

eICD2

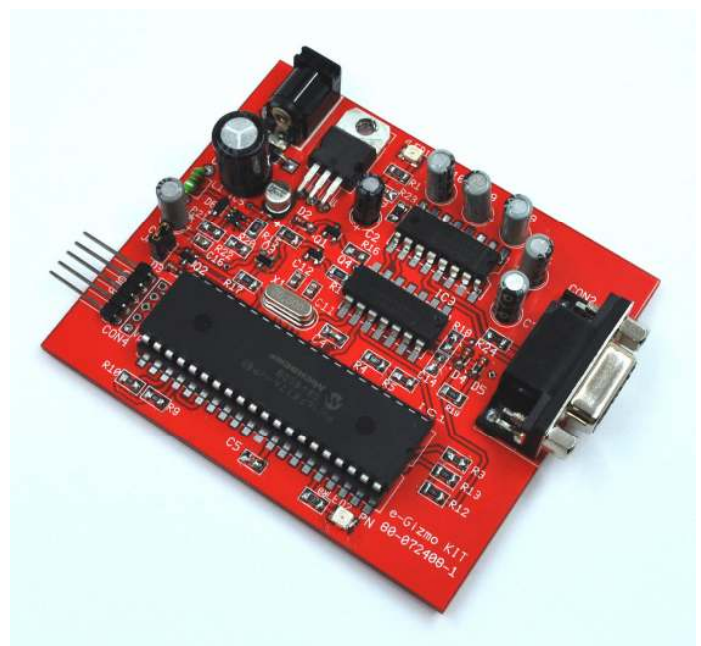
PIC Microcontroller Programmer/Debugger

Without a doubt, the PIC16F and PIC18F microcontrollers from Microchip rule the 8-bit microcontroller market. Due to its immense popularity, PIC users can find lots of resources in the Internet, from articles, books, to schematic of do-it-yourself (DIY) PIC programmer hardwares.

These DIY PIC programmers loved by hobbyists and students are typically of the serial port and parallel port-based designs. Its advantages are simplicity, ease of assembly, and low cost. Also, several Windows-based programmer applications that are freely downloadable from the Internet (i.e. WINPIC, ICPROG, etc.) support these hardware.

However, DIY programmers are severely limited in functionalities. Throughout the PIC application development cycle, the user will have to download the test code to the microcontroller many times. This process involves frequent removal/insertion of microcontroller chips between the target application board and the DIY programmer. This typical "remove-insert-program-and-pray" part of the code development cycle is simply inefficient.

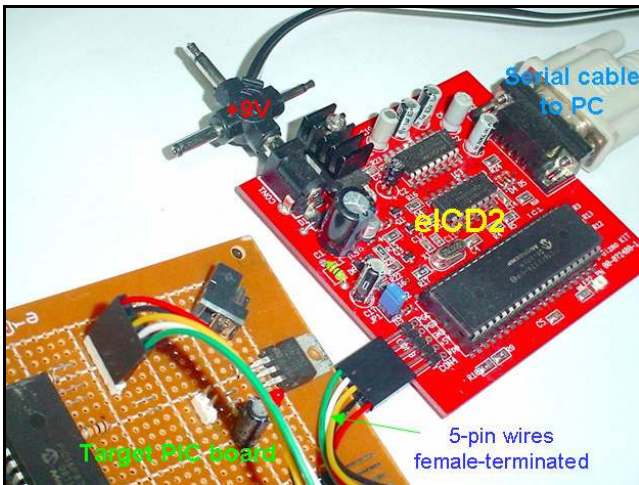
Wouldn't it be nice if instead of a DIY hardware, we can use a *specialized hardware* that can program the PIC microcontroller while it is in the target board to eliminate the frequent removal/insertion of the chip? And wouldn't it be nicer still if somehow this



specialized hardware can also communicate with the microcontroller *while the program is actually running inside the microcontroller* (a.k.a. "debugging")? We all knew that serious PIC users would love to look at the actual content of internal registers and program variables to check whether the code actually worked as intended. The good news is, here at E-gizmo, we have this special hardware just for you, the **eICD2 PIC Programmer** and **Debugger**.

The ultra low-cost eICD2 is a serial port-based ICSP programmer/debugger. The "IC" in the ICSP (in-circuit serial programmer) means that you can program/debug the microcontroller while the chip is mounted on its application board.

As an ICSP-equipped programmer, the eICD2 can program the microcontroller just like any programmer hardware would. But more importantly it can **debug** the target microcontroller. While in *debugger* mode, the eICD2 can access the internal registers of the PIC and variables of your program as it runs inside the microcontroller. The contents of these internal registers and variables can then be viewed, and even modified by you, via the MPLAB application. The eICD2's debugging capability is a must, if you ever hope of completing your PIC code bug-free.



eICD2 users readily acknowledge that it is the minimal PIC hardware development tool that one should have. If you ever hope of enhancing your PIC application development skill from that of a typical hobbyist to a full-pledge professional developer, you need the eICD2.

General Specification

Controller:	PIC16F877A
Connection:	RS232
Power supply:	9-12V
PC Development software:	MPLAB

Hardware Description

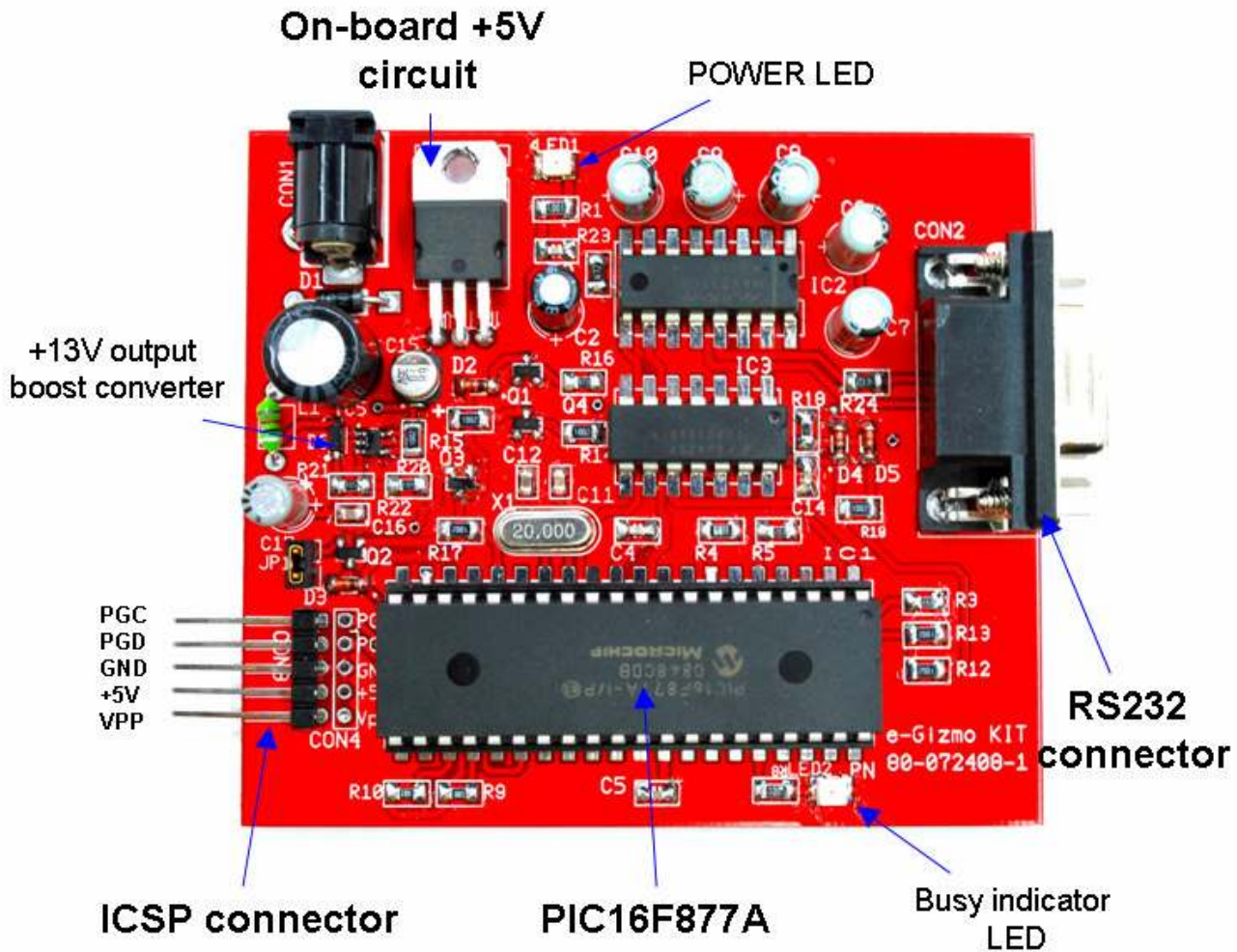
The eICD2 requires a 9-12V input supply voltage from a DC adapter. The onboard power supply circuit provides a +5V output to power up the rest of the board. The ICSP connector is 5-pin. Users will need a 5-pin female-header terminated wires to connect the eICD2 to the target board. The eICD2 itself is microcontroller-based. It has a 40-pin PIC16F877A. The eICD2 board communicates with the MPLAB application via serial port and with the target PIC board via the ICSP connectors. The microcontroller is pre-loaded with a bootloader + operating system program that does this communication task.

The board circuitry also includes a TPS61040-based low-power boost converter to generate the +13V signal needed to program the target PIC.

In all, the eICD2 needs the following extras:

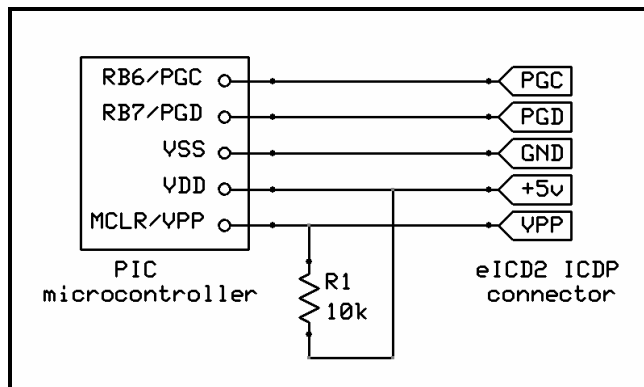
- (a) 9 or 12V output DC adapter
- (b) RS232 serial data cable
- (c) 5-wire ICSP connector

Though the board itself only consumes a small-amount of power, the regulator can run quite hot especially when you power the target board via the eICD2.



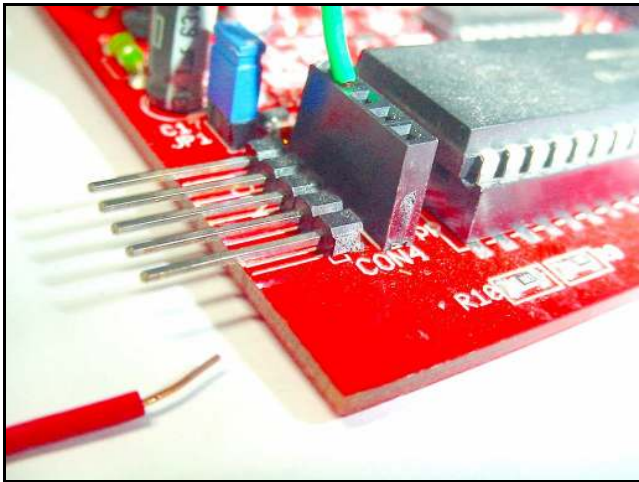
Connecting the eICD2 to Target microcontroller

The typical 5-pin **ICSP** connection between target PIC and **eICD2** is shown below.



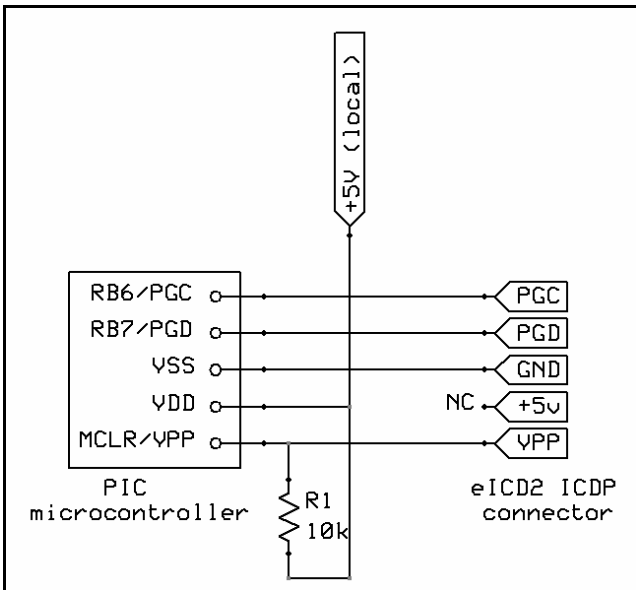
Using this connection, the target board derives its power from the eICD2. For most applications, it is recommended to disconnect any other external circuits on the target PIC's **PGD** (data) and **PGC** (clock) pin to prevent loading the pins and distorting the signals. Moreover, no capacitor of any value should be connected across the **MCLR/VPP** pin.

If you opt not to use the 5-wire female-terminated cable, you may modify the eICD2 board by soldering a 5-pin female connector (shown below) on the vacant ICSP slot on the PCB and use ordinary solid wires instead.



Some users might prefer using female header (above) for connection to microcontroller circuits assembled on breadboard.

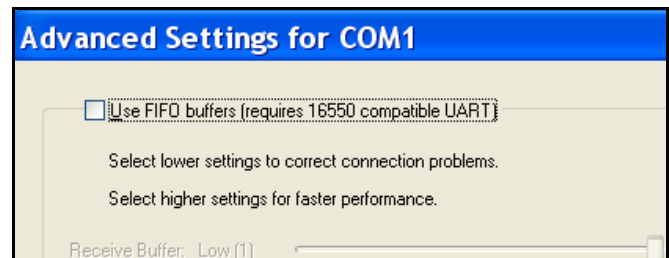
If the target board has its own +5V supply, the recommended ICSP connection is shown below.



If this is your preferred setup and you are using the 5-wire female-terminated cable, for convenience simply remove the jumper on **JP1** to disconnect the eICD2's +5v supply from the target PIC's V_{DD} .

Using eICD2 as Programmer

First, create your project in **MPLAB**. After successfully building the project, connect the eICD2 to a vacant serial port on your PC. The COM's FIFO buffer should also be disabled in Windows. To do this, open *Device Manager* and double-click on *Communication Port (COMx)*. On the *Properties* window that appear, select the *Port Settings* tab and click *Advanced...* Deselect the *Use FIFO buffers* checkbox and click *OK* (below). You only need to do this once the first time you use the eICD2.



At this time, connect the eICD2 to the target PIC board and connect the +9V supply to power up the eICD2 (and the target board). The target PIC should also be powered, either from the eICD2 or via its own supply.

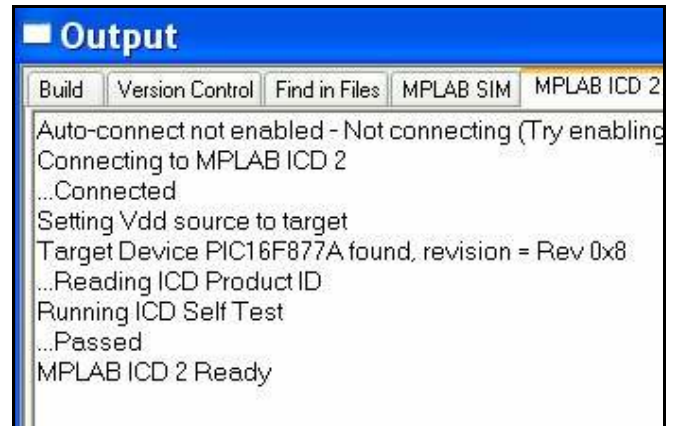
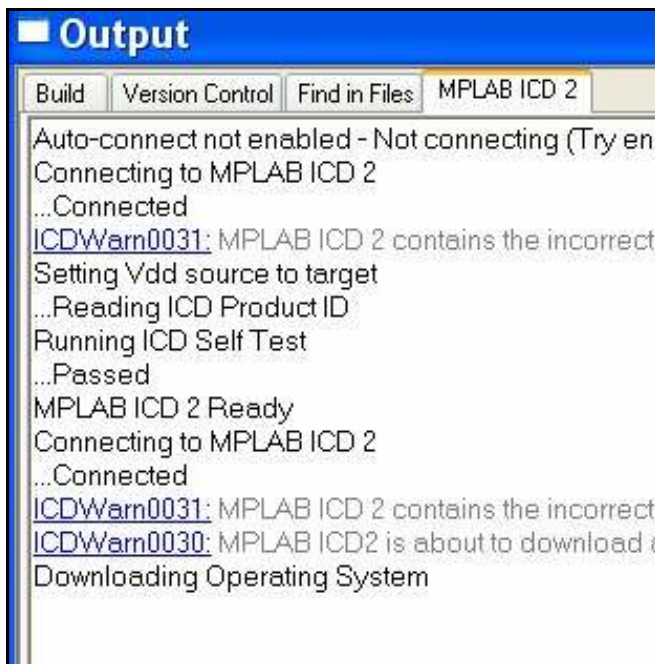
Next, in the **MPLAB** menu bar, select *Programmer > Select Programmer > MPLAB ICD2*. Click *Programmer > Settings*. In the *MPLAB ICD 2 Settings* window, click on the *Communication* tab and select the COM number corresponding to serial port where the eICD2 is connected. Also, select the desired *Baud Rate*. Click *OK*.



Now, to start the communication between MPLAB and the eICD2, select *Programmer > Connect*. If prompted to download a new operating system, as shown below, click Yes.



Downloading of the O.S. for the target PIC will take a few moments. If the *Output* window below is not visible, select *View > Output*.



After OS downloading is done, you are now ready to download the newly compiled and/or assembled program to the target PIC microcontroller. To do this, select *Programmer > Program*.



After programming, select *Programmer > Release from Reset* to run the program. To stop the program, select *Programmer > Hold in Reset*.

Using eICD2 as Debugger

To start a simple debugging session, open/create a PIC project in MPLAB. If you are using Assembly language, make sure you include the linker file required for debugging. The filename typically ends with *'i'* (i.e. use 16f877ai.lkr instead of 16f877a.lkr)

Power up the eICD2 and the target microcontroller. After successfully building the MPLAB project, select *Debugger > Select Tool > MPLAB ICD 2* to enable eICD2 as debugger. If the eICD2 was previously used as programmer, a message window shown below will prompt you to unload first the eICD2 as programmer.

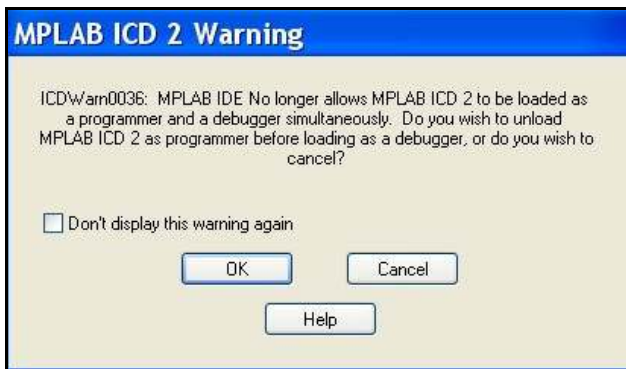
```

53          BANKSEL PORTC          ;se
54          bcf      PORTC, LED      ;in
55
56          loop
57          bsf      PORTC, LED      ;tu
58          call    one_sec_delay
59          bcf      PORTC, LED      ;tu
60          call    one_sec_delay
61          bcf      PORTC, LED      ;tu
62          call    one_sec_delay
63          goto    loop
64
65          one_sec_delay:
66          call    Delay            ;.25

```

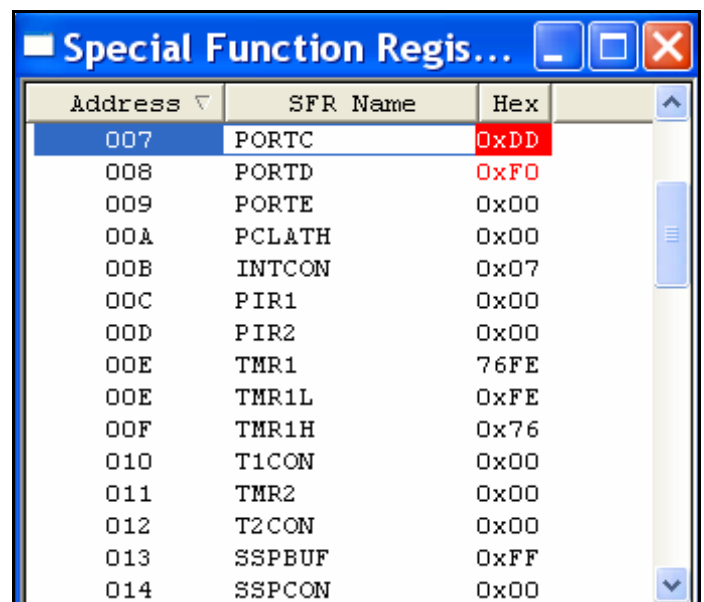
When program execution halts, you can now start single-stepping the program (*select Debugger > Step Into* or press F7 key). Single-stepping means the microcontroller will execute only one instruction and halt again. This is used to check the program flow line by line, instruction by instruction. Simply press F7 to single-step and wait until eICD2 finished communicating with the target microcontroller (and updating MPLAB) before pressing F7 again.

You may examine the actual content of the target PIC's special function registers (SFRs) and data RAM (variable locations) while single-stepping. To view the special function registers, select *View > Special Function Registers*.



A similar window will also appear if you are trying to select eICD2 as programmer while debugger mode is currently enable.

Next, select *Debugger > Connect* to connect to eICD2. Then, select *Debugger > Program* to download the program. Click *Debugger > Run* to run the program. To halt the program execution, *select Debugger > Halt*. The microcontroller will halt the program and MPLAB window will display what instruction, pointed to by the green arrow, is being executed next when program execution was halted.



To view the content of data RAM, select *View > File Registers*. You may switch between Hex view and Symbolic view.

Address	Hex	Decimal	Symbol Name
019	0x00	0	TXREG
01A	0x00	0	RCREG
01B	0xFF	255	CCPR2
01C	0x14	20	CCPR2H
01D	0x00	0	CCP2CON
01E	0x00	0	ADRESH
01F	0x00	0	ADCON0
020	0xFA	250	
021	0xC7	199	
022	0x01	1	
023	0x00	0	
024	0x00	0	
025	0x02	2	
026	0x00	0	
027	0x10	16	
028	0x02	2	
029	0x00	0	
02A	0x00	0	

You may also modify the content of the variables locations (data RAM) to influence the flow of the program execution.

There are many more debugging features for MPLAB and eICD2. You might want to use **breakpoint**. You can redo the previous exercise above, but this time, add a breakpoint on your code, preferably before the main loop. When you run the microcontroller, it will execute the lines of codes before and beside the breakpoint and then halt.

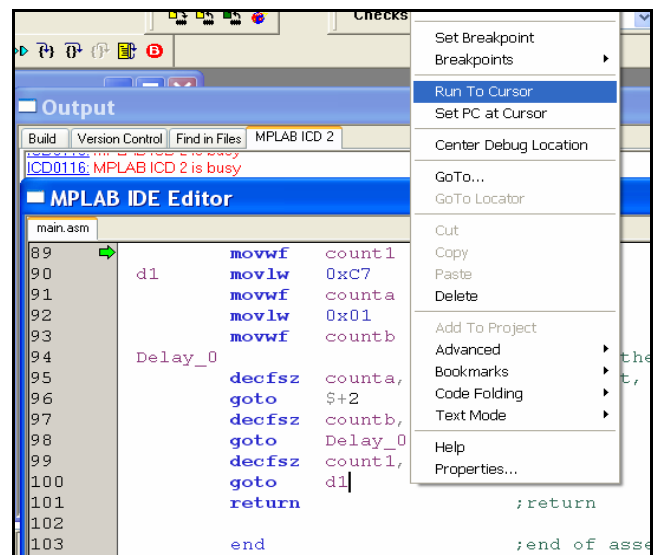
```

45     countb equ    UR22      ;DATA
46
47
48     main    ORG    0x00
49
50     BANKSEL TRISC      ;sele
51     bcf    TRISC, LED  ;RC0
52
53     BANKSEL PORTC     ;sele
54     bcf    PORTC, LED ;init
55
56     loop
57     bsf    PORTC, LED  ;turn
58     call  one_sec_delay
59
60     bcf    PORTC, LED  ;turn
61     call  one_sec_delay
  
```

You can also modify the 8-bit content of the SFRs. For example, you may change the PORTC register and write a 1 or 0 to turn on/off an LED at PC0 pin. Simply double-click on the register value under the HEX column in and enter a new desired value. Wait for eICD2 and MPLAB to finish updating and writing the new value to the microcontroller.

In addition, you can use the Run To Cursor command in the context menu or use the Step Over function to exit from very long loops (i.e. delay functions, etc).

Address	SFR Name	Hex
003	STATUS	0x1A
004	FSR	0x90
005	PORTA	0x10
006	PORTB	0x37
007	PORTC	0xFE
008	PORTD	0xFF
009	PORTE	0x00
00A	PCLATH	0x00
00B	INTCON	0x07
00C	PIR1	0x00
00D	PIR2	0x00
00E	TMR1	76FE
00E	TMR1L	0xFE
00F	TMR1H	0x76
010	T1CON	0x00



Refer to MPLAB Help for more information on debugging PIC projects.