NeoPixel LED's

(www.AdaFruit.com)

ECE 376 Embedded Systems

Jake Glower - Lecture #7

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

NeoPixel LED's

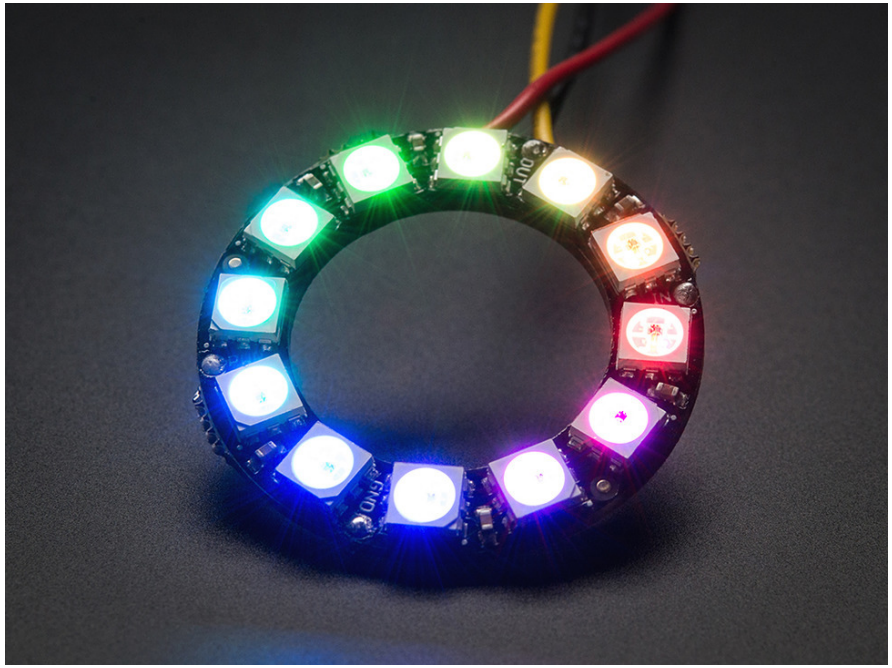
- www.AdaFruit.com
- search WS2812 LED on ebay

Bright, pretty, easy to use

- Useful for senior design
- Clothing (fashion)
- Desk lights
- Christmas lights
- etc.

RGB LED's with 1-wire interface

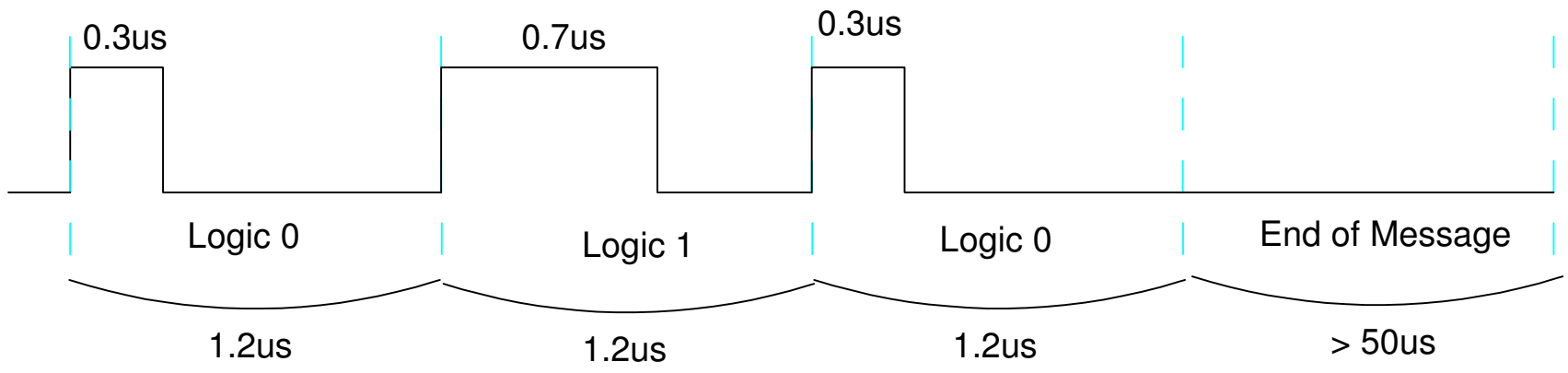
- 1st LED: 1st 24 bits of data
- 2nd LED: 2nd 24 bits of data
- 3rd LED: 3rd 24 bits of data
- etc.



Data In:

- 24-bits of data (G / R / B)
- Each bit is 1.2us long (+/- 150ns)
- Logic 1 is on for 0.7us (+/- 150ns)
- Logic 0 is on for 0.3us (+/- 150us)
- 50us pause = end of message

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



Assembler Coding - Bottom Up Programming

- 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
- It will get you a working design.
- It also saves a LOT of time.

Level 1: Send a bit

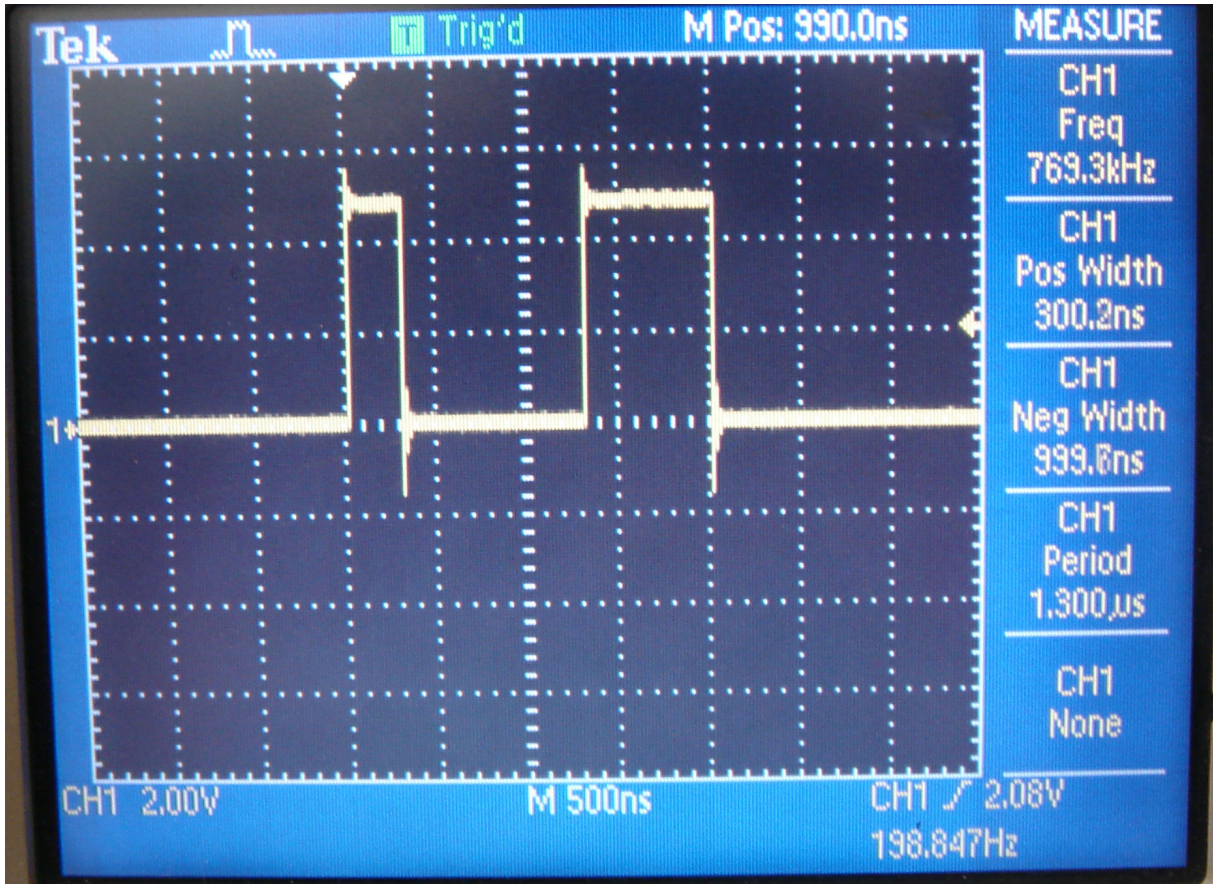
- Logic 0 or Logic 1
- Data in bit 7 of PIXEL

```
; 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)
```

Level 1: Send a Bit

- Testing

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



Level 2: Send a byte

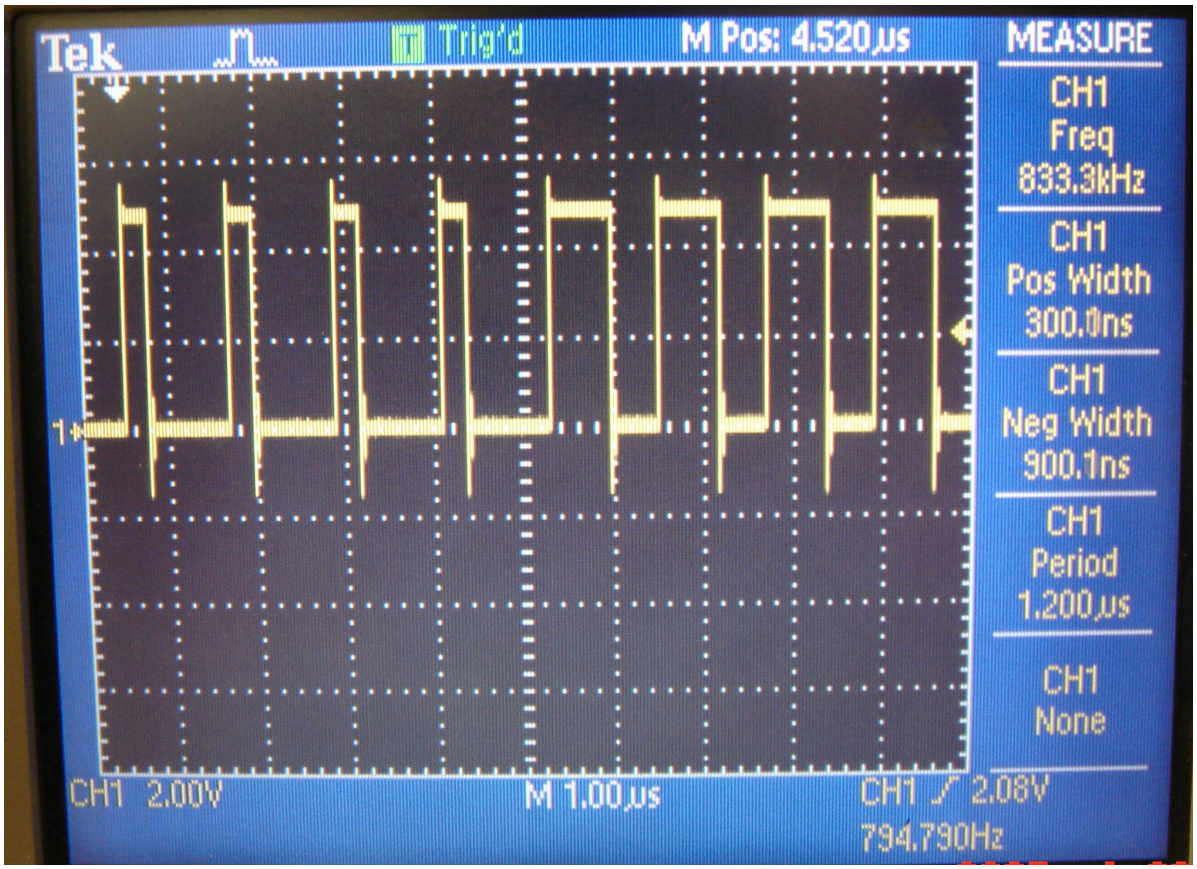
- Pass data in PIXEL

```
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
```

Level 2: Send a Byte

- Testing: Send 0000 1111:

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



Level 3: Send a GRB Pattern

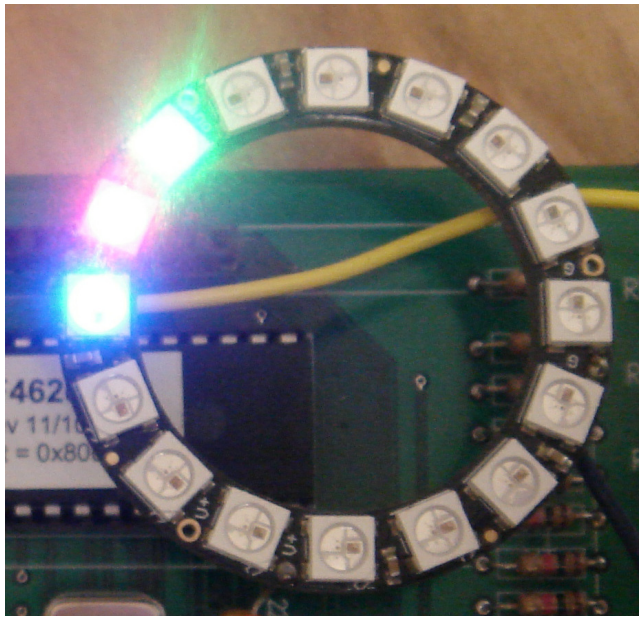
```
PixelGRB:
     movff GREEN, PIXEL
     call  Pixel_8
     movff RED, PIXEL
     call  Pixel_8
     movff BLUE, PIXEL
     call  Pixel_8
     return
```

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
```

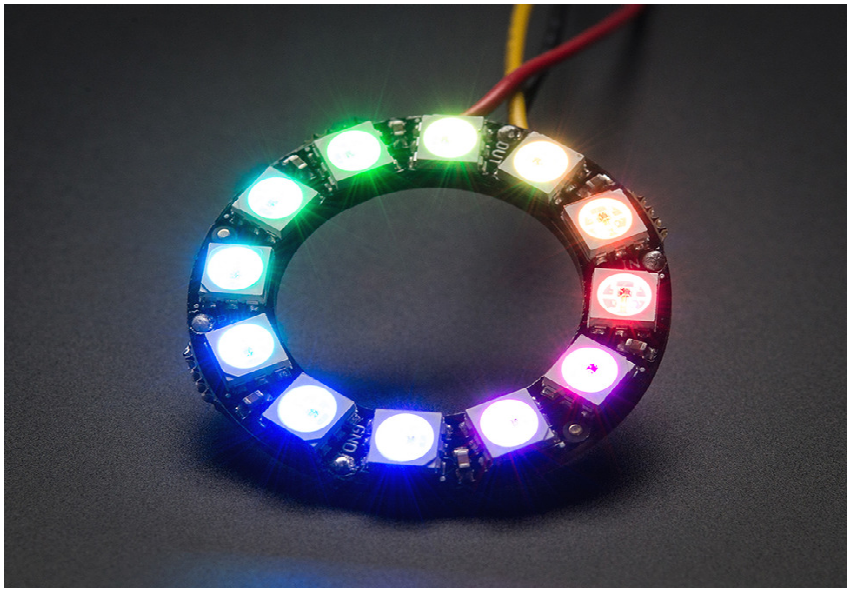
Testing: Make the first three lights Green, Red, Blue:

```
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
```



Program 1: Output a rainbow

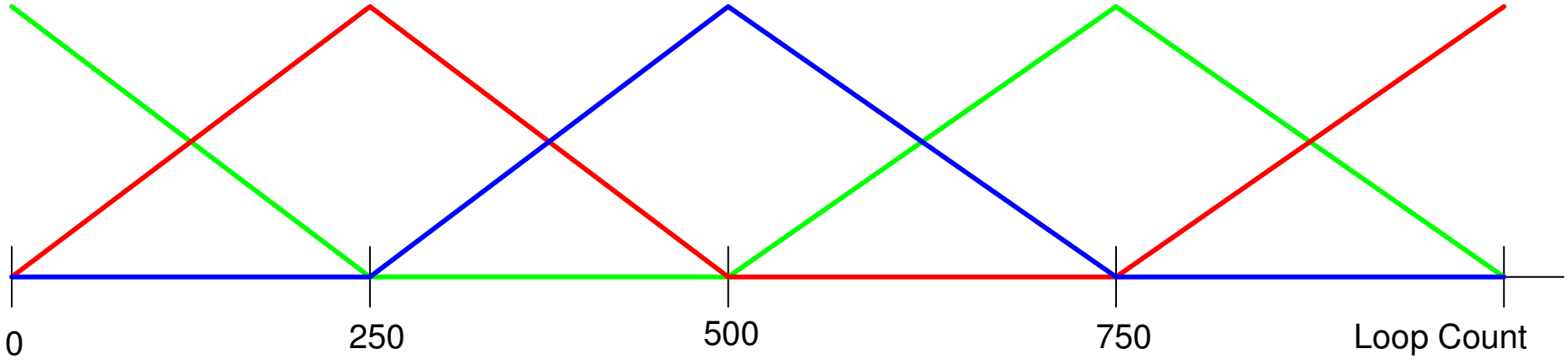
```
movlw 0
movwf RED
movlw 50
movwf GREEN
movlw 150
movwf BLUE
call  PixelRGB
```



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	

Program 2: Go through the color wheel

- Count to 750 (250 three times)
- Increase / Decrease GREEN / RED / BLUE each pass



Partial Code:

```
      movlw  250
      movwf  BLUE
      clrf   RED
      clrf   GREEN

Loop3:
      call PixelRGB

      movlw 10
      call  Wait_ms

      incf    GREEN,F
      decfsz  BLUE,F
      goto    Loop3

      goto  Loop1
```

Comments on NeoPixels

NeoPixels allow you to access a large number of RGB LEDs using a single wire

- Cascade as many NeoPixels as you want
- The 1st RGB pattern drives the 1st NeoPixel
- The 2nd RGB patter drives the 2nd NeoPixel
- etc

Timing is critical

- Each bit is 1.2us (12 clocks)
- 0.3us is logic 0 (3 clocks)
- 0.7us is logic 1 (7 clocks)
- 50us idle signifies end of message

You are almost forced to use assembler to drive NeoPixels