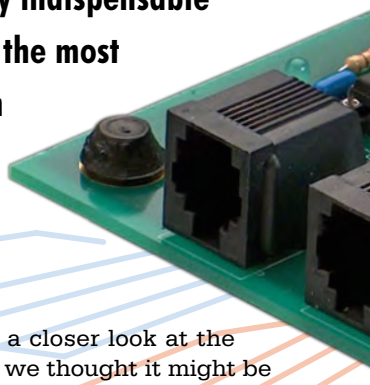
The Secrets of I²C

An I²C bus analyser to let you satisfy your curiosity

Etienne Boyer

In this article, we propose a microcomputing instrument that's valuable – not to say indispensable – when it comes to analysing what's happening on the I²C bus. It lets you examine the most interesting signals carried by this very common, easy-to-implement interconnection bus.



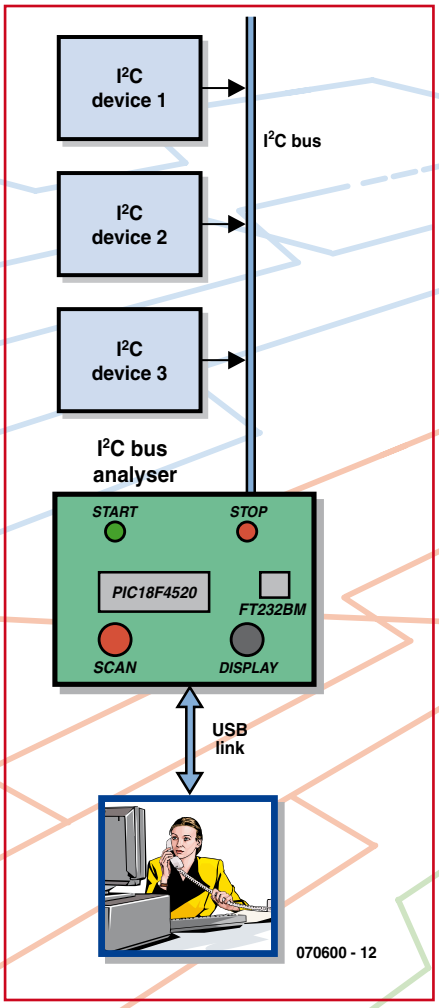


Figure 1. Block diagram of the I²C bus analyser stripped to its bare essentials.

The I²C bus analyser in this article connects to the I²C bus of an application in order to extract from it, for the purpose of examination, the characteristic information of the signals it's carrying; in particular, the START, STOP, ADDRESS, DATA, and ACKNOWLEDGE signals.

It can be used to troubleshoot a reluctant proprietary application or to 'reverse engineer' existing applications. The device communicates with a PC via a USB link configured as a virtual port (COMx) and so is powered directly from the USB, avoiding the need for an external mains adaptor (or even batteries).

Block diagram

Figure 1 gives the block diagram. As already explained, the I²C bus analyser is inserted between the subject being examined, using the application's I²C bus, and the 'watchful' PC. The analyser can have a maximum of three I²C modules connected to it.

The heart and brain of the circuit are combined in a single PIC, a PIC18F4520, the USB link achieved by the standard means of an FT232BM [1] from FTDI (hi Fred) – an IC that you'll already have encountered in many Elektor projects whenever they have anything to do with USB.

Electronics

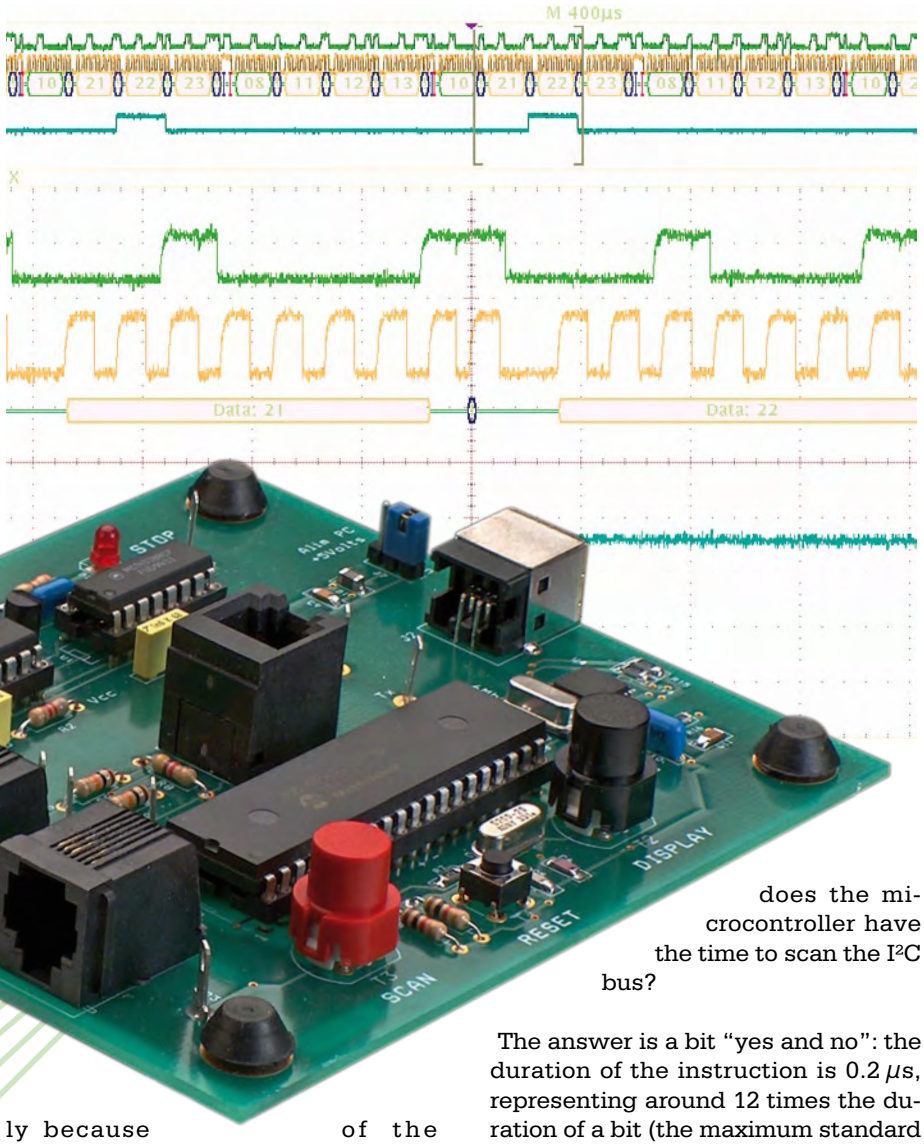
Before taking a closer look at the actual circuit, we thought it might be a good idea to highlight certain specific points of the circuit, by way of a little reminder. The box 'The secrets of I²C and its bus' recaps the most important elements of the I²C bus specifications.

So now let's get back to our circuit. In I²C, the START procedure consists of detecting a negative edge on the SDA while the SCL signal is high ('1'), performed via monostable IC2a of the 4538 IC configured as per the detail from the circuit in **Figure 2**.

The monostable time-constant R2*C2 produces an 8.2 µs pulse that faithfully mimics the I²C bus Start procedure. This pulse width is quite compatible with the reaction time of a microcontroller, but not with the persistence of the human eye! To display the presence of a Start using an LED requires a second monostable. IC2b lengthens the pulse to around 150 ms to produce visible illumination of the green LED D2.

The same goes for the STOP procedure, this time with the help of the monostables in IC1 and the red LED D1. The timing diagram in **Figure 3** shows the two START and STOP signals thus generated.

Note: the two pulses have different durations, as the START pulse ends ear-



Technical spec

- Analyses 100 and 400 kbit/s I²C bus
- Stores 620 contiguous I²C events
- Hardware detection of START and STOP. Display on 2 LEDs
- USB communication by virtual com port
- Self-powered at 5 V via USB port
- PIC programmed in C (CCS compiler)
- Windows man/machine interface in C++ Builder V5 (Borland)

ly because of the signal SCL going high. In practice, the START pulse lasts 3 µs, and the STOP pulse is longer than in theory (8.2 µs) as the circuit doesn't operate within the recommended range (100 µs – 1 second).

Studying the circuit in **Figure 4** shows that there's really very little in the way of electronics: a PIC, a USB interface IC from FTDI, and a pair of 4538 dual monostable ICs. Let's look at their functions a little more closely.

The heart of the circuit takes the form of a microcontroller from Microchip. The PIC18F4520 [2] is an improved version of the 18F452, but is still pin-compatible, not only with the latter, but also with the famous 16F877. Its function is to analyse, by scanning, input lines RC3 and RC4 of the PIC, directly driven by the SCL (**S**erial **CL**ock) and SDA (**S**erial **DA**ta) signals. Push-button S3 (SCAN) starts the analysis. Closer examination of the circuit leads us to ask the inevitable question: does the microcontroller have the time to scan the I²C bus?

The answer is a bit "yes and no": the duration of the instruction is 0.2 µs, representing around 12 times the duration of a bit (the maximum standard data rate is 400 kbit/s); however, this doesn't leave much room for manœuvre to carry out the full processing. What's more, the START and STOP procedures are recognized via edge-detection, which complicates the software. So the microcontroller does not have the time to scan the I²C bus. So a hardware solution comes along to back up the software solution, through the use of edge-detecting monostables (see details in the **inset** 'The secrets of I²C and its bus').

Now that START detection has taken place, all we have to do is detect each SCL clock pulse and sample the data SDA at this moment. Once the signals have been analysed, the microcontroller will store each event in memory in the same way as a data logger. The four events are START, BYTE, ACKNOWLEDGE, and STOP. The memory gets filled up at the rate of the traffic on the bus, and once full is transferred by a serial link to the PC (*USB is by definition a serial link, not a parallel one*).

If the traffic on the bus is too slow, push-button S2 (DISPLAY) lets us purge the memory to the PC so as to display the result without having to wait until the buffer is completely full.

Communication with the PC is achieved by means of the now-standard FT232 IC from FTDI, which uses the USB in the CDC (**C**ommunication **D**evice **C**lass) mode. There are just a very few components around this IC, principally a 6 MHz crystal and its two colleagues C7 and C8, and the type B 4-pin USB socket.
A pair of red and green LEDs are asso-

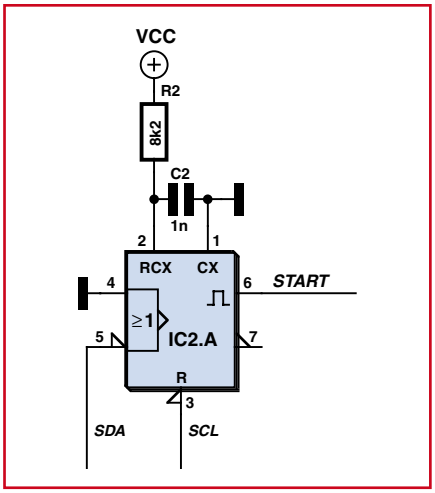


Figure 2. A monostable triggered by a falling edge.

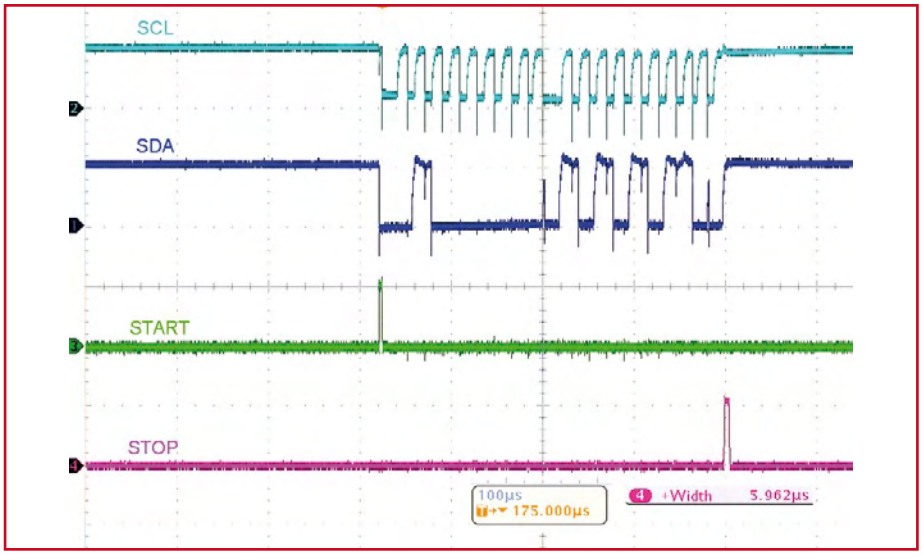


Figure 3. Creation of the Start and Stop.

ciated with the events STOP (D1) and START (D2) respectively.

Two 27 Ω resistors protect the Data+ and Data– lines. The FT232BM IC switches +5 V to resistor R16, indicating two things to the host USB (PC): first of all, presence of the peripheral, and secondly recognition of the Full Speed mode, since R16 is connected to Data+.

One interesting component is inductor L1, a ferrite bead intended to suppress high-frequency interference. Its impedance goes from 0.15 Ω at dc to 70 Ω at 100 MHz, thereby dissipating any electromagnetic interference (EMI) as heat. In addition, it acts as a fuse if you have the misguided idea of short-circuiting our circuit's +5 V sup-

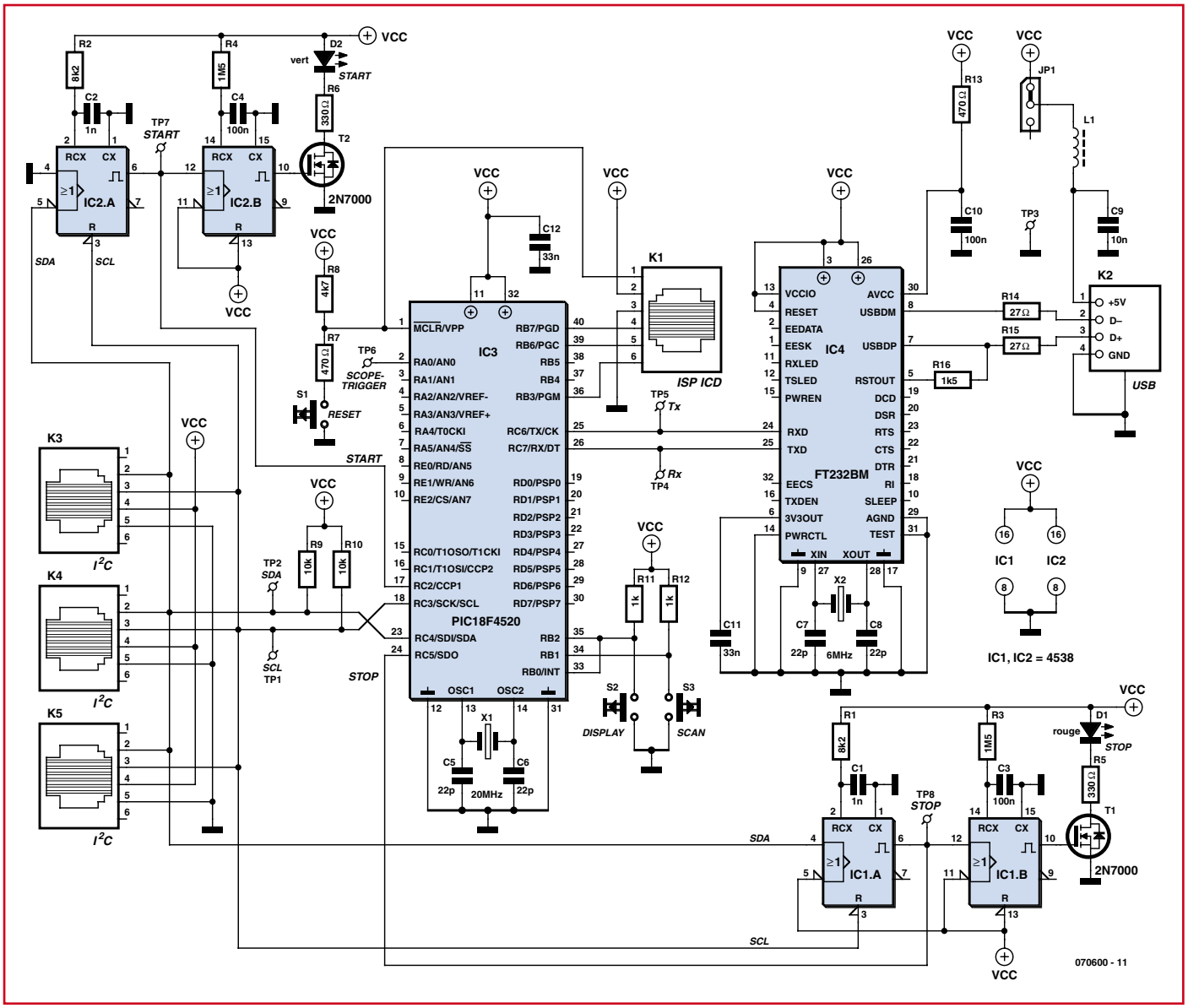


Figure 4. The circuit electronics don't amount to very much: a PIC/USB chip double-act surrounded by a handful of connectors of all types…

ply, thereby protecting the power supply from the PC. In connection with this, note the presence of a header + jumper JP1, which makes it possible to disconnect the PC supply.

PCB

It goes without saying that such a circuit merits a PCB. The screen-printed component overlay is shown in **Figure 5**.

The first step, and also the trickiest, consists of fitting IC4, the only SMD IC used in this project. As always with SMDs, soldering it requires a bit of care and a steady hand. The pads of IC4's LQFP-32 pinout have been lengthened to make it easier to solder this device using a soldering iron. Start by getting the orientation correct (pin 1 is the one immediately to the left of the round indent). On the component overlay, the position of pin 1 is identified by a small '1'. Start by soldering the two diagonally opposite legs. If the remaining legs line up properly with the remaining solder pads, they can be soldered quickly using a fine-tipped soldering iron and fine-gauge solder. Use a magnifying glass to check the quality of the soldered joints and that there are no shorts here. You can then go on to fit the remaining SMD components, in 1206 packages, followed by the small solder-through components, resistors, crystal, capacitors, LEDs and transistors (paying attention to polarities). Then you can fit the sockets (good quality ones!) for IC1, IC2, and IC3, and finish by installing the various sockets, RJ-11 for K1, K3, K4, and K5, and USB type B for K2.

All that then remains will be to fit push-buttons S2 and S3, and the reset button (S1).

Important: if you are powering the circuit via the analyser's USB port, which is the usual case, you need to fit the jumper in the 'on' position on header JP1.

After one last glance at the project to ensure there are no errors or shorts, now comes the moment to connect the board to the PC via a USB cable, to check for the presence of supply volts at the appropriate points on the sockets, with the aid of a multimeter. If everything is OK, you can disconnect the analyser and then fit the last ICs,

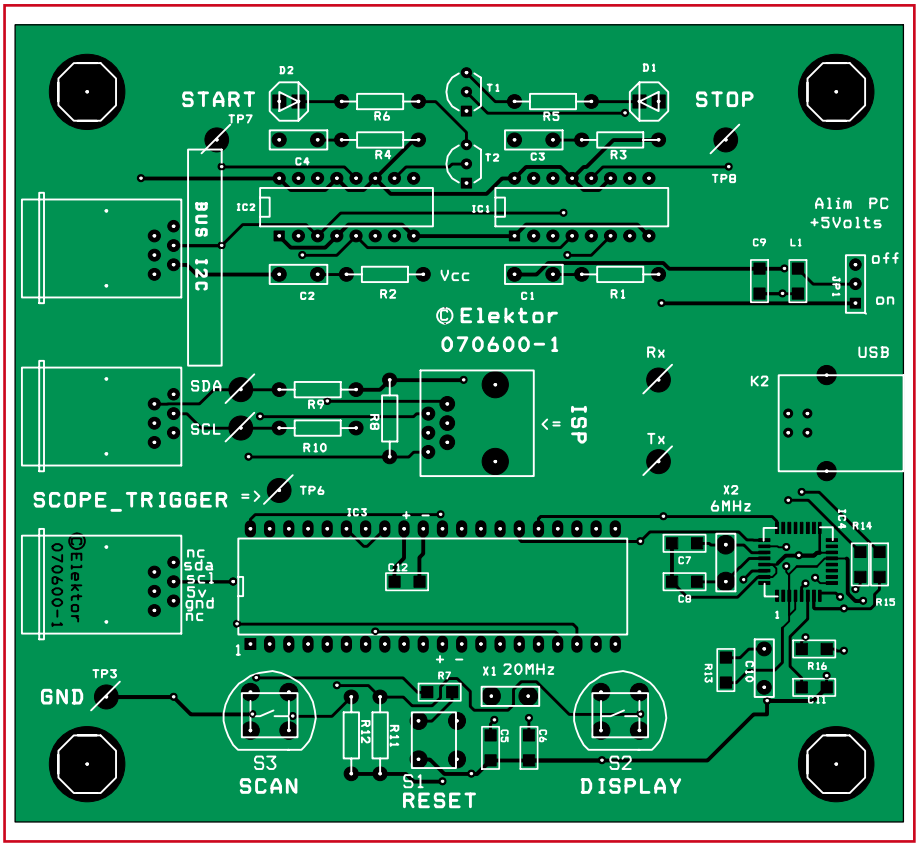


Figure 5. Component overlay for the I²C bus analyser.

IC1–IC3, watching out for their polarity. The board includes a number of test points (TP1–TP8 on the component overlay, test points TP1, TP2, TP4 and TP5 corresponding to the SCL, SDA, Rx, and Tx lines respectively), which can if desired be fitted with pins, as in our prototype.

For testing, the circuit can be powered at 5 V from the I²C bus connector or by wires soldered directly to the board.

Trouble-shooting the hardware part is made easier because the monostables are independent of the software: when the circuit is connected to an

COMPONENTS LIST

Resistors
R1,R2 = 8kΩ2
R3,R4 = 1MΩ5
R5,R6 = 330Ω
R7,R13 = 470Ω
R8 = 4kΩ7
R9,R10 = 10kΩ
R11,R12 = 1kΩ
R14,R15 = 27Ω
R16 = 1kΩ5

Capacitors
C1,C2 = 1nF
C3,C4,C10 = 100nF
C5-C8 = 22pF
C9 = 10nF
C11,C12 = 33nF

Semiconductors
D1 = LED, 3mm, red
D2 = LED, 3 mm, green
T1,T2 = 2N7000
IC1,IC2 = 4538
IC3 = PIC18F4520, programmed, Elektor shop item # 070600-41
IC4 = FT232BM (FTDI)

Miscellaneous
K1 = 6-way RJ-11 socket (vertical)
K3,K4,K5 = 6-way RJ-11 socket (horizontal)
K2 = USB socket, male, type B
L1 = ferrite bead
X1 = 20MHz quartz crystal, HC 49/4H case
X2 = 6 MHz quartz crystal, HC 49/4H case
S1 = miniature pushbutton
S2,S3 = 'D6' pushbutton (red and black)
JP1 = 3-way SIL pinheader with jumper
PCB, item # 070600-1
PCB artwork, free download from www.elektor.com
Project software (PC executable and .hex file), item # 070600-11, free download from www.elektor.com

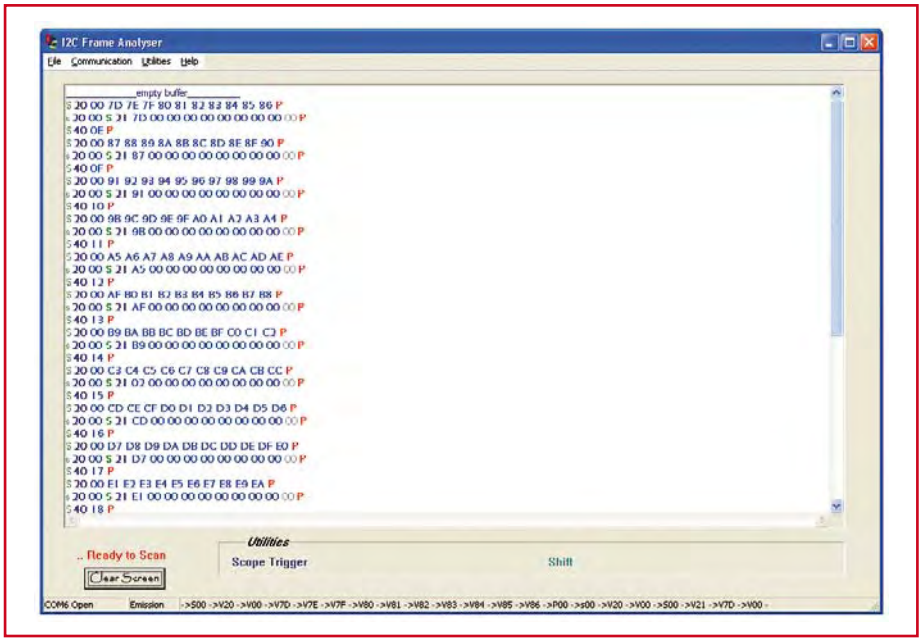


Figure 6. Screen dump of Monitor I2C program.

existing I²C bus, the four monostables ought to react to the arrival of a Start and a Stop by lighting the green and red LEDs respectively.
Then the PIC can be programmed.

Software

The microcontroller source program is written in C and compiled with the help of cross-compiler PCH compiler V4.010 from CCS (Custom Computer Services). This compiler tolerates a certain flexibility in the academic C language, and so is very well suited to programming by electronics technicians.
Development is carried out under MPLAB V.7.62. The program, which runs under Windows, is written in C++ Builder V5 (Borland). The software for Windows can also compile under CodeGear 11, the latest IDE from Borland, available in 30-day evaluation version. And there you have it all.
An ISP connector K1 is fitted in the middle of the circuit to allow debugging (ICD = **I**n-**C**ircuit **D**ebugging) and *in situ* programming (ISP = **I**n **S**ystem **P**rogramming) of the microcontroller.

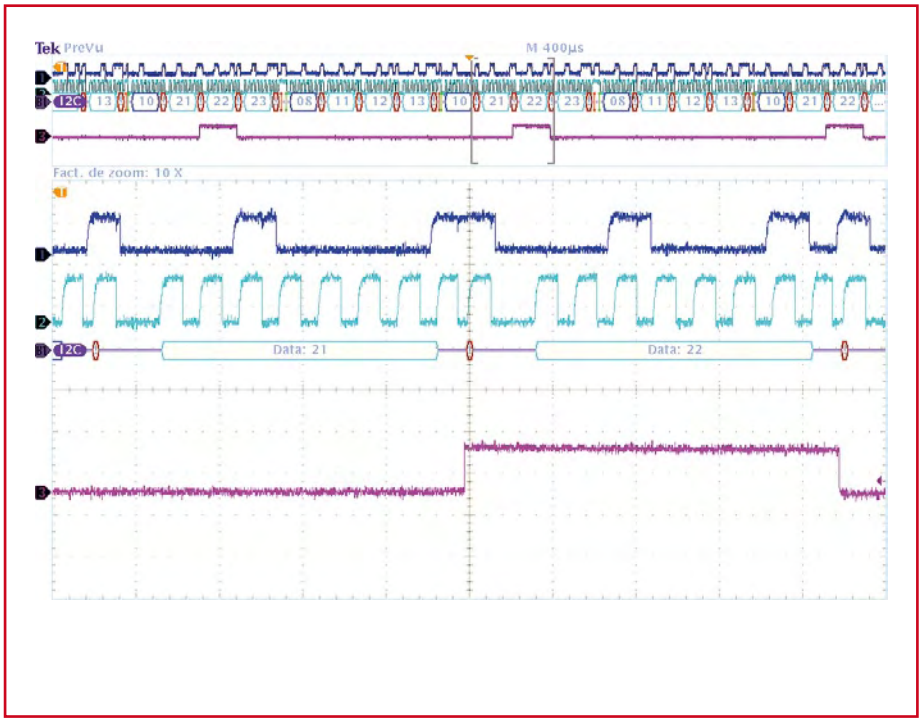


Figure 7. Scope_trigger screen dump

A quick glance at the software

Though the electronics are simple, the program loaded into the microcontroller needs to be all the more powerful. You can download it from our website (www.elektor.com) as archive file 070600-11.zip. Let's take a look at a few of its practical aspects.

The man/machine interface: installation

This application, written in C++ Builder V5.0, runs under Windows and is easy to install by copying the executable **monitor_I2C.exe**. This application requires prior installation of drivers on the PC. To do this, it's worth consulting the FTDI [3] website and the previous Elektor articles on this subject.

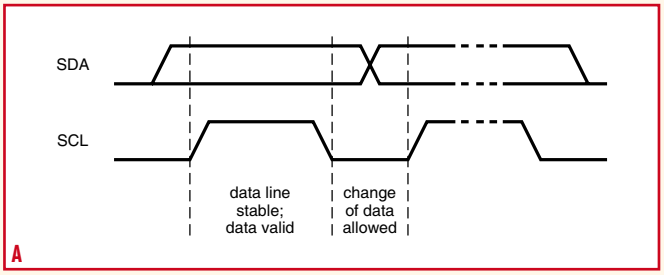
RS-232 configuration

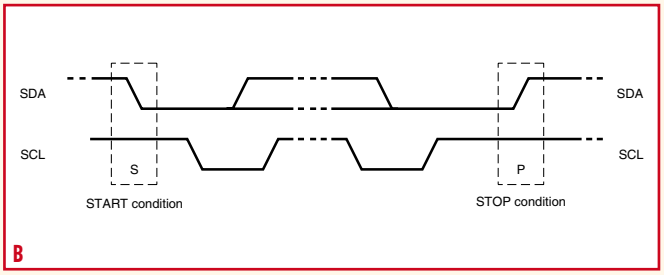
At start-up, an initial dialogue box lets you select the Virtual Com Port via which the USB link is going to receive the data sent by the PIC in the form of a standard asynchronous serial link (transfer speed 128,000 baud).
Then the status bar indicates that the serial port has been opened properly.

Displaying the results

Scanning is started by operating push-button S3 (SCAN). The I²C events are then recorded by the circuit and appear according to a colour code corresponding to the I²C event. Screen dump **Figure 6** shows the main screen when acquisition is finished; in it you will be able to recognize the STARTs in green, the STOPs in red, the addresses in ultramarine blue, and the data in royal blue. However, this display will be modified by the value of the acknowledge bit: if it is present, we see these two in blue, but if it is absent, they will be greyed out.
The status bar also shows the format of the codes transmitted on the serial link between the circuit and the PC (ASCII coding).
Example:

The secrets of I²C and its bus





The I²C (Inter Integrated Circuit) bus was dreamt up by Philips, at the time when they were one of the leading manufacturers of audio equipment. Its major fields of application were home automation and home electronics in the early 80s. Microprocessors were putting in an appearance in TVs, and they needed to find a cheap, easy technique for interconnecting the various electronic sub-assemblies in such devices.

The I²C bus is a synchronous serial bus with just 3 lines: Data, (SDA), Clock (SCL), and ground (used as a reference).

Operation of the bus is based on the concept of master (the peripheral that manages the communication, generates the clock, and transmits the data) and slave (the peripheral that receives the data and confirms reception by an 'acknowledge' signal). But don't let's be deceived: despite its rustic simplicity, this bus can handle several microcontrollers without conflicts, as long as certain rules are properly obeyed.

The four most important situations in this protocol are illustrated below.

1) Transfer of a bit onto the I²C bus (Figure A).

The clock doesn't really have the 'form' of a real clock, since it can have variable mark/space ratios (within constraints).

2) START and STOP conditions (Figure B).

At the start of communication, the SDA line goes to '0' while the SCL lines is at '1'. This is the StartBit. At the end of communication, when the SDA line has gone back to '1' and the SCL line is also at '1', we have the StopBit.

3) Transfer of data onto the I²C bus (Figure C).

This condition having been defined, the master places the MSB on the SDA line. It validates the data by briefly forcing the SCL line 'high'. It continues in the same way for the various bits right down to the LSB. The transmission over, the slave forces the SDA line low, this is the…

4) Acknowledge signal on the I²C bus (Figure D).

The slave component issues this signal to indicate reception of all the data. If everything is OK, it forces the line to '0'.

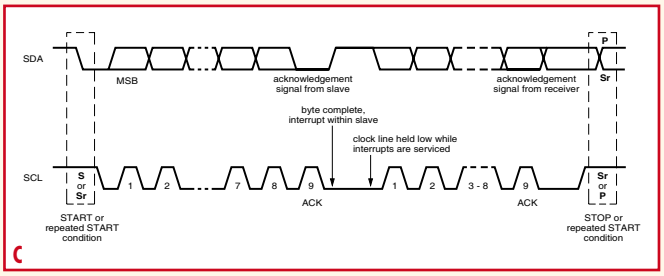
Despite its simplicity, the I²C bus allows handling of relatively complex operations. The various scope traces given in the article illustrate this operation very well.

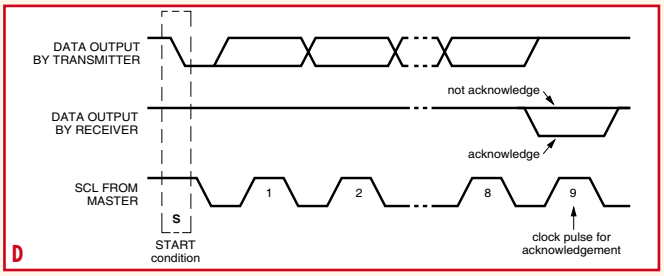
It's worth noting that a master-component can also receive data from a slave (master-receiver).

If you want to find out more, all is explained in the I²C bus specifications from January 2000:

http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

(Illustrations: Philips Semiconductors)





->S00 for the START
->V20 for an acknowledged byte with a value of 20 (HEX)
->v20 for an unacknowledged byte with a value of 20 (HEX)
->P00 for the STOP.

Oscilloscope synchronization function

This utility lets us configure the set-up with a synchronization byte, which when it is present in the frame triggers a synchronization pulse on the SCOPE_TRIGGER pin to synchronize a scope in external trigger mode. This makes it possible to display the shapes of the SDA and SCL signals just at that instant. For example, the screen dump in **Figure 7** represents the SCOPE_TRIGGER signal for the sync byte 21 (HEX). The top part of the trace clearly shows how each time this byte occurs, the signal toggles for the duration of the next byte. The central section shows the magnified portion with a zoom factor of 10 and reveals the transition of the SCOPE_TRIGGER signal at the moment of the ninth SCL clock pulse. This instant, which normally corresponds to the reply from the receiver, enables us to read a '1' on SDA, indicating that there has been no acknowledgment: from that moment on, the I²C circuit with the address 10 (HEX) for which data bytes 21 22 23 were intended is considered as absent from the bus or defective.

Delay function
Another utility function makes it possible to delay the start of recording of a certain number of events; in this case they are replaced on the display by a dot.

I²C summary
A built-in help page includes a brief summary of a few definitions of events present on an I²C bus. In addition, it shows the recording of real signals.
To transmit data over the I²C bus, it is necessary to monitor two specific conditions: Start and Stop. The Start condition corresponds to a falling edge on SDA while SCL is high.
The Stop condition corresponds to a rising edge on SDA while SCL is high.
Then, eight pulses supplied by the clock allow sampling of the eight bits of the byte, starting with the MSB.
The ninth clock pulse allows a response, an acknowledgement by the receiving component of the preceding byte. If the component is present, then it acknowledges by taking the SDA line low. This is the principle of 'handshaking'. If not, the line remains high, and the transmitter of the byte may react.

And what else?

Not a lot! All you have to do is connect an application's I²C bus to the I²C bus analyser, connect the latter to a PC, start the **monitor_I2C.exe** program, press the SCAN button, followed a few moments later by a press on S2, DISPLAY, and wait for the first data to appear on the screen.

Conclusion

This simple-to-use circuit using standard components (CMOS logic ICs, PIC microcontroller, USB interface) makes it possible to analyse the signals present on an I²C bus. One further development of the circuit might be to fit a PIC with a USB stack (PIC18F4550). This solution would simplify the hardware (by eliminating the USB interface IC and the 6 MHz crystal) and improve speed, as the PIC 18F4550 uses a PLL to generate a clock at 48 MHz. The disadvantage would be in the increased complexity of the software. Libraries provided by the publishers of C cross-compilers do exist and provide numerous source files (MPLABC18Compiler, CCS) but overall debugging is likely to be trickier.

This practical tool will let you to see what's happening on the bus of the datalogger project described elsewhere in this issue, as it too has an I²C bus interconnecting the real-time clock to the rest of the system. Happy hunting!

(070600-I)

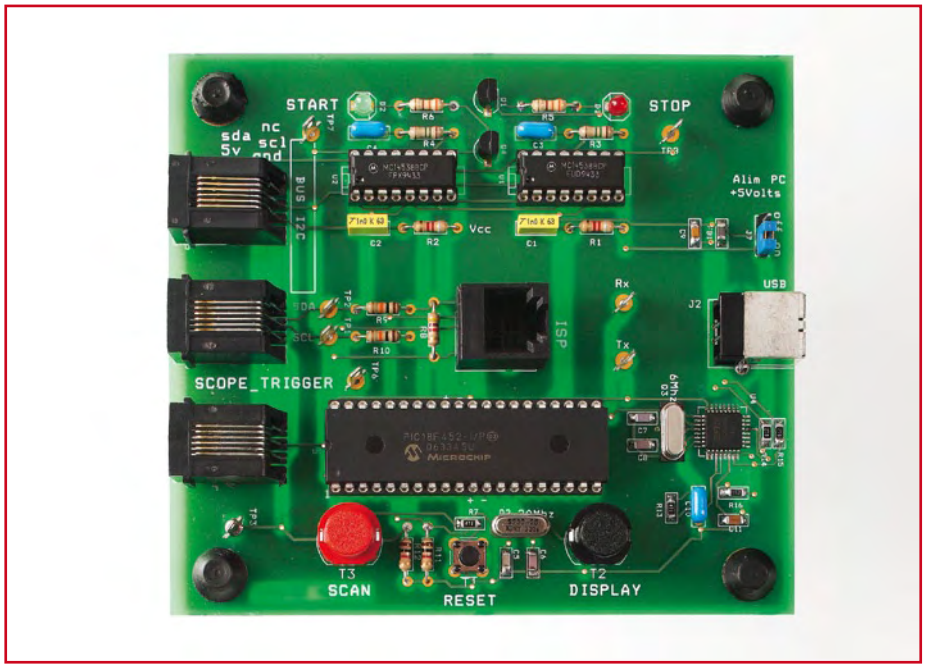


Photo of the prototype. Note that component designations have been changed in the final version.

The author

The author studied as an engineer at the INSA in Lyons and then moved into teaching, passing the competitive examination for National Education.

Teaching electronics to students in the BTS (Higher Technician's Certificate) section for many years, he has experienced and passed on the fantastic evolution in technology: discrete components (2N2646 unijunction transistor!), memories, microprocessors, mainframes, then the arrival of personal computing.

Training has also evolved, nowadays relying on an understanding of complex electronic systems: installation, configuration, and troubleshooting.

And sometimes even now, faced with a successful project, he still feels that good old maxim: What a fine profession it is to be a teacher!

Etienne Boyer

Web Links and bibliography

[1] FT232BM Data Sheet:
www.ftdichip.com/Documents/DataSheets/ds232b18.pdf

[2] PIC18F4520 Data Sheet:
ww1.microchip.com/downloads/en/DeviceDoc/39631a.pdf

[3] FTDI website:
www.ftdichip.com

[4] CCS compiler:
www.ccsinfo.com