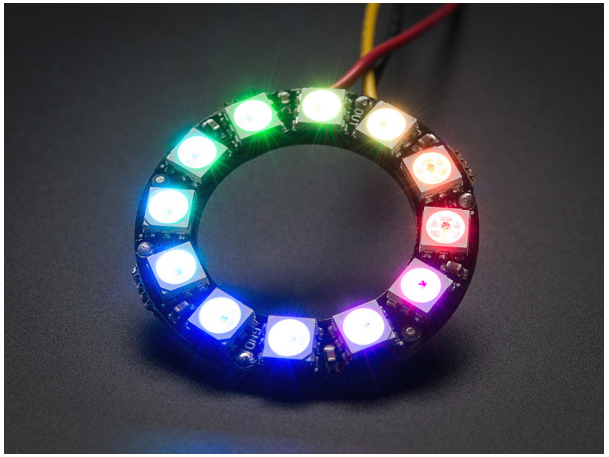
NeoPixels & In-Line Assembler

ECE 376 Embedded Systems

Jake Glower - Lecture #12

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

Assembler Code for NeoPixels



Assembler has advantages

- It lets you control the I/O pins
- It allows for precise timing

Disadvantages:

- *Really* hard to write, debug, maintain, reuse

To write to a NeoPixel, send a series of 24-bit commands:

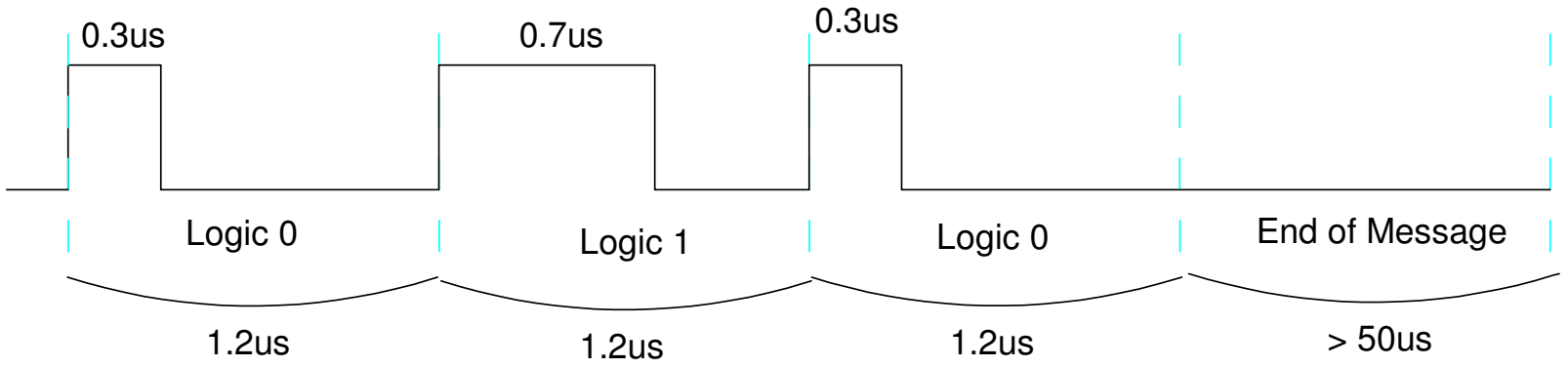
Green (byte 1)								Red (byte 2)								Blue (byte 3)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Timing is critical

- Each bit needs to be 1.2us long (12 clocks)
- Logic level 0 is a 300ns pulse (3 clocks)
- Logic level 1 is a 700ns pulse (7 clocks)
- A pause of 50us or more (500 clocks) signifies a new message

Such precise timing is

- Difficulty in C
- Easy in assembler



In-Line Assembler

Almost all C compilers offer this

- Makes it easy on the compiler: you write the assembler code for it
- Allows us to reuse the previous assembler routines

One Instruction

```
asm("    nop");
```

Multiple Instructions

```
#asm
    nop
    nop
    nop
#endasm
```

Global Variables

Intro to C: *Never never use global variables*

- Makes debugging hard
- Makes code hard to follow

Embedded Systems: *Given a choice, never use global variables.*

- Sometimes it's the best option
- *Everyone* can see global variables: C and assembler
- One way to pass date from a C program to an assembler program

Assembler

```
PIXEL equ 0x0000
```

C

```
unsigned char PIXEL @ 0x000;
```

In-Line Assembler and Bottom Up Programming

Level 1: (Assembler)

- Pixel_1.asm
- Send a bit

Level 2: (Assembler)

- Send a byte (8 bits)
- Pixel_8.asm

Level 3: (C)

- Send RED, GREEN, BLUE

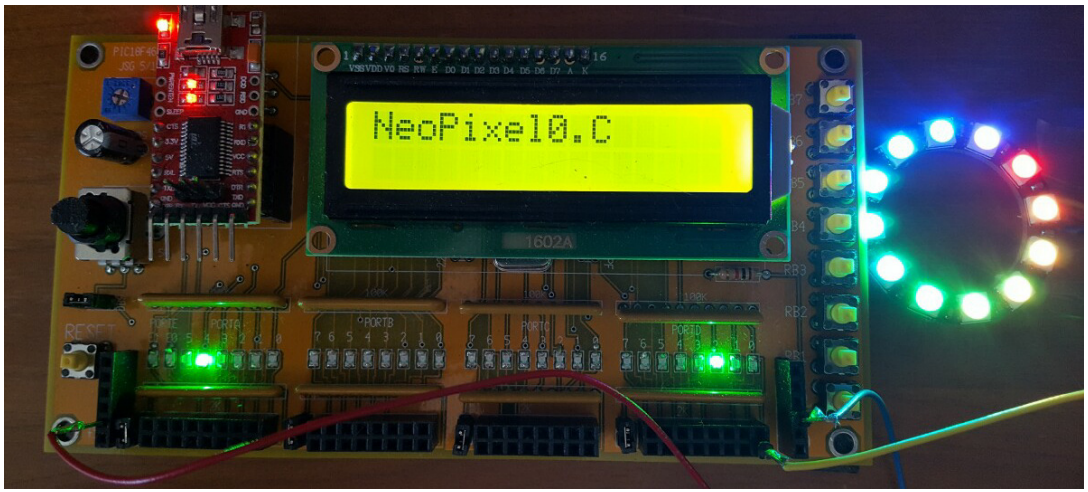
```
void NeoPixel_Display(char RED, char GREEN, char BLUE)
{
   PIXEL = GREEN;    asm(" call Pixel_8 ");
   PIXEL = RED;      asm(" call Pixel_8 ");
   PIXEL = BLUE;     asm(" call Pixel_8 ");
   asm(" return");

#asm
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
Pixel_1:
   bsf ((c:3971)),0 ; PORTD,0
   nop
   btfss ((c:0000)),7
   bcf ((c:3971)),0
   rlncf ((c:0000)),F
   nop
   nop
   bcf     ((c:3971)),0
   return
#endasm
}
```

NeoPixel0.C

Display a color wheel

```
while(1) {
    NeoPixel_Display(20,  0,  0);
    NeoPixel_Display(15,  5,  0);
    NeoPixel_Display(10, 10,  0);
    NeoPixel_Display( 5, 15,  0);
    NeoPixel_Display( 0, 20,  0);
    NeoPixel_Display( 0, 15,  5);
    NeoPixel_Display( 0, 10, 10);
    NeoPixel_Display( 0,  5, 15);
    NeoPixel_Display( 0,  0, 20);
    NeoPixel_Display( 5,  0, 15);
    NeoPixel_Display(10,  0, 10);
    NeoPixel_Display(15,  0,  5);
    Wait(100);
    }
```



NeoPixel1.C

Vary the color with buttons

- RB5: Red gets brighter (+1)
- RB4: Red gets dimmer (-1)
- RB3: Green gets brighter (+1)
- RB2: Green gets dimmer (-1)
- RB1: Blue gets brighter (+1)
- RB0: Blye gets dimmer (-1)

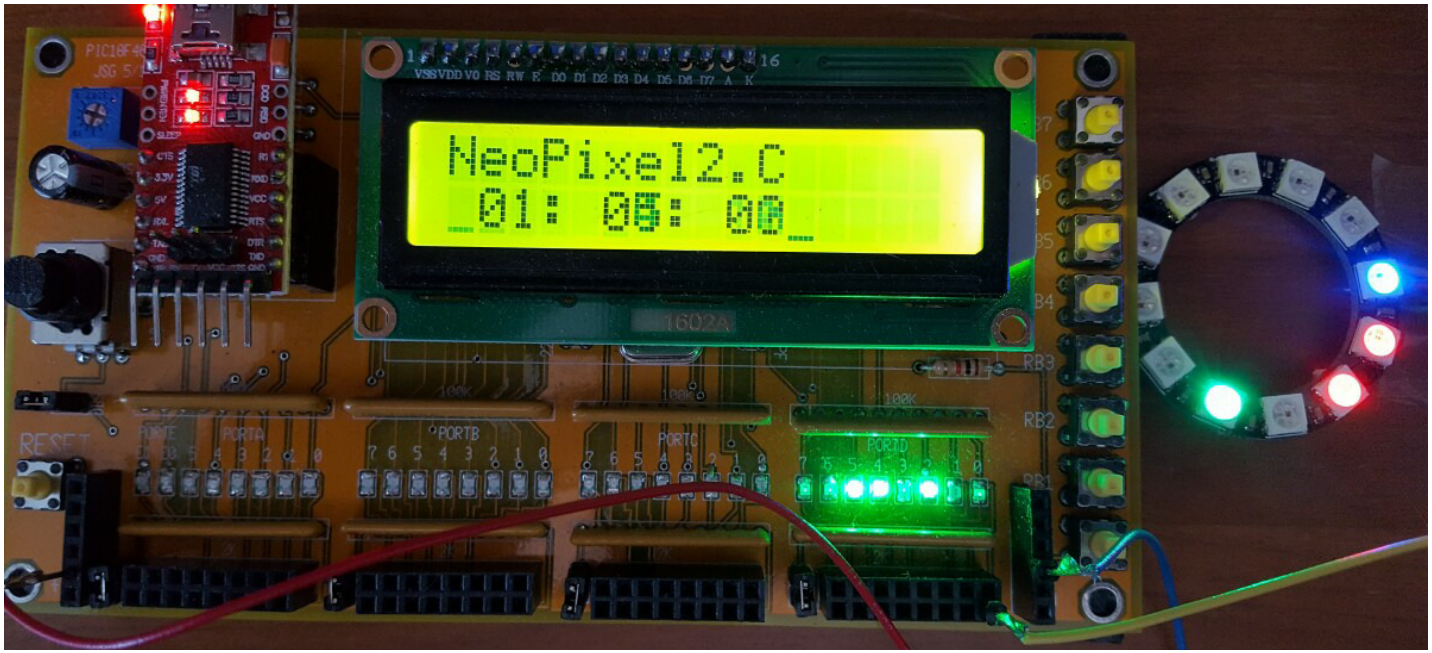
```
RED = 0;
GREEN = 0;
BLUE = 0;

while(1) {
   if(RB5) RED += 1;
   if(RB4) RED -= 1;
   if(RB3) GREEN += 1;
   if(RB2) GREEN -= 1;
   if(RB1) BLUE += 1;
   if(RB0) BLUE -= 1;
   LCD_Move(1,0);   LCD_Out(RED, 3, 0);
   LCD_Move(6,0);   LCD_Out(GREEN, 3, 0);
   LCD_Move(11,0);  LCD_Out(BLUE, 3, 0);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   Wait(50);
   }
```

NeoPixel2.C

Display time as a clock

- Red = seconds
- Green = minutes
- Blue = hours



Note 1: Uses global variables that are arrays

- Pass data from C to assembler using global variables
- One byte for each color of each pixel (R / G / B)
- 12-element NeoPixel used here

```
// Global Variables

unsigned char PIXEL @ 0x000;
const unsigned char MSG0[20] = "NeoPixel2.C ";


unsigned char RED[12];
unsigned char GREEN[12];
unsigned char BLUE[12];
```

Note 2: A subroutine fills in the array

- All LEDs off (000) except for three (hour, minute, second)
- Makes it easier for the main routine (bottom up programming)

```
void Update_RGB(char r, char g, char b)
{
   unsigned char i;
   for (i=0; i<16; i++) {
      RED[i] = 0;
      GREEN[i] = 0;
      BLUE[i] = 0;
      }
   RED[r] = 50;
   GREEN[g] = 50;
   BLUE[b] = 50;
   }
```

Note 3: Timing is critical

- Compute the current time
- Update the arrays (RED, GREEN, BLUE), then
- Drive the NeoPixel

When you start the NeoPixel driver routine, *don't do anything else*

- A 50us pause is interprited as the end of message

```
void NeoPixel_Display(void)

{
   PIXEL = GREEN[0]; asm(" call Pixel_8 ");
   PIXEL = RED[0]; asm(" call Pixel_8 ");
   PIXEL = BLUE[0]; asm(" call Pixel_8 ");

   PIXEL = GREEN[1]; asm(" call Pixel_8 ");
   PIXEL = RED[1]; asm(" call Pixel_8 ");
   PIXEL = BLUE[1]; asm(" call Pixel_8 ");

   (etc)
   }
```

Top Level: Update time (hour, minute, second)

```
while(1) {

    SEC = (SEC + 1) % 12;
    if (SEC == 0) {
       MIN = (MIN + 1) % 12;
       if (MIN == 0) {
          HOUR = (HOUR + 1) % 12;
          }
       }

    LCD_Move(1,0);
    LCD_Out(HOUR, 0, 2);
    LCD_Write(':');
    LCD_Out(MIN, 0, 2);
    LCD_Write(':');
    LCD_Out(SEC 0, 2);

    Update_RGB(SEC, MIN, HOUR);
    NeoPixel_Display();
    Wait(62);
    }
```

Final Results

- 2468 bytes (1234 lines of assembler)
- *Lots* more than I would like to write or to debug

That's also only 3.8% of program memory. A PIC can do a lot more.

```
Memory Summary:
Program space used 9A4h ( 2468) of 10000h bytes ( 3.8%)
Data space used 4Bh ( 75) of F80h bytes ( 1.9%)
EEPROM space used 0h ( 0) of 400h bytes ( 0.0%)
ID Location space used 0h ( 0) of 8h nibbles ( 0.0%)
Configuration bits used 0h ( 0) of 7h words ( 0.0%)
```