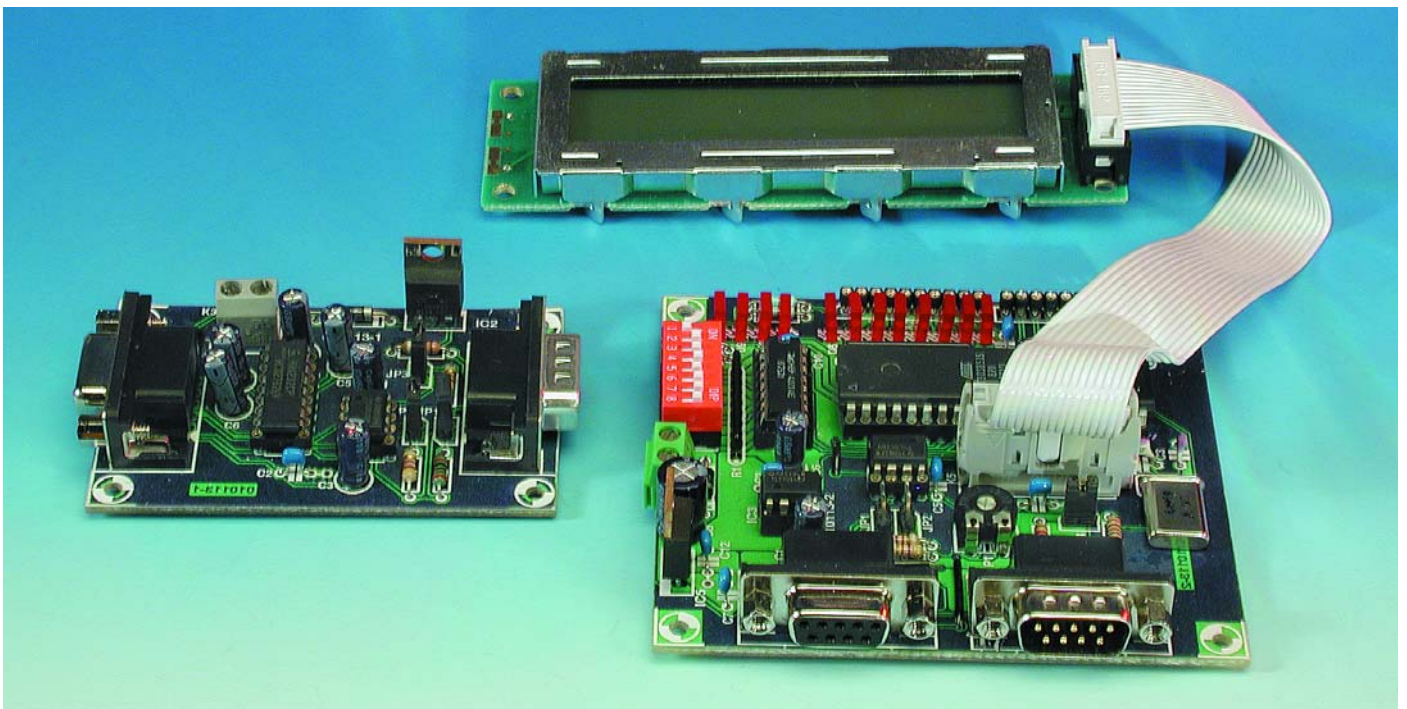


# DCI Bus

RS485 home network with a maximum of 64 terminals

Design by I. de Coninck

This universal bus has many applications. With a PC as a master, information can be exchanged with up to 64 slave terminals. Because each terminal contains eight selectable inputs and outputs as well as an LCD, this home network is not only suitable for use in the home, but it would also be perfect as a combined alarm/communication system for schools and small firms.



Before we delve into the technical details, we will first give a brief description of the system.

The backbone of the system is a complete serial bus that uses its own protocol. Data is sent to and received from a maximum of 64 terminals over long distances using an RS485 interface.

The hardware of the system consists of a

simple RS232 to RS485 converter and a microprocessor driven terminal. Of the latter you will obviously need to build as many as are required. The software consists of a program for the PC and one for the 'terminal processor'. Both files can be downloaded from this month's Free Downloads on our website ([www.elektor-](http://www.elektor-electronics.co.uk)

[elektor-electronics.co.uk](http://www.elektor-electronics.co.uk)). The terminal processor is also sold ready programmed. The PC software runs under Windows 95, 98 and ME and comes with an example program that controls the inputs and outputs of every terminal, as well as send the 2 x 20 characters that are displayed on the terminals.

## Overview of the system

The starting point of the system is a program that controls the serial port of the PC (baud rate = 115,200 bits/s). At the other end of the serial port sits an RS232 to RS485 converter. This is done with a fairly simple circuit, which has a MAX232 and a SN75LBC184 as the most important components. Then there is an RS485 bus (peer-to-peer) that can have 64 terminals connected to it.

Each terminal contains a compact circuit, using an AT90S8515 processor, a 4021 and again a SN75LBC184. Next there is an eight-bit input port, an eight-bit output port and a connector for a two-line LCD module. Each terminal can be given an address in the range 0 to 63. The 'terminal processor' runs a program that takes care of a number of tasks: communicating with the PC, driving the outputs, reading the inputs, sending text to the display and reading its address that has been set with its jumpers.

So how have the communications been implemented?

The program in the PC (master) can select a terminal (slave) with a certain address, then set the state of its eight outputs, followed by the transmission of data for the display, and finally request the state of its eight inputs. The total time taken for these three functions is approximately 25 ms.

The same procedure is then used for the next terminal. If the same terminal has to be selected again, then this takes an extra 0.8 s, because the data on the display needs to be refreshed first.

## RS485

The transmission of data in this system takes place over shielded twisted-pair cable, using the RS485 protocol. A description of this is first in order.

When small quantities of data have to be transmitted over larger distances, the RS485 interface is a good choice. This interface is an electrical specification for a multi-drop bus that uses differential data transmission. RS485 permits the use of several transmitters and receivers in one network (bus).

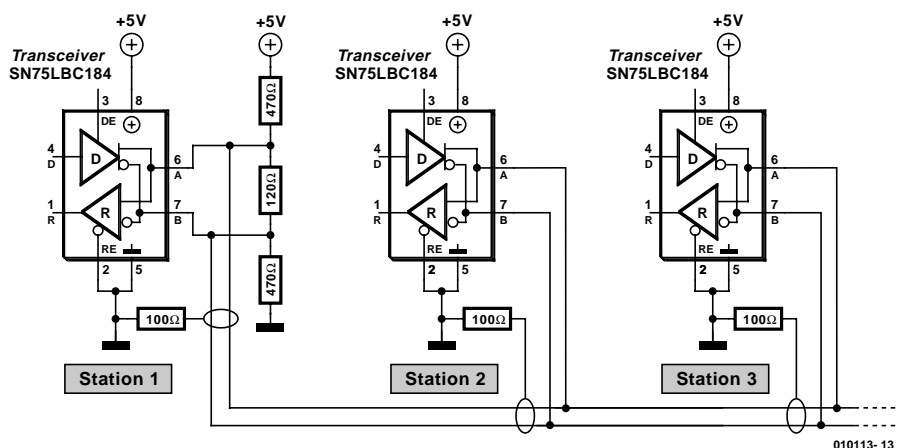


Figure 1. An example of an RS485 network.

The RS485 specification (TIA/EIA-485-A) defines the electrical characteristics of the bus and of the transmitters and receivers. There are also suggestions for the wiring and termination of the network, but the choice of connectors is left open, as is the software protocol that is used.

An RS485 network can have up to 256 terminals when use is made of receivers that have a higher input impedance. The length of the network can be up to 1200 metres at speeds up to 10 Mbps. For longer distances use is made of repeaters that restore the signal and start a new RS485 network. When this system was tested we didn't use extreme lengths of cable; when tested with a cable length of 75 meters everything was still working perfectly.

As mentioned earlier, the RS485 specification doesn't refer to the protocol that is used. In practice much use is made of a protocol that is compatible with that used by the UART in the PC. Several types of RS485 converters are available in the trade and a RS485 transceiver can be connected directly to the serial port of the microcontroller. Most networks use an extra signal to control the transceiver. On the PC side we can use the RTS signal for this purpose.

The reason why RS485 networks can communicate over such a large distance is that the receivers measure the difference in potential between the two conductors in the cable. Most of the interference that occurs in the conductors will be the

same for both; therefore the potential difference between the conductors doesn't change. A common-mode voltage (as in RS232) won't appear either since the signal return is via the second conductor.

The transmitter should provide a voltage difference of at least 1.5 V, giving the signal sufficient tolerance against noise and attenuation. At a node the wiring should be kept as short as possible. As a rule a twisted-pair shielded cable is used for this, because this has characteristics that prevent noise.

The datasheets of the interface chips refer to the non-inverting signal as 'Line A' and the inverting signal as 'Line B'. When the voltage at Line A is at least 200 mV more than at B, the output of the receiver will be 'high'. In the opposite case the output of the receiver will be 'low'. When the difference between Line A and Line B is less than 200 mV, the output state is undefined.

**Figure 1** shows an example of an RS485 network. At the start of the network we can see three resistors: two of 470  $\Omega$  and one of 120  $\Omega$ . These resistors keep the voltage stable on the bus when no drivers are active.

There should be a terminating resistor at the ends of an RS485 network in order to reduce signal reflections in the cable. The value for the terminating resistor in an RS485 network should be between 100 and 150  $\Omega$ . For a baud rate of 115,200, as is used here, a value of 120  $\Omega$  is a good choice. The terminating resistors should only be connected to the start and end points of the network. More resistors don't have any use and could cause a virtual short circuit on the network, overloading the transmitters. The RS485 specification also recommends that a 100  $\Omega$  resistor (0.5 W) is connected in series with the ground at each node in the network. Should there be a potential difference between the

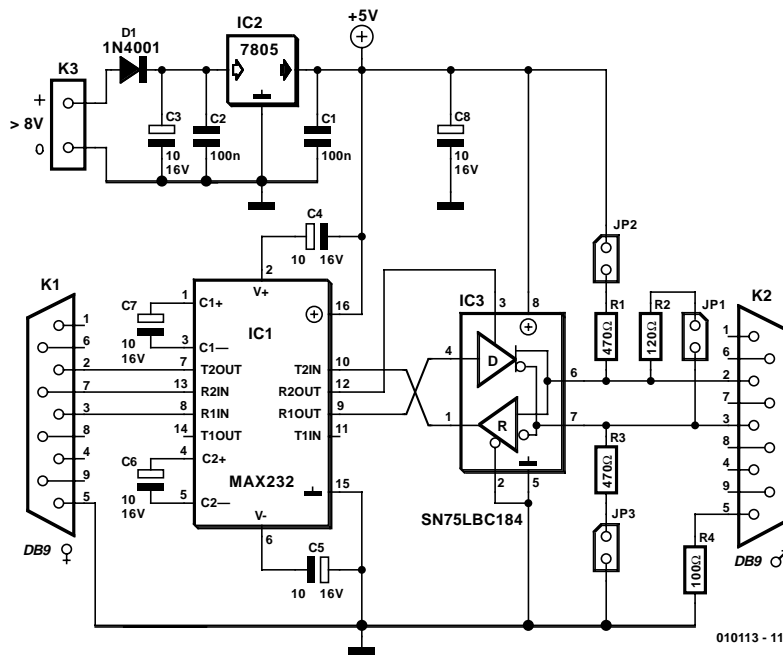


Figure 2. The RS232 to RS485 converter built round the SN75LBC184 transceiver is superbly simple.

grounds at two nodes then the resistors will limit the current flow.

## The RS232/485 converter

The circuit that converts the signals from RS232 to RS485 follows a very simple design, as shown in **Figure 2**. The voltage levels of the serial port first have to be adapted for use with the SN75LBC184 and for this we use a MAX232 (IC1). Electrolytic capacitors C4 to C7 are required by the internal DC/DC converter of the MAX232. The specially designed transceiver IC SN75LBC184 (IC3) takes care of the transmission and reception of data. A 75176 may also be used for this transceiver; but this part does not have ESD protection.

Because pin 2 of this IC is connected to ground, any data on the bus is always received. Pin 3 is connected to pin 7 of the serial port (RTS), via the MAX232. The RTS signal of the serial port will be used via the software to enable and disable the transmitter. Jumper 1 is used to connect the terminating resistor. Resistors R1 and R3 that can be connected via jumpers 2 and 3 to the positive supply and ground respectively, make sure that the voltage on the bus is stable when no drivers are active. Since the converter is at the start of the bus we need to use all three jumpers.

A standard 9 V mains adapter can be connected to K3 for the supply. No special demands are made of the mains adapter, because IC3 (7805) provides a properly sta-

bilised voltage of 5 V.

## The terminal circuit

The complete circuit diagram for the terminal is shown in **Figure 3**. Nearly all the work is performed by the AT90S8515 from Atmel (IC2). This microcontroller contains the software that controls the proper functioning of the terminal. The clock frequency is set to 3.6864 MHz by crystal X1. At this frequency the baud rate generated by the microcontroller is exactly the same as that used in the PC. The transceiver (IC4) sits between the RXD/TXD lines of the microcontroller and the sub-D connectors (K3 and K4) that are provided for the connection to the bus.

The setting of DIP-switch S1 is read with the help of a shift register (IC1). Switches 1 to 6 determine the address of the terminal. When all six are in an open position, the terminal address will be 63. When all six switches are closed, the terminal address is 0. In total there are therefore 64 terminals that can be connected to the bus. Take care never to put two terminals with the same address on the bus, because both terminals would try to respond at the same time, thereby corrupting the data on the bus!

Switches 7 and 8 in S1 are not used at this stage and can be used to provide extra functions to the terminal. The software for the microcontroller has to be modified for this, of course.

LED D9 indicates that the AT90S8515 is ready to receive data. During the time the terminal is active (receiving or transmitting data), LED D12 is lit. LED D10 and LED D11 are not used, but are provided for any future developments.

Pins 32 to 39 of the microcontroller function as outputs and drive LEDs D1 to D8. The outputs are also connected to a header (K2), to provide an interface with the outside world. The inputs go to pins 21 to 28; these have pull-up resistors (array R6) and header K1 provides an external interface.

The reset pulse is generated by a TL7705 (IC3). Electrolytic capacitor C1 determines the timing of the pulse. As can be seen, the reset pulse for the microcontroller is taken from the inverting output of IC3.

That leaves only the connection of a 2 by 20 character LCD display. This is what header K5 is for. The contrast can be adjusted via P1. Display control lines RS and E are driven by INT1 and INT0 respectively. Pin 5 of the display (R/W) is connected to ground, so it is only possible to send data to the LCD.

A simple 9-V mains adapter, as used for the transceiver board, can again provide the supply for the circuit. This adapter is connected to K6.

## The PCBs

**Figure 4** and **Figure 5** show the track and component layouts of the converter and terminal boards respectively. The first one especially is very compact and due to the small number of components it shouldn't take more than an hour to populate.

The terminal board contains a few more ICs and the population will take a bit longer. But populating this board shouldn't give those with average soldering skills any problems either. The connectors and headers have been placed along the edge of the board as much as possible, to make subsequent connections as easy as possible. The only exception to this is K5 for the LCD.

Because of the relatively small current consumption there is no need to fit heatsinks to the 5 V regulators on either board.

## Connections to the bus

**Figure 6** shows how three terminals should be connected to the bus and where the terminating resistors are to be placed.

As mentioned earlier, the address

of the terminal is set using the first six switches of DIP-switch S1. The example in **Figure 7** shows an address of 12 (binary value  $4 + 8 = 12$ ). Jumpers JP1 and JP2 in the terminal circuit are for use at the start of the bus (parallel resistors of 470  $\Omega$  to 5 V and GND respectively). Jumper JP3 is used to connect the 120  $\Omega$  terminating resistor; in **Figure 6** terminal 12 is at the end of the bus, so it has this jumper closed.

When the supply (a standard 9 V mains adapter) is connected, LED D9 will light up after a short period. This indicates that the IC is ready to receive or transmit data. If a liquid crystal display has been connected to the terminal, the following text will appear: 'DCI-BUS Addr: 12'

'waiting for data.U'

The number 12 is the value of the address that has been set for that terminal. The 'U' at the end of the second line is the version number of the software in the microcontroller.

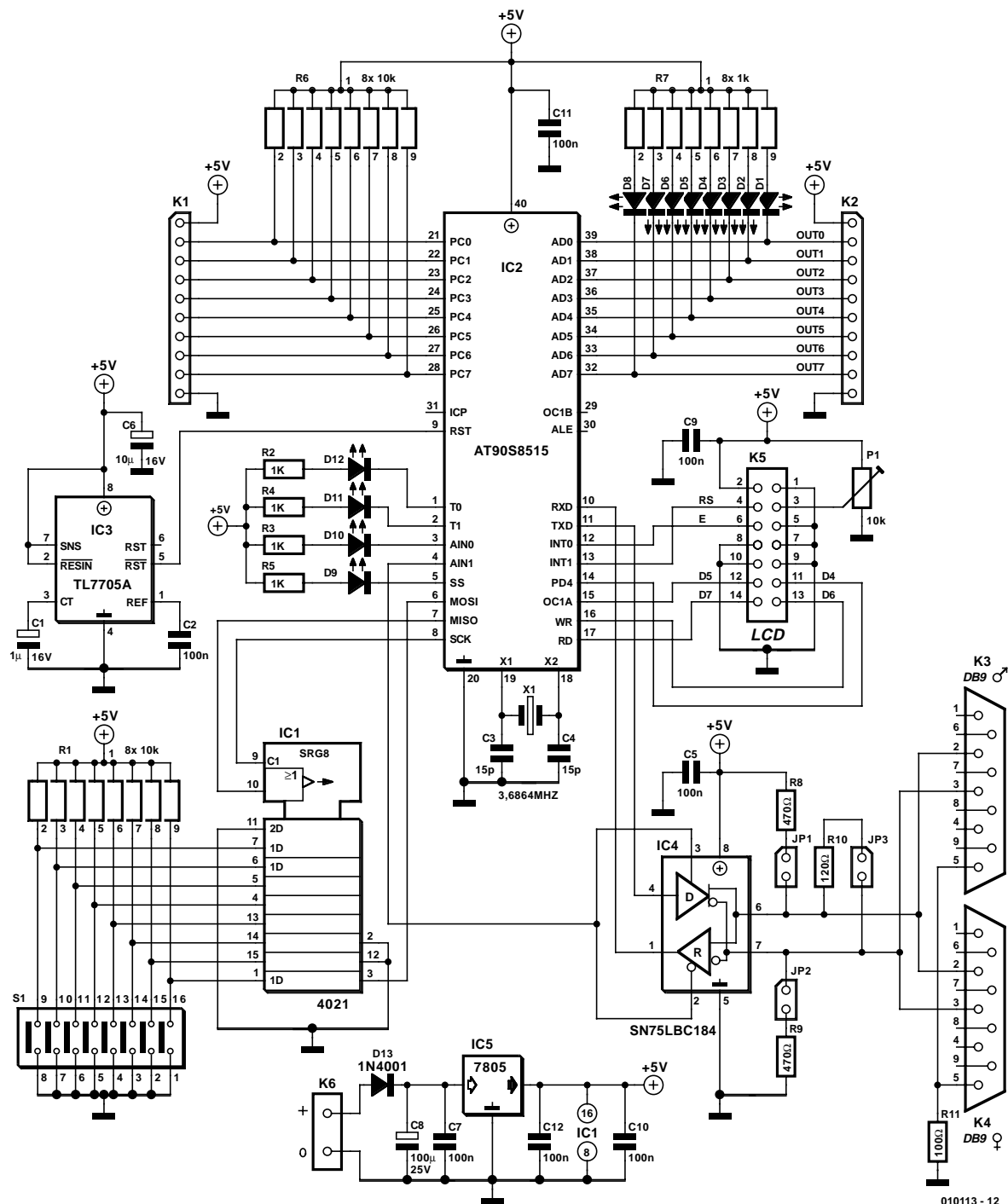


Figure 3. The complete diagram of a terminal circuit. The AT90S8515 controller plays a major part.



## COMPONENTS LIST

### Converter

#### Resistors:

R1, R3 = 470Ω  
R2 = 120Ω  
R4 = 100Ω

#### Capacitors:

C1, C2 = 100nF  
C3-C8 = 10μF 50V radial

#### Semiconductors:

D1 = 1N4001  
IC1 = MAX232  
IC2 = 7805  
IC3 = SN75LBC184 (or 75176)

#### Miscellaneous:

JP1, JP2, JP3 = 2-way pinheader w. jumper  
K1 = 9-way sub-D socket (female), PCB mount, angled pins  
K2 = 9-way sub-D plug (male), PCB mount, angled pins  
PCB, order code **010113-1** (see Readers Services page)

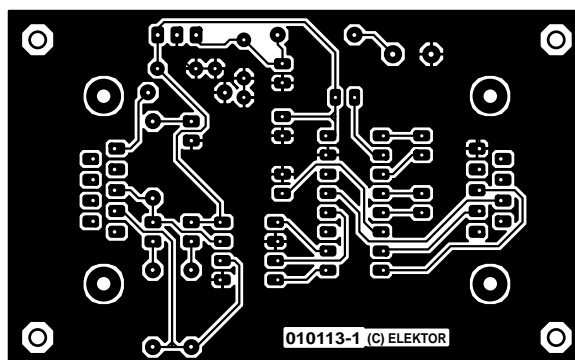
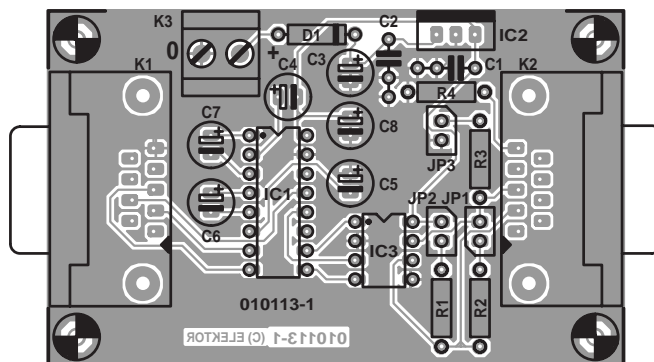


Figure 4. PCB of the RS232 to RS485 converter.

## COMPONENTS LIST

### Terminal

#### Resistors:

R1, R6 = 8-way SIL resistor array, 10kΩ  
R2-R5 = 1kΩ  
R7 = 8-way SIL resistor array, 1kΩ  
R8, R9 = 470Ω  
R10 = 120Ω  
R11 = 100Ω  
P1 = 10kΩ preset H

#### Capacitors:

C1 = 1μF 16V radial  
C2, C5, C7, C9-C12 = 100nF  
C3, C4 = 15pF  
C6 = 10μF 16V  
C8 = 100μF 25V

#### Semiconductors:

D1-D12 = LED, red, rectangular, high efficiency  
D13 = 1N4001  
IC1 = 4021  
IC2 = AT90S8515-8PC programmed, order code **010113-41**  
IC3 = TL7705-ACP  
IC4 = SN75LBC184 (or 75176)  
IC5 = 7805

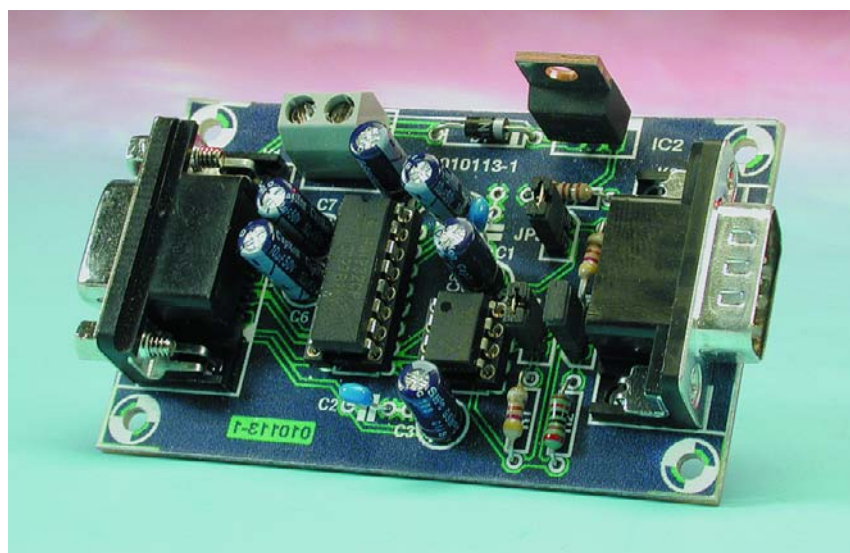
#### Miscellaneous:

JP1, JP2, JP3 = 2-way pinheader w. jumper

K1, K2 = 10way pinheader  
K3 = 9-way sub-D plug (male), PCB mount, angled pins  
K4 = 9-way sub-D socket (female), PCB mount, angled pins  
K5 = 14-way boxheader  
K6 = 2-way PCB terminal block, lead pitch 5mm  
S1 = 8-way DIP switch  
X1 = 3.6864MHz quartz crystal  
LCD module: 2 x 20 characters

PCB, order code **010113-2** (see Readers Services page)  
Disk, contains PC en controller software (incl. source code files), order code **010113-11**

PCB layouts and project software also available from 'Free Downloads' page at  
[www.elektor-electronics.co.uk](http://www.elektor-electronics.co.uk)



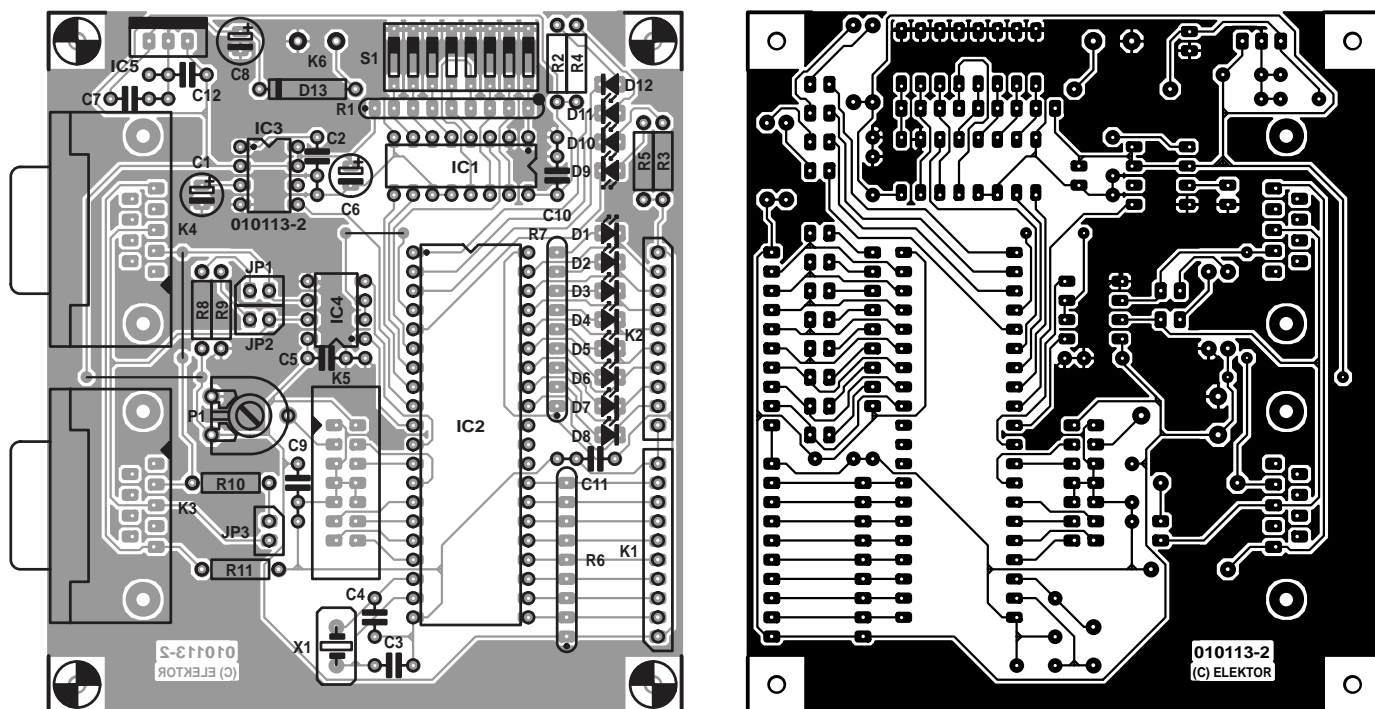


Figure 5. PCB of a terminal circuit. All connectors are at the edge of the board apart from K5.

## The software for the PC

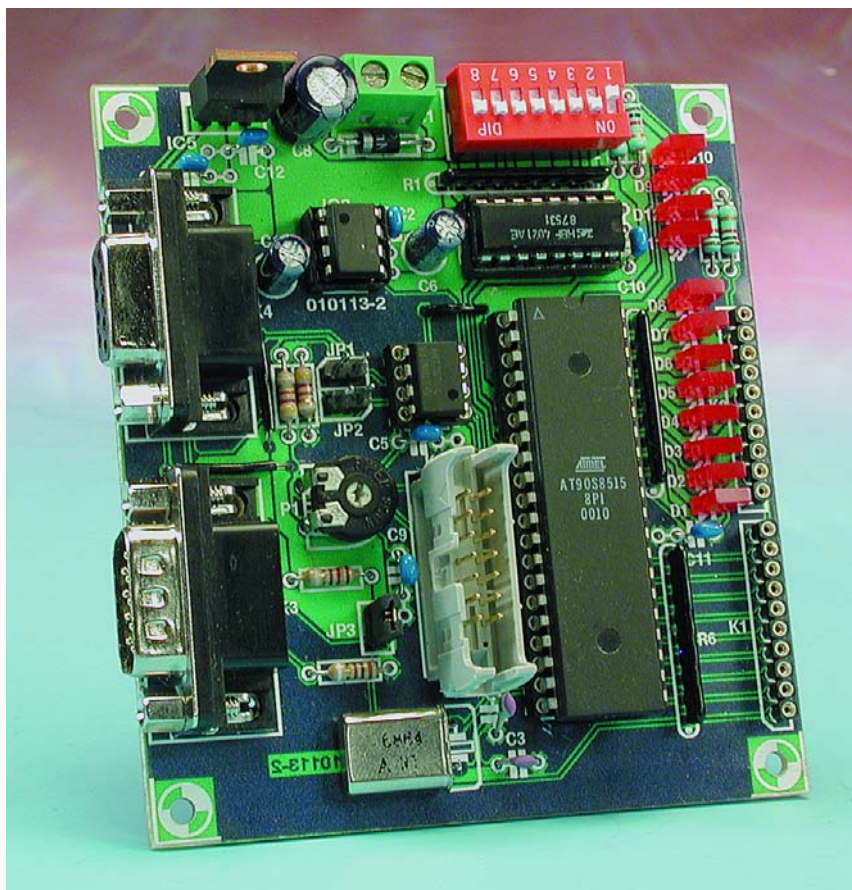
After the hardware has been connected and the software started, the

correct port has to be set up. Click on 'Select port' from the 'Ports' menu and choose the required serial port.

Now click on 'Open port' from the 'Ports' menu and a dialogue box should appear stating that the port has been opened successfully. When a port has been selected that doesn't exist then the message 'The system cannot find the file specified' appears. Next click on 'Run' from the 'Bus' menu and the program will be in RUN-mode (see **Figure 8**).

Next to each 'SetStation' button is an up/down component that can be used to select a terminal number. Then click on the 'SetStation' button and the terminal will be on-line; to the left of the button the number of the selected terminal will appear. From now on the text that is in the two text boxes will appear on the display of the selected terminal and the outputs that have been ticked will be set. At the same time the state of the inputs is also shown. The software can be used to control up to four terminals simultaneously.

The 'adjust bus' up/down button is used to set the so-called 'critical section' to a time between 0 and 20 ms. A value between 3 and 6 works well on a Pentium 400 and 475. This option is probably more useful with the current generation of PCs, which are much faster. Because a program running under Windows doesn't work in real-time, there is a possibility that errors can occur on the bus. This can be seen clearly in the diagram in **Figure 9**. If the RTS signal at the serial port is not 'low' before a terminal returns data, then this data will be lost. This can happen



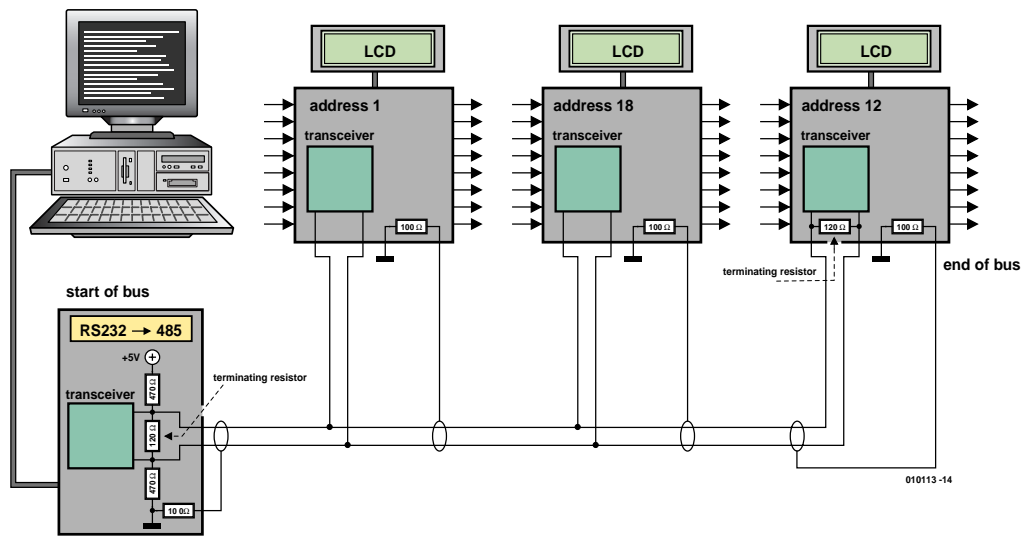


Figure 6. A practical example of a bus with three terminals.

when the operating system (Windows) deems it necessary to use all processing time for a different process, such as starting other software, dealing with a newly inserted CD, screen saver, anti-virus software, etc. Because of this, the RTS signal will sometimes not go 'low' in time.

This can be avoided by adding an error-checking routine to the software that prevents erroneous readings of the terminal inputs. This has been implemented as follows:

0. The transmit-enable output of all terminals will be 'low'.

1. The PC sets its RTS signal to 'low'.
2. The PC transmits 2 x ASCII-code 10 on the bus.
3. The PC puts the address of the terminal to be accessed on the bus.
4. The PC puts the value for the outputs of the terminal on the bus.
5. The PC sets its RTS signal to 'high'.
6. The terminal with the correct address sets its transmit-enable

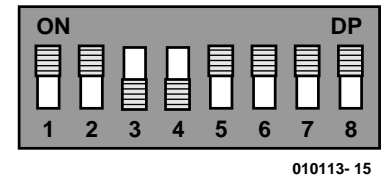


Figure 7. The terminal address has been set to 12 using DIP switch S1.

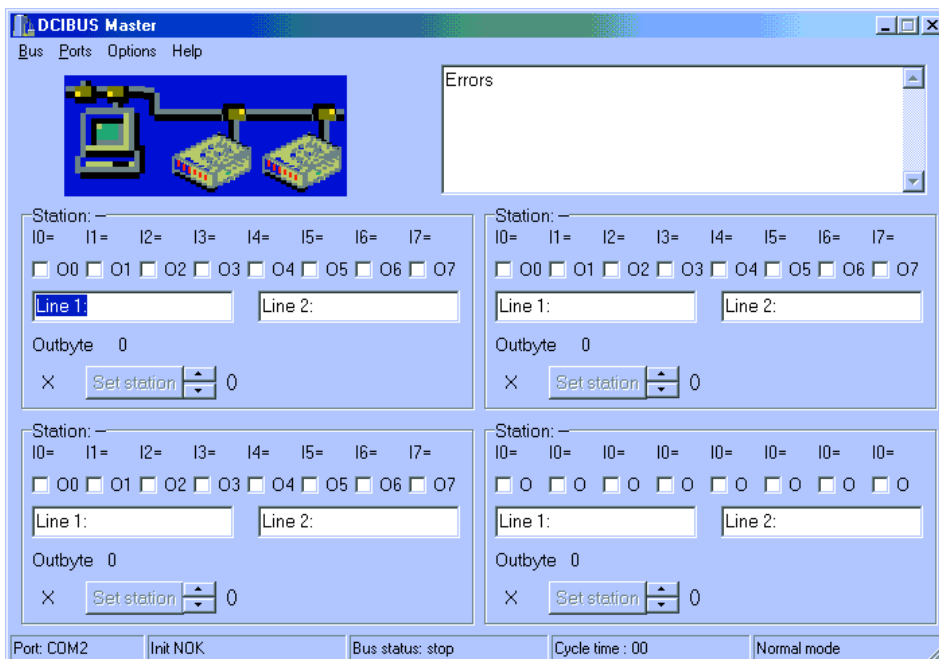


Figure 8. A screenshot of the program in RUN mode.

- to 'high'.
7. The terminal puts its address on the bus.
8. The terminal puts the status of its inputs on the bus.
9. The terminal puts its address on the bus again.
10. The terminal with the correct address sets its transmit-enable to 'low'.
11. The PC software checks that the first byte received is equal to the address that has been transmitted.
12. The PC software stores the value of the second byte (state of the inputs).
13. The PC software checks that the third byte received is equal to the address that has been transmitted.
14. If checkpoints 11 and 13 return the correct result, the PC uses the value returned in step 12 as the new state of the inputs. If either check failed then the software will wait for a cycle that has no errors.



In this way we can be certain that the inputs will always be read correctly. Every time an error occurs, a message is shown with the number of the terminal that caused the error. When we use the software to enable a terminal that is not present on the bus then this error message will also appear.

When our simple RS232 to RS485 converter is replaced by a microcontroller version with a data buffer, the whole communication problem will no longer be an issue. Such a converter is therefore a likely candidate for a future project.

Now for some general information about the software:

The accompanying software for this project comes on a disk (EPS010113-11), which can also be downloaded from the Elektor website. Apart from the PC program it also comes with the source code, written in C++ (Builder 4). The source code and HEX listing for the microcontroller are also on this disk. With these the system can be adapted as much as the user requires.

When modifying the software you should take into account that the terminal requires about a second to refresh the data on its display. Consequently there has to be a delay of one second before the same terminal can be selected again. This delay has to be programmed into the software of the PC!

Furthermore, it is not permitted to include the character with ASCII-code 10 in the text for the LCD, since this character is used to initiate the selection of a terminal!

## Control via an external program

The **Busmstr.exe** program contains a simple COM (Component Object Model) server interface. This allows an external program, such as Excel or Visual Basic, to start the Busmstr.exe program and use it to drive the hardware.

### As an example:

- First start the Busmstr.exe program and then close it.
- Start Excel, open the file 'Client.xls' and start the 'Visual Basic Editor' (Alt + F11).

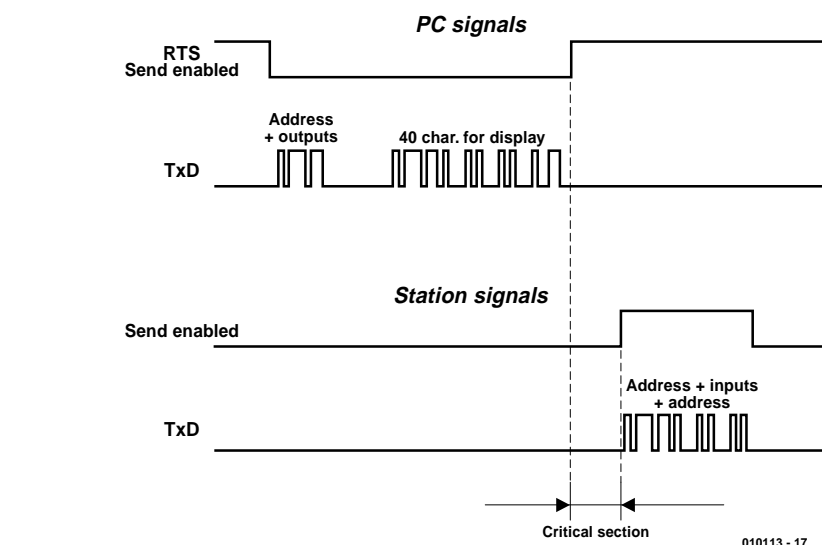


Figure 9. The RTS signal has to be 'low' when a terminal transmits data back, otherwise the data will be lost.

- From the 'View' menu select the 'Object Browser' (F2).
- Place the mouse pointer over the 'Object Browser' pane and click on it with the right-hand mouse button.
- Click on 'References' in the pop-up menu.
- Choose 'Browse' from the 'References' dialogue and choose as file type 'Executable Files (\*.exe,\*.dll)'.
- Navigate to the folder where Busmstr.exe is and choose Busmstr.exe and click on 'Open'.
- In the 'Classes' window there should now be an object with the name 'TBusServer'. Click on this and the members of the TBusServer object appear in the right-hand window. These should be 'About', 'InputGet', 'LCDTxt' and 'OutputSet'.
- Return to Excel and click on the 'INFO' button.
- The Busmstr.exe program is then started automatically. Choose the correct serial port, open it and start the bus.
- Click on 'Server mode' in the 'Options' menu.
- Set a terminal to number 4 and start it using the 'SetStation' button.
- Go back to Excel and close the dialogue box containing Info about the Busmstr.exe program.
- Now click on the 'Refresh' button.

The number in box C4 represents the state of the inputs to terminal 4. The text in box C8 appears on the LCD of terminal 4 and the number entered in box C6 is copied to the outputs of terminal 4. The text in box C8 must not exceed 40 characters in length.

- Now go back to the 'Visual Basic Editor' window and select 'Design Mode' from the 'Run' menu.
- Return to Excel and click with the right-hand mouse button on the 'Refresh' button and choose the option 'View Code'.
- Now the code can be edited. This should be studied with a view of possibly automating the Excel worksheet further.

### Observations:

1. If the Busmstr.exe program has been started via Excel, it will be closed every time 'Design Mode' is entered.
2. Never close the server program (Busmstr.exe) while the client program (Excel in this case) is still running!

(010113-1)

## Further information

may be found on the website of the author:  
<http://home.planetinternet.be/~dc11cd/index.html>

Comments and suggestions are welcome and should be sent via his email address:  
[ivo.deconinck@planetinternet.be](mailto:ivo.deconinck@planetinternet.be)