

Microcontroller **DEBUGGING** and **TESTING** tools

Professor Dr Dogan Ibrahim, lecturer at the Near East University of Nicosia in Cyprus, describes some of the currently available tools for microcontroller debugging and testing

DESIGNING A microcontroller-based system is a complex activity that involves hardware and software interfacing with the external world and many engineers find it difficult the first time.

Successful design of a microcontroller-based system requires skills to use the various debugging and testing tools available. A program may seem to be running but may not be effective, or it might give the wrong results when operated in real-time, so debugging may be the only tool to help eliminate the problems.

Tools for debugging and testing microcontroller-based systems can be divided into two groups: software-only tools and software-hardware tools. Software-only tools come in the form of monitors and simulators, which are independent of the hardware under development. Software-hardware tools are usually hardware dependent, are more expensive and range from in-circuit emulators and in-circuit simulators to in-circuit debuggers. In general, the higher the level of integration with the target hardware, the greater the benefit of a tool, resulting in a shorter development time, but the greater the cost as well.

This article describes very briefly some of the

commercially available tools for debugging and testing microcontroller-based systems. The factors to consider when choosing a debugging tool are cost, ease of use and the features offered during the debugging process.

In this article I will describe the details of one of the popular low-cost debugging tools, with an example showing how an in-circuit debugger can be used in a real design. I've used the PIC microcontroller family as the basis for this article.

Software Simulators

A software simulator is basically a computer program running on an independent hardware and it simulates the CPU, the instruction set and the I/O of the target microcontroller. Simulators offer the lowest-cost development tools for microcontroller-based systems and most companies offer their simulator programs free of charge.

Basically, the user program can be operated in a simulation environment where the user can insert breakpoints within the code to stop the code and then analyze the internal registers and memory, display and change the values of program variables and so on.

Incorrect logic or errors in computations can be analyzed by stepping through the code in simulation. Simulators run at speeds 100 to 1000 times slower than the actual microcontroller hardware and, thus, long time delays should be avoided when simulating a program.

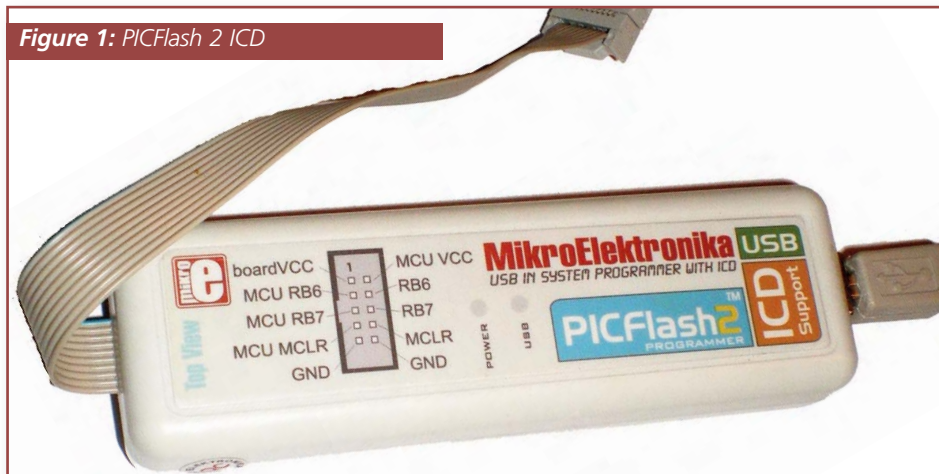
Microcontroller-based systems usually have interfaces to various external devices such as motors, I/O ports, timers, A/D converters, displays, push-buttons, sensors and signal generators, which are usually difficult to simulate. Some advanced simulators, such as the Proteus from Labcenter Electronics (www.labcenter.co.uk) for example, allow the simulation of various peripheral devices such as motors, LCDs, 7-segment displays and keyboards, and users can, in general, create new peripheral devices. Inputs to the simulator can come from files that may store complex digital I/O signals and waveforms. Outputs can be in the form of digital data or waveforms, usually stored in a file, or displayed on a screen. Some simulators accept only the assembly language of the target microcontroller.

Most microcontroller software is nowadays written using a high-level language such as C, Pascal or Basic, and it has become a necessity to simulate a program written in a high-level language. Some simulators – the microC, developed by mikroElektronika (www.mikroe.com) – can perform simulation using high-level languages. Oshon Software simulators (www.oshonsoft.com) are powerful low-cost high-level language simulators enabling the user to set breakpoints and display and modify registers and program variables.

Monitors

Monitors are software tools that usually run on the target microcontroller system. A monitor program usually resides in the bottom or top

Figure 1: PICFlash 2 ICD



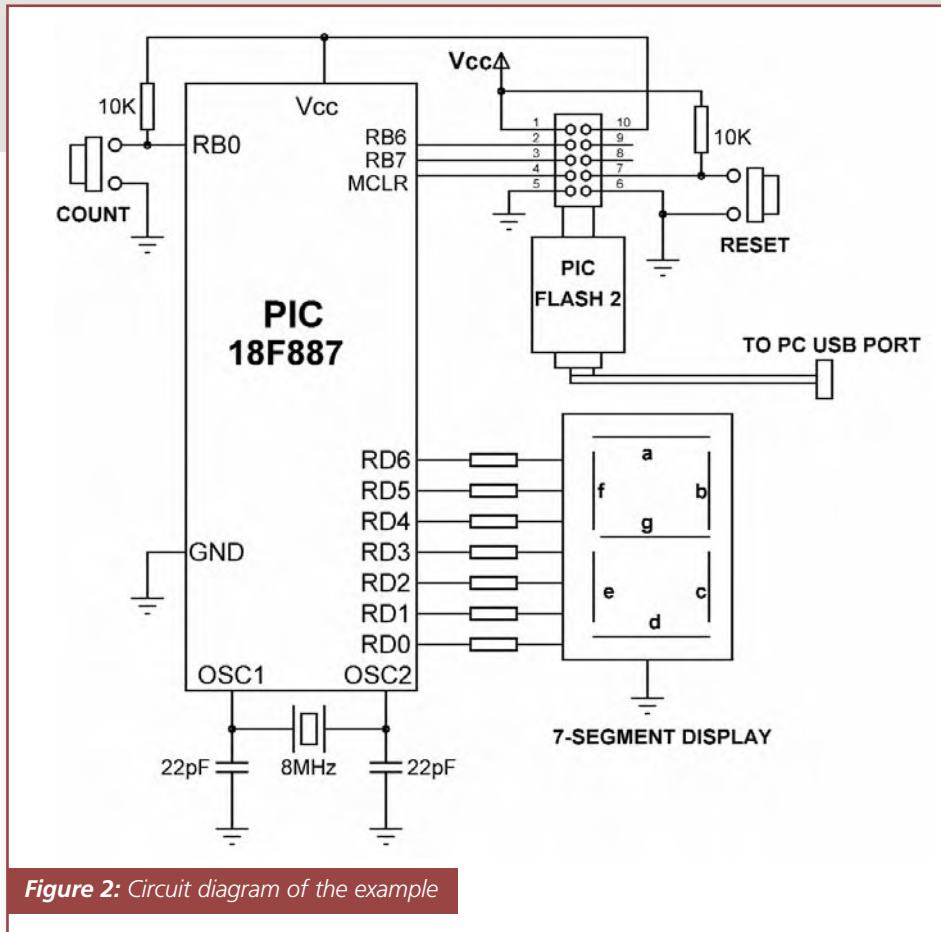


Figure 2: Circuit diagram of the example

part of target microcontroller’s memory and, with the aid of a monitor program, a user can download a program code, execute the code, set breakpoints in the code and examine and modify memory locations or registers.

The target microcontroller is usually connected to a terminal via an RS232 serial port (or USB) and the monitor commands are implemented directly on the target. One disadvantage of a monitor tool is that the execution of the application program must be stopped before a memory location or a register can be examined or changed. Monitor tools used to be very common in the early days of microprocessors and microcontrollers but they are seldom used nowadays.

In-Circuit Emulators

In-circuit emulators (ICE) offer real-time code execution, full peripheral implementation and breakpoint capability. An in-circuit emulator is plugged into the socket of the microcontroller in the target development system, replacing the target microcontroller. User program is loaded into the RAM memory of the emulator and the code executes in real-time.

In-circuit emulators offer real-time trace buffers, full internal register and memory access, and multi-level conditional breakpoints. Although an in-circuit emulator can be an invaluable debugging tool, its cost is usually much higher than the other debugging tools and different microcontrollers from different manufacturers, or even different models from the same manufacturer require different in-circuit emulator probes.

A popular ICE for PIC microcontrollers is the MPLAB ICE 4000 (www.microchip.com), consisting of an Emulator Pod, Processor Module, Device Adapter and a Transition Socket. A common emulator pod is used for all PIC microcontrollers and the pod communicates with the PC via an USB connection. The processor module, together with the device adapter and transition socket, is used for a specific processor, or processor family.

Burn and Learn Method

In many non-complex microcontroller-based system developments the ‘burn and learn’ method can be used for testing and debugging the system. This is basically a testing method,

although it can be used for debugging as well.

Here the microcontroller is programmed with the application code and then inserted into the target development system. Routines are then added to toggle the I/O pins, to dump data through the serial port, to send test data to an LCD and so on. For example, the result of a multiplication can be sent to the I/O ports and the program can be halted to examine the I/O ports, say by connecting LEDs or by measuring the voltage at the port pins to check whether or not the data is as expected. Alternatively, data can be sent to the LCD at various points of the program in order to trace the execution path of the program.

‘Burn and learn’ does not require any specialised software tools, and testing a system using this method is usually very inefficient, tedious and very slow.

In general, the following hardware can be useful in burn and learn method of testing:

- LED – An LED can be blinked on and off to tell what the code is doing. The disadvantage is that it cannot be used in fast time-critical applications.
- LCD – Characters can be displayed on an LCD to tell what the code is doing. The disadvantage is that it uses an 8-bit port.
- USART – Data can be sent to an LCD to see what the microcontroller is doing. The disadvantage is that the data transfer time is slow and extra hardware is usually needed.
- Logic Analyzer – Can be useful in testing the interface and control of peripheral devices such as USB, I2C, CAN or SPI bus. The disadvantage is that it can be expensive.
- Voltmeter or Oscilloscope – This is the simplest testing method where the state of a pin can be checked and, thus, some idea about the flow of program code can be obtained.

In-Circuit Simulation

An in-circuit simulator is very similar to a software simulator with the added advantage that inputs can be provided from the target hardware and, also, outputs can be sent to the

Parameter	Software Simulation	Burn and Learn	In-Circuit Simulator	In-Circuit Emulator	In-Circuit Debugger
Cost	very low	very low	Very low	high	low
Breakpoints	yes	no	no	yes	yes
Trace	yes	no	yes	yes	no
Single stepping	yes	no	yes	yes	yes
Loss of target pins	No	no	no	no	yes
Program downloading	yes	no	yes	Yes	yes
Real time execution	No	yes	no	yes	yes
Learning curve	medium	low	medium	medium	medium

Table 1: Features of debugging and testing tools

target hardware. In-Circuit Simulators are not very common and are only available for specific hardware platforms. Because an in-circuit simulator runs at the speed of the simulator software, it runs much slower than the target microcontroller and, thus, can not be used to simulate time critical applications, such as an USB port or the data transfer from an USART port.

In-Circuit Debuggers

In-circuit debuggers (ICD) are very clever tools used instead of the more expensive in-circuit emulators. An ICD is a small hardware device connected between a PC and the microcontroller development system (some development systems have built-in ICD circuits).

An ICD uses the In-Circuit Serial Programming (ICSP) capability of the target microcontroller where two or three pins of the microcontroller are allocated to the ICSP operation. The ICD downloads a small control program to the target microcontroller's flash program memory and then the user can develop and debug source code by setting breakpoints, watching variables, single-stepping the program, or running the program in full speed, thus enabling the time critical routines to be tested in real-time.

An ICD requires specialized circuitry to be present on a target microcontroller, such as logic to communicate to the ICD device, logic to single step program code and logic to halt and interrogate the target microcontroller. Although an ICD can be an invaluable debugging tool, it has two main disadvantages: it uses some memory of the target microcontroller and it requires the use of a few pins of the target microcontroller to communicate and control the target.

MPLAB ICD2 and, lately, the ICD3 are two of the popular in-circuit debuggers developed by Microchip. Some other popular PIC

microcontroller in-circuit debuggers are: ICD-U40 (by CCS Inc., www.ccsinfo.com), PICFlash 2 (by mikroElektronika) and others.

Table 1 gives a comparison of the various features of the debugging and testing tools.

In this article, an example is given to show how an ICD device can be used in a simple microcontroller-based system development. Although the operation and use of most ICDs are similar, the low-cost PICFlash 2 ICD is used in the example below for illustrative purposes.

PICFlash 2 In-Circuit Debugger

PICFlash 2 ICD is manufactured by mikroElektronika as a general purpose ICD and, also, for use in their development systems. A picture of the PICFlash 2 ICD is shown in **Figure 1**. The ICD is connected to the USB port of a PC and to the following pins of a PIC microcontroller:

- RB6
- RB7
- MCLR

Pins RB6 and RB7 are used for data and clock and pin MCLR provides the chip programming voltage. ICD manufacturers usually provide headers to connect their devices easily to the target microcontroller. Commands can be sent to PICFlash 2 from the mikroElektronika language compilers to watch and modify program variables, to set breakpoints and to single-step the program.

Example Use of PICFlash 2

A simple 7-segment display example is given here to illustrate how useful an ICD can be during the microcontroller-based system development cycle.

As shown in **Figure 2**, a 7-segment display is connected to PORT D of a PIC16F877 type microcontroller and a push-button is connected to bit 0 of PORT B (RB0). In this example, a count is incremented and its value sent to the 7-segment display whenever the push-button is

pressed. As such the display shows the count as 0, 1, 2, 3...9, 0, 1... as the button is pressed. Notice that ICD pins RB6 and RB7 and MCLR of the microcontroller are connected to a PICFlash 2 ICD.

The software in this example is based on the mikroC language, a popular C language compiler from mikroElektronika, supporting a built-in software simulator and the PICFlash 2 ICD. The program listing is given in **Figure 3**. Variable count is incremented every time the button (PButton) is pressed. Array LED stores the bit pattern to be sent to the 7-segment display to turn on a digit and this array is indexed by variable count.

In-Circuit Debugging

The debugging steps are given below:

- Write the program given in Figure 3, making sure that the compiler is configured as follows:

Device:	P16F887
Clock:	8MHZ
Build Type:	ICD debug
Debugger:	mikroICD Debugger
- Compile the program with no errors
- Connect the ICD to the target microcontroller system
- Send the program to the target microcontroller by pressing F11 or by selecting *Tools -> PicFlash Programmer*
- Select the mikroICD debugger. *Debugger -> Select Debugger -> mikroICD Debugger.*
- Start the debugger. *Click Run -> Start Debugger*
- Select variables and I/O ports count, PORTD and PORTB from the Watch Window.
- The program is now ready for debugging. Press F7 to single step through the program. As shown in **Figure 4**, a blue bar moves down the program lines to indicate the statement being executed. You should see the values of variables changing as the program is executing in single step mode.

```

/*****

```

ICD DEBUGGER EXAMPLE

This is an ICD example. In this example a 7-segment display is connected to PORT D and a push-button switch is connected to RB0. When the switch is pressed a count is incremented and displayed on the 7-segment display. The display counts as 0 1 2 3 ...9 0 1.....

A PIC16F887 type microcontroller is used in the project with an 8MHz crystal.

Date: January 2009

File: ICD.C

```

*****/

```

```

#define PButton PORTB.F0

```

```

void main()

```

```

{
char count = 0;
unsigned char LED[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67};

```

```

ANSELH = 0;

```

```

TRISB = 1;

```

```

TRISD = 0;

```

```

//

```

```

// Endless loop

```

```

//

```

```

for(;;)

```

```

{

```

```

while(PButton);           // Wait until Button is pressed

```

```

PORTD = LED[count];      // Send count to display

```

```

count++;                 // Increment count

```

```

if(count == 10)count = 0; // Count is between 0 and 9

```

```

}

```

```

}

```

Figure 3: Program listing

- Click on PORTD value to change the display type to hexadecimal. You should see the PORT D output sent to the 7-segment display as the value of count changes. The program stops whenever it is waiting for the push-button to be pressed. Pressing the button on the target system advances the program to the next statement, where the data is sent to PORT D and the next number is displayed on the 7-segment display.
- In addition to the Watch Window, the ICD can also display the EEPROM data (View -> Debug Windows -> View EEPROM), code data (View -> Debug Windows -> View Code) or the RAM data (View -> Debug Windows -> View RAM).

Most Popular Tool

In this paper the commonly used microcontroller debugging and testing tools are described. It is discussed that although an in-circuit emulator is probably the most useful tool, it is also the most expensive tool to use.

In-circuit debuggers are currently the most popular debugging tools because of their low-cost and the ability to debug in real-time using the target microcontroller development system. ■

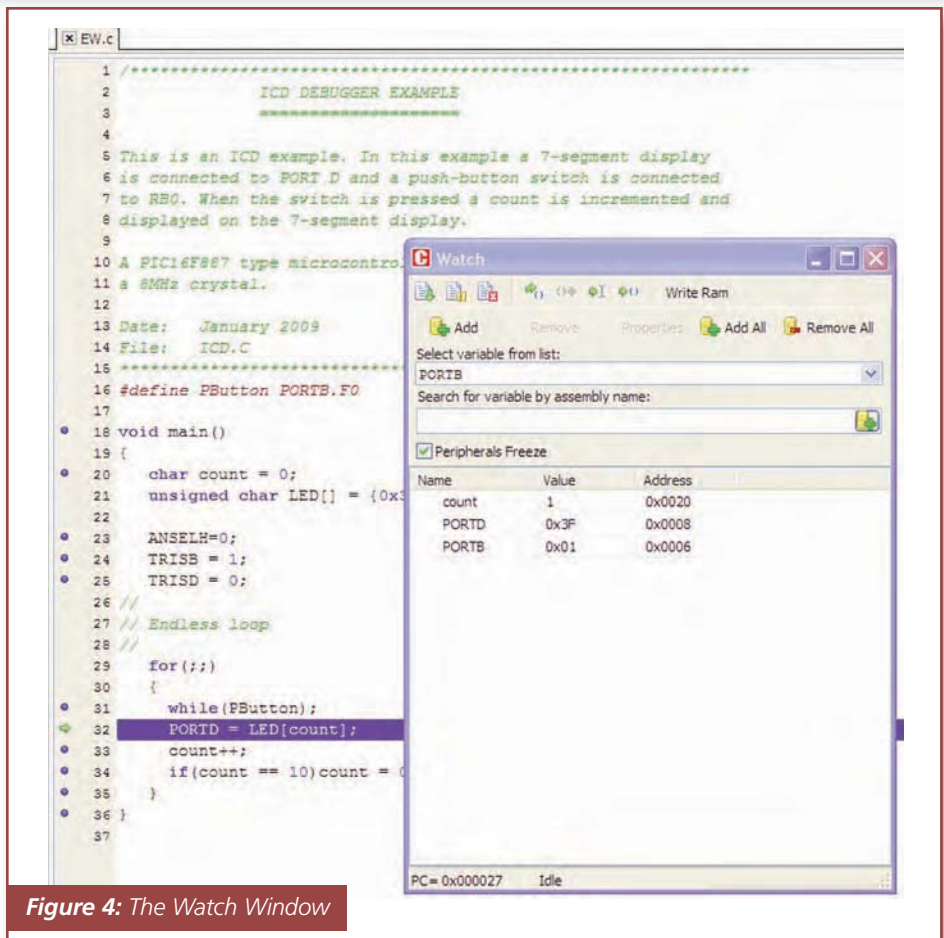


Figure 4: The Watch Window