Embedded Systems Module. 6EJ505

C tutorial 2

rev. 24.9.16 tjw

Main Learning Points

C Programming	Using MPLABX, and the 18F2420
Accessing and manipulating single bits	Setting configuration bits
Writing and calling simple functions	Simulating microcontroller inputs
Using library delay functions	Using simulation breakpoints
More on program loops and branches	Using the simulator Stopwatch

1. Introduction

All embedded programmes are written for a “target” piece of hardware, that’s the basis of “embedded C” programming. The target hardware used here is the core Derbot design, as seen in Figure 1. The programme must precisely reflect how the microcontroller is connected, and what the hardware is meant to do.

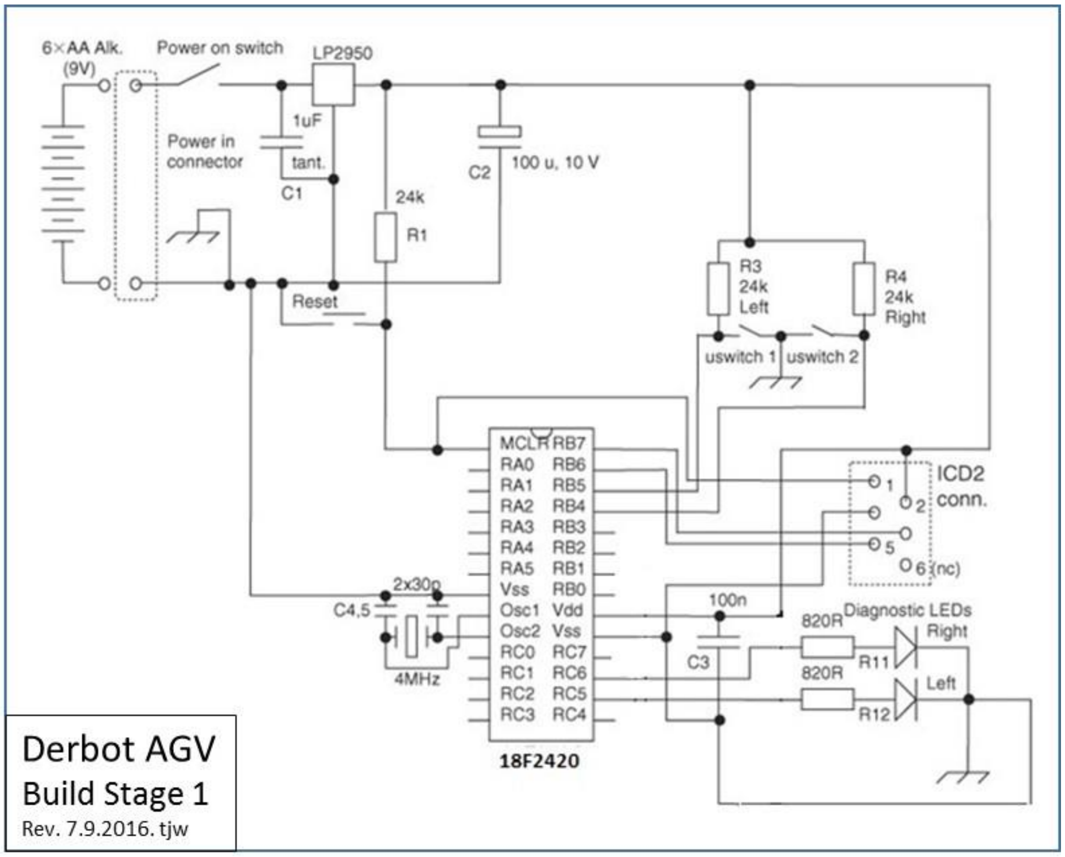


Figure 1: Derbot “Build Stage” 1

2. Controlling and branching on bit values

Fundamental to embedded systems programs is the ability to read and set single bits. Program Example 1, which moves the state of the microswitches on the Derbot to the LEDs, demonstrates how this is done in C. The program can only be fully understood by consulting the circuit diagram above.

The program further shows how to write and call simple functions, in addition to **main( )**. Here, the program contains two user-defined functions, **initialise( )** for initialisation and **diagnostic( )** for diagnostics. Both can be seen in the program listing. In implementing the diagnostic flashing of the LEDs, the program also makes use of an XC8 library delay function.

```
/*******************************************************************
Sw_to_led
Runs on Derbot. Moves state of front microswitches to leds
Revised for MPLABX                                 TJW rev. 08.09.16
*******************************************************************
Clock is 4MHz                                                     */

#include <xc.h>   //header file

//these config bit settings are useful for many Derbot programs
//Defaults settings are assumed for other config bits.
#pragma config OSC = HS               //HS oscillator
#pragma config PWRT = ON, BOREN = OFF //powerup timer on,
                                          //brownout detect off
#pragma config WDT = OFF              //watchdog timer off
#pragma config LVP = OFF              //low voltage programming off
#pragma config PBADEN = OFF           // Port B all digital i/o

#define _XTAL_FREQ 4000000  //specify clock frequency, for delay function.

//Function Prototypes (Library prototypes are in Header files)
void initialise (void);
void diagnostic (void);

void main (void){
initialise();       //call initialise function
diagnostic();       //call diagnostic function
//move microswitch states to diag leds
while (1){          //set up an endless loop
    if (PORTBbits.RB4 == 0)
    PORTCbits.RC6 = 0;
    else PORTCbits.RC6 = 1;
    if (PORTBbits.RB5 == 0)
    PORTCbits.RC5 = 0;
    else PORTCbits.RC5 = 1;
    }               //end of while(1)loop
}                   //end of main

//Initialises SFRs, and sets initial outputs.
//Assumes hardware is "Build Stage 1". All unused port bits set to output.
void initialise (void){
    TRISA = 0b00000000; //All bits output (none used in this program)
    TRISB = 0b00110000; //Only bits 5 and 4 (microswitches) are input
    TRISC = 0b00000000; //All bits output (none used in this program)
    PORTA = 0; //Switch all outputs off
    PORTB = 0;
    PORTC = 0;
}

//Diagnostic: switches leds on for 1s (Tcy = 1us)
void diagnostic (void){
    PORTCbits.RC6 = 1;
    PORTCbits.RC5 = 0;
    __delay_ms(100);          //library delay function
    PORTCbits.RC6 = 0;
    PORTCbits.RC5 = 1;
    __delay_ms(100);
}
```

Program Example 1 Derbot – moving microswitch states to LEDs

2.1 Controlling individual bits

The bits of each port are defined in the microcontroller header file. For the purposes of this program, it is enough to recognise that a port bit can be specified by the format **PORT*x*bits.R*xy***, where ***x*** indicates the port and ***y*** the bit in that port. For example, in the diagnostic function we see bits 5 and 6 of Port C being set to Logic 1 in the lines:

```
PORTCbits.RC6 = 1;
PORTCbits.RC5 = 1;
```

With this simple step we now have the ability to set or clear individual bits in a memory location or register, as long as they have been previously declared. As all microcontroller SFRs and their bits are declared in the `p18F2420.h` header file, this represents a great step forward.

2.2 The **if** and **if–else** conditional branch structures

The action in this program is built around a conditional **if**–**else** branching structure. This allows a program to contain a choice between two separate paths of action. An example of the structure appears in the **main** function, as quoted here:

```
if (PORTBbits.RB4 == 0)
   PORTCbits.RC6 = 0;
else PORTCbits.RC6 = 1;
```

This can be interpreted as: *if bit 4 of Port B is at Logic 0, then set bit 6 of Port C to 0; otherwise (else) set it to 1*. A block of code, rather than just a single line, can also be associated with either the **if** and/or the **else**, in which case it must be enclosed in curly brackets. For example:

```
if (PORTBbits.RB4 == 0){
  PORTCbits.RC6 = 0;
  PORTCbits.RC0 = 1;
  }
else PORTCbits.RC6 = 1;
```

This would cause two Port C bits to change, if Port B bit 4 was found to be zero.

It is also possible to use the **if** structure on its own. In this case there is no alternative action if the condition tested is not true. An example is:

```
if (PORTBbits.RB4 == 0)
  PORTCbits.RC6 = 0;
if (PORTBbits.RB5 == 0)
```

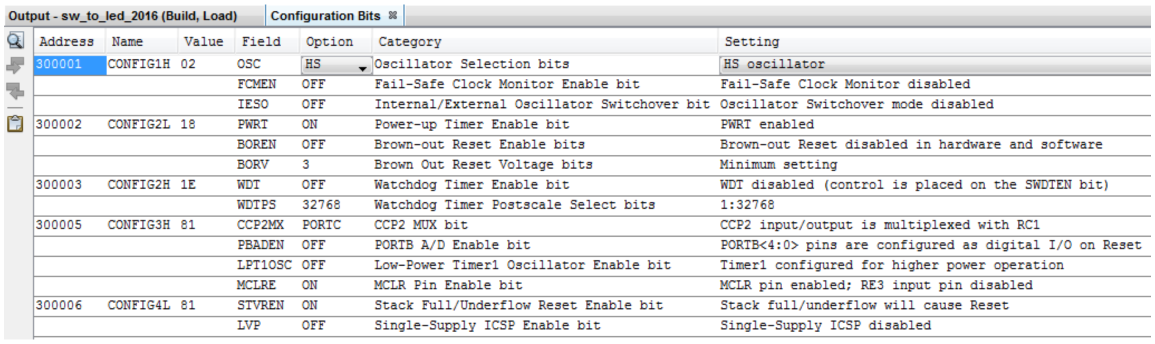
```
        PORTCbits.RC5 = 0;
```

In this case, Port C bit 6 is set to 0 if Port B bit 4 is at 0, but no action is taken if Port B bit 4 is at Logic 1. The same can be seen to happen in the lines which follow, with bit 5 of each port.

Notice in these examples how the assignment operator ‘=’ and the equal to operator ‘= =’ are used. As we have seen, the first is used to assign a value to a variable. The second is used, within the **if** construct, to test whether a variable is equal to a particular value.

3. Setting the configuration bits

Configuration bits control certain aspects of the microcontroller’s behavior and setup, and are downloaded at the same time as the program. If ignored, default settings take effect. Example settings for some configuration bits are indicated in the program listing, using the **#Pragma** construct. You can check all of these by following **Window > PIC Memory Views > Configuration Bits,** with example view seen in Figure 3**.** Bits not set in a program are left at their default value. It is essential that you understand what the configuration bits are doing to the microcontroller. It just takes one to be wrong, for major malfunction to occur.



Address	Name	Value	Field	Option	Category	Setting
300001	CONFIG1H	02	OSC	HS	Oscillator Selection bits	HS oscillator
			FCMEN	OFF	Fail-Safe Clock Monitor Enable bit	Fail-Safe Clock Monitor disabled
			IESO	OFF	Internal/External Oscillator Switchover bit	Oscillator Switchover mode disabled
300002	CONFIG2L	18	PWRT	ON	Power-up Timer Enable bit	PWRT enabled
			BOREN	OFF	Brown-out Reset Enable bits	Brown-out Reset disabled in hardware and software
			BORV	3	Brown Out Reset Voltage bits	Minimum setting
300003	CONFIG2H	1E	WDT	OFF	Watchdog Timer Enable bit	WDT disabled (control is placed on the SWDTEN bit)
			WDTPS	32768	Watchdog Timer Postscale Select bits	1:32768
300005	CONFIG3H	81	CCP2MX	PORTC	CCP2 MUX bit	CCP2 input/output is multiplexed with RC1
			PBADEN	OFF	PORTB A/D Enable bit	PORTB<4:0> pins are configured as digital I/O on Reset
			LPT1OSC	OFF	Low-Power Timer1 Oscillator Enable bit	Timer1 configured for higher power operation
			MCLRE	ON	MCLR Pin Enable bit	MCLR pin enabled; RE3 input pin disabled
300006	CONFIG4L	81	STVREN	ON	Stack Full/Underflow Reset Enable bit	Stack full/underflow will cause Reset
			LVP	OFF	Single-Supply ICSP Enable bit	Single-Supply ICSP disabled

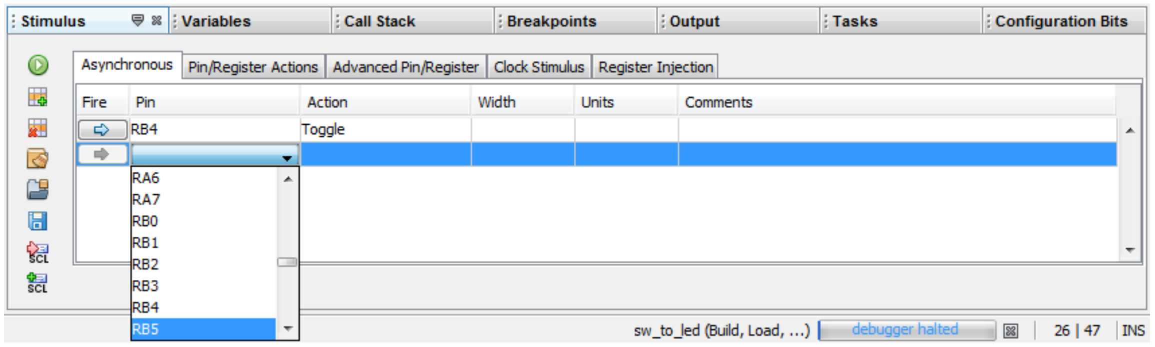
Figure 2: Configuration Bits Window

4. Simulating and running the example program

Simulate this program in the simulator. Create a new project around it, initially selecting simulator test when the project is defined. Follow the simulation steps outlined in C Tutorial 1, now displaying Port B and Port C in the Watch window.

4.1 Simulating External Inputs

We have two external inputs to the microcontroller, the microswitches connected to bits 4 and 5 of Port B. Therefore set up some external simulated inputs by selecting **Window > Simulator > Stimulus.** Simulate inputs for the microswitch inputs, RB5 and RB4, with **Toggle** as the action, as shown in Figure 2.



Fire	Pin	Action	Width	Units	Comments
⇨	RB4	Toggle			
⇨					

Figure 3: Setting up simulated digital inputs

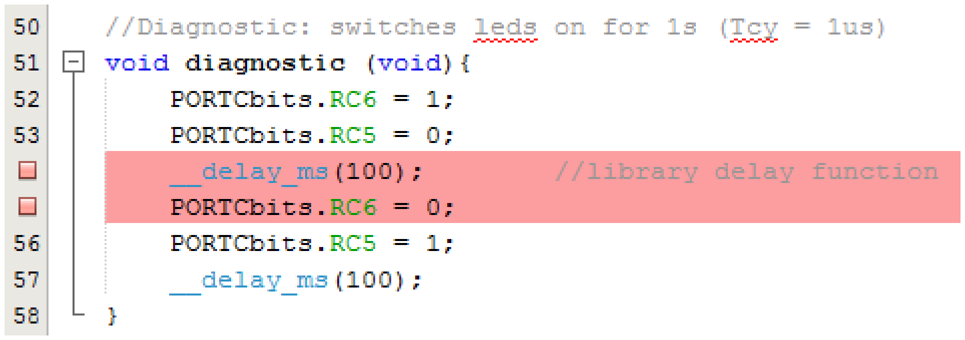
4.2 Using Breakpoints

For a simple program, we can accept single-stepping through it to see how it works. However for more complex programs, or for those where we want to explore timing, this is not acceptable. *Breakpoints* allow you to run the program up to a specific place, and then stop.

By clicking on each line in turn, place breakpoints in the **diagnostic( )** function, as shown in Figure 4a); little red boxes replace the line numbers. Use **File** > **Project Properties** > **Simulator**, to set the Instruction Frequency to 1 MHz, as shown in Figure 4b). This corresponds with the 4MHz clock frequency that is used in the Derbot. Finally select **Window > Debugging** > **Stopwatch** to display the Stopwatch window, as shown in Figure 4c). This will allow you to measure exactly how long a program section takes to execute, another essential in embedded systems.

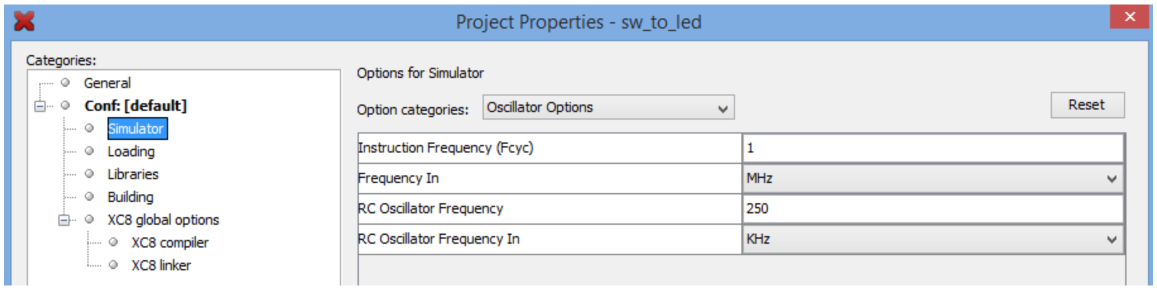
Now reset and run the program. You will see that it stops at the first breakpoint, i.e. the first in Figure 4a). At this point zero the Stopwatch and then run to the next breakpoint. This is just the line below. To get there, however, the program has to execute the **__delay_ms( )** function. The Stopwatch should now be similar Figure 4c). Almost exactly 100 ms of simulated time has elapsed in execution of the function. This is confirmation of the accuracy of this function.

Now set one further breakpoint in the line below **while (1).** Run the program to here and then single-step through the loop. Using the stimulus controller, change the values of the microswitch inputs (Port B bits 4 and 5) by clicking on the appropriate arrow buttons in the "Fire" column of the Stimulus window. Observe with care how the program loop responds.



```
//Diagnostic: switches leds on for 1s (Tcy = 1us)
void diagnostic (void){
    PORTCbits.RC6 = 1;
    PORTCbits.RC5 = 0;
    __delay_ms(100);          //library delay function
    PORTCbits.RC6 = 0;
    PORTCbits.RC5 = 1;
    __delay_ms(100);
}
```

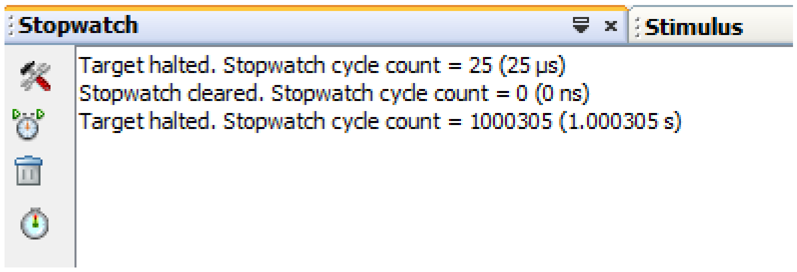
a) Suggested breakpoints in diagnostic function.



Option categories: Oscillator Options

Instruction Frequency (Fcyc)	1
Frequency In	MHz
RC Oscillator Frequency	250
RC Oscillator Frequency In	KHz

(b) Setting the instruction frequency



```
Target halted. Stopwatch cycle count = 25 (25 µs)
Stopwatch cleared. Stopwatch cycle count = 0 (0 ns)
Target halted. Stopwatch cycle count = 1000305 (1.000305 s)
```

(c) Stopwatch, after completion of delay function

Figure 4. Simulation settings

5. Using the for keyword

The **for** keyword provides another means of ‘packaging’ conditions for a loop. It has the general format:

```
      for (initialisation; condition; modification)
      statement, or statements in braces
```

The three expressions within the **for** brackets, called here **initialisation**, **condition** and **modification**, are all defined by the programmer. For example, the first loop in the Fibonacci Program (Tutorial 1) could be rewritten as:

```
for (counter = 0;counter<12;counter = counter + 1)
{fibtemp = fib1 + fib2;
//now shuffle numbers held, discarding the oldest
             fib0 = fib1;  //first move middle number, to overwrite oldest
             fib1 = fib2;
             fib2 = fibtemp;
      }
```

In the first expression, **counter** is initialised to 0. This occurs only once, when the loop is entered. The condition tested is whether **counter** is less than 12, and the modification caused is an increment to the value of **counter**. This does not occur on the first loop iteration. When the program runs, the loop is repeatedly executed, with **counter** being incremented each time. When it is incremented to 12, this is immediately detected by the condition expression and no further loop iterations occur.

Any of the three expressions associated with **for** can be omitted. If the condition is left out, then there is no test and the loop is continuous. Initialisation and modifications can still, however, apply. A simple way of creating an endless loop is by entering no expressions at all, giving:

```
for(;;)
{...
```

This is a direct alternative to:

```
while(1)
{...
```

6. Further Exercises

1. Try varying the delay value in the `__delay_ms(100);` lines, checking the new value chosen with the stopwatch. What is the maximum value possible? Why is this?

2. Try also `__delay_us()` and `__delay()`, and verify with the stopwatch.

3. Modify the Fibonacci program to use **for( )** instead of **while( ),** and simulate.

(End of Tutorial)