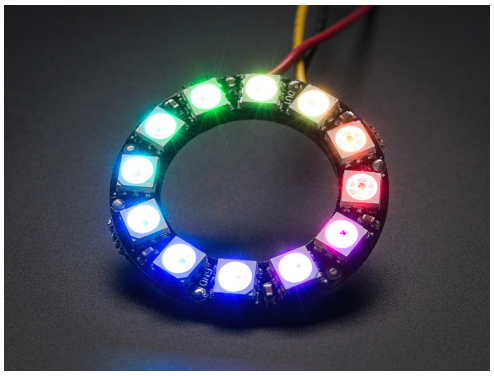
AdaFruit NeoPixel LED's



www.AdaFruit.com NeoPixel LED

There are several innovative companies I like

- www.SparkFun.com

  A bunch of students from Colorado State who cruise the web to find neat components and build interface boards to make them easier to use. Especially good spot for GPS and wireless communications
- www.AdaFruit.com

  Arduino, sensors, LEDs, such as the NeoPixel (1-wire wearable LED's for the fashion minded)
- www.ThinkGeek.com

  Useful devices like the Annoyatron (beeps at random times to annoy your neighbor)

AdaFruit NeoPixel

The NeoPixel is a GRB LED with a 1-wire interface. They are designed so that you can cascade them (as a ring as shown above, for example) and drive all the LEDs with a single data line.

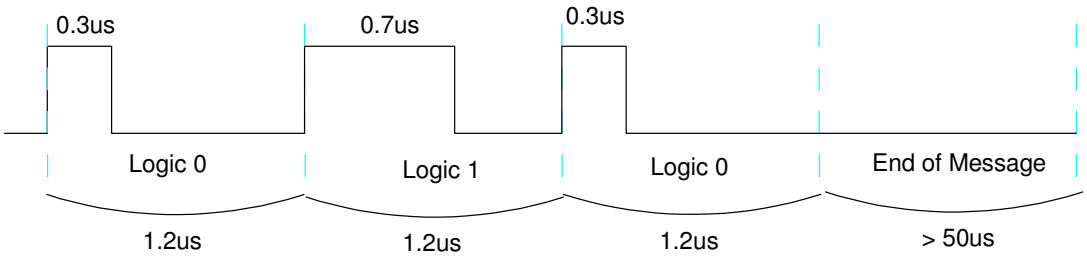
- The first 24 bits of data drive the fist NeoPixel
- The second 24 bits of data drive the second NeoPixel
- The third 24 bits of data drive the third NeoPixel
- etc.

To write to the NeoPixel, you send three bytes, most significant bit first.

Green (byte 1)								Red (byte 2)								Blue (byte 3)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Logic 1 and 0 is for each bit it determined by the length of a pulse:

- Each bit is 1.2us long (+/- 150ns)
- Logic 1 is on for 0.7us (+/- 150ns)
- Logic 0 is on for 0.3us (+/- 150us)



If communication to more than one LED,

- The first 3 bytes (GRB) is read by the first NeoPixel
- The next 3 bytes are read by the second NeoPixel
- etc.

The current to each LED is proportional to the number written, with 255 being 20mA.

The message is terminated by holding the data line low for more than 50us.

Assembler Coding - Bottom Up Programming

Note: This is one way to write programs.

- Start with the simplest (lowest) level, like output a bit. Test this routine to make sure it works.
- Once you can output 1 bit, output a byte (8 bits). Test this routine.
- Next, output 3 bytes (green / red / blue). Test this routine.
- Next, output 64 values for GRB to drive the display.

This is called 'bottom-up programming.' It is a methodical method to write programs and will get you a working design. It also saves a LOT of time.

Level 1: Write a subroutine which outputs a 1 or 0, determined by bit 7 of PIXEL

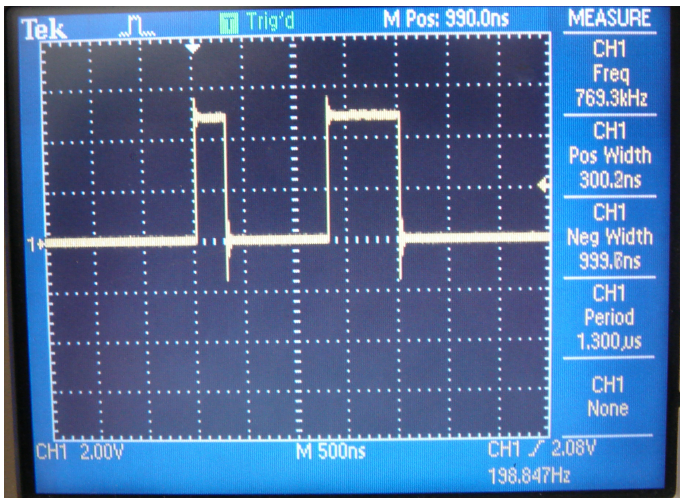
```
; Global Variables
PIXEL   EQU  xxxx      ; 0 is 0mA,  255 is 20mA

Pixel_1                                 ; clocks
        bsf     PORTD,0                 ; 0    bit set
        nop                             ; 1
        btfss   PIXEL,7                 ; 2
        bcf     PORTD,0                 ; 3    clear at 0.3us for a 0
        nop                             ; 4
        nop                             ; 5
        rlncf   PIXEL,F                 ; 6
        bcf     PORTD,0                 ; 7    clear at 0.7us for a 1
        return                          ; 8
                                        ; 9    (2 clocks for a goto)
        call    Pixel_1                 ; 10   (part of the next routine)
                                        ; 11   (2 clocks for a goto)
```

Note that you would normally end with three *nop* statements to make each bit 12 clocks long (1.2us). Since this will be a subrine, 2 clocks are used for the *return* statement with another 2 clocks used for the subsequent *call* statement.

Test: Test this code by sending a 01 signal every 100ms

```
Loop    bcf     PIXEL,7
        call    Pixel_1
        bsf     PIXEL,7
        call    Pixel_1
        movlw   100
        call    Wait
        goto    Loop
```

The oscilloscope trace is as follows:



Testing Pixel_1: '0' is high for 300ns, '1' is high for 700ns, each bit is 1200ns

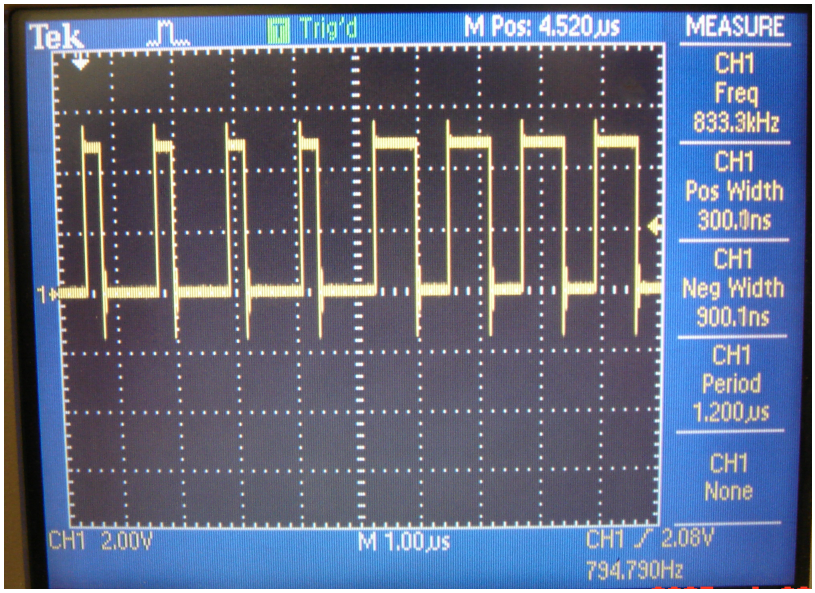
Level 2: Write a subroutine which outputs 8 bits. Pass the value to be written in the variable PIXEL

```
Pixel_8
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        call    Pixel_1
        return
```

Testing Pixel_8: To test this code, send the bit pattern 0000 1111:

```
Loop:
        movlw   0x0F
        movwf   PIXEL
        call    Pixel_8
        movlw   10
        call    Wait_ms
        goto    Loop
```

The resulting signal on RD0 is:



Testing Pixel_8: 0x0F is Sent to the NeoPixel - displayed as four 300ns pulses and four 700ns pulses

Level 3: Write a subroutine which drives the GRB lights with intensity levels set by variables GREEN, RED, and BLUE:

```
PixelGRB:
        movff   GREEN, PIXEL
        call    Pixel_8
        movff   RED, PIXEL
        call    Pixel_8
        movff   BLUE, PIXEL
        call    Pixel_8
        return
```

and just for fun, a routine which turns off a pixel (outputs 00 00 00)

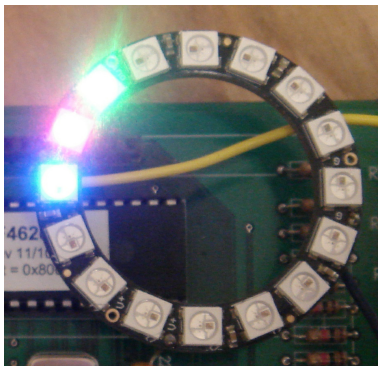
```
PixelOff:
        clrf    PIXEL
        call    Pixel_8
        clrf    PIXEL
        call    Pixel_8
        clrf    PIXEL
        call    Pixel_8
        return
```

Testing: Make the first three lights Green, Red, Blue:

```
Loop:
        movlw   250
        movwf   GREEN
        clrf    RED
        clrf    BLUE
        call    PixelRGB
```

```
        movlw   250
        movwf   RED
        clrf    BLUE
        clrf    GREEN
        call    PixelRGB

        movlw   250
        movwf   BLUE
        clrf    GREEN
        clrf    RED
        call    PixelRGB

        movlw   100
        call    Wait_ms

        goto    Loop
```

The output is as expected: the first three lights are green / red / blue



Testing PixelRGB: Green, Red, then Blue lights are displayed.

Fun with Neopixels: (Neopixel12): Display 12 different colors on a 12-neopixel ring.

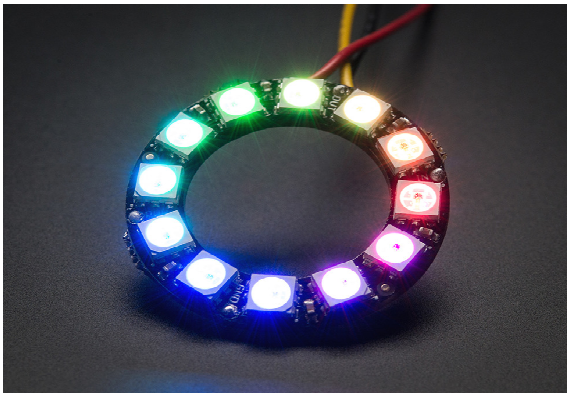
Vary the red-green-blue numbers with 12 calls:

```
        movlw 0
        movwf RED
        movlw 50
        movwf GREEN
        movlw 150
        movwf BLUE
        call  PixelRGB
```

For example, the weights

Pixel	0	1	2	3	4	5	6	7	8	9	10	11
Red	200	150	100	50	0	0	0	0	0	50	100	150
Green	0	50	100	150	200	150	100	50	0	0	0	0
Blue	0	0	0	0	0	50	100	150	200	150	100	50
color	red	orange	yellow		green		cyan		blue	purple	pink	

displays as the following:



Neopixel12: Twelve colors are displayed on the Neopixel

More Fun with Neopixels: Display a time-changing color on pixel #1 going through the spectrum.

Loop1:

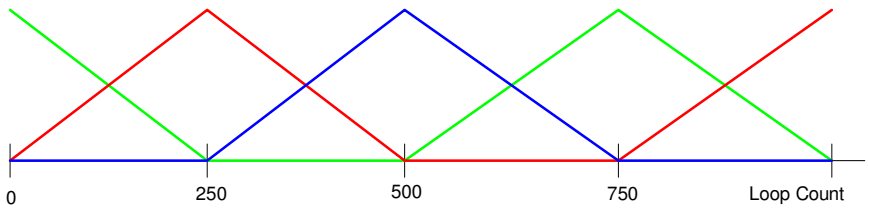
- Start with GREEN = 250, RED = 0, BLUE = 0
- Each loop, decrement GREEN and increment RED
- When you decrement GREEN to zero, skip out

Loop 2: At this point, GREEN = 0, RED = 250

- Each loop, decrement RED and increment BLUE
- When you decrement RED to zero, skip out

Loop 3: At this point, RED = 0 and BLUE = 250

- Each loop, decrement BLUE and increment GREEN
- When you decrement BLUE to zero, skip out



The main routine is then

```
      movlw 0
      movwf RED
      movlw 0;
      movwf BLUE;
      movlw 250;
      movwf GREEN;

  Loop1:
```

```
        call PixelRGB

        movlw 10
        call  Wait_ms

        incf    RED,F
        decfsz  GREEN,F
        goto    Loop1

Loop2:
        call PixelRGB

        movlw 10
        call  Wait_ms

        incf    BLUE,F
        decfsz  RED,F
        goto    Loop2

Loop3:
        call PixelRGB

        movlw 10
        call  Wait_ms

        incf    GREEN,F
        decfsz  BLUE,F
        goto    Loop3

        goto     Loop1
```