Controlling The Real World With Computers

::. Control And Embedded Systems .::

Putting It All Together - Controlling The Hardware With The Software
Programming Part 3

Programming Part 2 discussed how to vary the apparent brightness of an LED connected to pin 9 on the UNO board, which is an analog pin and so is designated by 9~. Pulse Width Modulation (PWM) is used to change the apparent brightness.

We will now consider a chase using analog rather than simple on/off control to control the LEDs. The fade program that comes with the UNO looks like this:

```
int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once with reset:
void setup()
{
        pinMode(led, OUTPUT);
}

void loop()
{
        // set the brightness of pin 9:
        analogWrite(led, brightness);

        // change the brightness for next time through the loop:
        brightness = brightness + fadeAmount;

        // reverse the direction of the fading at the ends of the fade:
        if (brightness == 0 || brightness == 255)
        {
                fadeAmount = -fadeAmount ;
        }

        delay(30);
}
```

The led variable is initialized to 9, then, in setup pin 9 is declared an output using pinMode(..) . A brightness variable is initialized to 0 and a variable called fadeAmount is set to 5. Inside loop(), the apparent brightness of the LED is accomplished with a call to analogWrite(led, brightness) which causes the LED to be of a brightness level dictated by the brightness variable.

The brightness variable is incremented by adding the fadeAmount to it each time around in the loop. This is accomplished with

```
brightness = brightness + fadeAmount;
```

A more efficient method would be

```
brightness += fadeAmount;
```

because it takes fewer instructions in the final machine code. The next section of code contains a potential error which can be problematic:

```
    if (brightness == 0 || brightness == 255)
    {
        fadeAmount = -fadeAmount ;
    }
```

If the brightness variable reaches 0 or 255, the direction is changed by changing the sign of fadeAmout. More efficient than the above might be

```
fadeAmount *= -1;
```

although it might be found that the method used is just as efficient, and it's more self-explanatory. But the decision that results in the reversal can be much more of a problem. In the above sketch, fadeAmount is initialized to 5. That means brightness will start at 0 then go to 5, etc. until it reaches 255, at which time the reversal takes place.

Problem is, fadeAmount can be initialized only to 1 or 5 or brightness will skip over 255 or go below 0 in the other direction. Consider what would happen if fadeAmount were set to 25. brightness would then be
0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275
It never reaches 255.

The solution is to change the decision slightly:

```
    if (brightness <= 0 || brightness >= 255)
        fadeAmount *= -1;
```

This says to reverse directions if brightness has become less than or equal to 0, OR has increased to a value greater than or equal to 255. Note that no curly braces are needed if there is only one command after the decision. Now it makes no difference what fadeAmount is set to. Also note the double bar (||) for the OR query. As noted in the Boolean Logic section, the setting OR is a single bar. Double symbols are used for queries (&&, ||, ==).

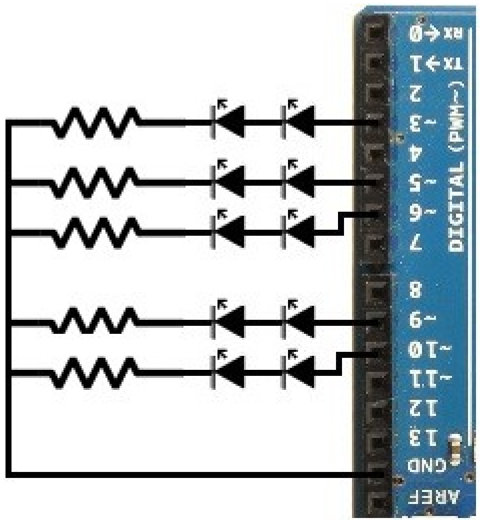
Let's now take a look at a fading chase arrangement. While we are at it, let's add more power to the display. Recall that an LED typically needs 2 volts at 20ma. Two LEDs in series would need 4 volts at 20ma. The UNO can safely provide 20ma at 5 volts.

The size of the current limiting resistors would be the source voltage minus 4 volts divided by 20ma:

(5 - 4)/.02 = 50 ohms

Two values are common, 47 and 51 ohms. Either will work just fine.

Connect the LEDs to the UNO analog outputs and the opposite side of the 47 or 51 ohm resistors to ground on the UNO as follows:



The variables and setup() are about the same as in chase6. The only difference is the change in the pins used and the addition of a PWM value variable:

```
const int LEDs[] = {3,5,6,9,10};
int NumberOfLEDs = sizeof(LEDs) / sizeof(int);
int x,PWM_Val;

void setup()
{
   // make pins outputs and set low
   for(x=0; x<NumberOfLEDs; x++)
   {
     pinMode(LEDs[x], OUTPUT);
     digitalWrite(LEDs[x], LOW);
   }
  //Serial.begin(9600);
}
```

A person could run through the LEDs like this:

```
void loop()
{

  for(x = 0; x < NumberOfLEDs; x ++)
  {

    for(PWM_Val = 0; PWM_Val < 256; PWM_Val ++)
    {
      analogWrite(LEDs[x], PWM_Val); // led number, value
    }

  } // end for(x = 0; x < NumberOfLEDs; x ++)

} // end void loop()
```

But that doesn't work quite right. Some pretty important constants are Just above loop() that can help make things work a little better:

```
const int DelayTime = 1, StepSize = 7, WaitDelay = 50;
```

Work on it and try to incorporate the above to make the program work more smoothly, fade the LEDs up in sequence then down. Then take a look at MultiFadeUpDown.c to see what I did.

Here's another challenge. The basic anatomy of a for loop was covered earlier. But the for loop can do much more. It can contain multiple variables starting at different places and modified in different ways.

Consider the following two variable loop and see if you can figure out what it prints. It's called MultipleForLoop.c

```
void setup()
{
  Serial.begin(9600);  //  set up serial
}

int x,w;

void loop()
{
        for(x = 0, w = 8; x < 4; x++, w /= 2)
        {
                Serial.print(w);
                Serial.print(" ");
        }

        Serial.println(" ");

        delay(1000);
}
```

Next time we will discuss changing a variable with a knob using Analog to Digital conversion.