Sierra College MECH-01
Jim Weir 530.272.2203
jweir@sierracollege.edu
www.rstengineering.com/sierra

Theory 10

Basic BASIC Programming for a PIC16F688

1. PIC 16F688 General Discussion

A. The PIC16F688 ("688") is a 14 pin microcontroller that comes in either a standard 14 pin dual inline through hole package (14-DIP) or a surface mount very small QFN 16 pin package. The 14-DIP pinout is available in the data sheet (see "C" below).

B. It operates from a standard +5 volt supply. There are 12 input-output pins of which 12 can be configured as high impedance input pins and 11 of which can be configured as low impedance output pins each of which can supply up to 25 mA of current at +5 volts.

C. It has a very comprehensive data sheet set of 202 pages detailing each and every function of which the 688 is capable. The full data sheet can be downloaded from here: http://ww1.microchip.com/downloads/en/devicedoc/41203d.pdf

D. The PIC may be programmed using a Microchip PICKIT programmer in conjunction with the Microchip IPE (Integrated Programming Environment), which is a subset of the IDE (Integrated Development Environment) programs which are free from Microchip. The IDE-IPE may be downloaded from here: https://www.microchip.com/mplab/mplab-x-ide Documentation for the PICKIT-4 programmer can be downloaded from here: http://ww1.microchip.com/downloads/en/DeviceDoc/50002721B.pdf

E. The IPE will require a hex code file to program the 688. The Microcode Engineering Lab PICBASIC PRO is a series of programs that will allow you to write a BASIC language code program in a program called Microcode Studio and the PicBasic Pro will then compile that BASIC into a hex file that IPE can the program (using the PICKIT-4) into the 688. A free Student copy (lifetime) of PICBASIC PRO (which includes Microcode Studio) can be downloaded from here: https://store.melabs.com/merchant.mvc?Session_ID=3a460a22a4106ad2c082827491e69096& and then at the top of the page search for "Student".

2. First Program

1. Here is a copy of the first program we are going to write. It has a very simple function. Take one of the 11 outputs and light a small Light Emitting Diode (LED) when power is applied to the 688.

```
' Name        : LEDjwC.4.bas
' Compiler    : PICBASIC PRO Compiler 3.1
' Assembler   : MPLAB X IPE v5.15
' Target PIC  : 16F688
' Hardware    : Lab Protoboard
' Oscillator  : Internal 4 MHz.
' Keywords    : LED
' Description : PICBASIC PRO program to light an LED connected to PORTC.4
'                   PORTC.4 is pin 6 of the 16F688.

 #CONFIG
      __config _INTRC_OSC_NOCLKOUT & _WDT_OFF & _MCLRE_OFF & _CP_OFF
 #ENDCONFIG


TRISA = %00001000
TRISC = %00000000

L1    var    PORTC.4          ' Alias PORTC.4 to L1

mainloop:
      High L1            ' Make PORTC.4 go to +5 volts to supply an LED with current through a resistor

      Goto mainloop    ' Go back to mainloop and light the LED forever

      End
```

Seems like a lot of trouble to go through when we could simply connect the LED to the + 5 volt supply through a current limiting resistor. And yes, this program is so simple that it seems to be a waste of time to simply light the LED. If that was all we were going to do with this circuit, it would be foolish to go through all this. However, this is only the first of some rather complex procedures that we will be building on.

3. Let's break the programming down into blocks.

A. The HEADER.

```
' Name        : LEDjwC.4.bas
' Compiler    : PICBASIC PRO Compiler 3.1
' Assembler   : MPLAB X IPE v5.15
' Target PIC  : 16F688
' Hardware    : Lab Protoboard
' Oscillator  : Internal 4 MHz.
' Keywords    : LED
' Description : PICBASIC PRO program to light an LED connected to PORTC.4
'                PORTC.4 is pin 6 of the 16F688.
```

<u>This part of the program will never be executed.</u>

These are only the things we want to remember about the program, what it is called, how we made it, what PIC devices we used, and what it does. Note VERY WELL that the name of the program tells me that this program is about an LED, it has my (jw) intials, it has the port(s) I want to use (C.4) and it has the extension .bas (BASIC Language)

Next we have the Compiler and the Assembler (Programmer) I want to use. Then the PIC (688) I am going to use. The Hardware I'm going to construct it on, the Oscillator frequency I want to use, any Keywords that may be useful, and then a description of what the program is going to do.

Note that EACH LINE of the program begins with an apostrophe ('). And not just any apostrophe. It turns out that Microsoft WORD uses a "smart apostrophe" that has a slant to it (‘). This is unacceptable in the programming world. You have to use something like Notepad that is a straight text editor rather than a word processor. Any true text editor may be used, but as we shall see shortly, Microcode Studio (comes free with the Student edition of PICBASIC PRO) has a built-in BASIC text editor that is very convenient to use.

B. The CONFIG commands

```
#CONFIG
    __config _INTRC_OSC_NOCLKOUT & _WDT_OFF & _MCLRE_OFF & _CP_OFF
#ENDCONFIG
```

This is the first real programming that we do. These configuration details will be programmed into the 688 during the programming step of the process. They mean as follows:

We start the configuration bits with a #CONFIG;

Then a double underscore and the word __config;

Then a single underscore and the command _INTRC_OSC_NOCLKOUT which means "Use the internal Resistor-Capacitor Oscillator and do not use pins 2 and 3 of the 688 for input or output of the clock signal;

Then an ampersand, an underscore, and the command & _WDT_OFF which means turn a timer off (called the WatchDog Timer) to detect whether the 688 has stopped running for any reason;

Then an ampersand, an underscore, and the command & _MCLRE_OFF which means allow pin 3 to set the Master Clear bit low to externally let the programmer clear the 688 before writing to it.

Then an ampersand, an underscore, and the command & _CP_OFF which turns off the protection on the separate memory for writing variables to semi-permanent memory.

Then we end the configuration bits with an #ENDCONFIG.

Quite honestly, I'm lazy. I don't like having to do the work to set up the configuration variables. I will either go back to a prior program I wrote for the 688 and cut and paste or I will go to the PBP folder in the PROGRAM FILES on the hard disk, go into the DEVICE_REFERENCE folder, find the 16F688.INFO file, and cut and paste the suggested configuration lines.

C. The Port Configuration

The 688 needs to know which of the individual port lines needs to be an input, and which needs to be an output. Remember, there are 12 I/O pins on the 688 and each of them needs to be defined as either input or output.

There are a couple of ways of doing this, but by far the easiest is the TRIS command.

There are two ports on the 688, the A port and the C port. These letters are arbitrary; they mean nothing Microchip could just as easily called them the M and the Z port.

We could go through and individually call out each port as an input or an output (i.e. INPUT A.3, meaning PortA bit 3 is an input).

It is FAR easier to set them all in one line with the TRIS command:

```
TRISA = %00001000
TRISC = %00000000
```

The TRIS command is read from right to left, so for the A port , A.0 (the far right digit) is a zero, which means that A.0 (pin 13 of the 688) is an output, as are A.1 (pin 12) and A.2 (pin 11). However, port A.3 (pin 4) is a 1, so A.3 is an input. Ports A.4 and A.5 (pins 3 and 2 respectively) are outputs. Ports A.6 and A.7 don't exist, but ports are 8-bit devices and it doesn't do any harm to set a nonexistent port to input or output.

The % symbol in front of the number defines it as a "binary" number, which means that only 0 and 1 are allowed.

Note that ALL 6 of the C ports are outputs.

D. Setting the variables.

Every time you say that something is equal to something else, or if you are going to change the value of something in the program, you have to define the variable. In this case, I didn't want to have to use the long term PORTC.4 everywhere in the program I wanted it, so I gave it the short name L1. I just as easily could have called it plain L or QRT, or GOBBLEDYGOOK, but Lamp number 1 seemed to be reasonable.

```
L1    var    PORTC.4          ' Alias PORTC.4 to L1
```

This means set variable L1 to be the same as saying PORTC.4. The comment ' Alias PORTC.4 to L1 is what is called "documenting". You will notice that there is an apostrophe (') in front of the comment, which is telling the programmer-compiler to disregard this part of the command when putting the program into the memory of the 688

E. Writing the working program.

```
mainloop:
    High L1          ' Make PORTC.4 go to +5 volts to supply an LED with current through a resistor

    Goto mainloop    ' Go back to mainloop and light the LED forever
```

End

mainloop: (note the colon at the end of mainloop:) is called a LABEL. It is just a spot in the program that you can use to send the program if and when it becomes necessary. The word "mainloop" has no special meaning. I could have called it "a:" or "bibbitybobbityboop:" if I had wanted without changing anything (except the Goto statement below).

Then I tell the 688 to make L1 (which is PORTC.4, pin 6) high, or +5 volts, and the comment is that this will turn on the LED through a resistor.

Then I tell the 688 to go back to mainloop, which keeps L1 high, which loops back to mainloop again and again and again… until turn the power off.

End isn't really necessary to the operation of the program, but some compilers require an "end" command to make sure that whatever happens the microcontroller won't just start wandering around the chip aimlessly if for some unknown reason the program runs past the last line before "end".

Stop at this point, go to Application 10 and program your 688.

4. Making the light BLINK

A. There is a command called "PAUSE". The PAUSE command tells the 688 to stop executing for a number of milliseconds. For example, if we want to make the program stop for a second, we would insert a PAUSE 1000 into the program.

B. .There is also a command called "LOW". The LOW command tells a port to go to zero volts (as opposed to HIGH, which causes the port to go to +5 volts).

C. If we pause the program after it goes high, the LED will stay bright during the pause. If we then tell the program to make the LED go off, and then pause the program again, the LED will blink on and off (forever). Like so:

```
mainloop:
    High L1          ' Make PORTC.4 go to +5 volts to supply an LED with current through a resistor

    PAUSE 500        ‘ stop executing for half a second (500 milliseconds)
    LOW L1           ‘ turn the LED off
    PAUSE 500        ‘ stop executing for half a second (500 milliseconds)

    Goto mainloop    ' Go back to mainloop and blink the LED forever

    End
```

D. Go to STUDIO and insert these three lines into the program. Rename the program BLINK, or whatever name you choose to call it. Save it like you saved your LED program
.

E. Compile the new program, program it, and see if the light blinks.

Stop at this point, go to Application 10 and program your 688.

5. The IF-THEN-ENDIF Function

A. Suppose that we want to execute the program at some time after power is supplied, say by a pushbutton switch. Since port A.3 (the MCLR always-an-input port) is held high by a large value resistor internal to the 688, all we need to do is make pin 4 of the 688 low for just a few milliseconds.

.

B. The problem is that the two pieces of metal inside the switch are like two hammers hitting one another face-to-face. They are going to bounce around for a few milliseconds before they settle down. What we need to do is detect that first hit and see if that was a random act or whether in fact the switch was thrown. Here is a "debounce" program that will do that for you.

```
pushbutn:

    IF PORTA.3 = 0 THEN     ;Button is pushed
    PAUSE 50                ;debounce sensitivity
    IF PORTA.3 = 0 THEN     ;Button is still pushed, debounced
    GOTO mainloop           ;Go to the main program
    ENDIF
    ENDIF
    PAUSE 500               ;Wait 500 milliseconds for things to settle down
    GOTO pushbutn
```

C. We are going to label this section "pushbutn". I could have named it "donutshop" or any other name you wish, but pushbutn seemed to be indicative of what the function is going to do.

.

D. So, IF the button is pushed, THEN we go to the next statement.

E. We pause the execution of the program for 50 milliseconds to make the hammers in the switch stop bouncing.

F. Then we say IF the switch is still showing LOW, we go to mainloop to execute the program

G. We then ENDIF the first IF statement, then ENDIF the second statement

H. We then pause for 500 milliseconds to let things settle down.

I. We then go back to pushbutn to look to see if the pushbutton has been pushed.]

J. And around and around we go until that button gets pushed.

K. The entire program now looks like this:

```
' Name        : LEDjwC.4.bas
' Compiler    : PICBASIC PRO Compiler 3.1
' Assembler   : MPLAB X IPE v5.15
' Target PIC  : 16F688
' Hardware    : Lab Protoboard
' Oscillator  : Internal 4 MHz.
' Keywords    : LED
' Description : PICBASIC PRO program to light an LED connected to PORTC.4
'                PORTC.4 is pin 6 of the 16F688.

 #CONFIG
     __config _INTRC_OSC_NOCLKOUT & _WDT_OFF & _MCLRE_OFF & _CP_OFF
 #ENDCONFIG

TRISA = %00001000
TRISC = %00000000

L1    var    PORTC.4          ' Alias PORTC.4 to L1

pushbutn:

    IF PORTA.3 = 0 THEN       ;Button is pushed
    PAUSE 50                  ;debounce sensitivity
    IF PORTA.3 = 0 THEN       ;Button is still pushed, debounced
    GOTO mainloop             ;Go to the main program
    ENDIF
    ENDIF
    PAUSE 500                 ;Wait 500 milliseconds for things to settle down
    GOTO pushbutn

mainloop:

    High L1          ' Make PORTC.4 go to +5 volts to supply an LED with current through a resistor
    PAUSE 500        ‘ stop executing for half a second (500 milliseconds)
    LOW L1           ‘ turn the LED off
    PAUSE 500        ‘ stop executing for half a second (500 milliseconds)
```

```
Goto mainloop    ' Go back to mainloop and blink the LED forever

End
```

L. Go to STUDIO and insert these nine lines into the program. Rename the program SWITCH, or whatever name you choose to call it. Save it like you saved your LED program
.

M. Compile the new program, program the 688, and see if the light blinks when you take pin 4 to LOW (ground).

Stop at this point, go to Application 10 and program your 688.

End.