



USER MANUAL



KD485 - STD
KD485 - ADE
KD485 - PROG

KKSYSYSTEMS.COM

Copyright

(C) 1992-2017 KK Systems Ltd. No reproduction of any part of this document, in any form, is allowed without prior written permission from KK Systems Ltd. All other copyrights and trademarks acknowledged.

Extract from Conditions of Sale

Any electronic device or system can fail, possibly resulting in the loss of valuable programs or data. It is your responsibility to ensure that all such valuable material is backed-up at all times. We are not liable for any direct, indirect or consequential loss caused directly or indirectly through the use of this product. All our software is sold on an "as is" basis without a warranty of any kind. We do not claim that this product is suitable for all potential applications. It is your responsibility to verify that the product works in its intended application. In the interest of progress, we reserve the right to alter prices and specifications without prior notice.

Safety Warning

The KD485 is not authorised for use in any situation where injury or death could result from its failure. Neither interface should be connected to a dangerous potential.

Year 2000 Compliance

There is no date dependence in the KD485, with the exception of the KD485-PROG with the RTC (real time clock) option. The RTC is also Y2K compliant.

Product versions

This manual covers the following versions:

KD485 firmware	v1.03 dated 13/11/95 or later
TERM.EXE	v1.02

Edition 13 15 December 2017

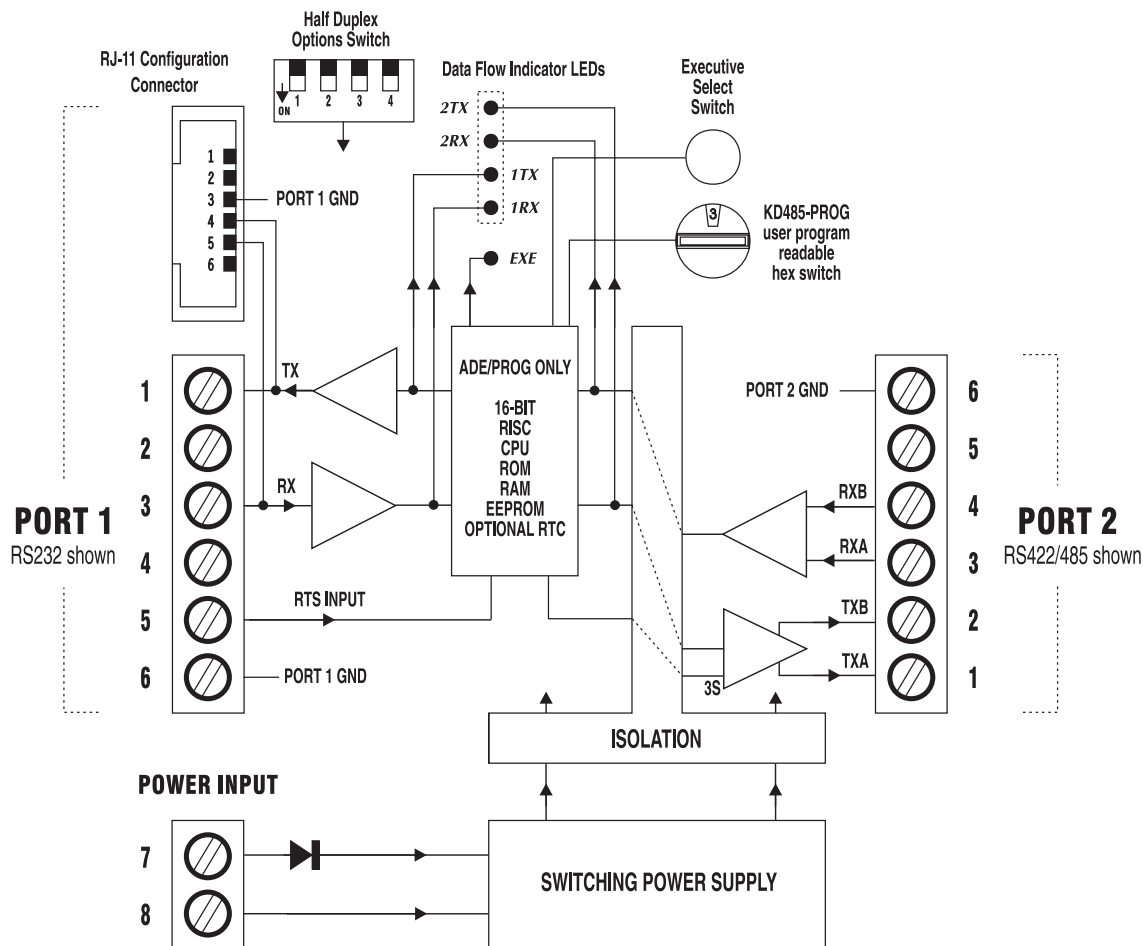
Table of Contents

KD485 Overview	1
Front Panel Description	2
DIP Switch	2
Standard KD485 Versions	3
1. KD485-STD Isolated Converter; standard model	3
2. KD485-ADE Isolated Converter; with Auto Driver Enable	3
3. KD485-PROG Isolated Converter; user programmable in ANSI C	4
Custom Versions	4
Specification	4
Changes from previous (1994-1998) KD485 version	5
KD485-ADE/PROG Mode 2 Detail Description	6
Installation	7
Product and Interface Type identification	7
DC Power Requirements	7
KD485-STD Configuration	8
KD485-ADE Configuration	8
Fitting KD485 on DIN Rail	10
KD485-PROG Configuration	11
Wiring Examples	12
Ports & Connections	18
RS232 Ports	18
RS232 Ports - Detail Description of Terminals	18
RS422/485 Ports	19
RS422/485 Ports - difference between RS422 and RS485?	20
RS422/485 Ports - Grounding	20
RS422/485 Ports - A/B Terminal Markings	20
RS422/485 Ports - Detail Description of Terminals	20
RS422/485 Ports - Terminators	21
RS422/485 Ports - Bus Pullups	21
20mA Loop Ports	22
20mA Loop Ports - Detail Description of Terminals	22
KD485-ADE and KD485-PROG - General Data Flow Details	24
I/O Queues	24
Transmit Queues	24
Receive Queues	24
Handshaking	24
The Executive	25
Basic Principles	25
Pn – Port Configuration	25
MD – KD485 Mode	26
AD – RS485 address – applies to Mode 2 only	26
UF – Upload a “user program” file – KD485-PROG only	27
RU – Run program	27
TE – Test ports	27
TS – Test Slave device	27
HE – Help	28
SOFTWARE UTILITIES	29
TERM.EXE	29

KDCFG.EXE	29
Troubleshooting	30
KD485 power-up problems	30
RS232 Communications Problems	30
RS422/485 Communications Problems	30
Cannot enter the KD485 Executive	31
Intermittent Comms Errors - General	31
Intermittent Comms Errors - RS422/485	31
Intermittent Comms Errors - KD485-PROG	31
C Introduction	32
Step 1: Get to know the KD485	32
Step 2: Compiler Installation	32
Step 3: KD485 C Software Utilities Installation	33
Step 4: Reboot your PC	33
Step 5: Create a program	33
Debug Port	33
Large Projects	34
C Reference	35
The Runtime Memory Map	35
Runtime Library	36
Using a non-Hi-Tech Compiler ?	36
Modifying Statically Initialised Data	36
Data Alignment	37
C Extensions	38
Serial Communications	38
Debug Port	39
I/O Queue Management Functions	40
I/O Port Configuration Functions	41
Tri-State Driver Control	41
Optional optocoupler output	42
Reading RTS Input	42
"Break" Sequences	42
Timing Functions	42
Real Time Clock	43
Checksums and CRC	43
Bit Operations and Rotation	43
EEPROM Access	43
Error Conditions	45
Switches, LEDs, miscellaneous	45
Watchdog	46
Interrupt Vectors	46
Performance Hints & Warnings	46
KD485 – PPC Differences	47
ASCII Character Codes	48
Overall Dimensions	49

KD485 Overview

The KD485 is a multi-mode interface converter, with two serial asynchronous ports which are available factory-fitted in any combination of RS232, RS422/485 or 20mA loop:



The default port configuration (shown above) is **RS232 on Port 1** and **RS422/485 on Port 2**. If the port types are not specified when ordering this is what is supplied. All other configurations are specified with a suffix, e.g.

KD485-ADE-422-20MA

specifies Port 1 as RS422/485 and Port 2 as 20mA loop.

Port 1, power input, and Port 2 are on removable screw terminals. For additional convenience during configuration of KD485-ADE and KD485-PROG, Port 1 is electrically duplicated on an RJ-11 connector. The RJ-11 connector can also be useful on the KD485-STD as it provides a quicker connection to Port 1.

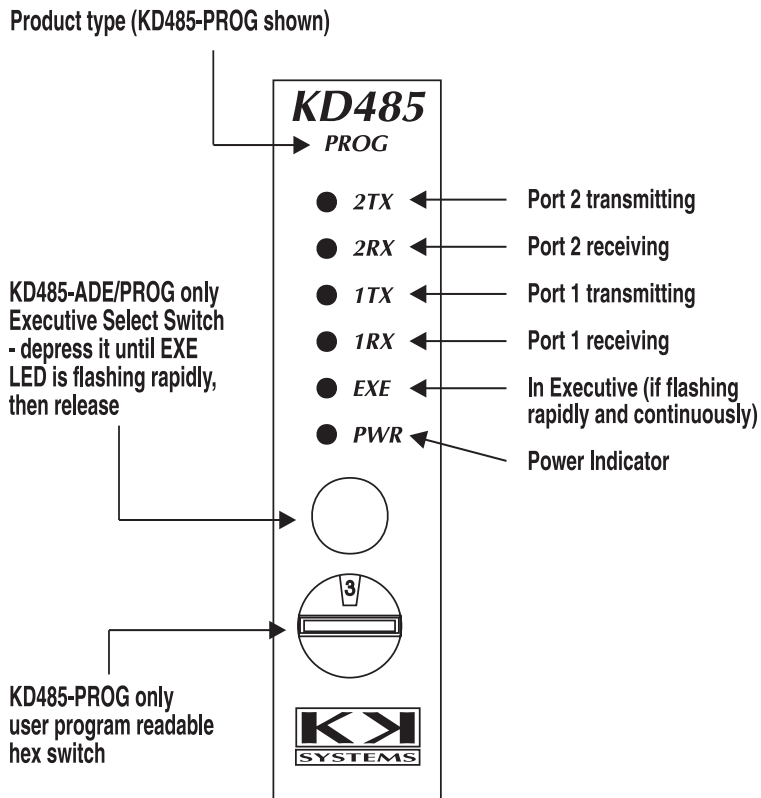
The KD485 is three-way isolated. This means that the two ports and the power supply input are all isolated from each other. This offers great installation flexibility while avoiding ground loops.

The KD485 is powered by a supply in the range +7V to +35V, 1-2 watts (typical). This is a high efficiency switching power supply which draws nearly constant power over the input voltage range and ensures very low power dissipation in the KD485.



The constant power property of the KD485 may require a larger power supply to be used. Please see the **Installation** chapter for details.

Front Panel Description



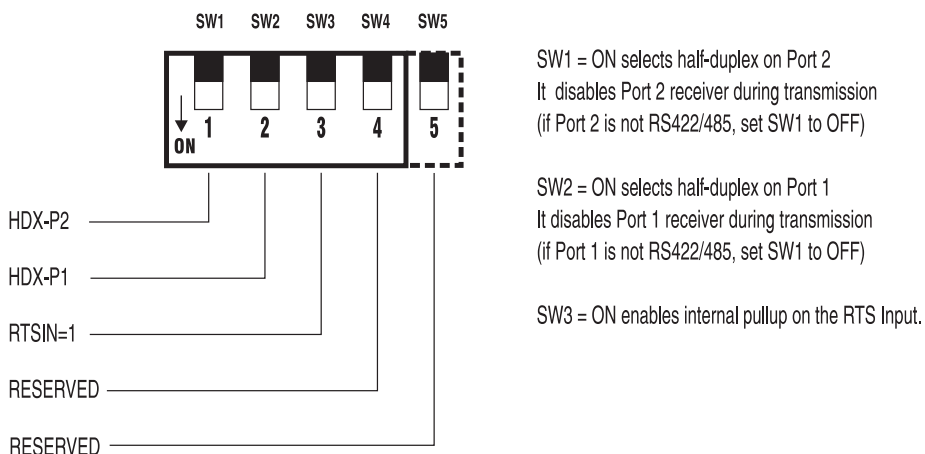
All KD485 versions feature four LEDs which indicate data flow on the two ports.

The KD485-ADE has an additional push-button switch for selecting the Executive (configuration) mode, and an “EXE” LED to indicate this mode, and other conditions.

The KD485-PROG has an additional 16-position rotary hex switch which is user program readable.

DIP Switch

RS485 Mode Switch



This switch is fitted on all KD485 versions; however, its function is presently limited to versions which have at least one RS422/485 port. Switches 4 and 5 are reserved for new features and separate documentation will be included if they are used.

- SW1** Controls if Port 2 receiver is enabled during transmission out of Port 2. This switch is relevant only if Port 2 is RS422/485 and is externally wired for 2-wire RS485 operation. Then, reception of own transmission is usually not desired (i.e. half-duplex operation is required) and therefore SW1=ON. There are special cases where one may want to receive everything in which case SW1=OFF. In general, **if Port 2 is not RS485, set SW1 to OFF.** SW1 replaces the "HDX" jumper in the previous (grey box) KD485 model.
- SW2** This is as SW1 but operates on Port 1. Since only the KD485-ADE/-PROG can control the Port 1 RS422/485 driver, this switch is relevant only if the KD485-ADE/-PROG is used as a converter from one RS485 bus to another RS485 bus. The most common such application is attaching 2-wire devices to an existing 4-wire bus, in which case reception of own transmission is not desired on either port, and SW1=SW2=ON.
- SW3** This merely enables an internal pullup on the RTS Input. The RTS Input is normally used only if Port 2 is RS422/485, to control its driver. Typically, with the KD485-STD converting from RS232 to RS422 when a permanently enabled Port 2 driver is desired, then SW3=ON. The RTS Input is also user program readable in the KD485-PROG. SW3 replaces the "RTS=1" jumper in the previous KD485 model.
- SW4,5** Currently unused.

Standard KD485 Versions

The KD485 is available in three standard versions:

1. KD485-STD Isolated Converter; standard model

This entry level product has no CPU. It is an interface converter/isolator only and is baud rate and character format independent.

In the default RS232-RS422/485 version, it enables RS232 devices to communicate with RS422 or 2/4-wire RS485 devices. When tri-state operation is required on Port 2, the RS232 host must provide RTS control.

The KD485-STD can also be used as a Master on a 4-wire RS485 system.

2. KD485-ADE Isolated Converter; with Auto Driver Enable

This inserts a CPU (with two UARTs) into the data path. The primary purpose of this version is for RS232 to RS485 conversion; it can however be used for character format / baud rate conversion also.

The two ports are individually configurable 30-115200 baud. Port 1 can be placed into an "Executive" configuration mode and the KD485-ADE can be configured using any ASCII terminal ranging from small hand-held devices (e.g. a Psion Organiser XP) to IBM PCs. A Windows-based pull-down-menu program can also be used. The configuration is stored in an EEPROM. The configuration software is included on a diskette.

The KD485-ADE can be user-configured to operate in one of the following modes:

- 0** Applicable to all port types. **Character format conversion** (baud rate, bits/word etc) only. The two ports can be individually configured with different settings. No auto driver enable for RS485.
- 1** Applicable only if Port 2 is RS422/485. Operation as for Mode 0, plus **auto driver enable**. This controls the Port 2 RS485 driver according to the presence of data arriving at Port 1, and eliminates the need for the Port 1 system to provide driver control. This mode can also be configured to work the other way, i.e. to control the Port 1 RS485 driver according to data arriving at Port 2, to form a bidirectional RS485-RS485 converter with auto driver enable at both ends.
- 2** Applicable only if Port 2 is RS422/485. This is an RS485 **addressable adapter**. This supports applications where multiple non-RS485 (i.e. RS232 or 20mA Loop) devices need to be multi-dropped on a 2-wire or 4-wire RS485 bus, but the devices themselves do not support address recognition, and/or do not have tri-state drivers. Devices which emit data only in response to a poll and devices which emit data continuously are both supported.

3. KD485-PROG Isolated Converter; user programmable in ANSI C

This does everything that KD485-ADE does and adds a large EEPROM, plus a means of uploading user-written programs into the EEPROM. User programs can be written in ANSI C, assembler, or other languages, and are uploaded (in Intel Hex) to the KD485-PROG using supplied software. The KD485-PROG is a very powerful high-performance programmable datacomms product suitable for highly complex projects.

A high quality ANSI C compiler is available separately and its multi-stream I/O and other features are supported with dedicated code in the KD485-PROG on-chip ROM. The result is a well-integrated system which greatly simplifies the creation of datacomms programs.

A MODBUS RTU SLAVE library is available which facilitates the rapid development of a MODBUS front end for a non-MODBUS instrument.

Custom Versions

Other KD485 versions are available. These include any combination of RS232, 422, 485 and 20mA loop interfaces, and custom software. Please contact Factory with your requirements.

Completely customised products, based on the KD485 or the 4-port PPC Programmable Protocol Converter, can also be supplied. Custom labelling and different physical layout (e.g. Eurocard) can be supplied.

Specification

Ports:	Two asynchronous ports, TX & RX signals only. XON/XOFF selectable.
Port parameters:	KD485-STD: 0 to 115200 baud, all character formats. KD485-ADE/PROG: 30-115200 baud, none/even/odd parity, 7/8 data bits, 1/2 stop bits. 20mA Loop ports: 30-19200 baud.
Interface combinations:	Standard product: port 1 is RS232; port 2 is RS422/485. Any combination of RS232, RS422/485 or 20mA loop can be supplied.
RS232 interface:	Receiver threshold +1.5V typ. Hysteresis 500mV typ. Receiver input impedance 5k Ω typ. Transmitter output swing \pm 8V typ into 3k load.
RS422/485 interface:	Receiver threshold 200mV typ (differential). Hysteresis 70mV typ. Receiver input impedance 12k Ω min. Transmitter output swing 0 to +5V (no DC load); +2 to +3V (120 Ω ohm load).
20mA loop interface:	Input: LED, nominal drop 2V Output: open collector transistor, Vce(sat) < 2V 20mA current source: accuracy \pm 20%; no-load output voltage is approximately equal to KD485 supply voltage.
I/O delay:	STD: <100 μ s. ADE/PROG: approx 2ms (@9600 baud, Mode 1).
Power supply:	+7V to +35V DC. +7V to +26V DC if 20mA Loop ports are fitted. Input power is approximately constant at 1-2 watts (startup current 300-600mA) depending on model. At startup, the supply voltage must reach 7V within 1 sec otherwise the power supply will not function.
Isolation:	64V PK, tested at >1000V AC RMS, 1 second.
Environmental:	Operating temperature -25C to +50C. Storage temperature -40C to +70C. Relative humidity (operating and storage) 0 to 90%, non-condensing.
Ventilation:	Rail-mounted KD485 must have a 50mm gap above and below.
Compliance:	Emissions EN 61000-6-4:2007. Immunity EN 61000-4-2:2010. EMC Directive CE 2014/30/EU. ROHS / REACH compliant 2011/65/EU.
Dimensions:	29mm (W) x 114mm (H) x 97mm (L) approx. including screw terminals.

Changes from previous (1994-1998) KD485 version

A number of improvements have been made:

- New enclosure with removable screw terminals.
- Three-way isolation; previous model shared a ground between the power supply and Port 1.
- Wide input range switching power supply. Much reduced heat dissipation, especially at 24V input.
- Front-accessible push button switch and LED for KD485-ADE/-PROG configuration; on previous model these were side accessible.
- Front panel visible data flow indicator LEDs.
- Externally accessible DIP switch replaces the internal jumpers.
- Port 1 can be switch configured to disable its receiver during transmission (for RS485). On previous model the Port 1 receiver was always enabled.
- Port 1 is duplicated on an RJ-11 connector. This makes KD485-ADE/-PROG configuration more convenient and also brings out the KD485-PROG TTL-level DEBUG output port.
- 16-position user program readable switch on KD485-PROG.
- 20mA Loop option, with active output (20mA current source) available on either or both ports.
- Factory-fittable plug-in I/O cards enable unusual versions, e.g. KD485-ADE-20MA-422 (for conversion between 20mA loop and 2-wire RS485) to be available on a short lead time.
- 14V (bidirectional) varistor protection on RS232 ports; previous model was varistor protected on RS422/485 ports (Port 2) only.
- +5V low power output on Port 2 screw terminal; can be used for powering external RS485 pullup resistors.
- On the KD485-STD, the polarity of the RTS Input can be reversed (as a factory option) to support users whose RTS source is HIGH to receive and LOW to transmit.
- The KD485-PROG has a Real Time Clock option.
- KD485-PROG has a factory-option optocoupler (open-collector) user-addressable output, available if Port 1 is not RS422/485.
- No need for installer to open the KD485 enclosure



The following features should be examined carefully when replacing the previous model with the new model:

- Due to the constant power characteristics of the new KD485 power supply, together with the low startup voltage (typically 4V) the unit draws a current which can reach 300-600mA (depending on model and port types) momentarily as the supply voltage rises through 4V. This may necessitate the use of a larger power supply than the previous KD485 required.
- The KD485-STD no longer has a default pullup on the RTS Input. If you are not driving RTS externally, you must set SW3=ON to obtain a HIGH level on the RTS Input.
- Only 35mm DIN rails are supported. Asymmetric rails are not.


KD485-ADE/PROG Mode 2 Detail Description

The KD485 monitors the RS485 bus for an (optional) **lead-in byte**, followed by a matching **address byte**. Upon address match, any subsequent data received from the RS485 bus is passed to the attached RS232/422 device on port 1; this continues until a user-specified **timeout** has passed since the last data byte transferred. These three parameters are configured with the **AD** command (see the **Executive** chapter), or with KDCFG under "Mode 1 Configuration".

When all RS485 data has been passed on to the attached device, and following the expiration of the timeout, the KD485 assumes that there is some data in the attached device's (port 1) input queue. In other words, it assumes that the attached device *has already started* to generate its response. The timeout must therefore be long enough to ensure this!

At the expiration of the timeout, the KD485 transfers any data from the attached device's queue back to the RS485 bus. This continues until there is no more data to transfer. If the timeout is too short, there will be no data *to start with*, and nothing will be transferred.

Mode 2 is also designed to be suitable for multi-dropping RS232 devices which generate data *continuously*, i.e. without being prompted. The KD485 simply returns everything which the device has generated since the last time the KD485 was polled. The size of the KD485 RX queue is usually sufficient, except when the KD485 is polled very infrequently.

 When Mode 2 is used with a 2-wire RS485 bus, ensure that SW1=ON. This inhibits the reception of own data back from Port 1.

Mode 2 supports two special features: an address of **255** is treated as a "broadcast" address and will always match. This is useful if one needs to send a message to all devices simultaneously. Of course, they must not respond to this message, otherwise bus contention will occur!

An address of **254** instructs the KD485 to return the number of bytes in the KD485's Port 1 RX queue, as a 7-character ASCII string in the form ":ddddd<CR>". The initial ":" and the final <CR> characters are useful delimiters. All five digits are always returned, with leading zeroes if necessary. This can be useful if you have a device which emits unsolicited data, and you need to periodically check how much data it has emitted. If there is more than one Mode 2 KD485 on the RS485 bus, *you need to use the lead-in byte* as an address byte otherwise bus contention will occur when all the KD485s respond!

Choosing Address and Lead-In bytes: If these values accidentally appear in the data on the RS485 bus (data flowing in *either* direction) this can cause false address recognition. This is a standard problem with RS485 and, apart from specialised solutions, it cannot be avoided. There are however two things you can do:

If possible, choose the message content and/or the addressing so the address byte (or the sequential combination of lead-in and address byte, if a lead-in byte is used) cannot appear within it.

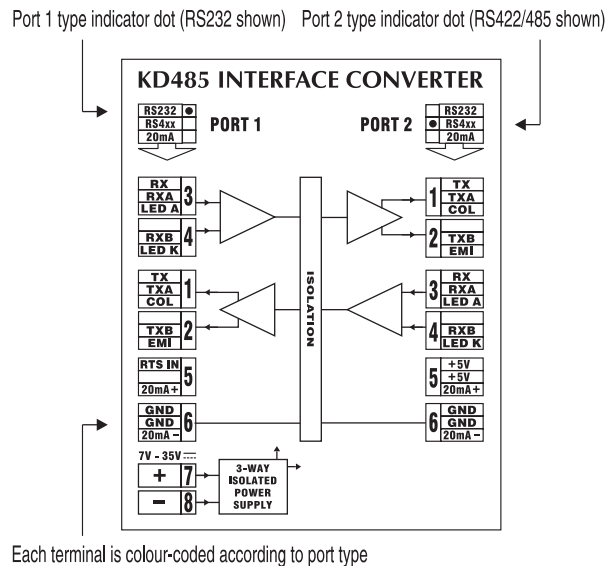
Use **4-wire** RS485 in preference to 2-wire RS485, for the multidrop bus. With a 4-wire system, the KD485 slaves do not see each others' responses and this eliminates the possibility of false addressing caused by the data being *returned*. However, you still need to ensure that the (optional) data *transmitted by the master* after the addressing byte(s) cannot cause false addressing of other slaves.

Installation

For detail information on the various port types (RS232, RS422/485, 20mA loop) and DIP switch configuration please also refer to the **Ports and Connections** section.

Product and Interface Type identification

The KD485 type, i.e. -STD, -ADE, -PROG is marked on the front label. The type of each of the two ports is marked on the side label and these are colour-coded to show which terminals are applicable to that port type:

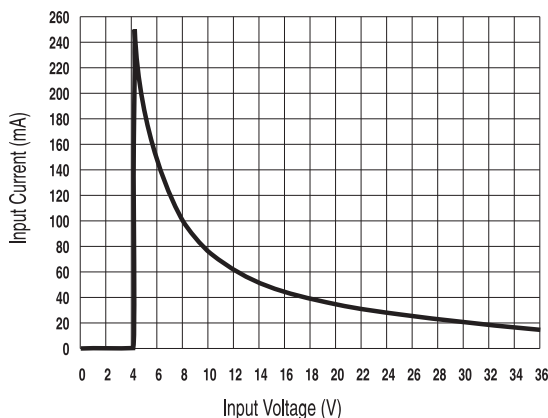


DC Power Requirements

The KD485 requires a regulated or unregulated DC power supply whose voltage must at all times be in the range +7V to +35V.

For efficiency, the KD485 uses a switching power supply. Therefore, **the current consumption varies with the supply voltage**, such that the power (in watts) remains fairly constant over the supply voltage range. The following graph shows the relationship for a typical KD485-ADE:

KD485-ADE Typical Power Input Characteristics



The peak current which occurs around 4V can reach 600mA on some models.

Input must reach the startup voltage (typ. 4V, guaranteed 7V) in less than 1 second otherwise the power supply will not start up.

It can be seen that to ensure reliable startup the power supply must be capable of delivering up to 600mA. If you have a choice, a +9V or +12V regulated supply is recommended. There is no advantage in using

higher voltages; the power supply needs to deliver (during startup) up to 600mA so if a higher voltage power supply is used, it needs to be a larger-wattage unit.

KD485-STD Configuration

The only potential configuration is on the DIP switch. The most common configurations are:

KD485-STD (RS232 to RS422): SW1,SW2=OFF, SW3=ON

KD485-STD (RS232 to 2-wire RS485, RTS control provided by RS232 device): SW1=ON, SW2,SW3=OFF

With 2-wire RS485 (on Port 2) applications, remember that the RS232 host must control the RTS signal to control the KD485 tri-state driver. Port 1 must therefore be RS232.

If the KD485-STD is a Master on a 4-wire RS485 system, or is used in a simple RS422 point-to-point link, leave RTS unconnected and permanently enable the Port 2 RS485 driver with SW3=ON.

Finally, with the power disconnected, connect the Port 1 and Port 2 signals and the power supply.

KD485-ADE Configuration

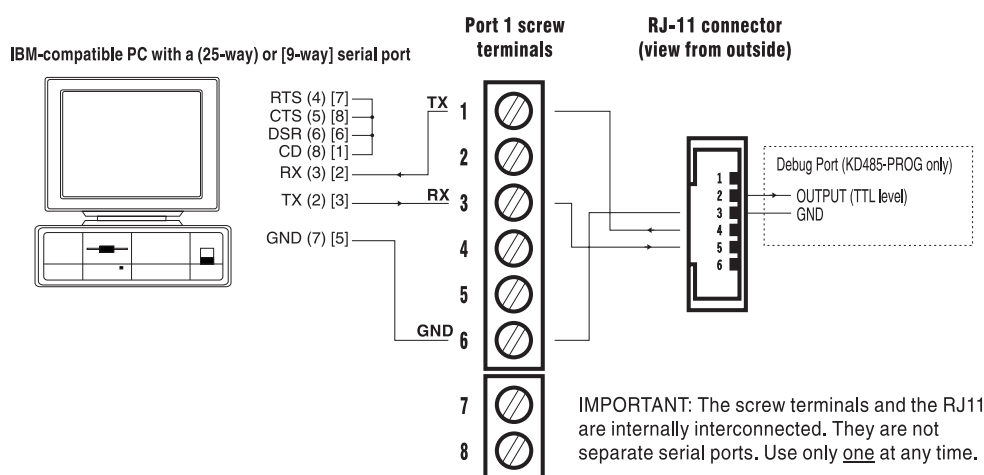
In addition to the DIP switch settings, **this version must be electronically configured before it can be used**, via Port 1. This involves mainly the port parameters, but you must also specify which of the several built-in programs (“Modes”) you wish to run. The configuration can be done in one of two ways:

- 1 With a “dumb” terminal, set to 9600 baud, 8 bits/word, no parity, 1 stop bit. This can be a PC-based terminal emulator; a simple utility called TERM.EXE is provided on the diskette. This uses a simple command-line interface where the KD485 provides a “>” prompt. Refer to the **Executive** chapter for details.
- 2 With a Windows-based program, called KDCFG.EXE. This is provided on the diskette. This method offers access to some additional (although very rarely used) KD485-ADE features.

Configuration is done via Port 1 which must be set to the Executive Mode, indicated by the “EXE” LED flashing rapidly and continuously. To enter Executive Mode, press the front panel pushbutton switch until the “EXE” LED starts flashing, then release it. A few seconds later, the LED will resume continuous flashing and you can start the configuration software.

Port 1 can be accessed either via the Port 1 screw terminals (numbered 1-6 on the 8-way terminal block), or via the RJ-11 connector. Do not connect to both simultaneously.

Configuration Connections for Port 1 = RS232



The supplied KDCFG.EXE and TERM.EXE programs require only the TX,RX,GND connections. The other connections shows at the PC end are sometimes needed with other terminal emulation programs.

The Psion Organiser pin numbers are those at the end of the Psion cable which comes as a part of the Psion Comms Link accessory. To use the Organiser as a dumb terminal, select COMMS, then select TERM. Configure to 9600, 8 bits/word, no parity, 1 stop bit.

Next, set up the DIP switches as applicable. The most common configuration is:

KD485-ADE (RS232 to 2-wire RS485): SW1=ON, SW2=OFF, SW3=ON)

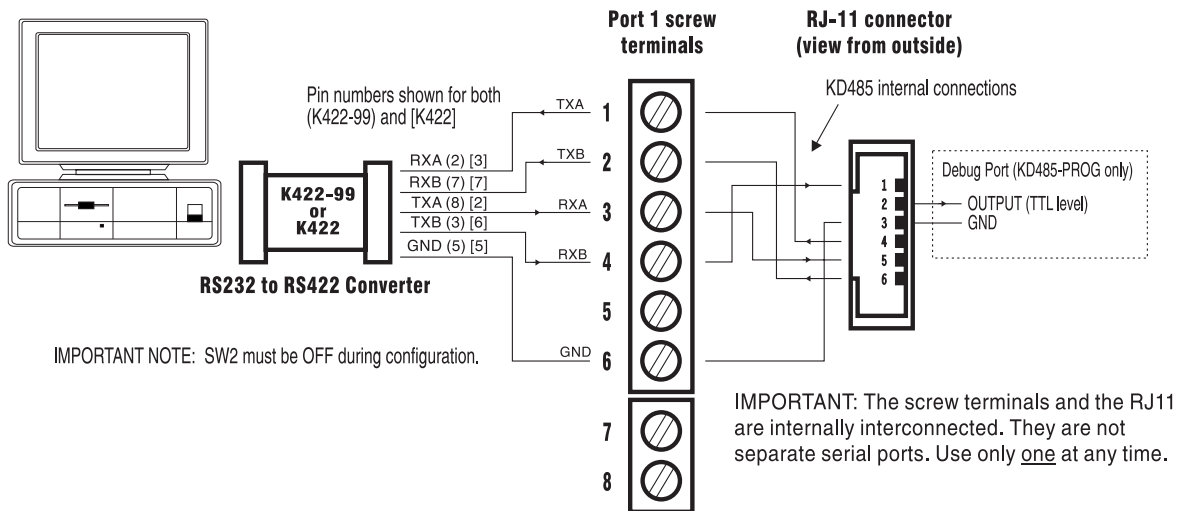
Finally, with the power disconnected, connect the Port 1 and Port 2 signals and the power supply.

If using the Mode 0 program and Port 2 is RS422/485, RTS input must be externally driven, or be tied to a RS232 HIGH level with SW3=ON.

If Port 1 is RS422/485 then a suitable RS232-RS422 interface converter must be used:

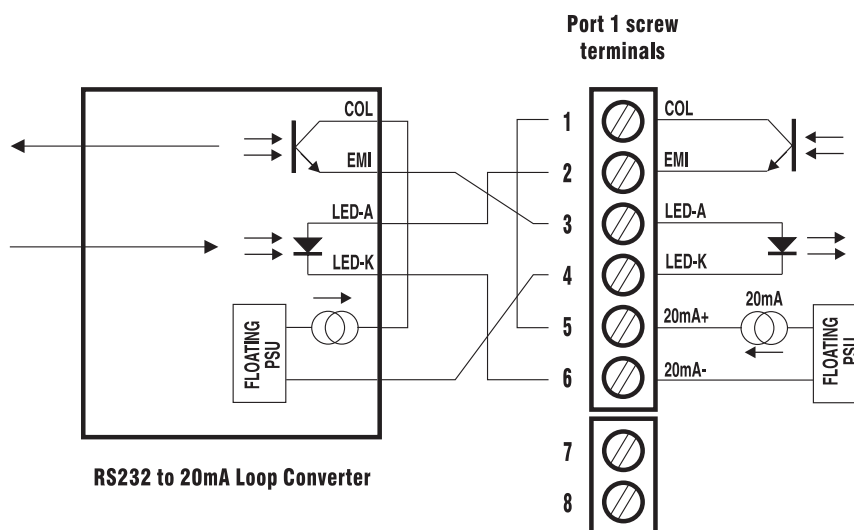
Configuration Connections for Port 1 = RS422/485

IBM-compatible PC with a (9-way) or [25-way] serial port



If Port 1 is 20mA Loop then a RS232 to 20mA converter must be used:

Configuration Connections for Port 1 = 20mA Loop



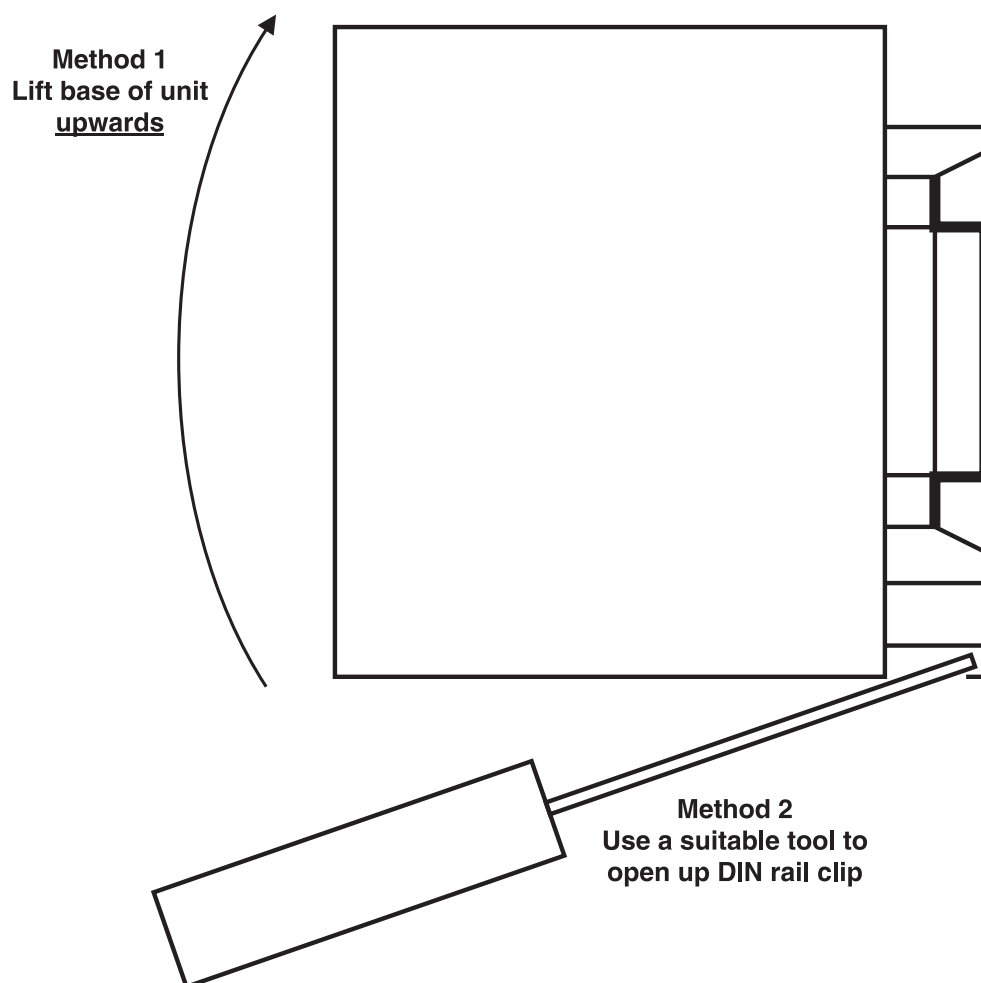
The RJ-11 connector cannot be used for the 20mA configuration since, due to insufficient pins, not all of the 20mA signals come out on it.

Fitting KD485 on DIN Rail

To fit the KD485 to the rail, simply clip it on.

To remove the KD485 from the rail, lift the front end **upwards**. However, this may not be successful, particularly with the taller and thicker 35mm rails, in which case use a suitable tool (e.g. a flat screwdriver) to gently (very little force is required) open up the DIN rail clip as shown below.

Removing unit from DIN rail



KD485-PROG Configuration

If a KD485-PROG is used to run one of the -ADE built-in programs, the installation is as per the -ADE product. If the KD485-PROG is used to run a user program, this program must first be uploaded to the KD485-PROG.

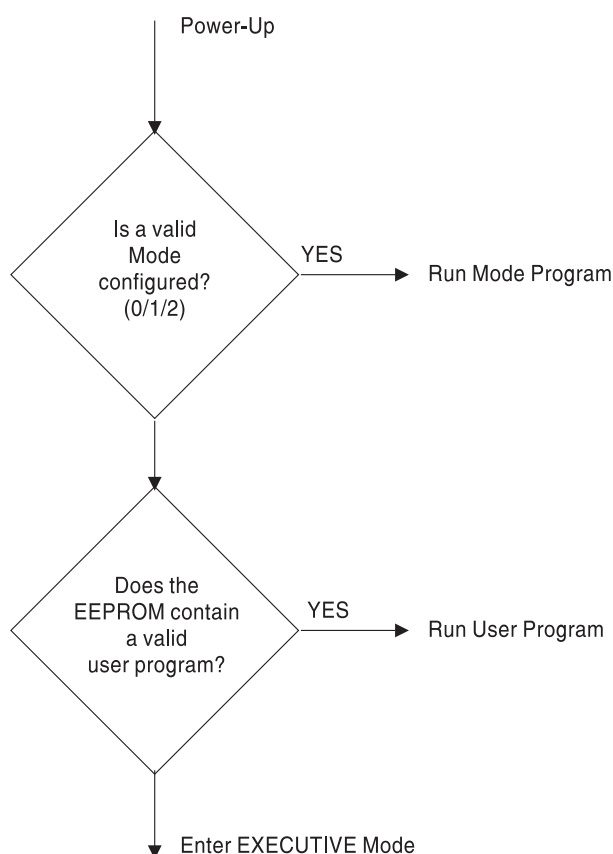
For information on how to produce such custom programs, refer to the **C Introduction** and **C Reference** chapters. The supplied hello.c and test.c are good examples to get you started.

User programs can be uploaded to the KD485-PROG in one of two ways:

- 1 Using TERM.EXE and the UF (Upload File) command. Refer to the **Executive** chapter. Any other terminal emulator which has an ASCII file upload facility can be used; note however that most of the modem-oriented comms programs (e.g. Procomm) will abort a file transfer if the PC's CD (carrier detect) input goes LOW and you may therefore need to wire this input to a HIGH level.
- 2 Using the Windows-based KDCFG.EXE program. This has a self-explanatory function for file upload.

The KD485-PROG executes programs in the following order:

Therefore, to execute your program, you need to set Mode to something invalid. With TERM, set Mode to



e.g. 100 (md=100). With KDCFG, there is a specific MODE=OFF option.

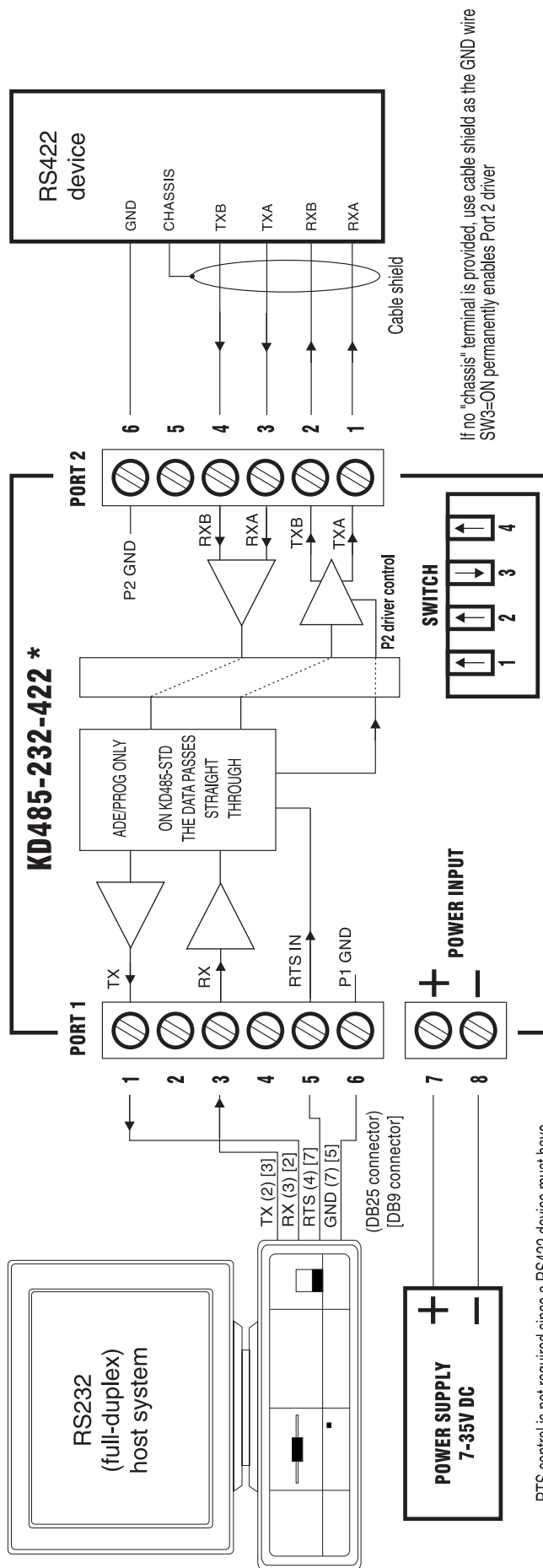


For program uploading under Windows 9x or Windows NT, you may need the latest version of KDCFG.EXE which can be downloaded from **www.kksystems.com**.

Wiring Examples

The following pages contain examples of typical applications.

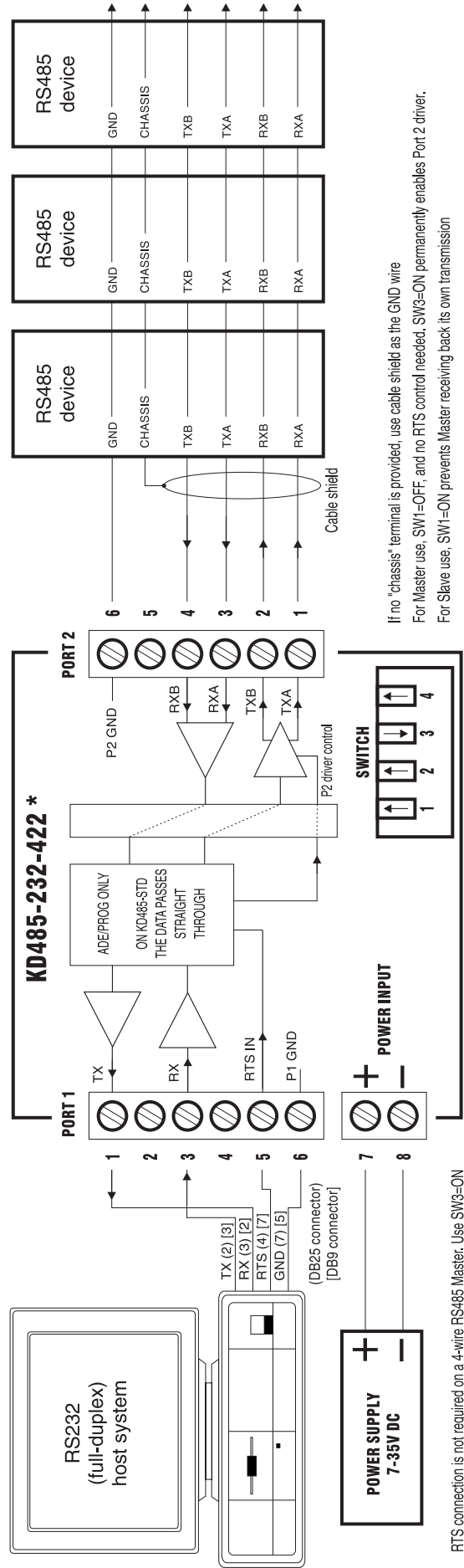
RS232 to RS422 (full-duplex, point-to-point) conversion



* The standard KD485 is RS232-RS422/485 therefore it is not necessary to specify the -232-422 suffix when ordering

RTS control is not required since a RS422 device must have its driver permanently enabled.
RTS connection above can be omitted if SW3 is ON as shown

RS232 to 4-wire RS485 (half-duplex, multidrop) conversion

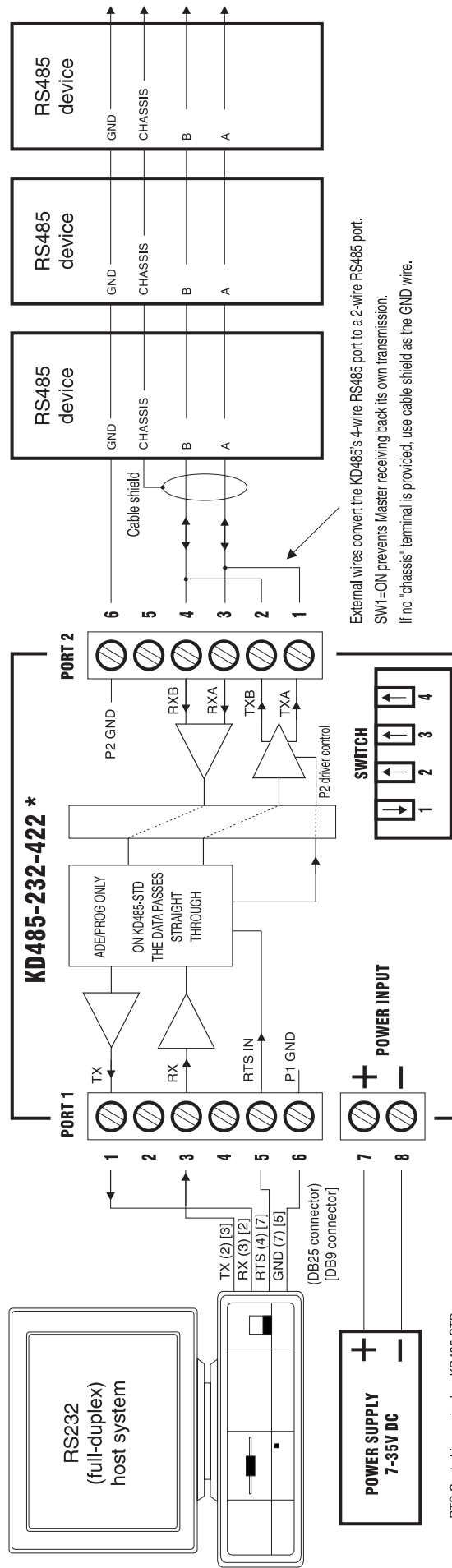


If no "chassis" terminal is provided, use cable shield as the GND wire
 For Master use, SW1=OFF, and no RTS control needed, SW3=ON permanently enables Port 2 driver.
 For Slave use, SW1=ON prevents Master receiving back its own transmission

* The standard KD485 is RS232-RS422/485 therefore it is not necessary to specify the -232-422 suffix when ordering

RTS connection is not required on a 4-wire RS485 Master. Use SW3=ON

RS232 to 2-wire RS485 (half-duplex, multidrop) conversion

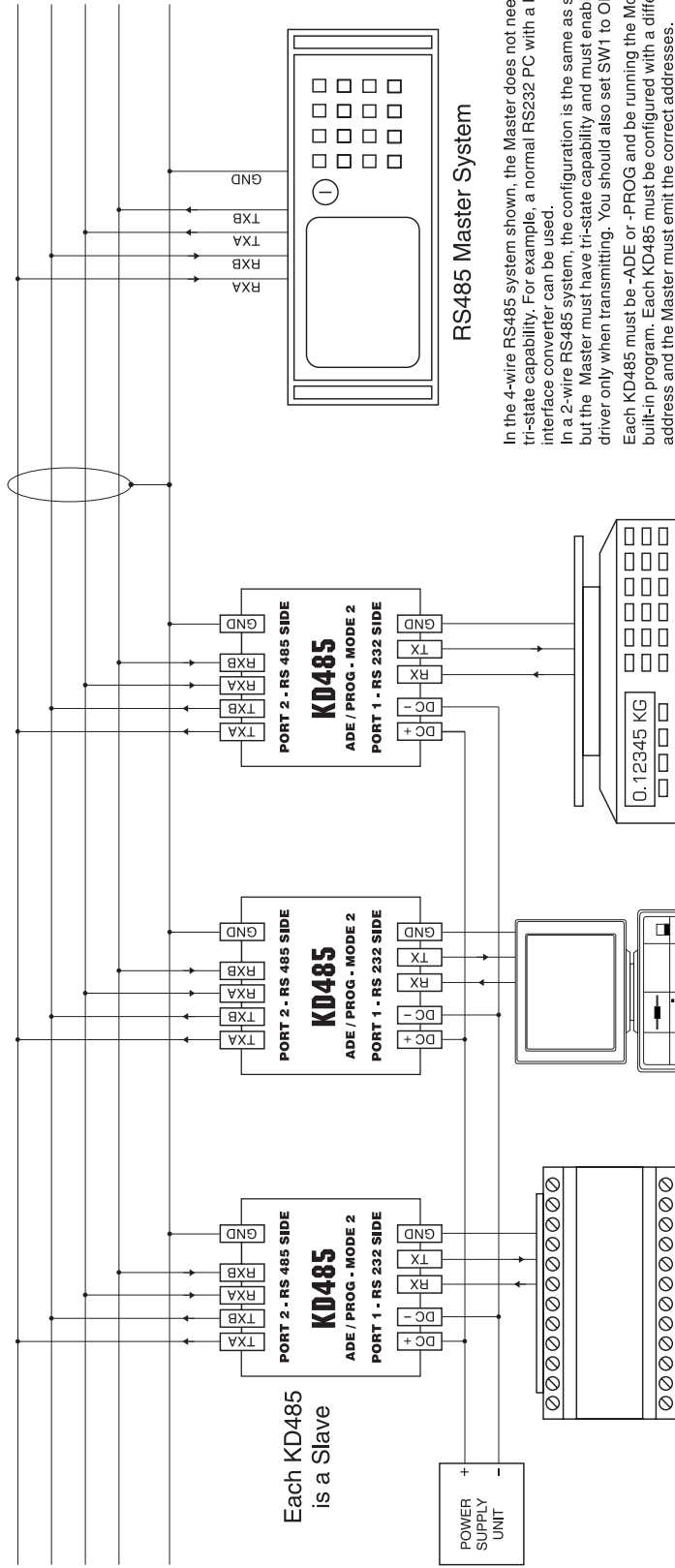


RTS Control is required on KD485-STD.
RTS connection above can be omitted on KD485-ADE/PROG

* The standard KD485 is RS232-RS422/485 therefore it is not necessary to specify the -232-422 suffix when ordering

External wires convert the KD485's 4-wire RS485 port to a 2-wire RS485 port.
SW1=ON prevents Master receiving back its own transmission.
If no 'chassis' terminal is provided, use cable shield as the GND wire.

KD485-ADE/PROG as an Addressable Adapter (Mode 2 program)



In the 4-wire RS485 system shown, the Master does not need tri-state capability. For example, a normal RS232 PC with a K422 interface converter can be used.

In a 2-wire RS485 system, the configuration is the same as shown but the Master must have tri-state capability and must enable its driver only when transmitting. You should also set SW1 to ON.

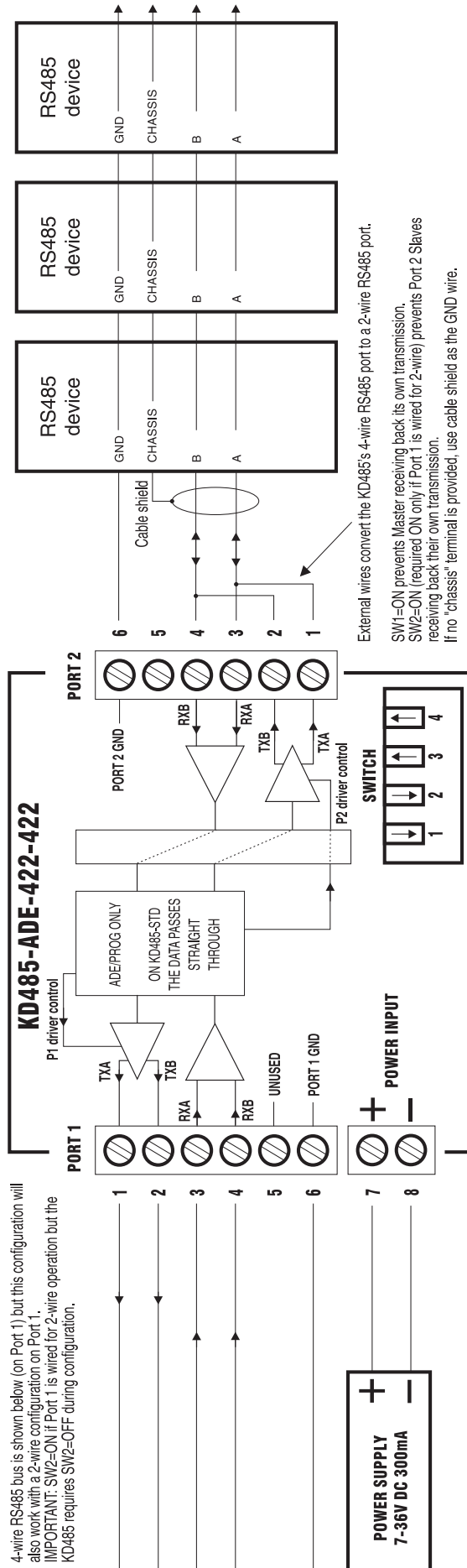
Each KD485 must be -ADE or -PROG and be running the Mode 2 built-in program. Each KD485 must be configured with a different address and the Master must emit the correct addresses.

This configuration is equally applicable to **RS422** devices which do not support multi-drop operation, using KD485-ADE-422-422.

RS232 devices with no built-in addressing or multi-drop support

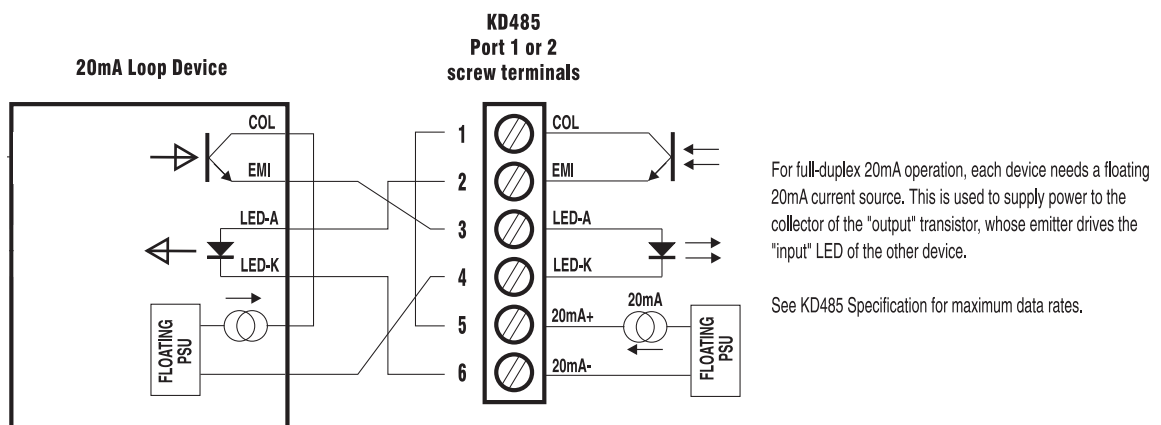
4-wire (or 2-wire) RS485 to 2-wire RS485 (half-duplex, multidrop) bus converter / repeater

4-wire RS485 bus is shown below (on Port 1) but this configuration will also work with a 2-wire configuration on Port 1.
IMPORTANT: SW2=ON if Port 1 is wired for 2-wire operation but the KD485 requires SW2=OFF during configuration.



IMPORTANT NOTE: If operating as 2-wire to 2-wire repeater, the KD485-ADE-422-422 requires a minimum gap of 4 character periods between messages, to ensure reliable line turn-around.

20mA Loop - General Connections



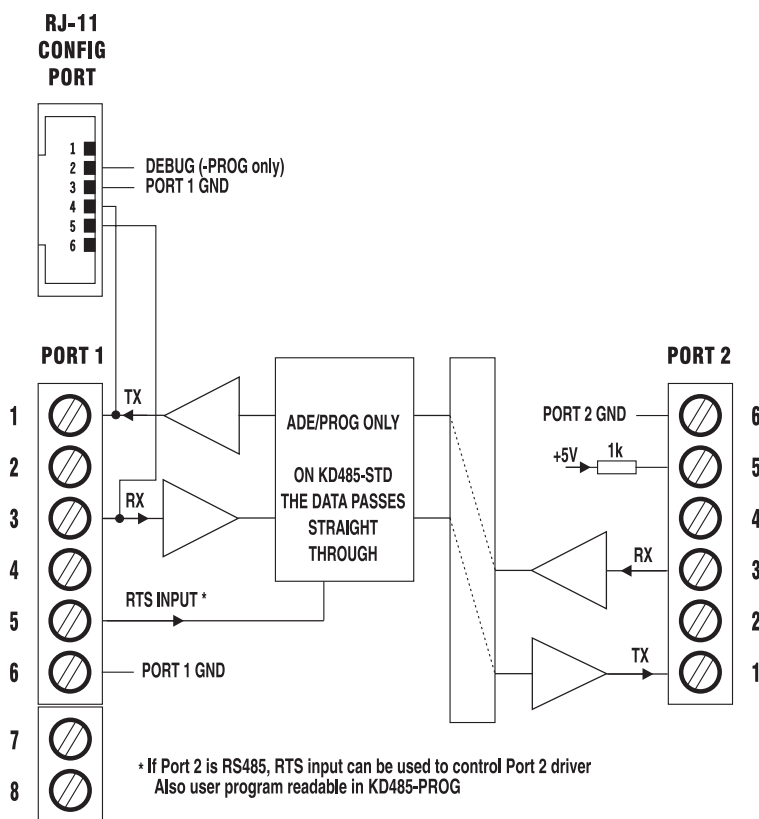
Ports & Connections

The KD485 has two ports, called Port 1 and Port 2. Each of these can be factory configured as RS232, RS422/485, or 20mA loop. The port types fitted to a particular KD485 are marked on its side label.

Port 1 is duplicated on an RJ-11 connector. The RJ-11 is not a third serial port; however it does carry a DEBUG output on the KD485-PROG.

The following text describes each of the three possible port types:

RS232 Ports



The RS232 port supports TX, RX signals only.

On Port 1 only, there is an additional input signal, RTS IN, which can be used to control an RS485 driver on Port 2. This operating mode is applicable only to the KD485-STD although the KD485-ADE Mode 0 program offers a similar function. On the KD485-PROG this input is user program readable.

RS232 Ports - Detail Description of Terminals

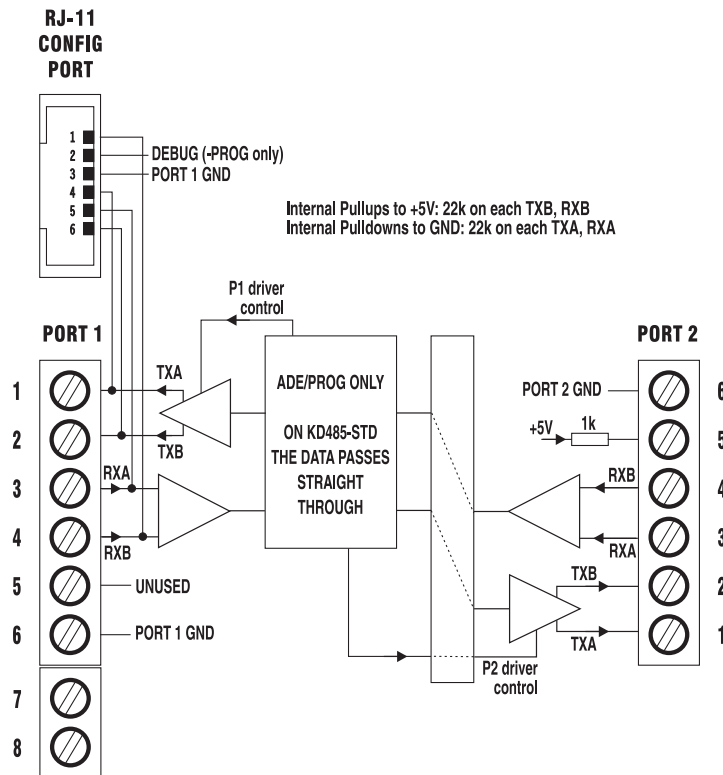
The numbers in brackets are the markings on the KD485 terminals.

- TX (1) "Transmit Data" output.
ADE/PROG: when XON/XOFF is enabled as an RX handshake, XON/XOFF characters are automatically transmitted from the TX terminal to control the data flow into the RX input.
- RX (3) "Receive Data" input.
ADE/PROG: when XON/XOFF is enabled as a TX handshake, any XON/XOFF characters arriving at this pin will be stripped by the KD485 and will control the data flow from the TX output.
- RTS IN (5) "RTS" input. On the KD485-STD, and on KD485-ADE Mode 0, this signal directly controls the tri-state state of the Port 2 driver if Port 2 is RS485. In other cases this signal is ignored. On the KD485-PROG this signal is software-readable.

GND (6)

This signal must always be connected. This combines both Protective Ground and Signal Ground functions.

RS422/485 Ports



This is a multi-purpose port which supports the following operating modes:

- **RS422.** This is a four-wire (plus ground) interface whose driver is permanently enabled.
- **4-wire RS485.** This is a four-wire (plus ground) interface whose driver can be enabled or disabled; this is called tri-state capability. This interface is often incorrectly called “RS422”.
- **2-wire RS485.** This is a two-wire (plus ground) interface whose driver must have tri-state capability.

The operation of the ports depends on whether it is Port 1 or Port 2, and on the KD485 version. The following notes cover the configuration of the most common product versions:

	STD Port 1	STD Port 2	ADE/PROG Port 1	ADE/PROG Port 2
RS422	SW2=OFF	SW1=OFF SW3=ON	SW2=OFF	Select Mode 0 program. SW1=OFF
4-wire RS485 **	Not supported	Requires RTS Control. Port 1 must be RS232. SW1=ON.	Select Mode 1 program & enable Port 1 driver to be controlled. * SW2=ON.	Select Mode 1 program. SW1=ON.
2-wire RS485	Not supported	Requires RTS Control. Port 1 must be RS232. SW1=ON. Interconnect TXA-RXA & TXB-RXB.	Select Mode 1 program & enable Port 1 driver to be “controlled”. * SW2=ON. Interconnect TXA-RXA & TXB-RXB.	Select Mode 1 program. SW1=ON. Interconnect TXA-RXA & TXB-RXB.

* This configuration option is available only via KDCFG.EXE. It cannot be done with a dumb terminal.

** The Master on a 4-wire RS485 bus can be an RS422 device because driver can be permanently enabled.

RS422/485 Ports - difference between RS422 and RS485?

RS422 and RS485 drivers and receivers have identical electrical characteristics. Both systems transmit each signal with two wires. Each signal is driven with a differential driver, and received with a differential receiver. Both of the driver outputs swing between 0V and +5V.

The main difference between the two systems is that while RS422 is a four-wire system suitable only for point-to-point use, the driver in a RS485 system has tri-state capability (its output can be disabled) which allows multiple transmitters to be connected to the same bus. RS485 thus supports “multi-drop” operation. In multi-drop systems there is always one device which is permanently a “master” and which periodically polls (sends messages to, requests data from) the “slaves”. A slave never initiates a communication.

There can be more than one Master (for fault tolerance, or other reasons) but separate precautions must exist to ensure only one can be driving the bus at any one time.

Furthermore, RS485 exists in two versions: 4-wire and 2-wire. The sole advantage of a 2-wire system is that it uses only two wires, (plus the ground connection, or the cable shield) and is thus cheaper to install.

The main advantage of a 4-wire system is more subtle: because the master is driving a pair of wires which no other device may drive, the master’s driver does not need tri-state capability. Many systems do not have tri-state capability, usually because their software was not written for multi-drop operation. The other advantage of a 4-wire system is that it is theoretically possible for the slave to respond before the master has finished transmitting its poll, without risk of bus contention, but no properly designed system relies on this.

Note that 4-wire RS485 with the driver left permanently enabled is the same as RS422.

RS422/485 Ports - Grounding

With an isolated device such as the KD485, an extra conductor is required for interconnecting the grounds between the devices connected to the RS485 bus. The cable shield can be used for this purpose.



Although it is not unusual to find products which have the ground connection missing, you can get away with this only where the common mode voltage between all the devices attached to the *same* cable will never exceed the RS422/485 common mode voltage range which is typically -7V to +12V. This is difficult to guarantee in practice; this is why the KD485 is an isolated converter with a GND connection!

RS422/485 Ports - A/B Terminal Markings

The standard specifies that the terminals should be labelled “A” and “B”. It also specifies their polarity: during the inter-character space (i.e. during the one or more stop bits which follow each character in asynchronous data) A is at ground potential and B is at +5V. Many manufacturers mis-label the terminals; some are reversed while others are marked + and – .

RS422/485 Ports - Detail Description of Terminals

The numbers in brackets are the markings on the KD485 terminals.

- | | |
|------------------|---|
| TXA (1), TXB (2) | “Transmit Data” output.
ADE/PROG: if port 2 is used as a RS422 port, and when XON/XOFF is enabled as an RX handshake, XON/XOFF characters are automatically transmitted from TX to control the data flow into the RX input. |
| RXA (3), RXB (4) | “Receive Data” input.
ADE/PROG: when XON/XOFF is enabled as a TX handshake, any XON/XOFF characters arriving at this pin will be stripped by port 2 and will control the data transmitted by the TX output. |
| TEST OUT (5) | This pin is internally connected, via a 1k resistor, to an internal DC +5V supply referenced to pin 6. It is provided mainly for test purposes, although current (limited only by the resistor) may be drawn from it for RS485 bus pullup purposes. |
| GND (6) | This is an isolated ground signal which should always be connected to the ground terminal(s) of the other RS422/485 equipment. |

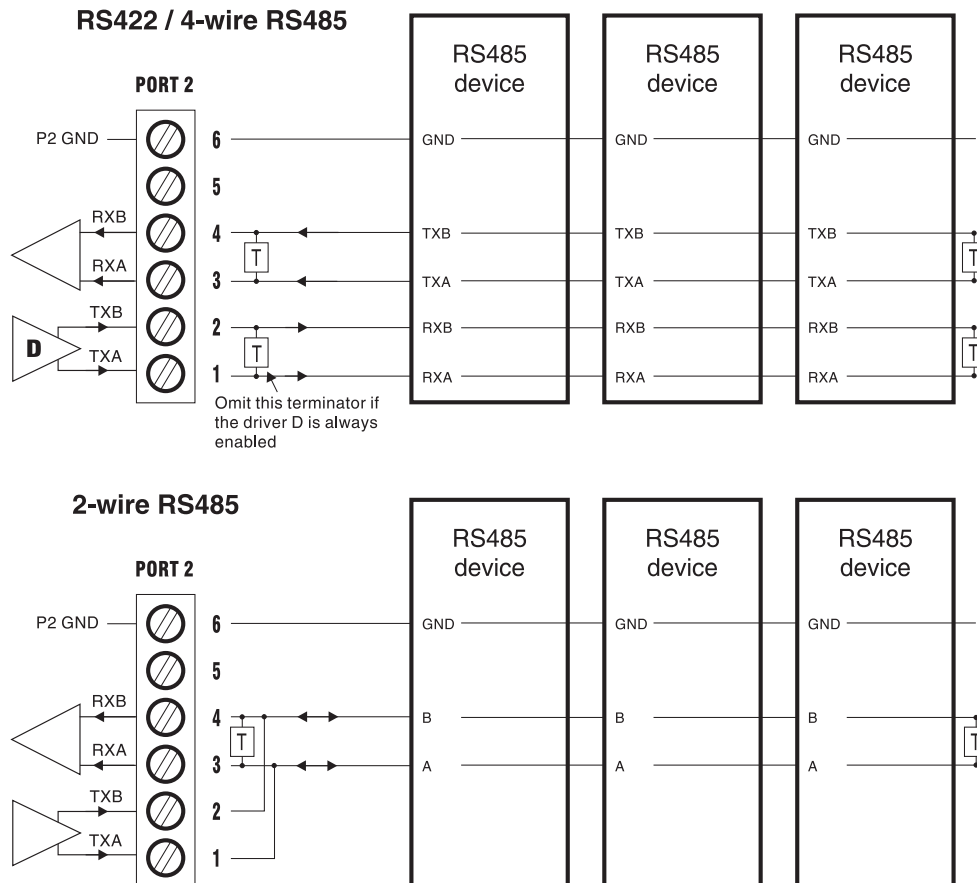
 Do not enable XON/XOFF on an RS485 port. Due to the half-duplex data flow, the handshake is unlikely to work as expected.

RS422/485 Ports - Terminators

Termination resistors are not normally required, because the KD485 uses controlled-slew-rate RS422/485 drivers and on cables below around 300m in length the reflections decay within the risetime of the signal.

When driving cables beyond 300m, you can connect a resistor equal to the cable impedance (typically 100-220 ohms) at each end of the cable (if RS485) or at the end farthest from the driver (if RS422).

Terminator Placement



To minimise DC loading on the driver(s) and to maximise the effectiveness of pullup/pulldown resistors (see below), a capacitor (e.g. 10nF) should be connected in series with any termination resistor.

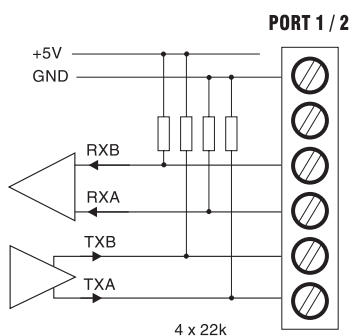
RS422/485 Ports - Bus Pullups

With multi-drop systems, it is important that when *none* of the connected devices is driving the bus, the level on the bus represents the inter-character space, i.e. B must be at a higher potential than A.

However, particularly where a number of devices are multi-dropped, driver leakage and other factors do sometimes cause A to float *above* B during the tri-state condition, in which case any attached receiver will see a permanent "start bit"; this is also known as a "break level". The effect of this depends on how well the firmware in the various devices is designed, and it can manifest itself as comms errors which are consistently cleared by a re-transmission of the data.

The KD485 contains internal pullup and pulldown resistors whose purpose is to ensure that when the bus is not being driven, B is above A:

Internal Pullup Resistors



However, these resistors may fail to work in the following cases:

- 1 Where another instrument also contains its own pullups/pulldowns and they are connected to the wrong (opposite) voltages. This rather fundamental error is not uncommon.
- 2 Where there is a terminator resistor (e.g. 100-200 ohms) connected directly across the 485 bus. The 22k resistors in the KD485 cannot develop a voltage across 100R which is sufficient to be a defined RS485 "1" or "0" level (200mV or more is needed). A solution might have been to use much lower value resistors but this would present an excessive load if many KD485 units were present on the bus.

To obtain a well defined level on the bus, one solution is to connect a capacitor (e.g. 10nF) in series with each of the terminator(s).

Another is to use external resistors to supplement those inside the KD485; this is where the +5V supply (available on pin 5, Port 2 only) is useful. Such external resistors are needed only in one place on a 485 bus; anywhere will do.

It is easy to verify, with a voltmeter, that B is above A when no communications are taking place. The voltages must be measured *between* A and B, or relative to the isolated ground.

20mA Loop Ports

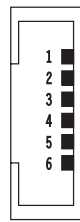
The 20mA port supports TX, RX signals only.

20mA Loop Ports - Detail Description of Terminals

The numbers in brackets are the markings on the KD485 terminals.

COL (1), EMI (2)	<p>"Transmit Data" output. These are the collector and emitter terminals of the output transistor which is used to switch the externally supplied 20mA loop current.</p> <p>ADE/PROG: when XON/XOFF is enabled as an RX handshake, XON/XOFF characters are automatically transmitted from the TX terminal to control the data flow into the RX input.</p>
LED-A (3), LED-K (4)	<p>"Receive Data" input. These are the anode and cathode terminals of the LED which is used to receive the externally switched 20mA loop current.</p> <p>ADE/PROG: when XON/XOFF is enabled as a TX handshake, any XON/XOFF characters arriving at this pin will be stripped by the KD485 and will control the data flow from the TX output.</p>
20mA+ (5), 20mA- (6)	This is the output of the floating 20mA constant current generator.

RJ-11 CONFIG PORT

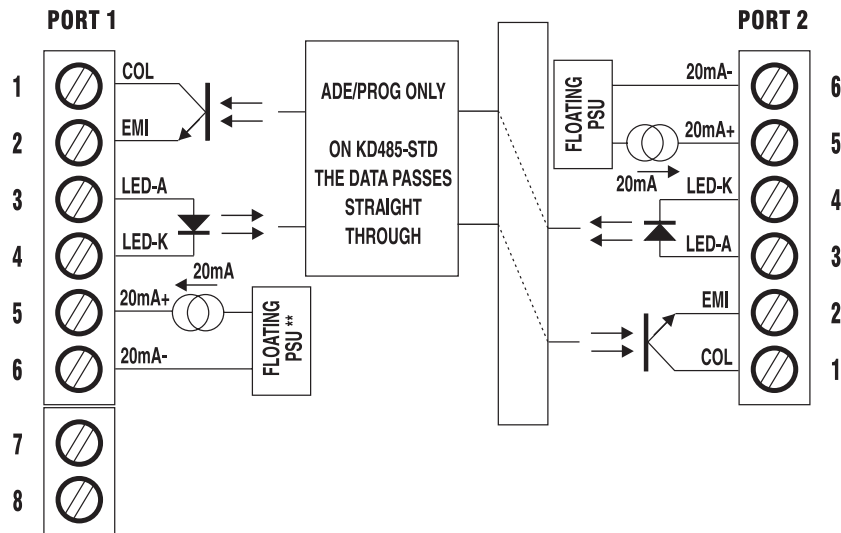



If Port 1 is 20mA, RJ-11 connector has no function except as a DEBUG output on the KD485-PROG

1 — DEBUG (-PROG only)
2 — PORT 1 GND

** Port 1 floating power supply is NOT isolated from the DEBUG output port!
Do NOT use both concurrently.

Floating power supply output voltage varies with KD485 supply voltage. Consult text for details.




 KD485-ADE/PROG only: The DEBUG output port is NOT isolated from the Port 1 20mA constant current generator. Therefore, if you are using a KD485-PROG whose Port 1 is a 20mA port, do not make simultaneous connections to Port 1 terminals 1,2 and to the DEBUG port (RJ-11 connector pins 2,3). The converter may otherwise be damaged. This requirement also applies to the KD485-ADE in so far as simultaneous connections to Port 1 terminals 1,2 and RJ-11 pins 2,3 are prohibited, even though the DEBUG port is not functional on the KD485-ADE.

Both the output transistor and the input LED are reverse polarity protected, with parallel diodes.

The internal supply voltage to the 20mA generator (i.e. its open circuit voltage) is not regulated. It is approximately equal to the KD485 supply voltage.

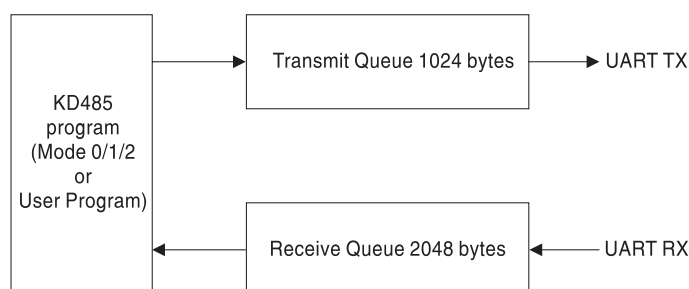
For example, if the KD485 is powered from +24V, the compliance of the 20mA current source is about 24V.

 If any KD485 ports are 20mA type, KD485 supply voltage is limited to **26V**, to limit the maximum possible internal heat dissipation.

KD485-ADE and KD485-PROG - General Data Flow Details

I/O Queues

Each KD485 port contains two queues: a transmit (TX) queue and a receive (RX) queue:



A user program in the KD485-PROG does not access the port hardware directly. Any I/O functions communicate only with the queues. The transfer of data between a queue and its port is performed as a background process (i.e. under interrupts) by the system firmware and is transparent to the user program.

Transmit Queues

A TX queue greatly enhances performance in both computational and datacomms areas, by allowing a user program to perform computations while previously-generated data is being transmitted under interrupts.

With 1024-byte TX queue, a user program could theoretically generate up to 1024 bytes of data even if the output device was BUSY throughout that time. However, to allow for insertion of XON/XOFF characters into the data stream and other reasons, the KD485 limits the filling-up of its TX queues to 1024-4 bytes.

Receive Queues

An RX queue is necessary to avoid data loss when handshaking the incoming data because most systems continue to transmit data for some time after being told to stop. An RX queue also permits the program to continue operations while data is arriving, and extract it from the queue when required.

When the KD485-ADE is used as an Addressable Adapter, the RX queue buffers any data received from the attached RS232 device, until it is retrieved by the RS485 bus Master.

Handshaking

The KD485 supports XON/XOFF handshakes only, and they are applicable to all port types except RS485. RS485 systems are usually half-duplex and do not have handshakes.

There are two types of handshakes: transmit (TX) handshakes and receive (RX) handshakes. A TX handshake signal is *received by* the KD485 and controls the transmit data flow *from the KD485*. An RX handshake signal is *transmitted by* the KD485 and controls the receive data flow *into the KD485*.

When the KD485 port is receiving data, if the free space in its RX queue falls below 512 bytes, it transmits an XOFF. When the space in the RX queue increases above 900 bytes the KD485 transmits an XON.



From the above it is clear that when a device which is sending data to the KD485 is told by the KD485 to stop transmitting, it must not transmit more than 512 additional bytes after that. A failure to meet this requirement will usually result in loss of data.

When transferring binary data, XON/XOFF characters could appear accidentally within the data. In such cases, an XON/XOFF handshake in which the XON/XOFFs flow in the same direction as the data must be disabled. Failure to observe this will result in a locked-up system, and/or in the XON/XOFF codes (11 and 13 hex) disappearing from your data. This may not be immediately apparent, and will certainly not be apparent if you are testing your system with printable data only, or only with short messages!

The Executive

The Executive is a special mode used to configure the KD485, using a serially-connected terminal or a PC. Only the “intelligent” KD485 versions (KD485-ADE and KD485-PROG) have the Executive. The KD485-STD requires no such configuration and does not have this feature.

The Executive is normally accessed using the supplied TERM.EXE terminal emulation program. A Windows-based configuration program, KDCFG.EXE, is also supplied and can be used as a more user-friendly alternative to the “dumb terminal” commands described here.

Basic Principles

The Executive is a command-line driven user interface, with a > prompt appearing after each line typed-in. Backspace editing is supported, but no cursor commands. Most KD485 installers will prefer to use the KDCFG.EXE graphical configuration utility.

The KD485 can enter the Executive at power-up, or following any attempt to run a program, if there is no program to run. This happens under the following conditions:

KD485-ADE: no Mode has been configured.

KD485-PROG: no Mode has been configured, and no valid user program is found.

Each configuration command consists of a two-letter command code, optionally followed by parameters. Unless specified otherwise, nothing is case-sensitive. No spaces are allowed as parameter separators; only commas may be used where shown. Optional items are shown in square [] brackets.

Each new setting is immediately stored in an EEPROM (non-volatile memory) and there is therefore no “save” command. To exit the Executive mode (i.e. to run whichever program has been selected or uploaded) you issue the RU (run) command, or interrupt the KD485 power.

The Executive always runs via Port 1 and always uses 9600,n,8,1. The terminal does not need to support any handshakes, although XON/XOFF should be enabled if possible.



The port 2 (RS485) driver is enabled while in the Executive. If port 2 is connected to a multidrop bus, ensure that no other device is trying to drive that bus while in the Executive.

The following section lists each command in the form of a definition, followed by an example.

Pn – Port Configuration

```
port=baudrate[,parity,bits/word,stopbits[,RXxon/xoff, TXxon/xoff]]
p2=9600,n,8,1,n,n
```

The following table lists valid values:

baud rates	30 37.5 50 75 100 110 134.5 150 300 600 1200 2000 2400 3600 4800 7200 9600 19200 38400 57600 115200
bits/word	7 8
parity	n e o (none/even/odd)
stop bits	1 2
RX XON/XOFF h/shake	y n (on/off)
TX XON/XOFF h/shake	y n (on/off)

Both ports have equal capabilities as far as this configuration is concerned. However, the handshakes should never be enabled on a port which is half-duplex.

Specifying the baud rate only (e.g. “p2=9600”) selects “n,8,1,n,n” for the other parameters. Specifying everything but the handshakes selects “n,n” for those.


Port 2 is *immediately* initialised to the new parameters, but port 1 remains at 9600,n,8,1 while in the Executive, otherwise the terminal would have to be reconfigured, too!

MD – KD485 Mode

```
mode=modenumber (0,1,2)
md=1
```

The above command sets the operating Mode. This simply selects one of several standard commonly used application programs which are already provided in the on-chip ROM. Both the -ADE and the -PROG products have these programs.

When one of these programs is enabled, it is executed at power-up, or following the RU command.

 On the -PROG, the Mode program is executed in preference to any user program. To run a user program instead, you must set the Mode to an invalid value, e.g. 100. If using KDCFG, set it to "OFF".

Port types: All the Mode programs currently provided assume that port 1 is never tri-state, i.e. is RS232 or RS422. Also, a tri-stateable (i.e. RS485) port 2 is implicit in modes 1 and 2.

Valid Modes are 0, 1 or 2. Please see the **Overview** chapter for their details.

AD – RS485 address – applies to Mode 2 only

```
address=address,lead-in,timeout
ad=\65,\101,25
```

The above command specifies the RS485 address, the optional lead-in byte, and the timeout. All three values must always be specified.

The first two values are entered in decimal, 1-3 digits, each with a "\ " prefix. Allowed range is 0-255 for both. Note that address values of 254 and 255 have special functions - see above. To disable the lead-in byte, enter "\0" (zero) for its value.

The timeout (in ms) is entered without a "\ " and can be any value 0-65535. In 9600 baud systems, a suggested initial value for this timeout is 20ms plus the worst-case response time of the device attached to port 1. The "response time" here is the delay between the device receiving the end of the poll and the first byte of the response. In most devices, this response time is only a few ms.

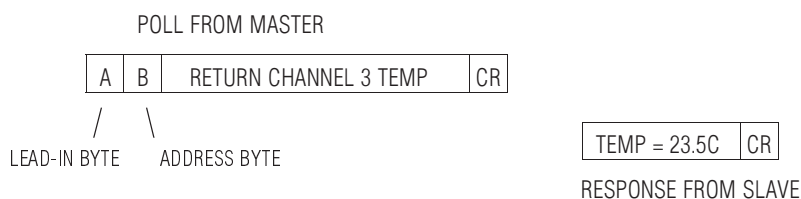
 RS485 systems do not have handshakes. Many RS232/422 systems do not have them either (although the KD485 can support XON/XOFF). If one KD485 port is set to e.g. 1200 baud and the other is set to e.g. 115200 baud, then you must ensure that the messages involved are short enough and/or infrequent enough to not overflow the KD485 RX buffers.

The value of the address byte must match the value emitted by your RS485 host system, and each KD485 on the RS485 bus must be configured to a *different* value.

The purpose of the lead-in byte is simply that it makes communications more robust. The receiving device (the KD485 in this case) can discard all data until this byte is received; only then does it look for a valid address byte. This scheme vastly reduces the probability of an accidental address match in a noisy system.

A lead-in byte also enables the use of an address value which may appear within the data.

The following illustrates one possible RS485 data exchange, demonstrating the use of the lead-in byte "A" and an RS485 address byte "B":



UF – Upload a “user program” file – KD485-PROG only

uf

The above command prompts for and requires a “Y” confirmation. Any other key terminates the upload.

A “Y” confirmation invalidates any existing program in the EEPROM (by overwriting its checksum) and places the KD485 in a receive mode where it expects data in the standard Intel Hex format.

At this point (within 60 seconds) you must initiate file upload in your terminal emulation program (with **ctrl-u** if using the supplied **term.exe** program), or press ESC to terminate the upload mode. Almost any terminal emulation program capable of uploading an ASCII file can be used. No handshake is required because, at 9600 baud, the KD485 can write each hex record to the EEPROM faster than the data can arrive.


When reading each hex line, the KD485 discards everything preceeding the initial “:” character. The line must end with an LF, optionally preceeded by a CR. A CR-only ending is not permitted. Maximum line length is 128 bytes, including the CRLF. Only type 00 (data record) and type 01 (end record) record types are used; all other types are ignored. Gaps in the data exceeding 5 seconds will terminate the upload mode.

Each hex record is written into the EEPROM as soon as it is received and its address and checksum are verified. When the end record is received, the program size and checksum fields in the EEPROM are updated. The program is now ready to run, either with the RU command, or by interrupting KD485 power.

The “end” record is mandatory, as are correct checksums on each line of the data. Any checksum errors, or address fields which are outside the EEPROM address range, terminate the upload mode and place the KD485 into a “data dump” mode where all data is discarded until there is a gap of a few seconds. This allows you to terminate the upload in your terminal emulator program in an orderly manner.

If you need to prevent the user program executing at power-up (e.g. to get to the Executive instead, without using the front panel pushbutton switch to do it) type “UF” followed by a “Y”, then press <enter>, wait for 5 seconds, and do not upload a program. This permanently invalidates the user program.

 A “UF” command with a “Y” confirmation irreversibly invalidates any existing program. Use it with care!

 The uploaded program will not execute if a valid Mode program is selected. The KD485-PROG executes a valid Mode program in preference to a User Program. Use e.g. md=100 to select an invalid Mode program.

RU – Run program

ru

The above command reboots the KD485. Also see the UF command above.

TE – Test ports

te

The above command runs a continuous loopback test on Port 2. The test stops when any key is pressed on the terminal. The test runs at the currently configured Port 2 baud rate. It uses the string “A”...“Z” repeated. This string is *not* output to the terminal; this function is intended for oscilloscope monitoring of data flow.

This test requires a loopback connection on Port 2: if Port 2 is RS422/485, interconnect TXA–RXA and TXB–RXB, and a 220 ohm load resistor may be connected TXA–TXB for additional confidence. Ensure that nothing else is connected to port 2.

TS – Test Slave device

ts [=h]

The above command is useful for testing slave devices attached to Port 2. A “:” prompt is displayed. All characters entered (except the final <enter>) are transmitted to the slave device. Non-printing characters are entered as three decimal digits with a “\” prefix, e.g. \013 is a CR. *Three* digits must always be entered. Up to 80 characters may be entered after the “:” prompt.

Any data returned from the slave device is displayed as received. If it contains non-printing characters, it can be displayed in hex instead, using the command form “ts=h”.

Port 2 = **RS485**: The port 2 driver is permanently enabled while in the Executive. To enable testing of 2-wire RS485 devices, the driver goes tri-state for 10 seconds following the transmission of the last character of the enquiry string and this allows more than ample time for the slave device to drive the bus and return its response. The Executive > prompt does not re-appear until this time has expired.

The timeout is extended, indefinitely if necessary, by any data returned from the slave device. This enables the KD485 to be used to monitor data on Port 2; you simply enter a null string (<enter> by itself) to the “:” prompt.

The TS function is not suitable for port 2 baud rates below 75 baud.

HE – Help

he

Presently this returns the firmware date and version only.

SOFTWARE UTILITIES

TERM.EXE

This is a basic terminal emulation program, with file upload and hex dump capability. It is a DOS application which works under DOS, and in a Windows 3.1, 95 or NT4 DOS box.

For usage details, type

`term`

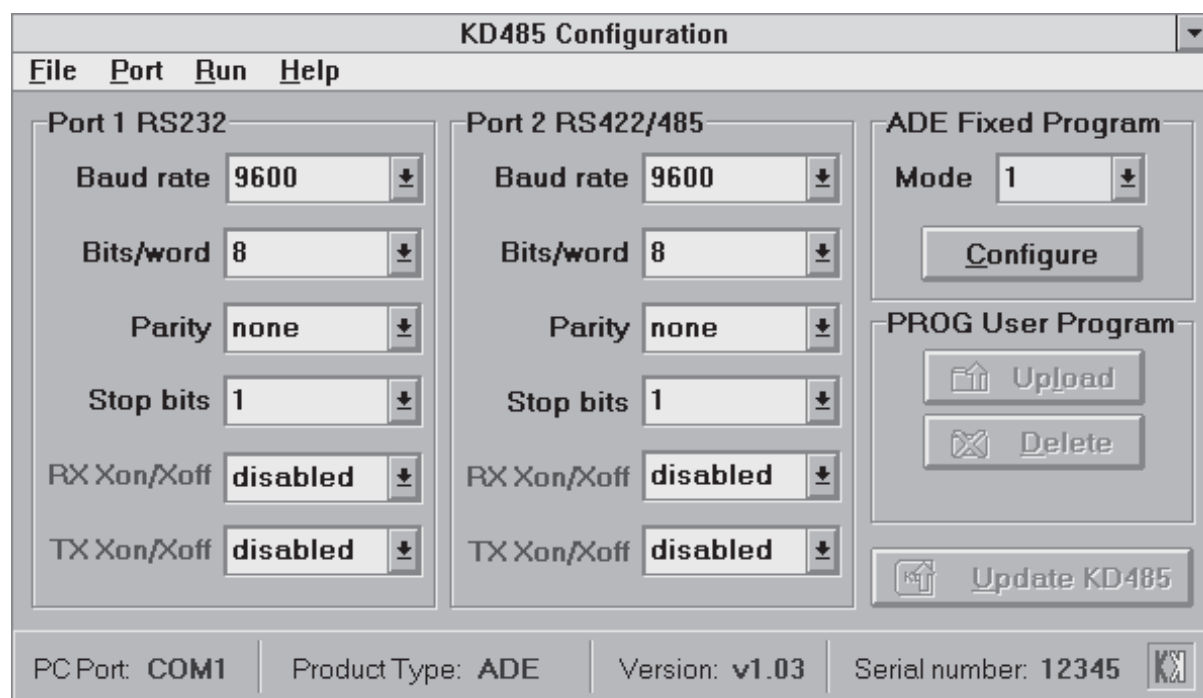
with no parameters.

By default, term sets the COM port to 9600,n,8,1 and leaves it so configured upon exit. This is correct for the KD485 but there is an override switch to stop this.

For non-KD485 applications, term internally translates cursor keys to ANSI ESCape sequences, so if you have ansi.sys loaded, term.exe provides a simple "ANSI" terminal.

KDCFG.EXE

KDCFG is a Windows based configuration program for the KD485-ADE and KD485-PROG. It is supplied on a CD with the unit, and can be found on our website <http://www.kksystems.com>. It is compatible with all versions of Windows up to Windows 10, 32-bit and 64-bit and supports COM ports 1-255.



The port types (RS232 or RS422/485) are displayed at the top, and the product type, firmware version and serial number are reported in the lower part of the display.

-  20mA ports are displayed as "RS232".
-  KDCFG requires that the KD485 is in the Executive mode (LED flashing rapidly) when it is started.

KDCFG cannot be used to run the TE and TS commands. It does however contain additional configuration items; for example, for "difficult" multi-drop systems containing poorly-designed instruments, customised RS485 driver turn-off delays can be specified for each of the baud rates above 4800 baud.

Troubleshooting

KD485 power-up problems

With a valid DC power input (7V to 35V), the voltage on the TEST OUT terminal (Port 2, pin 5) must be within the range +4.5V to +5.5V, measured relative to Port 2 pin 6.

KD485-ADE/-PROG only: When the KD485 is powered-up, the LED should blink twice, regardless of its configuration. If it does not illuminate at all, check DC power input. It must be in the range +7V to +35V and be connected solely to terminals 7 and 8 (on the Port 1 terminal block).

The KD485's power supply will also fail to start if the supply voltage rises from zero to the switching power supply startup voltage (typically 4V) very slowly. See the Specification section in the **Overview** chapter for details.

If after power-up the LED enters a repeating pattern of *n* rapid blinks followed by a longer pause, this indicates that a hardware fault has been detected by the KD485 power-up test firmware. The number of blinks "*n*" indicates the fault, but any such fault requires a return to the factory.

RS232 Communications Problems

- 1 Make sure TX,RX are correctly connected. RS232 is not a "bus" system; the TX terminal of one device feeds the RX terminal of the other device - unless one device is "DTE" and the other is "DCE". If in doubt, consult the device documentation.
- 2 KD485-ADE/PROG: check you have configured the KD485 to the correct baud rate and other parameters.

RS422/485 Communications Problems

- 1 Many RS422/485 equipment manufacturers have incorrectly labelled their A/B signals and you may need to swap them. Measured at the KD485 RS422/485 interface relative to the isolated ground M, a HIGH level (e.g. a start bit) is represented by A=+5V and B=0V, as per the published standard. Conversely, with no data being sent, A=0V and B=+5V. With significant loading (e.g. with cable terminators fitted) these voltages will be shifted towards the centre (+2.5V) but the *polarity of the difference* between A and B remains.

One way (other than swapping the wires) of determining how the manufacturer has labelled their terminals is to read the product's manual and look for a detailed description of the terminal functions.

Another is to measure the DC voltage on each of the product's terminals, with nothing connected and no data flowing, relative to ground. The terminal with the higher potential is our "B". This test is reliable only if the product has internal pullup/pulldowns and they are connected the right way!

- 2 Some RS485 systems use only the signal wires (2 or 4) and *no ground connection*. While this should work where the ground connections of the equipment involved are interconnected via another route, it is not a recommended practice because the finite common mode range (typically -7V to +12V) can easily be exceeded, resulting in comms errors or even equipment damage.

The isolation property of the KD485 allows the grounds to be properly connected to their respective equipment without the danger of creating ground loops.

- 3 On KD485-STD in 2-wire RS485 mode (Port 2) only: check that your RS232 host system properly controls its RTS signal: HIGH when transmitting and LOW when listening. Note that RTS must remain HIGH until the very last bit has shifted out of the host's UART.
- 5 On KD485-ADE/-PROG running the **Mode 0** program with the RTS input unconnected: set SW3=ON.
- 6 Check the DIP switch settings.

Cannot enter the KD485 Executive

If unable to communicate when using the KDCFG.EXE Windows configuration program, this problem is normally the result of the KD485-ADE/PROG not being in the Executive mode *when KDCFG is started*. Exit KDCFG, ensure the “EXE” LED is flashing rapidly (by pressing the front panel button - see **Installation**), then re-start KDCFG.

If, upon power-up, the “EXE” LED is not flashing this indicates that the KD485 is running a program. To force entry into the Executive mode, hold down the switch (this takes a number of seconds) until the LED is flashing rapidly, then immediately release it. Then wait (several more seconds) until it again starts flashing rapidly. If you have a terminal connected to Port 1, you should now see a PRESS ENTER message repeated on the terminal.

- 1 Under Windows 95/98 or Windows NT, there can be a conflict between KDCFG.EXE and another serial port driver. Download the latest version of KDCFG.EXE from **www.kksystems.com**.
- 2 If the KD485 has a suffix like -422-422 or -422-20MA, etc, Port 1 is not RS232 and a suitable interface converter is required.

The following applies to configuration using the command line mode:

- 3 If the “EXE” LED is flashing rapidly and continuously then the KD485 is already in the Executive. Pressing ENTER should produce the KD485 > prompt. If this does not appear, it is most likely due to a communications problem in the connection to the terminal. Try the following:
 - Reset the KD485 by interrupting power to it, and repeat 1 above if necessary.
 - Press ENTER on the terminal several times until the > prompt appears.
 - If using a non-TERM terminal, check it is configured for NO handshake (or XON/XOFF only, no hardware handshakes) and is set to **9600,N,8,1**.
 - If using TERM, exit it (with ctrl-c) and re-enter.
 - If using TERM, try a different serial port on the PC (e.g. COM2).

Intermittent Comms Errors - General

The usual cause of this is bad wiring or grounding, in an electrically hostile environment. Make sure the wiring is shielded, with the shield connected to a good ground. If interconnecting two devices, one of which is isolated (i.e. has a floating ground terminal) then you must interconnect the grounds of the two devices.

Intermittent Comms Errors - RS422/485

Check that your signal wiring is **properly shielded** and that the KD485 Port 1 / Port 2 **isolated ground** is connected to the ground of the other RS485 devices. While RS422/485 is known for its good noise immunity, this is true only if any induced common mode voltages are within the capability of the receiver. The DC common mode limits on RS485 are typically -7 to +12V and these can be exceeded under transient conditions if the isolated ground is not connected. Also, even if the common mode voltage is always within this range but includes a very high frequency component, the receiver may again not function properly. This is true for all RS485 devices, when the ground terminal of an RS485 device is left unconnected.

Another cause of persistent comms problems may be that the bus is floating into an invalid state in between messages. See the discussion of Pullups in the **Ports and Connections** section.

In some systems, problems can occur if the value of the RS485 address byte occurs *within* a message.

Intermittent Comms Errors - KD485-PROG

In the KD485-PROG you can experience runtime errors in your programs, if you have made programming errors. These may or may not cause the watchdog timer to trip. They are more likely to cause it to trip if your program has taken over the watchdog triggering. There are many possible runtime errors, ranging from endless loops to stack overflow. Some runtime errors can be triggered by invalid input data; your programs must be written to handle this.

C Introduction

This chapter is a quick “get started” guide to KD485-PROG C programming.

The C programming environment has been implemented in a way which makes programming much easier than it is on other hardware devices. The compiler is a full ANSI C compiler which generates efficient and well optimised code. Many KD485-specific comms and other extensions have also been added.

To program the KD485-PROG, you need the Hi-Tech H8/300 C cross compiler, version 7.60 or higher. This compiler is available in MS-DOS and other versions. It is priced separately from the KD485, and must be ordered separately. This compiler is a standard off-the-shelf unmodified Hi-Tech product.

You will also need a “make” utility to run the supplied makefile. Virtually any such utility, including those supplied with Borland or Microsoft languages can be used. The standard protected-mode make is preferred because it leaves more memory for the compiler.

To avoid the need for a make utility, a batch file mkhex.bat is supplied. Typing e.g. “mkhex test” with a program test.c will compile it, link it, and generate a hex file - if there are no compiler etc errors. This batch file must be edited before use, to set up the path to the compiler, linker, etc.

The Hi-Tech compiler includes a DOS protected-mode version which should always be installed unless your PC is a 286 or lower.

The following features of the Hi-Tech compiler are not supported with the KD485:

The compiler includes an IDE (integrated development environment). However, because of the ease with which a makefile-based system can be set-up (and because many programmers prefer to use their favourite editor anyway), the use of the IDE to achieve the various required compiler and linker operations is not documented here.

The compiler includes a source-level debugger called Lucifer. This is not supported. It would not be very useful in a datacomms product because the act of single-stepping through real-time comms code usually prevents correct system operation anyway.

Step 1: Get to know the KD485

You are strongly advised to acquire a general understanding of how the KD485 works. At least, please have a good “scan” through this manual.

Step 2: Compiler Installation

The Hi-Tech C compiler is a general-purpose H8/300 compiler and is not specific to the KD485. To install it, follow the steps in the compiler documentation.

Write down the serial number and the installation key (these are normally on the CD) in case these are needed in the future.

The installed copy is copy-protected with a scheme which ties the installed compiler to some property of the hard drive. You can replace individual files at any time but if the hard drive is reformatted or the compiler directory tree is modified, you will need to reinstall from the CD.

Unless this Hi-Tech compiler is the only Hi-Tech compiler which you are ever likely to use, choose an installation directory name like \HT8300 rather than \HT. This is because a version of the compiler for a different CPU may by default install in \HT also.

The Hi-Tech compiler disk includes C and assembler sources for the standard Hi-Tech runtime library. There is no need to install these except for reference. Note that the Hi-Tech sources do not necessarily correspond with the runtime library functions provided in the KD485-PROG ROM, many of which are hand-optimised.

None of the Hi-Tech libraries or .h files are used in KD485 programming. Please examine the supplied hello.c and test.c programs; these are useful templates for your own program.

Step 3: KD485 C Software Utilities Installation

Create a directory to work in, e.g. C:\KD485. Copy all files from the \c directory on the KD485 Software Utilities diskette into it. Read the readme.txt file.

At this point you can install the Windows-based KD485 configuration program KDCFG.EXE using the installation program provided with it.

Step 4: Reboot your PC

This is necessary to make effective any changes to your autoexec.bat file made by the compiler installation program. These changes are usually limited to a) adding the compiler path to your path and b) adding several environment variables to your environment. As is usual with programs which modify your autoexec.bat file, you should examine the changes it made and tidy-up if necessary!

Step 5: Create a program

Using a plain-text editor (e.g. Notepad, not MS Word), create the following program called hello.c

```
#include "KD485.h"

int main()
{
    int i;
    for (i=0; i<10; i++)
    {
        fprintf(port1,"Hello World \r\n");
        loadtimer(0,500); while (readtimer(0));
    }
}
```

The above program will output "Hello World" 10 times (at 500ms intervals) and then exit the main() function. This causes a return to the KD485 Executive.

A copy of hello.c is on the CD.

A ready-to-use makefile is provided which compiles, links and locates a program hello.c into a file hello.hex which is suitable for immediate transfer to the KD485. Type

```
make hello.hex
```

or

```
mkhex hello
```

and, when the file has been produced *with no errors*, use TERM.EXE to transfer it (ctrl-u initiates file upload) to the KD485's internal filing system. To prevent KD485 executing one of the built-in Mode programs in preference to yours, set Mode to an invalid value, e.g. md=100. Then reboot the KD485, either using the RU command, or by interrupting its power.

Use TERM.EXE for this exercise, because KDCFG.EXE is a configuration program only and it cannot be used as a terminal.

Following completion of the brief KD485 power-up test, the program will run and the text output will appear on your terminal connected to Port 1.

The supplied makefile contains two targets: hello.hex and test.hex. The latter is a larger C program which tests all the KD485-specific C extensions, and its source test.c is a very useful reference for writing comms programs for the KD485.

Debug Port

The KD485-PROG contains a debug output port. This can be used to emit diagnostic messages. It greatly helps with debugging a protocol converter which uses both of the KD485 ports for the data flow.

The debug port comes out on the RJ-11 connector, pins 2,3. Pin 3 is the ground. The data is TTL-level but it is compatible with most RS232 receivers. The data format is fixed at 115200 baud, 8 bits/word, no parity, 1 stop bit.

To output data from the debug port, use the `dprintf()` function. This has all the features of the normal `fprintf()`. Please refer to the DEBUG directory on the supplied diskette for details of how to emit diagnostic messages from your programs.

The debug port is not buffered, so if you output e.g. 10 bytes to it, your program will hang for approximately 1ms. Normally this does not affect the main program since the two main KD485-PROG ports have substantial transmit buffers already.



If Port 1 is a 20mA Loop port, the debug port is not isolated from the Port 1 20mA constant current source. Do not use both the debug port and the 20mA current source simultaneously, unless you have ensured that isolation is achieved in your system. For example, if the Port 1 20mA current source is driving a completely floating external circuit, that is acceptable.

Large Projects

The procedure is the same as for the simple programs. You need to edit the supplied makefile and add a new target to it. A new linkfile must also be created.

Larger programs should be split-up into several `.c` modules. These are normally compiled separately and then linked together. This is good programming practice.

From this point on, C programming on the KD485 is just like C programming on a PC. Most of the differences are in the I/O area where the KD485 is much more powerful than the primitive comms support of a standard PC.

The next section is a reference covering all KD485-specific C extensions and differences.

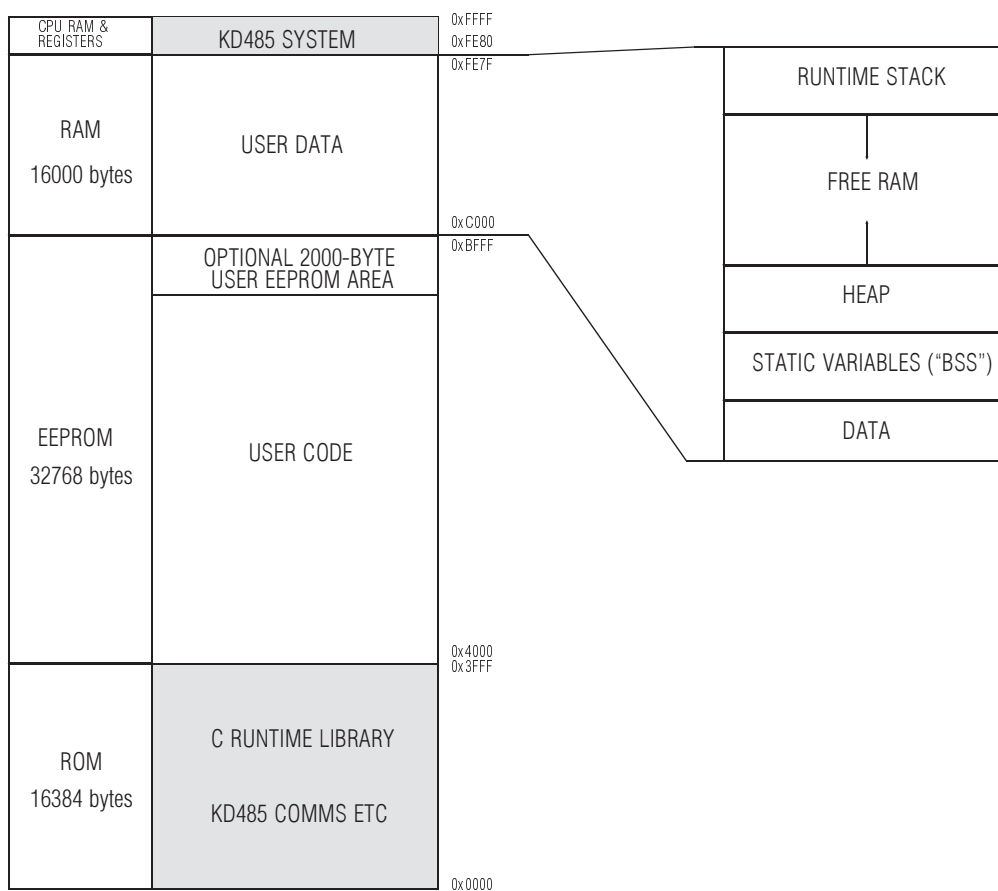
C Reference

This section covers KD485 C programming. It discusses KD485-specific C extensions and other features of the KD485 C programming environment.

The Runtime Memory Map

The user program executes in the EEPROM where it is stored. A RAM data area is available for its use as required. All the I/O and a part of the C runtime library (comprising some large functions such as fprintf, scanf, floats, etc) are ROM-resident and this greatly reduces the user program size.

The following diagram illustrates the runtime memory map:



The shaded areas are shown for information only. You never need to access these addresses directly, except in the unlikely case of setting-up your own interrupt vectors.

The first two words in the EEPROM are the program length and a 16-bit checksum. Your program is linked to execute at 0x4004. The first address of the resulting Intel Hex file is therefore 0x4004. This file is uploaded into the EEPROM and, if there were no errors, the Executive fills-in the program size and checksum.

The program can extend all the way up to 0xBFFF (i.e. 32768-4 bytes in length), unless you wish to make use of the 2000-byte User EEPROM Area (e.g. see function `upload`) in which case the maximum program size is reduced appropriately. The upload function warns if the program just uploaded has overwritten any part of the User EEPROM area. The upload function also aborts the upload if the program is too large, i.e. an attempt is made to load past 0xBFFF.

The first module in your program must be the supplied `init.obj` which ensures that execution starts at the function in your program, regardless of where in your program's `linkfile.usr` is. The supplied `linkfile.usr` shows `init.obj` as the first module.

Runtime Library

Only a part of the full ANSI C runtime library is in the KD485 on-chip ROM. Your program is therefore also linked with a library, `kd485.lib`, which contains all the rest. The supplied `linkfile.usr` shows this. As is usual with linking-in library-resident functions, only those modules which are actually used in your program will be linked-in.

There is a small amount of "clustering" in that e.g. if your program uses `log(x)` then the function `log10(x)` will be linked-in too. You can examine the contents of `kd485.lib` with

which outputs a list of the modules and the functions within each module. At the start of the module listing will be a module called `vectab.obj` which does not contain any code but instead contains the addresses of the vectors to the ROM-based runtime functions. Next is a module called `kd4_spec.obj` which contains all KD485-specific functions that are not in the on-chip PROM.

Using a non-Hi-Tech Compiler ?

The KD485 can be programmed with any assembler or compiler system which can generate code for the Hitachi H8/323 CPU, and which generates an Intel Hex file. However, using the Hi-Tech compiler (which was also used to write a part of the KD485 system software) has the following time-saving advantages:

- 1 The runtime functions provided in the on-chip ROM naturally assume certain parameter passing conventions. The Hi-Tech conventions have been used, and another compiler is unlikely to use the same conventions. One solution is to ignore the ROM-based functions and provide your own versions as part of your program (or in a custom library) but this will make your program slightly larger. Another solution is to write some interface routines to make the parameter passing compatible.
- 2 The KD485 buffered I/O ports are properly integrated into the supplied `init.obj`, `kd4_spec.obj`, etc functions. Because most embedded-system compilers have poor support for multiple I/O streams, accessing KD485 I/O from another compiler will require custom versions of `init.obj`, `kd4_spec.obj`, etc.
- 3 The supplied makefile, linkfile and other examples all assume Hi-Tech C.

None of the above are serious obstacles, if you already have an expensive H8/32x compiler which you know well and can use productively. All KD485 ROM-based functions are accessed via a jump table (located at a fixed address in the ROM) and the files `vectab.kd4` and `vectab.doc` contain all the information you need.

KD485-PROG users have successfully used the IAR H8/325 compiler. This is a good compiler but is very expensive and is not as good as the Hi-Tech compiler.

Modifying Statically Initialised Data

This issue will not affect most KD485 programmers.

The supplied `linkfile.usr`, in conjunction with the supplied `init.obj` startup module, causes all statically initialised data objects to be copied from the EEPROM to RAM at startup. This supports constructs such as

which are thus modifiable at runtime.

If your particular project requires every last drop of RAM and you do not need your initialised data to be runtime-modifiable, then you can prevent the above-mentioned copying by editing the linkfile and `init.kd4`, at the cost of not being able to modify `init.kd4` at runtime. Alternatively, for selected objects, you can use the `volatile` type.

Data Alignment

Unlike Z80 and 80x86 CPUs, the H8/300 requires that all word objects are aligned on even addresses. This includes all opcodes (which are words) and ints and longs.

The C compiler handles this automatically, so alignment is rarely if ever an issue in normal programming. However, when using certain dubious programming techniques it is possible to end up with word r/w instructions attempting to access data at a non-even address, because the compiler has no means of knowing what is happening.

Also, in assembler programming, it is quite easy to fall into this trap when embedding text strings within a program. With some assemblers, an odd-length text string will cause subsequent opcodes to be mis-aligned, with fatal results!

C Extensions

All functions which are implemented in the KD485 C system are defined in the **kd485.h** file. Some of these are located in the ROM; others are in the `kd485.lib` library.

This section documents all KD485-specific functions, i.e. those which are *not* part of ANSI C. It also documents those few functions which the KD485 implements *differently* from the ANSI C standard. Please refer to the Hi-Tech manual (or a C reference book) for details of all remaining KD485 ANSI C functions (their prototypes are listed in the `kd485.h` file).

 For usage examples of all KD485-specific functions, see the supplied `test.c` program.

 `kd485.h` is the only file which you *need* to include in your program. Do not include `stdio.h`, `conio.h`, or any other of the `.h` files supplied with the Hi-Tech compiler. Of course, you are free to use other include files particular to your project, as per standard C programming practice.

Serial Communications

The two KD485 ports correspond to two I/O streams called `kd485_1` and `kd485_2`. These streams are supplied as the first parameter to each of the various I/O functions provided. For convenience, the supplied `kd485.h` file #defines the symbols `PORT1` and `PORT2` as 1,2.

If your past I/O programming has been mainly in the PC file read/write arena, it is important that you appreciate that the KD485 I/O streams are not like disk files. They are in fact much simpler to understand and use. Just remember the following:

- 1 Since data flowing through the KD485 is “continuous”, you will never get EOF returned by any I/O function. (This is why most of the KD485-specific character-based functions use type `char`, (not `int`) to represent character data – this also produces more efficient code.) The standard C “file” behaviour (reading what there is to be read and then getting EOF) is generally not useful in a datacomms product because you would be getting EOF very frequently – whenever you have emptied the input queue.
- 2 All I/O is binary; there is no built-in conversion of LFs to CRLFs, no unexpected trapping of CRs, no hidden buffering of input until a CR arrives, no trapping of control-c (0x03) – anywhere. Pure and simple!
This is why `printf` statements in the example programs use `printf` to generate a CRLF rather than the usual `puts` which outputs only an LF. MS-DOS expands LF to CRLF on output.
- 3 All input functions wait (i.e. hang!) until the appropriate number of chars has been read from the KD485 input queue. If this is not desired, use an input queue test like `kd485_inq` to check the queue before reading. The KD485-PROG has a 2048-byte input (RX) queue for each port.
- 4 All output functions wait (i.e. hang!) until the appropriate number of chars has been written to the KD485 output queue. If this is not desired, use an output queue test like `kd485_outq` to check the queue before writing. The KD485-PROG has a 1024-byte output (TX) queue for each port.
- 5 There are no `scanf` functions like `scanf`, etc etc – whose precise operation (e.g. trapping of 0x03) differs from compiler to compiler in any case, and which usually dont support multiple ports.

In the KD485 C programming environment, the emphasis has been on simplicity, while providing everything that is needed to build the most complex asynchronous datacomms applications. The following are the *only* single-character I/O functions provided:

`kd485_getc` returns a char from specified port.
`kd485_getc` as above, non-destructive (i.e. does an `ungetc`)
`kd485_putc` outputs a char to specified port

The following are simplified character-oriented versions of `fwrite` and `fread`:

The number of chars to write/read is specified by the third parameter. These functions are pure binary – they do not stop at a null (0x00). For situations where they can be used, they are more efficient than making repeat calls to `write` within a loop.

The following are the only `printf`-style functions provided:

The above differ from the standard C versions only in that the first parameter is an `int` which specifies the KD485 port in the range 1..2. All ANSI features are supported including longs and floats. There is no `stdout` although you could define one using a macro in terms of `fprintf(stdout,...)` where `stdout` is defined in `kd485.h` as `port1`.

The following are identical to ANSI C and are therefore listed here for completeness only:

All "printf"-type functions return -1 if an invalid port is specified. Other returns are as per ANSI C.

All "printf"-type functions above have full float support. However, they are located in the `kd485.lib` library and using any of them will thus add around 4k to your program size. To avoid this, versions of the above functions which support only ints and longs (i.e. everything except floats) are provided in the ROM:

The usage of the above is identical to the library versions in all other respects. These functions also execute a lot faster.

The following are the only `printf`-style functions provided, and are identical to ANSI C:

The above are in the ROM and have full float support.

There is no `atoi` (i.e. no conversion of incoming data "as it arrives") because such a function is practically useless for building a robust system. Robust operation involves reading some data into a buffer and then stepping through the buffer and "intelligently" identifying its content. In any case, typical implementations read chars into a buffer (whose existence is hidden somewhere in the runtime library and whose size is fixed at e.g. 256 bytes!) and, upon receipt of a LF (0x0A, \n) process the buffer. Such mode of operation would be useless for data which does not have an LF, or which overflows the buffer.



Avoid using the name `PORT` for anything. This is a reserved word in Hi-Tech C. It is a typedef for a pointer to CPU I/O addresses. With the H8/300, these point to the on-chip peripheral configuration registers. The Hi-Tech manual contains full details. When programming the KD485, you never need to access these registers directly.

Debug Port

The KD485-PROG contains a debug output port. This can be used to emit diagnostic messages. It greatly helps with debugging a protocol converter which uses both of the KD485 ports for the data flow.

The debug port comes out on the RJ-11 connector, pins 2,3. Pin 3 is the ground. The data is TTL-level but it is compatible with most RS232 receivers. The data format is fixed at 115200 baud, 8 bits/word, no parity, 1 stop bit.

To output data from the debug port, use the `debug` function. This has all the features of the normal `printf`. Please refer to the DEBUG directory on the supplied diskette for details of how to emit diagnostic messages from your programs.

The debug port is not buffered, so if you output e.g. 10 bytes to it, your program will hang for approximately 1ms. Normally this does not affect the main program since the two main KD485-PROG ports have substantial transmit buffers already.

 To use the debug function you must link dfwritelc.obj to your program.

I/O Queue Management Functions

In a serial stream device like the KD485, data flow generally never ends (although it may of course have gaps in it) and one common “PC” disk-file-oriented programming style, where a program terminates when e.g. an EOF is returned when reading a file, is not suitable for the KD485. Instead, your program will typically contain a loop where it tests for data in the input queue, tests space in the output queue, if both are OK it reads the input data, processes it and outputs it, then returns to wait for more input. The following functions support this programming style and also allow the creation of programs which process communications on multiple ports concurrently:

IPQCOUNT – get #bytes in input queue

Each KD485 port has a 2048-byte input queue which is filled with received data under interrupts. If there is any unread data, this function will return an integer in the range 1..2048. In practice, if the handshakes are operating correctly, the queue size should never exceed around 1500. For full details of the serial port queues and handshakes, see the **Ports and Connections** chapter.

This function allows a program to check if any data has arrived on a specified port’s input queue. It allows a program to read data only when data is present, rather than “hang” until data arrives.

When used in conjunction with the KD485 timers, this function permits the construction of a program which waits for input data for a certain maximum period and then times-out and does something else. For an example of this, see the loadtimer and readtimer functions. When used in conjunction with the opqspace function, this function also permits the construction of a program which services multiple ports in an apparently multitasking manner. For an example see the `ipqcount` function below.

After this function has been invoked, the number of bytes in the input queue may increase as a result of new input data arriving, but it can never decrease except by being read by your program.

IPQCLEAR – clear input queue

This function clears-out any unread data in the port’s input queue. It does not re-initialise the UART. If any input handshakes are enabled, these are set to the READY state; this can result in an XON being returned to the transmitting device.

OPQSPACE – test output queue space

Each KD485 port has a 1024-byte output queue which is emptied under interrupts to the output device (handshakes permitting). Four bytes are always kept unused. If there is any room in this queue, this function will return an integer in the range 1..1024. If the queue is initially empty, a program can freely generate up to 1024 bytes of output data even if the output device has been BUSY all the time.

This function allows a program to check if there is any room in the specified port’s output queue. It allows a program to write output data only when there is room for it, rather than hang until room becomes available.

When used in conjunction with the KD485 timers, this function permits the construction of a program which times-out if the output device has been BUSY for too long.

When used in conjunction with the `opqspace` function, this function also permits the construction of a program which services multiple ports in an apparently multitasking manner. The following example shows

a trivial “multitasking” program which copies data port1→port2 and simultaneously port2→port1, and which can never hang:

OPQCOUNT – get #bytes in output queue

This function is the complement to the `OPQCOUNT` function. It returns the number of bytes of data currently in the output queue. To assist special programs which must at times ensure that the output queue is completely empty, this function has been implemented to return the *total* the number of bytes in the output queue *plus* the number of bytes held in the UART transmit circuitry. This is done to assist applications which need to ensure that e.g. all data has cleared the UART transmit buffer before disabling a tri-state RS485 bus driver.

However, a return value below 2 is ambiguous:

value=1	never returned
value=0	there are 0 or 1 bytes still in the UART

Therefore, a comprehensive detection of the “all sent” condition requires a timer. You must wait until the moment the `OPQCOUNT` return value falls to zero, and then wait for (at least) 2 character periods.

I/O Port Configuration Functions

The KD485 can be configured entirely using Executive commands, so these functions are needed only if your program needs to read and/or modify the port settings during its operation.

The first of the following two functions enables your C program to configure a KD485 port, and the second allows the present configuration to be read-back:

Both make use of the same structure which is defined in `kd485.h` as `PORT_CONFIG`. Both functions return 0 if successful, -1 otherwise.

Note that with `PORT_CONFIG` you must fill-in the port number; `PORT_CONFIG_GET` then fills-in the rest.

👉 Unlike port configuration done in the Executive, the port configuration produced by `PORT_CONFIG` is volatile. The only means to make it non-volatile is to write your program to store/retrieve the `PORT_CONFIG` structure data in the 2000-byte user EEPROM area.

👉 Function `PORT_CONFIG_GET` uses timer #7 internally. It waits for the UART to complete its initialisation and its execution time is approximately two character periods or 3ms, whichever is greater.

Tri-State Driver Control

The `PORT_CONFIG` function allows the user program to directly control the driver as required. The second parameter is 0 to disable the driver and 1 to enable it.

On RS232 ports (usually Port 1) both functions have no effect, but RS422 ports must have their driver permanently enabled, e.g.

For applications where you need to transmit a message and disable the driver very closely after the last character of the message has been shifted out of the KD485 UART, use the function

The `off_state` parameter is always 0 to turn the driver OFF at the end of the message. This function should be called immediately when the `kd485_send` function return value *becomes* zero:

The `kd485_send` function uses a spare hardware timer to turn the driver OFF approximately 3 bit periods after the stop bit of the last character transmitted. For baud rates of 9600 and above, this delay is adjustable using `KDCFG.EXE`, under the Mode 1 configuration settings. Normally you do not need to adjust it but there are some poorly-designed systems which for some reason need the bus driven for an extended time after the message is transmitted.

 The `kd485_send` function also uses timer #7 during its execution. This also means that it cannot be used from inside an interrupt service routine, when it is also called by the foreground program.

Optional optocoupler output

This is a factory option, if Port 1 is not RS422/485. The `kd485_send` function is used to drive this feature. Please contact Factory for connection details.

Reading RTS Input

The function `kd485_rts` returns 0 or 1 if the RS232 RTS IN input is at a RS232 LOW level or HIGH level respectively. The RTS input is the only “hardware handshake” signal available on the KD485, but it cannot be enabled as an *automatic* hardware handshake; its state must be tested by your program as required.

“Break” Sequences

KD485 ports are capable of detecting and generating an RS232 “break” level. This is in effect an over-length start bit, and can be used to indicate (to the receiving end) a condition which could not be represented simply by sending a normal message.

Unless you must use break generation for a very good reason, avoid it. It is rarely necessary in a properly designed system and is largely a hang-over from the origins of RS232 back in the 1960s.

This function transmits a break level of specified duration in ms.

 `kd485_send` uses KD485 timer #7 internally!

 Because the KD485 timers are decremented under a 1kHz timer interrupt, a value of e.g. 2 could produce a duration of anywhere between 2ms and just under 3ms.


To detect an incoming break level: this appears as null(s) (0x00) char(s) accompanied by a framing error. See the function `kd485_receive` for details of how to check for errors, and the break detection example in `test.c`.

Timing Functions

The KD485 contains eight timers, numbered 0..7, each of which is decremented to zero (under interrupt) at 1kHz. Each timer can be loaded with a value up to 65535ms.

`kd485_timer` returns -1 if an invalid timer # is specified, 0 otherwise.

The above timers are present in every KD485-PROG. The RTC option (below) is not required for them.

 Because the KD485 timers are decremented under an asynchronous 1kHz timer interrupt, a value of e.g. 2 could produce a duration of anywhere between 1ms and 2ms. A value of 1 should therefore be avoided.

Real Time Clock

This is an optional feature on the KD485-PROG. It is accessed using the functions

where `time_t` is the standard unix date/time structure defined in `kd485.h`. For RTC usage examples, see the `test.c` example program.

The RTC is the Dallas DS1302 which also contains 30 bytes of nonvolatile RAM. This RAM can be used by user programs. The following functions

provide low level access to the RAM. See `test.c` for usage examples.

 RTC support requires the linking of `rtc.obj` and `rtc2.obj` to your program.

Checksums and CRC

Many data communications systems use checksums or CRCs to ensure data integrity. A CRC function, in particular, is inefficient to implement in C and the following functions (written in assembler) are therefore provided:

This function computes a simple MOD 256 checksum over the buffer.

This function computes a XOR over the buffer. The buffer length must be 2 or greater. The 1st char is XORed with the second, the result is XORed with the 3rd, the result of that is XORed with the 4th, etc.

This function computes a 16-bit CRC over the buffer. The value `0xFFFF` is the initialisation value for the CRC generator and can be used to produce custom types of CRC generation. It can also be used to compute a CRC over multiple buffers, by using the return value of one invocation to initialise the next invocation.

The algorithm used is based on the standard CRC polynomial $x^{16}+x^{15}+x^2+1$. It is not the same as the MODBUS CRC algorithm, a C source code example of which is available on request.

Bit Operations and Rotation

These operations are not provided in standard C and, although they are easy to implement in C, the following self-explanatory functions have been provided for convenience and for performance-critical applications:

returns 011
returns 011

EEPROM Access

The 32768-byte KD485 EEPROM is used primarily for storage of user programs. However, the top 2000 bytes are accessible for writing using the function below:

where `blocksize` is a value in the range 0..1999 only.

Because the entire 32k-byte EEPROM is within the CPU address space (0x4000-0xBFFF) you can read any part of it simply by using pointer loaded with a specific value.

Writing the EEPROM (other than the top 2000 bytes using `EEPROM_WRITE`) is not normally intended because of the potential for corruption of the user program. However, an “unrestricted write” function

is provided just for that purpose. The value `unrestricted_write` is in the range 0..32767. Do not use this function unless you know what you are doing!

All above EEPROM write functions perform a byte-for-byte verify after write, and return the following values:

- 0 success (or blocksize=0)
- 1 invalid input parameter(s)
- 2 device programming failure
- 3 power supply voltage too low on entry to function (DC input is below approx. 6.9V)
- 4 verify error

 The above functions do not check for the possibility of overwriting a part of the user program!

Writing the EEPROM in any other way is not possible. The device has a special “software write protect” mechanism which ensures that a simple write instruction (or any other spurious write signal) is ignored.

 **FINITE EEPROM WRITE ENDURANCE:** An EEPROM has a limit on the number of writes which can be done on any particular address. If this limit is exceeded, that EEPROM address can be damaged.

The write limit in the device used in the KD485 (currently ATMEL 28C256) is guaranteed at 10,000 writes, although in practice failures do not typically occur until the number of writes *to the same location* is well past 100,000. This limit applies on a *per-location* basis, i.e. if you write a particular address until that location is totally destroyed (e.g. 100,000,000 times) you will not have affected any other addresses in the EEPROM.

Because of the above, whenever you write a program which writes to the EEPROM, you should comment-out the actual write function and insert debug statements which output the EEPROM addresses being written. This should prove that your program is writing the EEPROM only when it should be (i.e. not very often!). If your application requires the storage function in order to run at all, you can debug your program with a 2000-byte array in RAM simulating the EEPROM.

The `EEPROM_WRITE` and `EEPROM_BLOCK_WRITE` functions use fast page mode which takes typically 3ms per 64 bytes or part thereof; a 2000-byte block will therefore write in about 100ms.

All the write functions perform a verify after the write, so provided that the return code is zero, there is normally no need to read back the EEPROM to ensure the data was correctly written. An EEPROM has great advantages over the “battery-backed CMOS RAM” non-volatile storage schemes used by other equipment manufacturers: extremely high reliability and long life. It is almost impossible to corrupt the data stored in the KD485 EEPROM, other than by modifying it with the functions provided. The storage life is guaranteed at 10 years and the predicted typical figure (at 25C) is >100 years.

A power loss during an EEPROM write operation will cause the write operation to be terminated, and could theoretically result in loss of data at addresses other than those being written at the time of the power loss.

The KD485 checks that the supply voltage is sufficient before writing each 64-byte page and it does not proceed if it is too low, resulting in the “supply voltage too low” errorcode above.

The KD485 also contains a second small EEPROM device. This stores the Executive serial port settings, the KD485 device address, and all other non-volatile parameters which are configured in the Executive. This EEPROM is not accessible to user programs.

The Real Time Clock option contains 30 bytes of nonvolatile RAM which can be used for purposes where the EEPROM is not suitable, e.g. very frequent writes.

Error Conditions

This function returns the address of three consecutive byte error flag locations:

bit 2	floating point division by zero
bit 3	long division by zero
bit 4	word division by zero

bit 3	parity error
bit 4	framing error
bit 5	RX overrun

p2_errflg:

bit 3	parity error
bit 4	framing error
bit 5	RX overrun

All bits not listed above are reserved for future use.

In a program which looks at these flags, you should clear them when the program starts, and clear each flag *after* you have read it and found it set.

None of the above errors cause any exception handling in the KD485. In most applications, therefore, it is acceptable to ignore them and rely on other mechanisms such as robust programming.

Switches, LEDs, miscellaneous

The function below returns the states of the single **front panel push button switch** into a structure (defined in kd485.h) :

The structure has three members, for compatibility with the PPC Programmable Protocol Converter which has three switches.

The **16-position rotary hex switch** (present on 1998 and later KD485-PROG models only) is read with

which simply returns the switch state, 0-15, as an integer.


 To access the hex switch, you must link readhex.obj to your program.

The following function copies (0 or 1) into the "EXE" **LED**:

The following function fills-in a buffer with information on the KD485 firmware version, number of ports, firmware date, KD485 serial number, and other data. See test.c for usage details.

At present, a buffer of approximately 20 bytes is required. returns the exact size. The returned values represent the following:

+0	KD485 ROM version - MS byte
+1	KD485 ROM version - middle byte
+2	KD485 ROM version - LS byte
+3..+12	firmware date string, e.g. "08-DEC-94", 0x00-terminated
+13	# of ports, =2 on KD485
+14,+15	KD485 serial number (unsigned int)
+16	"Installed Options" byte (see test.c for current usage)

 When extracting the serial number (a 16-bit value) from the buffer, read one byte at a time. Do not cast the two bytes to an int, because the int might not be aligned on an even address!

The following function returns the checksum of the program presently loaded in the EEPROM. In high security or high integrity applications, a user program can check the return against a known (stored) value to ensure that it is what it should be:

The above function computes the checksum over the current program. If mode=0, it returns a nonzero value if the checksum is wrong. If mode=1, it returns the computed checksum.

Watchdog

The KD485-ADE and KD485-PROG contain a 1.6 second watchdog timer. By default, this is enabled at power-up and is pulsed (retriggered) every 1ms from the 1kHz system timer ISR. Should the CPU crash, it is likely (not guaranteed!) that the ISR will no longer execute, and 1.6 seconds later the CPU is reset.

If a program disables interrupts, e.g. with `cli()`, then 1.6 seconds later the KD485 is reset.

The purpose of a watchdog timer is to help ensure continued operation in electrically hostile environments which can be reasonably expected to *very occasionally* crash the CPU.

On the KD485-PROG it is possible for the user program to take over the pulsing function. For example, if your program contains a “main loop” which should never stop, and which can be expected to run fairly rapidly, then pulsing the watchdog at the head of that loop is a perfectly good idea.

The following function specifies whether the user program or the system should pulse the watchdog:


0 for system trigger, 1 for user trigger

If “user trigger” is specified, your program must call

at least once per second. If it does not, the CPU is reset.

The watchdog can be disabled completely, and re-enabled, with the functions below:

 Do not disable the watchdog, other than possibly for very short periods and under well known conditions.

 A watchdog timer makes a system more reliable. It is not a substitute for sound wiring, earthing and other aspects of system design. If a “spike” is powerful enough to crash the CPU in an electronic product, it is probably almost powerful enough to damage something. The next one probably *will* damage something!

The watchdog provides the only method of resetting the KD485. The KD485 cannot be reset by e.g. a jump to 0x0000.

Interrupt Vectors

Your programs are unlikely to need this feature, because the KD485 already contains a highly optimised set of interrupt service routines for the comms, the timer, etc. However, the function

enables you to redirect one of the existing interrupts to your own interrupt service routine (ISR). This ISR can be written in C. This function returns the address of the existing ISR, so you have the option of providing the *sole* ISR for that interrupt, or hooking into the chain. A return value of -1 indicates an invalid vecnum value.

For full usage examples see test.c, and further documentation is in vectab.doc.

Performance Hints & Warnings

The following general suggestions apply to all robust C programs, and are not in any order of importance:

Pass any large objects (e.g. arrays) by address rather than by value.

Automatic (stack-based) variables result in substantially smaller and faster programs. This is the opposite of what one normally finds with older-design CPUs such as the Z80.

For ultimate performance and when writing/reading integers, do not use `itoa` or `atoi` unless you really need their flexibility. Because these functions are always implemented in C as a runtime-interpreted template, it may be quicker to use `sprintf` or a dedicated integer-to-string conversion routine.

Avoid dynamic memory allocation (`malloc`, `calloc`, etc) unless you always free a previously-allocated block before allocating another one. This prevents memory fragmentation. Unlike a PC, the KD485 is unable to advise the user that it has run out of memory!

Do not rely on byte ordering within a word. On some systems, the LSB of an integer is stored at the lowest address. This applies to Z80, Z180 (e.g. the PPC Programmable Protocol Converter) and 80x86 systems (e.g. an IBM PC). Other systems (e.g. the H8/300 in the KD485) stores the LSB at the highest address.

KD485 – PPC Differences

This non-exhaustive list is relevant only when porting C programs written for the PPC Programmable Protocol Converter to the KD485.

- 1 All word variables must be aligned on **even** addresses. The compiler normally does this for you, but when doing dubious things (e.g. casting a 2-byte section of a char buffer into an integer) the compiler has no way of knowing what is happening.
- 2 The LSB of a word is stored at the higher address. On the PPC, the LSB is stored at the lower address. With properly written programs this will never matter!
- 3 The PPC function `putc` is different.
- 4 The KD485 has no `kbhit` function for detecting an incoming break condition. An incoming break is detected as a 0x00 accompanied by a framing error.
- 5 The KD485 does not have the same EEPROM read/write functions. Unlike on the PPC, the KD485 EEPROM is memory-mapped and therefore directly readable via a pointer.
- 6 The KD485 has only 1 program-accessible LED, 1 push button switch and one hex switch; the PPC has 8 LEDs and 3 switches, so the supporting functions differ.
- 7 The KD485 has only two ports, and has no hardware handshakes. It has much larger I/O queues. It also does not currently have a real-time clock.
- 8 The KD485 does not use the KTERM.EXE terminal emulator. TERM.EXE is used instead, and a Windows-based configuration program KDCFG.EXE is also supplied.

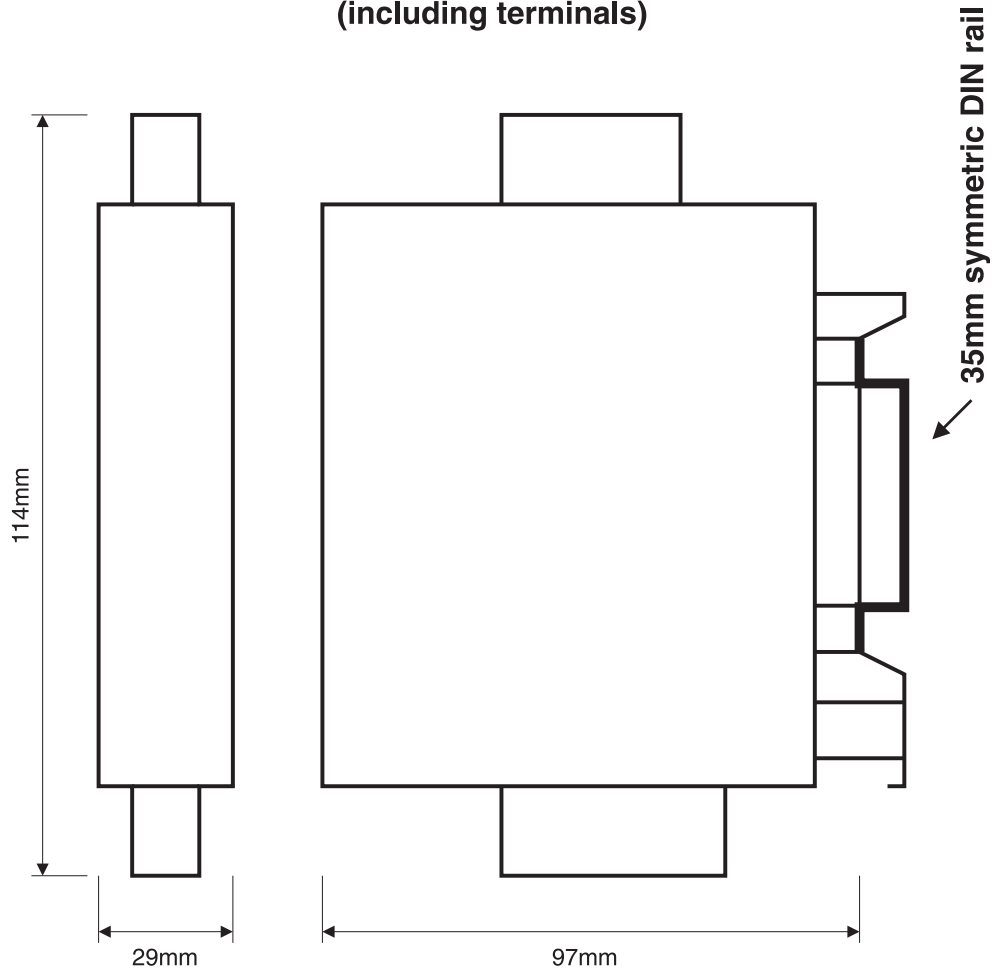
ASCII Character Codes

hex code	decimal code	ASCII value	common function (if any)	hex code	decimal code	ASCII value	common function (if any)
#00	0		null	#40	64	'@'	-
#01	1	ctrl-A	-	#41	65	'A'	-
#02	2	ctrl-B	STX	#42	66	'B'	-
#03	3	ctrl-C	ETX	#43	67	'C'	-
#04	4	ctrl-D	EOT	#44	68	'D'	-
#05	5	ctrl-E	ENQ	#45	69	'E'	-
#06	6	ctrl-F	ACK	#46	70	'F'	-
#07	7	ctrl-G	bell	#47	71	'G'	-
#08	8	ctrl-H	backspace	#48	72	'H'	-
#09	9	ctrl-I	tab	#49	73	'I'	-
#0A	10	ctrl-J	line feed	#4A	74	'J'	-
#0B	11	ctrl-K	-	#4B	75	'K'	-
#0C	12	ctrl-L	form feed	#4C	76	'L'	-
#0D	13	ctrl-M	carriage return	#4D	77	'M'	-
#0E	14	ctrl-N	-	#4E	78	'N'	-
#0F	15	ctrl-O	NAK	#4F	79	'O'	-
#10	16	ctrl-P	-	#50	80	'P'	-
#11	17	ctrl-Q	XON	#51	81	'Q'	-
#12	18	ctrl-R	-	#52	82	'R'	-
#13	19	ctrl-S	XOFF	#53	83	'S'	-
#14	20	ctrl-T	-	#54	84	'T'	-
#15	21	ctrl-U	NAK	#55	85	'U'	-
#16	22	ctrl-V	SYNC	#56	86	'V'	-
#17	23	ctrl-W	-	#57	87	'W'	-
#18	24	ctrl-X	-	#58	88	'X'	-
#19	25	ctrl-Y	-	#59	89	'Y'	-
#1A	26	ctrl-Z	-	#5A	90	'Z'	-
#1B	27	ESC	escape	#5B	91	'['	-
#1C	28		-	#5C	92	'\'	-
#1D	29		-	#5D	93	']'	-
#1E	30		-	#5E	94	'^'	-
#1F	31		-	#5F	95	'_'	(underscore)
#20	32	' '	space	#60	96	'`	-
#21	33	'!'	-	#61	97	'a'	-
#22	34	'\"'	(double quote)	#62	98	'b'	-
#23	35	'#'	-	#63	99	'c'	-
#24	36	'\$'	-	#64	100	'd'	-
#25	37	'%'	-	#65	101	'e'	-
#26	38	'&'	-	#66	102	'f'	-
#27	39	' '	(single quote)	#67	103	'g'	-
#28	40	'('	-	#68	104	'h'	-
#29	41	')'	-	#69	105	'i'	-
#2A	42	'*'	-	#6A	106	'j'	-
#2B	43	'+'	-	#6B	107	'k'	-
#2C	44	','	(comma)	#6C	108	'l'	-
#2D	45	'-'	(minus sign)	#6D	109	'm'	-
#2E	46	':'	-	#6E	110	'n'	-
#2F	47	'/'	-	#6F	111	'o'	-
#30	48	'0'	-	#70	112	'p'	-
#31	49	'1'	-	#71	113	'q'	-
#32	50	'2'	-	#72	114	'r'	-
#33	51	'3'	-	#73	115	's'	-
#34	52	'4'	-	#74	116	't'	-
#35	53	'5'	-	#75	117	'u'	-
#36	54	'6'	-	#76	118	'v'	-
#37	55	'7'	-	#77	119	'w'	-
#38	56	'8'	-	#78	120	'x'	-
#39	57	'9'	-	#79	121	'y'	-
#3A	58	':'	-	#7A	122	'z'	-
#3B	59	';'	-	#7B	123	'{'	-
#3C	60	'<'	-	#7C	124	' '	(vert bar)
#3D	61	'='	-	#7D	125	'}'	-
#3E	62	'>'	-	#7E	126	'~'	-
#3F	63	'?'	-				

Codes 126-255 vary.

Overall Dimensions

**KD485 Dimensions
(including terminals)**



A selection of other KK Systems industrial quality products



K2 range of interface converters

RS232 to RS422/RS485

Versions with automatic driver enable for robust 2-wire RS485
Removable terminal block option



K3 range of isolated interface converters

RS232 to RS422/RS485 and RS232-RS232 isolators

Versions with automatic driver enable for robust 2-wire RS485
Removable terminal block option



USB-485

Isolated USB to RS422/RS485 converter

30-115200 baud, 1.5kV isolation test voltage

Automatic driver enable on 2-wire RS485

Unique device ID - maintains COM port number



USB-232

Isolated USB to RS232 converter

30-115200 baud, 1.5kV isolation test voltage

Unique device ID - maintains COM port number

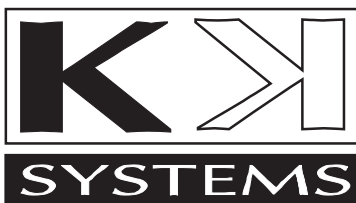


KD420 Modbus RTU interface for analog sensors

Interfaces a 4-20mA or voltage output sensor

to 2-wire RS485. High speed - to 220 readings/second.

Analog voltage range -30V to +30V.



KK Systems Ltd
Tates, London Road
Pyecombe
Brighton
BN45 7ED
United Kingdom

☎ +44 1273 857185
fax +44 1273 857186
info@kksystems.com
Full product data and
online shop:
<http://www.kksystems.com>