Objective

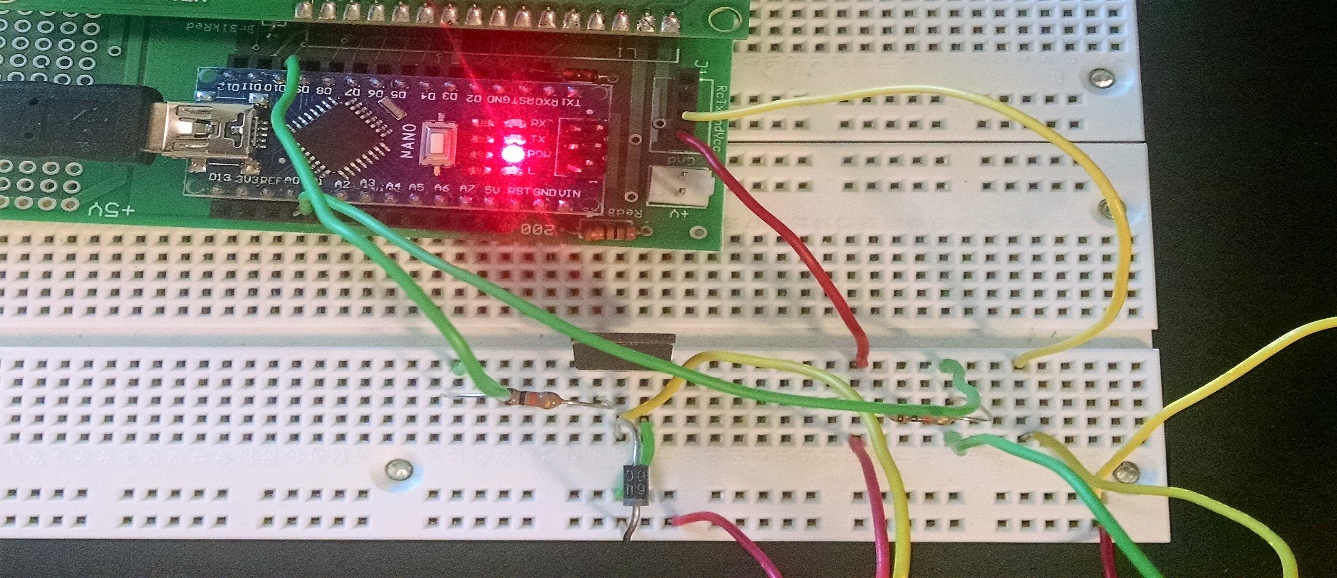
The object for our final project was to program a Motor Controller using the Arduino. This controller should correctly calculate the revolutions per minute (RPM) (using *RPM = (1.052632 x 10 ^ 6)/ΔT)*) of the motor, control the RPM of the motor, maintain good control under varying loads, and finally represent the RPM on the LCD.

Hardware

To complete this project, we had to have additional hardware. The extra hardware included:

1. A motor
2. 2 Resistors (10K)
3. 1 Diode
4. 1 MOSFET (IRF 510)
5. Several jump wires
6. A breadboard from the lab
7. A power supply (5V from the lab machines)

The wiring was a bit tricky at first, however we figured it out. **Figure 1** depicts a picture of what our wiring looked like on the project.



**Figure 1:** The wiring for this project is depicted here. We utilized an analog pin and digital pin. This

wiring coincides with the schematic given to us.

The MOSFET was used to drive the motor. This was protected by a 10K resistor. The other jump wires and resistor were used to send and receive from the motor encoder. Controlling the motor proved to be a challenging task. This portion will be covered the “Software” section.

Software

Writing programs for the motor proved to be a challenging task. Because of the nature of motor, it varies from HIGH to LOW too rapidly. To calculate the RPM, we needed to find the change in time from LOW to LOW (or HIGH to HIGH peak). We were able to accomplish through the usage of while loops. Inside the “findMotorRPM()” section of the code, we were able to find the “***ΔT”.*** We wrote more functions to deal with finding the state of the Motor Encoder. To do this, we wrote a header file for the encoders. This code is found in **Appendix** B. We called the “analogRead()” function on the motorEncoderPin and stored the resulting value inside a local integer variable. We then used this value to test if the pin was HIGH or LOW. The return value of this function was used inside the while loops to determine when the motor switched from HIGH to LOW or LOW to HIGH. Once we had found the ***ΔT*** we passed the value into the function, “RPMCalculator()” and stored the returned value in the global variable “motorRPM”. This was then displayed on the LCD in the loop.

One of the last functions we wrote was the “reachTargetRPM()” function. This function’s job was to take the value given by the Arduino encoder (user input), and force the motor to approach the RPM’s. However, we did have to perform some checks on this. To ensure our motor was protected, we capped the motor speed (from the encoder) at 255. The lowest speed was set to 20 because at that RPM the motor would not have enough voltage to run. If the target RPM (given by the Arduino encoder) was less than the motor RPM, then we incremented the motor speed by 1 until it was approximately the same as the target RPM. We performed the opposite if the target RPM was lower than the motor RPM.

To display the RPM (from the motor) and the target RPM (from the Arduino encoder), we used two variables to slow down the how frequently the two RPM’s were sent to the LCD. To send the RPM’s to the LCD, we wrote a function entitled “SendDisplay()”. This function sets the cursor on the LCD and prints each of the RPM’s with their respective labels. The bulk of our software may be found in **Appendix A.**

Testing

To test our program, we utilized the Serial libraries. This was a great way to see if our program was calculating the RPM correctly and printing the RPM’s on the correct interval. Once we had our program printing to the serial monitor correctly, we were easily able to translate our code into printing to the LCD (as previously described). Overall, we spent much more time running through or code and thinking it through than we did testing it.

**Appendix A (Main body of code)**

#include <LiquidCrystal.h> //Includes the LiquidCrystal Libraries

#include "Encoder\_Button.h" //Includes the header for the Encoder\_Button

LiquidCrystal LcdDriver(11, 9, 5, 6, 7, 8); //Type LiquidCrystal

//timer vars

unsigned long edgeTimer = 0; //Holds the time at which the Motor encoder is LOW

unsigned long displayTimer = 0; //Holds the time at which the display should print

int displayDelay = 1000; //Delays the display value

//RPM variables

float elapsedTime = 0; //Difference in time from LOW peak to LOW peak

float targetRPM = 200; //Desired RPM from the Arduino Encoder

float motorRPM = 0; //Actual RPM from the Motor

int motorSpeed = 100; //Base intialization of motorSpeed

//pin values

const int motorPin = 10; //Sets pin 10 up as the motorPin

const int motorEncoderPin = A0; //Sets analog pin "A0" as the motorEncoderPin

//Helper function sends the RPM to the LCD

void SendDisplay(float rpmRecieved, float rpmTarget)

{

LcdDriver.setCursor(0, 0); //Sets the position of the LCD

LcdDriver.print("RPM: "); //Prints the RPM's at (0, 0)

LcdDriver.print(rpmRecieved); //Prints the RPM's directly after "RPM: "

LcdDriver.setCursor(0, 1); //Sets the position at the 2nd row

LcdDriver.print("Target: "); //Prints "Target: "

LcdDriver.print(rpmTarget); //Prints the value of rpmTarget from Arduino encoder

}

//function for calculating RPM

float RPMCalculator(unsigned long timeChange) //timeChange in microseconds

{

return 1.052632 \* pow(10, 6) / timeChange; //Returns the RPM's

}

//function for checking state of motor encoder

boolean findMotorEncoderState()

{

// Sets the value of temp variable analogValue

int analogValue = analogRead(motorEncoderPin); //reads the value...

//if the reading is below 500...

if (analogValue < 500)

{

return LOW; //return low

}

//otherwise...

else

{

return HIGH; //return high

}

}

//This function helps the motor reach the Target RPM

void reachTargetRPM()

{

int acceleration = 1; //base intialization

//If RPM of the Motor is less than the desired from the Arduino Encoder

if (targetRPM > motorRPM)

{

motorSpeed += acceleration; //increments motorSpeed w/acceleration

//If the motorspeed has accelerated passed 255...

if (motorSpeed > 255)

{

motorSpeed = 255; //sets motorSpeed to 255

}

analogWrite(motorPin, motorSpeed); //writes motorPin to an analog signal w/motorSpeed

}

//Checks to see if the RPM of the motor > desired RPM of arduino encoder

else if (targetRPM < motorRPM)

{

motorSpeed -= acceleration; //decrements motorSpeed w/acceleration

//if the motorspeed has slowed below 0

if (motorSpeed < 0)

{

motorSpeed = 0; //sets motorSpeed to 0

}

analogWrite(motorPin, motorSpeed); //writes motorPin with motorSpeed to analog

}

}

//This function finds the current RPM of the motor

void findMotorRPM()

{

//while motor encoder state is HIGH

while (findMotorEncoderState() == HIGH)

{

//idles until LOW peak

}

//while motor encoder state is LOW

while (findMotorEncoderState() == LOW)

{

edgeTimer = micros(); //Sets edgeTimer to the amount of time passed in microseconds

}

//while motor encoder state is HIGH

while (findMotorEncoderState() == HIGH)

{

//idles until next LOW peak

}

//while motor encoder state is LOW

while (findMotorEncoderState() == LOW)

{

elapsedTime = micros() - edgeTimer; //Sets elapsedTime equal to difference in time (LOW to LOW)

}

motorRPM = RPMCalculator(elapsedTime); //Sets the motor RPM's value of RPMCalculator(elapsedTime)

}

//This function updates the targetRPM variable based on the encoder button

void updateTargetRPM()

{

int encoderConvert = encoderPosition;

targetRPM += encoderConvert \* 0.25; //update target RPM based on encoder value

//if targetRPM is less than 20...

if (targetRPM < 20)

{

targetRPM = 20; //sets targetRPM to 20

}

//else if targetRPM is greater than 270...

else if (targetRPM > 270)

{

targetRPM = 270; //sets targetRPM to 270

}

//otherwise

encoderPosition = 0; //sets the value for the encoderPosition

}

//setup code here, to run once

void setup()

{

Serial.begin(9600); //Sets the baud rate to 9600

displayTimer = millis(); //timer setup

//Encoder and button setup

EncoderButtonSetup(); //Calls EncoderButtonSetup

//LCD setup

LcdDriver.begin(16, 2); //for 16x2 display

LcdDriver.clear(); //clear screen

//pin setup

pinMode(motorPin, OUTPUT); //sets up motorPin "10" for OUTPUT

pinMode(motorEncoderPin, INPUT); //sets up motorEncoderPin "A0" for INPUT

analogWrite(motorPin, motorSpeed); //writes motorPin with the motorSpeed

}

//main code here, to run repeatedly

void loop()

{

updateTargetRPM(); //Calls updateTargetRPM

findMotorRPM(); //Calls findMotorRPM

reachTargetRPM(); //Calls reachTargetRPM

//This controls the LCD Display

if (millis() - displayTimer > displayDelay)

{

SendDisplay(motorRPM, targetRPM); //Sends the RPM of the motor and the RPM desired to LCD

displayTimer += displayDelay; //Increments displayTImer

}

}

**Appendix B – (Encoder header file)**

#ifndef Encoder\_Button\_H

#define Encoder\_Button\_H

//global vars

long encoderPosition;

unsigned long EncoderTimer;

//states for state machine

enum States {Idle, Wait, Low};

int ButtonState = 0; //global variable holding status of button

//function that is called to service the switch

int ButtonNextState(boolean input) {

switch (ButtonState) {

case Idle: //state where nothing is happening

if (input == LOW) {

EncoderTimer = millis(); //record time of high to low transition

ButtonState = Wait; //move to wait state

digitalWrite(13, HIGH); //turn on LED

}

break; //exit case

case Wait: //button has gone low and wait for it to remain low for 5 millisecs

if (input == HIGH) { //if button has gone high,

ButtonState = Idle; //reset back to Idle

}

else if (millis() - EncoderTimer >= 5) { //if 5 millisecs passes

ButtonState = Low; //move to low state

digitalWrite(13, LOW); //turn off LED

return 1; //to indicate button press

}

break; //exit case

case Low: //button is low and has been for 5 msecs

if (input == HIGH) { //once button released

ButtonState = Idle;

}

break; //exit case

}//end switch

return 0; //default: nothing happening

}//end ButtonNextState

//interrupt service routines

void MonitorA() {

if (digitalRead(2) == digitalRead(3)) { //if pin 2 and 3 are equal,

encoderPosition++; //incr encoderPosition

}

else {

encoderPosition--; //otherwise decr

}

}

void MonitorB() {

if (digitalRead(2) == digitalRead(3)) { //if pin 2 and 3 are equal,

encoderPosition--; //decr encoderPostion

}

else {

encoderPosition++; //otherwise incr

}

}

//set up for encoder and button

void EncoderButtonSetup() {

//attach interrupt to the ISR fuctions

attachInterrupt(0, MonitorA, CHANGE); //pin2

attachInterrupt(1, MonitorB, CHANGE); //pin3

//button setup

pinMode(13, OUTPUT); //sets LED pin for output

pinMode(4, INPUT); //sets button pin for input

ButtonState = Idle;

EncoderTimer = millis();}#endif