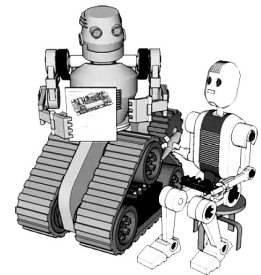


BCHS Advanced Computer Programming
Introduction to Robotics
By Daniel Johnson



P_{ulse} W_{idth} M_{odulation}

The principal behind PWM is simple. For a given signal we vary the ratio of on to off time. This signal can be used to control the brightness of an LED, the speed of a motor and other things.

If we have a LED connected to PORTA bit 0 we can turn the LED on with.

```
PORTA=0x01;
```

To turn the LED off use.

```
PORTA=0x00;
```

Signal modulation was first used in radio communication. A carrier wave frequency or amplitude is modulated to convey voice communication. For the purpose of this tutorial we will use PWM to power devices.

One might think that the code fragment below would produce a PWM signal that is on 50% and off 50% of the time.

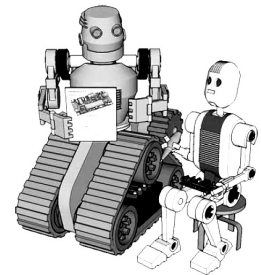
```
PWM - MPLAB IDE v8.14
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
MPLAB IDE Editor
test_lcd111_8.c LCD111_8_2321.h lcd111_8.c c018i.c LCD111_8.h timer.c test.c PWM.c* 18f2321i.lkr __init.c
// PWM1
// Daniel Johnson BCHS 2008
#pragma config OSC=INTIO2, WDT=OFF, LVP=OFF
#include <p18f1320.h>
#define byte unsigned char

void main (void)
{
    byte i;

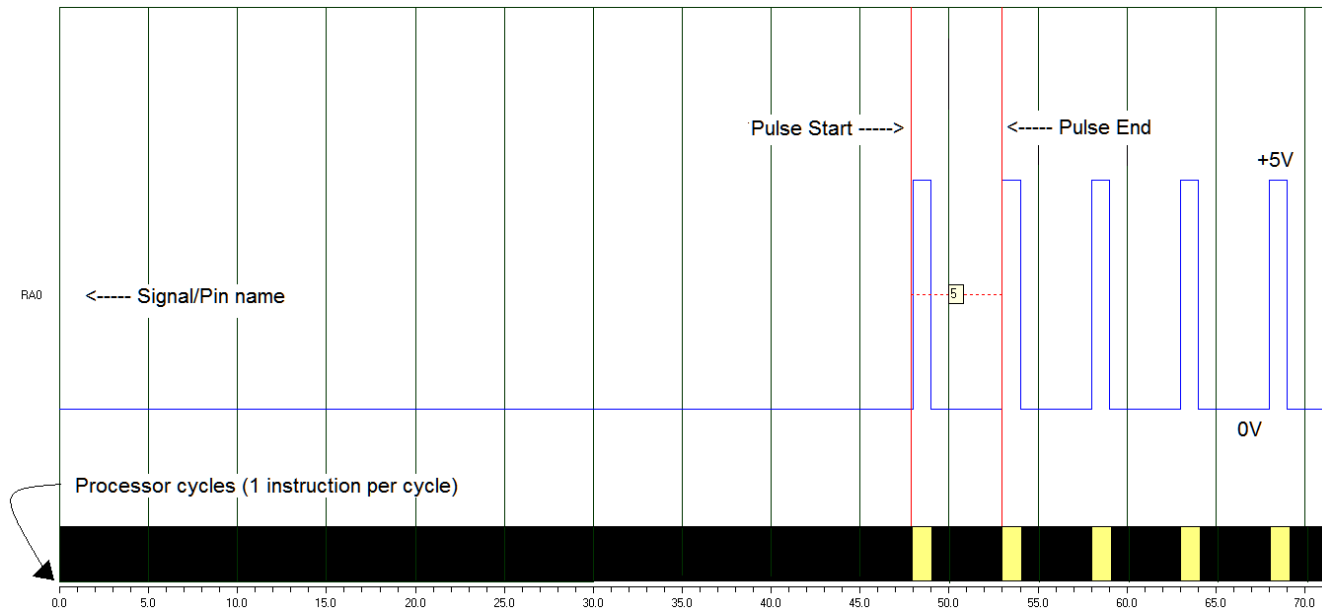
    ADCON1 = 0x7F;
    PORTA=0x00; // no refrigerator problems!
    TRISA=0x00; // bit 0 is output

    while(1)
    {
        PORTA=0x01;
        PORTA=0x00;
    }
}
```

BCHS Advanced Computer Programming Introduction to Robotics By Daniel Johnson



The resulting signal on PORTA is can be seen from the MPLAB simulator Logic Analyzer tool. The display has been augmented to help you understand it better.



The program setup in C018.c and the our code prior to the while loop execute in 46 cycles. RA0 goes high at the start of cycle 48. This tells us that cycle 47 sets RA0 high.

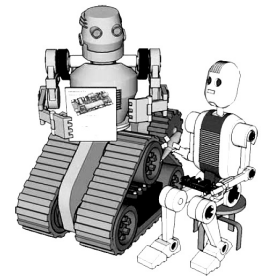
Cycle 48 sets RA0 low. The problem is that we expected cycle 49 to set it high but that did not happen, why?

There are two things going on. The first it that it takes each line of C code maps into one or more machine instructions. The other is that we forgot that the compiler will need to use one or more instructions to create the while loop.

If we were more advanced we might be tempted to look at the machine code and determine exactly what we needed to do to fix the problem. (We wanted a 50% duty cycle and ended up with 20%). There is an easier an more intuitive method.

For the reasons given two paragraphs above our off time is too long and we can not do anything about that. But we can extend the on time to equal that of the off time by inserting Nop() instructions after setting RA0 high. The question is how many Nop()'s are required? We know our on time is 1 cycle and the off time is 4. If we add 3 Nop()'s they will each be 4.

BCHS Advanced Computer Programming Introduction to Robotics By Daniel Johnson



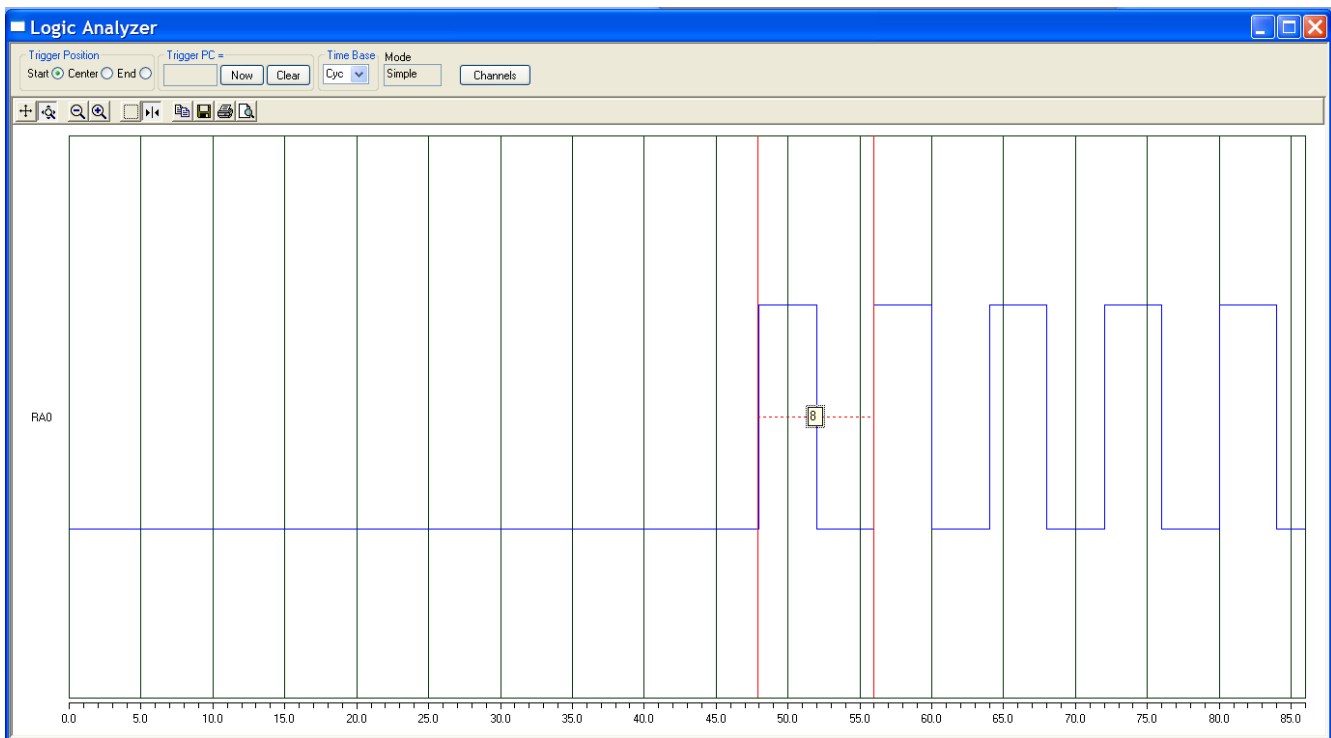
```
MPLAB IDE Editor
test_lcd111_8.c LCD111_8_2321.h lcd111_8.c c018i.c LCD111_8.h timer.c test.c PWM.c 18f2321i.lkr __init.c
#pragma config OSC=INTIO2, WDT=OFF, LVP=OFF
#include <p18f1320.h>
#define byte unsigned char

void main (void)
{
    byte i;

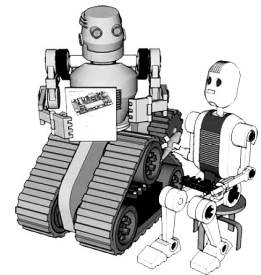
    ADCON1 = 0x7F;
    PORTA=0x00; // no refrigerator problems!
    TRISA=0x00; // bit 0 is output

    while(1)
    {
        PORTA=0x01;
        Nop(); Nop(); Nop();
        PORTA=0x00;
    }
}
```

We return to the LA (Logic Analyzer) window and see that we have achieved a 50% duty cycle.

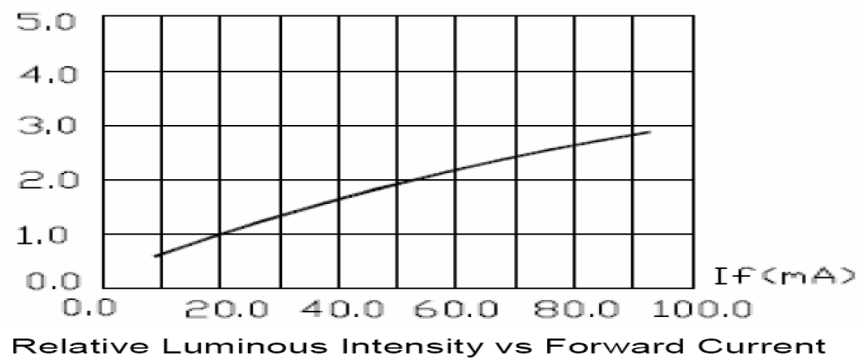


BCHS Advanced Computer Programming
Introduction to Robotics
By Daniel Johnson



It is worth noting that in the majority of cases a 50% duty cycle will not result in 50% illumination in the case of a LED or half speed in the case of a motor. Most real world devices do not generate an output in direct proportion to their input. The input/output is can be represented by a graph.

The following graph shows luminous intensity (brightness) for one specific LED. This LED starts to emit light when the current reaches 10mA and increases at a stead rate up to about 90 mA.



Each LED or motor will have its own characteristic graph. <http://www.theledlight.com/technical3.html> shows such a graph for various colored LEDs.

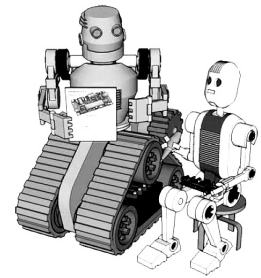
How does Duty Cycle relate to current ?

If you know what the current is for the LED when it is on all the time it is easy to determine the current for a given duty cycle.

Lets assume that we have an LED resistor pair setup to provide 20mA (iF or current Forward). Simply multiply that current time the duty cycle.

Duty Cycle	iF (current Forward)
100	20mA
75	15mA
50	10mA
25	5mA
0	0mA

BCHS Advanced Computer Programming
Introduction to Robotics
By Daniel Johnson



Advanced Study. (Advanced students will be tested on this)

A good embedded system programmer needs to understand the machine code generated by the C compiler.

This is a partial listing of our code from the MPLAB disassembly window from the above tutorial. The first numbers on the left correspond to the line number or each C source statement shown in black.

```
15:          while(1)
    00E8      D7F9      BRA 0xdc
16:          {
17:          PORTA=0x01;
    00DC      0E01      MOVLW 0x1
    00DE      6E80      MOVWF 0xf80, ACCESS
18:          Nop(); Nop(); Nop();
    00E0      0000      NOP
    00E2      0000      NOP
    00E4      0000      NOP
19:          PORTA=0x00;
    00E6      6A80      CLRWF 0xf80, ACCESS
20:          }
21:          }
```

The colored text without line numbers are generated by the compiler. The first number (blue) is the address where the instruction is stored in memory. The second (red) is the actual hex value that represents that instruction. The green text is the code an assembly language programmer might use to generate the code.

It should not be a surprise that each line of C code can require several machine instructions. Also note that machine instructions are 16 bits long. They require 2 address locations. That is why all the addresses shown in the listing are even numbers.

NOP is the most easy to identify. An assembler language programmer uses the **NOP** opcode instead of the C statement `Nop()`. The compiler generates the hex instruction **0000**. But when talking about its value in text we should use `0x0000` to indicate it is a hex value.

It is interesting to note how the `while(1)` loop is created. Notice that is a single instruction placed at the end of the loop that jumps or branches back to the start of the loop.