

Junebug Tutorial 2

Light a LED Part II

In part I of this tutorial we started work on a program to turn on a LED. In this tutorial we will complete the examination of that program.

DRAFT

Junebug Tutorial 2

Light a LED Part II

Declaring Functions

Definition: Code Block a section of code inside a pair of curly braces(brackets) `{ }`.

All executable code in C must be inside a function. Every function consists of the function declaration followed by a code block. Every function just like every variable must have a type. In the case of a function the type indicates what type of data is returned. A function that returns no data has the the type `void`.

We declare a function using the following syntax or form:

```
type name(parameter list)
{
    ... code
}
```

The Function Main

Up to now every thing we have done was in preparation for what we are about to do now. Write executable code.

All programs must have a **main()** function. Prior to executing the user program the PIC must do some setup. The traditional name for the setup code file is `c0.c`, the name for this routine in our case is `c018.c` or `c018i.c`, The only thing you need to know about `c018i` is that when it has finished it passes the control over to the user program which starts at **main()**.

```
void main(void)
{
    ...
}
```

The function **main()** takes no parameters and returns no data. You can not pass parameters/ data into **main()** or return a value form it. Thus the parameter list is void as is the function type.

Junebug Tutorial 2

Light a LED Part II

Initialization - Configuring PORTA for Digital IO

A big part of learning to program micro controllers is learning to read the processor datasheet. In this example I am providing needed information from the processor datasheet.

In the datasheet you will find

Note: On a Power-on Reset, RA3:RA0 are configured as analog inputs and read as '0'. RA4 is always a digital pin.

Had we neglected to read the datasheet for the processor we would not have known this. It is a common mistake is to use PORTA for digital IO when it is, in its default analog configuration.

We are going to use RA0 and RA5 (port A bit 0 and 5) as a digital outputs to drive the LED. We need to change RA0 and RA5 from analog inputs to digital I/O pins. We find out how to do so by consulting the datasheet again.

Pins RA6 and RA7 are multiplexed with the main oscillator pins...
The other PORTA pins are multiplexed with analog inputs, the analog VREF+ and VREF- inputs and the LVD input. The operation of pins RA3:RA0 as A/D converter inputs is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register 1).

We now know we need to set values in the ADCON1 register. In section 17.0 of the datasheet we find the below information.

Junebug Tutorial 2

Light a LED Part II

REGISTER 17-2: 10-BIT ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

REGISTER 17-2: ADCON1: A/D CONTROL REGISTER 1							
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'
bit 6 **PCFG6:** A/D Port Configuration bit – AN6
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 5 **PCFG5:** A/D Port Configuration bit – AN5
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 4 **PCFG4:** A/D Port Configuration bit – AN4
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 3 **PCFG3:** A/D Port Configuration bit – AN3
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 2 **PCFG2:** A/D Port Configuration bit – AN2
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 1 **PCFG1:** A/D Port Configuration bit – AN1
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'
bit 0 **PCFG0:** A/D Port Configuration bit – AN0
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input disabled and reads '0'

We now have all the information we need to setup RA0 and RA5 as a digital IO pin. Because we do not need any analog pins we make all of port A digital by setting `ADCON1 = 0xFF`.

It is important to know that the names used in the processor header file may not always be exactly the same as used in the manufactures processor data sheet. We use the datasheet to figure out what to do, but the C code is written using the names from the header file.

Junebug Tutorial 2

Light a LED Part II

Action – Turn on LED1

At this point all bits of PORTA are in the digital IO mode. To make LED1 light we need to do two additional things. First we have to setup the correct pins as outputs and then we have to make these output pins high or low as required by the LED1.

```
void main(void)
{
  ADCON1 = 0; // make RA0 digital
  TRISA = 0b10111110;
  PORTA=0b00000001; // Turn LED on
  while(1); // loop forever
}
```

The six LEDs on the Junebug are controlled by 3 IO pins. The method used to accomplish this is known as charlieplexing. We will cover charlieplexing in tutorial 4. For now it is enough to know that to turn on LED1 all port A bits must be inputs but for bit0 and bit5.

In the datasheet we find the following.

10.1 PORTA, TRISA and LATA Registers

PORTA is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

We make port A bit0 and bit5 outputs by setting the TRISA:

```
TRISA = 0b10111110; // hex 0xBE
```

Making PORTA bit0 low and PORTA bit5 high causes LED1 to turn on:

```
PORTA = 0b00000001; // hex 0x01
```

As programmers we need to be careful about what our programs do. A `while(1);` statement will prevent the program from doing anything unexpected. We will learn more about `while` in the next tutorial.

Junebug Tutorial 2

Light a LED Part II

Statements

Lets examine our program one more time. I have removed the comments and replaced them with comments that indicate what each line is.

```
// compiler directives
#pragma config OSC=INTIO2, WDT=OFF, LVP=OFF, DEBUG=ON
#include <p18f1320.h>

void main(void) // function declaration
{ // start of code block
  // statements
  ADCON1 = 0;
  TRISA = 0b10111110;
  PORTA = 0b00000001;
  while(1);
} // end of code block
```

You may have noticed that sometimes we end a line with a semicolon. The lines ending with semicolons are C statements. The ones without are not.