DEVELOPING APPLICATIONS AROUND THE PIC ARCHITECTURE

PART 6

Building a Complete Solution

by Stephen Waddington

After separately considering both the hardware and software elements of the PIC family of microprocessors, this month, we take a look at developing complete applications. Here, Stephen Waddington matches hardware and software with a collection of design tools to create a very basic application based around the PIC architecture.

In the beginning, there is always a big idea. What's yours? What do you want to develop? All electronic design engineers start with an idea. In fact, we're short-cutting the design process before we've even started by deciding to work with the PIC family of microprocessors. You'll have to forgive me for this supposition, but I want to use this article to demonstrate how to create a PIC based solution from initial concept, through to a working design.

My idea is straightforward. I want to create a circuit which will flash two LEDs alternately on and off. And here, you'll understand the reason for my apology. This is not the most complex design – and could be solved using a basic flip-flop – but its simplicity will allow us to examine each element of the development cycle in detail. Once we've got the basics right, we can start to look at more complex problems.

Development Cycle

At the end of last months' feature, you may remember that we examined the development cycle for a microprocessor application. This is repeated in detail in Figure 1. We'll be concentrating on the left-hand side of the process without the luxury of an In Circuit Emulator (ICE) to test code. While the ICE approach can simplify debugging, it has a major downside in that the emulator hardware is expensive. A basic ICE costs approximately £500, placing it beyond the realms of the amateur developer. Instead of taking the ICE approach, we'll either download code directly to EPROM for testing or test it first using an event-driven software simulator.

As with the other parts of this series, we're focusing on the PIC16C84 as the target device. From a hardware point of view, this is complete overkill. The flip-flop application will require a fraction of the 1k-byte program memory available, will not require any of the four available interrupt lines, and will require only two of the available 13 output lines.

But the PIC16C84 is an excellent device to learn the basics of microprocessor and PIC development. It uses the standard PIC instruction set of 33 instructions, with only two additions to the address registers, which are unique to the device. This means

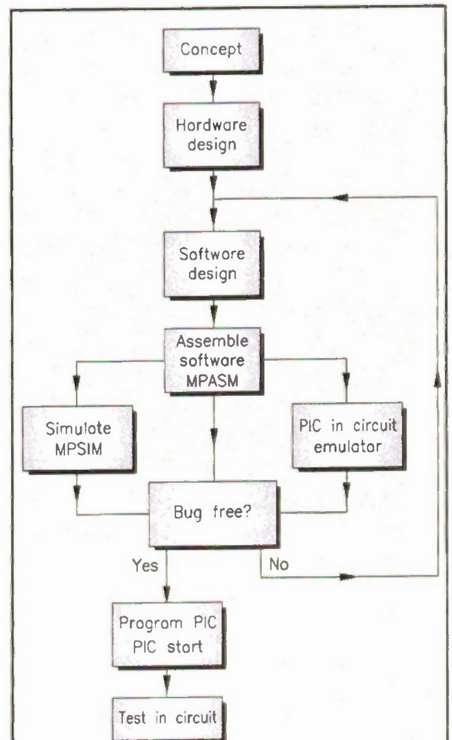

Figure 1. Development cycle for microprocessor application.

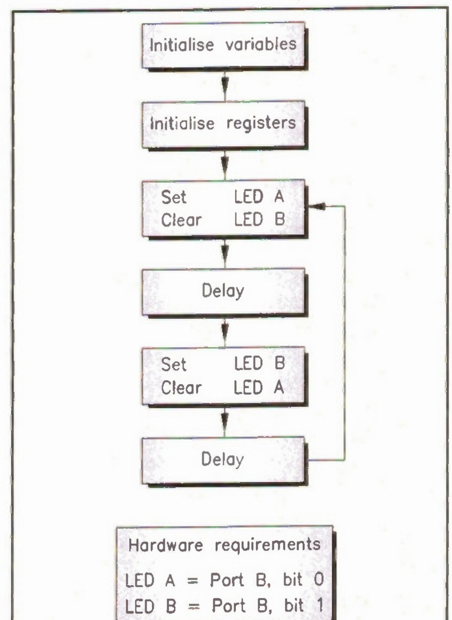

Figure 2. Development cycle for flip-flop application.

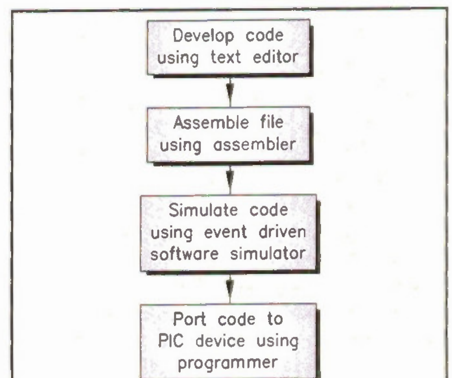

Figure 3. Assembling and porting PIC machine code.

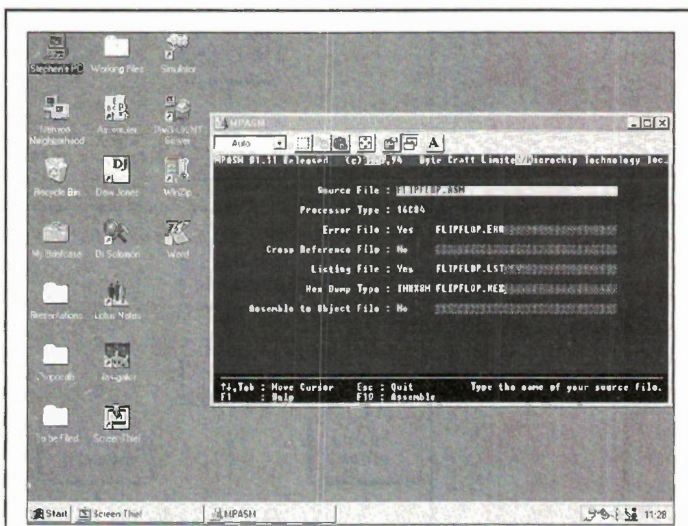

Photo 1. Microchip MPASM assembler environment.

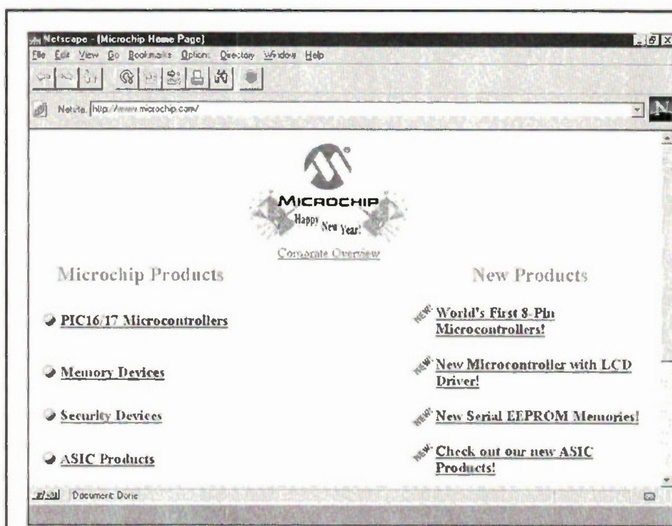

Photo 2. Microchip home page at `http://www.microchip.com`.

that once you have got to grips with programming the PIC16C84, you will be able to work with any of the other devices in the PIC family.

The PIC16C84 has other benefits in terms of the design process. Its 1k-byte of local EEPROM means that the device can be programmed and re-programmed electronically within 20 seconds. The majority of the other members of the PIC family are EPROM-based and require exposure to ultra-violet light to erase the contents of memory, before being reprogrammed. This typically takes up to 20 minutes. In later parts of this series, we will look in greater detail at other members of the PIC family and how to best match a device against a particular design requirement.

Design Process

The first requirement of the design process is to develop a flowchart for our flashing LED flip-flop problem. I've done this in Figure 2. The flowchart starts by initialising both the hardware elements of the PIC16C84 and software variables required by the application.

The remainder of the flowchart considers the configuration of the required outputs. In this sense, it is self-explanatory. Of two LEDs, LED A is switched on, while LED B is switched off. Following a delay, the situation is reversed so that LED B is switched on and LED A switched off. After a second delay, the program jumps back to the start to continuously repeat this loop.

After defining the function of the flip-flop application, it would be normal to match the hardware requirements against a target device, but we have already decided on the PIC16C84. What we can do though, is define the required outputs against available ports. For this exercise, we'll use PORTB of the PIC16C84 and defined bit 0 as LED A and bit 1 as LED B.

Our next task is to convert the flowchart into code. This is created in a text editor or word-processor. You can use the Notepad in Windows, Word or WordPerfect, since all enable files to be saved in ASCII format. Once complete, the draft program is saved as an assembler file in ASCII or raw text format with an .asm extension. A PIC assembler is used to convert the source code into hexadecimal format which can then be ported directly to the target microprocessor. This process is shown in Figure 3.

The Assembler

There are numerous PIC assemblers available on the market which could be used to convert the source code into a hex file. One of the most popular is a DOS based package from Microchip, called MPASM. This is a neat piece of software which, unlike many other PIC assemblers, has a reasonable user interface, as shown in Photo 1. The majority of assemblers don't even have this luxury, relying instead on the user inputting a complex DOS string. The first-time user should avoid these at all costs. The complexity which they add to the design process is unnecessary.

Writing Code

The MPASM assembler accepts source code in a standard ASCII format and allows the user to select the required output format on screen using a mixture of the <TAB>, <ENTER> and Function keys. It also has a reasonable level of error reporting for when things inevitably don't go right first time.

Each line of the source file may contain up to four types of information: labels, mnemonics, operands and comments.

The order and position of these are important. The MPASM assembler separates a line into a series of columns each denoted by a tab space. Labels must start in column one directly against the left edge of the page. Mnemonics may start in column two or beyond and operands always follow the mnemonic.

Comments may be added after either an operand, mnemonic or label, or can start in any column if the first character is either an asterisk or a semi-colon. The maximum column width is 255 characters. One or more spaces must separate the label and the mnemonic, or the mnemonic and the operands. Operands may be separated by a comma.

Labels

All labels, such as subroutine names, must start in column 1. They may be followed by a colon, space, tab or the end of line. Comments may also start in column 1 if one of the valid comment denotations is used. Labels must begin with an alphabetical-character and may thereafter contain alphanumeric characters. Labels may be up to 32 characters long.

Mnemonics

Assembler instruction mnemonics, assembler directives and macro calls must begin in at least column 2. If there is a label on the same line, they must be separated from that label by a colon or by one or more spaces or tabs.

Operands

Operands must be separated from mnemonics by one or more spaces or tabs. Operand lists must be separated by commas. If the operand requires a fixed number of operands, anything on the line after the operands is ignored. Comments are allowed at the end of the line. If the mnemonics permits a variable number of operands, the end of the operand list is determined by the end of the line or the comment.

Comments

Comments which are on a line by themselves must start with either of the comment characters, namely an asterisk or semi-colon. Comments at the end of a source line must be separated from the rest of the line by one or more spaces or tabs. Anything encountered on the line following the comment character is ignored by the assembler.

Developing Code

Before we examine the code for the LED flip-flop application in detail, let's run through some basic rules of microprocessor software development. These rules apply as much to development around the PIC architecture as any other microprocessor family.

Structure Software

Creating a flowchart of an application is a good way to start the development process, since it imposes structure from the outset. It enables you to remain focused on what it is you want to achieve.

Layout

Use a logical format to the structure of your software. Convention and logic dictates that initialisation routines come first, followed by the main program, and then any subroutines which the main program calls upon.

Labels

All subroutines should carry a logical name. This simplifies coding, since it is easier to jump to a logical name rather than specifying a hex address when you need to switch to a subroutine.

Variables

The same applies to variables. Wherever possible, use a logical naming convention for variables, rather than specifying a number. This makes code easier to debug and means that the function of a program is understandable at first glance.

Comment

Always annotate your software with comments. You will inevitably need to come back to review your software. Concise annotation makes that process far easier.

Header

Create a universal hearer for your programs. This reduces workload, creates a consistent format and limits the number of variables you have to remember.

If you find all these rules daunting don't be too concerned at this stage. The easiest way to learn a new software language such as PIC machine code is to examine plenty of examples. Consult the books recommended in the reading list at the end of this feature, or check out Microchip's home page at **http://www.microchip.com**, as shown in Photo 2. It provides numerous links to Web sites created by PIC development engineers. Also, flick through your back copies of *Electronics and Beyond*. Revisit PIC projects designed by the team of Maplin project engineers to check their code and target circuit designs.

Flip-Flop Application

The code for the flip-flop is shown in Figure 4. There are essentially four components to the programme, as denoted by comments in the program and the descriptions below.

Set Variables

Here, variables used throughout the programme, such as PORTB and COUNT, are defined.

Initialisation Routine

This is used to set the memory location for the programme, initialise PORTB for output rather than input and clear PORTB to zero.

Main Programme

This is the heart of the programme. It alternately switches bits 0 (LED A) and 1 (LED B) of PORTB on and off. A delay of 0·2ms is maintained between each state.

Delay Routine

The programme uses the PIC16C84's internal real time clock (RTCC) to create a delay. Bit 7 of the RTCC register – which hits 1 after 128 clock pulses – is used in a nested loop to create a delay. LONG2 loops until bit 7 of the RTCC register is set. This is equivalent to a delay of 32·768ms – 128 clock pulses multiplied by 256μs, assuming 4MHz clock, which gives a clock pulse width of 256μs. LONG2 is nested with JUMP which counts from 8 – the value of the variable COUNT – to zero. This creates an overall delay of 0·256s.

```
; Flipflop Routine - Filename: flipflop.asm
; Stephen Waddington
; Building A Complete Solution
; Developing Applications Around the PIC Architecture - Part6

; Set variables
PORTB        EQU 06H        ; PORTB is register 6
RTCC         EQU 01H        ; PIC RTCC timer register
COUNT        EQU 00H        ; Timer counter
TIME         EQU 08H        ; Timer period

; Initialisation routine
INIT    ORG   00H           ; Store programme at location 00H
        TRIS PORTB          ; Set PORTB as outputs
        CLRF PORTB          ; Clear PORTB

; Main programme
MAIN MOVLW B'00000001'      ; Set LEDA on, LEDB off
        MOVWF PORTB
        CALL DELAY          ; Hold LEDA on for 0.256ms
        MOVLW B'00000010'   ; Set LEDB on, LEDA off
        MOVWF PORTB
        CALL DELAY          ; Hold LEDB on for 0.256ms
        GOTO MAIN           ; Loop back to the beginning of MAIN

; Delay routine
DELAY   CLRWDT              ; Clear Watchdog timer
        MOVLW TIME
        MOVWF COUNT
        CLRF RTCC           ; Clear RTCC register
LONG    BTFSC RTCC,7        ; Test RTCC bit 7 (128 X 256µS = 32µ768ms)
        GOTO JUMP           ; If RTCC bit 7 set goto JUMP
        GOTO LONG           ; If RTCC bit 7 not set loop until set
JUMP    CLRF RTCC           ; If RTCC bit 7 is set clear RTCC
        DECFSZ COUNT,F      ; Decrement COUNT by 1 until reach zero
                            ; (32ms X 8 = 0·256s)
        GOTO LONG           ; Loop LONG if COUNT not equal to zero
        RETURN              ; Return to call location
RESET   GOTO INIT           ; On RESET goto INIT

    END
```

Listing 1. Flip-flop assembler code.

Having creating the source code in a text editor, it must be save in an ASCII format with an **.asm** extension, in this case, **flipflop.asm**. It can then be loaded into the MPASM assembler. After selecting the appropriate target microprocessor – in this case, the PIC16C84 – and Hex output – in this case, INHX8M – the file can be compiled. The MPASM assembler is able to create four different Hex output formats, depending on the format required by the PIC programmer. Make sure you select the correct format.

The assembler returns a series of statistics relating to the length of the assembled code as well as the number of warning and error messages, as shown in Photo 3. If the code contains any bugs, it will not run when downloaded to the microprocessor. Errors reported by the assembler can be examined in either the List or Error files. The assembler creates three other files in addition to the List and Error Files. These are detailed below and shown in Photo 4.

- **<filename>.asm**
  Default source code file.
- **<filename>.lst**
  Default output extension for listing files generated from the assembler
- **<filename>.err**
  Default output extension from MPASM for error details.
- **<filename>.hex**
  Default output code for porting to target microprocessor.
- **<filename>.cod**
  Default output extension for the symbol and debug file.

Software Debugging and Simulation

Using the Error and List files, the source code should be debugged and reassembled until it is error free. This can be a very tedious process and is why professional development engineers use an In Circuit Emulator (ICE). Microchip has developed a compromise solution in terms of cost, in the form of a discrete event software simulator. The Microchip MPSIM enables PIC code to be emulated by a PC and various program variables, interrupts, and ports to be monitored.

Like the Microchip assembler MPASM, MPSIM is DOS-based and as such, is not very user-friendly. It uses a set of proprietary instructions to both initialise the simulator environment and run an actual simulation. It's almost as if you need to learn an additional software language before you can run a simulation. For this reason, the majority of hobbyists tend to test software by downloading it directly to the target microprocessor.

Photo 5 shows the flip-flop application being simulated using MPSIM. The assembled PIC software has been loaded in hex format into the simulator and the different variables used by the programme set as flags to be monitored in the upper half of the screen. When the code is run and simulated by the PC, its effects on registers, variables, interrupts and the I/O ports can be monitored by observing the flags.

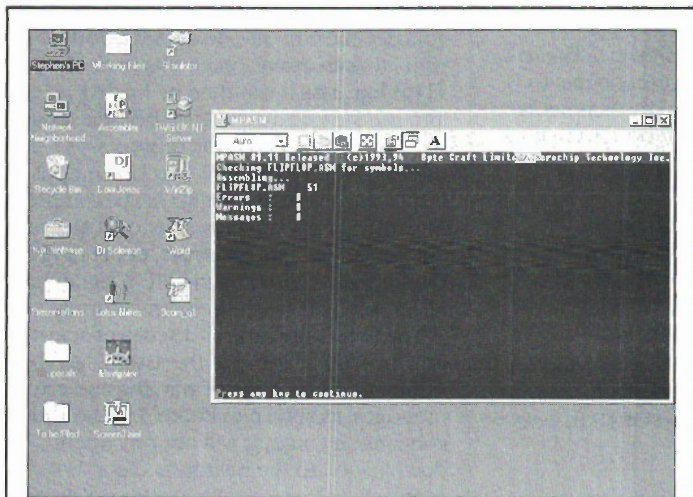

Photo 3. MPASM returns a series of statistics relating to assembled file.

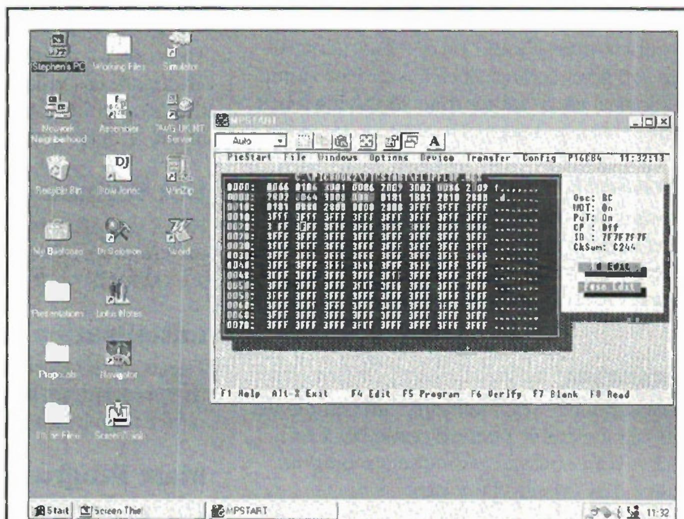

Photo 6. Porting code to PIC16C84 using PICStart.

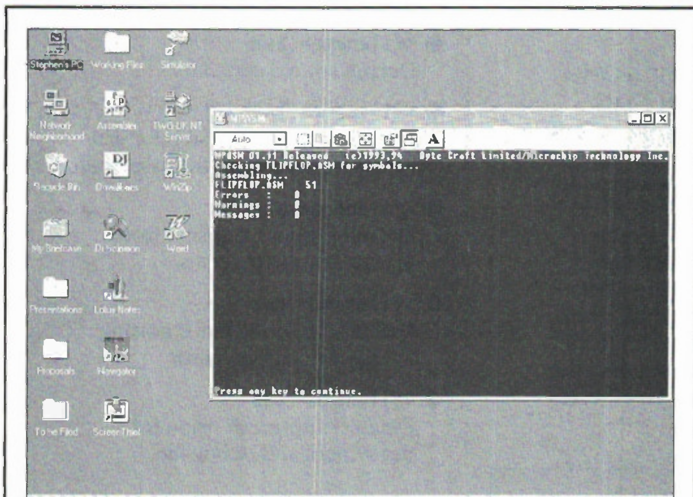

Photo 4. Output files from MPASM.

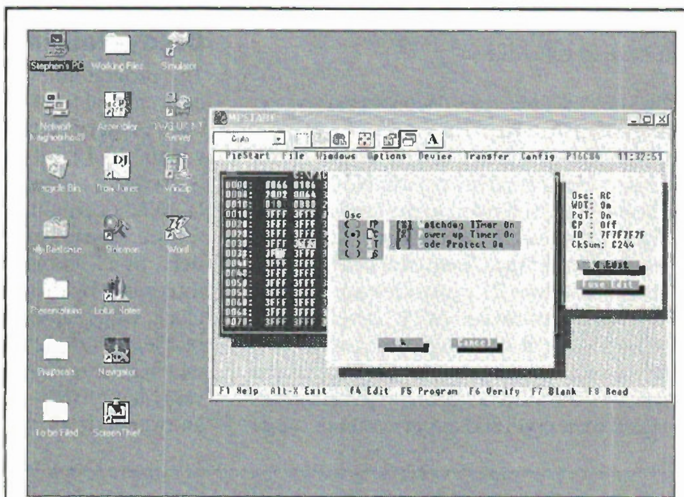

Photo 7. Making fuse selecting in PICStart development environment.

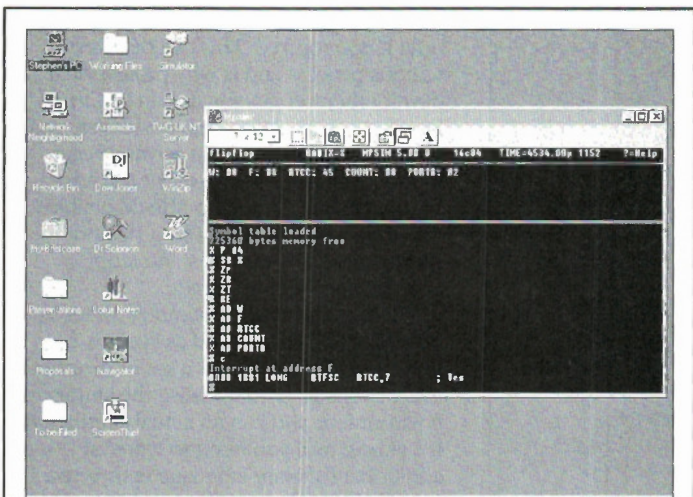

Photo 5. Simulating flip-flop application using Microchip MPSIM.

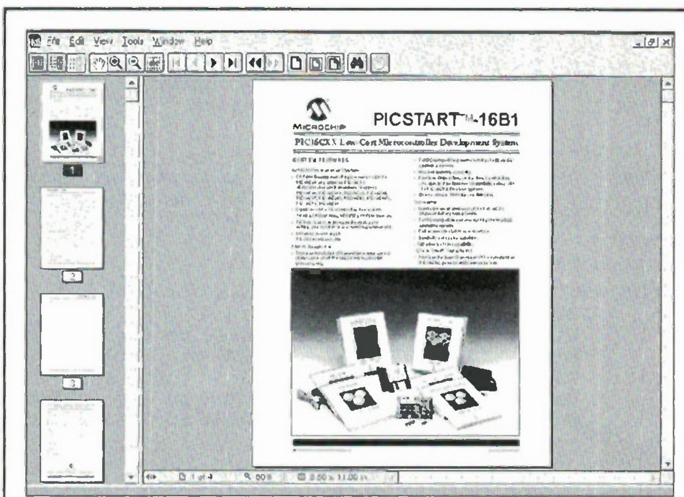

Photo 8. Datasheets in Acrobat PDF format can be downloaded from the Microchip Web site.

Hardware

Whether or not you decide to simulate your PIC code using MPSIM or a similar software simulator, you'll eventually have to download it directly to a PIC device. There are a number of programming devices available on the market. The most versatile and consequently, the most popular, is Microchip's PICStart. PICStart enables any device in the PIC microprocessor family to be programmed. By comparison, Maplin has developed a programmer kit project specifically for programming the PIC16C84. While this machine is specific to the PIC16C84, at approximately £20, it is relatively inexpensive compared with PICStart. The PIC16C84 was profiled in Issue 105 of *Electronics and Beyond*.

PICStart consists of two elements – a software programme and a programming board. The software programme, as shown in Photo 6, is used to drive the programmer. From here, the target programming board (shown in Figure 4) connects directly to the serial port of a PC. Power is provided by an auxiliary 9V mains adapter. The target PIC device is inserted on the programming board in a zero insertion force (ZIF) socket.

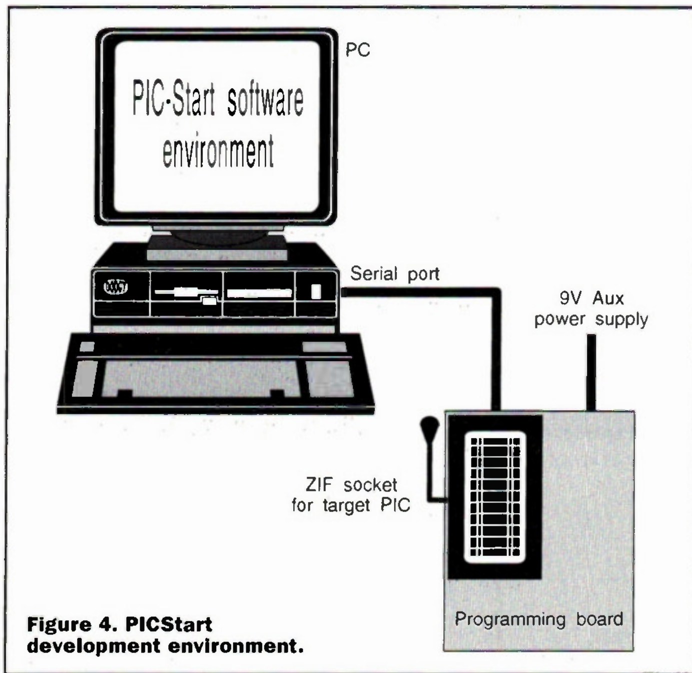

Figure 4. PICStart development environment.

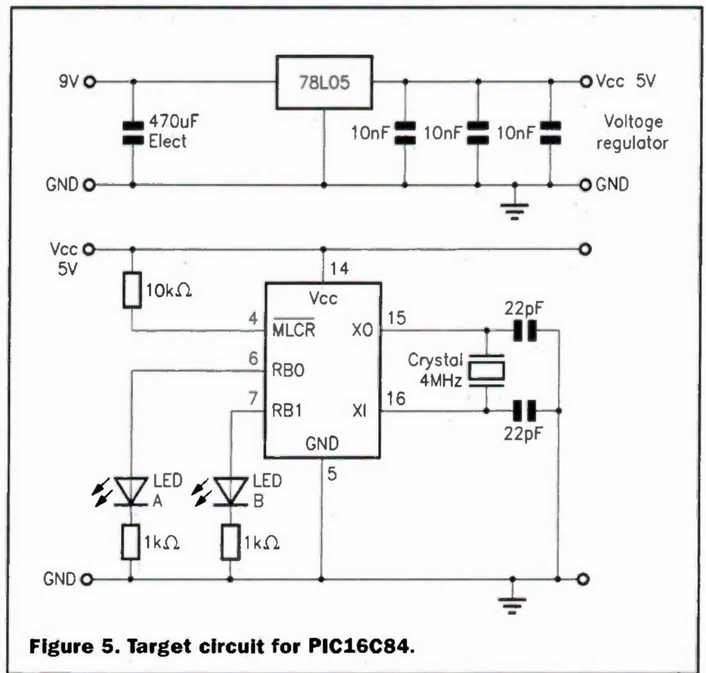

Figure 5. Target circuit for PIC16C84.

Unlike other elements of the development process, programming a PIC device is very straightforward. Having loaded the PICStart application and connected the programming board, the target device is selected. Next, the assembled code to be ported to the PIC device is loaded. Like the MPSIM Simulator, PICStart uses the INHX8M version of hex code.

Before the device can be programmed, the software programmable fuses such as Watchdog Timer, Start on Power up and clock format must be configured, as shown in Photo 7. Once this is complete, the target device is mounted on the programming board and the code downloaded. It takes approximately 20 to 30 seconds to programme a PIC device. During this time, the PC transfers the hex code and fuse selections to the memory of PIC device, before verifying the contents of all EEPROM memory.

Target Circuit

So, we've developed the flip-flop design from an initial concept, created a flowchart, written a routine in PIC assembly code, simulated it and finally downloaded it to a PIC16C84. What we need to do now is build a target circuit and test the device. Figure 5 shows a basic target circuit for the PIC16C84. There are four key elements to this design as follows:

Voltage Regulator

The operating range for the PIC16C84 is 2 to 6V. Consequently, a 78L05 voltage regulator is used to stabilise the voltage from a 9V battery at a constant 5V. This is not necessary if you have access to a stabilised 5V power supply.

Clock

Figure 5 shows a 4MHz crystal with two 22pF capacitors. This option has been selected for its simplicity. A resistor capacitor combination with an RC time constant of 4MHz could be equally used. Whichever clocking method you adopt, ensure that you select the appropriate fuse option when programming the target device. A 4MHz crystal falls within the XT region, as shown in Table 1 and discussed in Part 3 (Issue 109) of this series.

For timing insensitive applications an external RC clock offers cost savings. Figure 6 shows how the RC combination is implemented. For R_{ext} values below 2k2Ω, the oscillator operation may become unstable, or stop completely. For very high R_{ext} values above say, 1MΩ, the oscillator becomes sensitive to noise, humidity and leakage. Microchip recommend that R_{ext} is kept between 3kΩ and 100kΩ.

Although the oscillator will operate with no external capacitor (C_{ext} = 0pF), a value above 20pF should be used for noise and stability reasons. For these reasons, it is recommended that the novice developer use either a crystal or ceramic clock, since this avoids added complication. If you intend using a RC combination, select the RC option when programming the target device.

To achieve a clock speed of 4MHz using RC components, try around values of Rext = 20pF and Cext = 10kΩ.

Output

The PIC16C84 has an output current capability of 25mA. At 5V, this is sufficient to blow an LED. Consequently, 1kΩ resistors are used on the leg of each output to reduce the current to the LED to a moderate 5mA.

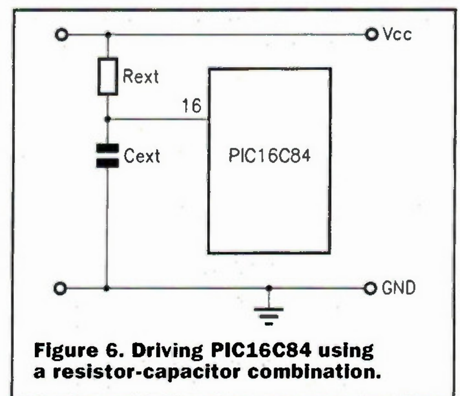

Figure 6. Driving PIC16C84 using a resistor-capacitor combination.

Option	Clock Speed
LP	0 to 200kHz
XT	100kHz to 4MHz
HS	4MHz to 10MHz
RC	0 to 10MHz

Table 1. Clocking rates and options

Reading List

The following books discuss many of the issues raised in this article in greater depth. They also provide examples of the PIC development cycle including many specific projects.

Microchip has produced documentation for all of its development tools. It has also produced datasheets for each of the PIC microprocessors. These can be downloaded in Acrobat PDF format, as shown in Photo 8, from the Microchip Web site. An Acrobat reader can be downloaded from the Acrobat Web site at **http://www.adobe.com**.

Description	Reference/Order Code	Cost
A Beginners Guide to the Microchip PIC	AD31J	£19.95
PIC Cookbook	DT76H	£19.95
Embedded Control Handbook	AD28F	£9.50
Microchip Databook	AD29G	£9.50
MPSIM for DOS User's Guide	http://www.microchip2.com/devtools/devtools.htm	–
MPASM User's Guide	http://www.microchip2.com/devtools/devtools.htm	–
PICStart-16B1 User Guide	http://www.microchip2.com/devtools/devtools.htm	–
PIC16C84 Application Note	http://www.microchip2.com/devtools/devtools.htm	–

Reset

The reset pin – pin 4 – is connected high. This causes the PIC16C84 to enter a power-up delay phase of approximately 72ms in order to enable the clock to stabilise at power on. This eliminates the need for external components usually required for Power On Reset. To reset the device at any stage during operation, pin 4 should be connected low for a brief period.

Initially, you should experiment using breadboard. It always takes a couple of iterations of both software and hardware to achieve the required operation. That said, having built the target PIC16C84 circuit, the programme should fire-up immediately on power-up and LEDs A and B flash alternately.

Next Month

Next month, we'll examine more advanced software techniques including the use of interrupts, look-up tables, reset vectors and the watchdog timer. We'll also look in greater details at how to debug object code.

Download List

Shareware versions of the majority of the development tools discussed in this feature can be downloaded from the Microchip Web site at **http://www.microchip2.com/softupdt.htm**. Specific references are detailed below. All files have been compressed using PKZIP. PKUNZIP – the decompress utility – can also be downloaded for the Microchip Web site.

Description	Software Tool	Full Length
MPASM Assembler	MPASM 1.40.00	487k-bytes
MPSIM Simulator	MPSIM 5.20.00	286k-bytes
PICStart (Software only)	PICSTART-16B1 5.00	125k-bytes

Catalogue References

Many of the items discussed in this feature can be purchased directly from Maplin, either by mail order or directly from one of the Maplin shops. Catalogue references and costs are outlined below.

Pic Resources

Description	Order Code	Cost
PIC16C84 Programmer Kit	95128	£19.99
PICStart-16B1 Development System	DM79L	£154.90
ICEPIC16CXX Real Time Emulator System	DT77J	£689.00

Target PIC16C84 Circuit

Component Type	No	Description	Order Code	Cost
Semiconductors	1	78L05 5V voltage regulator	WQ85G	£0.49
	1	PIC16C84	AD50E	£10.70
Capacitors	2	22pF mica	WX05F	£0.55
	3	10nF ceramic	BX00A	£0.11
	1	470μF, 35V electrolytic	AT62S	£0.36
Resistors	2	1kΩ resistor	U1K	£0.05
	1	10kΩ resistor	U10K	£0.05
Miscellaneous	2	Red LEDs	WL27E	£0.10
	1	9V PP3 battery	JY60Q	£1.90
	1	PP3 Battery clip	HF28F	£0.19