EE 474 Winter 2017 Lab 2: Timer Interrupts and Pulse-Width Modulation

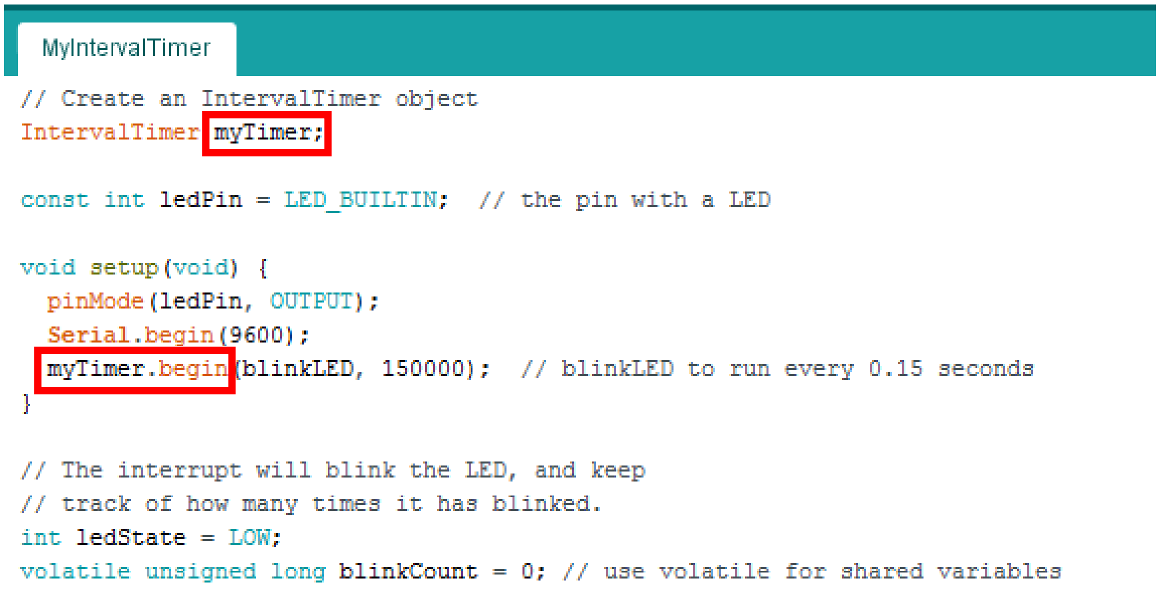
University of Washington

by Ryan Linden, Khalid Alzuhair, and Brandon Ngo

January 24, 2017

Question 1: What starts the timer running?

The myTimer.begin(blinkLED, 150000) call in the sample code MyIntervalTimer is what starts the timer running. This is because it calls myTimer, which is an IntervalTimer that interrupts to call a function at a precise timing interval as seen in Figure 1.

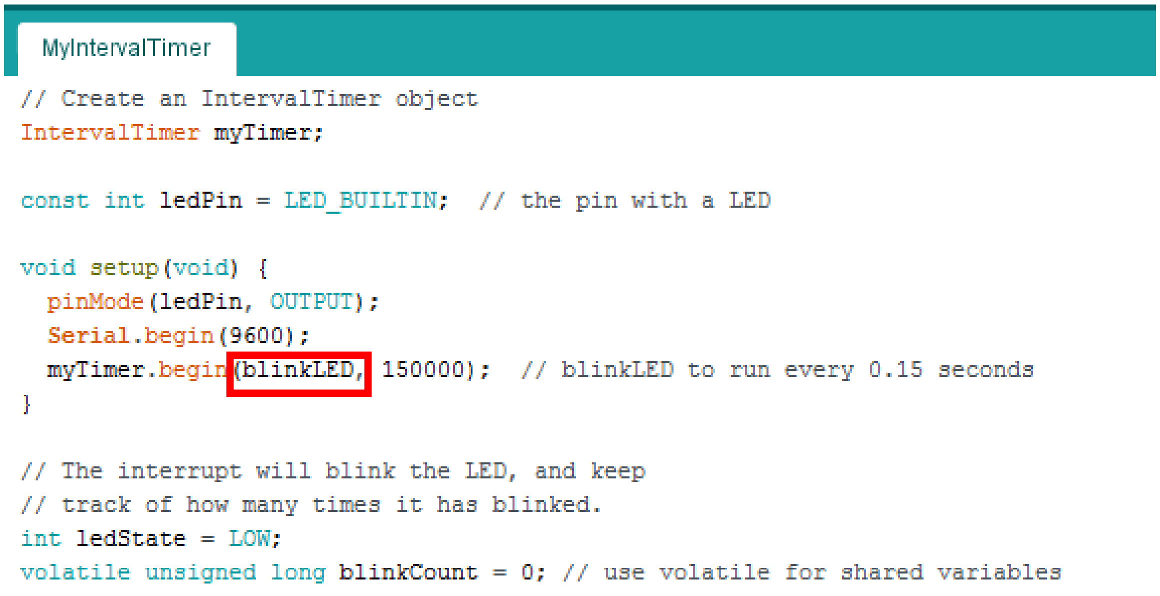


```
// Create an IntervalTimer object
IntervalTimer myTimer;

const int ledPin = LED_BUILTIN;  // the pin with a LED

void setup(void) {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  myTimer.begin(blinkLED, 150000);  // blinkLED to run every 0.15 seconds
}

// The interrupt will blink the LED, and keep
// track of how many times it has blinked.
int ledState = LOW;
volatile unsigned long blinkCount = 0; // use volatile for shared variables
```

Figure 1. Snippet of the sample code MyIntervalTimer with the IntervalTimer instantiation and the call to IntervalTimer myTimer in the setup() highlighted.

Question 2: What function executes when the timer interrupt occurs?

The function blinkLED() executes when the timer interrupt occurs as seen in **Figure 2** with the call to the IntervalTimer myTimer in setup().

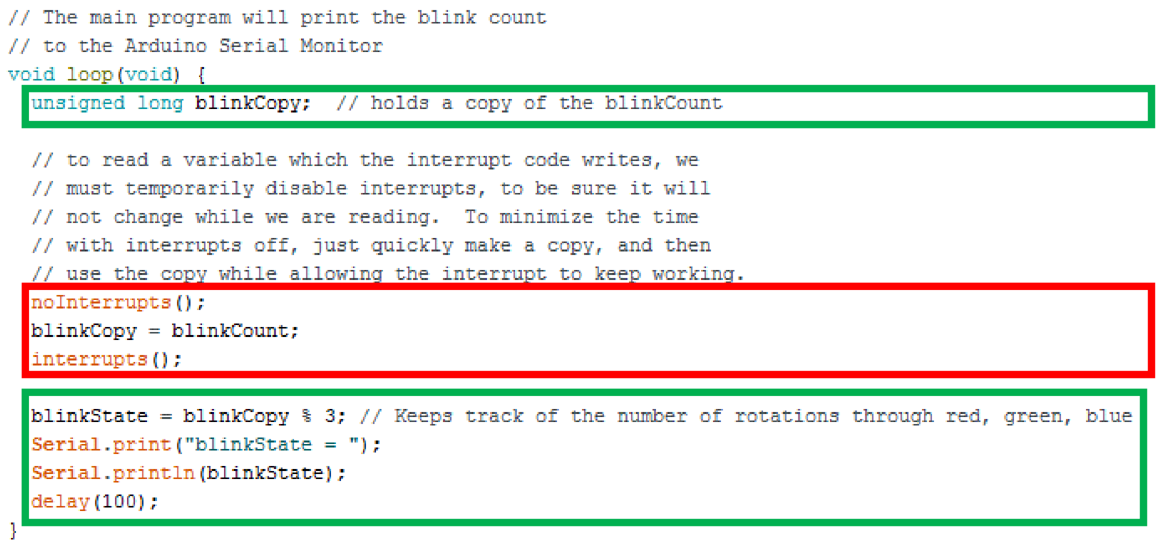


```
// Create an IntervalTimer object
IntervalTimer myTimer;

const int ledPin = LED_BUILTIN;  // the pin with a LED

void setup(void) {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  myTimer.begin(blinkLED, 150000);  // blinkLED to run every 0.15 seconds
}

// The interrupt will blink the LED, and keep
// track of how many times it has blinked.
int ledState = LOW;
volatile unsigned long blinkCount = 0; // use volatile for shared variables
```

Figure 2. Snippet of the sample code MyIntervalTimer with the blinkLED being the function that executes when the timer interrupt occurs.

Question 3: What section of the code is interrupted by the timer interrupt?

The section of code **excluding** that between noInterrupts() and interrupts() in loop() as seen in Figure 3.



```
// The main program will print the blink count
// to the Arduino Serial Monitor
void loop(void) {
  unsigned long blinkCopy;  // holds a copy of the blinkCount

  // to read a variable which the interrupt code writes, we
  // must temporarily disable interrupts, to be sure it will
  // not change while we are reading.  To minimize the time
  // with interrupts off, just quickly make a copy, and then
  // use the copy while allowing the interrupt to keep working.
  noInterrupts();
  blinkCopy = blinkCount;
  interrupts();

  blinkState = blinkCopy % 3; // Keeps track of the number of rotations through red, green, blue
  Serial.print("blinkState = ");
  Serial.println(blinkState);
  delay(100);
}
```

Figure 3. Snippet of the sample code MyIntervalTimer with the sections of code interrupted by the timer interrupt highlighted in green and the section of code **not** interrupted by the timer interrupt highlighted in red.

Question 4: What is the purpose of the following functions:

noInterrupts(): disables interrupts and doesn't allow interrupts to occur
interrupts(): re-enables or allows interrupts to occur

Question 5: What is the purpose of bypass capacitors? Why might they be useful for this lab?

The purpose of bypass capacitors is to reduce noise. If there are significant variations in voltage of these microcontroller circuits, then the circuit may operate incorrectly. With this lab specifically, the bypass capacitors will prevent false interrupts with the cleared out noise.

Question 6: Is the LED affected by the period of the PWM output, or by the duty cycle (on time), or both? Can you run it too fast or too slowly?

Yes, the LED is affected by both the period of the PWM output and the duty cycle. Period is $T = \frac{1}{f}$, so with a larger period the LED would blink fewer times per second while a smaller period would result in the LED blinking more times per second. The PWM duty cycle would also affect the LED. With the larger duty cycle, the LED would appear to be off for longer since the high voltage is on for longer according to Figure 4. With a smaller duty cycle, the LED would appear to be on for longer since the low voltage is on for longer also according to Figure 4. Recall that since the LED is common-anode, OFF is a 1 and ON is a 0 on the respective port pins.

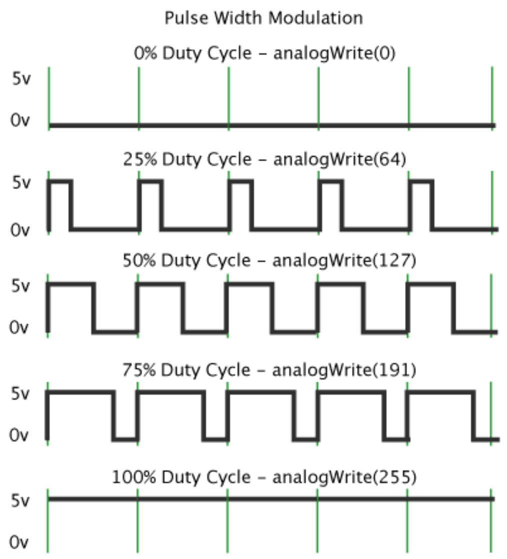

Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

Figure 4. Snippet of PWM on the Arduino webpages. Notice that higher duty cycle corresponds to the higher voltage being on for longer while lower duty cycle corresponds to the lower voltage being on for longer.

Yes, you can run it too fast or too slowly. If the period is too long (smaller frequency) then the LED would appear to be blinking rather than fading since it is blinking fewer times per second.

If the period is too short (higher frequency) then the LED would appear to be always on rather than changing state since it is blinking more times per second.