ECE 241 **Spring 2017**
Jacob Peterson
Brycen Parker **Final Project Report**

Objective: Control the speed of a motor by adjusting the duty cycle of a Pulse Width Modulated (PWM) signal. The duty cycle of the motor will be controlled by an encoder, and the result can be printed to the LCD and the serial monitor.

Hardware: For our circuit, we used 2 transistors, the IRF510 and the 2N2222 (circuit diagrams shown in figures 1 and 2, respectively). The IRF510 was used in the PWM circuit and the 2N2222 was used in the feedback circuit.

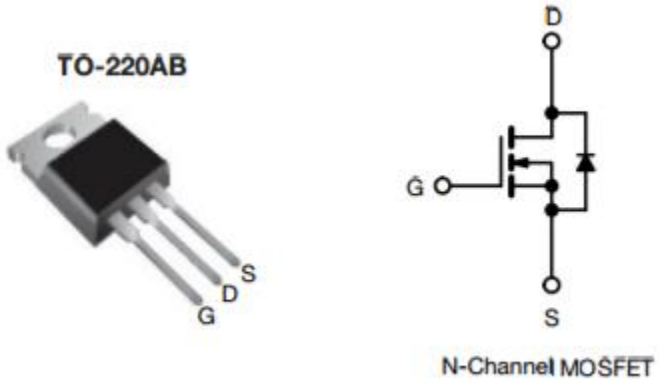

Figure 1: IRF510 Transistor

The IRF510 allowed for switching and amplification of signals coming from the PWM signal line of the Arduino. This transistor was located on our circuit between the Arduino's PWM signal, the 12 V external supply, and ground. The IRF510 is shown in Figure 1.

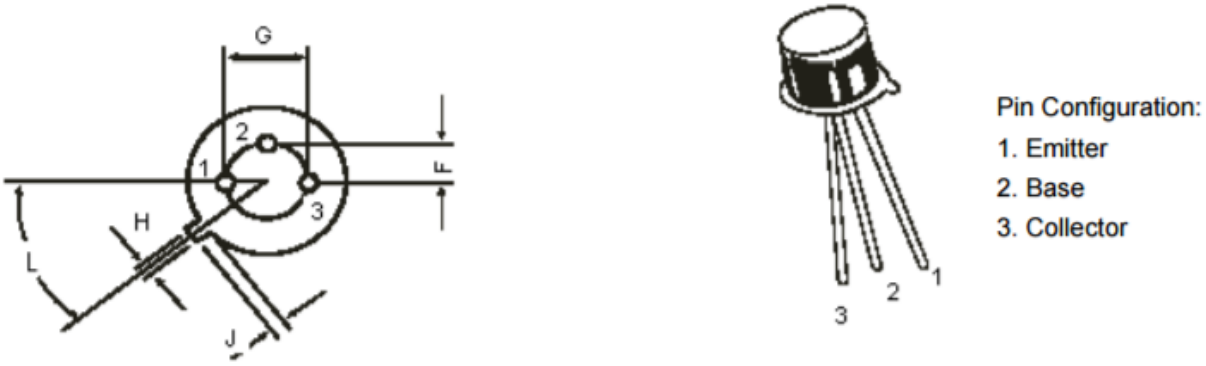

Figure 2: 2N2222 Transistor

The 2N2222 allowed for switching and amplification of signals into the feedback pin (pin 12). This transistor was located on our circuit between the feedback pin, the 5 volt supply from the Arduino, and ground. The 2N2222 is shown in Figure 2.

Aside from the motor, the Arduino, and the two transistors, our circuit also used 2 diodes (1N4001 and 1N4148) and 2 10K resistors.

Software: The basis of our code is an Interrupt Service Routine which controls the speed of the motor from the encoder position. This ISR uses the value of the encoder position (we also included code for encoder position handling) to set the duty cycle of the motor using the PWM signal pin on the Arduino. The code for the ISR can be found in Figure 3 below.

```
x=EncoderValue/220.0 *1023;  //converts the encoder position into a duty
cycle and stores in x
  if (x>1023)  //prevents x from going above a full duty cycle
  {
    x=1023;  //caps duty cycle at 1023
  }
  if (x<0)  //prevents x from going below a full duty cycle
  {
    x=0;  //sets x to a minimum of 0
  }
  Serial.println(x);  //test code which prints x to serial monitor to monitor
duty cycle
  Timer1.setPwmDuty(10, x);  //sets the duty cycle of the motor to x
```

Figure 3: ISR for motor control

Another important part of our code was the code segment which controlled capturing the period and printing the RPM to either the LCD or the serial monitor. These statements capture the rising edge and falling edge of the square wave generated by the motor, and uses them to calculate and display the RPM to the LCD or serial monitor. These statements were performed in the loop. The pseudocode for our loop can be found in Figure 4 below.

```
Inside loop:

Set MeasurementTimer to micros

Stall for 10 microseconds

Capture time when pin 12 goes high using micros function
```

```
Stall for 12 to go low

Capture time when 12 goes low and stores difference between high
and low in MeasurementTimer

If 100 milliseconds has passed:

     If cycle time is non-zero:

          Clear LCD

          Print RPM to LCD using provided formula

     Increment timer
```

Figure 4: pseudocode for loop

Our entire code for this project can be found in Appendix A below.

Testing: The biggest hurdle for us was getting the loop to successfully display the RPM. We used the serial monitor to help test our code by displaying things at various times to help us visualize what was working in the code. We also used the Analog Discovery tool to monitor the square wave generated by the motor in order to gain an understanding of the times we were trying to capture.

The encoder code and the main loop to step up the motor ran very smoothly, the RPM calculation was the difficult part of the project. However, using the serial monitor statements seen in the code and the analog discovery tool, we were able to work through how to successfully calculate RPM values.

We used a slightly different circuit diagram to wire our board than the one provided in the project description. Our diagram used different pieces of hardware to drive to motor (documented above) and was an easier circuit to visualize.

Appendix A: Entire code for project

```
#include <LiquidCrystal.h>  //include LCD library
#include <TimerOne.h>  //include TimerOne library
#include <MsTimer2.h>  //include MsTimer2 library

LiquidCrystal LcdDriver(11,9,5,6,7,8);  //pins required to interface to the
LCD
float x;  //variable used in duty cycle
unsigned long MeasurementTimer = 0, Timer;

#define EncoderRead_H
volatile int EncoderValue = 0;  // Variable for keeping track of encoder
change.
int EncAPin = 2,
EncBPin = 3; int EncA, EncB;

void PinA(void) {  // Service Routine for Encoder channel A, active on
channel A changing.

    if (digitalRead(EncBPin) != digitalRead(EncAPin)) {  // Check the two
inputs and then if not equal
        EncoderValue++;  // Increment the encoder.
    }
    else {
        EncoderValue--;  // otherwise decrement
    }

}  // End of PinA

void PinB(void) {  // Service Routine for Encoder channel B, active on
channel B changing.

    if (digitalRead(EncBPin) == digitalRead(EncAPin)) {  // Check the two
inputs and then if equal
        EncoderValue++; // Increment the encoder.
    }
    else {
        EncoderValue--;// otherwise decrement
    }

} // End of PinB

void EncoderInitialize() {  // Setup interrupt services and pinModes for the
Encoder lines.

    attachInterrupt(0, PinA, CHANGE); // ISR's
    attachInterrupt(1, PinB, CHANGE);
    pinMode(EncAPin, INPUT); // Pin Modes.
    pinMode(EncBPin, INPUT);

}// End of EncoderInitialize
```

```
////////////////////////////////////////////////////////////////////////////


void setup() {

    MsTimer2::set(10, Adc_Dac_ISR);  //set ISR to run every 10 milliseconds
    MsTimer2::start();  //start MsTimmer2
    //Serial.begin(115200); serial monitor used for testing
    pinMode(10, OUTPUT);  //set pin 10 (pwm pin) to output
    pinMode(12, INPUT);  //set pin 12 (feedback pin) to read input
    EncoderInitialize();  //initializes encoder components
    Timer = millis();  //sets Timer to count milliseconds
    Timer1.initialize(500);  //sets period for Timer1 function
    Timer1.pwm(10, 512);  //sets initial duty cycle

}


void loop() {

MeasurementTimer = micros();

while (micros() - MeasurementTimer < 10);  //stall for 10 micros

while (digitalRead(12) == LOW);

MeasurementTimer = micros();  // Time when 12 went high.

while (digitalRead(12) == HIGH);  // Wait for 12 to go low. (Stalls as long
as it is high ).

MeasurementTimer = micros() - MeasurementTimer; //  compute delta t

if (millis()-Timer>=100)  //checks if 100 milliseconds have passed
  {
    if (MeasurementTimer>0) //  checks if cycle length is non-zero
    {
   LcdDriver.clear();  //clear LCD
   LcdDriver.print(60000000/(8*MeasurementTimer));  //print RPM to LCD

    }
    Timer += 100;  //increment timer
  }

}

void Adc_Dac_ISR()  //ISR to control motor speed
{

  x=EncoderValue/220.0 *1023;  //converts the encoder position into a duty
cycle and stores in x
  if (x>1023)  //prevents x from going above a full duty cycle
  {
    x=1023;  //caps duty cycle at 1023
```

```
  }
  if (x<0)  //prevents x from going below a full duty cycle
  {
    x=0;  //sets x to a minimum of 0
  }
  //Serial.println(x); test code which prints x to serial monitor to monitor
duty cycle
  Timer1.setPwmDuty(10, x);  //sets the duty cycle of the motor to x

}
```