Smiley's Workshop 5 Supplement 1 – Cylon Optometry

Cylon Optometry

Joe Pardue © October 11, 2008

Okay, so I ran out of space in the Smiley's Workshop 5 article in Nuts&Volts, and here is stuff left out. I'll also include some things that were in the article so you don't have to keep jumping back and forth.

Hardware

We will recycle the Port I/O project, but rewire the DIP switch, reversing the order of the pins so that the switches match the bit order shown in Figure 1.

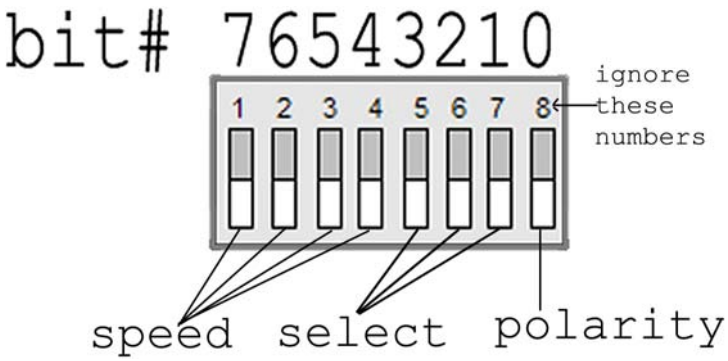

Figure 1: DIP Switch Use

Figure 2 shows what it will look like all wired up, while Figures 3 and 4 show the schematic of the Port B to DIP switch and Port B to LEDs respectively.

Smiley's Workshop 5 Supplement 1 – Cylon Optometry

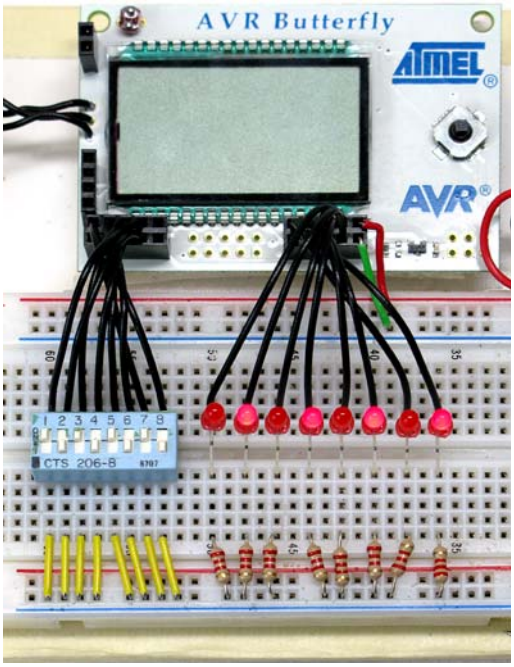

Figure 2: All Wired Up

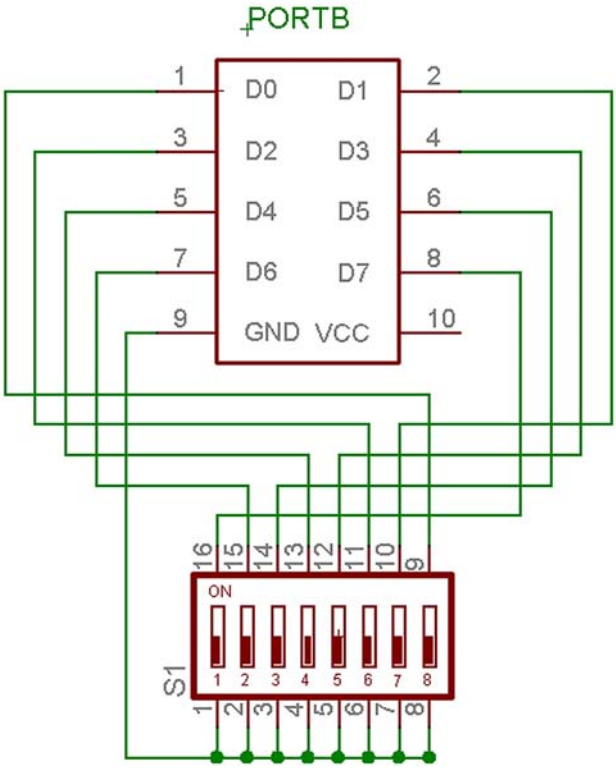

Figure 3: Port B to DIP Switch Wiring

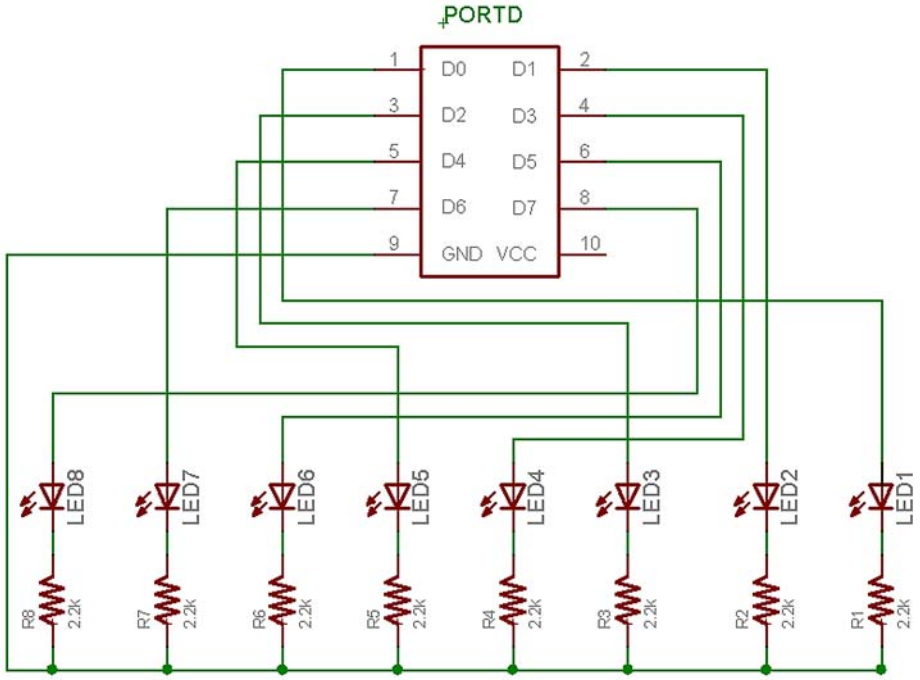

Figure 4: Port D to LED wiring

We will have the following patterns:

cylonEyes – one LED sweeps back and forth.
cylonEyes2 – two LEDs sweep back and forth.
cylonEyes3 – threeLEDs sweep back and forth.
wallEyes – two LEDs start at the center then sweep to the edges and back.
antEyes – three lit LEDs with two dark LEDs between them crawl right to left.
vibroEyes – 10101010 to 01010101 vibrates back and forth.
blinkinEyes – all LEDs blink on and off.
randomEyes – random patterns are shown.

We will be able to set the sweep speed to one of 16 values. And we will be able to set the polarity of the LEDs.

Control Bits on the DIP Switch

As shown in Figure 1, bits used to control which pattern is displayed, the speed and the polarity.

Smiley's Workshop 5 Supplement 1 – Cylon Optometry
Bit 0 - polarity
Bit 1 – pattern bit 0
Bit 2 – pattern bit 1
Bit 3 – pattern bit 2
Bit 4 – speed bit 0
Bit 5 – speed bit 1
Bit 6 – speed bit 2
Bit 7 – speed bit 3

Polarity Bit:

We will use the rightmost switch, which corresponds to bit 0 of PORTB, to control the polarity, meaning that if we have the original Cylon Eyes pattern that sweeps one lit LED across seven dark LEDs, then if we change the polarity, we will sweep one dark LED across seven lit

Pattern Bits:

The switches for bits 1, 2, and 3 are used to set the LED pattern. Three bits can uniquely encode 8 values that we will map to 8 patterns:

Binary	Decimal	Pattern Select
000	0	cylonEyes
001	1	cylonEyes2
010	2	cylonEyes3
011	3	wallEyes
100	4	antEyes
101	5	vibroEyes
110	6	blinkinEyes
111	7	randomEyes

Speed Bits:

The switches for bits 4, 5, 6, and 7 are used to set the LED sweep speed. four bits can uniquely encode 16 values that we will use to set 16 speeds from subjectively slow to fast.

Software Details of One Pattern:

This time, instead of calculating the pattern we will use arrays of bytes with the pattern. For instance the Cylon Eye pattern is:

```
00000001 == 0x01
00000010 == 0x02
00000100 == 0x04
00001000 == 0x08
00010000 == 0x10
```

```
00100000 == 0x20
01000000 == 0x40
10000000 == 0x80
01000000 == 0x40
00100000 == 0x20
00010000 == 0x10
00001000 == 0x08
00000100 == 0x04
00000010 == 0x02
```

We can create an array containing the first 8 values, move forward thru them to get the right to left sweep, then reverse to get the left to right sweep.

```
uint8_t cyloneye[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
```

And we can write a function to step through this array and display the values on the LEDs. This, of course, only sweeps right to left, so if we want to sweep left to right, we have to run through the array backwards.

```
// run up the array
for(i = 0; i <= 7; i++)
{
      if(dillyDally()) return;     // delay or bail out
      if(Polarity) PORTD = ce[i];  // show non-inverted
      else PORTD = ~ce[i];         // show inverted
}
// run down the array
for(i = 6; i >= 1; i--)
{
      if(dillyDally()) return;     // delay or bail out
      if(Polarity) PORTD = ce[i];  // show non-inverted
      else PORTD = ~ce[i];         // show inverted
}
```

We will modify this somewhat for our actual program, but you get the idea? Oh, and dillydally() is a delay function that controls the perceived speed of the sweep.

The remaining patterns are similarly coded and you can see the details in the source code.

Other Patterns:

cylonEyes2

```
00000011 == 0x03
00000110 == 0x06
00001100 == 0x0C
00011000 == 0x18
00110000 == 0x30
01100000 == 0x60
```

Smiley's Workshop 5 Supplement 1 – Cylon Optometry

```
11000000 == 0xC0
01100000 == 0x60
00110000 == 0x30
00011000 == 0x18
00001100 == 0x0C
00000110 == 0x06
00000011 == 0x03
```

cylonEyes3

```
00000111 == 0x07
00001110 == 0x0E
00011100 == 0x1C
00111000 == 0x31
01110000 == 0x70
11100000 == 0xE0
01110000 == 0x70
00111000 == 0x31
00011100 == 0x1C
00001110 == 0x0E
00000111 == 0x07
```

wallEyes

```
00000000 == 0x00
00011000 == 0x18
00100100 == 0x24
01000010 == 0x42
10000001 == 0x81
01000010 == 0x42
00100100 == 0x24
00011000 == 0x18
```

antEyes

```
01001001 == 0x49
10010010 == 0x92
00100100 == 0x24
01001001 == 0x49
```

vibroEyes

```
01010101 == 0x55
10101010 == 0xAA
```

blinkinEyes

```
11111111 == 0xFF
00000000 == 0x00
```

randomEyes

Any and all possibilities (more or less, sort of…).

Smiley's Workshop 5 Supplement 1 – Cylon Optometry

One of the problems we run into with a program like this is that we need to check the switches while continuing to sweep the LEDs. You'll be surprised at how perceptive the eye is in judging that something is out of sync with the sweep pattern. I'm not sure what the minimum 'glitch' time is that is noticeable, but I personally was bothered while writing this code by the time it takes to restart a sweep loop in a while(1) loop. I'd think that the loop ending then the while(1) causing it to start again would be imperceptible, but it isn't. To prevent this, I incorporated checking the switch state in the delay function, dillyDally.

By now you should be able to read the source code with little problems and run the hardware easily. I say this having muffed a couple of demo trying to remember which switches controlled what. Play with it, have fun.