Swiss Army

Jim Spence

In a fix? Need to get a microcontroller project off the ground FAST? This article proves that an ultra-versatile microcontroller board can be built and programmed even by relative newcomers. The main circuit is based on the Atmel 89C8252 which has an 8052 architecture.

Knife

Tiny BASIC, 8051 assembler, RS232 and USB all in one project

Included is some very special (free) software that enables even the most code shy among you to make the controller do something. There are two alternatives to communicating with the circuit: USB 2.0 or RS232.

The original intention of the project was to produce a controller with built in BASIC that could retain the program after switch off, automatically start at switch on and not consist of too many chips. It should also be very easy to use and not require any special software on the PC. This has been achieved with more or less a single chip. This is possible because the 89C8252 has 2 kB of data EEPROM that can be programmed with high level instructions. The AT89C8252 micro was first used in the Elektor *'51 Flash Micro Board* published in December 2001 and now a 'classic' with PCB sales in the 'k' range.
You may well be wondering why not use a BASIC or C compiler on your PC and blow the object code into the micro. These are good options, however you loose the immediacy of your actions. The Swiss Army Knife board *immediately* executes any code sent to it. Typing:

```
Pz1=0
```

at the console will immediately set all of the port 1 lines to Low. Also, a level of knowledge is required of the BASIC and C compilers before anything can be achieved. If you simply want to get on with your favourite robot project, then this is the way to do it. Mind you, you can still use the compilers or assembler later when you have done some experimenting.

RS232 and/or USB

Along the way the author got fed up with incorporating a dedicated communication system with every board especially when it was not usually needed in the final application. This lead to a separate power supply and RS232 connectivity that could be used with other circuits. However, as the RS232 format is getting a bit rusty and is not even included on some modern PCs it was considered nice to produce a USB interface as well and of course the USB can power the circuit. The result of the design effort is that you, the user, may choose between RS232 and USB when it comes to talking to the micro in your Swiss Army Knife.
The Swiss Army Knife board consists of three sections: microcontroller (MCU), RS232 interface and USB interface. Each of these will be discussed separately below.

Microcontroller circuit

The MCU circuit shown in **Figure 1** contains the popular AT89C8252 microcontroller clocked at 22.118 MHz and programmed with a version of Tiny Basic called *Tiny Control Basic* (TCB). The circuit shown is about as minimal as you can get. It is possible to program the code memory allowing a mix of high-level and assembler can be used. To facilitate this IC2 has been included although the chip could be omitted in the finished application or if in-circuit programming is not required.
Regarding connectors, K1 takes all of the I/O lines to a 40-way header and K2 is a 14-way header that mates with either of the two other boards. Pins 1 and 2 form the serial communication. The handshaking pins are not involved in the serial communication.
The AT89C8252 can be programmed serially using just three lines: SCK (clock), MISO (output) and MOSI (input). Using these three lines and sending special bytes it is possible to get direct access to the code memory without removing the chip from the circuit, hence the term *in-circuit programming*. Pin 5 on K1, the DTR line, initiates this process and the other pins are as described in **Table 1**. The DTR signal enables the top half of IC2, a tri-state inverting buffer. All of the lines are inverted by IC2 so the software must take this into account. The ByVac terminal described further on has suitable software built in.
Along with the in-circuit programming there are various other signals available that apply to the USB interface. Pin 6 is the Ring Indicator. Providing the PC is set up correctly and the USB board is also configured correctly, this line can wake up a sleeping PC. By taking the line low the USB chip will send the correct signals to the PC to activate the PC from a suspended state. For this to work properly the microcontroller cannot be self powered, as the current available on standby is insufficient to operate the microcontroller without that also being in power down mode. In the power down mode the microcontroller will not be able to initiate a signal.
The USB specification has strict control over how much power can be taken from it and at what time. A normal USB is capable of providing a limited amount of current (100 mA) until it is properly configured. Once it is, up to 500 mA can be delivered by the USB bus. Pin 7 goes low to indicate that the USB is properly configured. This action enables the lower half of IC2 and as a result the PWR LED illuminates indicating that PWR is active and that current can be drawn from it. Power to external devices is taken from pin 8. The USB circuit (see later) will not provide any power to this pin until it is configured properly.
Pin 9 is a signal from the USB interface that goes low during suspend. The user can monitor this line. Pin 14 is power provided from the USB bus, con-

Features

- Built-in integer BASIC (TCB)
- BASIC can call assembler programs
- Instant code execution, no compiling phase
- Only needs a simple terminal emulator to program
- Auto Baud detection
- User BASIC program, line 10 runs on reset
- Can program code memory in-circuit using ByVac-Terminal
- I/O connectivity
- USB 2.0 option
- RS232 option
- All software available free of charge

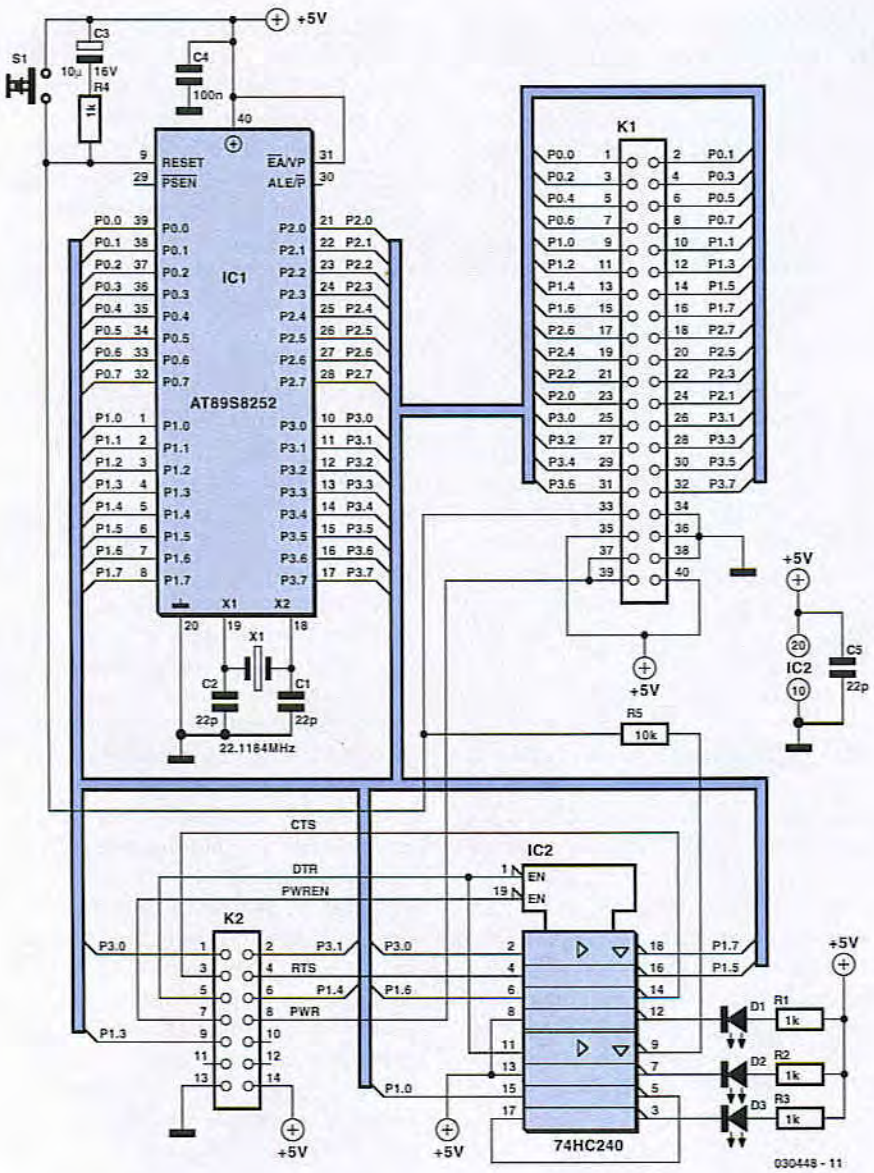


Figure 1. Circuit diagram of the microcontroller main board (MCU). It employs the 89S8252 Flash micro from Atmel.

figured or not. This is only capable of supplying about 100mA when the bus is not configured. For this reason, external devices must use the PWR lines rather than Vcc.

By monitoring the above lines IC2 can be put into power down mode when the PC goes into the suspend state. $\overline{RI}$ Can be used to awaken the PC provided the USB and PC interfaces are set up correctly. To do this IC1 must also be self powered.

The P1.0 LED is connected via two inverters to the P1.0 I/O line of IC1, this can be used for anything. It is useful to have something there for testing purposes.

USB interface

The USB interface shown in **Figure 2** uses a chip from Future Technology Devices Intl. Ltd. (FTDI) and the circuit follows their data sheet. The FT232BM is designed as a quasi-replacement RS232 interface and when installing the device drivers, it becomes a COM port on the PC. Drivers for other operating systems are available. The chip and associated software drivers remove any hassle from building a USB interface.

There are only a few aspects of this circuit worthy of attention. The first is the N channel MOSFET that supplies power to pin 8 of K2 when the $\overline{PWREN}$ signal is low. It is set up this way so that external devices can be powered from the USB bus when the FT232BM is initiated. The initiation process is called enumeration. The other part of the circuit is IC2. This part is optional and is only required if you need to use the special features of the chip, USB 2.0 for example, or are using two chips in the same system. The circuit will work perfectly well without it.

Before going any further it is helpful to know something about the USB interface. The interface is incredibly useful in that not only does it provide a high-speed interface but power as well.

When first plugging in the device it must draw no more than 100 mA until the device is enumerated. Once enumerated the bus is capable of supplying up to 500 mA. In suspend mode this reduces to 500 µA. All USB devices have a unique, registered ID. The FT232BM has a built in, default ID. For

Jim Spence

Jim was born in 1953 and has a degree in Construction (1979). He is currently an IT Project manager working for a global IT company. Jim's last article was published 10 years ago in Electronics Today International. That was another single board computer based on a Z80 which ran Forth. Before managing projects Jim was a Lecturer in Computer Aided Design. His main interests are electronics and software languages. Jim says he goes back to the time of valves and has been lucky enough to watch the introduction of transistors, integrated circuits and then see the personal computer industry grow from nothing.

jim@byvac.com

commercial use the ID must be registered. Also if you intend to use more than one device on the same system then each device must have a unique product ID. It is possible to set all of this up by fitting the EEPROM IC2.

Once enumeration has taken place, which is all taken care of by the FT232BM chip, the PWREN line is taken low. This is passed on to pin 7 of K2 so that it can be monitored by the microcontroller if necessary. It also activates T1 which in turn allows 500 mA to be drawn from the USB bus via pin 8 of K2.

There are two main types of socket: 'A' and 'B'. The 'A' socket is a power provider and is the type fitted to the PC. The 'B' type is a power user and to our knowledge comes in about three different shapes. The one used on the PCB is the most common although smaller ones can be obtained with difficulty, these are usually found on cameras and the like.

The FT232BM requires a device driver, even on an XP system. All the device drivers, software and documentation required for the device can be obtained from the FTDI website. The driver needed is the Virtual COM Port (VCP). This will enable you to use the device as if it were a COM port.

The EEPROM device, if fitted, can be programmed using the D2XX drivers and one of the many software utilities provided at the site. Note **the drivers will no co-exist**, you need to uninstall one to use the other.

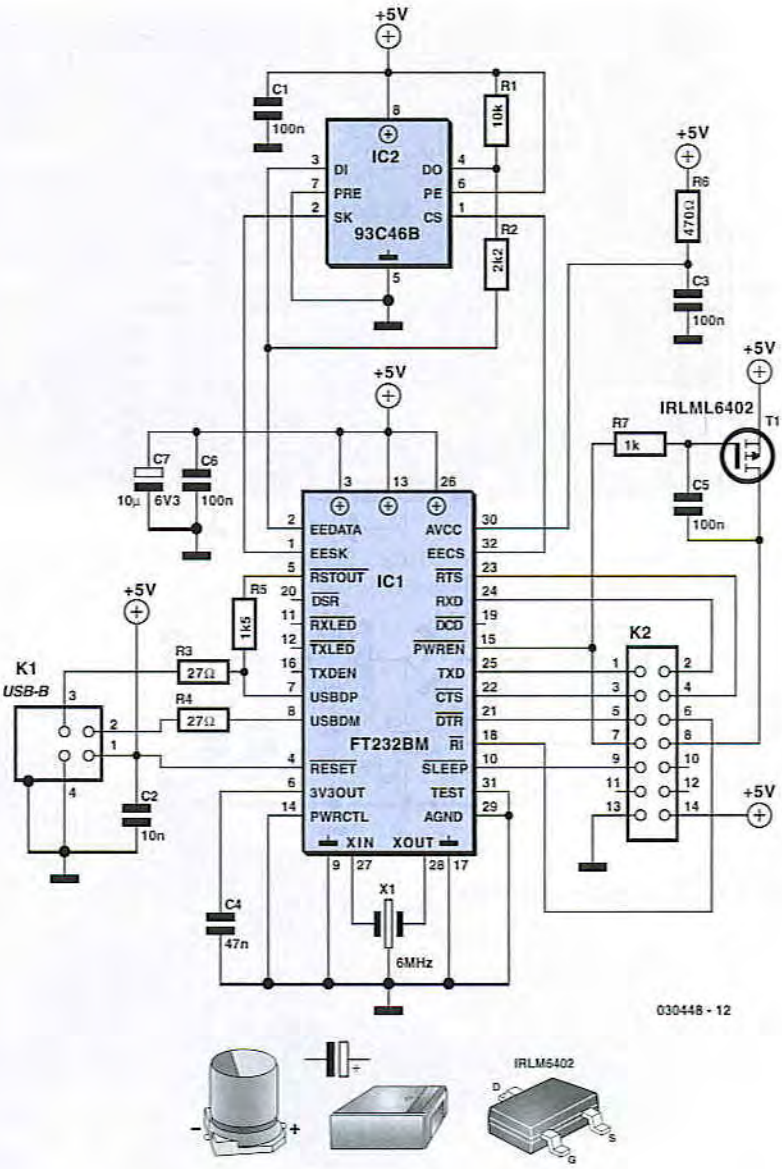


Figure 2. USB interface schematic.

Good old RS232

This is for those of you who do not fancy USB. The RS232 interface circuit diagram appears in **Figure 3.** As RS232 port is not capable of enough supplying power it is necessary to incorporate a simple 5 V supply into the design. Bridge rectifier B1 may strike you as unusual but is well worth the extra cost, not only can you use an AC output power supply but also it does not matter which way round the polarity is on a DC power supply — just plug it in and it works.

Power from the supply is taken to pin 8 on K2 to indicate to the main board that power is available. Pin 7 is permanently at ground indicating to the main board that power is available. The circuit

COMPONENTS LIST

PCB, order code **030448-1**
Project software on two disks, order code **030448-11** or Free Download.

MCU board

Resistors:
R1-R4 = 1kΩ
R5 = 10kΩ

C1,C2 = 22pF
C3 = 10µF 16V radial
C4,C5 = 100nF

Semiconductors:
D1,D2,D3 = LED, low current, colours to personal taste
IC1 = AT89S8252-24PC, Dip40 case, programmed, order code **030448-41**
IC2 = 74HC240
K1 = 40-way boxheader (two pin rows)
K2 = 14-way socket, angled pins, two receptacle rows
S1 = pushbutton, 1 make contact, miniature
X1 = 22.1184MHz quartz crystal

USB interface

All parts SMD, case shape 1206

Resistors:
R1 =10kΩ
R2 = 2kΩ2
R3,R4 = 27Ω
R5 = 1kΩ5
R6 = 470Ω
R7= 1kΩ

Capacitors:
C1,C3,C5,C6 = 100nF
C2 = 10nF
C4 =47nF

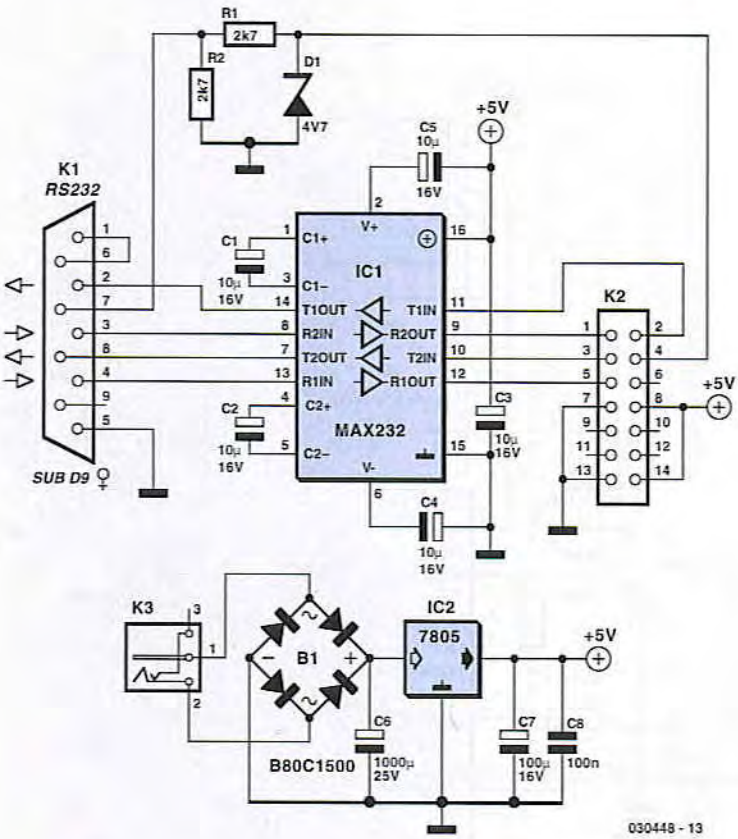


Figure 3. RS232 interface schematic.

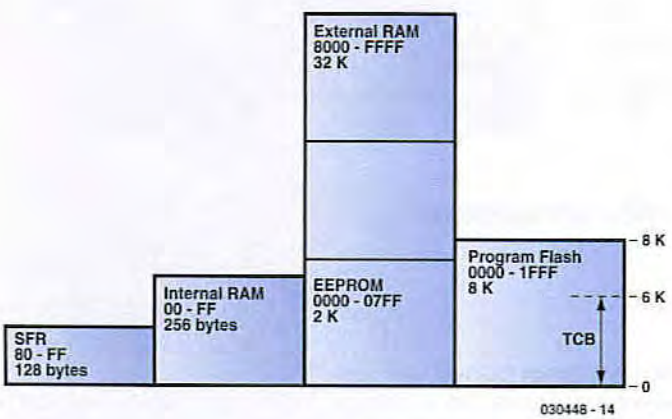


Figure 5. Harvard memory structure applicable to the 89S8252 Flash controller in this project. TCB occupies 6k of the 8k available in the Program Flash area.

formed by R1, R2 and D1 limits the voltage on the RTS line and make it suitable for feeding into IC1 on the main board.
All of the I/O lines are used on IC1 in order to facilitate the in-circuit programming. Only TXD and RXD are required if the in-circuit programming is not required.
Some of the functionality of the USB board is not available to the RS232 interface, so pins 6 and 9 of K2 are left unconnected.

Construction & test: MCU and RS232...

If you look at the PCB artwork shown in **Figure 4**, you'll notice that the MCU, USB and RS232 sections are supplied as one board, order code **030448-1** from our Readers Services.
Depending on the connectivity you have in mind for the Swiss Army Knife you will have to populate the USB or the RS232. Sure, you can build both sections but do remember you can't use them at the same time.
The MCU and RS232 sections are nothing special when it comes to building them — simply work along the lines indicated by the parts list and the component overlay. Sockets are of course recommended for the ICs. The 14-way connector can be cut from a larger one if required. Voltage regulator IC2 does not require a heat-sink.
Before inserting the ICs into their sockets, connect the MCU board to a 5-V supply and power up. Check with a meter that 5 volts appears at the correct polarity across pins 20 and 40 for IC1 and 10 and 20 for IC2. If all is well disconnect the power, insert the IC's and re-connect the power. If you have a logic probe or oscilloscope, monitor pin 2 on K2 (pin 11 on IC1). Press reset and after 1 or 2 seconds the sign-on signal should be seen. This will be a short burst of pulses at 9600 baud.

C7 = 10µF 6.3V SMD

Semiconductors:
T1 = IRLML6402
IC1 = FT232BM (FTDI, www.ftdichip.com)
IC2 = do not fit (93C46B SO8)

Miscellaneous:
K1 = USB connector, type 'B', PCB mount
K2 = 14-way socket, angled pins, two receptacle rows
X1 = 6MHz ceramic resonator, 3 pins

RS232 board

Resistors:
R1,R2 = 2kΩ7

Capacitors:
C1-C5 = 10µF 16V radial
C6 = 1000µF 25V radial
C7 = 100µF 16V radial
C8 = 100nF

Semiconductors:
B1=B80C1500, round case (80V piv, 1.5A)
D1 = 4.7V zener diode, 500mW

IC1 = MAX232 (Dip16 case)
IC2 = 7805

Miscellaneous:
K3 = mains adapter socket, PCB mount, angled pins
K1 = 9-way sub-D socket (female), angled pins, PCB mount
K2 = 14-way angled pinheader, two pin rows

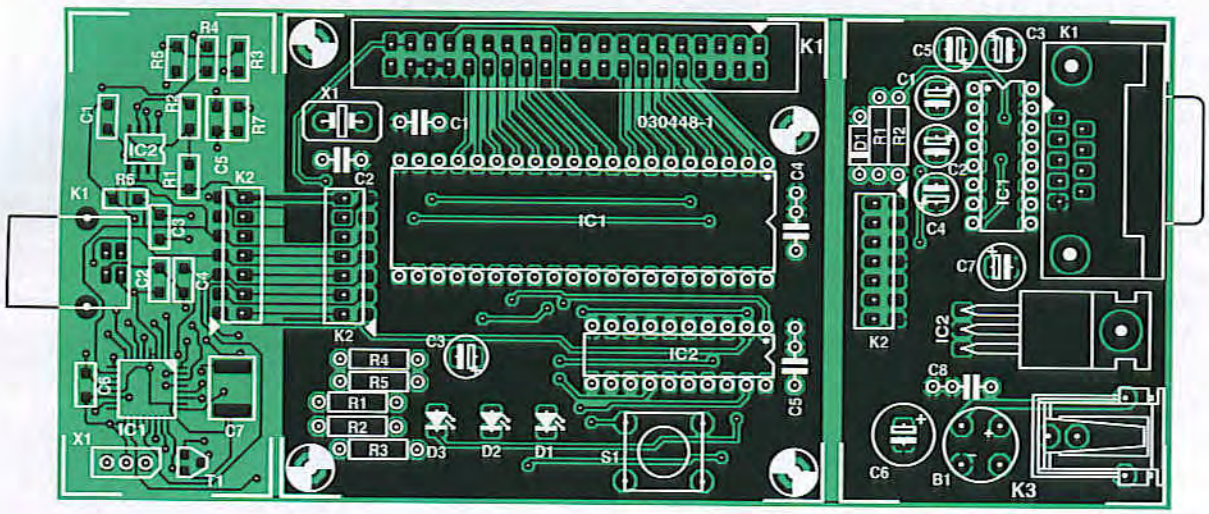


Figure 4. Component overlay for the combined MCU / RS232 / USB board, order code 030448-1. If you want to use USB either leave the USB interface section attached to the MCU or separate the two and connect them with a flatcable.

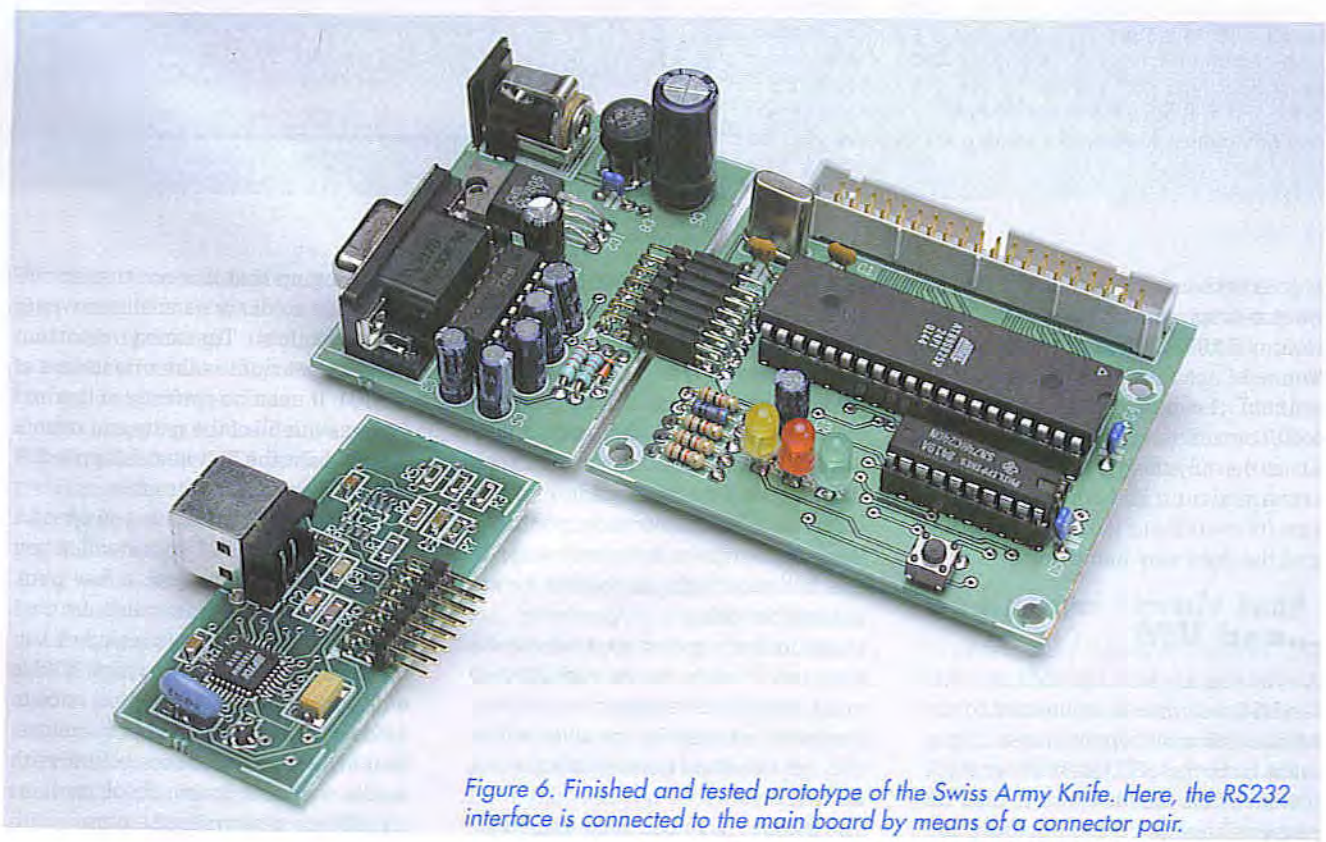
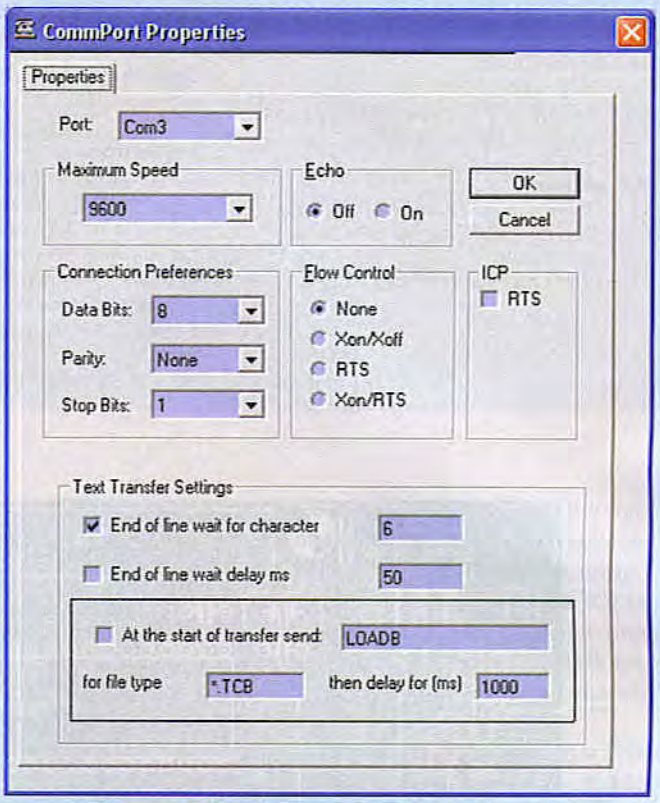
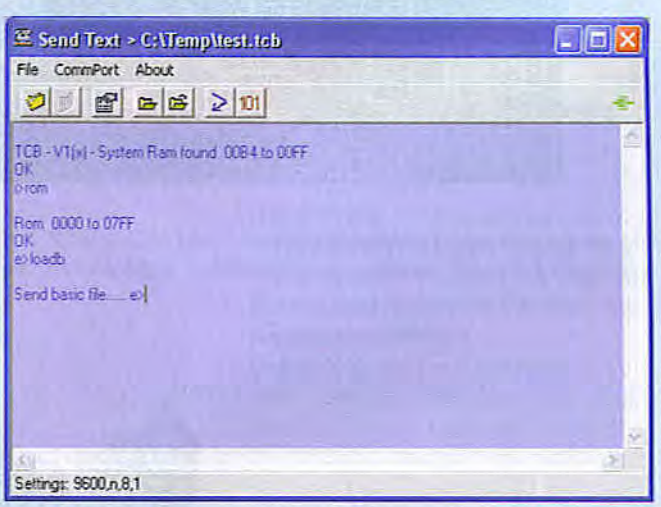


Figure 6. Finished and tested prototype of the Swiss Army Knife. Here, the RS232 interface is connected to the main board by means of a connector pair.

ByVac-Terminal



A



B

ByVac-Terminal does have some built-in features to get maximum performance and ease of use from the Swiss Army Knife. Version 1.0 is free from the Free Downloads section of our website: www.elektor-electronics.co.uk. As an example, to download a program from your text editor you can simply use 'Send File'. If however you are using the EEPROM memory space then there is a noticeable delay while the program line is written. An option on some terminal emulators is to send line by line and insert a delay between lines. This works fine but you have to cater for the worst case meaning the download is much slower than it should be.

Built into TCB is the 'LOADB' command that has a very simple protocol and it works like this. After issuing the command TCB waits for a line of BASIC to be sent, when the line is received it is processed and an ASCII code 6 (ACK) is sent back to the terminal to indicate it is ready for the next line. This simple protocol works exceptionally well and the ByVac terminal has this protocol built in.

Another advantage is that it is capable of using the in-circuit programming features so assembler code can be written and downloaded into the code space.

We've already seen how to turn the on board LED on and off using a simple program that you typed in at the terminal. You can verify that everything is okay by typing:

```
Pz1=254
Pz1=255
```

The above action should have turned the LED on and then off again. You can edit TCB programs by re-typing the line, to remove a line simply type the line number. This however can become very tedious. A much better way is to create and edit the program using "Notepad" or a similar text editor and then downloading this to the board.

Open your text editor and enter the program below. This is very similar to the program you used in the TCB introduction. Note that at lines 20 and 40, instead of assigning a value directly to port 1 a logical operator ('and' 'or') is used. This will have the same effect but it does not affect any of the other lines on the same port. Observe also the

It goes without saying that IC1 should be pre-programmed with TCB (Tiny Control BASIC) for the above to work. There is not much else you can do without the other two boards; the MCU circuit is so simple that just about the only thing that can go wrong is the soldering and component location (is everything in its correct place and the right way round).

...and USB

As you can see from the PCB artwork, the USB interface is connected to the MCU section by copper tracks. If you want to fit the USB interface at some distance from the MCU board, the PCB sections have to be separated by cutting or sawing and a small connection cable installed between the respective connectors.

Now for some bad news. The FT232BM chip only comes in a surface mount version and it's a small one at that. You need to be brave to build this but it can be done with a simple soldering iron, solder wick and some solder paste.

We would urge you to have a go at this, it's not a beginner task but it's not impossible either.

Unfortunately, solder paste is expensive but it does make the process much easier. An alternative to using the paste is to use far too much solder and get rid of the excess with the solder wick.

The solder wick does such a good job at tidying up that it is not that important if the solder goes in all the wrong places at first. The most important thing to get right is the orientation of the IC1. It must be perfectly in line and square with all of the pads and remain there whilst the first heat is applied. If it slips then you're in trouble.

We would recommend that you spend a lot of time lining IC1 up and when you are satisfied, solder just a few pins, (one pin if you can) the minimum coverage of the author's fine tipped soldering iron is two to three pins. If all is well solder the rest of the pins, once in place it is almost impossible to remove. Remove all of the excess solder with solder wick and double check no short circuits exist between the pins.

"end" without a line number at the bottom of the program. This will tell TCB that the download has finished. In practice, if you forget to do this it still works okay.

```
10   for j = 1 to 10
20   pz1 = pz1 and 254
30   gosub 500
40   pz1 = pz1 or 1
50   gosub 500
60   next j
70   end

500  for k = 1 to 20
510  next k
530  return
end
```

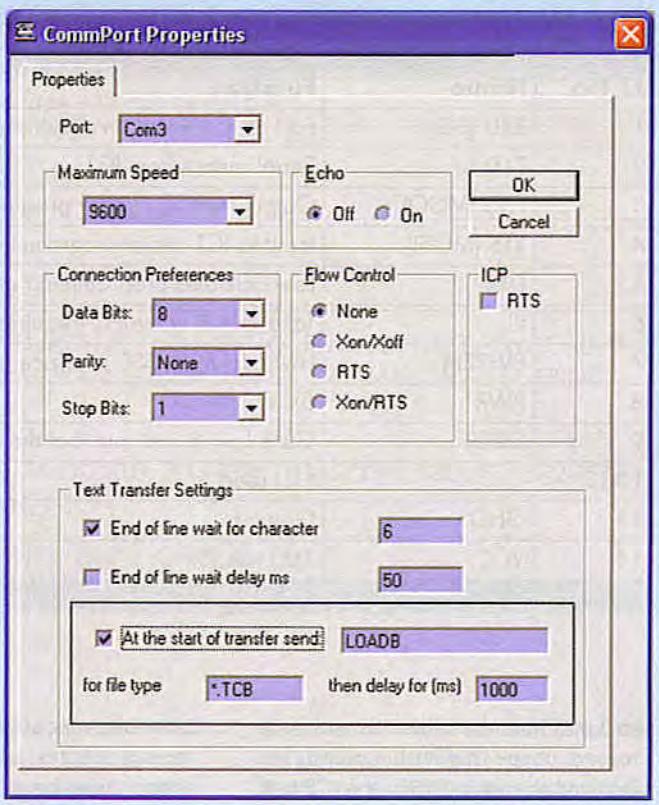
Save the program with the extension TCB, e.g. "LedFlash.tcb" If you are using Notepad be careful to select all file types before saving otherwise you end up with a file called "LedFlash.tcb.txt". This only happens on the first save but it is something to be aware of.

As indicated TCB has a very simple but effective file transfer protocol that the ByVac-Terminal takes advantage of. Start the terminal and use the initial settings as shown in **Figure A**. Note that the "End of line wait for character" is checked. At the terminal type LOADB as **Figure B**.

TCB is now ready to accept a basic program using the simple protocol. Use File and "Transmit text file" or use the forth icon from the left. Select your file "LedFlash.tcb" and it will be loaded into TCB.

The program will load, type RUN to see the fruits of your efforts. To speed things up if you alter the text transfer settings to that of **Figure C**. The terminal will now type in LOADB for you if you select a file with an extension "tcb". Changes to the RS232 settings require the connection between the PC and the Swiss Army Knife to be broken and then restored again.

Use the fifth icon from the left to reload the same file. The



cycle for development then becomes:

1. Edit program
2. Save
3. Use the reload icon
4. RUN
5. Back to 1

If you want to stop the program running use CTRL-C or simply press reset. The beauty of an interactive system such as this is that everything is immediate, you can try things out at the command line and it will be instantly activated. At the end of development you have your finished product.

Two dice programs, one simple and the other, well, slightly less simple, to test the above sequence may be found on our website.

The other components are not too bad. They were chosen for their large size (relatively speaking). The connector K2 is a pinheader that will probably need cutting from a larger one.

Before plugging in the device to the PC, download the FTDI device drivers and unzip to a suitable directory. If all is well when you first plug the device in you will be asked for the device driver. Take some time after building to inspect the circuit for shorts, use a meter if necessary. Plug the device into the USB port on the PC or preferably onto a hub to prevent any possible damage to the PC. Although the USB specification calls for short circuit protection, you never know, and a new hub is much cheaper than a PC. If it's any consolation the prototype circuit was not checked well enough the first time it was plugged into a PC and all the USB devices suddenly stopped working, this included the mouse. It took a reset of the PC to restore things back to normal — not a pleasant experience.

Assuming all is well the PC will detect the new device and request the location of the device drivers. Install this just as you would any device driver. If you are not sure how to do this look on the FTDI website for information, installation instructions also come with the device drivers for Windows and other operating systems.

All being well you should now have a new COM port. To find out which port number has been allocated depends on the operating system. For Windows XP this is in the Control Panel → System → Hardware → Device Manager then open up the Ports tree by clicking on the + sign. You should see a new port. If not, re-install the device driver and make a note of any error messages.

Complete circuit test

Connect the two boards together, use either the RS232 circuit or the USB circuit. Launch the free ByVac terminal utility, see the 'ByVac-Terminal' inset. If using the RS232 board you need a straight-through (1:1) cable where pin 2 goes to pin 2 and pin 3 goes to

Table 1 Programming & USB interface		
J1 Pin	**Name**	**Function**
1	RXD (SCK)	Part of IC1 serial, in-circuit programming interface. This is the clock line.
2	TXD	Serial output from IC1
3	CTS (MISO)	Output from IC1 serial programming interface.
4	RTS (MOSI)	Input to IC1 serial programming interface
5	DTR	Low activates programming mode, also low and back to high again will reset IC1
6	RI	Take low to wake PC (requires special set up)
7	PWREN	Low o/p from USB interface indicates that 500mA is available on the PWR line
8	PWR	5V @ 500mA
9	SLEEP	Goes low to indicate that the connected PC has gone to sleep
10-12		Not used
13	GND	Ground
14	VCC	100 mA, 5V

pin 3 etc. In some cables the pins are crossed, check this with a meter. You also need to enable RTS in the ICP box. Press reset (S1), wait for a few seconds and you should see the sign-on message. If not, check that the sign on message is coming from IC1. If there is a signal and you still do not see the sign-on message check the settings, Baud rate etc., check the connectors, cable and wiring.

Here's TCB

At the heart of this project is the TCB (Tiny Control BASIC) software that will incidentally run on any 89C8252 system with or without external RAM. If there is any RAM present it will automatically detect it, obviously there is no RAM in this project.
There are three ways you can get the TCB software onto IC1:

1. use the programming interface;
2. buy a pre-programmed chip, order code 030448-41 from Readers Services;
3. use an 89C8252 programmer.

Although possible, option 1 is not recommended, the programming interface and software being designed for short programs(<100 lines), also it will mean that you can't properly test the circuits until you have done this so how do you know if the circuit has been built correctly? It is possible however but will take about 25 minutes, see the Assembly Code inset. Options 2 or 3 are recommended if you are building the circuit for the first time.
This processor and its architecture have been mentioned many times in various articles, however the memory aspect must be at least partly understood so it is briefly discussed here.

Memory

The memory space of the 8052 uses an architecture that shares parallel areas of memory. By logically combining the OE and PSEN signals to access the RAM it means that the external RAM can be used as program memory. The 8051 architecture is old and we're sure that separating out the data and program memory seemed like a good idea at the time. This is what is known as Harvard Architecture and in theory at least it means that it is possible to fetch a code instruction at the same time as data in a single machine cycle.

Figure 5 shows the memory map and may cause some confusion for those of you who are more used to the conventional memory arrangements. The EEPROM space is pure data memory and assembler programs cannot run in this space, high level programs can, see later. There is provided 256 bytes of internal RAM that shares the upper half of this space with special function registers. The trick to 'getting at' the various spaces lays in the instruction set. It would take up too much room here to go into detail but to give an example when accessing the code memory the MOVC instruction is used but when accessing external memory MOVX is used. Other techniques of direct and indirect addressing are used to access the internal RAM spaces.
The on-board Flash memory contains TCB, version 1 and only occupies about the first 6 k. If this is good enough for your applications then you can forget all about the various memory addressing modes, TCB will take care of it.

Introduction to programming in TCB

The integer BASIC is a modified version of Tiny BASIC called Tiny Control Basic (TCB) that was designed specifically to enable users to get the maximum out of a microcontroller in the shortest possible time without having to go through a massive learning curve, or installing any special software.

The 8052 architecture has three memory spaces that are dealt with by TCB. Internal RAM, EEPROM and external RAM. These are accessed by the keywords IRAM, ROM & RAM respectively. Type IRAM and you should see this message:

```
i>iram

Internal Ram  00B4 to 00FF
OK
i>
```

The 'i>' prompt indicates that we are now in internal ram space. As you can see there are only a few bytes but this is good enough for a 2 or 3 line program. TCB will let you know if you run out of space. To access the EEPROM type ROM:

Table 2. Tiny Control BASIC main specification	
Numbers:	16 bit signed integers range from –32767 to 32767
Variables:	single letter A through L (12)
Arithmetic:	+, -, *, /, and MOD
Logic:	NOT, AND, OR, XOR
Comparisons:	>, <, =, <>, >=, <=
Commands:	RUN, LIST, NEW, DUMP, RND, ABS, IF, THEN, GOTO, FOR, TO, NEXT, REM, CALL, RETURN, GOSUB, ROM, RAM, IRAM, LOADH, LOADB, DECIMAL, HEX, LET, PRINT, INPUT, BAUD
Special function registers:	PZ0, PZ1, PZ2, PZ3, TCON, TMOD, TL0, TL1, TH0, TH1, SCON, SBU, IE, IP, T2CON, WMCON, SPCR, SPSR, SPDR, PCON
Interrupt:	ONEX0, ONEX1, ONT0, ONT1, ONT2, ONSP, EI, DI

```
i>rom

Rom  0000 to 07FF
OK
e>
```

We are now in ROM space and as you can see there is 2 k of memory from 0000 to 07FF. At any time you can type 'DUMP' to see the contents of the memory. The advantage of using this space is that it will retain the program even after power down. The disadvantage is that it is slower than RAM to write to and there is a limit to how many times you can write to it, approximately 100,000 times. In this project there is no external RAM so typing RAM will return an error. All TCB programs will be written to the EEPROM. TCB is capable of running a program at start up but it must begin with line 10. If line 10 does not exist any the program will effectively be erased. The start up procedure is as follows:

1. Check for input from user (space bar) – waits about 1 to 2 seconds
2. If no input, check ROM space for a program starting at line 10
3. Run it if it exists, if not check RAM space for a program starting at line 10 and run that.

If no line 10 exists and there is RAM it will erase (NEW command) the program in RAM and the sign on will be in RAM space. If there is no RAM then it will come up in internal RAM space (IRAM). Memory will not be erased if you press the space bar within one to two seconds of switch on. In this project of course there is no external RAM so the default will be to come up in internal RAM space "i>" unless there is a line 10 in ROM space in which case it will run the program there.
If the space bar is pressed within approximately two seconds of reset or switch on, TCB will detect the Baud rate and come up in internal RAM space.
TBC does not actually erase all of the RAM but simply puts the end of program marker (FF) into the first byte. You can verify this using the DUMP command after the NEW command.
Referring to the MCU circuit, LED D3 is connected to port line P1.0 via two buffers in IC2. By default it will be off. This is because at start up all of the port lines are taken high.
To turn on the LED simply type: PZ1=254

```
i>pz1=254
OK
i>
```

PZ1 is the port 1 variable, anything you set this to will occur on port 1. By setting port 1 to 254 which is 1111 1110 in binary it will set pin 0 of port 1 to 0. By convention this pin is referred to as p1.0

Try:

```
ROM
10  FOR J = 1 TO 10
20  PZ1=254
30  GOSUB 200
40  PZ1=255
50  GOSUB 200
60  NEXT J
70  END

200 FOR K = 1 TO 50
210 NEXT K
220 RETURN
```

The above should flash the LED on and off 20 times.
Table 2 gives a brief description of the language.

Odds & ends

The massive amount of documentation Jim produced for this project would easily fill half this magazine, hence some items had to be moved to our website from where they can be downloaded free of charge. The items include the illustrated *Quick Start Guide — Swiss Army Knife*, the *Tiny Control BASIC Manual* and *Simple Dice*, so get downloading...

(030448-1)

Web pointer

FT232BM USB drivers: www.ftdichip.com

Free Downloads

Byvac-Terminal for PCs (install file with supporting OCX files), TCB (hex file), File number: 030448-11.zip

Two simple dice programs (Word file). File number: 030448-12.zip

Quick Start Guide for Swiss Army Knife (Word file).
File number 030448-13.zip

Tiny Control BASIC manual (pdf file).
File number: 030448-14.zip

PCB layout in PDF format.
File number: 030448-1.zip

www.elektor-electronics.co.uk/dl/dl.htm, select month of publication.

Assembly code

At some point in time more control or faster speed may be required than a high level language can give. There is no alternative but to resort to use assembler, which is the direct equivalent to programming the processor itself.

To do this effectively you will need an assembler. This is a program that translates 2 to 4 letter mnemonics into numbers that the processor can understand. There are many available free for this processor. In the examples shown "ASM51" has been used. It can be downloaded from various sources.

RAM would be required to run assembly code using the LOADH feature but none is available in this project. However, we have free code space in EEPROM starting at 1700h which is accessible through the in-circuit programming feature.

Getting Started

Always the hardest part? You can't really program in assembler without knowing something about the processor and for this you will need at least a data sheet for the processor and some knowledge of the 8051 / 8052 instruction set but the following example will get you on your way.

Example program

We will keep on familiar ground by flashing the on board LED again. As you know, to do this we need to set P1.0 to zero and back to 1 again.

```
; Example of flashing an LED
; Use Asm51
; P1.0 has the LED connected to it
;
$MOD8252

        cseg
        org     1700h
flash:  ; pulse P1.0 up and down
        setb    p1.0
        call    delay
        clr     p1.0
        call    delay
        jmp     flash
;
delay:  ; delay
        mov     r0,#0f0h
del2:   mov     r1,#0
del1:   nop
        djnz    r1,del1
        djnz    r0,del2
        ret

        end
```

The operation of the program is conveniently explained by means of a table.

`&MOD8252`	First of all this contains all of the special names associated with that processor so this needs to be in place for our 8958252.
`Cseg`	Tells the assembler that the following is code, there is a corresponding "dseg" for data.
`Org`	This is the program origin address. TCB finishes at about 1650h so we don't want to be any lower than this otherwise we will damage TCB.
`setb p1.0`	Sets the bit 0 of port 1 to high, just as pz1=1 would do.
`Call`	Is the equivalent of gosub
`delay:`	This is a subroutine, notice the ':', asm51 requires this. Because of the raw speed of the processor 2 delays are required. In fact this carries out over 61,000 instructions before returning which gives a delay of about 0.25 second.
`R0,#0f0h`	R0 is a general purpose register of which there are 8, R0-R7. There are in fact 4 banks of them but only one bank can be used at a time. What this instruction does is to place the value of F0 (hex) into register R0. The '#' is known as an 'immediate' modifier. If you did R0,0f0h then the contents of memory address F0 would go into R0. (yes there is plenty of scope for error).
`Djnz`	This is Decrement and Jump if not Zero so on the inner loop R1 will be decremented and if it is not zero then the program will jump to the label "del1:". Something to note here is that the register is decremented before zero is tested so a register starting out with a value of zero will be decremented to FF before being tested.

Assembling

Use Notepad or any text editor to create the above program, save it with an 'asm' extension, for example FlashLed.asm. Now that the program is written it needs converting to a format that the processor can understand. To do this you will need to use the command prompt if you are using the Windows operating system. If you are using DOS then you are already there.

Choose a directory to put the files in, you should have FlashLed.asm, Asm51 and MOD8252 in the same directory (unless you know how to set paths up). At the prompt type "asm51 FlashLed". On pressing enter you should see something similar to that of **Figure A**. Note that Asm51or indeed any other 8051 assembler has to be obtained separately, it is not included with the software for this project.

This process produces two files using the same file name but with different extensions thus: FlasLed.hex and FlasLed.lst. The LST file is where you will find any errors if there are any and the HEX file is a special format developed by Intel to enable code to be loaded into processors. As a matter of interest the format consists of lines of text preceded by a colon.

```
:10170000D29012170CC29012170C80F478F0790066
```

The first two numbers '10' is the length of the code in hex, the next 4 numbers is the memory address, in our case 1700 this is followed by a record type which is almost always 00. The rest of the line consisted of the actual code to be loaded at the given memory address except for the last two numbers, '66' which is a checksum to verify that the line has been received correctly.

Load Up

The next job is to load this program into code space. This is where the special features come in. Click on the "101" icon and the background will turn black. We are now in flash programming mode. Press 'p' and the text will turn yellow, this indicates that we will be using Program or Code space (the flash memory area). Typing "v" "88 <enter>" and then "99<enter>" will enable you to view the contents of the code memory from memory address 0088 to 0099. This is where the sign-on message of TCB is located. All numbers are in hexadecimal format, see **Figure B**.

To program this area with our assemble program type "h" This will pop a dialog box that will allow you to select the FlashLed.hex file. This will be loaded into the memory and you should get something like **Figure C**. Programming will now commence automatically (and it does take some time to complete).

Running the program

First of all get back into terminal mode by pressing 'x' or the '101' icon. To run this program TCB has provided CALL. The syntax is CALL n where n is the address you want to jump to. The address we chose to put the program at was 1700h, TCB works by default in decimal, to change this type HEX. Another word of caution, everything is in hex from now on until you reset or type DECIMAL. This applies to normal basic program, hex line numbers look very strange.

Type the following to run the program:

```
HEX
CALL 1700
```

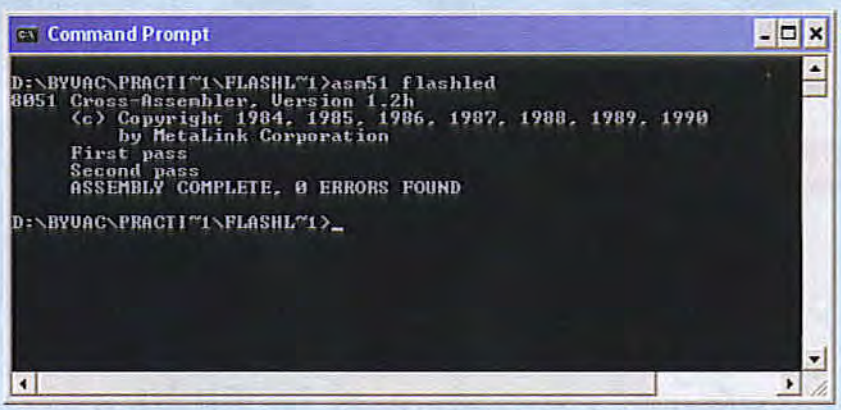
If you don't want to use HEX then CALL 5888 works just as well, 5888 is the decimal equivalent of 1700h.

The LED should now flash indicating that you are running the assembler program. Note because this is such a simple program, the only way to stop it is to press reset. Do this at the terminal using the icon next to "101" or using the push button on the main board. Once in flash memory, the program will remain after power is removed. Because of the slow serial programming nature it is not really suitable for large files. Although this can be done, expect TCB to take about 25 minutes for the basic.hex file. Some things simply can't be done using TCB alone, this feature makes the project completely versatile.

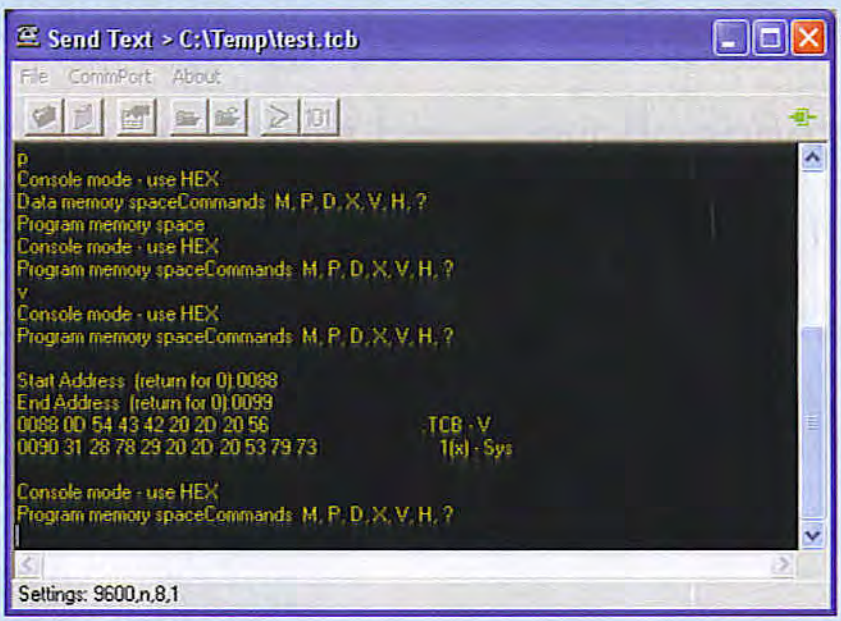
You can quite happily use a mixture of TCB and assembler. You can for example assemble the code at address 1700 and use a simple 2 line basic program:

```
10 HEX
20 CALL 1700
```

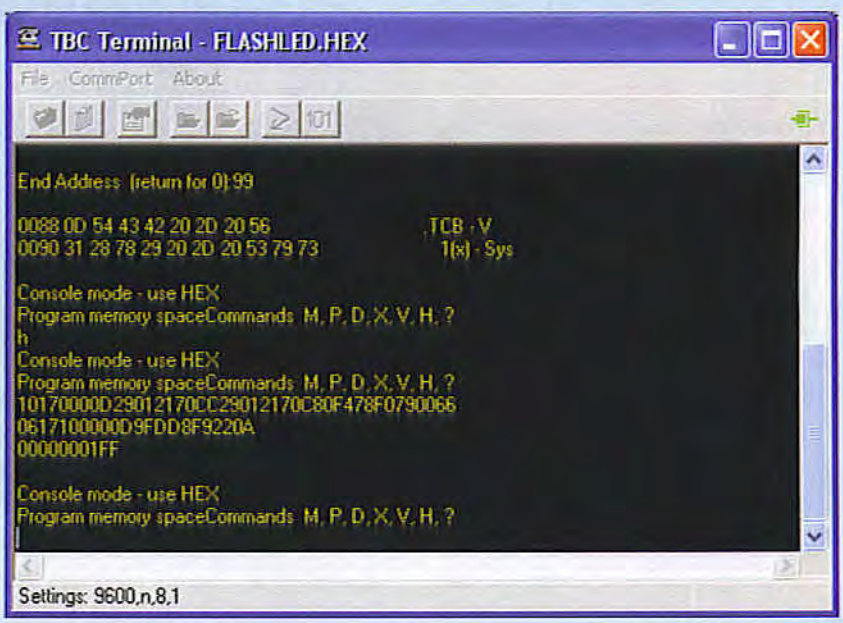
This will start flashing the LED whenever there is power applied to the board. Okay so a flashing LED has its limitations but we're sure you get the idea.



A



B



C