DATA SHEET

PCK210

Low voltage dual 1:5 differential ECL/PECL clock driver

Product data
Supersedes data of 2002 Apr 11

2002 Nov 13

Philips
Semiconductors

PHILIPS

Low voltage dual 1:5 differential ECL/PECL clock driver

PCK210

FEATURES

- 85 ps part-to-part skew typical
- 20 ps output-to-output skew typical
- Differential design
- V_{BB} output
- Voltage and temperature compensated outputs
- Low voltage V_{EE} range of -2.25 V to -3.8 V
- 75 kΩ input pull-down resistors
- Form, fit, and function compatible with MC100EP210

DESCRIPTION

The PCK210 is a low skew 1-to-5 dual differential driver, designed with clock distribution in mind. The input signals can be either differential or single-ended if the V_{BB} output is used. The signal is fanned out to 5 identical differential outputs.

The PCK210 is specifically designed, modeled and produced with low skew as the key goal. Optimal design and layout serve to minimize gate-to-gate skew within a device, and empirical modeling is used to determine process control limits that ensure consistent t_{PD} distributions from lot to lot. The net result is a dependable, guaranteed low skew device.

To ensure that the tight skew specification is met, it is necessary that both sides of the differential output are terminated into 50 Ω, even if only one side is being used. In most applications, all ten differential pairs will be used, and therefore terminated. In the case where fewer than ten pairs are used, it is necessary to terminate at least the output pairs on the same package side as the pair(s) being used on that side, in order to maintain minimum skew. Failure to do this will result in small degradations of propagation delay (on the order of 10-20 ps) of the output(s) being used, which, while not being catastrophic to most designs, will mean a loss of skew margin.

The PCK210, as with most other ECL devices, can be operated from a positive V_{CC} supply in PECL mode. This allows the PCK210 to be used for high performance clock distribution in +3.3 V or +2.5 V systems. Designers can take advantage of the PCK210's performance to distribute low skew clocks across the backplane or the board. In a PECL environment, series or Thevenin line terminations are typically used as they require no additional power supplies.

The PCK210 may be driven single-endedly utilizing the V_{BB} bias output with the $\overline{CLKA}$ or $\overline{CLKB}$ input. If a single-ended signal is to be used, the V_{BB} pin should be connected to the $\overline{CLKA}$ or $\overline{CLKB}$ input and bypassed to ground via a 0.01 μF capacitor. The V_{BB} output can only source/sink 0.3 mA, therefore, it should be used as a switching reference for the PCK210 only. Part-to-part skew specifications are not guaranteed when driving the PCK210 single-endedly.

PINNING

Pin configuration

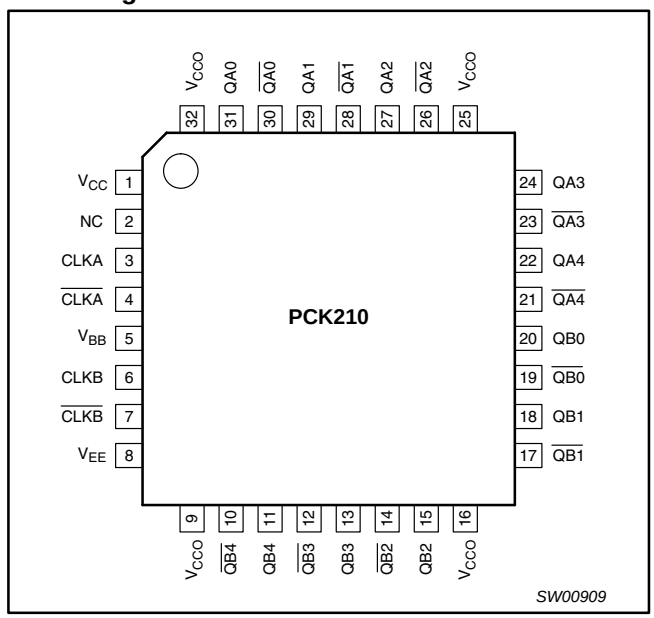


Figure 1. Pin configuration

Pin description

SYMBOL	PIN	DESCRIPTION
V_{CC}	1	Supply voltage
NC	2	Not connected
CLKA, $\overline{CLKA}$	3, 4	Differential input pair
V_{BB}	5	V_{BB} output
CLKB, $\overline{CLKB}$	6, 7	Differential input pair
V_{EE}	8	Ground
V_{CCO}	9, 16, 25, 32	Output drive power supply voltage
QA0-QA4, QB0-QB4	31, 29, 27, 24, 22, 20, 18, 15, 13, 11	Differential outputs
$\overline{QA0}$-$\overline{QA4}$, $\overline{QB0}$-$\overline{QB4}$	30, 28, 26, 23, 21, 19, 17, 14, 12, 10	Differential outputs

ORDERING INFORMATION

Type number	Package			Temperature range
	Name	Description	Version	
PCK210BD	LQFP32	plastic low profile quad flat package; 32 leads; body 7 × 7 × 1.4 mm	SOT358-1	-40 to +70 °C

LOGIC SYMBOL

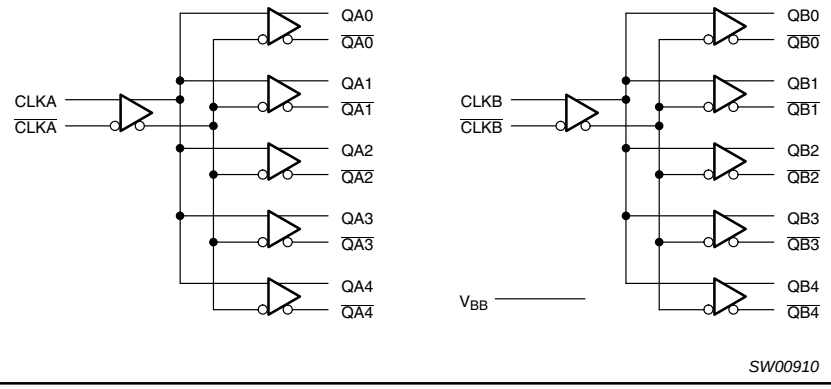


Figure 2. Logic symbol

ABSOLUTE MAXIMUM RATINGS[1]

In accordance with the Absolute Maximum Rating System (IEC 134)

SYMBOL	PARAMETER	LIMITS		UNIT
		MIN	MAX	
V_{CC}	Supply voltage	-0.3	+4.6	V
V_I	Input voltage	-0.3	V_{CC} + 0.3	V
I_{IN}	Input current	-	±20	mA
T_{stg}	Storage temperature range	-40	+125	°C
ESD_{HBM}	Electrostatic discharge (Human Body Model; 1.5 kΩ, 100 pF)	-	>1750	V
ESD_{MM}	Electrostatic discharge (Machine Model; 0 kΩ, 100 pF)	-	>200	V
ESD_{CDM}	Electrostatic discharge (Charge Device Model)	-	>1000	V

NOTE:
1. Absolute maximum continuous ratings are those values beyond which damage to the device may occur. Exposure to these conditions or conditions beyond those indicated may adversely affect device reliability. Functional operation under absolute-maximum-rated conditions is not implied.

RECOMMENDED OPERATING CONDITIONS

SYMBOL	PARAMETER	CONDITIONS	MIN	MAX	UNIT
V_{CC}	Supply voltage		2.25	3.8	V
V_{IR}	Receiver input voltage		V_{EE}	V_{CC}	V
V_{DIFF}	Input differential voltage[1]	$V_{(CLKinN)}$- $V_{(CLKin)}$	—	1.00	V
T_{amb}	Operating ambient temperature range in free air		-40	+85	°C

NOTE:
1. To idle an unused differential clock input, connect one input terminal (e.g. CLK1) to V_{BB} and leave its complimentary input terminal (e.g. $\overline{CLK1}$) open-circuit, in which case $\overline{CLK1}$ will default low by its internal pull-down reistor. Inputs should not be shorted to ground or V_{CC}.

THERMAL CHARACTERISTICS

Proper thermal management is critical for reliable system operation. This is especially true for high fan-out and high drive capability products.

DC ELECTRICAL CHARACTERISTICS

V_{supply}: $V_{CC} = V_{CCO} = 0.0$ V; $V_{EE} = -2.25$ V to -3.80 V.

SYMBOL	PARAMETER	CONDITIONS	-40 °C		+25 °C		+85 °C		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
I_{EE}	Internal supply current	Absolute value of current	20	80	20	85	30	90	mA
I_{CC}	Output and internal supply current	All outputs terminated 50 Ω to V_{CC} - 2.0 V	270	390	270	395	270	405	mA
I_{IN}	Input current	Includes pull-up/pull-down resistors	-	150	-	150	-	150	µA
V_{BB}	Internally generated bias voltage	for $V_{EE} = -2.25$ V to -3.8 V	-1.38	-1.16	-1.38	-1.16	-1.38	-1.16	V
V_{PP}	Input amplitude	Difference of input ≈ V_{IH} - V_{IL} (Note 1)	0.5	1.3	0.5	1.3	0.5	1.3	V
V_{CMR}	Common mode voltage	Crosspoint of input ≈ average (V_{IH}, V_{IL})	V_{EE} + 1.0	-0.3	V_{EE} + 1.0	-0.3	V_{EE} + 1.0	-0.3	V
V_{OH}	HIGH-level output voltage	$I_{OH} = -30$ mA	-1.30	-0.95	-	-	-1.20	-0.85	V
V_{OL}	LOW-level output voltage	$I_{OL} = -5$ mA	-1.85	-1.40	-	-	-1.90	-1.50	V
V_{OUTpp}	Differential output swing		350	-	-	-	500	-	mV

DC ELECTRICAL CHARACTERISTICS

V_{supply}: $V_{CC} = V_{CCO} = 2.25$ V to 3.80 V; $V_{EE} = 0.0$ V.

SYMBOL	PARAMETER	CONDITIONS	-40 °C		+25 °C		+85 °C		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
I_{EE}	Internal supply current	Absolute value of current	20	80	20	85	30	90	mA
I_{CC}	Output and internal supply current	All outputs terminated 50 Ω to V_{CC} - 2.0 V	270	390	270	395	270	405	mA
I_{IN}	Input current	Includes pull-up/ pull-down resistors	-	150	-	150	-	150	µA
V_{BB}	Internally generated bias voltage	$V_{CC} = 2.25$ V to 3.8 V	V_{CC} - 1.38	V_{CC} - 1.16	V_{CC} - 1.38	V_{CC} - 1.16	V_{CC} - 1.38	V_{CC} - 1.16	V
V_{PP}	Input amplitude	Difference of input ≈ V_{IH} - V_{IL} (Note 1)	0.5	1.3	0.5	1.3	0.5	1.3	V
V_{CMR}	Common mode voltage	Crosspoint of input ≈ average (V_{IH}, V_{IL})	1	V_{CC} - 0.3	1	V_{CC} - 0.3	1	V_{CC} - 0.3	V
V_{OH}	HIGH-level output voltage	$I_{OH} = -30$ mA	V_{CC} - 1.30	V_{CC} - 0.95	-	-	V_{CC} - 1.20	V_{CC} - 0.85	V
V_{OL}	LOW-level output voltage	$I_{OL} = -5$ mA	V_{CC} - 1.85	V_{CC} - 1.40	-	-	V_{CC} - 1.90	V_{CC} - 1.50	V
V_{OUTpp}	Differential output swing		350	-	-	-	500	-	mV

NOTE:

1. V_{PP} minimum and maximum required to maintain AC specifications. Actual device function will tolerate minimum V_{PP} of 100 mV.

AC CHARACTERISTICS — PECL input

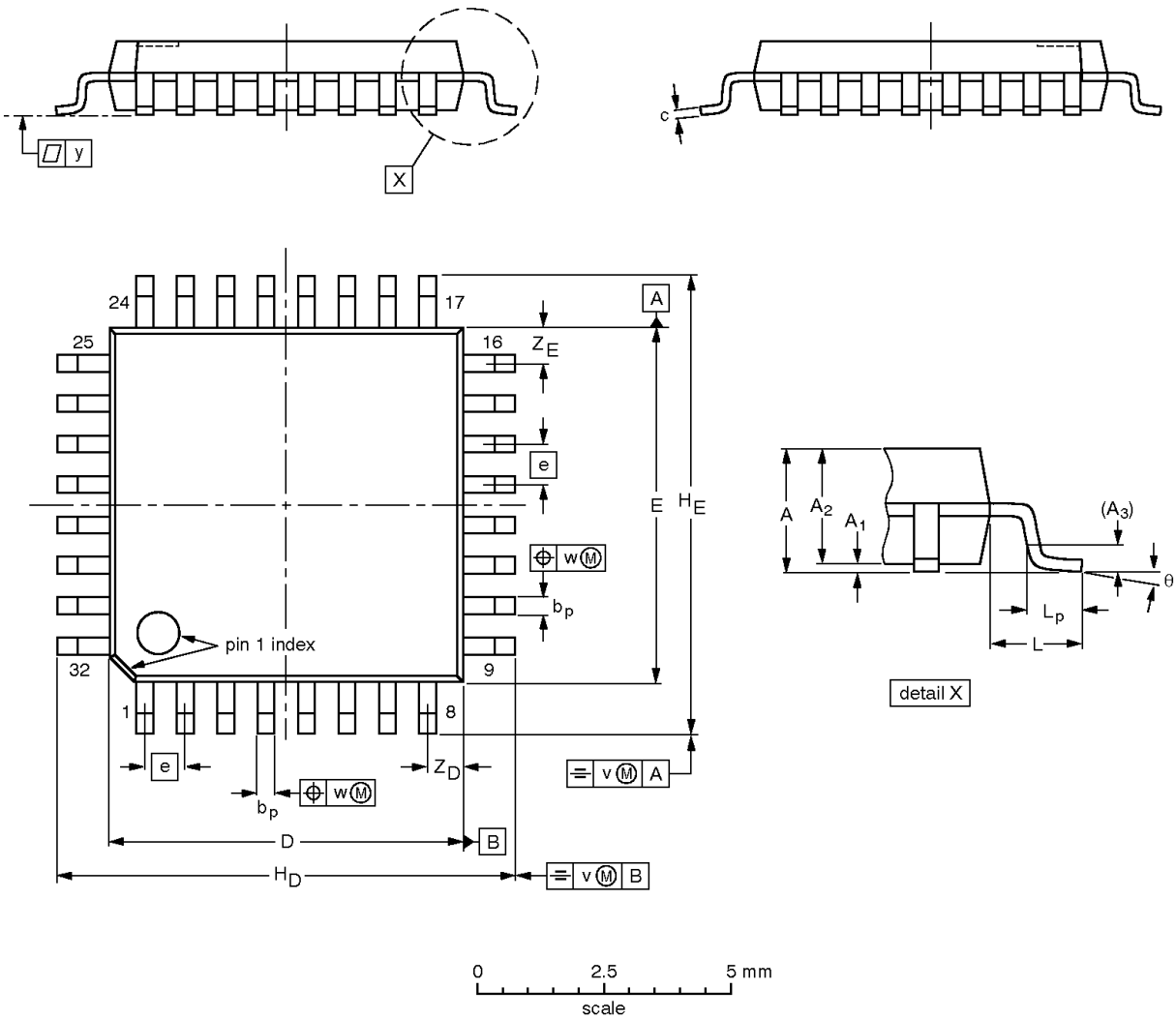
V_{supply}: $V_{CC} = V_{CCO}$ = 2.25 V to 3.80 V; V_{EE} = 0.0 V *-OR-* $V_{CC} = V_{CCO}$ = 0.0 V; V_{EE} = -2.25 V to -3.80 V.

SYMBOL	PARAMETER	CONDITIONS	-40 °C			+25 °C			+85 °C			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
t_{PD}	Differential propagation delay CLK,$\overline{CLK}$ to all Q0,$\overline{Q0}$ through Q4,$\overline{Q4}$	Nominal (single input condition) V_{PP} = 0.650 V, $V_{CMR} = V_{CC}$ - 0.800 V Applies to 500 MHz reference. (Note 1)	270	-	420	300	-	450	380	-	530	ps
$t_{SK(part)}$	Part-to-part skew	Single input condition (Note 1)	-	-	110	-	-	110	-	-	110	ps
$t_{SK(output)}$	Output-to-output skew for given part	Single input condition (Note 1)	-	15	50	-	15	50	-	15	50	ps
t_{PD}	Differential propagation delay CLK,$\overline{CLK}$ to all Q0,$\overline{Q0}$ through Q4,$\overline{Q4}$	All input conditions (Note 1)	220	-	520	250	-	550	320	-	620	ps
$t_{SK(part)}$	Part-to-part skew	(Note 1)	-	-	160	-	-	160	-	-	160	ps
$t_{SK(output)}$	Output-to-output skew for given part	(Note 1)	-	15	50	-	15	50	-	15	50	ps
t_{jitter}	Cyle-to-cycle jitter		-	-	1	-	-	1	-	-	1	ns
f_{MAX}	Maximum frequency	Functional to 1.5 GHz Timing specifications apply up to 1.0 GHz	-	-	1500	-	-	1500	-	-	1500	MHz
t_r, t_f	Output rise and fall times (20%, 80%)	(Note 1)	100	-	320	100	-	320	100	-	320	ps

NOTE:

1. For operation with 2.5 V supply, the output termination is 50 Ω to V_{EE}.
   For operation at 3.3 V supply, the output termination is 50 Ω to V_{CC} - 2 V.

LQFP32: plastic low profile quad flat package; 32 leads; body 7 x 7 x 1.4 mm **SOT358-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L	L_p	v	w	y	$Z_D^{(1)}$	$Z_E^{(1)}$	θ
mm	1.60	0.20 0.05	1.45 1.35	0.25	0.4 0.3	0.18 0.12	7.1 6.9	7.1 6.9	0.8	9.15 8.85	9.15 8.85	1.0	0.75 0.45	0.2	0.25	0.1	0.9 0.5	0.9 0.5	7^o 0^o

Note

1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT358 -1	136E03	MS-026				~~99-12-27~~ 00-01-19

REVISION HISTORY

Rev	Date	Description
_2	20021213	Product data (9397 750 10866); ECN 853-2336 29225 of 22 November 2002. Modifications: • Addition of jitter specification to datasheet.
_1	20020411	Product data (9397 750 09657); ECN 853-2336 27995 of 11 April 2002.

Low voltage dual 1:5 differential ECL/PECL clock driver

PCK210

Data sheet status

Level	Data sheet status[1]	Product status[2] [3]	Definitions
I	Objective data	Development	This data sheet contains data from the objective specification for product development. Philips Semiconductors reserves the right to change the specification in any manner without notice.
II	Preliminary data	Qualification	This data sheet contains data from the preliminary specification. Supplementary data will be published at a later date. Philips Semiconductors reserves the right to change the specification without notice, in order to improve the design and supply the best possible product.
III	Product data	Production	This data sheet contains data from the product specification. Philips Semiconductors reserves the right to make changes at any time in order to improve the design, manufacturing and supply. Relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN).

[1] Please consult the most recently issued data sheet before initiating or completing a design.

[2] The product status of the device(s) described in this data sheet may have changed since this data sheet was published. The latest information is available on the Internet at URL http://www.semiconductors.philips.com.

[3] For data sheets describing multiple type numbers, the highest-level product status determines the data sheet status.

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 60134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products—including circuits, standard cells, and/or software—described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information

For additional information please visit
http://www.semiconductors.philips.com. Fax: **+31 40 27 24825**

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com.

Date of release: 12-02

Document order number: 9397 750 10866

Let's make things better.

Philips Semiconductors

PHILIPS

Virtex-E High Performance Differential Solutions: Low Voltage Differential Signalling (LVDS)

Introduction

As the need for higher bandwidth accelerates, system designers are choosing differential signaling to satisfy high bandwidth requirements while reducing power, increasing noise immunity, and decreasing EMI emissions. LVDS is a low swing, differential signaling technology providing very fast data transmission, common-mode noise rejection, and low power consumption over a broad frequency range. The Virtex-E family delivers the programmable industry's highest bandwidth and most flexible differential signaling solution for direct interfacing to industry-standard LVDS devices.

With up to 36 I/O pairs operating at 622 Megabits per second (Mb/s) or up to 344 I/O pairs operating at over 311 Mb/s, the Virtex-E family supports multiple 10 Gb/s ports while maintaining high signal integrity with low power consumption. Unlike other PLD solutions, all Virtex-E LVDS I/Os support input, output, and I/O signaling, providing a system designer unparalleled flexibility in board layout. Table 1 summarizes the LVDS support in the Virtex-E family.

Table 1: Virtex-E High-Bandwidth LVDS Support Summary

LVDS Configuration	Bandwidth
Point-to-Point	36 pairs @ 622 Mb/s or 344 pairs @ 311 Mb/s
Multi-Drop 	344 pairs @ 311 Mb/s

The LVDS Standard

LVDS is defined by two industry standards: ANSI/TIA/EIA-644 and IEEE 1596.3 SCI-LVDS.

- The ANSI/TIA/EIA-644 standard defines LVDS electrical specs including driver output and receiver input electrical characteristics. It does not cover functional specifications, protocols, or transmission medium characteristics since these are application dependent. The ANSI/TIA/EIA-644 is the more generic of the two standards, and is intended for multiple applications.
- The IEEE 1596.3 SCI-LVDS standard is a subset of SCI (Scalable Coherent Interface). The SCI-LVDS standard defines electrical specifications for the physical layer interface of SCI. It is similar to the ANSI/TIA/EIA-644 standard but differs in the intended usage of the interface. The IEEE committee created the SCI-LVDS standard for communication between SCI nodes.

The Virtex-E LVDS solution conforms to the ANSI/TIA/EIA-644 standard. Table 2 summarizes the pertinent Virtex-E LVDS DC specifications.

Table 2: LVDS DC Specifications

DC Parameter	Conditions	Min	Typ	Max	Units
V_{CCO}		2.375	2.5	2.625	V
Output High Voltage for Q and $\overline{Q}$	R_T = 100 Ω across Q and $\overline{Q}$ signals	1.25	1.425	1.6	V
Output Low Voltage for Q and $\overline{Q}$	R_T = 100 Ω across Q and $\overline{Q}$ signals	0.9	1.075	1.25	V
Differential Output Voltage (Q - $\overline{Q}$), Q = High ($\overline{Q}$ - Q), $\overline{Q}$ = High	R_T = 100 Ω across Q and $\overline{Q}$ signals	250	350	450	mV
Output Common-Mode Voltage	R_T = 100 Ω across Q and $\overline{Q}$ signals	1.125	1.25	1.375	V
Differential Input Voltage (Q - $\overline{Q}$), Q = High ($\overline{Q}$ - Q), $\overline{Q}$ = High	Common-mode input voltage = 1.25 V	100	350	NA	mV
Input Common-Mode Voltage	Differential input voltage = ± 350 mV	0.2	1.25	2.2	V

Advantages

- LVDS is specified to be technology and process independent.
- LVDS is EMI tolerant. Common-mode noise is equally removed by two conductors and rejected by the receiver.
- No transmission medium is defined in the standard. The medium can be tailored to meet the specific application requirements.
- The typical LVDS voltage swing is 350 mV, resulting in a higher transfer rate and lower power consumption.

Configurations

There are two configurations that are used in LVDS applications, point-to-point and multi-drop. The Virtex-E family supports both LVDS configurations.

Point-to-Point

In point-to-point configuration, there is one transmitter and one receiver. The LVDS driver is a current source that drives a differential pair of lines. The typical current drive is 3.5 mA. The receiver has high DC impedance. The majority of the driver current flows across the termination resistor generating about 350 mV at the receiver inputs (Figure 1).

Multi-Drop

A multi-drop LVDS configuration has one transmitter and multiple receivers. The differential termination resistor is placed close to the last receiver (Figure 3).

Applications

Applications for LVDS include:

- Switches
- Repeaters
- Hubs
- Routers
- Wireless base stations
- Flat panel displays
- Digital cameras
- Printers
- Copiers

- Multimedia peripherals
- Backplane applications

Terminations

LVDS is widely used for high-speed point-to-point interface as well as multi-drop applications. Depending on the exact interconnect topology, precision resistors are required to match specific impedance characteristics to minimize reflection and ensure high signal integrity. The Virtex-E family supports the most flexible LVDS high-speed interface by supporting a flexible external termination scheme. This enables system designers select resistor values most appropriate for maximum performance.

Point-to-Point

Figure 1 shows the schematic of a standard LVDS driver driving the Virtex-E receiver. An LVDS driver drives the two 50 Ω transmission lines into a Virtex-E LVDS receiver. The two 50 Ω single-ended transmission lines can be micro-strip, strip-line, a 100 Ω differential twisted pair, or a similar balanced differential transmission line.

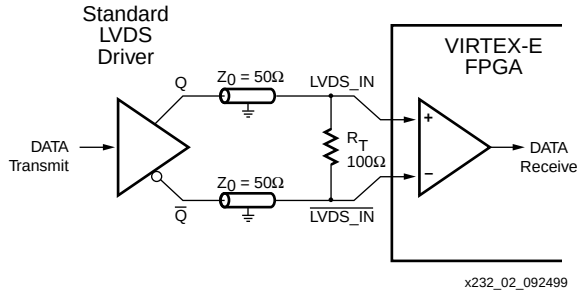


Figure 1: A Standard LVDS Driver Driving a Virtex-E LVDS Receiver

Figure 2 shows the complete schematic of the Virtex-E LVDS line driver and receiver. The standard LVDS 100 Ω termination resistor is connected across the LVDS_OUT and $\overline{\text{LVDS\_OUT}}$ outputs at the end of the transmission line. The resistors R_S and R_{DIV} attenuate signals from the Virtex-E LVDS drivers and provide a matched source impedance (series termination) to the transmission lines. Standard termination packs are available from **Bourns**. Other resistor vendors provide termination networks with up to 16-pins per pack.

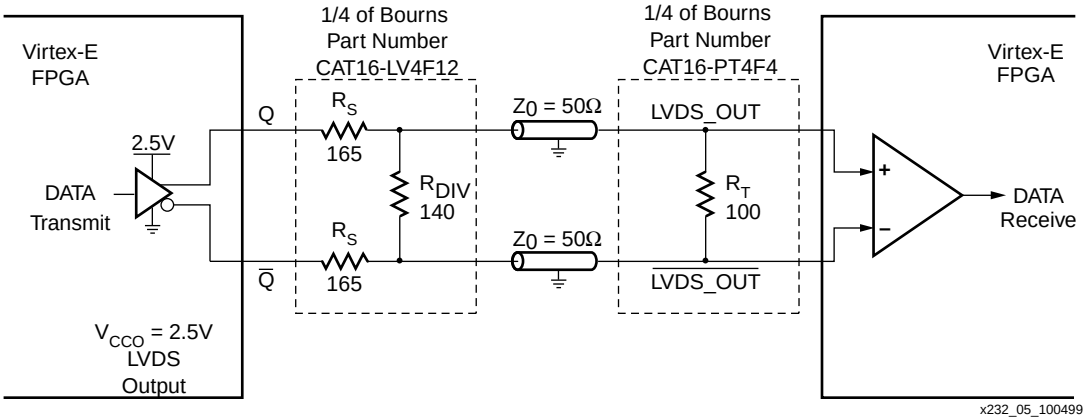


Figure 2: Virtex-E LVDS Line Driver and Receiver Schematic

Virtex-E LVDS driver meets ANSI/TIA/EIA-644 LVDS DC specifications. The matched source impedance of the Virtex-E LVDS driver absorbs nearly all differential reflections from the

capacitive load at the LVDS destination, reducing standing waves, undershoot, and signal swing on data bursts or clocks.

Data and clocks can be transmitted over cables longer than 5 ns electrical length, limited only by the quality of the cable (the cable attenuation caused by skin effect losses at high frequencies).

The 622 Mb/s data rate, or 311 MHz clock is achievable with a Virtex-E -7 speed grade devices. See "**XAPP233: LVDS Transceivers at 622 Mb/s using General-Purpose I/O**" for details of the reference design.

Multi-Drop

Multi-drop LVDS configuration allows many receivers to be driven by one Virtex-E LVDS driver. With simple source and differential termination, Virtex-E LVDS driver can drive lines with fan-outs of 20 to 1, making Virtex-E LVDS I/Os suitable for a broad variety of high-load applications.

Figure 3 illustrates a Virtex-E LVDS driver driving 20 LVDS receivers in a multi-drop configuration. The receivers can be either Virtex-E receivers or other off-the-shelf LVDS receivers. The LVDS signal is driven from a Virtex-E LVDS driver, and is daisy-chained with two 29 Ω transmission lines and stubs to all 20 LVDS receivers. Each LVDS receiver is connected to the main multi-drop lines every 2.5" for a multi-drop line length of 50". Each LVDS receiver tap line has a 1" maximum stub length with a 50 Ω transmission line impedance to ground, or a differential impedance of 100 Ω between the two stubs. A 44 Ω termination resistor R_T is placed across the differential lines close to the last LVDS receiver. Resistors R_S and R_{DIV} attenuate the signals from the Virtex-E drivers and provide a 22 Ω source impedance (series termination) to the 29 Ω transmission lines. The 22 Ω source impedance is used because the added load of the LVDS receivers brings the 29 Ω line down to an effective average impedance of 22 Ω. The capacitor C_{SLEW} reduces the slew rate from the Virtex-E LVDS driver, resulting in smaller reflections and less ringing at the receivers.

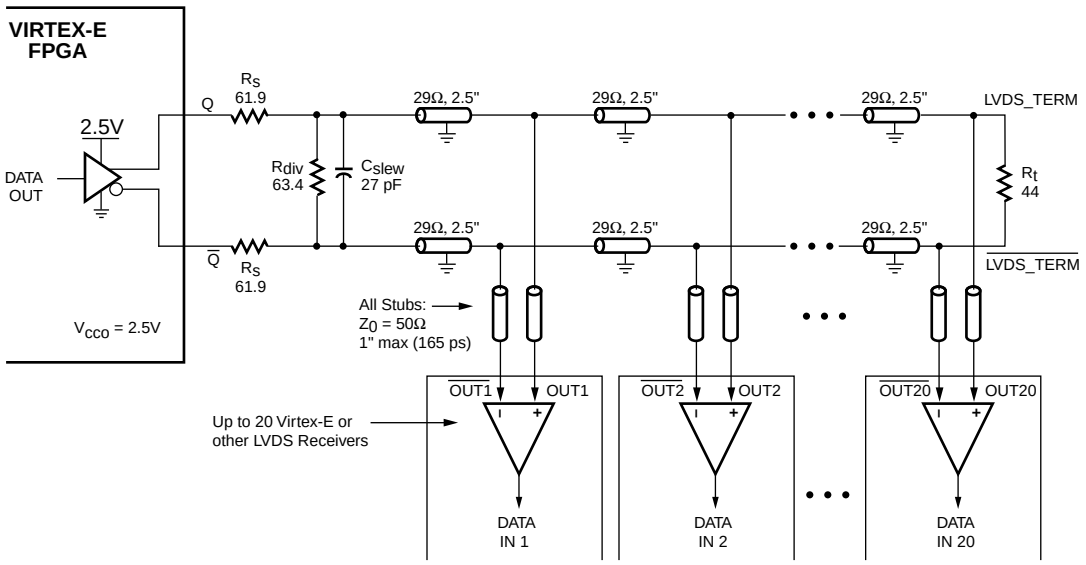


Figure 3: Virtex-E 20-load Multi-Drop LVDS Schematic

The two 29 Ω single-ended transmission lines can be micro-strip, strip-line, the single-ended equivalent of a 58 Ω twisted pair, or a similar balanced differential transmission line. The

resistors R_S and R_{DIV} should be placed close to the Virtex-E driver outputs. The parallel termination resistor R_T should be placed close to the last LVDS receiver inputs at the far end of the multi-drop line. The capacitor C_{SLEW} should be placed close to the resistors R_S and R_{DIV}.

The Virtex-E multi-drop LVDS driver adheres to all the ANSI/TIA/EIA-644 LVDS standard DC input level specifications, and is fully compatible with LVDS receivers from National Semiconductor and other companies.

The maximum data rate is 311 Mb/s or a clock of 155.5 MHz for a Virtex-E -7 speed grade device. Reliable data transmission is possible for up to 20 LVDS receivers over a multi-drop line length of 50 inches, limited only by skin effect losses in the PCB trace.

Waveform

A typical LVDS output waveform for Virtex-E devices is shown in Figure 4.

± 175 mV Swing
@ 1.25 V mid-point.

Computed Signal
Differential (Q - $\overline{Q}$)

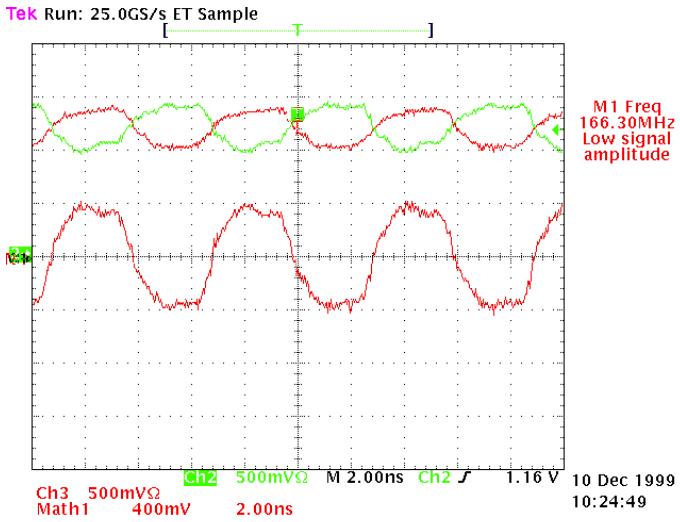


Figure 4: LVDS Output Waveform

Virtex Advantages

The Virtex-E devices are the first programmable logic devices available in the market incorporating advanced LVDS I/O capability with support for other differential standards (Bus LVDS and LVPECL). Unlike other announced architectures (for example, APEX E), the Virtex-E LVDS capability provides an abundance of LVDS-capable user I/O and clock pins, and the architectural flexibility shown in Table 3 to address true high-speed system issues. This capability works in concert with a robust delay locked loop (DLL) technology enabling designers to achieve maximum performance in their LVDS applications.

Table 3: Virtex-E High-Bandwidth LVDS Solution Summary

Feature	Virtex-E	APEX E
Offer LVDS as a standard feature	Yes, in all devices, packages, and speeds	Large "X" device only, fast speed only (More $$)
LVDS	Y	Y
Bus LVDS	Y	N
LVPECL	Y	N
LVDS Configurations	Point-to-point, Multi-drop, and Multi-point	Point-to-point and Multi-drop [1]
Maximum I/O Bandwidth	22 Gb/s (622 Mb/s/pair x 36 pairs) or 107 Gb/s (311 Mb/s/pair x 344 pairs)	10 Gb/s (622 Mb/s x 16 pairs)
Termination	Flexible External Termination	Inflexible Internal Termination on Outputs
High-Speed Differential Clock Pairs	4	1
Maximum # of differential pairs	344 In/Out	Dedicated 16 input and 16 output pairs. Not layout friendly.
Maximum Speed	622 Mb/s	622 Mb/s
Serializer/Deserializer	Flexible in CLB	Dedicated 8:1
Clock Recovery	N	N

1. APEX E has internal termination on the outputs and cannot guarantee high signal integrity due to an inability to impedance match.

Unlike other PLD solutions that only offer LVDS capability in the most expensive and highest speed grade options, the Virtex-E DLL and LVDS I/O capabilities are standard features available in all Virtex-E device/package combinations. The Virtex-E family offers the option to use up to 36 LVDS I/O pairs operating at 622 Mb/s or up to 344 LVDS I/O pairs operating at over 311 Mb/s to achieve over 100 Gb/s aggregate bandwidth. This enables system designers to support multiple 10 Gb/s ports architecture for high-performance DSP and data communication systems. Table 4 demonstrates the high-bandwidth LVDS solution provided by the Virtex-E family. In addition to offering a high-performance and highly flexible LVDS solution, Xilinx also works closely with other component vendors (e.g., Bourn for the resistor pack) to ensure inter-operability and help system designers further reduce the overall design complexity and system cost.

Table 4: Virtex-E High-Bandwidth LVDS Solution Summary

I/O Standard	Type	1	2	32	72	688
Virtex-E LVDS	Differential	NA	622 Mb/s	10 Gb/s	22 Gb/s	107 Gb/s
APEX E LVDS	Differential	NA	622 Mb/s	10 Gb/s	NA	NA

References

Standards

ANSI/TIA/EIA-644 **http://www.eia.org/eng**
IEEE1596.3 **http://www.ieee.org**

Related Xilinx Documents

XAPP230: "The LVDS I/O Standard" at: **http://www.xilinx.com/xapp/xapp230.pdf**
XAPP231: "Multi-drop LVDS" at: **http://www.xilinx.com/xapp/xapp231.pdf**
XAPP232: "The LVDS Drivers and Receivers: Interface Guidelines" at:
http://www.xilinx.com/xapp/xapp232.pdf

XAPP233: "LVDS Transceivers at 622 Mb/s using General Purpose I/O" at:
http://www.xilinx.com/xapp/xapp233.pdf

19-2511; Rev 2; 10/02

MAXIM

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

General Description

The MAX9325 low-skew, 2:8 differential driver features extremely low output-to-output skew (50ps max) and part-to-part skew (225ps max). These features make the device ideal for clock and data distribution across a backplane or board. The device selects one of the two differential HSTL or LVECL/LVPECL inputs and repeats them at eight differential outputs. Outputs are compatible with LVECL and LVPECL, and can directly drive 50Ω terminated transmission lines.

The differential inputs can be configured to accept a single-ended signal when the unused complementary input is connected to the on-chip reference output voltage V_{BB}. All inputs have internal pulldown resistors to V_{EE}. The internal pulldowns and a fail-safe circuit ensure differential low default outputs when the inputs are left open or at V_{EE}.

The MAX9325 operates over a 2.375V to 3.8V supply range for interfacing to differential HSTL and LVPECL signals. This allows high-performance clock or data distribution in systems with a nominal +2.5V or +3.3V supply. For LVECL operation, the device operates with a -2.375V to -3.8V supply.

The MAX9325 is offered in 28-lead PLCC and space-saving 28-lead QFN packages. The MAX9325 is specified for operation from -40°C to +85°C.

Applications

Precision Clock Distribution

Low-Jitter Data Repeaters

Features

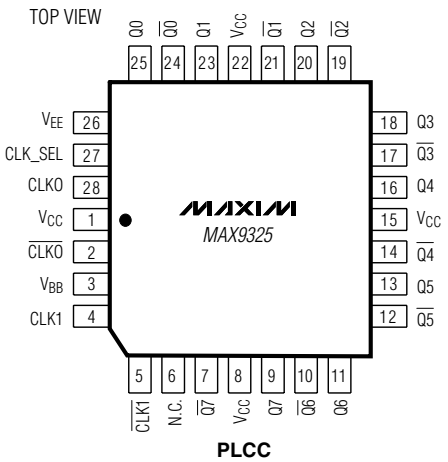
- **50ps (max) Output-to-Output Skew**
- **1.5ps$_{RMS}$ (max) Random Jitter**
- **Guaranteed 300mV Differential Output at 700MHz**
- **+2.375V to +3.8V Supplies for Differential HSTL/LVPECL**
- **-2.375V to -3.8V Supplies for Differential LVECL**
- **Two Selectable Differential Inputs**
- **On-Chip Reference for Single-Ended Inputs**
- **Outputs Low for Inputs Open or at V_{EE}**
- **Pin Compatible with MC100LVE310**

Ordering Information

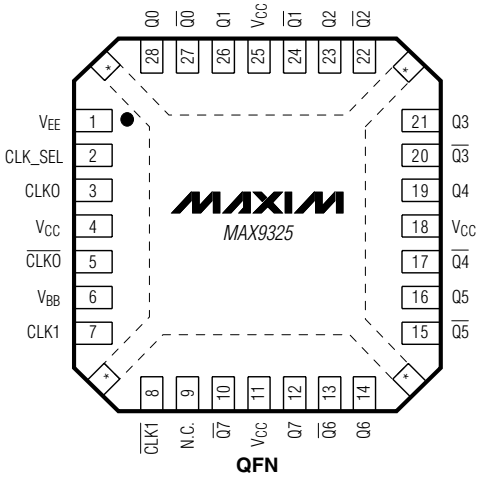
PART	TEMP RANGE	PIN-PACKAGE
MAX9325EQI	-40°C to +85°C	28 PLCC
MAX9325EGI	-40°C to +85°C	28 QFN 5mm x 5mm

Functional Diagram appears at end of data sheet.

Pin Configurations



PLCC



QFN

*CORNER PINS AND EXPOSED PAD ARE CONNECTED TO V_{EE}.

INPUT SELECT TRUTH TABLE

CLK_SEL	INPUT CLOCK
L	CLK0, $\overline{CLK0}$ SELECTED
H	CLK1, $\overline{CLK1}$ SELECTED

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

ABSOLUTE MAXIMUM RATINGS

V_{CC} - V_{EE} ..... -0.3V to +4.1V
Inputs (CLK_, $\overline{CLK\_}$, CLK_SEL) to V_{EE} ..... -0.3V to (V_{CC} + 0.3V)
CLK_ to $\overline{CLK\_}$ ..... ±3.0V
Continuous Output Current 50mA
Surge Output Current 100mA
V_{BB} Sink/Source Current ..... ±0.65mA
Continuous Power Dissipation (T_A = +70°C)
28-Lead PLCC (derate 10.5mW/°C above +70°C) 842mW
θ_{JA} in Still Air ..... +95°C/W
θ_{JC} ..... +25°C/W
28-Lead QFN (derate 20.8mW/°C above +70°C) 1667mW
θ_{JA} in Still Air ..... +48°C/W
θ_{JC} ..... +2°C/W
Operating Temperature Range ..... -40°C to +85°C
Junction Temperature ..... +150°C
Storage Temperature Range ..... -65°C to +150°C
ESD Protection
Human Body Model (CLK_, $\overline{CLK\_}$, Q_, $\overline{Q\_}$) ..... ≥2kV
Soldering Temperature (10s) ..... +300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC ELECTRICAL CHARACTERISTICS

((V_{CC} - V_{EE}) = 2.375V to 3.8V, R_L = 50Ω ±1% to V_{CC} - 2V. Typical values are at (V_{CC} - V_{EE}) = 3.3V, V_{IH} = (V_{CC} - 1V), V_{IL} = (V_{CC} - 1.5V).) (Notes 1–4)

PARAMETER	SYMBOL	CONDITIONS	-40°C			+25°C			+85°C			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
SINGLE-ENDED INPUT (CLK_SEL)												
Single-Ended Input High Voltage	V_{IH}	Figure 1	V_{CC} - 1.165		V_{CC}	V_{CC} - 1.165		V_{CC}	V_{CC} - 1.165		V_{CC}	V
Single-Ended Input Low Voltage	V_{IL}	Figure 1	V_{EE}		V_{CC} - 1.475	V_{EE}		V_{CC} - 1.475	V_{EE}		V_{CC} - 1.475	V
Input Current	I_{IN}	V_{IH}, V_{IL}	-10.0		+150	-10.0		+150	-10.0		+150	µA
DIFFERENTIAL INPUT (CLK_, $\overline{CLK\_}$)												
Single-Ended Input High Voltage	V_{IH}	Figure 1	V_{CC} - 1.165		V_{CC}	V_{CC} - 1.165		V_{CC}	V_{CC} - 1.165		V_{CC}	V
Single-Ended Input Low Voltage	V_{IL}	Figure 1	V_{EE}		V_{CC} - 1.475	V_{EE}		V_{CC} - 1.475	V_{EE}		VCC - 1.475	V
Differential Input High Voltage	V_{IHD}	Figure 1	V_{EE} + 1.2		V_{CC}	V_{EE} + 1.2		V_{CC}	V_{EE} + 1.2		V_{CC}	V

DC ELECTRICAL CHARACTERISTICS (continued)

((V_{CC} - V_{EE}) = 2.375V to 3.8V, R_L = 50Ω ±1% to V_{CC} - 2V. Typical values are at (V_{CC} - V_{EE}) = 3.3V, V_{IH} = (V_{CC} - 1V), V_{IL} = (V_{CC} - 1.5V).) (Notes 1–4)

PARAMETER	SYMBOL	CONDITIONS	-40°C			+25°C			+85°C			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
Differential Input Low Voltage	V_{ILD}	Figure 1	V_{EE}		V_{CC} - 0.095	V_{EE}		V_{CC} - 0.095	V_{EE}		V_{CC} - 0.095	V
Differential Input Voltage	V_{IHD} - V_{ILD}	(V_{CC} - V_{EE}) < 3.0V, Figure 1	0.095		V_{CC} - V_{EE}	0.095		V_{CC} - V_{EE}	0.095		V_{CC} - V_{EE}	V
		(V_{CC} - V_{EE}) ≥ 3.0V, Figure 1	0.095		3.0	0.095		3.0	0.095		3.0	
Input Current	I_{IN}	V_{IH}, V_{IL}, V_{IHD}, V_{ILD}	-10.0		+150.0	-10.0		+150.0	-10.0		+150.0	µA
OUTPUT (Q_, $\overline{Q}$_)												
Single-Ended Output High Voltage	V_{OH}	Figure 2	V_{CC} - 1.085	V_{CC} - 0.977	V_{CC} - 0.880	V_{CC} - 1.025	V_{CC} - 0.949	V_{CC} - 0.88	V_{CC} - 1.025	V_{CC} - 0.929	V_{CC} - 0.88	V
Single-Ended Output Low Voltage	V_{OL}	Figure 2	V_{CC} - 1.810	V_{CC} - 1.695	V_{CC} - 1.620	V_{CC} - 1.810	V_{CC} - 1.697	V_{CC} - 1.62	V_{CC} - 1.810	V_{CC} - 1.698	V_{CC} - 1.62	V
Differential Output Voltage	V_{OH} - V_{OL}	Figure 2	535	718		595	749		595	769		mV
REFERENCE VOLTAGE OUTPUT (V_{BB})												
Reference Voltage Output	V_{BB}	I_{BB} = ±0.5mA (Note 5)	V_{CC} - 1.38	V_{CC} - 1.318	V_{CC} - 1.26	V_{CC} - 1.38	V_{CC} - 1.325	V_{CC} - 1.26	V_{CC} - 1.38	V_{CC} - 1.328	V_{CC} - 1.26	V
SUPPLY												
Supply Current	I_{EE}	(Note 6)		35	50		39	55		42	65	mA

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

AC ELECTRICAL CHARACTERISTICS—PLCC Package

((V_{CC} - V_{EE}) = 2.375V to 3.8V, R_L = 50Ω ±1% to V_{CC} - 2V, f_{IN} ≤ 500MHz, input transition time = 125ps (20% to 80%). Typical values are at (V_{CC} - V_{EE}) = 3.3V, V_{IH} = (V_{CC} - 1V), V_{IL} = (V_{CC} - 1.5V).) (Note 7)

PARAMETER	SYMBOL	CONDITIONS	-40°C			+25°C			+85°C			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
Differential Input-to-Output Delay	t_{PLHD} t_{PHLD}	Figure 2	475		650	460		710	490		740	ps
Single-Ended Input-to-Output Delay	t_{PLH} t_{PHL}	Figure 3 (Note 8)	440		780	430		790	450		800	ps
Output-to-Output Skew	t_{SKOO}	(Note 9)			50			50			50	ps
Part-to-Part Skew	t_{SKPP}	Differential input (Note 10)			160			190			225	ps
Added Random Jitter	t_{RJ}	f_{IN} = 0.5GHz clock pattern (Note 11)			1.5			1.5			1.5	ps_{RMS}
Added Deterministic Jitter	t_{DJ}	f_{IN} = 1.0Gbps, $2E^{23}$ - 1 PRBS pattern (Note 11)			100			100			100	ps_{P-P}
Switching Frequency	f_{MAX}	V_{OH} - V_{OL} ≥ 300mV clock pattern	1.5			1.5			1.5			GHz
Output Rise/Fall Time (20% to 80%)	t_R, t_F	Figure 2	140		440	140		440	140		440	ps

AC ELECTRICAL CHARACTERISTICS—QFN Package

((V_{CC} - V_{EE}) = 2.375V to 3.8V, R_L = 50Ω ±1% to V_{CC} - 2V, f_{IN} ≤ 500MHz, input transition time = 125ps (20% to 80%). Typical values are at (V_{CC} - V_{EE}) = 3.3V, V_{IH} = (V_{CC} - 1V), V_{IL} = (V_{CC} - 1.5V).) (Note 7)

PARAMETER	SYMBOL	CONDITIONS	-40°C			+25°C			+85°C			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
Differential Input-to-Output Delay	t_{PLHD} t_{PHLD}	Figure 2	250		575	298		553	309		576	ps
Single-Ended Input-to-Output Delay	t_{PLH} t_{PHL}	Figure 3 (Note 8)	253		581	310		586	324		606	ps
Output-to-Output Skew	t_{SKOO}	(Note 9)			50			50			50	ps
Part-to-Part Skew	t_{SKPP}	Differential input (Note 10)			192			215			218	ps
Added Random Jitter	t_{RJ}	f_{IN} = 0.5GHz clock pattern (Note 11)			1.5			1.5			1.5	ps_{RMS}
Added Deterministic Jitter	t_{DJ}	f_{IN} = 1.0Gbps, $2E^{23}$ - 1 PRBS pattern (Note 11)			95			95			95	ps_{P-P}
Switching Frequency	f_{MAX}	V_{OH} - V_{OL} ≥ 300mV clock pattern	1.5			1.5			1.5			GHz
Output Rise/Fall Time (20% to 80%)	t_R, t_F	Figure 2	97		411	104		210	111		232	ps

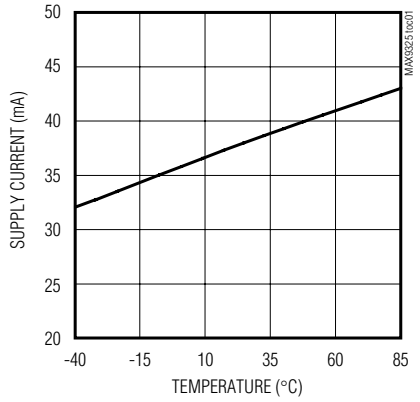
Note 1: Measurements are made with the device in thermal equilibrium.
Note 2: Current into a pin is defined as positive. Current out of a pin is defined as negative.
Note 3: DC parameters production tested at T_A = +25°C and guaranteed by design over the full operating temperature range.
Note 4: Single-ended input operation using V_{BB} is limited to (V_{CC} - V_{EE}) = 3.0V to 3.8V.
Note 5: Use V_{BB} only for inputs that are on the same device as the V_{BB} reference.
Note 6: All pins open except V_{CC} and V_{EE}.
Note 7: Guaranteed by design and characterization. Limits are set at ±6 sigma.
Note 8: Measured from the 50% point of the input signal with the 50% point equal to V_{BB}, to the 50% point of the output signal.
Note 9: Measured between outputs of the same part at the signal crossing points for a same-edge transition. Differential input signal.
Note 10: Measured between outputs of different parts under identical condition for same-edge transition.
Note 11: Device jitter added to the input signal. Differential input signal.

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

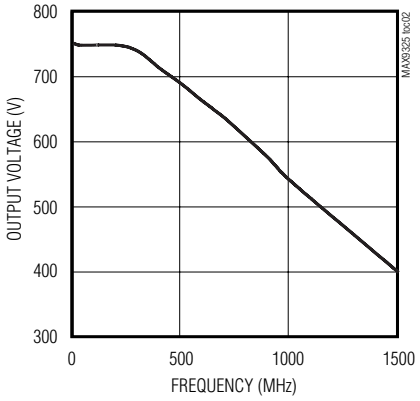
Typical Operating Characteristics

(PLCC package, typical values are at (V_{CC} - V_{EE}) = 3.3V, V_{IH} = (V_{CC} - 1V), V_{IL} = (V_{CC} - 1.5V), R_L = 50Ω ±1% to V_{CC} - 2V, f_{IN} = 500MHz, input transition time = 125ps (20% to 80%).)

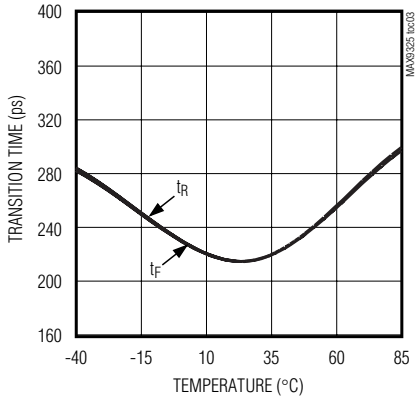
SUPPLY CURRENT (I_{EE}) vs. TEMPERATURE



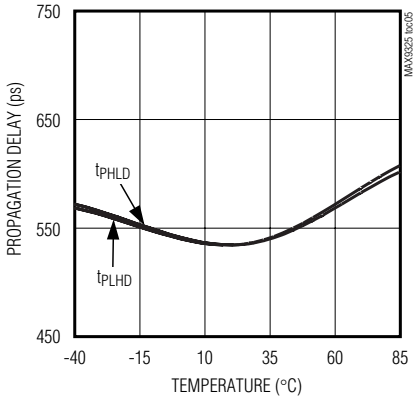
OUTPUT AMPLITUDE (V_{OH} - V_{OL}) vs. FREQUENCY



TRANSITION TIME vs. TEMPERATURE



PROPAGATION DELAY vs. TEMPERATURE



Pin Description

PIN		NAME	FUNCTION
PLCC	QFN		
1, 8, 15, 22	4, 11, 18, 25	V_{CC}	Positive Supply Voltage. Bypass each V_{CC} to V_{EE} with 0.1µF and 0.01µF ceramic capacitors. Place the capacitors as close to the device as possible, with the smaller value capacitor closest to the device.
2	5	$\overline{CLK0}$	Inverting Differential Clock Input 0. Internal 105kΩ pulldown to V_{EE}.
3	6	V_{BB}	Reference Output Voltage. Connect to the inverting or noninverting clock input to provide a reference for single-ended operation. When used, bypass V_{BB} to V_{CC} with a 0.01µF ceramic capacitor. Otherwise leave open.
4	7	CLK1	Noninverting Differential Clock Input 1. Internal 105kΩ pulldown to V_{EE}.
5	8	$\overline{CLK1}$	Inverting Differential Clock Input 1. Internal 105kΩ pulldown to V_{EE}.
6	9	N.C.	Not Connected
7	10	$\overline{Q7}$	Inverting Q7 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
9	12	Q7	Noninverting Q7 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
10	13	$\overline{Q6}$	Inverting Q6 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
11	14	Q6	Noninverting Q6 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
12	15	$\overline{Q5}$	Inverting Q5 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
13	16	Q5	Noninverting Q5 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
14	17	$\overline{Q4}$	Inverting Q4 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
16	19	Q4	Noninverting Q4 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
17	20	$\overline{Q3}$	Inverting Q3 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
18	21	Q3	Noninverting Q3 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
19	22	$\overline{Q2}$	Inverting Q2 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
20	23	Q2	Noninverting Q2 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
21	24	$\overline{Q1}$	Inverting Q1 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
23	26	Q1	Noninverting Q1 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
24	27	$\overline{Q0}$	Inverting Q0 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
25	28	Q0	Noninverting Q0 Output. Typically terminate with 50Ω resistor to V_{CC} - 2V.
26	1	V_{EE}	Negative Supply Voltage
27	2	CLK_SEL	Clock Select Input. When driven low, the CLK0 input is selected. Drive high to select the CLK1 Input. The CLK_SEL threshold is equal to V_{BB}. Internal 75kΩ pulldown to V_{EE}.
28	3	CLK0	Noninverting Differential Clock Input 0. Internal 105kΩ pulldown to V_{EE}.
Exposed	Exposed Pad	—	Internally Connected to V_{EE}

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

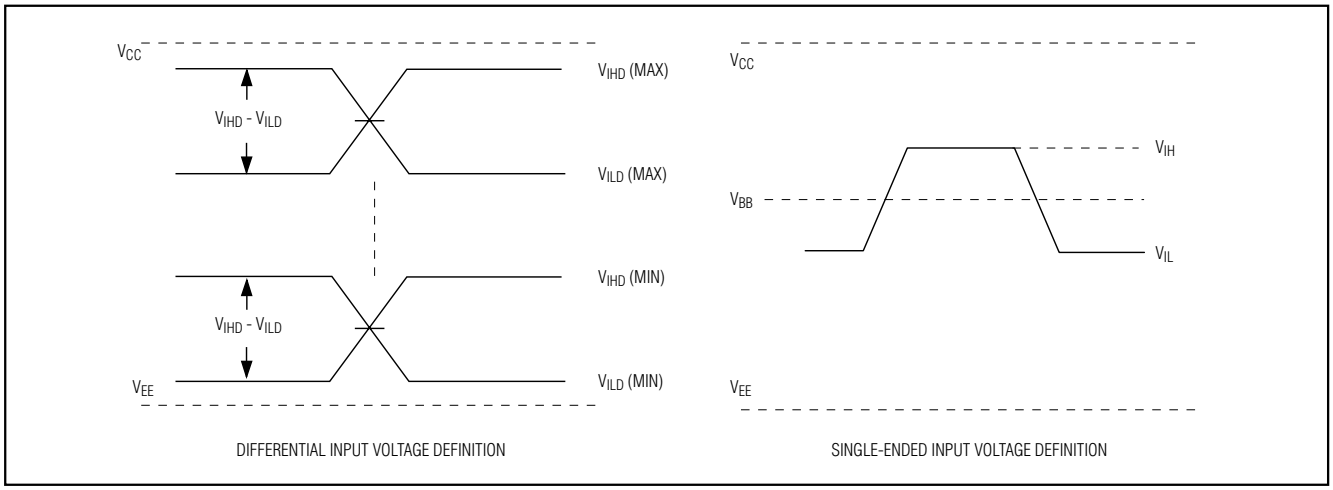


Figure 1. Input Voltage Definitions

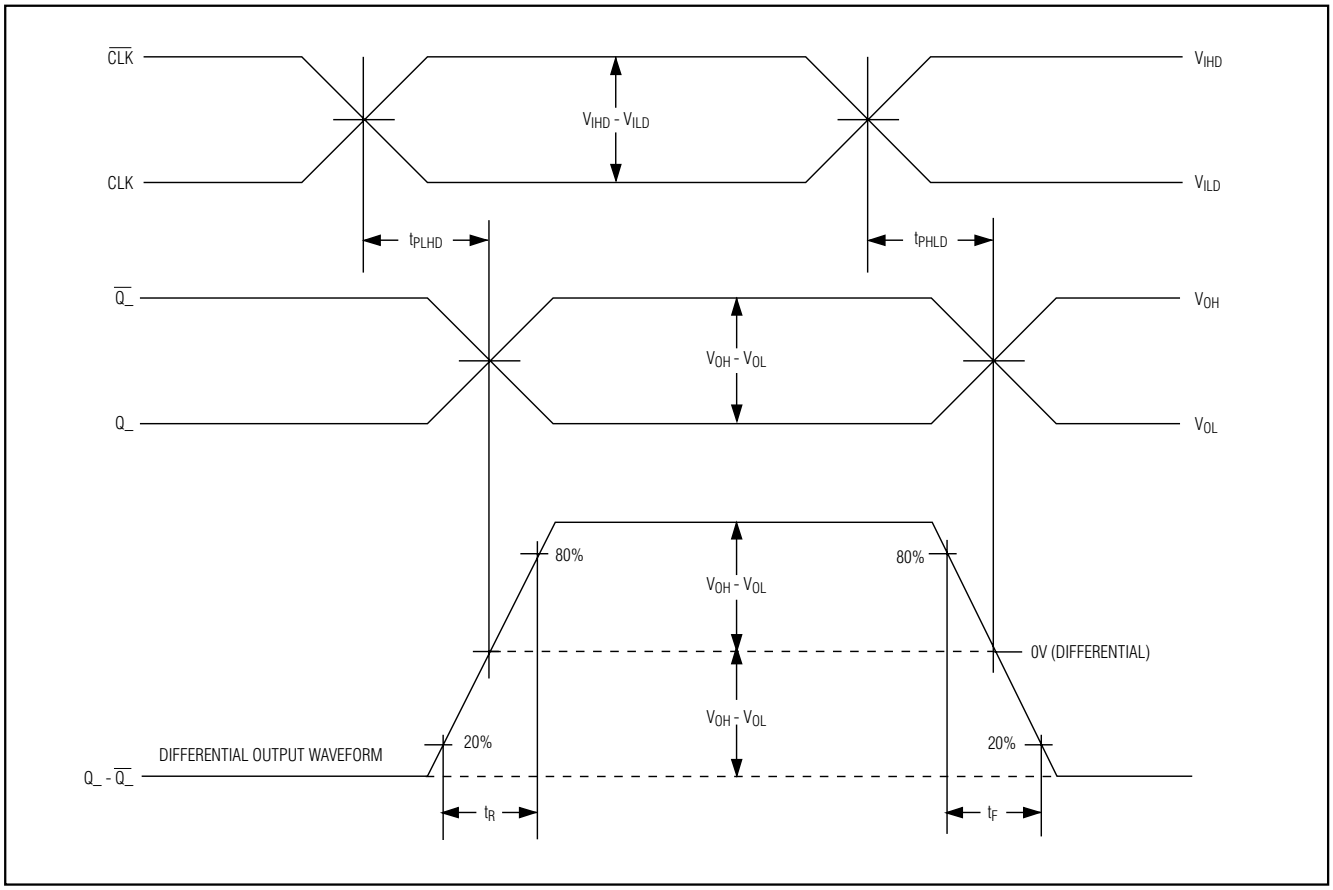


Figure 2. Differential Input (CLK_, $\overline{CLK\_}$) to Output (Q_, $\overline{Q\_}$) Delay Timing Diagram

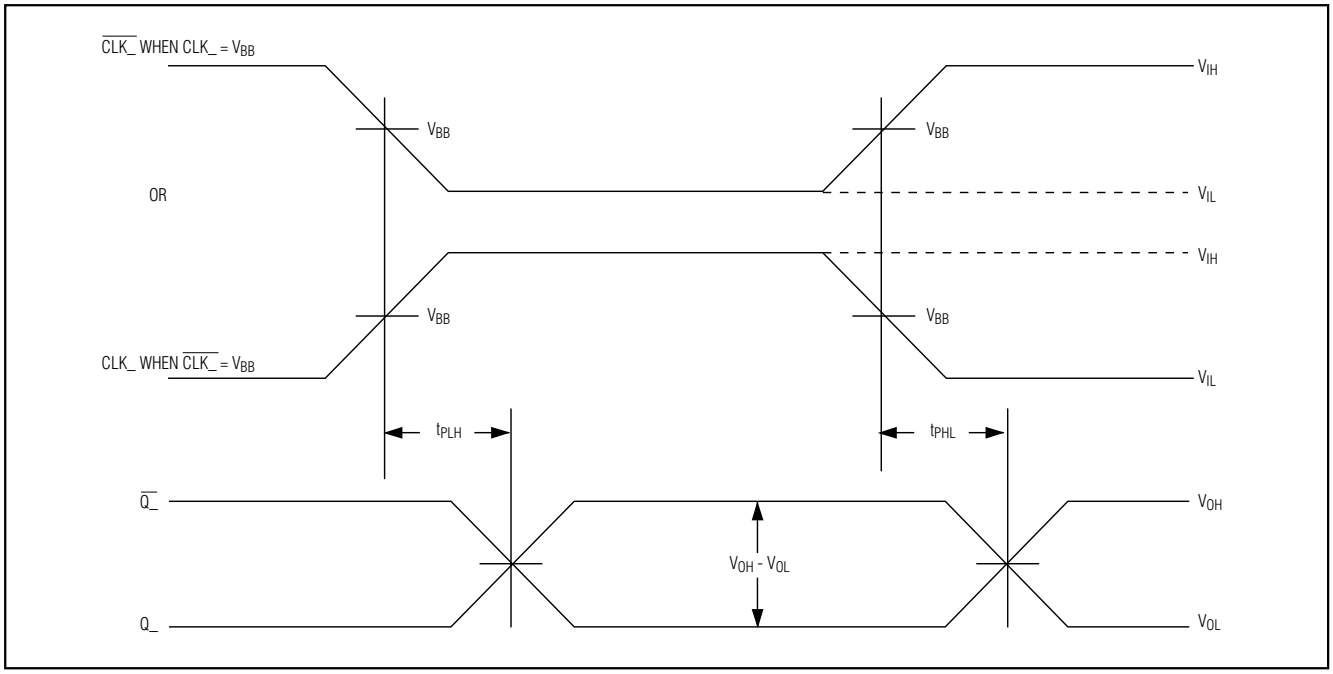


Figure 3. Single-Ended Input (CLK_, CLK_) to Output (Q_, Q_) Delay Timing Diagram

Detailed Description

The MAX9325 low-skew, 2:8 differential driver features extremely low output-to-output skew (50ps max) and part-to-part skew (225ps max). These features make the device ideal for clock and data distribution across a backplane or board. The device selects one of the two differential HSTL or LVECL/LVPECL inputs, and repeats them at eight differential outputs. Outputs are compatible with LVECL and LVPECL, and can directly drive 50Ω terminated transmission lines.

A 2:1 mux selects between the two differential inputs, CLK0, CLK0 and CLK1, CLK1. The 2:1 mux is switched by the single-ended CLK_SEL input. A logic low selects the CLK0, CLK0 input. A logic high selects the CLK1, CLK1 input. The logic threshold for CLK_SEL is set by an internal V_{BB} voltage reference. The selected input is reproduced at eight differential outputs at speeds up to 700MHz.

The differential inputs can be configured to accept a single-ended signal when the unused complementary input is connected to the on-chip reference output voltage (V_{BB}). A single-ended input of at least V_{BB} ±95mV or a differential input of at least 95mV switches the outputs to the V_{OH} and V_{OL} levels specified in the *DC Electrical Characteristics.* The maximum magnitude of the differential input from CLK_ to CLK_ is ±3.0V or ±(V_{CC} - V_{EE}), whichever is less. This limit also applies to the difference between a single-ended input and any reference voltage input.

The single-ended CLK_SEL input has a 75kΩ pulldown to V_{EE} that selects the default input, CLK0, CLK0, when CLK_SEL is left open or at V_{EE}. All the differential inputs have 105kΩ pulldowns to V_{EE}. Internal pulldowns and a fail-safe circuit ensure differential low default outputs when the inputs are left open or at V_{EE}.

Specifications for the high and low voltages of a differential input (V_{IHD} and V_{ILD}) and the differential input voltage (V_{IHD} - V_{ILD}) apply simultaneously.

For interfacing to differential HSTL and LVPECL signals, these devices operate over a +2.375V to +3.8V supply range, allowing high-performance clock or data distribution in systems with a nominal +2.5V or +3.3V supply. For differential LVECL operation, these devices operate from a -2.375V to -3.8V supply.

Single-Ended Operation

CLK_SEL is a single-ended input with the input threshold internally set to V_{BB}, and can be driven to V_{CC} or V_{EE} or by a single-ended LVPECL/LVECL signal. The CLK_, CLK_ are differential inputs but can be configured to accept single-ended inputs when operating at supply voltages greater than 2.58V. The recommended supply voltage for single-ended operation is 3.0V to 3.8V. A dif-

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

ferential input is configured for single-ended operation by connecting the on-chip reference voltage, V_{BB}, to an unused complementary input as a reference. For example, the differential CLK0, $\overline{CLK0}$ input is converted to a noninverting, single-ended input by connecting V_{BB} to $\overline{CLK0}$ and connecting the single-ended input to CLK0. Similarly, an inverting input is obtained by connecting V_{BB} to CLK0 and connecting the single-ended input to $\overline{CLK0}$. With a differential input configured as single-ended (using V_{BB}), the single-ended input can be driven to V_{CC} or V_{EE} or with a single-ended LVPECL/LVECL signal.

When configuring a differential input as a single-ended input, a user must ensure that the supply voltage (V_{CC} - V_{EE}) is greater than 2.58V. This is because the input high minimum level must be at (V_{EE} + 1.2V) or higher for proper operation. The reference voltage V_{BB} must be at least (V_{EE} + 1.2V) or higher for the same reason because it becomes the high-level input when the other single-ended input swings below it. The minimum V_{BB} output for the MAX9325 is (V_{CC} - 1.38V). Substituting the minimum V_{BB} output for (V_{BB} = V_{EE} + 1.2V) results in a minimum supply (V_{CC} - V_{EE}) of 2.58V. Rounding up to standard supplies gives the single-ended operating supply ranges (V_{CC} - V_{EE}) of 3.0V to 3.8V for the MAX9325.

When using the V_{BB} reference output, bypass it with a 0.01µF ceramic capacitor to V_{CC}. If not used, leave it open. The V_{BB} reference can source or sink 0.5mA, which is sufficient to drive two inputs.

Applications Information

Output Termination

Terminate the outputs through 50Ω to (V_{CC} - 2V) or use equivalent Thevenin terminations. Terminate each Q and $\overline{Q}$ output with identical termination on each for low output distortion. When a single-ended signal is taken from the differential output, terminate both Q_ and $\overline{Q}$_.

Ensure that output currents do not exceed the current limits as specified in the *Absolute Maximum Ratings* table. Under all operating conditions, the device's total thermal limits should be observed.

Supply Bypassing

Bypass each V_{CC} to V_{EE} with high-frequency surface-mount ceramic 0.1µF and 0.01µF capacitors. Place the capacitors as close to the device as possible with the 0.01µF capacitor closest to the device pins.

Use multiple vias when connecting the bypass capacitors to ground. When using the V_{BB} reference output, bypass it with a 0.01µF ceramic capacitor to V_{CC}. If the V_{BB} reference is not used, it can be left open.

Traces

Circuit board trace layout is very important to maintain the signal integrity of high-speed differential signals. Maintaining integrity is accomplished in part by reducing signal reflections and skew, and increasing common-mode noise immunity.

Signal reflections are caused by discontinuities in the 50Ω characteristic impedance of the traces. Avoid discontinuities by maintaining the distance between differential traces, not using sharp corners or using vias. Maintaining distance between the traces also increases common-mode noise immunity. Reducing signal skew is accomplished by matching the electrical length of the differential traces.

Exposed-Pad Package

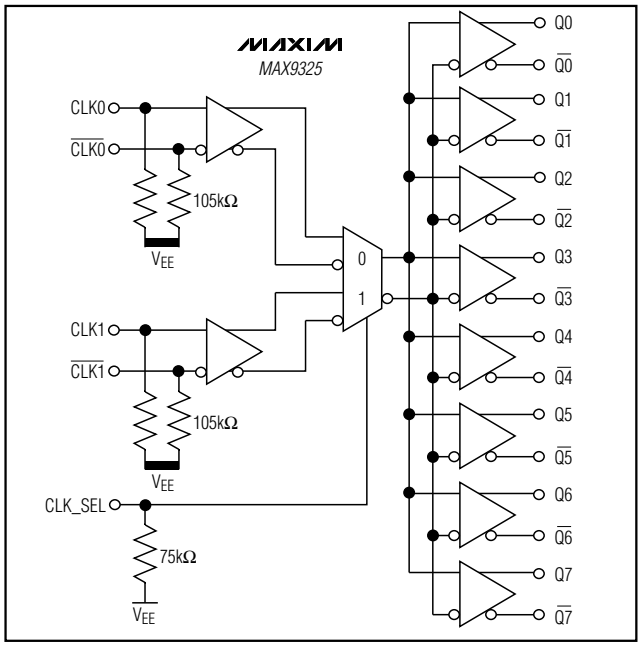
The 28-lead QFN package (MAX9325EGI) has the exposed paddle on the bottom of the package that provides the primary heat removal path from the IC to the PC board, as well as excellent electrical grounding to the PC board. **The MAX9325EGI's exposed pad is internally connected to V_{EE}. Do not connect the exposed pad to a separate circuit ground plane unless V_{EE} and the circuit ground are the same.**

Chip Information

TRANSISTOR COUNT: 1030

PROCESS: Bipolar

Functional Diagram

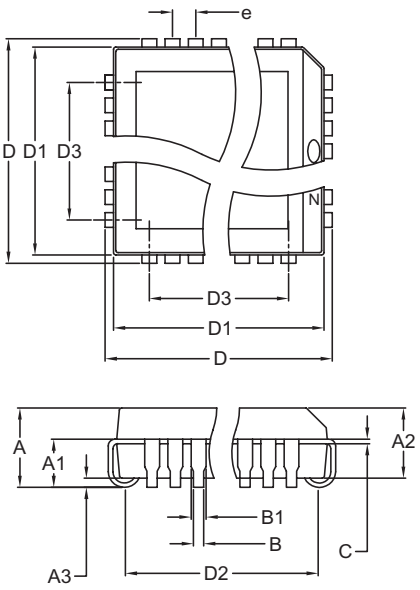


2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

Package Information

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information, go to **www.maxim-ic.com/packages**.)

PLCC.EPS



	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.165	0.180	4.20	4.57
A1	0.090	0.120	2.29	3.04
A2	0.145	0.156	3.69	3.96
A3	0.020	---	0.51	---
B	0.013	0.021	0.33	0.53
B1	0.026	0.032	0.66	0.81
C	0.009	0.011	0.23	0.28
e	0.050		1.27	

	INCHES		MILLIMETERS			
	MIN	MAX	MIN	MAX	N	MO047
D	0.385	0.395	9.78	10.03	20	AA
D1	0.350	0.356	8.89	9.04		
D2	0.290	0.330	7.37	8.38		
D3	0.200	REF	5.08	REF		
D	0.485	0.495	12.32	12.57	28	AB
D1	0.450	0.456	11.43	11.58		
D2	0.390	0.430	9.91	10.92		
D3	0.300	REF	7.62	REF		
D	0.685	0.695	17.40	17.65	44	AC
D1	0.650	0.656	16.51	16.66		
D2	0.590	0.630	14.99	16.00		
D3	0.500	REF	12.70	REF		
D	0.785	0.795	19.94	20.19	52	AD
D1	0.750	0.756	19.05	19.20		
D2	0.690	0.730	17.53	18.54		
D3	0.600	REF	15.24	REF		
D	0.985	0.995	25.02	25.27	68	AE
D1	0.950	0.958	24.13	24.33		
D2	0.890	0.930	22.61	23.62		
D3	0.800	REF	20.32	REF		

NOTES:

1. D1 DOES NOT INCLUDE MOLD FLASH.
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED .20mm (.008") PER SIDE.
3. LEADS TO BE COPLANAR WITHIN .10mm.
4. CONTROLLING DIMENSION: MILLIMETER
5. MEETS JEDEC MO047-XX AS SHOWN IN TABLE.
6. N = NUMBER OF PINS.

DALLAS SEMICONDUCTOR MAXIM

PROPRIETARY INFORMATION

TITLE: FAMILY PACKAGE OUTLINE: 20L, 28L, 44L, 52L, 68L PLCC

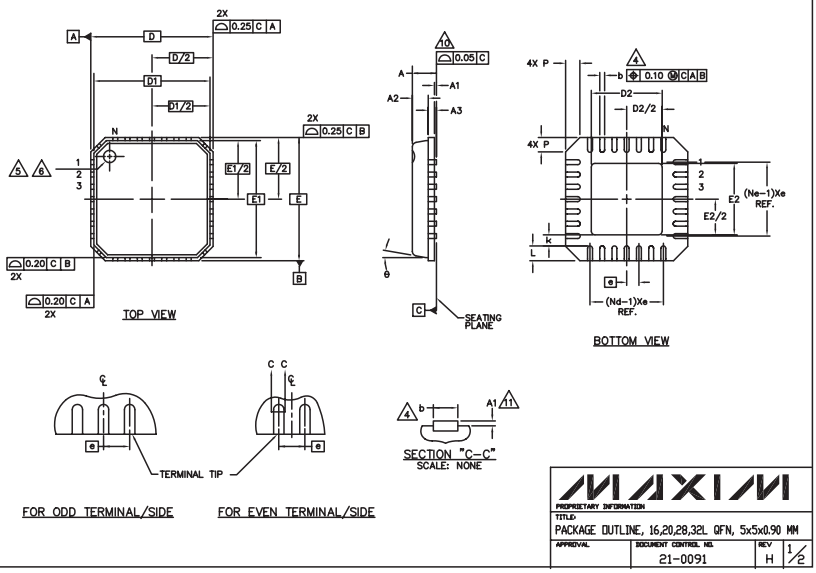
APPROVAL	DOCUMENT CONTROL NO.	REV.	
	21-0049	D	1/1

2:8 Differential LVPECL/LVECL/HSTL Clock and Data Driver

Package Information (continued)

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information, go to **www.maxim-ic.com/packages**.)

32L QFN.EPS



COMMON DIMENSIONS												
PKG	16L 5x5			20L 5x5			28L 5x5			32L 5x5		
SYMBOL	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	0.80	0.90	1.00	0.80	0.90	1.00	0.80	0.90	1.00	0.80	0.90	1.00
A1	0.00	0.01	0.05	0.00	0.01	0.05	0.00	0.01	0.05	0.00	0.01	0.05
A2	0.00	0.65	1.00	0.00	0.65	1.00	0.00	0.65	1.00	0.00	0.65	1.00
A3	0.20 REF			0.20 REF			0.20 REF			0.20 REF		
b	0.28	0.33	0.40	0.23	0.28	0.35	0.18	0.23	0.30	0.18	0.23	0.30
D	4.90	5.00	5.10	4.90	5.00	5.10	4.90	5.00	5.10	4.90	5.00	5.10
D1	4.75 BSC			4.75 BSC			4.75 BSC			4.75 BSC		
E	4.90	5.00	5.10	4.90	5.00	5.10	4.90	5.00	5.10	4.90	5.00	5.10
E1	4.75 BSC			4.75 BSC			4.75 BSC			4.75 BSC		
e	0.80 BSC			0.65 BSC			0.50 BSC			0.50 BSC		
k	0.25	–	–	0.25	–	–	0.25	–	–	0.25	–	–
L	0.35	0.55	0.75	0.35	0.55	0.75	0.35	0.55	0.75	0.30	0.40	0.50
N	16			20			28			32		
ND	4			5			7			8		
NE	4			5			7			8		
P	0.00	0.42	0.60	0.00	0.42	0.60	0.00	0.42	0.60	0.00	0.42	0.60
θ	0°		12°	0°		12°	0°		12°	0°		12°

EXPOSED PAD VARIATIONS						
PKG. CODES	D2			E2		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
G1655-3	2.95	3.10	3.25	2.95	3.10	3.25
G2055-1	2.55	2.70	2.85	2.55	2.70	2.85
G2055-2	2.95	3.10	3.25	2.95	3.10	3.25
G2855-1	2.55	2.70	2.85	2.55	2.70	2.85
G2855-2	2.95	3.10	3.25	2.95	3.10	3.25
G3255-1	2.95	3.10	3.25	2.95	3.10	3.25

NOTES:

1. DIE THICKNESS ALLOWABLE IS 0.305mm MAXIMUM (.012 INCHES MAXIMUM)
2. DIMENSIONING & TOLERANCES CONFORM TO ASME Y14.5M. – 1994.
3. N IS THE NUMBER OF TERMINALS.
   Nd IS THE NUMBER OF TERMINALS IN X-DIRECTION & Ne IS THE NUMBER OF TERMINALS IN Y-DIRECTION.
4. DIMENSION b APPLIES TO PLATED TERMINAL AND IS MEASURED BETWEEN 0.20 AND 0.25mm FROM TERMINAL TIP.
5. THE PIN #1 IDENTIFIER MUST BE EXISTED ON THE TOP SURFACE OF THE PACKAGE BY USING INDENTATION MARK OR INK/LASER MARKED.
6. EXACT SHAPE AND SIZE OF THIS FEATURE IS OPTIONAL.
7. ALL DIMENSIONS ARE IN MILLIMETERS.
8. PACKAGE WARPAGE MAX 0.05mm.
9. APPLIED FOR EXPOSED PAD AND TERMINALS.
   EXCLUDE EMBEDDED PART OF EXPOSED PAD FROM MEASURING.
10. MEETS JEDEC MO220.
11. THIS PACKAGE OUTLINE APPLIES TO ANVIL SINGULATION (STEPPED SIDES).

PROPRIETARY INFORMATION
TITLE:
PACKAGE OUTLINE, 16,20,28,32L QFN, 5x5x0.90 MM
APPROVAL
DOCUMENT CONTROL NO.
21-0091
REV
H
2/2

VTT009 (v1.2) September 28, 2000

Virtex™-E High-Performance Differential Solutions: Low Voltage Differential Signaling (LVDS)

Introduction

As the need for higher bandwidth accelerates, system designers are choosing differential signaling to satisfy high bandwidth requirements while reducing power, increasing noise immunity, and decreasing EMI/RFI emissions. LVDS is a low swing, differential signaling technology providing very fast data transmission, common-mode noise rejection, and low power consumption over a broad frequency range. The Virtex-E family delivers the programmable industry's highest bandwidth and most flexible differential signaling solution for direct interfacing to industry-standard LVDS devices.

With up to 36 I/O pairs operating at 622 Megabits per second (Mb/s) or up to 344 I/O pairs operating at over 311 Mb/s, the Virtex-E family supports multiple 10 Gb/s ports while maintaining high signal integrity with low power consumption. Unlike other PLD solutions, all Virtex-E LVDS I/Os support input, output, and I/O signaling, providing a system designer unparalleled flexibility in board layout. Figure 1 summarizes LVDS support in the Virtex-E family.

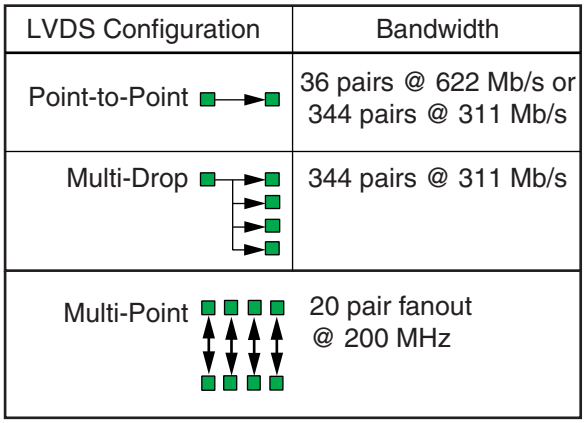


Figure 1: **Virtex-E High-Bandwidth LVDS Support Summary**

LVDS Standard

LVDS is defined by two industry standards: ANSI/TIA/EIA-644 and IEEE 1596.3 SCI-LVDS.

- The ANSI/TIA/EIA-644 standard defines LVDS electrical specs including driver output and receiver input electrical characteristics. It does not cover functional specifications, protocols, or transmission medium characteristics since these are application dependent. The ANSI/TIA/EIA-644 is the more generic of the two standards, and is intended for multiple applications.
- The IEEE 1596.3 SCI-LVDS standard is a subset of SCI (Scalable Coherent Interface). The SCI-LVDS standard defines electrical specifications for the physical layer interface of SCI. It is similar to the ANSI/TIA/EIA-644 standard but differs in the intended usage of the interface. The IEEE committee created the SCI-LVDS standard for communication between SCI nodes.

The Virtex-E LVDS solution conforms to the ANSI/TIA/EIA-644 standard. Table 1 summarizes the pertinent Virtex-E LVDS DC specifications.

Table 1: **LVDS DC Specifications**

DC Parameter	Conditions	Min	Typ	Max	Units
V_{CCO}		2.375	2.5	2.625	V
Output High Voltage for Q and $\overline{Q}$	R_T = 100 Ω across Q and $\overline{Q}$ signals	1.25	1.425	1.6	V
Output Low Voltage for Q and $\overline{Q}$	R_T = 100 Ω across Q and $\overline{Q}$ signals	0.9	1.075	1.25	V
Differential Output Voltage (Q - $\overline{Q}$), Q = High (($\overline{Q}$ - Q), $\overline{Q}$ = High	R_T = 100 Ω across Q and $\overline{Q}$ signals	250	350	450	mV
Output Common-Mode Voltage	R_T = 100 Ω across Q and $\overline{Q}$ signals	1.125	1.25	1.375	V
Differential Input Voltage (Q - $\overline{Q}$), Q = High (($\overline{Q}$ - Q), $\overline{Q}$ = High	Common-mode input voltage = 1.25 V	100	350	NA	mV
Input Common-Mode Voltage	Differential input voltage = ± 350 mV	0.2	1.25	2.2	V

Advantages

- LVDS is specified to be technology and process independent.
- LVDS is EMI/RFI tolerant. Common-mode noise affects both signals in a differential pair equally and is rejected by the receiver.
- LVDS produces low EMI/RFI because of its low swing and because of field cancellation between matched differential traces.
- No transmission medium is defined in the standard. The medium can be tailored to meet the specific application requirements.
- The typical LVDS voltage swing is 350 mV, resulting in a higher data transfer rate and lower CV^2f power consumption.

Configurations

There are three configurations that are used in LVDS applications, point-to-point, multi-point, and multi-drop. The Virtex-E family supports all LVDS configurations.

Point-to-Point

In point-to-point configuration, there is one transmitter and one receiver. The LVDS driver is a current source that drives a differential pair of lines. The typical current drive is 3.5 mA. The receiver has high DC impedance. The majority of the driver current flows across the receiver termination resistor generating about 350 mV at the receiver inputs (Figure 2).

Multi-Drop

A multi-drop LVDS configuration has one transmitter and multiple receivers. The differential termination resistor is placed close to the last receiver (Figure 3).

Multi-Point

Multi-point LVDS configurations feature multiple transmitters and receivers. Frequently separate data channels come from separate clock domains. Virtex-E excels in multi-domain clock support.

Applications

Applications for LVDS include:

- Switches
- Repeaters
- Hubs
- Routers

- Wireless base stations
- Flat panel displays
- Digital cameras
- Printers
- Copiers
- Multimedia peripherals
- Backplane applications

Terminations

LVDS is widely used for high-speed point-to-point interface as well as multi-drop and multi-point applications. Depending on the exact interconnect topology, precision resistors are required to match specific impedance characteristics to minimize reflection and ensure high signal integrity. The Virtex-E family supports the most flexible LVDS high-speed interface by supporting a flexible external termination scheme. This enables system designers select resistor values most appropriate for maximum performance.

Point-to-Point

Figure 2 shows the schematic of a standard LVDS driver driving the Virtex-E receiver. An LVDS driver drives the two 50 Ω transmission lines into a Virtex-E LVDS receiver. The two 50 Ω single-ended transmission lines can be microstrip, stripline, a 100 Ω differential twisted pair, or a similar balanced differential transmission line.

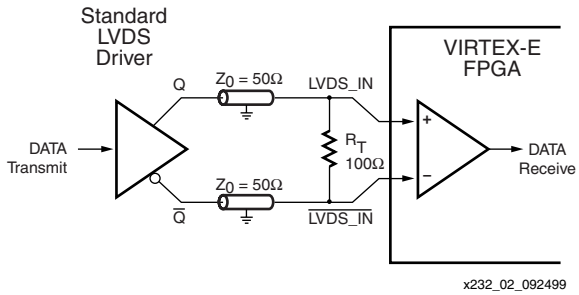


Figure 2: **A Standard LVDS Driver Driving a Virtex-E LVDS Receiver**

Figure 3 shows the complete schematic of the Virtex-E LVDS line driver and receiver. The standard LVDS 100 Ω termination resistor is connected across the LVDS_OUT and $\overline{LVDS\_OUT}$ outputs at the end of the transmission line. The resistors R_S and R_{DIV} attenuate signals from the Virtex-E LVDS drivers and provide a matched source impedance (series

termination) to the transmission lines. Standard termination packs are available from **Bourns**. Other resistor vendors provide termination networks with up to 16-pins per pack.

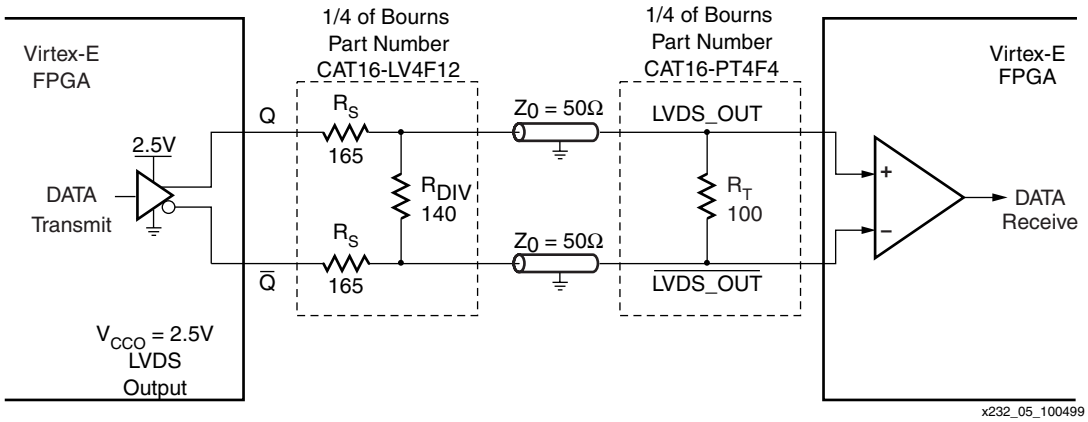


Figure 3: **Virtex-E LVDS Line Driver and Receiver Schematic**

Virtex-E LVDS driver meets ANSI/TIA/EIA-644 LVDS DC specifications. The matched source impedance of the Virtex-E LVDS driver absorbs nearly all differential reflections from the capacitive load at the LVDS destination, reducing standing waves, undershoot, and signal swing on data bursts or clocks.

Data and clocks can be transmitted over cables longer than 5 ns electrical length, limited only by the quality of the cable (the cable attenuation caused by skin effect losses at high frequencies).

The 622 Mb/s data rate, or 311 MHz clock is achievable with a Virtex-E -7 speed grade devices. See "**XAPP233: LVDS Transceivers at 622 Mb/s using General-Purpose I/O**" for details of the reference design.

Multi-Drop

Multi-drop LVDS configuration allows many receivers to be driven by one Virtex-E LVDS driver. With simple source and differential termination, Virtex-E LVDS driver can drive lines with fan-outs of 20 to 1, making Virtex-E LVDS I/Os suitable for a broad variety of high-load applications.

Figure 4 illustrates a Virtex-E LVDS driver driving 20 LVDS receivers in a multi-drop configuration. The receivers can be either Virtex-E receivers or other off-the-shelf LVDS receivers. The LVDS signal is driven from a Virtex-E LVDS driver, and is daisy-chained with two 29 Ω transmission lines and stubs to all 20 LVDS receivers. Each LVDS receiver is connected to the main multi-drop lines every 2.5" for a multi-drop line length of 50". Each LVDS receiver tap line has a 1" maximum stub length with a 50 Ω transmission line impedance to ground, or a differential impedance of 100 Ω between the two stubs. A 44 Ω termination resistor R_T is placed across the differential lines close to the last LVDS receiver. Resistors R_S and R_{DIV} attenuate the signals from the Virtex-E drivers and provide a 22 Ω source impedance (series termination) to the 29 Ω transmission lines. The 22 Ω source impedance is used because the added load of the LVDS receivers brings the 29 Ω line down to an effective average impedance

of 22 Ω. The capacitor C_{SLEW} reduces the slew rate from the Virtex-E LVDS driver, resulting in smaller reflections and less ringing at the receivers.

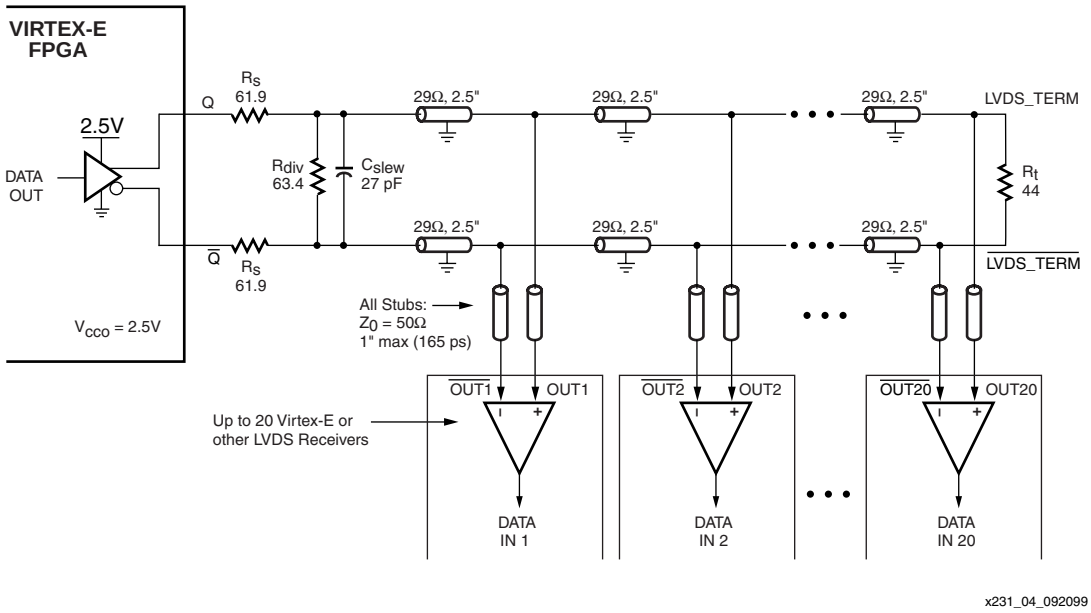


Figure 4: **Virtex-E 20-load Multi-Drop LVDS Schematic**

Multi-Point

Figure 5 shows a 20-load multi-point LVDS schematic using Virtex-E devices. Each Virtex-E BLVDS transceiver is uniformly spaced two inches apart on a 40-inch multi-point length line. Each BLVDS transceiver tap line has 1.5-inch maximum stub length. The transceivers in Figure 5 have an effective load capacitance of roughly 8 pF, including receiver capacitance, trace and stub capacitance. When all 20 Virtex-E BLVDS transceivers are fully populated, the effective impedance of the fully loaded backplane is calculated using Equation 1 as 70 ohms. The effective impedance is the same as in the case of the 10-load scenario because the spacing between the stubs and the stub-lengths have not increased. An 80-ohm termination resistor is placed across the two lines at each end of the backplane to approximately match the effective impedance of the fully loaded line and to provide a differential voltage swing of approximately 250 mV at the receivers. A 50-ohm series resistor on each signal line, next to the Virtex-E transceivers, reduces the magnitude of the reflections (which can be caused by the long stubs) and attenuates the signals coming out of the Virtex-E devices.

The two 29 Ω single-ended transmission lines can be microstrip, stripline, the single-ended equivalent of a 58 Ω twisted pair, or a similar balanced differential transmission line. The resistors R_S and R_{DIV} should be placed close to the Virtex-E driver outputs. The parallel termination resistor R_T should be placed close to the last LVDS receiver inputs at the far end of the multi-drop line. The capacitor C_{SLEW} should be placed close to the resistors R_S and R_{DIV}.

The Virtex-E multi-drop LVDS driver adheres to all the ANSI/TIA/EIA-644 LVDS standard DC input level specifications, and is fully compatible with LVDS receivers from National Semiconductor and other companies.

The maximum data rate is 311 Mb/s or a clock of 155.5 MHz for a Virtex-E -7 speed grade device. Reliable data transmission is possible for up to 20 LVDS receivers over a multi-drop line length of 50 inches, limited only by skin effect losses in the PCB trace.

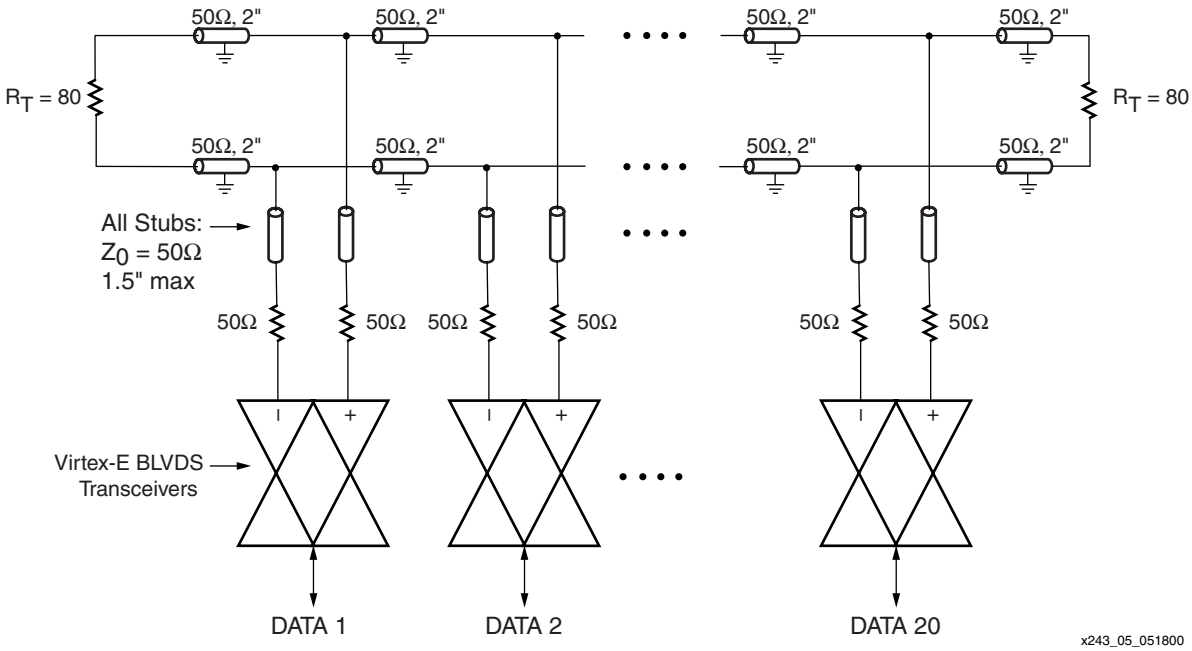


Figure 5: **20-Load Multi-Point LVDS Using Virtex-E Devices**

Waveform

A typical LVDS output waveform for Virtex-E devices is shown in Figure 6.

± 175 mV Swing
@ 1.25 V mid-point.

Computed Signal
Differential (Q - $\overline{Q}$)

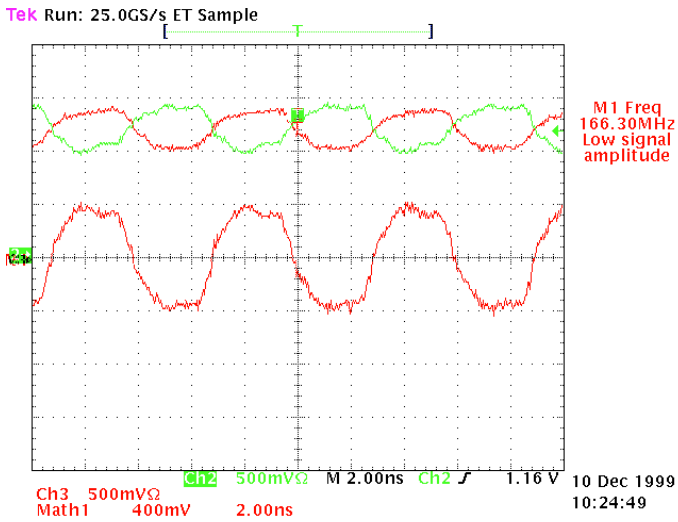


Figure 6: **LVDS Output Waveform**

Virtex Advantages

The Virtex-E devices are the first programmable logic devices available in the market incorporating advanced LVDS I/O capability with support for other differential standards (Bus LVDS and LVPECL). Unlike other announced architectures (for example, APEX E), the Virtex-E LVDS capability provides an abundance of LVDS-capable user I/O and clock pins, and the architectural flexibility shown in Table 2 to address true high-speed system issues. This

capability works in concert with a robust delay locked loop (DLL) technology enabling designers to achieve maximum performance in their LVDS applications.

Table 2: **Virtex-E High-Bandwidth LVDS Solution Summary**

Feature	Virtex-E	APEX E
LVDS offered as a standard feature	Yes, in all devices, packages, and speeds	Large “X” device only, fast speed only (More $$)
LVDS	Y	Y
Bus LVDS	Y	N
LVPECL	Y	N
LVDS Configurations	Point-to-point, Multi-drop, and Multi-point	Point-to-point and Multi-drop [1]
Maximum I/O Bandwidth	22 Gb/s (622 Mb/s/pair x 36 pairs) or 107 Gb/s (311 Mb/s/pair x 344 pairs)	10 Gb/s (622 Mb/s x 16 pairs)
Termination	Flexible External Termination	Inflexible Internal Termination on Outputs
High-Speed Differential Clock Pairs	4	1
Maximum # of Differential Pairs	344 In/Out	Dedicated 16 input and 16 output pairs. Not layout friendly.
Maximum Speed	622 Mb/s	622 Mb/s
Serializer/Deserializer	Flexible in CLB	Dedicated 8:1
Clock Recovery	N	N

Unlike other PLD solutions that only offer LVDS capability in the most expensive and highest speed grade options, the Virtex-E DLL and LVDS I/O capabilities are standard features available in all Virtex-E device/package combinations. The Virtex-E family offers the option to use up to 36 LVDS I/O pairs operating at 622 Mb/s or up to 344 LVDS I/O pairs operating at over 311 Mb/s to achieve over 100 Gb/s aggregate bandwidth. This enables system designers to support multiple 10 Gb/s ports architecture for high-performance DSP and data communication systems. Table 3 demonstrates the high-bandwidth LVDS solution provided by the Virtex-E family. In addition to offering a high-performance and highly flexible LVDS solution, Xilinx also works closely with other component vendors (e.g., Bourn for the resistor pack) to ensure inter-operability and help system designers further reduce the overall design complexity and system cost.

Table 3: **Virtex-E High-Bandwidth LVDS Solution Summary**

I/O Standard	Type	1	2	32	72	688
Virtex-E LVDS	Differential	NA	622 Mb/s	10 Gb/s	22 Gb/s	107 Gb/s
APEX E LVDS	Differential	NA	622 Mb/s	10 Gb/s	NA	NA

References

Standards

- ANSI/TIA/EIA-644 http://www.eia.org/eng
- IEEE1596.3 http://www.ieee.org

Related Xilinx Documents

- XAPP230: “The LVDS I/O Standard” at: http://www.xilinx.com/xapp/xapp230.pdf
- XAPP231: “Multi-Drop LVDS” at: http://www.xilinx.com/xapp/xapp231.pdf

- XAPP232: “The LVDS Drivers and Receivers: Interface Guidelines” at:
  http://www.xilinx.com/xapp/xapp232.pdf
- XAPP233: “LVDS Transceivers at 622 Mb/s using General Purpose I/O” at:
  http://www.xilinx.com/xapp/xapp233.pdf
- XAPP243: “Bus LVDS with Virtex-E Devices” at:
  http://www.xilinx.com/xapp/xapp243.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/14/00	1.0	Initial Xilinx release.
08/21/00	1.1	Revisions made.
09/28/00	1.2	Minor changes.

I²C-Bus—Interface with HPC

National Semiconductor
Application Note 561
Hubert Utz
November 1989

INTRODUCTION

There are many applications in which microcontrollers are used as a central processor. These systems are designed with the following aspects:

— reduce and minimize system costs

— provide system flexibility

— simple connections to other peripheral devices (no high speed requirements)

A serial bus structure fulfills the above subjects.

The National Semiconductor microcontroller family provides the MICROWIRE/PLUS™ interface as a synchronous serial line to communicate with peripherals.

Another important serial bus is the I²C-Bus (Inter IC-Bus) which was developed by Valvo/Philips. It is mainly used in the customer area. This article describes a simple I²C-Bus interface with National's microcontroller family HPC 16xxx and two different software routines to work the interface:

a. Softwarepolling

b. Using the MICROWIRE™ shift register

THE I²C-Bus

The I²C-Bus is a bidirectional two line serial communication bus. The two wires, SDA (serial data) and SCL (serial clock) carry information between the different devices connected to the bus.

The devices can operate either as a receiver or a transmitter, depending on their functions.

The I²C-Bus also supports multimaster mode. Each device has its own 7-bit address.

This address consists commonly of a fixed hardwired part (4 Bits chip intern) and a variable address part (3 Pins of the device).

The I²C-Bus is based on the following definitions:

—TRANSMITTER:	the device which sends the data to the serial data line
—RECEIVER:	the device which receives the data from the serial data line
—MASTER:	the device which starts a transfer, supplies the clock signals and terminates a current transfer cycle
—SLAVE:	the device which is addressed by the master
—MULTIMASTER:	more than one device can get the master to control the serial data bus and the serial clock bus
—ARBITRATION:	if more than one device simultaneously tries to control the bus, a simple arbitration procedure takes place, so that only one device can get the master
—SYNCHRONIZATION:	procedure to synchronize the clock signals of two or more devices (slow slaves)

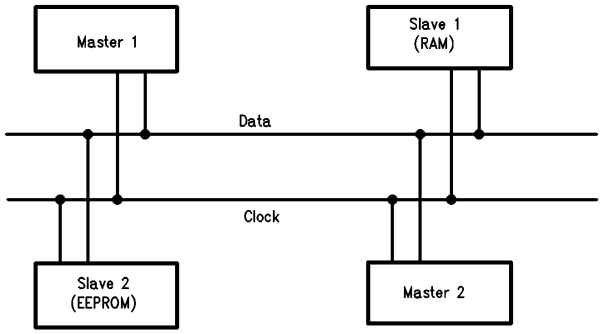
The maximum transmission rate is 100 kbit/s.

The maximum number of devices connected to the bus is limited by the maximum bus capacitance of 400 pF (typical device capacitance 10 pF).

Start-and Stop Conditions

The bus is not busy if both data- and clock lines remain HIGH because there are only two lines available, the start- and stop conditions have special timing definitions between these two lines:

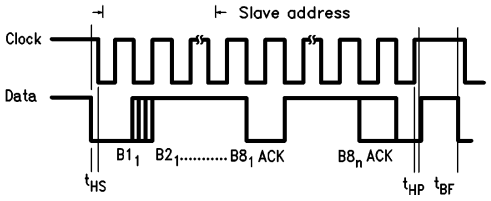
—start conditions:	HIGH-to-LOW transition of the data line, while the clock line is in a HIGH state.
—stop conditions:	LOW-to-HIGH transition of the data line, while the clock line is in a HIGH state.



TL/DD/10080–1

FIGURE 1. I²C-Bus Configurations

RRD-B30M105/Printed in U. S. A.



TL/DD/10080–2

Startcondition

— Clock-, Dataline high (Bus free)

— change Dataline from high to low level

— after $t_{HS\ Min} = 4\ \mu s$ the master supplies the clock

Acknowledge

— transmitting Device releases the Dataline

— the receiving Device pulls the Dataline low during ACK-clock if there is no error

— if there is no ACK the master will generate a Stopcondition to abort the transfer

Stopcondition

— clockline goes high

— after $t_{HP\ Min} = 4.7\ \mu s$ datalines goes high

— the master remains the Data-, Clockline high

— next Startcondition after $t_{FB\ Min} = 4.7\ \mu s$ possible

FIGURE 2. I²C-Bus Timing

The master always generates the start and stop conditions. After the start condition the bus is in the busy state. The bus becomes free after the stop condition.

DATA BIT TRANSFER

After a start condition 'S' one databit is transferred during each clock pulse. The data must be stable during the HIGH-period of the clock. The data line can only change when the clock line is at a LOW level.

Normally each data transfer is done with 8 data bits and 1 acknowledge bit (byte format with acknowledge).

ACKNOWLEDGE

Each data transfer needs to be acknowledged. The master generates the acknowledge clock pulse. The transmitter releases the data line (SDA = HIGH) during the acknowledge clock pulse. If there was no error detected, the receiver will pull down the SDA-line during the HIGH period of the acknowledge clock pulse.

If a slave receiver is not able to acknowledge, the slave will keep the SDA line HIGH and the master can then generate a STOP condition to abort the transfer.

If a master receiver keeps the SDA line HIGH, during the acknowledge clock pulse the master signals the end of data transmission and the slave transmitter release the data line to allow the master to generate a STOP- condition.

ARBITRATION

Only in multi master systems.

If more than one device could be master and more than one wants to access the bus, an arbitration procedure takes place: if a master transmits a HIGH level and another master transmits a LOW level the master with the LOW level will get the bus and the other master will release the bus and the clockline immediately and switches to the slave receiver mode. This arbitration could carry on through many bits (address bits and data bits are used for arbitration).

FORMATS

There are three data transfer formats supported:

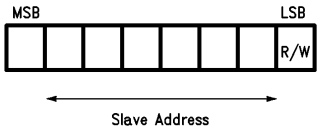
— master transmitter writes to slave receiver; no direction change

— master reads immediate after sending the address byte

— combined format with multiple read or write transfers (see . . .)

ADDRESSING

The 7-bit address of an I²C device and the direction of the following data is coded in the first byte after the start condition:

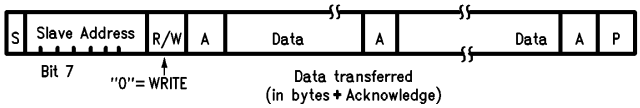


TL/DD/10080–3

A "0" on the least significant bit means that the master will write information to the selected Slave address device: a "1" means that the master will read data from the slave.

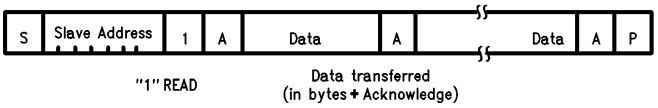
Some slave addresses are reserved for future use. These are all addresses with the bit combinations 1111XXX and 0000XXX. The address 00000000 is used for a general call address, for example to initialize all I²C devices (refer to I²C bus specification for detailed information).

—Master Transmits to Slave, No Direction Change



TL/DD/10080–4

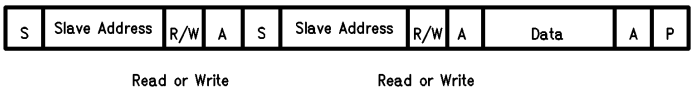
—Master Reads Slave Immediately after First Byte



TL/DD/10080–5

The master becomes a master receiver after first ACK

—Combined Formats



TL/DD/10080–6

nbytes Data + ACK nbytes Data + ACK

S = Startcondition A = Acknowledge P = Stopcondition

FIGURE 3. I²C-Bus Transfer Formats

TIMING

The master can generate a maximum clock frequency of 100 kHz. The minimum LOW period is defined with 4.17 μs, the minimum HIGH period width is 4 μs, the maximum rise time on SDA and SCL is 1 μs and the maximum fall time on SDA and SCL is 300 ns.

Figure 4 shows the detailed timing requirements.

Symbol	Parameter	Min	Max	Units
f_{SCL}	SCL Clock Frequency	0	100	kHz
t_{BUF}	Time the Bus Must Be Free before a New Transmission Can Start	4.7		μs
t_{HD}; STA	Hold Time Start Condition. After This Period the First Clock Pulse Is Generated	4.0		μs
t_{LOW}	The LOW Period of the Clock	4.7		μs
t_{SU}; STA	Setup Time for Start Condition (Only Relevant for a Repeated Start Condition)	4.7		μs
t_{HD}; DAT	Hold Time DATA for CBUS Compatible Masters (See Also NOTE, Section 8.1.3.) for I²C Device	5 0*		μs μs
t_{SU}; DAT	Setup Time Data	250		ns
t_R	Rise Time of Both SDA and SCL Lines		1	μs
t_F	Fall Time of Both SDA and SCL Lines		300	ns
t_{SU};STO	Setup Time for Stop Condition	4.7		μs

All values referred to $V_{IH\ Min}$ = 3.0V and $V_{IL\ Min}$ = 1.5V levels at 5V supply voltage

*Note that a transmitter must internally provide at least a hold time to bridge the undefined region (max. 300 ns) of the falling edge of SCL.

FIGURE 4. I²C-Bus Timing Requirements

—Two Wire Serial Bus with
- *Data line
- *Clock line

—Features:
- *Multimaster Bus (Master/Slave)
- *Busarbitration
- *Transfer rate up to 100 kbits/s
- *Bytetransfers
- *Protocols with
  - —Start Condition
  - —Address
  - —Ackn.
  - —Data
  - —Ackn. (Each Byte)
  - —Stop Condition
- *Read, Write, Multiple R/W

FIGURE 5. I²C-Bus Features

The I²C test hardware uses the following components:

2 x PC F 8570: 256 x 8-Bit RAM
- RAM 1: Address: 1010000X
- RAM 2: Address: 1010010X

2 x PC F 8582: 256 x 8 Bit EEPROM
- EEPROM 1: Address: 1010001X
- EEPROM 2: Address: 1010011X

2 x PC F 8574: Remote 8-Bit I/O Expander
- I/O 1: Address: 0100000X Used as 8-Bit LED Output Port
- I/O 2: Address: 0100001X Used as 8-Bit Input Port

1 x HCT04:	Inverter
1 x LS05:	Inverter, Open Collector
8 x LEDs:	Connected Via Pull Up Resistors to Output Pins of PCF 8574
2 x Rp:	Pull Up Resistors for Clock Line and Data Line
8 Switches:	Connected Via Pull Up Resistors to Input Pins of PCF 8574
1 x Pin Grid Socket:	Socket for MOLE™ Connection
1 x Power Connector:	5V Power Supply

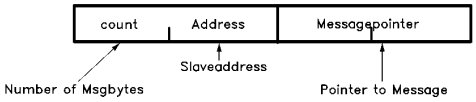
Four I/O lines of the HPC are used to connect a HPC-MOLE or a HPC-Designer Kit to the I²C-board: SO, SI, P0, and SK. SO drives the data bus line; SDA and P0 drive the clock bus line SCL.

The data on the SDA line is monitored by the input SI and the I²C-bus clock is available at input SK.

SI, SO and SK are μWire interface lines.

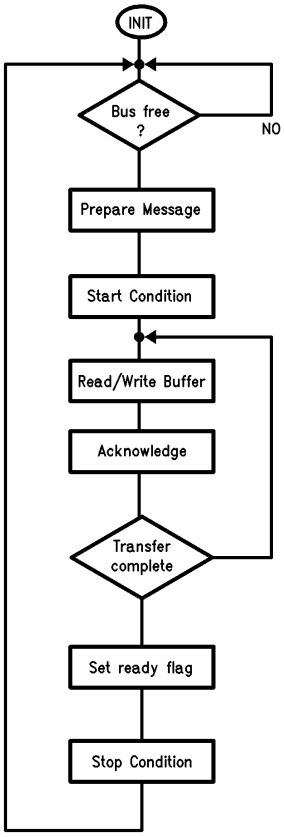
P0 is used as a continuous timer output during the transfer. All rise and fall times meet the I²C-bus specification. The highest I²C-clock frequency you can get with a 17 MHz HPC oscillator/4 Waitstates is ca. 20 kHz.

I²C-Bus Software HPC
- *Master only Mode
- *Tested I²C-Clock Frequency 16 kHz–20 kHz
- *3 Possible Formats Supported
  - —Read
  - —Write
  - —Multiple Read or Write
- *Two Program Versions
  - —Programmed I/O
  - —Interrupt Driven I/O
- *Demo Loop:
  - —Write Output
  - —RAM test
  - —Read Input
  - —Increment Output or Set Output = Input
- *IRQ Driven Program Uses μWire Shift register
- *Message Field:



TL/DD/10080–7

Figure 7. Software Features



TL/DD/10080–8

FIGURE 8. Programmed I/O Flowchart

```
        ;*****************************************************
        ;*  INTER-IC-BUS (I2C-BUS) WITH HPC  : APPL.NOTE
        ;*****************************************************

        ;* 12.10.87                                    HU
        ;* 20.10.87                                    HU

        .INCLD  HPC16083.MAP    ; MEMORY MAP FOR HPC16083

        ;DEFINITIONS
        CLK = 32                ;CLOCK LOW/HIGH TIME = 33us

        .MACRO SAVE_AB

        PUSH    A               ;SAVE A-REG
        PUSH    B               ;SAVE B-REG

        .ENDM

        .MACRO SAVE_ABX

        SAVE_AB                 ;SAVE REGS A,B
        PUSH    X               ;SAVE X-REG

        .ENDM

        .MACRO RESTORE_AB

        POP     B               ;RESTORE B-REG
        POP     A               ;RESTORE A_REG

        .ENDM

        .MACRO RESTORE_ABX

        POP     X               ;RESTORE X-REG
        RESTORE_AB              ;RESTORE REGS B,A

        .ENDM

        .SECT   B1,BASE         ;DEFINE BASEPAGE SECTION

WBUF1:  .DSW    1               ;WORDBUFFER FOR I2C-TABLE
WBUF2:  .DSW    1               ;WORDBUFFER FOR I2C-TABLE
INDEX:  .DSW    1               ;POINTER TO THE NEXT TABLEENTRY
STATUS: .DSB    1               ;STATUSBYTE
DUMMY:  .DSB    1               ;DUMMY BYTE
WPORT:  .DSB    1               ;8-BIT VALUE WRITE TO PORT0
RPORT:  .DSB    1               ;8-BIT VALUE READ FROM PORT1
WRBUFF: .DSB    10              ;RAM WRITE BUFFER
RDBUFF: .DSB    10              ;RAM READ BUFFER

        .ENDSECT

        .SECT   I2C,ROM16
```

TL/DD/10080–9

```
        ;*************************************************
        ;* INIT : THIS SUBROUTINE INITIALIZES TIMER T4, TIMER T5
        ;*        AND THE uWIRE-INTERFACE TO OPERATE AS I2C-BUS
        ;*
        ;* INPUT : NONE
        ;* OUTPUT : NONE
        ;* USED REGS : A, B, X   ( ALL REGS ARE SAVED )
        ;*************************************************

INIT:   SAVE_ABX                ;SAVE REGS A,B,X
        LD      X,#PWMODE       ;ADDR. PWMODE-REG -> X
        LD      B,#PORTP        ;ADDR. PORTP-REG -> B
        LD      A,#0C           ;VALUE TO STOP TIMER
        ST      A,[X].B         ;STOP T4, NO IRQ, ACK TIP-FLAG
        NOP
        ST      A,[X].B         ;MAKE SHURE TIP-FLAGS ARE CLEARED
        LD      A,#01           ;DISABLE TOGGLE AND SET OUTPUT HIGH
        ST      A,[B].B         ;ON PINS PO AND P1
        SBIT    3,[B].B         ;TOGGLE ON AT P0
        LD      T4.W,#CLK-1     ;LOAD T4 (33us)
        LD      R4.W,#CLK-1     ;LOAD R4 (33us)
        SBIT    5,PORTB.B       ;DATALINE OUTPUT = HIGH
        SBIT    5,DIRB.B        ;B5 = OUTPUT
        RBIT    5,BFUN.B        ;NO ALTERNATE FUNCTION SELECTED
        RBIT    6,DIRB.B        ;B6 = INPUT
        RBIT    6,BFUN.B        ;
        RESTORE_ABX             ;RESTORE REGS X,B,A
        RET


RWI2C:  PUSH    K               ;SAVE REG K
        SAVE_ABX                ;SAVE-REGISTER
        LD      B,#PORTB        ;ADDRESS OF PORTB -> REG B
RWRST:  LD      STATUS.B,#0     ;RESET STATUSBYTE
        JSR     TSTBUS          ;BUS FREE ?
        IFC
        JP      RWERR           ;IF ERROR -> EXIT
        LD      A,[X+].W        ;GET 2 BYTES OF TABLE
        ST      A,WBUF1.W       ;SAVE TABLE CONTENTS
        IFBIT   0,A             ;TEST RECEIVE/TRANSMIT BIT
        JP      RWRECV          ;BIT = 1 -> RECEIVE
        RBIT    0,STATUS.B      ;STATUS = TRANSMIT
        JP      RW01            ;CONTINUE AT RW01
RWRECV: SBIT    0,STATUS.B      ;STATUS = RECEIVE
RW01:   SBIT    1,STATUS.B      ;STATUS = FIRST BYTE
        LD      A,[X+].W        ;GET NEXT 2 BYTES OF TABLE
        ST      A,WBUF2.W       ;SAVE TABLE CONTENTS
        LD      A,X
        ST      A,INDEX.W       ;SAVE INDEX
        LD      A,[X].W         ;GET NEXT WORD OF TABLE
        IFEQ    A,#0            ;ANY MORE TO TRANSFER
        JP      RW02            ;NO, EXIT
        SBIT    3,STATUS.B      ;STATUS = MULTISTART
RW02:   SBIT    7,STATUS.B      ;STATUS = BUSY
```

TL/DD/10080–10

```
        RC                      ;CLR CARRY = NO ERROR

        JSR     STRTCD          ;STARTCONDITION
        IFC
        JP      STPERR
        LD      X,WBUF2.W       ;X = BUFFERINDEX
        LD      A,WBUF1.W       ;A.B = ADDRESS
        JSR     TRANSF          ;TRANSMITT 1.BYTE = ADDRESS
        IFC
        JP      STPERR
        DECSZ   WBUF1+1.B       ;DECREMENT BYTECOUNT
        NOP                     ;DUMMY FOR DECSZ
        IFBIT   0,STATUS.B      ;STATUS = RECEIVE ?
        JP      RCVE            ;YES -> RCVE
TRMIT:  LD      A,[X+].B        ;GET NEXT BYTE
        DECSZ   WBUF1+1.B       ;DECREMENT BYTECOUNT
        JP      TRM1            ;BYTECOUNT <> 0
        SBIT    2,STATUS.B      ;FLAG LAST BYTE
TRM1:   JSR     TRANSF          ;SEND BYTE
        IFC                     ;TEST ERROR
        JP      STPERR          ;ERROR DETECTED
        IFBIT   2,STATUS.B      ;LAST BYTE ?
        JP      TRM2            ;YES
        JP      TRMIT           ;NO -> TRANSFER AGAIN
TRM2:   IFBIT   3,STATUS.B      ;MULTISTART ?
        JP      TRM3            ;YES
        JP      TRM6            ;NO -> STOPCONDITION
TRM3:   SBIT    5,[B].B         ;DATALINE = HIGH
TRM4:   IFBIT   6,[B].B         ;WAIT UNTIL CLOCKLINE = HIGH
        JP      TRM5            ;CLOCK = HIGH
        JP      TRM4            ;CLOCK = LOW
TRM5:   SBIT    2,PWMODE.B      ;STOP TIMER 4
        LD      T4.W,#2*CLK-1   ;SET START TIME
        LD      X,INDEX.W       ;GET NEXT TABLEENTRY
        JP      RWRST           ;PERFORM NEXT STARTCONDITION

TRM6:   JP      RWEND
RCVE:   DECSZ   WBUF1+1.B       ;DECREMENT BYTECOUNT
        JP      RCV1            ;BYTECOUNT <> 0
        SBIT    2,STATUS.B      ;FLAG LAST BYTE
RCV1:   JSR     RECEIV          ;GET 1 BYTE
        ST      A,[X].B         ;PUT BYTE TO BUFFER
        INC     X               ;X += 1
        IFC                     ;ERROR ?
        JP      STPERR          ;YES -> STOPCONDITION
        IFBIT   2,STATUS.B      ;LAST BYTE FLAGGED ?
        JP      TRM2            ;YES, CHECK MULTISTART
        JP      RCVE            ;GET NEXT BYTE
RWEND:  RC                      ;NO ERROR
STPERR: JSR     STOPCD          ;STOPCONDITION

RWERR:  RESTORE_ABX             ;RESTORE REGISTER
        POP     K               ;RESTORE REG K
        RET
```

TL/DD/10080–11

```
STRTCD: IFBIT   5,PORTI.B       ;TEST DATALINE
        JP      STRT01          ;IF HIGH -> CONTINUE
STRTER: SC                      ;ELSE ERROR
        RET
STRT01: IFBIT   6,[B].B         ;TEST CLOCKLINE
        JP      STRT02          ;IF HIGH -> CONTINUE
        JP      STRTER          ;ELSE ERROR
STRT02: RBIT    5,[B].B         ;DATALINE = LOW
        AND     PWMODE.B,#0FB   ;START TIMER 4
STRT03: IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = LOW
        JP      STRT03
        RC                      ;SIGNAL NO ERROR
        RET

TRANSF: IFBIT   7,A             ;TEST FOR THE NEXT DATA
        JP      TRNF1           ;PUT DATALINE HIGH
        RBIT    5,[B].B         ;PUT DATALINE LOW
        JP      TRNF2
TRNF1:  SBIT    5,[B].B         ;PUT DATALINE HIGH
TRNF2:  SWAP    A
        SWAP    A               ;EXCHANGE LOWER/HIGHER BYTE
        LD      K,#8            ;SET LOOP COUNT
        SHL     A               ;DUMMY SHIFT
        JP      TRF2            ;JUMP INTO THE LOOP
TRF1:   SHL     A               ;SHIFT MSB -> CARRY
        IFC
        SBIT    5,[B].B         ;DATALINE = HIGH
        IFNC
        RBIT    5,[B].B         ;DATALINE = LOW
TRF2:   IFBIT   6,[B].B         ;WAIT UNTIL CLOCK HIGH
        JP      TRF3            ;CLOCK = HIGH
        JP      TRF2            ;CLOCK = LOW
TRF3:   IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = LOW
        JP      TRF3            ;CLOCK = HIGH
        DECSZ   K               ;DECREMENT LOOP COUNT
        JP      TRF1            ;NEXT BIT
        JSR     GETACK          ;LOOK FOR ACKNOWLEDGE
        RET

SETACK: RBIT    5,[B].B         ;DATALINE = LOW
        RC
        JP      ACK01
GETACK: SBIT    5,[B].B         ;DATALINE = HIGH
        RC
ACK01:  IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = HIGH
        JP      ACK02           ;CLOCK = HIGH
        JP      ACK01           ;CLOCK = LOW , WAIT
ACK02:  IFBIT   5,PORTI.B       ;TEST DATALINE
        SC                      ;FLAG EROR IF HIGH
ACK03:  IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = LOW
        JP      ACK03
        SBIT    5,[B].B         ;DATALINE = HIGH
        RET
```

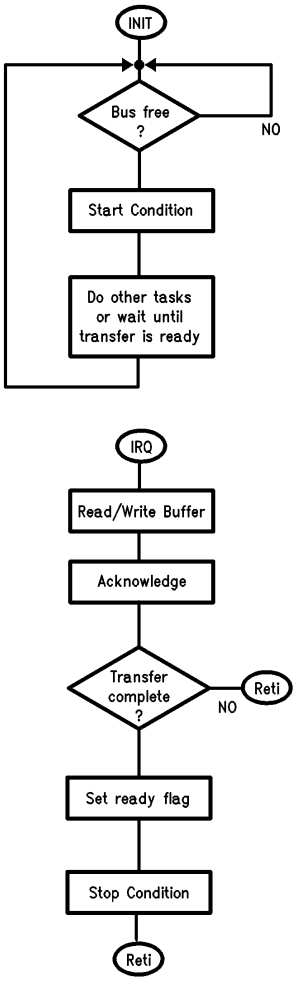
TL/DD/10080–12

```
RECEIV: PUSH    K               ;SAVE REG K
        LD      K,#8            ;SET LOOP COUNT
REC1:   RC
REC2:   IFBIT   6,[B].B         ;WAIT UNTIL CLOCK HIGH
        JP      REC3            ;CLOCK = HIGH
        JP      REC2            ;CLOCK = LOW
REC3:   IFBIT   5,PORTI.B       ;TEST DATALINE
        SC                      ;IF HIGH SET CARRY
        RLC     A               ;ROTATE LEFT WITH CARRY
REC4:   IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = LOW
        JP      REC4            ;CLOCK = HIGH
        DECSZ   K               ;DECREMENT LOOP COUNT
        JP      REC1            ;NEXT BIT
        IFBIT   2,STATUS.B      ;LAST BYTE FLAGGED ?
        JP      REC5            ;YES, NO ACKNOWLEDGE
        JSR     SETACK          ;SET ACKNOWLEDGE
        JP      REC6
REC5:   JSR     GETACK          ;LOOK FOR ACKNOWLEDGE
REC6:   POP     K               ;RESTORE REG K
        RET


STOPCD: RBIT    5,[B].B         ;DATALINE = LOW
STOP01: IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = HIGH
        JP      STOP02          ;CLOCK = HIGH -> STOP02
        JP      STOP01          ;WAIT
STOP02: SBIT    2,PWMODE.B      ;STOP TIMER 4
        LD      T4.W,#CLK-1     ;INITIALIZE T4 TO STARTCONDITION
        RBIT    7,STATUS.B      ;STATUS = I2CBUS NOT BUSY
        SBIT    5,[B].B         ;PERFORM STOPCONDITION
        RET

TSTBUS: RC
        IFBIT   5,PORTI.B       ;TEST DATALINE
        JP      TST1
        SC
TST1:   IFBIT   6,[B].B         ;TEST CLOCKLINE
        JP      TST2
        SC
TST2:   RET

RESET:  LD      ENIR.B,#0       ;DISABLE ALL INTERRUPTS
        LD      PWMODE.W,#0CCCC ;STOP AND CLEAR ALL TIMERS
        LD      TMMODE.W,#0CCCC
START:  JSR     INIT
        LD      B,#WRBUFF
        LD      K,#WRBUFF+8     ;CLEAR 9 BYTES
START0: CLR     A
        XS      A,[B+].W
        JP      START0
        LD      RDBUFF.B,#0     ;SET READADDRESS TO 0

        LD      WPORT.B,#0FF    ;INITIALISE PORT1 AS INPUT
        LD      X,#INIPO1
        JSR     RWI2C
```

TL/DD/10080–13

```
        LD      WPORT.B,#0FF    ;PUT ALL LED'S OFF
START1: LD      X,#WRPO0
        JSR     RWI2C

        LD      X,#WRRAM0       ;WRITE TO RAM
        JSR     RWI2C           ;START TRANSMISSION
        JSR     WAIT

        LD      X,#RDRAM0       ;READ RAM0
        JSR     RWI2C
        JSR     WAIT

        ADD     WRBUFF.B,#8     ;WRITE/READ NEXT 8 BYTES RAM
        ADD     RDBUFF.B,#8
        IFEQ    WRBUFF.B,#0     ;IF WRAP
        DECSZ   WPORT.B         ;DECREMENT LED VALUE
        NOP                     ;ONLY DECREMENT

        LD      X,#RDPO1        ;READ INPUT
        JSR     RWI2C
        JSR     WAIT

        IFBIT   7,RPORT         ;IF BIT SET FREE-RUN-LED
        JP      START1
        LD      WPORT.B,RPORT.B ;ELSE COPY INPUT TO OUTPUT

        JP      START1

WAIT:   PUSH    X               ;SAVE X-REG
        LD      X,#010          ;INITIALIZE WAITLOOP
WAIT1:  DECSZ   X
        JP      WAIT1
        POP     X               ;RESTORE X-REG
        RET

INIPO1: .DW     0242,WPORT,0            ;INITIALIZE PORT1 AS INPUT
RDPO1:  .DW     0243,RPORT,0            ;READ 1 BYTE FROM PORT1
WRPO0:  .DW     0240,WPORT,0            ;WRITE 1 BYTE TO PORT0
WRRAM0: .DW     0AA0,WRBUFF,0           ;WRITE 8 BYTES TO RAM
RDRAM0: .DW     02A0,WRBUFF,0AA1,RDBUFF+1,0     ;READ 10 BYTES


        .END RESET
```

TL/DD/10080–14



TL/DD/10080–15

FIGURE 9. Interrupt Driver Flowchart

```
        ;*****************************************************
        ;*  INTER-IC-BUS (I2C-BUS) WITH HPC  : APPL.NOTE
        ;*  using uWire interface with interrupt
        ;*****************************************************

        ;* 12.10.87                                        HU
        ;* 20.10.87                                        HU
        ;* 04.11.87                                        HU
        ;* 08.02.88                                        HU

        .INCLD  HPC16083.MAP    ; MEMORY MAP FOR HPC16083

        ;DEFINITIONS
        CLK = 30                ;CLOCK LOW/HIGH TIME = 33us

        .MACRO SAVE_AB

        PUSH    A               ;SAVE A-REG
        PUSH    B               ;SAVE B-REG

        .ENDM

        .MACRO SAVE_ABX

        SAVE_AB                 ;SAVE REGS A,B
        PUSH    X               ;SAVE X-REG

        .ENDM

        .MACRO RESTORE_AB

        POP     B               ;RESTORE B-REG
        POP     A               ;RESTORE A_REG

        .ENDM

        .MACRO RESTORE_ABX

        POP     X               ;RESTORE X-REG
        RESTORE_AB              ;RESTORE REGS B,A

        .ENDM

        .SECT   B1,BASE         ;DEFINE BASEPAGE SECTION

WBUF1:  .DSW    1               ;WORDBUFFER FOR I2C-TABLE
WBUF2:  .DSW    1               ;WORDBUFFER FOR I2C-TABLE
INDEX:  .DSW    1               ;POINTER TO THE NEXT TABLEENTRY
STATUS: .DSB    1               ;STATUSBYTE
DUMMY:  .DSB    1               ;DUMMY BYTE
WPORT:  .DSB    1               ;8-BIT VALUE WRITE TO PORT0
RPORT:  .DSB    1               ;8-BIT VALUE READ FROM PORT1
WRBUFF: .DSB    10              ;RAM WRITE BUFFER
RDBUFF: .DSB    10              ;RAM READ BUFFER
```

TL/DD/10080–16

```
        .ENDSECT

        .SECT   I2C,ROM16

        ;*****************************************************
        ;* INIT : THIS SUBROUTINE INITIALIZES TIMER T4, TIMER T5
        ;*        AND THE uWIRE-INTERFACE TO OPERATE AS I2C-BUS
        ;*
        ;* INPUT : NONE
        ;* OUTPUT : NONE
        ;* USED REGS : A, B, X   ( ALL REGS ARE SAVED )
        ;*****************************************************

INIT:   SAVE_ABX                ;SAVE REGS A,B,X
        LD      X,#PWMODE       ;ADDR. PWMODE-REG -> X
        LD      B,#PORTP        ;ADDR. PORTP-REG -> B
        LD      A,#0CC          ;VALUE TO STOP TIMERS
        ST      A,[X].B         ;STOP T4, T5, NO IRQ, ACK TIP-FLAG
        NOP
        ST      A,[X].B         ;MAKE SHURE TIP-FLAGS ARE CLEARED
        LD      A,#011          ;DISABLE TOGGLE AND SET OUTPUT HIGH
        ST      A,[B].B         ;ON PINS P0 AND P1
        SBIT    3,[B].B         ;TOGGLE ON AT P0
        SBIT    7,[B].B         ;TOGGLE ON AT P1
        LD      T4.W,#CLK-1     ;LOAD T4 (33us)
        LD      R4.W,#CLK-1     ;LOAD R4 (33us)
        LD      T5.W,#17*CLK-1  ;9-BIT SHIFT TIME (STARTCONDITION)
        LD      R5.W,#18*CLK-1  ;9-BIT SHIFT TIME (NORMAL MODE)
        SBIT    5,PORTB.B       ;DATALINE OUTPUT = HIGH
        SBIT    5,DIRB.B        ;B5 = OUTPUT
        RBIT    5,BFUN.B        ;NO ALTERNATE FUNCTION SELECTED
        RBIT    6,DIRB.B        ;B6 = INPUT
        SBIT    6,BFUN.B        ;SELECT SK-INPUT
        LD      A,DIVBY.B       ;SET UWIRE-DEVIDE
        AND     A,#0F0
        OR      A,#02           ;SET CLKI /16
        ST      A,DIVBY.B       ;STORE NEW VALUE
        SBIT    1,IRCD.B        ;ACTIVATE UWIRE
INIT1:  IFBIT   0,IRPD.B        ;TEST IF READY
        JP      INIT2           ;YES CONTINUE
        JP      INIT1           ;NO WAIT
INIT2:  RBIT    1,IRCD.B        ;SELECT SLAVE MODE
        RBIT    6,BFUN.B        ;
        SBIT    4,[X].B         ;ENABLE T5-IRQ
        OR      ENIR.B,#021     ;ENABLE GLOBAL TIMER IRQ
        RESTORE_ABX             ;RESTORE REGS X,B,A
        RET


RWI2C:  PUSH    A               ;SAVE A-REGISTER
        LD      STATUS.B,#0     ;RESET STATUSBYTE
        LD      A,[X+].W        ;GET 2 BYTES OF TABLE
        JSR     STRTCD          ;PERFORM STARTCONDITION
        IFC
```

TL/DD/10080–17

```
        JP      RWERR           ;IF ERROR -> EXIT
        SBIT    5,BFUN.B        ;ENABLE SO-OUTPUT
        ST      A,WBUF1.W       ;SAVE TABLE CONTENTS
        IFBIT   0,A             ;TEST RECEIVE/TRANSMIT BIT
        JP      RWRECV          ;BIT = 1 -> RECEIVE
        RBIT    0,STATUS.B      ;STATUS = TRANSMIT
        JP      RW01            ;CONTINUE AT RW01
RWRECV: SBIT    0,STATUS.B      ;STATUS = RECEIVE
RW01:   SBIT    1,STATUS.B      ;STATUS = FIRST BYTE
        DECSZ   WBUF1+1.B       ;DEC BYTECOUNT
        JP      RW02            ;MORE THAN 1 BYTE TO PROCESS
        SBIT    2,STATUS.B      ;STATUS = LAST BYTE
RW02:   LD      A,[X+].W        ;GET NEXT 2 BYTES OF TABLE
        ST      A,WBUF2.W       ;SAVE TABLE CONTENTS
        LD      A,X
        ST      A,INDEX.W       ;SAVE INDEX
        LD      A,[X].W         ;GET NEXT WORD OF TABLE
        IFEQ    A,#0            ;ANY MORE TO TRANSFER
        JP      RW03            ;NO, EXIT
        SBIT    3,STATUS.B      ;STATUS = MULTISTART
RW03:   SBIT    7,STATUS.B      ;STATUS = BUSY
        RC                      ;CLR CARRY = NO ERROR
RWERR:  POP     A               ;RESTORE A-REGISTER
        RET

STRTCD:
        IFBIT   5,PORTI.B       ;TEST DATALINE
        JP      STRT01          ;IF HIGH -> CONTINUE
STRTER: SC                      ;ELSE ERROR
        RET
STRT01: IFBIT   6,PORTB.B       ;TEST CLOCKLINE
        JP      STRT02          ;IF HIGH -> CONTINUE
        JP      STRTER          ;ELSE ERROR
STRT02: RBIT    5,PORTB.B       ;DATALINE = LOW
        AND     PWMODE.B,#0BB   ;START TIMER 4 AND 5
STRT03: IFBIT   6,PORTB.B       ;WAIT UNTIL CLOCK = LOW
        JP      STRT03
        ST      A,SIO.B         ;WRITE 1 BYTE TO SIO AND ENABLE SHIFT
        RC                      ;SIGNAL NO ERROR
        RET

TIMIRQ:
        IFBIT   5,PWMODE.B      ;TIMER 5 IRQ ?
        JP      T1IRQ           ;YES, CONTINUE
        JP      IRQRET          ;NO TIMER IRQ
T1IRQ:  SBIT    5,PORTB.B
        RBIT    5,BFUN.B
        SBIT    7,PWMODE.B      ;ACK IRQ
        SAVE_ABX                ;SAVE REGS A,B,X
        LD      B,#PORTB        ;PORTB-ADDR -> REG B
        LD      X,#STATUS       ;STATUS-ADDR -> REG-X
        PUSH    PSW.W           ;
        IFBIT   2,[X].B         ;LAST BYTE ?
        JP      LAST            ;YES -> JUMP
        IFBIT   1,[X].B         ;FIRST BYTE ?
```

TL/DD/10080–18

```
        JP      FIRST           ;YES -> JUMP
        IFBIT   0,[X].B         ;RECEIVEMODE ?
        JP      RECVE           ;YES -> JUMP
        JSR     GETACK          ;ACKNOWLEDGE
        IFC                     ;ERROR ?
        JP      STOPCD          ;STOP TRANSMISSION
TRMIT:  LD      A,[WBUF2].B     ;GET NEXT BYTE
        ST      A,SIO.B         ;ENABLE SHIFT
        JP      TINC            ;-> INCREMENT POINTERS
RECVE:  LD      A,SIO.B         ;GET DATA
        ST      A,[WBUF2].B     ;PUT INTO BUFFER
        JSR     SETACK          ;ACKNOWLEDGE
        IFC                     ;ERROR ?
        JP      STOPCD          ;STOP TRANSMISSION
        LD      SIO.B,#0FF      ;ENABLE SHIFT
TINC:   SBIT    5,BFUN.B        ;SELECT ALTERNATE MODE
TINC1:  INC     WBUF2.W         ;INC BUFFERPOINTER
TDEC:   DECSZ   WBUF1+1.B       ;DEC BYTECOUNT
        JP      T5IR02          ;MORE THAN 1 BYTE TO PROCESS
        SBIT    2,[X].B         ;STATUS = LAST BYTE
T5IR02: JP      IRQEND          ;EXIT AND WAIT FOR NEXT IRQ

LAST:   IFBIT   0,[X].B         ;RECEIVE ?
        JP      LASTRD          ;YES -> JUMP
LAST1:  JSR     GETACK          ;ACKNOWLEDGE
        IFBIT   3,[X].B         ;RESTART ?
        JP      RESTRT          ;YES -> JUMP
        JP      STOPCD          ;STOP TRANSMISSION

LASTRD: LD      A,SIO.B         ;GET LAST CHARACTER
        ST      A,[WBUF2].B     ;PUT INTO BUFFER
        JP      LAST1

FIRST:  LD      A,[WBUF2].B     ;GET NEXT BYTE
        JSR     GETACK          ;ACKNOWLEDGE
        IFC                     ;ERROR ?
        JP      STOPCD          ;STOP TRANSMISSION
        RBIT    1,[X].B         ;RESET FIRST BYTE FLAG
        IFBIT   0,[X].B         ;RECEIVE ?
        JP      IRRCV           ;YES -> JUMP
        ST      A,SIO.B         ;ENABLE SHIFT
        SBIT    5,BFUN.B        ;SELECT ALTERNATE FUNCTION
        JP      TINC1           ;-> INCREMENT POINTERS

IRRCV:  LD      SIO.B,#0FF      ;ACTIVATE SHIFT
        SBIT    5,BFUN.B        ;SELECT ALTERNATE FUNCTION
        JP      TDEC

IRQEND: POP     PSW.W
        RESTORE_ABX             ;RESTORE REGS X,B,A
IRQRET: RETI


RESTRT: LD      X,INDEX.W       ;GET NEXT POINTER TO ENTRYTABLE
        SBIT    5,[B].B         ;DATALINE = HIGH
```

TL/DD/10080–19

```
        RBIT    5,BFUN.B        ;DISABLE SO-OUTPUT
REST01: IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = HIGH
        JP      REST02          ;CLOCK = HIGH -> REST02
        JP      REST01          ;WAIT
REST02: SBIT    2,PWMODE.B      ;STOP TIMER 4
        SBIT    6,PWMODE.B      ;STOP TIMER 5
        LD      T4.W,#2*CLK-1   ;LOAD TIMER 4
        LD      T5.W,#18*CLK-1  ;LOAD TIMER 5
        JSR     RWI2C           ;INITIALIZE READ/WRITE TO I2C-BUS
        JP      IRQEND

STOPCD: RBIT    5,[B].B         ;DATALINE = LOW
        RBIT    5,BFUN.B        ;DISABLE SO-OUTPUT
STOP01: IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = HIGH
        JP      STOP02          ;CLOCK = HIGH -> STOP02
        JP      STOP01          ;WAIT
STOP02: SBIT    2,PWMODE.B      ;STOP TIMER 4
        SBIT    6,PWMODE.B      ;STOP TIMER 5
        LD      T4.W,#CLK-1     ;INITIALIZE T4 TO STARTCONDITION
        LD      T5.W,#17*CLK-1  ;INITIALIZE T5 TO STARTCONDITION
        RBIT    7,[X].B         ;STATUS = I2CBUS NOT BUSY
        SBIT    5,[B].B         ;PERFORM STOPCONDITION
        JP      IRQEND


SETACK: RBIT    5,[B].B         ;DATALINE = LOW
        RC
        JP      ACK01
GETACK: SBIT    5,[B].B         ;DATALINE = HIGH
        RC
ACK01:  IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = HIGH
        JP      ACK02           ;CLOCK = HIGH
        JP      ACK01           ;CLOCK = LOW , WAIT
ACK02:  IFBIT   5,PORTI.B       ;TEST DATALINE
        SC                      ;FLAG EROR IF HIGH
ACK03:  IFBIT   6,[B].B         ;WAIT UNTIL CLOCK = LOW
        JP      ACK03
        SBIT    5,[B].B         ;DATALINE = HIGH
        RET

TSTBUS: RC
        IFBIT   5,PORTI.B       ;TEST DATALINE
        JP      TST1
        SC
TST1:   IFBIT   6,[B].B         ;TEST CLOCKLINE
        JP      TST2
        SC
TST2:   RET


RESET:  LD      ENIR.B,#0       ;DISABLE ALL INTERRUPTS
        LD      PWMODE.W,#0CCCC ;STOP AND CLEAR ALL TIMERS
        LD      TMMODE.W,#0CCCC
START:  JSR     INIT
        LD      B,#WRBUFF
```

TL/DD/10080–20

```
        LD      K,#WRBUFF+8     ;CLEAR 9 BYTES
START0: CLR     A
        XS      A,[B+].W
        JP      START0
        LD      RDBUFF.B,#0     ;SET READADDRESS TO 0

        LD      WPORT.B,#0FF    ;INITIALISE PORT1 AS INPUT
        LD      X,#INIPO1
        JSR     RWI2C
        JSR     WAIT

        LD      WPORT.B,#0FF    ;PUT ALL LED'S OFF
START1: LD      X,#WRPO0
        JSR     RWI2C
        JSR     WAIT

        LD      X,#WRRAM0       ;WRITE TO RAM
        JSR     RWI2C           ;START TRANSMISSION
        JSR     WAIT

        LD      X,#RDRAM0       ;READ RAM0
        JSR     RWI2C
        JSR     WAIT

        ADD     WRBUFF.B,#8     ;WRITE/READ NEXT 8 BYTES RAM
        ADD     RDBUFF.B,#8
        IFEQ    WRBUFF.B,#0     ;IF WRAP
        DECSZ   WPORT.B         ;DECREMENT LED VALUE
        NOP                     ;ONLY DECREMENT

        LD      X,#RDPO1        ;READ INPUT
        JSR     RWI2C
        JSR     WAIT

        IFBIT   7,RPORT         ;IF BIT SET FREE-RUN-LED
        JP      START1
        LD      WPORT.B,RPORT.B ;ELSE COPY INPUT TO OUTPUT

        JP      START1

WAIT:   PUSH    X               ;SAVE X-REG
        LD      X,#010          ;INITIALIZE WAITLOOP
        IFC
        INC     DUMMY.B
WAIT1:  IFBIT   7,STATUS.B      ;WAIT UNTIL READY
        JP      WAIT1
WAIT2:  DECSZ   X
        JP      WAIT2
        POP     X               ;RESTORE X-REG
        RET

INIPO1: .DW     0242,WPORT,0            ;INITIALIZE PORT1 AS INPUT
RDPO1:  .DW     0243,RPORT,0            ;READ 1 BYTE FROM PORT1
WRPO0:  .DW     0240,WPORT,0            ;WRITE 1 BYTE TO PORT0
WRRAM0: .DW     0AA0,WRBUFF,0           ;WRITE 8 BYTES TO RAM
```

TL/DD/10080–21

```
RDRAM0: .DW     02A0,WRBUFF,0AA1,RDBUFF+1,0     ;READ 10 BYTES


        .IPT    5,TIMIRQ        ;SET TIMER IRQ ENTRY
        .END RESET
```

TL/DD/10080–22

```
;
;****** INCLUDE FILE HPC16083.MAP *******
;
;****** HPC-REGISTER DEFINITIONS ********

        PSW     = 0C0       ;PROCESSOR STATUS REGISTER
;       SP      = 0C4       ;STACK POINTER
;       PC      = 0C6       ;PROGRAM COUNTER
;       A       = 0C8       ;ACCUMULATOR
;       K       = 0CA       ;K REGISTER
;       B       = 0CC       ;B REGISTER
;       X       = 0CE       ;X REGISTER
        PORTA   = 0E0       ;PORTA DATA / OUTPUT BUFFER
        DIRA    = 0F0       ;PORTA DIRECTION / INPUT BUFFER
        PORTB   = 0E2       ;PORTB DATA REGISTER
        DIRB    = 0F2       ;PORTB DIRECTION REGISTER
        BFUN    = 0F4       ;PORTB ALTERNATE FUNCTION REG
        PORTI   = 0D8       ;PORTI DATA REGISTER
        PORTD   = 0104      ;PORTD DATA REGISTER
        PORTP   = 0152      ;PORTP REGISTER
        ENIR    = 0D0       ;INTERRUPT ENABLE REGISTER
        IRCD    = 0D4       ;INTERRUPT AND CAPTURE CONDITION REG
        IRPD    = 0D2       ;INTERRUPT PENDING REGISTER
        HLTEN   = 0DC       ;HALT ENABLE CONTROL CIRCUIT
        DIVBY   = 018E      ;DIVIDE BY REGISTER
        PWMODE  = 0150      ;PULSE WIDTH MODE REGISTER
        TMMODE  = 0190      ;TIMER MODE REGISTER
        I2CR    = 0184      ;I2 CAPTURE REGISTER / R1
        I3CR    = 0182      ;I3 CAPTURE REGISTER / T1
        I4CR    = 0180      ;I4 CAPTURE REGISTER
        EICR    = 015E      ;EI CAPTURE REGISTER
        EICON   = 015C      ;EI CONFIGURATION REGISTER
        T0CON   = 0192      ;T0 CAPTURE CONFIGURATION REG
        T2      = 0188      ;TIMER2
        R2      = 0186      ;TIMER2 MODULUS REGISTER
        T3      = 018C      ;TIMER3
        R3      = 018A      ;TIMER3 MODULUS REGISTER
        T4      = 0140      ;TIMER4
        R4      = 0142      ;TIMER4 MODULUS REGISTER
        T5      = 0144      ;TIMER5
        R5      = 0146      ;TIMER5 MODULUS REGISTER
        T6      = 0148      ;TIMER6
        R6      = 014A      ;TIMER6 MODULUS REGISTER
        T7      = 014C      ;TIMER7
        R7      = 014E      ;TIMER7 MODULUS REGISTER
        WD      = 0194      ;WATCHDOG REGISTER
        SIO     = 0D6       ;SERIAL INPUT OUTPUT SHIFT REG
        ENU     = 0120      ;UART CONTROL AND STATUS REGISTER
        ENUI    = 0122      ;UART INTERRUPT AND CLOCK SOURCE REG
        RBUF    = 0124      ;UART RECEIVE BUFFER
        TBUF    = 0126      ;UART TRANSMIT BUFFER
        ENUR    = 0128      ;UART RECEIVE CONTROL AND STATUS REG
        UPIC    = 0E6       ;UPI CONTROL REGISTER
```

TL/DD/10080–23

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

THE I²C-BUS SPECIFICATION

VERSION 2.1

JANUARY 2000

The I²C-bus specification

CONTENTS

The I²C-bus specification

1 PREFACE

1.1 Version 1.0 - 1992

This version of the 1992 I²C-bus specification includes the following modifications:

- Programming of a slave address by software has been omitted. The realization of this feature is rather complicated and has not been used.
- The “low-speed mode” has been omitted. This mode is, in fact, a subset of the total I²C-bus specification and need not be specified explicitly.
- The Fast-mode is added. This allows a fourfold increase of the bit rate up to 400 kbit/s. Fast-mode devices are downwards compatible i.e. they can be used in a 0 to 100 kbit/s I²C-bus system.
- 10-bit addressing is added. This allows 1024 additional slave addresses.
- Slope control and input filtering for Fast-mode devices is specified to improve the EMC behaviour.

NOTE: Neither the 100 kbit/s I²C-bus system nor the 100 kbit/s devices have been changed.

1.2 Version 2.0 - 1998

The I²C-bus has become a de facto world standard that is now implemented in over 1000 different ICs and licensed to more than 50 companies. Many of today’s applications, however, require higher bus speeds and lower supply voltages. This updated version of the I²C-bus specification meets those requirements and includes the following modifications:

- The High-speed mode (Hs-mode) is added. This allows an increase in the bit rate up to 3.4 Mbit/s. Hs-mode devices can be mixed with Fast- and Standard-mode devices on the one I²C-bus system with bit rates from 0 to 3.4 Mbit/s.
- The low output level and hysteresis of devices with a supply voltage of 2 V and below has been adapted to meet the required noise margins and to remain compatible with higher supply voltage devices.
- The 0.6 V at 6 mA requirement for the output stages of Fast-mode devices has been omitted.
- The fixed input levels for new devices are replaced by bus voltage-related levels.
- Application information for bi-directional level shifter is added.

1.3 Version 2.1 - 2000

Version 2.1 of the I²C-bus specification includes the following minor modifications:

- After a repeated START condition in Hs-mode, it is possible to stretch the clock signal SCLH (see Section 13.2 and Figs 22, 25 and 32).
- Some timing parameters in Hs-mode have been relaxed (see Tables 6 and 7).

1.4 Purchase of Philips I²C-bus components

Purchase of Philips I²C components conveys a license under the Philips’ I²C patent to use the components in the I²C system provided the system conforms to the I²C specification defined by Philips.

2 THE I²C-BUS BENEFITS DESIGNERS AND MANUFACTURERS

In consumer electronics, telecommunications and industrial electronics, there are often many similarities between seemingly unrelated designs. For example, nearly every system includes:

- Some intelligent control, usually a single-chip microcontroller
- General-purpose circuits like LCD drivers, remote I/O ports, RAM, EEPROM, or data converters
- Application-oriented circuits such as digital tuning and signal processing circuits for radio and video systems, or DTMF generators for telephones with tone dialling.

To exploit these similarities to the benefit of both systems designers and equipment manufacturers, as well as to maximize hardware efficiency and circuit simplicity, Philips developed a simple bi-directional 2-wire bus for efficient inter-IC control. This bus is called the Inter IC or I²C-bus. At present, Philips' IC range includes more than 150 CMOS and bipolar I²C-bus compatible types for performing functions in all three of the previously mentioned categories. All I²C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I²C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits.

Here are some of the features of the I²C-bus:

- Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL)
- Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers
- It's a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer
- Serial, 8-bit oriented, bi-directional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, or up to 3.4 Mbit/s in the High-speed mode
- On-chip filtering rejects spikes on the bus data line to preserve data integrity
- The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance of 400 pF.

Figure 1 shows two examples of I²C-bus applications.

2.1 Designer benefits

I²C-bus compatible ICs allow a system design to rapidly progress directly from a functional block diagram to a prototype. Moreover, since they 'clip' directly onto the I²C-bus without any additional external interfacing, they allow a prototype system to be modified or upgraded simply by 'clipping' or 'unclipping' ICs to or from the bus.

Here are some of the features of I²C-bus compatible ICs which are particularly attractive to designers:

- Functional blocks on the block diagram correspond with the actual ICs; designs proceed rapidly from block diagram to final schematic.
- No need to design bus interfaces because the I²C-bus interface is already integrated on-chip.
- Integrated addressing and data-transfer protocol allow systems to be completely software-defined
- The same IC types can often be used in many different applications
- Design-time reduces as designers quickly become familiar with the frequently used functional blocks represented by I²C-bus compatible ICs
- ICs can be added to or removed from a system without affecting any other circuits on the bus
- Fault diagnosis and debugging are simple; malfunctions can be immediately traced
- Software development time can be reduced by assembling a library of reusable software modules.

In addition to these advantages, the CMOS ICs in the I²C-bus compatible range offer designers special features which are particularly attractive for portable equipment and battery-backed systems.

They all have:

- Extremely low current consumption
- High noise immunity
- Wide supply voltage range
- Wide operating temperature range.

The I²C-bus specification

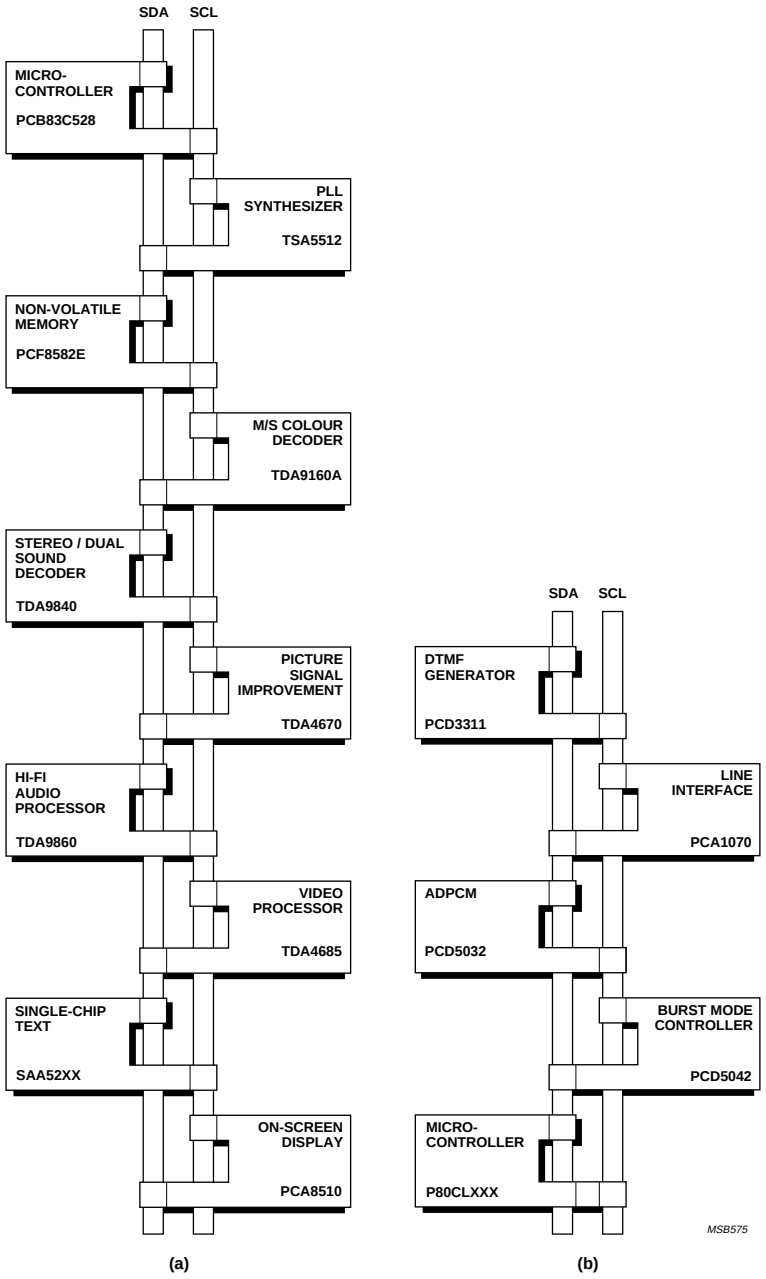


Fig.1 Two examples of I²C-bus applications: (a) a high performance highly-integrated TV set (b) DECT cordless phone base-station.

2.2 Manufacturer benefits

I²C-bus compatible ICs don't only assist designers, they also give a wide range of benefits to equipment manufacturers because:

- The simple 2-wire serial I²C-bus minimizes interconnections so ICs have fewer pins and there are not so many PCB tracks; result - smaller and less expensive PCBs
- The completely integrated I²C-bus protocol eliminates the need for address decoders and other 'glue logic'
- The multi-master capability of the I²C-bus allows rapid testing and alignment of end-user equipment via external connections to an assembly-line
- The availability of I²C-bus compatible ICs in SO (small outline), VSO (very small outline) as well as DIL packages reduces space requirements even more.

These are just some of the benefits. In addition, I²C-bus compatible ICs increase system design flexibility by allowing simple construction of equipment variants and easy upgrading to keep designs up-to-date. In this way, an entire family of equipment can be developed around a basic model. Upgrades for new equipment, or enhanced-feature models (i.e. extended memory, remote control, etc.) can then be produced simply by clipping the appropriate ICs onto the bus. If a larger ROM is needed, it's simply a matter of selecting a micro-controller with a larger ROM from our comprehensive range. As new ICs supersede older ones, it's easy to add new features to equipment or to increase its performance by simply unclipping the outdated IC from the bus and clipping on its successor.

3 INTRODUCTION TO THE I²C-BUS SPECIFICATION

For 8-bit oriented digital control applications, such as those requiring microcontrollers, certain design criteria can be established:

- A complete system usually consists of at least one microcontroller and other peripheral devices such as memories and I/O expanders
- The cost of connecting the various devices within the system must be minimized
- A system that performs a control function doesn't require high-speed data transfer
- Overall efficiency depends on the devices chosen and the nature of the interconnecting bus structure.

To produce a system to satisfy these criteria, a serial bus structure is needed. Although serial buses don't have the throughput capability of parallel buses, they do require less wiring and fewer IC connecting pins. However, a bus is not merely an interconnecting wire, it embodies all the formats and procedures for communication within the system.

Devices communicating with each other on a serial bus must have some form of protocol which avoids all possibilities of confusion, data loss and blockage of information. Fast devices must be able to communicate with slow devices. The system must not be dependent on the devices connected to it, otherwise modifications or improvements would be impossible. A procedure has also to be devised to decide which device will be in control of the bus and when. And, if different devices with different clock speeds are connected to the bus, the bus clock source must be defined. All these criteria are involved in the specification of the I²C-bus.

4 THE I²C-BUS CONCEPT

The I²C-bus supports any IC fabrication process (NMOS, CMOS, bipolar). Two wires, serial data (SDA) and serial clock (SCL), carry information between the devices connected to the bus. Each device is recognized by a unique address (whether it's a microcontroller, LCD driver, memory or keyboard interface) and can operate as either a transmitter or receiver, depending on the function of the device. Obviously an LCD driver is only a receiver, whereas a memory can both receive and transmit data. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers (see Table 1). A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

The I²C-bus specification

Table 1 Definition of I²C-bus terminology

TERM	DESCRIPTION
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	The device addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

The I²C-bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it. As masters are usually micro-controllers, let's consider the case of a data transfer between two microcontrollers connected to the I²C-bus (see Fig.2).

This highlights the master-slave and receiver-transmitter relationships to be found on the I²C-bus. It should be noted that these relationships are not permanent, but only depend on the direction of data transfer at that time. The transfer of data would proceed as follows:

1) Suppose microcontroller A wants to send information to microcontroller B:

- microcontroller A (master), addresses microcontroller B (slave)
- microcontroller A (master-transmitter), sends data to microcontroller B (slave- receiver)
- microcontroller A terminates the transfer

2) If microcontroller A wants to receive information from microcontroller B:

- microcontroller A (master) addresses microcontroller B (slave)
- microcontroller A (master- receiver) receives data from microcontroller B (slave- transmitter)
- microcontroller A terminates the transfer.

Even in this case, the master (microcontroller A) generates the timing and terminates the transfer.

The possibility of connecting more than one microcontroller to the I²C-bus means that more than one master could try to initiate a data transfer at the same time. To avoid the chaos that might ensue from such an event - an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all I²C interfaces to the I²C-bus.

If two or more masters try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line (for more detailed information concerning arbitration see Section 8).

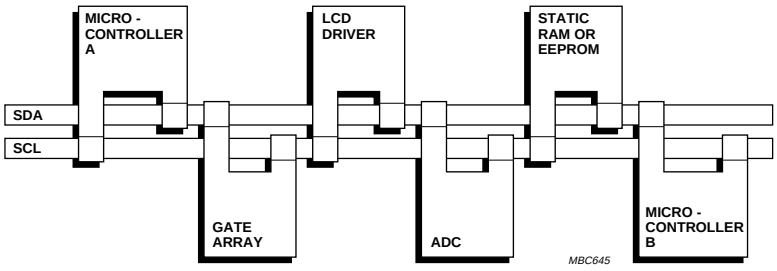


Fig.2 Example of an I²C-bus configuration using two microcontrollers.

Generation of clock signals on the I²C-bus is always the responsibility of master devices; each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can only be altered when they are stretched by a slow-slave device holding-down the clock line, or by another master when arbitration occurs.

5 GENERAL CHARACTERISTICS

Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a current-source or pull-up resistor (see Fig.3). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function. Data on the I²C-bus can be transferred at rates of up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, or up to 3.4 Mbit/s in the High-speed mode. The number of interfaces connected to the bus is solely dependent on the bus capacitance limit of 400 pF. For information on High-speed mode master devices, see Section 13.

6 BIT TRANSFER

Due to the variety of different technology devices (CMOS, NMOS, bipolar) which can be connected to the I²C-bus, the levels of the logical ‘0’ (LOW) and ‘1’ (HIGH) are not fixed and depend on the associated level of V_{DD} (see Section 15 for electrical specifications). One clock pulse is generated for each data bit transferred.

6.1 Data validity

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Fig.4).

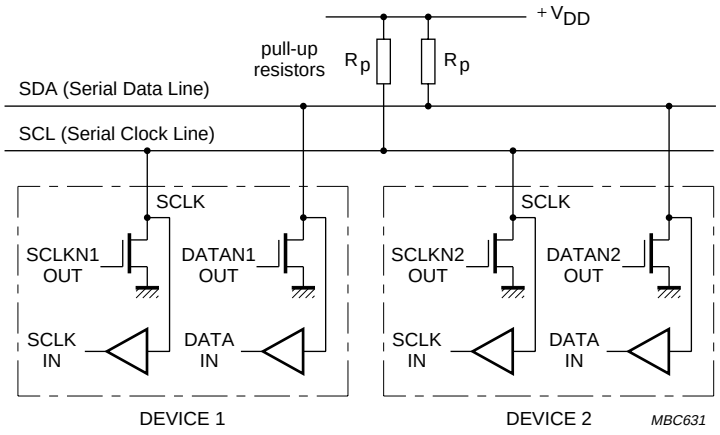


Fig.3 Connection of Standard- and Fast-mode devices to the I²C-bus.

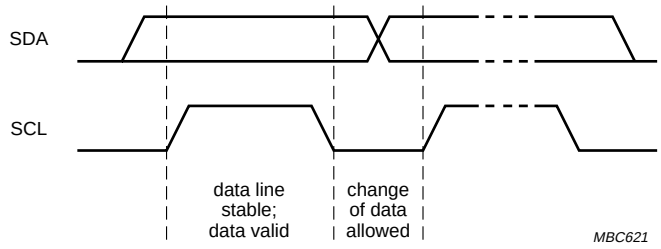


Fig.4 Bit transfer on the I²C-bus.

6.2 START and STOP conditions

Within the procedure of the I²C-bus, unique situations arise which are defined as START (S) and STOP (P) conditions (see Fig.5).

A HIGH to LOW transition on the SDA line while SCL is HIGH is one such unique case. This situation indicates a START condition.

A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition. This bus free situation is specified in Section 15.

The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition. In this respect, the START (S) and repeated START (Sr) conditions are functionally identical (see Fig. 10). For the remainder of this document, therefore, the S symbol will be used as a generic term to represent both the START and repeated START conditions, unless Sr is particularly relevant.

Detection of START and STOP conditions by devices connected to the bus is easy if they incorporate the necessary interfacing hardware. However, microcontrollers with no such interface have to sample the SDA line at least twice per clock period to sense the transition.

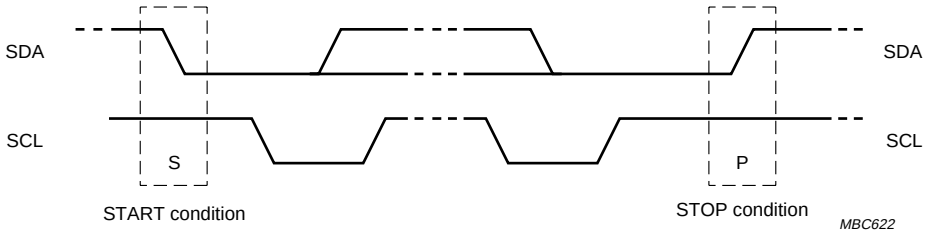


Fig.5 START and STOP conditions.

7 TRANSFERRING DATA

7.1 Byte format

Every byte put on the SDA line must be 8-bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (see Fig.6). If a slave can't receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the master into a wait state. Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.

In some cases, it's permitted to use a different format from the I²C-bus format (for CBUS compatible devices for example). A message which starts with such an address can be terminated by generation of a STOP condition, even during the transmission of a byte. In this case, no acknowledge is generated (see Section 10.1.3).

7.2 Acknowledge

Data transfer with acknowledge is obligatory. The acknowledge-related clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse.

The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse (see Fig.7). Of course, set-up and hold times (specified in Section 15) must also be taken into account.

Usually, a receiver which has been addressed is obliged to generate an acknowledge after each byte has been received, except when the message starts with a CBUS address (see Section 10.1.3).

When a slave doesn't acknowledge the slave address (for example, it's unable to receive or transmit because it's performing some real-time function), the data line must be left HIGH by the slave. The master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer.

If a slave-receiver does acknowledge the slave address but, some time later in the transfer cannot receive any more data bytes, the master must again abort the transfer. This is indicated by the slave generating the not-acknowledge on the first byte to follow. The slave leaves the data line HIGH and the master generates a STOP or a repeated START condition.

If a master-receiver is involved in a transfer, it must signal the end of data to the slave- transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

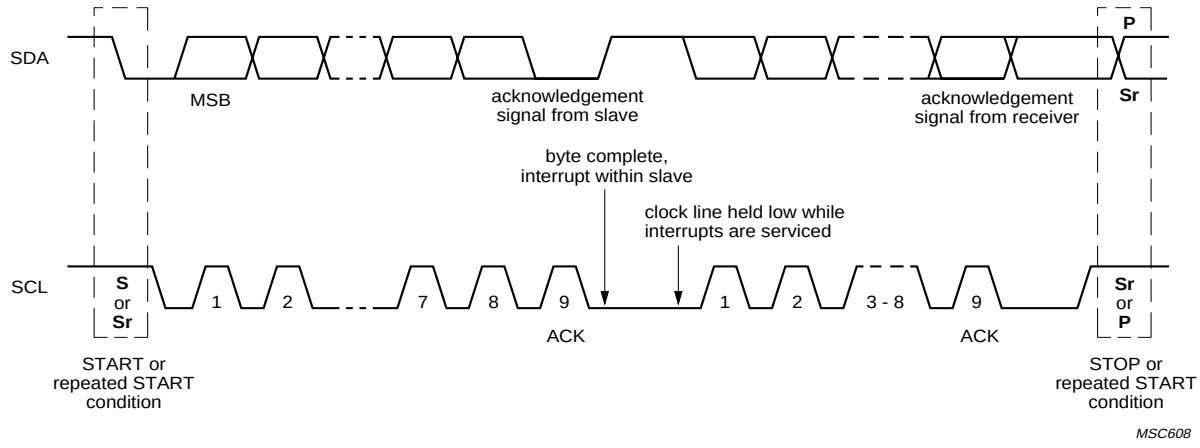


Fig.6 Data transfer on the I²C-bus.

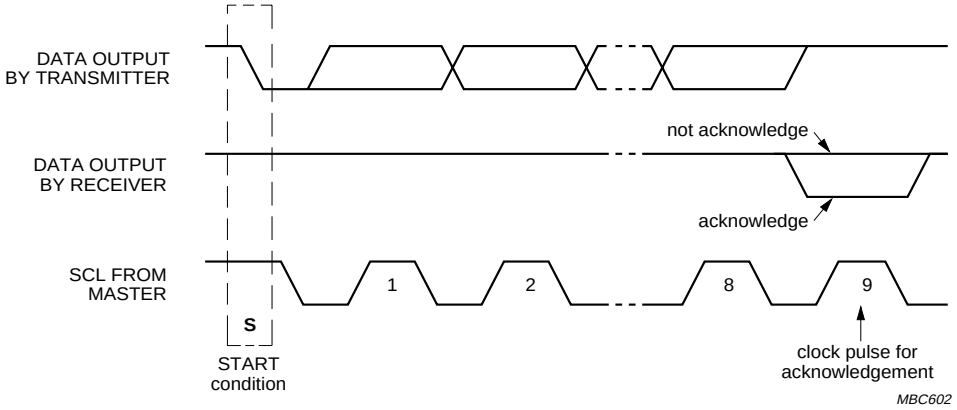


Fig.7 Acknowledge on the I²C-bus.

8 ARBITRATION AND CLOCK GENERATION

8.1 Synchronization

All masters generate their own clock on the SCL line to transfer messages on the I²C-bus. Data is only valid during the HIGH period of the clock. A defined clock is therefore needed for the bit-by-bit arbitration procedure to take place.

Clock synchronization is performed using the wired-AND connection of I²C interfaces to the SCL line. This means that a HIGH to LOW transition on the SCL line will cause the devices concerned to start counting off their LOW period and, once a device clock has gone LOW, it will hold the SCL line in that state until the clock HIGH state is reached (see Fig.8). However, the LOW to HIGH transition of this clock may not change the state of the SCL line if another clock is still within its LOW period. The SCL line will therefore be held LOW by the device with the longest LOW period. Devices with shorter LOW periods enter a HIGH wait-state during this time.

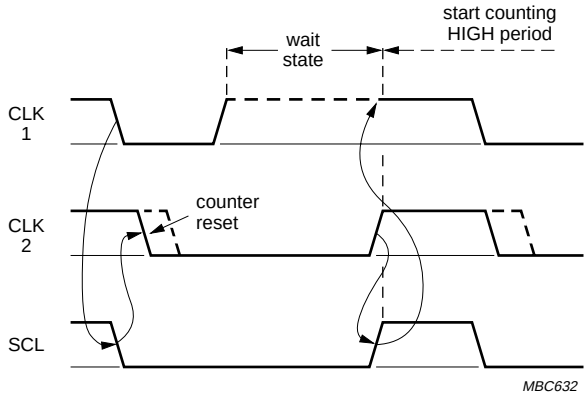


Fig.8 Clock synchronization during the arbitration procedure.

When all devices concerned have counted off their LOW period, the clock line will be released and go HIGH. There will then be no difference between the device clocks and the state of the SCL line, and all the devices will start counting their HIGH periods. The first device to complete its HIGH period will again pull the SCL line LOW.

In this way, a synchronized SCL clock is generated with its LOW period determined by the device with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

8.2 Arbitration

A master may start a transfer only if the bus is free. Two or more masters may generate a START condition within the minimum hold time ($t_{HD;STA}$) of the START condition which results in a defined START condition to the bus.

Arbitration takes place on the SDA line, while the SCL line is at the HIGH level, in such a way that the master which transmits a HIGH level, while another master is transmitting a LOW level will switch off its DATA output stage because the level on the bus doesn't correspond to its own level.

Arbitration can continue for many bits. Its first stage is comparison of the address bits (addressing information is given in Sections 10 and 14). If the masters are each trying to address the same device, arbitration continues with comparison of the data-bits if they are master-transmitter, or acknowledge-bits if they are master-receiver. Because address and data information on the I²C-bus is determined by the winning master, no information is lost during the arbitration process.

A master that loses the arbitration can generate clock pulses until the end of the byte in which it loses the arbitration.

As an Hs-mode master has a unique 8-bit master code, it will always finish the arbitration during the first byte (see Section 13).

If a master also incorporates a slave function and it loses arbitration during the addressing stage, it's possible that the winning master is trying to address it. The losing master must therefore switch over immediately to its slave mode.

Figure 9 shows the arbitration procedure for two masters. Of course, more may be involved (depending on how many masters are connected to the bus). The moment there is a difference between the internal data level of the master generating DATA 1 and the actual level on the SDA line, its data output is switched off, which means that a HIGH output level is then connected to the bus. This will not affect the data transfer initiated by the winning master.

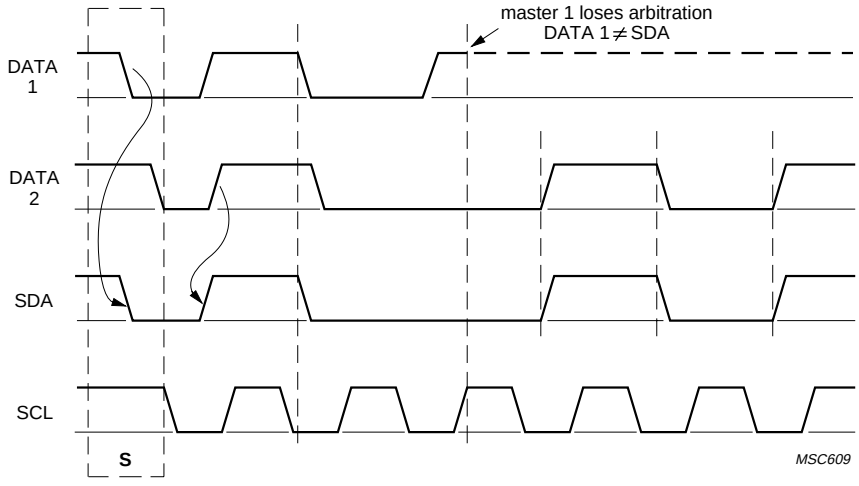


Fig.9 Arbitration procedure of two masters.

Since control of the I²C-bus is decided solely on the address or master code and data sent by competing masters, there is no central master, nor any order of priority on the bus.

Special attention must be paid if, during a serial transfer, the arbitration procedure is still in progress at the moment when a repeated START condition or a STOP condition is transmitted to the I²C-bus. If it's possible for such a situation to occur, the masters involved must send this repeated START condition or STOP condition at the same position in the format frame. In other words, arbitration isn't allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition.

Slaves are not involved in the arbitration procedure.

8.3 Use of the clock synchronizing mechanism as a handshake

In addition to being used during the arbitration procedure, the clock synchronization mechanism can be used to enable receivers to cope with fast data transfers, on either a byte level or a bit level.

On the byte level, a device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slaves can then hold the SCL line LOW after reception and acknowledgment of a byte to force the master into a wait state until the slave is ready for the next byte transfer in a type of handshake procedure (see Fig.6).

On the bit level, a device such as a microcontroller with or without limited hardware for the I²C-bus, can slow down the bus clock by extending each clock LOW period. The speed of any master is thereby adapted to the internal operating rate of this device.

In Hs-mode, this handshake feature can only be used on byte level (see Section 13).

9 FORMATS WITH 7-BIT ADDRESSES

Data transfers follow the format shown in Fig.10. After the START condition (S), a slave address is sent. This address is 7 bits long followed by an eighth bit which is a data direction bit (R/$\overline{W}$) - a 'zero' indicates a transmission (WRITE), a 'one' indicates a request for data (READ). A data transfer is always terminated by a STOP condition (P) generated by the master. However, if a master still wishes to communicate on the bus, it can generate a repeated START condition (Sr) and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such a transfer.

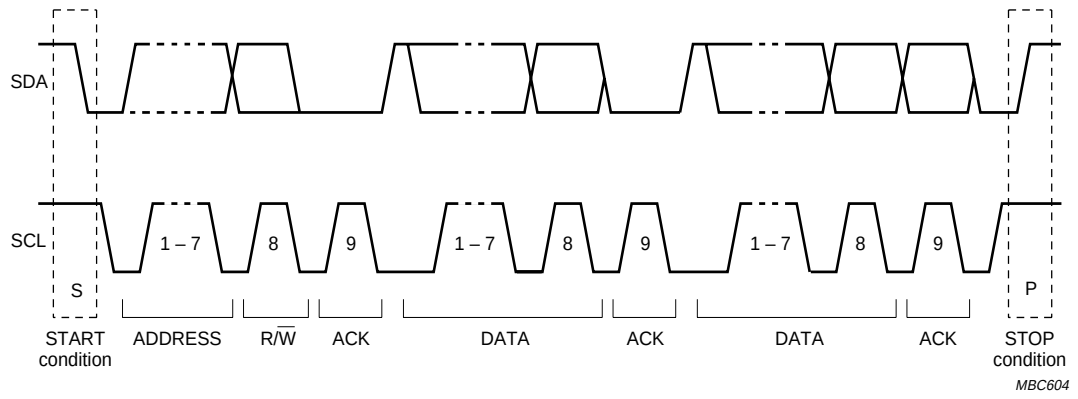


Fig.10 A complete data transfer.

The I²C-bus specification

Possible data transfer formats are:

- Master-transmitter transmits to slave-receiver. The transfer direction is not changed (see Fig.11).
- Master reads slave immediately after first byte (see Fig.12). At the moment of the first acknowledge, the master- transmitter becomes a master- receiver and the slave-receiver becomes a slave-transmitter. This first acknowledge is still generated by the slave. The STOP condition is generated by the master, which has previously sent a not-acknowledge ($\overline{A}$).
- Combined format (see Fig.13). During a change of direction within a transfer, the START condition and the slave address are both repeated, but with the R/$\overline{W}$ bit reversed. If a master receiver sends a repeated START condition, it has previously sent a not-acknowledge ($\overline{A}$).

NOTES:

1. Combined formats can be used, for example, to control a serial memory. During the first data byte, the internal memory location has to be written. After the START condition and slave address is repeated, data can be transferred.
2. All decisions on auto-increment or decrement of previously accessed memory locations etc. are taken by the designer of the device.
3. Each byte is followed by an acknowledgment bit as indicated by the A or $\overline{A}$ blocks in the sequence.
4. I²C-bus compatible devices must reset their bus logic on receipt of a START or repeated START condition such that they all anticipate the sending of a slave address, even if these START conditions are not positioned according to the proper format.
5. A START condition immediately followed by a STOP condition (void message) is an illegal format.

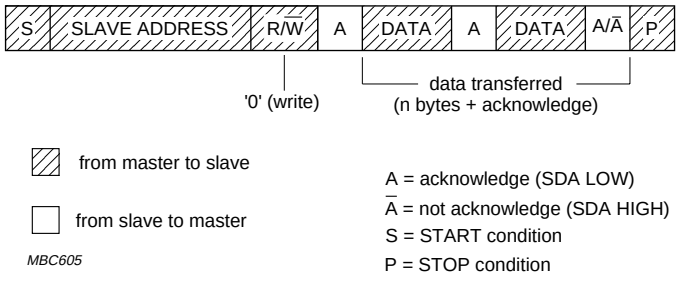


Fig.11 A master-transmitter addressing a slave receiver with a 7-bit address. The transfer direction is not changed.

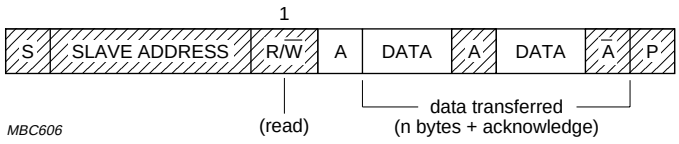


Fig.12 A master reads a slave immediately after the first byte.

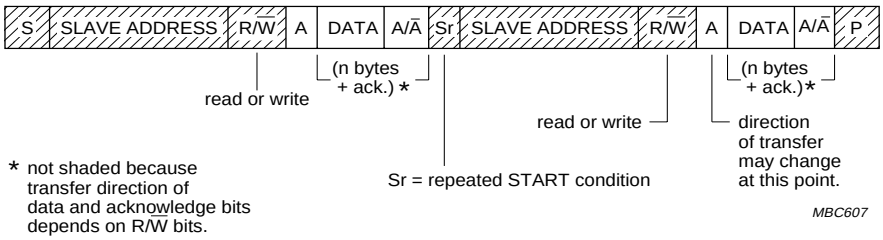


Fig.13 Combined format.

10 7-BIT ADDRESSING

The addressing procedure for the I²C-bus is such that the first byte after the START condition usually determines which slave will be selected by the master. The exception is the 'general call' address which can address all devices. When this address is used, all devices should, in theory, respond with an acknowledge. However, devices can be made to ignore this address. The second byte of the general call address then defines the action to be taken. This procedure is explained in more detail in Section 10.1.1. For information on 10-bit addressing, see Section 14

10.1 Definition of bits in the first byte

The first seven bits of the first byte make up the slave address (see Fig.14). The eighth bit is the LSB (least significant bit). It determines the direction of the message. A 'zero' in the least significant position of the first byte means that the master will write information to a selected slave. A 'one' in this position means that the master will read information from the slave.

When an address is sent, each device in a system compares the first seven bits after the START condition with its address. If they match, the device considers itself addressed by the master as a slave-receiver or slave-transmitter, depending on the R/$\overline{W}$ bit.

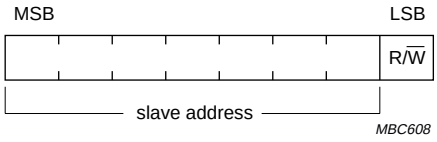


Fig.14 The first byte after the START procedure.

A slave address can be made-up of a fixed and a programmable part. Since it's likely that there will be several identical devices in a system, the programmable part of the slave address enables the maximum possible number of such devices to be connected to the I²C-bus. The number of programmable address bits of a device depends on the number of pins available. For example, if a device has 4 fixed and 3 programmable address bits, a total of 8 identical devices can be connected to the same bus.

The I²C-bus committee coordinates allocation of I²C addresses. Further information can be obtained from the Philips representatives listed on the back cover. Two groups of eight addresses (0000XXX and 1111XXX) are reserved for the purposes shown in Table 2. The bit combination 11110XX of the slave address is reserved for 10-bit addressing (see Section 14).

Table 2 Definition of bits in the first byte

SLAVE ADDRESS	R/$\overline{W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte(1)
0000 001	X	CBUS address(2)
0000 010	X	Reserved for different bus format(3)
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Notes

1. No device is allowed to acknowledge at the reception of the START byte.
2. The CBUS address has been reserved to enable the inter-mixing of CBUS compatible and I²C-bus compatible devices in the same system. I²C-bus compatible devices are not allowed to respond on reception of this address.
3. The address reserved for a different bus format is included to enable I²C and other protocols to be mixed. Only I²C-bus compatible devices that can work with such formats and protocols are allowed to respond to this address.

10.1.1 General call address

The general call address is for addressing every device connected to the I²C-bus. However, if a device doesn't need any of the data supplied within the general call structure, it can ignore this address by not issuing an acknowledgment. If a device does require data from a general call address, it will acknowledge this address and behave as a slave- receiver. The second and following bytes will be acknowledged by every slave- receiver capable of handling this data. A slave which cannot process one of these bytes must ignore it by not-acknowledging. The meaning of the general call address is always specified in the second byte (see Fig.15).

There are two cases to consider:

- When the least significant bit B is a 'zero'.
- When the least significant bit B is a 'one'.

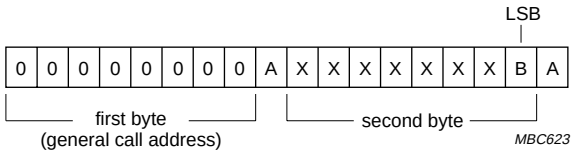


Fig.15 General call address format.

When bit B is a 'zero'; the second byte has the following definition:

- 00000110 (H'06'). Reset and write programmable part of slave address by hardware. On receiving this 2-byte sequence, all devices designed to respond to the general call address will reset and take in the programmable part of their address. Pre-cautions have to be taken to ensure that a device is not pulling down the SDA or SCL line after applying the supply voltage, since these low levels would block the bus.
- 00000100 (H'04'). Write programmable part of slave address by hardware. All devices which define the programmable part of their address by hardware (and which respond to the general call address) will latch this programmable part at the reception of this two byte sequence. The device will not reset.
- 00000000 (H'00'). This code is not allowed to be used as the second byte.

Sequences of programming procedure are published in the appropriate device data sheets.

The remaining codes have not been fixed and devices must ignore them.

When bit B is a 'one'; the 2-byte sequence is a 'hardware general call'. This means that the sequence is transmitted by a hardware master device, such as a keyboard scanner, which cannot be programmed to transmit a desired slave address. Since a hardware master doesn't know in advance to which device the message has to be transferred, it can only generate this hardware general call and its own address - identifying itself to the system (see Fig.16).

The seven bits remaining in the second byte contain the address of the hardware master. This address is recognized by an intelligent device (e.g. a microcontroller) connected to the bus which will then direct the information from the hardware master. If the hardware master can also act as a slave, the slave address is identical to the master address.

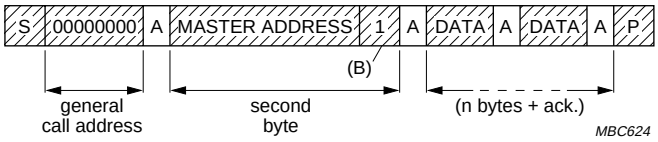


Fig.16 Data transfer from a hardware master-transmitter.

In some systems, an alternative could be that the hardware master transmitter is set in the slave-receiver mode after the system reset. In this way, a system configuring master can tell the hardware master-transmitter (which is now in slave-receiver mode) to which address data must be sent (see Fig.17). After this programming procedure, the hardware master remains in the master-transmitter mode.

10.1.2 START BYTE

Microcontrollers can be connected to the I²C-bus in two ways. A microcontroller with an on-chip hardware I²C-bus interface can be programmed to be only interrupted by requests from the bus. When the device doesn't have such an interface, it must constantly monitor the bus via software. Obviously, the more times the microcontroller monitors, or polls the bus, the less time it can spend carrying out its intended function.

There is therefore a speed difference between fast hardware devices and a relatively slow microcontroller which relies on software polling.

In this case, data transfer can be preceded by a start procedure which is much longer than normal (see Fig.18). The start procedure consists of:

- A START condition (S)
- A START byte (00000001)
- An acknowledge clock pulse (ACK)
- A repeated START condition (Sr).

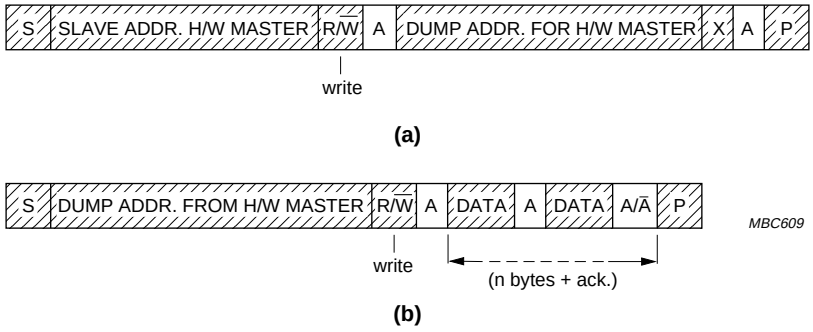


Fig.17 Data transfer by a hardware-transmitter capable of dumping data directly to slave devices.
(a) Configuring master sends dump address to hardware master
(b) Hardware master dumps data to selected slave.

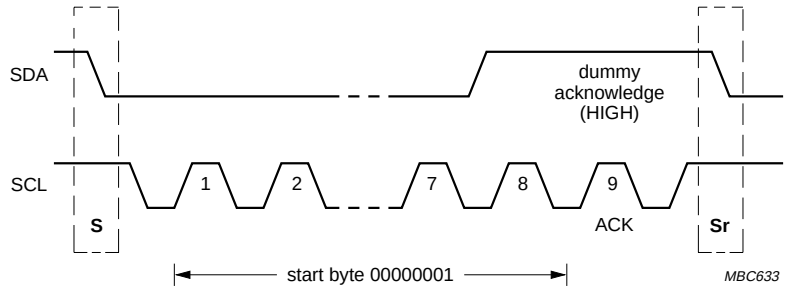


Fig.18 START byte procedure.

After the START condition S has been transmitted by a master which requires bus access, the START byte (00000001) is transmitted. Another microcontroller can therefore sample the SDA line at a low sampling rate until one of the seven zeros in the START byte is detected. After detection of this LOW level on the SDA line, the microcontroller can switch to a higher sampling rate to find the repeated START condition Sr which is then used for synchronization.

A hardware receiver will reset on receipt of the repeated START condition Sr and will therefore ignore the START byte.

An acknowledge-related clock pulse is generated after the START byte. This is present only to conform with the byte handling format used on the bus. No device is allowed to acknowledge the START byte.

10.1.3 CBUS COMPATIBILITY

CBUS receivers can be connected to the Standard-mode I²C-bus. However, a third bus line called DLEN must then be connected and the acknowledge bit omitted. Normally, I²C transmissions are sequences of 8-bit bytes; CBUS compatible devices have different formats.

In a mixed bus structure, I²C-bus devices must not respond to the CBUS message. For this reason, a special CBUS address (0000001X) to which no I²C-bus compatible device will respond, has been reserved. After transmission of the CBUS address, the DLEN line can be made active and a CBUS-format transmission sent (see Fig.19). After the STOP condition, all devices are again ready to accept data.

Master-transmitters can send CBUS formats after sending the CBUS address. The transmission is ended by a STOP condition, recognized by all devices.

NOTE: If the CBUS configuration is known, and expansion with CBUS compatible devices isn't foreseen, the designer is allowed to adapt the hold time to the specific requirements of the device(s) used.

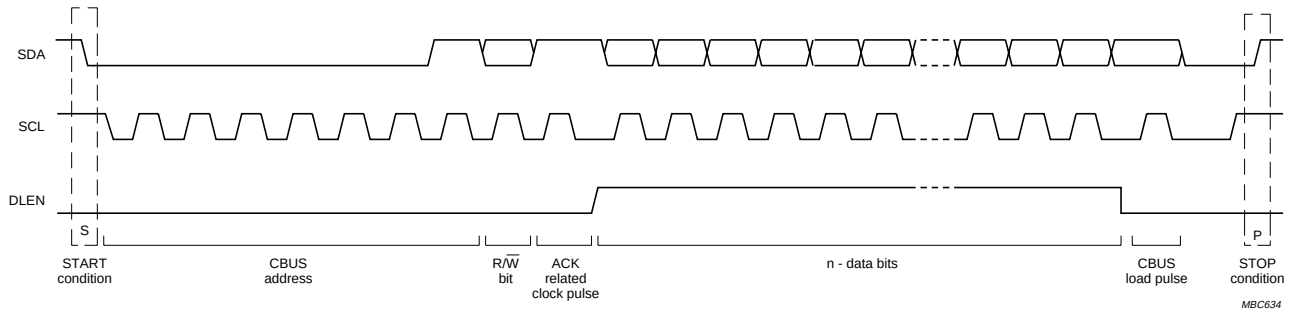


Fig.19 Data format of transmissions with CBUS transmitter/receiver.

11 EXTENSIONS TO THE STANDARD-MODE I²C-BUS SPECIFICATION

The Standard-mode I²C-bus specification, with its data transfer rate of up to 100 kbit/s and 7-bit addressing, has been in existence since the beginning of the 1980's. This concept rapidly grew in popularity and is today accepted worldwide as a de facto standard with several hundred different compatible ICs on offer from Philips Semiconductors and other suppliers. To meet the demands for higher speeds, as well as make available more slave address for the growing number of new devices, the Standard-mode I²C-bus specification was upgraded over the years and today is available with the following extensions:

- **Fast-mode**, with a bit rate up to 400 kbit/s.
- **High-speed mode (Hs-mode)**, with a bit rate up to 3.4 Mbit/s.
- **10-bit addressing**, which allows the use of up to 1024 additional slave addresses.

There are two main reasons for extending the regular I²C-bus specification:

- Many of today's applications need to transfer large amounts of serial data and require bit rates far in excess of 100 kbit/s (Standard-mode), or even 400 kbit/s (Fast-mode). As a result of continuing improvements in semiconductor technologies, I²C-bus devices are now available with bit rates of up to 3.4 Mbit/s (Hs-mode) without any noticeable increases in the manufacturing cost of the interface circuitry.
- As most of the 112 addresses available with the 7-bit addressing scheme were soon allocated, it became apparent that more address combinations were required to prevent problems with the allocation of slave addresses for new devices. This problem was resolved with the new 10-bit addressing scheme, which allowed about a tenfold increase in available addresses.

New slave devices with a Fast- or Hs-mode I²C-bus interface can have a 7- or a 10-bit slave address. If possible, a 7-bit address is preferred as it is the cheapest hardware solution and results in the shortest message length. Devices with 7- and 10-bit addresses can be mixed in the same I²C-bus system regardless of whether it is an F/S- or Hs-mode system. Both existing and future masters can generate either 7- or 10-bit addresses.

12 FAST-MODE

With the Fast-mode I²C-bus specification, the protocol, format, logic levels and maximum capacitive load for the SDA and SCL lines quoted in the Standard-mode I²C-bus specification are unchanged. New devices with an I²C-bus interface must meet at least the minimum requirements of the Fast- or Hs-mode specification (see Section 13).

Fast-mode devices can receive and transmit at up to 400 kbit/s. The minimum requirement is that they can synchronize with a 400 kbit/s transfer; they can then prolong the LOW period of the SCL signal to slow down the transfer. Fast-mode devices are downward-compatible and can communicate with Standard-mode devices in a 0 to 100 kbit/s I²C-bus system. As Standard-mode devices, however, are not upward compatible, they should not be incorporated in a Fast-mode I²C-bus system as they cannot follow the higher transfer rate and unpredictable states would occur.

The I²C-bus specification

The Fast-mode I²C-bus specification has the following additional features compared with the Standard-mode:

- The maximum bit rate is increased to 400 kbit/s.
- Timing of the serial data (SDA) and serial clock (SCL) signals has been adapted. There is no need for compatibility with other bus systems such as CBUS because they cannot operate at the increased bit rate.
- The inputs of Fast-mode devices incorporate spike suppression and a Schmitt trigger at the SDA and SCL inputs.
- The output buffers of Fast-mode devices incorporate slope control of the falling edges of the SDA and SCL signals.
- If the power supply to a Fast-mode device is switched off, the SDA and SCL I/O pins must be floating so that they don't obstruct the bus lines.
- The external pull-up devices connected to the bus lines must be adapted to accommodate the shorter maximum permissible rise time for the Fast-mode I²C-bus. For bus loads up to 200 pF, the pull-up device for each bus line can be a resistor; for bus loads between 200 pF and 400 pF, the pull-up device can be a current source (3 mA max.) or a switched resistor circuit (see Fig.43).

13 Hs-MODE

High-speed mode (Hs-mode) devices offer a quantum leap in I²C-bus transfer speeds. Hs-mode devices can transfer information at bit rates of up to 3.4 Mbit/s, yet they remain fully downward compatible with Fast- or Standard-mode (F/S-mode) devices for bi-directional communication in a mixed-speed bus system. With the exception that arbitration and clock synchronization is not performed during the Hs-mode transfer, the same serial bus protocol and data format is maintained as with the F/S-mode system. Depending on the application, new devices may have a Fast or Hs-mode I²C-bus interface, although Hs-mode devices are preferred as they can be designed-in to a greater number of applications.

13.1 High speed transfer

To achieve a bit transfer of up to 3.4 Mbit/s the following improvements have been made to the regular I²C-bus specification:

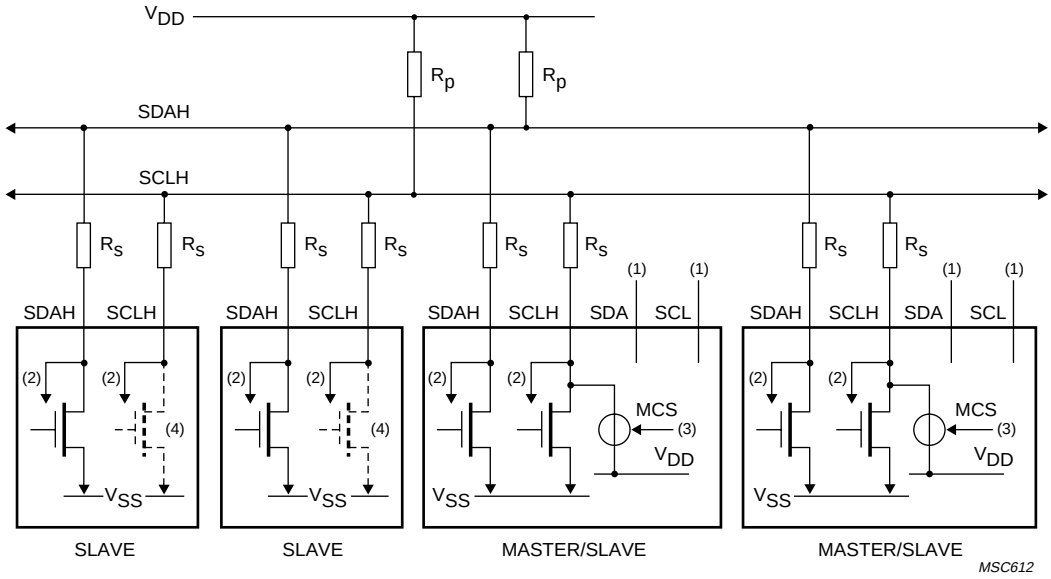
- Hs-mode master devices have an open-drain output buffer for the SDAH signal and a combination of an open-drain pull-down and current-source pull-up circuit on the SCLH output(1). This current-source circuit shortens the rise time of the SCLH signal. Only the current-source of one master is enabled at any one time, and only during Hs-mode.
- No arbitration or clock synchronization is performed during Hs-mode transfer in multi-master systems, which speeds-up bit handling capabilities. The arbitration procedure always finishes after a preceding master code transmission in F/S-mode.
- Hs-mode master devices generate a serial clock signal with a HIGH to LOW ratio of 1 to 2. This relieves the timing requirements for set-up and hold times.
- As an option, Hs-mode master devices can have a built-in bridge(1). During Hs-mode transfer, the high speed data (SDAH) and high-speed serial clock (SCLH) lines of Hs-mode devices are separated by this bridge from the SDA and SCL lines of F/S-mode devices. This reduces the capacitive load of the SDAH and SCLH lines resulting in faster rise and fall times.
- The only difference between Hs-mode slave devices and F/S-mode slave devices is the speed at which they operate. Hs-mode slaves have open-drain output buffers on the SCLH and SDAH outputs. Optional pull-down transistors on the SCLH pin can be used to stretch the LOW level of the SCLH signal, although this is only allowed after the acknowledge bit in Hs-mode transfers.
- The inputs of Hs-mode devices incorporate spike suppression and a Schmitt trigger at the SDAH and SCLH inputs.
- The output buffers of Hs-mode devices incorporate slope control of the falling edges of the SDAH and SCLH signals.

Figure 20 shows the physical I²C-bus configuration in a system with only Hs-mode devices. Pins SDA and SCL on the master devices are only used in mixed-speed bus systems and are not connected in an Hs-mode only system. In such cases, these pins can be used for other functions.

Optional series resistors R_s protect the I/O stages of the I²C-bus devices from high-voltage spikes on the bus lines and minimize ringing and interference.

Pull-up resistors R_p maintain the SDAH and SCLH lines at a HIGH level when the bus is free and ensure the signals are pulled up from a LOW to a HIGH level within the required rise time. For higher capacitive bus-line loads (>100 pF), the resistor R_p can be replaced by external current source pull-ups to meet the rise time requirements. Unless proceeded by an acknowledge bit, the rise time of the SCLH clock pulses in Hs-mode transfers is shortened by the internal current-source pull-up circuit MCS of the active master.

(1) Patent application pending.



(1) SDA and SCL are not used here but may be used for other functions.
(2) To input filter.
(3) Only the active master can enable its current-source pull-up circuit
(4) Dotted transistors are optional open-drain outputs which can stretch the serial clock signal SCLH.

Fig.20 I²C-bus configuration with Hs-mode devices only.

13.2 Serial data transfer format in Hs-mode

Serial data transfer format in Hs-mode meets the Standard-mode I²C-bus specification. Hs-mode can only commence after the following conditions (all of which are in F/S-mode):

1. START condition (S)
2. 8-bit master code (00001XXX)
3. not-acknowledge bit ($\overline{A}$)

Figures 21 and 22 show this in more detail. This master code has two main functions:

- It allows arbitration and synchronization between competing masters at F/S-mode speeds, resulting in one winning master.
- It indicates the beginning of an Hs-mode transfer.

Hs-mode master codes are reserved 8-bit codes, which are not used for slave addressing or other purposes. Furthermore, as each master has its own unique master code, up to eight Hs-mode masters can be present on the one I²C-bus system (although master code 0000 1000 should be reserved for test and diagnostic purposes). The master code for an Hs-mode master device is software programmable and is chosen by the System Designer.

Arbitration and clock synchronization only take place during the transmission of the master code and not-acknowledge bit ($\overline{A}$), after which one winning master remains active. The master code indicates to other devices that an Hs-mode transfer is to begin and the connected devices must meet the Hs-mode specification. As no device is allowed to acknowledge the master code, the master code is followed by a not-acknowledge ($\overline{A}$).

After the not-acknowledge bit ($\overline{A}$), and the SCLH line has been pulled-up to a HIGH level, the active master switches to Hs-mode and enables (at time t_H, see Fig.22) the current-source pull-up circuit for the SCLH signal. As other devices can delay the serial transfer before t_H by stretching the LOW period of the SCLH signal, the active master will enable its current-source pull-up circuit when all devices have released the SCLH line and the SCLH signal has reached a HIGH level, thus speeding up the last part of the rise time of the SCLH signal.

The active master then sends a repeated START condition (Sr) followed by a 7-bit slave address (or 10-bit slave

address, see Section 14) with a R/$\overline{W}$ bit address, and receives an acknowledge bit (A) from the selected slave.

After a repeated START condition and after each acknowledge bit (A) or not-acknowledge bit ($\overline{A}$), the active master disables its current-source pull-up circuit. This enables other devices to delay the serial transfer by stretching the LOW period of the SCLH signal. The active master re-enables its current-source pull-up circuit again when all devices have released and the SCLH signal reaches a HIGH level, and so speeds up the last part of the SCLH signal's rise time.

Data transfer continues in Hs-mode after the next repeated START (Sr), and only switches back to F/S-mode after a STOP condition (P). To reduce the overhead of the master code, it's possible that a master links a number of Hs-mode transfers, separated by repeated START conditions (Sr).

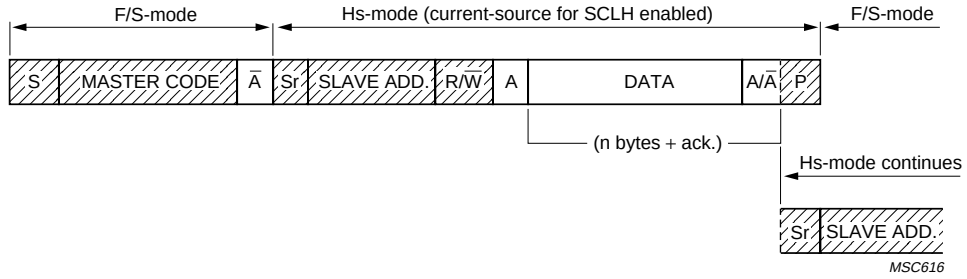


Fig.21 Data transfer format in Hs-mode.

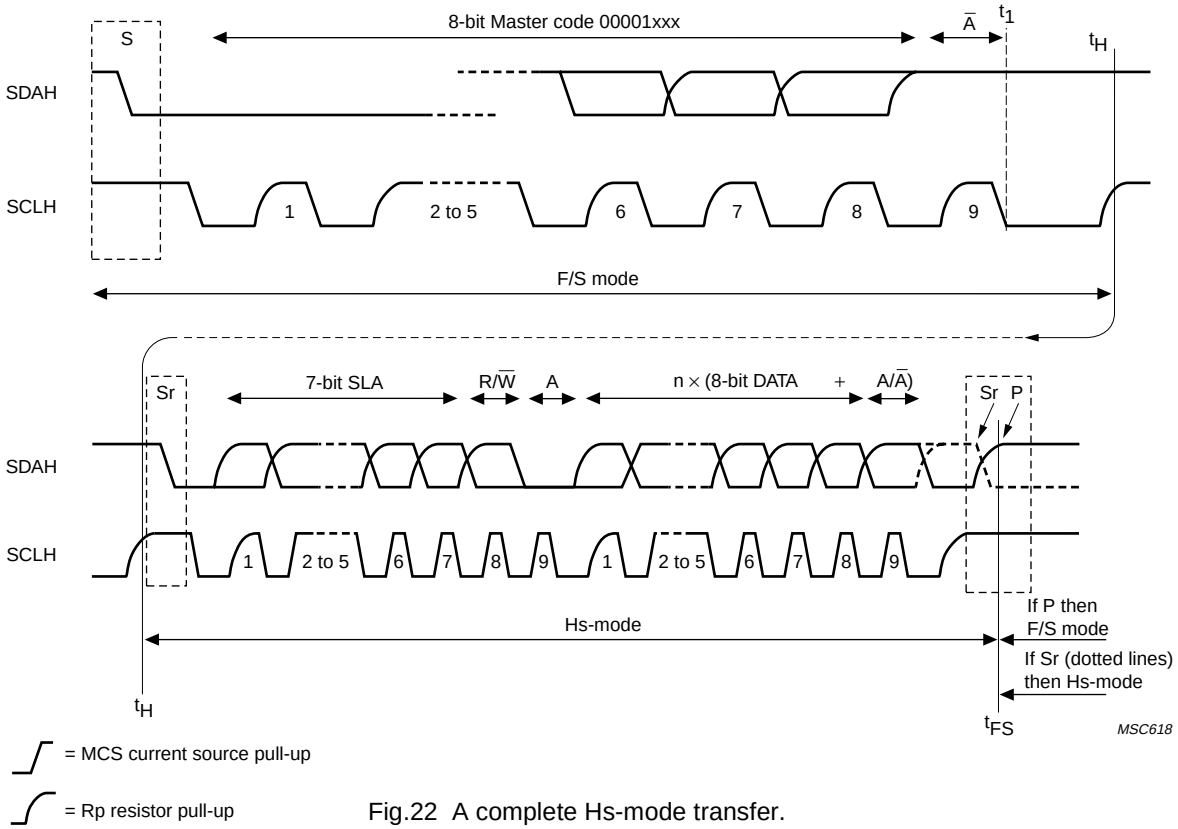


Fig.22 A complete Hs-mode transfer.

13.3 Switching from F/S- to Hs-mode and back

After reset and initialization, Hs-mode devices must be in Fast-mode (which is in effect F/S-mode as Fast-mode is downward compatible with Standard-mode). Each Hs-mode device can switch from Fast- to Hs-mode and back and is controlled by the serial transfer on the I²C-bus.

Before time t_1 in Fig.22, each connected device operates in Fast-mode. Between times t_1 and t_H (this time interval can be stretched by any device) each connected device must recognize the "S 00001XXX $\overline{A}$" sequence and has to switch its internal circuit from the Fast-mode setting to the Hs-mode setting. Between times t_1 and t_H the connected master and slave devices perform this switching by the following actions.

The active (winning) master:

1. Adapts its SDAH and SCLH input filters according to the spike suppression requirement in Hs-mode.
2. Adapts the set-up and hold times according to the Hs-mode requirements.
3. Adapts the slope control of its SDAH and SCLH output stages according to the Hs-mode requirement.
4. Switches to the Hs-mode bit-rate, which is required after time t_H.
5. Enables the current source pull-up circuit of its SCLH output stage at time t_H.

The non-active, or losing masters:

1. Adapt their SDAH and SCLH input filters according to the spike suppression requirement in Hs-mode.
2. Wait for a STOP condition to detect when the bus is free again.

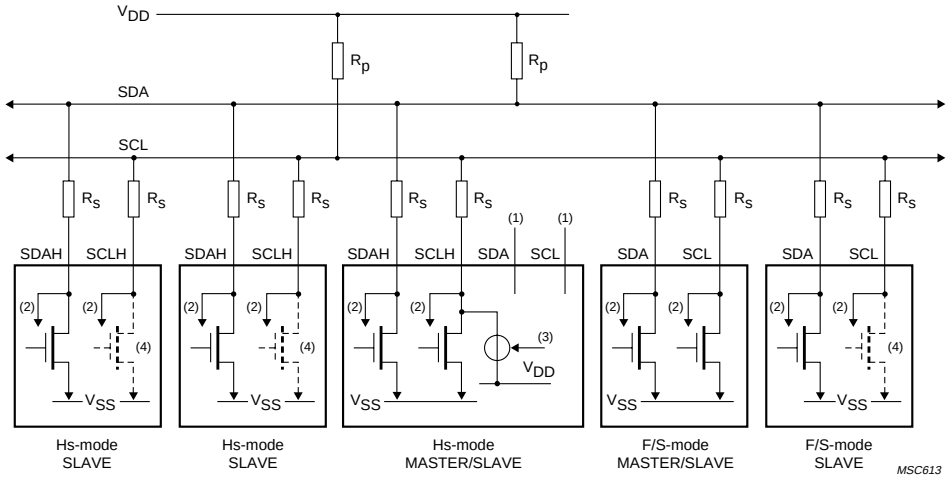
All slaves:

1. Adapt their SDAH and SCLH input filters according to the spike suppression requirement in Hs-mode.
2. Adapt the set-up and hold times according to the Hs-mode requirements. This requirement may already be fulfilled by the adaptation of the input filters.
3. Adapt the slope control of their SDAH output stages, if necessary. For slave devices, slope control is applicable for the SDAH output stage only and, depending on circuit tolerances, both the Fast- and Hs-mode requirements may be fulfilled without switching its internal circuit.

At time t_{FS} in Fig.22, each connected device must recognize the STOP condition (P) and switch its internal circuit from the Hs-mode setting back to the Fast-mode setting as present before time t_1. This must be completed within the minimum bus free time as specified in Table 5 according to the Fast-mode specification.

13.4 Hs-mode devices at lower speed modes

Hs-mode devices are fully downwards compatible, and can be connected to an F/S-mode I²C-bus system (see Fig.23). As no master code will be transmitted in such a configuration, all Hs-mode master devices stay in F/S-mode and communicate at F/S-mode speeds with their current-source disabled. The SDAH and SCLH pins are used to connect to the F/S-mode bus system, allowing the SDA and SCL pins (if present) on the Hs-mode master device to be used for other functions.



(1) Bridge not used. SDA and SCL may have an alternative function.

(2) To input filter.

(3) The current-source pull-up circuit stays disabled.

(4) Dotted transistors are optional open-drain outputs which can stretch the serial clock signal SCL.

Fig.23 Hs-mode devices at F/S-mode speed.

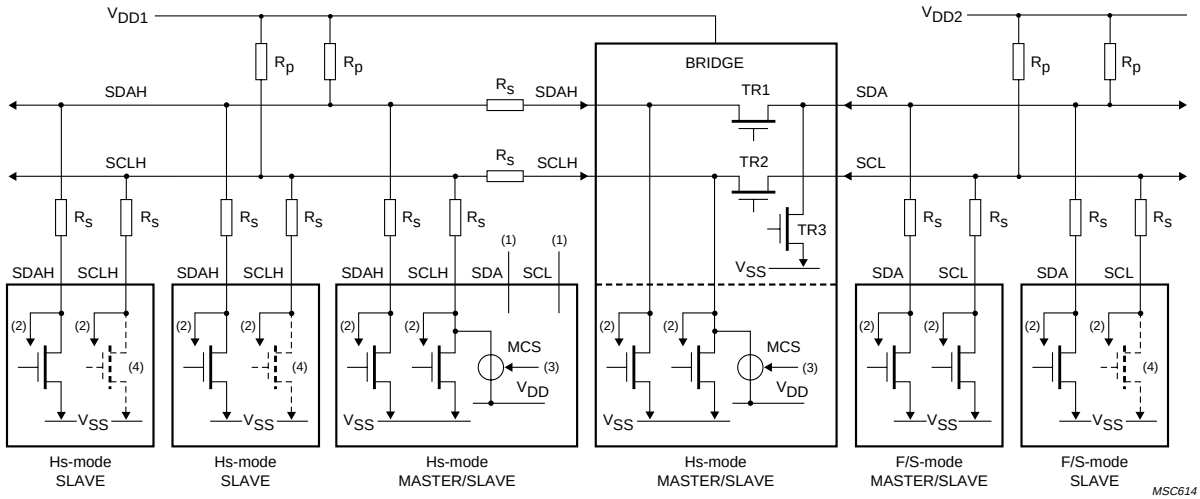
13.5 Mixed speed modes on one serial bus system

If a system has a combination of Hs-, Fast- and/or Standard-mode devices, it's possible, by using an interconnection bridge, to have different bit rates between different devices (see Figs 24 and 25).

One bridge is required to connect/disconnect an Hs-mode section to/from an F/S-mode section at the appropriate time. This bridge includes a level shift function that allows devices with different supply voltages to be connected. For example F/S-mode devices with a V_{DD2} of 5 V can be connected to Hs-mode devices with a V_{DD1} of 3 V or less (i.e. where $V_{DD2} \geq V_{DD1}$), provided SDA and SCL pins are 5 V tolerant. This bridge is incorporated in Hs-mode master devices and is completely controlled by the serial signals SDAH, SCLH, SDA and SCL. Such a bridge can be implemented in any IC as an autonomous circuit.

TR1, TR2 and TR3 are N-channel transistors. TR1 and TR2 have a transfer gate function, and TR3 is an open-drain pull-down stage. If TR1 or TR2 are switched on they transfer a LOW level in both directions, otherwise when both the drain and source rise to a HIGH level there will be a high impedance between the drain and source of each switched on transistor. In the latter case, the transistors will act as a level shifter as SDAH and SCLH will be pulled-up to V_{DD1} and SDA and SCL will be pulled-up to V_{DD2}

During F/S-mode speed, a bridge on one of the Hs-mode masters connects the SDAH and SCLH lines to the corresponding SDA and SCL lines thus permitting Hs-mode devices to communicate with F/S-mode devices at slower speeds. Arbitration and synchronization is possible during the total F/S-mode transfer between all connected devices as described in Section 8. During Hs-mode transfer, however, the bridge opens to separate the two bus sections and allows Hs-mode devices to communicate with each other at 3.4 Mbit/s. Arbitration between Hs-mode devices and F/S-mode devices is only performed during the master code (00001XXX), and normally won by one Hs-mode master as no slave address has four leading zeros. Other masters can win the arbitration only if they send a reserved 8-bit code (00000XXX). In such cases, the bridge remains closed and the transfer proceeds in F/S-mode. Table 3 gives the possible communication speeds in such a system.



(1) Bridge not used. SDA and SCL may have an alternative function.

(2) To input filter.

(3) Only the active master can enable its current-source pull-up circuit.

(4) Dotted transistors are optional open-drain outputs which can stretch the serial clock signal SCL or SCLH.

Fig.24 Bus system with transfer at Hs- and F/S-mode speeds.

Table 3 Communication bit-rates in a mixed speed bus system

TRANSFER BETWEEN	SERIAL BUS SYSTEM CONFIGURATION			
	Hs + FAST + STANDARD	Hs + FAST	Hs + STANDARD	FAST + STANDARD
Hs <–> Hs	0 to 3.4 Mbit/s	0 to 3.4 Mbit/s	0 to 3.4 Mbit/s	–
Hs <–> Fast	0 to 100 kbit/s	0 to 400 kbit/s	–	–
Hs <–> Standard	0 to 100 kbit/s	–	0 to 100 kbit/s	–
Fast <–> Standard	0 to 100 kbit/s	–	–	0 to 100 kbit/s
Fast <–> Fast	0 to 100 kbit/s	0 to 400 kbit/s	–	0 to 100 kbit/s
Standard <–> Standard	0 to 100 kbit/s	–	0 to 100 kbit/s	0 to 100 kbit/s

13.5.1 F/S-MODE TRANSFER IN A MIXED-SPEED BUS SYSTEM

The bridge shown in Fig.24 interconnects corresponding serial bus lines, forming one serial bus system. As no master code (00001XXX) is transmitted, the current-source pull-up circuits stay disabled and all output stages are open-drain. All devices, including Hs-mode devices, communicate with each other according the protocol, format and speed of the F/S-mode I²C-bus specification.

13.5.2 Hs-MODE TRANSFER IN A MIXED-SPEED BUS SYSTEM

Figure 25 shows the timing diagram of a complete Hs-mode transfer, which is invoked by a START condition, a master code, and a not-acknowledge $\overline{A}$ (at F/S-mode speed). Although this timing diagram is split in two parts, it should be viewed as one timing diagram were time point t_H is a common point for both parts.

The I²C-bus specification

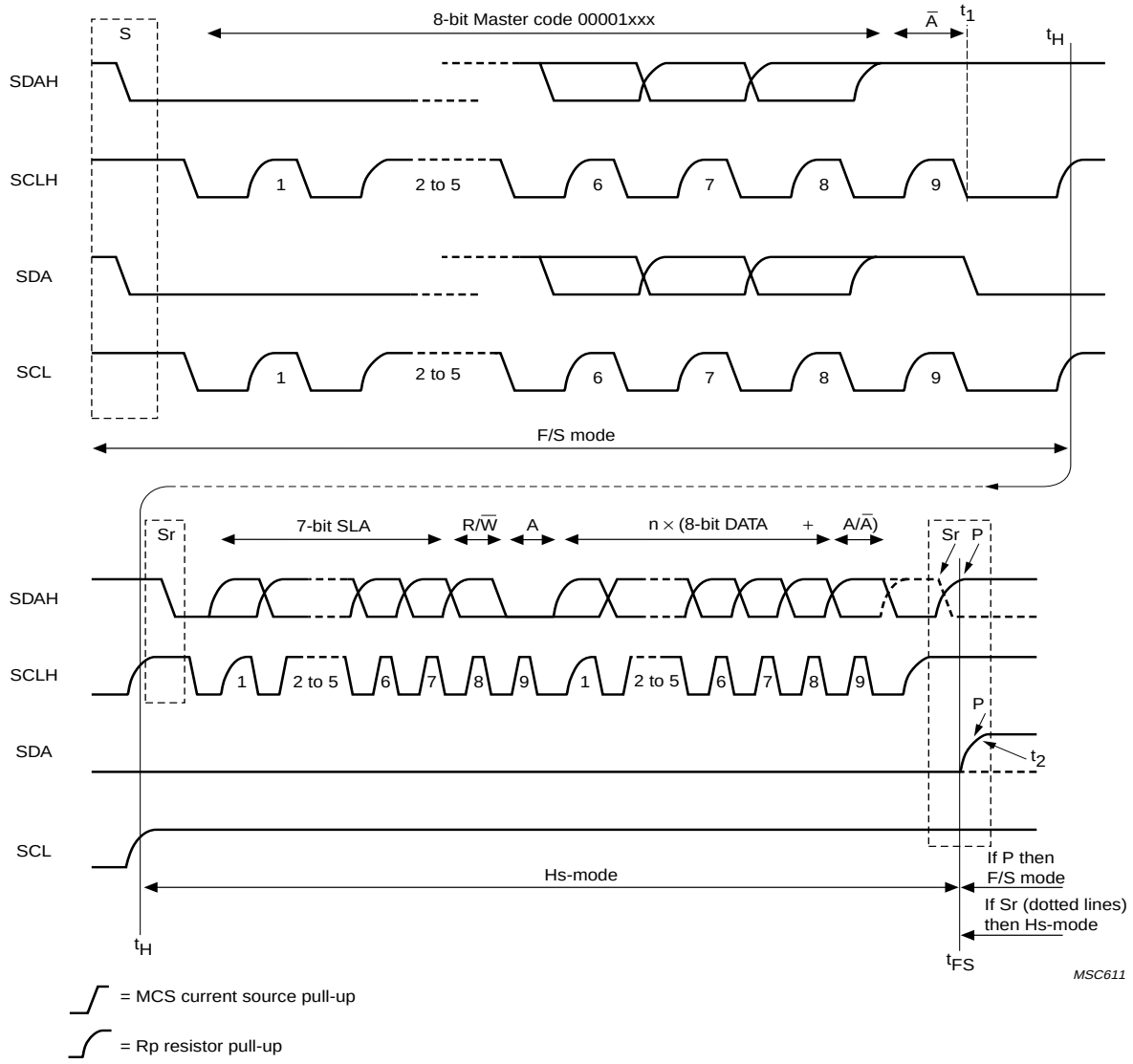


Fig.25 A complete Hs-mode transfer in a mixed-speed bus system.

The master code is recognized by the bridge in the active or non-active master (see Fig.24). The bridge performs the following actions:

1. Between t_1 and t_H (see Fig.25), transistor TR1 opens to separate the SDAH and SDA lines, after which transistor TR3 closes to pull-down the SDA line to V_{SS}.
2. When both SCLH and SCL become HIGH (t_H in Fig.25), transistor TR2 opens to separate the SCLH and SCL lines. TR2 must be opened before SCLH goes LOW after Sr.

Hs-mode transfer starts after t_H with a repeated START condition (Sr). During Hs-mode transfer, the SCL line stays at a HIGH and the SDA line at a LOW steady-state level, and so is prepared for the transfer of a STOP condition (P).

After each acknowledge (A) or not-acknowledge bit ($\overline{A}$) the active master disables its current-source pull-up circuit. This enables other devices to delay the serial transfer by stretching the LOW period of the SCLH signal. The active master re-enables its current-source pull-up circuit again when all devices are released and the SCLH signal reaches a HIGH level, and so speeds up the last part of the SCLH signal's rise time. In irregular situations, F/S-mode devices can close the bridge (TR1 and TR2 closed, TR3 open) at any time by pulling down the SCL line for at least 1 μs, e.g. to recover from a bus hang-up.

Hs-mode finishes with a STOP condition and brings the bus system back into the F/S-mode. The active master disables its current-source MCS when the STOP condition (P) at SDAH is detected (t_{FS} in Fig.25). The bridge also recognizes this STOP condition and takes the following actions:

1. Transistor TR2 closes after t_{FS} to connect SCLH with SCL; both of which are HIGH at this time. Transistor TR3 opens after t_{FS}, which releases the SDA line and allows it to be pulled HIGH by the pull-up resister R_p. This is the STOP condition for the F/S-mode devices. TR3 must open fast enough to ensure the bus free time between the STOP condition and the earliest next START condition is according to the Fast-mode specification (see t_{BUF} in Table 5).
2. When SDA reaches a HIGH (t_2 in Fig.25) transistor TR1 closes to connect SDAH with SDA. (Note: interconnections are made when all lines are HIGH, thus preventing spikes on the bus lines). TR1 and TR2 must be closed within the minimum bus free time according to the Fast-mode specification (see t_{BUF} in Table 5).

13.5.3 Timing requirements for the bridge in a mixed-speed bus system

It can be seen from Fig.25 that the actions of the bridge at t_1, t_H and t_{FS} must be so fast that it does not affect the SDAH and SCLH lines. Furthermore the bridge must meet the related timing requirements of the Fast-mode specification for the SDA and SCL lines.

14 10-BIT ADDRESSING

This section describes 10-bit addressing and can be disregarded if only 7-bit addressing is used.

10-bit addressing is compatible with, and can be combined with, 7-bit addressing. Using 10 bits for addressing exploits the reserved combination 1111XXX for the first seven bits of the first byte following a START (S) or repeated START (Sr) condition as explained in Section 10.1. The 10-bit addressing does not affect the existing 7-bit addressing. Devices with 7-bit and 10-bit addresses can be connected to the same I²C-bus, and both 7-bit and 10-bit addressing can be used in F/S-mode and Hs-mode systems.

Although there are eight possible combinations of the reserved address bits 1111XXX, only the four combinations 11110XX are used for 10-bit addressing. The remaining four combinations 11111XX are reserved for future I²C-bus enhancements.

14.1 Definition of bits in the first two bytes

The 10-bit slave address is formed from the first two bytes following a START condition (S) or a repeated START condition (Sr).

The first seven bits of the first byte are the combination 11110XX of which the last two bits (XX) are the two most-significant bits (MSBs) of the 10-bit address; the eighth bit of the first byte is the R/$\overline{W}$ bit that determines the direction of the message. A 'zero' in the least significant position of the first byte means that the master will write information to a selected slave. A 'one' in this position means that the master will read information from the slave.

If the R/$\overline{W}$ bit is 'zero', then the second byte contains the remaining 8 bits (XXXXXXXX) of the 10-bit address. If the R/$\overline{W}$ bit is 'one', then the next byte contains data transmitted from a slave to a master.

14.2 Formats with 10-bit addresses

Various combinations of read/write formats are possible within a transfer that includes 10-bit addressing. Possible data transfer formats are:

- Master-transmitter transmits to slave-receiver with a 10-bit slave address.
  The transfer direction is not changed (see Fig.26). When a 10-bit address follows a START condition, each slave compares the first seven bits of the first byte of the slave address (11110XX) with its own address and tests if the eighth bit (R/$\overline{W}$ direction bit) is 0. It is possible that more than one device will find a match and generate an acknowledge (A1). All slaves that found a match will compare the eight bits of the second byte of the slave address (XXXXXXXX) with their own addresses, but only one slave will find a match and generate an acknowledge (A2). The matching slave will remain addressed by the master until it receives a STOP

condition (P) or a repeated START condition (Sr) followed by a different slave address.

- Master-receiver reads slave- transmitter with a 10-bit slave address.
  The transfer direction is changed after the second R/$\overline{W}$ bit (Fig.27). Up to and including acknowledge bit A2, the procedure is the same as that described for a master-transmitter addressing a slave-receiver. After the repeated START condition (Sr), a matching slave remembers that it was addressed before. This slave then checks if the first seven bits of the first byte of the slave address following Sr are the same as they were after the START condition (S), and tests if the eighth (R/$\overline{W}$) bit is 1. If there is a match, the slave considers that it has been addressed as a transmitter and generates acknowledge A3. The slave-transmitter remains addressed until it receives a STOP condition (P) or until it receives another repeated START condition (Sr) followed by a different slave address. After a repeated START condition (Sr), all the other slave devices will also compare the first seven bits of the first byte of the slave address (11110XX) with their own addresses and test the eighth (R/$\overline{W}$) bit. However, none of them will be addressed because R/$\overline{W}$ = 1 (for 10-bit devices), or the 11110XX slave address (for 7-bit devices) does not match.
- Combined format. A master transmits data to a slave and then reads data from the same slave (Fig.28). The same master occupies the bus all the time. The transfer direction is changed after the second R/$\overline{W}$ bit.
- Combined format. A master transmits data to one slave and then transmits data to another slave (Fig.29). The same master occupies the bus all the time.
- Combined format. 10-bit and 7-bit addressing combined in one serial transfer (Fig.30). After each START condition (S), or each repeated START condition (Sr), a 10-bit or 7-bit slave address can be transmitted. Figure 30 shows how a master transmits data to a slave with a 7-bit address and then transmits data to a second slave with a 10-bit address. The same master occupies the bus all the time.

NOTES:

1. Combined formats can be used, for example, to control a serial memory. During the first data byte, the internal memory location has to be written. After the START condition and slave address is repeated, data can be transferred.
2. All decisions on auto-increment or decrement of previously accessed memory locations etc. are taken by the designer of the device.
3. Each byte is followed by an acknowledgment bit as indicated by the A or blocks in the sequence.
4. I²C-bus compatible devices must reset their bus logic on receipt of a START or repeated START condition such that they all anticipate the sending of a slave address.

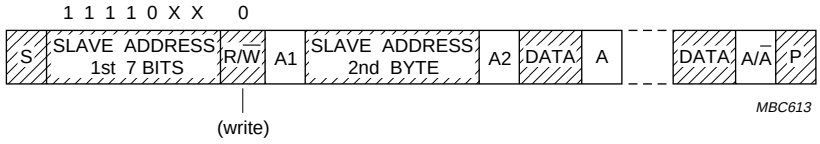


Fig.26 A master-transmitter addresses a slave-receiver with a 10-bit address.

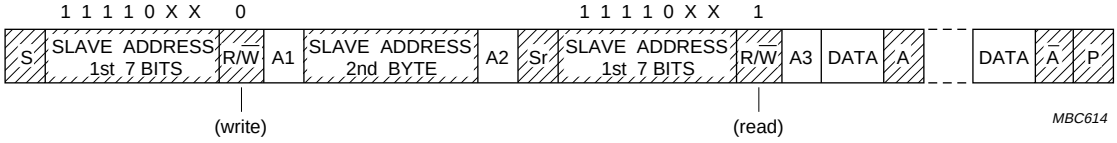


Fig.27 A master-receiver addresses a slave-transmitter with a 10-bit address.

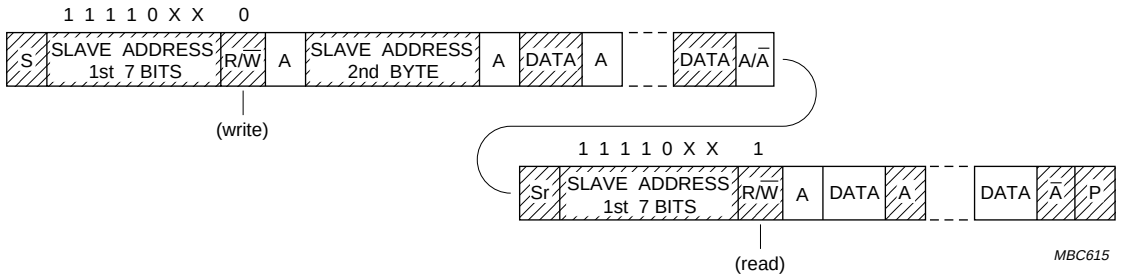


Fig.28 Combined format. A master addresses a slave with a 10-bit address, then transmits data to this slave and reads data from this slave.

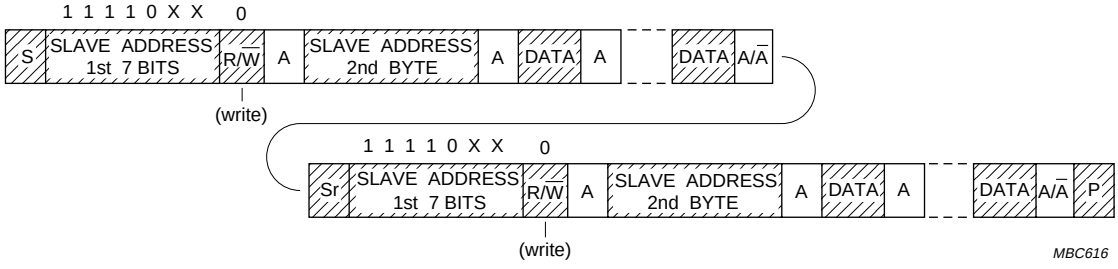


Fig.29 Combined format. A master transmits data to two slaves, both with 10-bit addresses.

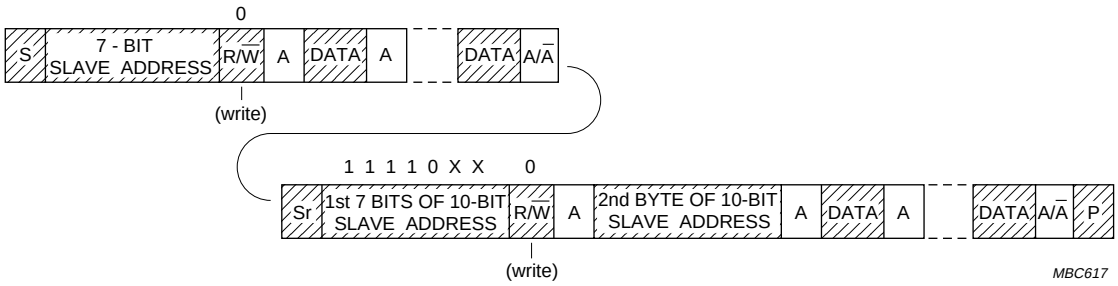


Fig.30 Combined format. A master transmits data to two slaves, one with a 7-bit address, and one with a 10-bit address.

14.3 General call address and start byte with 10-bit addressing

The 10-bit addressing procedure for the I²C-bus is such that the first two bytes after the START condition (S) usually determine which slave will be selected by the master. The exception is the “general call” address 00000000 (H‘00’). Slave devices with 10-bit addressing will react to a “general call” in the same way as slave devices with 7-bit addressing (see Section 10.1.1).

Hardware masters can transmit their 10-bit address after a ‘general call’. In this case, the ‘general call’ address byte is followed by two successive bytes containing the 10-bit address of the master-transmitter. The format is as shown in Fig.10 where the first DATA byte contains the eight least-significant bits of the master address.

The START byte 00000001 (H‘01’) can precede the 10-bit addressing in the same way as for 7-bit addressing (see Section 10.1.2).

15 ELECTRICAL SPECIFICATIONS AND TIMING FOR I/O STAGES AND BUS LINES

15.1 Standard- and Fast-mode devices

The I/O levels, I/O current, spike suppression, output slope control and pin capacitance for F/S-mode I²C-bus devices are given in Table 4. The I²C-bus timing characteristics, bus-line capacitance and noise margin are given in Table 5. Figure 31 shows the timing definitions for the I²C-bus.

The minimum HIGH and LOW periods of the SCL clock specified in Table 5 determine the maximum bit transfer rates of 100 kbit/s for Standard-mode devices and 400 kbit/s for Fast-mode devices. Standard-mode and Fast-mode I²C-bus devices must be able to follow transfers at their own maximum bit rates, either by being able to transmit or receive at that speed or by applying the clock synchronization procedure described in Section 8 which will force the master into a wait state and stretch the LOW period of the SCL signal. Of course, in the latter case the bit transfer rate is reduced.

The I²C-bus specification

Table 4 Characteristics of the SDA and SCL I/O stages for F/S-mode I²C-bus devices

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
LOW level input voltage: fixed input levels V_{DD}-related input levels	V_{IL}	 –0.5 –0.5	 1.5 $0.3V_{DD}$	 n/a –0.5	 n/a $0.3V_{DD}$ (1)	 V V
HIGH level input voltage: fixed input levels V_{DD}-related input levels	V_{IH}	 3.0 $0.7V_{DD}$	 (2) (2)	 n/a $0.7V_{DD}$(1)	 n/a (2)	 V V
Hysteresis of Schmitt trigger inputs: V_{DD} > 2 V V_{DD} < 2 V	V_{hys}	 n/a n/a	 n/a n/a	 $0.05V_{DD}$ $0.1V_{DD}$	 – –	 V V
LOW level output voltage (open drain or open collector) at 3 mA sink current: V_{DD} > 2 V V_{DD} < 2 V	 V_{OL1} V_{OL3}	 0 n/a	 0.4 n/a	 0 0	 0.4 $0.2V_{DD}$	 V V
Output fall time from V_{IHmin} to V_{ILmax} with a bus capacitance from 10 pF to 400 pF	t_{of}	–	250(4)	$20 + 0.1C_b$(3)	250(4)	ns
Pulse width of spikes which must be suppressed by the input filter	t_{SP}	n/a	n/a	0	50	ns
Input current each I/O pin with an input voltage between $0.1V_{DD}$ and $0.9V_{DDmax}$	I_i	–10	10	–10(5)	10(5)	µA
Capacitance for each I/O pin	C_i	–	10	–	10	pF

Notes

1. Devices that use non-standard supply voltages which do not conform to the intended I²C-bus system levels must relate their input levels to the V_{DD} voltage to which the pull-up resistors R_p are connected.
2. Maximum $V_{IH} = V_{DDmax} + 0.5$ V.
3. C_b = capacitance of one bus line in pF.
4. The maximum t_f for the SDA and SCL bus lines quoted in Table 5 (300 ns) is longer than the specified maximum t_{of} for the output stages (250 ns). This allows series protection resistors (R_s) to be connected between the SDA/SCL pins and the SDA/SCL bus lines as shown in Fig.36 without exceeding the maximum specified t_f.
5. I/O pins of Fast-mode devices must not obstruct the SDA and SCL lines if V_{DD} is switched off.

n/a = not applicable

Table 5 Characteristics of the SDA and SCL bus lines for F/S-mode I²C-bus devices(1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	$20 + 0.1C_b$(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	$20 + 0.1C_b$(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs
Capacitive load for each bus line	C_b	–	400	–	400	pF
Noise margin at the LOW level for each connected device (including hysteresis)	V_{nL}	$0.1V_{DD}$	–	$0.1V_{DD}$	–	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V_{nH}	$0.2V_{DD}$	–	$0.2V_{DD}$	–	V

Notes

1. All values referred to V_{IHmin} and V_{ILmax} levels (see Table 4).
2. A device must internally provide a hold time of at least 300 ns for the SDA signal (referred to the V_{IHmin} of the SCL signal) to bridge the undefined region of the falling edge of SCL.
3. The maximum $t_{HD;DAT}$ has only to be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal.
4. A Fast-mode I²C-bus device can be used in a Standard-mode I²C-bus system, but the requirement $t_{SU;DAT} \geq 250$ ns must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line $t_{r\,max} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the Standard-mode I²C-bus specification) before the SCL line is released.
5. C_b = total capacitance of one bus line in pF. If mixed with Hs-mode devices, faster fall-times according to Table 6 are allowed.

n/a = not applicable

The I²C-bus specification

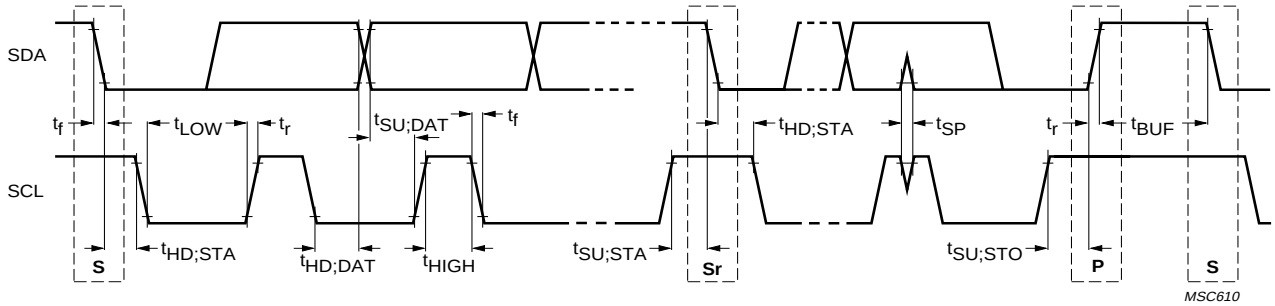


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

15.2 Hs-mode devices

The I/O levels, I/O current, spike suppression, output slope control and pin capacitance for I²C-bus Hs-mode devices are given in Table 6. The noise margin for HIGH and LOW levels on the bus lines are the same as specified for F/S-mode I²C-bus devices.

Figure 32 shows all timing parameters for the Hs-mode timing.The “normal” START condition S does not exist in Hs-mode. Timing parameters for Address bits, R/$\overline{W}$ bit, Acknowledge bit and DATA bits are all the same. Only the rising edge of the first SCLH clock signal after an acknowledge bit has an larger value because the external R_p has to pull-up SCLH without the help of the internal current-source.

The Hs-mode timing parameters for the bus lines are specified in Table 7. The minimum HIGH and LOW periods and the maximum rise and fall times of the SCLH clock signal determine the highest bit rate.

With an internally generated SCLH signal with LOW and HIGH level periods of 200 ns and 100 ns respectively, an Hs-mode master can fulfil the timing requirements for the external SCLH clock pulses (taking the rise and fall times into account) for the maximum bit rate of 3.4 Mbit/s. So a basic frequency of 10 MHz, or a multiple of 10 MHz, can be used by an Hs-mode master to generate the SCLH signal. There are no limits for maximum HIGH and LOW periods of the SCLH clock, and there is no limit for a lowest bit rate.

Timing parameters are independent for capacitive load up to 100 pF for each bus line allowing the maximum possible bit rate of 3.4 Mbit/s. At a higher capacitive load on the bus lines, the bit rate decreases gradually. The timing parameters for a capacitive bus load of 400 pF are specified in Table 7, allowing a maximum bit rate of 1.7 Mbit/s. For capacitive bus loads between 100 pF and 400 pF, the timing parameters must be interpolated linearly. Rise and fall times are in accordance with the maximum propagation time of the transmission lines SDAH and SCLH to prevent reflections of the open ends.

Table 6 Characteristics of the SDAH, SCLH, SDA and SCL I/O stages for Hs-mode I²C-bus devices

PARAMETER	SYMBOL	Hs-MODE		UNIT
		MIN.	MAX.	
LOW level input voltage	V_{IL}	−0.5	$0.3V_{DD}$(1)	V
HIGH level input voltage	V_{IH}	$0.7V_{DD}$(1)	V_{DD} + 0.5(2)	V
Hysteresis of Schmitt trigger inputs	V_{hys}	$0.1V_{DD}$(1)	–	V
LOW level output voltage (open drain) at 3 mA sink current at SDAH, SDA and SCLH for: V_{DD} > 2 V V_{DD} < 2 V	V_{OL}	 0 0	 0.4 $0.2V_{DD}$	 V V
On resistance of the transfer gate, for both current directions at V_{OL} level between SDA and SDAH or SCL and SCLH at 3 mA	R_{onL}	–	50	Ω
On resistance of the transfer gate between SDA and SDAH or SCL and SCLH if both are at V_{DD} level	R_{onH}(2)	50	–	kΩ
Pull-up current of the SCLH current-source. Applies for SCLH output levels between $0.3V_{DD}$ and $0.7V_{DD}$	I_{CS}	3	12	mA
Output rise time (current-source enabled) and fall time at SCLH with a capacitive load from 10 to 100 pF	t_{rCL}, t_{fCL}	10	40	ns
Output rise time (current-source enabled) and fall time at SCLH with an external pull-up current source of 3 mA and a capacitive load of 400 pF	t_{rCL}(3), t_{fCL}(3)	20	80	ns
Output fall time at SDAH with a capacitive load from 10 to 100 pF	t_{fDA}	10	80	ns
Output fall time at SDAH with a capacitive load of 400 pF	t_{fDA}(3)	20	160	ns
Pulse width of spikes at SDAH and SCLH that must be suppressed by the input filters	t_{SP}	0	10	ns
Input current each I/O pin with an input voltage between $0.1V_{DD}$ and $0.9V_{DD}$	I_i(4)	–	10	μA
Capacitance for each I/O pin	C_i	–	10	pF

Notes

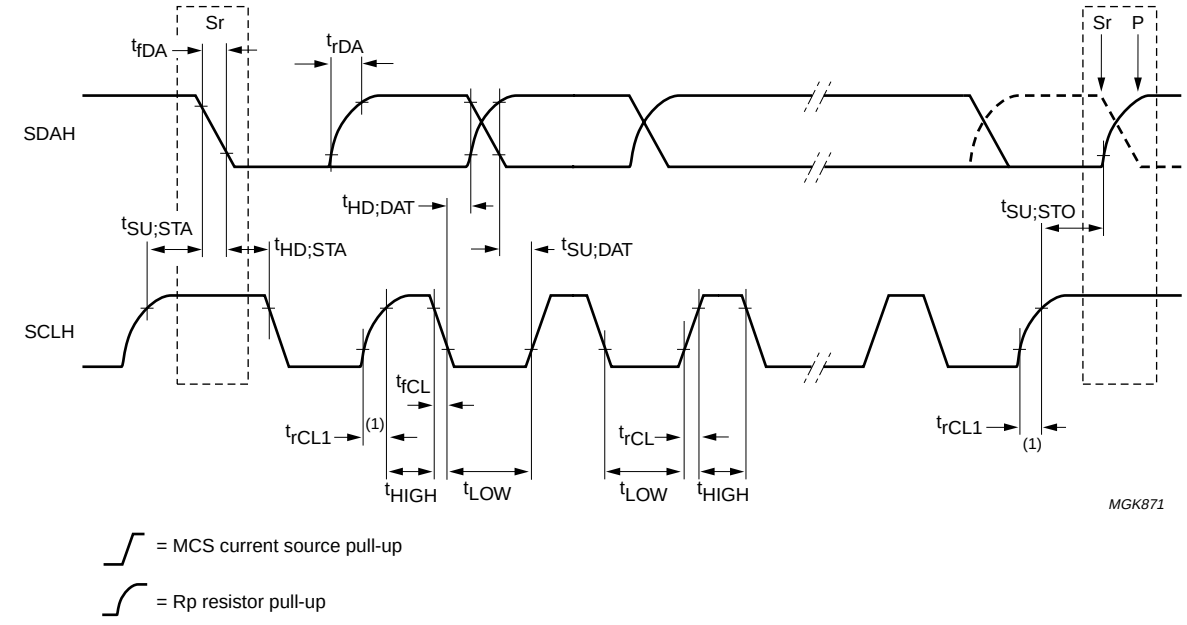
1. Devices that use non-standard supply voltages which do not conform to the intended I²C-bus system levels must relate their input levels to the V_{DD} voltage to which the pull-up resistors R_p are connected.
2. Devices that offer the level shift function must tolerate a maximum input voltage of 5.5 V at SDA and SCL.
3. For capacitive bus loads between 100 and 400 pF, the rise and fall time values must be linearly interpolated.
4. SDAH and SCLH I/O stages of Hs-mode slave devices must have floating outputs if their supply voltage has been switched off. Due to the current-source output circuit, which normally has a clipping diode to V_{DD}, this requirement is not mandatory for the SCLH or the SDAH I/O stage of Hs-mode master devices. This means that the supply voltage of Hs-mode master devices cannot be switched off without affecting the SDAH and SCLH lines.

Table 7 Characteristics of the SDAH, SCLH, SDA and SCL bus lines for Hs-mode I²C-bus devices(1)

PARAMETER	SYMBOL	C_b = 100 pF MAX.		C_b = 400 pF(2)		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCLH clock frequency	f_{SCLH}	0	3.4	0	1.7	MHz
Set-up time (repeated) START condition	$t_{SU;STA}$	160	–	160	–	ns
Hold time (repeated) START condition	$t_{HD;STA}$	160	–	160	–	ns
LOW period of the SCLH clock	t_{LOW}	160	–	320	–	ns
HIGH period of the SCLH clock	t_{HIGH}	60	–	120	–	ns
Data set-up time	$t_{SU;DAT}$	10	–	10	–	ns
Data hold time	$t_{HD;DAT}$	0(3)	70	0(3)	150	ns
Rise time of SCLH signal	t_{rCL}	10	40	20	80	ns
Rise time of SCLH signal after a repeated START condition and after an acknowledge bit	t_{rCL1}	10	80	20	160	ns
Fall time of SCLH signal	t_{fCL}	10	40	20	80	ns
Rise time of SDAH signal	t_{rDA}	10	80	20	160	ns
Fall time of SDAH signal	t_{fDA}	10	80	20	160	ns
Set-up time for STOP condition	$t_{SU;STO}$	160	–	160	–	ns
Capacitive load for SDAH and SCLH lines	C_b(2)	–	100	–	400	pF
Capacitive load for SDAH + SDA line and SCLH + SCL line	C_b	–	400	–	400	pF
Noise margin at the LOW level for each connected device (including hysteresis)	V_{nL}	$0.1V_{DD}$	–	$0.1V_{DD}$	–	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V_{nH}	$0.2V_{DD}$	–	$0.2V_{DD}$	–	V

Notes

1. All values referred to V_{IHmin} and V_{ILmax} levels (see Table 6).
2. For bus line loads C_b between 100 and 400 pF the timing parameters must be linearly interpolated.
3. A device must internally provide a Data hold time to bridge the undefined part between V_{IH} and V_{IL} of the falling edge of the SCLH signal. An input circuit with a threshold as low as possible for the falling edge of the SCLH signal minimizes this hold time.



(1) First rising edge of the SCLH signal after Sr and after each acknowledge bit.

Fig.32 Definition of timing for Hs-mode devices on the I²C-bus.

16 ELECTRICAL CONNECTIONS OF I²C-BUS DEVICES TO THE BUS LINES

The electrical specifications for the I/Os of I²C-bus devices and the characteristics of the bus lines connected to them are given in Section 15.

I²C-bus devices with fixed input levels of 1.5 V and 3 V can each have their own appropriate supply voltage. Pull-up resistors must be connected to a 5 V ± 10% supply (Fig.33). I²C-bus devices with input levels related to V_{DD} must have one common supply line to which the pull-up resistor is also connected (Fig.34).

When devices with fixed input levels are mixed with devices with input levels related to V_{DD}, the latter devices must be connected to one common supply line of 5 V ± 10% and must have pull-up resistors connected to their SDA and SCL pins as shown in Fig.35.

New Fast- and Hs-mode devices must have supply voltage related input levels as specified in Tables 4 and 6.

Input levels are defined in such a way that:

- The noise margin on the LOW level is $0.1V_{DD}$
- The noise margin on the HIGH level is $0.2V_{DD}$
- As shown in Fig.36, series resistors (R_S) of e.g. 300 Ω can be used for protection against high-voltage spikes on the SDA and SCL lines (resulting from the flash-over of a TV picture tube, for example).

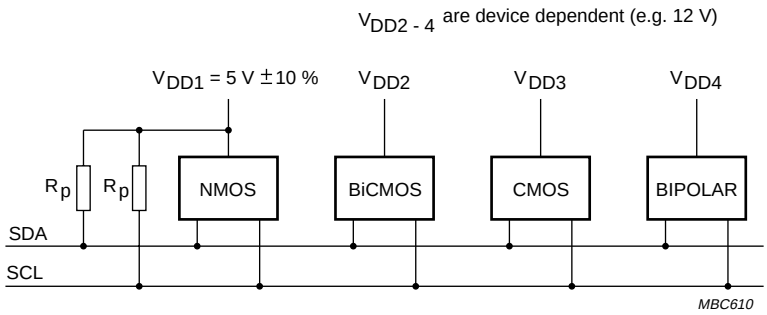


Fig.33 Fixed input level devices connected to the I²C-bus.

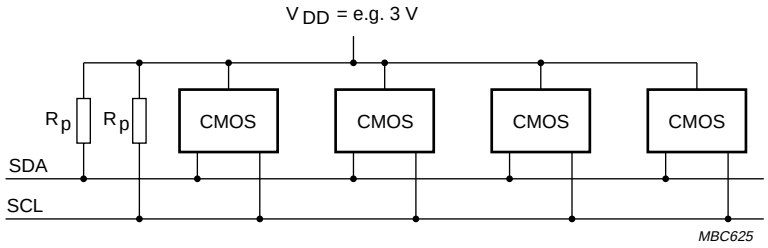


Fig.34 Devices with wide supply voltage range connected to the I²C-bus.

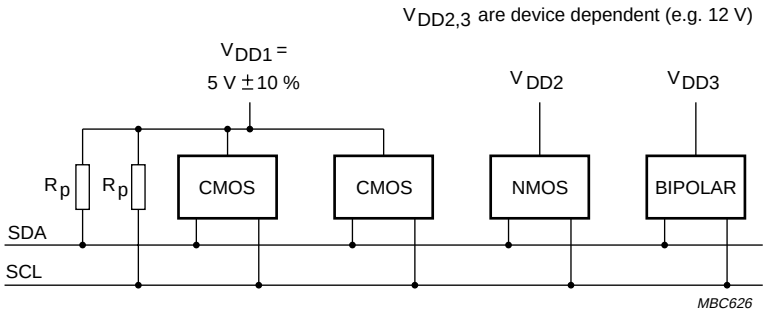


Fig.35 Devices with input levels related to V_{DD} (supply V_{DD1}) mixed with fixed input level devices (supply $V_{DD2,3}$) on the I²C-bus.

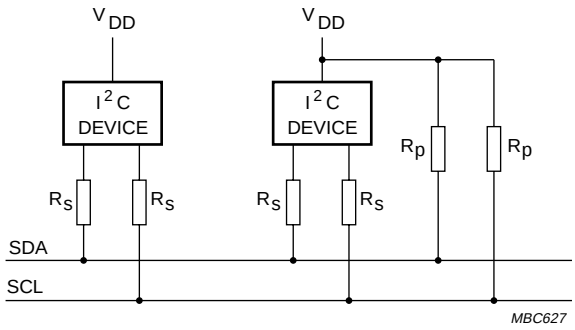


Fig.36 Series resistors (R_s) for protection against high-voltage spikes.

16.1 Maximum and minimum values of resistors R_p and R_s for Standard-mode I²C-bus devices

For Standard-mode I²C-bus systems, the values of resistors R_p and R_s in Fig.33 depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current).

The supply voltage limits the minimum value of resistor R_p due to the specified minimum sink current of 3 mA at V_{OLmax} = 0.4 V for the output stages. V_{DD} as a function of $R_{p\ min}$ is shown in Fig.37. The required noise margin of $0.1V_{DD}$ for the LOW level, limits the maximum value of R_s. $R_{s\ max}$ as a function of R_p is shown in Fig.38.

The bus capacitance is the total capacitance of wire, connections and pins. This capacitance limits the maximum value of R_p due to the specified rise time. Fig.39 shows $R_{p\ max}$ as a function of bus capacitance.

The maximum HIGH level input current of each input/output connection has a specified maximum value of 10 μA. Due to the required noise margin of 0.2 V_{DD} for the HIGH level, this input current limits the maximum value of R_p. This limit depends on V_{DD}. The total HIGH level input current is shown as a function of $R_{p\ max}$ in Fig.40.

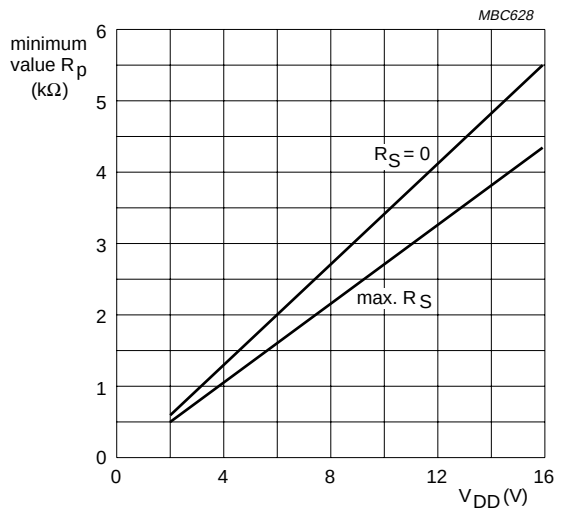


Fig.37 Minimum value of R_p as a function of supply voltage with the value os R_s as a parameter.

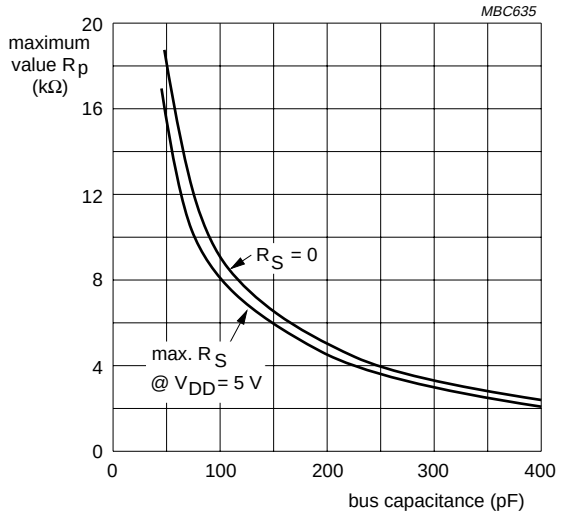


Fig.39 Maximum value of R_p as a function of bus capacitance for a Standard-mode I²C-bus.

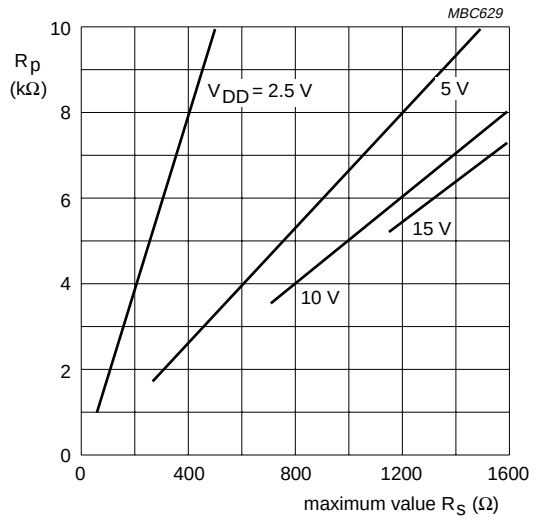


Fig.38 Maximum value of R_s as a function of the value of R_p with supply voltage as a parameter.

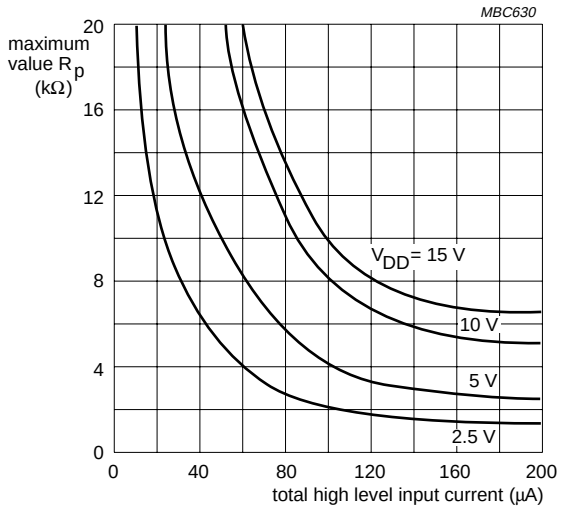


Fig.40 Total HIGH level input current as a function of the maximum value of R_p with supply voltage as a parameter.

17 APPLICATION INFORMATION

17.1 Slope-controlled output stages of Fast-mode I²C-bus devices

The electrical specifications for the I/Os of I²C-bus devices and the characteristics of the bus lines connected to them are given in Section 15.

Figures 41 and 42 show examples of output stages with slope control in CMOS and bipolar technology. The slope of the falling edge is defined by a Miller capacitor (C1) and a resistor (R1). The typical values for C1 and R1 are indicated on the diagrams. The wide tolerance for output fall time t_{of} given in Table 4 means that the design is not critical. The fall time is only slightly influenced by the external bus load (C_b) and external pull-up resistor (R_p). However, the rise time (t_r) specified in Table 5 is mainly determined by the bus load capacitance and the value of the pull-up resistor.

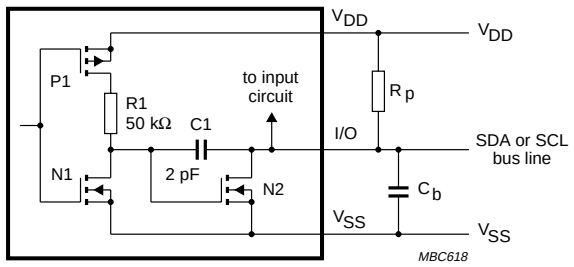


Fig.41 Slope-controlled output stage in CMOS technology.

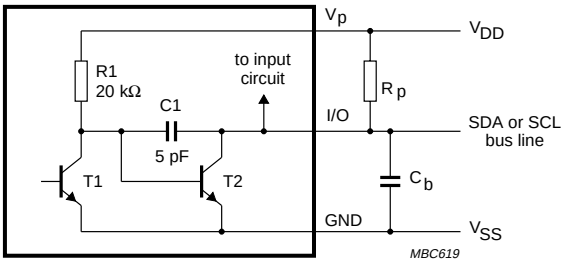


Fig.42 Slope-controlled output stage in bipolar technology.

17.2 Switched pull-up circuit for Fast-mode I²C-bus devices

The supply voltage (V_{DD}) and the maximum output LOW level determine the minimum value of pull-up resistor R_p (see Section 16.1). For example, with a supply voltage of V_{DD} = 5 V ± 10% and V_{OLmax} = 0.4 V at 3 mA, $R_{p\,min}$ = (5.5 – 0.4)/0.003 = 1.7 kΩ. As shown in Fig.33, this value of R_p limits the maximum bus capacitance to about 200 pF to meet the maximum t_r requirement of 300 ns. If the bus has a higher capacitance than this, a switched pull-up circuit as shown in Fig.43 can be used.

The switched pull-up circuit in Fig.43 is for a supply voltage of V_{DD} = 5 V ± 10% and a maximum capacitive load of 400 pF. Since it is controlled by the bus levels, it needs no additional switching control signals. During the rising/falling edges, the bilateral switch in the HCT4066 switches pull-up resistor R_{p2} on/off at bus levels between 0.8 V and 2.0 V. Combined resistors R_{p1} and R_{p2} can pull-up the bus line within the maximum specified rise time (t_r) of 300 ns.

Series resistors R_s are optional. They protect the I/O stages of the I²C-bus devices from high-voltage spikes on the bus lines, and minimize crosstalk and undershoot of the bus line signals. The maximum value of R_s is determined by the maximum permitted voltage drop across this resistor when the bus line is switched to the LOW level in order to switch off R_{p2}.

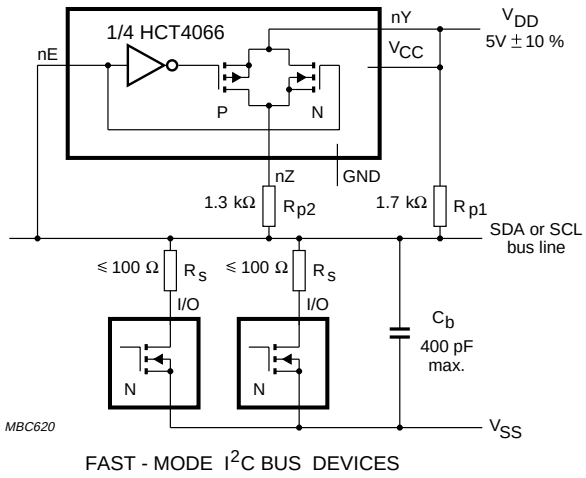


Fig.43 Switched pull-up circuit.

17.3 Wiring pattern of the bus lines

In general, the wiring must be so chosen that crosstalk and interference to/from the bus lines is minimized. The bus lines are most susceptible to crosstalk and interference at the HIGH level because of the relatively high impedance of the pull-up devices.

If the length of the bus lines on a PCB or ribbon cable exceeds 10 cm and includes the V_{DD} and V_{SS} lines, the wiring pattern must be:

SDA —————————

V_{DD} —————————

V_{SS} —————————

SCL —————————

If only the V_{SS} line is included, the wiring pattern must be:

SDA —————————

V_{SS} —————————

SCL —————————

These wiring patterns also result in identical capacitive loads for the SDA and SCL lines. The V_{SS} and V_{DD} lines can be omitted if a PCB with a V_{SS} and/or V_{DD} layer is used.

If the bus lines are twisted-pairs, each bus line must be twisted with a V_{SS} return. Alternatively, the SCL line can be twisted with a V_{SS} return, and the SDA line twisted with a V_{DD} return. In the latter case, capacitors must be used to decouple the V_{DD} line to the V_{SS} line at both ends of the twisted pairs.

If the bus lines are shielded (shield connected to V_{SS}), interference will be minimized. However, the shielded cable must have low capacitive coupling between the SDA and SCL lines to minimize crosstalk.

17.4 Maximum and minimum values of resistors R_p and R_s for Fast-mode I²C-bus devices

The maximum and minimum values for resistors R_p and R_s connected to a Fast-mode I²C-bus can be determined from Figs 37, 38 and 40 in Section 16.1. Because a Fast-mode I²C-bus has faster rise times (t_r) the maximum value of R_p as a function of bus capacitance is less than that shown in Fig.39 The replacement graph for Fig.39 showing the maximum value of R_p as a function of bus capacitance (C_b) for a Fast-mode I²C-bus is given in Fig.44.

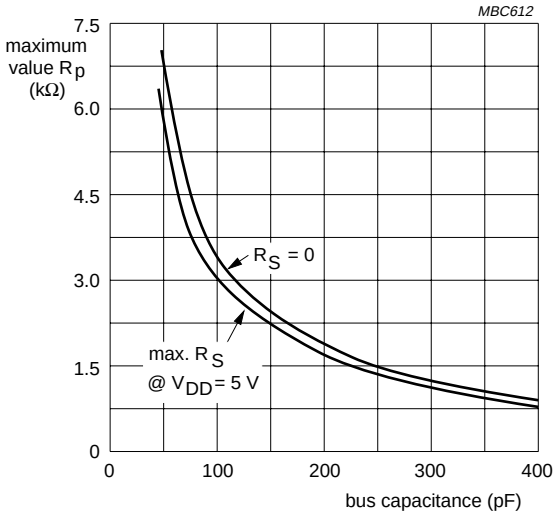


Fig.44 Maximum value of R_p as a function of bus capacitance for meeting the $t_{r\ max}$ requirement for a Fast-mode I²C-bus.

17.5 Maximum and minimum values of resistors R_p and R_s for Hs-mode I²C-bus devices

The maximum and minimum values for resistors R_p and R_s connected to an Hs-mode I²C-bus can be calculated from the data in Tables 6 and 7. Many combinations of these values are possible, owing to different rise and fall times, bus line loads, supply voltages, mixed speed systems and level shifting. Because of this, no further graphs are included in this specification.

18 BI-DIRECTIONAL LEVEL SHIFTER FOR F/S-MODE I²C-BUS SYSTEMS

Present technology processes for integrated circuits with clearances of 0.5 μm and less limit the maximum supply voltage and consequently the logic levels for the digital I/O signals. To interface these lower voltage circuits with existing 5 V devices, a level shifter is needed. For bi-directional bus systems like the I²C-bus, such a level shifter must also be bi-directional, without the need of a direction control signal(1). The simplest way to solve this problem is by connecting a discrete MOS-FET to each bus line.

(1) US 5,689,196 granted; corresponding patent applications pending.

The I²C-bus specification

In spite of its surprising simplicity, such a solution not only fulfils the requirement of bi-directional level shifting without a direction control signal, it also:

- isolates a powered-down bus section from the rest of the bus system
- protects the “lower voltage” side against high voltage spikes from the “higher-voltage” side.

The bi-directional level shifter can be used for both Standard-mode (up to100 kbit/s) or in Fast-mode (up to 400 kbit/s) I²C-bus systems. It is not intended for Hs-mode systems, which may have a bridge with a level shifting possibility (see Section 13.5)

18.1 Connecting devices with different logic levels

Section 16 described how different voltage devices could be connected to the same bus by using pull-up resistors to the supply voltage line. Although this is the simplest solution, the lower voltage devices must be 5 V tolerant, which can make them more expensive to manufacture. By using a bi-directional level shifter, however, it’s possible to interconnect two sections of an I²C-bus system, with each section having a different supply voltage and different logic levels. Such a configuration is shown in Fig.45. The left “low-voltage” section has pull-up resistors and devices connected to a 3.3 V supply voltage, the right “high-voltage” section has pull-up resistors and devices connected to a 5 V supply voltage. The devices of each section have I/Os with supply voltage related logic input levels and an open drain output configuration.

The level shifter for each bus line is identical and consists of one discrete N-channel enhancement MOS-FET; TR1 for the serial data line SDA and TR2 for the serial clock line SCL. The gates (g) have to be connected to the lowest supply voltage V_{DD1}, the sources (s) to the bus lines of the “lower-voltage” section, and the drains (d) to the bus lines of the “higher-voltage” section. Many MOS-FETs have the substrate internally connected with its source, if this is not the case, an external connection should be made. Each MOS-FET has an integral diode (n-p junction) between the drain and substrate.

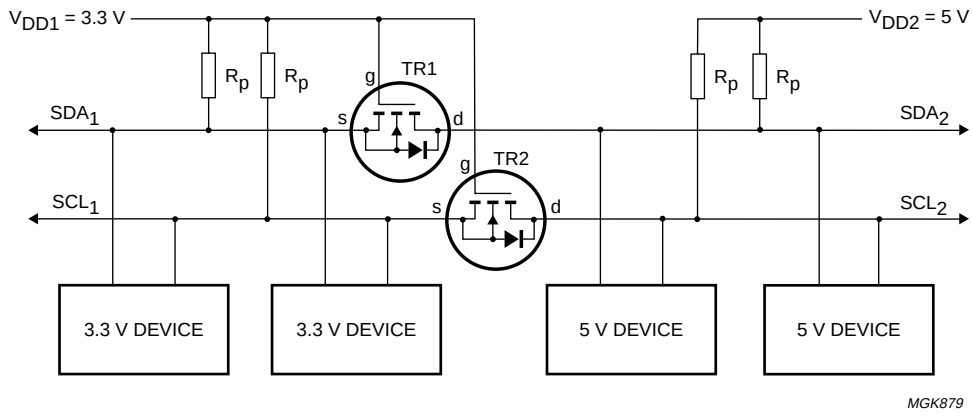


Fig.45 Bi-directional level shifter circuit connecting two different voltage sections in an I²C-bus system.

18.1.1 Operation of the level shifter

The following three states should be considered during the operation of the level shifter:

1. No device is pulling down the bus line.
   The bus line of the “lower-voltage” section is pulled up by its pull-up resistors R_p to 3.3 V. The gate and the source of the MOS-FET are both at 3.3 V, so its V_{GS} is below the threshold voltage and the MOS-FET is not conducting. This allows the bus line at the “higher-voltage” section to be pulled up by its pull-up resistor R_p to 5 V. So the bus lines of both sections are HIGH, but at a different voltage level.
2. A 3.3 V device pulls down the bus line to a LOW level. The source of the MOS-FET also becomes LOW, while the gate stay at 3.3 V. V_{GS} rises above the threshold and the MOS-FET starts to conduct. The bus line of the “higher-voltage” section is then also pulled down to a LOW level by the 3.3 V device via the conducting MOS-FET. So the bus lines of both sections go LOW to the same voltage level.
3. A 5 V device pulls down the bus line to a LOW level. The drain-substrate diode of the MOS-FET the “lower-voltage” section is pulled down until V_{GS} passes the threshold and the MOS-FET starts to conduct. The bus line of the “lower-voltage” section is then further pulled down to a LOW level by the 5 V device via the conducting MOS-FET. So the bus lines of both sections go LOW to the same voltage level.

The three states show that the logic levels are transferred in both directions of the bus system, independent of the driving section. State 1 performs the level shift function. States 2 and 3 perform a “wired AND” function between the bus lines of both sections as required by the I²C-bus specification.

Supply voltages other than 3.3 V for V_{DD1} and 5 V for V_{DD2} can also be applied, e.g. 2 V for V_{DD1} and 10 V for V_{DD2} is feasible. In normal operation V_{DD2} must be equal to or higher than V_{DD1} (V_{DD2} is allowed to fall below V_{DD1} during switching power on/off).

19 DEVELOPMENT TOOLS AVAILABLE FROM PHILIPS

Table 8 I²C evaluation boards

PRODUCT	DESCRIPTION
OM4151/ S87C00KSD	I²C-bus evaluation board with microcontroller, LCD, LED, Par. I/O, SRAM, EEPROM, Clock, DTMF generator, AD/DA conversion.
OM5500	Demo kit for the PCF2166 LCD driver and PCD3756A telecom microcontroller

Table 9 Development tools for 80C51-based systems

PRODUCT	DESCRIPTION
PDS51	A board-level, full featured, in-circuit emulator: RS232 interface to PC, universal motherboard, controlled via terminal emulation

Table 10 Development tools for 68000-based systems

PRODUCT	DESCRIPTION
OM4160/2	Microcore-2 demonstration/evaluation board with SCC68070
OM4160/4	Microcore-4 demonstration/evaluation board with 90CE201
OM4160/5	Microcore-5 demonstration/evaluation board with 90CE301

Table 11 I²C analyzers

PRODUCT	DESCRIPTION
OM1022	PC I²C-bus analyzer with multi-master capability. Hardware and software (runs on IBM or compatible PC) to experiment with and analyze the behaviour of the I²C-bus (includes documentation)
OM4777	Similar to OM1022 but for single-master systems only
PF8681	I²C-bus analyzer support package for the PM3580 logic analyzer family

20 SUPPORT LITERATURE

Table 12 Data handbooks

TITLE	ORDERING CODE
IC01: Semiconductors for Radio, Audio and CD/DVD Systems	9397 750 02453
IC02: Semiconductors for Television and Video Systems	9397 750 01989
IC03: Semiconductors for Wired Telecom Systems (parts a & b)	9397 750 00839, 9397 750 00811
IC12: I²C Peripherals	9397 750 01647
IC14: 8048-based 8-bit microcontrollers	9398 652 40011
IC17: Semiconductors for wireless communications	9397 750 01002
IC18: Semiconductors for in-car electronics	9397 750 00418
IC19: ICs for data communications	9397 750 00138
IC20: 80C51-based 8-bit microcontrollers + Application notes and Development tools	9397 750 00963
IC22: Multimedia ICs	9397 750 02183

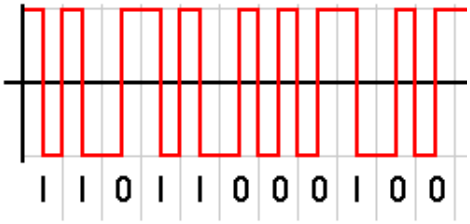
Table 13 Brochures/leaflets/lab. reports/books etc.

TITLE	ORDERING CODE
Can you make the distance... with I²C-bus (information about the P82B715 I²C-bus extender IC)	9397 750 00008
I²C-bus multi-master & single-master controller kits	9397 750 00953
Desktop video (CD-ROM)	9397 750 00644
80C51 core instructions quick reference	9398 510 76011
80C51 microcontroller selection guide	9397 750 01587
OM5027 I²C-bus evaluation board for low-voltage, low-power ICs & software	9398 706 98011
P90CL301 I²C driver routines	AN94078
User manual of Microsoft Pascal I²C-bus driver (MICDRV4.OBJ)	ETV/IR8833
C routines for the PCF8584	AN95068
Using the PCF8584 with non-specified timings and other frequently asked questions	AN96040
User's guide to I²C-bus control programs	ETV8835
The I²C-bus from theory to practice (book and disk)	Author: D. Paret Publisher: Wiley ISBN: 0-471-96268-6
Bi-directional level shifter for I²C-bus and other systems	AN97055
OM5500 demo kit for the PCF2166 LCD driver and PCD3756A telecom microcontroller	9397 750 00954

For more information about Philips Semiconductors and how we can help with your I²C-bus design, contact your nearest Philips Semiconductors national organization from the address list of the back of this book, or visit our worldwide web site at http://www.semiconductors.philips.com/i2c for the latest products, news and applications notes.

Manchester code

From Wikipedia, the free encyclopedia.



Encoding of 11011000100 in Manchester code

In telecommunication, **Manchester code** is a form of data communication in which each bit of data is signified by at least one transition. Manchester encoding is therefore considered to be self-clocking, which means that accurate synchronisation of a data stream is possible. Each bit is transmitted over a predefined time period.

There are two opposing conventions for the representations of data.

The first of these was first published by G. E. Thomas in 1949 and is followed by numerous authors (e.g., Tannenbaum). It specifies that for a 0 bit the signal levels will be Low-High (assuming an amplitude physical encoding of the data) - with a low level in the first half of the bit period, and a high level in the second half. For a 1 bit the signal levels will be High-Low.

The second convention is also followed by numerous authors (e.g., Stallings) as well as by the IEEE 802.4 standard. It states that a logic 0 is represented by a High-Low signal sequence and a logic 1 is represented by a Low-High signal sequence.

A consequence of the transitions for each bit is that the bandwidth requirements for Manchester encoded signals is doubled compared with asynchronous communications, and the signal spectrum is considerably wider. Although Manchester encoding is a highly reliable form of communication, the bandwidth requirements are seen as a disadvantage, and most modern communication takes place with asynchronous communications protocols.

One consideration with Manchester encoding is synchronising the receiver with the transmitter. At first sight it might seem that a half bit period error would lead to an inverted output at the receiver end, but further consideration reveals that on typical data this will lead to code violations. The hardware used can detect these code violations, and use this information to synchronise accurately on the correct interpretation of the data.

A related technique is differential Manchester encoding.

In summary:

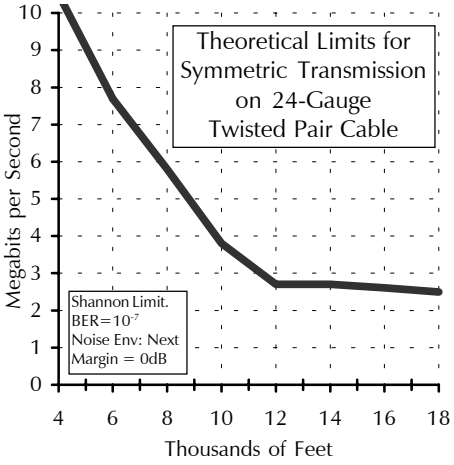
- data and clock signals are combined to form a single self-synchronizing data stream
- each encoded bit contains a transition at the midpoint of a bit period
- the direction of transition determines whether the bit is a "0" or a "1," and
- the first half is the true bit value and the second half is the complement of the true bit value. *Contrast with* **non-return-to-zero.**

Some source: from Federal Standard 1037C in support of MIL-STD-188

Driving High-Speed Data Through Telephone Wires

Telemanagement explains almost everything you need to know about DSL

IAN ANGUS



There's a hot new acronym family in town. It's "xDSL," in which the last three letters stand for "Digital Subscriber Line," and the "x" stands for just about any letter of the alphabet — so you'll hear about ADSL, CDSL, HDSL, IDSL, SDSL, VDSL, and more.

After years of talk without much action, DSL is now taking off. Traditional phone companies and Competitive Local Exchange Carriers are rushing to install it — in the first three months of 2000, the number of working ADSL-based services in Canada jumped from 95,000 to 127,000. This year, Bell Canada's installation rate has grown from 1,200 ADSL-based Internet access services a week in January to 4,400 a week by May. There is even a proposal — hotly contested by many — to have the CRTC recognize a new category of carrier called "DSL Service Providers."

Most of the ADSL activity to date has been in the consumer market, but business-oriented services are beginning to appear. If you are involved with voice or data communications today, you need to know about DSL — where it came from, what it is, what it can do. But where will you learn such things? Of course! *Telemanagement* to the rescue...

The Many Varieties of Digital Subscriber Line ...

ADSL: Asymmetric DSL — transmission over one pair with higher speeds downstream than upstream
CDSL: Consumer DSL — Rockwell's name for low-speed ADSL without splitters
G.lite: Standard for low-speed ADSL without splitters
HDSL: High-bit-rate DSL — 1.544 or 2.048 Mbps over two pairs
HDSL2: Emerging standard for 1.544 or 2.048 Mbps over one twisted pair
IDSL: ISDN DSL — Basic Rate ISDN for data only, with no PSTN connection
RADSL: Rate Adaptive DSL — ADSL technology which adjusts data speed in response to transmission quality.
SDSL: Symmetric DSL — a one-pair version of HDSL, with data rates from 128 Kbps to 2.3 Mbps
VDSL: Very high-speed DSL — transmission over one pair with downstream rates from 13 to 52 Mbps over relatively short distances
VoDSL: Voice over DSL
xDSL: General term for all DSLs

Speed Limits

DSL stands for "Digital Subscriber Line," but it isn't a type of line. It's a family of modem technologies that provide high-speed transmission over the most widely available communications medium in the world — copper telephone wire.

Anyone who talks about technological limits in any area of telecom is likely to be embarrassed by news of yet another breakthrough. Nonetheless, it now seems certain that we are at or near the speed limit for voice-band modems. The modems which ship with most PCs max out at 56 Kbps for downloads and 33.6 Kbps for uploads — and the download speeds seldom reach the maximum.

Don't blame telephone *wires* for the speed limits, blame the telephone *network*. Telephone switches are physically designed for voice transmission, which requires 64,000 bits per second, no more. As the graph on this page shows, twisted pair phone wire isn't so limited: it can theoretically carry data at speeds of many megabits per second.

So why hasn't it been done before now?

In the first place, the "theoretical maximums" assume consistently high-quality wiring, and a lot of the installed cable is less than superb. The box on page 5 outlines some common problems that can interfere with data transmission on phone wire.

But even if the cable itself is in excellent condition, the laws of physics dictate that transmitting high-speed data accurately over thousands of feet of copper isn't a simple plug-and-go process. The two big barriers to high speed are the fact that all electrical signals get weaker over distance (attenuation) and that electrical signals in adjacent wires tend to interfere with each other (crosstalk).

Both of these problems are more severe for high-speed, high-frequency data communication than for voice. Until recently, they made the "theoretical maximums" irrelevant for any practical application. To preserve accuracy, data had to be transmitted many times more slowly than the theory said was possible.

DSL was made possible by two important technological developments: advanced methods of coding data for high-speed transmission, and low-cost Digital Signal Processing (DSP) chips that made possible affordable high-speed transmitters and receivers.

These technologies, together with some very sophisticated telecom engineering, have enabled the three major families of DSL technology: HDSL, ADSL, and VDSL. Although they have similar names, the three types of DSL are very different in origin, function, and application.

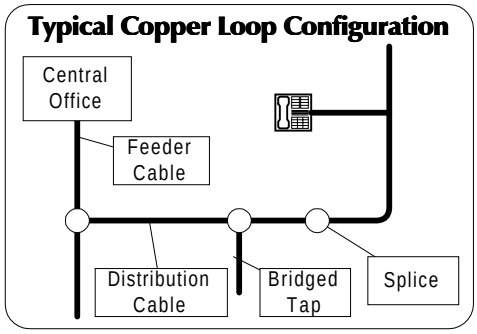
HDSL

High-bit-rate Digital Subscriber Line (HDSL) is the oldest and most mature of

Bridged Taps, Loading Coils, and Other Enemies

In an ideal world, the copper wires that enter your home or office would extend back, unbroken and uninterrupted, to the switching center. Such a world would be perfect for DSL, because the only difference between one loop and another would be length.

But we don't live in an ideal world, and the telephone distribution network isn't ideal for high-speed data. The following are some of the issues DSL's designers have had to deal with.



Loading Coils: For loops longer than 18,000 feet (and some shorter ones), installers insert coils that maintain voice quality but block high frequencies, making the line unusable for high-speed data. Only about 15% of all loops have loading coils, but finding out which those are can be a challenge. Loading coils must be physically removed from any loop used for DSL, a process that increases costs and slows installation. This is a much bigger problem in the U.S. than in Canada.

Bridged Taps: The copper pairs in the distribution cable that runs down the street are not cut to connect into a given customer's premises. Instead, the "drop wire" pairs are spliced into the distribution cable. This allows the same distribution pairs to be reused for another customer when your service is disconnected. Those entrance pairs — called "bridged taps" — are seldom removed when a change is made. Such leftover taps cause echoes and other forms of interference.

Splices: Telephone cable is usually manufactured in 500-foot lengths, so an 18,000-foot loop may have 36 or more splices along its length. They cause echoes that interfere with data quality.

Gauge Changes: Most distribution cables contain 24-gauge wires, which are 0.5 mm in diameter. But in the Feeder Cable near the switching center, where thousands of pairs of wires come together and space is at a premium, it's common to use smaller 26-gauge (0.4 mm) wires. This increases attenuation and adds another source of echo.

AM Radio: Telephone wires can act as antennas, picking up AM radio frequencies. Nortel Networks says that in the U.S. this affects about 15% of ADSL installations, reducing download speeds by as much as 40%, especially for customers who are at the end of very long loops and near AM radio transmitters.

Remote Concentrators: There isn't always a direct copper connection from the telco switch to each customer. In some areas — especially in new suburbs — the copper wire goes to a curbside box, called a Remote Concentrator, which is connected to the switch by optical fiber. Canadian telcos have begun working on the problem of locating DSLAMs in curbside boxes, but it won't be easy or inexpensive. It will be even more difficult for non-telco carriers to offer DSL to customers who are connected to Remote Concentrators.

the DSL technologies. It was developed to reduce the cost and complexity of installing T-1 services for customers. HDSL provides 1.544 Mbps over two pairs of copper wires, up to 12,000 feet long. Unlike traditional T-1, HDSL does not require repeaters (digital amplifiers), and it works on pairs that have bridged taps.

In 1992, Bell Canada became the first phone company in the world to use HDSL commercially. Today, about 15% of all North American T-1 and Primary Rate ISDN (Megalink) services use HDSL. In 1998 the International Telecommunications Union approved Recommendation G.991.1 for HDSL, in effect setting an international standard.

HDSL2, currently being considered by standards committees, provides T-1 speeds over a single pair of wires, allowing carriers to double the number of HDSL lines in service without adding any copper.

HDSL has been used almost exclusively to provide T-1 or Primary Rate services from the public network over local loops, but we are beginning to see it used in campus environments, by customers who want high-speed inter-building communications but don't want to pay monthly bills for the privilege. It's worth investigating, if you have high-speed data requirements and spare cable pairs.

ADSL

In the early nineties, researchers realized that twisted copper wire could handle even higher speeds if the communication was faster in one direction than the other. This led to the development of ADSL — Asymmetric Digital Subscriber Line.

ADSL development initially focused on Video Dial Tone, which the telephone industry believed would allow them to offer cable television and to launch lucrative video-on-demand services. Video Dial Tone seemed like the perfect application for ADSL — customers could order video services over a slow upstream channel, and receive them over a very fast downstream channel.

As time progressed, it became clear that no one could make money offering video on demand at the prices consumers were willing to pay, so by 1995 most phone companies had moved their Video Dial Tone projects into the "been there, tried that" file. ADSL might have disappeared, but for a completely unexpected development — the World Wide Web.

Internet access and ADSL are a near-perfect match, for two reasons:

- Most individual use of Internet access involves sending short messages to request files, followed by receipt of larger files, which matches the slow-upload, fast-download operation of ADSL.
- In most cases, ADSL can use existing telephone lines, without interfering with voice calls over those lines. As a result, it can be installed almost anywhere there is a phone, without replacing the local distribution infrastructure.

With cable TV, all channels travel through the cable at the same time, but

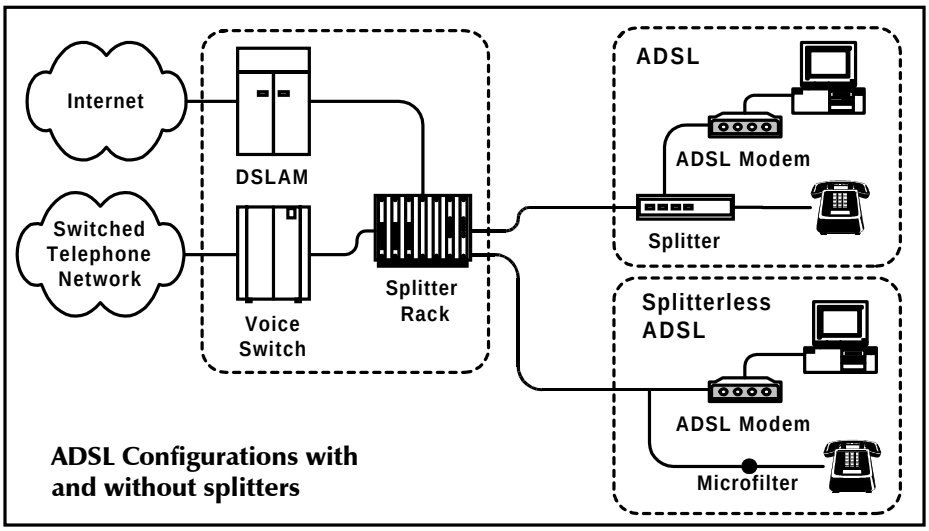
G.lite: Mass Market Breakthrough or Dead End?

ADSL carries voice and data in separate frequency bands, so they don't interfere with each other. That sounds good in theory, but is very hard to do in practice.

First, the high frequencies used for data tend to produce "side lobes" — lower frequencies that overlap into the voice frequency bands, generating noise that can be heard on telephones.

On the other side, telephones and other analog devices (modems and fax machines, most notably) produce signals that can interfere with high frequency data. The voltages that cause the phone to ring are a particularly difficult problem for ADSL.

Interference can be eliminated with a "splitter" that isolates the phone wiring from the data wiring.



ADSL Configurations with and without splitters

That solves one problem, but it creates another. Almost anyone can install an ADSL modem, but installing a splitter usually involves changes to the wiring system, and that requires a site visit by a trained technician. That creates an insuperable barrier to mass deployment of ADSL — installation will take too long and cost too much if a "truck roll" is required in every case.

If splitters were needed, no one could see how to implement ADSL for millions of customers and still make a profit.

In 1997, within a few months of each other, Nortel, Lucent, and Rockwell announced proprietary ADSL implementations that didn't require splitters. At virtually the same time, Compaq, Intel, and Microsoft, supported by the major U.S. telephone companies, announced the "Universal ADSL Working Group," which proposed to develop a standard for "splitterless ADSL" that could be routinely installed in every new PC for about the same cost as an analog modem

The price to be paid for these splitterless ADSL proposals was lower speed. All of them limited the top speed to 1 or 1.5 Mbps downstream and 512 Kbps upstream, as a means of reducing interference.

In 1998 and 1999, ADSL deployment focused on pre-standard splitterless devices. Bell Canada in particular made a major commitment to Nortel's "Megabit Modem" as the basis for consumer and small business ADSL, and thus for the "Sympatico High Speed Edition" Internet service.

There is now a standard for these slower, splitterless versions of ADSL — G.992, often called G.lite — but the entire process may prove to have been a technological detour, for two reasons:

- Experience has shown that even G.lite requires splitters in many cases — there is just too much poor-quality wiring in homes and businesses, and too many telephones with unpredictable electrical characteristics, for the technology to work everywhere.
- Major manufacturers have introduced "microfilters" — simple, small devices that the customer plugs into each phone jack to isolate the telephone set. These microfilters, combined with improved modem designs, allow ADSL to operate at full speed without splitters.

If carriers can deliver multi-megabit ADSL to most customers without sending out technicians, most of the case for G.lite evaporates. But lack of justification has never prevented any product for being marketed, so don't count G.lite out just yet.

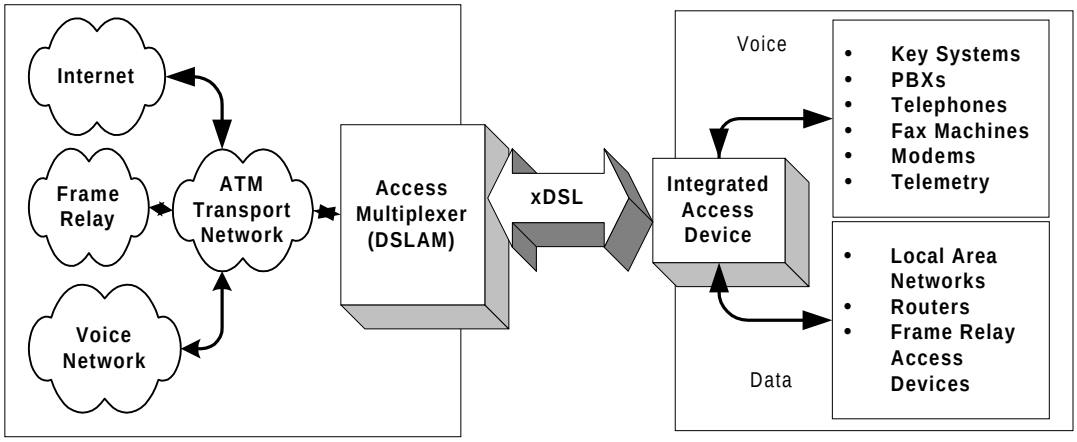
they don't interfere with each other because each channel uses a distinct range of frequencies. ADSL uses a similar approach with copper wire — regular phone calls use the 300 to 3400 Hz (Hertz — cycles per second) frequency range they have always used, while data is carried in the frequencies from 25 KHz to 1 Mhz.

Those higher frequencies are subdivided into a narrow band used for upstream communication and a wider band for downstream. The actual division and the method of dividing varies among manufacturers — not only do the standards allow more than one method, but many companies use non-standard technologies or proprietary enhancements to the standards.

Typically, North American carriers offer up to 800 Kbps upstream, and up to 8 Mbps downstream, but whether you can actually get the maximum will depend on the length, gauge, and quality of the loops terminating in your building.

The speed also depends, of course, on the equipment installed in the carrier's switching center. ADSL lines terminate on a DSL Access Multiplexer (DSLAM, pronounced Dee-Slam) that connects multiple ADSL connections to the appropriate wide area networks. Although everyone is promising "standards-based" ADSL, there is a lot of leeway for variation in the standards, so in most cases you'll be required to use an ADSL modem produced by the company that makes your carrier's DSLAMs. If you change ADSL carriers, you'll probably have to invest in new modems.

Multi-Service Networks? DSL is emerging as an important technology for some Competitive Local Exchange Carriers, who see it as a means of delivering a full range of business services to small and medium-sized businesses, without having to rent a separate loop for every line.



Canada's phone companies are betting heavily on ADSL as their key competitive weapon against the cable TV companies' high-speed Internet service for residential customers, so most of the marketing effort and hype will target that market. NBTel is taking the competition further, using ADSL to deliver cable television to homes in some areas.

In addition, we expect to see significant adoption of ADSL by business customers, especially for smaller offices and telecommuters. ADSL is likely to replace most existing basic rate ISDN (Microlink) services, and to bring multi-megabit data communications to locations that couldn't previously afford them. In many cases, ADSL will be the ideal way to connect branch offices to corporate nets.

VDSL

Although they can be used in other ways, HDSL and ADSL were both designed to provide high-speed connections over 12,000 to 18,000 feet of copper between a carrier's switching center and a customer's premises. The third major member of the DSL family plays a different role. Very high-speed DSL (VDSL) is designed for shorter distances, to extend the reach of optical fiber networks from the curb or the basement to customer premises.

VDSL can be configured for either symmetric or asymmetric operation, with speeds from 13 to 52 Mbps.

The big application for VDSL is likely to be multiple-service networks, delivering voice, data, and video over in-building cabling systems. Telephone companies see it as a way to offer cable TV services in apartment buildings, and some Competitive Local Exchange Carriers are installing VDSL-based multi-service networks in office buildings.

VDSL also has potential as a replacement for T-1 trunks and multiple copper pairs in corporate communications. In a campus environment, for example, buildings could be linked with fiber, with VDSL for distribution to in-building LANs.

Is DSL for You?

DSL offers high-speed communication links that support multiple services, at prices that are lower — much lower in some cases — than other digital technologies. So why not make it a key element in your organization's networking strategy right away? What's the catch?

Many organizations will probably use one or more flavors of DSL eventually. But your adoption plan should take some caveats into account:

■ DSL is evolving very rapidly. If you order it this year, you can expect to be one of the first to try the latest release.

■ Customer-friendly management and diagnostic tools are hard to come by in the DSL world. Organizations that are used to end-to-end network management systems may find that these systems can't "see" into the DSL world very well yet.

■ Even more important, the carriers offering DSL to end-customers are new — either new in every sense (the new local carriers) or new to this technology (the phone companies). Count on it: there *will* be problems with ordering, provisioning, and support, with every carrier.

■ DSL availability will be spotty for a while. The odds are that it won't be available in some of the places you want it.

Being an early adopter is always risky, but it also has the potential for big payoffs. Implementing DSL on a small scale — in one small office or department, perhaps — may offer you the opportunity to learn about the technology, evaluate vendor capabilities, and lay the basis for larger applications. Your task in 2000 is to weigh the risks, and move ahead appropriately.

The Future

In our fast-moving industry, no technology, however good, has a clear path to market success. There are multiple alternatives for every application, and products that seem to be in the lead often find themselves leapfrogged by competitors.

Even within the DSL family, there is overlap. Both HDSL and VDSL can be used for high-capacity campus distribution. Both ADSL and VDSL can be used for television distribution. All three can be used for business applications.

More significantly, DSL faces strong competition from other technologies. It is, for example, an open question whether ADSL will continue to be more cost effective than fiber directly to the end customer: some manufacturers are now arguing that fiber-to-the-home can be deployed as cheaply as DSL. Similarly, the companies that won broadband wireless licenses in last year's auction will compete directly for ADSL-type applications.

The stage is set for a major battle in the access market. DSL's future will depend both on how its supporters price, market, deliver, and support it — and on how well supporters of alternative technologies do the same things. ■

Telemanagement

The Angus Report on Business Telecommunications in Canada

Angus TeleManagement Group Inc.
8 Old Kingston Road, Ajax ON L1T 2Z7
Tel: 905-686-5050 ext 500 • Fax: 905-686-2655
www.angustel.ca • admin@angustel.ca

Keeping decision makers ahead of today's revolution in business telecom and networking

Since 1983, decision makers across Canada have depended on *Telemanagement* for independent, management-oriented analysis and information about the fast-changing world of business telecommunications and networking.

Telemanagement cuts across hype and oversell, delivering the information you need in order to determine what is really useful and practical for your organization today, and how changes in networking and telecom will affect you. It stresses concise, clearly written and tightly edited reports, for managers who must make hard decisions about telecommunications and the role it plays in their organizations' strategic plans.

Each issue of *Telemanagement* puts Angus TeleManagement Group's expertise to work for you ... translating technical jargon ... making sense of complex regulations ... explaining the impact and importance of all key developments, as they affect you and your organization.

Business telecommunications is too important to be left to chance, ignored, or managed using yesterday's concepts and information. Start benefiting from the number one telecommunications information source in Canada.

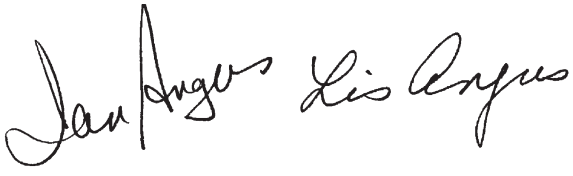
We invite you to subscribe ... and to protect your investment with this unique and ironclad

100% Money-Back Guarantee of Satisfaction

Your subscription is absolutely risk free. Here's why:

> If *Telemanagement* doesn't provide the accurate, reliable information you need, just write and tell us why. We'll immediately refund your money. Not just for the undelivered issues, but *every cent you paid for this subscription*. The issues you have already received will be yours to keep with our thanks.

This is the strongest, fairest guarantee you'll find anywhere. We make this unique offer because we're confident that you'll find *Telemanagement* to be worth many times the subscription price, and that once you start reading it, you'll continue to subscribe for years to come.



Ian and Lis Angus have been the Publishers and Senior Editors of *Telemanagement* since 1983. *The Financial Post* called them "Canada's best-known telecom consultants." *OA Magazine* described them as "visionaries in telecommunications." They have twice won major awards from the Canadian Business Telecommunications Alliance, "in recognition of their leadership and influence on the Canadian telecommunications environment."

FOUR EASY WAYS TO SUBSCRIBE

WEB: www.angustel.ca
(Secure on-line payment available)
PHONE: 1-800-263-4415 x500
FAX: 905-686-2655
MAIL: Telemanagement, 8 Old Kingston Road, Ajax, Ontario Canada L1T 2Z7

In Canada:

- ☐ One Year (10 issues) C $385.00
- ☐ Two Years (20 issues) C $725.00 (Best Deal: Save 52%)

Other countries:

- ☐ One Year (10 issues) US $325.00
- ☐ Two Years (20 issues) US $600.00

PLEASE PRINT CLEARLY

Name _____

Title _____

Organization _____

Address _____

City _____

Prov/State _____

Postal/Zip Code _____

Phone _____

Fax _____

E-mail _____

Charge to: ☐ Visa ☐ American Express ☐ Mastercard

Account # _____
For MasterCard, please provide full 19-digit account # on back of card.

Expiry _____

Signature _____

☐ Full payment enclosed: $_____
(Please add applicable GST or HST - BN# 10021 5995 RT)

☐ Please Invoice

☐ **FREE WEEKLY UPDATES!** Check here to confirm that you also want to receive Telecom *Update* – Angus TeleManagement's telecom newsbulletin – by e-mail every week, absolutely free! (It will be sent to your e-mail address, so please make sure you have printed it clearly and correctly.)

The RS232 STANDARD

A Tutorial with Signal Names and Definitions

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Send Us Your Comments . . .

Contents

What is EIA232?
Next Topic | TOC

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the

interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

Likely Problems when Using an EIA232 Interface
Next Topic | Previous Topic | TOC

During this 40-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

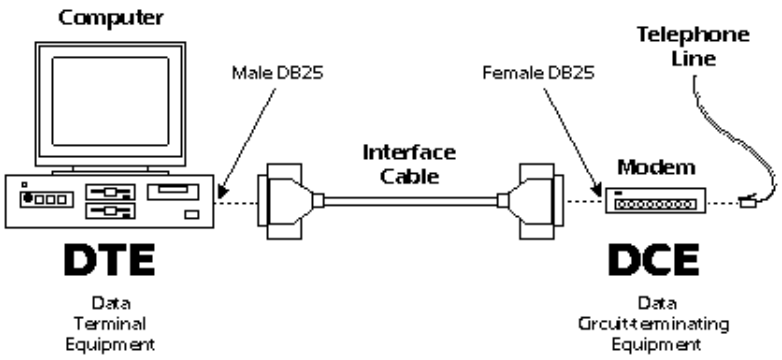
Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

Pin Assignments
Next Topic | Previous Topic | TOC

Go to DTE Pinout (looking into the computer’s serial connector)
Go to DCE Pinout (looking into the modem’s serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is

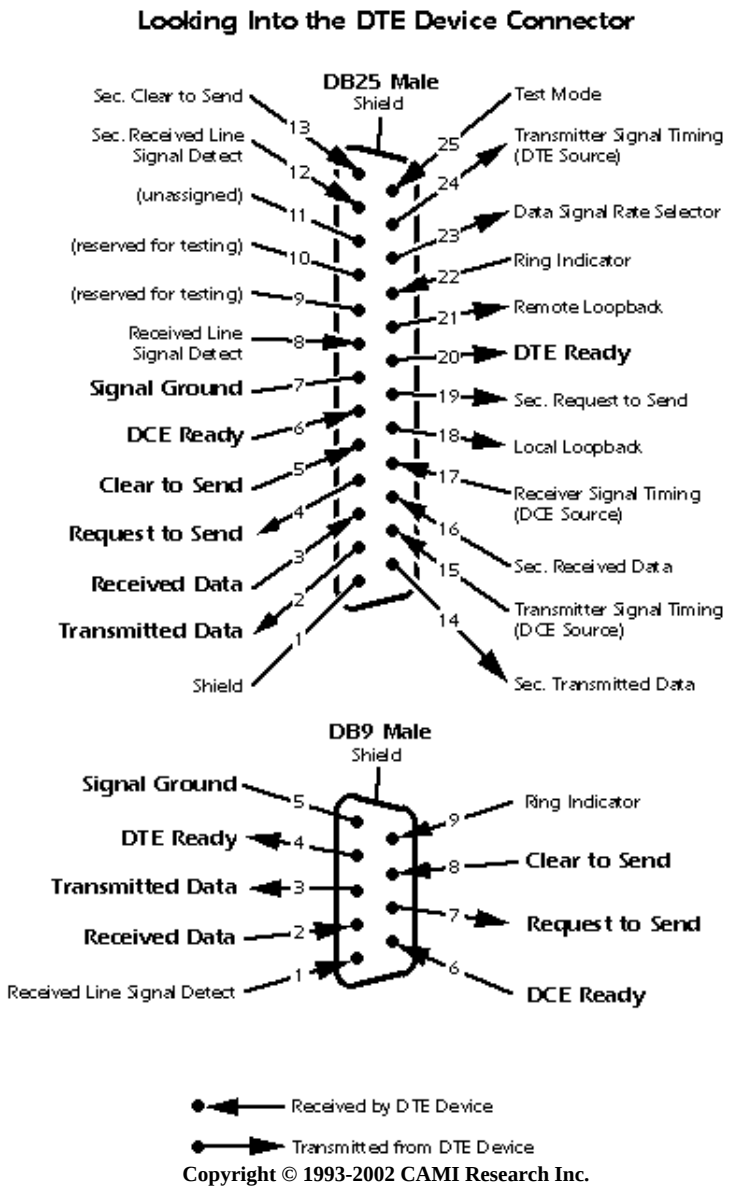
named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.

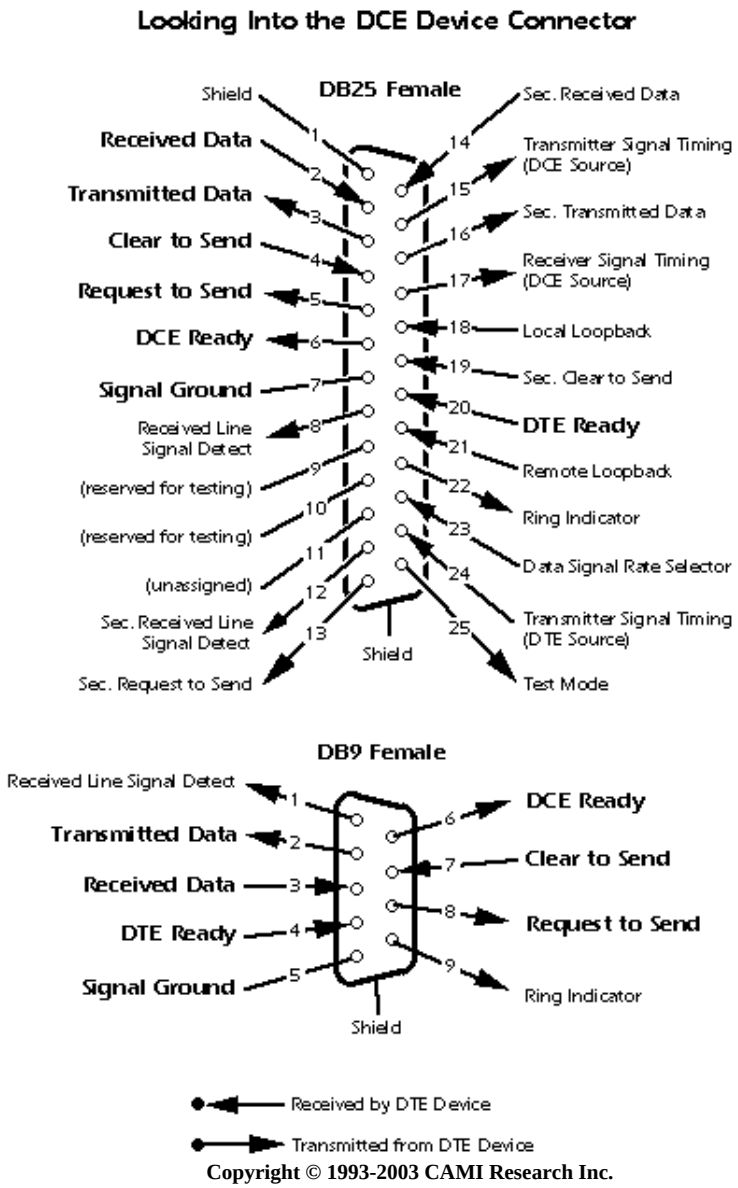
Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



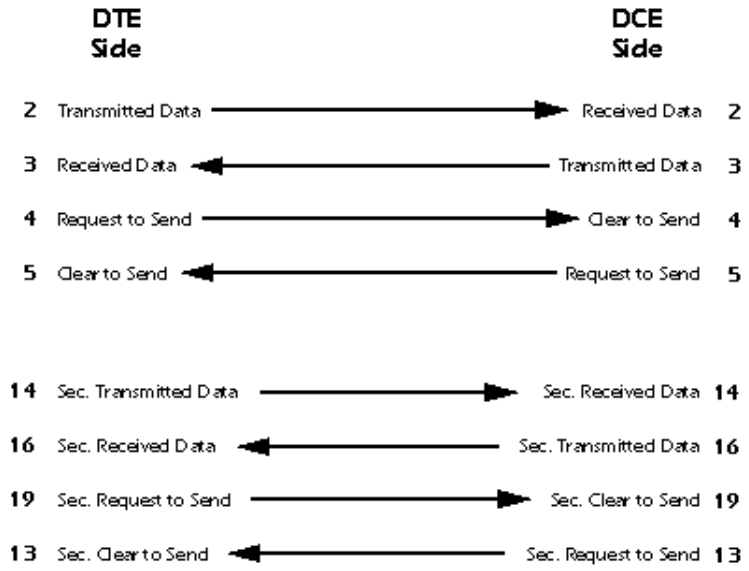
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

> ***IMPORTANT:*** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:



Cable Wiring Examples
Next Topic | Previous Topic | TOC

The following wiring diagrams come from actual cables scanned by the CableEye® PC-Based Cable Test System. CableEye's software automatically draws schematics whenever it tests a cable. Click here to learn more about CableEye.

1 - DB9 All-Line Direct Extension
Next Cable | (no previous cable) || Next Topic

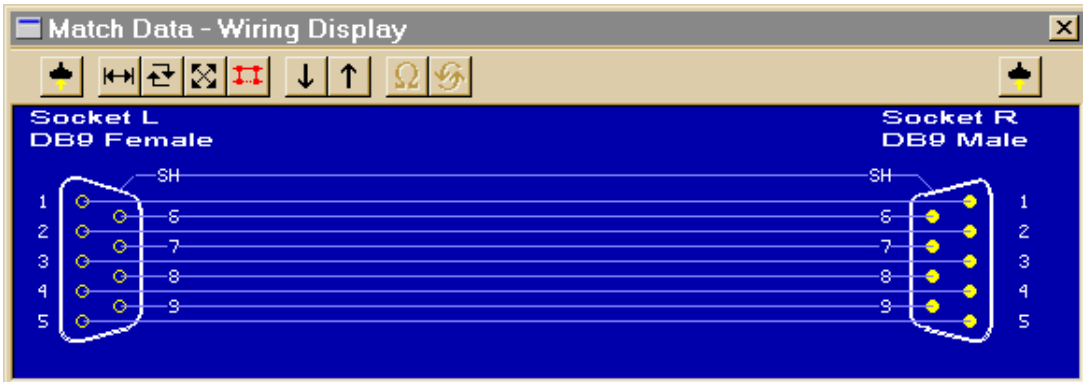
80K

This shows a 9-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 9 pins plus shield are directly extended from DB9 Female to DB9 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any device that uses a DB9 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between a computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Left Side: Connect to **DTE** (computer)

Right Side: Connect to **DCE** (modem or other serial device)



Cable image created by CableEye®

---

2 - DB9 Loopback Connector

Next Cable | Previous Cable || Next Topic

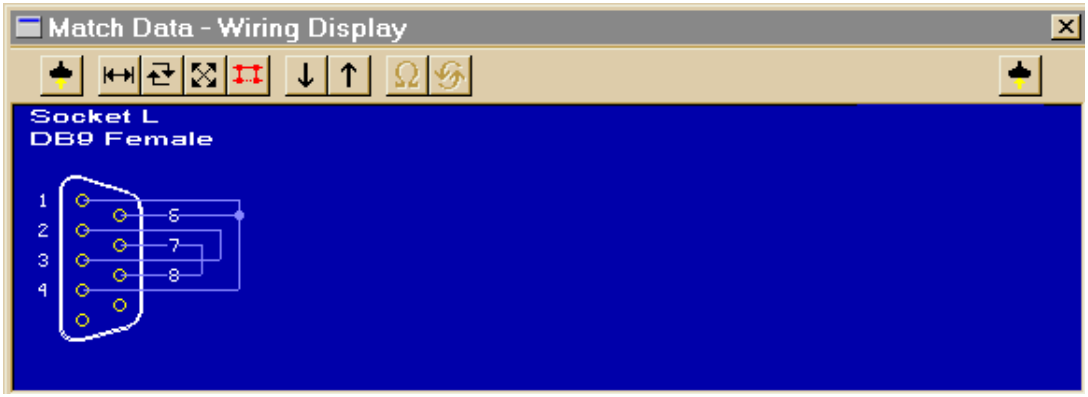
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

3 - DB9 Null Modem Cable
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

80K

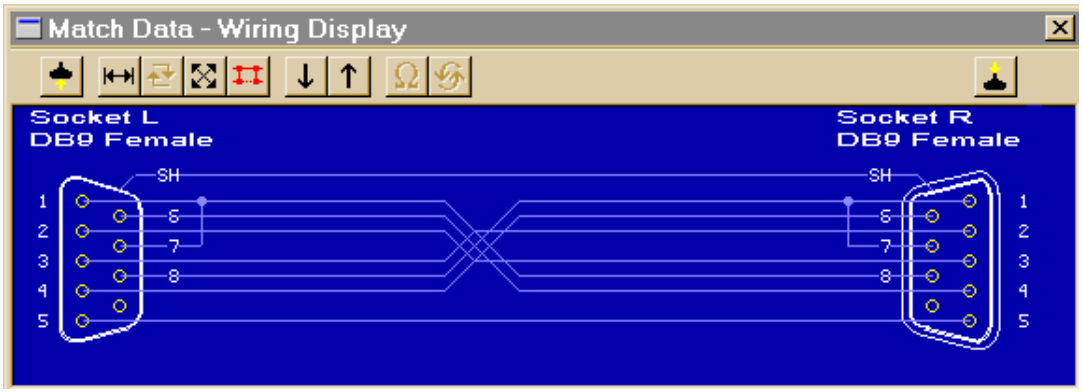
The cable shown below is intended for RS232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals, and a DB25 connector would be necessary.

NOTE: Not all null modem cables connect handshaking lines the same way. In this cable, Request-to-Send (RTS, pin 7) asserts the Carrier Detect (pin 1) on the same side and the Clear-to-Send (CTS, pin 8) on the other side of the cable.

This device may also be available in the form of an adapter.

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 9-pin **DTE** (computer)



Cable image created by CableEye®

---

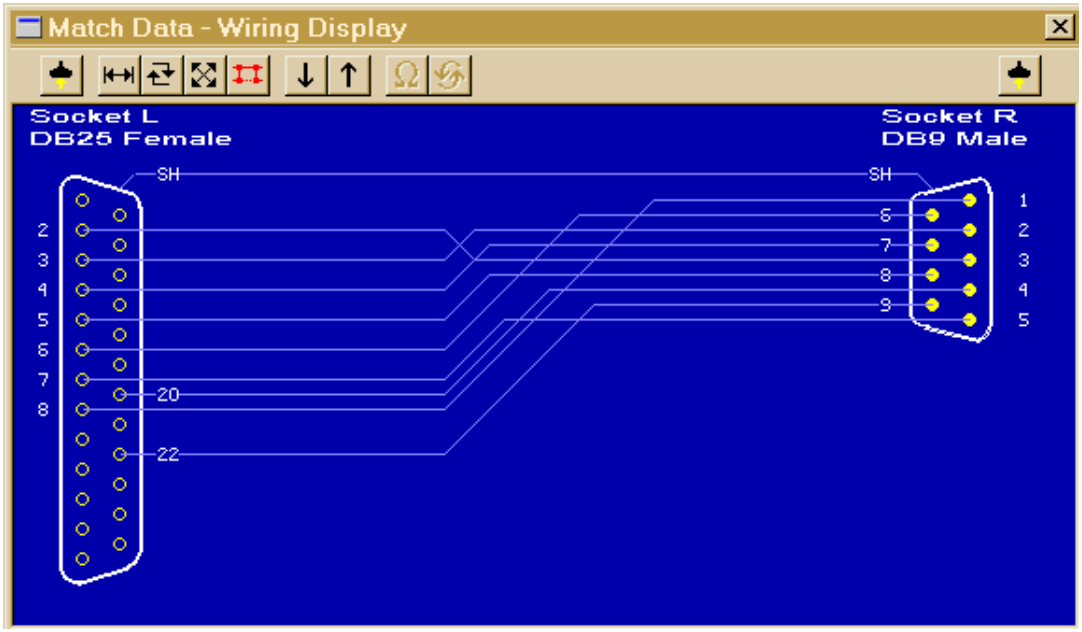
4 - DB25 to DB9 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB25 DTE side are directly mapped to the DB9 assignments for a DTE device. Use this to adapt a 25-pin COM connector on the back of a computer to mate with a 9-pin serial DCE device, such as a 9-pin serial mouse or modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

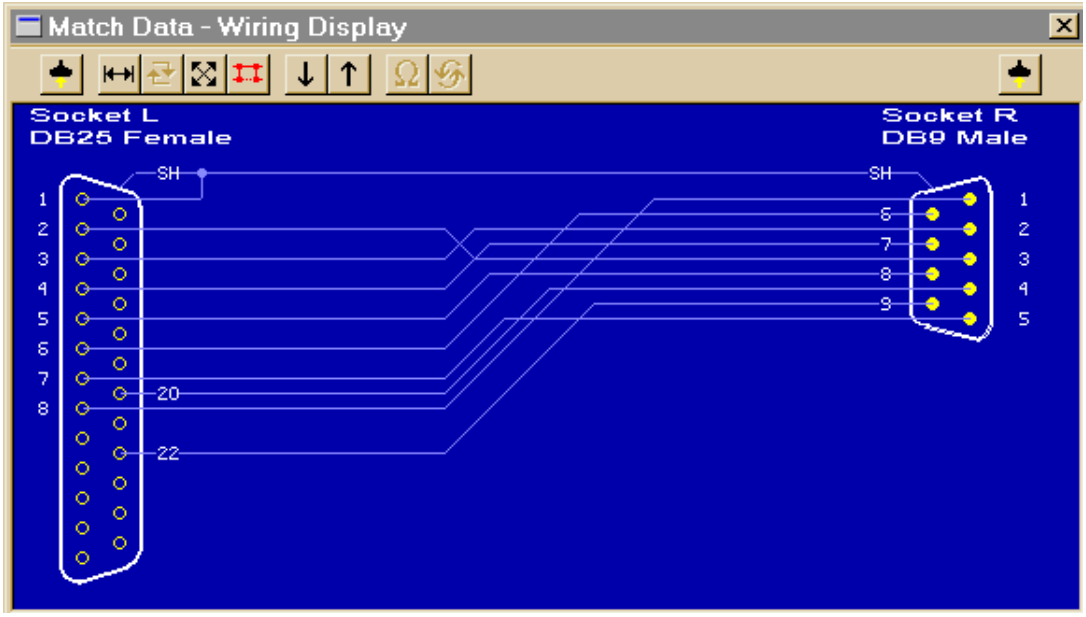
5 - DB25 to DB9 Adapter (pin 1 connected to shield)
Next Cable | Previous Cable || Next Topic

84K

This adapter has the same wiring as the previous cable (#4) except that pin 1 is wired to the connector shell (shield). Note that the cable's shield is usually a foil blanket surrounding all conductors running the length of the cable and joining the connector shells. Pin 1 of the EIA232 specification, called out as "shield", may be separate from the earth ground usually associated with the connector shells.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

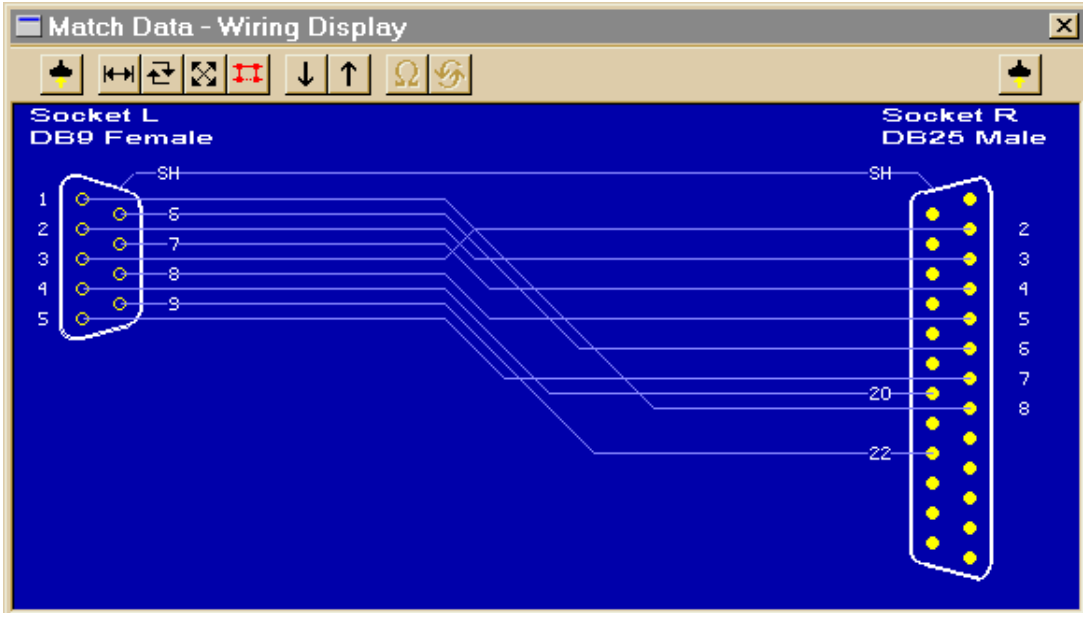
6 - DB9 to DB25 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB9 DTE side are directly mapped to the DB25 assignments for a DTE device. Use this to adapt a 9-pin COM connector on the back of a computer to mate with a 25-pin serial DCE devices, such as a modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

7 - DB25 All-Line Direct Extension
Next Cable | Previous Cable || Next Topic

84K

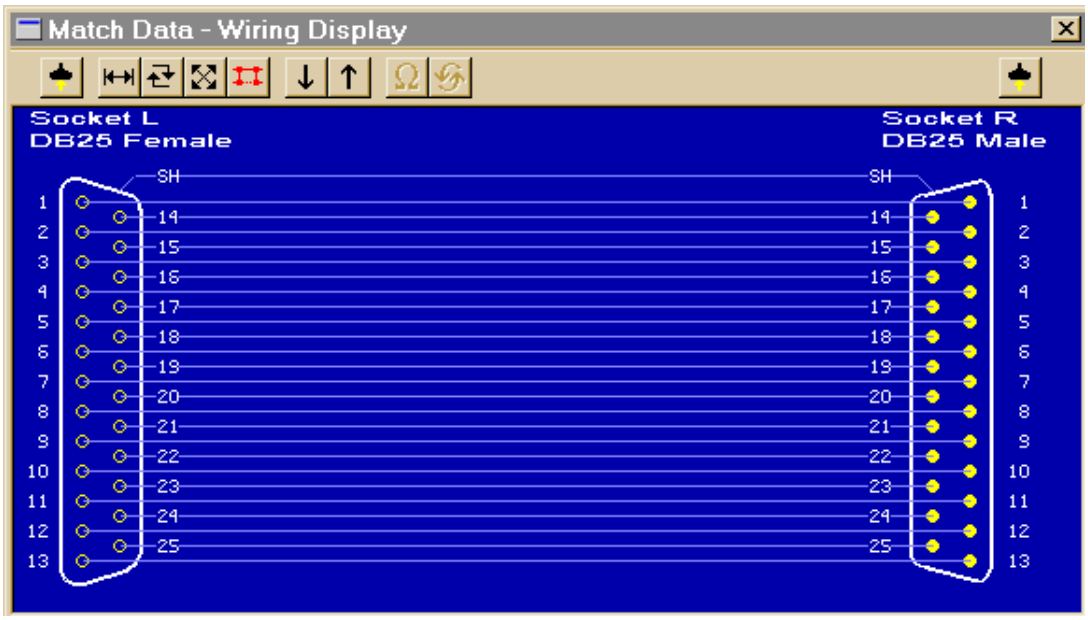
This shows a 25-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 25 pins plus shield are directly extended from DB25 Female to DB25 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any serial device that uses a DB25 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Caution: the male end of this cable (right) also fits a PC's parallel printer port. You may use this cable to extend the length of a printer cable, but DO NOT attach a serial device to the computer's parallel port. Doing so may cause damage to both devices.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

8 - DB25 Loopback Connector
Next Cable | Previous Cable || Next Topic

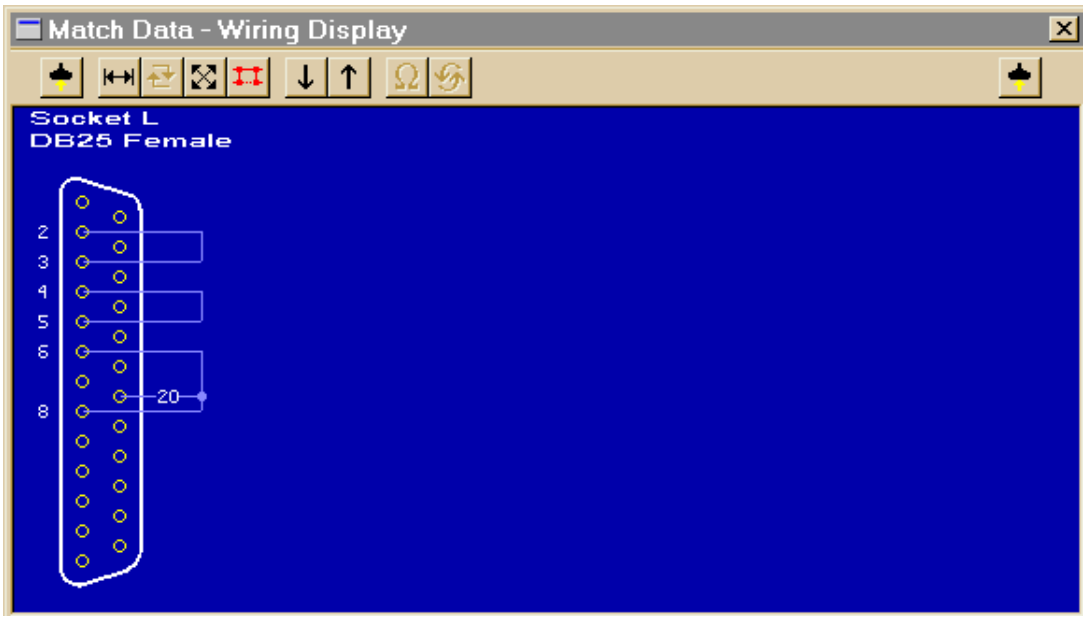
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB25 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable the verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

9 - DB25 Null Modem (no handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

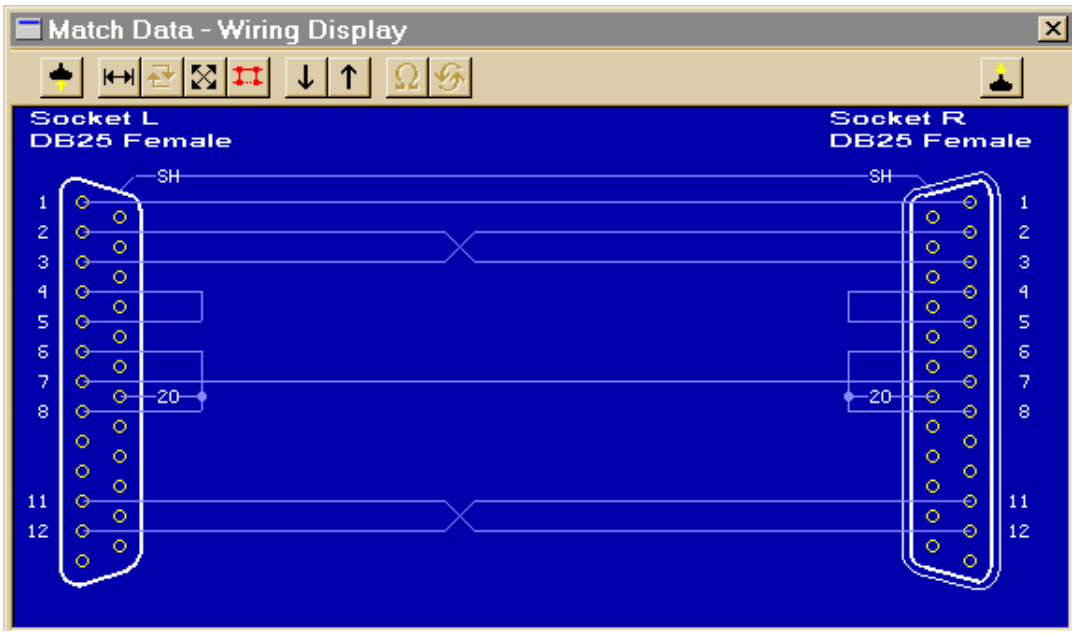
84K

Note that Pins 11 and 12 are not necessary for this null modem cable to work. As is often the case, the manufacturer of equipment that uses this cable had a proprietary application in mind. We show it here to emphasize that custom serial cables may include connections for which no purpose is clear.

IMPORTANT: This cable employs NO handshaking lines between devices. The handshake signals on each side are artificially made to appear asserted by the use of self-connects on each side of the cable (for example, between pins 4 and 5). Without hardware handshaking, you risk buffer overflow at one or both ends of the transmission unless STX and ETX commands are inserted in the dataflow by software.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

10 - DB25 Null Modem (standard handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

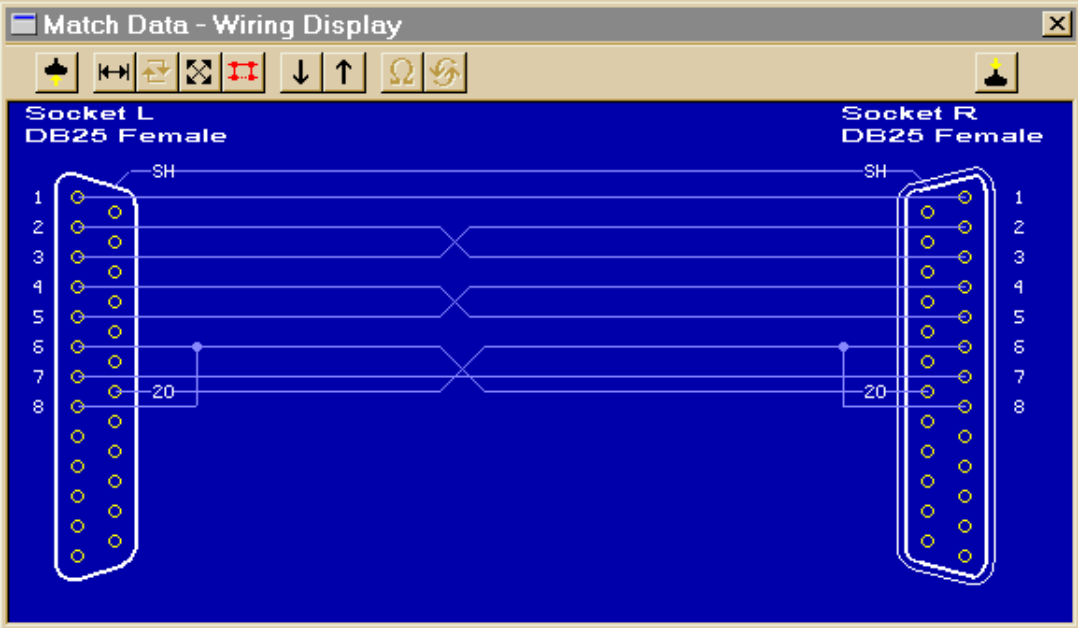
84K

The cable shown below is intended for EIA232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals not shown here.

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6) and the Request to Send (pin 5) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

11 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

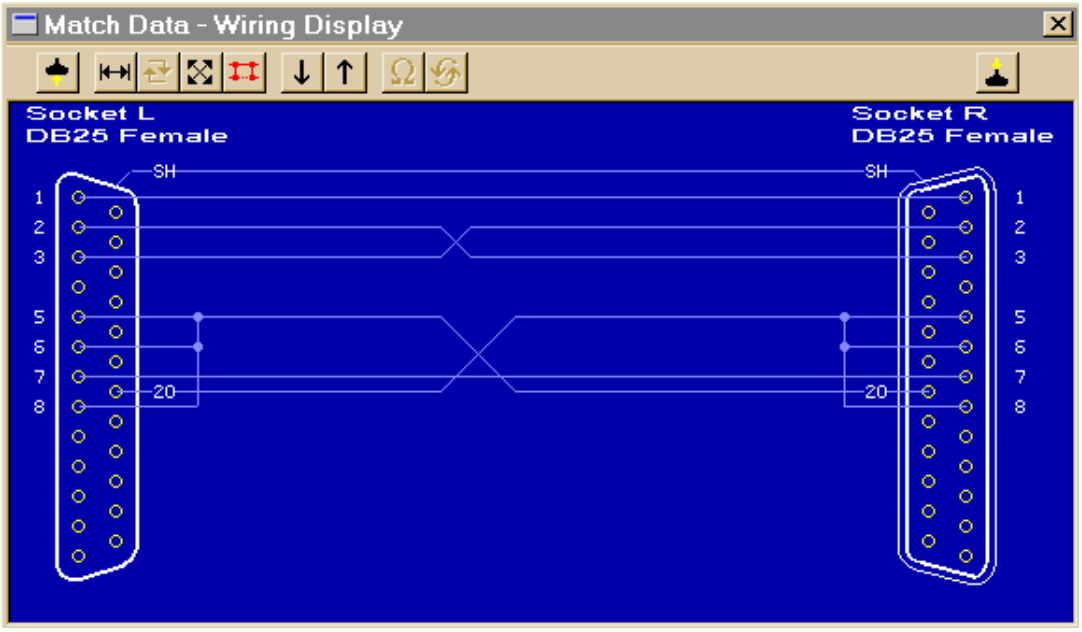
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear to Send (pin 5), DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

12 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

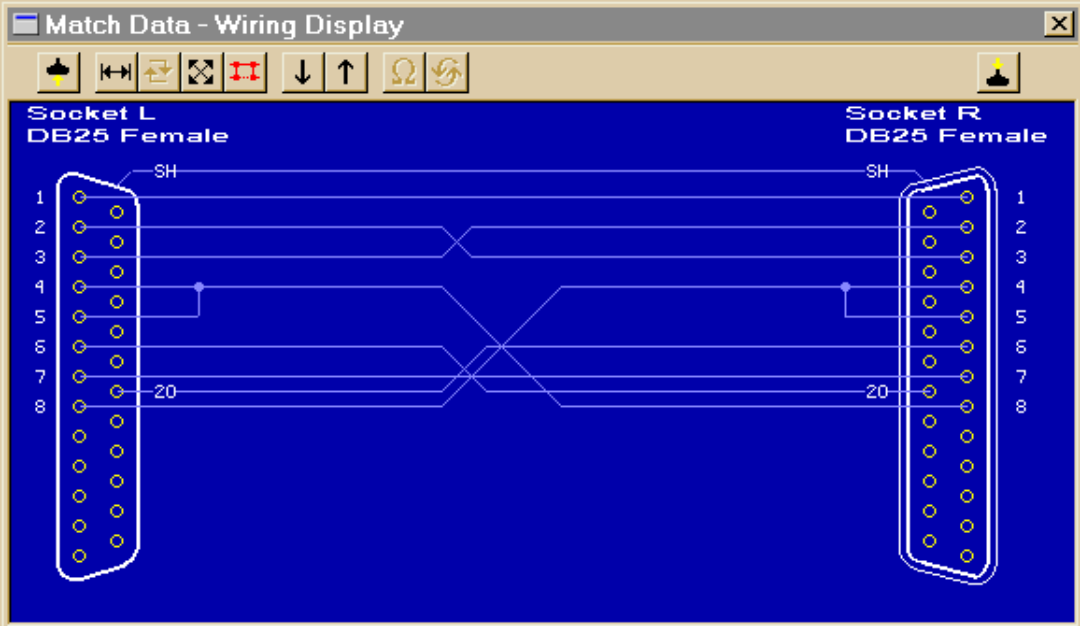
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the Request-to-Send (pin 4) on one side asserts the Clear-to-Send (pin 5) on the SAME side (self-connect) and the Carrier Detect (pin 8) on the other side. The other handshaking signals are employed in a conventional manner.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

13 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

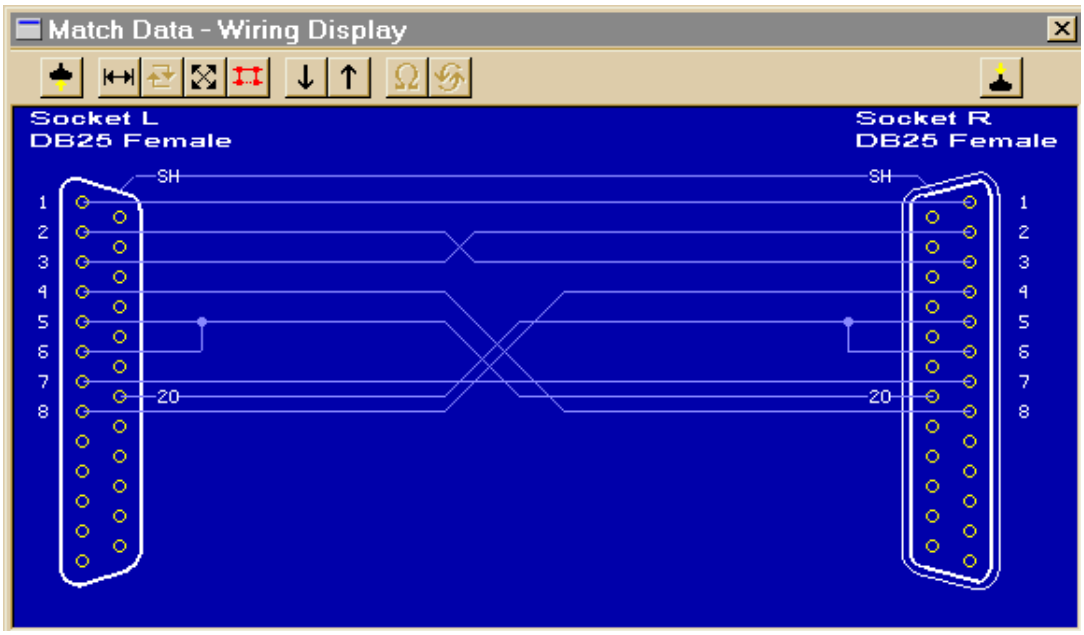
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear-to-Send (pin 5) and the DCE Ready (pin 6) on the other side. Request-to-Send (pin 4) on one side asserts Received Line Signal Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

14 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

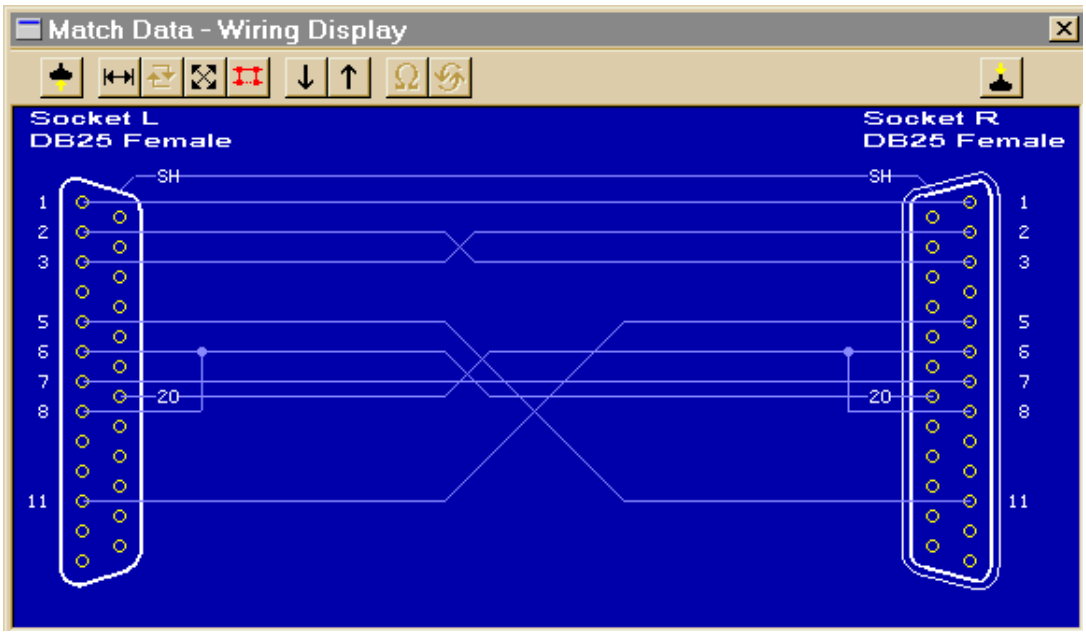
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side. Request to Send (pin 4) is unused, and Clear-to-Send (pin 5) is driven by a proprietary signal (pin 11) determined by the designer of this cable.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

15 - DB25 Null Modem Cable (synchronous communications)
Next Cable | Previous Cable || Next Topic

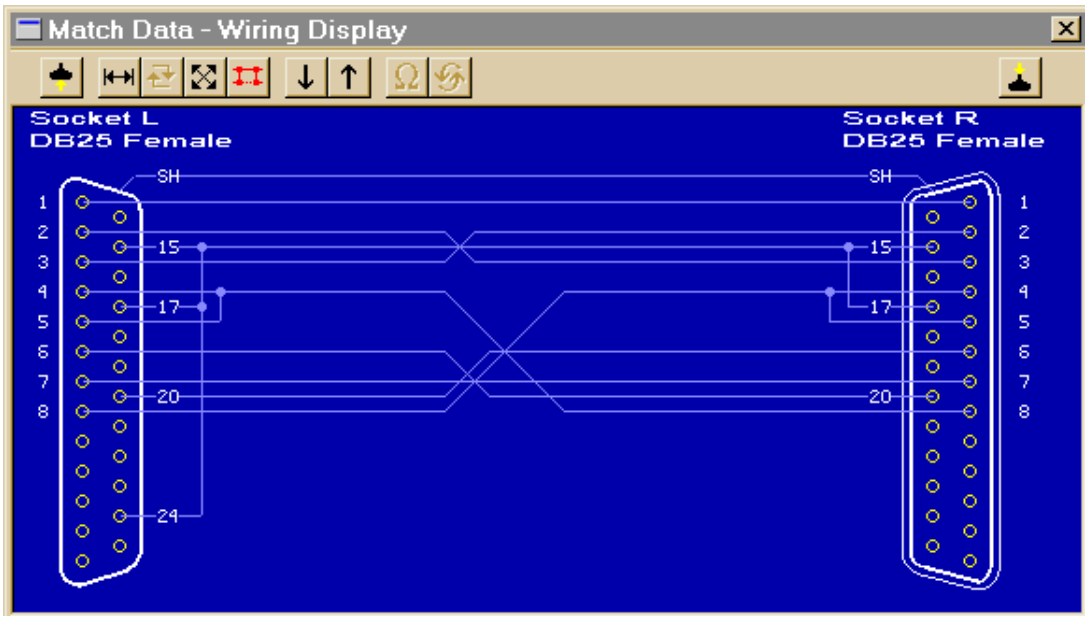
84K

This female-to-female cable is intended for **synchronous** EIA232 connections, and is designed to connect two DTE devices. It contains the standard connections of an asynchronous null modem cable, plus additional connections on pins 15, 17, and 24 for synchronous timing signals. To connect two DCE devices, use a male-to-male equivalent of this cable.

For synchronous communications, the null modem cable includes an additional conductor for timing signals, and joins pins 15, 17, and 24 on one side to pins 15 and 17 on the other. Pin 24 on the right side should connect to the timing signal source.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

16 - DB25 Null Modem Cable (unconventional, may pose risk)
(no more) | Previous Cable || Next Topic

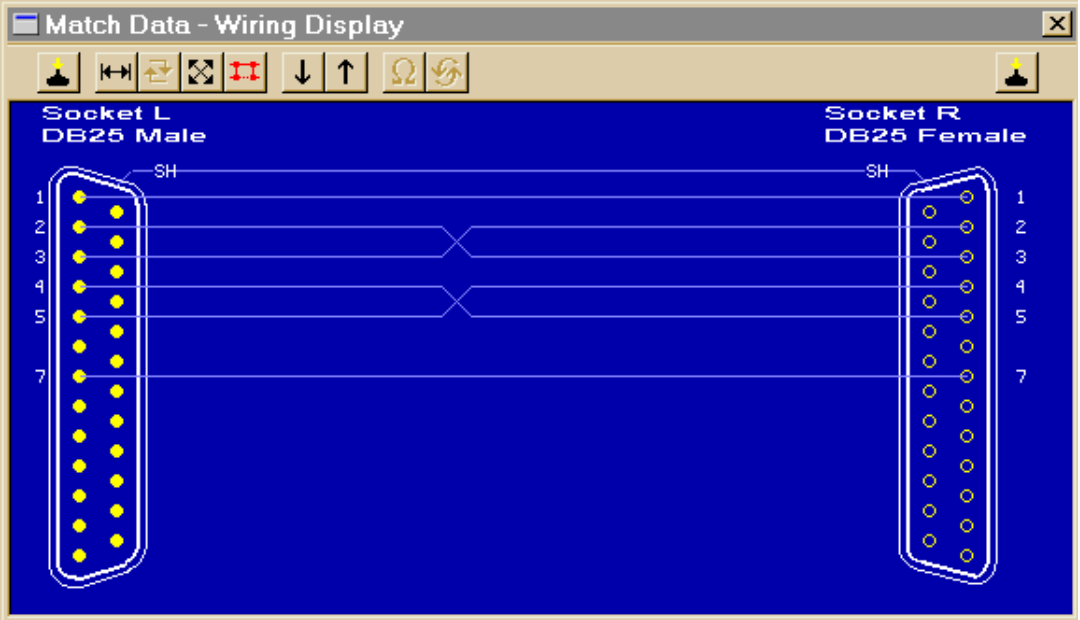
This simplified null modem cable uses only Request-to-Send (pin 4) and Clear-to-Send (pin 5) as handshaking lines; DTE Ready, DCE Ready, and Carrier Detect are not employed, so this cable should not be used with modems.

80K

CAUTION! Normally, null modem cables have the same gender on each connector (either both male for two DTE devices, or both female for two DCE devices). This cable would be used when the gender on one of the devices does not conform to the standard. However, the opposite genders imply usage as a straight through cable, and if used in that manner will not function. Further, if used as a standard null-modem between two computers, the opposite gender allows you to connect one end to the parallel port, an impermissible situation that may cause hardware damage.

Left Side: Connect to 25-pin **DTE** (computer) with Gender Changer

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

Signal Definitions
Next Topic | Previous Topic | TOC

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

Signal Ground and Shield

Pin 7, Pin 1, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 - Ground All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

Primary Communications Channel

Pin 2 - Transmitted Data (TxD) This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD) This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS) This signal is asserted (logic ’0’, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

> *NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS) This signal is asserted (logic ’0’, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

> *NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the

EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

Secondary Communications Channel
Next Topic | Previous Topic | TOC

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

Modem Status and Control Signals
Next Topic | Previous Topic | TOC

Pin 6 - DCE Ready (DSR) When originating from a modem, this signal is asserted (logic ’0’, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic ’1’, negative voltage).

IMPORTANT: If DCE Ready originates from a device other than a modem, it may be asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic ’0’, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR) This signal is asserted (logic ’0’, positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the

connection. When DTE Ready is deasserted (logic ’1’, negative voltage), the modem is switched to "on-hook" to terminate the connection.

> *IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

Pin 8 - Received Line Signal Detector (CD) (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic ’0’, positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem’s requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD) This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI) This signal is relevant when the DCE device is a modem, and is asserted (logic ’0’, positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic ’0’, positive voltage) selects the higher baud rate.

Transmitter and Receiver Timing Signals
Next Topic | Previous Topic | TOC

Pin 15 - Transmitter Signal Element Timing (TC) (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic ’1’ to logic ’0’ (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC) (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC) (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic ’1’ to logic ’0’ transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

Channel Test Signals

Next Topic | Previous Topic | TOC

Pin 18 - Local Loopback (LL) This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic ’0’, positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL) This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic ’0’, positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

Pin 25 - Test Mode (TM) This signal is relevant only when the DCE device is a modem. When asserted (logic ’0’, positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

Electrical Standards

Next Topic | Previous Topic | TOC

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic ’1’, and positive voltage represents logic ’0’. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic ’1’. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

Common Signal Ground

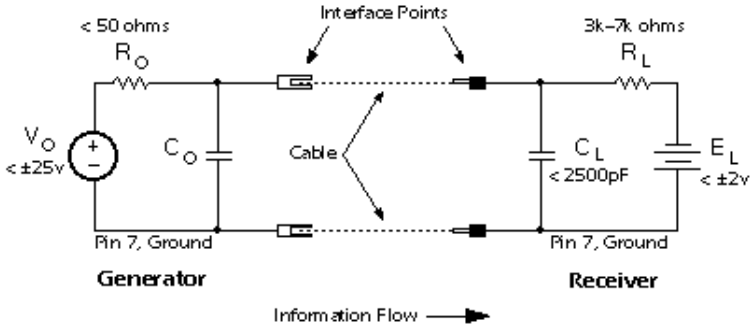
Next Topic | Previous Topic | TOC

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

NOTE: optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

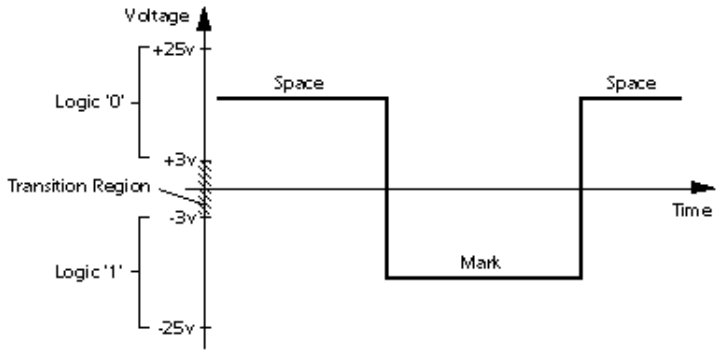
Signal Characteristics
Next Topic | Previous Topic | TOC

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "Co" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "Ro" and "Vo" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.

Signal State Voltage Assignments - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic ’1’ (the marking condition), whereas voltages of +3v to +25v are considered logic ’0’ (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern

digital designs.

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic ’1’ (mark), and +8 to +14 volts for logic ’0’ (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

> *IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of -3v for logic ’1’, and +3v for logic ’0’. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

Short-Circuit Tolerance - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of ±25 volts without sustaining damage.

> *CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the ±25-volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

Fail-Safe Signals - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic ’1’ (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.
Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic ’0’) to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic ’1’ to logic ’0’ or vice versa must abide by several requirements, as follows:

1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.

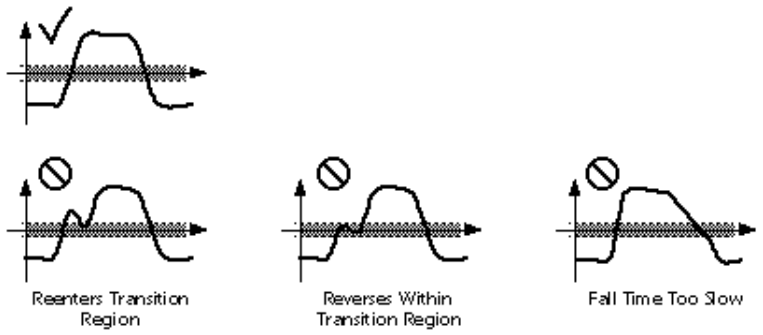
2 - For control signals, the transit time through the transition region should be less than 1ms.

3 - For Data and Timing signals, the transit time through the transition region should be

a - less than 1ms for bit periods greater than 25ms,

b - 4% of the bit period for bit periods between 25ms and 125µs,

c - less than 5µs for bit periods less than 125µs.
The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.

---

4 - The slope of the rising and falling edges of a transition should not exceed 30v/µS. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

Accepted Simplifications of the Standard
Previous Topic | TOC

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

Return to TOC

The RS232 STANDARD

A Tutorial with Signal Names and Definitions

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Send Us Your Comments . . .

Contents

What is EIA232?
Next Topic | TOC

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers, thereby fostering the benefits of

mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 30+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

Likely Problems when Using an EIA232 Interface
Next Topic | Previous Topic | TOC

During this 30-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

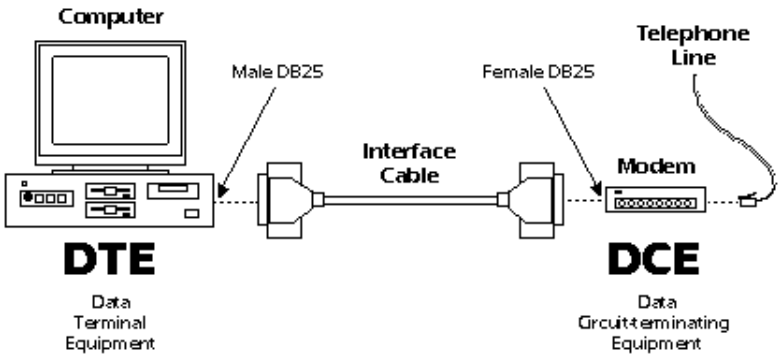
Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

Pin Assignments
Next Topic | Previous Topic | TOC

Go to DTE Pinout (looking into the computer’s serial connector)
Go to DCE Pinout (looking into the modem’s serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25

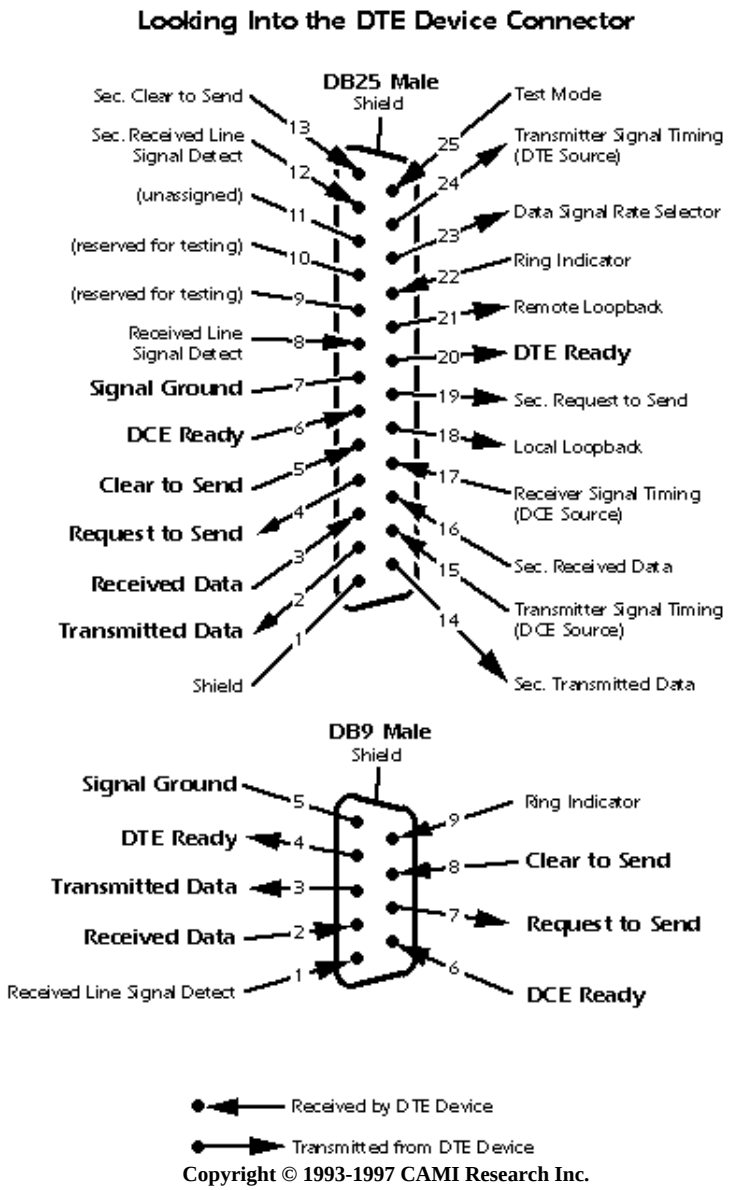
connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.

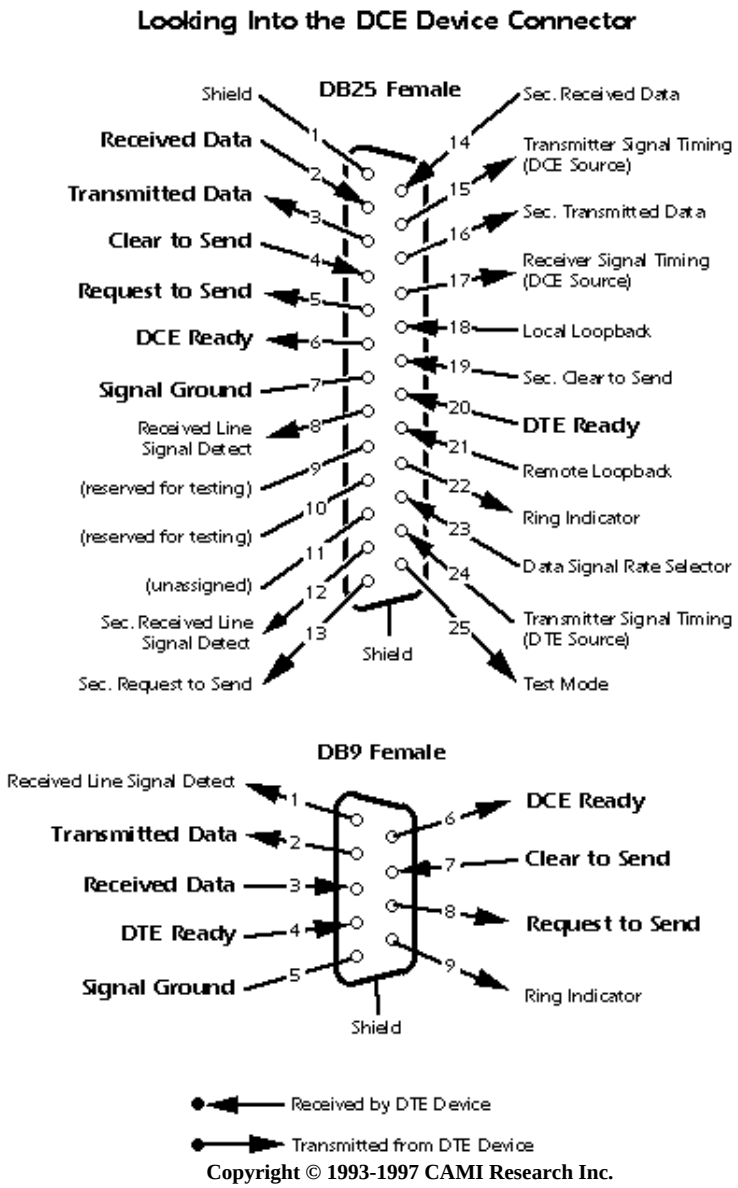
Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



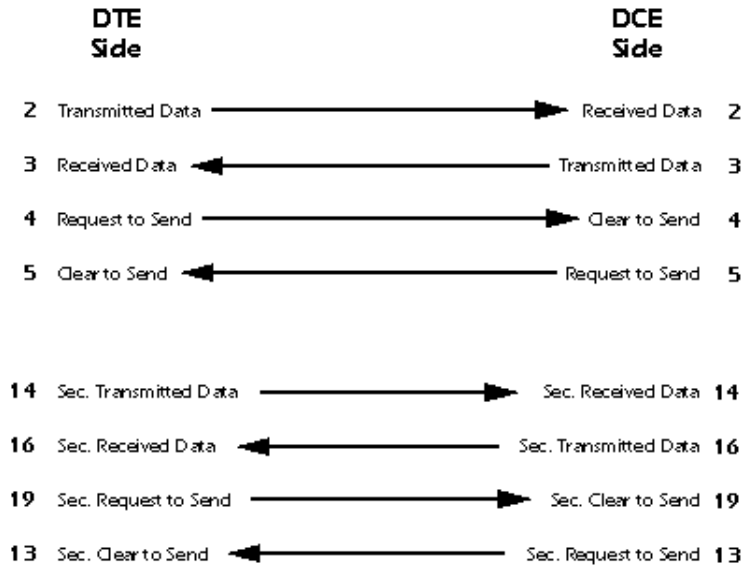
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

> ***IMPORTANT:*** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:



Signal Definitions
Next Topic | Previous Topic | TOC

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals

provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

Signal Ground and Shield
Next Topic | Previous Topic | TOC

Pin 7, Pin 1, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 - Ground All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

Primary Communications Channel
Next Topic | Previous Topic | TOC

Pin 2 - Transmitted Data (TxD) This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

NOTE: Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD) This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

NOTE: Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS) This signal is asserted (logic ’0’, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

NOTE: Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the

EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS) This signal is asserted (logic ’0’, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

NOTE: Pin 5 on the DCE device is commonly labeled "Request to Send", although by the EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

Secondary Communications Channel
Next Topic | Previous Topic | TOC

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

Modem Status and Control Signals
Next Topic | Previous Topic | TOC

Pin 6 - DCE Ready (DSR) When originating from a modem, this signal is asserted (logic ’0’, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic ’1’, negative voltage).

IMPORTANT: If DCE Ready originates from a device other than a modem, it may be

asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic ’0’, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR) This signal is asserted (logic ’0’, positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the connection. When DTE Ready is deasserted (logic ’1’, negative voltage), the modem is switched to "on-hook" to terminate the connection.

IMPORTANT: If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

Pin 8 - Received Line Signal Detector (CD) (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic ’0’, positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem’s requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD) This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI) This signal is relevant when the DCE device is a modem, and is asserted (logic ’0’, positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic ’0’, positive voltage) selects the higher baud rate.

Transmitter and Receiver Timing Signals
Next Topic | Previous Topic | TOC

Pin 15 - Transmitter Signal Element Timing (TC) (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic ’1’ to logic ’0’ (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC) (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC) (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic '1' to logic '0' transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

Channel Test Signals

Next Topic | Previous Topic | TOC

Pin 18 - Local Loopback (LL) This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic '0', positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL) This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic '0', positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

Pin 25 - Test Mode (TM) This signal is relevant only when the DCE device is a modem. When asserted (logic '0', positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

Electrical Standards

Next Topic | Previous Topic | TOC

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic '1', and positive voltage represents logic '0'. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic '1'. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

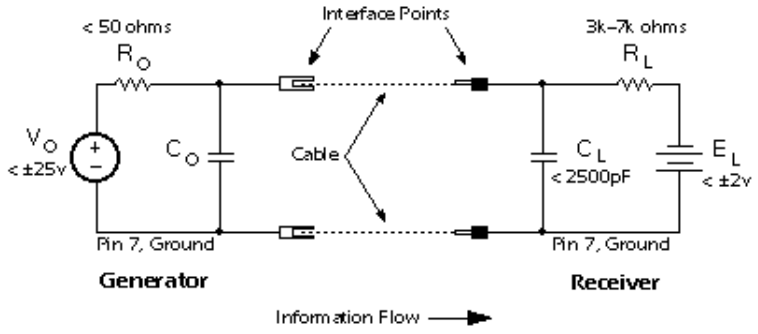
Common Signal Ground
Next Topic | Previous Topic | TOC

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

> *NOTE:* optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

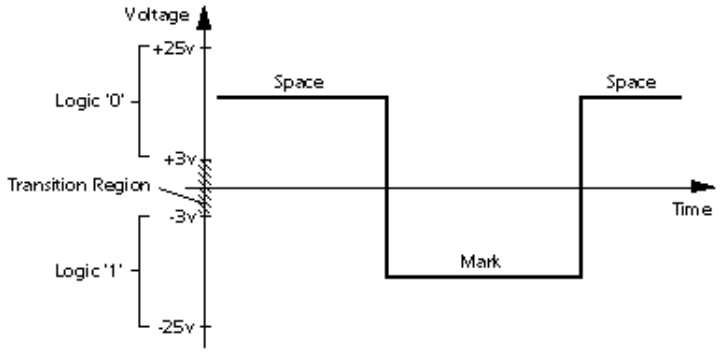
Signal Characteristics
Next Topic | Previous Topic | TOC

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "Co" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "Ro" and "Vo" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.

Signal State Voltage Assignments - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic ’1’ (the marking condition), whereas voltages of +3v to +25v are considered logic ’0’ (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern digital designs.

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic ’1’ (mark), and +8 to +14 volts for logic ’0’ (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

> *IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of &shyp;3v for logic ’1’, and +3v for logic ’0’. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

Short-Circuit Tolerance - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of ±25 volts without sustaining damage.

> *CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the ±25-volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

Fail-Safe Signals - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic ’1’ (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.
Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic ’0’) to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

Signal Timing
Next Topic | Previous Topic | TOC

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic ’1’ to logic ’0’ or vice versa must abide by several requirements, as follows:

1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.

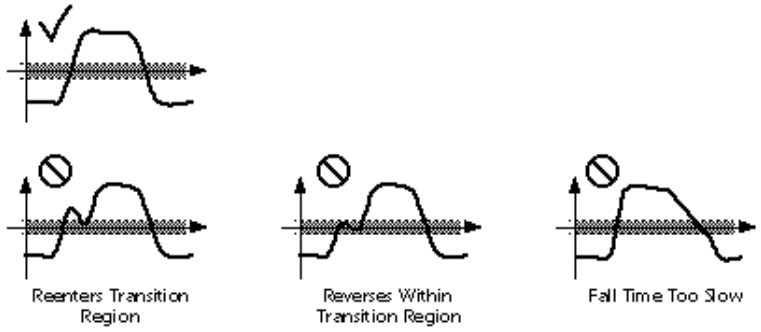
2 - For control signals, the transit time through the transition region should be less than 1ms.

3 - For Data and Timing signals, the transit time through the transition region should be

a - less than 1ms for bit periods greater than 25ms,

b - 4% of the bit period for bit periods between 25ms and 125µs,

c - less than 5µs for bit periods less than 125µs.
The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.

4 - The slope of the rising and falling edges of a transition should not exceed 30v/µS. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

Accepted Simplifications of the Standard
Previous Topic | TOC

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

Return to TOC

SDTP, PPP Serial Data Transport Protocol

Description:

Protocol suite: PPP.
Type: PPP network layer protocol.
PPP protocol: 0x0049
Working groups: pppext, Point-to-Point Protocol Extensions.

Serial Data Transport Protocol (SDTP) is used for synchronous serial data compression over a PPP link.

Before any SDTP packets may be communicated, PPP must reach the Network-Layer Protocol phase, and the SDTP Control Protocol must reach the Opened state.

The maximum length of the SDTP datagram transmitted over a PPP link is limited only by the negotiated Maximum-Frame-Size and the maximum length of the Information field of a PPP encapsulated packet. Note that if compression is used on the PPP link, this the maximum length of the SDTP datagram may be larger or smaller than the maximum length of the Information field of a PPP encapsulated packet, depending on the particular compression algorithm and protocol used.

RFC 1963, pages 1 - 3:

> This document describes a new Network level protocol (from the PPP point of view), PPP Serial Data Transport Protocol, that provides encapsulation and an associated Serial Data Control Protocol (SDCP) for transporting serial data streams over a PPP link. This protocol was developed for the purpose of using PPP's many features to provide a standard method for synchronous data compression. The encapsulation uses a header structure based on that of the ITU-T Recommendation V.120.
>
> This document is a product of the TR30.1 ad hoc committee on compression of synchronous data. It represents a component of a proposal to use PPP to provide compression of synchronous data in DSU/CSUs.
>
> In addition to providing support for multi-protocol datagrams, the Point-to-Point Protocol (PPP) has defined an effective and robust negotiating mechanism that can be used on point to point links. When used in conjunction with the PPP Compression Control Protocol and one of the PPP Compression Protocols, PPP provides an interoperable method of employing data compression on a point-to- point link.
>
> This document provides a PPP encapsulation for serial data, specifying a transport protocol, PPP Serial Data Transport Protocol (PPP-SDTP), and an associated control protocol, PPP Serial Data Control Protocol (PPP-SDCP). When these protocols are added to above mentioned PPP protocols, PPP can be used to provide compression of serial data on a point-to-point link.

This first edition of PPP-SDTP/SDCP covers HDLC-like synchronous serial data and asynchronous serial data. It does this by using a terminal adaption header based on that of ITU-T Recommendation V.120. Support may be added in the future for other synchronous protocols as the marketplace demands.

The V.120 terminal adaption header allows transported data frames to be split over several packets, supports the transport of DTE port idle and error information, and optionally supports the transport of DTE control state information.

In addition to the V.120 Header, fields can be added to the packet format through negotiation to provide support for features not included in the V.120 header. The extra fields are: a Length Field, which is used to distinguish packets in compound frames, and a Port field, which is used to provide multi-port multiplexing capability. The protocol also allows reserved bits in the V.120 header to be used to transport non-octet aligned frames and to provide a flow control mechanism.

To provide these features, PPP-SDTP permits a single frame format to be selected from several possible formats by using PPP-SDCP negotiation. The terminal adaption header can be either fixed length or variable length, to allow either simplicity or flexibility.

The default frame format places the terminal adaption header at the end of the packet. This permits optimal transmitter timelines when user frames are segmented and compression is also used in conjunction with this protocol.

Packet format:

Glossary:

V.120.
CCITT Recommendation V.120 (09/92), "Support by an ISDN of Data Terminal Equipment with V-Series Type Interfaces with Provision for Statistical Multiplexing", 1993.

RFCs:

[RFC 1963] PPP Serial Data Transport Protocol (SDTP).

Disclaimer: This description is completely unofficial. Most of the information presented here is discovered by me, Eugene Crosser, while snooping the serial line and by trial and error. I never had an official protocol description, have never seen any related software source code, and have never done reverse engineering of any related software. This description may be incomplete, inaccurate or completely wrong. You are warned.

Some information is taken from 'camediaplay' package by Jun-ichiro Itoh <itojun@itojun.org>, from the findings of Thierry Bousch <bousch%linotte.uucp@topo.math.u-psud.fr> TsuruZoh Tachibanaya <tsuruzoh@butaman.ne.jp> and from other (open) sources and not checked by me.

Serial Protocol of Some Digital Cameras

Several models of digital cameras, namely Epson, Sanyo, Agfa and Olympus cameras, seem to use the same protocol for communication with the host. Follows the description of the high-level protocol they use over the serial line.

Protocol Basics

The host and the camera exchange with data packets and individual bytes. Serial line paramaters used are: 8bit, no parity. No flow control is used. All arithmetic data is transmitted least significant byte first ("little endian").

Protocol Elements

The elementary units of the protocol are:

Initialization Byte	NUL	0x00
Action Complete Notification	ENQ	0x05
Positive Achnowledgement	ACK	0x06
Unable to Execute Command	DC1	0x11
Negative Acknowledgement, also Camera Signature	NAK	0x15
Packet	Variable length sequence of bytes	
Termination Byte		0xff

Packet structure

The packet has the following structure:

Offset	Length	Meaning
0	1	Packet type
1	1	Packet subtype/sequence
2	2	Length of data
4	variable	Data
-2	2	checksum

Known packet types are:

Type	Description
0x02	Data packet that is not last in sequence
0x03	Data packet that is last in sequence
0x1b	Command packet

Data packets that are sent in responce to a single command are numbered starting from zero. If all requested data fits in one packet, it has type 0x03 and sequence 0.

Command packet has subtype 0x43 or 0x53. Only the first command packet in a session has subtype 0x53.

Maximum length of data field in a packet is 2048 bytes, which yields in 2054 total packet length.

Checksum is a simple 16 bit arithmetic sum of all bytes in the data field. As already mentioned above, length and checksum values are transmitted least significant byte first.

Flow of Control

A communication session flow is as follows:

Host	Camera
Port speed set to 19200 baud	
Host sends init byte 0x00	
	Camera responds with signature 0x15
Host sends command packet with subtype 0x53 and "set speed" command	
	Camera sends ACK 0x06
Port speed set to the new value	
Host sends command	
	Camera responds with either ACK plus optionally "action taken" notifier or data packet sequence
Host sends ACK to every data packet	
... Command - reply cycle repeated ...	
	Camera sends 0xff and resets after a few seconds (value is model-dependant) of inactivity

If the camera does not respond to a command in reasonable time, or responds with a NAK, the command can be resent. If the camera does not provide a complete data packet in reasonable time, or the data packet is corrupt (checksum does not match), the host can request resending of the packet by sending NAK instead of ACK.

Command format and codes

Command is a sequence of bytes sent in the data field of a command packet. Command format is as follows:

Offset	Length	Description
0	1	Command code
1	1	Register number or subcode
2	variable	Optional argument

Five command codes are known:

Code	Argument	Description
0	int32	Set value of integer register
1	none	Read value of integer register
2	vdata	Take action unrelated to registers
3	vdata	Set value of vdata register
4	none	Read value of vdata register

Commands 0 and 3 are replied with a single ACK 0x06. Command 2 is replied with an ACK 0x06 followed by an "action complete" notifier 0x05. Commands 1 and 4 are replied with a sequence of data

packets, each of them must be ACK’ed by the host.

Command 0 must be issued with a 4 byte argument containg the new value for the register (bytes in "LSB first" order). Command 2 typically is issued with a single zero byte as an argument. Command 3 is issued with an argument of variable number of bytes. If this is a printable string, it should **not** include the trailing zero byte.

Camera replies to the command 1 with a single data packet containing 4 bytes of a 32bit integer (in "LSB first" order). Camera replies to the command 4 with a sequence of data packets with variable number of data bytes. Note that if a printable string is returned, it **is** terminated with a zero byte, and thus may be safely printed or otherwise treated as a normal C language character string.

Registers

The following registers are known (read/writablity info is inaccurate):

No.	Type	R/W	Description
1	int32	R/W	Resolution: 1 - Std, 2 - Hi, 3 - Ext, other values possible
2	int32	R/W	Clock in UNIX time_t format
3	int32	R/W	Shutter speed (microseconds), 0 - auto
4	int32	W	Current frame number (or animation number if hi order byte is 0xff)
5	int32	R/W	Aperture: 0 - Auto, 1 - Low, 2 - Med, 3 - ?, 4 - Hi
6	int32	R/W	Color mode: 1 - Color, 2 - B/W
7	int32	R/W	Flash mode: 0 - Auto, 1 - Force, 2 - Off, 3 - Anti RedEye, 4 - Slow sync
8	int32	R/W	Unknown (128)
9	int32	R/W	Unknown (128)
10	int32	R	No. of frames in current folder
11	int32	R	No. of frames left
12	int32	R	Length of current frame *
13	int32	R	Length of current thumbnail *
14	vdata	R	Current frame data *
15	vdata	R	Current thumbnail data *
16	int32	R	Battery capacity percentage
17	int32	R/W	Communication speed 1 - 9600 .. 5 - 115200, 6 - 230400, 256 - 9600 .. 264 - 911600 (sync?)
18	int32	R	Unknown (1)
19	int32	R/W	Bright/Contrast: 0 - Normal, 1 - Contrast+, 2 - Contrast-, 3 - Brightnes+, 4 - Brightnes-
20	int32	R/W	White balance: 0 - Auto, 1 - Sunny, 2 - Incandescent, 3 - Fluorescent, 5 - Flash, 6- White preset, 255 - Cloudy
21	vdata	R/W	Unused

22	vdata	R/W	Camera I.D.
23	int32	R/W	Autoshut on host timer (seconds)
24	int32	R/W	Autoshut in field timer (seconds)
25	vdata	R/W	Serial No. (string)
26	vdata	R	Version
27	vdata	R/W	Model
28	int32	R	Available memory left
29	vdata	R/W	Upload image data to this register
30	int32	W	LED: 0 - Off, 1 - On, 2 - Blink
31	vdata	R	Unknown ("\0")
32	int32	R	Put "magic spell" 0x0FEC000E here before uploading image data
33	int32	R/W	Focus mode: 1 - Macro, 2 - Normal, 3 - Infinity/fisheye
34	int32	R	Operation mode: 1 - Off, 2 - Record, 3-Play, 6-Thumbnail
35	int32	R/W	LCD brightness 1 to 7
36	int32	R	Unknown (3)
37	vdata	R	Unknown ("\0")
38	int32	R	LCD autoshut timer (seconds)
39	int32	R	Protection state of current frame *
40	int32	R	True No. of frames taken
41	int32	R/W	LCD date format: 1 - ’YY MM DD, 2 - DD MM ’HH
42	vdata	R	Unknown ("")
43	vdata	R	Audio data description block * 0: expanded .wav length 1: compressed .wav length 3: Unknown (0) 4: Unknown (0) 5: Unknown (0) 6: Unknown (0) 7: Unknown (0)
44	vdata	R	Audio data *
45	vdata	R	Unknown ("")
46	vdata	R	Camera summary data: 32 bytes with copies of 8 other registers 0: Reg 1 (Resolution) 1: Reg 35 (LCD brightness) or Reg 7 (Flash mode) 2: Reg 10 (Frames taken) or Unknown 3: Unknown (0) or Unknown 4: Unknown (0) or Reg 16 (Battery capacity) 5: Unknown (0) or Reg 10 (Frames taken) 6: Unknown (0) or Reg 11 (Frames left) 7: Number of animations taken

47	vdata	R	Picture summary data: 32 bytes or 8 int32's * 0: Hi order byte: unknown, next 3 bytes: Length of current image 1: Length of current thumbnail 2: Audio data length (expanded) 3: Resolution 4: Protection state 5: TimeDate 6: Unknown (0) 7: Animation type: 1 - 10ms, 2 - 20ms
48	vdata	R	Manufacturer
49	vdata	R	Unknown ("")
50	int32	R	Unknown (0)
51	int32	R/W	Card detected: 1 - No, 2 - Yes
52	vdata	R/W	Unknown ("")
53	int32	R/W	Language: 3 - english, 4 - french, 5 - german, 6 - italian, 8 - spanish, 10 - dutch
54-59	vdata	R	Unknown ("")
60	int32	R	True No. of frames taken
61-68	vdata	R	Unknown ("")
69	vdata	R	Exposure Compensation 8 bytes 0: compensation value -20 to +20 1: 0 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
70	int32	R/W	Exp. meter: 2 - Center weighted, 3 - Spot, 5 - Multi element matrix
71	vdata	R/W	Effective zoom in tenths of millimeters: 8 bytes 0: LSB 1: MSB 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
72	int32	R/W	Bitmap: 1 - AEL/WBL, 2 - Fisheye, 4 - Wide, 8 - Manual zoom, 16 - B/W, 256 - 1.25x, 512 - 1.6x, 768 - 2.0x, 1024 - 2.5x, 1280 - off
73-76	vdata	R	Unknown ("")
77	int32	W	Size of data packet from camera (default 0x800)
78	vdata	R	Unknown ("")

79	vdata	R	Filename of current frame *
80-81	vdata	R	Unknown ("")
82	int32	W	Unknown (enable folder features? Write 60 here)
83	int32	R/W	Folder navigation When read, return number of folders on the card. When written **without** data, reset folder system (?) Or select current folder by its number
84	vdata	R/W	Current folder name (may read or set)
85	vdata	R	Unknown ("")
86	int32	R/W	Digital zoom; 0 - 1X, otherwise zoom factor x 100 (i.e. in percent)
87-90	vdata	R	Unknown ("")
91	vdata	R	Current folder I.D. and name

*** Note:** Marked registers only become useful for reading after setting register 4. If value of 0 assigned to register 4 after doing action 5, subsequent retrieval of picture data gives the "live preview".

For command 2, the second byte is action code not register number. The following action codes are known:

Code	Argument	Description
0	single zero byte	Erase last picure
1	single zero byte	Erase all picures (but not animations)
2	single zero byte	Take picture
4	single zero byte	Finish session immediately
5	single zero byte	Take preview snapshot (retreivable as frame zero)
6	single byte	Calibration / testing. Arg value: 1 Calibrate autofocus 3 Test zoom/exposure 4-6 Store 0 in Reg 32 9 Load LCD Brightness (0-31) from Reg 32 10 Load LCD size (25 for Nikon Coolpix 950) from Reg 32 11 LCD Saturation (0-32) from Reg 32 13 LCD Red-Green (0-32) from Reg 32 14 LCD Blue (0-32) from Reg 32 15 Store -1 in Reg 32 16 Multi shot (locks up if lcd is on) 17 Take picture 18 Store -1 in Reg 32 20-23 locks up if lcd is on 24-255 Store -1 in Reg 32
7	single zero byte	Erase current frame *

8	single byte	Switch LCD mode. Arg value: 1 - Off 2 - Record 3 - Play (show current frame fullscreen) 4 - preview thumbnails (?) 5 - Thumbnail view (smaller?) 6 - Thumbnail view (larger?) 7 - Next 8 - Previous
9	single byte	Set protection state of current frame to the value of parameter (binary 0 or 1)*
11	single zero byte	Store freshly uploaded image into NVRAM (see appendix A)
12	single byte	LCD test. Arg value: 0 - white 1 - gray 2 - black 3 - red 4 - green 5 - blue 6 - test pattern

*** Note:** actions 7 and 9 only useful after setting register 0x04.

Appendix A

```
Date: Sun, 14 Jul 2002 01:28:39 +0200 (CEST)
From: =?iso-8859-1?Q?Peter_=C5strand?= <astrand(at)lysator.liu.se>
To: allyn(at)fratkin.com, <wolfgang(at)charlotte.wsrcc.com>, <crosser(at)average.org
Subject: Upload on Olympus C-860L


FYI.

Tonight, I've been struggling with uploading arbitrary pictures to my
Olympus C-860L. I've finally found out that for the camera to accept the
picture, to two conditions must be met:

1) The subsampling must be 2x1, 1x1, 1x1
2) The EXIF info must be just like the pictures the camera itself
produces.

So, I've made a small script to fix this. Feel free to include it in FAQs
and/or photopc dists.

/more/data/pics/olympus-reference-pic.jpg is just some picture taken with
the camera.

photopc-upload-all:

#!/bin/sh

TMPFILE='mktemp /tmp/photopc-upload.XXXXXX' || exit 1
```

```
for file in $@; do
    echo Converting $file...
    djpeg $file | cjpeg -sample 2x1 > $TMPFILE
    jhead -te /more/data/pics/olympus-reference-pic.jpg $TMPFILE
    echo Uploading $file...
    photopc upload $TMPFILE
    sleep 2;
done

rm -f $TMPFILE


--
/Peter Åstrand <astrand(at)lysator.liu.se>
```

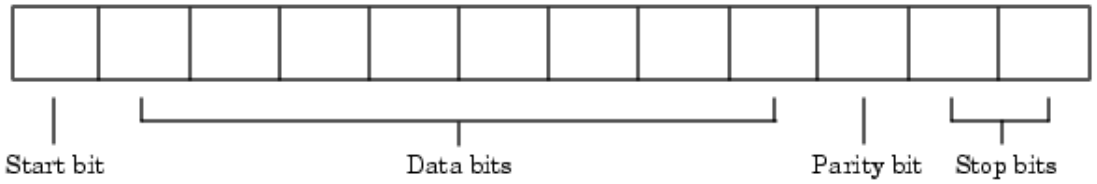
Appendix B

Some Nikon models support an extension to the protocol described above, specifically designed for remote control units. This protocol allows to control zoom, emulate half-depress of the shutter release button, bulb operation and possibly more. <*vladimir.vyskocil(at)wanadoo.fr*> compiled a partial description of this protocol, available here.

Please mail your corrections/additions to <*crosser at average dot org*>
See http://photopc.sourceforge.net/ for possible updates.

Serial Data Format

The serial data format includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well. The diagram below illustrates the serial data format.



The format for serial port data is often expressed using the following notation

- number of data bits - parity type - number of stop bits

For example, 8-N-1 is interpreted as eight data bits, no parity bit, and one stop bit, while 7-E-2 is interpreted as seven data bits, even parity, and two stop bits.

The data bits are often referred to as a *character* because these bits usually represent an ASCII character. The remaining bits are called *framing bits* because they frame the data bits.

Bytes Versus Values

The collection of bits that comprise the serial data format is called a *byte*. At first, this term might seem inaccurate because a byte is 8 bits and the serial data format can range between 7 bits and 12 bits. However, when serial data is stored on your computer, the framing bits are stripped away, and only the data bits are retained. Moreover, eight data bits are always used regardless of the number of data bits specified for transmission, with the unused bits assigned a value of 0.

When reading or writing data, you might need to specify a *value*, which can consist of one or more bytes. For example, if you read one value from a device using the `int32` format, then that value consists of four bytes. For more information about reading and writing values, refer to Writing and Reading Data.

Synchronous and Asynchronous Communication

The RS-232 standard supports two types of communication protocols: synchronous and asynchronous.

Using the synchronous protocol, all transmitted bits are synchronized to a common clock signal. The two devices initially synchronize themselves to each other, and then continually send characters to stay synchronized. Even when actual data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each bit that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required.

Using the asynchronous protocol, each device uses its own internal clock resulting in bytes that are transferred at arbitrary times. So, instead of using time as a way to synchronize the bits, the data format is used.

In particular, the data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word. The requirement to send these additional bits causes asynchronous communications to be slightly slower than synchronous. However, it has the advantage that the processor does not have to deal with the additional idle characters. Most serial ports operate asynchronously.

> **Note** When used in this guide, the terms "synchronous" and "asynchronous" refer to whether read or write operations block access to the MATLAB command line. Refer to Controlling Access to the MATLAB Command Line for more information.

How Are the Bits Transmitted?

By definition, serial data is transmitted one bit at a time. The order in which the bits are transmitted is given below:

1. The start bit is transmitted with a value of 0.
2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB).
3. The parity bit (if defined) is transmitted.
4. One or two stop bits are transmitted, each with a value of 1.

The number of bits transferred per second is given by the *baud rate*. The transferred bits include the start bit, the data bits, the parity bit (if defined), and the stop bits.

Start and Stop Bits

As described in Synchronous and Asynchronous Communication, most serial ports operate asynchronously. This means that the transmitted byte must be identified by start and stop bits. The start bit indicates when the data byte is about to begin and the stop bit(s) indicates when the data byte has been transferred. The process of identifying bytes with the serial data format follows these steps:

1. When a serial port pin is idle (not transmitting data), then it is in an "on" state.
2. When data is about to be transmitted, the serial port pin switches to an "off" state due to the start bit.
3. The serial port pin switches back to an "on" state due to the stop bit(s). This indicates the end of the byte.

Data Bits

The data bits transferred through a serial port might represent device commands, sensor readings, error messages, and so on. The data can be transferred as either binary data or ASCII data.

Most serial ports use between five and eight data bits. Binary data is typically transmitted as eight bits. Text-based data is transmitted as either seven bits or eight bits. If the data is based on the ASCII character set, then a minimum of seven bits is required because there are 2^7 or 128 distinct characters. If an eighth bit is used, it must have a value of 0. If the data is based on the extended ASCII character set, then eight bits must be used because there are 2^8 or 256 distinct characters.

The Parity Bit

The parity bit provides simple error (parity) checking for the transmitted data. The types of parity checking are given below.

Table 9-2: Parity Types

Parity Type	Description
Even	The data bits plus the parity bit result in an even number of 1's.
Mark	The parity bit is always 1.
Odd	The data bits plus the parity bit result in an odd number of 1's.
Space	The parity bit is always 0.

Mark and space parity checking are seldom used because they offer minimal error detection. You might choose to not use parity checking at all.

The parity checking process follows these steps:

1. The transmitting device sets the parity bit to 0 or to 1 depending on the data bit values and the type of parity checking selected.
2. The receiving device checks if the parity bit is consistent with the transmitted data. If it is, then the data bits are accepted. If it is not, then an error is returned.

> **Note** Parity checking can detect only 1-bit errors. Multiple-bit errors can appear as valid data.

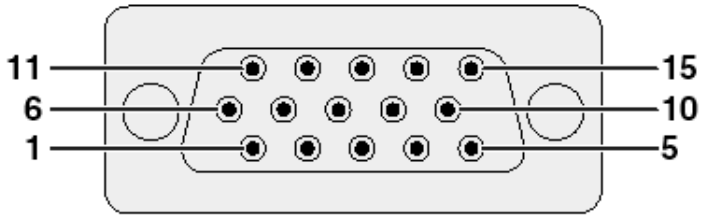
For example, suppose the data bits 01110001 are transmitted to your computer. If even parity is selected, then the parity bit is set to 0 by the transmitting device to produce an even number of 1's. If odd parity is selected, then the parity bit is set to 1 by the transmitting device to produce an odd number of 1's.

Serial Port Signals and Pin Assignments Finding Serial Port Information for Your Platform

Connectors

RGB Signal Input Port:

15-pin Mini D-sub female connector



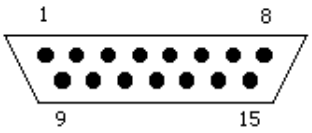
RGB Input

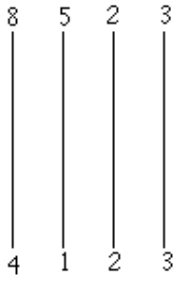
Analog

1. Video input (red)
2. Video input (green/sync on green)
3. Video input (blue)
4. Not connected
5. Composite sync
6. Earth (red)
7. Earth (green/sync on green)
8. Earth (blue)
9. Not connected
10. GND
11. GND
12. Bi-directional data
13. Horizontal sync signal
14. Vertical sync signal
15. Data clock

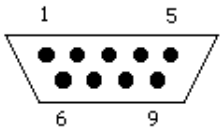
Computer - Pioneer DVD9300 player cable

DVD player DB-15 Male





Computer DB-9 female

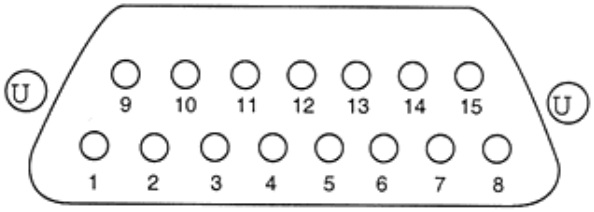


Pioneer DVD 7300 (7400)

2.1 Interface Connector

A computer may be connected to the DVD-V7400 using a 15-pin D-Sub connector (e.g., a JAE DALC-J15SAF connector with suitable plug such as the JAE DA-15PF-N) to the RS-232C serial port or to the parallel port.

The pins are identified below:

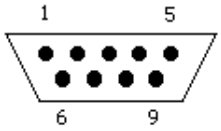


2.2 Serial Interface Pin Specification

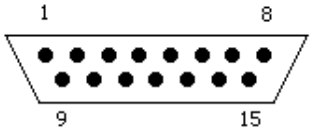
Pin #	Terminal	Input/Output	Function
1	GND	--	ground
2	TxD	Output	send data
3	RxD	Input	receive data
4	DTR	Output	enable data receiving
5	POWER	Output	external power control
6	SW1	Input	
7	SW2	Input	
8	SW3	Input	
9	SW4	Input	
10	SW5	Input	
11	SW6	Input	
12	SW7	Input	
13	SW8	Input	
14	DLTST	Input	used only for servicing the unit – do not connect
15	V +8V	Output	used only for servicing the unit – do not connect

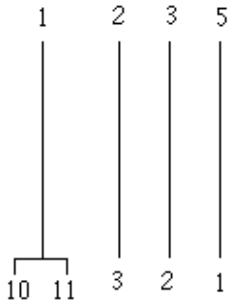
Serial controller - DVD player cable

Computer DB-9 male



DVD player DB-15 Male

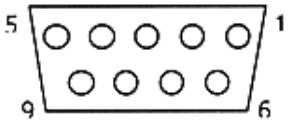




HT200/HT250

RS-232C Control Port:

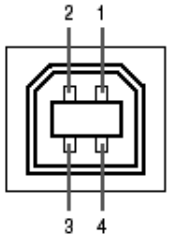
D-SUB 9-pin
(female)



Pin No	Signal	Definition
1	N/A	Not used
2	TD	Transmit data
3	RD	Receive data
4	N/A	Not used
5	GND	Ground
6	N/A	Not used
7	N/A	Not used
8	N/A	Not used
9	N/A	Not used

Parameter	Value
Transfer Rate	19200 b.p.s.
Data Length	8 bits
Parity Bit	None
Stop Bit	1 bit
Flow Control	None

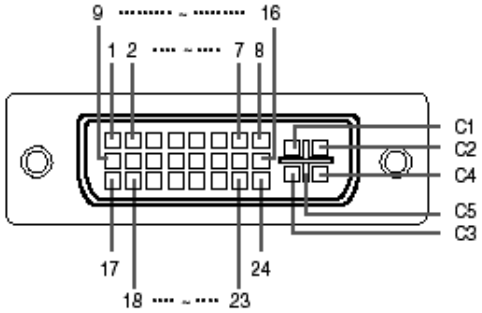
4-pin USB connector



- **USB connector: 4 pin B-type USB connector**

Pin no.	Signal	Name
1	VCC	USB power
2	USB–	USB data–
3	USB+	USB data+
4	SG	Signal Ground

DVI Digital / Analog INPUT 1 port : 29 pin connector



- **DVI Digital INPUT**

Pin No.	Signal	Pin No.	Signal
1	T.M.D.S data 2–	16	Hot plug detection
2	T.M.D.S data 2+	17	T.M.D.S data 0–
3	T.M.D.S data 2 shield	18	T.M.D.S data 0+
4	Not connected	19	T.M.D.S data 0 shield
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	T.M.D.S clock shield
8	Not connected	23	T.M.D.S clock+
9	T.M.D.S data 1–	24	T.M.D.S clock–
10	T.M.D.S data 1+	C1	Not connected
11	T.M.D.S data 1 shield	C2	Not connected
12	Not connected	C3	Not connected
13	Not connected	C4	Not connected
14	+5V power	C5	Ground
15	Ground		

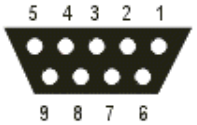
- **DVI Analog RGB Input**

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Hot plug detection
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	Not connected
8	Vertical sync	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Red
11	Not connected	C2	Analog input Green
12	Not connected	C3	Analog input Blue
13	Not connected	C4	Horizontal sync
14	+5V power	C5	Ground
15	Ground		

- **DVI Analog Component Input**

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Not connected
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	Not connected	21	Not connected
7	Not connected	22	Not connected
8	Not connected	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Pr/Cr
11	Not connected	C2	Analog input Y
12	Not connected	C3	Analog input Pb/Cb
13	Not connected	C4	Not connected
14	Not connected	C5	Ground
15	Ground		

Serial mouse



Pin	Description	
1	DCD	Data carried detect
2	RD	Receive data
3	TD	Transmit data
4	DTR	Data terminal ready
5	SG	Signal ground
6	DSR	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	Ring	

Image Dimensions in Common Usage

Collated by Paul Bourke
May 2000

Dimensions	Ratio	Comments
8x8	1	
16x16	1	Macintosh cursor size Supported by Windows icon format
32x32	1	Macintosh icon size Supported by Windows icon format
64x64	1	Supported by Windows icon format
88x31	2.8387	WWW micro banner
128x96	1 1/3	
160x120	1 1/3	NTSC 13" 1/16th CuSeeMe small image size Considered a "small" QuickTime movie
160x144	1.1111	
176x144	1.2222	
180x132	1.3636	
180x135	1 1/3	
192x144	1 1/3	PAL 1/16
234x60	3.9	WWW small banner
256x192	1 1/3	10"/12" 1/4
320x200	1.6	Called CGA, IBM PS/2
320x240	1 1/3	NTSC 13" 1/4 CuSeeMe image size Considered a "large" QuickTime movie
320x288	1.1111	
320x400	0.8	Amigo

352x288	1.2222	PAL Video CD
352x240	1.46666	NTSC Video CD
384x256	1.5	Photo CD
384x288	1 1/3	PAL 1/4
392x72	5.444444	WWW banner
400x300	1 1/3	
460x55	8.36363636	WWW banner
468x32	14.625	WWW banner
468x60	7.8	WWW banner
512x342	1.497	Original Macintosh screen
512x384	1 1/3	
544x372	1.4623	WebTV image size
640x350	1.82857	Called EGA, IBM
640x480	1 1/3	NTSC full Called PGA, IBM VGA
640x576	1.1111	
704x576	1.22222	Full image size from ASUS video capture with TNT2
720x350	2.05714286	Called MDA, IBM / VESA
720x400	1.8	Called MCGA, IBM/VESA
720x480	1.5	Format 480p. NTSC video format as used by DPS PVR video hardware. NTSC DV SDTV
720x483	1.49068323	SDTV
720x484	1.48760331	Alternative Media-100 NTSC, eg: Media-100
720x486	40/27	CCIR 601 NTSC
720x540	1 1/3	CCIR 601 NTSC Sq
720x576	1.25	CCIR 601 DV PAL and DV SECAM

729x348	2.094828	Hercules graphics
768x576 Subtract 75 from left/right and 55 top/bottom for the PAL "safe" area	1 1/3	CCIR 601 PAL full
800x600	1 1/3	Called SVGA resolution. Used in the first generation of LCD projectors (their native resolution).
832x624	1 1/3	Macintosh
856x480	1.78333333	
896x600	1.49333333	
960x720	1 1/3	Non standard digital TV format supported by some suppliers.
1024x768	1 1/3	XGA. Native resolution of many LCD projectors. Used by VisionStation and VisionStation 3.
1080x720	1.5	HDTV
1152x768	1.5	
1152x864	1 1/3	VESA
1152x870	1.3241	Macintosh
1152x900	1.28	Sun / SGI
1280x720	16/9	Format 720p. HDTV WXGA
1280x800	1.6	
1280x854	1.498829	Mac G4 15" laptop
1280x960	1 1/3	SXVGA
1280x992	1.29	Optimised on the PowerStorm 350 OpenGL card/drivers from Compaq
1280x1024	1.25	SXGA
1360x766	1.77545692	

1365x768	16:9 (nearly, 1.77734375)	NEC 61" plasma
1365x1024	1 1/3	VisionStation 3 with upgrade and VisionStation 5.
1400x1050	1.3671875	DELL Laptop
1440x900	1.6	Apple 17" G4 laptop
1520x856	1.77570093	
1600x900	16:9	
1600x1024	1.5625	
1600x1200	1 1/3	VESA UXGA
1792x1120	1.6	
1792x1344	1 1/3	
1824x1128	1.61702128	
1824x1368	1 1/3	
1856x1392	1 1/3	
1920x1080	16/9	1080i format. HDTV, known as 1K
1920x1200	1.6	
1920x1440	1 1/3	
2000x1280	1.5625	QXGA
2048x1152	16:9	
2048x1536	1 1/3	Feature film, known as 2K Also used by DOME display controllers
2048x2048	1	Tiger high resolution display
2500x1340	1.86567	
3072x2252	1.3641	Sometimes used for IMAX (?)
3600x2613	1.37772675	Sometimes used for IMAX (?)
4096x3072	1 1/3	Image size for IMAX 3D rendering, known as 4K
4096x3840	1.0666666	NCSA tiled wall (2001)

Serial Mouse Data Formats

The Microsoft Serial Mouse format is the defacto standard for serial mice. The Microsoft mouse format allows for only two buttons. Three button mice working in Microsoft mode ignore the middle button.

The data packets are sent at 1200 baud with 1 stop bit and no parity. Each packet consists of 3 bytes. It is sent to the computer every time the mouse changes state (ie. the mouse is moved or the buttons are pressed/released).

	D6	D5	D4	D3	D2	D1	D0
1st byte	1	LB	RB	Y7	Y6	X7	X6
2nd byte	0	X5	X4	X3	X2	X1	X0
3rd byte	0	Y5	Y4	Y3	Y2	Y1	Y0

```
LB is the state of the left  button, 1 = pressed, 0 = released.
RB is the state of the right button, 1 = pressed, 0 = released
X0-7 is movement of the mouse in the X direction since the
     last packet.  Positive movement is toward the right.
Y0-7 is movement of the mouse in the Y direction since the
     last packet.  Positive movement is back, toward the user.
```

The mouse driver software collects the X and Y movement bits from the different bytes in the packet. All moves are sent as two’s complement binary numbers.

Although the Microsoft format only requires 7 data bits per byte, most mice actually send 8-bit data with the most significant bit set to 1. Since the most-significant-bit (D7) is last in the serial data stream, this is the same as sending two stop bits instead of one. The Joymouse sends data packets as shown below.

	D7	D6	D5	D4	D3	D2	D1	D0
1st byte	1	1	LB	RB	Y7	Y6	X7	X6
2nd byte	1	0	X5	X4	X3	X2	X1	X0
3rd byte	1	0	Y5	Y4	Y3	Y2	Y1	Y0

[Back to Data & Documentation] [Home]

Next: Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)

RS-232 Serial Protocol

The RS-232 serial communication protocol is a standard protocol used in asynchronous serial communication. It is the primary protocol used over modem lines. It is the protocol used by the MicroStamp11 when it communicates with a host PC.

Figure 23 shows the relationship between the various components in a serial ink. These components are the *UART*, the serial channel, and the interface logic. An interface chip known as the **universal asynchronous receiver/transmitter** or **UART** is used to implement serial data transmission. The UART sits between the host computer and the *serial channel*. The serial channel is the collection of wires over which the bits are transmitted. The output from the UART is a standard TTL/CMOS logic level of 0 or 5 volts. In order to improve bandwidth, remove noise, and increase range, this TTL logical level is converted to an RS-232 logic level of -12 or $+12$ volts before being sent out on the serial channel. This conversion is done by the interface logic shown in figure 23. In your system the interface logic is implemented by the comm stamp.

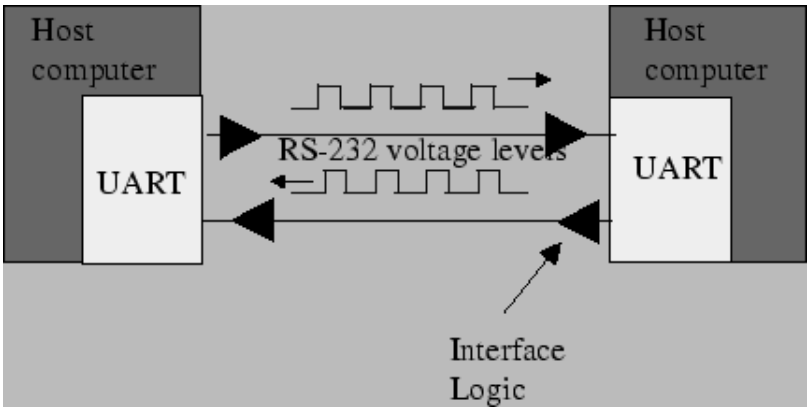


Figure 23: Asynchronous (RS-232) serial link

A **frame** is a complete and nondivisible packet of bits. A frame includes both *information* (e.g., data and characters) and *overhead* (e.g., start bit, error checking and stop bits). In asynchronous serial protocols such as RS-232, the frame consists of one start bit, seven or eight data bits, parity bits, and stop bits. A timing diagram for an RS-232 frame consisting of one start bit, 7 data bits, one parity bits and two stop bits is shown below in figure 24. Note that the exact structure of the frame must be agreed upon by both transmitter and receiver before the comm-link must be opened.

Figure 24: RS-232 Frame (1 start bit, 7 data bits, 1 parity bits,

and 2 stop bits)

Most of the bits in a frame are self-explanatory. The start bit is used to signal the beginning of a frame and the stop bit is used to signal the end of a frame. The only bit that probably needs a bit of explanation is the *parity* bit. Parity is used to detect transmission errors. For *even parity* checking, the number of 1's in the data plus the parity bit must equal an even number. For *odd parity*, this sum must be an odd number. Parity bits are used to detect errors in transmitted data. Before sending out a frame, the transmitter sets the parity bit so that the frame has either even or odd parity. The receiver and transmitter have already agreed upon which type of parity check (even or odd) is being used. When the frame is received, then the receiver checks the parity of the received frame. If the parity is wrong, then the receiver knows an error occurred in transmission and the receiver can request that the transmitter re-send the frame.

In cases where the probability of error is extremely small, then it is customary to ignore the parity bit. For communication between the MicroStamp11 and the host computer, this is usually the case and so we ignore the parity bit.

The *bit time* is the basic unit of time used in serial communication. It is the time between each bit. The transmitter outputs a bit, waits one bit time and then outputs the next bit. The start bit is used to synchronize the transmitter and receiver. After the receiver senses the true-false transition in the start bit, it waits one half bit time and then starts reading the serial line once every bit time after that. The *baud rate* is the total number of bits (information, overhead, and idle) per time that is transmitted over the serial link. So we can compute the baud rate as the reciprocal of the bit time.

Next: Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)
Bill Goodwine 2002-09-29


Next: LCD display for the **Up:** Serial Communication **Previous:** RS-232 Serial Protocol

Motorola 68HC11 SCI Interface

The Motorola 68HC11 supports one SCI. We'll discuss both transmitting and receiving ends of the SCI. The programmer controls the operation of the SCI interface through a set of hardware registers that are memory mapped into the processor's address space. There are 5 control registers shown below in figure 25. This figure also shows the logical names for individual bits in the registers. The `BAUD` register is used to set the serial link's baud rate. There are two control registers `SCCR1` and `SCCR2` that specify how the SCI should work. There is a status register, `SCSR` that the programmer can use to check whether the transmission/reception of a frame has been completed. Finally there is the data register `SCDR` that holds the transmitted or received information bits.

BAUD

TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0

SCCR1

R8	T8	0	M	WAK	0	0	0

SCCR2

TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

SCSR

TDRE	TC	RDRF	IDLE	OR	NF	FE	0

SCDR

R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

Figure 25: 68HC11 SCI Registers

From the number of control bits in the `SCCR1` and `SCCR2` registers you can see that the programmer has quite a bit of control over the SCI interface. Most of the situations we'll be using, however, have standard set-ups so we won't need to discuss the control register bits in detail. Instead, we'll provide standard functions that encapsulate the user's interface to SCI devices such as a personal computers and simply discuss how these functions work. The proper setup of the SCI subsystem is usually done in your program's `init()` initialization routine.

To understand how the SCI subsystem works, let's examine figure 26. Figure 26 shows how the programmer interacts with the SCI transmit and receive buffers. In order to transmit, the programmer first loads the 8-bit data register `SCDR` with the data to be sent. The SCI module automatically fills in the start bit, stop bit, and the extra `T8` bit from control register `SCCR1`. The `T8` bit can be used as a parity bit.

Once the SCDR register is loaded, the subsystem loads this data into the SCI module's transmit buffer. The SCI transmit buffer then holds a single frame and this frame is then clocked out of the transmit register one bit at a time at the rate specified in the BAUD register. Once all of the bits have been clocked out of the transmit buffer, the SCI module sets the TDRE (transmit data register empty) bit in the SCI's status register SCSR. This single bit can then be used to check whether or not the frame has been successfully transmitted.

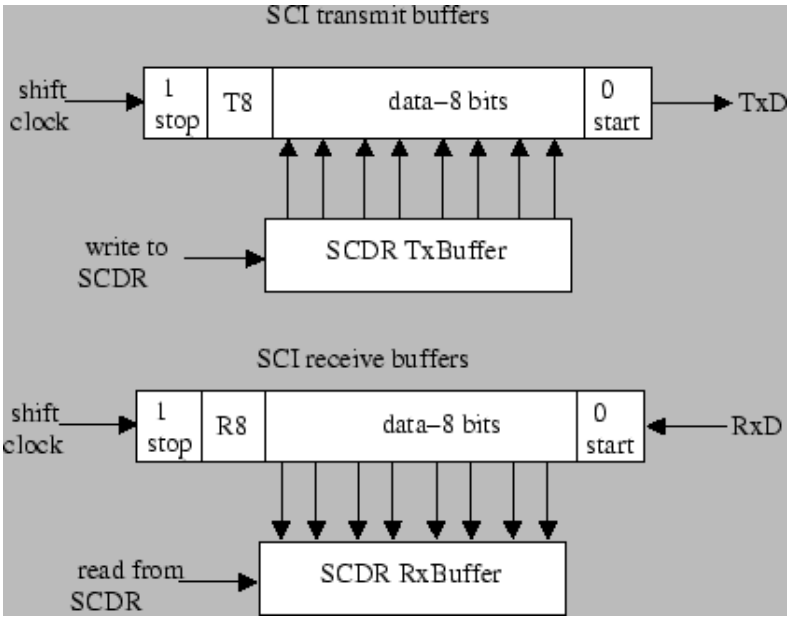


Figure 26: SCI transmit and receive buffers

A similar set of steps can be used to check and see if the receive data register has been filled. Once initialized, the receive data component of the SCI subsystem will wait for the true-to-false transition on the input line signalling a start bit. After the start bit has been detected, the receive subsystem will shift in 10 or 11 bits into the receive data register. The start and stop bits are removed and 8 bits of data are loaded into the SCI data register SCDR. The ninth parity bit R8 is put in the SCCR1 control register. When the receive data register is full, then the SCI subsystem sets the RDRF (receive data register full) flag in the status register SCSR. The programmer can then check this status flag to see if a full frame has been received.

The following code segment from an init() function can be used to initialize the SCI module to transmit and receive at 38 kbaud. This setup was used in our earlier kernel.c functions to send characters back and forth between the MicroStamp11 and the PC.

```
void init(void){
  asm(" sei");
    CONFIG = 0x04;
    BAUD=BAUD38K;
    SCCR1 = 0x00;
    SCCR2 = 0x0C;
  asm(" cli");
}
```

The instructions in this function do the following. The first instruction CONFIG = 0x04 turns off the

micro-controller's watchdog timer. The second instruction `BAUD=BAUD38K` sets the SCI subsystem's baud rate to 38 kilo-baud. The variable `BAUD38K` is a logical name whose actual value will be found in `hc11.h`. The next two lines set up the SCI subsystem's parameters. By zeroing `SCCR1`, we are ignoring the parity bit and creating a 10-bit frame. By setting `SCCR2=0x0C`, we've disabled all transmit and receive interrupts and we've enabled the transmit and receive modules in the SCI subsystem.

Note that the SCI module has hardware interrupts associated with the hardware events

- Transmission Complete (TC): which is set when the transmit shift register is empty.
- Transmit Data Register Empty (TDRE): which is set when the transmit data register is empty.
- Receive Data Register Full (RDRF): which is set when the receive data register is full.
- Idle line (ILIE): which is when the receive module detects an idle line.

These hardware interrupts can be used to do parity-bit processing on a transmitted or received frame. In our particular examples, however, we assume no parity checking so these interrupts have been disabled.

As specific examples of how the SCI interface can be used, we consider the two function `InChar()` and `OutChar()`. These functions are used to receive and transmit, respectively, a single byte (frame) of data.

```
void OutChar(char data){
  while((SCSR & TDRE) == 0);
  SCDR=data;
}

void InChar(void){
  while((SCSR & RDRF) == 0);
  return(SCDR);
}
```

The first function, `OutChar()` transmits a single byte. The function simply waits in a loop until the `TDRE` bit (transmit data register empty) is set. Once this is done, we know that any previously loaded bytes have been successfully shifted out of the transmit data register. The function then reloads the SCI's data register `SCDR` with the new data. The other function `InChar()` receives a single byte. The function waits in a while loop until the `RDRF` bit (receive data register full) is set, thereby indicating that 8-bits of have been shifted into the receive data register. Once this is done, the function returns a pointer to the `SCDR` data register.

Next: LCD display for the **Up:** Serial Communication **Previous:** RS-232 Serial Protocol
Bill Goodwine 2002-09-29

Serial Data Format

The serial data that we are interested in sending to and from the terminal is byte-wide ASCII data. ASCII is a standard code for sending alphanumeric data and is actually only 7-bits wide. The 8th bit is often used to indicate the parity of the 7-bit data word and used for error detection. For our circuit, the high-order bit will always be 0, but you should always send it anyway. So, each packet that is sent will consist of 8 bits, 7-bits of ASCII and one 0 in the high-order bit. A table of the ASCII code is shown in Figure 1.

$B_3B_2B_1B_0$	$B_6B_5B_4$ 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	—
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Figure 1: ASCII Character Codes

Once we agree to send ASCII, we should also agree on how that data should be send as a serial data stream. The protocol we will use defines that the bits in the byte are passed least-significant bit first in a serial stream. So, send the bits one at a time, starting with the least-significant. How long each bit is asserted in this serial stream depends on how fast your baud rate is. At 9600 baud, for example, each bit will be asserted for 1/9600 of a second, or about 104 μ sec. Now the problem is how to decide when to look at the data wire in order to see the bit. The DCE and DTE will not be synchronized to a common clock, so in order to decide when to look at the data line to see new data being passed, they must synchronize with each new byte that is being passed. This is why the UART is ‘‘asynchronous’’ in operation. The data is being passed at a known frequency, but the starting time of each new byte is unknown. So, the receiving circuit must resynchronize at the start of each new byte.

In practice this is quite easy. You only need some sort of protocol that tells you when to expect new data. For our system (and most asynchronous serial protocols in general) the data line must be held in a 1 state (+5v in our case) until a byte is ready to be passed. When a byte is to be sent, the data line drops to 0 (gnd in our case) for one bit time to signal that a byte will follow. This is the *start bit* and its purpose is to wake up the receiver and alert it to the byte that is about to be sent. The 8 data bits then follow, least significant bit first, each asserted for one bit time (which depends on the baud rate). Finally, one or two *stop bits* are sent to indicate the end of the byte. Stop bits are 1-bits that are asserted for one bit time each. In our system the receiver will assume that only one stop bit is sent, and the sender will send two stop bits. This is the most general and safest solution. For example, if the receiver expects a single stop bit and two are sent, nothing bad happens except that some extra time elapses between that byte and the next due to the extra stop bit. On the other hand, if the sender sends only a single stop bit but the external receiver expects two, a mistake might be made. So, it's safer to expect only a single stop bit, but always send two (note that in practice, most terminals and other communication devices have settings to control how many stop bits are sent or expected.) A picture of this data protocol is shown in Figure 2.

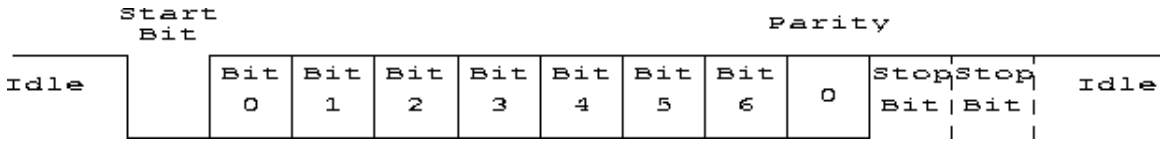


Figure 2: Data Byte Transmission Format

Erik Brunvand
Tue Apr 11 15:50:32 MDT 2000

ANALOG DEVICES

AN-397
APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

Electrically Induced Damage to Standard Linear Integrated Circuits: The Most Common Causes and the Associated Fixes to Prevent Reoccurrence

by Niall Lyne

INTRODUCTION

The sensitivity of electronic components to transient electrical overstress events is a well-known problem, exacerbated by the continuing evolution of integrated circuits. Smaller geometries, increased circuit densities, and the limited area allotted to on-chip protection all tend to increase this sensitivity. In an effort to minimize costs in each particular segment of system implementation, the burden of transient protection is often shifted to other, less efficient means.

Techniques for protection from "zapping" depend on the stage of manufacture. During the manufacturing of integrated circuits and assembly of electronic equipment, protection is achieved through the use of well-known measures such as static dissipative table tops, wrist straps, ionized air blowers, antistatic shipping tubes, etc. These methods will be discussed only briefly here in relation to Electrostatic Discharge (ESD) protection. Likewise this application note is not addressed to precautionary measures employed during shipping, installation, or repair of equipment. Rather, the main thrust will be limited to protection aspects called upon during printed circuit board assembly, normal operation of the equipment (often by operating personnel who are untrained in preventative measures), and in service conditions where the transient environment may not be well characterized.

The transient environment varies widely. There are substantial differences among those experienced by, say, automotive systems, airborne or shipborne equipment, space systems, industrial equipment or consumer products. All types of electronic components can be destroyed or degraded. [1] Even capacitors, relays, connectors, printed circuit boards, etc., are susceptible, although their threshold levels are much higher than integrated circuits. Microwave diodes and transistors are among the most sensitive components. However, this application note will be restricted to standard linear integrated circuits because of their wide usage, and to limit the scope of coverage.

This application note will first review the nature of the threat to integrated circuits in an operating environment, and then briefly discuss overall device protection from the following: (1) *ESD* events caused by human handling, automatic board insertion equipment, etc., (2) *LATCH-UP* generated from power-up/down sequencing errors, floating ground(s) due to a loose edge connectors, etc., and finally, (3) *HIGH VOLTAGE TRANSIENTS* generated from a power supply, a defective circuit board, during circuit board troubleshooting, etc.

Electrostatic Discharge

Electrostatic discharge is a single, fast, high current transfer of electrostatic charge that results from:

- Direct contact transfer between two objects at different potentials, *or*
- A high electrostatic field between two objects when they are in close proximity.

The prime sources of static electricity are mostly insulators and are typically synthetic materials, e.g., vinyl or plastic work surfaces, insulated shoes, finished wood chairs, Scotch tape, bubble pack, soldering irons with ungrounded tips, etc. Voltage levels generated by these sources can be extremely high since their charge is not readily distributed over their surfaces or conducted to other objects.

The generation of static electricity caused by rubbing two substances together is called the triboelectric effect. Examples of sources of triboelectric electrostatic charge generation in a high RH (≈60%) environment include:

- Walking across a carpet ⇒ 1000 V–1500 V generated.
- Walking across a vinyl floor ⇒ 150 V–250 V generated.
- Handling material protected by clear plastic covers ⇒ 400 V–600 V generated.
- Handling polyethylene bags ⇒ 1000 V–1200 V generated.
- Pouring polyurethane foam into a box ⇒ 1200 V–1500 V generated.

- ICs sliding down an open antistatic shipping tube ⇒ 25 V–250 V generated.

Note: For low RH (<30%) environments, generated voltages can be >10 × those listed above.

ESD Models

To evaluate the susceptibility of devices to simulated stress environments a host of test waveforms have been developed. The three most prominent of these waveforms currently in general use for simulating ESD events in semiconductor or discrete devices are: The Human Body Model (HBM), the Machine Model (MM), and the Charged Device Model (CDM). The test circuits and current waveform characteristics for these three models are shown in Figures 1 to 3. Each of these models represents a fundamentally different ESD event. Consequently, correlation between the test results for these models is minimal.

Human Body Model:[2]

Simulates the discharge event that occurs when a person charged to either a positive or negative potential touches an IC at a different potential.

RLC = 1.5 kΩ, ~0 nH, 100 pF.

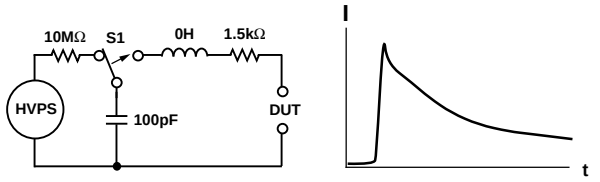


Figure 1. Human Body Model

Machine Model:

Japanese model based on a worst-case HBM.

RLC = 0 Ω, 500 nH, 200 pF.

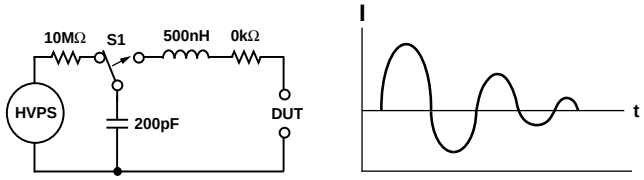


Figure 2. Machine Model

Charged Device Model:

Simulates the discharge that occurs when a pin on an IC charged to either a positive or negative potential contacts a conductive surface at a different (usually ground) potential.

RLC = 0 Ω, ~0 nH, 1 pF–20 pF.

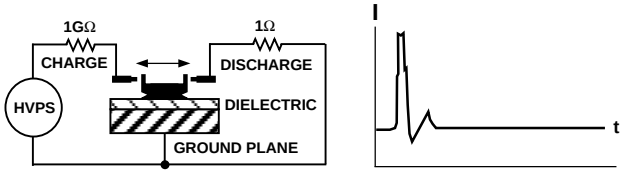


Figure 3. Charged Device Model

Comparison of HBM, MM, and CDM Waveforms

Figure 4 shows 400 V HBM, MM, and CDM discharge waveforms on the same current vs. time scale. These waveforms are of great use in predicting what failure mechanism may result on a particular device type due to ESD events simulated by one of these three models.

The rise time for the HBM waveform is <10 ns (typically 6 ns–9 ns), and this waveform decays exponentially towards 0 V with a fall time of >150 ns. MIL-STD-883[3] Method 3015 *Electrostatic Discharge Sensitivity Classification* requires a rise time of <10 ns and a delay time of 150 ± 20 ns (Method 3015 defines delay time as the time for the waveform to drop from 90% of the peak current to 36.8% of the peak current). The peak current for the HBM waveform is ≈400 V/1500 Ω or 0.267A. Although this peak current is much lower than that for 400 V CDM and MM events, the relatively long duration of the total HBM event results in a discharge of relatively high energy.

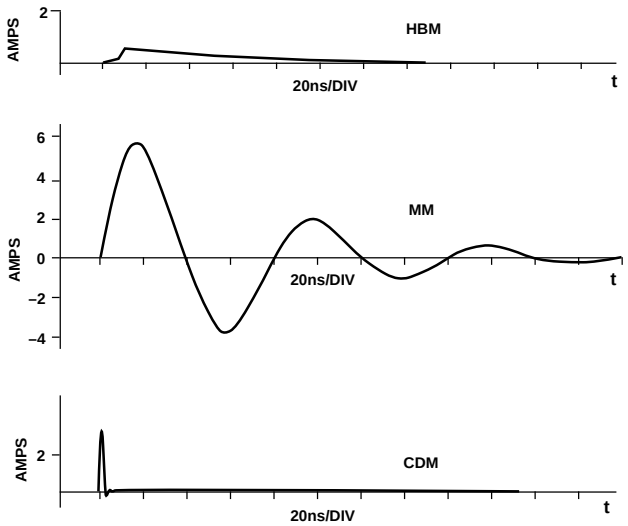


Figure 4. Relative Comparison of 400 V HBM, MM, and CDM Discharges

The MM waveform consists of both positive-going and negative-going sinusoidal peaks with peak magnitudes that decay exponentially. The initial MM peak has a rise time of ≈14 ns, i.e., only slightly greater than that of the single HBM peak. The total duration of the MM waveform is comparable to that for the HBM waveform. However, the peak current for the first peak of the 400 V MM event is ≈5.8 A, which is the highest of the three models. The next four peaks, though decreasing in current, still all have magnitudes of >1 A. These multiple high current peaks of substantial duration result in an overall discharge energy that is by far the highest of the three models because there is no current limiting; R = 0 Ω.

The CDM waveform corresponds to the shortest known real-world ESD event. The socketed CDM waveform has a rise time of 400 ps, with the total duration of the CDM event of ≈2 ns. The CDM waveform is essentially unipolar, although some slight ringing occurs at the end of the CDM event that results in some negative-going peaks.

With a 400 V charging voltage, a socketed CDM discharge will have a peak current of 2.1 A. However, the very short duration of the overall CDM event results in an overall discharge of relatively low energy.

Summary of ESD Models

Table I is a reference table that compares the most important characteristics of the three ESD simulation models.

Table I.

Model	HBM	MM	Socketed CDM
Simulate	Human Body	Machine	Charged Device
Origin	US Military Late 1960s	Japan 1976	AT&T 1974
Real World	Yes	Generally No	Yes
RC	1.5 kΩ, 100 pF	0 Ω, 200 pF	1 Ω, 1 pF–20 pF
Rise Time	<10 ns	14 ns*	400ps**
I_{PEAK} at 400 V	0.27 A	5.8 A*	2.1 A**
Package Dependent	No	No	Yes
Leakage Recovery	No	No	Yes
Standard	MIL-STD-883 Method 3015	ESD Assoc. Standard S5.2; EIAJ Standard ED-4701, Method C-111	ESD Assoc. Draft Standard DS5.3

* These values are per ESD Association Standard S5.2. EIAJs standard ED-4701 Method C-111 includes no waveform specifications.
** These values are for the direct charging (socketed) method.

Prevention

When auditing a facility in which ESD protective measures will be taken, the following should be considered: [4]

- There must be a grounded workbench on which to handle static sensitive devices incorporating:
  a) Personal ground strap (wrist strap)
  b) Conductive trays or shunts, etc.
  c) Conductive work surface
  d) Conductive floor or mat
  e) A common ground point
- All steel shelving or cabinets used to store devices must be grounded.
- The relative humidity should be controlled; the desirable range is 40 to 60 percent. Where high relative humidity levels cannot be maintained, the use of ionized air should be used to dissipate electrostatic charges.
- All electrical equipment used in the area must be grounded.
- Prohibit the use of prime static generators, e.g., Scotch tape.
- Follow up with ESD audits at a minimum of three month intervals.
- Training: Keep in mind, the key to an effective ESD control program is "TRAINING." Training should be given to all personnel who come in contact with integrated circuits and should be documented for certification purposes, e.g., ISO 9000 audits.

Determining whether a device failed as a result of ESD or Electrical Overstress (EOS) can be difficult and is often best left to Failure Analysis Engineers. Typically ESD damage is less obvious than that of EOS when electrical analysis and internal visual analysis are performed. In the case of ESD, events of 1 kV or more (depending on the ESD rating of the device) can rupture oxides (inter layer dielectric of the die) and damage junctions in less than 10 ns (see Figure 6). Alternately, EOS conditions leading to 1 to 3 amps of current for a duration of ≥1 ms can cause sufficient self-heating of bond wires to fuse them. Such conditions can occur as a result of latch-up. Lower currents can cause rapid melting of chip metallization and other interconnect layers (see Figure 5).

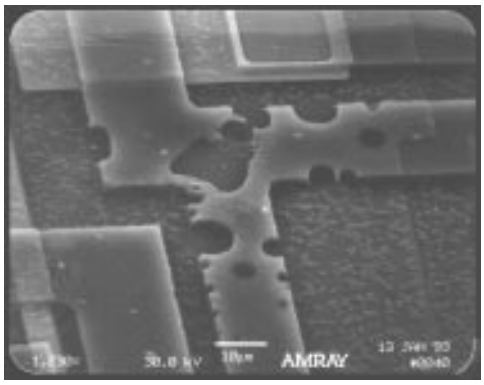


Figure 5. Scanning Electron Microscope View of a Fused Metallization Site, as a Result of Electrical Overstress

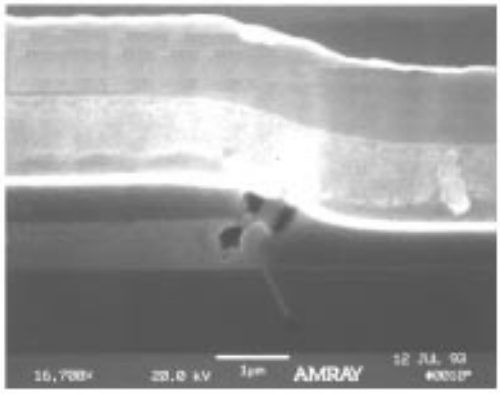


Figure 6. Scanning Electron Microscope Cross-Sectional View of a CDM ESD Site. This subsurface site could not be viewed from the surface with an optical microscope.

A quick analysis can be performed on site to evaluate if a device may have been overstressed or may have been subjected to an ESD event. In order to perform this analysis, to compare the pin-to-pin I/V results of the suspect device to those of a known good device, a curve tracer or similar equipment should be used. A typical set of I/V traces for a short circuit, open circuit or ESD leakage on a digital input pin (with reference to the V_{SS} supply pin) of a 12-bit DAC is shown in Figure 7.

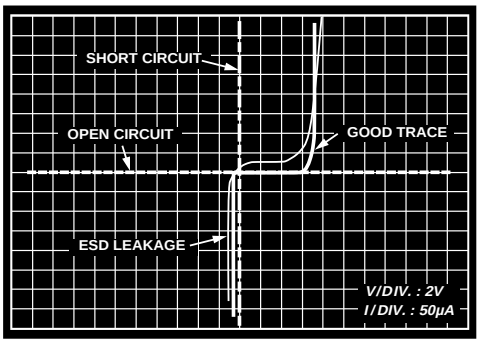


Figure 7. Example of an Unpowered Curve Trace Analysis of a Digital Pin versus a Supply Pin (V_{SS})

LATCH-UP

Latch-up is a potentially destructive situation in which a parasitic active device is triggered, shorting the positive and negative supplies together. If current flow is not limited, electrical overstress will occur. The classic case of latch-up occurs in CMOS output devices, in which the driver transistors and wells form a *pnpn* SCR structure when one of the two parasitic base-emitter junctions is momentarily forward biased during an overvoltage event. The SCR turns on and essentially causes a short between V_{DD} and ground.

Triggering Mechanisms

There are two main triggering mechanisms. *First*, if the input/output (I/O) pin voltage is raised above the positive supply, or lowered below the negative supply, one of the parasitic transistors is turned on. The current returning to the supply through the collector causes a voltage drop across the base-emitter of the second parasitic transistor. In turn, the collector current of the second transistor maintains a forward bias on the base-emitter of the first transistor. If the product of the two transistor gains is greater than unity, the condition may be self-sustaining and can persist even after the external voltage is removed.

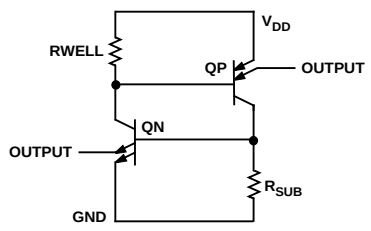


Figure 8. Parasitic SCR. The Diffusions in a CMOS output form a parasitic SCR. The resistors are labeled for an n-well process.

This triggering mechanism can occur if excessive voltage overshoot is present at the I/O pin, or if the signal arrives at the input before the power supplies are applied to the device, or due to electrostatic discharge. This latch-up is usually limited to the devices directly connected to the pin.

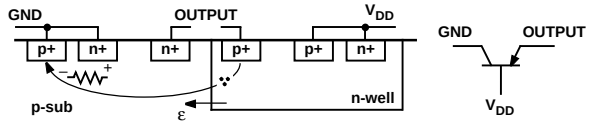


Figure 9a. Output Overvoltage Triggering. Initial hole current flows when the output voltage is raised above V_{DD}. This current causes a voltage rise in the substrate under the NMOS device.

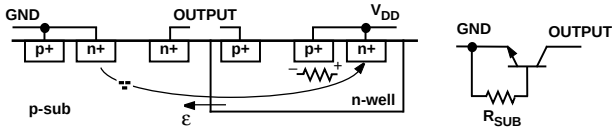


Figure 9b. Current Multiplication. The substrate voltage rise actively biases the second parasitic transistor into conduction. The electron current subsequently causes a voltage drop in the n-well, further turning on the first transistor. If the product of the current gains is larger than one, the final current flow between the supplies can be self-sustaining, limited only by internal resistance's, i.e., an SCR.

Although triggering is by an overvoltage event (typically of only a diode drop above or below the power supplies), the industry practice is to classify the I/O susceptibility in terms of the amount of excess current the pin can source or sink in this overvoltage condition before the internal parasitic resistance's develop enough voltage drop to sustain the latch-up condition. A value of 100 mA is generally considered adequate, with 200 mA considered immune to latch-up.

The *second* triggering mechanism occurs if a supply voltage is raised enough to break down an internal junction, injecting current into the SCR previously described.

This triggering mechanism can occur due to supply transients, or electrostatic discharges shunted to a supply rail. Unlike the case of I/O triggering, latch-up can occur anywhere on the die and is not limited to the vicinity of the external power connections or I/O pins.

The susceptibility to power supply overvoltage is usually limited by the fabrication process on which the device is manufactured, and can be found in the data sheet under the Absolute Maximum Rating specification. If this rating is exceeded, permanent EOS damage may occur. Operating a device near the maximum ratings may degrade the long term reliability of the device. Also the electrical specifications are applicable only at the supply specified on the data sheet and will not be guaranteed above these ratings.

Design Rules

The following is a set of rules to be followed for all designers using CMOS and Bipolar-CMOS ICs: [5]

1. Digital inputs and outputs should not be allowed to exceed V_{DD} by more than 0.3 volts at any time. This includes a power-down situation when $V_{DD} = 0$ volts.
2. Digital inputs and outputs should also not be allowed to go below DGND by more than –0.3 volts.
3. For mixed signal devices, DGND should not be allowed to exceed AGND by 0.3 volts.
4. For a CMOS or Bipolar-CMOS DAC, I_{OUT} should, in general, not be allowed to drop below AGND by more than 0.3 volts. Some DACs can tolerate significant I_{OUT} current flow, however, without any danger of latch-up.

Latch-Up Prevention Techniques

The following recommendations should be implemented in general, for all applications with CMOS and Bipolar-CMOS ICs that violate one or more of the previously discussed rules:

1. If the digital inputs or outputs of a device can go beyond V_{DD} at any time, a diode (such as a 1N914) connected in series with V_{DD} will prevent SCR action and subsequent latch-up. This works because the diode prevents the base current of the parasitic lateral-PNP transistor from flowing out the V_{DD} pin, thus preventing SCR triggering. This is shown in Figure 10.

   Diodes are also a reliable solution if power-up sequencing is identified as the failure mechanism. In such a case, the insertion of a Schottky diode between the logic inputs and the V_{DD} supply rail (the anode of the diode connected to the logic inputs), will ensure that the logic inputs do not exceed the V_{DD} supply by more than 0.3 volts, thus preventing latch-up of the device.

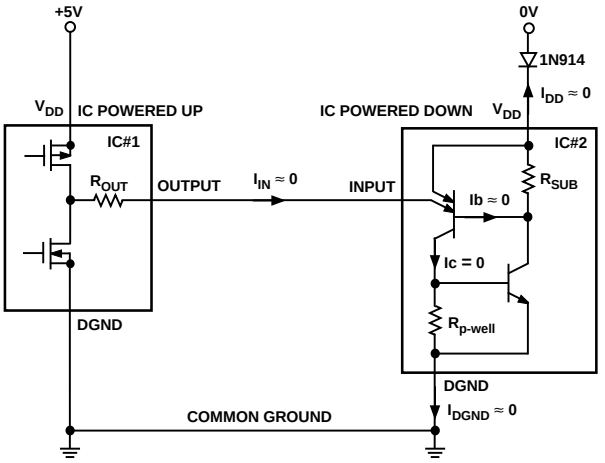


Figure 10. Adding an inexpensive silicon diode in series with the V_{DD} pin of the unpowered IC effectively prevents the parasitic lateral-PNP transistor's base current from flowing and inhibits SCR action.

However, the one *exception* to this rule is when the input range of a device exceeds the supply voltage range of the device, e.g., by design the AD7893-10 12-bit A/D subsystem, the input range is ±10 V and the supply is +5 V.

2. If the digital inputs and outputs of a device can go below DGND at any time, a Schottky diode (such as an HP5082-2835) connected from those inputs or outputs to DGND will effectively clamp negative excursions at –0.3 volts to –0.4 volts. This prevents the emitter-base junction of the parasitic NPN transistor from being turned on, and also prevents SCR triggering. Figure 11 shows the connections for the Schottky diodes.

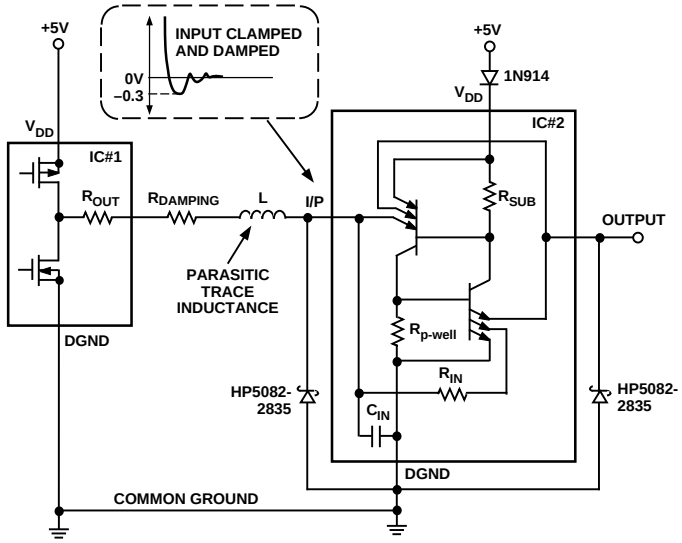


Figure 11. Adding Schottky diodes from the inputs and outputs of a CMOS IC to DGND protects against undervoltages causing conduction of the parasitic NPN, thus inhibiting SCR action. The series damping resistor makes ringing due to long PC board traces die out more quickly.

3. If the DGND potential can occasionally exceed AGND by more than 0.3 volts, a Schottky diode placed between the two pins of the device will prevent conduction of the associated parasitic NPN transistor. This provides additional protection against latch-up as shown in Figure 12. An extra diode connected in inverse parallel with the one just mentioned provides clamping of DGND to AGND in the other direction and will help to minimize digital noise from being injected into the IC.

To identify over- and under-voltage events as described in points (2) and (3) above, the use of a storage oscilloscope is suggested, set at the maximum ratings specification for each pin. Set the Time/Div. to the minimum setting on the oscilloscope (preferably in the ns range). This test should be conducted over a long period of time, e.g., overnight.

4. In circuits where the I_{OUT} pin of a CMOS IC can be pulled below AGND, another Schottky diode clamp between these two terminals will prevent sensitive

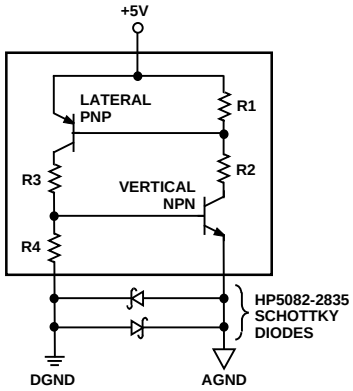


Figure 12. Connecting Schottky diodes between DGND and AGND prevents conduction of the parasitic NPN transistor, and helps to minimize injected noise from DGND to the analog output.

ICs from latching up. This condition sometimes occurs with high speed bipolar operational amplifiers that are used as current-to-voltage converters following a DAC. During power-up or power-down transitions, the op amp's inverting input presents a low impedance from I_{OUT} to the negative supply rail. An unprotected DAC may fail without the recommended Schottky diode clamp to AGND.

5. In designs that have long digital PC board traces between components and are therefore prone to inductive ringing problems, a series damping resistor of 10 Ω–100 Ω will be beneficial. This resistor increases the damping factor of the equivalent series RLC network and causes the ringing to decay more quickly. This will help to prevent conduction of the input or output protection diodes.

High Voltage Transients

If power supply overvoltaging is identified as the failure mechanism, a reliable solution is the insertion of a TransZorb* transient voltage suppressor (TVS). What is a TVS and how does it work?

Transient voltage suppressors [6] (TVSs) are devices used to protect vulnerable circuits from electrical overstress such as that caused by ESD, inductive load switching and lightning-induced line transients. Within the TVS, damaging voltage spikes are limited by clamping or avalanche action of a rugged silicon pn junction which reduces the amplitude of the transient to a nondestructive level.

In a circuit, the TVS should be "invisible" until a transient appears. Electrical parameters such as breakdown voltage (V_{BR}), standby (leakage) current (I_D), and capacitance should have no effect on normal circuit performance.

To limit standby current and to allow for variations in V_{BR} caused by the temperature coefficient of the TVS, the TVS breakdown voltage is usually 10% above the reverse standoff voltage (V_R), which approximates the circuit absolute maximum operating voltage. When a transient occurs, the TVS clamps instantly to limit the spike voltage to a safe level, called the clamping voltage (V_C), while conducting potentially damaging current away from the protected component.

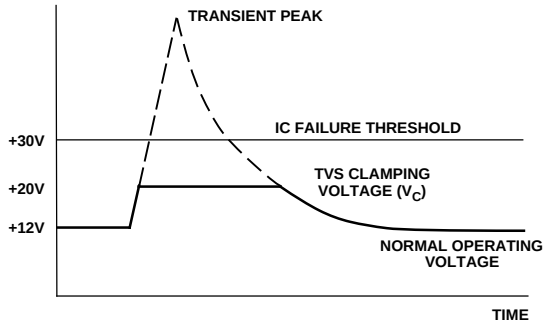


Figure 13. Transients of several thousand volts can be clamped to a safe level by the TVS.

TVSs are designed, specified and tested for transient voltage protection, while a Zener diode is designed and specified for voltage regulation. Therefore, for transient protection the TVS should be selected over the Zener.

The surge power and surge current capability of the TVS are proportional to its junction area. Surge ratings for silicon TVS families are normally specified in kilowatts of peak pulse power (P_P) during a given waveform. Early devices were specified with a 10/1000 μs waveform (10 μs rise to peak and 1000 μs exponential decay to one half peak), while more recent devices are rated for an 8/20 μs test waveform. Power ratings range from 5 kW for 10/1000 μs, down to 400 W for 8/20 μs. This power is derived from the product of the peak voltage across the TVS and the peak current conducted through the device.

TVSs have circuit operating voltages available in increments from 5 V up to 376 V for some families. Because of the broad range of voltages and power ratings available (as well as the universal presence of transient voltages), TVSs are used in a wide variety of circuits and applications.

As an example, consider a pressure transducer which operates at 28 V, placed in an environment in which it encounters a transient voltage of 140 V peak, having a source impedance of 2 Ω and a duration of 10/1000 μs. The failure threshold of the transducer is 40 V, therefore the TVS must clamp at 40 V or less. The current delivered by this transient is:

$$I = (140\ V - 40\ V)/2\ \Omega = 50\ A$$

Note that the voltage clamping action of the TVS results in a voltage divider whereby the open circuit voltage of the transient appears across the combination of the source impedance and the TVS device. Thus the TVS

*TransZorb is a registered trademark of General Semiconductor Industries, Inc.

clamping voltage is subtracted from the transient voltage leaving a net source voltage of 100 V. When the clamping voltage is high compared to the transient peak voltage, the current is significantly reduced.

This circuit can be protected with a 5 kW rated TransZorb TVS which will easily sustain the surge current.

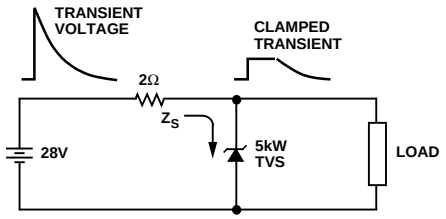


Figure 14. A 5 kW TVS is required to handle the surge current.

An alternate and more economical approach is to add a series resistor to effectively increase the source impedance thus limiting surge current as illustrated in Figure 15. Since the current drawn by the transducer under normal operation is small (<20 mA typical), performance is not adversely affected by a reduction in supply current.

For a small load current, 10 mA, the voltage drop across the added resistance is minimal, about 250 mV for a 25 ohm resistor. Adding this resistor reduces the surge current to:

$$I = (140\ V - 40\ V)/(2\ \Omega + 25\ \Omega) = 3.7\ A$$

This is less than one-tenth the surge current without the resistor. A TVS with lower power rating is able to handle the resulting current. In this case a 500 W suppressor replaces the 5 kW device, saving board space and cost.

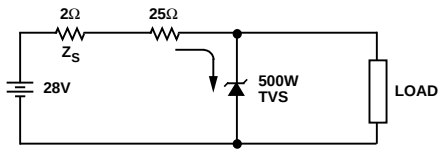


Figure 15. The series resistor reduces transient current allowing a much smaller TVS to be used.

Carbon composition resistors are recommended for this application because of their energy dissipation capability. Steady state power dissipated by the resistor (V ×I) is 2.5 mW requiring the lowest rated resistor available for adequate margin.

TYPICAL TVS APPLICATIONS

DC Line Applications

TransZorb TVSs on power lines prevent IC failures caused by transients, power supply reversals or during switching of the power supply between on and off (Figure 16).

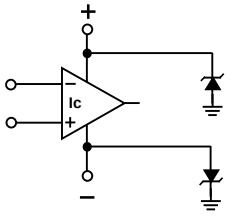


Figure 16.

For power sources utilizing the TransZorb TVS, the TransZorb TVS is chosen such that the reverse stand-off voltage is equal to or greater than the dc output voltage. For certain applications it may be more desirable to replace the series resistor (R) with an inductor (Figure 17).

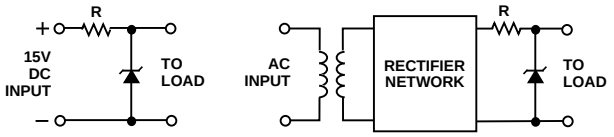


Figure 17.

Signal Line Applications

Input pins are vulnerable to low energy, high voltage static discharges or crosstalk transmitted to the signal wires. Limited protection is provided by the clamp diode or an input network within the IC substrate (Figure 18).

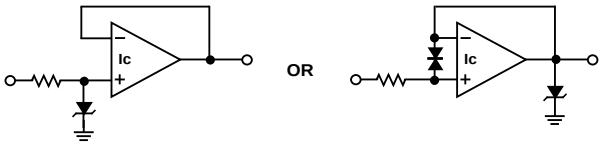


Figure 18.

Transients generated on the line can vary from a few microseconds to several milliseconds in duration and up to 10,000 volts in magnitude. Excess current passing through the diode can cause an open circuit condition or a slow degradation of the circuit performance. TransZorb TVSs located on the signal line can absorb this excess energy (Figure 19).

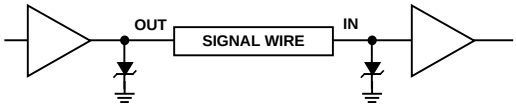


Figure 19.

A further reference on the subject of using TransZorbs for circuit protection is Analog Devices Application Note AN-311, entitled "How to Reliably Protect CMOS Circuits against Power Supply Overvoltaging."

IN SUMMARY

Designing an application with maximum protection of the integrated circuits is a challenging problem with a solution that depends on many factors. The following is a brief summary of the protection schemes discussed in this application note:

1. Personnel should be trained in the proper handling techniques for prevention of EOS/ESD damage.
2. A good facilities ground system including shielding of equipment and data lines should be implemented.
3. Use transient suppressors judiciously, i.e., check if there are spikes on the supply and the ground lines which may exceed the maximum ratings of those pins.
4. Review the proper power-up sequence of the device(s). The correct order should normally be: GND, Main supplies (if possible the substrate supply being first), V_{CC}, $V_{REF+/-}$ and finally all other pins.
5. Review the data sheet, in particular the maximum ratings section.

Remaining devices from a lot that may have been mistested or subjected to the same conditions as those of any failing devices should be evaluated to determine if latent damage may be present. This analysis should be performed due to the possibility that overstress conditions existed which did not cause immediate failure but induced subtle damage that could result in long-term reliability problems.

Finally, the issue of input overvoltage protection for amplifiers is not discussed in this application note. However, it is exclusively discussed in two other Analog Devices publications; (1) Joe Buxton, "Simple Techniques Protect Amplifiers from Input Overvoltage," *Analog Dialogue* 28-3, 1994, and (2) Joe Buxton, "Input Overvoltage Protection," *System Applications Guide*, Analog Devices, 1993, pp 1–56 to 1–74.

REFERENCES

[1]Henry Domingos, "Circuit Design for EOS/ESD Protection," Proc. 1982 EOS/ESD Symp., pp. 1–17 to 1–21.

[2]John A. Schmidt, Manager of Technical Services IMCS Corporation, Santa Clara, CA, "CDM–The Newest ESD Test Model," 1991.

[3]MIL-STD-883 Method 3015, "Electrostatic Discharge Sensitivity Classification," Military Standard Test Methods and Procedures for Microelectronics.

[4]*ESD Prevention Manual*, 1986. Norwood MA; Analog Devices Inc., pp 9-11. Contains additional references.

[5]Mark Alexander, "Understanding and Preventing Latch-Up in CMOS DACs," AN-109. Free from Analog Devices, PMI Division.

[6]*General Instrument, Power Semiconductor Division Data Book/11th edition*, pp. 633, 696–703. Contains additional references.

Andrew Olney, Analog Devices, Inc., personal communication

E2054–15–8/95

PRINTED IN U.S.A.

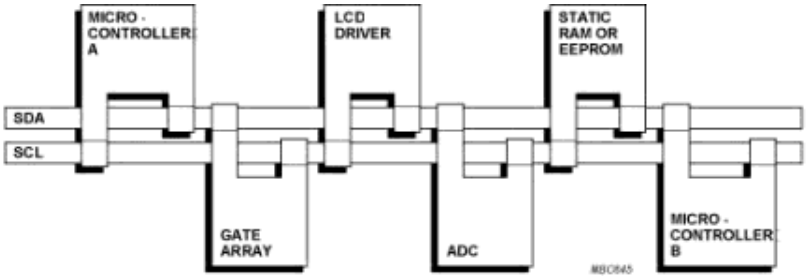
I²C-bus

About the I2C-bus

- What is the I2C-bus?
- Level-shifting I²C
- Fast-mode I²C
- High-Speed mod I²C
- 10-bit I²C addressing
- The I²C-bus specification

Buses
I²C-bus
*Products
*Facts & figures
*News & events
*Support & tools
*Requesting I²C slave addresses
*Licensing

In modern electronic systems there are a number of peripheral ICs that have to communicate with each other and the outside world. To maximize hardware efficiency and simplify circuit design, Philips developed a simple bi-directional 2-wire, serial data (SDA) and serial clock (SCL) bus for inter-IC control. This I²C-bus supports any IC fabrication process and, with the extremely broad rang I²C-compatible chips from Philips and other suppliers, it has become the world industry standard proprietary control bus.



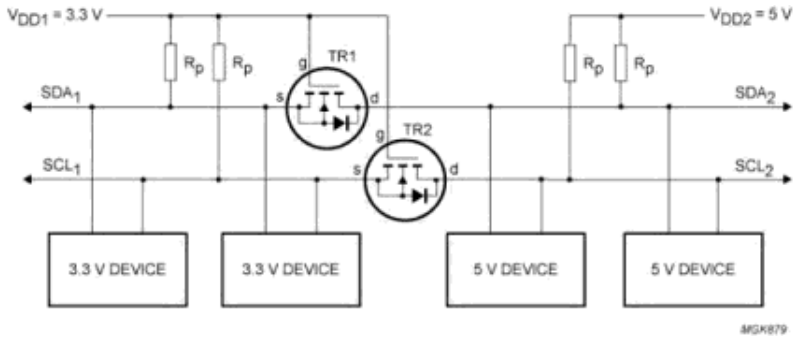
Each device is recognized by a unique address and can operate as either a receiver-only device (e.g. an LCD driver or a transmitter with the capabilit

y to both receive and send information (such as memory). Transmitters and/or receivers can operate in either master or slave mode, depending on whether the has to initiate a data transfer or is only addressed. I²C is a multi-master bus, i.e. can be controlled by more than one IC connected to it.

The basic I²C-bus, with a data transfer rate up to 100 kbits/s and 7-bit addressin was originally introduced nearly 20 years ago. But, as data transfer rates and application functionality rapidly increased, the I²C-bus specification was enhan to include Fast-mode and 10-bit addressing, meeting the demand for higher spe and more address space.

The I²C-bus continues to keep pace with advancing technology while retaining backwards compatibility. Mixed designs incorporating new low voltage devices supported via the I²C-bus' level shifting capability. And, most recently, High-sp mode has been added; with speeds of up to 3.4 Mbits/s it ensures the capacity o I²C-bus to support existing and future high speed serial transfer rates for applica such as EEPROM and Flash memory.

Level-shifting I²C

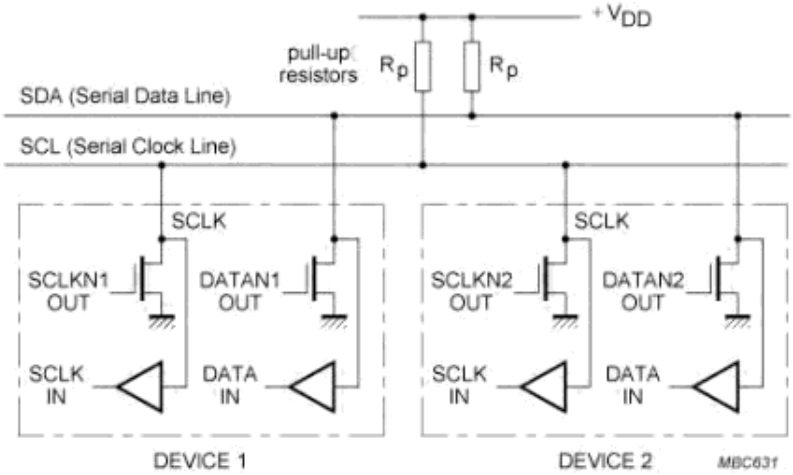


Philips Semiconductors has developed a simple 'level-shifting' enhancement to standard I²C-bus, offering fully bi-directional data transfer between I²C-devices operating from different supply voltages. It provides an elegant solution for allo ICs with different supply voltages to communicate and be controlled, for almos additional design-in effort or cost.

The simple addition of two low-cost transistors, placed between the different vo level sections of the I²C-bus, separates and transfers the logic voltage levels of bus lines on either side of the level-shifter. In fact, with the addition of these transistors, the I²C-bus answers all level-shifting needs for a complete multi-sup voltage system design. This set-up also allows the level-shifter to be used to iso section of powered-down devices from the bus, allowing powered I²C devices t operate normally.

At the same time, the I²C-bus specification has been extended for devices opera below 2.7 V. This extension, together with compatibility of lower and higher vo devices provided by the bi-directional level-shifter, ensures the I²C-bus will ren the serial bus of choice for future systems into the next century.

Fast-mode I²C



Until 1992, the I²C-bus was mainly used for the transfer of control and status information and its originally defined bit rate of 100 kbits/s remained sufficient this purpose. As it became a de-facto standard, it began to be used for text and d transfer and to meet these needs, in 1992 the specification was upgraded with a Fast-mode, supporting bit rates up to 400 kbits/s. Compatible with Standard-mo devices, all Philips Semiconductors devices developed since 1992 have include Fast-mode I²C-bus interface. Support for even higher data transfer speeds is no available through High-speed mode (which also offers the level-shifting enhancement).

High-speed mode I²C-Bus

Developments in high-speed serial RAMs and mixed technology telecom syste have created a demand for buses capable of operating at high speeds and with a variety of supply voltages. I²C's High-speed mode meets these needs without sacrificing its compatibility with existing Standard and Fast-mode devices or its low-cost simplicity - no special logic levels, timing or drive capability are need

A purpose-designed bridge in a High-speed mode master allows for bi-directio communications between Fast- and Standard-mode devices within a single I²C- system and, if required, this master can also perform bi-directional level shiftin supporting a variety of different voltage devices.

As a true multi-master bus, I²C's High-speed mode requires no additional wires pins for slave devices. Also, there are two additional pins on a High-speed mast which, if they are not used (because there is no system bridge or only Fast/Standard-mode are in use), then other I/O functions can be assigned to thes pins, making I²C an extremely flexible solution.

10-bit I²C addressing

10-bit addressing allows the use of up to 1024 additional addresses to prevent problems with the allocation of slave addresses as the number of I²C devices ra

expands. It does not change the format for addresses defined in the I²C-bus specification, using addresses reserved in the existing specification.

10-bit addressing does not affect the existing 7-bit addressing, allowing devices 7-bit or 10-bit addresses to be connected to the same I²C-bus, and both types of devices can be used in Standard-, Fast- or High-speed mode systems.

I²C-bus specification

The latest update to the I²C-bus specification, version 2.1, includes full details of the standard I²C-bus interface plus specifications for all the enhancements including Fast-mode, High-speed mode, 10-bit addressing and details on the bi-directional level shifter.

I²C-bus specification

How to use your RS232 or IRDA port for Remote Control

Update History - skip to Introduction if this is your first visit

26/02/2001
FAQ page added

28/02/2001
FAQ No.2 updated to include extra information on component selection

26/08/2002
Theory updated
Hardware schematic and text updated
Software Section text and code updated (winsamp now version 1.3)
FAQ updated
******* This was a big update - please let me know if I've broken anything (that used to work!) *******

Introduction

Firstly, an apology. There is a lot to read on this page and I'm struggling to make it even slightly visually appealing. If you feel that the subject is worthy of your attention, then I suggest you print this page out or at least save it to disk, snag the zip file mentioned in the software section (not a big download) then study it all offline.

Unlike some of my other stuff, this one is fairly serious. It is intended to introduce a concept and demonstrate its application rather than present the definitive finished product, but having said that, everything needed to get it working properly on a PC is included, with enough information and flexibility to enable the software as supplied to be incorporated into much more elaborate Windows front-ends if desired. The idea is not limited to PC platforms, however - most devices with serial or IRDA ports are possible candidates for the technique as described, and samples made on one platform should transfer easily to other platforms.

A number of people have done good work in the field of turning a PC into a 'learning' universal infrared remote control, for various appliances such as TVs, video recorders, satellite decoders etc. The common methods appear to use either an IR detector and transmitter circuit attached to the parallel port, or an 'intelligent' device accepting high-level commands attached to a serial port. A quick online search will turn up schematics, construction details, driver software and some very attractive Windows front-ends for this purpose.

Following on from my Furby experiments (http://www.veg.nildram.co.uk/furby.htm), I set out to prove that it is possible to control the TV etc using either an IRDA port or a standard RS232 port with extremely minimalist hardware. The reasons were threefold. First, the challenge. Second, the convenience or geek value - many PCs, PDAs and other devices already come with IRDA as standard, wouldn't it be nice to use it for something different. Third, DEVICE INDEPENDENCE AND PORTABILITY OF SAMPLES - serial ports of one type or another are fairly ubiquitous,

more so than parallel ports anyway, and a drawback with sampling and playback through the parallel port is the heavy dependence on processor speed and environment - samples made on one speed of machine may not work reliably on a different speed of machine (no criticism intended, but this is what I found when I tried it), so to have a good probability of success you have to sample your remotes on the machine you intend to use for playing them back - consequently, you can't simply sample your Sony CD remote and send the file off to your pal in Greenland when he loses his [ahaaa - until now :-)]. Also, most systems have more than one COM port, so it's no great hardship tying one of them up permanently for remote control. The next bit is a disclaimer of sorts, then we'll get on with the theory.

Disclaimer - Please read this - Important

What I am about to describe is my own original idea, my own software, my own project. As far as I am aware, nobody else does it, or has done it, this way, but if they have, what can I say? Great minds think alike? I am not intentionally ripping off anyone else's work, and have worked long and hard to get it into its present state, for my own amusement, with no prior knowledge that it can or cannot be done or indeed that it has been attempted in this fashion. The ideas and software have been put here because I thought you might be interested in how I did it. It all works as far as I'm concerned, but I'm not asking or telling you to use it in any way, shape or form. The software (should you choose to try it) can write small text and binary files containing sample information to your disk, or whatever you run it from. Also, the Windows demo application keeps a couple of settings in the Registry to enable it to remember preferences from one session to the next. None of this should cause you any distress, but as with all things, it could always go horribly wrong, so you have been warned. If you don't want to risk using my software, you should be able to write your own using the information in the theory section. Also be advised that there is a class of IR remote control (so-called 'flash' controls) which is not supported by the software in its present form, but the project is still in a state of flux and I may add support later - the issue here is one of trying to keep it simple enough for anyone to use when they don't have the use of a proper 'scope - I'm sure you don't wish to become an expert on remotes (not that I am, but a certain amount of it inevitably rubs off).

Anyway, with the software and hardware described, I routinely control a Toshiba TV, JVC video recorder and Maspro satellite receiver either using a laptop irda port or a normal 232 port on my desktop PC, using samples made from the original remotes and without having to know details of the three different protocols actually employed by the devices. There is a good chance that most other appliances will work equally well ('flash' devices being known exceptions). If you can't get a device to work, the sampler software will at least provide information to help identify the problem if you wish to look deeper into it.

Finally, if you would like me to investigate the implementation of this technique on other platforms, you should send me non-returnable hardware and development software for the experiments, and if you make a fortune based on the idea, I certainly wouldn't mind you giving me a share :-)

Blank Frank 24th November, 1999.

Theory - Updated 26th August, 2002

Ok. Now to the good bit. Without getting too technical, here is an idea of what's going on and why. Though there is a mind-bogglingly large selection of remote control protocols at the bit level, based on quite a number of types of protocol at the packet level, the vast majority of ir remote controls function by sending data at relatively low rates by transmitting bursts of (carrier) pulses of various lengths with periods of silence inbetween, also of various lengths. The data is encoded in the lengths of the bursts and silences, and receivers demodulate the carrier to recover the baseband data. The carrier frequency is typically somewhere in the range 36 - 40 kHz, and receivers are designed to allow signals of this frequency range through (+/- a few kHz) while rejecting signals outwith this range in order to reject such things as strip lights.

You may wish to brush up on your serial port theory for this - lots of info online, maybe enough on the Furby page. I make the assumption that most receivers will be happy with combs of pulses at 38.4kHz, since they are designed round R-C filters and diode pumps (at least conceptually, though most use one integrated device). I can achieve this by transmitting suitable characters at 115200 baud out of any serial port, so long as there are no gaps between characters. The precise pattern for a comb is the character value 0xdb, at 115k2/8/e/2 (0x5b at 115k2/7/n/1 would also be possible but would require data to be loaded into the uart more frequently). There is sufficient tolerance in most receivers to cope with the effective 4-pulse granularity (or 3-pulse if 7/n/1) in the comb. The next assumption I make is that out-of-frequency-band pulses will be rejected by the receiver, so if I transmit 0xfe, still at 115k2/8/e/2, this will look like silence to the receiver. So that's the transmission part covered - by sending correctly-sized groups of characters with values 0xdb and 0xfe, I can effectively construct an infrared stream which looks like it came from a 'real' remote control from the receiver's point of view, so long as I keep the uart continuously stuffed with data. The timing is derived from the uart, not the raw speed of the PC or other host device, and it is fairly easy to keep up.

Sampling works as follows. Using the very simple hardware described below, (or your own equivalent - doesn't need to be a handshake line, could be a joystick or printer input, for example), I transmit a junk character (0xff as it happens, but it doesn't matter) continuously out the serial port with no gaps between characters. While waiting for each character to be transmitted, I sample the IR input as frequently as possible (in my case a handshake line) and if I ever see an 'active' condition on it (ie if IR is ever detected) I assume that there is a comb (36 - 40kHz) of data being received from a remote control during this period, and if I don't see activity I assume this is a silent period. I grab a big array, with one bucket for each character time, and the buckets will either be marked as active or silent. After a preset number of characters have been sent, I stop grabbing and process the data. What I end up with is another array of buckets which contain alternating active counts and silent counts, stored as numbers of character times, adjusted to compensate for the inherent tendency of this method to round up comb lengths and round down silence lengths. Arrays like this provide a compact method of sample storage, and are very easy to play back when I want to transmit the sample as described above, and again, the timing is absolute, based on the uart, not the processor raw speed, so it is accurate and repeatable. By understanding the actual protocol used, it would be easy to compute the packets rather than store samples as such and make the storage space requirement much smaller, but I'm trying to avoid this to keep things simple and flexible. There are a few details to do with timeouts, sample sizes,

waiting for the data to start, suspending interrupts etc which will be explained elsewhere, but the basic principle is very simple and apparently rock solid.

It is not possible to sample remotes through the IRDA port - at least with my software, but probably not at all on a PC - due to the design of the port hardware. Samples have to be made using a 232 port using the hardware described below, or equivalent, but this should not be a problem. Playback through the IRDA port works fine so long as you are aware of the potential pitfalls (again covered to some extent on my Furby page, and in particular relating to operation under Windows).

I have run the same software on an 8MHz 286 (yes, I managed to find one that still works) and a K6-2 333 and got the same results. Samples made on both extremes of machine look the same and can be exchanged without problems. This suggests that the technique will work well on a wide range of devices with serial ports, not just PCs. The actual sample-grabbing process is designed to scream along, even on a slow machine, though the post-grab processing can take as much time as it needs since it is not real-time critical.

Update 26th August, 2002
Things move on. Since writing the above section, PCs have got considerably faster. I am happy to report that the system has now beem tested unchanged on Athlon 1000 boxes and above, without problems, under Windows 95 / 98SE, and, from the feedback I've had, the project has been constructed and run on a variety of PCs, and implemented on and some other platforms, in various parts of the world.

Now the interesting news. I have figured out a way of retaining all the uart-derived timing properties of transmission described above while achieving 'real' silence during the silent periods rather than the combs of out-of-frequency-band pulses which have been used up to now. It works on all the PCs I can get my hands on. The latest version of the DOS program (version 1.3) implements this and no longer has the option to switch in software timing loops to achieve (with some calibration effort) the same effect. It takes advantage of some particular features of the PC uart hardware, so I wouldn't class it as a new method, rather as an optimisation aimed at PCs. Since the vast majority of users are PC users, this new version should be of general interest, though I stand by the original method as described above as the more general solution. I have to say that I personally don't have a problem using the original silence method with the receivers I'm controlling, but this reworked version may be of some help with borderline cases, and this is now the 'official version' - I've moved over to it anyway, because it has a couple of other features I wanted, which are described elsewhere in the docs.

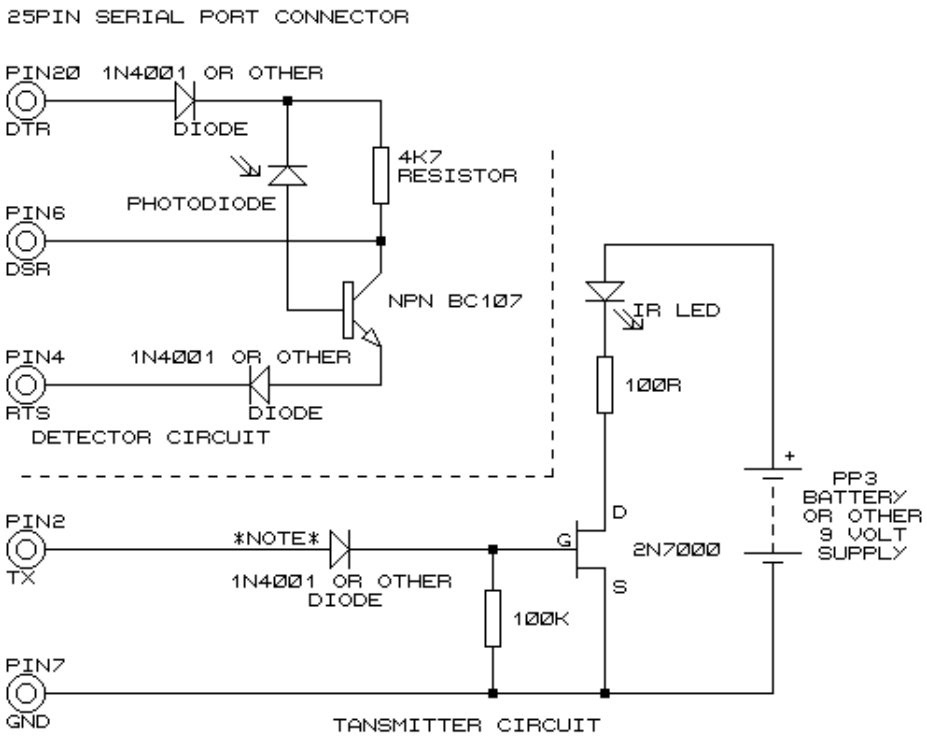
If you are a first-time user of this project, don't worry, just use the latest version of the software.

If you are an existing user of the system, there are several reasons why you might wish to install the latest version. Firstly, real silence rather than silence combs should work with a larger selection of equipment - the original silence combs might somewhat saturate the IR receiver or indeed totally confuse some poorly-filtered equipment (always one of the known drawbacks of the method), so the new method will hopefully improve the IR on/off contrast ratio and thus improve the operating range or make it work properly with your target equipment for the first time.

Secondly, the data stream will be much easier to observe on a proper oscilloscope than it used to be. Thirdly, you can do fun things like transmit a sample from one PC to another and get a close match (I know, you could do it more easily using a floppy or whatever). Fourthly, the new software has some helpful features in DOS-screen mode which make it easier to experiment, diagnose problems and manipulate data sets. Fifthly, the average power consumption of transmission is reduced, prolonging your battery life, saving the planet and so on. The main thing here is I can see no downside. All you need to do is replace the DOS program - the change is invisible to the Windows program, and any samples you already have remain totally compatible. At worst, you get the benefit of the new features; at best you may also get better performance.

Hardware - Section updated 26th August, 2002

Here is a schematic for the hardware I use. This has been updated on 26th August, 2002 to include the diode explained below. If you built yours before this date, and you think your LED might be glowing slightly (in an IR sense) when it should be off, you should consider adding the diode even if everything seems to work fine. It may give you improved performance. You can check to see if you have the problem using a video camera or by measuring the current through the LED in the idle state - don't bother changing it if you don't see the problem. Having said that, it only hit me when I started playing with substitute transistors which were intended to be near-equivalents.



There are really two separate circuits, and it would be ok just to build the receiver section for sampling and use the IRDA port for transmission. The receiver section takes the power it needs from handshake lines (so no external source required in this configuration). The receiver range is

only a couple of cm, so the remote has to be close to work, but there is no particular reason why a proper circuit shouldn't be built which would operate over several metres, and indeed software could be written to let the PC decode the signal properly, though it would be a non-starter as a mouse substitute under Windows or whatever. There are additional notes about the driver transistor (2N7000) on my Furby page (here).

The diode marked *NOTE* between the TX pin and the gate of the FET has been added because I occasionally observed the LED to be slightly on when the TX line was low and the LED should have been off, with the result that the on/off IR contrast ratio was reduced (and consequently maybe the useful transmission range). There are a couple of possible explanations for the LED being on, which I didn't investigate further, opting rather to kill the problem once and for all by including the diode. Shame really, because the tolerance to large positive and negative Vgs values was one of the reasons I chose the FET in the first place. Anyway, the diode effectively prevents the gate from going more negative than the source and the problem is gone. If you suspect that the FET is not turning off fast enough with the 100k pulldown (due to extremely high gate / track capacitance or whatever) you could safely reduce the resistor to 10k to see if that helps, but I have no reason to think the change is needed.

The FAQ page has been updated, and now includes a couple of (untested) suggestions for increasing the output of the transmitter, and the simple change needed in order to replace the FET with a normal NPN transistor.

Software - section updated 26th August, 2002

There are now two versions of the software package available for download, and unless you have a pressing desire to do comparisons between the original version (winsamp version 1.1) and the current one (V1.3), you should ignore REMOTE11.ZIP (37k) and go for REMOTE13.ZIP (39k). The packages contain versions of the same set of 5 files - MANYBUTT.EXE, BUTT1.BUT and BUTT18.BUT are unchanged, WINSAMP.EXE and README.TXT have been updated in the newer (remote13.zip) version. If you are updating your installation from 1.1 to 1.3, only the two changed files need to be replaced (obviously). Just for information, V1.2 was an intermediate version which was never published because I didn't want to have to do two page and docs revisions in quick succession, but you're not missing anything because 1.3 contains all of 1.2 plus extra features. At risk of repeating myself, the upgrade will not require you to do anything to your existing sample files, does not change the command line interface, does not modify the settings of the GUI (manybutt.exe), and only requires you to replace winsamp.exe (and the readme file, if you want to keep things in step - and then probably only if you actually want to read it). Note that the remainder of this section is virtually the same as what was here before, so there is no nead to read it again if you're already familiar with the contents.

The software here is enough to get you going both in dos and Windows95/98, but is not pretty, clever or feature-laden. It goes some way beyond proving the point that the method works, but falls well short of some of the applications which have been written around the parallel-port system. It does everything I need it to do as far as controlling multiple devices and producing platform-independent samples is concerned, so I have drawn the line there for the time being, but I hope I will have provided enough information to allow anyone with an inclination towards

programming to write their own much nicer versions or to port the method to other platforms. I have in mind some particular features I might add to the VB side to address some specific future project requirements, but these are so far off the normal track that there would be little point in trying to build them in to a general app, and anyway they still depend on utilising the same dos-based core program.

How to get started

Click here to download REMOTE13.ZIP (39k) , a .zip file which contains five files - WINSAMP.EXE, MANYBUTT.EXE, README.TXT, BUTT1.BUT and BUTT18.BUT. Create a directory (wherever you want and whatever you want to call it) and put these into it. WINSAMP.EXE is the main dos app, and is all you really need to get started. MANYBUTT.EXE is my VB5 demo for Win95/98 which calls the dos program as required. The *.BUT files are example buttons used by MANYBUTT.EXE. README.TXT is all the explanation and detail I didn't want to put on this page, written in plain text, and I only suggest that it be in the same directory so it doesn't get lost or overwrite any other file of the same name - it overlaps with this file, but you would be expected to have both. This html file (REMOTE.HTM) and README.TXT are expected to evolve as I find problems, develop the idea further and hopefully if and as I get feedback. Create a shortcut to MANYBUTT.EXE on your desktop for convenience. You can also create one for WINSAMP.EXE if you prefer to access it this way rather than from dos prompt, but if you want to see what's going on, it would be best if you make sure (by tweaking properties if necessary) that it runs Full-Screen. Remember that if you want to run it on anything other than COM2, you'll need to edit the command line to WINSAMP.EXE Cx. You can actually look at and fiddle with the software without having any sample/playback hardware. If you already built the IR Thingy from the Furby page, you're half way there hardware-wise, and if you managed to get the Furby stuff working, there's a very good chance this will work too. What I suggest is you save this page or print it out, get the software and read the readme file, maybe have a look at the programs then decide whether or not you can be bothered to build the hardware. If you get that far, then go to dos prompt and run WINSAMP.EXE, using appropriate COM port selection and start trying to sample remotes. If that seems to function, then use MANYBUTT.EXE from the desktop - it's a lot more convenient in the long run. Note that only COM1 - COM4 are supported by WINSAMP.EXE, and that MANYBUTT.EXE does not touch the ports as such. MANYBUTT simply passes a string, which the dos app uses to determine the hardware addresses it hits directly.

Some more detail

The main software (winsamp.exe, 21k) is a dos application which can be run in stand-alone mode or shelled from Windows (tested with 95 and 98). It is not Windows-aware, and consequently takes more processor time when idle than it really needs, but the normal way to use it under Windows is to shell it with various command line options set, which causes it to start, do what you want (eg grab a sample or play back a sample) then exit without any keyboard / mouse / screen activity, so it's normally only loaded briefly, and the program is so small that it takes a negligible amount of time to load. Note that once a sample starts to arrive or starts to be played back, interrupts are suspended until the operation has finished. In the case of sampling, the default sample window is approx 250ms, but this can be pushed up to 500ms using command line options. For playback, this time is the duration of the stored sample, which again can be controlled from

the command line but defaults to around 150ms and cannot exceed 500ms. Be aware that suspending interrupts in order to make sure there are no gaps in the data streams will inevitably result in system ticks being missed (I certainly hope so, this is one of the reasons for doing it), and the result is that your clock will gradually lose time, depending on how much you use the software. Sorry. An easy way round it would be to install one of these programs which syncs your PC to an atomic clock every time you go online if it bothers you - to some, the ability to slow down time might be seen as a bonus. A more messy way round it would be for me to attempt something clever with the real-time-clock hardware following each interrupt suspension, but it doesn't bother me that much to lose a few seconds now and again - when the battery was going flat on my PC I used to lose minutes per day and managed to cope, so this is nothing in comparison.

Because the interrupts are not suspended during sampling until the start of the IR stream from the remote is detected, there is a very small chance that a system tick or similar might happen between detection and the instruction which does the suspending. The chance of this happening is extremely small, but becomes more possible on slower machines, and the ISR would probably take longer to complete if it did happen. The effect of this would be that the recorded width of the first comb would be smaller than it actually was, which would perhaps make the sample invalid on playback for some protocols. I think I've seen this once in all the playing around during development, but of course it might be that pressing the button on the remote causes it to move into range of the receiver just at the critical time or whatever. I would suggest observing a few samples on the sillyscope screen to get a feel for what a good sample looks like, and have a quick look at the sample files to see if any are outstandingly different before burning them onto a million CD Roms or whatever. And of course test the samples.

The most usual way to operate the program in 'real' dos or dos prompt under Windows is to type 'WINSAMP' or WINSAMP Cx where x is the COM port you want to use if the default of COM2 does not suit. A 80*50 text-mode screen is presented (which might not be compatible with _very_ old graphics cards, but if your machine can run Windows it should be ok), and this has lots of things squished onto it. You need to read all the bits of writing on it because the user controls are all hinted at somewhere there. For details of command line options, type WINSAMP ? and a horrible text screen will appear, which might help. A separate README.TXT file is included explaining the command line options and controls in more detail, along with examples of how to make it work from Windows programs. I have also included a simple Windows front-end demonstration written in VB5 which uses the dos app for all the low-level stuff. This is only the actual .exe file - due to bandwidth restrictions I can't put up a full distribution - the .exe is only a few k, but all the support bits take up megs of space, so unless you already have VB5 or have acquired the support files through other software installations you've done over the years, I'm afraid you're stuck, but there is a somewhat less satisfactory but quite functional method for accessing the dos program under Windows using nothing more than program groups and shortcuts which is explained in README.TXT.

It may be possible to rewrite the whole thing as a 'proper' Windows application, but using the dos program makes it relatively simple to achieve the necessary low-level hardware access and timing. Certainly, VB or similar could be used to provide sophisticated sample management (archiving, selection, grouping, exporting, importing etc), but I have only done enough to suit my own needs for the time being.

I hope you like it.

I have added a FAQ page. I welcome feedback on the project, but if you have specific questions or requests, please do me a favour and make sure they haven't already been covered by this web page, the docs or the FAQ. Your question and / or my response might be added in a generic and unattributed sort of way to the next revision of the FAQ.

View my other offerings

RLE - Run Length Encoding

Written by Paul Bourke
August 1995

Source code

Standard compression C source: rle.c
Example code based upon the above: rletest.c

Introduction

Run length encoding is a straightforward way of encoding data so that it takes up less space. It is relies on the string being encoded containing runs of the same character. Consider storing the following short string.

```
abcddddddcbbbbabcdef
```

There are 20 letters above, if each is stored as a single byte that is 20 bytes in all. However, the runs of "d" and "b' above can be stored as two bytes each, the first indicating how many letters in the run. For example, the following run length encoded string takes only 14 bytes.

```
abc6dc4babcdef
```

In general of course it needs to be a bit more sophisticated than the above. For example, there is no way in the above encoding to encode strings with numbers, that is, how would one know whether the number was the length of the run or part of the string content. Also, one would not want to encode runs of length 1 so how does one tell when a run starts and when a literal sequence starts.

The common approach is to use only 7 of the 8 bits to indicate the run length, this is normally interpreted as a signed byte. If the length byte is positive it indicates a replicated run (run of the following byte). If the number is negative then it indicates a literal run, that is, that number of following bytes is copied as is. To illustrate this the following sequence of bytes would encode the example string given above, it requires 17 bytes.

```
-3 a b c 6 d -1 c 4 b 6 a b c d e f
```

While 17 bytes to encode what would take 20 without RLE may not sound like much, but as the frequency and length of the repeating characters increases the compression ratio gets better.

Worst case

Of course RLE will not always result in a compression, consider a string where the next character is different from the current character. Every 127 bytes will require a extra byte to indicate a new literal run length.

Best case

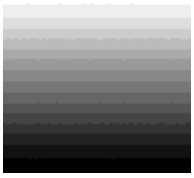
The best case is when 128 identical characters follow each other, this is compressed into 2 bytes instead

of 128 giving a compression ratio of 64.

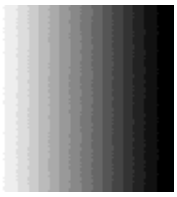
Example

For this reason RLE is most often used to compress black and white or 8 bit indexed colour images where long runs are likely. RLE compression is therefore what was used for the original low colour images expected for the Macintosh PICT file format. RLE is not generally used for high colour images such as photographs where in general each pixel will vary from the last.

The following 3 images illustrate the different extremes, the first image contains runs along each row and will compress well. The second image is the same as the first but rotated 90 degrees so there are no runs giving worse case and a larger file. This suggests a natural extension to RLE for images, that is, one compresses vertically and horizontally and uses the best, the flag indicating which one is used is stored in the image header. The last case is the best scenario where the whole image is a constant value.



Original size: 10000 bytes
Compressed size: 5713 bytes
Ratio: 1.75



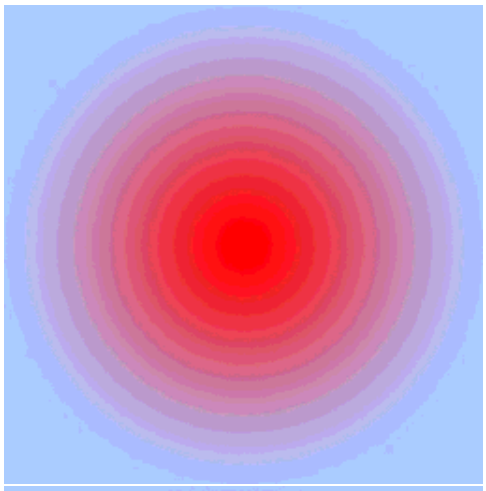
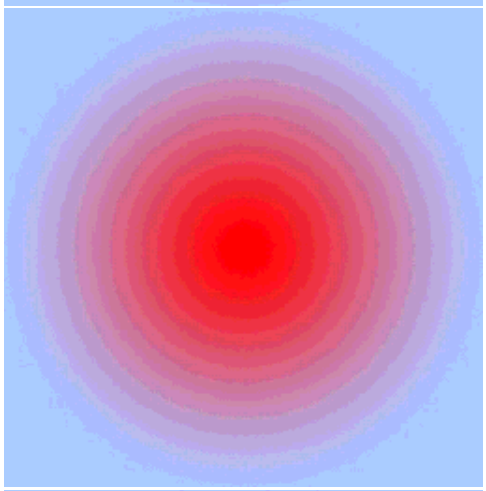
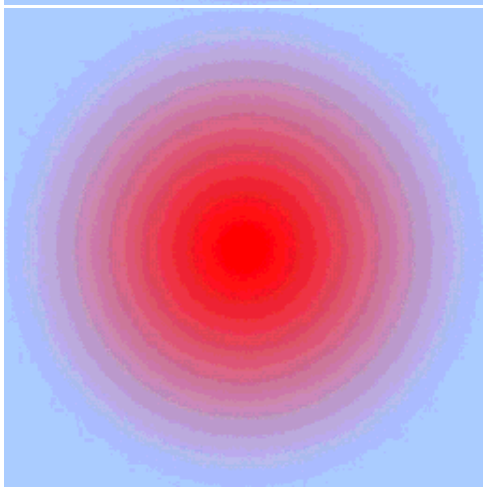
Original size: 10000 bytes
Compressed size: 10100
Ratio: 0.99

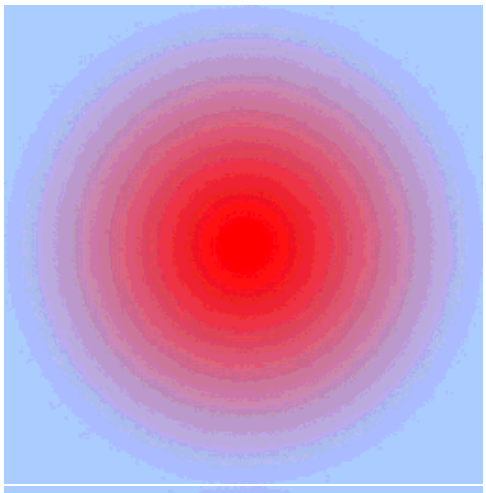
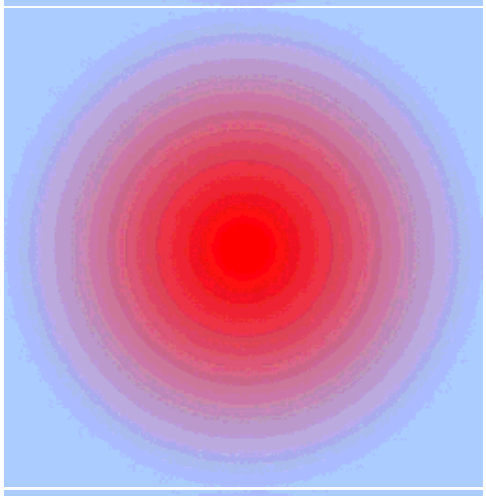
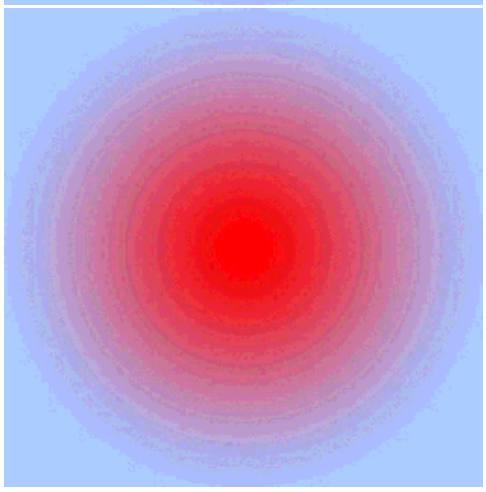
Original size: 10000 bytes
Compressed size: 200
Ratio: 50

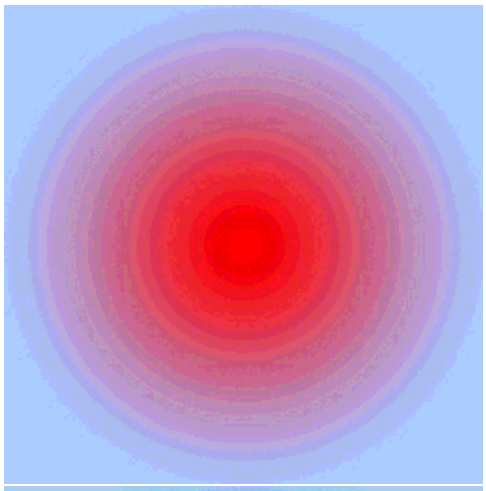
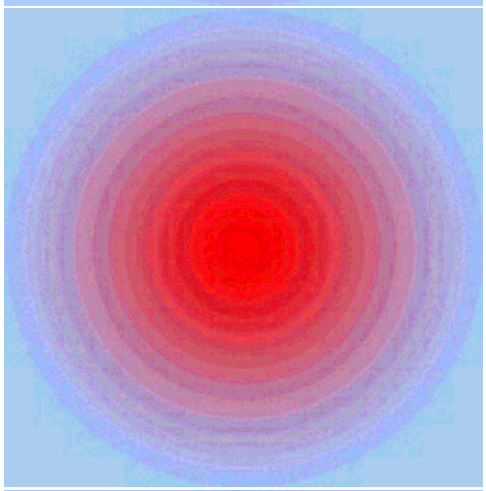
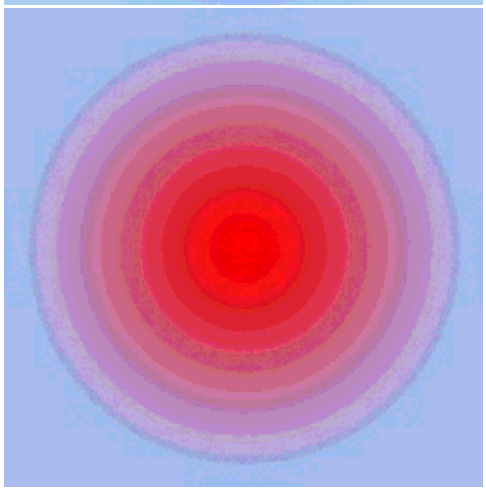
Image comparison

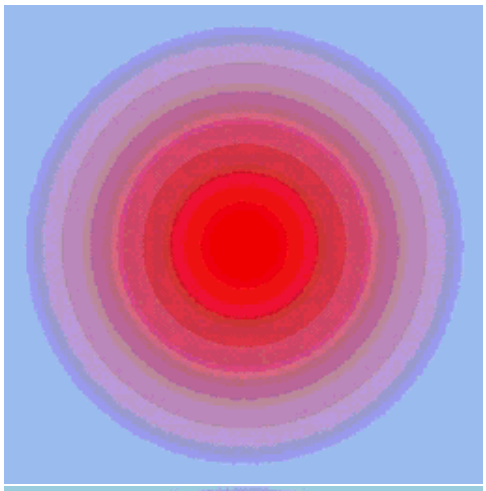
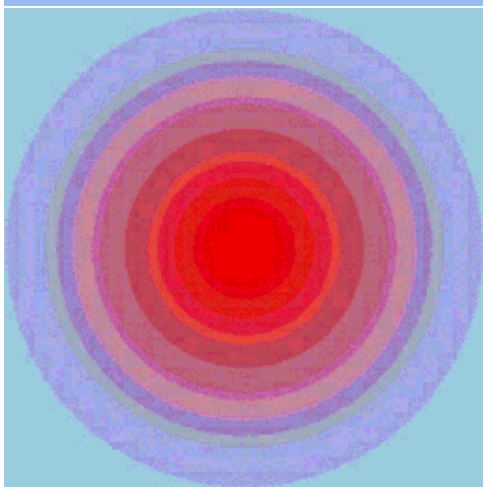
Run length encoding is used within a number of image formats, for example PNG, TIFF, and TGA. While RLE is normally used as a lossless compression, it can be assisted (to create small files) by quantising the rgb values thus increasing the chances of runs of the same colour. There are two ways one can run length encode the pixels, the first as used in the TGA format is to look for runs of all three components, the other is to compress each colour plane seperately. The second approach normally gives smaller files. Below is a table for two different images along with the file size for the image quantised to different levels and saved in rgb order or planar order.

Image details and quantisation level	Image example	RLE on RGB (KBytes)	RLE on Planes (KBytes)

Uncompressed		197	197
1 (none)		151	141
2		135	110

4		101	70
6		81	49
8		64	36

12		46	24.5
16		35	18.5
20		28	14.5

26		22	11.5
32		17.5	9.5

The above example was chosen because it doesn't have long runs of equal colour and because any banding due to quantisation should be obvious to spot. This occurs somewhere between 4 and 8 depending on how fussy one is. Note the planar compression works much better than the rgb based compression.

Image details and quantisation level	Image example	RLE on RGB (KBytes)	RLE on Planes (KBytes)

Uncompressed		197	197
1 (none)		195	190
2		188	173

4		163	140
6		142	119
8		127	106

12		105	88
16		86.5	74
20		92	73.5

26		74	60.5
32		60	50.5

Unlike the first example, because each of the colour layers in this images are "busy" the difference between rgb and planar RLE compression is not so marked. Note that the visual artifacts that occur to so on the wall where there is a smooth and subtle shade variation, even at the highest quantisation level the artifacts on the vase are hard to pick.

RS232 Data Interface

a Tutorial on Data Interface and cables

RS-232 is simple, universal, well understood and supported but it has some serious shortcomings as a data interface. The standards to 256kbps or less and line lengths of 15M (50 ft) or less but today we see high speed ports on our home PC running very high speeds and with high quality cable maxim distance has increased greatly. The rule of thumb for the length a data cable depends on speed of the data, quality of the cable.

---

a Tutorial

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumors for its early demise, has remained widely used through the industry.

Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol.

The RS-232 interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption when a short cable connects the DTE to the DCE, but with longer lines and connections between devices that may be on different electrical busses with different grounds, this may not be true.

RS232 data is bi-polar.... +3 TO +12 volts indicates an "ON or 0-state (SPACE) condition" while A -3 to -12 volts indicates an "OFF" 1-state (MARK) condition.... Modern computer equipment ignores the negative level and accepts a zero voltage level as the "OFF" state. In fact, the "ON" state may be achieved with lesser positive potential. This means circuits powered by 5 VDC are capable of driving RS232 circuits directly, however, the overall range that the RS232 signal may be transmitted/received may be dramatically reduced.

The output signal level usually swings between +12V and -12V. The "dead area" between +3v and -3v is designed to absorb line noise. In the various RS-232-like definitions this dead area may vary. For instance, the definition for V.10 has a dead area from +0.3v to -0.3v. Many receivers designed for RS-232 are sensitive to differentials of 1v or less.

This can cause problems when using pin powered widgets - line drivers, converters, modems etc.

These type of units need enough voltage & current to power them self's up. Typical URART (the RS-232 I/O chip) allows up to 50ma per output pin - so if the device needs 70ma to run we would need to use at least 2 pins for power. Some devices are very efficient and only require one pin (some times the Transmit or DTR pin) to be high - in the "SPACE" state while idle.

An RS-232 port can supply only limited power to another device. The number of output lines, the type of interface driver IC, and the state of the output lines are important considerations.

The types of driver ICs used in serial ports can be divided into three general categories:

- **Drivers which require plus (+) and minus (-) voltage power supplies such as the 1488 series of interface integrated circuits. (Most desktop and tower PCs use this type of driver.)**
- **Low power drivers which require one +5 volt power supply. This type of driver has an internal charge pump for voltage conversion. (Many industrial microprocessor controls use this type of driver.)**
- **Low voltage (3.3 v) and low power drivers which meet the EIA-562 Standard. (Used on notebooks and laptops.)**

Data is transmitted and received on pins 2 and 3 respectively. Data Set Ready (DSR) is an indication from the Data Set (i.e., the modem or DSU/CSU) that it is on. Similarly, DTR indicates to the Data Set that the DTE is on. Data Carrier Detect (DCD) indicates that a good carrier is being received from the remote modem.

Pins 4 RTS (Request To Send - from the transmitting computer) and 5 CTS (Clear To Send - from the Data set) are used to control. In most Asynchronous situations, RTS and CTS are constantly on throughout the communication session. However where the DTE is connected to a multipoint line, RTS is used to turn carrier on the modem on and off. On a multipoint line, it's imperative that only one station is transmitting at a time (because they share the return phone pair). When a station wants to transmit, it raises RTS. The modem turns on carrier, typically waits a few milliseconds for carrier to stabilize, and then raises CTS. The DTE transmits when it sees CTS up. When the station has finished its transmission, it drops RTS and the modem drops CTS and carrier together.

Clock signals (pins 15, 17, & 24) are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

Note: Transmit and receive leads (2 or 3) can be reversed depending on the use of the equipment - DCE Data Communications Equipment or a DTE Data Terminal Equipment.

Glossary of Abbreviations etc.

CTS	**Clear To Send [DCE --> DTE]**
DCD	**Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
DCE	**Data Communications Equipment eg. modem**
DSR	**Data Set Ready [DCE --> DTE]**
DSRS	**Data Signal Rate Selector [DCE --> DTE] (Not commonly used)**
DTE	**Data Terminal Equipment eg. computer, printer**
DTR	**Data Terminal Ready [DTE --> DCE]**
FG	**Frame Ground (screen or chassis)**
NC	**No Connection**
RCk	**Receiver (external) Clock input**
RI	**Ring Indicator (ringing tone detected)**
RTS	**Ready To Send [DTE --> DCE]**
RxD	**Received Data [DCE --> DTE]**
SG	**Signal Ground**
SCTS	**Secondary Clear To Send [DCE --> DTE]**
SDCD	**Secondary Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
SRTS	**Secondary Ready To Send [DTE --> DCE]**
SRxD	**Secondary Received Data [DCE --> DTE]**
STxD	**Secondary Transmitted Data [DTE --> DTE]**
TxD	**Transmitted Data [DTE --> DTE]**

Is Your Interface a DTE or a DCE?

Find out by following these steps: The point of reference for all signals is the terminal (or PC).

1) Measure the DC voltages between (DB25) pins 2 & 7 and between pins 3 & 7. Be sure the black lead is connected to pin 7 (Signal Ground) and the red lead to whichever pin you are measuring.

2) If the voltage on pin 2 (TD) is more negative than -3 Volts, then it is a DTE, otherwise it should be near zero volts.

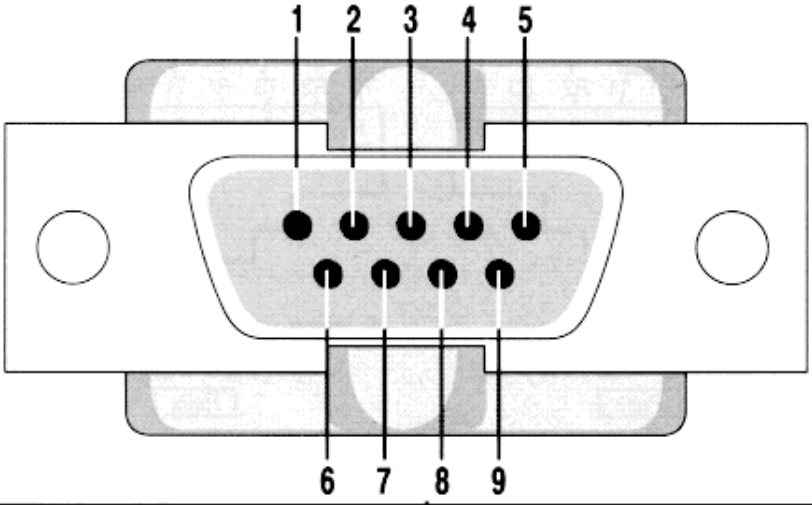
3) If the voltage on pin 3 (RD) is more negative than -3 Volts, then it is a DCE.

4) If both pins 2 & 3 have a voltage of at least 3 volts, then either you are measuring incorrectly, or your device is not a standard EIA-232 device. Call technical support.

5) In general, a DTE provides a voltage on TD, RTS, & DTR, whereas a DCE provides voltage on RD, CTS, DSR, & CD.

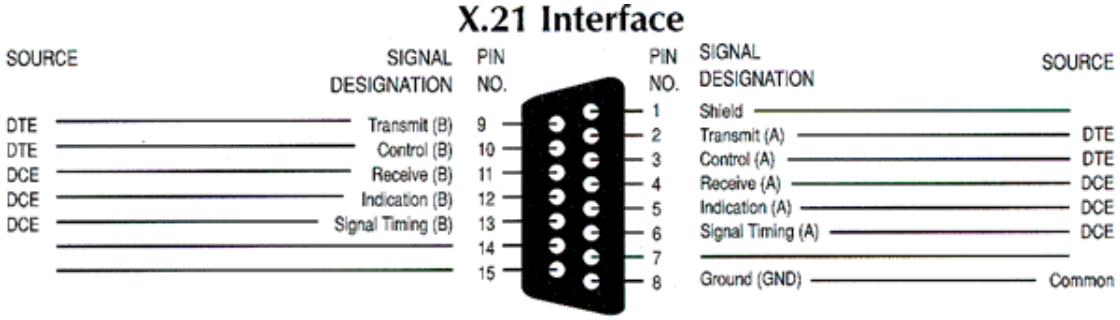
PC Com Port - EIA-574
RS-232/V.24 pin out on a DB-9 pin

used for Asynchronous Data



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

X.21 interface on a DB 15 connector



also see X.21 write up
also see end of page for more info

X.21

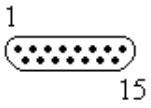
General

Voltages:	+/- 0.3Vdc
Speeds:	Max. 100Kbps (X.26)
	Max. 10Mbps (X.27)

The X.21 interface was recommended by the CCITT in 1976. It is defined as a digital signalling interface between customers (DTE) equipment and carrier's equipment (DCE). And thus primarally used for telecom equipment.

All signals are balanced. Meaning there is always a pair (+/-) for each signal, like used in RS422. The X.21 signals are the same as RS422, so please refer to RS422 for the exact details.

Pinning according to ISO 4903

	
Sub-D15 Male	Sub-D15 Female

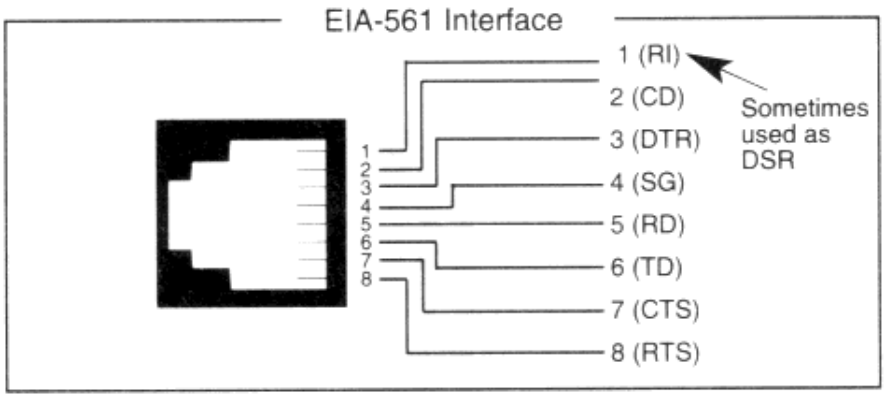
Pin	Signal	abbr.	DTE	DCE
1	Shield		-	-
2	Transmit (A)		Out	In
3	Control (A)		Out	In
4	Receive (A)		In	Out
5	Indication (A)		In	Out
6	Signal Timing (A)		In	Out
7	Unassigned			
8	Ground		-	-
9	Transmit (B)		Out	In
10	Control (B)		Out	In
11	Receive (B)		In	Out
12	Indication (B)		In	Out
13	Signal Timing (B)		In	Out
14	Unassigned			
15	Unassigned			

Functional Description
As can be seen from the pinning specifications, the Signal Element Timing (clock) is provided by the DCE. This means that your provider (local telco office) is responsible for the correct clocking and that X.21 is a synchronous interface. Hardware handshaking is done by the Control and Indication lines. The Control is used by the DTE and the Indication is the DCE one.

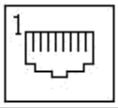
Cross-cable pinning

X.21 Cross Cable	
X.21	**X.21**
1	1
2	4
3	5
4	2
5	3
6	7
7	6
8	8
9	11
10	12
11	9
12	10
13	14
14	13
15	

EIA-561 defines RS-232 on a modular connector. (For nonsynchronous applications only, since it does not provide for the synchronous clocking signals.)



RS232D uses RJ45 type connectors (similar to telephone connectors)

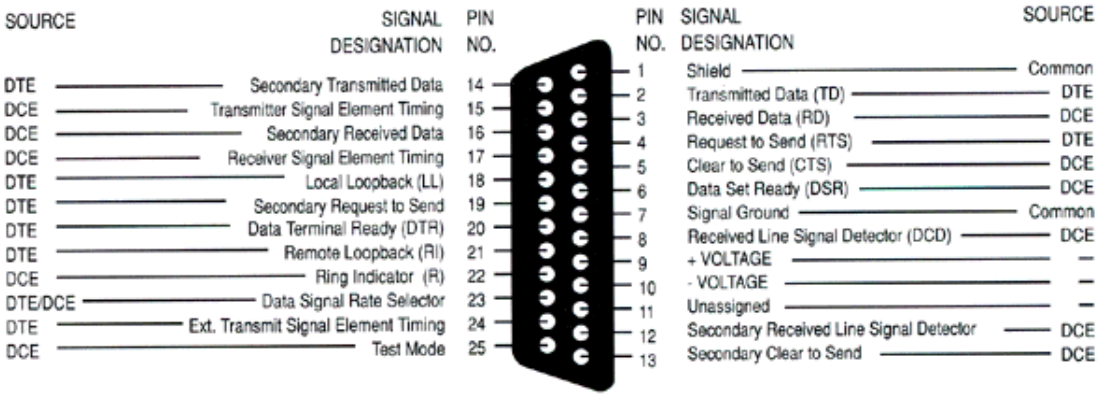
	Male	Female
RJ45		

Pin No.	Signal Description	Abbr.	DTE	DCE
1	DCE Ready, Ring Indicator	DSR/RI	←	→
2	Received Line Signal Detector	DCD	←	→
3	DTE Ready	DTR	→	←
4	Signal Ground	SG		
5	Received Data	RxD	←	→
6	Transmitted Data	TxD	→	←
7	Clear To Send	CTS	←	→
8	Request To Send	RTS	→	←

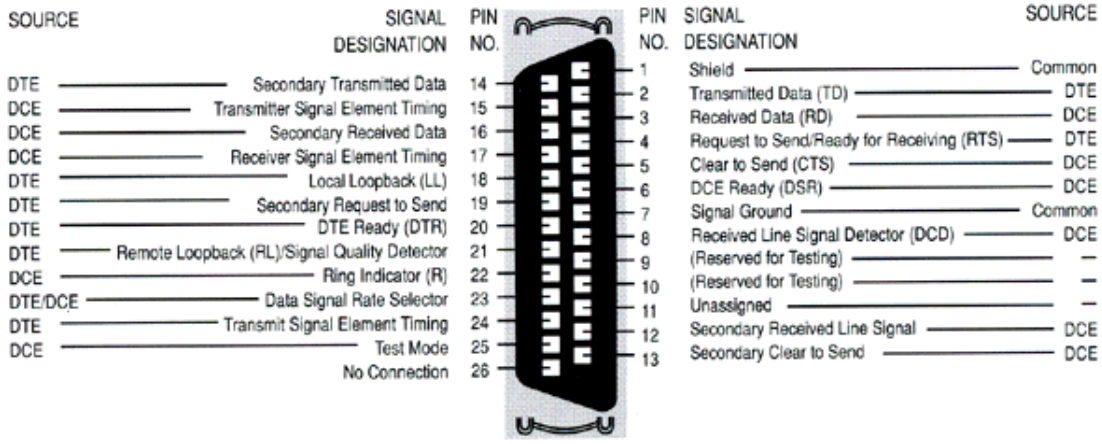
This is a standard 9 to 25 pin cable layout for async data on a PC AT serial cable

Description	Signal	9-pin DTE	25-pin DCE	Source DTE or DCE
Carrier Detect	CD	1	8	from Modem
Receive Data	RD	2	3	from Modem
Transmit Data	TD	3	2	from Terminal/Computer
Data Terminal Ready	DTR	4	20	from Terminal/Computer
Signal Ground	SG	5	7	from Modem
Data Set Ready	DSR	6	6	from Modem
Request to Send	RTS	7	4	from Terminal/Computer
Clear to Send	CTS	8	5	from Modem
Ring Indicator	RI	9	22	from Modem

V.24/RS-232 Interface



V.24/RS-232E ALT A Connector



---

25 pin D-shell connector RS232

commonly used for Async. data

PIN SIGNAL DESCRIPTION

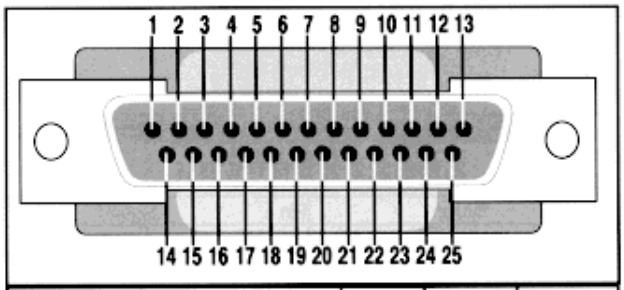
1 PGND Protective Ground
2 TXD Transmit Data
3 RXD Receive Data
4 RTS Ready To Send
5 CTS Clear To Send

6 DSR Data Set Ready
7 SG Signal Ground
8 CD Carrier Detect
20 DTR Data Terminal Ready
22 RI Ring Indicator

Some applications require more
than a simple async. configurat

RS-232 Interface

RS-232 (EIA Std.) applicable to the 25 pin interconnection of Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) using serial binary data

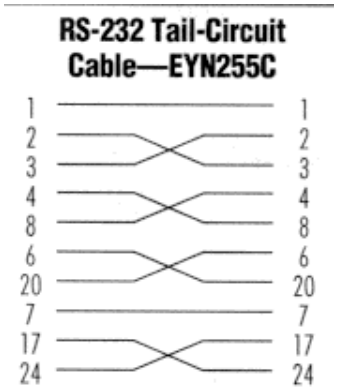


Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CI	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

Pins used for Synchronous data

jump to Other Connector pages

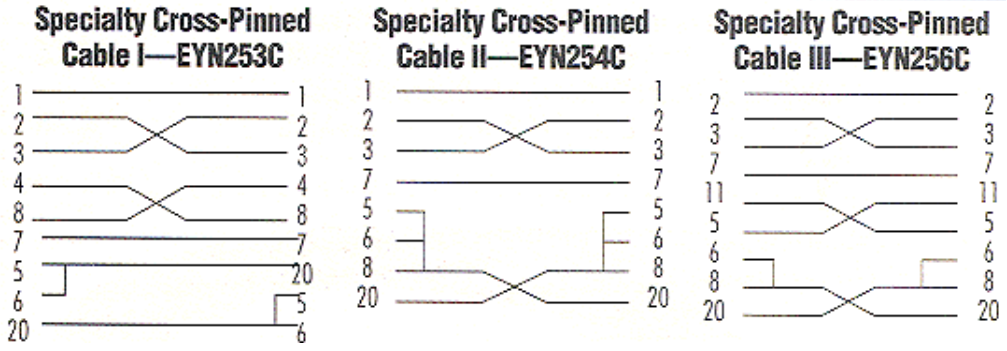
RS232 (25 pin) Tail Circuit Cable



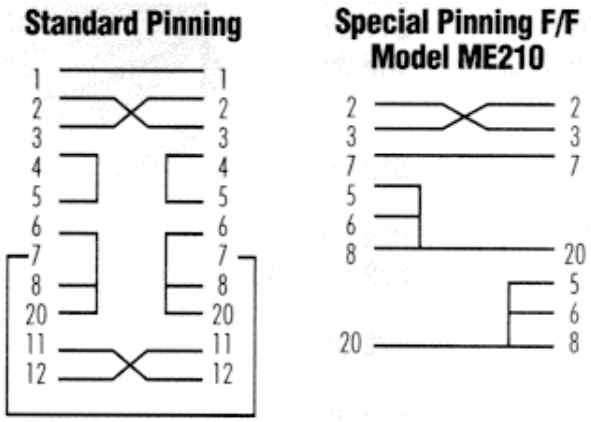
Null Modem cable diagrams

- Nullmodem (9p to 9p)
- Nullmodem (9p to 25p)
- Nullmodem (25p to 25p)

Cross Pinned cables for Async data.



Pin out for local Async Data transfer



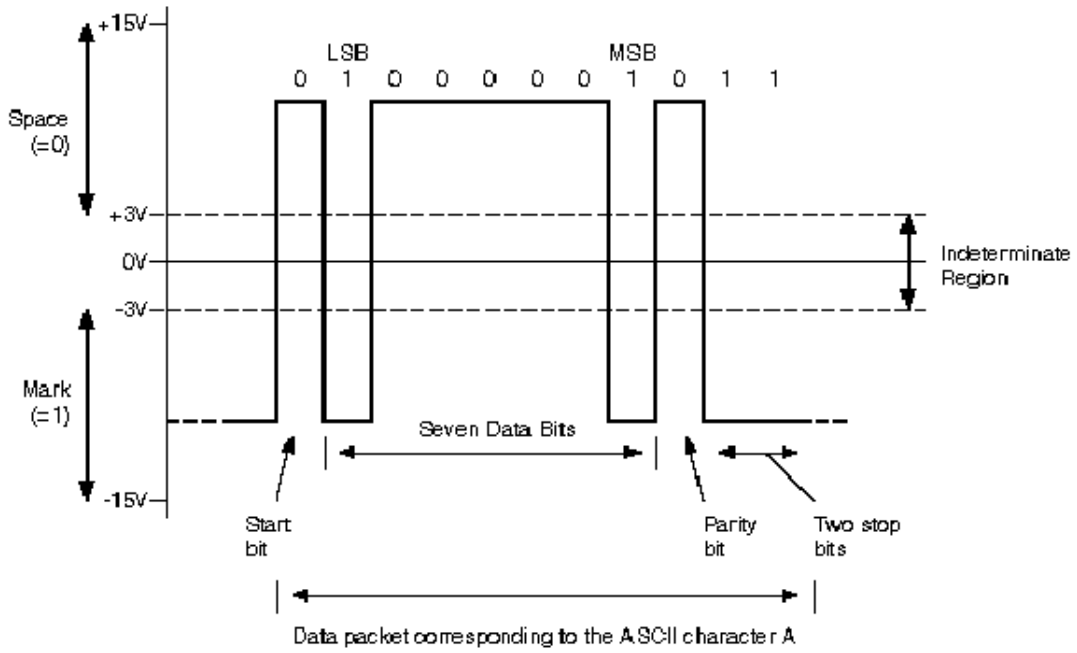
Loopback plugs:

- Serial Port Loopback (9p)
- Serial Port Loopback (25p)

RS-232 Specs.

SPECIFICATIONS		RS232	RS423
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR
Maximum Cable Length		50 FT.	4000 FT.
Maximum Data Rate		20kb/s	100kb/s
Maximum Driver Output Voltage		+/-25V	+/-6V
Driver Output Signal Level (Loaded Min.)	**Loaded**	+/-5V to +/-15V	+/-3.6V
Driver Output Signal Level (Unloaded Max)	**Unloaded**	+/-25V	+/-6V
Driver Load Impedance (Ohms)		3k to 7k	>=450
Max. Driver Current in High Z State	**Power On**	N/A	N/A
Max. Driver Current in High Z State	**Power Off**	+/-6mA @ +/-2v	+/-100uA
Slew Rate (Max.)		30V/uS	Adjustable
Receiver Input Voltage Range		+/-15V	+/-12V
Receiver Input Sensitivity		+/-3V	+/-200mV
Receiver Input Resistance (Ohms)		3k to 7k	4k min.

One byte of async data



Cabling considerations - you should use cabling made for RS-232 data but I have seen low speed data go over 250’ on 2 pair phone cable. Level 5 cable can also be used but for best distance use a low capacitance data grade cable.

The standard maxim length is 50’ but if data is async you can increase that distance to as much as 500’ with a good grade of cable.

The RS-232 signal on a single cable is impossible to screen effectively for noise. By screening the entire cable we can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the different lines introduces serious crosstalk (this especially true on synchronous data - because of the clock lines) until a point is reached where the data itself is unreadable. Signal Crosstalk can be reduced by using low capacitance cable and shielding each pair

Using a high grade cable (individually shield low capacitance pairs) the distance can be extended to 4000’

At higher frequencies a new problem comes to light. The high frequency component of the data signal is lost as the cable gets longer resulting in a rounded, rather than square wave signal.

The maxim distance will depend on the speed and noise level around the cable run.

On longer runs a line driver is needed. This is a simple modem used to increase the maxim distance you can run RS-232 data.

Making sense of the specifications

Selecting data cable isn't difficult, but often gets lost in the shuffle of larger system issues. Care should be taken. however, because intermittent problems caused by marginal cable can be very difficult to troubleshoot.

Beyond the obvious traits such as number of conductors and wire gauge, cable specifications include a handful of less intuitive terms.

Characteristic Impedance (Ohms): A value based on the inherent conductance, resistance, capacitance and inductance of a cable that represents the impedance of an infinitely long cable. When the cable is out to any length and terminated with this Characteristic Impedance, measurements of the cable will be identical to values obtained from the infinite length cable. That is to say that the termination of the cable with this impedance gives the cable the appearance of being infinite length, allowing no reflections of the transmitted signal. If termination is required in a system, the termination impedance value should match the Characteristic Impedance of the cable.

Shunt Capacitance (pF/ft): The amount of equivalent capacitive load of the cable, typically listed in a per foot basis One of the factors limiting total cable length is the capacitive load. Systems with long lengths benefits from using low capacitance cable.

Propagation velocity (% of c): The speed at which an electrical signal travels in the cable. The value given typically must be multiplied by the speed of light (c) to obtain units of meters per second. For example, a cable that lists a propagation velocity of 78% gives a velocity of 0.78 X 300 X 106 - 234 X 106 meters per second.

Plenum cable

Plenum rated cable is fire resistant and less toxic when burning than non-plenum rated cable. Check building and fire codes for requirements. Plenum cable is generally more expensive due to the sheathing material used.

The specification recommends 24AWG twisted pair cable with a shunt capacitance of 16 pF per foot and 100 ohm characteristic impedance.

It can be difficult to qualify whether shielding is required in a particular system or not, until problems arise. We recommend erring on the safe side and using shielded cable. Shielded cable is

only slightly more expensive than unshielded.

There are many cables available meeting the recommendations of RS-422 and RS-485, made specifically for that application. Another choice is the same cable commonly used in the Twisted pair Ethernet cabling. This cable, commonly referred to as Category 5 cable, is defined by the ElA/TIA/ANSI 568 specification The extremely high volume of Category 5 cable used makes it widely available and very inexpensive, often less than half the price of specialty RS422/485 cabling. The cable has a maximum capacitance of 17 pF/ft (14.5 pF typical) and characteristic impedance of 100 ohms.

Category 5 cable is available as shielded twisted pair (STP) as well as unshielded twisted pair (UTP) and generally exceeds the recommendations making it an excellent choice for RS232 systems.

RS232 - V.24/V.28 - IS2110 - X.20 bis (for Async) - X.21 bis (for Sync)

General

In this document the term RS232 will be used when refered to this serial interface. The description of RS232 is an EIA/TIA norm and is identical to CCITT V.24/V.28, X.20bis/X.21bis and ISO IS2110. The only difference is that CCITT has split the interface into its electrical description (V.28) and a mechanical part (V.24) or Asynchronous (X.20 bis) and Synchronous (X.21 bis) where the EIA/TIA describes everything under RS232.

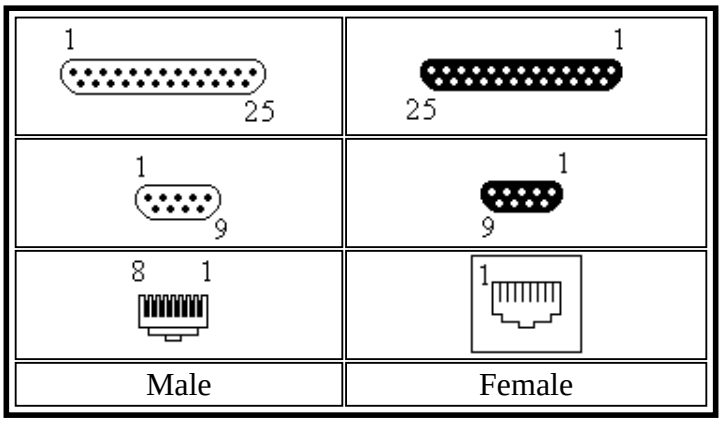
As said before RS232 is a serial interface. It can be found in many different applications where the most common ones are modems and Personal Computers. All pinning specifications are writen for the DTE side.

All DTE-DCE cables are straight through meaning the pins are connected one on one. DTE-DTE and DCE-DCE cables are cross cables. To make a destiction between all different types of cables we have to use a naming convention.
DTE - DCE: Straight Cable
DTE - DTE: Null-Modem Cable
DCE - DCE: Tail Circuit Cable

Interface Mechanical

RS232 can be found on different connectors. There are special specifications for this. The CCITT only defines a Sub-D 25 pins version where the EIA/TIA has two versions RS232C and RS232D which are resp. on a Sub-D25 and a RJ45. Next to this IBM has added a Sub-D 9 version which is found an almost

all Personal Computers and is described in TIA 457.



Pinning

RS232-C	Description	Circuit EIA	Circuit CCITT	RJ45	TIA 457
1	Shield Ground	AA			
7	Signal Ground	AB	102	4	5
2	Transmitted Data	BA	103	6	3
3	Received Data	BB	104	5	2
4	Request To Send	CA	105	8	7
5	Clear To Send	CB	106	7	8
6	DCE Ready	CC	107	1	6
20	DTE Ready	CD	108.2	3	4
22	Ring Indicator	CE	125	1	9
8	Received Line Signal Detector	CF	109	2	1
23	Data Signal Rate Select (DTE/DCE Source>	CH/CI	111/112		
24	Transmit Signal Element Timing (DTE Source)	DA	113		
15	Transmitter Signal Element Timing (DCE Source)	DB	114		
17	Receiver Signal Element Timing (DCE Source)	DD	115		
18	Local Loopback / Quality Detector	LL	141		
21	Remote Loopback	RL/CG	140/110		
14	Secondary Transmitted Data	SBA	118		
16	Secondary Received Data	SBB	119		
19	Secondary Request To Send	SCA	120		
13	Secondary Clear To Send	SCB	121		
12	Secondary Received Line Signal Detector/ Data signal Rate Select (DCE Source)	SCF/CI	122/112		
25	Test Mode	TM	142		
9	Reserved for Testing				
10	Reserved for Testing				
11	Unassigned				

Interface Electrical

All signals are measured in reference to a common ground, which is called the signal ground (AB). A positive voltage between 3 and 15 Vdc represents a logical 0 and a negative voltage between 3 and 15 Vdc represents a logical 1.
This switching between positive and negative is called bipolar. The zero state is not defined in RS232

and is considered a fault condition (this happens when a device is turned off).
According to the above a maximum distance of 50 ft or 15 m. can be reached at a maximum speed of 20k bps. This is according to the official specifications, the distance can be exceeded with the use of Line Drivers.

Functional description

Description	Circuit	Function
Shield Ground	AA	Also known as protective ground. This is the chassis ground connection between DTE and DCE.
Signal Ground	AB	The reference ground between a DTE and a DCE. Has the value 0 Vdc.
Transmitted Data	BA	Data send by the DTE.
Received Data	BB	Data received by the DTE.
Request To Send	CA	Originated by the DTE to initiate transmission by the DCE.
Clear To Send	CB	Send by the DCE as a reply on the RTS after a delay in ms, which gives the DCEs enough time to energize their circuits and synchronize on basic modulation patterns.
DCE Ready	CC	Known as DSR. Originated by the DCE indicating that it is basically operating (power on, and in functional mode).
DTE Ready	CD	Known as DTR. Originated by the DTE to instruct the DCE to setup a connection. Actually it means that the DTE is up and running and ready to communicate.
Ring Indicator	CE	A signal from the DCE to the DTE that there is an incomming call (telephone is ringing). Only used on switched circuit connections.
Received Line Signal Detector	CF	Known as DCD. A signal send from DCE to its DTE to indicate that it has received a basic carrier signal from a (remote) DCE.
Data Signal Rate Select (DTE/DCE Source>	CH/CI	A control signal that can be used to change the transmission speed.
Transmit Signal Element Timing (DTE Source)	DA	Timing signals used by the DTE for transmission, where the clock is originated by the DTE and the DCE is the slave.
Transmitter Signal Element Timing (DCE Source)	DB	Timing signals used by the DTE for transmission.
Receiver Signal Element Timing (DCE Source)	DD	Timing signals used by the DTE when receiving data.
Local Loopback / Quality Detector	LL	
Remote Loopback	RL/CG	Originated by the DCE that changes state when the analog signal received from the (remote) DCE becomes marginal.

Test Mode	TM	
Reserved for Testing		

The secondary signals are used on some DCE’s. Those units have the possibility to transmit and/or receive on a secondary channel. Those secondary channels are mostly of a lower speed than the normal ones and are mainly used for administrative functions.

Cable pinning
Here are some cable pinning that might be useful. Not all applications are covered, it is just a help:

Straight DB25 Cable

Pin	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25

DB25 Null- modem cable (Async)

Pin	Pin
1	1
2	3
3	2
4	5
5	4
6, 8	20
7	7
20	6, 8

</

DB9 Null- modem cable

1,6	4
2	3
3	2
4	1,6
5	5
7	8
8	7

DB25 Tail- circuit cable (Sync)

Pin	Pin
1	1
2	3
3	2
4	8
6	20
7	7
8	4
17	24
20	6
24	17

DB25 to DB9 DTE - DCE cable

jump to **related fiber cable pages**

---

jump to The Belden Cable Company's cable selection tutorial pages

jump to Data Communication by CAMI Research good write up

jump to RS-232 by CAMI Research good write up

jump to Interfacing the Serial / RS232 Port good write up
(in-depth very technical)

jump to Data Modems for phone lines

jump to Data Modems for fiber optics

jump to Interface converters

ARC Electronics ...
800-926-0226
Home Page

arc@arcelect.com

PC serial port buffer

Summary of circuit features

- Brief description of operation: Buffer to run RS-232 data to longer distanced as normally
- Circuit protection: No special protection circuits used
- Circuit complexity: Very simple two transistor buffer circuit
- Circuit performance: Worked nicely in one special application, doubled the line throughput
- Availability of components: Widely available components at the time when the circuit was built
- Design testing: Circuit was in constatant use by my friend for over a year
- Applications: Maximizing RS-232 line throughput on long cable runs
- Power supply: +-12V DC power supply 80 mA
- Estimated component cost: Few dollars
- Safety considerations: No special safety considerations

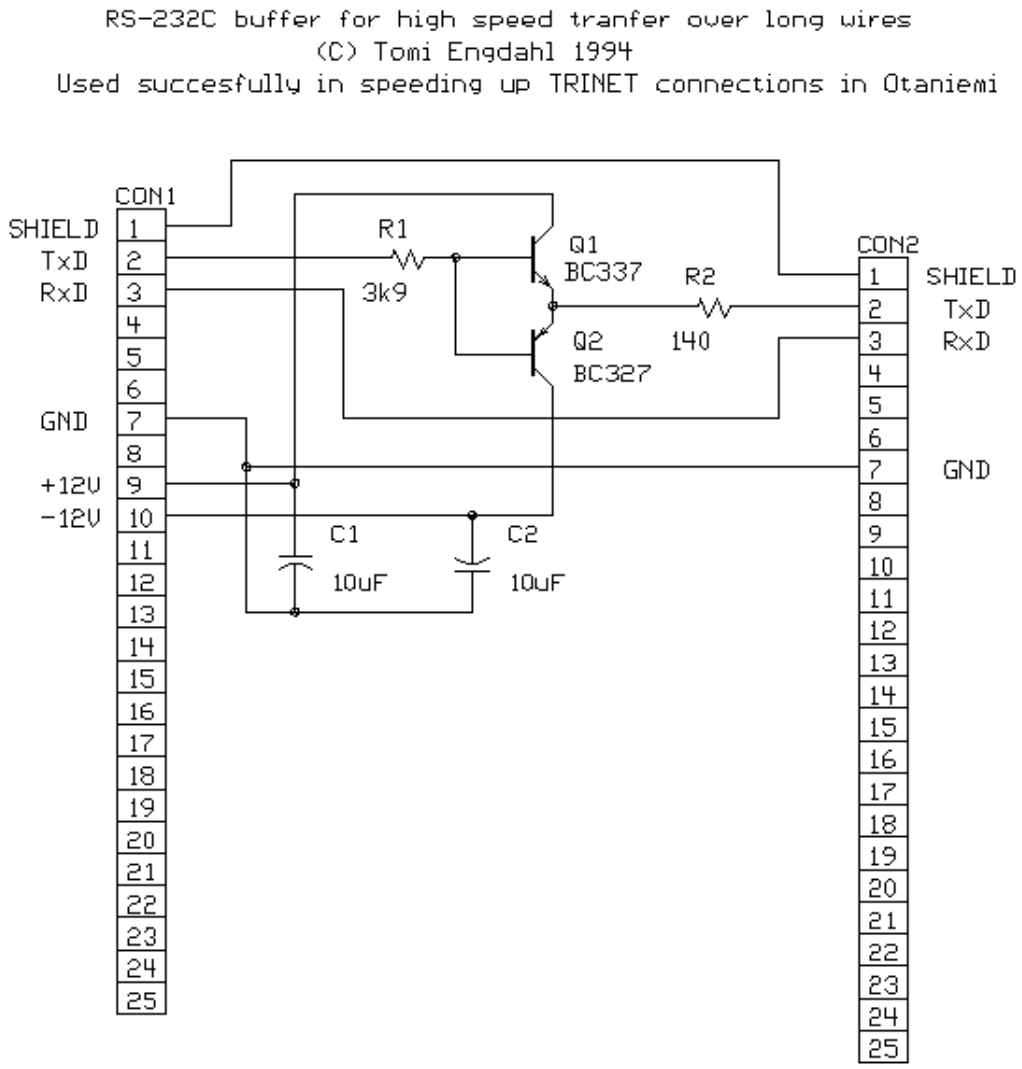
Circuit description

This is a simple serial port buffer circuit I designed for a friend to speed up his SLIP connection in campus computer network "TRINET" of Helsinki University of Technology. The problem in the network was that the RS232 commections from rooms to terminal server were long and made of bad quality wiring.

The circuit is a simple buffer which adds more driving capacity to PC serial port for the signal to go succesfully from PC computer to terminal server (other direction had no problems). The computer is connected to connector CON1 and the buffered output is available ar CON2. With this circuit the speed of RS232 connection to termial server could be succesfully raised for 9600 bps to 38400 bps.

The circuit is basically a two transistor buffer consistong of transistors Q1 and Q2 which can drive up to 1A current pulses, but the maximum putput current of the circuit is limited by resistor R2. Value R2 was experimentally selected by testing resistor values in range of 22 ohm to 270 ohm and value 140 ohm gave best results (it provides quite good impedance matching to cable used). It is a good idea to use at least 1W resistor in place of R2 to make sure that it does not overheat in output short circuit situation (RS232 devices must withstand that to meet the standard).

The circuit was designed to be a compact box which is powered through D25 connector as some commercial RS232 buffer circuits. The idea is to feed the power to the buffer unit through serial port voltage test pins 9 and 10. The power was taken from an external power supply (cheap universal wall transformer) and wired to the D25 connector by modifying the cable connected between computer and the buffer circuit. The circuit in this configuration takes maximally continuous current of about 100 mA.



Tomi Engdahl <then@delta.hut.fi>

Interfacing the Serial / RS232 Port

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

So what are the advantages of using serial data transfer rather than parallel?

1. Serial Cables can be longer than Parallel cables. The serial port transmits a '1' as -3 to -25 volts and a '0' as +3 to +25 volts where as a parallel port transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5 Volts. Therefore cable loss is not going to be as much of a problem for serial cables than they are for parallel.
2. You don't need as many wires than parallel transmission. If your device needs to be mounted a far distance away from the computer then 3 core cable (Null Modem Configuration) is going to be a lot cheaper that running 19 or 25 core cable. However you must take into account the cost of the interfacing at each end.
3. Infra Red devices have proven quite popular recently. You may of seen many electronic diaries and palmtop computers which have infra red capabilities build in. However could you imagine transmitting 8 bits of data at the one time across the room and being able to (from the devices point of view) decipher which bits are which? Therefore serial transmission is used where one bit is sent at a time. IrDA-1 (The first infra red specifications) was capable of 115.2k baud and was interfaced into a UART. The pulse length however was cut down to 3/16th of a RS232 bit length to conserve power considering these devices are mainly used on diaries, laptops and palmtops.
4. Microcontroller's have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

Table of Contents

Part One : Hardware (PC's)

Hardware Properties

Devices which use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) Data Communications Equipment are devices such as your modem, TA adapter, plotter etc while Data Terminal Equipment is your Computer or Terminal.

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as -

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.
3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!)

Above is no where near a complete list of the EIA standard. Line Capacitance, Maximum Baud Rates etc are also included. For more information please consult the EIA RS232-C standard. It is interesting to note however, that the RS232C standard specifies a maximum baud rate of 20,000 BPS!, which is rather slow by today's standards. A new standard, RS-232D has been recently released.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. Below is a table of pin connections for the 9 pin and 25 pin D-Type connectors.

Serial Pinouts (D25 and D9 Connectors)

D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

Table 1 : D Type 9 Pin and D Type 25 Pin Connectors

Pin Functions

Abbreviation	Full Name	Function
TD	Transmit Data	Serial Data Output (TXD)
RD	Receive Data	Serial Data Input (RXD)
CTS	Clear to Send	This line indicates that the Modem is ready to exchange data.
DCD	Data Carrier Detect	When the modem detects a "Carrier" from the modem at the other end of the phone line, this Line becomes active.
DSR	Data Set Ready	This tells the UART that the modem is ready to establish a link.
DTR	Data Terminal Ready	This is the opposite to DSR. This tells the Modem that the UART is ready to link.
RTS	Request To Send	This line informs the Modem that the UART is ready to exchange data.
RI	Ring Indicator	Goes active when modem detects a ringing signal from the PSTN.

Null Modems

A Null Modem is used to connect two DTE's together. This is commonly used as a cheap way to network games or to transfer files between computers using Zmodem Protocol, Xmodem Protocol etc. This can also be used with many Microprocessor Development Systems.

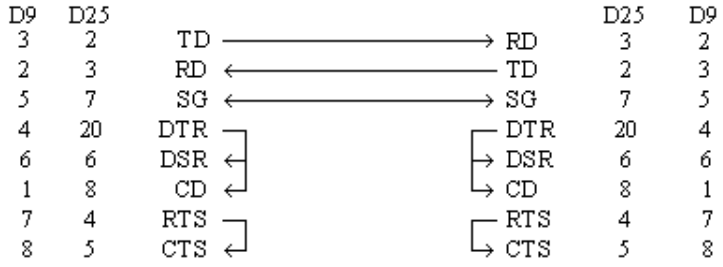


Figure 1 : Null Modem Wiring Diagram

Above is my preferred method of wiring a Null Modem. It only requires 3 wires (TD, RD & SG) to be wired straight through thus is more cost effective to use with long cable runs. The theory of operation is reasonably easy. The aim is to make to computer think it is talking to a modem rather than another computer. Any data transmitted from the first computer must be received by the second thus TD is connected to RD. The second computer must have the same set-up thus RD is connected to TD. Signal Ground (SG) must also be connected so both grounds are common to each computer.

The Data Terminal Ready is looped back to Data Set Ready and Carrier Detect on both computers. When the Data Terminal Ready is asserted active, then the Data Set Ready and Carrier Detect immediately become active. At this point the computer thinks the Virtual Modem to which it is connected is ready and has detected the carrier of the other modem.

All left to worry about now is the Request to Send and Clear To Send. As both computers communicate together at the same speed, flow control is not needed thus these two lines are also linked together on each computer. When the computer wishes to send data, it asserts the Request to Send high and as it's hooked together with the Clear to Send, It immediately gets a reply that it is ok to send and does so.

Notice that the ring indicator is not connected to anything of each end. This line is only used to tell the computer that there is a ringing signal on the phone line. As we don't have a modem connected to the phone line this is left disconnected.

LoopBack Plug

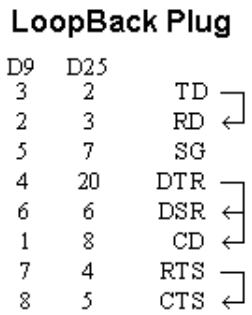


Figure 2 : Loopback Plug Wiring Diagram

This loopback plug can come in extremely handy when writing Serial / RS232 Communications Programs. It has the receive and transmit lines connected together, so that anything transmitted out of the Serial Port is immediately received by the same port. If you connect this to a Serial Port an load a Terminal Program, anything you type will be immediately displayed on the screen. This can be used with the examples later in this tutorial.

Please note that this is not intended for use with Diagnostic Programs and thus will probably not work. For these programs you require a differently wired Loop Back plug which may vary from program to program.

DTE / DCE Speeds

We have already talked briefly about DTE & DCE. A typical Data Terminal Device is a computer and a typical Data Communications Device is a Modem. Often people will talk about DTE to DCE or DCE to DCE speeds. DTE to DCE is the speed between your modem and computer, sometimes referred to as your terminal speed. This should run at faster speeds than the DCE to DCE speed. DCE to DCE is the link between modems, sometimes called the line speed.

Most people today will have 28.8K or 33.6K modems. Therefore we should expect the DCE to DCE speed to be either 28.8K or 33.6K. Considering the high speed of the modem we should expect the DTE to DCE speed to be about 115,200 BPS.(Maximum Speed of the 16550a UART) This is where some people often fall into a trap. The communications program which they use have settings for DCE to DTE speeds. However they see 9.6 KBPS, 14.4 KBPS etc and think it

is your modem speed.

Today's Modems should have Data Compression build into them. This is very much like PK-ZIP but the software in your modem compresses and decompresses the data. When set up correctly you can expect compression ratios of 1:4 or even higher. 1 to 4 compression would be typical of a text file. If we were transferring that text file at 28.8K (DCE-DCE), then when the modem compresses it you are actually transferring 115.2 KBPS between computers and thus have a DCE-DTE speed of 115.2 KBPS. Thus this is why the DCE-DTE should be much higher than your modem's connection speed.

Some modem manufacturers quote a maximum compression ratio as 1:8. Lets say for example its on a new 33.6 KBPS modem then we may get a maximum 268,800 BPS transfer between modem and UART. If you only have a 16550a which can do 115,200 BPS tops, then you would be missing out on a extra bit of performance. Buying a 16C650 should fix your problem with a maximum transfer rate of 230,400 BPS.

However don't abuse your modem if you don't get these rates. These are MAXIMUM compression ratios. In some instances if you try to send a already compressed file, your modem can spend more time trying the compress it, thus you get a transmission speed less than your modem's connection speed. If this occurs try turning off your data compression. This should be fixed on newer modems. Some files compress easier than others thus any file which compresses easier is naturally going to have a higher compression ratio.

Flow Control

So if our DTE to DCE speed is several times faster than our DCE to DCE speed the PC can send data to your modem at 115,200 BPS. Sooner or later data is going to get lost as buffers overflow, thus flow control is used. Flow control has two basic varieties, Hardware or Software.

Software flow control, sometimes expressed as Xon/Xoff uses two characters Xon and Xoff. Xon is normally indicated by the ASCII 17 character where as the ASCII 19 character is used for Xoff. The modem will only have a small buffer so when the computer fills it up the modem sends a Xoff character to tell the computer to stop sending data. Once the modem has room for more data it then sends a Xon character and the computer sends more data. This type of flow control has the advantage that it doesn't require any more wires as the characters are sent via the TD/RD lines. However on slow links each character requires 10 bits which can slow communications down.

Hardware flow control is also known as RTS/CTS flow control. It uses two wires in your serial cable rather than extra characters transmitted in your data lines. Thus hardware flow control will not slow down transmission times like Xon-Xoff does. When the computer wishes to send data it takes active the Request to Send line. If the modem has room for this data, then the modem will reply by taking active the Clear to Send line and the computer starts sending data. If the modem does not have the room then it will not send a Clear to Send.

The UART (8250 and Compatibles)

UART stands for Universal Asynchronous Receiver / Transmitter. Its the little box of tricks found on your serial card which plays the little games with your modem or other connected devices. Most cards will have the UART's integrated into other chips which may also control your parallel port, games port, floppy or hard disk drives and are typically surface mount devices. The 8250 series, which includes the 16450, 16550, 16650, & 16750 UARTS are the most commonly found type in your PC. Later we will look at other types which can be used in your homemade devices and projects.

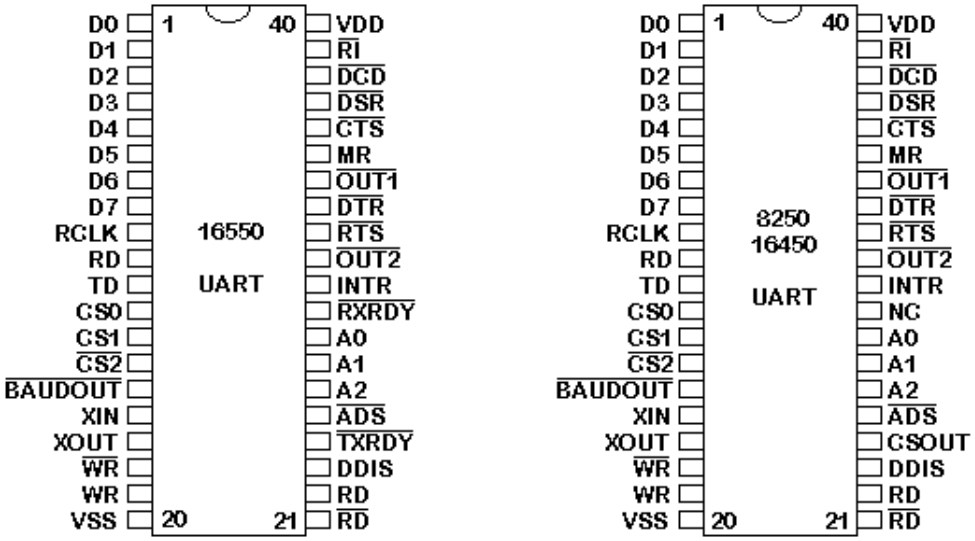


Figure 3 : Pin Diagrams for 16550, 16450 & 8250 UARTs

The 16550 is chip compatible with the 8250 & 16450. The only two differences are pins 24 & 29. On the 8250 Pin 24 was chip select out which functioned only as a indicator to if the chip was active or not. Pin 29 was not connected on the 8250/16450 UARTs. The 16550 introduced two new pins in their place. These are Transmit Ready and Receive Ready which can be implemented with DMA (Direct Memory Access). These Pins have two different modes of operation. Mode 0 supports single transfer DMA where as Mode 1 supports Multi-transfer DMA.

Mode 0 is also called the 16450 mode. This mode is selected when the FIFO buffers are disabled via Bit 0 of the FIFO Control Register or When the FIFO buffers are enabled but DMA Mode Select = 0. (Bit 3 of FCR) In this mode RXRDY is active low when at least one character (Byte) is present in the Receiver Buffer. RXRDY will go inactive high when no more characters are left in the Receiver Buffer. TXRDY will be active low when there are no characters in the Transmit Buffer. It will go inactive high after the first character / byte is loaded into the Transmit Buffer.

Mode 1 is when the FIFO buffers are active and the DMA Mode Select = 1. In Mode 1, RXRDY will go active low when the trigger level is reached or when 16550 Time Out occurs and will return to inactive state when no more characters are left in the FIFO. TXRDY will be active when no characters are present in the Transmit Buffer and will go inactive when the FIFO Transmit Buffer is completely Full.

All the UARTs pins are TTL compatible. That includes TD, RD, RI, DCD, DSR, CTS, DTR and

RTS which all interface into your serial plug, typically a D-type connector. Therefore RS232 Level Converters (which we talk about in detail later) are used. These are commonly the DS1489 Receiver and the DS1488 as the PC has +12 and -12 volt rails which can be used by these devices. The RS232 Converters will convert the TTL signal into RS232 Logic Levels.

Pin No.	Name	Notes
Pin 1:8	D0:D7	Data Bus
Pin 9	RCLK	Receiver Clock Input. The frequency of this input should equal the receivers baud rate * 16
Pin 10	RD	Receive Data
Pin 11	TD	Transmit Data
Pin 12	CS0	Chip Select 0 - Active High
Pin 13	CS1	Chip Select 1 - Active High
Pin 14	nCS2	Chip Select 2 - Active Low
Pin 15	nBAUDOUT	Baud Output - Output from Programmable Baud Rate Generator. Frequency = (Baud Rate x 16)
Pin 16	XIN	External Crystal Input - Used for Baud Rate Generator Oscillator
Pin 17	XOUT	External Crystal Output
Pin 18	nWR	Write Line - Inverted
Pin 19	WR	Write Line - Not Inverted
Pin 20	VSS	Connected to Common Ground
Pin 21	RD	Read Line - Inverted
Pin 22	nRD	Read Line - Not Inverted
Pin 23	DDIS	Driver Disable. This pin goes low when CPU is reading from UART. Can be connected to Bus Transceiver in case of high capacity data bus.
Pin 24	nTXRDY	Transmit Ready
Pin 25	nADS	Address Strobe. Used if signals are not stable during read or write cycle
Pin 26	A2	Address Bit 2
Pin 27	A1	Address Bit 1
Pin 28	A0	Address Bit 0
Pin 29	nRXRDY	Receive Ready
Pin 30	INTR	Interrupt Output
Pin 31	nOUT2	User Output 2
Pin 32	nRTS	Request to Send
Pin 33	nDTR	Data Terminal Ready

Pin 34	nOUT1	User Output 1
Pin 35	MR	Master Reset
Pin 36	nCTS	Clear To Send
Pin 37	nDSR	Data Set Ready
Pin 38	nDCD	Data Carrier Detect
Pin 39	nRI	Ring Indicator
Pin 40	VDD	+ 5 Volts

Table 2 : Pin Assignments for 16550A UART

The UART requires a Clock to run. If you look at your serial card a common crystal found is either a 1.8432 MHZ or a 18.432 MHZ Crystal. The crystal in connected to the XIN-XOUT pins of the UART using a few extra components which help the crystal to start oscillating. This clock will be used for the Programmable Baud Rate Generator which directly interfaces into the transmit timing circuits but not directly into the receiver timing circuits. For this an external connection mast be made from pin 15 (BaudOut) to pin 9 (Receiver clock in.) Note that the clock signal will be at Baudrate * 16.

If you are serious about pursuing the 16550 UART used in your PC further, then would suggest downloading a copy of the PC16550D data sheet from National Semiconductors Site. Data sheets are available in .PDF format so you will need Adobe Acrobat Reader to read these. Texas Instruments has released the 16750 UART which has 64 Byte FIFO's. Data Sheets for the TL16C750 are available from the Texas Instruments Site.

Types of UARTS (For PC's)

8250	First UART in this series. It contains no scratch register. The 8250A was an improved version of the 8250 which operates faster on the bus side.
8250A	This UART is faster than the 8250 on the bus side. Looks exactly the same to software than 16450.
8250B	Very similar to that of the 8250 UART.
16450	Used in AT's (Improved bus speed over 8250's). Operates comfortably at 38.4KBPS. Still quite common today.
16550	This was the first generation of buffered UART. It has a 16 byte buffer, however it doesn't work and is replaced with the 16550A.
16550A	Is the most common UART use for high speed communications eg 14.4K & 28.8K Modems. They made sure the FIFO buffers worked on this UART.
16650	Very recent breed of UART. Contains a 32 byte FIFO, Programmable X-On / X-Off characters and supports power management.
16750	Produced by Texas Instruments. Contains a 64 byte FIFO.

Part Two : Serial Port's Registers (PC's)

Port Addresses & IRQ’s

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Table 3 : Standard Port Addresses

Above is the standard port addresses. These should work for most P.C’s. If you just happen to be lucky enough to own a IBM P/S2 which has a micro-channel bus, then expect a different set of addresses and IRQ’s. Just like the LPT ports, the base addresses for the COM ports can be read from the BIOS Data Area.

Start Address	Function
0000:0400	COM1’s Base Address
0000:0402	COM2’s Base Address
0000:0404	COM3’s Base Address
0000:0406	COM4’s Base Address

Table 4 - COM Port Addresses in the BIOS Data Area;

The above table shows the address at which we can find the Communications (COM) ports addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your communications ports.

```
#include
#include

void main(void)
{
 unsigned int far *ptraddr;  /* Pointer to location of Port Addresses */
 unsigned int address;       /* Address of Port */
 int a;

 ptraddr=(unsigned int far *)0x00000400;

 for (a = 0; a <  4; a++)
   {
    address = *ptraddr;
    if (address == 0)
                printf("No port found for COM%d \n",a+1);
    else
                printf("Address assigned to COM%d is %Xh\n",a+1,address);
    *ptraddr++;
```

```
    }
}
```

Table of Registers

Base Address	DLAB	Read/Write	Abr.	Register Name
+ 0	=0	Write	-	Transmitter Holding Buffer
	=0	Read	-	Receiver Buffer
	=1	Read/Write	-	Divisor Latch Low Byte
+ 1	=0	Read/Write	IER	Interrupt Enable Register
	=1	Read/Write	-	Divisor Latch High Byte
+ 2	-	Read	IIR	Interrupt Identification Register
	-	Write	FCR	FIFO Control Register
+ 3	-	Read/Write	LCR	Line Control Register
+ 4	-	Read/Write	MCR	Modem Control Register
+ 5	-	Read	LSR	Line Status Register
+ 6	-	Read	MSR	Modem Status Register
+ 7	-	Read/Write	-	Scratch Register

Table 5 : Table of Registers

DLAB ?

You will have noticed in the table of registers that there is a DLAB column. When DLAB is set to ’0’ or ’1’ some of the registers change. This is how the UART is able to have 12 registers (including the scratch register) through only 8 port addresses. DLAB stands for Divisor Latch Access Bit. When DLAB is set to ’1’ via the line control register, two registers become available from which you can set your speed of communications measured in bits per second.

The UART will have a crystal which should oscillate around 1.8432 MHZ. The UART incorporates a divide by 16 counter which simply divides the incoming clock signal by 16. Assuming we had the 1.8432 MHZ clock signal, that would leave us with a maximum, 115,200 hertz signal making the UART capable of transmitting and receiving at 115,200 Bits Per Second (BPS). That would be fine for some of the faster modems and devices which can handle that speed, but others just wouldn’t communicate at all. Therefore the UART is fitted with a Programmable Baud Rate Generator which is controlled by two registers.

Lets say for example we only wanted to communicate at 2400 BPS. We worked out that we would have to divide 115,200 by 48 to get a workable 2400 Hertz Clock. The "Divisor", in this case 48, is stored in the two registers controlled by the "Divisor Latch Access Bit". This divisor can be any number which can be stored in 16 bits (ie 0 to 65535). The UART only has a 8 bit data bus, thus this is where the two registers are used. The first register (Base + 0) when DLAB = 1 stores the "Divisor latch low byte" where as the second register (base + 1 when DLAB = 1)

stores the "Divisor latch high byte."

Below is a table of some more common speeds and their divisor latch high bytes & low bytes. Note that all the divisors are shown in Hexadecimal.

Speed (BPS)	Divisor (Dec)	Divisor Latch High Byte	Divisor Latch Low Byte
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

Table 6 : Table of Commonly Used Baudrate Divisors

Interrupt Enable Register (IER)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Enables Low Power Mode (16750)
Bit 4	Enables Sleep Mode (16750)
Bit 3	Enable Modem Status Interrupt
Bit 2	Enable Receiver Line Status Interrupt
Bit 1	Enable Transmitter Holding Register Empty Interrupt
Bit 0	Enable Received Data Available Interrupt

Table 7 : Interrupt Enable Register

The Interrupt Enable Register could possibly be one of the easiest registers on a UART to understand. Setting Bit 0 high enables the Received Data Available Interrupt which generates an interrupt when the receiving register/FIFO contains data to be read by the CPU.

Bit 1 enables Transmit Holding Register Empty Interrupt. This interrupts the CPU when the transmitter buffer is empty. Bit 2 enables the receiver line status interrupt. The UART will interrupt when the receiver line status changes. Likewise for bit 3 which enables the modem status interrupt. Bits 4 to 7 are the easy ones. They are simply reserved. (If only everything was that easy!)

Interrupt Identification Register (IIR)

Bit	Notes		
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but Unusable
	1	1	FIFO Enabled
Bit 5	64 Byte Fifo Enabled (16750 only)		
Bit 4	Reserved		
Bit 3	0 Reserved on 8250, 16450		
	1 16550 Time-out Interrupt Pending		
Bits 1 and 2	Bit 2	Bit 1	
	0	0	Modem Status Interrupt
	0	1	Transmitter Holding Register Empty Interrupt
	1	0	Received Data Available Interrupt
	1	1	Receiver Line Status Interrupt
Bit 0	0 Interrupt Pending		
	1 No Interrupt Pending		

Table 8 : Interrupt Identification Register

The interrupt identification register is a read only register. Bits 6 and 7 give status on the FIFO Buffer. When both bits are ’0’ no FIFO buffers are active. This should be the only result you will get from a 8250 or 16450. If bit 7 is active but bit 6 is not active then the UART has it’s buffers enabled but are unusable. This occurs on the 16550 UART where a bug in the FIFO buffer made the FIFO’s unusable. If both bits are ’1’ then the FIFO buffers are enabled and fully operational.

Bits 4 and 5 are reserved. Bit 3 shows the status of the time-out interrupt on a 16550 or higher.

Lets jump to Bit 0 which shows whether an interrupt has occurred. If an interrupt has occurred it’s status will shown by bits 1 and 2. These interrupts work on a priority status. The Line Status Interrupt has the highest Priority, followed by the Data Available Interrupt, then the Transmit Register Empty Interrupt and then the Modem Status Interrupt which has the lowest priority.

First In / First Out Control Register (FCR)

Bit	Notes		
Bits 6 and 7	Bit 7	Bit 6	Interrupt Trigger Level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

Table 9 : FIFO Control Register

The FIFO register is a write only register. This register is used to control the FIFO (First In / First Out) buffers which are found on 16550's and higher.

Bit 0 enables the operation of the receive and transmit FIFO's. Writing a '0' to this bit will disable the operation of transmit and receive FIFO's, thus you will loose all data stored in these FIFO buffers.

Bit's 1 and 2 control the clearing of the transmit or receive FIFO's. Bit 1 is responsible for the receive buffer while bit 2 is responsible for the transmit buffer. Setting these bits to 1 will only clear the contents of the FIFO and will not affect the shift registers. These two bits are self resetting, thus you don't need to set the bits to '0' when finished.

Bit 3 enables the DMA mode select which is found on 16550 UARTs and higher. More on this later. Bits 4 and 5 are those easy type again, Reserved.

Bits 6 and 7 are used to set the triggering level on the Receive FIFO. For example if bit 7 was set to '1' and bit 6 was set to '0' then the trigger level is set to 8 bytes. When there is 8 bytes of data in the receive FIFO then the Received Data Available interrupt is set. See (IIR)

Line Control Register (LCR)

Bit 7	1 Divisor Latch Access Bit			
	0 Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register			
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0 One Stop Bit			
	1 2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.			
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Table 10 : Line Control Register

The Line Control register sets the basic parameters for communication. Bit 7 is the Divisor Latch Access Bit or DLAB for short. We have already talked about what it does. (See DLAB?) Bit 6 Sets break enable. When active, the TD line goes into "Spacing" state which causes a break in the receiving UART. Setting this bit to ’0’ Disables the Break.

Bits 3,4 and 5 select parity. If you study the 3 bits, you will find that bit 3 controls parity. That is, if it is set to ’0’ then no parity is used, but if it is set to ’1’ then parity is used. Jumping to bit 5, we can see that it controls sticky parity. Sticky parity is simply when the parity bit is always transmitted and checked as a ’1’ or ’0’. This has very little success in checking for errors as if the first 4 bits contain errors but the sticky parity bit contains the appropriately set bit, then a parity error will not result. Sticky high parity is the use of a ’1’ for the parity bit, while the opposite, sticky low parity is the use of a ’0’ for the parity bit.

If bit 5 controls sticky parity, then turning this bit off must produce normal parity provided bit 3 is still set to ’1’. Odd parity is when the parity bit is transmitted as a ’1’ or ’0’ so that there is a odd number of 1’s. Even parity must then be the parity bit produces and even number of 1’s. This provides better error checking but still is not perfect, thus CRC-32 is often used for software error correction. If one bit happens to be inverted with even or odd parity set, then a parity error will occur, however if two bits are flipped in such a way that it produces the correct parity bit then an parity error will no occur.

Bit 2 sets the length of the stop bits. Setting this bit to ’0’ will produce one stop bit, however setting it to ’1’ will produce either 1.5 or 2 stop bits depending upon the word length. Note that the receiver only checks the first stop bit.

Bits 0 and 1 set the word length. This should be pretty straight forward. A word length of 8 bits is most commonly used today.

Modem Control Register (MCR)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Autoflow Control Enabled (16750 only)
Bit 4	LoopBack Mode
Bit 3	Aux Output 2
Bit 2	Aux Output 1
Bit 1	Force Request to Send
Bit 0	Force Data Terminal Ready

Table 11 : Modem Control Register

The Modem Control Register is a Read/Write Register. Bits 5,6 and 7 are reserved. Bit 4 activates the loopback mode. In Loopback mode the transmitter serial output is placed into marking state. The receiver serial input is disconnected. The transmitter out is looped back to the receiver in. DSR, CTS, RI & DCD are disconnected. DTR, RTS, OUT1 & OUT2 are connected to the modem control inputs. The modem control output pins are then place in an inactive state. In this mode any data which is placed in the transmitter registers for output is received by the receiver circuitry on the same chip and is available at the receiver buffer. This can be used to test the UARTs operation.

Aux Output 2 maybe connected to external circuitry which controls the UART-CPU interrupt process. Aux Output 1 is normally disconnected, but on some cards is used to switch between a 1.8432MHZ crystal to a 4MHZ crystal which is used for MIDI. Bits 0 and 1 simply control their relevant data lines. For example setting bit 1 to ’1’ makes the request to send line active.

Line Status Register (LSR)

Bit	Notes
Bit 7	Error in Received FIFO
Bit 6	Empty Data Holding Registers
Bit 5	Empty Transmitter Holding Register
Bit 4	Break Interrupt
Bit 3	Framing Error
Bit 2	Parity Error
Bit 1	Overrun Error
Bit 0	Data Ready

Table 12 : Line Status Register

The line status register is a read only register. Bit 7 is the error in received FIFO bit. This bit is high when at least one break, parity or framing error has occurred on a byte which is contained in the FIFO.

When bit 6 is set, both the transmitter holding register and the shift register are empty. The UART's holding register holds the next byte of data to be sent in parallel fashion. The shift register is used to convert the byte to serial, so that it can be transmitted over one line. When bit 5 is set, only the transmitter holding register is empty. So what's the difference between the two? When bit 6, the transmitter holding and shift registers are empty, no serial conversions are taking place so there should be no activity on the transmit data line. When bit 5 is set, the transmitter holding register is empty, thus another byte can be sent to the data port, but a serial conversion using the shift register may be taking place.

The break interrupt (Bit 4) occurs when the received data line is held in a logic state '0' (Space) for more than the time it takes to send a full word. That includes the time for the start bit, data bits, parity bits and stop bits.

A framing error (Bit 3) occurs when the last bit is not a stop bit. This may occur due to a timing error. You will most commonly encounter a framing error when using a null modem linking two computers or a protocol analyzer when the speed at which the data is being sent is different to that of what you have the UART set to receive it at.

A overrun error normally occurs when your program can't read from the port fast enough. If you don't get an incoming byte out of the register fast enough, and another byte just happens to be received, then the last byte will be lost and a overrun error will result.

Bit 0 shows data ready, which means that a byte has been received by the UART and is at the receiver buffer ready to be read.

Modem Status Register (MSR)

Bit	Notes
Bit 7	Carrier Detect
Bit 6	Ring Indicator
Bit 5	Data Set Ready
Bit 4	Clear To Send
Bit 3	Delta Data Carrier Detect
Bit 2	Trailing Edge Ring Indicator
Bit 1	Delta Data Set Ready
Bit 0	Delta Clear to Send

Table 13 : Modem Status Register

Bit 0 of the modem status register shows delta clear to send, delta meaning a change in, thus delta clear to send means that there was a change in the clear to send line, since the last read of this register. This is the same for bits 1 and 3. Bit 1 shows a change in the Data Set Ready line where as Bit 3 shows a change in the Data Carrier Detect line. Bit 2 is the Trailing Edge Ring Indicator which indicates that there was a transformation from low to high state on the Ring Indicator line.

Bits 4 to 7 show the current state of the data lines when read. Bit 7 shows Carrier Detect, Bit 6 shows Ring Indicator, Bit 5 shows Data Set Ready & Bit 4 shows the status of the Clear To Send line.

Scratch Register

The scratch register is not used for communications but rather used as a place to leave a byte of data. The only real use it has is to determine whether the UART is a 8250/8250B or a 8250A/16450 and even that is not very practical today as the 8250/8250B was never designed for AT's and can't hack the bus speed.

Part 3 : Programming (PC's)

Polling or Interrupt Driven?
Source Code - Termpoll.c (Polling Version)
Source Code - Buff1024.c (ISR Version)
Interrupt Vectors
Interrupt Service Routine
UART Configuration
Main Routine (Loop)
Determining the type of UART via Software

Part 4 : External Hardware - Interfacing Methods

RS-232 Waveforms
RS-232 Level Converters
Making use of the Serial Format

8250 and compatable UART's
CDP6402, AY-5-1015 / D36402R-9 etc UARTs
Microcontrollers

RS232 Data Interface

a Tutorial on Data Interface and cables

RS-232 is simple, universal, well understood and supported but it has some serious shortcomings as a data interface. The standards to 256kbps or less and line lengths of 15M (50 ft) or less but today we see high speed ports on our home PC running very high speeds and with high quality cable maxim distance has increased greatly. The rule of thumb for the length a data cable depends on speed of the data, quality of the cable.

---

a Tutorial

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumors for its early demise, has remained widely used through the industry.

Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol.

The RS-232 interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption when a short cable connects the DTE to the DCE, but with longer lines and connections between devices that may be on different electrical busses with different grounds, this may not be true.

RS232 data is bi-polar.... +3 TO +12 volts indicates an "ON or 0-state (SPACE) condition" while A -3 to -12 volts indicates an "OFF" 1-state (MARK) condition.... Modern computer equipment ignores the negative level and accepts a zero voltage level as the "OFF" state. In fact, the "ON" state may be achieved with lesser positive potential. This means circuits powered by 5 VDC are capable of driving RS232 circuits directly, however, the overall range that the RS232 signal may be transmitted/received may be dramatically reduced.

The output signal level usually swings between +12V and -12V. The "dead area" between +3v and -3v is designed to absorb line noise. In the various RS-232-like definitions this dead area may vary. For instance, the definition for V.10 has a dead area from +0.3v to -0.3v. Many receivers designed for RS-232 are sensitive to differentials of 1v or less.

This can cause problems when using pin powered widgets - line drivers, converters, modems etc.

These type of units need enough voltage & current to power them self's up. Typical URART (the RS-232 I/O chip) allows up to 50ma per output pin - so if the device needs 70ma to run we would need to use at least 2 pins for power. Some devices are very efficient and only require one pin (some times the Transmit or DTR pin) to be high - in the "SPACE" state while idle.

An RS-232 port can supply only limited power to another device. The number of output lines, the type of interface driver IC, and the state of the output lines are important considerations.

The types of driver ICs used in serial ports can be divided into three general categories:

- **Drivers which require plus (+) and minus (-) voltage power supplies such as the 1488 series of interface integrated circuits. (Most desktop and tower PCs use this type of driver.)**
- **Low power drivers which require one +5 volt power supply. This type of driver has an internal charge pump for voltage conversion. (Many industrial microprocessor controls use this type of driver.)**
- **Low voltage (3.3 v) and low power drivers which meet the EIA-562 Standard. (Used on notebooks and laptops.)**

Data is transmitted and received on pins 2 and 3 respectively. Data Set Ready (DSR) is an indication from the Data Set (i.e., the modem or DSU/CSU) that it is on. Similarly, DTR indicates to the Data Set that the DTE is on. Data Carrier Detect (DCD) indicates that a good carrier is being received from the remote modem.

Pins 4 RTS (Request To Send - from the transmitting computer) and 5 CTS (Clear To Send - from the Data set) are used to control. In most Asynchronous situations, RTS and CTS are constantly on throughout the communication session. However where the DTE is connected to a multipoint line, RTS is used to turn carrier on the modem on and off. On a multipoint line, it's imperative that only one station is transmitting at a time (because they share the return phone pair). When a station wants to transmit, it raises RTS. The modem turns on carrier, typically waits a few milliseconds for carrier to stabilize, and then raises CTS. The DTE transmits when it sees CTS up. When the station has finished its transmission, it drops RTS and the modem drops CTS and carrier together.

Clock signals (pins 15, 17, & 24) are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

Note: Transmit and receive leads (2 or 3) can be reversed depending on the use of the equipment - DCE Data Communications Equipment or a DTE Data Terminal Equipment.

Glossary of Abbreviations etc.

CTS	**Clear To Send [DCE --> DTE]**
DCD	**Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
DCE	**Data Communications Equipment eg. modem**
DSR	**Data Set Ready [DCE --> DTE]**
DSRS	**Data Signal Rate Selector [DCE --> DTE] (Not commonly used)**
DTE	**Data Terminal Equipment eg. computer, printer**
DTR	**Data Terminal Ready [DTE --> DCE]**
FG	**Frame Ground (screen or chassis)**
NC	**No Connection**
RCk	**Receiver (external) Clock input**
RI	**Ring Indicator (ringing tone detected)**
RTS	**Ready To Send [DTE --> DCE]**
RxD	**Received Data [DCE --> DTE]**
SG	**Signal Ground**
SCTS	**Secondary Clear To Send [DCE --> DTE]**
SDCD	**Secondary Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
SRTS	**Secondary Ready To Send [DTE --> DCE]**
SRxD	**Secondary Received Data [DCE --> DTE]**
STxD	**Secondary Transmitted Data [DTE --> DTE]**
TxD	**Transmitted Data [DTE --> DTE]**

Is Your Interface a DTE or a DCE?

Find out by following these steps: The point of reference for all signals is the terminal (or PC).

1) Measure the DC voltages between (DB25) pins 2 & 7 and between pins 3 & 7. Be sure the black lead is connected to pin 7 (Signal Ground) and the red lead to whichever pin you are measuring.

2) If the voltage on pin 2 (TD) is more negative than -3 Volts, then it is a DTE, otherwise it should be near zero volts.

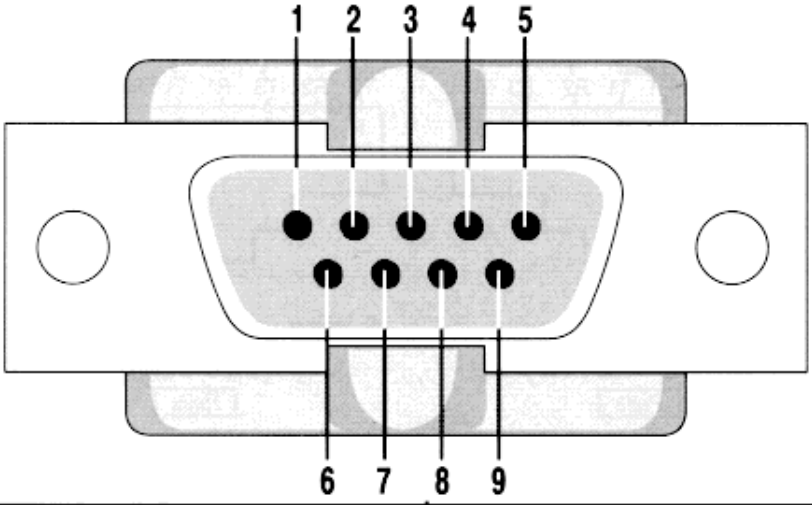
3) If the voltage on pin 3 (RD) is more negative than -3 Volts, then it is a DCE.

4) If both pins 2 & 3 have a voltage of at least 3 volts, then either you are measuring incorrectly, or your device is not a standard EIA-232 device. Call technical support.

5) In general, a DTE provides a voltage on TD, RTS, & DTR, whereas a DCE provides voltage on RD, CTS, DSR, & CD.

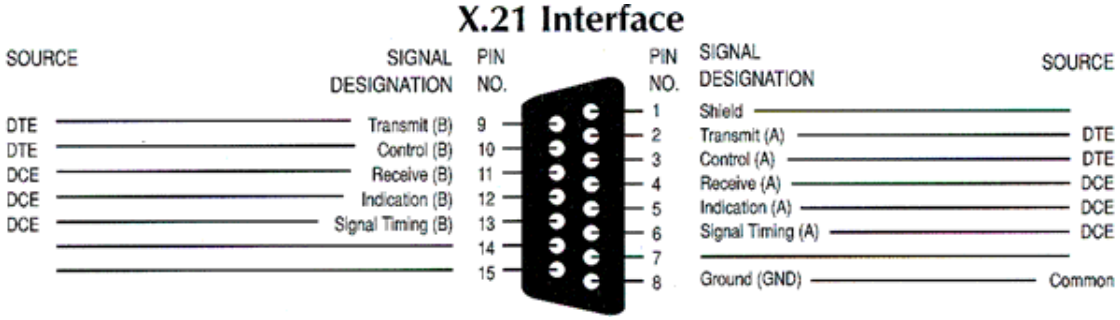
PC Com Port - EIA-574
RS-232/V.24 pin out on a DB-9 pin

used for Asynchronous Data



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

X.21 interface on a DB 15 connector



also see X.21 write up
also see end of page for more info

X.21

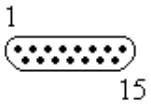
General

Voltages:	+/- 0.3Vdc
Speeds:	Max. 100Kbps (X.26)
	Max. 10Mbps (X.27)

The X.21 interface was recommended by the CCITT in 1976. It is defined as a digital signalling interface between customers (DTE) equipment and carrier's equipment (DCE). And thus primarally used for telecom equipment.

All signals are balanced. Meaning there is always a pair (+/-) for each signal, like used in RS422. The X.21 signals are the same as RS422, so please refer to RS422 for the exact details.

Pinning according to ISO 4903

	
Sub-D15 Male	Sub-D15 Female

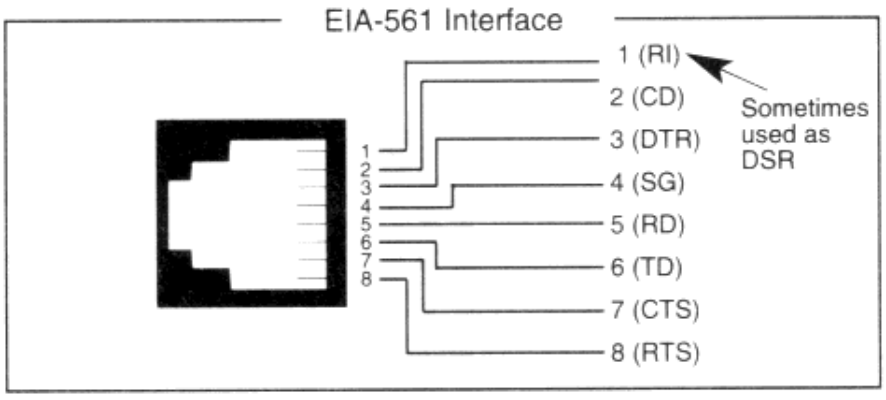
Pin	Signal	abbr.	DTE	DCE
1	Shield		-	-
2	Transmit (A)		Out	In
3	Control (A)		Out	In
4	Receive (A)		In	Out
5	Indication (A)		In	Out
6	Signal Timing (A)		In	Out
7	Unassigned			
8	Ground		-	-
9	Transmit (B)		Out	In
10	Control (B)		Out	In
11	Receive (B)		In	Out
12	Indication (B)		In	Out
13	Signal Timing (B)		In	Out
14	Unassigned			
15	Unassigned			

Functional Description
As can be seen from the pinning specifications, the Signal Element Timing (clock) is provided by the DCE. This means that your provider (local telco office) is responsible for the correct clocking and that X.21 is a synchronous interface. Hardware handshaking is done by the Control and Indication lines. The Control is used by the DTE and the Indication is the DCE one.

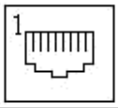
Cross-cable pinning

X.21 Cross Cable	
X.21	**X.21**
1	1
2	4
3	5
4	2
5	3
6	7
7	6
8	8
9	11
10	12
11	9
12	10
13	14
14	13
15	

EIA-561 defines RS-232 on a modular connector. (For nonsynchronous applications only, since it does not provide for the synchronous clocking signals.)



RS232D uses RJ45 type connectors (similar to telephone connectors)

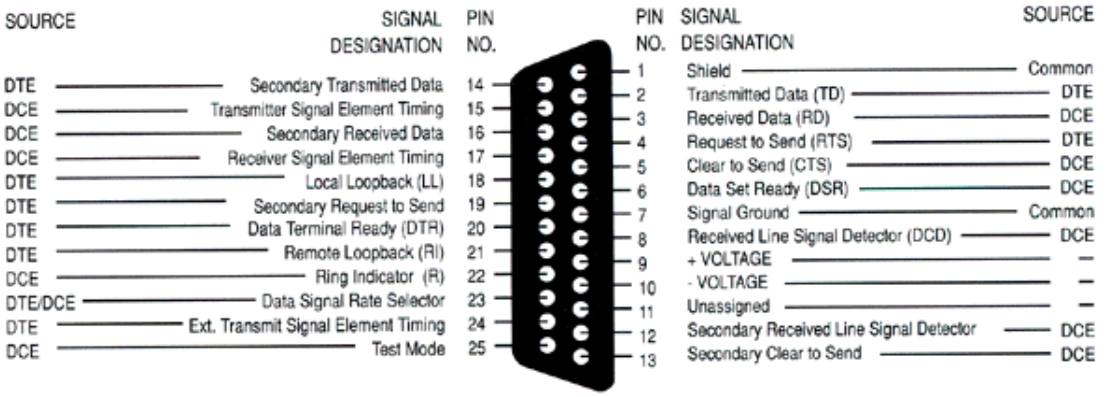
	Male	Female
RJ45		

Pin No.	Signal Description	Abbr.	DTE	DCE
1	DCE Ready, Ring Indicator	DSR/RI	←	→
2	Received Line Signal Detector	DCD	←	→
3	DTE Ready	DTR	→	←
4	Signal Ground	SG		
5	Received Data	RxD	←	→
6	Transmitted Data	TxD	→	←
7	Clear To Send	CTS	←	→
8	Request To Send	RTS	→	←

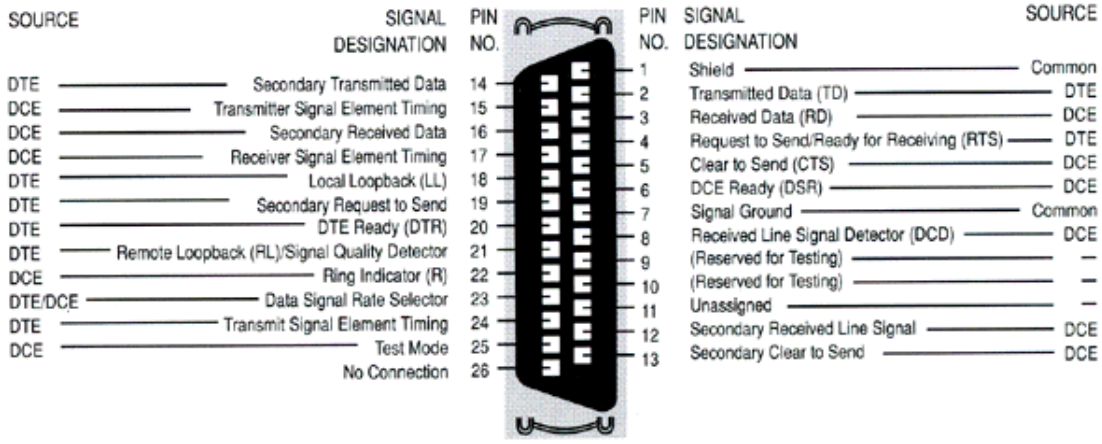
This is a standard 9 to 25 pin cable layout for async data on a PC AT serial cable

Description	Signal	9-pin DTE	25-pin DCE	Source DTE or DEC
Carrier Detect	CD	1	8	from Modem
Receive Data	RD	2	3	from Modem
Transmit Data	TD	3	2	from Terminal/Computer
Data Terminal Ready	DTR	4	20	from Terminal/Computer
Signal Ground	SG	5	7	from Modem
Data Set Ready	DSR	6	6	from Modem
Request to Send	RTS	7	4	from Terminal/Computer
Clear to Send	CTS	8	5	from Modem
Ring Indicator	RI	9	22	from Modem

V.24/RS-232 Interface



V.24/RS-232E ALT A Connector



---

25 pin D-shell connector RS232

commonly used for Async. data

PIN SIGNAL DESCRIPTION

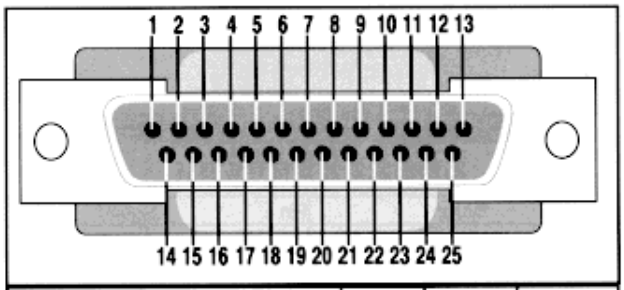
1 PGND Protective Ground
2 TXD Transmit Data
3 RXD Receive Data
4 RTS Ready To Send
5 CTS Clear To Send

6 DSR Data Set Ready
7 SG Signal Ground
8 CD Carrier Detect
20 DTR Data Terminal Ready
22 RI Ring Indicator

Some applications require more
than a simple async. configurat

RS-232 Interface

RS-232 (EIA Std.) applicable to the 25 pin interconnection of Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) using serial binary data

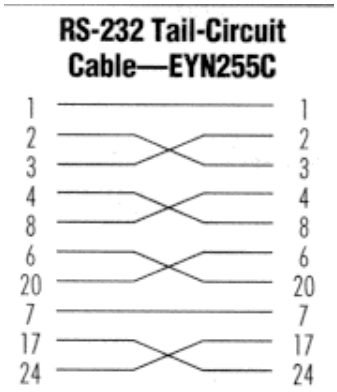


Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CI	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

Pins used for Synchronous data

jump to Other Connector pages

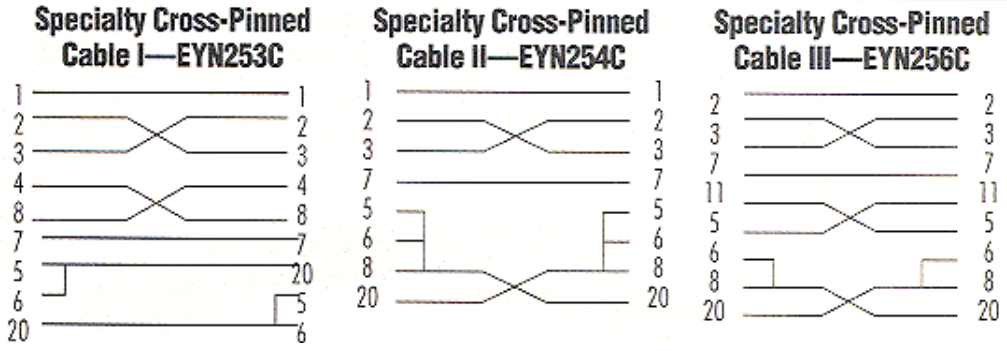
RS232 (25 pin) Tail Circuit Cable



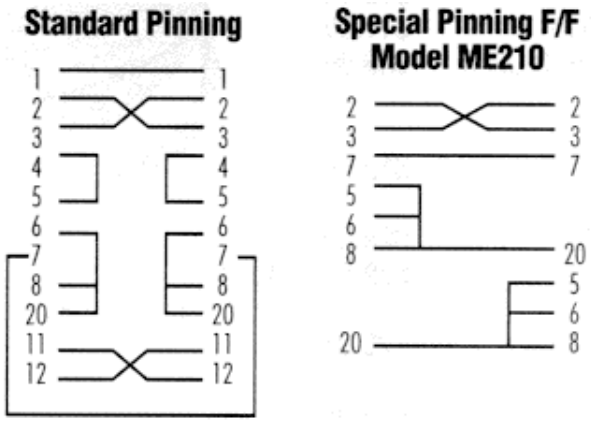
Null Modem cable diagrams

- Nullmodem (9p to 9p)
- Nullmodem (9p to 25p)
- Nullmodem (25p to 25p)

Cross Pinned cables for Async data.



Pin out for local Async Data transfer



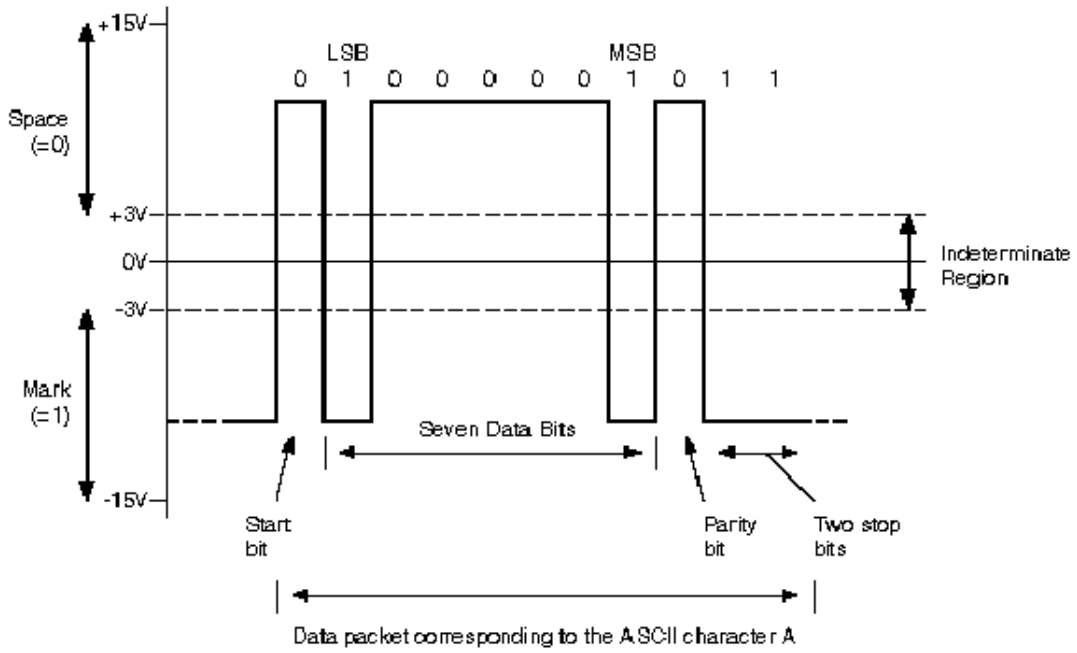
Loopback plugs:

- Serial Port Loopback (9p)
- Serial Port Loopback (25p)

RS-232 Specs.

SPECIFICATIONS		RS232	RS423
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR
Maximum Cable Length		50 FT.	4000 FT.
Maximum Data Rate		20kb/s	100kb/s
Maximum Driver Output Voltage		+/-25V	+/-6V
Driver Output Signal Level (Loaded Min.)	**Loaded**	+/-5V to +/-15V	+/-3.6V
Driver Output Signal Level (Unloaded Max)	**Unloaded**	+/-25V	+/-6V
Driver Load Impedance (Ohms)		3k to 7k	>=450
Max. Driver Current in High Z State	**Power On**	N/A	N/A
Max. Driver Current in High Z State	**Power Off**	+/-6mA @ +/-2v	+/-100uA
Slew Rate (Max.)		30V/uS	Adjustable
Receiver Input Voltage Range		+/-15V	+/-12V
Receiver Input Sensitivity		+/-3V	+/-200mV
Receiver Input Resistance (Ohms)		3k to 7k	4k min.

One byte of async data



Cabling considerations - you should use cabling made for RS-232 data but I have seen low speed data go over 250’ on 2 pair phone cable. Level 5 cable can also be used but for best distance use a low capacitance data grade cable.

The standard maxim length is 50’ but if data is async you can increase that distance to as much as 500’ with a good grade of cable.

The RS-232 signal on a single cable is impossible to screen effectively for noise. By screening the entire cable we can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the different lines introduces serious crosstalk (this especially true on synchronous data - because of the clock lines) until a point is reached where the data itself is unreadable. Signal Crosstalk can be reduced by using low capacitance cable and shielding each pair

Using a high grade cable (individually shield low capacitance pairs) the distance can be extended to 4000’

At higher frequencies a new problem comes to light. The high frequency component of the data signal is lost as the cable gets longer resulting in a rounded, rather than square wave signal.

The maxim distance will depend on the speed and noise level around the cable run.

On longer runs a line driver is needed. This is a simple modem used to increase the maxim distance you can run RS-232 data.

Making sense of the specifications

Selecting data cable isn't difficult, but often gets lost in the shuffle of larger system issues. Care should be taken. however, because intermittent problems caused by marginal cable can be very difficult to troubleshoot.

Beyond the obvious traits such as number of conductors and wire gauge, cable specifications include a handful of less intuitive terms.

Characteristic Impedance (Ohms): A value based on the inherent conductance, resistance, capacitance and inductance of a cable that represents the impedance of an infinitely long cable. When the cable is out to any length and terminated with this Characteristic Impedance, measurements of the cable will be identical to values obtained from the infinite length cable. That is to say that the termination of the cable with this impedance gives the cable the appearance of being infinite length, allowing no reflections of the transmitted signal. If termination is required in a system, the termination impedance value should match the Characteristic Impedance of the cable.

Shunt Capacitance (pF/ft): The amount of equivalent capacitive load of the cable, typically listed in a per foot basis One of the factors limiting total cable length is the capacitive load. Systems with long lengths benefits from using low capacitance cable.

Propagation velocity (% of c): The speed at which an electrical signal travels in the cable. The value given typically must be multiplied by the speed of light (c) to obtain units of meters per second. For example, a cable that lists a propagation velocity of 78% gives a velocity of 0.78 X 300 X 106 - 234 X 106 meters per second.

Plenum cable

Plenum rated cable is fire resistant and less toxic when burning than non-plenum rated cable. Check building and fire codes for requirements. Plenum cable is generally more expensive due to the sheathing material used.

The specification recommends 24AWG twisted pair cable with a shunt capacitance of 16 pF per foot and 100 ohm characteristic impedance.

It can be difficult to qualify whether shielding is required in a particular system or not, until problems arise. We recommend erring on the safe side and using shielded cable. Shielded cable is

only slightly more expensive than unshielded.

There are many cables available meeting the recommendations of RS-422 and RS-485, made specifically for that application. Another choice is the same cable commonly used in the Twisted pair Ethernet cabling. This cable, commonly referred to as Category 5 cable, is defined by the ElA/TIA/ANSI 568 specification The extremely high volume of Category 5 cable used makes it widely available and very inexpensive, often less than half the price of specialty RS422/485 cabling. The cable has a maximum capacitance of 17 pF/ft (14.5 pF typical) and characteristic impedance of 100 ohms.

Category 5 cable is available as shielded twisted pair (STP) as well as unshielded twisted pair (UTP) and generally exceeds the recommendations making it an excellent choice for RS232 systems.

RS232 - V.24/V.28 - IS2110 - X.20 bis (for Async) - X.21 bis (for Sync)

General

In this document the term RS232 will be used when refered to this serial interface. The description of RS232 is an EIA/TIA norm and is identical to CCITT V.24/V.28, X.20bis/X.21bis and ISO IS2110. The only difference is that CCITT has split the interface into its electrical description (V.28) and a mechanical part (V.24) or Asynchronous (X.20 bis) and Synchronous (X.21 bis) where the EIA/TIA describes everything under RS232.

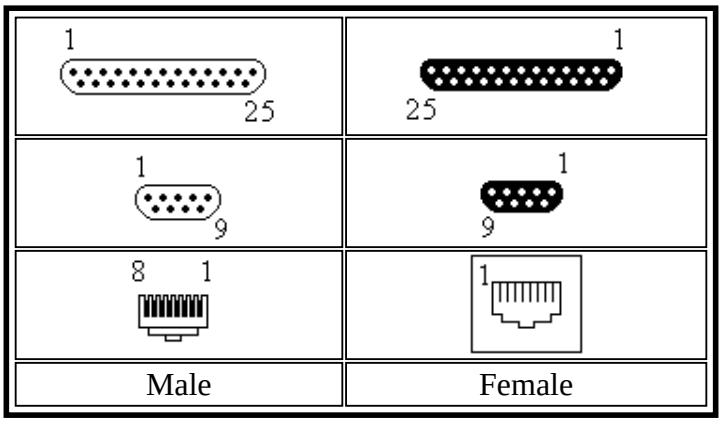
As said before RS232 is a serial interface. It can be found in many different applications where the most common ones are modems and Personal Computers. All pinning specifications are writen for the DTE side.

All DTE-DCE cables are straight through meaning the pins are connected one on one. DTE-DTE and DCE-DCE cables are cross cables. To make a destiction between all different types of cables we have to use a naming convention.
DTE - DCE: Straight Cable
DTE - DTE: Null-Modem Cable
DCE - DCE: Tail Circuit Cable

Interface Mechanical

RS232 can be found on different connectors. There are special specifications for this. The CCITT only defines a Sub-D 25 pins version where the EIA/TIA has two versions RS232C and RS232D which are resp. on a Sub-D25 and a RJ45. Next to this IBM has added a Sub-D 9 version which is found an almost

all Personal Computers and is described in TIA 457.



Pinning

RS232-C	Description	Circuit EIA	Circuit CCITT	RJ45	TIA 457
1	Shield Ground	AA			
7	Signal Ground	AB	102	4	5
2	Transmitted Data	BA	103	6	3
3	Received Data	BB	104	5	2
4	Request To Send	CA	105	8	7
5	Clear To Send	CB	106	7	8
6	DCE Ready	CC	107	1	6
20	DTE Ready	CD	108.2	3	4
22	Ring Indicator	CE	125	1	9
8	Received Line Signal Detector	CF	109	2	1
23	Data Signal Rate Select (DTE/DCE Source>	CH/CI	111/112		
24	Transmit Signal Element Timing (DTE Source)	DA	113		
15	Transmitter Signal Element Timing (DCE Source)	DB	114		
17	Receiver Signal Element Timing (DCE Source)	DD	115		
18	Local Loopback / Quality Detector	LL	141		
21	Remote Loopback	RL/CG	140/110		
14	Secondary Transmitted Data	SBA	118		
16	Secondary Received Data	SBB	119		
19	Secondary Request To Send	SCA	120		
13	Secondary Clear To Send	SCB	121		
12	Secondary Received Line Signal Detector/ Data signal Rate Select (DCE Source)	SCF/CI	122/112		
25	Test Mode	TM	142		
9	Reserved for Testing				
10	Reserved for Testing				
11	Unassigned				

Interface Electrical

All signals are measured in reference to a common ground, which is called the signal ground (AB). A positive voltage between 3 and 15 Vdc represents a logical 0 and a negative voltage between 3 and 15 Vdc represents a logical 1.
This switching between positive and negative is called bipolar. The zero state is not defined in RS232

and is considered a fault condition (this happens when a device is turned off).
According to the above a maximum distance of 50 ft or 15 m. can be reached at a maximum speed of 20k bps. This is according to the official specifications, the distance can be exceeded with the use of Line Drivers.

Functional description

Description	Circuit	Function
Shield Ground	AA	Also known as protective ground. This is the chassis ground connection between DTE and DCE.
Signal Ground	AB	The reference ground between a DTE and a DCE. Has the value 0 Vdc.
Transmitted Data	BA	Data send by the DTE.
Received Data	BB	Data received by the DTE.
Request To Send	CA	Originated by the DTE to initiate transmission by the DCE.
Clear To Send	CB	Send by the DCE as a reply on the RTS after a delay in ms, which gives the DCEs enough time to energize their circuits and synchronize on basic modulation patterns.
DCE Ready	CC	Known as DSR. Originated by the DCE indicating that it is basically operating (power on, and in functional mode).
DTE Ready	CD	Known as DTR. Originated by the DTE to instruct the DCE to setup a connection. Actually it means that the DTE is up and running and ready to communicate.
Ring Indicator	CE	A signal from the DCE to the DTE that there is an incomming call (telephone is ringing). Only used on switched circuit connections.
Received Line Signal Detector	CF	Known as DCD. A signal send from DCE to its DTE to indicate that it has received a basic carrier signal from a (remote) DCE.
Data Signal Rate Select (DTE/DCE Source>	CH/CI	A control signal that can be used to change the transmission speed.
Transmit Signal Element Timing (DTE Source)	DA	Timing signals used by the DTE for transmission, where the clock is originated by the DTE and the DCE is the slave.
Transmitter Signal Element Timing (DCE Source)	DB	Timing signals used by the DTE for transmission.
Receiver Signal Element Timing (DCE Source)	DD	Timing signals used by the DTE when receiving data.
Local Loopback / Quality Detector	LL	
Remote Loopback	RL/CG	Originated by the DCE that changes state when the analog signal received from the (remote) DCE becomes marginal.

Test Mode	TM	
Reserved for Testing		

The secondary signals are used on some DCE’s. Those units have the possibility to transmit and/or receive on a secondary channel. Those secondary channels are mostly of a lower speed than the normal ones and are mainly used for administrative functions.

Cable pinning
Here are some cable pinning that might be useful. Not all applications are covered, it is just a help:

Straight DB25 Cable	**DB25 Null- modem cable (Async)**	**DB25 Tail- circuit cable (Sync)**	**DB25 to DB9 DTE - DCE cable**	**DB25 to DB9 DTE - DTE cable**

Pin	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25

Pin	Pin
1	1
2	3
3	2
4	5
5	4
6, 8	20
7	7
20	6, 8

</

DB9 Null- modem cable

1,6	4
2	3
3	2
4	1,6
5	5
7	8
8	7

Pin	Pin
1	1
2	3
3	2
4	8
6	20
7	7
8	4
17	24
20	6
24	17

---

jump to The Belden Cable Company's cable selection tutorial pages

jump to Data Communication by CAMI Research good write up

jump to RS-232 by CAMI Research good write up

jump to Interfacing the Serial / RS232 Port good write up
(in-depth very technical)

jump to Data Modems for phone lines

jump to Data Modems for fiber optics

jump to Interface converters

ARC Electronics ...
800-926-0226
Home Page

arc@arcelect.com

Functions Used

Serial Data Transmission in Asynchronous Mode

---

1. In this task example, a Serial Communication Interface (SCI) is used for serial data transmission in asynchronous mode.Figure 2 shows a block diagram of serial data transmission in asynchronous mode which is described below.
   - In asynchronous mode, serial data communication is performed asynchronously, with synchronization provided character by character.
   - Serial data can by communicated with standard asynchronous communication LSIs such as Universal Asynchronous Receiver/Transmitter (UART) and Asynchronous Communication Interface Adapter (ACIA).
   - A multi-processor communication function is provided to enable serial data communications with multiple processors.
   - The transfer format can be selected from 16 transfer format types.
   - The transmitter and receiver are independent, enabling simultaneous transmission and reception. Both the transmitter and receiver have a double-buffer architecture to achieve continuous transmission and reception.
   - Any desired bit rate can be selected using the on-chip baud rate generator.
   - The transmit/receive clock source can be selected from internal or external clocks.
   - There are six interrupt factors, namely, transmit complete, transmit data empty, receive data full, overrun error, framing error and parity error.
   - The Receive Shift Register (RSR) is a register to receive serial data. Serial data input from RXD32 pin is set in RSR in the receiving order that is starting from the LSB (Bit 0), and is converted into parallel data. When one-byte data is received, it is transferred automatically to RDR. RSR cannot be read from or written to directly by the CPU.
   - The Receive Data Register (RDR) is an 8-bit register to store received serial data. Receiving one-byte data, the received data is transferred from RSR to RDR to complete receive operation. RSR is then ready to receive data. RSR and RDR have a double buffer, enabling continuous receive operations. RDR is a receive-only register and cannot be written to by the CPU.
   - The Transmission Shift Register (TSR) is a register to transmit serial data. Transmit data is temporarily transferred from TDR to TSR and is sent to TXD32 pin starting from the LSB (Bit 0) for serial data transmission. Transmitting one-byte data, the next transmit data is transferred automatically from TDR to TSR to start transmitting. If data is not

written in TDR (1 is set in TDRE), data is not transferred from TDR to TSR. TSR cannot be read from or written to directly by the CPU.

- The Transmit Data Register (TDR) is an 8-bit register to store transmit data. Detecting that TSR is "empty", transmit data written in TDR is transferred to TSR to start serial data transmission. By writing next transmit data in TDR during TSR serial data transmission, continuous transmission is possible. TDR can always be read from or written to by the CPU.
- The Serial Mode Register (SMR) is an 8-bit register for setting of a serial data tranfer format and selecting a clock source for the baud rate generator. SMR can always be read from or written to by the CPU.
- The Serial Control Register 3 (SCR3) is an 8-bit register for selecting transmit/receive operation, clock output in asynchronous mode, interrupt request enable/disable, and transmit/receive clock source. SCR3 can always be read from or written to by the CPU.
- The Serial Port Control Register (SPCR) is an 8-bit register to control P42/TXD32 pin. In this task example, P42/TXD32 pin is set as TXD32 output pin, and the input data of TXD32 pin is set not to be inverted.
- The Serial Status Register (SSR) is an 8-bit register with on-chip status flags indicating operation status of SCI3, and multi-processor bits. SSR can always be read from or written to by the CPU, except 1 cannot be written in TDRE, RDRF, OER, PER or FER. 1 must be read in advance to clear them by writing 0. TEND and MPBR are for read only and data cannot be written in them.
- The Bit Rate Register (BRR) is an 8-bit register to set a transmit/receive bit rate matched to the operating clock for baud rate generator selected by CKS0 and CKS1 in SMR. BRR can always be read from or written to by the CPU.
- Table 1 shows an example of BRR setting in asynchronous mode. Table 1 shows values in the active mode when OSC is 10 MHz.

Table 1 Example of BRR Settings for Bit Rates (Asynchronous Mode)

R Bit Rate (Bps)	110	150	200	250	1200	2400	31250
n	2	2	2	2	0	0	0
N	88	64	48	38	129	64	4
Error(%)	-0.25	+0.16	-0.35	+0.16	+0.16	+0.16	0

Notes: 1.Set errors to be less than 1%.

2.BRR set values can be calculated as follows:

$$N = \frac{OSC}{64 \times 2^{2n} \times B} \times 10^6 - 1$$

B:Bit rate (bps)

N:Set value of baud rate generator BRR (0 <= N <= 255)

OSC:Value of f_{OSC} (MHz) = 10 MHz or subclock f w 32.768 kHz

n:Value set in CKS1 and CKS0 in SMR (0 <= n <= 3)

(See Table 2 for the relation between n and clock.)

Table 2 Relationship between n and Clock

n	Clock	Set Value of SMR	
		CKS1	CKS0
0	¦Õ	0	0
1	¦Õw/4,¦Õw	0	1
2	¦Õ/16	1	0
3	¦Õ/64	1	1

3.The error shown in Table 1 is given by the following equation.(rounded off to two decimals)

$$\text{Error (\%)} = \left\{ \frac{\phi \times 10^6}{(N+1) \times B \times 64 \times 2^{2n-1}} - 1 \right\} \times 100$$

4.When OSC is 10 MHz, the maximum bit rate (asynchronous mode) is 31250 bps, provided n=0 and N=4 are set.

- In asynchronous mode, serial communication is performed with synchronization provided character by character, transmitting and receiving characters added with a start bit indicating the start of communication and a stop bit indicating the end of communication.
- The transmitter and receiver are independent inside SCI3 and full duplex communications are possible. Both the transmitter and receiver have a double-buffer architecture to achieve continuous transmission and reception. Data writing during transmission and data reading during reception can make continuous transmission and reception possible.
- Figure 3 shows data format of asynchronous communications. In asynchronous communications, the communication line is normally maintained in the mark state ("High" level). SCI3 monitors communication line and starts serial communications when it detects the place which has become a space ("Low" level) to serve as a start bit.
- One character in communication data consists of the start bit ("Low" level), followed by transmit/receive data (LSB first, starting from the least significant bit), parity bit ("High" or "Low" level) and stop bit ("High" level) at the end.
- In asynchronous mode, synchronization is achieved by the falling edge of the start bit during reception. Data is sampled on the eighth clock of a frequency obtained by multiplying 16 times the one bit period and communication data is fetched in the center of each bit.

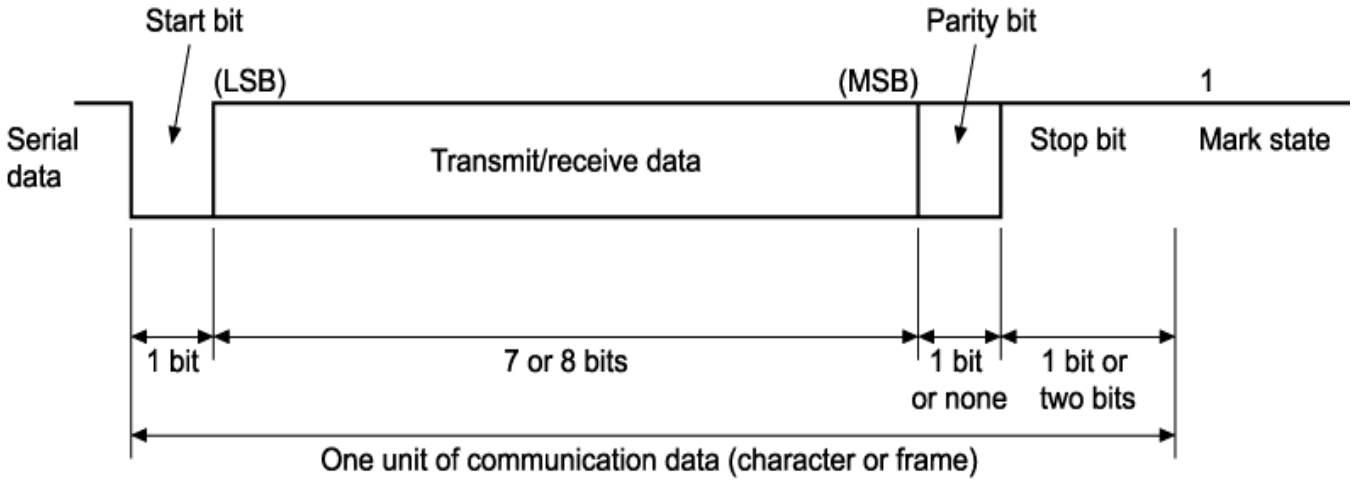


Figure 3 Data Format in Asynchronous Communications

- SCI3 clock (SCK32) is a clock input/output pin of SCI3.
- SCI3 receive data input (RXD32) is a receive data input pin of SCI3.
- SCI3 transmit data output (TXD32) is a transmit data output pin of SCI3.
- SCI3 interrupt factors total six, transmit complete, transmit data empty, receive data full and three receive errors (overrun error, framing error and parity error). Common vector address is assigned to them.
- Each interrupt request can be enabled/disabled by TIE and RIE in SCR3.
- If TDRE in SSR is set to 1, TXI is generated. If TEND in SSR is set to 1, TEI is generated. These two interrupts are generated during transmission.
- The initial value of TDRE in SSR is 1. Therefore, by setting TIE in SCR3 to 1 and by enabling a transmit data empty interrupt request (TXI) before transferring transmit data to TDR, TXI is generated even when transmit data is not ready.
- The initial value of TEND in SSR is 1. Therefore, by setting TEIE in SCR3 to 1 and by enabling a transmit end interrupt request (TEI) before transferring transmit data to TDR, TEI is generated even when transmit data is not sent.
- By processing which transfers transmit data to TDR within the interrupt handling routine, these interrupts can be utilized effectively. To prevent these interrupt requests (TXI and TEI), the enable bits (TIE and TEIE) interacting to these interrupt requests should be set to 1 after transmit data has been transferred to TDR.
- RXI is generated when RDRF in SSR is set to 1. ERI is generated when OER, PER or FER is set to 1. These two interrupt requests are generated during reception.

2. Table 3 shows assignment of functions in this task example. Serial data transmission in asynchronous mode is performed by assigning the functions as shown in Table 3.

Table 3 Assignment of Functions

Function	Assignment
TSR	A register to transmit serial data
TDR	A register to store transmit data
SMR	Sets a serial data transfer format and clock source for baud rate generator
SSR	Status flags to indicate operation status of SCI3
BRR	Sets transmit/receive bit rate
SCR3	Enables transmit operation and sets TXD32 output pin
TXD32	SC13 transmit data output pin
SPCR	Sets TXD32 output pin

8-bit Counter Count-Up by Interval Function	Simultaneous Serial Data Transmission and Reception in Asynchronous Mode	Asynchronous Event Counter
LED Flickering by Clock Time-Base Function	Multiprocessor Communication	LCD Display with Static Duty
Interrupt Period Setting by Auto-Reload Timer Function	Voltage Measurement by 4-Channel A/D Converter	LCD Display with 1/4 Duty
Pulse Frequency Measurement by Event Counter Function	Duty Pulse Output by 10-Bit PWM Function	Oscillation Stabilization Time Settings
Interrupt Counting by 16-Bit Timer Counter Function	Flickering of LEDs Connected to I/O Port	Module Standby Mode Settings
Count of Input Pulses by 16-Bit Event Counter Function	Count Start by External Interrupt	Clock Operation Using Timer F
PWM Output by Output Compare Function	Multiple Interrupt Operation by Internal Interrupt	¡¡
Pulse Period Measurement by Input Capture Function	Transition to Sleep (Medium Speed) Mode	¡¡
Watchdog Timer	Transition to Sleep (High Speed) Mode	¡¡
Serial Data Transmission in Synchronous Mode	Transition to Standby Mode	¡¡
Serial Data Reception in Synchronous Mode	Transition to Watch Mode	¡¡
Simultaneous Serial Data Transmission and Reception in Synchronous mode	Transition to Subsleep Mode	¡¡
Serial Data Transmission in Asynchronous Mode	Transition to Subactive Mode	¡¡
Serial Data Reception in Asynchronous Mode	Transition to Active (Medium Speed) Mode	¡¡

Older One For All IR Remote Serial Protocol

This protocol supposedly works for:

- URC-4000 (One For All 6)
- URC-5000 (One For All 12)

For newer remotes, click here.

Serial Settings

4800 baud, 1 start bit, 8 data bits, no parity, 1 stop bit, half-duplex.

DTR High, RTS Low

The active components hidden in the serial cable's DB9 housing draw power from DTR. Before communication, you should lower DTR and CTS: this resets the circuitry. During communication, you must raise DTR to power the serial cable.

Wake Up Sequence

You must repeat this wake up sequence for each command you send to the remote.

To wake up the remote:

1. raise DTR (to power the serial cable)
2. send a serial BREAK for at least 50 msec (15 msec minimum, but some remotes take longer than that, maybe even 100 msec)
3. receive a wake-up acknowlege from the remote: 0x6E
4. send a serial execute command to the remote: 0xBC
5. receive a serial execute command acknowlege: 0x6F

Once the remote's awake, you can send any single-byte keycode to the remote. The remote will go to sleep after execute the keycode. The remote will echo back the keycode.

If you send a macro command that issues multiple keycodes, the remote echoes back only the last keycode.

After each command:

1. lower DTR (power down the serial cable)
2. wait 200 msec

Table of Keycodes

These keycodes are probably wrong: each new OFA model has a unique set of keycodes. If you map out a particular unit's keycodes, please send a list to Rob at remotes@stormloader.com to add to the codes page.

Name	Code	Name	Code	Name	Code	Name	Code	Name	Code	Name	Code	Name	C
	00	6	10		20	Aux	30	Enter	40	Play	50		6
Mute	01	3	11	B/Audio	21	TV	31		41	Pause	51	Sleep	6
Vol -	02		12	Amp	22		32		42	Stop	52	F1	6
Vol +	03		13	VCR	23	CH +	33	FF	43	Display	53		6
	04	C/Tuner /Video	14	7	24	CH -	34		44		54		6
Power	05	Cable	15	4	25		35		45	F2	55		6
CD	06	8	16	1	26		36	A/B	46	Recall	56		6
Satellite	07	5	17		27	Record	37	F3	47		57		6
9	08		18		28	Program	38	0	48		58		6
	09	2	19	A	29	F4	39		49	Rewind	59		6

The RS232 STANDARD

A Tutorial with Signal Names and Definitions

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Send Us Your Comments . . .

Contents

What is EIA232?
Next Topic | TOC

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the

interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

Likely Problems when Using an EIA232 Interface
Next Topic | Previous Topic | TOC

During this 40-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

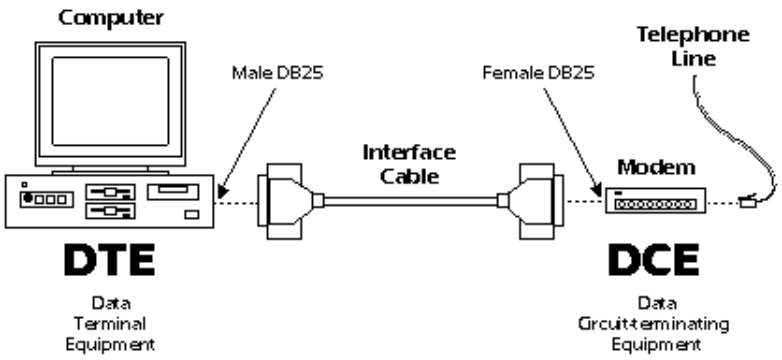
Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

Pin Assignments
Next Topic | Previous Topic | TOC

Go to DTE Pinout (looking into the computer’s serial connector)
Go to DCE Pinout (looking into the modem’s serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is

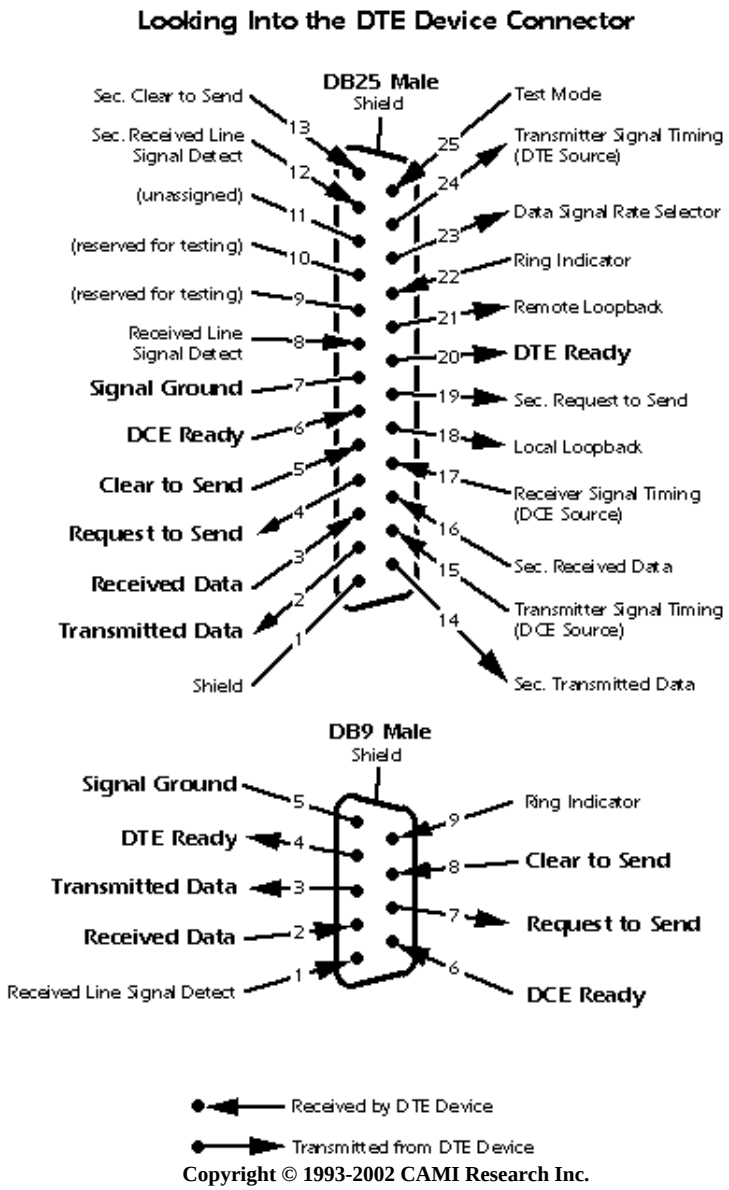
named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.

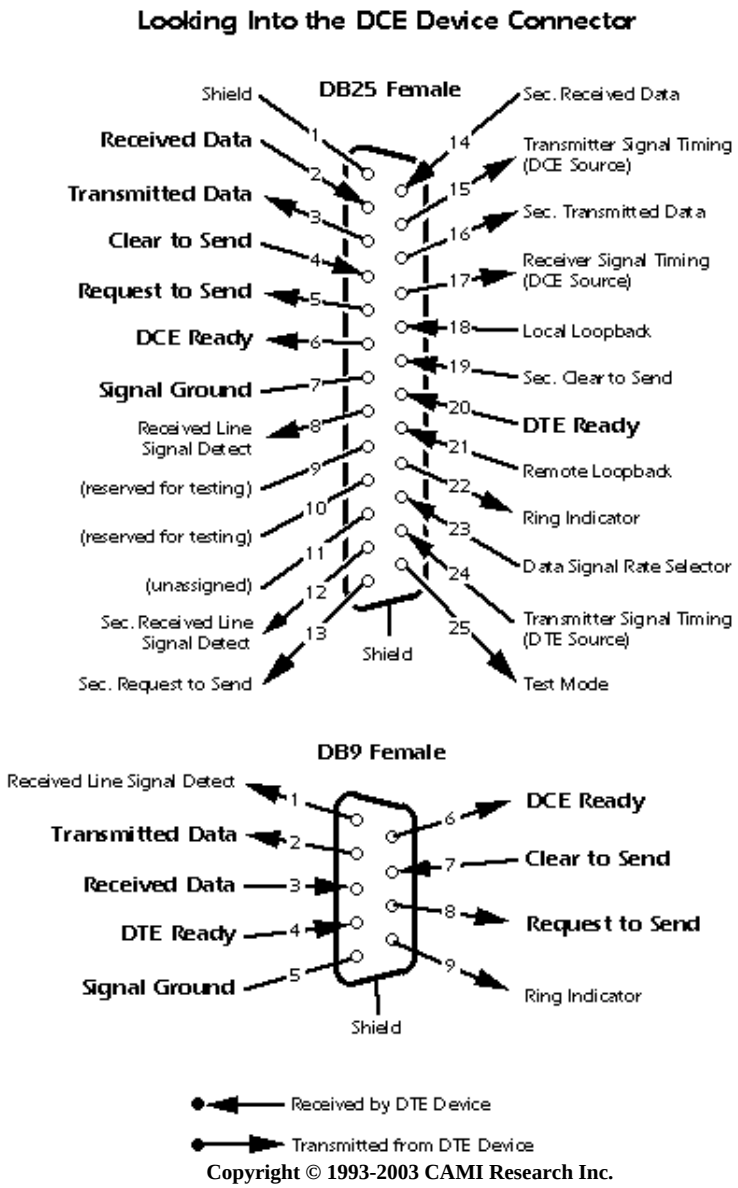
Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



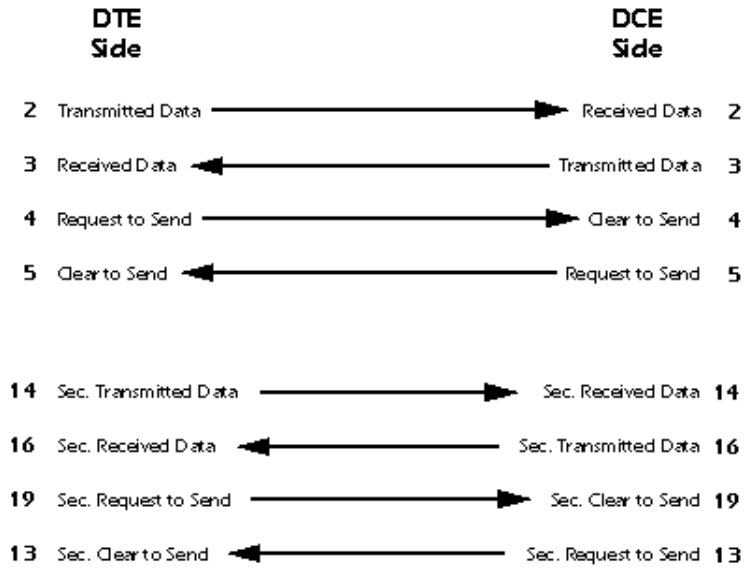
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

> ***IMPORTANT:*** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:



Cable Wiring Examples
Next Topic | Previous Topic | TOC

The following wiring diagrams come from actual cables scanned by the CableEye® PC-Based Cable Test System. CableEye's software automatically draws schematics whenever it tests a cable. Click here to learn more about CableEye.

1 - DB9 All-Line Direct Extension
Next Cable | (no previous cable) || Next Topic

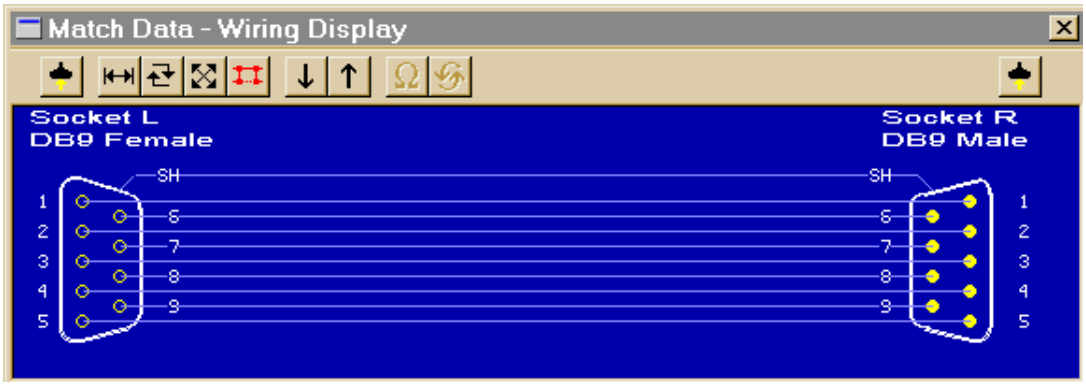
80K

This shows a 9-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 9 pins plus shield are directly extended from DB9 Female to DB9 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any device that uses a DB9 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between a computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Left Side: Connect to **DTE** (computer)

Right Side: Connect to **DCE** (modem or other serial device)



Cable image created by CableEye®

---

2 - DB9 Loopback Connector

Next Cable | Previous Cable || Next Topic

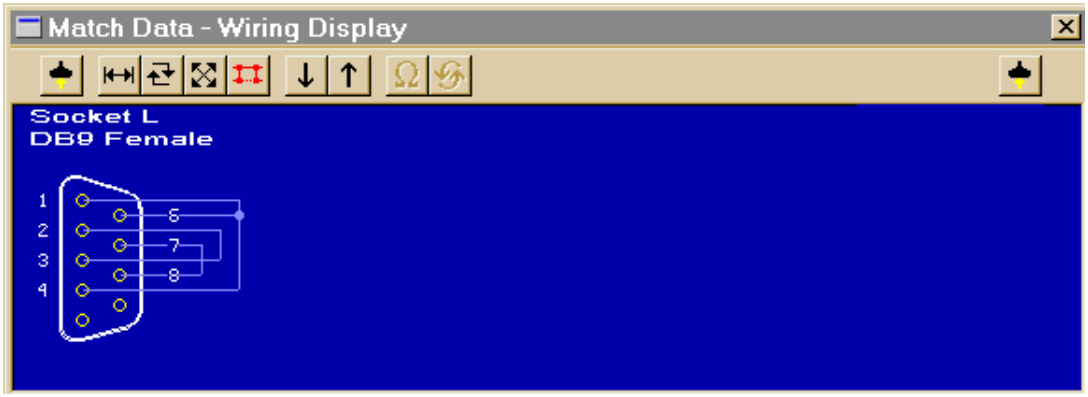
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

3 - DB9 Null Modem Cable
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

80K

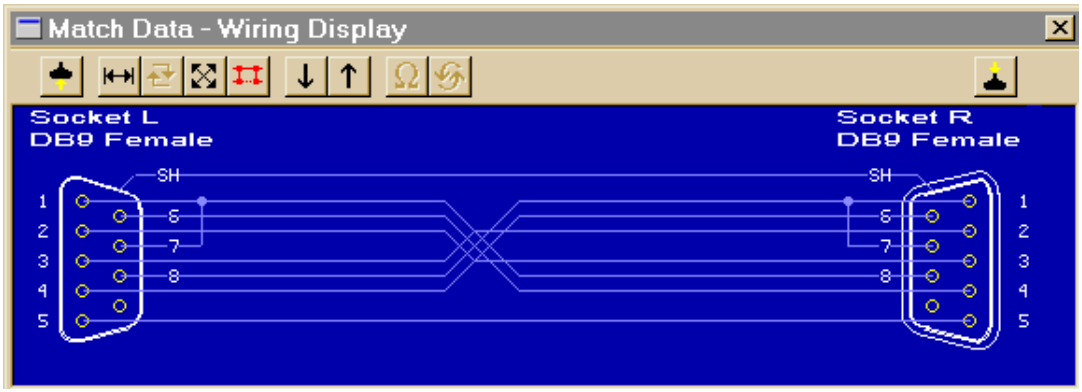
The cable shown below is intended for RS232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals, and a DB25 connector would be necessary.

NOTE: Not all null modem cables connect handshaking lines the same way. In this cable, Request-to-Send (RTS, pin 7) asserts the Carrier Detect (pin 1) on the same side and the Clear-to-Send (CTS, pin 8) on the other side of the cable.

This device may also be available in the form of an adapter.

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 9-pin **DTE** (computer)



Cable image created by CableEye®

---

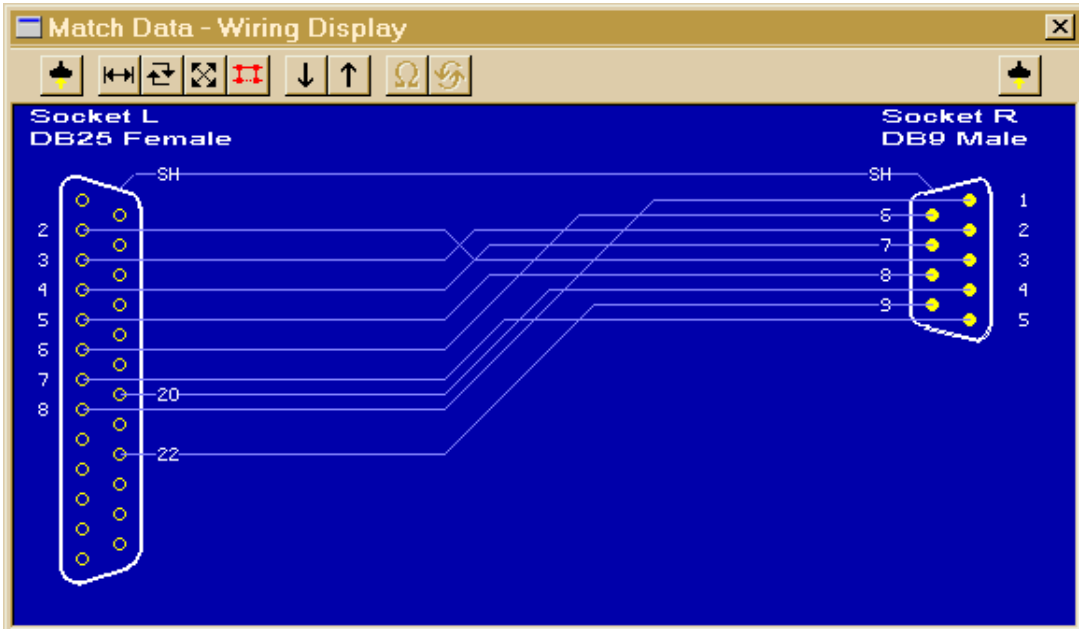
4 - DB25 to DB9 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB25 DTE side are directly mapped to the DB9 assignments for a DTE device. Use this to adapt a 25-pin COM connector on the back of a computer to mate with a 9-pin serial DCE device, such as a 9-pin serial mouse or modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

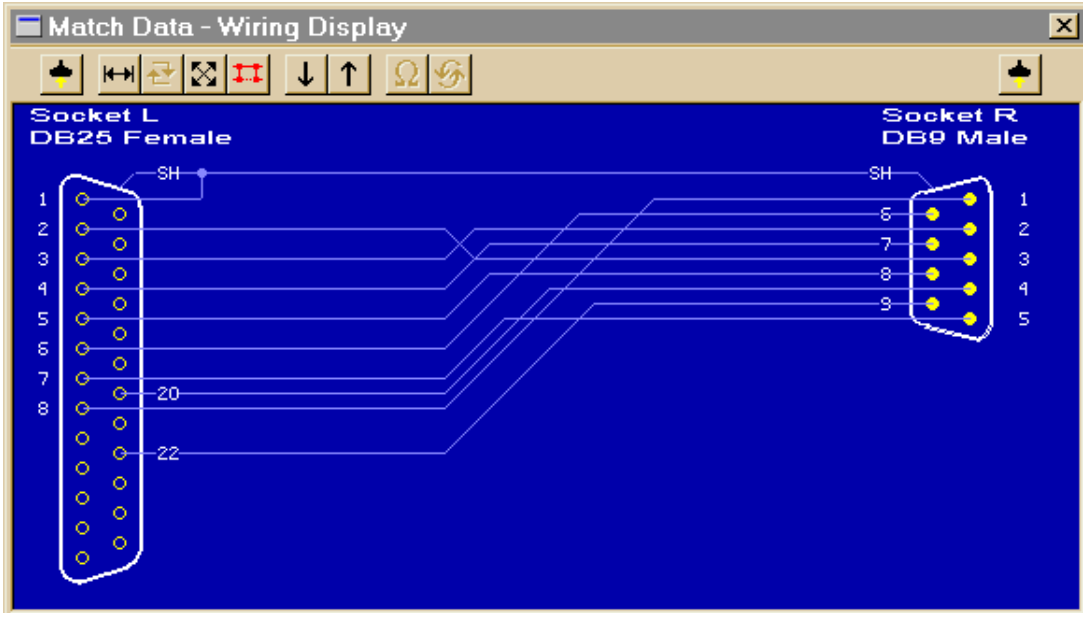
5 - DB25 to DB9 Adapter (pin 1 connected to shield)
Next Cable | Previous Cable || Next Topic

84K

This adapter has the same wiring as the previous cable (#4) except that pin 1 is wired to the connector shell (shield). Note that the cable's shield is usually a foil blanket surrounding all conductors running the length of the cable and joining the connector shells. Pin 1 of the EIA232 specification, called out as "shield", may be separate from the earth ground usually associated with the connector shells.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

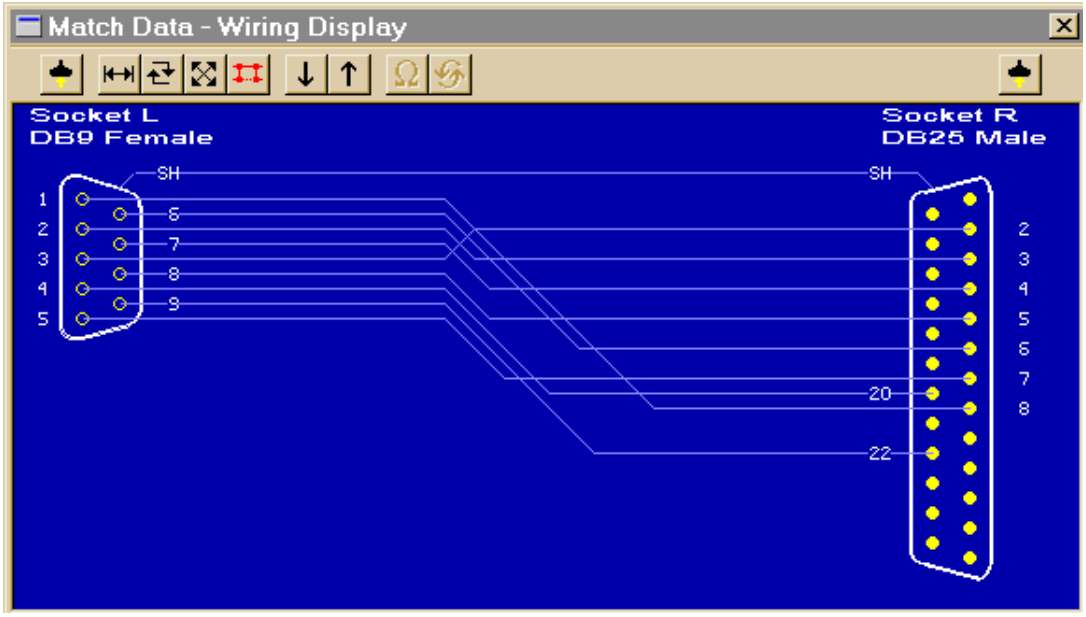
6 - DB9 to DB25 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB9 DTE side are directly mapped to the DB25 assignments for a DTE device. Use this to adapt a 9-pin COM connector on the back of a computer to mate with a 25-pin serial DCE devices, such as a modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

7 - DB25 All-Line Direct Extension
Next Cable | Previous Cable || Next Topic

84K

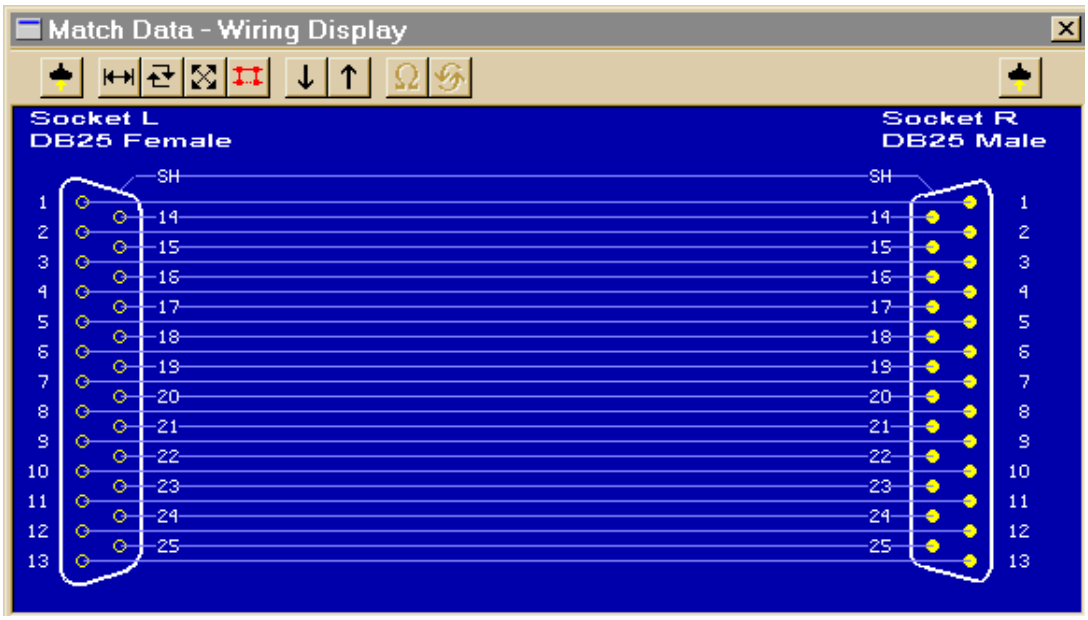
This shows a 25-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 25 pins plus shield are directly extended from DB25 Female to DB25 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any serial device that uses a DB25 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Caution: the male end of this cable (right) also fits a PC's parallel printer port. You may use this cable to extend the length of a printer cable, but DO NOT attach a serial device to the computer's parallel port. Doing so may cause damage to both devices.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

8 - DB25 Loopback Connector
Next Cable | Previous Cable || Next Topic

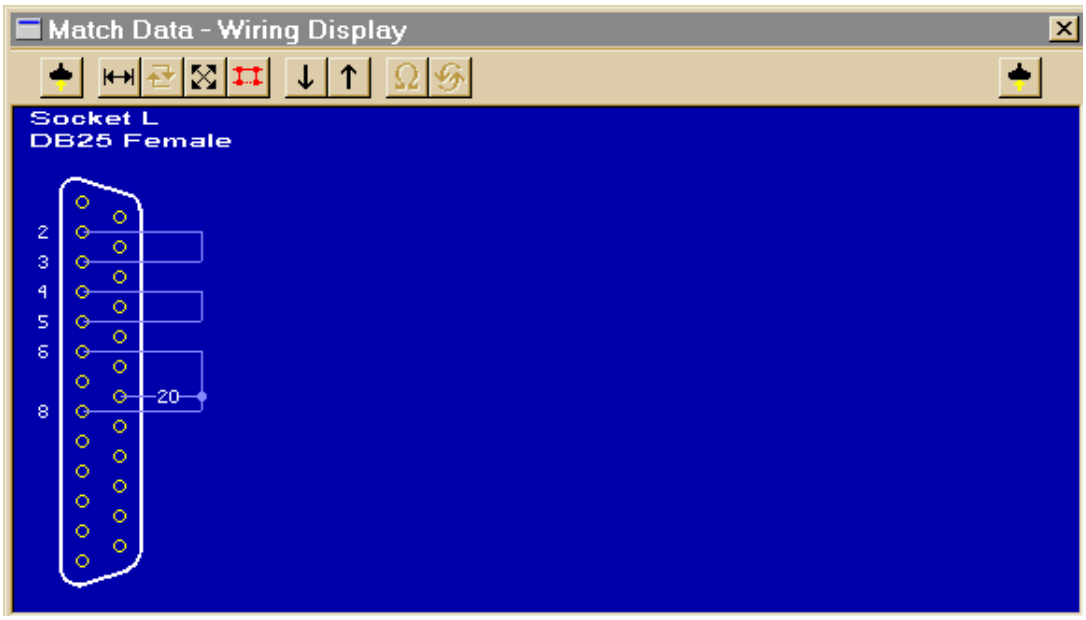
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB25 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable the verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

9 - DB25 Null Modem (no handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

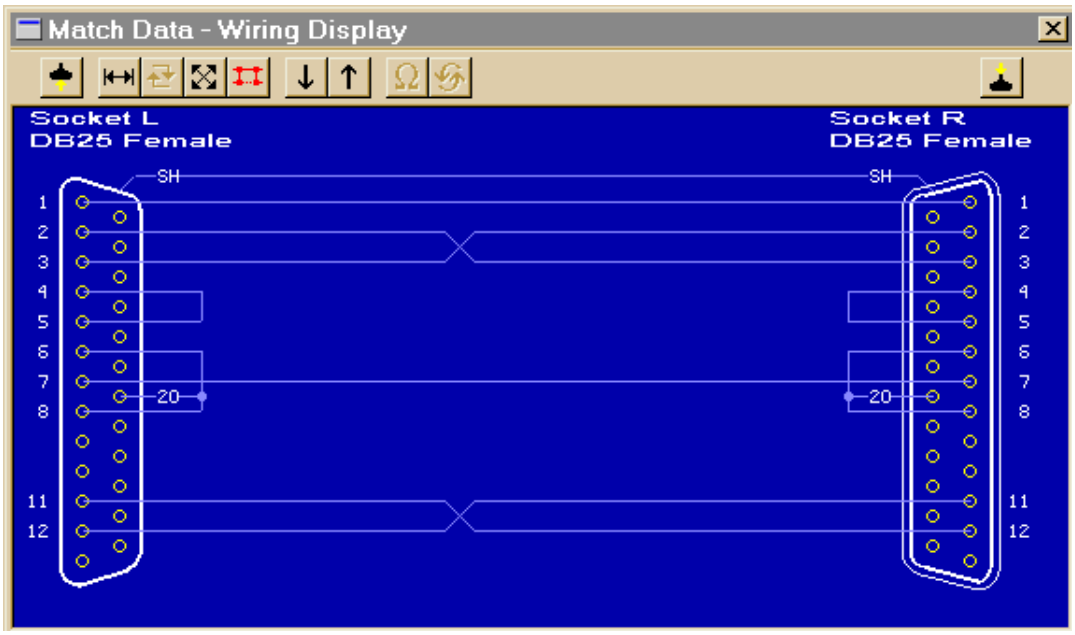
84K

Note that Pins 11 and 12 are not necessary for this null modem cable to work. As is often the case, the manufacturer of equipment that uses this cable had a proprietary application in mind. We show it here to emphasize that custom serial cables may include connections for which no purpose is clear.

IMPORTANT: This cable employs NO handshaking lines between devices. The handshake signals on each side are artificially made to appear asserted by the use of self-connects on each side of the cable (for example, between pins 4 and 5). Without hardware handshaking, you risk buffer overflow at one or both ends of the transmission unless STX and ETX commands are inserted in the dataflow by software.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

10 - DB25 Null Modem (standard handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

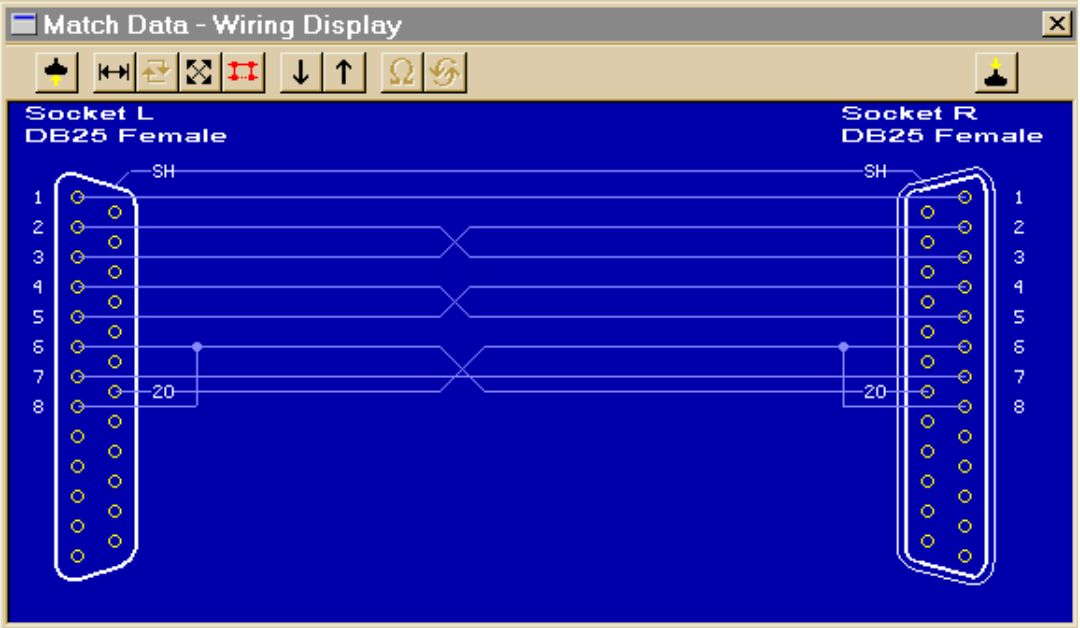
84K

The cable shown below is intended for EIA232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals not shown here.

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6) and the Request to Send (pin 5) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

11 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

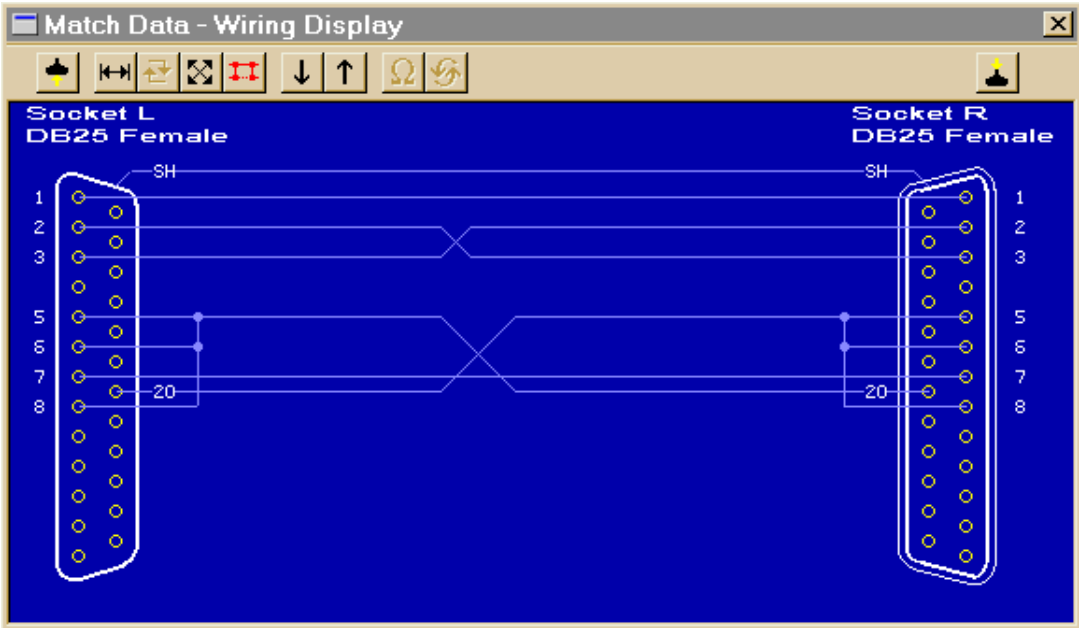
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear to Send (pin 5), DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

12 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

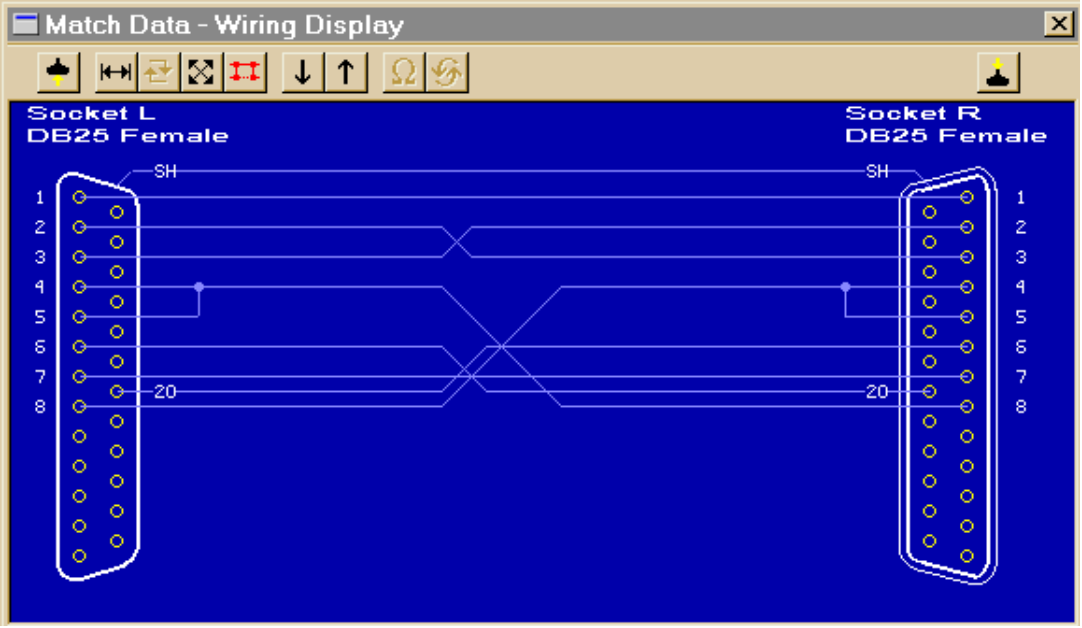
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the Request-to-Send (pin 4) on one side asserts the Clear-to-Send (pin 5) on the SAME side (self-connect) and the Carrier Detect (pin 8) on the other side. The other handshaking signals are employed in a conventional manner.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

13 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

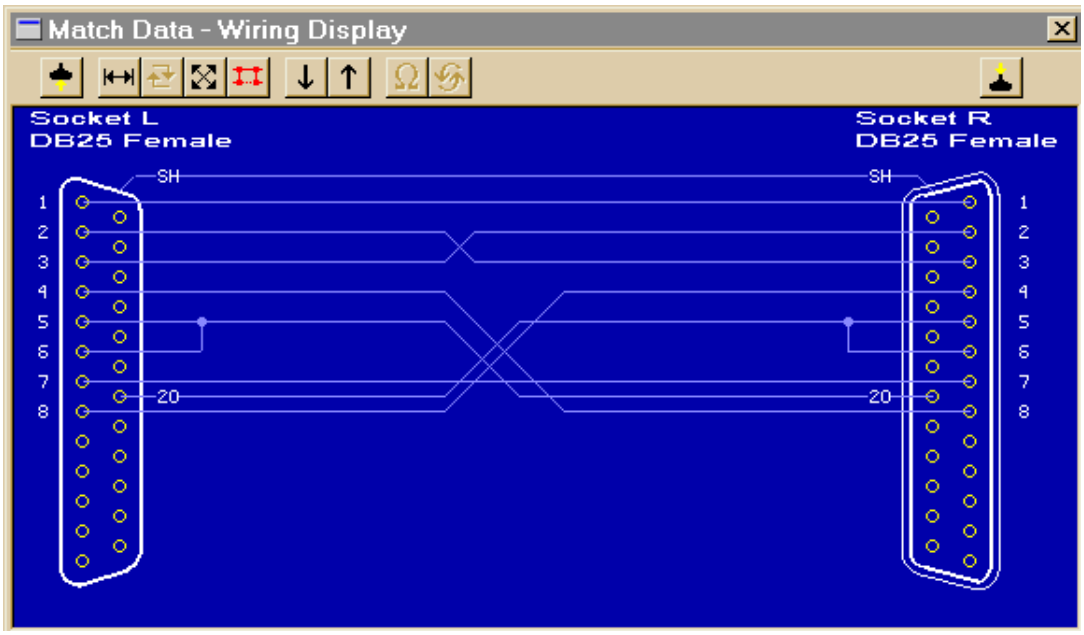
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear-to-Send (pin 5) and the DCE Ready (pin 6) on the other side. Request-to-Send (pin 4) on one side asserts Received Line Signal Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

14 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

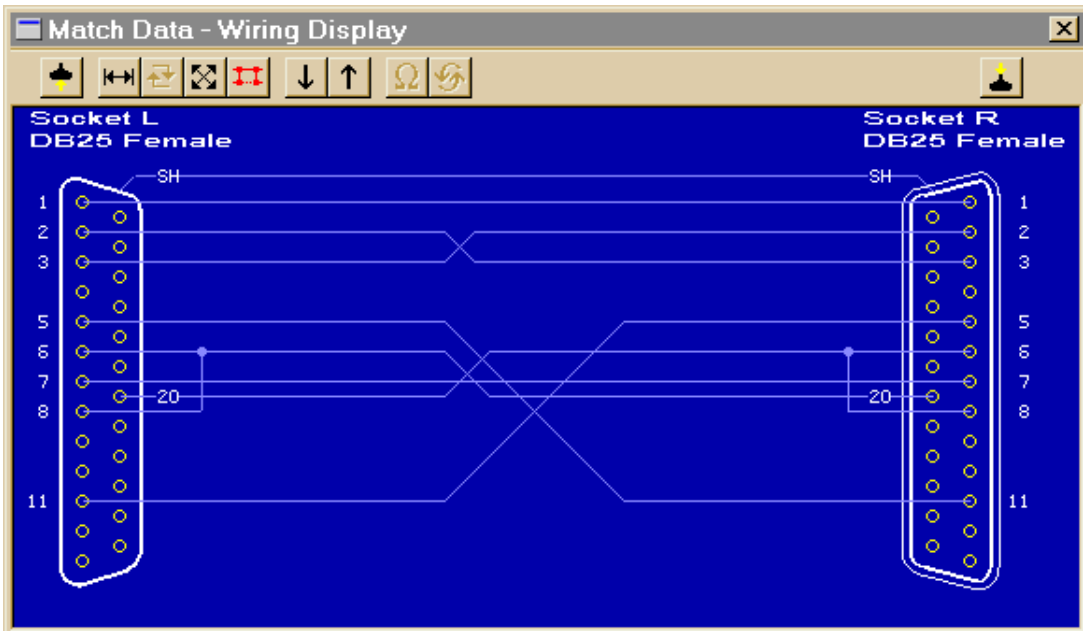
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side. Request to Send (pin 4) is unused, and Clear-to-Send (pin 5) is driven by a proprietary signal (pin 11) determined by the designer of this cable.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

15 - DB25 Null Modem Cable (synchronous communications)
Next Cable | Previous Cable || Next Topic

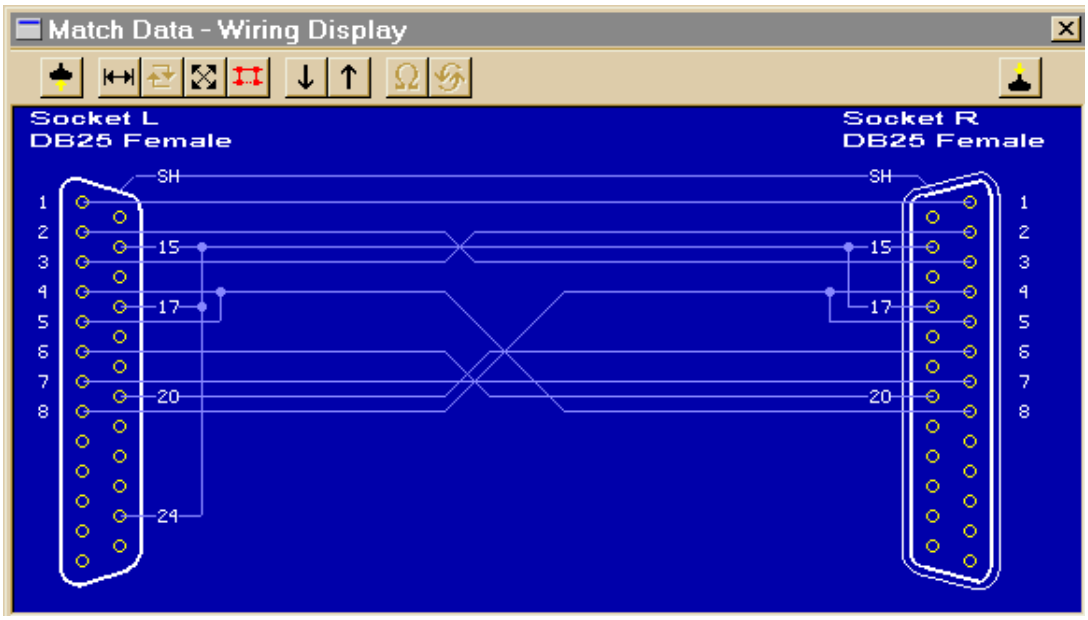
84K

This female-to-female cable is intended for **synchronous** EIA232 connections, and is designed to connect two DTE devices. It contains the standard connections of an asynchronous null modem cable, plus additional connections on pins 15, 17, and 24 for synchronous timing signals. To connect two DCE devices, use a male-to-male equivalent of this cable.

For synchronous communications, the null modem cable includes an additional conductor for timing signals, and joins pins 15, 17, and 24 on one side to pins 15 and 17 on the other. Pin 24 on the right side should connect to the timing signal source.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

16 - DB25 Null Modem Cable (unconventional, may pose risk)
(no more) | Previous Cable || Next Topic

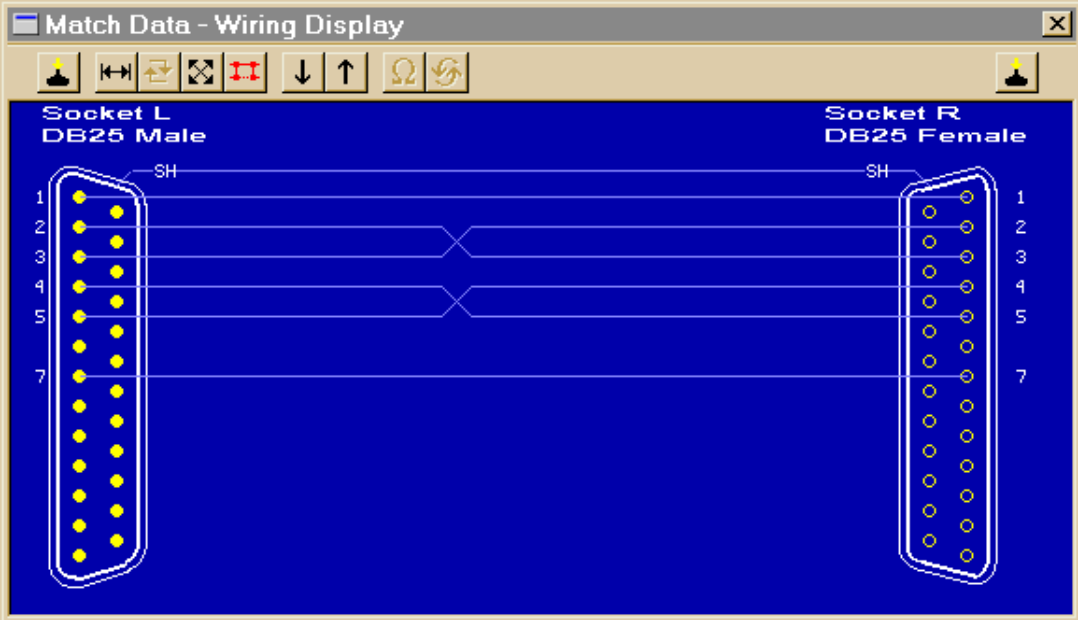
This simplified null modem cable uses only Request-to-Send (pin 4) and Clear-to-Send (pin 5) as handshaking lines; DTE Ready, DCE Ready, and Carrier Detect are not employed, so this cable should not be used with modems.

80K

CAUTION! Normally, null modem cables have the same gender on each connector (either both male for two DTE devices, or both female for two DCE devices). This cable would be used when the gender on one of the devices does not conform to the standard. However, the opposite genders imply usage as a straight through cable, and if used in that manner will not function. Further, if used as a standard null-modem between two computers, the opposite gender allows you to connect one end to the parallel port, an impermissible situation that may cause hardware damage.

Left Side: Connect to 25-pin **DTE** (computer) with Gender Changer

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

Signal Definitions
Next Topic | Previous Topic | TOC

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

Signal Ground and Shield

Pin 7, Pin 1, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 - Ground All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

Primary Communications Channel

Pin 2 - Transmitted Data (TxD) This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD) This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS) This signal is asserted (logic ’0’, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

> *NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS) This signal is asserted (logic ’0’, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

> *NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the

EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

Secondary Communications Channel
Next Topic | Previous Topic | TOC

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

Modem Status and Control Signals
Next Topic | Previous Topic | TOC

Pin 6 - DCE Ready (DSR) When originating from a modem, this signal is asserted (logic ’0’, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic ’1’, negative voltage).

IMPORTANT: If DCE Ready originates from a device other than a modem, it may be asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic ’0’, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR) This signal is asserted (logic ’0’, positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the

connection. When DTE Ready is deasserted (logic ’1’, negative voltage), the modem is switched to "on-hook" to terminate the connection.

> *IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

Pin 8 - Received Line Signal Detector (CD) (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic ’0’, positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem’s requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD) This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI) This signal is relevant when the DCE device is a modem, and is asserted (logic ’0’, positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic ’0’, positive voltage) selects the higher baud rate.

Transmitter and Receiver Timing Signals
Next Topic | Previous Topic | TOC

Pin 15 - Transmitter Signal Element Timing (TC) (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic ’1’ to logic ’0’ (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC) (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC) (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic ’1’ to logic ’0’ transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

Channel Test Signals

Next Topic | Previous Topic | TOC

Pin 18 - Local Loopback (LL) This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic ’0’, positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL) This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic ’0’, positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

Pin 25 - Test Mode (TM) This signal is relevant only when the DCE device is a modem. When asserted (logic ’0’, positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

Electrical Standards

Next Topic | Previous Topic | TOC

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic ’1’, and positive voltage represents logic ’0’. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic ’1’. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

Common Signal Ground

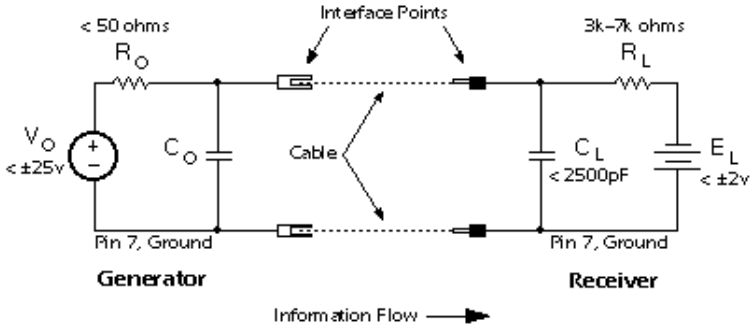
Next Topic | Previous Topic | TOC

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

NOTE: optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

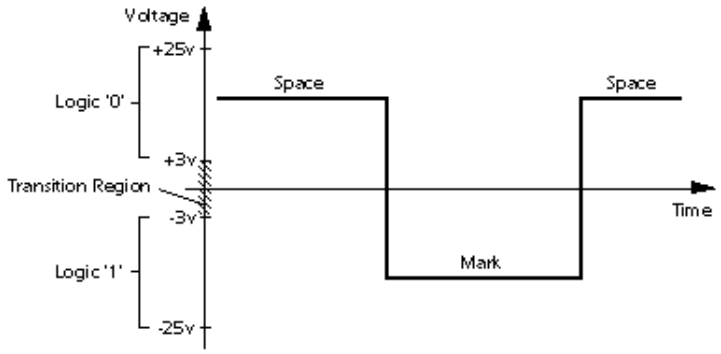
Signal Characteristics
Next Topic | Previous Topic | TOC

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "Co" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "Ro" and "Vo" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.

Signal State Voltage Assignments - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic ’1’ (the marking condition), whereas voltages of +3v to +25v are considered logic ’0’ (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern

digital designs.

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic ’1’ (mark), and +8 to +14 volts for logic ’0’ (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

> *IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of -3v for logic ’1’, and +3v for logic ’0’. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

Short-Circuit Tolerance - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of ±25 volts without sustaining damage.

> *CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the ±25-volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

Fail-Safe Signals - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic ’1’ (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.
Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic ’0’) to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic ’1’ to logic ’0’ or vice versa must abide by several requirements, as follows:

1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.

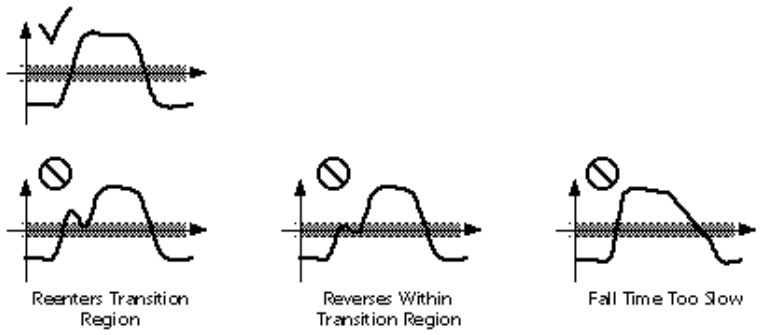
2 - For control signals, the transit time through the transition region should be less than 1ms.

3 - For Data and Timing signals, the transit time through the transition region should be

a - less than 1ms for bit periods greater than 25ms,

b - 4% of the bit period for bit periods between 25ms and 125µs,

c - less than 5µs for bit periods less than 125µs.
The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.

---

4 - The slope of the rising and falling edges of a transition should not exceed 30v/µS. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

Accepted Simplifications of the Standard
Previous Topic | TOC

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

Return to TOC

The RS232 STANDARD

A Tutorial with Signal Names and Definitions

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Send Us Your Comments . . .

Contents

What is EIA232?
Next Topic | TOC

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the

interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

Likely Problems when Using an EIA232 Interface
Next Topic | Previous Topic | TOC

During this 40-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

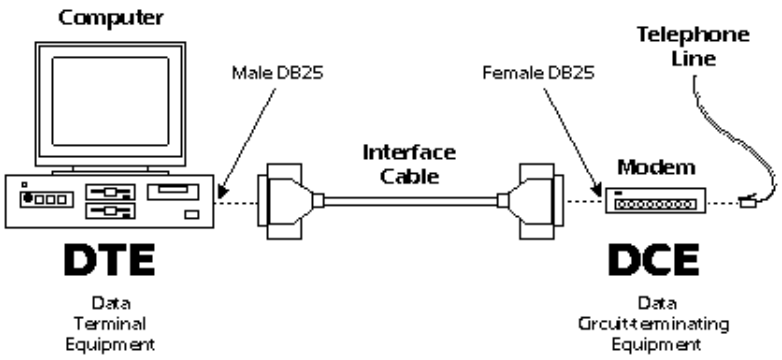
Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

Pin Assignments
Next Topic | Previous Topic | TOC

Go to DTE Pinout (looking into the computer’s serial connector)
Go to DCE Pinout (looking into the modem’s serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is

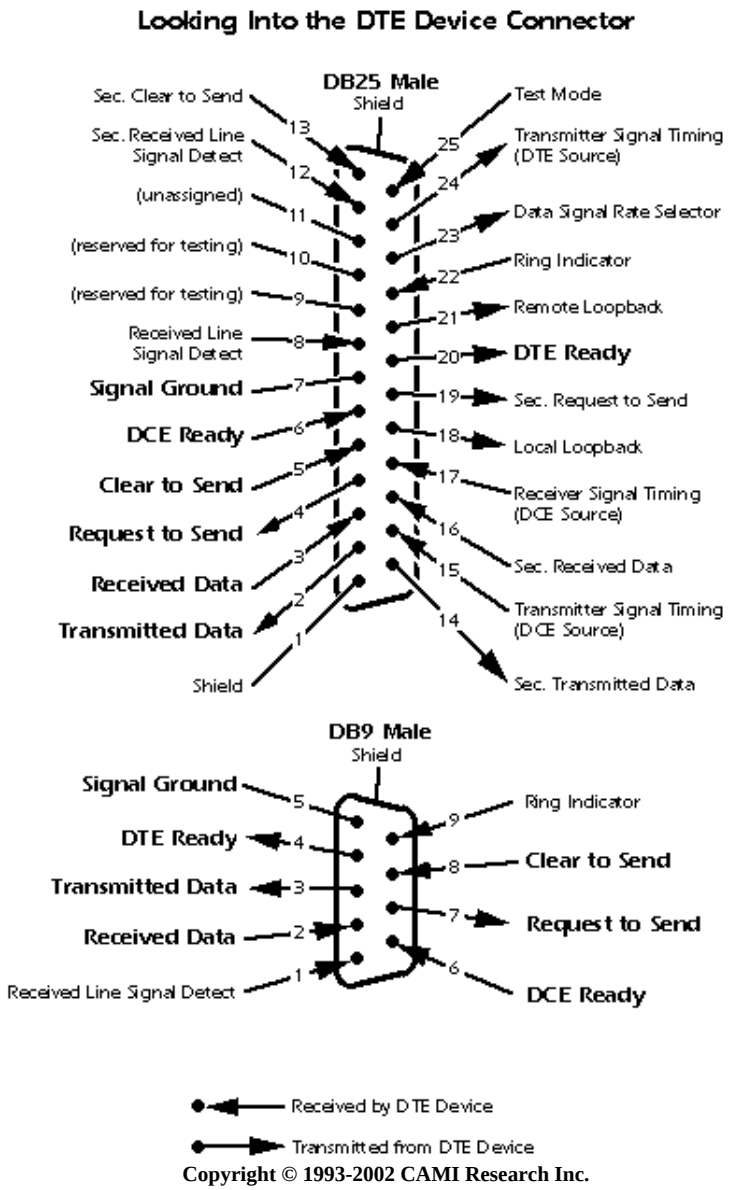
named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.

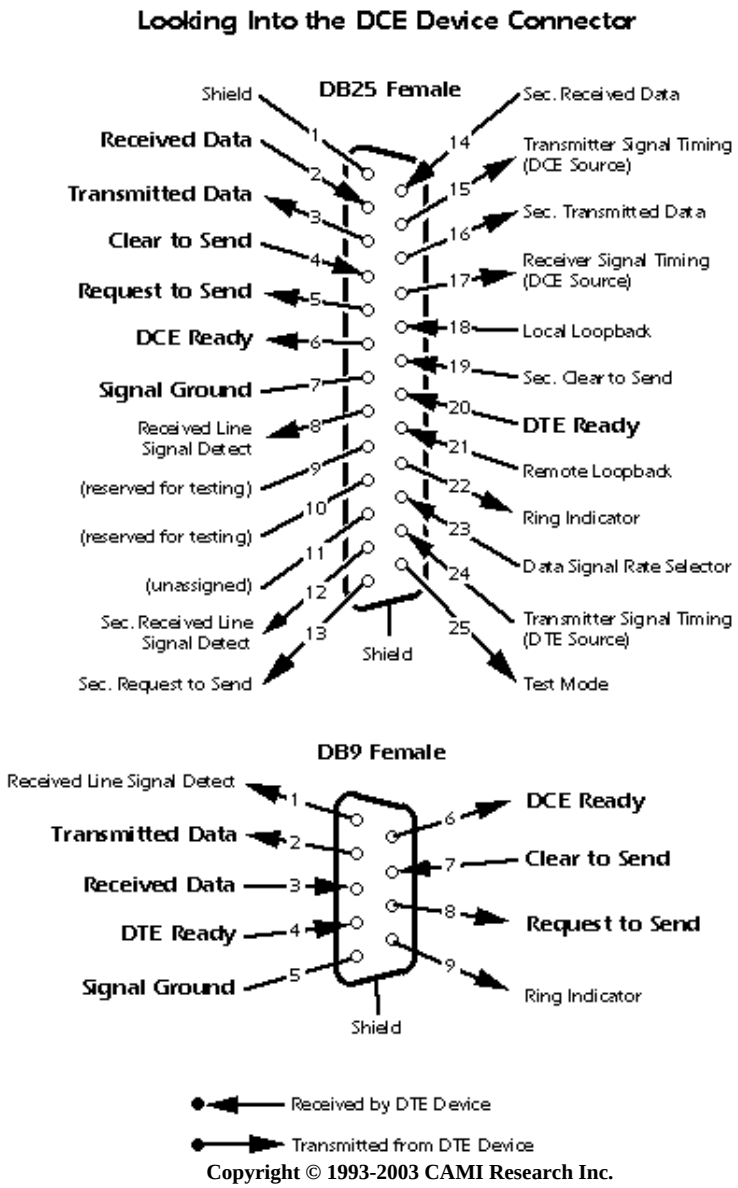
Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



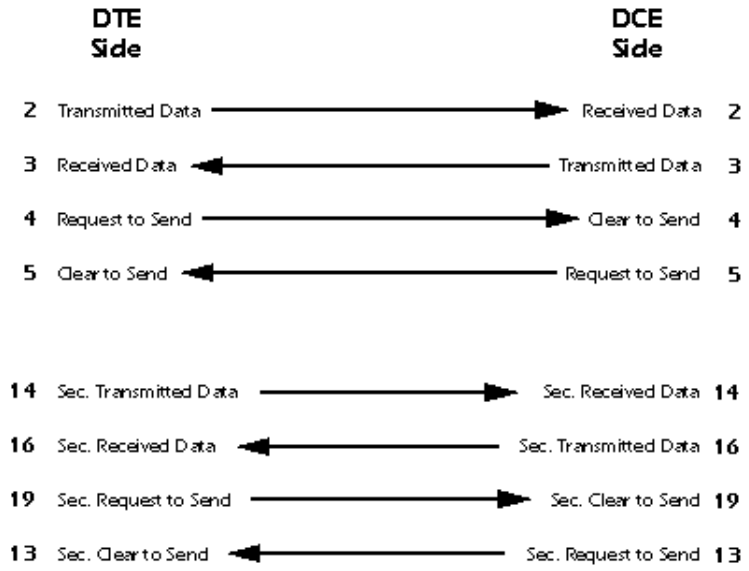
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

> ***IMPORTANT:*** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:



Cable Wiring Examples
Next Topic | Previous Topic | TOC

The following wiring diagrams come from actual cables scanned by the CableEye® PC-Based Cable Test System. CableEye's software automatically draws schematics whenever it tests a cable. Click here to learn more about CableEye.

1 - DB9 All-Line Direct Extension
Next Cable | (no previous cable) || Next Topic

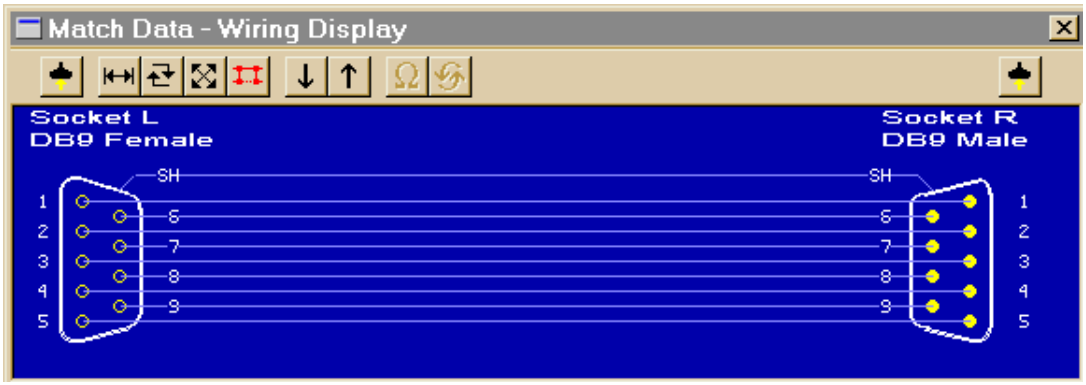
80K

This shows a 9-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 9 pins plus shield are directly extended from DB9 Female to DB9 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any device that uses a DB9 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between a computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Left Side: Connect to **DTE** (computer)

Right Side: Connect to **DCE** (modem or other serial device)



Cable image created by CableEye®

---

2 - DB9 Loopback Connector

Next Cable | Previous Cable || Next Topic

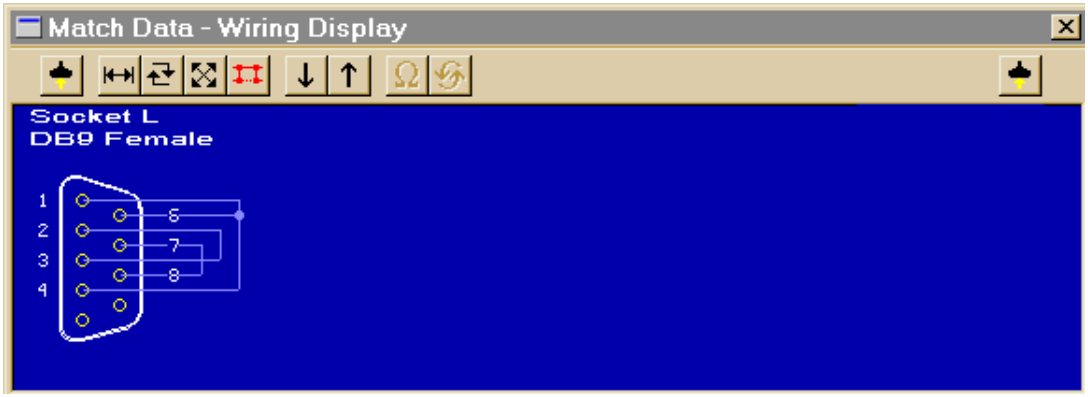
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

3 - DB9 Null Modem Cable
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

80K

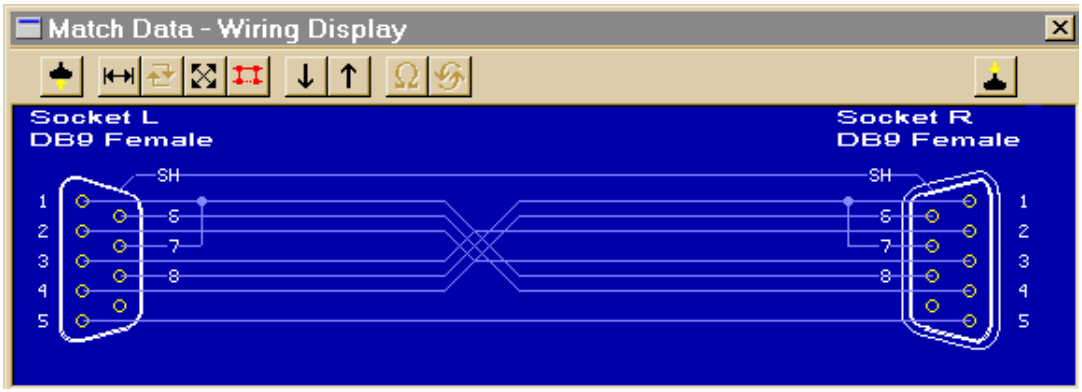
The cable shown below is intended for RS232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals, and a DB25 connector would be necessary.

NOTE: Not all null modem cables connect handshaking lines the same way. In this cable, Request-to-Send (RTS, pin 7) asserts the Carrier Detect (pin 1) on the same side and the Clear-to-Send (CTS, pin 8) on the other side of the cable.

This device may also be available in the form of an adapter.

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 9-pin **DTE** (computer)



Cable image created by CableEye®

---

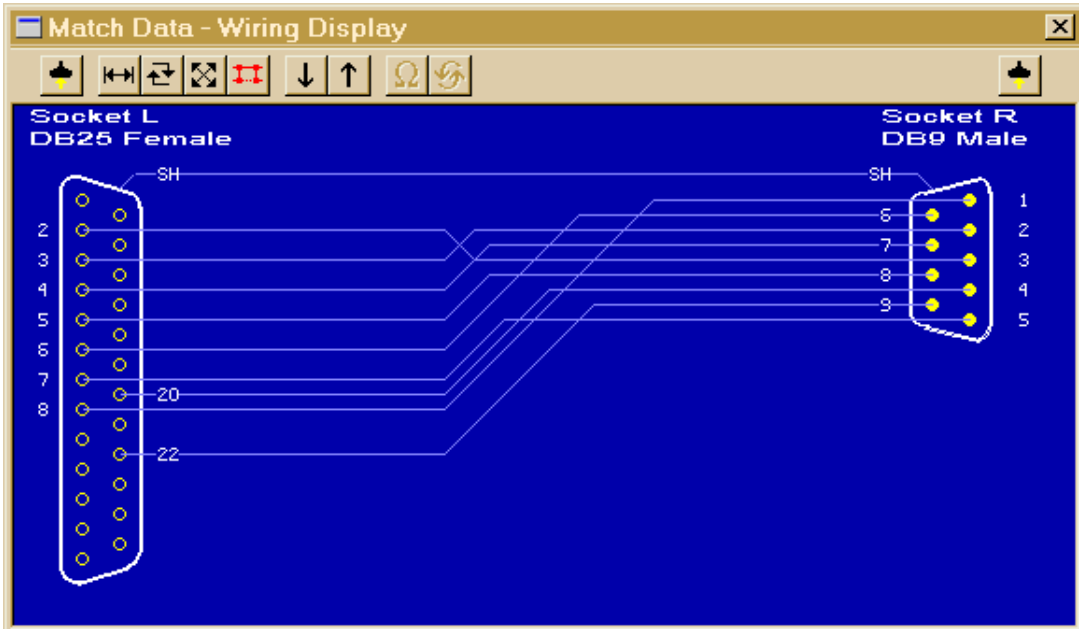
4 - DB25 to DB9 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB25 DTE side are directly mapped to the DB9 assignments for a DTE device. Use this to adapt a 25-pin COM connector on the back of a computer to mate with a 9-pin serial DCE device, such as a 9-pin serial mouse or modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

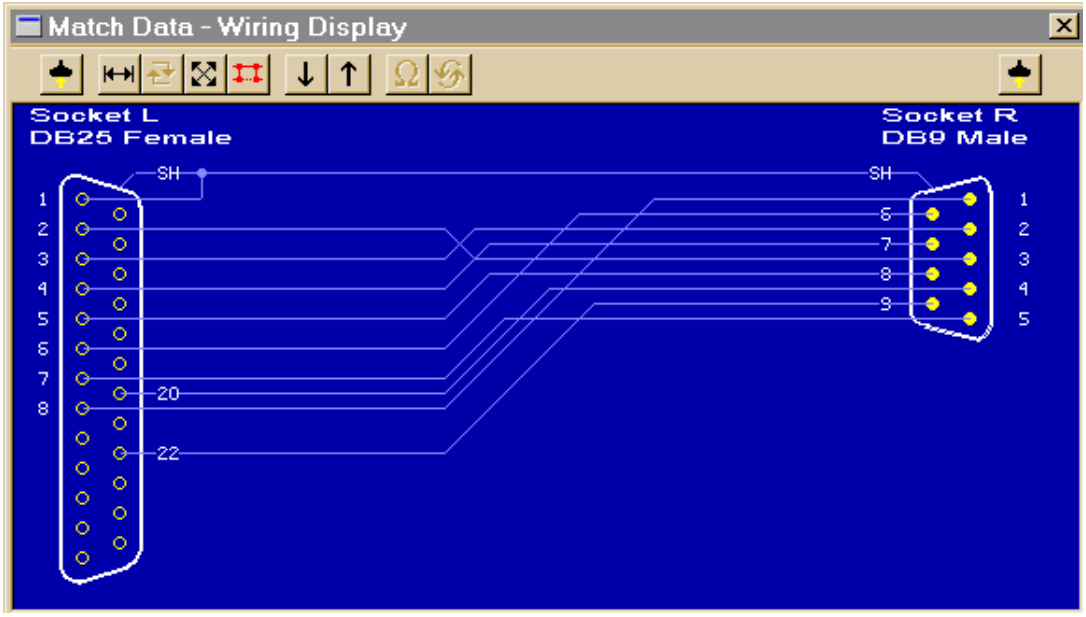
5 - DB25 to DB9 Adapter (pin 1 connected to shield)
Next Cable | Previous Cable || Next Topic

84K

This adapter has the same wiring as the previous cable (#4) except that pin 1 is wired to the connector shell (shield). Note that the cable's shield is usually a foil blanket surrounding all conductors running the length of the cable and joining the connector shells. Pin 1 of the EIA232 specification, called out as "shield", may be separate from the earth ground usually associated with the connector shells.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 9-pin **DCE** (modem)



Cable image created by CableEye®

---

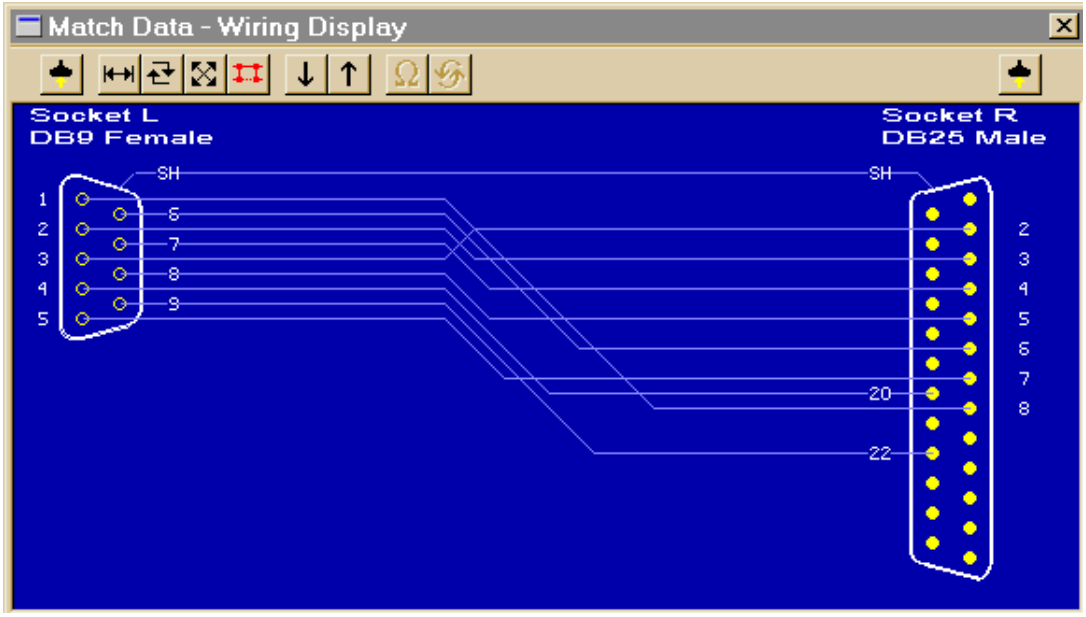
6 - DB9 to DB25 Adapter
Next Cable | Previous Cable || Next Topic

Signals on the DB9 DTE side are directly mapped to the DB25 assignments for a DTE device. Use this to adapt a 9-pin COM connector on the back of a computer to mate with a 25-pin serial DCE devices, such as a modem. This adapter may also be in the form of a cable.

80K

Left Side: Connect to 9-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

7 - DB25 All-Line Direct Extension
Next Cable | Previous Cable || Next Topic

84K

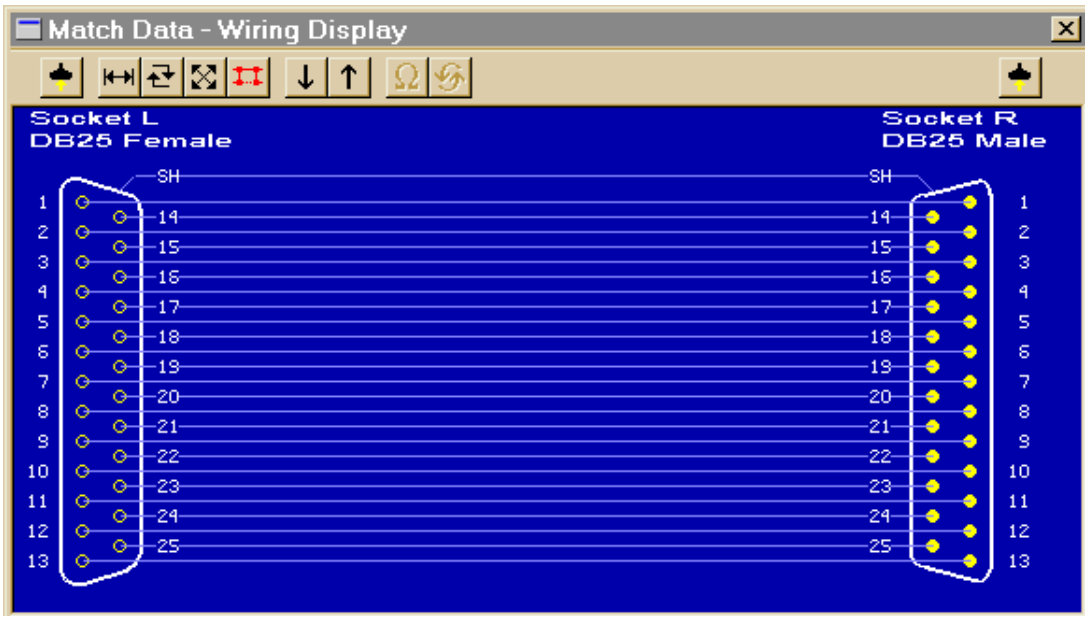
This shows a 25-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 25 pins plus shield are directly extended from DB25 Female to DB25 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any serial device that uses a DB25 connector to a PC's serial port.

This cable may also serve as an extension cable to increase the distance between computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

Caution: the male end of this cable (right) also fits a PC's parallel printer port. You may use this cable to extend the length of a printer cable, but DO NOT attach a serial device to the computer's parallel port. Doing so may cause damage to both devices.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DCE** (modem)



Cable image created by CableEye®

---

8 - DB25 Loopback Connector
Next Cable | Previous Cable || Next Topic

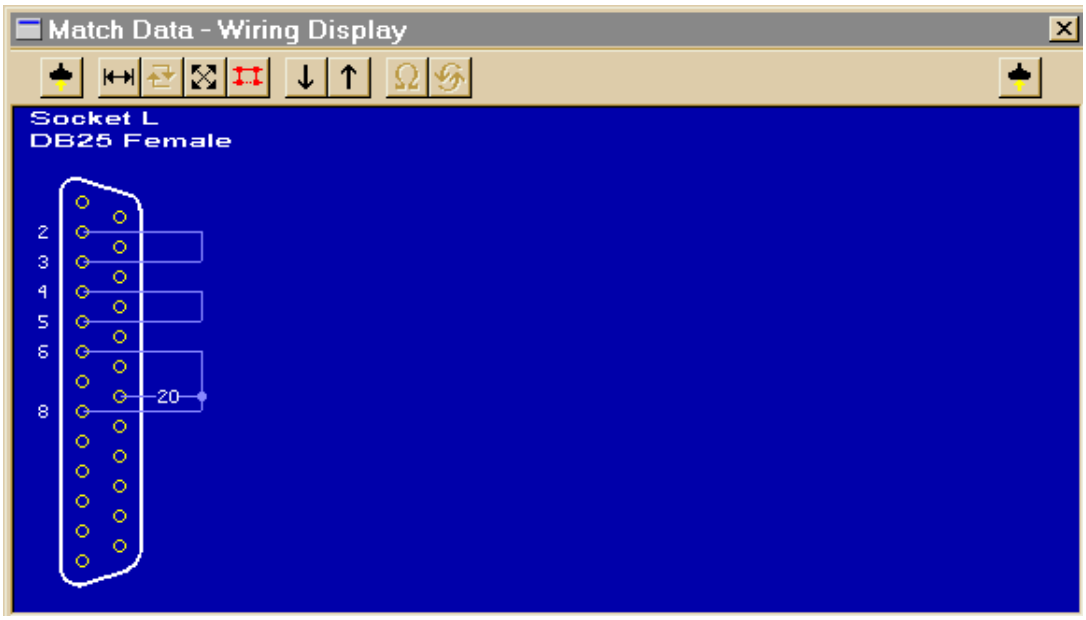
80K

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB25 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable the verify the cable.

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: (none)



Cable image created by CableEye®

---

9 - DB25 Null Modem (no handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

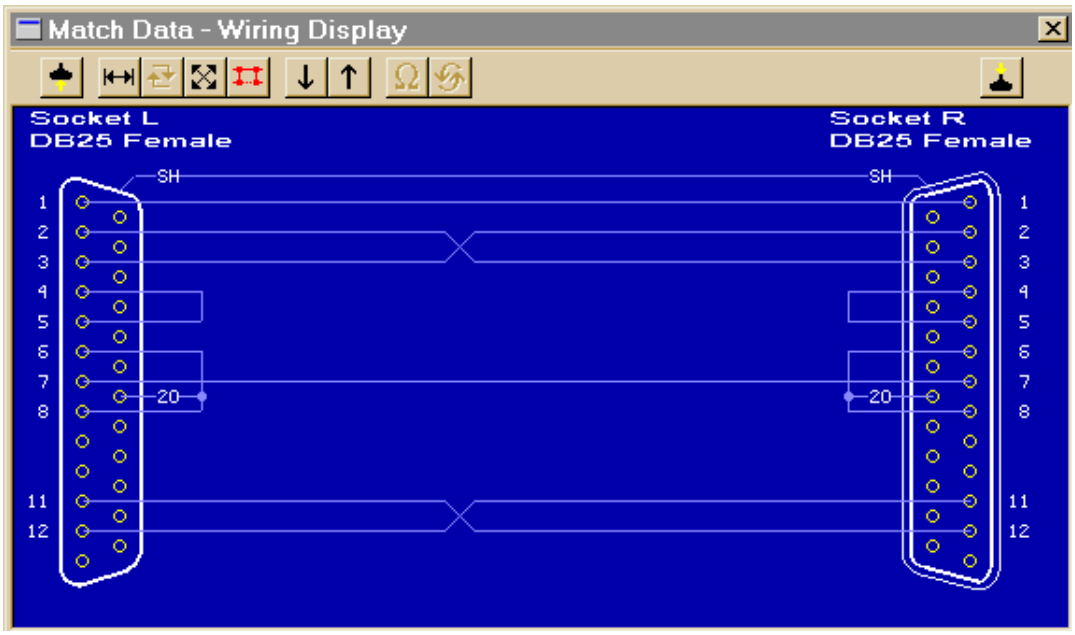
84K

Note that Pins 11 and 12 are not necessary for this null modem cable to work. As is often the case, the manufacturer of equipment that uses this cable had a proprietary application in mind. We show it here to emphasize that custom serial cables may include connections for which no purpose is clear.

IMPORTANT: This cable employs NO handshaking lines between devices. The handshake signals on each side are artificially made to appear asserted by the use of self-connects on each side of the cable (for example, between pins 4 and 5). Without hardware handshaking, you risk buffer overflow at one or both ends of the transmission unless STX and ETX commands are inserted in the dataflow by software.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

10 - DB25 Null Modem (standard handshaking)
Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

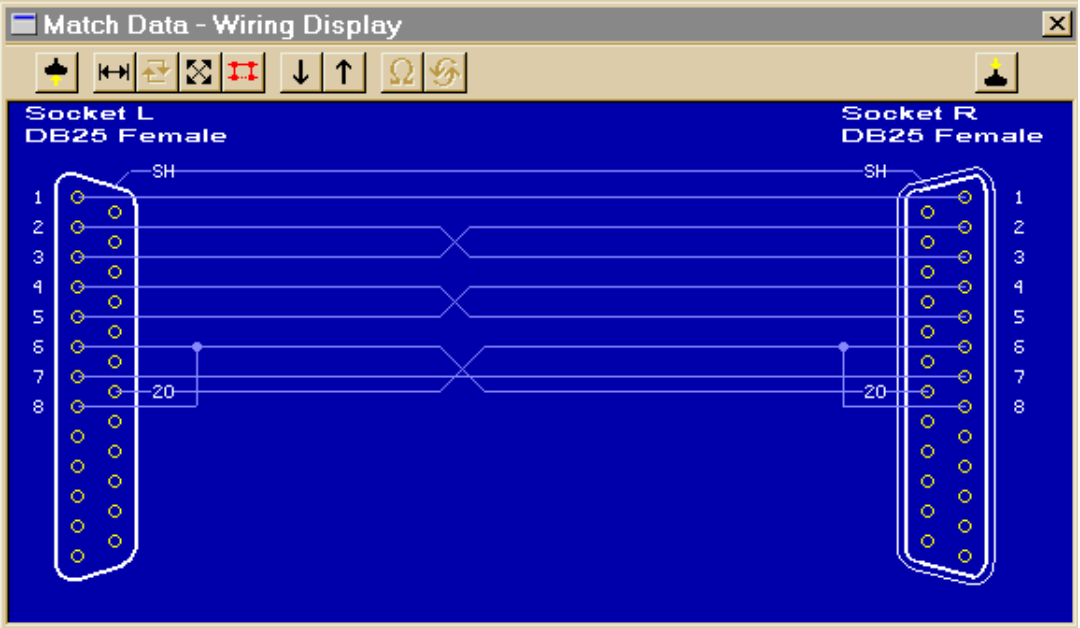
84K

The cable shown below is intended for EIA232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals not shown here.

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6) and the Request to Send (pin 5) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

11 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

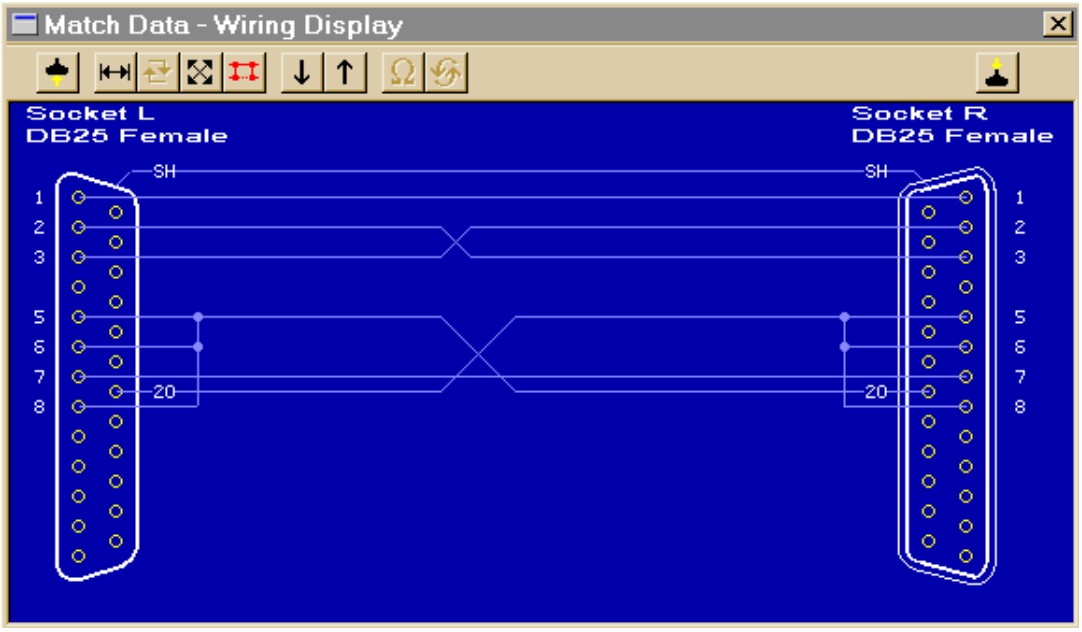
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear to Send (pin 5), DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

12 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

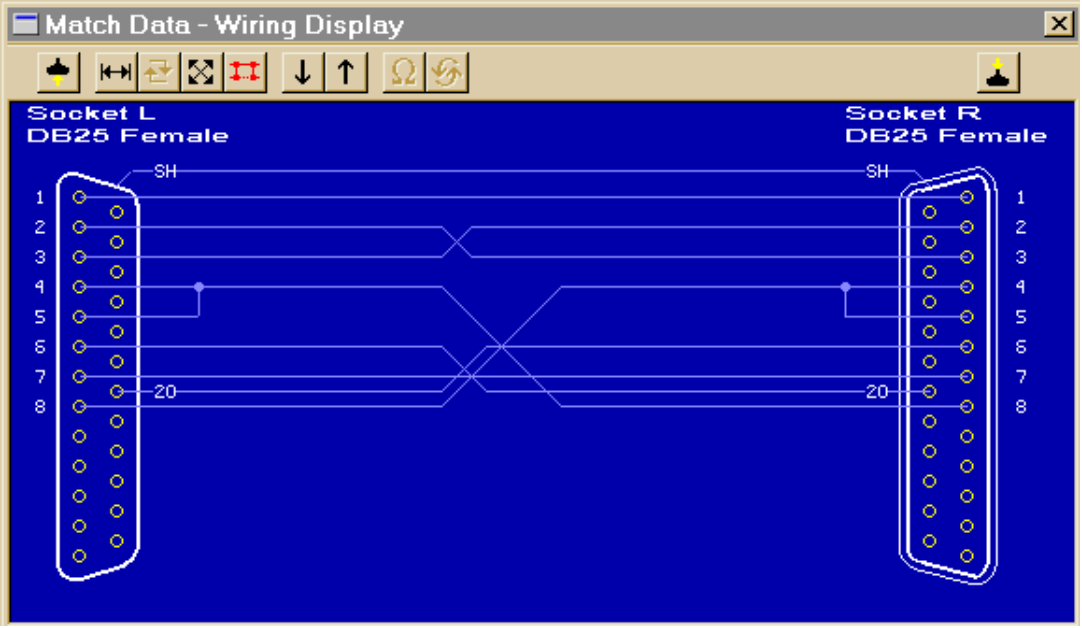
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the Request-to-Send (pin 4) on one side asserts the Clear-to-Send (pin 5) on the SAME side (self-connect) and the Carrier Detect (pin 8) on the other side. The other handshaking signals are employed in a conventional manner.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

13 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

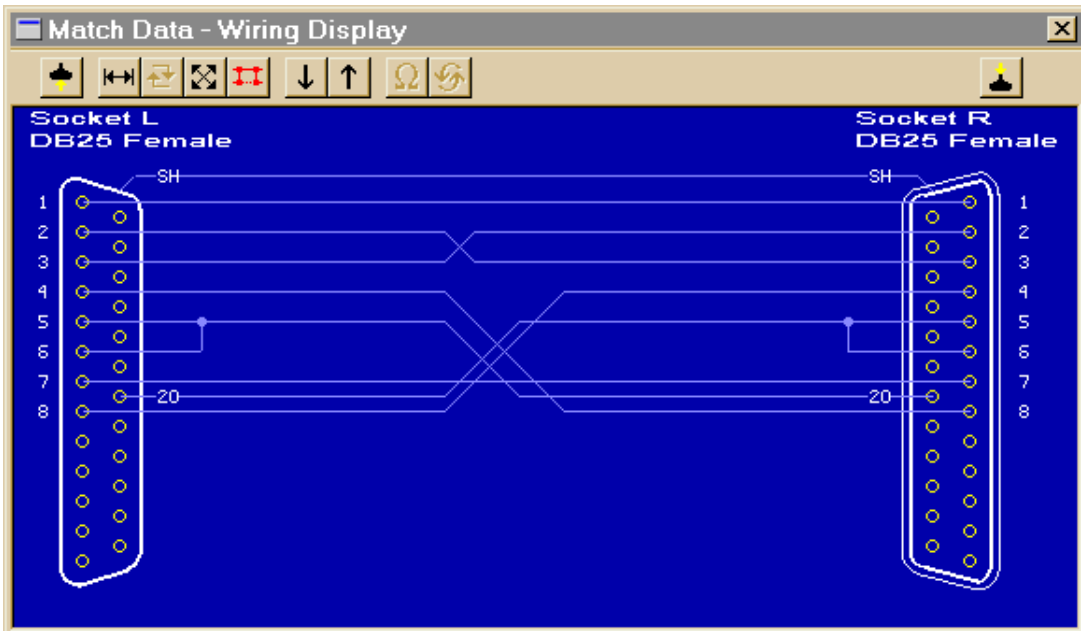
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear-to-Send (pin 5) and the DCE Ready (pin 6) on the other side. Request-to-Send (pin 4) on one side asserts Received Line Signal Detect (pin 8) on the other side.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

14 - DB25 Null Modem (unusual handshaking)
Next Cable | Previous Cable || Next Topic

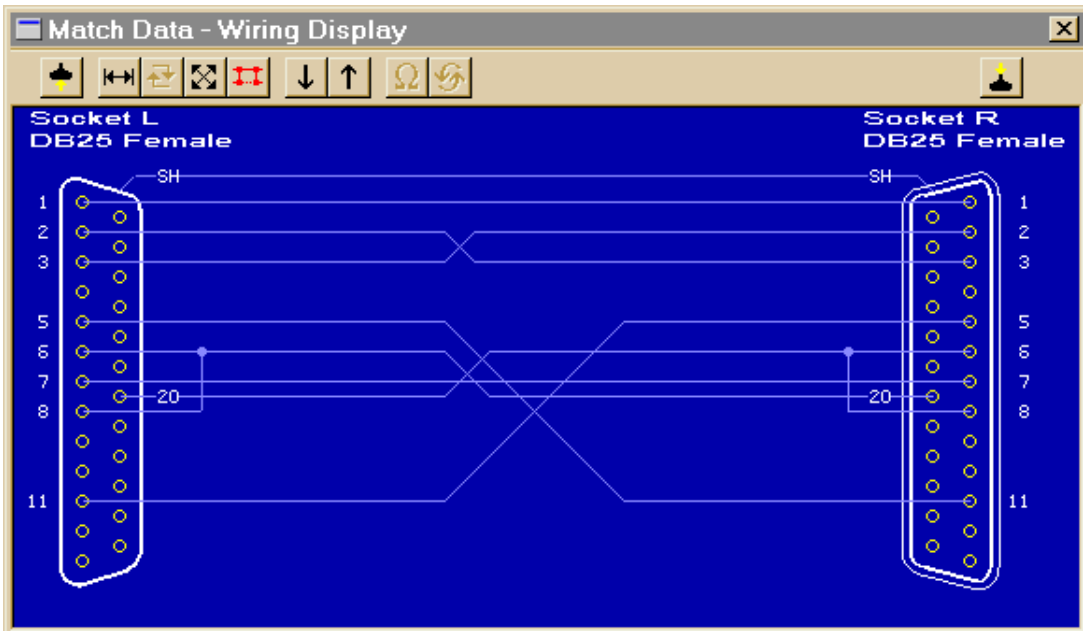
Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side. Request to Send (pin 4) is unused, and Clear-to-Send (pin 5) is driven by a proprietary signal (pin 11) determined by the designer of this cable.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

15 - DB25 Null Modem Cable (synchronous communications)
Next Cable | Previous Cable || Next Topic

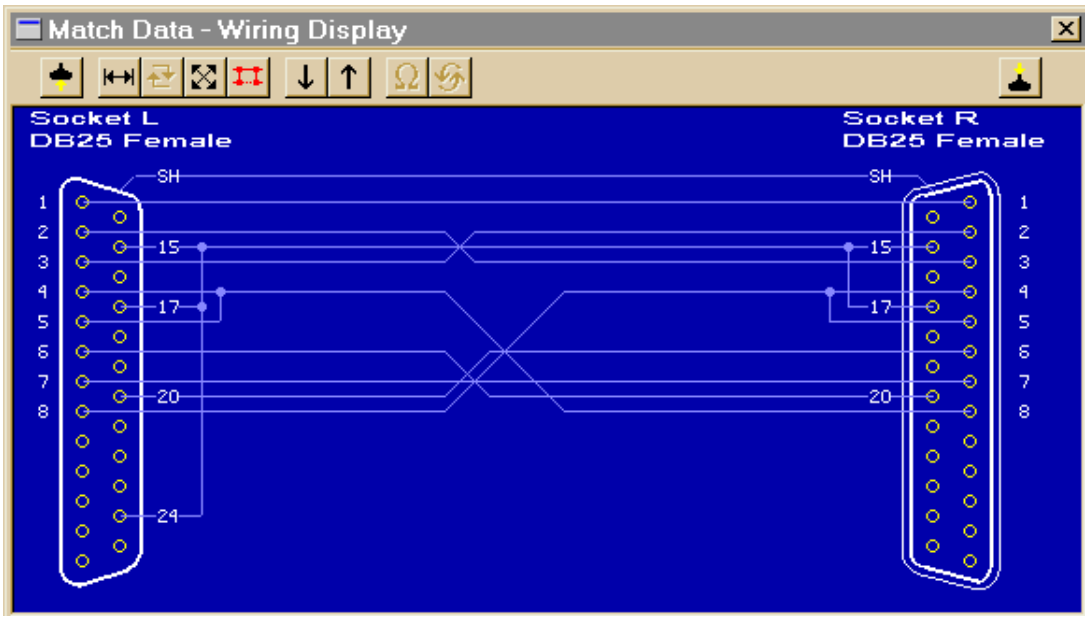
84K

This female-to-female cable is intended for **synchronous** EIA232 connections, and is designed to connect two DTE devices. It contains the standard connections of an asynchronous null modem cable, plus additional connections on pins 15, 17, and 24 for synchronous timing signals. To connect two DCE devices, use a male-to-male equivalent of this cable.

For synchronous communications, the null modem cable includes an additional conductor for timing signals, and joins pins 15, 17, and 24 on one side to pins 15 and 17 on the other. Pin 24 on the right side should connect to the timing signal source.

Left Side: Connect to 25-pin **DTE** (computer)

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

16 - DB25 Null Modem Cable (unconventional, may pose risk)
(no more) | Previous Cable || Next Topic

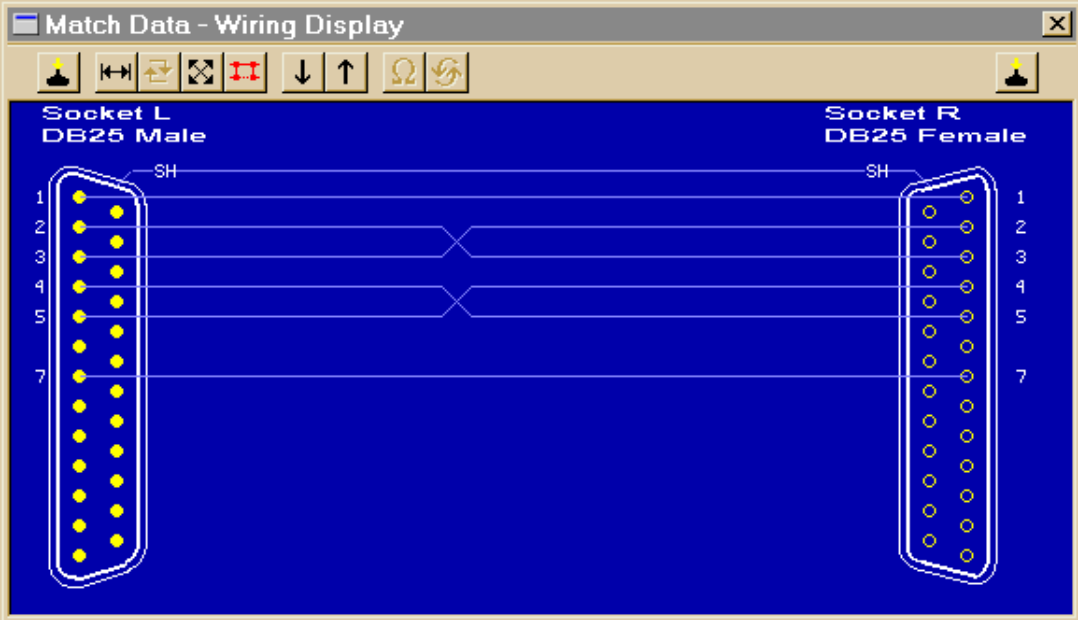
This simplified null modem cable uses only Request-to-Send (pin 4) and Clear-to-Send (pin 5) as handshaking lines; DTE Ready, DCE Ready, and Carrier Detect are not employed, so this cable should not be used with modems.

80K

CAUTION! Normally, null modem cables have the same gender on each connector (either both male for two DTE devices, or both female for two DCE devices). This cable would be used when the gender on one of the devices does not conform to the standard. However, the opposite genders imply usage as a straight through cable, and if used in that manner will not function. Further, if used as a standard null-modem between two computers, the opposite gender allows you to connect one end to the parallel port, an impermissible situation that may cause hardware damage.

Left Side: Connect to 25-pin **DTE** (computer) with Gender Changer

Right Side: Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

Signal Definitions
Next Topic | Previous Topic | TOC

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

Signal Ground and Shield

Pin 7, Pin 1, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 - Ground All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

Primary Communications Channel

Pin 2 - Transmitted Data (TxD) This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD) This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

> *NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS) This signal is asserted (logic ’0’, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

> *NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS) This signal is asserted (logic ’0’, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

> *NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the

EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

Secondary Communications Channel
Next Topic | Previous Topic | TOC

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

Modem Status and Control Signals
Next Topic | Previous Topic | TOC

Pin 6 - DCE Ready (DSR) When originating from a modem, this signal is asserted (logic ’0’, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic ’1’, negative voltage).

IMPORTANT: If DCE Ready originates from a device other than a modem, it may be asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic ’0’, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR) This signal is asserted (logic ’0’, positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the

connection. When DTE Ready is deasserted (logic ’1’, negative voltage), the modem is switched to "on-hook" to terminate the connection.

> *IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

Pin 8 - Received Line Signal Detector (CD) (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic ’0’, positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem’s requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD) This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI) This signal is relevant when the DCE device is a modem, and is asserted (logic ’0’, positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic ’0’, positive voltage) selects the higher baud rate.

Transmitter and Receiver Timing Signals
Next Topic | Previous Topic | TOC

Pin 15 - Transmitter Signal Element Timing (TC) (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic ’1’ to logic ’0’ (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC) (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC) (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic ’1’ to logic ’0’ transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

Channel Test Signals

Next Topic | Previous Topic | TOC

Pin 18 - Local Loopback (LL) This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic ’0’, positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL) This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic ’0’, positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

Pin 25 - Test Mode (TM) This signal is relevant only when the DCE device is a modem. When asserted (logic ’0’, positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

Electrical Standards

Next Topic | Previous Topic | TOC

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic ’1’, and positive voltage represents logic ’0’. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic ’1’. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

Common Signal Ground

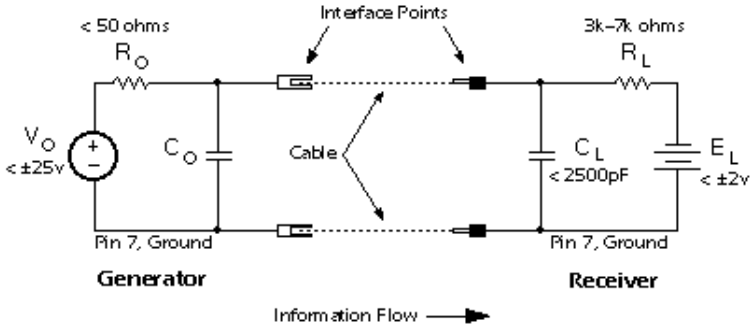
Next Topic | Previous Topic | TOC

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

NOTE: optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

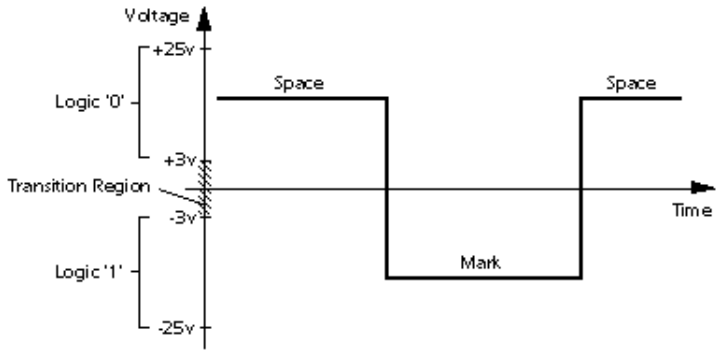
Signal Characteristics
Next Topic | Previous Topic | TOC

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "Co" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "Ro" and "Vo" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.

Signal State Voltage Assignments - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic ’1’ (the marking condition), whereas voltages of +3v to +25v are considered logic ’0’ (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern

digital designs.

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic ’1’ (mark), and +8 to +14 volts for logic ’0’ (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

> *IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of -3v for logic ’1’, and +3v for logic ’0’. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

Short-Circuit Tolerance - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of ±25 volts without sustaining damage.

> *CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the ±25-volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

Fail-Safe Signals - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic ’1’ (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.
Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic ’0’) to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic ’1’ to logic ’0’ or vice versa must abide by several requirements, as follows:

1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.

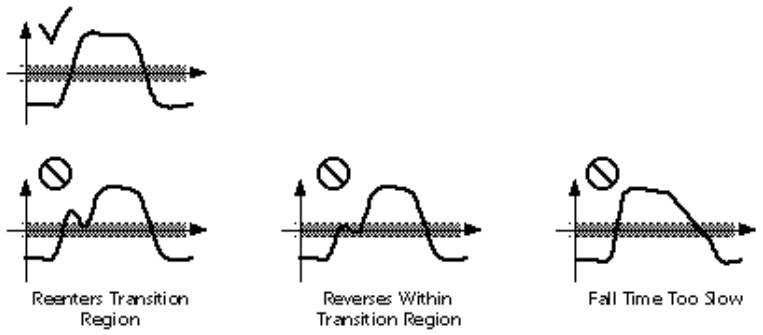
2 - For control signals, the transit time through the transition region should be less than 1ms.

3 - For Data and Timing signals, the transit time through the transition region should be

a - less than 1ms for bit periods greater than 25ms,

b - 4% of the bit period for bit periods between 25ms and 125µs,

c - less than 5µs for bit periods less than 125µs.
The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.

---

4 - The slope of the rising and falling edges of a transition should not exceed 30v/µS. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

Accepted Simplifications of the Standard
Previous Topic | TOC

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

Return to TOC

The RS232 STANDARD

A Tutorial with Signal Names and Definitions

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Send Us Your Comments . . .

Contents

What is EIA232?
Next Topic | TOC

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers, thereby fostering the benefits of

mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 30+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

Likely Problems when Using an EIA232 Interface
Next Topic | Previous Topic | TOC

During this 30-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

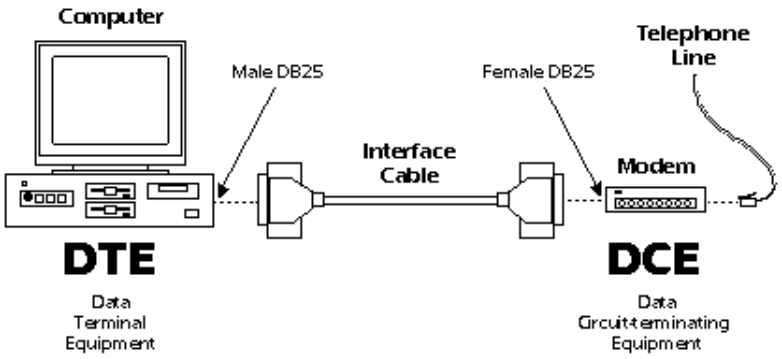
Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

Pin Assignments
Next Topic | Previous Topic | TOC

Go to DTE Pinout (looking into the computer’s serial connector)
Go to DCE Pinout (looking into the modem’s serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25

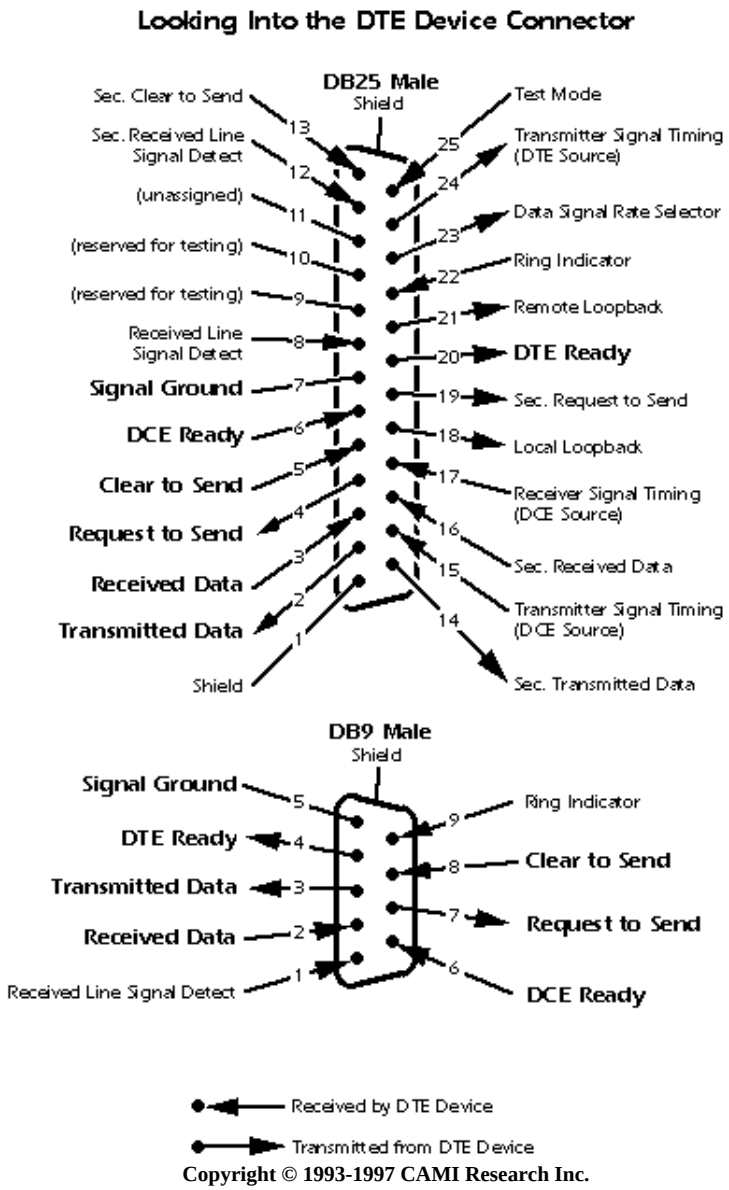
connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.

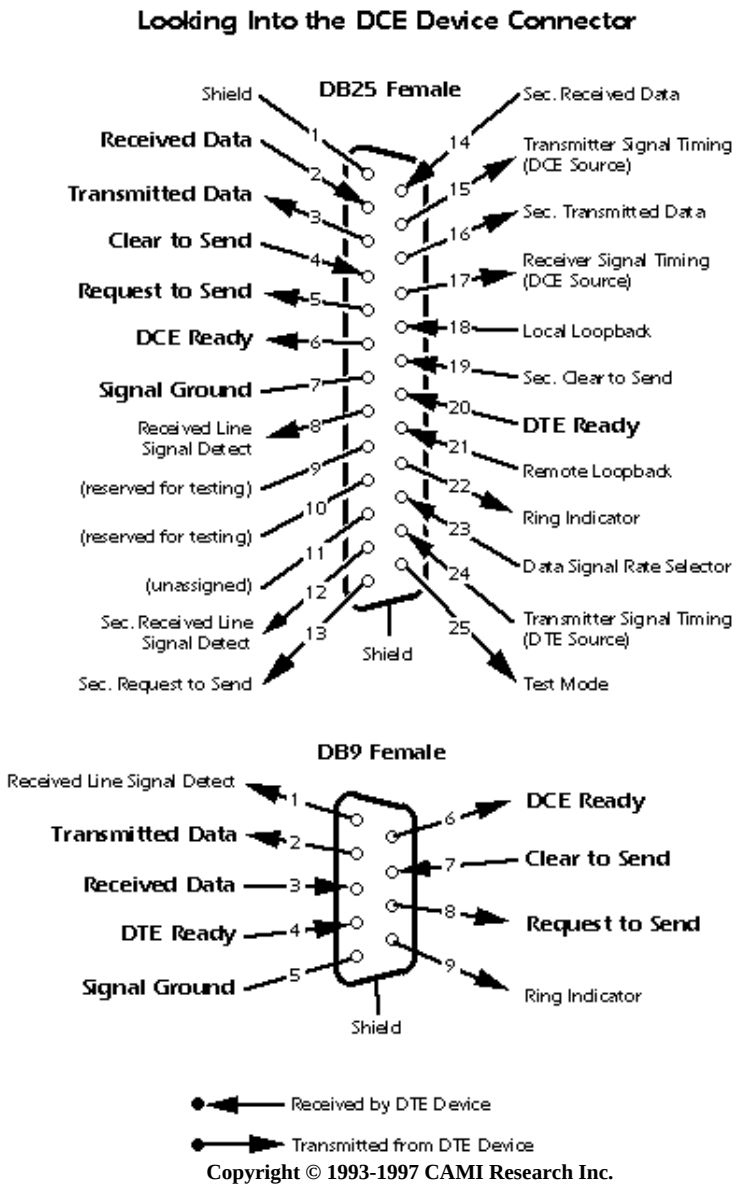
Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



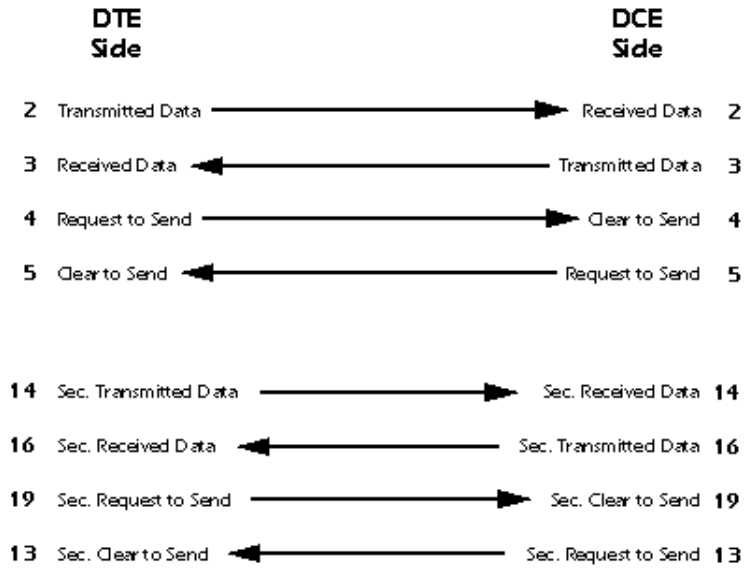
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

> ***IMPORTANT:*** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:



Signal Definitions
Next Topic | Previous Topic | TOC

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals

provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

Signal Ground and Shield
Next Topic | Previous Topic | TOC

Pin 7, Pin 1, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

Pin 7 - Ground All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

Primary Communications Channel
Next Topic | Previous Topic | TOC

Pin 2 - Transmitted Data (TxD) This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

NOTE: Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

Pin 3 - Received Data (RxD) This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic ’1’, negative voltage).

NOTE: Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

Pin 4 - Request to Send (RTS) This signal is asserted (logic ’0’, positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

NOTE: Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the

EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

Pin 5 - Clear to Send (CTS) This signal is asserted (logic ’0’, positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

NOTE: Pin 5 on the DCE device is commonly labeled "Request to Send", although by the EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

Secondary Communications Channel
Next Topic | Previous Topic | TOC

Pin 14 - Secondary Transmitted Data (STxD)

Pin 16 - Secondary Received Data (SRxD)

Pin 19 - Secondary Request to Send (SRTS)

Pin 13 - Secondary Clear to Send (SCTS)

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

Modem Status and Control Signals
Next Topic | Previous Topic | TOC

Pin 6 - DCE Ready (DSR) When originating from a modem, this signal is asserted (logic ’0’, positive voltage) when the following three conditions are all satisfied:

1 - The modem is connected to an active telephone line that is "off-hook";

2 - The modem is in data mode, not voice or dialing mode; and

3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic ’1’, negative voltage).

IMPORTANT: If DCE Ready originates from a device other than a modem, it may be

asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic ’0’, positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

Pin 20 - DTE Ready (DTR) This signal is asserted (logic ’0’, positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the connection. When DTE Ready is deasserted (logic ’1’, negative voltage), the modem is switched to "on-hook" to terminate the connection.

IMPORTANT: If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

Pin 8 - Received Line Signal Detector (CD) (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic ’0’, positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem’s requirements (perhaps due to a noisy channel).

Pin 12 - Secondary Received Line Signal Detector (SCD) This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

Pin 22 - Ring Indicator (RI) This signal is relevant when the DCE device is a modem, and is asserted (logic ’0’, positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

Pin 23 - Data Signal Rate Selector This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic ’0’, positive voltage) selects the higher baud rate.

Transmitter and Receiver Timing Signals
Next Topic | Previous Topic | TOC

Pin 15 - Transmitter Signal Element Timing (TC) (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic ’1’ to logic ’0’ (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

Pin 17 - Receiver Signal Element Timing (RC) (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

Pin 24 - Transmitter Signal Element Timing (ETC) (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic '1' to logic '0' transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

Channel Test Signals

Next Topic | Previous Topic | TOC

Pin 18 - Local Loopback (LL) This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic '0', positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

Pin 21 - Remote Loopback (RL) This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic '0', positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

Pin 25 - Test Mode (TM) This signal is relevant only when the DCE device is a modem. When asserted (logic '0', positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

Electrical Standards

Next Topic | Previous Topic | TOC

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic '1', and positive voltage represents logic '0'. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic '1'. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

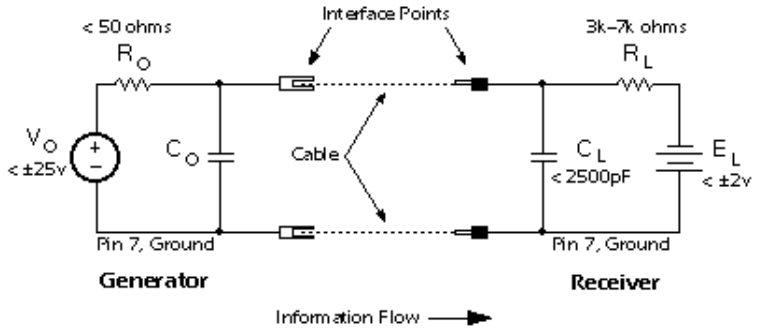
Common Signal Ground
Next Topic | Previous Topic | TOC

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

> *NOTE:* optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

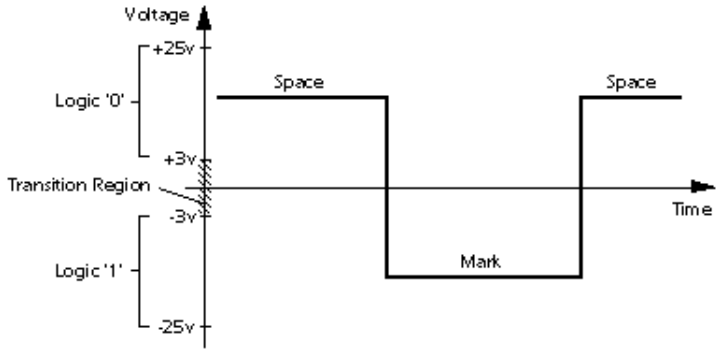
Signal Characteristics
Next Topic | Previous Topic | TOC

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "Co" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "Ro" and "Vo" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.

Signal State Voltage Assignments - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic ’1’ (the marking condition), whereas voltages of +3v to +25v are considered logic ’0’ (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern digital designs.

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic ’1’ (mark), and +8 to +14 volts for logic ’0’ (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

> *IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of &shyp;3v for logic ’1’, and +3v for logic ’0’. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

Short-Circuit Tolerance - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of ±25 volts without sustaining damage.

> *CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the ±25-volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

Fail-Safe Signals - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic ’1’ (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.
Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic ’0’) to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

Signal Timing
Next Topic | Previous Topic | TOC

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic ’1’ to logic ’0’ or vice versa must abide by several requirements, as follows:

1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.

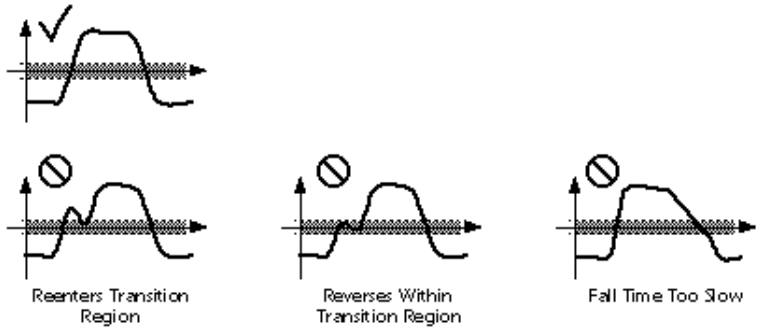
2 - For control signals, the transit time through the transition region should be less than 1ms.

3 - For Data and Timing signals, the transit time through the transition region should be

a - less than 1ms for bit periods greater than 25ms,

b - 4% of the bit period for bit periods between 25ms and 125µs,

c - less than 5µs for bit periods less than 125µs.
The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.

4 - The slope of the rising and falling edges of a transition should not exceed 30v/µS. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

Accepted Simplifications of the Standard
Previous Topic | TOC

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

Return to TOC

SDTP, PPP Serial Data Transport Protocol

Description:

Protocol suite: PPP.
Type: PPP network layer protocol.
PPP protocol: 0x0049
Working groups: pppext, Point-to-Point Protocol Extensions.

Serial Data Transport Protocol (SDTP) is used for synchronous serial data compression over a PPP link.

Before any SDTP packets may be communicated, PPP must reach the Network-Layer Protocol phase, and the SDTP Control Protocol must reach the Opened state.

The maximum length of the SDTP datagram transmitted over a PPP link is limited only by the negotiated Maximum-Frame-Size and the maximum length of the Information field of a PPP encapsulated packet. Note that if compression is used on the PPP link, this the maximum length of the SDTP datagram may be larger or smaller than the maximum length of the Information field of a PPP encapsulated packet, depending on the particular compression algorithm and protocol used.

RFC 1963, pages 1 - 3:

> This document describes a new Network level protocol (from the PPP point of view), PPP Serial Data Transport Protocol, that provides encapsulation and an associated Serial Data Control Protocol (SDCP) for transporting serial data streams over a PPP link. This protocol was developed for the purpose of using PPP's many features to provide a standard method for synchronous data compression. The encapsulation uses a header structure based on that of the ITU-T Recommendation V.120.
>
> This document is a product of the TR30.1 ad hoc committee on compression of synchronous data. It represents a component of a proposal to use PPP to provide compression of synchronous data in DSU/CSUs.
>
> In addition to providing support for multi-protocol datagrams, the Point-to-Point Protocol (PPP) has defined an effective and robust negotiating mechanism that can be used on point to point links. When used in conjunction with the PPP Compression Control Protocol and one of the PPP Compression Protocols, PPP provides an interoperable method of employing data compression on a point-to- point link.
>
> This document provides a PPP encapsulation for serial data, specifying a transport protocol, PPP Serial Data Transport Protocol (PPP-SDTP), and an associated control protocol, PPP Serial Data Control Protocol (PPP-SDCP). When these protocols are added to above mentioned PPP protocols, PPP can be used to provide compression of serial data on a point-to-point link.

This first edition of PPP-SDTP/SDCP covers HDLC-like synchronous serial data and asynchronous serial data. It does this by using a terminal adaption header based on that of ITU-T Recommendation V.120. Support may be added in the future for other synchronous protocols as the marketplace demands.

The V.120 terminal adaption header allows transported data frames to be split over several packets, supports the transport of DTE port idle and error information, and optionally supports the transport of DTE control state information.

In addition to the V.120 Header, fields can be added to the packet format through negotiation to provide support for features not included in the V.120 header. The extra fields are: a Length Field, which is used to distinguish packets in compound frames, and a Port field, which is used to provide multi-port multiplexing capability. The protocol also allows reserved bits in the V.120 header to be used to transport non-octet aligned frames and to provide a flow control mechanism.

To provide these features, PPP-SDTP permits a single frame format to be selected from several possible formats by using PPP-SDCP negotiation. The terminal adaption header can be either fixed length or variable length, to allow either simplicity or flexibility.

The default frame format places the terminal adaption header at the end of the packet. This permits optimal transmitter timelines when user frames are segmented and compression is also used in conjunction with this protocol.

Packet format:

Glossary:

V.120.
CCITT Recommendation V.120 (09/92), "Support by an ISDN of Data Terminal Equipment with V-Series Type Interfaces with Provision for Statistical Multiplexing", 1993.

RFCs:

[RFC 1963] PPP Serial Data Transport Protocol (SDTP).

Disclaimer: This description is completely unofficial. Most of the information presented here is discovered by me, Eugene Crosser, while snooping the serial line and by trial and error. I never had an official protocol description, have never seen any related software source code, and have never done reverse engineering of any related software. This description may be incomplete, inaccurate or completely wrong. You are warned.

Some information is taken from 'camediaplay' package by Jun-ichiro Itoh <itojun@itojun.org>, from the findings of Thierry Bousch <bousch%linotte.uucp@topo.math.u-psud.fr> TsuruZoh Tachibanaya <tsuruzoh@butaman.ne.jp> and from other (open) sources and not checked by me.

Serial Protocol of Some Digital Cameras

Several models of digital cameras, namely Epson, Sanyo, Agfa and Olympus cameras, seem to use the same protocol for communication with the host. Follows the description of the high-level protocol they use over the serial line.

Protocol Basics

The host and the camera exchange with data packets and individual bytes. Serial line paramaters used are: 8bit, no parity. No flow control is used. All arithmetic data is transmitted least significant byte first ("little endian").

Protocol Elements

The elementary units of the protocol are:

Initialization Byte	NUL	0x00
Action Complete Notification	ENQ	0x05
Positive Achnowledgement	ACK	0x06
Unable to Execute Command	DC1	0x11
Negative Acknowledgement, also Camera Signature	NAK	0x15
Packet	Variable length sequence of bytes	
Termination Byte		0xff

Packet structure

The packet has the following structure:

Offset	Length	Meaning
0	1	Packet type
1	1	Packet subtype/sequence
2	2	Length of data
4	variable	Data
-2	2	checksum

Known packet types are:

Type	Description
0x02	Data packet that is not last in sequence
0x03	Data packet that is last in sequence
0x1b	Command packet

Data packets that are sent in responce to a single command are numbered starting from zero. If all requested data fits in one packet, it has type 0x03 and sequence 0.

Command packet has subtype 0x43 or 0x53. Only the first command packet in a session has subtype 0x53.

Maximum length of data field in a packet is 2048 bytes, which yields in 2054 total packet length.

Checksum is a simple 16 bit arithmetic sum of all bytes in the data field. As already mentioned above, length and checksum values are transmitted least significant byte first.

Flow of Control

A communication session flow is as follows:

Host	Camera
Port speed set to 19200 baud	
Host sends init byte 0x00	
	Camera responds with signature 0x15
Host sends command packet with subtype 0x53 and "set speed" command	
	Camera sends ACK 0x06
Port speed set to the new value	
Host sends command	
	Camera responds with either ACK plus optionally "action taken" notifier or data packet sequence
Host sends ACK to every data packet	
... Command - reply cycle repeated ...	
	Camera sends 0xff and resets after a few seconds (value is model-dependant) of inactivity

If the camera does not respond to a command in reasonable time, or responds with a NAK, the command can be resent. If the camera does not provide a complete data packet in reasonable time, or the data packet is corrupt (checksum does not match), the host can request resending of the packet by sending NAK instead of ACK.

Command format and codes

Command is a sequence of bytes sent in the data field of a command packet. Command format is as follows:

Offset	Length	Description
0	1	Command code
1	1	Register number or subcode
2	variable	Optional argument

Five command codes are known:

Code	Argument	Description
0	int32	Set value of integer register
1	none	Read value of integer register
2	vdata	Take action unrelated to registers
3	vdata	Set value of vdata register
4	none	Read value of vdata register

Commands 0 and 3 are replied with a single ACK 0x06. Command 2 is replied with an ACK 0x06 followed by an "action complete" notifier 0x05. Commands 1 and 4 are replied with a sequence of data

packets, each of them must be ACK’ed by the host.

Command 0 must be issued with a 4 byte argument containg the new value for the register (bytes in "LSB first" order). Command 2 typically is issued with a single zero byte as an argument. Command 3 is issued with an argument of variable number of bytes. If this is a printable string, it should **not** include the trailing zero byte.

Camera replies to the command 1 with a single data packet containing 4 bytes of a 32bit integer (in "LSB first" order). Camera replies to the command 4 with a sequence of data packets with variable number of data bytes. Note that if a printable string is returned, it **is** terminated with a zero byte, and thus may be safely printed or otherwise treated as a normal C language character string.

Registers

The following registers are known (read/writablity info is inaccurate):

No.	Type	R/W	Description
1	int32	R/W	Resolution: 1 - Std, 2 - Hi, 3 - Ext, other values possible
2	int32	R/W	Clock in UNIX time_t format
3	int32	R/W	Shutter speed (microseconds), 0 - auto
4	int32	W	Current frame number (or animation number if hi order byte is 0xff)
5	int32	R/W	Aperture: 0 - Auto, 1 - Low, 2 - Med, 3 - ?, 4 - Hi
6	int32	R/W	Color mode: 1 - Color, 2 - B/W
7	int32	R/W	Flash mode: 0 - Auto, 1 - Force, 2 - Off, 3 - Anti RedEye, 4 - Slow sync
8	int32	R/W	Unknown (128)
9	int32	R/W	Unknown (128)
10	int32	R	No. of frames in current folder
11	int32	R	No. of frames left
12	int32	R	Length of current frame *
13	int32	R	Length of current thumbnail *
14	vdata	R	Current frame data *
15	vdata	R	Current thumbnail data *
16	int32	R	Battery capacity percentage
17	int32	R/W	Communication speed 1 - 9600 .. 5 - 115200, 6 - 230400, 256 - 9600 .. 264 - 911600 (sync?)
18	int32	R	Unknown (1)
19	int32	R/W	Bright/Contrast: 0 - Normal, 1 - Contrast+, 2 - Contrast-, 3 - Brightnes+, 4 - Brightnes-
20	int32	R/W	White balance: 0 - Auto, 1 - Sunny, 2 - Incandescent, 3 - Fluorescent, 5 - Flash, 6- White preset, 255 - Cloudy
21	vdata	R/W	Unused

22	vdata	R/W	Camera I.D.
23	int32	R/W	Autoshut on host timer (seconds)
24	int32	R/W	Autoshut in field timer (seconds)
25	vdata	R/W	Serial No. (string)
26	vdata	R	Version
27	vdata	R/W	Model
28	int32	R	Available memory left
29	vdata	R/W	Upload image data to this register
30	int32	W	LED: 0 - Off, 1 - On, 2 - Blink
31	vdata	R	Unknown ("\0")
32	int32	R	Put "magic spell" 0x0FEC000E here before uploading image data
33	int32	R/W	Focus mode: 1 - Macro, 2 - Normal, 3 - Infinity/fisheye
34	int32	R	Operation mode: 1 - Off, 2 - Record, 3-Play, 6-Thumbnail
35	int32	R/W	LCD brightness 1 to 7
36	int32	R	Unknown (3)
37	vdata	R	Unknown ("\0")
38	int32	R	LCD autoshut timer (seconds)
39	int32	R	Protection state of current frame *
40	int32	R	True No. of frames taken
41	int32	R/W	LCD date format: 1 - ’YY MM DD, 2 - DD MM ’HH
42	vdata	R	Unknown ("")
43	vdata	R	Audio data description block * 0: expanded .wav length 1: compressed .wav length 3: Unknown (0) 4: Unknown (0) 5: Unknown (0) 6: Unknown (0) 7: Unknown (0)
44	vdata	R	Audio data *
45	vdata	R	Unknown ("")
46	vdata	R	Camera summary data: 32 bytes with copies of 8 other registers 0: Reg 1 (Resolution) 1: Reg 35 (LCD brightness) or Reg 7 (Flash mode) 2: Reg 10 (Frames taken) or Unknown 3: Unknown (0) or Unknown 4: Unknown (0) or Reg 16 (Battery capacity) 5: Unknown (0) or Reg 10 (Frames taken) 6: Unknown (0) or Reg 11 (Frames left) 7: Number of animations taken

47	vdata	R	Picture summary data: 32 bytes or 8 int32's * 0: Hi order byte: unknown, next 3 bytes: Length of current image 1: Length of current thumbnail 2: Audio data length (expanded) 3: Resolution 4: Protection state 5: TimeDate 6: Unknown (0) 7: Animation type: 1 - 10ms, 2 - 20ms
48	vdata	R	Manufacturer
49	vdata	R	Unknown ("")
50	int32	R	Unknown (0)
51	int32	R/W	Card detected: 1 - No, 2 - Yes
52	vdata	R/W	Unknown ("")
53	int32	R/W	Language: 3 - english, 4 - french, 5 - german, 6 - italian, 8 - spanish, 10 - dutch
54-59	vdata	R	Unknown ("")
60	int32	R	True No. of frames taken
61-68	vdata	R	Unknown ("")
69	vdata	R	Exposure Compensation 8 bytes 0: compensation value -20 to +20 1: 0 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
70	int32	R/W	Exp. meter: 2 - Center weighted, 3 - Spot, 5 - Multi element matrix
71	vdata	R/W	Effective zoom in tenths of millimeters: 8 bytes 0: LSB 1: MSB 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
72	int32	R/W	Bitmap: 1 - AEL/WBL, 2 - Fisheye, 4 - Wide, 8 - Manual zoom, 16 - B/W, 256 - 1.25x, 512 - 1.6x, 768 - 2.0x, 1024 - 2.5x, 1280 - off
73-76	vdata	R	Unknown ("")
77	int32	W	Size of data packet from camera (default 0x800)
78	vdata	R	Unknown ("")

79	vdata	R	Filename of current frame *
80-81	vdata	R	Unknown ("")
82	int32	W	Unknown (enable folder features? Write 60 here)
83	int32	R/W	Folder navigation When read, return number of folders on the card. When written **without** data, reset folder system (?) Or select current folder by its number
84	vdata	R/W	Current folder name (may read or set)
85	vdata	R	Unknown ("")
86	int32	R/W	Digital zoom; 0 - 1X, otherwise zoom factor x 100 (i.e. in percent)
87-90	vdata	R	Unknown ("")
91	vdata	R	Current folder I.D. and name

*** Note:** Marked registers only become useful for reading after setting register 4. If value of 0 assigned to register 4 after doing action 5, subsequent retrieval of picture data gives the "live preview".

For command 2, the second byte is action code not register number. The following action codes are known:

Code	Argument	Description
0	single zero byte	Erase last picure
1	single zero byte	Erase all picures (but not animations)
2	single zero byte	Take picture
4	single zero byte	Finish session immediately
5	single zero byte	Take preview snapshot (retreivable as frame zero)
6	single byte	Calibration / testing. Arg value: 1 Calibrate autofocus 3 Test zoom/exposure 4-6 Store 0 in Reg 32 9 Load LCD Brightness (0-31) from Reg 32 10 Load LCD size (25 for Nikon Coolpix 950) from Reg 32 11 LCD Saturation (0-32) from Reg 32 13 LCD Red-Green (0-32) from Reg 32 14 LCD Blue (0-32) from Reg 32 15 Store -1 in Reg 32 16 Multi shot (locks up if lcd is on) 17 Take picture 18 Store -1 in Reg 32 20-23 locks up if lcd is on 24-255 Store -1 in Reg 32
7	single zero byte	Erase current frame *

8	single byte	Switch LCD mode. Arg value: 1 - Off 2 - Record 3 - Play (show current frame fullscreen) 4 - preview thumbnails (?) 5 - Thumbnail view (smaller?) 6 - Thumbnail view (larger?) 7 - Next 8 - Previous
9	single byte	Set protection state of current frame to the value of parameter (binary 0 or 1)*
11	single zero byte	Store freshly uploaded image into NVRAM (see appendix A)
12	single byte	LCD test. Arg value: 0 - white 1 - gray 2 - black 3 - red 4 - green 5 - blue 6 - test pattern

*** Note:** actions 7 and 9 only useful after setting register 0x04.

Appendix A

```
Date: Sun, 14 Jul 2002 01:28:39 +0200 (CEST)
From: =?iso-8859-1?Q?Peter_=C5strand?= <astrand(at)lysator.liu.se>
To: allyn(at)fratkin.com, <wolfgang(at)charlotte.wsrcc.com>, <crosser(at)average.org
Subject: Upload on Olympus C-860L


FYI.

Tonight, I've been struggling with uploading arbitrary pictures to my
Olympus C-860L. I've finally found out that for the camera to accept the
picture, to two conditions must be met:

1) The subsampling must be 2x1, 1x1, 1x1
2) The EXIF info must be just like the pictures the camera itself
produces.

So, I've made a small script to fix this. Feel free to include it in FAQs
and/or photopc dists.

/more/data/pics/olympus-reference-pic.jpg is just some picture taken with
the camera.

photopc-upload-all:

#!/bin/sh

TMPFILE='mktemp /tmp/photopc-upload.XXXXXX' || exit 1
```

```
for file in $@; do
    echo Converting $file...
    djpeg $file | cjpeg -sample 2x1 > $TMPFILE
    jhead -te /more/data/pics/olympus-reference-pic.jpg $TMPFILE
    echo Uploading $file...
    photopc upload $TMPFILE
    sleep 2;
done

rm -f $TMPFILE


--
/Peter Åstrand <astrand(at)lysator.liu.se>
```

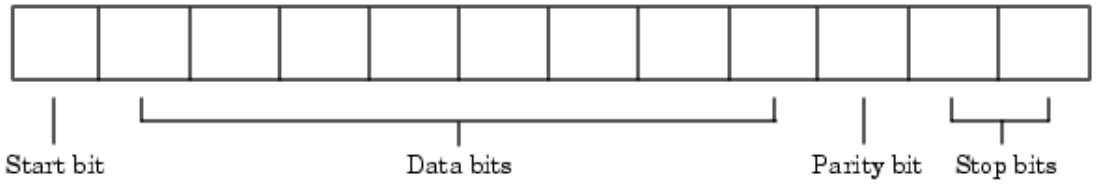
Appendix B

Some Nikon models support an extension to the protocol described above, specifically designed for remote control units. This protocol allows to control zoom, emulate half-depress of the shutter release button, bulb operation and possibly more. <*vladimir.vyskocil(at)wanadoo.fr*> compiled a partial description of this protocol, available here.

Please mail your corrections/additions to <*crosser at average dot org*>
See http://photopc.sourceforge.net/ for possible updates.

Serial Data Format

The serial data format includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well. The diagram below illustrates the serial data format.



The format for serial port data is often expressed using the following notation

- number of data bits - parity type - number of stop bits

For example, 8-N-1 is interpreted as eight data bits, no parity bit, and one stop bit, while 7-E-2 is interpreted as seven data bits, even parity, and two stop bits.

The data bits are often referred to as a *character* because these bits usually represent an ASCII character. The remaining bits are called *framing bits* because they frame the data bits.

Bytes Versus Values

The collection of bits that comprise the serial data format is called a *byte*. At first, this term might seem inaccurate because a byte is 8 bits and the serial data format can range between 7 bits and 12 bits. However, when serial data is stored on your computer, the framing bits are stripped away, and only the data bits are retained. Moreover, eight data bits are always used regardless of the number of data bits specified for transmission, with the unused bits assigned a value of 0.

When reading or writing data, you might need to specify a *value*, which can consist of one or more bytes. For example, if you read one value from a device using the `int32` format, then that value consists of four bytes. For more information about reading and writing values, refer to Writing and Reading Data.

Synchronous and Asynchronous Communication

The RS-232 standard supports two types of communication protocols: synchronous and asynchronous.

Using the synchronous protocol, all transmitted bits are synchronized to a common clock signal. The two devices initially synchronize themselves to each other, and then continually send characters to stay synchronized. Even when actual data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each bit that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required.

Using the asynchronous protocol, each device uses its own internal clock resulting in bytes that are transferred at arbitrary times. So, instead of using time as a way to synchronize the bits, the data format is used.

In particular, the data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word. The requirement to send these additional bits causes asynchronous communications to be slightly slower than synchronous. However, it has the advantage that the processor does not have to deal with the additional idle characters. Most serial ports operate asynchronously.

> **Note** When used in this guide, the terms "synchronous" and "asynchronous" refer to whether read or write operations block access to the MATLAB command line. Refer to Controlling Access to the MATLAB Command Line for more information.

How Are the Bits Transmitted?

By definition, serial data is transmitted one bit at a time. The order in which the bits are transmitted is given below:

1. The start bit is transmitted with a value of 0.
2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB).
3. The parity bit (if defined) is transmitted.
4. One or two stop bits are transmitted, each with a value of 1.

The number of bits transferred per second is given by the *baud rate*. The transferred bits include the start bit, the data bits, the parity bit (if defined), and the stop bits.

Start and Stop Bits

As described in Synchronous and Asynchronous Communication, most serial ports operate asynchronously. This means that the transmitted byte must be identified by start and stop bits. The start bit indicates when the data byte is about to begin and the stop bit(s) indicates when the data byte has been transferred. The process of identifying bytes with the serial data format follows these steps:

1. When a serial port pin is idle (not transmitting data), then it is in an "on" state.
2. When data is about to be transmitted, the serial port pin switches to an "off" state due to the start bit.
3. The serial port pin switches back to an "on" state due to the stop bit(s). This indicates the end of the byte.

Data Bits

The data bits transferred through a serial port might represent device commands, sensor readings, error messages, and so on. The data can be transferred as either binary data or ASCII data.

Most serial ports use between five and eight data bits. Binary data is typically transmitted as eight bits. Text-based data is transmitted as either seven bits or eight bits. If the data is based on the ASCII character set, then a minimum of seven bits is required because there are 2^7 or 128 distinct characters. If an eighth bit is used, it must have a value of 0. If the data is based on the extended ASCII character set, then eight bits must be used because there are 2^8 or 256 distinct characters.

The Parity Bit

The parity bit provides simple error (parity) checking for the transmitted data. The types of parity checking are given below.

Table 9-2: Parity Types

Parity Type	Description
Even	The data bits plus the parity bit result in an even number of 1's.
Mark	The parity bit is always 1.
Odd	The data bits plus the parity bit result in an odd number of 1's.
Space	The parity bit is always 0.

Mark and space parity checking are seldom used because they offer minimal error detection. You might choose to not use parity checking at all.

The parity checking process follows these steps:

1. The transmitting device sets the parity bit to 0 or to 1 depending on the data bit values and the type of parity checking selected.
2. The receiving device checks if the parity bit is consistent with the transmitted data. If it is, then the data bits are accepted. If it is not, then an error is returned.

> **Note** Parity checking can detect only 1-bit errors. Multiple-bit errors can appear as valid data.

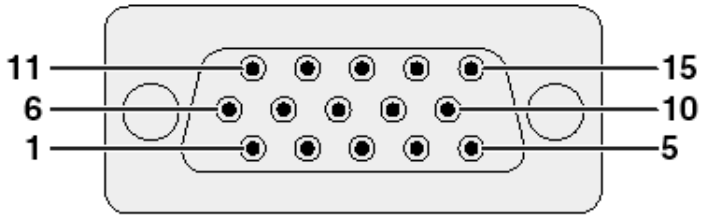
For example, suppose the data bits 01110001 are transmitted to your computer. If even parity is selected, then the parity bit is set to 0 by the transmitting device to produce an even number of 1's. If odd parity is selected, then the parity bit is set to 1 by the transmitting device to produce an odd number of 1's.

Serial Port Signals and Pin Assignments Finding Serial Port Information for Your Platform

Connectors

RGB Signal Input Port:

15-pin Mini D-sub female connector



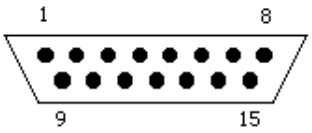
RGB Input

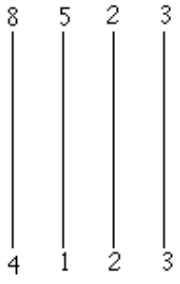
Analog

1. Video input (red)
2. Video input (green/sync on green)
3. Video input (blue)
4. Not connected
5. Composite sync
6. Earth (red)
7. Earth (green/sync on green)
8. Earth (blue)
9. Not connected
10. GND
11. GND
12. Bi-directional data
13. Horizontal sync signal
14. Vertical sync signal
15. Data clock

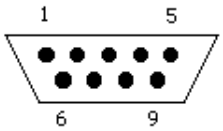
Computer - Pioneer DVD9300 player cable

DVD player DB-15 Male





Computer DB-9 female

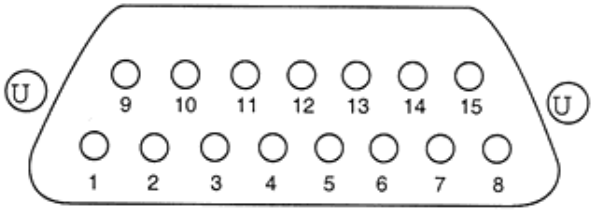


Pioneer DVD 7300 (7400)

2.1 Interface Connector

A computer may be connected to the DVD-V7400 using a 15-pin D-Sub connector (e.g., a JAE DALC-J15SAF connector with suitable plug such as the JAE DA-15PF-N) to the RS-232C serial port or to the parallel port.

The pins are identified below:

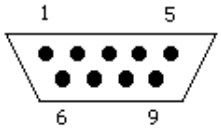


2.2 Serial Interface Pin Specification

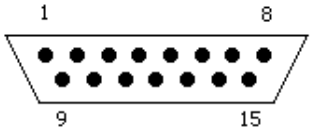
Pin #	Terminal	Input/Output	Function
1	GND	--	ground
2	TxD	Output	send data
3	RxD	Input	receive data
4	DTR	Output	enable data receiving
5	POWER	Output	external power control
6	SW1	Input	
7	SW2	Input	
8	SW3	Input	
9	SW4	Input	
10	SW5	Input	
11	SW6	Input	
12	SW7	Input	
13	SW8	Input	
14	DLTST	Input	used only for servicing the unit – do not connect
15	V +8V	Output	used only for servicing the unit – do not connect

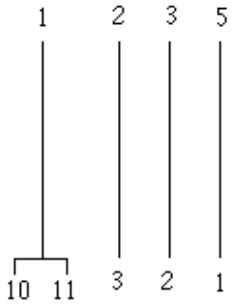
Serial controller - DVD player cable

Computer DB-9 male



DVD player DB-15 Male

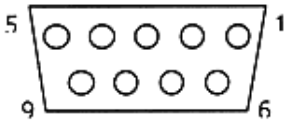




HT200/HT250

RS-232C Control Port:

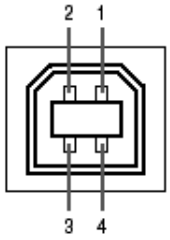
D-SUB 9-pin
(female)



Pin No	Signal	Definition
1	N/A	Not used
2	TD	Transmit data
3	RD	Receive data
4	N/A	Not used
5	GND	Ground
6	N/A	Not used
7	N/A	Not used
8	N/A	Not used
9	N/A	Not used

Parameter	Value
Transfer Rate	19200 b.p.s.
Data Length	8 bits
Parity Bit	None
Stop Bit	1 bit
Flow Control	None

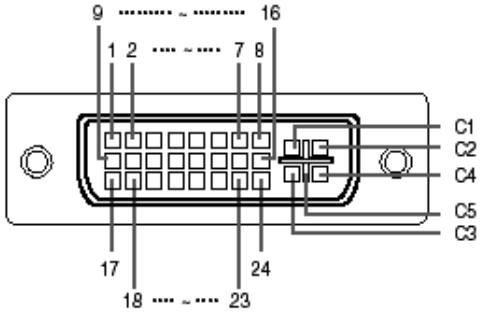
4-pin USB connector



- **USB connector: 4 pin B-type USB connector**

Pin no.	Signal	Name
1	VCC	USB power
2	USB–	USB data–
3	USB+	USB data+
4	SG	Signal Ground

DVI Digital / Analog INPUT 1 port : 29 pin connector



- **DVI Digital INPUT**

Pin No.	Signal	Pin No.	Signal
1	T.M.D.S data 2–	16	Hot plug detection
2	T.M.D.S data 2+	17	T.M.D.S data 0–
3	T.M.D.S data 2 shield	18	T.M.D.S data 0+
4	Not connected	19	T.M.D.S data 0 shield
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	T.M.D.S clock shield
8	Not connected	23	T.M.D.S clock+
9	T.M.D.S data 1–	24	T.M.D.S clock–
10	T.M.D.S data 1+	C1	Not connected
11	T.M.D.S data 1 shield	C2	Not connected
12	Not connected	C3	Not connected
13	Not connected	C4	Not connected
14	+5V power	C5	Ground
15	Ground		

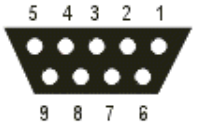
- **DVI Analog RGB Input**

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Hot plug detection
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	Not connected
8	Vertical sync	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Red
11	Not connected	C2	Analog input Green
12	Not connected	C3	Analog input Blue
13	Not connected	C4	Horizontal sync
14	+5V power	C5	Ground
15	Ground		

- **DVI Analog Component Input**

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Not connected
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	Not connected	21	Not connected
7	Not connected	22	Not connected
8	Not connected	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Pr/Cr
11	Not connected	C2	Analog input Y
12	Not connected	C3	Analog input Pb/Cb
13	Not connected	C4	Not connected
14	Not connected	C5	Ground
15	Ground		

Serial mouse



Pin	Description	
1	DCD	Data carried detect
2	RD	Receive data
3	TD	Transmit data
4	DTR	Data terminal ready
5	SG	Signal ground
6	DSR	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	Ring	

Image Dimensions in Common Usage

Collated by Paul Bourke
May 2000

Dimensions	Ratio	Comments
8x8	1	
16x16	1	Macintosh cursor size Supported by Windows icon format
32x32	1	Macintosh icon size Supported by Windows icon format
64x64	1	Supported by Windows icon format
88x31	2.8387	WWW micro banner
128x96	1 1/3	
160x120	1 1/3	NTSC 13" 1/16th CuSeeMe small image size Considered a "small" QuickTime movie
160x144	1.1111	
176x144	1.2222	
180x132	1.3636	
180x135	1 1/3	
192x144	1 1/3	PAL 1/16
234x60	3.9	WWW small banner
256x192	1 1/3	10"/12" 1/4
320x200	1.6	Called CGA, IBM PS/2
320x240	1 1/3	NTSC 13" 1/4 CuSeeMe image size Considered a "large" QuickTime movie
320x288	1.1111	
320x400	0.8	Amigo

352x288	1.2222	PAL Video CD
352x240	1.46666	NTSC Video CD
384x256	1.5	Photo CD
384x288	1 1/3	PAL 1/4
392x72	5.444444	WWW banner
400x300	1 1/3	
460x55	8.36363636	WWW banner
468x32	14.625	WWW banner
468x60	7.8	WWW banner
512x342	1.497	Original Macintosh screen
512x384	1 1/3	
544x372	1.4623	WebTV image size
640x350	1.82857	Called EGA, IBM
640x480	1 1/3	NTSC full Called PGA, IBM VGA
640x576	1.1111	
704x576	1.22222	Full image size from ASUS video capture with TNT2
720x350	2.05714286	Called MDA, IBM / VESA
720x400	1.8	Called MCGA, IBM/VESA
720x480	1.5	Format 480p. NTSC video format as used by DPS PVR video hardware. NTSC DV SDTV
720x483	1.49068323	SDTV
720x484	1.48760331	Alternative Media-100 NTSC, eg: Media-100
720x486	40/27	CCIR 601 NTSC
720x540	1 1/3	CCIR 601 NTSC Sq
720x576	1.25	CCIR 601 DV PAL and DV SECAM

729x348	2.094828	Hercules graphics
768x576 Subtract 75 from left/right and 55 top/bottom for the PAL "safe" area	1 1/3	CCIR 601 PAL full
800x600	1 1/3	Called SVGA resolution. Used in the first generation of LCD projectors (their native resolution).
832x624	1 1/3	Macintosh
856x480	1.78333333	
896x600	1.49333333	
960x720	1 1/3	Non standard digital TV format supported by some suppliers.
1024x768	1 1/3	XGA. Native resolution of many LCD projectors. Used by VisionStation and VisionStation 3.
1080x720	1.5	HDTV
1152x768	1.5	
1152x864	1 1/3	VESA
1152x870	1.3241	Macintosh
1152x900	1.28	Sun / SGI
1280x720	16/9	Format 720p. HDTV WXGA
1280x800	1.6	
1280x854	1.498829	Mac G4 15" laptop
1280x960	1 1/3	SXVGA
1280x992	1.29	Optimised on the PowerStorm 350 OpenGL card/drivers from Compaq
1280x1024	1.25	SXGA
1360x766	1.77545692	

1365x768	16:9 (nearly, 1.77734375)	NEC 61" plasma
1365x1024	1 1/3	VisionStation 3 with upgrade and VisionStation 5.
1400x1050	1.3671875	DELL Laptop
1440x900	1.6	Apple 17" G4 laptop
1520x856	1.77570093	
1600x900	16:9	
1600x1024	1.5625	
1600x1200	1 1/3	VESA UXGA
1792x1120	1.6	
1792x1344	1 1/3	
1824x1128	1.61702128	
1824x1368	1 1/3	
1856x1392	1 1/3	
1920x1080	16/9	1080i format. HDTV, known as 1K
1920x1200	1.6	
1920x1440	1 1/3	
2000x1280	1.5625	QXGA
2048x1152	16:9	
2048x1536	1 1/3	Feature film, known as 2K Also used by DOME display controllers
2048x2048	1	Tiger high resolution display
2500x1340	1.86567	
3072x2252	1.3641	Sometimes used for IMAX (?)
3600x2613	1.37772675	Sometimes used for IMAX (?)
4096x3072	1 1/3	Image size for IMAX 3D rendering, known as 4K
4096x3840	1.0666666	NCSA tiled wall (2001)

Serial Mouse Data Formats

The Microsoft Serial Mouse format is the defacto standard for serial mice. The Microsoft mouse format allows for only two buttons. Three button mice working in Microsoft mode ignore the middle button.

The data packets are sent at 1200 baud with 1 stop bit and no parity. Each packet consists of 3 bytes. It is sent to the computer every time the mouse changes state (ie. the mouse is moved or the buttons are pressed/released).

	D6	D5	D4	D3	D2	D1	D0
1st byte	1	LB	RB	Y7	Y6	X7	X6
2nd byte	0	X5	X4	X3	X2	X1	X0
3rd byte	0	Y5	Y4	Y3	Y2	Y1	Y0

```
LB is the state of the left  button, 1 = pressed, 0 = released.
RB is the state of the right button, 1 = pressed, 0 = released
X0-7 is movement of the mouse in the X direction since the
     last packet.  Positive movement is toward the right.
Y0-7 is movement of the mouse in the Y direction since the
     last packet.  Positive movement is back, toward the user.
```

The mouse driver software collects the X and Y movement bits from the different bytes in the packet. All moves are sent as two’s complement binary numbers.

Although the Microsoft format only requires 7 data bits per byte, most mice actually send 8-bit data with the most significant bit set to 1. Since the most-significant-bit (D7) is last in the serial data stream, this is the same as sending two stop bits instead of one. The Joymouse sends data packets as shown below.

	D7	D6	D5	D4	D3	D2	D1	D0
1st byte	1	1	LB	RB	Y7	Y6	X7	X6
2nd byte	1	0	X5	X4	X3	X2	X1	X0
3rd byte	1	0	Y5	Y4	Y3	Y2	Y1	Y0

[Back to Data & Documentation] [Home]

Next: Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)

RS-232 Serial Protocol

The RS-232 serial communication protocol is a standard protocol used in asynchronous serial communication. It is the primary protocol used over modem lines. It is the protocol used by the MicroStamp11 when it communicates with a host PC.

Figure 23 shows the relationship between the various components in a serial ink. These components are the *UART*, the serial channel, and the interface logic. An interface chip known as the **universal asynchronous receiver/transmitter** or **UART** is used to implement serial data transmission. The UART sits between the host computer and the *serial channel*. The serial channel is the collection of wires over which the bits are transmitted. The output from the UART is a standard TTL/CMOS logic level of 0 or 5 volts. In order to improve bandwidth, remove noise, and increase range, this TTL logical level is converted to an RS-232 logic level of -12 or $+12$ volts before being sent out on the serial channel. This conversion is done by the interface logic shown in figure 23. In your system the interface logic is implemented by the comm stamp.

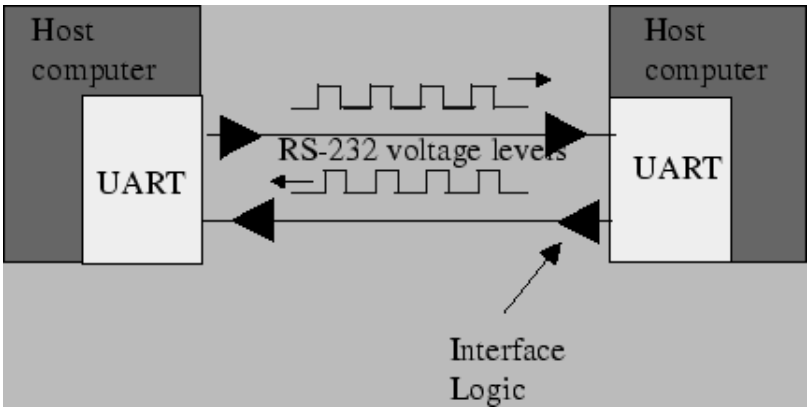


Figure 23: Asynchronous (RS-232) serial link

A **frame** is a complete and nondivisible packet of bits. A frame includes both *information* (e.g., data and characters) and *overhead* (e.g., start bit, error checking and stop bits). In asynchronous serial protocols such as RS-232, the frame consists of one start bit, seven or eight data bits, parity bits, and stop bits. A timing diagram for an RS-232 frame consisting of one start bit, 7 data bits, one parity bits and two stop bits is shown below in figure 24. Note that the exact structure of the frame must be agreed upon by both transmitter and receiver before the comm-link must be opened.

Figure 24: RS-232 Frame (1 start bit, 7 data bits, 1 parity bits,

and 2 stop bits)

Most of the bits in a frame are self-explanatory. The start bit is used to signal the beginning of a frame and the stop bit is used to signal the end of a frame. The only bit that probably needs a bit of explanation is the *parity* bit. Parity is used to detect transmission errors. For *even parity* checking, the number of 1's in the data plus the parity bit must equal an even number. For *odd parity*, this sum must be an odd number. Parity bits are used to detect errors in transmitted data. Before sending out a frame, the transmitter sets the parity bit so that the frame has either even or odd parity. The receiver and transmitter have already agreed upon which type of parity check (even or odd) is being used. When the frame is received, then the receiver checks the parity of the received frame. If the parity is wrong, then the receiver knows an error occurred in transmission and the receiver can request that the transmitter re-send the frame.

In cases where the probability of error is extremely small, then it is customary to ignore the parity bit. For communication between the MicroStamp11 and the host computer, this is usually the case and so we ignore the parity bit.

The *bit time* is the basic unit of time used in serial communication. It is the time between each bit. The transmitter outputs a bit, waits one bit time and then outputs the next bit. The start bit is used to synchronize the transmitter and receiver. After the receiver senses the true-false transition in the start bit, it waits one half bit time and then starts reading the serial line once every bit time after that. The *baud rate* is the total number of bits (information, overhead, and idle) per time that is transmitted over the serial link. So we can compute the baud rate as the reciprocal of the bit time.

Next: Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)
Bill Goodwine 2002-09-29


Next: LCD display for the **Up:** Serial Communication **Previous:** RS-232 Serial Protocol

Motorola 68HC11 SCI Interface

The Motorola 68HC11 supports one SCI. We'll discuss both transmitting and receiving ends of the SCI. The programmer controls the operation of the SCI interface through a set of hardware registers that are memory mapped into the processor's address space. There are 5 control registers shown below in figure 25. This figure also shows the logical names for individual bits in the registers. The `BAUD` register is used to set the serial link's baud rate. There are two control registers `SCCR1` and `SCCR2` that specify how the SCI should work. There is a status register, `SCSR` that the programmer can use to check whether the transmission/reception of a frame has been completed. Finally there is the data register `SCDR` that holds the transmitted or received information bits.

BAUD

TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0

SCCR1

R8	T8	0	M	WAK	0	0	0

SCCR2

TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

SCSR

TDRE	TC	RDRF	IDLE	OR	NF	FE	0

SCDR

R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

Figure 25: 68HC11 SCI Registers

From the number of control bits in the `SCCR1` and `SCCR2` registers you can see that the programmer has quite a bit of control over the SCI interface. Most of the situations we'll be using, however, have standard set-ups so we won't need to discuss the control register bits in detail. Instead, we'll provide standard functions that encapsulate the user's interface to SCI devices such as a personal computers and simply discuss how these functions work. The proper setup of the SCI subsystem is usually done in your program's `init()` initialization routine.

To understand how the SCI subsystem works, let's examine figure 26. Figure 26 shows how the programmer interacts with the SCI transmit and receive buffers. In order to transmit, the programmer first loads the 8-bit data register `SCDR` with the data to be sent. The SCI module automatically fills in the start bit, stop bit, and the extra `T8` bit from control register `SCCR1`. The `T8` bit can be used as a parity bit.

Once the SCDR register is loaded, the subsystem loads this data into the SCI module's transmit buffer. The SCI transmit buffer then holds a single frame and this frame is then clocked out of the transmit register one bit at a time at the rate specified in the BAUD register. Once all of the bits have been clocked out of the transmit buffer, the SCI module sets the TDRE (transmit data register empty) bit in the SCI's status register SCSR. This single bit can then be used to check whether or not the frame has been successfully transmitted.

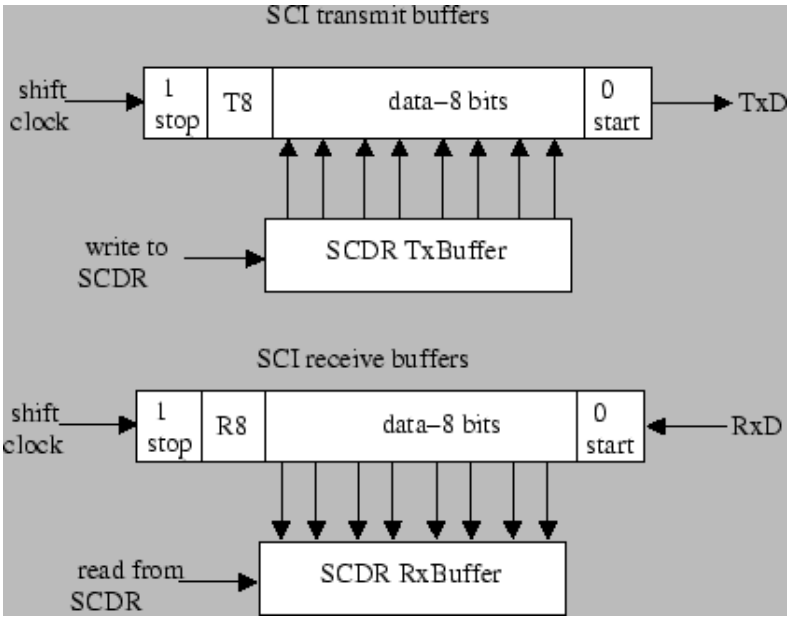


Figure 26: SCI transmit and receive buffers

A similar set of steps can be used to check and see if the receive data register has been filled. Once initialized, the receive data component of the SCI subsystem will wait for the true-to-false transition on the input line signalling a start bit. After the start bit has been detected, the receive subsystem will shift in 10 or 11 bits into the receive data register. The start and stop bits are removed and 8 bits of data are loaded into the SCI data register SCDR. The ninth parity bit R8 is put in the SCCR1 control register. When the receive data register is full, then the SCI subsystem sets the RDRF (receive data register full) flag in the status register SCSR. The programmer can then check this status flag to see if a full frame has been received.

The following code segment from an init() function can be used to initialize the SCI module to transmit and receive at 38 kbaud. This setup was used in our earlier kernel.c functions to send characters back and forth between the MicroStamp11 and the PC.

```
void init(void){
  asm(" sei");
    CONFIG = 0x04;
    BAUD=BAUD38K;
    SCCR1 = 0x00;
    SCCR2 = 0x0C;
  asm(" cli");
}
```

The instructions in this function do the following. The first instruction CONFIG = 0x04 turns off the

micro-controller's watchdog timer. The second instruction `BAUD=BAUD38K` sets the SCI subsystem's baud rate to 38 kilo-baud. The variable `BAUD38K` is a logical name whose actual value will be found in `hc11.h`. The next two lines set up the SCI subsystem's parameters. By zeroing `SCCR1`, we are ignoring the parity bit and creating a 10-bit frame. By setting `SCCR2=0x0C`, we've disabled all transmit and receive interrupts and we've enabled the transmit and receive modules in the SCI subsystem.

Note that the SCI module has hardware interrupts associated with the hardware events

- Transmission Complete (TC): which is set when the transmit shift register is empty.
- Transmit Data Register Empty (TDRE): which is set when the transmit data register is empty.
- Receive Data Register Full (RDRF): which is set when the receive data register is full.
- Idle line (ILIE): which is when the receive module detects an idle line.

These hardware interrupts can be used to do parity-bit processing on a transmitted or received frame. In our particular examples, however, we assume no parity checking so these interrupts have been disabled.

As specific examples of how the SCI interface can be used, we consider the two function `InChar()` and `OutChar()`. These functions are used to receive and transmit, respectively, a single byte (frame) of data.

```
void OutChar(char data){
  while((SCSR & TDRE) == 0);
  SCDR=data;
}

void InChar(void){
  while((SCSR & RDRF) == 0);
  return(SCDR);
}
```

The first function, `OutChar()` transmits a single byte. The function simply waits in a loop until the `TDRE` bit (transmit data register empty) is set. Once this is done, we know that any previously loaded bytes have been successfully shifted out of the transmit data register. The function then reloads the SCI's data register `SCDR` with the new data. The other function `InChar()` receives a single byte. The function waits in a while loop until the `RDRF` bit (receive data register full) is set, thereby indicating that 8-bits of have been shifted into the receive data register. Once this is done, the function returns a pointer to the `SCDR` data register.

Next: LCD display for the **Up:** Serial Communication **Previous:** RS-232 Serial Protocol
Bill Goodwine 2002-09-29

Serial Data Format

The serial data that we are interested in sending to and from the terminal is byte-wide ASCII data. ASCII is a standard code for sending alphanumeric data and is actually only 7-bits wide. The 8th bit is often used to indicate the parity of the 7-bit data word and used for error detection. For our circuit, the high-order bit will always be 0, but you should always send it anyway. So, each packet that is sent will consist of 8 bits, 7-bits of ASCII and one 0 in the high-order bit. A table of the ASCII code is shown in Figure 1.

$B_3B_2B_1B_0$	$B_6B_5B_4$ 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	—
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Figure 1: ASCII Character Codes

Once we agree to send ASCII, we should also agree on how that data should be send as a serial data stream. The protocol we will use defines that the bits in the byte are passed least-significant bit first in a serial stream. So, send the bits one at a time, starting with the least-significant. How long each bit is asserted in this serial stream depends on how fast your baud rate is. At 9600 baud, for example, each bit will be asserted for 1/9600 of a second, or about 104 μ sec. Now the problem is how to decide when to look at the data wire in order to see the bit. The DCE and DTE will not be synchronized to a common clock, so in order to decide when to look at the data line to see new data being passed, they must synchronize with each new byte that is being passed. This is why the UART is ‘‘asynchronous’’ in operation. The data is being passed at a known frequency, but the starting time of each new byte is unknown. So, the receiving circuit must resynchronize at the start of each new byte.

In practice this is quite easy. You only need some sort of protocol that tells you when to expect new data. For our system (and most asynchronous serial protocols in general) the data line must be held in a 1 state (+5v in our case) until a byte is ready to be passed. When a byte is to be sent, the data line drops to 0 (gnd in our case) for one bit time to signal that a byte will follow. This is the *start bit* and its purpose is to wake up the receiver and alert it to the byte that is about to be sent. The 8 data bits then follow, least significant bit first, each asserted for one bit time (which depends on the baud rate). Finally, one or two *stop bits* are sent to indicate the end of the byte. Stop bits are 1-bits that are asserted for one bit time each. In our system the receiver will assume that only one stop bit is sent, and the sender will send two stop bits. This is the most general and safest solution. For example, if the receiver expects a single stop bit and two are sent, nothing bad happens except that some extra time elapses between that byte and the next due to the extra stop bit. On the other hand, if the sender sends only a single stop bit but the external receiver expects two, a mistake might be made. So, it's safer to expect only a single stop bit, but always send two (note that in practice, most terminals and other communication devices have settings to control how many stop bits are sent or expected.) A picture of this data protocol is shown in Figure 2.

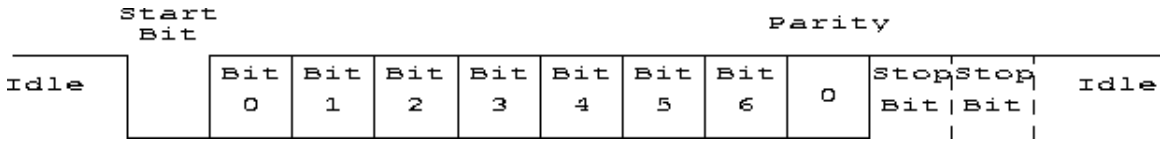


Figure 2: Data Byte Transmission Format

Erik Brunvand
Tue Apr 11 15:50:32 MDT 2000

PostScript Tutorial

Written by Paul Bourke
Original November 1990. Last updated December 1998

Introduction

Postscript is a programming language that was designed to specify the layout of the printed page. Postscript printers and postscript display software use an interpreter to convert the page description into the displayed graphics.

The following information is designed as a first tutorial to the postscript language. It will concentrate on how to use postscript to generate graphics rather than explore it as a programming language. By the end you should feel confident about writing simple postscript programs for drawing graphics and text. Further information and a complete specification of the language can be obtained from **The Postscript Language Reference Manual** from Adobe Systems Inc, published by Addison-Wesley, Reading, Massachuchusetts, 1985.

Why learn postscript, after all, many programs can generate it for you and postscript print drivers can print to a file? Some reasons might be:

- Having direct postscript output can often result in much more efficient postscript, postscipt that prints faster than the more generic output from printer drivers.
- There are many cases where generating postscript directly can result in much better quality. For example when drawing many types of fractals where high resolution is necessary, being able to draw at the native high resolution of a postscript printer is desirable.
- It isn't uncommon for commercial packages to make errors with their postscript output. Being able to look at the postscript and make some sense of what is going on can sometimes give insight on how to fix the problem.

The Basics

Postscript files are (generally) plain text files and as such they can easily be generated by hand or as the output of user written programs. As with most programming languages, postscript files (programs) are intended to be, at least partially, human-readable. As such, they are generally free format, that is, the text can be split across lines and indented to hilight the logical structure.
Comments can be inserted anywhere within a postscript file with the percent (%) symbol, the comment applies from the % until the end of the line.
While not part of the postscript specification the first line of a postscript file often starts as %!. This is so that spoolers and other printing software detect that the file is to interpreted as postscript instead of a plain text file. The inline example below will not include this but the postscript files linked from this page will include it since they are design for direct printing.
The first postscript command to learn is **showpage**, it forces the printer to print a page with whatever is currently drawn on it. The examples given below print on single pages and therefore there is a showpage at the end of the file in each example, see the comments later regarding EPS.

A Path

A path is a collection of, possibly disconnected, lines and areas describing the image. A path is itself not drawn, after it is specified it can be stroked (lines) or filled (areas) making the appropriate marks on the page. There is a special type of path called the clipping path, this is a path within which future drawing is constrained. By default the clipping path is a rectangle that matches the border of the paper, it will not be changed during this tutorial.

The Stack

Postscript uses a stack, otherwise known as a LIFO (Last In First Out) stack to store programs and data. A postscript interpreter places the postscript program on the stack and executes it, instructions that require data will read that data from the stack. For example, there is an operator in postscript for multiplying two numbers, **mul**. it requires two arguments, namely the two numbers that are to be multiplied together. In postscript this might be specified as

```
10 20 mul
```

The interpreter would place 10 and then 20 onto the stack. The operator **mul** would remove 20 and then 10 from the stack, multiply them together and leave the result, 200, on the stack.

Coordinate system

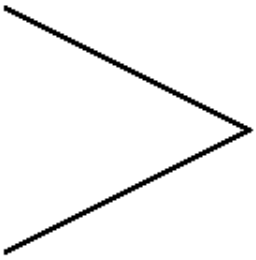
Postscript uses a coordinate system that is device independent, that is, it doesn't rely on the resolution, paper size, etc of the final output device. The initial coordinate system has the x axis to the right and y axis upwards, the origin is located at the bottom left hand corner of the page. The units are of "points" which are 1/72 of an inch long. In other words, if we draw a line from postscript coordinate (72,72) to (144,72) we will have a line starting one inch in from the left and right of the page, the line will be horizontal and be one inch long.

The coordinate system can be changed, that is, scaled, rotated, and translated. This is often done to form a more convenient system for the particular drawing being created.

Basic Drawing Commands

Time to draw something. The following consists of a number of operators and data, some operators like **newpath** don't need arguments, others like **lineto** take two arguments from the stack. All the examples in this text are shown as postscript on the left with the resulting image on the right. The text on the left also acts as a link to a printable form of the postscript file.

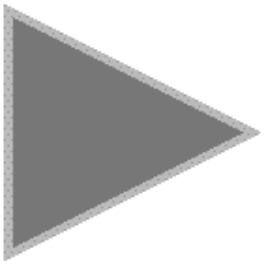
```
newpath
100 200 moveto
200 250 lineto
100 300 lineto
2 setlinewidth
stroke
```



There are also a relative moveto and lineto commands, namely, **rmoveto** and **rlineto**.

In this next example a filled object will be drawn in a particular shade, both for the outline and the interior. Shades range from 0 (black) to 1 (white). Note the **closepath** that joins the first vertex of the path with the last.

```
newpath
100 200 moveto
200 250 lineto
100 300 lineto
closepath
gsave
0.5 setgray
fill
grestore
4 setlinewidth
0.75 setgray
stroke
```



The drawing commands such as **stroke** and **fill** destroy the current path, the way around this is to use **gsave** that saves the current path so that it can be reinstated with **grestore**.

Text

Text is perhaps the most sophisticated and powerful aspect of postscript, as such only a fraction of its capabilities will be discussed here. One of the nice things is that the way characters are placed on the page is no different to any other graphic. The interpreter creates a path for the character and it is then either stroked or filled as per usual.

```
/Times-Roman findfont
12 scalefont
setfont
newpath
100 200 moveto
(Example 3) show
```

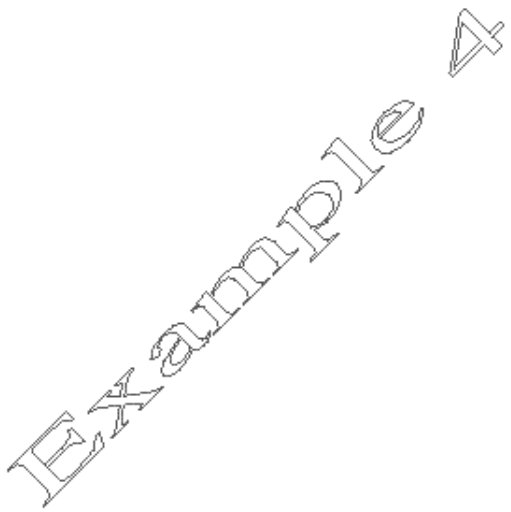
Example 3

As might be expected the position (100,200) above specifies the position of the bottom left corner of the text string. The first three lines in the above example are housekeeping that needs to be done the first time a font is used. By default the font size is 1 point, scalefont then sets the font size in units of points

(1/72 inch). The brackets around the words "Example 3" indicate that it is a string.

A slightly modified version of the above uses **charpath** to treat the characters in the string as a path which can be stroked or filled.

```
/Times-Roman findfont
32 scalefont
setfont
100 200 translate
45 rotate
2 1 scale
newpath
0 0 moveto
(Example 4) true charpath
0.5 setlinewidth
0.4 setgray
stroke
```

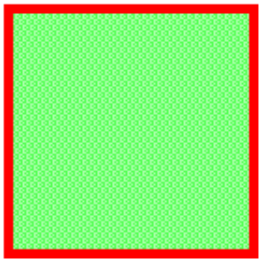


You should make sure you understand the order of the operators above and the resulting orientation and scale of the text, procedurally it draws the text, **scale** the y axis by a factor of 2, **rotate** counter clockwise about the origin, finally **translate** the coordinate system to (100,200).

Colour

For those with colour LaserWriters the main instruction of interest that replaces the **setgray** is previous examples is **setrgbcolor**. It requires 3 arguments, the red-green-blue components of the colour each varying from 0 to 1.

```
newpath
100 100 moveto
0 100 rlineto
100 0 rlineto
0 -100 rlineto
-100 0 rlineto
closepath
gsave
0.5 1 0.5 setrgbcolor
fill
grestore
1 0 0 setrgbcolor
4 setlinewidth
stroke
```

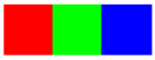


Programming

As mentioned in the introduction postscript is a programming language. The extend of this language will

not be covered here except to show some examples of procedures that can be useful to simplify postscript generation and make postscript files smaller.
Lets assume one needed to draw lots of squares with no border but filled with a particular colour. One could create the path repeatedly for each one, alternatively one could define something like the following.

```
/csquare {
   newpath
   0 0 moveto
   0 1 rlineto
   1 0 rlineto
   0 -1 rlineto
   closepath
   setrgbcolor
   fill
} def

20 20 scale

5 5 translate
1 0 0 csquare

1 0 translate
0 1 0 csquare

1 0 translate
0 0 1 csquare
```



This procedure draws three coloured squares next to each other, each 20/72 inches square, note the **scale** of 20 on the coordinate system. The procedure draws a unit square and it expects the RGB colour to be on the stack. This could be used as a method (albeit inefficient) of drawing a bitmap image.

Even if one is simply drawing lots of lines on the page, in order to reduce the file size it is common to define a procedure as shown below. It just defines a single character "l" to draw a line segment, one can then use commands like 100 200 200 200 l" to draw a line segment from (100,200) to (200,200).

```
/l { newpath moveto lineto stroke } def
```

Some other useful Commands

The following are some other commonly used commands along with a brief description, again you should consult a reference manual for the entire set of commands.

arc	Draws an arc (including a circle). The arguments are xcenter, ycenter, radius, start angle, stop angle. The arc is drawn counterclockwise, the angles are in units of degrees.
centerpoint	This is an example of an instruction that takes no arguments but leaves numbers on the stack, namely the coordinates of the current point.
setdash	This sets the dash attribute of a line in terms of a mark-space array. Just as strings are denoted by round braces (), arrays are denoted by square braces []. For example the following command "[3 3] 0 setdash" would make any following lines have a 3 unit dash followed by a 3 unit space. The argument after the dash array is the offset for the start of

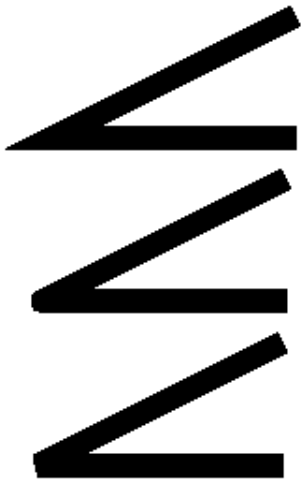
the first dash.

setlinecap This specifies what the ends of a stroked line look like. It takes one argument which may be 0 (butt caps), 1 (round caps), or 2 (extended butt caps). The radius of round caps and the extension of the butt caps is determined by the line thickness.

```
/LINE {
        newpath
        0 0 moveto
        100 0 lineto
        stroke
} def

100 200 translate
10 setlinewidth 0 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE

0 20 translate
10 setlinewidth 1 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE

0 20 translate
10 setlinewidth 2 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE
```

setlinejoin This determines the appearance of joining lines. It takes one argument which may be 0 (miter join), 1 (round join), or 2 (bevel join).

```
/ANGLE {
        newpath
        100 0 moveto
        0 0 lineto
        100 50 lineto
        stroke
} def

10 setlinewidth
0 setlinejoin
100 200 translate
ANGLE

1 setlinejoin
0 70 translate
ANGLE

2 setlinejoin
0 70 translate
ANGLE
```



curveto This draws a bezier curve through the three points given as arguments. The curve starts at the first point, end at the last point, and the tangents are given by the line between the first-second and second-third pair.

save and **restore** Instead of having to "undo" changes to the graphics state it is possible using **save** to push the entire graphics state onto the stack and then reinstate it later with a **restore**.

Drawing "small" Images

Printing images using postscript is somewhat more involved than the graphics discussed so far. Examples of two image types will be presented, grey scale images (which could also be used for black and white images) and 24 bit RGB images.

8 Bit Grey Scale

The simplest method that is applicable to small images is the postscript command **image**. This take 5 arguments: the width and height of the image, the bits per pixel, a transformation matrix (6 numbers) and finally a procedure for acquiring the image data. In the following example, the grey scale (8 bit) image is 24 pixels wide by 34 pixels tall. Since in postscript an image is defined from the lower left corner, and the image is defined from its top left corner, the transformation matrix used here does the appropriate vertical flipping. The procedure used here is esentially a null procedure, it is a hexadecimal string as indicated by the angle brackets <>. Postscript renders images as one point square hence the scaling by 24 horizontally and 34 vertically. There are many details left out of this discussion, the reader needs to consult a postscript reference manual for more information.

```
100 200 translate
26 34 scale
26 34 8 [26 0 0 -34 0 34]
{<
ffffffffffffffffffffffffffffffffffffffffffffffffff
ff000000000000000000000000000000000000ffffffffffff
ff00efefefefefefefefefefefefefefef0000ffffffffffff
ff00efefefefefefefefefefefefefefef00ce00ffffffffff
ff00efefefefefefefefefefefefefefef00cece00ffffffff
ff00efefefefefefefefefefefefefefef00cecece00ffffff
ff00efefefefefefefefefefefefefefef00cecececе00ffff
ff00efefefefefefefefefefefefefefef000000000000000ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efef000000ef000000ef000000ef0000ef0000efefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efef000000ef00000000ef00000000ef000000efefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efef0000ef00000000000000ef000000ef0000efefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00efefefefefefefefefefefefefefefefefefefefef00ff
ff00000000000000000000000000000000000000000000000ff
fffffffffffffffffffffffffffffffffffffffffffffffffff
```

```
>}
image
```

24 Bit RGB Colour

RGB images with 8 bits per pixel can be represented in postscript using the command **colorimage** which is very similar to the **image** command. In the following example the image is 32 pixels wide by 38 pixels tall.

```
100 200 translate
32 38 scale
32 38 8 [32 0 0 -38 0 38]
{<
1dfb0023fb002afb0031fb0037fb003ffb00
66fb006cfb0073fb0079fb0081fb0086fb00
adfb00b5fb00bbfb00c3fb00c8fb00cffb00
23f5002af50031f50037f5003ff50044f500

           ...cut...

3807003f08004508004c0800520800590800
8108008608008d07009308009a0700a20800
c90800d00800d60800dd0800e40700ea0700
>}
false 3 colorimage
```

What is EPS?

EPS (Encapsulated PostScript) is normal postscript with a few restrictions and a few comments in a specified format that provides more information about the postscript that follows. It was design to make it easier for applications to include postscript generated elsewhere within their own pages. The full specification can be obtained from Adobe but in order to make a postscript file DSC (Adobe's Document Structuring Convention) compliant the following must be true:

- There shouldn't be a **showpage**, since EPS is designed to be included inside other documents a **showpage** would obviously ruin the intended effect. In reality most programs that import EPS redefine **showpage** so that if it does exist it doesn't cause problems, a common definition is "/showpage { } def"
- The file should consist of just one page.
- The first line of the file should be "%!PS-Adobe EPSF-3.0"
- There must be a correctly formed BoundingBox comment, this looks like
  %%BoundingBox: xmin ymin xmax ymax
  and tells application that plans to include the postscript how large the image is.
- The file should not use any operators that change the global drawing state. In particular the following command may not be used:

```
banddevice      exitserver    initmatrix     setshared
clear           framedevice   quit           startjob
cleardictstack  grestoreall   renderbands    copypage
initclip        setglobal     initgraphics   setpagedevice
erasepage       nulldevice    sethalftone    setscreen
```

```
setgstate       setmatrix     settransfer     undefinefont
```

- The stack must be left EXACTLY in the same state at the end of the EPS file as it was at the start of the EPS file.
- The lines in EPS files cannot exceed 255 characters in length.

Perhaps most importantly, since usually an application that supports postscript file insertion doesn't have the full postscript interpreter, an EPS file generally has a preview image associated with it. The application dealing with the EPS file can display the preview in the user interface giving a better idea what will print. It should be noted that EPS previews are one of the more machine/OS dependent aspects of EPS.

Frequently Used Comments

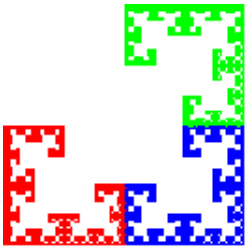
Comments can of course be be added anywhere and they will be ignored by the interpreter. There are some standard comments the most common of which are be listed below. The text within the square brackets should be replaced with the appropriate text for the file in which they appear (without the []).

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: [generally the program that generated the postscript]
%%Title: [descriptive name or just the file name]
%%CreationDate: [date the file was created]
%%DocumentData: Clean7Bit
%%Origin: [eg: 0 0]
%%BoundingBox: xmin ymin xmax ymax
%%LanguageLevel: 2 [could be 1 2 or 3]
%%Pages: 1
%%Page: 1 1
%%EOF
```

Drawing "large" images

Due to line length and other restrictions, turning 'large" bitmaps into postscript requires a modification to the methods discussed earlier. The following will describe the most general case of representing a 24 bit RGB colour image as an EPS file. While inefficient this can also be used for greyscale and even black and white images. In the following code "width" and "height" should be replaced with the numbers appropriate to the image.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: someone or something
%%BoundingBox: 0 0 width height
%%LanguageLevel: 2
%%Pages: 1
%%DocumentData: Clean7Bit
width height scale
width height 8 [width 0 0 -height 0 height
{currentfile 3 width mul string readhexstring pop} bind
false 3 colorimage

...hexadecimal information cut...

%%EOF
```



The modifications for greyscale images are quite simple, change the line

```
{currentfile 3 width mul string readhexstring pop} bind
```

to

```
{currentfile width string readhexstring pop} bind
```

and of course only write one hexadecimal number (representing the grey level of the pixel) for each pixel of the image. This technique should work for images of any size.

Paper sizes

Paper Size	Dimension (in points)
Comm #10 Envelope	297 x 684
C5 Envelope	461 x 648
DL Envelope	312 x 624
Folio	595 x 935
Executive	522 x 756
Letter	612 x 792
Legal	612 x 1008
Ledger	1224 x 792
Tabloid	792 x 1224
A0	2384 x 3370
A1	1684 x 2384
A2	1191 x 1684
A3	842 x 1191
A4	595 x 842
A5	420 x 595
A6	297 x 420
A7	210 x 297
A8	148 x 210
A9	105 x 148
B0	2920 x 4127
B1	2064 x 2920
B2	1460 x 2064
B3	1032 x 1460
B4	729 x 1032
B5	516 x 729
B6	363 x 516
B7	258 x 363
B8	181 x 258
B9	127 x 181
B10	91 x 127

QD3D

Apple QuickTime 3D Meta file format

Converted by Paul Bourke

Metafile Header

Full name: 3DMF, 3DMetafile
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: 3DMF
Ascii type: 3DMetafile
Binary size: 16
Parent Objects: none
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The metafile header is the first object to appear in any metafile.
Metafile versions of 1.x are expected to maintain some degree of compatibility.
Flags indicate to a general degree of how the file is structured or should be read.

A database file indicates that the metafile is a library, and all objects that are shared appear in the table of contents.
A stream file indicates that no references exist in the metafile, so that a parsing program may discard encountered data when it is through with it.

If the toc location (Table of Contents location) is NULL, the entire file must be parsed to find a Table Of Contents.

Data structure:
Uns16 majorVersion
Uns16 minorVersion
MetafileFlags flags
FilePointer tocLocation

As of this release:

majorVersion = 0
minorVersion = 8

The final release of the metafile will be:

majorVersion = 1
minorVersion = 0

MetafileFlags bitfield is:

Binary Text
0x00000000 Normal
0x00000001 Stream
0x00000002 Database

Text samples:

```
3DMetafile (
  1 0 # version
  Normal
  toc>
)
...
toc: TableOfContents (
  ...
)
```

Begin Group

Full name: 3DMF, BeginGroup
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: bgng
Ascii type: BeginGroup
Binary size: sizes of contained objects + (8 * number of child objects)
Parent Objects: special
Format: Data Format
Subobjects:
Inherited: No
Referencable: No

Description:
The begin group object is used similarly to the container object, except it is used as the starting delimiter for a group. This allows a naive parser to traverse a metafile without special casing the many types of groups that appear in the metafile spec. It also allows for a single mechanism that is used to declare a group.

Please note that all objects of type group MUST be contained in a begin group, to allow them to be identified as starting a group.

Data structure:

Text samples:

```
BeginGroup ( DisplayGroup ( ) )
  Triangle ( 0 0 1 0 0 0 0 1 0 )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

BeginGroup (
  OrderedDisplayGroup ( )
  DisplayGroupState ( DoNotDraw )
)
  Triangle ( 0 0 1 0 0 0 0 1 0 )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

BeginGroup ( InfoGroup ( ) )
  CString ( Copyright (c) 1995 )
EndGroup ( )
```

Container

Full name: 3DMF, Container
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: cntr
Ascii type: Container
Binary size: sizes of contained objects + (8 * number of child objects)
Parent Objects: special
Format: Data Format
Subobjects: special
Inherited: No
Referencable: No

Description:
Used to bind objects together to form a single object.
Container objects always contain other objects.
The first object in the container is called the root object, and sets the scope of the remaining objects in the container, called subobjects.
In general, the root object instantiates the object with its default values, and subobjects append information to the original root object.
There is one exception to these encapsulation rules, which is group objects. Although group objects contain a list of other objects, they are delimited with another 3DMF object, the end group object.

Data structure:

Text samples:

```
Container (
  Box ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
  )
)
```

End Group

Full name: 3DMF, EndGroup
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: endg
Ascii type: EndGroup
Binary size: 0
Parent Objects: none
Format: No Data
Subobjects: none
Inherited: No
Referencable: No

Description:
This object is used as a delimiter for all group objects.

Data structure:
Groups should be arranged into non-overlapping pairs of BeginGroup (group type/data) and an EndGroup object.

All groups must be arranged into DAGs. (no cycles are permitted)

Text samples:

```
# Empty group
BeginGroup ( OrderedDisplayGroup ( ) )
EndGroup ( )

# Group containing 1 object
BeginGroup ( DisplayGroup ( ) )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

# Inline group referenced elsewhere

REDColor:
BeginGroup (
  DisplayGroup ( )
```

```
  DisplayGroupState ( IsInline )
)
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 )
  )
EndGroup ( )

BeginGroup ( DisplayGroup ( ) )
  Reference ( 1 ) # REDColor
  Cone () # Cone is RED
EndGroup ()
toc: TableOfContents (
  nextTOC> -1 2 0 12
  1
  1 REDColor>
)
```

Junk

Full name: 3DMF, Junk
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: junk
Ascii type: Junk
Binary size: any size
Parent Objects: special
Format: Data Format
Subobjects: any
Inherited: No
Referencable: No

Description:
The junk object contains garbage, and serves as a placeholder for deleted information in the metafile. Junk objects should always be skipped and never parsed.

Data structure:

Text samples:

```
Container (
  Box ( )
  Junk (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
  )
)

is equivalent to:

Box ()
```

Reference

Full name: 3DMF, Reference
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: rfrn
Ascii type: Reference
Binary size: 4
Parent Objects: may be substituted for any Shared object
Format: Data Format
Subobjects: 1 Storage object (optional)
Inherited: No
Referencable: No

Description:
The reference object is used to instantiate an object multiple times in a metafile.

It may be substituted anywhere in the metafile for another Shared object. Only shared objects may be referenced.

References are resolved in the Table Of Contents. If a Storage object is specified as a subobject, it is assumed that the reference is external to the current metafile, and should be resolved in that external storages table of contents.

Data structure:
Uns32 refID

if refID = 0, must contain subobjects

if refID > 0, a TOC must exist in current metafile that contains refIDs resolution

This refID is resolved in the current metafile unless a Storage subobject is found in the Reference

Text samples:

```
Reference ( 23 ) # internal reference
...
toc: TableOfContents (
  nextTOC> 35 -1 0 12
  ...
  20 CarFrame>
  21 Axle>
  23 WheelOfCar>
  ...
)

Container ( # external reference
  Reference ( 23 )
```

```
  UnixPath ( parts/car.eb )
)
```

Table Of Contents

Full name: 3DMF, TableOfContents
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: toc
Ascii type: TableOfContents
Binary size: 28 + (tocEntrySize * nEntries)
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The table of contents provides a means of resolving references within a file. The nextTOC file pointer points to the next table of contents in the file, or is NULL if no other table of contents exists.

The reference seed indicates the next available reference id available for reference objects. It is an unsigned positive number that is incremented with each addtional reference in a file. It is always one more than the maximum reference seed in a file.

The type seed indicates the next available type ID available for type objects. It is a negative number that is decremented with each additional type in a file. It is always one less than the minimum type seed in a file.

The tocEntryType and tocEntrySize are a set of paired values which indicate the size and type of information stored in a tocEntry.

The tocEntries are sorted by reference ID, in increasing order, to allow fast searching of the table of contents.

Data structure:
FilePointer nextTOC
Uns32 refSeed
Int32 typeSeed
Uns32 tocEntryType
Uns32 tocEntrySize
Uns32 nEntries
TOCEntry tocEntries

refSeed > 0
typeSeed < 0

tocEntryType = 0 or 1
tocEntrySize = 12 or 16, based upon tocEntryType
the TOCEntry structure is:
- tocEntryType 0, tocEntrySize 12 is:

Uns32 refID
FilePointer objLocation

- tocEntryType 1, tocEntrySize 16 is:

Uns32 refID
FilePointer objLocation
ObjectType objType

Text samples:

```
3DMetafile (
  1 0
  Normal
  toc>
)
box23:
Mesh (
  45 # nVertices
  ...
)
Reference ( 1 )
Arrows:
BeginGroup ( DisplayGroup ( ) )
  Cone ( )
  Scale ( 0.2 0.1 0.2)
  Cylinder ( )
EndGroup ( )
Reference ( 2 )
Reference ( 4 )
...
Type ( -1 Joes Garage:RepairHistory )
...

-1 ( Jim Fixed lug nut 0.23 0.2 1.2 )

toc:
TableOfContents (
  nextTOC>
  5  # refSeed
  -2 # typeSeed
  0 12 # tocEntry Type/Size
  3 # nEntries
  1 box23>
  2 Arrows>
  4 Geom34>
)
```

Type

Full name: 3DMF, Type
Drawable: No
Parent Class Heirarchy: 3DMF
Binary type: type
Ascii type: Type
Binary size: 4 + sizeof(String)
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
A type definition is used to declare a custom data type. A type definition may appear anywhere in a file, however, the custom type must be encountered before the custom object of that type is encountered..

All custom types in the metafile are negative numbers, and the typeID field begins at -1 and is decremented for each additional type. Only 2147483648 (or 2^31) custom types are permitted in a single metafile.

The owner string is an ISO 9070 registered owner string. Owner strings are unique globally for each type of custom data.

In the binary and text metafile, the typeID is used as the object type later in the file.

Data structure:
Int32 typeID
String owner

typeID < 0
owner string

Text samples:

```
Type (
  -1
  Joes Garage:BoltData
)

...

-1 (
   -2.3 34 # Stress (kPA/area)
)
```

Face Attribute Set List

Full name: Data, AttributeSetList, FaceAttributeSetList
Drawable: No
Parent Class Heirarchy: Data, AttributeSetList
Binary type: fasl
Ascii type: FaceAttributeSetList
Binary size: 12 + nIndices * sizeof(Uns) + padding
Parent Objects: ALWAYS: Box, GeneralPolygon, Mesh, TriGrid
Format: Data Format
Subobjects: many AttributeSet (order-dependent)
Inherited: No
Referencable: No

Description:
The face attribute set list specifies a list of attributes to be attached to a set of faces determined by the parents topology.

nObjects indicates the total number of objects being mapped to.

packing indicates how AttributeSet objects are mapped to indices. Include packing lists the face indices, in sequential order, of those faces to be assigned face attribute sets. Exclude packing lists the face indices, in sequential order, of those faces to NOT be assigned face attribute sets.

So, for example, supposing nObjects was 5, Include packing with a list of 3 indices after it means that there are 3 subobjects, each assigned to the indices in their order. Exclude packing with a list of 3 indices after it means there are 2 attribute sets subobjects, assigned to the indices NOT in the exclude list, in order.

The face attribute set list is padded to the nearest long word.

The values in indices always appear in increasing order.

If a packing value other than Include or Exclude is found, this object and its subobjects should be ignored.

Data structure:
Uns32 nObjects
PackingEnum packing
Uns32 nIndices
Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum is:

Binary Text

0x00000000 Include
0x00000001 Exclude

0 indices < nObjects

Text samples:

```
Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Include 2
      0 1
    )
    Container ( # assigned to 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # assigned to 1
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)

Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Exclude 2
      2 4
    )
    Container ( # assigned to 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # assigned to 1
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
    Container ( # assigned to 3
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
    Container ( # assigned to 5
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Geometry Attribute Set List

Full name: Data, AttributeSetList, GeometryAttributeSetList
Drawable: No

Parent Class Heirarchy: Data, AttributeSetList
Binary type: gasl
Ascii type: GeometryAttributeSetList
Binary size: 12 + nIndices * 4 + padding
Parent Objects: ALWAYS: PolyLine
Format: Data Format
Subobjects: many AttributeSet (order-dependent)
Inherited: No
Referencable: No

Description:
The geometry attribute set list specifies a list of attributes to be attached to a set of geometric entities determined by the parents topology.

Currently, only the PolyLine primitive uses this object. Each attribute set is mapped to a line segment in the PolyLine.

Packing for this object is identical to the other attribute set lists.

Data structure:
Uns32 nObjects
PackingEnum packing
Uns32 nIndices
Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum described in FaceAttributeSetList

Text samples:

```
Container (
  PolyLine (
   3
   10 2 3
   0 0 0
   2 8.5 3
  )
  Container (
    GeometryAttributeSetList (
      3 Exclude 1 1
    )
    Container ( # segment 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # segment 2
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Vertex Attribute Set List

Full name: Data, AttributeSetList, VertexAttributeSetList
Drawable: No
Parent Class Heirarchy: Data, AttributeSetList
Binary type: vasl
Ascii type: VertexAttributeSetList
Binary size: 12 + nIndices * sizeof(Uns) + padding
Parent Objects: ALWAYS: GeneralPolygon, Line, Mesh, Polygon, PolyLine, Triangle, TriGrid
Format: Data Format
Subobjects: many AttributeSet (order-dependent)br> **Inherited:** No
Referencable: No

Description:
The vertex attribute set list specifies a list of attributes to be attached to a set of vertices determined by the parents topology.

Packing for this object is identical to the other attribute set lists.

Data structure:
Uns32 nObjects
PackingEnum packing
Uns32 nIndices
Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum described in FaceAttributeSetList

Text samples:

```
Container (
  Triangle (
    0 0 0
    0 2 0
    0 0 2
  )
  Container (
    VertexAttributeSetList (
      3 Exclude 0
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 0 )
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
```

```
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
)
```

Camera Placement

Full name: Data, CameraData, CameraPlacement
Drawable: No
Parent Class Heirarchy: Data, CameraData
Binary type: cmpl
Ascii type: CameraPlacement
Binary size: 36
Parent Objects: ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The camera placement specifies the location and orientation of the camera in space, by a camera location, a point of interest, and an up vector. This placement locates and orients the camera, and defines a space in which the rest of the parameters are interpreted.

If the up vector is not of unit length upon reading, it should be normalized by the reading program.

The camera placement is affected by the current transformation state in a hierarchy. The location and point of interest are multiplied by the current transformation directly, and the up vector is multiplied by the current transformation minus any translation component of the transform, and unitized.

The camera vector is defined as:
camera vector = (pointOfInterest - location)

Data structure:
Point3D location
Point3D pointOfInterest
Vector3D upVector

upVector ^ (pointOfInterest - location)

|upVector| = 1.0

Default Values:
0 0 1 # location

0 0 0 # pointOfInterest
0 1 0 # upVector

Text samples:

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraPlacement (
     10 0 0 # located along X axis
      0 0 0 # point of interest is origin
      0 1 0 # Y is up
  )
)
```

Camera Range

Full name: Data, CameraData, CameraRange
Drawable: No
Parent Class Heirarchy: Data, CameraData
Binary type: cmrg
Ascii type: CameraRange
Binary size: 8
Parent Objects: ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The camera range affects the clipping of the viewing frustum.

This is used to bound the range of the set of objects of interest.

Hither is the frontmost clipping plane (sometimes referred to as near), yon is the backmost clipping plane (sometimes referred to as far).

Each of these distances is measured along the camera vector, described in the Camera Placement object.

Data structure:
Float32 hither
Float32 yon

0 < hither yon

default is:

hither e
yon

Text samples:

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraRange (
     0.1 2 # hither, yon
  )
)
```

Camera ViewPort

Full name: Data, CameraData, CameraViewPort
Drawable: No
Parent Class Heirarchy: Data, CameraData
Binary type: cmvp
Ascii type: CameraViewPort
Binary size: 16
Parent Objects: ALWAYS: any Camera object: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The camera viewport specifies a rectangular region of the viewing frustum to which the image is clipped. Effectively the view port may be used to zoom in on a particular feature of an image.

The view port uses the cartesian coordinate system, with Y towards the top of the screen, X to the right, and Z coming towards the viewer, as shown in the diagram.

Data structure:
Point2D origin
Float32 width
Float32 height

-1 origin.x 1
-1 origin.y 1
0 < width 2
0 < height 2

Default is:
-1 1 # origin
2 # width
2 # height

Text samples:

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraViewPort ( # zoom to 200%
    -0.5 0.5 1 1
  )
)
```

Bottom Cap Attribute Set

Full name: Data, CapData, BottomCapAttributeSet
Drawable: No
Parent Class Heirarchy: Data, CapData
Binary type: bcas
Ascii type: BottomCapAttributeSet
Binary size: 0
Parent Objects: ALWAYS: Cone, Cylinder
Format: No Data
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: No

Description:
This object simply allows the attributes associated with the bottom cap of a Cone or Cylinder to be encapsulated.

Presence of a bottom cap attribute set does not neccessarily mean the bottom cap is drawn.

The Caps object determines whether the Cone and Cylinder caps are drawn or not.

Data structure:

Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
```

```
    capColor: Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
  )
)
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Reference (1)
  )
)
...
toc: TableOfContents (
   ...
   1 capColor>
)
```

Caps

Full name: Data, CapData, Caps
Drawable: No
Parent Class Heirarchy: Data, CapData
Binary type: caps
Ascii type: Caps
Binary size: 4
Parent Objects: ALWAYS: Cone, Cylinder
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
In the binary file, the upper 28 bits of the caps bitfield should be ignored. In the text file, unknown bitfield strings should be skipped. The default caps value is 0, or None.

The Top cap bit (label) is ignored in the Cone.

Data structure:
CapsFlags caps

CapsFlags is defined as:

Binary Text
0x00000000 None
0x00000001 Bottom
0x00000002 Top

Default is:
None

Text samples:

```
Container (
  Cylinder ( )
  Caps ( Bottom | Top )
)

Container ( # Cone with a blue bottom
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Face Cap Attribute Set

Full name: Data, CapData, FaceCapAttributeSet
Drawable: No
Parent Class Heirarchy: Data, CapData
Binary type: fcas
Ascii type: FaceCapAttributeSet
Binary size: 0
Parent Objects: ALWAYS: Cone, Cylinder
Format: No Data
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: No

Description:
Attaches a set of attributes to the face cap of the cone and cylinder primitives. For the cone, its indicated in the diagram.

Data structure:

Text samples:

```
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    FaceCapAttributeSet ( )
    Container (
```

```
      AttributeSet ( )
      DiffuseColor ( 0.2 0.9 0.4 )
    )
  )
)
```

Top Cap Attribute Set

Full name: Data, CapData, TopCapAttributeSet
Drawable: No
Parent Class Heirarchy: Data, CapData
Binary type: tcas
Ascii type: TopCapAttributeSet
Binary size: 0
Parent Objects: ALWAYS: Cylinder
Format: No Data
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: No

Description:
Attaches a set of attributes to the top cap of the cylinder primitive.

Presence of a top cap attribute set does not neccessarily mean the top cap is drawn.

The Caps object determines whether the Cylinder caps are drawn or not.

Data structure:

Text samples:

```
Container (
  Cylinder ( )
  Caps ( Top )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0.2 0.9 0.4 )
    )
  )
)
```

Display Group State

Full name: Data, DisplayGroupState

Drawable: No
Parent Class Heirarchy: Data
Binary type: dgst
Ascii type: DisplayGroupState
Binary size: 4
Parent Objects: ALWAYS: DisplayGroup, OrderedDisplayGroup
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
This piece of data is a subobject only to objects of type display group. It affects how a display group is traversed. These flags allow any display group to have the following characteristics:

To have invisible objects in a scene which may act as user interface items, or may aid in bounding complex geometries
To have non-user interface items which may serve only as decoration and should not be picked.
To have a group of shaders/attributes which affects the state as an inline group so it may be instantiated and inherited in many parts of a hierarchy.

Data structure:
DisplayGroupStateFlags traversalFlags

DisplayGroupStateFlags is:

Binary Text
0x00000000 None
0x00000001 Inline
0x00000002 DoNotDraw
0x00000004 NoBoundingBox
0x00000008 NoBoundingSphere
0x00000010 DoNotPick

default is:

Binary Text
0x00000000 None

Text samples:

```
# to pick a chess piece by a box around it

BeginGroup ( DisplayGroup ( ) )
  PickIDStyle ( 1 )
  BeginGroup (
    DisplayGroup ( )
    DisplayGroupState ( DoNotDraw )
  )
    Scale ( 2 4 2 )
```

```
    Box ( )
  EndGroup ( )

  Container (
    DisplayGroup ( )
    DisplayGroupState ( DoNotPick )
  )
  Mesh ( # chess piece
    56 # nVertices
    0.2 0.3 0.5
    ...
  )
  EndGroup ( )
EndGroup ( )
```

General Polygon Hint

Full name: Data, GeneralPolygonHint
Drawable: No
Parent Class Heirarchy: Data
Binary type: gplh
Ascii type: GeneralPolygonHint
Binary size: 4
Parent Objects: ALWAYS: GeneralPolygon
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The GeneralPolygonHint gives a reading application some hint of what shape a general polygon is.

A Complex general polygon may contain intersecting, concave, or convex polygons.

A Concave general polygon contains no intersecting polygons, but contains 1 or more concave polygons.

A Convex general polygon indicates that all contained polygons are convex and non-intersecting.

Data structure:
GeneralPolygonHintEnum shapeHint

GeneralPolygonHintEnum is:

Binary Text
0x00000000 Complex
0x00000001 Concave
0x00000002 Convex

default is:
Complex

Text samples:

```
Container (
  GeneralPolygon (
     1
     3
     0 2 3
     0 2 1
     2 0 0
  )
  GeneralPolygonHint ( Convex )
)
```

Light Data

Full name: Data, LightData
Drawable: No
Parent Class Heirarchy: Data
Binary type: lght
Ascii type: LightData
Binary size: 20
Parent Objects: ALWAYS: any Light: SpotLight, AmbientLight, PointLight, DirectionalLight
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The light data object affects information about a light that is common among all lights.

A light may be on or off, may vary in intensity, or may have different colors.

Data structure:
Boolean isOn
Float32 intensity
ColorRGB color

0 intensity 1

Default is:
True # isOn
1.0 # intensity
1 1 1 # color

Text samples:

```
Container (
  AmbientLight ( )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

Mesh Corners

Full name: Data, MeshCorners
Drawable: No
Parent Class Heirarchy: Data
Binary type: crnr
Ascii type: MeshCorners
Binary size: 4 + sizeof(corners[0..nCorners-1]) sizeof(MeshCorner) = 8 + nFaces * 4
Parent Objects: ALWAYS: Mesh
Format: Data Format
Subobjects: nCorners AttributeSets (order-dependent)
Inherited: No
Referencable: No

Description:
Mesh Corners allow you to attach AttributeSets to a mesh vertex, to allow for attributes to be associated with a particular face-vertex pair. This may be used to allow sharp corners in an object (diagram above), to set different shading parameters for adjacent faces, etc.

Mesh corners supplies a vertex index, a list of face indices, and a vertex attribute set for each corner.

The mesh corners object most often appears inside a container, and always has AttributeSet subobjects. The first corner in the mesh corners data is mapped to the first attribute set subobject, the second corner to the second attribute set, etc.

Data structure:
Uns32 nCorners
MeshCorner corners[nCorners]

0 < nCorners
where MeshCorner is:

Uns32 vertexIndex
Uns32 nFaces
Uns32 faces[nFaces]

0 < nFaces

Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshCorners (
      2 # numCorners

      # Corner 0
      5 # vertexIndex
      2 # faces
      25 26 # face indices

      # Corner 1
      5 # vertexIndex
      2 # faces
      23 24 # face indices
    )
    Container (
      AttributeSet ( )
      Normal ( -0.2 0.8 0.3 )
    )
    Container (
      AttributeSet ( )
      Normal ( -0.7 -0.1 0.4 )
    )
  )
)
```

Mesh Edges

Full name: Data, MeshEdges
Drawable: No
Parent Class Heirarchy: Data
Binary type: edge
Ascii type: MeshEdges
Binary size: 4 + sizeof(corners[0..nCorners-1]) sizeof(MeshEdges) = 2 * sizeof(Uns)
Parent Objects: ALWAYS: Mesh
Format: Data Format
Subobjects: nCorners AttributeSets (order-dependent)
Inherited: No
Referencable: No

Description:
Mesh Edges allow you to attach AttributeSets to a mesh edge.

You may attach mesh edges to any edge in the mesh that corresponds to a face edge. To specify and

edge that should have an attribute set attached to it, include it as the nth edge the list of edges, and specify the attribute set as the nth attribute set subobject.

Data structure:
Uns32 nEdges
MeshEdge edges[nEdges]

0 < nEdges
where MeshEdge is:

Uns32 vertexIndex1
Uns32 vertexIndex2

Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshEdges (
      2 # numEdges
      0 1 # 1st edge vertexIndices
      1 2 # 2nd edge vertexIndices
    )
    Container ( /* 1st edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.2 0.8 0.3 )
    )
    Container ( /* 2nd edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.8 0.2 0.3 )
    )
  )
)
```

NURB Curve 2D

Full name: Data, NURBCurve2D
Drawable: No
Parent Class Heirarchy: Data
Binary type: nb2c
Ascii type: NURBCurve2D
Binary size: 8 + 12 * nPoints + 4 * (order + nPoints)
Parent Objects: ALWAYS: TrimCurves
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The NURB Curve 2D is a subobject of the TrimCurves object, and supplies a 2 dimensional curve to trim NURB Patches.

Data structure:
Uns32 order
Uns32 nPoints
RationalPoint3D points[nPoints]
Float32 knots[order + nPoints]

2 order
2 nPoints
0 < points[...].w (weights of points)

Text samples:

Shader Data

Full name: Data, ShaderData
Drawable: No
Parent Class Heirarchy: Data
Binary type: shdr
Ascii type: ShaderData
Binary size: 8
Parent Objects: ALWAYS: any Shader
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The shader data initializes boundary wrapping conditions for a shader.

Data structure:
ShaderUVBoundaryEnum uBounds
ShaderUVBoundaryEnum vBounds

ShaderUVBoundaryEnum is:

Binary Text
0x00000000 Wrap
0x00000001 Clamp

default is:
Wrap Wrap

Text samples:

```
Container (
  CustomShader ( ... )
  ShaderData ( Wrap Clamp )
)
```

Shader Transform

Full name: Data, ShaderTransform
Drawable: No
Parent Class Heirarchy: Data
Binary type: sdxf
Ascii type: ShaderTransform
Binary size: 64
Parent Objects: ALWAYS: any Shader
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
This transforms a shaded object into another world space coordinate system. It does not affect how the object is drawn, or the current state of the hierarchy.

Data structure:
Matrix4x4 shaderTransform

Text samples:

```
Container (
  3DMarbleShader ( )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
...
Type ( -3 Apple:ATG:3DMarbleShader )
Container (
  -3 ( 2.3 1.0 -10 )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
```

Shader UV Transform

Full name: Data, ShaderUVTransform
Drawable: No
Parent Class Heirarchy: Data
Binary type: sduv
Ascii type: ShaderUVTransform
Binary size: 36
Parent Objects: ALWAYS: any Shader
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The Shader UV transform allows the uvs on a geometric object to be transformed before shading occurs.

This allows you to rotate a texture map, for example.

Data structure:
Matrix3x3 matrix

Text samples:

```
Container (
  TextureShader ( )
  ShaderUVTransform (
    1 0 0
    0 1 0
    0.2 0.3 1
  )
  PixmapTexture (
    ...
  )
)
```

Trim Curves

Full name: Data, TrimLoop
Drawable: No
Parent Class Heirarchy: Data
Binary type: trml
Ascii type: TrimLoop
Binary size: 0

Parent Objects: ALWAYS: NURBPatch
Format: No Data
Subobjects: many NURBCurve2D (order-dependent)
Inherited: No
Referencable: No

Description:
The Trim Loop subobject allows users to attach trimming loops to a NURB Patch. The Trim Loop object contains no data, and serves only as an encapsulation of various 2-dimensional curves used for trimming.

The Trim loop object contains a sequence of 2 dimensional curves which are concatenated together to form a loop. The subobjects are order-dependent. Each trim loop subobject should contain loops that are geometrically continuous, meaning the first trim curves end point ends at the next trim curves starting point.

In the metafile version 1.0, the only 2-dimensional curve allowed is a NURBCurve2D.

In future releases of the metafile, we expect to add additional types of 2d trim curves for trimming NURBS.

Data structure:

Text samples:

```
Container (
 NURBPatch (
  4 4 4 4 # u,v order, num M,N points
  -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1
  -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1
  -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
  -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots
 )
 Container (
  TrimLoop ( )
  NURBCurve2D (
    ...
  )
  NURBCurve2D (
    ...
  )
 )
)
```

Image Clear Color

Full name: Data, ViewHintsData, ImageClearColor
Drawable: No

Parent Class Heirarchy: Data, ViewHintsData
Binary type: imcc
Ascii type: ImageClearColor
Binary size: 12
Parent Objects: ALWAYS: ViewHints
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
This specifies the preferred rgb color with should be used to clear the drawing areas background.

Data structure:
ColorRGB clearColor

Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageClearColor ( 1 1 1 ) # white
)
Box ( )
```

Image Dimensions

Full name: Data, ViewHintsData, ImageDimensions
Drawable: No
Parent Class Heirarchy: Data, ViewHintsData
Binary type: imdm
Ascii type: ImageDimensions
Binary size: 8
Parent Objects: ALWAYS: ViewHints
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The image dimensions specifies the preferred image width and height in bits. It is a subobject of the view hints, which aids an application in determining how to display an image.

Data structure:
Uns32 width
Uns32 height

0 < width
0 < height

Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageDimensions ( 32 32 )
  ImageClearColor ( 1 1 1 )
)
Rotate ( X 0.75 )
Rotate ( Y 0.75 )
Container (
  AttributeSet ( )
  DiffuseColor ( 1 0 0 )
)
Box ( )
```

Image Mask

Full name: Data, ViewHintsData, ImageMask
Drawable: No
Parent Class Heirarchy: Data, ViewHintsData
Binary type: immk
Ascii type: ImageMask
Binary size: 12 + (rowBytes * height) + padding
Parent Objects: ALWAYS: ViewHints
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The image mask is a bitmap that specifies how an images rendered pixels should be clipped. The origin of the bitmap (the upper-left) is aligned with the origin (upper left) of the drawing area. Generally, the image mask and the image dimensions are used simultaneously to specify an image which is partially clipped.

The example to the right specifies a mask to clip a 32x32 image. The application using this data uses this clip mask to only render to a clipped portion of a custom document icon in this case, the bitmap will only draw inside of a document icon, providing a small preview in the Finder with a black document icon. The image mask to the right was used to render the example above.

Data structure:
Uns32 width
Uns32 height
Uns32 rowBytes

EndianEnum bitOrder
RawData image[rowBytes * height]

width, height in bits
0 < width
0 < height
((width >> 3) + ((width & 0x7) ? 1 : 0)) rowBytes
EndianEnum is:

Binary Text
0x00000000 BigEndian
0x00000001 LittleEndian

Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageDimensions ( 32 32 )
  ImageClearColor ( 1 1 1 )
  ImageMask (
    32 32 # width, height
    4 # rowBytes
    BigEndian # bitOrder
    0x000000000FFFF8000FFFF8000FFFF800
    0x0FFFF8000FFFF8000FFFF8000FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0C61FFE00F24FFE00E64FFE00F24FFE0
    0x0F24FFE00C61FFE00FFFFFE000000000
  )
)
Rotate ( X 0.25 )
Rotate ( Y 0.23 )
Container (
  Torus ( 0 0.7 0 0 0 1 1 0 0 0 0 0 0.7 )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.9 0.9 )
  )
)
```

Ambient Coefficient

Full name: Element, Attribute, AmbientCoefficient
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: camb
Ascii type: AmbientCoefficient
Binary size: 4

Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The ambient coefficient describes the intensity of the ambient light that is reflected by a surface.

Data structure:
Float32 ambientCoefficent

0 ambientCoefficient 1.0

Text samples:

```
Container (
  AttributeSet ( )
  AmbientCoefficient ( 0.7 )
)
```

Diffuse Color

Full name: Element, Attribute, DiffuseColor
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: kdif
Ascii type: DiffuseColor
Binary size: 12
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The diffuse color indicates the amount of diffuse light reflected by a surface.

Data structure:
ColorRGB diffuseColor

Text samples:

```
Container (
  AttributeSet ( )
  DiffuseColor ( 1 0 0 ) # red
)
```

Highlight State

Full name: Element, Attribute, HighlightState
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: hlst
Ascii type: HighlightState
Binary size: 4
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The highlight state attribute, when true, indicates that the current attribute state is overridden with the current highlight styles attribute set. The highlight state attribute allows various portions of a geometry object to be highlighted for user interface, etc. while retaining the integrity of a geometrys attribute set.

Data structure:
Boolean highlighted

Text samples:

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 ) # RED
  )
)
...
Container (
  Polygon (
    3
    0 1 2
    0 0 0
    0 -1 2
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 1 2 )
    HighlightState ( True )
    # polygon is drawn RED
  )
)
```

Normal

Full name: Element, Attribute, Normal
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: nrml
Ascii type: Normal
Binary size: 12
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
If normal is not of unit length upon reading, it should be normalized. (npi)

The normal indicates the surface normal at a vertex.

Data structure:
Vector3D normal

|normal| = 1

Text samples:

```
Container (
  Polygon (
     5
     0.23423 0.56434 0.2312
     ...
  )
  Container (
    VertexAttributeSetList ( 5 Exclude 0 )
    Container (
      AttributeSet ( )
      Normal ( 0.8 -0.1 -0.1 )
    )
  )
)
```

Shading UV

Full name: Element, Attribute, ShadingUV
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: shuv
Ascii type: ShadingUV

Binary size: 8
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The shading UV indicates an alternate UV to the Surface UV for shading purposes.

Shading UVs are generally used by shaders that affect appearance information, such as texture maps, which alter the color on a geometric surface.

Surface UVs are generally used for trimming.

Data structure:
Param2D shadingUV

Any UV parametrization is allowed, however, shading generally occurs with the following values.

0 shadingUV.u 1
0 shadingUV.v 1

Text samples:

```
Container (
  AttributeSet ( )
  ShadingUV ( 0 0 )
)
```

Specular Color

Full name: Element, Attribute, SpecularColor
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: kspc
Ascii type: SpecularColor
Binary size: 12
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The specular color indicates the color of specular highlights on a surface.

Data structure:
ColorRGB specularColor

Text samples:

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.1 0.1 0.1 ) # near-black
  SpecularColor ( 1 1 1 ) # white
)
Sphere (
  0 0 0
  0 1 0
  1 0 0
  0 0 1
)
```

Specular Control

Full name: Element, Attribute, SpecularControl
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: cspc
Ascii type: SpecularControl
Binary size: 4
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The specular control attribute indicates the power to which the specular component of lighting computations is raised.

Data structure:
Float32 specularControl

0 specularControl

Text samples:

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.5 0.5 0.5 ) # near-black
  SpecularColor ( 0.5 ) # white highlights
  SpecularControl ( 1 ) # larger highlight area
)
```

```
Sphere ( )
```

Surface Tangent

Full name: Element, Attribute, SurfaceTangent
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: srtn
Ascii type: SurfaceTangent
Binary size: 24
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The surface tangent attribute indicates the direction of changing U and V on a surface.

Data structure:
Vector3D paramU
Vector3D paramV

Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      ...
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0.1 0.293 )
      SurfaceTangent (
        1 0 0
        0 1 0
      )
    )
  )
)
```

Surface UV

Full name: Element, Attribute, SurfaceUV

Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: sruv
Ascii type: SurfaceUV
Binary size: 8
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The surface UV indicates an alternate UV to the shading UV for shading purposes.

Surface UVs are generally used for trim shaders.

Shading UVs are generally used by shaders that affect appearance information, such as texture maps, which alter the color on a geometric surface.

Data structure:
Param2D surfaceUV

Any UV parametrization is allowed, however, shading generally occurs with the following values.

0 surfaceUV 1
0 surfaceUV 1

Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      200 Include 4 10 21 22 11
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 0 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 0 )
    )
```

```
  )
)
```

Transparency Color

Full name: Element, Attribute, TransparencyColor
Drawable: No
Parent Class Heirarchy: Element, Attribute
Binary type: kxpr
Ascii type: TransparencyColor
Binary size: 12
Parent Objects: ALWAYS: AttributeSet
Format: Data Format
Subobjects: none
Inherited: No
Referencable: No

Description:
The transparency color indicates the degree of light allowed to pass though the various channels (r,g,b) of a surface.

A color of (1, 1, 1) indicates complete transparency (meaning 100% of the light behind an object is allowed to pass through), a color of (0, 0, 0) indicates complete opacity (meaning no light passes through an object.)

Data structure:
ColorRGB transparency

Text samples:

```
Container (
  Polygon (
    ...
  )
  Container (
    AttributeSet ( )
    TransparencyColor ( 1 0 0 )
  )
)
```

Generic Renderer

Full name: Shared, Renderer, GenericRenderer
Drawable: No
Parent Class Heirarchy: Shared, Renderer

Binary type: gnrr
Ascii type: GenericRenderer
Binary size: 0
Parent Objects: SOMETIMES: ViewHints
Format: No Data
Subobjects: none
Inherited: No
Referencable: Yes

Description:
A renderer that doesnt do anything, but may be used to accumulate state or for picking.

Data structure:

Text samples:

```
Container (
  ViewHints ( )
  GenericRenderer ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Interactive Renderer

Full name: Shared, Renderer, InteractiveRenderer
Drawable: No
Parent Class Heirarchy: Shared, Renderer
Binary type: ctwn
Ascii type: InteractiveRenderer
Binary size: 0
Parent Objects: SOMETIMES: ViewHints
Format: No Data
Subobjects: none
Inherited: No
Referencable: Yes

Description:
The interactive renderer.

This will be renamed later when the corresponding product is named.

Data structure:

Text samples:

```
Container (
  ViewHints ( )
  InteractiveRenderer ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Wire Frame Renderer

Full name: Shared, Renderer, WireFrame
Drawable: No
Parent Class Heirarchy: Shared, Renderer
Binary type: wrfr
Ascii type: WireFrame
Binary size: 0
Parent Objects: SOMETIMES: ViewHints
Format: No Data
Subobjects: none
Inherited: No
Referencable: Yes

Description:
A wireframe renderer.

Data structure:

Text samples:

```
Container (
  ViewHints ( )
  Wireframe ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Attribute Set

Full name: Shared, Set, AttributeSet
Drawable: No
Parent Class Heirarchy: Shared, Set
Binary type: attr
Ascii type: AttributeSet
Binary size: 0
Parent Objects: any AttributeSetList, any Geometry, any Group, any CapAttributeSet
Format: No Data
Subobjects: 1 AmbientCoefficient (optional) 1 DiffuseColor (optional) 1 HighlightState (optional) 1 Normal (optional) 1 ShadingUV (optional) 1 SpecularColor (optional) 1 SpecularControl (optional) 1 SurfaceTangent (optional) 1 SurfaceUV (optional) 1 TransparencyColor (optional) 1 SurfaceShader (optional)
Inherited: No
Referencable: Yes

Description:
A attribute set groups sets of unique attributes together and is associated with a vertex, face, or an entire geometry. Any object that is an Element may be placed in an attribute set.

An attribute set also may be placed in a group. The various attributes in an attribute set are inherited to nodes lower than it in a hierarchy.

Data structure:

Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      30 Exclude 2
      29 30
    )
    ...
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
      SurfaceUV ( 0.87 0.57 )
    )
    ...
  )
)
```

Orthographic Camera

Full name: Shared, Shape, Camera, OrthographicCamera
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Camera
Binary type: orth
Ascii type: OrthographicCamera
Binary size: 16
Parent Objects: SOMETIMES: ViewHints
Format: Data Format
Subobjects: 1 CameraPlacement (optional, default) 1 CameraViewPort (optional, default) 1 CameraRange (optional, default)
Inherited: No
Referencable: Yes

Description:
The lens characteristics are set with the dimensions of a rectangular view port in the frame of the camera.

Data structure:
Float32 left
Float32 top
Float32 right
Float32 bottom

left < right
bottom < top

Text samples:

```
OrthographicCamera (
  -1 -1 1 1
)

Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraPlacement (
    0 0 20
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

View Angle Aspect Camera

Full name: Shared, Shape, Camera, ViewAngleAspectCamera
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Camera
Binary type: vana
Ascii type: ViewAngleAspectCamera
Binary size: 8
Parent Objects: SOMETIMES: ViewHints
Format: Data Format
Subobjects: 1 CameraPlacement (optional, default)1 CameraViewPort (optional, default)1 CameraRange (optional, default)
Inherited: No
Referencable: Yes

Description:
A perspective camera specified in terms of the minimum view angle and the aspect ratio of X to Y.

Data structure:
Float32 fieldOfView
Float32 aspectRatioXtoY

0 < fieldOfView
0 < aspectRatioXtoY

Text samples:

```
ViewAngleAspectCamera (
  1.7 1.0
)

Container (
  ViewAngleAspectCamera (
    1.7 1.0
  )
  CameraPlacement (
    0 0 20
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

View Plane Camera

Full name: Shared, Shape, Camera, ViewPlaneCamera

Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Camera
Binary type: vwpl
Ascii type: ViewPlaneCamera
Binary size: 20
Parent Objects: SOMETIMES: ViewHints
Format: Data Format
Subobjects: 1 CameraPlacement (optional, default)1 CameraViewPort (optional, default)1 CameraRange (optional, default)
Inherited: No
Referencable: Yes

Description:
A view plane camera is a view angle aspect camera specified in terms of an arbitrary view plane. This is most useful when setting the camera to look at a particular object.

The viewPlane is set to distance from the camera to the object.

The halfWidth is set to half the width of the cross section of the object, and the halfHeight equal to the halfWidth divided by the aspect ratio of the viewPort.

This is the only perspective camera with specifications for off-axis viewing, which is desirable for scrolling.

Data structure:
Float32 viewPlane
Float32 halfWidthAtViewPlane
Float32 halfHeightAtViewPlane
Float32 centerXOnViewPlane
Float32 centerYOnViewPlane

0 < viewPlane
0 < halfWidthAtViewPlane
0 < halfHeightAtViewPlane
centerXOnViewPlane, centerYOnViewPlane may be any value

Text samples:

```
ViewPlaneCamera (
  ...
)

Container (
  ViewPlaneCamera (
    20
    15.0 15.0
    18 29
  )
  CameraPlacement (
    0 0 20
```

```
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

Box

Full name: Shared, Shape, Geometry, Box
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: box
Ascii type: Box
Binary size: 0 or 48
Parent Objects:
Format: Data Format
Subobjects: 1 FaceAttributeSetList (optional, nObjects = 6)1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
This is a rectangular parallelepiped

A size of zero indicates the default values, helpful in instantiating a unit-cube.

The Face Attribute Set List subobject assigns color to the following faces:

Face ^ orientation at origin + orientation
Face ^ orientation at origin
Face ^ majorAxis at origin + majorAxis
Face ^ majorAxis at origin
Face ^ minorAxis at origin + minorAxis
Face ^ minorAxis at origin

Basically, the faces perpendicular to the orientation direction are assigned first, then the majorAxis, then the minorAxis.

Data structure:
Vector3D orientation
Vector3D majorAxis
Vector3D minorAxis
Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin

Text samples:

```
Box ( )
```

```
Box (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)
```

```
Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Exclude 0
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

Cone

Full name: Shared, Shape, Geometry, Cone
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry

Binary type: cone
Ascii type: Cone
Binary size: 0 or 48
Parent Objects:
Format: Data Format
Subobjects: 1 Caps (optional, default) 1 FaceCapAttributeSet (optional) 1 BottomCapAttributeSet (optional) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
A cone may have a cap, and may have attributes assigned to the entire geometry, to the face cap, or to the bottom cap.

The default parametrization is shown in the diagram.

Data structure:
Vector3D orientation
Vector3D majorAxis
Vector3D minorAxis
Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin

Text samples:

```
Cone ( )
```

```
Cone (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)
```

```
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
```

```
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

Cylinder

Full name: Shared, Shape, Geometry, Cylinder
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: cyln
Ascii type: Cylinder
Binary size: 0 or 48
Parent Objects:
Format: Data Format
Subobjects: 1 Caps (optional, default) 1 TopCapAttributeSet (optional) 1 FaceCapAttributeSet (optional) 1 BottomCapAttributeSet (optional) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
A cylinder may have either top or bottom caps, and may have attributes assigned to the entire geometry, to the face cap, the bottom cap, or the top cap.

The default parametrization is shown in the diagram.

Data structure:
Vector3D orientation
Vector3D majorRadius
Vector3D minorRadius
Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin

Text samples:

```
Cylinder ( )
```

```
Cylinder (
  2 0 0
  0 1 1
```

```
  2 3 0
  0 0 0
)

Container (
  Cylinder ( )
  Caps ( Bottom | Top )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
  )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

Disk

Full name: Shared, Shape, Geometry, Disk
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: disk
Ascii type: Disk
Binary size: 0 or 36
Parent Objects:
Format: Data Format
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
This is an elliptical disk at the given origin with two vectors specifying the dimensions.

The default parametrization is shown in the diagram.

Data structure:
Vector3D majorRadius

Vector3D minorRadius
Point3D origin

For 0-sized objects, default is:

1 0 0 # majorRadius
0 1 0 # minorRadius
0 0 0 # origin

Text samples:

```
Disk ( )
```

```
Disk (
  2 0 0
  0 1 1
  0 0 0
)
```

```
Container (
  Cylinder ( )
  Caps ( Bottom | Top )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

Ellipse

Full name: Shared, Shape, Geometry, Ellipse
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: elps
Ascii type: Ellipse
Binary size: 0 or 36
Parent Objects:
Format: Data Format

Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
This is an ellipse at the given origin with two vectors specifying its dimensions.

There is no default parametrization for an ellipse.

Data structure:
Vector3D majorAxis
Vector3D minorAxis
Point3D origin

For 0-sized objects, default is:

1 0 0 # majorAxis
0 1 0 # minorAxis
0 0 0 # origin

Text samples:

```
Ellipse ( )

Ellipse (
  2 0 0
  0 1 1
  0 0 0
)

Container (
  Ellipse ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

Ellipsoid

Full name: Shared, Shape, Geometry, Ellipsoid
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: elpd
Ascii type: Ellipsoid
Binary size: 0 or 48
Parent Objects:
Format: Data Format

Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
An ellipsoid may have an attribute set attached to it.

The default parametrization is shown in the diagram. V is zero to the left of majorRadius, and is 1 to the right. U is zero at the orientation vector, and 0 at the bottom.

Data structure:
Vector3D orientation
Vector3D majorRadius
Vector3D minorRadius
Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorRadius
0 0 1 # minorRadius
0 0 0 # origin

Text samples:

```
Sphere ( )
```

```
Sphere (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)
```

```
Container (
  Sphere ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

General Polygon

Full name: Shared, Shape, Geometry, GeneralPolygon
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: gpgn

Ascii type: GeneralPolygon
Binary size: 4 + sizeof(polygons[0..nCoutours-1]) sizeof(PolygonData) = 4 + nVertices * 12
Parent Objects:
Format: Data Format
Subobjects: 1 VertexAttributeSetList (optional, nObjects = nVertices[0] + ... + nVertices[nContours-1]) 1 AttributeSet (optional) 1 GeneralPolygonHint (optional)
Inherited: No
Referencable: Yes

Description:
A general polygon is a polygon that may be convex or may contain holes. A general polygon also assumes that all faces are planar within floating point tolerances.

Holes are indicated by specifying a contour of the generalPolygon in clockwise order.

Polygons that cross use the even-odd rule to specify holes (see diagram).

You may specify the complexity of a GeneralPolygon by adding a viewHints object.

Data structure:
Uns32 nContours
PolygonData polygons[nContours]

where PolygonData is:

Uns32 nVertices
Point3D vertices[nVertices]

0 < nContours
2 < nVertices

Text samples:

```
Container (
 GeneralPolygon (
  2 # nContours
  3 # nVertices
  -1 0 0
  1 0 0
  0 1.7 0
  3 # nVertices
  -1 0.4 0
  1 0.4 0
  0 2.1 0
 )
 Container (
  VertexAttributeSetList ( 6 Exclude 2 0 4 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 0 1 )
  )
  Container (
```

```
   AttributeSet ( )
   DiffuseColor ( 0 1 1 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 1 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 1 0 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 1 1 1 )
 )
)
```

Line

Full name: Shared, Shape, Geometry, Line
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: line
Ascii type: Line
Binary size: 24
Parent Objects:
Format: Data Format
Subobjects: 1 VertexAttributeSetList (optional, nObjects = 2) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
Our basic line primitive is a line segment, a simple line drawn between two vertices.

Optional vertex attributes may be attached using a VertexAttributeSetList.

A set of attributes may be applied to the entire line segment by attaching an attribute set.

Data structure:
Point3D start
Point3D end

Text samples:

```
Line ( 0 0 0 1 0 0 )
```

```
Container (
  Line (
    0 0 0
```

```
    1 0 0
  )
  Container (
    VertexAttributeSetList ( 2 Exclude 0 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Marker

Full name: Shared, Shape, Geometry, Marker
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: mrkr
Ascii type: Marker
Binary size: 32 + (rowBytes * height) + padding
Parent Objects:
Format: Data Format
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The marker is used to rasterize bitmaps parallel to the viewing plane. They are used for annotation of an image.

Data structure:
Point3D location
Int32 xOffset
Int32 yOffset
Uns32 width
Uns32 height
Uns32 rowBytes
EndianEnum bitEndian
RawData data[height * rowBytes]

0 < width
0 < height
(((width / 8) + ((width & 7) > 0)) rowBytes
EndianEnum is:

Binary Text
0x00000000 BigEndian
0x00000001 LittleEndian

Text samples:

```
Container (
 Marker (
  0.5 0.5 0.5 # origin
  -28 # xOffset
  -3 # yOffset
  56 # width
  6 # height
  7 # rowBytes
  BigEndian # bitOrder
  0x7E3C3C667E7C18606066666066187C3C
  0x607E7C661860066066607C1860066666
  0x6066007E3C3C667E6618
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.2 0.6 )
 )
)
```

Mesh

Full name: Shared, Shape, Geometry, Mesh
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: mesh
Ascii type: Mesh
Binary size: 4 + nVertices * 12 + 8 + (nFaces+nContours) * sizeof(faces[0..nFaces+nContours-1] sizeof(MeshFace) = sizeof(Int) + sizeof(Uns) * nFaceVertexIndices
Parent Objects:
Format: Data Format
Subobjects: 1 FaceAttributeSetList (optional, nObjects = nFaces) 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 MeshCorners (optional) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The mesh is used for representing complex topological objects. It contains enough information to determine which polygonal faces are adjacent to each other without numerical ambiguity. This metafile object contains topological as well as geometrical information.

A contour (hole) in a face is indicated by supplying a negative number for the number of vertices, and adds a hole to the previous face that was not a contour.

The size of nFaceVertexIndices and faceVertexIndices is based on the value of nVertices.

We introduce a special subobject used only with the mesh, called MeshCorners. This object allows multiple attribute sets to be attached to a single vertex, where each attribute set is bound to a set of vertex-face pairs. It can be used to place a sharp edge in the mesh (if the attribute set contains a normal, for instance).

Data structure:
Uns32 nVertices
Point3D vertices[nVertices]
Uns32 nFaces
Uns32 nContours
MeshFace faces[nFaces + nContours]

where MeshFace is:

Int32 nFaceVertexIndices
Uns32 faceVertexIndices[nFaceVertexIndices]

3 nVertices
3 nFaceVertexIndices

Text samples:

```
Mesh (
 10 # nVertices
 -1 1 1
 -1 1 -1
 1 1 -1
 1 -1 -1
 1 -1 1
 0 -1 1
 -1 -1 0
 -1 -1 -1
 1 1 1
 -1 0 1
 7 # nFaces
 0 # nContours
 3 6 5 9
 5 7 6 9 0 1
 4 2 3 7 1
 4 2 8 4 3
 4 1 0 8 2
 5 4 8 0 9 5
 5 3 4 5 6 7
)
```

NURB Curve

Full name: Shared, Shape, Geometry, NURBCurve

Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: nrbc
Ascii type: NURBCurve
Binary size: 8 + (nPoints * 12) + ((nPoints + order) * 4)
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
NURB curves are Non-Uniform Rational B-spline curves. A rational B-spline curve is a curve in 4D space, which has been projected down to 3D space. Thus, the control points for a 3D rational curve have four components - x, y, z, and w (usually known as the weight). For such a point, the corresponding point in 3D space is (x/w, y/w, z/w)

Weights (w) are always positive.

Data structure:
Uns32 order
Uns32 nPoints
RationalPoint4D points[nPoints]
Float32 knots[order + nPoints]

2 order
2 nPoints
0 < points[...].w (weights of points)

Text samples:

```
NURBCurve (
 4 7 # order, nPoints
 0 0 0 1 # points
 1 1 0 1
 2 0 0 1
 3 1 0 1
 4 0 0 1
 5 1 0 1
 6 0 0 1
 0 0 0 0 0.25 0.5 0.75 1 1 1 1 # knots
)
```

NURB Patch

Full name: Shared, Shape, Geometry, NURBPatch
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry

Binary type: nrbp
Ascii type: NURBPatch
Binary size: 16 + (16 * numColumns * numRows) + ((uOrder + numColumns) * 4) + ((vOrder + numRows) * 4)
Parent Objects:
Format: Data Format
Subobjects: 1 TrimCurves (optional)
Inherited: No
Referencable: Yes

Description:
Non-Uniform Rational B-Spline (NURB) Patches are closed under projective transformations, can represent quadrics exactly, and can be refined locally to allow additional detail.

The default parametrization is given by the knot vectors.

Weights (w) are always positive.

Data structure:
Uns32 uOrder
Uns32 vOrder
Uns32 numColumns
Uns32 numRows
RationalPoint4D points[numMPoints*numNPoints]
Float32 uKnots[uOrder + numColumns]
Float32 vKnots[vOrder + numRows]

2 numColumns
2 numRows
2 uOrder
2 vOrder
0 < points[...].w (weights of points)

Text samples:

```
NURBPatch (
 4 4 4 4 # u,v order, num M,N points
 -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1
 -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1
 -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
 -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
  0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots
)
```

Point

Full name: Shared, Shape, Geometry, Point

Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: pnt
Ascii type: Point
Binary size: 12
Parent Objects:
Format: Data Format
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The basic point primitive is an infinitesimally small point in space. It is specified as a 3D point plus an optional attribute set.

A 3D point has no default parametrization.

Data structure:
Point3D point

Text samples:

```
Point ( 0 1 2 )
```

Polygon

Full name: Shared, Shape, Geometry, Polygon
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: plyg
Ascii type: Polygon
Binary size: 4 + nVertices * 12
Parent Objects:
Format: Data Format
Subobjects: 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The polygon is convex with no holes. To describe concave polygons or polygons with holes, use the general polygon primitive.

The points that make up a polygons face are assumed to be planar within floating point tolerances.

Data structure:

Uns32 nVertices
Point3D vertices[nVertices]

2 nVertices

Text samples:

```
Polygon (
  4
  0 1 1
  0 -1 1
  0 -1 -1
  0 1 -1
)
```

Poly Line

Full name: Shared, Shape, Geometry, PolyLine
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: plyl
Ascii type: PolyLine
Binary size: 4 + nVertices * 12
Parent Objects:
Format: Data Format
Subobjects: 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 GeometryAttributeSetList (optional, nObjects = nVertices - 1) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
An extension of the basic line primitive is a polyline, where simple lines are drawn between adjacent points in a point list

A polyline is NOT closed, and the last point is never connected to the first point.

A polyline has no default parametrization.

Data structure:
Uns32 nVertices
Point3D vertices[nVertices]

2 nVertices

Text samples:

```
Container (
```

```
  PolyLine (
    4
    -1 -0.5 -0.25
    -0.5 1.5 0.45
    0 0 0
    1.5 1.5 1
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.4 0.2 0.9 )
  )
)
```

Torus

Full name: Shared, Shape, Geometry, Torus
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: tors
Ascii type: Torus
Binary size: 0 or 52
Parent Objects:
Format: Data Format
Subobjects: 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The orientation length specifies the radius of the circular along the orientation vector of the torus cross-section.

The major and minor axes are vectors to the center of the torus cross-section (as in the diagram).

The ratio is the change in the orientation length in the axial direction. A ratio of 2, for example, creates a fatter torus cross-section along the major and minor axes, a ratio of 0.5 creates a fatter cross-section along the orientation.

As far as anyone knows, the torus is useful for drawing donuts and bagels, and makes a great demo.

The default parametrization is shown in the diagram.

Data structure:
Vector3D orientation
Vector3D majorAxis
Vector3D minorAxis
Point3D origin
Float32 ratio

For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
1 # ratio

Text samples:

```
Torus ( )
```

```
Torus (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
  1
)
```

```
Container (
  Torus ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

Triangle

Full name: Shared, Shape, Geometry, Triangle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: trng
Ascii type: Triangle
Binary size: 36
Parent Objects:
Format: Data Format
Subobjects: 1 VertexAttributeSetList (optional, nObjects = 3) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:
The most basic polygon is a triangle, which contains 3 points.

A VertexAttributeSetList may be used to attach attribute sets to the vertices (containing three vertex attribute sets) or an optional AttributeSet may be added to attach to the face.

There is no default parametrization for a triangle.

Data structure:
Point3D vertices[3]

Text samples:

```
Container (
 Triangle (
  -1 -0.5 -0.25
  0 0 0
  -0.5 1.5 0.45
 )
 Container (
  VertexAttributeSetList ( 3 Exclude 0 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 0 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 1 0 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 0 1 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.5 0.2 )
 )
)
```

Tri Grid

Full name: Shared, Shape, Geometry, TriGrid
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Geometry
Binary type: trig
Ascii type: TriGrid
Binary size: 8 + (nColumns * nRows * 12)
Parent Objects:
Format: Data Format
Subobjects: 1 FaceAttributeSetList (optional, nObjects = (numNVertices - 1) * (numMVertices - 1) * 2) 1 VertexAttributeSetList (optional, nObjects = numNVertices * numMVertices attribute sets) 1 AttributeSet (optional)
Inherited: No
Referencable: Yes

Description:

Points specified are given in row major order.

You may add a FaceAttributeSetList to attach a set of attributes for each of the triangles generated by this primitive.

You may also add a VertexAttributeSetList to attach attributes to each vertex.

Data structure:
Uns32 nColumns
Uns32 nRows
Point3D points[numMVertices * numNVertices]

2 nColumns
2 nRows

Text samples:

```
Container (
 TriGrid (
  3 4 # nUVertices nVVertices
  -1 1 1      -0.5 1 0  0 1 0
   0.7 1 0.5  -1 0 0    -0.5 0 0.3
   0 0.2 0    0.5 0 0   -1 -1 0
   -0.5 -1 0  0 -1 0.1  0.2 -1.3 0.2
 )
 Container (
  FaceAttributeSetList ( 12 Include 1 5 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 0.5 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.7 0.3 )
 )
)
```

Group

Full name: Shared, Shape, Group
Drawable: Yes
Parent Class Heirarchy: Shared, Shape
Binary type: grup
Ascii type: Group
Binary size: 0
Parent Objects: none
Format: No Data
Subobjects: none
Inherited: No

Referencable: Yes

Description:
The group is useful for grouping any type of shared objects together.

It is delimited by an end group object.

Data structure:

Text samples:

```
BeginGroup ( Group ( ) )
  CString ( This is the first day of the rest of your life. )
  Torus ( )
EndGroup ( )
```

Display Group

Full name: Shared, Shape, Group, DisplayGroup
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Group
Binary type: dspg
Ascii type: DisplayGroup
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 DisplayGroupState (optional)
Inherited: No
Referencable: Yes

Description:
A display group contains only objects that are drawable.

A display group adds the ability to be traversed for various operations via the DisplayGroupState subobject.

It is delimited by an end group object.

Data structure:

Text samples:

IO Proxy Display Group

Full name: Shared, Shape, Group, DisplayGroup, IOProxyDisplayGroup
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Group, DisplayGroup
Binary type: iopx
Ascii type: IOProxyDisplayGroup
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 DisplayGroupState (optional, default)
Inherited: No
Referencable: Yes

Description:
The IO proxy display group contains drawable objects that are similar representations of the same object in different formats. For example, if it is known that a particular application does not understand NURBPatchs, the writing application may write the NURBPatch in an IO proxy group along with a mesh which is the tesselated NURBPatch.

The objects in a IO proxy display group appear in their preferencial order. The first object is the most preferred representation, the last object the least. The first object that is understood by a reading application should be used.

You may specify a group of objects inside a IOProxyDisplayGroup, as a group (up to its EndGroup) delimiter is a single object.

It is understood that ONLY the first understood object in an IO proxy display group is traversed while drawing, bounding, or picking.

In other words, if an IO proxy display group contains many objects, only one of them will be drawn when it comes time to render an image, etc.

Data structure:

Text samples:

```
BeginGroup ( IOProxyDisplayGroup ( ) )
  Mesh (
    8
    0 0 0
    0 0 1
    0 1 0
    1 0 0
    1 1 0
    0 1 1
    1 0 1
    1 1 1
    ... etc.
  )
  Box ( )
EndGroup ( )
```

```
BeginGroup ( IOProxyDisplayGroup ( ) )
  NURBPatch (       # preferred object
    ...
  )
  DisplayGroup ( )  # 2nd choice object
    Translate ( 1 2 3 )
    Box ( )
  EndGroup ( )
EndGroup ( )
```

Ordered Display Group

Full name: Shared, Shape, Group, DisplayGroup, OrderedDisplayGroup
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Group, DisplayGroup
Binary type: ordg
Ascii type: OrderedDisplayGroup
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 DisplayGroupState (optional, default)
Inherited: No
Referencable: Yes

Description:
The ordered display group is simply a display group except that objects are sorted by type. Objects always appear in an ordered group in the following order:

Transforms
Styles
AttributeSets
Shaders
Geometries
DisplayGroups

It is delimited by an end group object.

Data structure:

Text samples:

Info Group

Full name: Shared, Shape, Group, InfoGroup
Drawable: Yes

Parent Class Heirarchy: Shared, Shape, Group
Binary type: info
Ascii type: InfoGroup
Binary size: 0
Parent Objects: none
Format: No Data
Subobjects: none
Inherited: No
Referencable: Yes

Description:
An info group contains nothing but String objects. It is used to add human-readable information pertaining to a files origin or history. A use that comes to mind is copyright notices.

The info group object should be preserved by a reading application, and appended with additional information if a file is re-written.

It is delimited by an end group object.

Data structure:

Text samples:

```
BeginGroup ( InfoGroup ( ) )
  CString (
    Copyright  1995 Apple Computer, Inc. )
  CString (
    Author: Bonanza Jellybean )
EndGroup ( )
```

Light Group

Full name: Shared, Shape, Group, LightGroup
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Group
Binary type: lghg
Ascii type: LightGroup
Binary size: 0
Parent Objects: none
Format: No Data
Subobjects: none
Inherited: No
Referencable: Yes

Description:
A light group contains nothing but lights.

It is delimited by an end group object.

Data structure:

Text samples:

```
BeginGroup ( LightGroup ( ) )
  AmbientLight ( )
  DirectionalLight ( 1 0 0 False )
EndGroup ( )
```

Ambient Light

Full name: Shared, Shape, Light, AmbientLight
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Light
Binary type: ambn
Ascii type: AmbientLight
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 LightData (optional, default)
Inherited: No
Referencable: Yes

Description:
An ambient light supplies light that comes from secondary reflections.

In lieu of other light sources, the ambient light illuminates the scene with a flat, uniform light.

Data structure:

Text samples:

```
AmbientLight ( )

Container (
  AmbientLight ( )
  LightData (
    EcTrue # isOn
    1.0 # intensity
    1 0 0 # red color
  )
)
```

Directional Light

Full name: Shared, Shape, Light, DirectionalLight
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Light
Binary type: drct
Ascii type: DirectionalLight
Binary size:
Parent Objects:
Format: Data Format
Subobjects: 1 LightData (optional, defaults)
Inherited: No
Referencable: Yes

Description:
A directional light is far enough away from the scene that we may treat it as though it were infinitely far away. This produces shading results faster than any other type of light (except ambient).

It is specified with a vector pointing in the same direction as the light rays, an attenuation and a boolean value indicating whether this light casts shadows or not.

Data structure:
Vector3D direction
Boolean castsShadows

|direction| = 1.0

Text samples:

```
DirectionalLight ( 1 0 0 True )
```

```
Container (
  DirectionalLight ( 1 0 0 True )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

Point Light

Full name: Shared, Shape, Light, PointLight
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Light
Binary type: pntl
Ascii type: PointLight
Binary size:

Parent Objects:
Format: Data Format
Subobjects: 1 LightData (optional, defaults)
Inherited: No
Referencable: Yes

Description:
A point light is a light at an infinitesimally small point in space. It may be attenuated or it may cast shadows.

Data structure:
Point3D location
Attenuation attenuation
Boolean castsShadows

where Attenuation is the structure:

Float32 c0
Float32 c1
Float32 c2

attenuation is computed, using d as the distance from location:

1
c0 + c1*d + c2 * d^2

0 < c0
0 < c1
0 < c2

attenuation is not clamped to [0,1] to allow for lighting washout (such as in a nuclear explosion)

Text samples:

```
PointLight (
  12 23 2
  0 0 1 # InverseDistanceSquared
  True
)

Container (
  PointLight (
    12 23 2
    0 0 1 # InverseDistanceSquared
    True
  )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

Spot Light

Full name: Shared, Shape, Light, SpotLight
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Light
Binary type: spot
Ascii type: SpotLight
Binary size:
Parent Objects:
Format: Data Format
Subobjects: 1 LightData (optional, defaults)
Inherited: No
Referencable: Yes

Description:
A spot light radiates with a circular cone of light that tapers toward the edge of the cone.

The hotSpotAngle is the angle (in radians) from the axis of the spot light for which the spot light has maximum, constant intensity. The outer angle is the angle for which the light falls to zero. Between these two, the light intensity tapers to zero according to the FallOff enumerated type.

Data structure:
Point3D location
Vector3D orientation
Boolean castsShadows
Attenuation attenuation
Float32 hotAngle
Float32 outerAngle
FallOffEnum fallOff

|orientation| = 1

Attenuation is described in the Point Light

0 < hotAngle outerAngle

FallOffEnum is:

Binary Text
0x00000000 None
0x00000001 Linear
0x00000002 Exponential
0x00000003 Cosine

Text samples:

```
SpotLight (
  12 0 0
  0 1 0
  True
  0 0 1 # InverseDistanceSquared
  0.7 # hotAngle
  0.8 # outerAngle
  Cosine
)

Container (
  SpotLight (
    12 0 0
    0 1 0
    True
    0 0 1 # InverseDistanceSquared
    0.7 # hotAngle
    0.8 # outerAngle
   Cosine
  )
  LightData (
    True
    0.4
    1 0 1
  )
)
```

Lambert Illumination

Full name: Shared, Shape, Shader, IlluminationShader, LambertIllumination
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Shader, IlluminationShader
Binary type: lmil
Ascii type: LambertIllumination
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The lambertian illumination model.

Data structure:

Text samples:

```
LambertIllumination ( )
```

Phong Illumination

Full name: Shared, Shape, Shader, IlluminationShader, PhongIllumination
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Shader, IlluminationShader
Binary type: phil
Ascii type: PhongIllumination
Binary size: 0
Parent Objects:
Format: No Data
Subobjects:
Inherited: Yes
Referencable: Yes

Description:
The phong illumination model.

Data structure:

Text samples:

```
PhongIllumination ( )
```

Texture Shader

Full name: Shared, Shape, Shader, SurfaceShader, TextureShader
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Shader, SurfaceShader
Binary type: txsu
Ascii type: TextureShader
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 PixmapTexture (required)
Inherited: Yes
Referencable: Yes

Description:
The texture shader is used to perform shading using a texture (in this case, a PixmapTexture).

Data structure:

Text samples:

```
Container (
  TextureShader ( )
  PixmapTexture (
    ...
  )
)
```

Backfacing Style

Full name: Shared, Shape, Style, BackfacingStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: bckf
Ascii type: BackfacingStyle
Binary size:
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The backfacing style tells a renderer how to clip backfacing polygons while rendering.

Data structure:
BackfacingEnum backfacing

where BackfacingEnum is:

Text Binary
0x00000000 Both
0x00000001 Culled
0x00000002 Flipped

Text samples:

```
BackfacingStyle ( Culled )
```

Fill Style

Full name: Shared, Shape, Style, FillStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style

Binary type: fist
Ascii type: FillStyle
Binary size: 4
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The fill style tells a renderer what parts of a polygon to draw.

Data structure:
FillStyleEnum fillStyle

where FillStyleEnum is:

Text Binary
0x00000000 Filled
0x00000001 Edges
0x00000002 Points
0x00000003 Empty

Text samples:

```
FillStyle ( Edges )
```

Highlight Style

Full name: Shared, Shape, Style, HighlightStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: high
Ascii type: HighlightStyle
Binary size: 0
Parent Objects:
Format: No Data
Subobjects: 1 AttributeSet (required)
Inherited: Yes
Referencable: Yes

Description:
The highlight style sets the binding for highlighting features of a geometry via the HighlightState attribute. The attribute set subobject sets the highlight attribute set.

Data structure:

Text samples:

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 0 1 )
  )
)
```

Interpolation Style

Full name: Shared, Shape, Style, InterpolationStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: intp
Ascii type: InterpolationStyle
Binary size: 4
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The interpolation style tells a renderer how to interpolate shading values on a polygon.

Data structure:
InterpolationStyleEnum interpolationStyle

where InterpolationStyleEnum is:

Binary Text
0x00000000 None
0x00000001 Vertex
0x00000002 Pixel

Text samples:

```
InterpolationStyle ( Vertex )
```

Orientation Style

Full name: Shared, Shape, Style, OrientationStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: ornt
Ascii type: OrientationStyle
Binary size: 4
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The Orientation style is used to change the orientation of polygons.

Data structure:
OrientationEnum orientation

where OrientationEnum is:

Binary Text
0x00000000 CounterClockwise
0x00000001 Clockwise

Text samples:

```
OrientationStyle ( Clockwise )
```

Pick ID Style

Full name: Shared, Shape, Style, PickIDStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: pkid
Ascii type: PickIDStyle
Binary size: 4
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The pick ID style is used to allow the user to insert ids within a hierarchy to aid in picking a hierarchy.

Data structure:
Uns32 id

Text samples:

```
PickIDStyle ( 23 )
```

Pick Parts Style

Full name: Shared, Shape, Style, PickPartsStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: pkpt
Ascii type: PickPartsStyle
Binary size:
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The pick parts style determines the level of granularity for picking.

Data structure:
PickPartsFlags pickParts

where PickPartsFlags is:

Text Binary
0x00000000 Object
0x00000001 Face
0x00000002 Edge
0x00000004 Vertex

default is:
Object

Text samples:

```
PickPartsStyle ( Object | Vertex )
```

Receive Shadows Style

Full name: Shared, Shape, Style, ReceiveShadowsStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: rcsh
Ascii type: ReceiveShadowsStyle
Binary size: 4
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The receive shadows style determines whether a geometry receives shadows when rendering. It is coupled with the casts shadows field in all lights, excluding the ambient light.

Data structure:
Boolean receiveShadows

Text samples:

```
ReceiveShadowsStyle ( True )
```

Subdivision Style

Full name: Shared, Shape, Style, SubdivisionStyle
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Style
Binary type: sbdv
Ascii type: SubdivisionStyle
Binary size: (subdivisionMethod == Constant) ? 12 : 8
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The subdivision style tells a geometric decomposition the courseness of a geometric primitive tesselation. There are three methods of subdivision: constant, world space, and screen space subdivision.

Constant subdivision supplies 2 integral values, which indicate the number of sections the u and v axes of a decomposition should be divided into.

The Screen Space value indicates average size of a single polygon in a tesselation in screen space.

The world space value indicates the average size of a single polygon in a tesselation in world space.

Data structure:
This object has two forms, based on the subdivison method field:

for subdivisionMethod == WorldSpace or ScreenSpace the structure is:

SubdivisionMethodEnum subdivisionMethod
Float32 value1

for subdivisionMethod == Constant, the values are integral:

SubdivisionMethodEnum subdivisionMethod
Uns32 value1
Uns32 value2

where SubdivisionMethodEnum is:

Binary Text
0x00000000 Constant
0x00000001 WorldSpace
0x00000002 ScreenSpace

Text samples:

```
SubdivisionStyle (
   Constant 12 12
)

SubdivisionStyle (
   WorldSpace 50
)

SubdivisionStyle (
   ScreenSpace 50
)
```

Matrix Transform

Full name: Shared, Shape, Transform, Matrix
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: mtrx
Ascii type: Matrix
Binary size: 64
Parent Objects:

Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
A custom, invertible matrix transform.

Data structure:
Matrix4x4 matrix

matrix is invertible

Text samples:

Quaternion Transform

Full name: Shared, Shape, Transform, Quaternion
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: qtrn
Ascii type: Quaternion
Binary size: 16
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
The quaternion specifies three axes of rotation and a twist value.

Useful for user interface.

Data structure:
Float32 w
Float32 x
Float32 y
Float32 z

Text samples:

```
Quaternion ( 0.2 0.7 0.2 1.57 )
```

Rotate Transform

Full name: Shared, Shape, Transform, Rotate
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: rott
Ascii type: Rotate
Binary size:
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
Rotate about the X, Y, or Z axes.

Data structure:
AxisEnum axis
Float32 radians

AxisEnum is:

Binary Text
0x00000000 X
0x00000001 Y
0x00000002 Z

Text samples:

```
Rotate ( X 1.57 )
```

Rotate About Axis Transform

Full name: Shared, Shape, Transform, RotateAboutAxis
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: rtaa
Ascii type: RotateAboutAxis
Binary size: 28
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes

Referencable: Yes

Description:
Rotate about an arbitrary axis in space.

Data structure:
Point3D origin
Vector3D orientation
Float32 radians

|orientation| = 1

Text samples:

```
RotateAboutAxis (
  20 0 0 # origin
  0  1 0 # orientation
  1.57 # radians
)
```

Rotate About Point Transform

Full name: Shared, Shape, Transform, RotateAboutPoint
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: rtap
Ascii type: RotateAboutPoint
Binary size: 20
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
To rotate about the X, Y, or Z axes at an arbitrary point in space.

Data structure:
AxisEnum axis
Float32 radians
Point3D origin

AxisEnum is:

Binary Text
0x00000000 X

0x00000001 Y
0x00000002 Z

Text samples:

Scale Transform

Full name: Shared, Shape, Transform, Scale
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: scal
Ascii type: Scale
Binary size:
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
A scale transform.

Data structure:
Vector3D scale

scale.x 0.0
scale.y 0.0
scale.z 0.0

Text samples:

```
Scale ( 1 1 2 )
```

Translate Transform

Full name: Shared, Shape, Transform, Translate
Drawable: Yes
Parent Class Heirarchy: Shared, Shape, Transform
Binary type: trns
Ascii type: Translate
Binary size: 12
Parent Objects:
Format: Data Format

Subobjects: none
Inherited: Yes
Referencable: Yes

Description:
A translate transfrom.

Data structure:
Vector3D translate

Text samples:

```
Translate ( 1 2 100 )
```

Unknown Binary

Full name: Shared, Shape, UnknownBinary
Drawable: Yes
Parent Class Heirarchy: Shared, Shape
Binary type: ukbn
Ascii type: UnknownBinary
Binary size: 12 +
Parent Objects:
Format: Data Format
Subobjects:
Inherited: No
Referencable: Yes

Description:
The unknown binary object is a way of transporting unknown data found in a binary file. It is an encapsulated replica of the original data found in a binary metafile, containing the object type (an Int32), the object size (in bytes), the byte order of the original file, and the data itself. The byte order is needed if unknown data is transported across different processors, and allows for parsing endian-specific primitives within the raw data block.

Unknown binary objects may be written in either the text or binary files.

When an unknown binary object is encountered in a metafile, it is up to the reading program to either:
transport the data around
validate it and convert it to a known object
discard the data

Unknown objects are inherently dirty, meaning you may assume the unknown binary object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

Data structure:
Int32 objectType
Uns32 objectSize
EndianEnum byteOrder
RawData objectData[objectSize]

Text samples:

```
UnknownBinary (
  1701605476
  4
  BigEndian
  0x0AB2
)
```

Unknown Text

Full name: Shared, Shape, UnknownText
Drawable: Yes
Parent Class Heirarchy: Shared, Shape
Binary type: uktx
Ascii type: UnknownText
Binary size: sizeof(name) + sizeof(data)
Parent Objects: any
Format: Data Format
Subobjects:
Inherited: No
Referencable: Yes

Description:
The unknown text object is a way of transporting unknown data found in a text file. It is an encapsulated replica of the original data found in a text metafile, containing the object type (a String), and a text string containing the original data. In some cases, white space and comments may have been stripped from the contents field.

Unknown text objects may be written in either the text or binary files.

When an unknown text object is encountered in a metafile, it is up to the reading program to either:
transport the data around
validate it and convert it to a known object
discard the data

Unknown objects are inherently dirty, meaning you may assume the unknown text object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

Data structure:
String asciiName

String contents

Text samples:

```
UnknownText (
  Ellipsoid

)
```

Macintosh Path

Full name: Shared, Storage, MacintoshPath
Drawable: No
Parent Class Heirarchy: Shared, Storage
Binary type: macp
Ascii type: MacintoshPath
Binary size: sizeof(String)
Parent Objects: ALWAYS: Reference
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
The Macintosh path specifies the pathname of an external file reference using the pathname specification found in the Inside Macintosh volumes. (essentially, a colon-based separator)

Data structure:
String pathName

Text samples:

```
Container (
  Reference ( 43 )
  MacintoshPath ( :::Foo:Bar:Models:Cheryl )
)
```

Unix Path

Full name: Shared, Storage, UnixPath
Drawable: No
Parent Class Heirarchy: Shared, Storage
Binary type: unix
Ascii type: UnixPath

Binary size: sizeof(String)
Parent Objects: ALWAYS: Reference
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
The unix path object serves as a way to reference files on a unix file system.

The path should obey naming standards for unix operating systems.

Data structure:
String unixPath

Text samples:

```
Container (
  Reference ( 23 )
  UnixPath ( ./shaders.eb )
)
```

C String

Full name: Shared, String, CString
Drawable: No
Parent Class Heirarchy: Shared, String
Binary type: strc
Ascii type: CString
Binary size: sizeof(String)
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
The CString is a way of embedding text in a metafile.

Other string types allow for internationalization.

The only allowable characters in a CString are 7-bit ASCII numbers.

The following characters may be escaped with the \ character:

'\a', '\b', '\f', '\n', '\r', '\t', '\v', '\', '\\'

Data structure:
String cString

Text samples:

```
CString (
  Copyright (c) 1994 Apple Computer, Inc.
)
```

Unicode

Full name: Shared, String, Unicode
Drawable: No
Parent Class Heirarchy: Shared, String
Binary type: uncd
Ascii type: Unicode
Binary size: 4 + length * 2
Parent Objects:
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
The unicode object is another way of embedding text in a metafile.

See UNICODE reference for details.

Data structure:
Uns32 length
RawData unicode[length * 2]

Text samples:

```
Unicode (
  6
  0x457363686572
)
```

Pixmap Texture

Full name: Shared, Texture, PixmapTexture
Drawable: No
Parent Class Heirarchy: Shared, Texture

Binary type: txpm
Ascii type: PixmapTexture
Binary size: 28 + rowBytes * height + padding
Parent Objects: SOMETIMES: TextureShader
Format: Data Format
Subobjects: none
Inherited: No
Referencable: Yes

Description:
A generic means of transferring pixmap data. Used in the Texture Shader.

Data structure:
Uns32 width
Uns32 height
Uns32 rowBytes
Uns32 pixelSize
PixelTypeEnum pixelType
EndianEnum bitOrder
EndianEnum byteOrder
RawData image[rowBytes * height]

0 < width
0 < height
0 < pixelSize < 32
width * pixelSize rowBytes
PixelTypeEnum is:

Binary Text
0x00000000 RGB8
0x00000001 RGB16
0x00000002 RGB24
0x00000003 RGB32

EndianEnum is:

Binary Text
0x00000000 BigEndian
0x00000001 LittleEndian

Text samples:

```
PixmapTexture (
   256 256 # width/height
   128 # rowBytes
   32 # pixelSize
   RGB24
   BigEndian BigEndian
   0x00123232...
```

```
   0x...
)
```

View Hints

Full name: Shared, ViewHints
Drawable: No
Parent Class Heirarchy: Shared
Binary type: vwhn
Ascii type: ViewHints
Binary size: 0
Parent Objects: none
Format: No Data
Subobjects: 1 Renderer (optional) 1 Camera (optional)many Lights (optional) 1 AttributeSet (optional) 1 ImageDimensions (optional) 1 ImageMask (optional) 1 ImageClearColor (optional)
Inherited: No
Referencable: Yes

Description:
The subobjects of the view hints object specifies the preferences supplied by a writing application when rendering a scene.

The semantic to be followed when a view hints object is encountered in the metafile is that the view hints is specified previous to a list of objects to be rendered to that particular view hints preference. The subobjects of the view hints object are inherited from the previous view hints in a metafile.

For example, if a modelling application contains 10 camera locations for viewing various portions of a scene, it would first store the default view as the first object in a metafile, then the group representing the scene, then a view containing the second camera position, then a reference to the scene, etc.

Data structure:

Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 0 30
      0 0 0
      0 1 0
    )
  )
  DirectionalLight ( -0.7 -0.7 -0.65 )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
```

```
    SpecularControl ( 3 )
  )
  ImageDimensions ( 200 200 )
)
ref1:
BeginGroup ( DisplayGroup ( ) )
...
EndGroup ( )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 10 0
      0 0 0
      0 1 0
    )
  )
)
Reference ( 1 )
```

How to use your RS232 or IRDA port for Remote Control

Update History - skip to Introduction if this is your first visit

26/02/2001
FAQ page added

28/02/2001
FAQ No.2 updated to include extra information on component selection

26/08/2002
Theory updated
Hardware schematic and text updated
Software Section text and code updated (winsamp now version 1.3)
FAQ updated
******* This was a big update - please let me know if I've broken anything (that used to work!) *******

Introduction

Firstly, an apology. There is a lot to read on this page and I'm struggling to make it even slightly visually appealing. If you feel that the subject is worthy of your attention, then I suggest you print this page out or at least save it to disk, snag the zip file mentioned in the software section (not a big download) then study it all offline.

Unlike some of my other stuff, this one is fairly serious. It is intended to introduce a concept and demonstrate its application rather than present the definitive finished product, but having said that, everything needed to get it working properly on a PC is included, with enough information and flexibility to enable the software as supplied to be incorporated into much more elaborate Windows front-ends if desired. The idea is not limited to PC platforms, however - most devices with serial or IRDA ports are possible candidates for the technique as described, and samples made on one platform should transfer easily to other platforms.

A number of people have done good work in the field of turning a PC into a 'learning' universal infrared remote control, for various appliances such as TVs, video recorders, satellite decoders etc. The common methods appear to use either an IR detector and transmitter circuit attached to the parallel port, or an 'intelligent' device accepting high-level commands attached to a serial port. A quick online search will turn up schematics, construction details, driver software and some very attractive Windows front-ends for this purpose.

Following on from my Furby experiments (http://www.veg.nildram.co.uk/furby.htm), I set out to prove that it is possible to control the TV etc using either an IRDA port or a standard RS232 port with extremely minimalist hardware. The reasons were threefold. First, the challenge. Second, the convenience or geek value - many PCs, PDAs and other devices already come with IRDA as standard, wouldn't it be nice to use it for something different. Third, DEVICE INDEPENDENCE AND PORTABILITY OF SAMPLES - serial ports of one type or another are fairly ubiquitous,

more so than parallel ports anyway, and a drawback with sampling and playback through the parallel port is the heavy dependence on processor speed and environment - samples made on one speed of machine may not work reliably on a different speed of machine (no criticism intended, but this is what I found when I tried it), so to have a good probability of success you have to sample your remotes on the machine you intend to use for playing them back - consequently, you can't simply sample your Sony CD remote and send the file off to your pal in Greenland when he loses his [ahaaa - until now :-)]. Also, most systems have more than one COM port, so it's no great hardship tying one of them up permanently for remote control. The next bit is a disclaimer of sorts, then we'll get on with the theory.

Disclaimer - Please read this - Important

What I am about to describe is my own original idea, my own software, my own project. As far as I am aware, nobody else does it, or has done it, this way, but if they have, what can I say? Great minds think alike? I am not intentionally ripping off anyone else's work, and have worked long and hard to get it into its present state, for my own amusement, with no prior knowledge that it can or cannot be done or indeed that it has been attempted in this fashion. The ideas and software have been put here because I thought you might be interested in how I did it. It all works as far as I'm concerned, but I'm not asking or telling you to use it in any way, shape or form. The software (should you choose to try it) can write small text and binary files containing sample information to your disk, or whatever you run it from. Also, the Windows demo application keeps a couple of settings in the Registry to enable it to remember preferences from one session to the next. None of this should cause you any distress, but as with all things, it could always go horribly wrong, so you have been warned. If you don't want to risk using my software, you should be able to write your own using the information in the theory section. Also be advised that there is a class of IR remote control (so-called 'flash' controls) which is not supported by the software in its present form, but the project is still in a state of flux and I may add support later - the issue here is one of trying to keep it simple enough for anyone to use when they don't have the use of a proper 'scope - I'm sure you don't wish to become an expert on remotes (not that I am, but a certain amount of it inevitably rubs off).

Anyway, with the software and hardware described, I routinely control a Toshiba TV, JVC video recorder and Maspro satellite receiver either using a laptop irda port or a normal 232 port on my desktop PC, using samples made from the original remotes and without having to know details of the three different protocols actually employed by the devices. There is a good chance that most other appliances will work equally well ('flash' devices being known exceptions). If you can't get a device to work, the sampler software will at least provide information to help identify the problem if you wish to look deeper into it.

Finally, if you would like me to investigate the implementation of this technique on other platforms, you should send me non-returnable hardware and development software for the experiments, and if you make a fortune based on the idea, I certainly wouldn't mind you giving me a share :-)

Blank Frank 24th November, 1999.

Theory - Updated 26th August, 2002

Ok. Now to the good bit. Without getting too technical, here is an idea of what's going on and why. Though there is a mind-bogglingly large selection of remote control protocols at the bit level, based on quite a number of types of protocol at the packet level, the vast majority of ir remote controls function by sending data at relatively low rates by transmitting bursts of (carrier) pulses of various lengths with periods of silence inbetween, also of various lengths. The data is encoded in the lengths of the bursts and silences, and receivers demodulate the carrier to recover the baseband data. The carrier frequency is typically somewhere in the range 36 - 40 kHz, and receivers are designed to allow signals of this frequency range through (+/- a few kHz) while rejecting signals outwith this range in order to reject such things as strip lights.

You may wish to brush up on your serial port theory for this - lots of info online, maybe enough on the Furby page. I make the assumption that most receivers will be happy with combs of pulses at 38.4kHz, since they are designed round R-C filters and diode pumps (at least conceptually, though most use one integrated device). I can achieve this by transmitting suitable characters at 115200 baud out of any serial port, so long as there are no gaps between characters. The precise pattern for a comb is the character value 0xdb, at 115k2/8/e/2 (0x5b at 115k2/7/n/1 would also be possible but would require data to be loaded into the uart more frequently). There is sufficient tolerance in most receivers to cope with the effective 4-pulse granularity (or 3-pulse if 7/n/1) in the comb. The next assumption I make is that out-of-frequency-band pulses will be rejected by the receiver, so if I transmit 0xfe, still at 115k2/8/e/2, this will look like silence to the receiver. So that's the transmission part covered - by sending correctly-sized groups of characters with values 0xdb and 0xfe, I can effectively construct an infrared stream which looks like it came from a 'real' remote control from the receiver's point of view, so long as I keep the uart continuously stuffed with data. The timing is derived from the uart, not the raw speed of the PC or other host device, and it is fairly easy to keep up.

Sampling works as follows. Using the very simple hardware described below, (or your own equivalent - doesn't need to be a handshake line, could be a joystick or printer input, for example), I transmit a junk character (0xff as it happens, but it doesn't matter) continuously out the serial port with no gaps between characters. While waiting for each character to be transmitted, I sample the IR input as frequently as possible (in my case a handshake line) and if I ever see an 'active' condition on it (ie if IR is ever detected) I assume that there is a comb (36 - 40kHz) of data being received from a remote control during this period, and if I don't see activity I assume this is a silent period. I grab a big array, with one bucket for each character time, and the buckets will either be marked as active or silent. After a preset number of characters have been sent, I stop grabbing and process the data. What I end up with is another array of buckets which contain alternating active counts and silent counts, stored as numbers of character times, adjusted to compensate for the inherent tendency of this method to round up comb lengths and round down silence lengths. Arrays like this provide a compact method of sample storage, and are very easy to play back when I want to transmit the sample as described above, and again, the timing is absolute, based on the uart, not the processor raw speed, so it is accurate and repeatable. By understanding the actual protocol used, it would be easy to compute the packets rather than store samples as such and make the storage space requirement much smaller, but I'm trying to avoid this to keep things simple and flexible. There are a few details to do with timeouts, sample sizes,

waiting for the data to start, suspending interrupts etc which will be explained elsewhere, but the basic principle is very simple and apparently rock solid.

It is not possible to sample remotes through the IRDA port - at least with my software, but probably not at all on a PC - due to the design of the port hardware. Samples have to be made using a 232 port using the hardware described below, or equivalent, but this should not be a problem. Playback through the IRDA port works fine so long as you are aware of the potential pitfalls (again covered to some extent on my Furby page, and in particular relating to operation under Windows).

I have run the same software on an 8MHz 286 (yes, I managed to find one that still works) and a K6-2 333 and got the same results. Samples made on both extremes of machine look the same and can be exchanged without problems. This suggests that the technique will work well on a wide range of devices with serial ports, not just PCs. The actual sample-grabbing process is designed to scream along, even on a slow machine, though the post-grab processing can take as much time as it needs since it is not real-time critical.

Update 26th August, 2002
Things move on. Since writing the above section, PCs have got considerably faster. I am happy to report that the system has now beem tested unchanged on Athlon 1000 boxes and above, without problems, under Windows 95 / 98SE, and, from the feedback I've had, the project has been constructed and run on a variety of PCs, and implemented on and some other platforms, in various parts of the world.

Now the interesting news. I have figured out a way of retaining all the uart-derived timing properties of transmission described above while achieving 'real' silence during the silent periods rather than the combs of out-of-frequency-band pulses which have been used up to now. It works on all the PCs I can get my hands on. The latest version of the DOS program (version 1.3) implements this and no longer has the option to switch in software timing loops to achieve (with some calibration effort) the same effect. It takes advantage of some particular features of the PC uart hardware, so I wouldn't class it as a new method, rather as an optimisation aimed at PCs. Since the vast majority of users are PC users, this new version should be of general interest, though I stand by the original method as described above as the more general solution. I have to say that I personally don't have a problem using the original silence method with the receivers I'm controlling, but this reworked version may be of some help with borderline cases, and this is now the 'official version' - I've moved over to it anyway, because it has a couple of other features I wanted, which are described elsewhere in the docs.

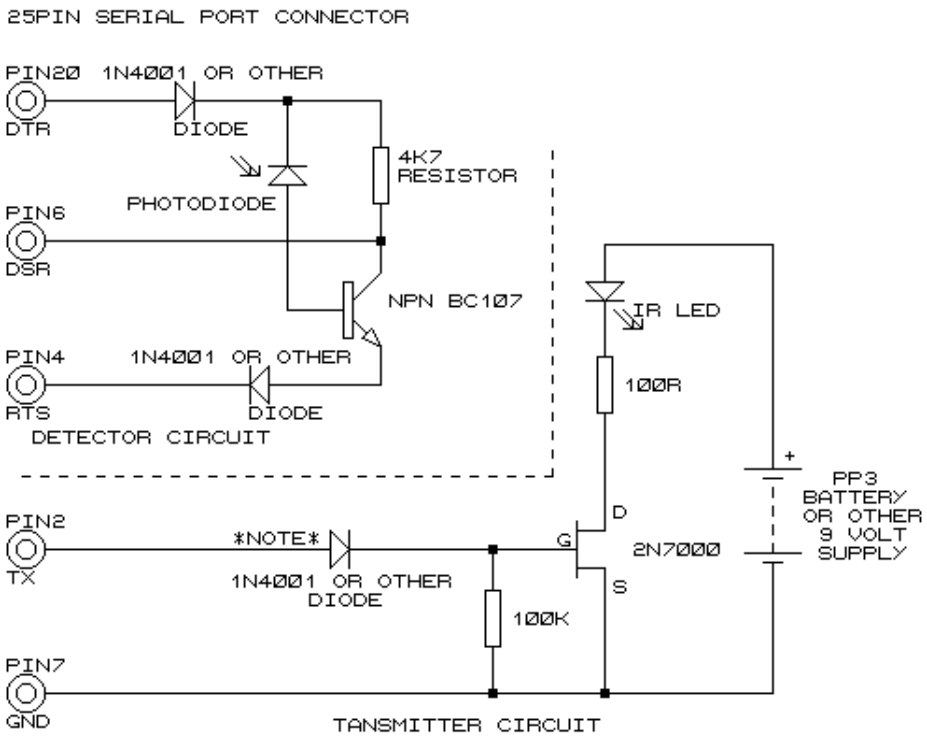
If you are a first-time user of this project, don't worry, just use the latest version of the software.

If you are an existing user of the system, there are several reasons why you might wish to install the latest version. Firstly, real silence rather than silence combs should work with a larger selection of equipment - the original silence combs might somewhat saturate the IR receiver or indeed totally confuse some poorly-filtered equipment (always one of the known drawbacks of the method), so the new method will hopefully improve the IR on/off contrast ratio and thus improve the operating range or make it work properly with your target equipment for the first time.

Secondly, the data stream will be much easier to observe on a proper oscilloscope than it used to be. Thirdly, you can do fun things like transmit a sample from one PC to another and get a close match (I know, you could do it more easily using a floppy or whatever). Fourthly, the new software has some helpful features in DOS-screen mode which make it easier to experiment, diagnose problems and manipulate data sets. Fifthly, the average power consumption of transmission is reduced, prolonging your battery life, saving the planet and so on. The main thing here is I can see no downside. All you need to do is replace the DOS program - the change is invisible to the Windows program, and any samples you already have remain totally compatible. At worst, you get the benefit of the new features; at best you may also get better performance.

Hardware - Section updated 26th August, 2002

Here is a schematic for the hardware I use. This has been updated on 26th August, 2002 to include the diode explained below. If you built yours before this date, and you think your LED might be glowing slightly (in an IR sense) when it should be off, you should consider adding the diode even if everything seems to work fine. It may give you improved performance. You can check to see if you have the problem using a video camera or by measuring the current through the LED in the idle state - don't bother changing it if you don't see the problem. Having said that, it only hit me when I started playing with substitute transistors which were intended to be near-equivalents.



There are really two separate circuits, and it would be ok just to build the receiver section for sampling and use the IRDA port for transmission. The receiver section takes the power it needs from handshake lines (so no external source required in this configuration). The receiver range is

only a couple of cm, so the remote has to be close to work, but there is no particular reason why a proper circuit shouldn't be built which would operate over several metres, and indeed software could be written to let the PC decode the signal properly, though it would be a non-starter as a mouse substitute under Windows or whatever. There are additional notes about the driver transistor (2N7000) on my Furby page (here).

The diode marked *NOTE* between the TX pin and the gate of the FET has been added because I occasionally observed the LED to be slightly on when the TX line was low and the LED should have been off, with the result that the on/off IR contrast ratio was reduced (and consequently maybe the useful transmission range). There are a couple of possible explanations for the LED being on, which I didn't investigate further, opting rather to kill the problem once and for all by including the diode. Shame really, because the tolerance to large positive and negative Vgs values was one of the reasons I chose the FET in the first place. Anyway, the diode effectively prevents the gate from going more negative than the source and the problem is gone. If you suspect that the FET is not turning off fast enough with the 100k pulldown (due to extremely high gate / track capacitance or whatever) you could safely reduce the resistor to 10k to see if that helps, but I have no reason to think the change is needed.

The FAQ page has been updated, and now includes a couple of (untested) suggestions for increasing the output of the transmitter, and the simple change needed in order to replace the FET with a normal NPN transistor.

Software - section updated 26th August, 2002

There are now two versions of the software package available for download, and unless you have a pressing desire to do comparisons between the original version (winsamp version 1.1) and the current one (V1.3), you should ignore REMOTE11.ZIP (37k) and go for REMOTE13.ZIP (39k). The packages contain versions of the same set of 5 files - MANYBUTT.EXE, BUTT1.BUT and BUTT18.BUT are unchanged, WINSAMP.EXE and README.TXT have been updated in the newer (remote13.zip) version. If you are updating your installation from 1.1 to 1.3, only the two changed files need to be replaced (obviously). Just for information, V1.2 was an intermediate version which was never published because I didn't want to have to do two page and docs revisions in quick succession, but you're not missing anything because 1.3 contains all of 1.2 plus extra features. At risk of repeating myself, the upgrade will not require you to do anything to your existing sample files, does not change the command line interface, does not modify the settings of the GUI (manybutt.exe), and only requires you to replace winsamp.exe (and the readme file, if you want to keep things in step - and then probably only if you actually want to read it). Note that the remainder of this section is virtually the same as what was here before, so there is no nead to read it again if you're already familiar with the contents.

The software here is enough to get you going both in dos and Windows95/98, but is not pretty, clever or feature-laden. It goes some way beyond proving the point that the method works, but falls well short of some of the applications which have been written around the parallel-port system. It does everything I need it to do as far as controlling multiple devices and producing platform-independent samples is concerned, so I have drawn the line there for the time being, but I hope I will have provided enough information to allow anyone with an inclination towards

programming to write their own much nicer versions or to port the method to other platforms. I have in mind some particular features I might add to the VB side to address some specific future project requirements, but these are so far off the normal track that there would be little point in trying to build them in to a general app, and anyway they still depend on utilising the same dos-based core program.

How to get started

Click here to download REMOTE13.ZIP (39k) , a .zip file which contains five files - WINSAMP.EXE, MANYBUTT.EXE, README.TXT, BUTT1.BUT and BUTT18.BUT. Create a directory (wherever you want and whatever you want to call it) and put these into it. WINSAMP.EXE is the main dos app, and is all you really need to get started. MANYBUTT.EXE is my VB5 demo for Win95/98 which calls the dos program as required. The *.BUT files are example buttons used by MANYBUTT.EXE. README.TXT is all the explanation and detail I didn't want to put on this page, written in plain text, and I only suggest that it be in the same directory so it doesn't get lost or overwrite any other file of the same name - it overlaps with this file, but you would be expected to have both. This html file (REMOTE.HTM) and README.TXT are expected to evolve as I find problems, develop the idea further and hopefully if and as I get feedback. Create a shortcut to MANYBUTT.EXE on your desktop for convenience. You can also create one for WINSAMP.EXE if you prefer to access it this way rather than from dos prompt, but if you want to see what's going on, it would be best if you make sure (by tweaking properties if necessary) that it runs Full-Screen. Remember that if you want to run it on anything other than COM2, you'll need to edit the command line to WINSAMP.EXE Cx. You can actually look at and fiddle with the software without having any sample/playback hardware. If you already built the IR Thingy from the Furby page, you're half way there hardware-wise, and if you managed to get the Furby stuff working, there's a very good chance this will work too. What I suggest is you save this page or print it out, get the software and read the readme file, maybe have a look at the programs then decide whether or not you can be bothered to build the hardware. If you get that far, then go to dos prompt and run WINSAMP.EXE, using appropriate COM port selection and start trying to sample remotes. If that seems to function, then use MANYBUTT.EXE from the desktop - it's a lot more convenient in the long run. Note that only COM1 - COM4 are supported by WINSAMP.EXE, and that MANYBUTT.EXE does not touch the ports as such. MANYBUTT simply passes a string, which the dos app uses to determine the hardware addresses it hits directly.

Some more detail

The main software (winsamp.exe, 21k) is a dos application which can be run in stand-alone mode or shelled from Windows (tested with 95 and 98). It is not Windows-aware, and consequently takes more processor time when idle than it really needs, but the normal way to use it under Windows is to shell it with various command line options set, which causes it to start, do what you want (eg grab a sample or play back a sample) then exit without any keyboard / mouse / screen activity, so it's normally only loaded briefly, and the program is so small that it takes a negligible amount of time to load. Note that once a sample starts to arrive or starts to be played back, interrupts are suspended until the operation has finished. In the case of sampling, the default sample window is approx 250ms, but this can be pushed up to 500ms using command line options. For playback, this time is the duration of the stored sample, which again can be controlled from

the command line but defaults to around 150ms and cannot exceed 500ms. Be aware that suspending interrupts in order to make sure there are no gaps in the data streams will inevitably result in system ticks being missed (I certainly hope so, this is one of the reasons for doing it), and the result is that your clock will gradually lose time, depending on how much you use the software. Sorry. An easy way round it would be to install one of these programs which syncs your PC to an atomic clock every time you go online if it bothers you - to some, the ability to slow down time might be seen as a bonus. A more messy way round it would be for me to attempt something clever with the real-time-clock hardware following each interrupt suspension, but it doesn't bother me that much to lose a few seconds now and again - when the battery was going flat on my PC I used to lose minutes per day and managed to cope, so this is nothing in comparison.

Because the interrupts are not suspended during sampling until the start of the IR stream from the remote is detected, there is a very small chance that a system tick or similar might happen between detection and the instruction which does the suspending. The chance of this happening is extremely small, but becomes more possible on slower machines, and the ISR would probably take longer to complete if it did happen. The effect of this would be that the recorded width of the first comb would be smaller than it actually was, which would perhaps make the sample invalid on playback for some protocols. I think I've seen this once in all the playing around during development, but of course it might be that pressing the button on the remote causes it to move into range of the receiver just at the critical time or whatever. I would suggest observing a few samples on the sillyscope screen to get a feel for what a good sample looks like, and have a quick look at the sample files to see if any are outstandingly different before burning them onto a million CD Roms or whatever. And of course test the samples.

The most usual way to operate the program in 'real' dos or dos prompt under Windows is to type 'WINSAMP' or WINSAMP Cx where x is the COM port you want to use if the default of COM2 does not suit. A 80*50 text-mode screen is presented (which might not be compatible with _very_ old graphics cards, but if your machine can run Windows it should be ok), and this has lots of things squished onto it. You need to read all the bits of writing on it because the user controls are all hinted at somewhere there. For details of command line options, type WINSAMP ? and a horrible text screen will appear, which might help. A separate README.TXT file is included explaining the command line options and controls in more detail, along with examples of how to make it work from Windows programs. I have also included a simple Windows front-end demonstration written in VB5 which uses the dos app for all the low-level stuff. This is only the actual .exe file - due to bandwidth restrictions I can't put up a full distribution - the .exe is only a few k, but all the support bits take up megs of space, so unless you already have VB5 or have acquired the support files through other software installations you've done over the years, I'm afraid you're stuck, but there is a somewhat less satisfactory but quite functional method for accessing the dos program under Windows using nothing more than program groups and shortcuts which is explained in README.TXT.

It may be possible to rewrite the whole thing as a 'proper' Windows application, but using the dos program makes it relatively simple to achieve the necessary low-level hardware access and timing. Certainly, VB or similar could be used to provide sophisticated sample management (archiving, selection, grouping, exporting, importing etc), but I have only done enough to suit my own needs for the time being.

I hope you like it.

I have added a FAQ page. I welcome feedback on the project, but if you have specific questions or requests, please do me a favour and make sure they haven't already been covered by this web page, the docs or the FAQ. Your question and / or my response might be added in a generic and unattributed sort of way to the next revision of the FAQ.

View my other offerings

RLE - Run Length Encoding

Written by Paul Bourke
August 1995

Source code

Standard compression C source: rle.c
Example code based upon the above: rletest.c

Introduction

Run length encoding is a straightforward way of encoding data so that it takes up less space. It is relies on the string being encoded containing runs of the same character. Consider storing the following short string.

```
abcddddddcbbbbabcdef
```

There are 20 letters above, if each is stored as a single byte that is 20 bytes in all. However, the runs of "d" and "b' above can be stored as two bytes each, the first indicating how many letters in the run. For example, the following run length encoded string takes only 14 bytes.

```
abc6dc4babcdef
```

In general of course it needs to be a bit more sophisticated than the above. For example, there is no way in the above encoding to encode strings with numbers, that is, how would one know whether the number was the length of the run or part of the string content. Also, one would not want to encode runs of length 1 so how does one tell when a run starts and when a literal sequence starts.

The common approach is to use only 7 of the 8 bits to indicate the run length, this is normally interpreted as a signed byte. If the length byte is positive it indicates a replicated run (run of the following byte). If the number is negative then it indicates a literal run, that is, that number of following bytes is copied as is. To illustrate this the following sequence of bytes would encode the example string given above, it requires 17 bytes.

```
-3 a b c 6 d -1 c 4 b 6 a b c d e f
```

While 17 bytes to encode what would take 20 without RLE may not sound like much, but as the frequency and length of the repeating characters increases the compression ratio gets better.

Worst case

Of course RLE will not always result in a compression, consider a string where the next character is different from the current character. Every 127 bytes will require a extra byte to indicate a new literal run length.

Best case

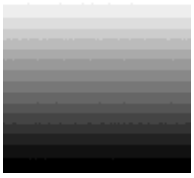
The best case is when 128 identical characters follow each other, this is compressed into 2 bytes instead

of 128 giving a compression ratio of 64.

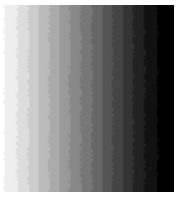
Example

For this reason RLE is most often used to compress black and white or 8 bit indexed colour images where long runs are likely. RLE compression is therefore what was used for the original low colour images expected for the Macintosh PICT file format. RLE is not generally used for high colour images such as photographs where in general each pixel will vary from the last.

The following 3 images illustrate the different extremes, the first image contains runs along each row and will compress well. The second image is the same as the first but rotated 90 degrees so there are no runs giving worse case and a larger file. This suggests a natural extension to RLE for images, that is, one compresses vertically and horizontally and uses the best, the flag indicating which one is used is stored in the image header. The last case is the best scenario where the whole image is a constant value.



Original size: 10000 bytes
Compressed size: 5713 bytes
Ratio: 1.75



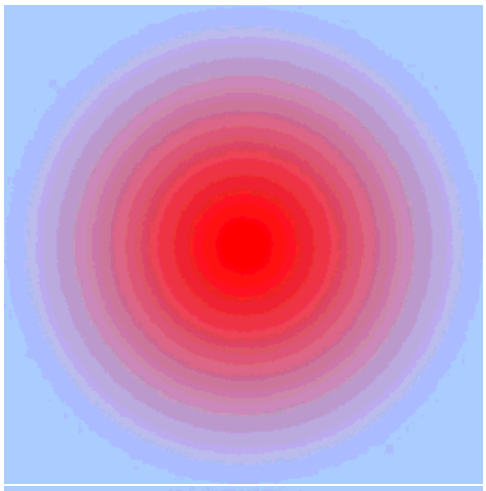
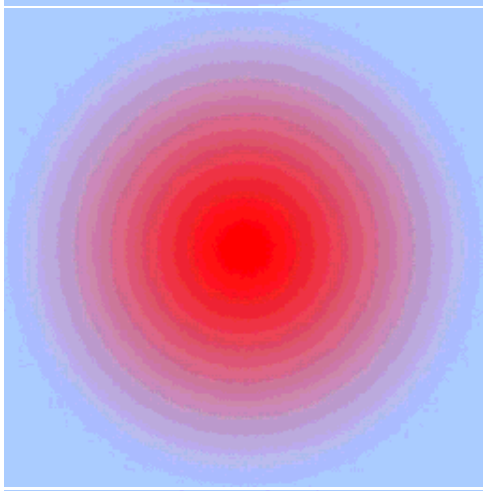
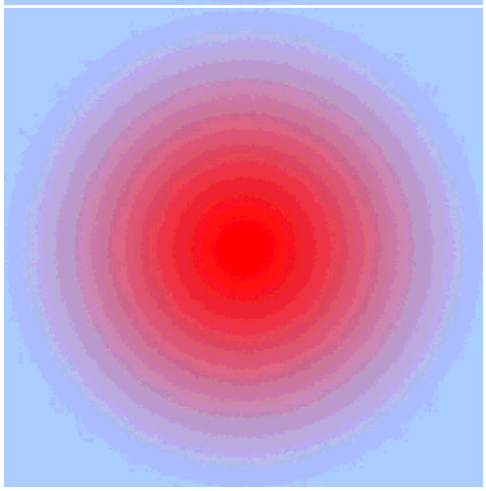
Original size: 10000 bytes
Compressed size: 10100
Ratio: 0.99

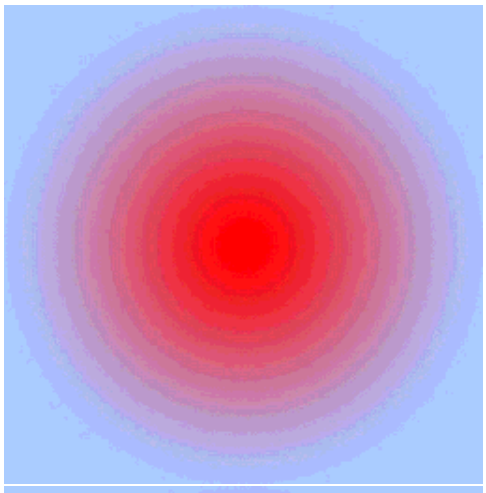
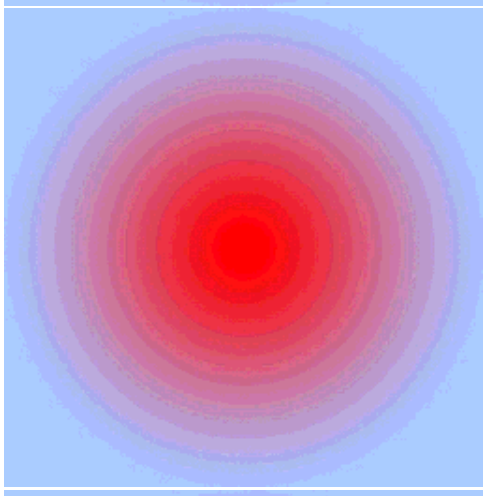
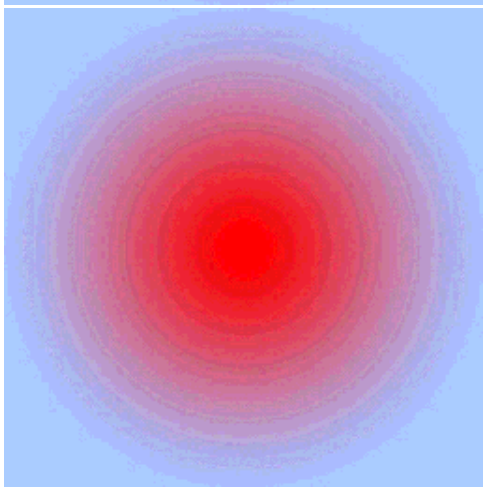
Original size: 10000 bytes
Compressed size: 200
Ratio: 50

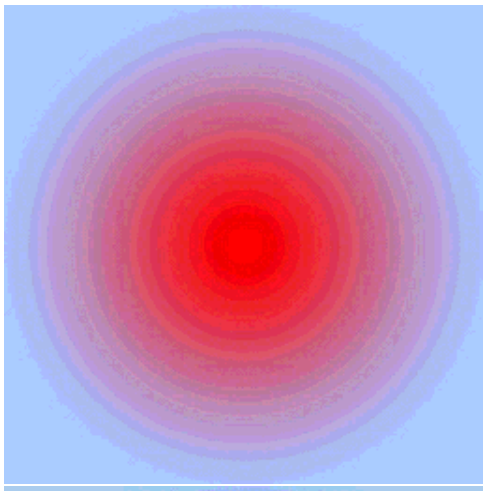
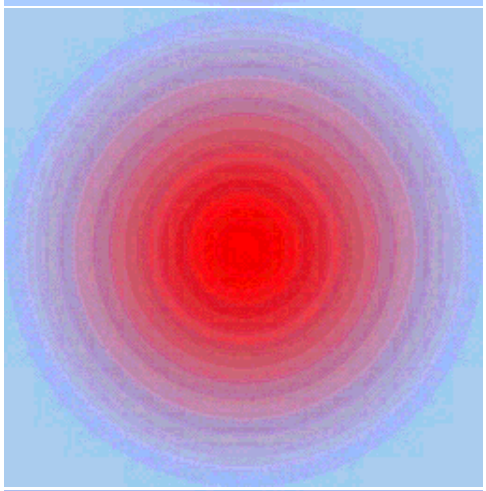
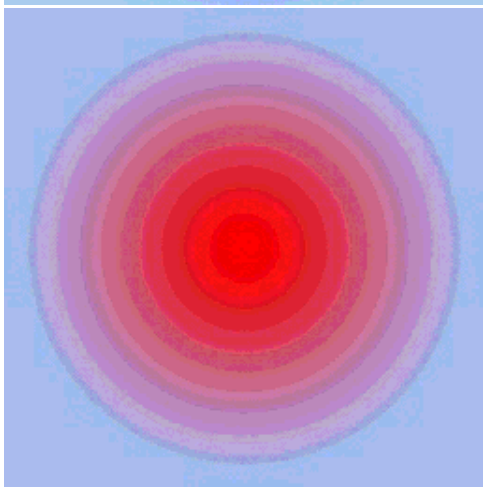
Image comparison

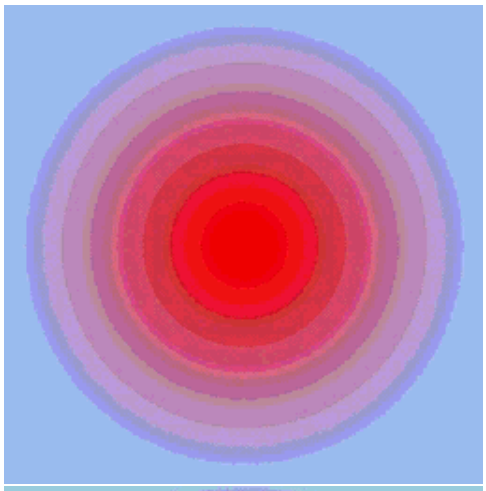
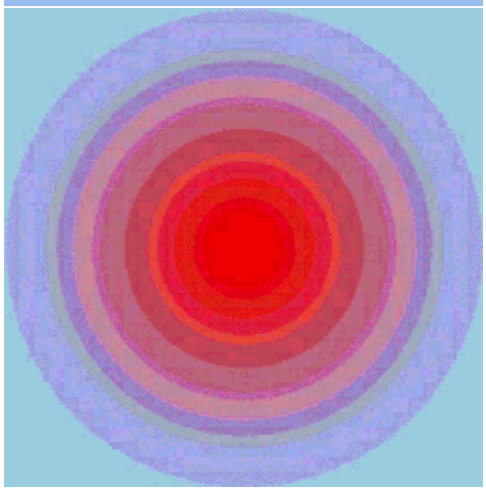
Run length encoding is used within a number of image formats, for example PNG, TIFF, and TGA. While RLE is normally used as a lossless compression, it can be assisted (to create small files) by quantising the rgb values thus increasing the chances of runs of the same colour. There are two ways one can run length encode the pixels, the first as used in the TGA format is to look for runs of all three components, the other is to compress each colour plane seperately. The second approach normally gives smaller files. Below is a table for two different images along with the file size for the image quantised to different levels and saved in rgb order or planar order.

Image details and quantisation level	Image example	RLE on RGB (KBytes)	RLE on Planes (KBytes)

Uncompressed		197	197
1 (none)		151	141
2		135	110

4		101	70
6		81	49
8		64	36

12		46	24.5
16		35	18.5
20		28	14.5

26		22	11.5
32		17.5	9.5

The above example was chosen because it doesn't have long runs of equal colour and because any banding due to quantisation should be obvious to spot. This occurs somewhere between 4 and 8 depending on how fussy one is. Note the planar compression works much better than the rgb based compression.

Image details and quantisation level	Image example	RLE on RGB (KBytes)	RLE on Planes (KBytes)

Uncompressed		197	197
1 (none)		195	190
2		188	173

4		163	140
6		142	119
8		127	106

12		105	88
16		86.5	74
20		92	73.5

26		74	60.5
32		60	50.5

Unlike the first example, because each of the colour layers in this images are "busy" the difference between rgb and planar RLE compression is not so marked. Note that the visual artifacts that occur to so on the wall where there is a smooth and subtle shade variation, even at the highest quantisation level the artifacts on the vase are hard to pick.

RS232 Data Interface

a Tutorial on Data Interface and cables

RS-232 is simple, universal, well understood and supported but it has some serious shortcomings as a data interface. The standards to 256kbps or less and line lengths of 15M (50 ft) or less but today we see high speed ports on our home PC running very high speeds and with high quality cable maxim distance has increased greatly. The rule of thumb for the length a data cable depends on speed of the data, quality of the cable.

---

a Tutorial

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumors for its early demise, has remained widely used through the industry.

Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol.

The RS-232 interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption when a short cable connects the DTE to the DCE, but with longer lines and connections between devices that may be on different electrical busses with different grounds, this may not be true.

RS232 data is bi-polar.... +3 TO +12 volts indicates an "ON or 0-state (SPACE) condition" while A -3 to -12 volts indicates an "OFF" 1-state (MARK) condition.... Modern computer equipment ignores the negative level and accepts a zero voltage level as the "OFF" state. In fact, the "ON" state may be achieved with lesser positive potential. This means circuits powered by 5 VDC are capable of driving RS232 circuits directly, however, the overall range that the RS232 signal may be transmitted/received may be dramatically reduced.

The output signal level usually swings between +12V and -12V. The "dead area" between +3v and -3v is designed to absorb line noise. In the various RS-232-like definitions this dead area may vary. For instance, the definition for V.10 has a dead area from +0.3v to -0.3v. Many receivers designed for RS-232 are sensitive to differentials of 1v or less.

This can cause problems when using pin powered widgets - line drivers, converters, modems etc.

These type of units need enough voltage & current to power them self's up. Typical URART (the RS-232 I/O chip) allows up to 50ma per output pin - so if the device needs 70ma to run we would need to use at least 2 pins for power. Some devices are very efficient and only require one pin (some times the Transmit or DTR pin) to be high - in the "SPACE" state while idle.

An RS-232 port can supply only limited power to another device. The number of output lines, the type of interface driver IC, and the state of the output lines are important considerations.

The types of driver ICs used in serial ports can be divided into three general categories:

- **Drivers which require plus (+) and minus (-) voltage power supplies such as the 1488 series of interface integrated circuits. (Most desktop and tower PCs use this type of driver.)**
- **Low power drivers which require one +5 volt power supply. This type of driver has an internal charge pump for voltage conversion. (Many industrial microprocessor controls use this type of driver.)**
- **Low voltage (3.3 v) and low power drivers which meet the EIA-562 Standard. (Used on notebooks and laptops.)**

Data is transmitted and received on pins 2 and 3 respectively. Data Set Ready (DSR) is an indication from the Data Set (i.e., the modem or DSU/CSU) that it is on. Similarly, DTR indicates to the Data Set that the DTE is on. Data Carrier Detect (DCD) indicates that a good carrier is being received from the remote modem.

Pins 4 RTS (Request To Send - from the transmitting computer) and 5 CTS (Clear To Send - from the Data set) are used to control. In most Asynchronous situations, RTS and CTS are constantly on throughout the communication session. However where the DTE is connected to a multipoint line, RTS is used to turn carrier on the modem on and off. On a multipoint line, it's imperative that only one station is transmitting at a time (because they share the return phone pair). When a station wants to transmit, it raises RTS. The modem turns on carrier, typically waits a few milliseconds for carrier to stabilize, and then raises CTS. The DTE transmits when it sees CTS up. When the station has finished its transmission, it drops RTS and the modem drops CTS and carrier together.

Clock signals (pins 15, 17, & 24) are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

Note: Transmit and receive leads (2 or 3) can be reversed depending on the use of the equipment - DCE Data Communications Equipment or a DTE Data Terminal Equipment.

Glossary of Abbreviations etc.

CTS	**Clear To Send [DCE --> DTE]**
DCD	**Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
DCE	**Data Communications Equipment eg. modem**
DSR	**Data Set Ready [DCE --> DTE]**
DSRS	**Data Signal Rate Selector [DCE --> DTE] (Not commonly used)**
DTE	**Data Terminal Equipment eg. computer, printer**
DTR	**Data Terminal Ready [DTE --> DCE]**
FG	**Frame Ground (screen or chassis)**
NC	**No Connection**
RCk	**Receiver (external) Clock input**
RI	**Ring Indicator (ringing tone detected)**
RTS	**Ready To Send [DTE --> DCE]**
RxD	**Received Data [DCE --> DTE]**
SG	**Signal Ground**
SCTS	**Secondary Clear To Send [DCE --> DTE]**
SDCD	**Secondary Data Carrier Detected (Tone from a modem) [DCE --> DTE]**
SRTS	**Secondary Ready To Send [DTE --> DCE]**
SRxD	**Secondary Received Data [DCE --> DTE]**
STxD	**Secondary Transmitted Data [DTE --> DTE]**
TxD	**Transmitted Data [DTE --> DTE]**

Is Your Interface a DTE or a DCE?

Find out by following these steps: The point of reference for all signals is the terminal (or PC).

1) Measure the DC voltages between (DB25) pins 2 & 7 and between pins 3 & 7. Be sure the black lead is connected to pin 7 (Signal Ground) and the red lead to whichever pin you are measuring.

2) If the voltage on pin 2 (TD) is more negative than -3 Volts, then it is a DTE, otherwise it should be near zero volts.

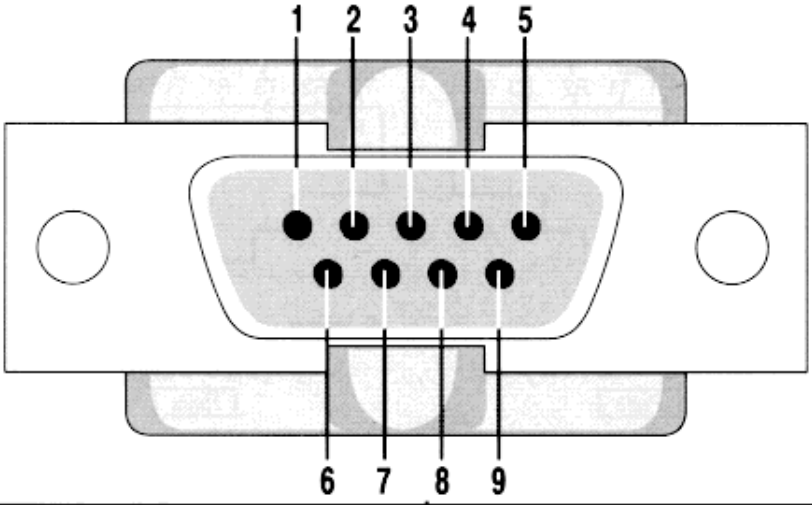
3) If the voltage on pin 3 (RD) is more negative than -3 Volts, then it is a DCE.

4) If both pins 2 & 3 have a voltage of at least 3 volts, then either you are measuring incorrectly, or your device is not a standard EIA-232 device. Call technical support.

5) In general, a DTE provides a voltage on TD, RTS, & DTR, whereas a DCE provides voltage on RD, CTS, DSR, & CD.

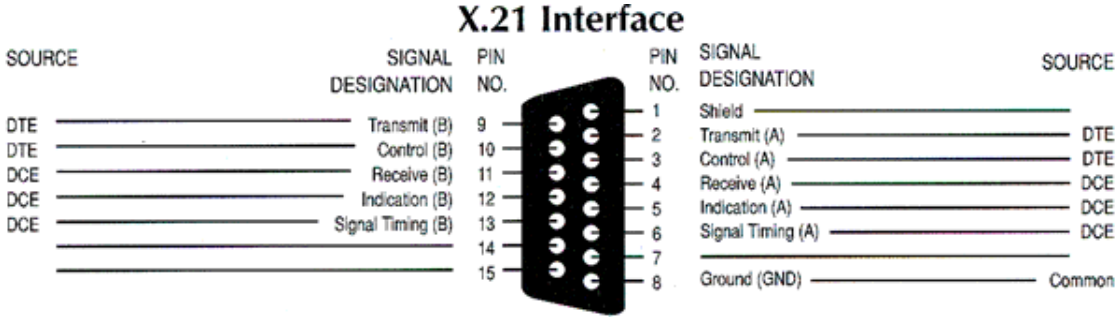
PC Com Port - EIA-574
RS-232/V.24 pin out on a DB-9 pin

used for Asynchronous Data



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

X.21 interface on a DB 15 connector



also see X.21 write up
also see end of page for more info

X.21

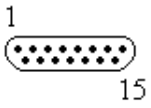
General

Voltages:	+/- 0.3Vdc
Speeds:	Max. 100Kbps (X.26)
	Max. 10Mbps (X.27)

The X.21 interface was recommended by the CCITT in 1976. It is defined as a digital signalling interface between customers (DTE) equipment and carrier's equipment (DCE). And thus primarally used for telecom equipment.

All signals are balanced. Meaning there is always a pair (+/-) for each signal, like used in RS422. The X.21 signals are the same as RS422, so please refer to RS422 for the exact details.

Pinning according to ISO 4903

	
Sub-D15 Male	Sub-D15 Female

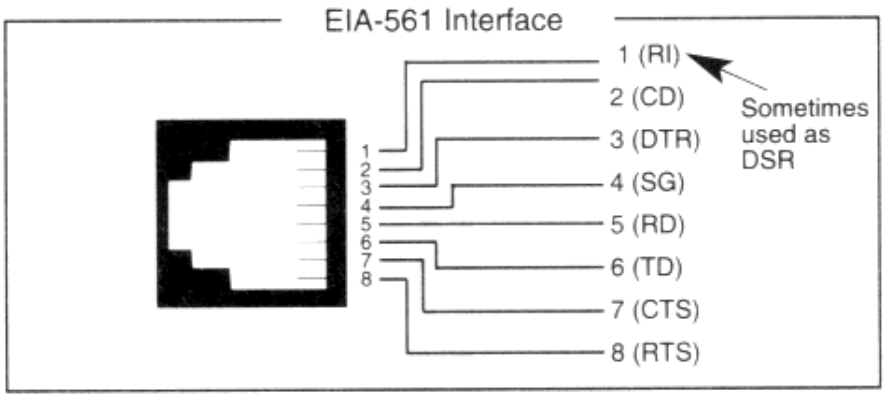
Pin	Signal	abbr.	DTE	DCE
1	Shield		-	-
2	Transmit (A)		Out	In
3	Control (A)		Out	In
4	Receive (A)		In	Out
5	Indication (A)		In	Out
6	Signal Timing (A)		In	Out
7	Unassigned			
8	Ground		-	-
9	Transmit (B)		Out	In
10	Control (B)		Out	In
11	Receive (B)		In	Out
12	Indication (B)		In	Out
13	Signal Timing (B)		In	Out
14	Unassigned			
15	Unassigned			

Functional Description
As can be seen from the pinning specifications, the Signal Element Timing (clock) is provided by the DCE. This means that your provider (local telco office) is responsible for the correct clocking and that X.21 is a synchronous interface. Hardware handshaking is done by the Control and Indication lines. The Control is used by the DTE and the Indication is the DCE one.

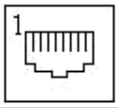
Cross-cable pinning

X.21 Cross Cable	
X.21	**X.21**
1	1
2	4
3	5
4	2
5	3
6	7
7	6
8	8
9	11
10	12
11	9
12	10
13	14
14	13
15	

EIA-561 defines RS-232 on a modular connector. (For nonsynchronous applications only, since it does not provide for the synchronous clocking signals.)



RS232D uses RJ45 type connectors (similar to telephone connectors)

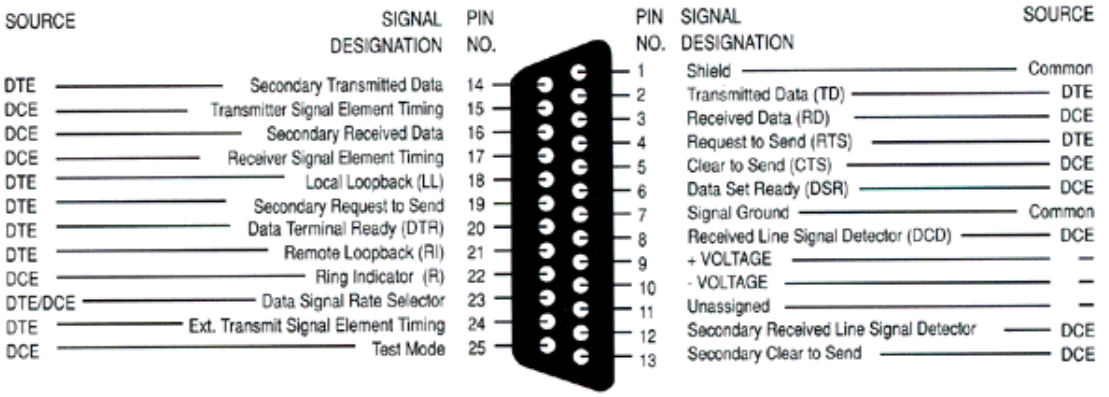
	Male	Female
RJ45		

Pin No.	Signal Description	Abbr.	DTE	DCE
1	DCE Ready, Ring Indicator	DSR/RI	←	→
2	Received Line Signal Detector	DCD	←	→
3	DTE Ready	DTR	→	←
4	Signal Ground	SG		
5	Received Data	RxD	←	→
6	Transmitted Data	TxD	→	←
7	Clear To Send	CTS	←	→
8	Request To Send	RTS	→	←

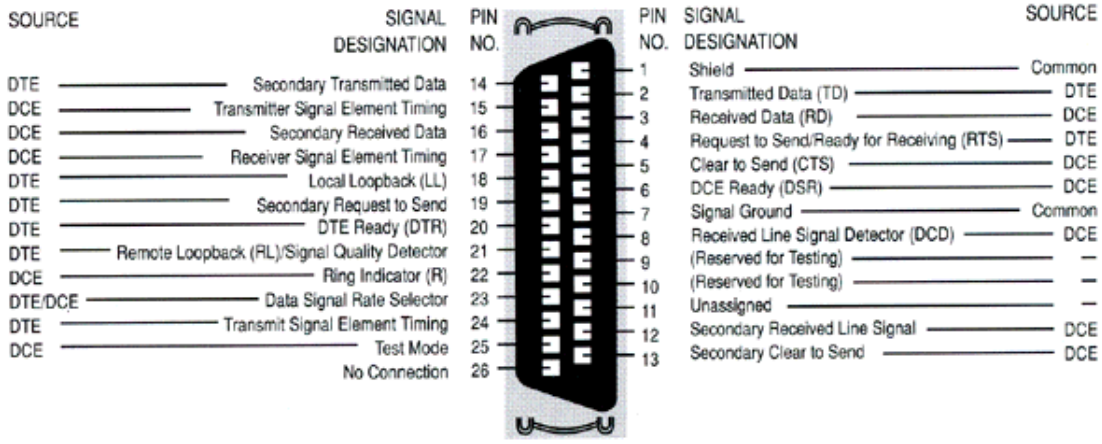
This is a standard 9 to 25 pin cable layout for async data on a PC AT serial cable

Description	Signal	9-pin DTE	25-pin DCE	Source DTE or DCE
Carrier Detect	CD	1	8	from Modem
Receive Data	RD	2	3	from Modem
Transmit Data	TD	3	2	from Terminal/Computer
Data Terminal Ready	DTR	4	20	from Terminal/Computer
Signal Ground	SG	5	7	from Modem
Data Set Ready	DSR	6	6	from Modem
Request to Send	RTS	7	4	from Terminal/Computer
Clear to Send	CTS	8	5	from Modem
Ring Indicator	RI	9	22	from Modem

V.24/RS-232 Interface



V.24/RS-232E ALT A Connector



---

25 pin D-shell connector RS232

commonly used for Async. data

PIN SIGNAL DESCRIPTION

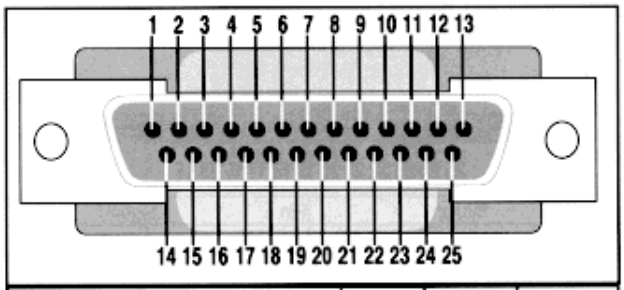
1 PGND Protective Ground
2 TXD Transmit Data
3 RXD Receive Data
4 RTS Ready To Send
5 CTS Clear To Send

6 DSR Data Set Ready
7 SG Signal Ground
8 CD Carrier Detect
20 DTR Data Terminal Ready
22 RI Ring Indicator

Some applications require more
than a simple async. configurat

RS-232 Interface

RS-232 (EIA Std.) applicable to the 25 pin interconnection of Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) using serial binary data

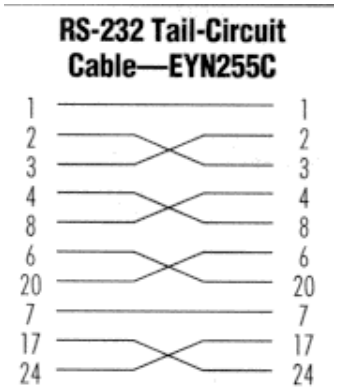


Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CI	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

Pins used for Synchronous data

jump to Other Connector pages

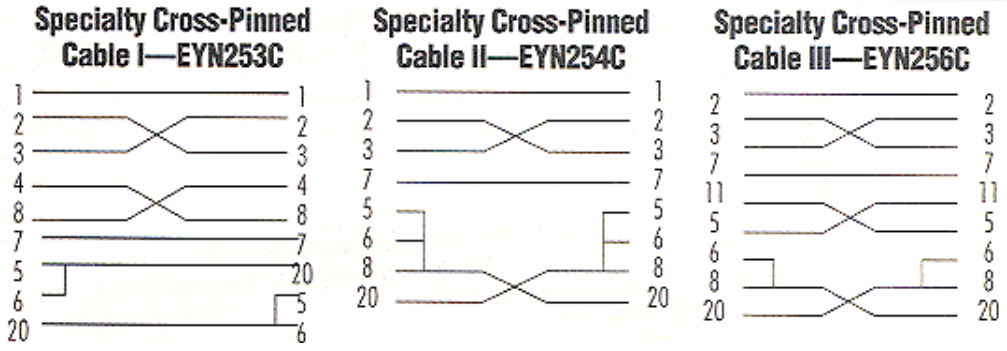
RS232 (25 pin) Tail Circuit Cable



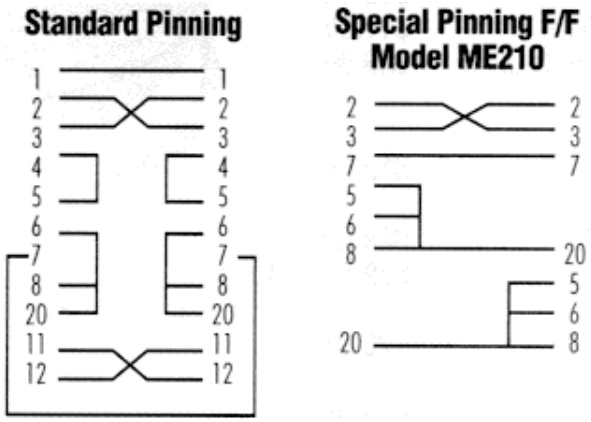
Null Modem cable diagrams

- Nullmodem (9p to 9p)
- Nullmodem (9p to 25p)
- Nullmodem (25p to 25p)

Cross Pinned cables for Async data.



Pin out for local Async Data transfer



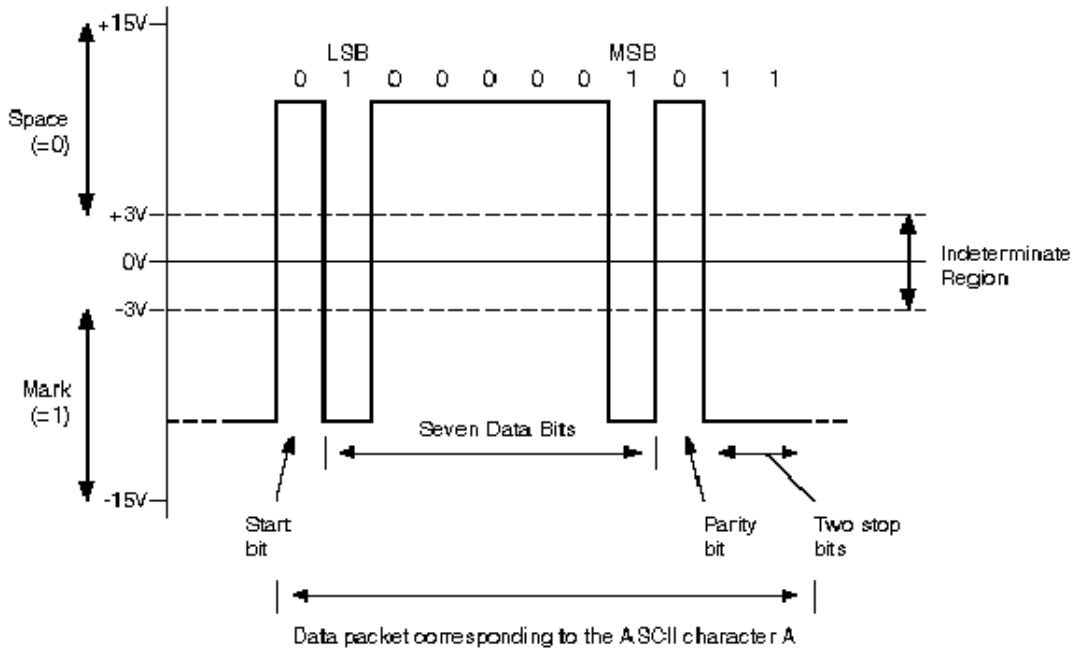
Loopback plugs:

- Serial Port Loopback (9p)
- Serial Port Loopback (25p)

RS-232 Specs.

SPECIFICATIONS		RS232	RS423
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR
Maximum Cable Length		50 FT.	4000 FT.
Maximum Data Rate		20kb/s	100kb/s
Maximum Driver Output Voltage		+/-25V	+/-6V
Driver Output Signal Level (Loaded Min.)	**Loaded**	+/-5V to +/-15V	+/-3.6V
Driver Output Signal Level (Unloaded Max)	**Unloaded**	+/-25V	+/-6V
Driver Load Impedance (Ohms)		3k to 7k	>=450
Max. Driver Current in High Z State	**Power On**	N/A	N/A
Max. Driver Current in High Z State	**Power Off**	+/-6mA @ +/-2v	+/-100uA
Slew Rate (Max.)		30V/uS	Adjustable
Receiver Input Voltage Range		+/-15V	+/-12V
Receiver Input Sensitivity		+/-3V	+/-200mV
Receiver Input Resistance (Ohms)		3k to 7k	4k min.

One byte of async data



Cabling considerations - you should use cabling made for RS-232 data but I have seen low speed data go over 250’ on 2 pair phone cable. Level 5 cable can also be used but for best distance use a low capacitance data grade cable.

The standard maxim length is 50’ but if data is async you can increase that distance to as much as 500’ with a good grade of cable.

The RS-232 signal on a single cable is impossible to screen effectively for noise. By screening the entire cable we can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the different lines introduces serious crosstalk (this especially true on synchronous data - because of the clock lines) until a point is reached where the data itself is unreadable. Signal Crosstalk can be reduced by using low capacitance cable and shielding each pair

Using a high grade cable (individually shield low capacitance pairs) the distance can be extended to 4000’

At higher frequencies a new problem comes to light. The high frequency component of the data signal is lost as the cable gets longer resulting in a rounded, rather than square wave signal.

The maxim distance will depend on the speed and noise level around the cable run.

On longer runs a line driver is needed. This is a simple modem used to increase the maxim distance you can run RS-232 data.

Making sense of the specifications

Selecting data cable isn't difficult, but often gets lost in the shuffle of larger system issues. Care should be taken. however, because intermittent problems caused by marginal cable can be very difficult to troubleshoot.

Beyond the obvious traits such as number of conductors and wire gauge, cable specifications include a handful of less intuitive terms.

Characteristic Impedance (Ohms): A value based on the inherent conductance, resistance, capacitance and inductance of a cable that represents the impedance of an infinitely long cable. When the cable is out to any length and terminated with this Characteristic Impedance, measurements of the cable will be identical to values obtained from the infinite length cable. That is to say that the termination of the cable with this impedance gives the cable the appearance of being infinite length, allowing no reflections of the transmitted signal. If termination is required in a system, the termination impedance value should match the Characteristic Impedance of the cable.

Shunt Capacitance (pF/ft): The amount of equivalent capacitive load of the cable, typically listed in a per foot basis One of the factors limiting total cable length is the capacitive load. Systems with long lengths benefits from using low capacitance cable.

Propagation velocity (% of c): The speed at which an electrical signal travels in the cable. The value given typically must be multiplied by the speed of light (c) to obtain units of meters per second. For example, a cable that lists a propagation velocity of 78% gives a velocity of 0.78 X 300 X 106 - 234 X 106 meters per second.

Plenum cable

Plenum rated cable is fire resistant and less toxic when burning than non-plenum rated cable. Check building and fire codes for requirements. Plenum cable is generally more expensive due to the sheathing material used.

The specification recommends 24AWG twisted pair cable with a shunt capacitance of 16 pF per foot and 100 ohm characteristic impedance.

It can be difficult to qualify whether shielding is required in a particular system or not, until problems arise. We recommend erring on the safe side and using shielded cable. Shielded cable is

only slightly more expensive than unshielded.

There are many cables available meeting the recommendations of RS-422 and RS-485, made specifically for that application. Another choice is the same cable commonly used in the Twisted pair Ethernet cabling. This cable, commonly referred to as Category 5 cable, is defined by the ElA/TIA/ANSI 568 specification The extremely high volume of Category 5 cable used makes it widely available and very inexpensive, often less than half the price of specialty RS422/485 cabling. The cable has a maximum capacitance of 17 pF/ft (14.5 pF typical) and characteristic impedance of 100 ohms.

Category 5 cable is available as shielded twisted pair (STP) as well as unshielded twisted pair (UTP) and generally exceeds the recommendations making it an excellent choice for RS232 systems.

RS232 - V.24/V.28 - IS2110 - X.20 bis (for Async) - X.21 bis (for Sync)

General

In this document the term RS232 will be used when refered to this serial interface. The description of RS232 is an EIA/TIA norm and is identical to CCITT V.24/V.28, X.20bis/X.21bis and ISO IS2110. The only difference is that CCITT has split the interface into its electrical description (V.28) and a mechanical part (V.24) or Asynchronous (X.20 bis) and Synchronous (X.21 bis) where the EIA/TIA describes everything under RS232.

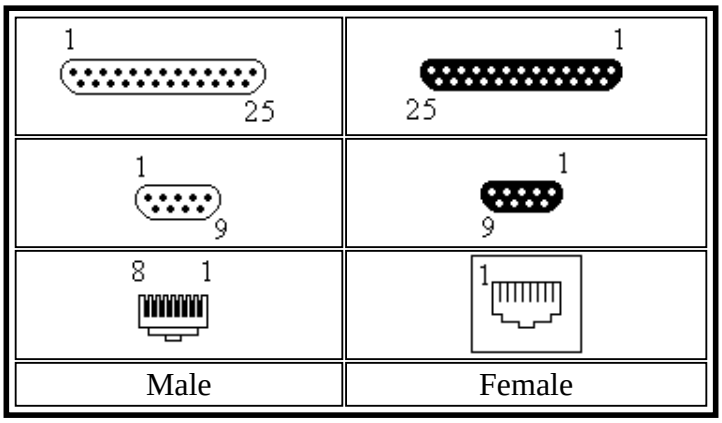
As said before RS232 is a serial interface. It can be found in many different applications where the most common ones are modems and Personal Computers. All pinning specifications are writen for the DTE side.

All DTE-DCE cables are straight through meaning the pins are connected one on one. DTE-DTE and DCE-DCE cables are cross cables. To make a destiction between all different types of cables we have to use a naming convention.
DTE - DCE: Straight Cable
DTE - DTE: Null-Modem Cable
DCE - DCE: Tail Circuit Cable

Interface Mechanical

RS232 can be found on different connectors. There are special specifications for this. The CCITT only defines a Sub-D 25 pins version where the EIA/TIA has two versions RS232C and RS232D which are resp. on a Sub-D25 and a RJ45. Next to this IBM has added a Sub-D 9 version which is found an almost

all Personal Computers and is described in TIA 457.



Pinning

RS232-C	Description	Circuit EIA	Circuit CCITT	RJ45	TIA 457
1	Shield Ground	AA			
7	Signal Ground	AB	102	4	5
2	Transmitted Data	BA	103	6	3
3	Received Data	BB	104	5	2
4	Request To Send	CA	105	8	7
5	Clear To Send	CB	106	7	8
6	DCE Ready	CC	107	1	6
20	DTE Ready	CD	108.2	3	4
22	Ring Indicator	CE	125	1	9
8	Received Line Signal Detector	CF	109	2	1
23	Data Signal Rate Select (DTE/DCE Source>	CH/CI	111/112		
24	Transmit Signal Element Timing (DTE Source)	DA	113		
15	Transmitter Signal Element Timing (DCE Source)	DB	114		
17	Receiver Signal Element Timing (DCE Source)	DD	115		
18	Local Loopback / Quality Detector	LL	141		
21	Remote Loopback	RL/CG	140/110		
14	Secondary Transmitted Data	SBA	118		
16	Secondary Received Data	SBB	119		
19	Secondary Request To Send	SCA	120		
13	Secondary Clear To Send	SCB	121		
12	Secondary Received Line Signal Detector/ Data signal Rate Select (DCE Source)	SCF/CI	122/112		
25	Test Mode	TM	142		
9	Reserved for Testing				
10	Reserved for Testing				
11	Unassigned				

Interface Electrical

All signals are measured in reference to a common ground, which is called the signal ground (AB). A positive voltage between 3 and 15 Vdc represents a logical 0 and a negative voltage between 3 and 15 Vdc represents a logical 1.
This switching between positive and negative is called bipolar. The zero state is not defined in RS232

and is considered a fault condition (this happens when a device is turned off).
According to the above a maximum distance of 50 ft or 15 m. can be reached at a maximum speed of 20k bps. This is according to the official specifications, the distance can be exceeded with the use of Line Drivers.

Functional description

Description	Circuit	Function
Shield Ground	AA	Also known as protective ground. This is the chassis ground connection between DTE and DCE.
Signal Ground	AB	The reference ground between a DTE and a DCE. Has the value 0 Vdc.
Transmitted Data	BA	Data send by the DTE.
Received Data	BB	Data received by the DTE.
Request To Send	CA	Originated by the DTE to initiate transmission by the DCE.
Clear To Send	CB	Send by the DCE as a reply on the RTS after a delay in ms, which gives the DCEs enough time to energize their circuits and synchronize on basic modulation patterns.
DCE Ready	CC	Known as DSR. Originated by the DCE indicating that it is basically operating (power on, and in functional mode).
DTE Ready	CD	Known as DTR. Originated by the DTE to instruct the DCE to setup a connection. Actually it means that the DTE is up and running and ready to communicate.
Ring Indicator	CE	A signal from the DCE to the DTE that there is an incomming call (telephone is ringing). Only used on switched circuit connections.
Received Line Signal Detector	CF	Known as DCD. A signal send from DCE to its DTE to indicate that it has received a basic carrier signal from a (remote) DCE.
Data Signal Rate Select (DTE/DCE Source>	CH/CI	A control signal that can be used to change the transmission speed.
Transmit Signal Element Timing (DTE Source)	DA	Timing signals used by the DTE for transmission, where the clock is originated by the DTE and the DCE is the slave.
Transmitter Signal Element Timing (DCE Source)	DB	Timing signals used by the DTE for transmission.
Receiver Signal Element Timing (DCE Source)	DD	Timing signals used by the DTE when receiving data.
Local Loopback / Quality Detector	LL	
Remote Loopback	RL/CG	Originated by the DCE that changes state when the analog signal received from the (remote) DCE becomes marginal.

Test Mode	TM	
Reserved for Testing		

The secondary signals are used on some DCE’s. Those units have the possibility to transmit and/or receive on a secondary channel. Those secondary channels are mostly of a lower speed than the normal ones and are mainly used for administrative functions.

Cable pinning
Here are some cable pinning that might be useful. Not all applications are covered, it is just a help:

Straight DB25 Cable

Pin	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25

DB25 Null- modem cable (Async)

Pin	Pin
1	1
2	3
3	2
4	5
5	4
6, 8	20
7	7
20	6, 8

</

DB9 Null- modem cable

1,6	4
2	3
3	2
4	1,6
5	5
7	8
8	7

DB25 Tail- circuit cable (Sync)

Pin	Pin
1	1
2	3
3	2
4	8
6	20
7	7
8	4
17	24
20	6
24	17

DB25 to DB9 DTE - DCE cable

jump to **related fiber cable pages**

---

jump to The Belden Cable Company's cable selection tutorial pages

jump to Data Communication by CAMI Research good write up

jump to RS-232 by CAMI Research good write up

jump to Interfacing the Serial / RS232 Port good write up
(in-depth very technical)

jump to Data Modems for phone lines

jump to Data Modems for fiber optics

jump to Interface converters

ARC Electronics ...
800-926-0226
Home Page

arc@arcelect.com

PC serial port buffer

Summary of circuit features

- Brief description of operation: Buffer to run RS-232 data to longer distanced as normally
- Circuit protection: No special protection circuits used
- Circuit complexity: Very simple two transistor buffer circuit
- Circuit performance: Worked nicely in one special application, doubled the line throughput
- Availability of components: Widely available components at the time when the circuit was built
- Design testing: Circuit was in constatant use by my friend for over a year
- Applications: Maximizing RS-232 line throughput on long cable runs
- Power supply: +-12V DC power supply 80 mA
- Estimated component cost: Few dollars
- Safety considerations: No special safety considerations

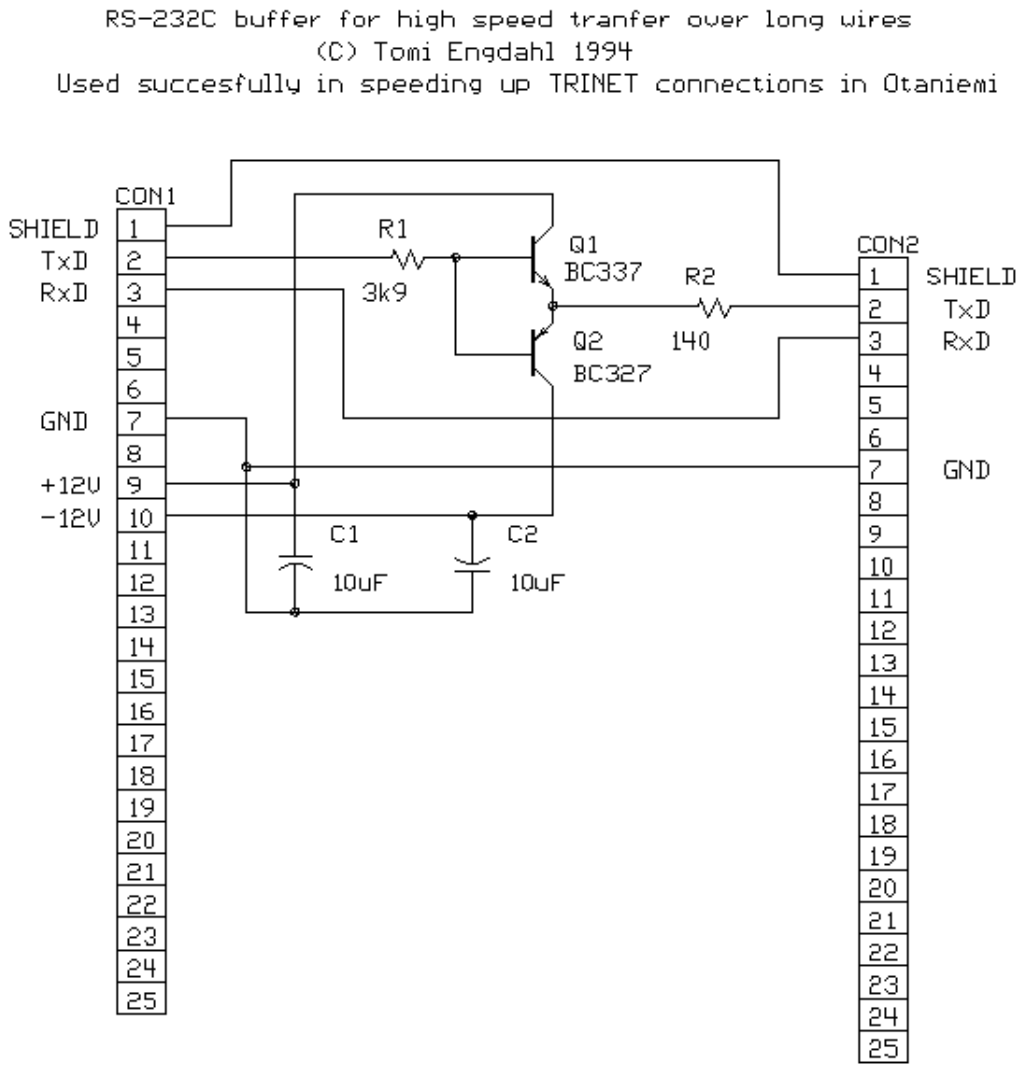
Circuit description

This is a simple serial port buffer circuit I designed for a friend to speed up his SLIP connection in campus computer network "TRINET" of Helsinki University of Technology. The problem in the network was that the RS232 commections from rooms to terminal server were long and made of bad quality wiring.

The circuit is a simple buffer which adds more driving capacity to PC serial port for the signal to go succesfully from PC computer to terminal server (other direction had no problems). The computer is connected to connector CON1 and the buffered output is available ar CON2. With this circuit the speed of RS232 connection to termial server could be succesfully raised for 9600 bps to 38400 bps.

The circuit is basically a two transistor buffer consistong of transistors Q1 and Q2 which can drive up to 1A current pulses, but the maximum putput current of the circuit is limited by resistor R2. Value R2 was experimentally selected by testing resistor values in range of 22 ohm to 270 ohm and value 140 ohm gave best results (it provides quite good impedance matching to cable used). It is a good idea to use at least 1W resistor in place of R2 to make sure that it does not overheat in output short circuit situation (RS232 devices must withstand that to meet the standard).

The circuit was designed to be a compact box which is powered through D25 connector as some commercial RS232 buffer circuits. The idea is to feed the power to the buffer unit through serial port voltage test pins 9 and 10. The power was taken from an external power supply (cheap universal wall transformer) and wired to the D25 connector by modifying the cable connected between computer and the buffer circuit. The circuit in this configuration takes maximally continuous current of about 100 mA.



Tomi Engdahl <then@delta.hut.fi>

Interfacing the Serial / RS232 Port

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

So what are the advantages of using serial data transfer rather than parallel?

1. Serial Cables can be longer than Parallel cables. The serial port transmits a '1' as -3 to -25 volts and a '0' as +3 to +25 volts where as a parallel port transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5 Volts. Therefore cable loss is not going to be as much of a problem for serial cables than they are for parallel.
2. You don't need as many wires than parallel transmission. If your device needs to be mounted a far distance away from the computer then 3 core cable (Null Modem Configuration) is going to be a lot cheaper that running 19 or 25 core cable. However you must take into account the cost of the interfacing at each end.
3. Infra Red devices have proven quite popular recently. You may of seen many electronic diaries and palmtop computers which have infra red capabilities build in. However could you imagine transmitting 8 bits of data at the one time across the room and being able to (from the devices point of view) decipher which bits are which? Therefore serial transmission is used where one bit is sent at a time. IrDA-1 (The first infra red specifications) was capable of 115.2k baud and was interfaced into a UART. The pulse length however was cut down to 3/16th of a RS232 bit length to conserve power considering these devices are mainly used on diaries, laptops and palmtops.
4. Microcontroller's have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

Table of Contents

Part One : Hardware (PC's)

Hardware Properties

Devices which use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) Data Communications Equipment are devices such as your modem, TA adapter, plotter etc while Data Terminal Equipment is your Computer or Terminal.

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as -

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.
3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!)

Above is no where near a complete list of the EIA standard. Line Capacitance, Maximum Baud Rates etc are also included. For more information please consult the EIA RS232-C standard. It is interesting to note however, that the RS232C standard specifies a maximum baud rate of 20,000 BPS!, which is rather slow by today's standards. A new standard, RS-232D has been recently released.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. Below is a table of pin connections for the 9 pin and 25 pin D-Type connectors.

Serial Pinouts (D25 and D9 Connectors)

D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

Table 1 : D Type 9 Pin and D Type 25 Pin Connectors

Pin Functions

Abbreviation	Full Name	Function
TD	Transmit Data	Serial Data Output (TXD)
RD	Receive Data	Serial Data Input (RXD)
CTS	Clear to Send	This line indicates that the Modem is ready to exchange data.
DCD	Data Carrier Detect	When the modem detects a "Carrier" from the modem at the other end of the phone line, this Line becomes active.
DSR	Data Set Ready	This tells the UART that the modem is ready to establish a link.
DTR	Data Terminal Ready	This is the opposite to DSR. This tells the Modem that the UART is ready to link.
RTS	Request To Send	This line informs the Modem that the UART is ready to exchange data.
RI	Ring Indicator	Goes active when modem detects a ringing signal from the PSTN.

Null Modems

A Null Modem is used to connect two DTE's together. This is commonly used as a cheap way to network games or to transfer files between computers using Zmodem Protocol, Xmodem Protocol etc. This can also be used with many Microprocessor Development Systems.

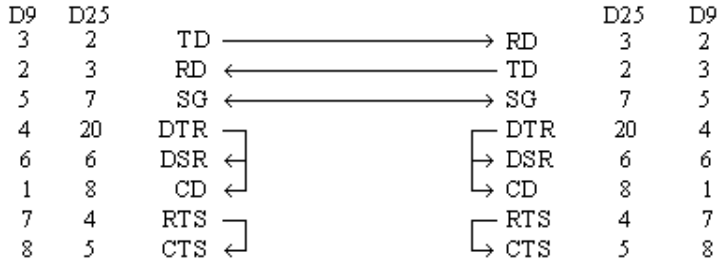


Figure 1 : Null Modem Wiring Diagram

Above is my preferred method of wiring a Null Modem. It only requires 3 wires (TD, RD & SG) to be wired straight through thus is more cost effective to use with long cable runs. The theory of operation is reasonably easy. The aim is to make to computer think it is talking to a modem rather than another computer. Any data transmitted from the first computer must be received by the second thus TD is connected to RD. The second computer must have the same set-up thus RD is connected to TD. Signal Ground (SG) must also be connected so both grounds are common to each computer.

The Data Terminal Ready is looped back to Data Set Ready and Carrier Detect on both computers. When the Data Terminal Ready is asserted active, then the Data Set Ready and Carrier Detect immediately become active. At this point the computer thinks the Virtual Modem to which it is connected is ready and has detected the carrier of the other modem.

All left to worry about now is the Request to Send and Clear To Send. As both computers communicate together at the same speed, flow control is not needed thus these two lines are also linked together on each computer. When the computer wishes to send data, it asserts the Request to Send high and as it's hooked together with the Clear to Send, It immediately gets a reply that it is ok to send and does so.

Notice that the ring indicator is not connected to anything of each end. This line is only used to tell the computer that there is a ringing signal on the phone line. As we don't have a modem connected to the phone line this is left disconnected.

LoopBack Plug

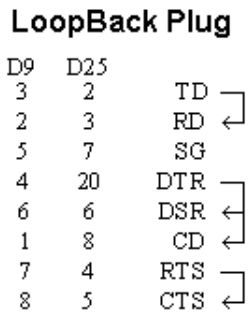


Figure 2 : Loopback Plug Wiring Diagram

This loopback plug can come in extremely handy when writing Serial / RS232 Communications Programs. It has the receive and transmit lines connected together, so that anything transmitted out of the Serial Port is immediately received by the same port. If you connect this to a Serial Port an load a Terminal Program, anything you type will be immediately displayed on the screen. This can be used with the examples later in this tutorial.

Please note that this is not intended for use with Diagnostic Programs and thus will probably not work. For these programs you require a differently wired Loop Back plug which may vary from program to program.

DTE / DCE Speeds

We have already talked briefly about DTE & DCE. A typical Data Terminal Device is a computer and a typical Data Communications Device is a Modem. Often people will talk about DTE to DCE or DCE to DCE speeds. DTE to DCE is the speed between your modem and computer, sometimes referred to as your terminal speed. This should run at faster speeds than the DCE to DCE speed. DCE to DCE is the link between modems, sometimes called the line speed.

Most people today will have 28.8K or 33.6K modems. Therefore we should expect the DCE to DCE speed to be either 28.8K or 33.6K. Considering the high speed of the modem we should expect the DTE to DCE speed to be about 115,200 BPS.(Maximum Speed of the 16550a UART) This is where some people often fall into a trap. The communications program which they use have settings for DCE to DTE speeds. However they see 9.6 KBPS, 14.4 KBPS etc and think it

is your modem speed.

Today's Modems should have Data Compression build into them. This is very much like PK-ZIP but the software in your modem compresses and decompresses the data. When set up correctly you can expect compression ratios of 1:4 or even higher. 1 to 4 compression would be typical of a text file. If we were transferring that text file at 28.8K (DCE-DCE), then when the modem compresses it you are actually transferring 115.2 KBPS between computers and thus have a DCE-DTE speed of 115.2 KBPS. Thus this is why the DCE-DTE should be much higher than your modem's connection speed.

Some modem manufacturers quote a maximum compression ratio as 1:8. Lets say for example its on a new 33.6 KBPS modem then we may get a maximum 268,800 BPS transfer between modem and UART. If you only have a 16550a which can do 115,200 BPS tops, then you would be missing out on a extra bit of performance. Buying a 16C650 should fix your problem with a maximum transfer rate of 230,400 BPS.

However don't abuse your modem if you don't get these rates. These are MAXIMUM compression ratios. In some instances if you try to send a already compressed file, your modem can spend more time trying the compress it, thus you get a transmission speed less than your modem's connection speed. If this occurs try turning off your data compression. This should be fixed on newer modems. Some files compress easier than others thus any file which compresses easier is naturally going to have a higher compression ratio.

Flow Control

So if our DTE to DCE speed is several times faster than our DCE to DCE speed the PC can send data to your modem at 115,200 BPS. Sooner or later data is going to get lost as buffers overflow, thus flow control is used. Flow control has two basic varieties, Hardware or Software.

Software flow control, sometimes expressed as Xon/Xoff uses two characters Xon and Xoff. Xon is normally indicated by the ASCII 17 character where as the ASCII 19 character is used for Xoff. The modem will only have a small buffer so when the computer fills it up the modem sends a Xoff character to tell the computer to stop sending data. Once the modem has room for more data it then sends a Xon character and the computer sends more data. This type of flow control has the advantage that it doesn't require any more wires as the characters are sent via the TD/RD lines. However on slow links each character requires 10 bits which can slow communications down.

Hardware flow control is also known as RTS/CTS flow control. It uses two wires in your serial cable rather than extra characters transmitted in your data lines. Thus hardware flow control will not slow down transmission times like Xon-Xoff does. When the computer wishes to send data it takes active the Request to Send line. If the modem has room for this data, then the modem will reply by taking active the Clear to Send line and the computer starts sending data. If the modem does not have the room then it will not send a Clear to Send.

The UART (8250 and Compatibles)

UART stands for Universal Asynchronous Receiver / Transmitter. Its the little box of tricks found on your serial card which plays the little games with your modem or other connected devices. Most cards will have the UART's integrated into other chips which may also control your parallel port, games port, floppy or hard disk drives and are typically surface mount devices. The 8250 series, which includes the 16450, 16550, 16650, & 16750 UARTS are the most commonly found type in your PC. Later we will look at other types which can be used in your homemade devices and projects.

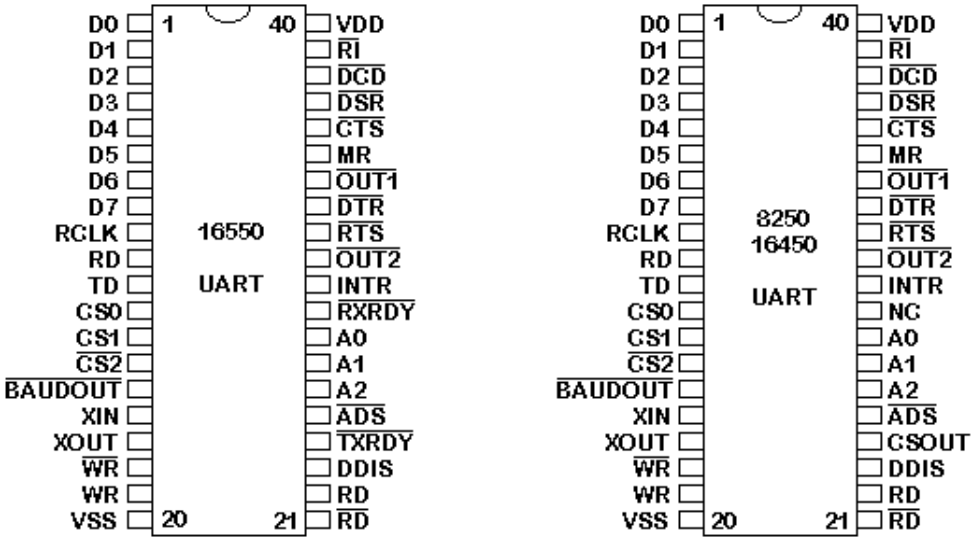


Figure 3 : Pin Diagrams for 16550, 16450 & 8250 UARTs

The 16550 is chip compatible with the 8250 & 16450. The only two differences are pins 24 & 29. On the 8250 Pin 24 was chip select out which functioned only as a indicator to if the chip was active or not. Pin 29 was not connected on the 8250/16450 UARTs. The 16550 introduced two new pins in their place. These are Transmit Ready and Receive Ready which can be implemented with DMA (Direct Memory Access). These Pins have two different modes of operation. Mode 0 supports single transfer DMA where as Mode 1 supports Multi-transfer DMA.

Mode 0 is also called the 16450 mode. This mode is selected when the FIFO buffers are disabled via Bit 0 of the FIFO Control Register or When the FIFO buffers are enabled but DMA Mode Select = 0. (Bit 3 of FCR) In this mode RXRDY is active low when at least one character (Byte) is present in the Receiver Buffer. RXRDY will go inactive high when no more characters are left in the Receiver Buffer. TXRDY will be active low when there are no characters in the Transmit Buffer. It will go inactive high after the first character / byte is loaded into the Transmit Buffer.

Mode 1 is when the FIFO buffers are active and the DMA Mode Select = 1. In Mode 1, RXRDY will go active low when the trigger level is reached or when 16550 Time Out occurs and will return to inactive state when no more characters are left in the FIFO. TXRDY will be active when no characters are present in the Transmit Buffer and will go inactive when the FIFO Transmit Buffer is completely Full.

All the UARTs pins are TTL compatible. That includes TD, RD, RI, DCD, DSR, CTS, DTR and

RTS which all interface into your serial plug, typically a D-type connector. Therefore RS232 Level Converters (which we talk about in detail later) are used. These are commonly the DS1489 Receiver and the DS1488 as the PC has +12 and -12 volt rails which can be used by these devices. The RS232 Converters will convert the TTL signal into RS232 Logic Levels.

Pin No.	Name	Notes
Pin 1:8	D0:D7	Data Bus
Pin 9	RCLK	Receiver Clock Input. The frequency of this input should equal the receivers baud rate * 16
Pin 10	RD	Receive Data
Pin 11	TD	Transmit Data
Pin 12	CS0	Chip Select 0 - Active High
Pin 13	CS1	Chip Select 1 - Active High
Pin 14	nCS2	Chip Select 2 - Active Low
Pin 15	nBAUDOUT	Baud Output - Output from Programmable Baud Rate Generator. Frequency = (Baud Rate x 16)
Pin 16	XIN	External Crystal Input - Used for Baud Rate Generator Oscillator
Pin 17	XOUT	External Crystal Output
Pin 18	nWR	Write Line - Inverted
Pin 19	WR	Write Line - Not Inverted
Pin 20	VSS	Connected to Common Ground
Pin 21	RD	Read Line - Inverted
Pin 22	nRD	Read Line - Not Inverted
Pin 23	DDIS	Driver Disable. This pin goes low when CPU is reading from UART. Can be connected to Bus Transceiver in case of high capacity data bus.
Pin 24	nTXRDY	Transmit Ready
Pin 25	nADS	Address Strobe. Used if signals are not stable during read or write cycle
Pin 26	A2	Address Bit 2
Pin 27	A1	Address Bit 1
Pin 28	A0	Address Bit 0
Pin 29	nRXRDY	Receive Ready
Pin 30	INTR	Interrupt Output
Pin 31	nOUT2	User Output 2
Pin 32	nRTS	Request to Send
Pin 33	nDTR	Data Terminal Ready

Pin 34	nOUT1	User Output 1
Pin 35	MR	Master Reset
Pin 36	nCTS	Clear To Send
Pin 37	nDSR	Data Set Ready
Pin 38	nDCD	Data Carrier Detect
Pin 39	nRI	Ring Indicator
Pin 40	VDD	+ 5 Volts

Table 2 : Pin Assignments for 16550A UART

The UART requires a Clock to run. If you look at your serial card a common crystal found is either a 1.8432 MHZ or a 18.432 MHZ Crystal. The crystal in connected to the XIN-XOUT pins of the UART using a few extra components which help the crystal to start oscillating. This clock will be used for the Programmable Baud Rate Generator which directly interfaces into the transmit timing circuits but not directly into the receiver timing circuits. For this an external connection mast be made from pin 15 (BaudOut) to pin 9 (Receiver clock in.) Note that the clock signal will be at Baudrate * 16.

If you are serious about pursuing the 16550 UART used in your PC further, then would suggest downloading a copy of the PC16550D data sheet from National Semiconductors Site. Data sheets are available in .PDF format so you will need Adobe Acrobat Reader to read these. Texas Instruments has released the 16750 UART which has 64 Byte FIFO's. Data Sheets for the TL16C750 are available from the Texas Instruments Site.

Types of UARTS (For PC's)

8250	First UART in this series. It contains no scratch register. The 8250A was an improved version of the 8250 which operates faster on the bus side.
8250A	This UART is faster than the 8250 on the bus side. Looks exactly the same to software than 16450.
8250B	Very similar to that of the 8250 UART.
16450	Used in AT's (Improved bus speed over 8250's). Operates comfortably at 38.4KBPS. Still quite common today.
16550	This was the first generation of buffered UART. It has a 16 byte buffer, however it doesn't work and is replaced with the 16550A.
16550A	Is the most common UART use for high speed communications eg 14.4K & 28.8K Modems. They made sure the FIFO buffers worked on this UART.
16650	Very recent breed of UART. Contains a 32 byte FIFO, Programmable X-On / X-Off characters and supports power management.
16750	Produced by Texas Instruments. Contains a 64 byte FIFO.

Part Two : Serial Port's Registers (PC's)

Port Addresses & IRQ’s

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Table 3 : Standard Port Addresses

Above is the standard port addresses. These should work for most P.C’s. If you just happen to be lucky enough to own a IBM P/S2 which has a micro-channel bus, then expect a different set of addresses and IRQ’s. Just like the LPT ports, the base addresses for the COM ports can be read from the BIOS Data Area.

Start Address	Function
0000:0400	COM1’s Base Address
0000:0402	COM2’s Base Address
0000:0404	COM3’s Base Address
0000:0406	COM4’s Base Address

Table 4 - COM Port Addresses in the BIOS Data Area;

The above table shows the address at which we can find the Communications (COM) ports addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your communications ports.

```
#include
#include

void main(void)
{
 unsigned int far *ptraddr;  /* Pointer to location of Port Addresses */
 unsigned int address;       /* Address of Port */
 int a;

 ptraddr=(unsigned int far *)0x00000400;

 for (a = 0; a <  4; a++)
   {
    address = *ptraddr;
    if (address == 0)
                printf("No port found for COM%d \n",a+1);
    else
                printf("Address assigned to COM%d is %Xh\n",a+1,address);
    *ptraddr++;
```

```
    }
}
```

Table of Registers

Base Address	DLAB	Read/Write	Abr.	Register Name
+ 0	=0	Write	-	Transmitter Holding Buffer
	=0	Read	-	Receiver Buffer
	=1	Read/Write	-	Divisor Latch Low Byte
+ 1	=0	Read/Write	IER	Interrupt Enable Register
	=1	Read/Write	-	Divisor Latch High Byte
+ 2	-	Read	IIR	Interrupt Identification Register
	-	Write	FCR	FIFO Control Register
+ 3	-	Read/Write	LCR	Line Control Register
+ 4	-	Read/Write	MCR	Modem Control Register
+ 5	-	Read	LSR	Line Status Register
+ 6	-	Read	MSR	Modem Status Register
+ 7	-	Read/Write	-	Scratch Register

Table 5 : Table of Registers

DLAB ?

You will have noticed in the table of registers that there is a DLAB column. When DLAB is set to ’0’ or ’1’ some of the registers change. This is how the UART is able to have 12 registers (including the scratch register) through only 8 port addresses. DLAB stands for Divisor Latch Access Bit. When DLAB is set to ’1’ via the line control register, two registers become available from which you can set your speed of communications measured in bits per second.

The UART will have a crystal which should oscillate around 1.8432 MHZ. The UART incorporates a divide by 16 counter which simply divides the incoming clock signal by 16. Assuming we had the 1.8432 MHZ clock signal, that would leave us with a maximum, 115,200 hertz signal making the UART capable of transmitting and receiving at 115,200 Bits Per Second (BPS). That would be fine for some of the faster modems and devices which can handle that speed, but others just wouldn’t communicate at all. Therefore the UART is fitted with a Programmable Baud Rate Generator which is controlled by two registers.

Lets say for example we only wanted to communicate at 2400 BPS. We worked out that we would have to divide 115,200 by 48 to get a workable 2400 Hertz Clock. The "Divisor", in this case 48, is stored in the two registers controlled by the "Divisor Latch Access Bit". This divisor can be any number which can be stored in 16 bits (ie 0 to 65535). The UART only has a 8 bit data bus, thus this is where the two registers are used. The first register (Base + 0) when DLAB = 1 stores the "Divisor latch low byte" where as the second register (base + 1 when DLAB = 1)

stores the "Divisor latch high byte."

Below is a table of some more common speeds and their divisor latch high bytes & low bytes. Note that all the divisors are shown in Hexadecimal.

Speed (BPS)	Divisor (Dec)	Divisor Latch High Byte	Divisor Latch Low Byte
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

Table 6 : Table of Commonly Used Baudrate Divisors

Interrupt Enable Register (IER)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Enables Low Power Mode (16750)
Bit 4	Enables Sleep Mode (16750)
Bit 3	Enable Modem Status Interrupt
Bit 2	Enable Receiver Line Status Interrupt
Bit 1	Enable Transmitter Holding Register Empty Interrupt
Bit 0	Enable Received Data Available Interrupt

Table 7 : Interrupt Enable Register

The Interrupt Enable Register could possibly be one of the easiest registers on a UART to understand. Setting Bit 0 high enables the Received Data Available Interrupt which generates an interrupt when the receiving register/FIFO contains data to be read by the CPU.

Bit 1 enables Transmit Holding Register Empty Interrupt. This interrupts the CPU when the transmitter buffer is empty. Bit 2 enables the receiver line status interrupt. The UART will interrupt when the receiver line status changes. Likewise for bit 3 which enables the modem status interrupt. Bits 4 to 7 are the easy ones. They are simply reserved. (If only everything was that easy!)

Interrupt Identification Register (IIR)

Bit	Notes		
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but Unusable
	1	1	FIFO Enabled
Bit 5	64 Byte Fifo Enabled (16750 only)		
Bit 4	Reserved		
Bit 3	0 Reserved on 8250, 16450		
	1 16550 Time-out Interrupt Pending		
Bits 1 and 2	Bit 2	Bit 1	
	0	0	Modem Status Interrupt
	0	1	Transmitter Holding Register Empty Interrupt
	1	0	Received Data Available Interrupt
	1	1	Receiver Line Status Interrupt
Bit 0	0 Interrupt Pending		
	1 No Interrupt Pending		

Table 8 : Interrupt Identification Register

The interrupt identification register is a read only register. Bits 6 and 7 give status on the FIFO Buffer. When both bits are ’0’ no FIFO buffers are active. This should be the only result you will get from a 8250 or 16450. If bit 7 is active but bit 6 is not active then the UART has it’s buffers enabled but are unusable. This occurs on the 16550 UART where a bug in the FIFO buffer made the FIFO’s unusable. If both bits are ’1’ then the FIFO buffers are enabled and fully operational.

Bits 4 and 5 are reserved. Bit 3 shows the status of the time-out interrupt on a 16550 or higher.

Lets jump to Bit 0 which shows whether an interrupt has occurred. If an interrupt has occurred it’s status will shown by bits 1 and 2. These interrupts work on a priority status. The Line Status Interrupt has the highest Priority, followed by the Data Available Interrupt, then the Transmit Register Empty Interrupt and then the Modem Status Interrupt which has the lowest priority.

First In / First Out Control Register (FCR)

Bit	Notes		
Bits 6 and 7	Bit 7	Bit 6	Interrupt Trigger Level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

Table 9 : FIFO Control Register

The FIFO register is a write only register. This register is used to control the FIFO (First In / First Out) buffers which are found on 16550's and higher.

Bit 0 enables the operation of the receive and transmit FIFO's. Writing a '0' to this bit will disable the operation of transmit and receive FIFO's, thus you will loose all data stored in these FIFO buffers.

Bit's 1 and 2 control the clearing of the transmit or receive FIFO's. Bit 1 is responsible for the receive buffer while bit 2 is responsible for the transmit buffer. Setting these bits to 1 will only clear the contents of the FIFO and will not affect the shift registers. These two bits are self resetting, thus you don't need to set the bits to '0' when finished.

Bit 3 enables the DMA mode select which is found on 16550 UARTs and higher. More on this later. Bits 4 and 5 are those easy type again, Reserved.

Bits 6 and 7 are used to set the triggering level on the Receive FIFO. For example if bit 7 was set to '1' and bit 6 was set to '0' then the trigger level is set to 8 bytes. When there is 8 bytes of data in the receive FIFO then the Received Data Available interrupt is set. See (IIR)

Line Control Register (LCR)

Bit 7	1 Divisor Latch Access Bit			
	0 Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register			
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0 One Stop Bit			
	1 2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.			
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Table 10 : Line Control Register

The Line Control register sets the basic parameters for communication. Bit 7 is the Divisor Latch Access Bit or DLAB for short. We have already talked about what it does. (See DLAB?) Bit 6 Sets break enable. When active, the TD line goes into "Spacing" state which causes a break in the receiving UART. Setting this bit to ’0’ Disables the Break.

Bits 3,4 and 5 select parity. If you study the 3 bits, you will find that bit 3 controls parity. That is, if it is set to ’0’ then no parity is used, but if it is set to ’1’ then parity is used. Jumping to bit 5, we can see that it controls sticky parity. Sticky parity is simply when the parity bit is always transmitted and checked as a ’1’ or ’0’. This has very little success in checking for errors as if the first 4 bits contain errors but the sticky parity bit contains the appropriately set bit, then a parity error will not result. Sticky high parity is the use of a ’1’ for the parity bit, while the opposite, sticky low parity is the use of a ’0’ for the parity bit.

If bit 5 controls sticky parity, then turning this bit off must produce normal parity provided bit 3 is still set to ’1’. Odd parity is when the parity bit is transmitted as a ’1’ or ’0’ so that there is a odd number of 1’s. Even parity must then be the parity bit produces and even number of 1’s. This provides better error checking but still is not perfect, thus CRC-32 is often used for software error correction. If one bit happens to be inverted with even or odd parity set, then a parity error will occur, however if two bits are flipped in such a way that it produces the correct parity bit then an parity error will no occur.

Bit 2 sets the length of the stop bits. Setting this bit to ’0’ will produce one stop bit, however setting it to ’1’ will produce either 1.5 or 2 stop bits depending upon the word length. Note that the receiver only checks the first stop bit.

Bits 0 and 1 set the word length. This should be pretty straight forward. A word length of 8 bits is most commonly used today.

Modem Control Register (MCR)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Autoflow Control Enabled (16750 only)
Bit 4	LoopBack Mode
Bit 3	Aux Output 2
Bit 2	Aux Output 1
Bit 1	Force Request to Send
Bit 0	Force Data Terminal Ready

Table 11 : Modem Control Register

The Modem Control Register is a Read/Write Register. Bits 5,6 and 7 are reserved. Bit 4 activates the loopback mode. In Loopback mode the transmitter serial output is placed into marking state. The receiver serial input is disconnected. The transmitter out is looped back to the receiver in. DSR, CTS, RI & DCD are disconnected. DTR, RTS, OUT1 & OUT2 are connected to the modem control inputs. The modem control output pins are then place in an inactive state. In this mode any data which is placed in the transmitter registers for output is received by the receiver circuitry on the same chip and is available at the receiver buffer. This can be used to test the UARTs operation.

Aux Output 2 maybe connected to external circuitry which controls the UART-CPU interrupt process. Aux Output 1 is normally disconnected, but on some cards is used to switch between a 1.8432MHZ crystal to a 4MHZ crystal which is used for MIDI. Bits 0 and 1 simply control their relevant data lines. For example setting bit 1 to ’1’ makes the request to send line active.

Line Status Register (LSR)

Bit	Notes
Bit 7	Error in Received FIFO
Bit 6	Empty Data Holding Registers
Bit 5	Empty Transmitter Holding Register
Bit 4	Break Interrupt
Bit 3	Framing Error
Bit 2	Parity Error
Bit 1	Overrun Error
Bit 0	Data Ready

Table 12 : Line Status Register

The line status register is a read only register. Bit 7 is the error in received FIFO bit. This bit is high when at least one break, parity or framing error has occurred on a byte which is contained in the FIFO.

When bit 6 is set, both the transmitter holding register and the shift register are empty. The UART's holding register holds the next byte of data to be sent in parallel fashion. The shift register is used to convert the byte to serial, so that it can be transmitted over one line. When bit 5 is set, only the transmitter holding register is empty. So what's the difference between the two? When bit 6, the transmitter holding and shift registers are empty, no serial conversions are taking place so there should be no activity on the transmit data line. When bit 5 is set, the transmitter holding register is empty, thus another byte can be sent to the data port, but a serial conversion using the shift register may be taking place.

The break interrupt (Bit 4) occurs when the received data line is held in a logic state '0' (Space) for more than the time it takes to send a full word. That includes the time for the start bit, data bits, parity bits and stop bits.

A framing error (Bit 3) occurs when the last bit is not a stop bit. This may occur due to a timing error. You will most commonly encounter a framing error when using a null modem linking two computers or a protocol analyzer when the speed at which the data is being sent is different to that of what you have the UART set to receive it at.

A overrun error normally occurs when your program can't read from the port fast enough. If you don't get an incoming byte out of the register fast enough, and another byte just happens to be received, then the last byte will be lost and a overrun error will result.

Bit 0 shows data ready, which means that a byte has been received by the UART and is at the receiver buffer ready to be read.

Modem Status Register (MSR)

Bit	Notes
Bit 7	Carrier Detect
Bit 6	Ring Indicator
Bit 5	Data Set Ready
Bit 4	Clear To Send
Bit 3	Delta Data Carrier Detect
Bit 2	Trailing Edge Ring Indicator
Bit 1	Delta Data Set Ready
Bit 0	Delta Clear to Send

Table 13 : Modem Status Register

Bit 0 of the modem status register shows delta clear to send, delta meaning a change in, thus delta clear to send means that there was a change in the clear to send line, since the last read of this register. This is the same for bits 1 and 3. Bit 1 shows a change in the Data Set Ready line where as Bit 3 shows a change in the Data Carrier Detect line. Bit 2 is the Trailing Edge Ring Indicator which indicates that there was a transformation from low to high state on the Ring Indicator line.

Bits 4 to 7 show the current state of the data lines when read. Bit 7 shows Carrier Detect, Bit 6 shows Ring Indicator, Bit 5 shows Data Set Ready & Bit 4 shows the status of the Clear To Send line.

Scratch Register

The scratch register is not used for communications but rather used as a place to leave a byte of data. The only real use it has is to determine whether the UART is a 8250/8250B or a 8250A/16450 and even that is not very practical today as the 8250/8250B was never designed for AT's and can't hack the bus speed.

Part 3 : Programming (PC's)

Polling or Interrupt Driven?
Source Code - Termpoll.c (Polling Version)
Source Code - Buff1024.c (ISR Version)
Interrupt Vectors
Interrupt Service Routine
UART Configuration
Main Routine (Loop)
Determining the type of UART via Software

Part 4 : External Hardware - Interfacing Methods

RS-232 Waveforms
RS-232 Level Converters
Making use of the Serial Format

8250 and compatable UART's
CDP6402, AY-5-1015 / D36402R-9 etc UARTs
Microcontrollers

TIFF Image Creation

Written by Paul Bourke
August 1998

The following demonstrates how to create 24 bit colour RGB TIFF (Tagged Image FIle Format) files. That is, how to create images from your own software that can be then opened and manipulated with image handling software, for example: GIMP, PhotoShop, etc. Given this aim, this document illustrates the "minimal" requirements necessary to create a TIFF file, it does not provide enough information for writing a TIFF file reader. For more information on the full TIFF specification the following postscript and pdf files describe the TIFF version 6.

tiff.ps.gz tiff.pdf.gz

The basic structure of a TIFF file is as follows:

The first 8 bytes forms the **header**. The first two bytes of which is either "II" for little endian byte ordering or "MM" for big endian byte ordering. In what follows we'll be assuming little endian ordering. Note: any true TIFF reading software is supposed to be handle both types. The next two bytes of the header should be 0 and 42_{dec} ($2a_{hex}$). The remaining 4 bytes of the header is the offset from the start of the file to the first "Image File Directory" (IFD), this normally follows the image data it applies to. In the example below there is only one image and one IFD.

An **IFD** consists of two bytes indicating the number of entries followed by the entries themselves. The IFD is terminated with 4 byte offset to the next IFD or 0 if there are none. A TIFF file must contain at least one IFD!

Each **IFD entry** consists of 12 bytes. The first two bytes identifies the tag type (as in Tagged Image File Format). The next two bytes are the field type (byte, ASCII, short int, long int, ...). The next four bytes indicate the number of values. The last four bytes is either the value itself or an offset to the values. Considering the first IFD entry from the example gievn below:

```
   0100 0003 0000 0001 0064 0000
      |         |    |         |
tag --+         |    |         |
short int -----+    |         |
one value ----------+         |
value of 100 -----------------+
```

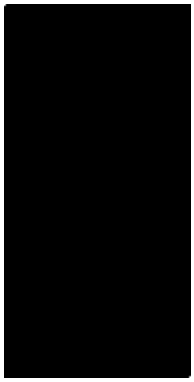
Example

The following is an example using the TIFF file shown on the right, namely a black image with a single white pixel at the top left and the bottom right position. The image is 100 pixels wide by 200 pixels high.

A hex dump is given below along with matching pointers and locations marked in matching colours,

these colours further match the appropriate parts of the source code gievn later. The tags are underlined.

```
4d4d 002a 0000 ea68 ffff ff00 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

.... ....  black 0's deleted  .... ....

0000 0000 00ff ffff 000e 0100 0003 0000
0001 0064 0000 0101 0003 0000 0001 00c8
0000 0102 0003 0000 0003 0000 eb16 0103
0003 0000 0001 0001 0000 0106 0003 0000
0001 0002 0000 0111 0004 0000 0001 0000
0008 0112 0003 0000 0001 0001 0000 0115
0003 0000 0001 0003 0000 0116 0003 0000
0001 00c8 0000 0117 0004 0000 0001 0000
ea60 0118 0003 0000 0003 0000 eb1c 0119
0003 0000 0003 0000 eb22 011c 0003 0000
0001 0001 0000 0153 0003 0000 0003 0000
eb28 0000 0000 0008 0008 0008 0000 0000
0000 00ff 00ff 00ff 0001 0001 0001
```



The above example uses 14_{dec} ($000ei_{hex}$) directory entries.

0100 - Image width
0101 - Image height
0102 - Bits per sample (8)
0103 - Compression method (1 = uncompressed)
0106 - Photometric Interpretation (2 = RGB)
0111 - Strip Offsets
0112 - Orientation (1 = 0 top, 0 left hand side) 0115 - Samples per pixel (1)
0116 - Rows per strip (200 = image height)
0117 - Strip Byte Counts (60000 = 100 x 200 x 3)
0118 - Minimum sample value (0,0,0)
0119 - Maximum sample value (255,255,255)
011c - Planar configuration (1 = single image plane)
0153 - Sample format

Source code example

The following is the guts of a C program to create a TIFF file of width nx, height ny. Each pixel is made up of 3 bytes, one byte for each of Red, Green, Blue. Each colour component ranges from 0 (black) to 255 (white).

```
/* Write the header */
WriteHexString(fptr,"4d4d002a");    /* Little endian & TIFF identifier */
offset = nx * ny * 3 + 8;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);
```

```
  /* Write the binary data */
  for (j=0;j<ny;j++) {
     for (i=0;i<nx;i++) {
... calculate the RGB value between 0 and 255 ...
        fputc(red,fptr);
        fputc(green,fptr);
        fputc(blue,fptr);
     }
  }

  /* Write the footer */
  WriteHexString(fptr,"000e");  /* The number of directory entries (14) */

  /* Width tag, short int */
  WriteHexString(fptr,"0100000300000001");
  fputc((nx & 0xff00) / 256,fptr);    /* Image width */
  fputc((nx & 0x00ff),fptr);
  WriteHexString(fptr,"0000");

  /* Height tag, short int */
  WriteHexString(fptr,"0101000300000001");
  fputc((ny & 0xff00) / 256,fptr);    /* Image height */
  fputc((ny & 0x00ff),fptr);
  WriteHexString(fptr,"0000");

  /* Bits per sample tag, short int */
  WriteHexString(fptr,"0102000300000003");
  offset = nx * ny * 3 + 182;
  putc((offset & 0xff000000) / 16777216,fptr);
  putc((offset & 0x00ff0000) / 65536,fptr);
  putc((offset & 0x0000ff00) / 256,fptr);
  putc((offset & 0x000000ff),fptr);

  /* Compression flag, short int */
  WriteHexString(fptr,"0103000300000000100010000");

  /* Photometric interpolation tag, short int */
  WriteHexString(fptr,"0106000300000000100020000");

  /* Strip offset tag, long int */
  WriteHexString(fptr,"0111000400000000100000008");

  /* Orientation flag, short int */
  WriteHexString(fptr,"0112000300000000100010000");

  /* Sample per pixel tag, short int */
  WriteHexString(fptr,"0115000300000000100030000");

  /* Rows per strip tag, short int */
  WriteHexString(fptr,"0116000300000001");
  fputc((ny & 0xff00) / 256,fptr);
  fputc((ny & 0x00ff),fptr);
  WriteHexString(fptr,"0000");

  /* Strip byte count flag, long int */
  WriteHexString(fptr,"0117000400000001");
  offset = nx * ny * 3;
  putc((offset & 0xff000000) / 16777216,fptr);
  putc((offset & 0x00ff0000) / 65536,fptr);
  putc((offset & 0x0000ff00) / 256,fptr);
  putc((offset & 0x000000ff),fptr);
```

```
/* Minimum sample value flag, short int */
WriteHexString(fptr,"0118000300000003");
offset = nx * ny * 3 + 188;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);

/* Maximum sample value tag, short int */
WriteHexString(fptr,"0119000300000003");
offset = nx * ny * 3 + 194;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);

/* Planar configuration tag, short int */
WriteHexString(fptr,"011c00030000000100010000");

/* Sample format tag, short int */
WriteHexString(fptr,"0153000300000003");
offset = nx * ny * 3 + 200;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);

/* End of the directory entry */
WriteHexString(fptr,"00000000");

/* Bits for each colour channel */
WriteHexString(fptr,"000800080008");

/* Minimum value for each component */
WriteHexString(fptr,"000000000000");

/* Maximum value per channel */
WriteHexString(fptr,"00ff00ff00ff");

/* Samples per pixel for each channel */
WriteHexString(fptr,"000100010001");
```