Development Kit
For the PIC® MCU
Exercise Book

PIC12F675

December 2010

CCS Inc.

Custom Computer Services, Inc.
Brookfield, Wisconsin, USA
262-522-6500

Recognized
Microchip
Third-Party
Tool Provider

**Custom Computer Services, Inc.
proudly supports the Microchip
brand with highly optimized
C compilers and embedded
software development tools.**

Inventory

❑ Use of this kit requires a PC with Windows 95, 98, ME, NT, 2000 or XP. The PC must have a spare 9-Pin Serial or USB port, a CD-ROM drive and 75 MB of disk space.

❑ The diagram on the following page shows each component in the PIC12F675 kit. Ensure every item is present.

Software

❑ Insert the CD into the computer and wait for the installation program to start. If your computer is not set up to auto-run CDs, then select **My Computer** and double-click on the CD drive.

❑ Click on **Install** and use the default settings for all subsequent prompts by clicking NEXT, OK, CONTINUE…as required.

❑ Identify a directory to be used for the programs in this booklet. The install program will have created an empty directory **c:\program files\picc\projects** that may be used for this purpose.

❑ Select the compiler icon on the desktop. In the PCW IDE, click **Help>About** and verify a version number is shown for the IDE and PCM to ensure the software was installed properly. Exit the software.

Hardware

❑ Connect the PC to the ICD(6) using the USB cable.[1] Connect the prototyping board (10) to the ICD using the modular cable. Plug in the DC adaptor (9) to the power socket and plug it into the prototyping board (10). The first time the ICD-U is connected to the PC, Windows will detect new hardware. Install the USB driver from the CD or website using the new hardware wizard. The driver needs to be installed properly before the device can be used.

❑ The LED should be red[2] on the ICD-U to indicate the unit is connected properly.

❑ Run the Programmer Control Software by clicking on the CCSLOAD icon on the desktop. Use CCSLOAD Help File for assistance.

❑ The software will auto-detect the programmer and target board and the LED should be illuminated green. If any errors are detected, go to Diagnostic tab. If all tests pass, the hardware is installed properly.

❑ Disconnect the hardware until you are ready for Chapter 4. Always disconnect the power to the Prototyping board before connecting/disconnecting the ICD or changing the jumper wires to the Prototyping board.

[1] ICS-S40 can also be used in place of ICD-U. Connect it to an available serial port on the PC using the 9 pin serial cable. There is no driver required for S40.

[2] ICD-U40 units will be dimly illuminated green and may blink while connecting.
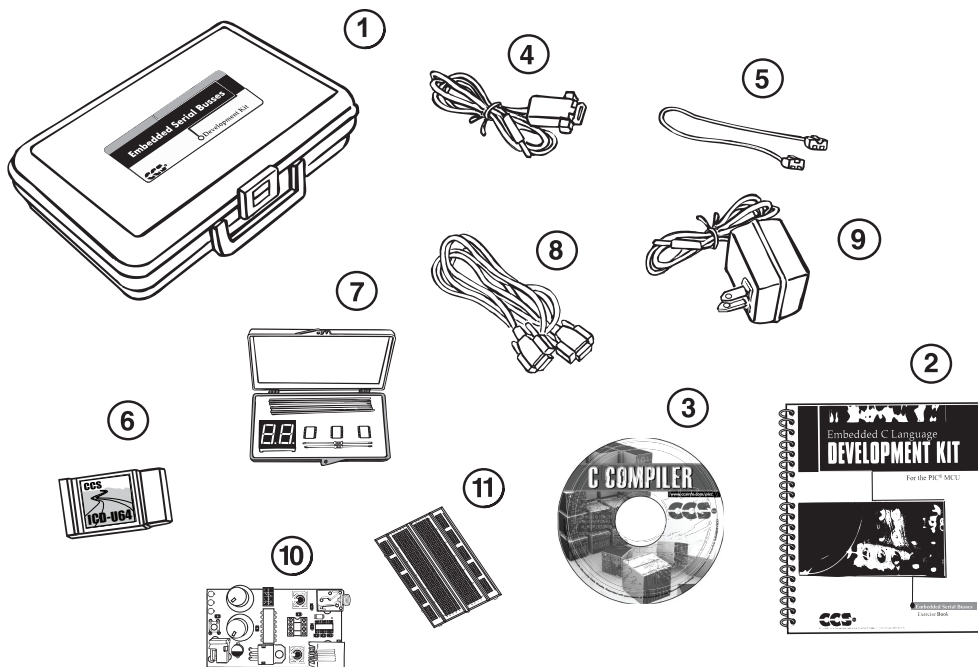
① Carrying case

② Exercise booklet

③ CD-ROM of C compiler (optional)

④ Serial PC cable to Prototyping board

⑤ Modular ICD cable to Prototyping board

⑥ ICD unit for programming and debugging of the PIC® MCU

⑦ Parts box includes:

    PIC12F675 chip
    Green, Yellow and Red LEDs
    Thermistor
    Two 10K resistors
    Jumpers to connect the Prototyping board to the breadboard

⑧ Serial (or USB) PC to ICD cable

⑨ AC Adaptor (9VDC)

⑩ Prototyping board with a PIC12F675 processor chip (See inside front and back cover for details on the board layout and schematic)

⑪ Breadboard for prototyping circuits

⑫ IC Extractor

2 USING THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Editor

❑ Open the PCW IDE. If any files are open, click **File>Close All**

❑ Click **File>Open**. Select the file: **c:\program files\picc\examples\ex_stwt.c**

❑ Scroll down to the bottom of this file. Notice the editor shows comments, preprocessor directives and C keywords in different colors.

❑ Move the cursor over the **Set_timer0** and click. Press the F1 key. Notice a Help file description for set_timer0 appears. The cursor may be placed on any keyword or built-in function and F1 will find help for the item.

❑ Review the editor special functions by clicking on **Edit**. The IDE allows various standard cut, paste and copy functions along with setting bookmarks and various C specific functions.

❑ Review the editor option settings by clicking on **Options>Editor Properties**. The IDE allows selection of the tab size, editor colors, fonts, and many more. Click on **Options>Customize** to select which icons appear on the toolbars.
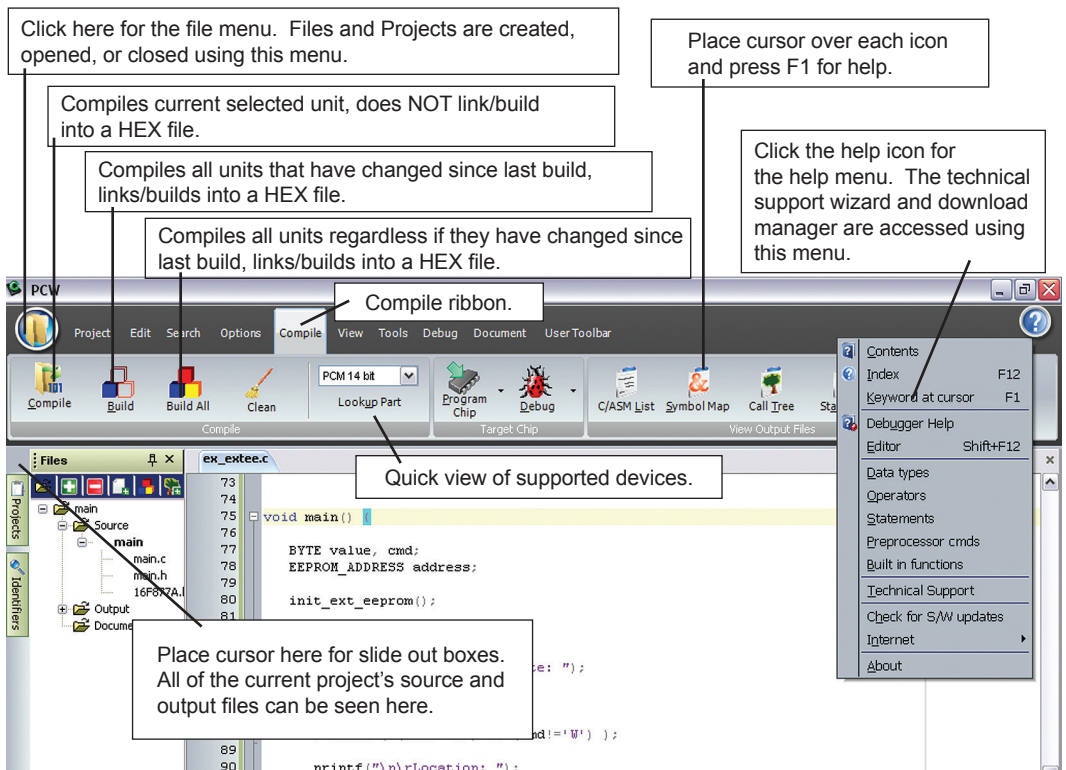
Compiler

❑ Use the white box on the toolbar to select the compiler. CCS offers different compilers for each family of Microchip parts. All the exercises in this booklet are for the PIC12F675 chip, a 14-bit opcode part. Make sure **14 bit** is selected in the white box.

❑ The main program compiled is always shown in the lower right corner of the IDE. If this is not the file you want to compile, then click on the tab of the file you want to compile. Right click into editor and select **Make file project**.

❑ Click **Options>Include Dirs…** and review the list of directories the compiler uses to search for included files. The install program should have put two directories in this list to point to the device: *.h files and the device drivers*.

❑ Normally the file formats need not be changed and global defines are not used in these exercises. To review these settings, click **Options>File Formats** and **Options>Global Defines**.

❑ Click the compile icon to compile. Notice the compilation box shows the files created and the amount of ROM and RAM used by this program. Press any key to remove the compilation box.

Viewer

❑ Click **View>Symbol Map**. This file shows how the RAM in the micro-controller is used. Identifiers that start with @ are compiler generated variables. Notice some locations are used by more than one item. This is because those variables are not active at the same time.

❑ Click **View>C/ASM List.** This file shows the original C code and the assembly code generated for the C. Scroll down to the line:

```
int_count=INTS_PER_SECOND;
```

❑ Notice there are two assembly instructions generated. The first loads 4C into the W register. INTS_PER_SECOND is #defined in the file to 76. 4C hex is 76 decimal. The second instruction moves W into memory location. Switch to the Symbol Map to find the memory location where int_count is located.

❑ Click **View>Data Sheet**, then **View.** This brings up the Microchip data sheet for the microprocessor being used in the current project.

Click here for the file menu. Files and Projects are created, opened, or closed using this menu.

Place cursor over each icon and press F1 for help.

Compiles current selected unit, does NOT link/build into a HEX file.

Click the help icon for the help menu. The technical support wizard and download manager are accessed using this menu.

Compiles all units that have changed since last build, links/builds into a HEX file.

Compiles all units regardless if they have changed since last build, links/builds into a HEX file.

Compile ribbon.

Quick view of supported devices.

Place cursor here for slide out boxes. All of the current project's source and output files can be seen here.

3 PIC12F675 OVERVIEW

- ❑ Key features of the PIC12F675:
  - Small 8 pin package (up to 6 I/O pins)
  - Internal 4 MHz oscillator
  - Four channel, 10 bit A/D converter
  - Analog comparator
  - Two timers
  - Operates 2V to 5 V

- ❑ Memory:
  - 1024 words of reprogrammable program memory
  - 64 bytes of RAM
  - 128 bytes of data EEPROM

- ❑ Because of the small number of pins and numerous features on this chip each pin can have a number of different functions. It is very important to be sure the chip is properly programmed to use each pin in the desired fashion. The following table identifies each pin, what it can be used for and how to use it in that way. Note that the data sheet identifies the I/O pins as GP0-GP5. In the compiler and this booklet the pins are identified as A0-A5 to make the code more compatible with other chips.

- ❑ For debugging Microchip makes a 14-pin version of the PIC12F675. It is a PIC12F675-ICD. The extra pins allow an ICD to be connected to the chip with the normal eight pins are used for the application under development. The protoboard has a switch to connect the ICD unit to either the 14-pin socket for debugging or the eight-pin socket for programmin g an eight-pin part.

A0	Analog Input	`setup_ADC_ports( AN0_ANALOG );`
	Comparator + input	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_comparator( A0_A1 );` [2]
	General I/O	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_comparator( NC_NC );` [1] `enable_interrupts(INT_RA0);` [3]
A1	Analog Input	`setup_ADC_ports ( AN1_ANALOG );`
	Comparator - input	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_comparator( A0_A1 );` [2]
	Ref Voltage for ADC	`setup_ADC_ports ( AN1_ANALOG | VSS_VREF);` [2]
	General I/O	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_comparator( NC_NC );` [1] `enable_interrupts(INT_RA1);` [3]

A2	Analog Input	`setup_ADC_ports ( AN2_ANALOG );` `setup_timer0(RTCC_INTERNAL);`
	Comparator Output	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_timer0(RTCC_INTERNAL);` `setup_comparator( A0_A1_OUT_ON_A2 );` [2]
	Timer 0 Clock Input	`setup_timer0(RTCC_EXT_L_TO_H); // or H_TO_L`
	General I/O	`setup_ADC_ports ( NO_ANALOGS);` [1] `setup_timer0(RTCC_INTERNAL);` [1] `setup_comparator( NC_NC );` [1] `enable_interrupts(INT_EXT);` [3] `ext_int_edge(L_TO_H);` [3]
A3	Chip Reset (MCLR)	`#fuses MCLR`
	General Input	`#fuses NOMCLR` `enable_interrupts(INT_RA3);` [3]
A4	Analog Input	`setup_ADC_ports ( AN3_ANALOG );`
	Timer 1 Run/Pause	`#fuses  INTRC_IO` `setup_timer1 (T1_INTERNAL | T1_GATE);` [2]
	Crystal	`#fuses  XT` [4]
	Buffered Clock/4 Output	`#fuses  INTRC` [5]
	General I/O	`#fuses  INTRC_IO` `setup_ADC_ports (NO_ANALOGS);` [1] `enable_interrupts(INT_RA4);` [3]
A5	Timer 1 Clock Input	`setup_timer1(T1_EXTERNAL );`
	Crystal or Osc Input	`#fuses  XT` [4]
	General I/O	`#fuses  INTRC_IO` [6] `enable_interrupts(INT_RA5);` [3]

[1] or any option set that excludes the pin
[2] or any option set that includes the pin
[3] Optional - Use only if an interrupt is required for the pin
[4] Any option other than INTRC or INTRC_IO
[5] Any option other than INTRC_IO or RC_IO
[6] INTRC or INTRC_IO

NOTES

- In addition to the above pin usage at run time, the eight-pin part can use A0, A1 and A3 for in circuit programming when the chip is not running.

4 COMPILING AND RUNNING A PROGRAM

❑ Open the PCW IDE. If any files are open, click **File>Close All**

❑ Click **File>New>Source File** and enter the filename **EX3.C**

❑ Type in the following program and **Compile.**

```
#include <12f675.H>
#device ICD=TRUE
#fuses intrc_io,nomclr,nowdt,noprotect
#use delay(clock=4000000)

#define RED_LED    PIN_A4
#define YELLOW_LED  PIN_A5
#define GREEN_LED   PIN_A2

void main() {
    setup_comparator(NC_NC);
    setup_adc_ports(NO_ANALOGS);
    do {
        output_low(RED_LED);
        delay_ms(500);
        output_high(RED_LED);
        output_low(YELLOW_LED);
        delay_ms(500);
        output_high(YELLOW_LED);
        output_low(GREEN_LED);
        delay_ms(500);
        output_high(GREEN_LED);
    } while(TRUE);
}
```

❑ Connect the ICD to the Prototyping board using the modular cable, and connect the ICD to the PC. Power up the Prototyping board.

❑ Click **Debug>Enable Debugger** and wait for the program to load.

❑ Click the green go icon: **GO**
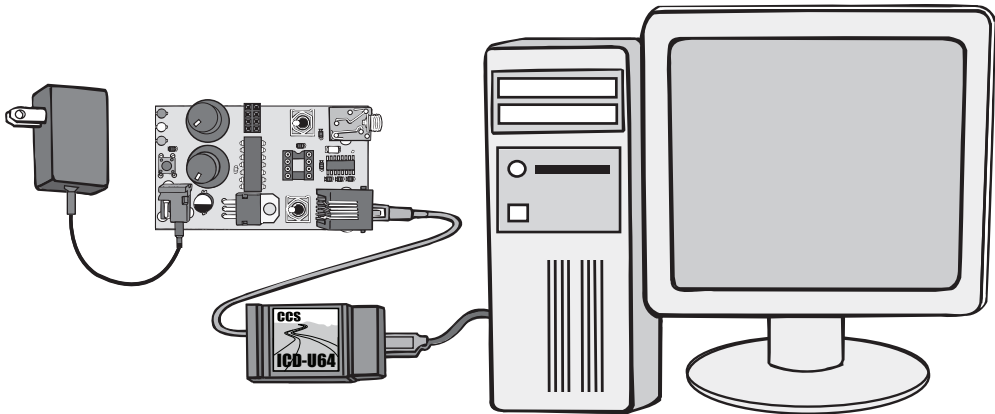
❑ Expect the debugger window status block to turn yellow indicating the program is running.

❑ The green LED on the Prototyping board should be flashing. One second on and one second off.

❑ The program can be stopped by clicking on the stop icon: 🛑

❑ Add the following to the #defines:
```
#define PUSHBUTTON   PIN_A3
```

❑ Add the function to the program:
```
Wait_for_one_press() {
  while(input(PUSHBUTTON)) ;
  while(!input(PUSHBUTTON)) ;
}
```

❑ Replace each delay_ms(500) with the following line:
```
Wait_for_one_press();
```

❑ Save the program as **EX4A.C.**

❑ Compile and run the program. The LEDs on the protoboard should now only change when the button is pressed.

FURTHER STUDY

> *A   Note that the "output_low" turns the LED on because the other end of the LED is +5V.*
> *B   PIN_A3 reads a 1 when the pushbutton is not pressed.*

5 ▶ DEBUGGING

❑ Open **EX4.C** and start the debugger **Debug>Enable Debugger**.

❑ Click the reset icon to ensure the target is ready.

❑ Click the step-over 🐾 icon until `wait _ for _ one _ press ()` is highlighted. This is the step over command. Each click causes a line of C code to be executed. The highlighted line has not been executed, but the line about to be executed.

❑ Step over the `wait _ for _ one _ press` and press the pushbutton. Notice the debugger now stops since the function terminates. Notice that one click executed the entire function. This is the way step-over works.

❑ Click the **Watch** tab, then the add icon ➕ to add a watch. Enter **count or choose count the variables from list**, then click **Add Watch**. Notice the value shown. Continue to step over through the loop a few more times (press the button as required) and notice the count watch increments.

❑ Step-over until the call to `show _ binary _ on _ leds(count);` is highlighted. This time, instead of step over, use the standard step icon 🐾 several times and notice the debugger is now stepping into the function.

❑ Click the GO icon 🟢 to allow the program to run. Press the prototype button a couple of times to verify that the program is running normally. Click the stop icon to halt execution. Notice the C source line that the program stopped on. This is the line were the program is waiting for a button press.

❑ In the editor, click on `show _ binary _ on _ leds(count);` to move the editor cursor to that line. Then click the Breaks tab and click the add ➕ icon to set a breakpoint. The debugger will now stop every time that line is reached in the code. Click the GO icon and then press the prototype button. The debugger should now stop on the breakpoint. Repeat this a couple of times to see how the breakpoint works. Note that the ICD with PIC16 chips only allow one breakpoint at a time.

❑ Click **View>C/ASM list**. Scroll down to the highlighted line. Notice that one assembly instruction was already executed for the next line. This is another side effect of the ICD debugger. Sometimes breakpoints slip by one ASM instruction.

❑ Click the step-over icon a few times and note that when the list file is the selected window, the debugger has executed one assembly instruction per click instead of one entire C line.

❑ Close all files and start a new file **EX5.C** as follows:

```
#include <12f675.h>
#device ICD=TRUE
#fuses intrc_io,nomclr, nowdt,noprotect
#use delay(clock=4000000)

void main() {
    int a,b,c;

    a=11;
    b=5;
    c=a+b;
    c=b-a;
    while(TRUE);
}
```

❑ Compile the program and step-over until the c=a+b is executed. Add a watch for c and the expected value is 16.

❑ Step-over the subtraction and notice the value of c. The **int** data type by default is not signed, so c cannot be the expected –6. The modular arithmetic works like a car odometer when the car is in reverse only in binary. For example, 00000001 minus 1 is 00000000, subtract another 1 and you get 11111111.

❑ Reset and again step up to the c=a+b. Click the **Eval** tab. This pane allows a one time expression evaluation. Type in a+b and click **Eval** to see the debugger and calculate the result. The complete expression may also be put in the watch pane as well. Now enter b=10 and click **Eval**. This expression will actually change the value of B if the "keep side effects" check box of the evaluation tab is checked. Check it and click **Eval** again. Step over the addition line and click the **Watch** tab to observe the c value was calculated with the new value of b.

6 ANALOG INPUT

❑ The PIC12F675 chip has four pins that may be used to read an analog voltage. The following is a simple program (**EX6.c)** to read one analog pin.

```
#include <12f675.H>
#device ICD=TRUE
#fuses intrc_io,nomclr,nowdt,noprotect
#use delay(clock=4000000)

#define RED_LED  PIN_A4
#define YELLOW_LED  PIN_A5
#define GREEN_LED  PIN_A2
#define cutoff 128        // 2.5 Volts
#define neutral_zone 25   // 0.5 Volts

void main() {
    int reading;

    setup_adc_ports( AN0_ANALOG );
    setup_adc( ADC_CLOCK_INTERNAL );
    set_adc_channel( 0 );

    while(TRUE) {
        output_high(GREEN_LED);
        output_high(YELLOW_LED);
        output_high(RED_LED);
        reading = read_adc();
        if(reading<(cutoff-neutral_zone/2))
            output_low(GREEN_LED);
        else if (reading>(cutoff+neutral_zone/2))
            output_low(RED_LED);
        else
            output_low(YELLOW_LED);
        delay_ms(100);
    }
}
```

❑ Compile and Run the program. Verify that the Prototyping board knob (A0) is turned so the green LED is on when it is low, the red LED when high and the yellow LED for a small region in the center.

- By default, the analog to digital converter is 8 bits. Thus, a range of 0 to 5 volts analog is represented by the numbers 0-255. The A/D reading can be converted to volts by the formula:

    Volts = reading*(5.0/255)

- The setup_adc_ports function call determines what pins are set to be analog inputs. The setup_adc function call determines how fast the conversion is done. The internal clock option uses an internal RC clock. Although the timing is not exact, it is long enough for an accurate conversion. The time can be based off the instruction clock for more precise timing.

- The set_adc_channel function sets the A/D converter to channel 0 (AN0 or A0). This switches an internal mux in the part, but does not start an A/D conversion. Even though a conversion has not started, there is a small capacitor in the chip that must charge up after the port switch and before the voltage is read. This is fast with a low impedance input, but for a higher impedance input, a small delay should be put in after the channel is changed.

- The call to read_adc starts a conversion, waits for it to complete and returns the result. The conversion time is around 20μs.

SPECIAL NOTE

- For programming the PIC12F675 prototyping board with the CCS ICD, make sure potentiometer knobs on board are set to the far left (or 8:00)

❑ The PIC12F675 has one internal analog comparator. When the analog voltage on the + input is greater than the - input the output will be high. The inputs can either be a pin on the chip or an internal voltage reference. The output can be a pin on the chip, an interrupt or just a bit in a register than can be pulled.

❑ Enter, compile and load the following program:

```
#include <12f675.H>
#device ICD=TRUE
#fuses intrc_io,nomclr,nowdt,noprotect
#use delay(clock=4000000)

void main() {
    setup_adc_ports( NO_ANALOGS );
    setup_comparator( A0_A1_OUT_ON_A2 );

    while(TRUE) {
    }
}
```

❑ Run the program and notice that when POT A0 is less than POT A1 that LED A2 (green) is on. After initialization the software in this program does nothing. The hardware is doing it all.

❑ Add the following code to the inside of the while loop:

```
if(C1OUT)
output_low(PIN_A4);  // Red LED
else
output_high(PIN_A4); // Red LED
```

❑ Compile and run the program. The C1OUT variable will be true if the comparator output is high. This should set the Red LED on when the comparator output is high (opposite of the Green LED).

❑ There are two formulas for the internal voltage reference, high and low:
  HIGH: Voltage = Vdd/4 + N*(Vdd/32)
  LOW: Voltage = N*(Vdd/24)
In order to set the voltage reference to 2.03 volts using the high range (with Vdd=5V) set N to 5.

❑ Replace the setup_comparator line in the program with the following:

```
setup_comparator( A1_VR_OUT_ON_A2 );
setup_vref( VREF_HIGH | 5 );
```

❑ Compile and run the program. This should light the Green LED on when the POT A1 is set greater than 2.03V and the Red LED otherwise.

❑ The PIC12F675 has 128 bytes of internal data EEPROM. This memory retains the data even when the power is removed.

❑ RS-232 is a popular serial communications standard used on most PCs and many embedded systems. Two wires are used (in addition to ground), one for outgoing data and one for incoming data. The top switch on the prototyping board is used to route A4 and A5 to the RS232 level converter instead of the LEDs.

❑ Create the file **EX8.c** as follows:

```c
#include <12f675.H>
#device ICD=TRUE
#fuses intrc_io,nomclr,nowdt,noprotect
#use delay(clock=4000000)
#use RS232(baud=9600, xmit=PIN_A5, rcv=PIN_A4)
#include <input.c>

void main() {
    int count,i,j,address,value;

    count = read_eeprom(0);
    do {
        printf("\r\n\nEEPROM:\r\n");
        for(i=0; i<=7; ++i) {
            for(j=0; j<=15; ++j)
                printf( "%2x ", read_eeprom( i*16+j ) );
            printf("\n\r");
        }
        printf("\r\nLocation to change: ");
        address = gethex();
        printf("\r\nNew value: ");
        value = gethex();

        write_eeprom( address, value );

    } while (TRUE);
}
```
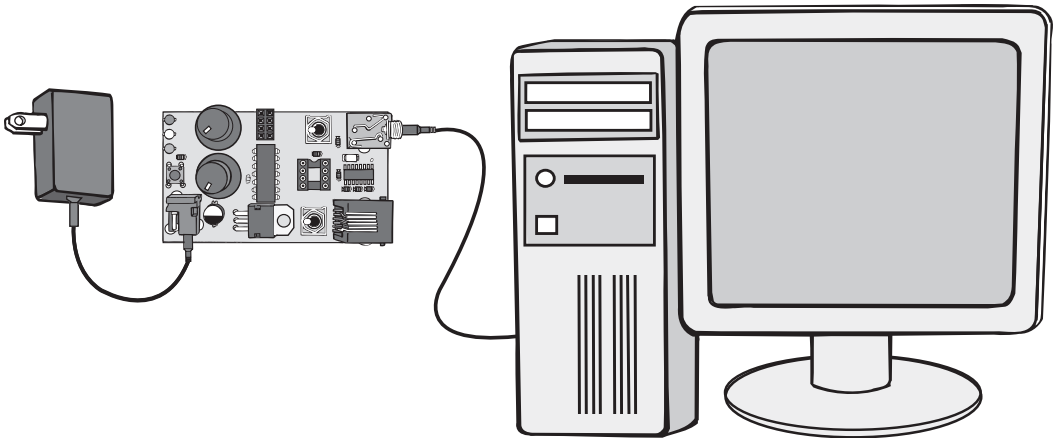
- ☐ Connect the proto board to the PC as shown above. Move the top switch position to RS-232 and bottom switch position to debug.

- ☐ Connect the serial cable to a second port on the PC or a dump terminal and run a serial I/O program. Set the correct port and baud rate settings.

- ☐ Power up the proto board, compile the program, enable the debugger and run the program. A dump of the data EEPROM should appear. Enter an address and new value to change a byte.

- ☐ Exit the Debugger. Remove power and reconnect the power to the board. Enable the debugger and run the program. Notice the EEPROM shows the last values written. Change the values, cycle power and verify the data is retained.

> **NOTES**
>
> - The first argument to read/write _eeprom is the address in the EEPROM to write the byte to. The PIC12F675 part ranges from 0 to 127.
>
> - There is a limit as to how many times a given location in the data EEPROM can be written to. For example, the PIC12F675 chip allows 1,000,000 times. For this reason, a program should be designed not to write any more often than is necessary. For example, if the volume setting for a TV is being saved, one might wait until there are no changes for 5 seconds before saving a new value to EEPROM. Some system designs can give an early warning on power down and the program can only save to EEPROM at power down.

9 TIMERS

- ❑ The PIC12F675 has two built-in timers. Each timer has a different set of features. The following example will use Timer #1 to measure the time it takes to execute some C code.

- ❑ Create the file EX9.c as follows:

```c
#include <12f675.h>
#device ICD=TRUE
#fuses intrc_io,nomclr, nowdt,noprotect
#use delay(clock=4000000)
#use RS232(baud=9600, xmit=PIN_A5, rcv=PIN_A4)

void main() {
    long time;

    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
    set_timer1(0);

    time = get_timer1();
    printf("Time in ticks is %lu\r\n",time);
    while(TRUE);
}
```

- ❑ Compile and Run the program. Check the monitor tab to see the result.

- ❑ This number is the number of timer ticks that it took to set and read the timer. The T1_INTERNAL indicates the instruction clock is the source for the timer. The instruction clock is the oscillator divided by 4, or in our case, 0.2ms. This time represents the overhead of our timer code and may now be used in a more useful example.

- ❑ Modify the program as follows and replace the ??? with the number of ticks determined in the above program.

```
#include <12f675.h>
#device ICD=TRUE
#fuses intrc_io,nomclr, nowdt,noprotect
#use delay(clock=4000000)
#use RS232(baud=9600, xmit=PIN_A5, rcv=PIN_A4)

void main() {
    long time;
    long a,b,c;

    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
    set_timer1(0);
    a=b*c;
    time = get_timer1();
    time -= ???; // subtract overhead
    printf("Time is %lu microseconds.\r\n", time);
        while(TRUE);
}
```

❑ Compile and run the program. Check the signal output to see the result.

NOTES

- Since "time" represents the number of 1 microsecond ticks that it takes to do *a=b*c", then time is the number of microseconds it takes to do that one line of C code.
- All the timers on the PIC12F675 count up and when the maximum value is reached, the timer restarts at 0. The set_ timer1(0) resets the timer to 0. Timer 1 is 16 bits and the range is 0 to 65535. This means it will overflow every 65535us. This is the largest time the program will be able to measure.
- If using T1_EXTERNAL instead of INTERNAL, then the timer would increment every time pin A5 cycled. This makes it more of a counter.
- If using T1_DIV_BY_2 instead of BY_1, then the timer would increment once for every 2 instruction clocks. This makes the timer tick 2us and the range of the timer is now 131070us.
- The following is a summary of the timers on the PIC12F675 chip:

#0	Input is Instruction Clock or external pin Range is 0-255 Input can be divided by 1,2,4,8,16,32,64,128,256 Can generate interrupt on each overflow
#1	Input is Instruction Clock or external pin Range is 0-65535 Input can be divided by 1,2,4,8 Can generate interrupt on each overflow

10 INTERRUPTS

❑ An interrupt is a specific event that causes the normal program execution to be suspended wherever it is and an interrupt function is executed. Normal program execution continues when the interrupt function returns. The PIC12F675 has a number of interrupt sources such as a timer overflow, a change on a pin, and external interrupt. Use **View>Valid Interrupts** for a complete list of interrupts for this chip.

❑ In this exercise, the timer 1 overflow interrupt will be used to extend the timer 1 timer to see all the valid interrupts from 16 bits to 32 bits by counting the number of times the timer overflows. Create the file **EX10.c** as follows:

```c
#include <12f675.h>
#device ICD=TRUE
#fuses intrc_io,nomclr, nowdt,noprotect
#use delay(clock=4000000)
#use RS232(baud=9600, xmit=PIN_A5, rcv=PIN_A4)

#define PUSH_BUTTON PIN_A3
int16 overflow_count;

#int_timer1
void timer1_isr() {
    overflow_count++;
}

void main() {
    int32 time;

    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
    enable_interrupts(int_timer1);
    while(TRUE) {
        enable_interrupts(global);

        while(input(PUSH_BUTTON));// Wait for press
        set_timer1(0);
        overflow_count=0;

        while(!input(PUSH_BUTTON));//Wait for release
        disable_interrupts(global);
        time = get_timer1();
        time = time + ((int32)overflow_count<<16);
        time -= 15; // subtract overhead
        printf("Time is %lu.%06u seconds.\r\n",
            time/1000000, (time%1000000));
    }
}
```

❑ Compile and Run the program. Press the button, release, and note the time it was held down is shown to 6 decimal places in the Monitor pane.

FURTHER STUDY

**A** Make a version of this program that prints in the format
  MM:SS.FFFFFF
  Where MM is minutes, SS is seconds and FFFFFF is fractions of a second.
B *Add a second interrupt using timer 0 to interrupt every 13.1ms. In the interrupt routine, count interrupts and when 76 interrupts have happened, do a putc('.');. This should display a period every second while interrupts are enabled.*

❑ The protoboard can be used to program a eight-pin PIC12F675 chip. Load the **EX4A.C** program. Remove the #device ICD=TRUE line, add the following before the while loop and save the file as **EX11.C.**

```
port_a_pullups(8);  // Turn on pull-up on A3
```

❑ Remove the power from the protoboard and insert the 12F675 8 pin chip as shown in the diagram on the opposite page.

❑ Make sure the bottom switch is in the PROGRAM position and power up the board.

❑ Compile the **EX11.C** program.

❑ Use **Tools>ICD** to burn the program into the chip.

❑ Remove power, Use the IC extractor to remove the 12F675 chip.

❑ Use the white breadboard to wire up the circuit as shown in the following diagram:
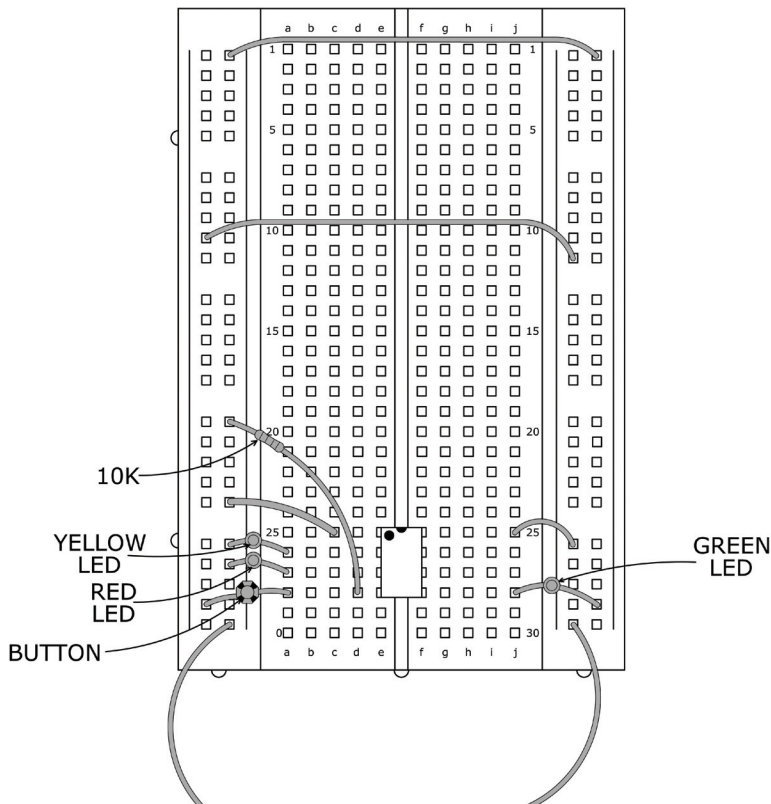
❑ The Prototyping board can be used to program a eight-pin PIC12F675 chip. Load the
   EX4A. The Prototyping board is now used only to supply 5V to the breadboard. First,
   make sure the board is wired up right, then power up the units.

❑ Verify the program works as it did in Chapter 4.

+	G
A5	A0
A4	A1
A3	A2

LED A4

LED A5

LED A2

Push
Button
A3

POT A0

POT A1

Power
9V DC

Use for
programming
8 pin chips only

RS-232
A4, A5

ICD
Connector

Debug using
ICD and
14 pin chip

Debug using
ICD and
8 pin socket
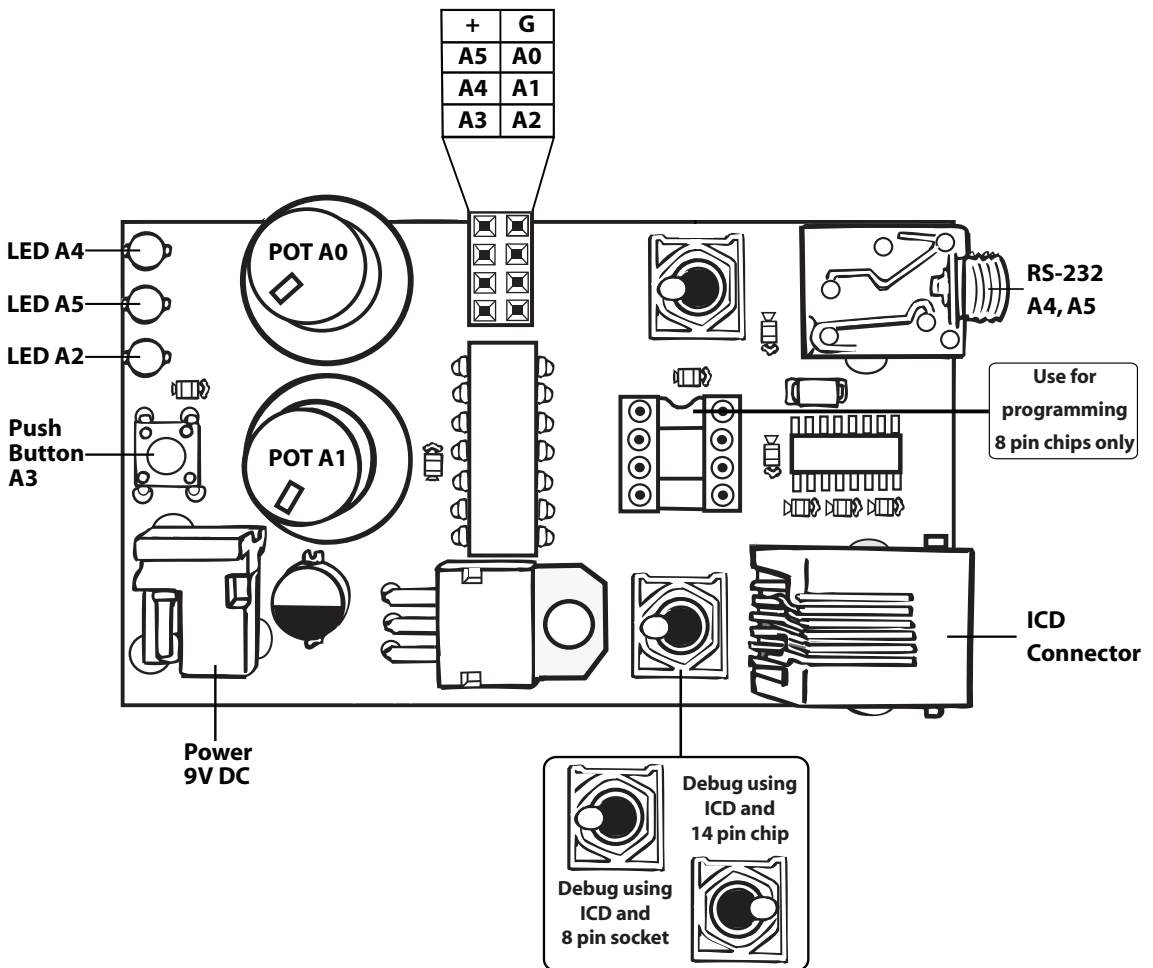
❑ The thermistor included in this kit changes resistance with temperature. The following program will read the temperature (analog voltage) and light an LED depending on whether the temperature is going up, down or unchanged. Enter the program and compile.

```
#include <12f675.H>
    #device ADC=10
#fuses intrc_io,nomclr,nowdt,noprotect
#use delay(clock=4000000)


    #define RED_LED   PIN_A4
    #define YELLOW_LED  PIN_A5
    #define GREEN_LED  PIN_A2

    void main() {
        long reading,last_reading;

        setup_adc_ports( AN0_ANALOG );
        setup_adc( ADC_CLOCK_INTERNAL );
        set_adc_channel( 0 );
        delay_ms(1000);
                    last_reading = read_adc();

        while(TRUE) {
            port_a_pullups(8);
                output_high(GREEN_LED);
            output_high(YELLOW_LED);
            output_high(RED_LED);
            reading = read_adc();
            if(reading<last_reading)
                output_low(GREEN_LED);
            else if (reading>last_reading)
                output_low(RED_LED);
            else
                output_low(YELLOW_LED);
                last_reading = reading;
            delay_ms(2000);
        }
    }
```
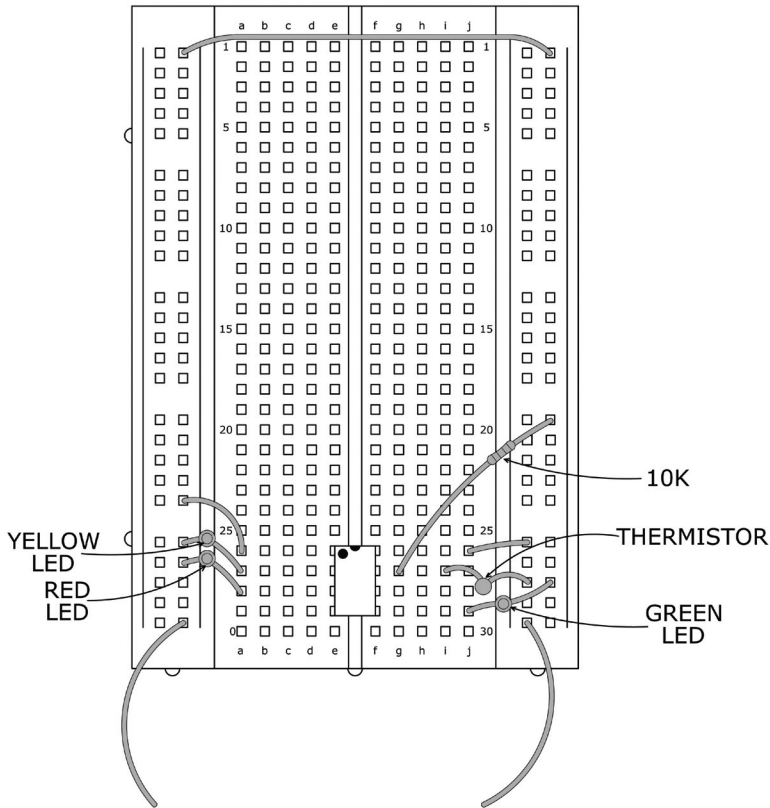
❑ Remove power, insert the 12F675 8 pin DIP in the protoboard (use the IC extractor whenever removing the chip).

❑ Use **Tools>ICD** to burn the program into the chip.

❑ Use the white breadboard to wire up the circuit as shown in the following diagrams:

❑ Make sure the board is wired up correctly, then power up the unit.

❑ Verify the program works by first squeezing the thermistor with your fingers to heat it up, then allowing it to cool.

Other Development Tools

EMULATORS

The ICD used in this booklet uses two I/O pins on the chip to communicate with a small debug program in the chip. This is a basic debug tool that takes up some of the chip's resources (I/O pins and memory). An emulator replaces the chip with a special connector that connects to a unit that emulates the chip. The debugging works in a simulator manner except that the chip has all of its normal resources, the debugger runs faster and there are more debug features. For example an emulator typically will allow any number of breakpoints. Some of the emulators can break on an external event like some signal on the target board changing. Some emulators can break on an external event like some that were executed before a breakpoint was reached. Emulators cost between $500 and $3000 depending on the chips they cover and the features.

DEVICE PROGRAMMERS

The ICD can be used to program FLASH chips as was done in these exercises. A stand alone device programmer may be used to program all the chips. These programmers will use the .HEX file output from the compiler to do the programming. Many standard EEPROM programmers do know how to program the Microchip parts. There are a large number of Microchip only device programmers in the $100-$200 price range. Note that some chips can be programmed once (OTP) and some parts need to be erased under a UV light before they can be re-programmed (Windowed). CCS offers the Mach X which is a stand-alone programmer and can be used as an in-circuit debugger.

PROTOTYPING BOARDS

There are a large number of Prototyping boards available from a number of sources. Some have an ICD interface and others simply have a socket for a chip that is externally programmed. Some boards have some advanced functionality on the board to help design complex software. For example, CCS has a Prototyping board with a full 56K modem on board and a TCP/IP stack chip ready to run internet applications such as an e-mail sending program or a mini web server. Another Prototyping board from CCS has a USB interface chip, making it easy to start developing USB application programs.

SIMULATORS

A simulator is a program that runs on the PC and pretends to be a microcontroller chip. A simulator offers all the normal debug capability such as single stepping and looking at variables, however there is no interaction with real hardware. This works well if you want to test a math function but not so good if you want to test an interface to another chip. With the availability of low cost tools, such as the ICD in this kit, there is less interest in simulators. Microchip offers a free simulator that can be downloaded from their web site. Some other vendors offer simulators as a part of their development packages.

CCS Programmer Control Software

The CCSLOAD software will work for all the CCS device programmers and replaces the older ICD.EXE and MACHX.EXE software. The CCSLOAD software is stand-alone and does not require any other software on the PC. CCSLOAD supports ICD-Sxx, ICD-Uxx, Mach X, Load-n-Go, and PRIME8.

Powerful Command Line Options in Windows and Linux
- Specify operational settings at the execution level
- Set-up software to perform, tasks like save, set target Vdd
- Preset with operational or control settings for user

Easy to use Production Interface
- Simply point, click and program
- Additions to HEX file organization include associating comments or a graphic image to a file to better ensure proper file selection for programming
- Hands-Free mode auto programs each time a new target is connected to the programmer
- PC audio cues indicate success and fail

Extensive Diagnostics
- Each target pin connection can be individually tested
- Programming and debugging is tested with known good programs
- Various PC driver tests to identify specific driver installation problems

Enhanced Security Options
- Erase chips that failed programming
- Verify protected code cannot be read after programming
- File wide CRC checking

Automatic Serial Numbering Options
- Program memory or Data EEPROM
- Incremented, from a file list or by user prompt
- Binary, ASCII string or UNICODE string

CCS IDE owners can use the CCSLOAD program with:
- MPLAB®ICD 2/ICD 3
- MPLAB®REAL ICE™
- **All CCS programmers and debuggers**

How to Get Started:
Step 1: *Connect Programmer to PC and target board. Software will auto-detect the programmer and device.*
Step 2: *Select Hex File for target board.*
Step 3: *Select Test Target. Status bar will show current progress of the operation.*
Step 4: *Click "Write to Chip" to program the device.*

Use the Diagnostics tab for troubleshooting or the ccsload.chm help file for additional assistance.

+	G
A5	A0
A4	A1
A3	A2

A4, A5 connected to LEDs

Debug using ICD and 8 pin socket

LED A4

LED A5

LED A2

POT A0

POT A1

Push Button A3

Power 9V DC

RS-232 A4, A5

Use for programming 8 pin chips only

ICD Connector

Debug using ICD and 14 pin chip

Debug using ICD and 8 pin socket