ELCT 201 – EE LABORATORY [#7]

DIGITAL TRANSFORMATION OF D&C CIRCUIT ELEMENTS

KAILEN KING

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF SOUTH CAROLINA
COLUMBIA, SC 29208

ABSTRACT

The objective of this project is to use c++ programing with compilers in conjunction with a microcontroller to run a program. A terminal emulator will also be used to interact with a microcontroller. Code segments will be troubleshooted as sensor readings are observed.

INTRODUCTION

This project combines the skills of computer science with electrical engineering. There will be code segments that will be used to sense light, temperature, and motor torque states. A LDR will be used to sense light and power the circuit. A thermistor will be used to sense overheating of a motor and turn it off.

DESCRPITION OF CIRCIUT CONSTRUCTION

Figure one shows the complete construction of the circuit.

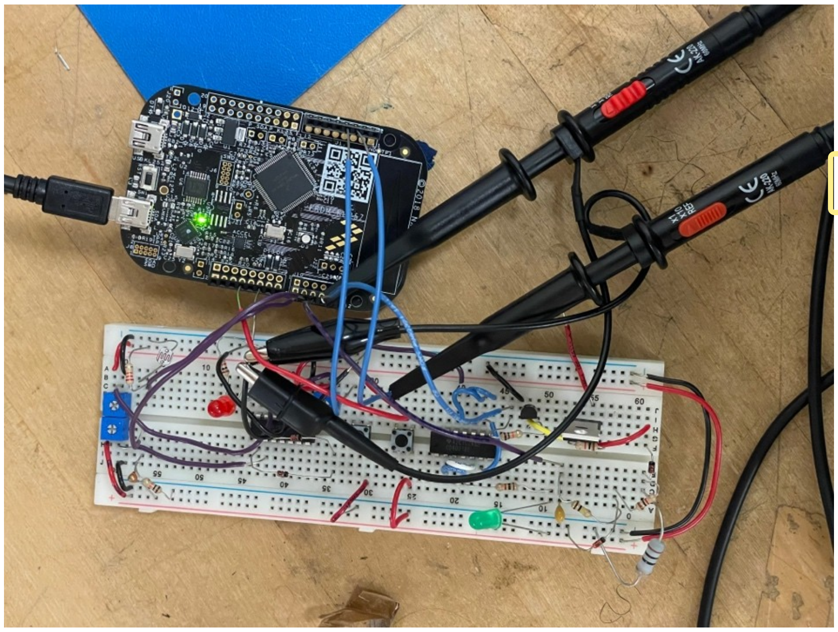


Figure 1

First input ports are soldered to the microcontroller so Jumper wires can connect from the breadboard to give power to the circuit. Ports on the microcontroller are shown in figure 2. PTD4 and PTD12 are used to connect the start and stop buttons from the micro controller to the circuit board. After that power of either port labeled 3v3 is connected to the circuit to run the circuit.

Either port named GND is used to ground the microcontroller with the circuit board. Port PTB0 is used to connect the LDR with the microcontroller. The light dependent resistor relates to a 3.3k resistor for reference. The thermistor will relate to a 10k resistor and jump to the microcontroller port PTB1. PTB2 will be used to connect the motor from the circuit to the microcontroller. PTC0 is used as the digital output to send a signal. This signal goes through a 10k ohm resistor in series with an npn transistor to activate a PMOS which allows the motor to receive 12v of power. A 100k resistor and diode will be connected in parallel along with a series 1k resistor to stop excess current from the motor damaging the microcontroller.

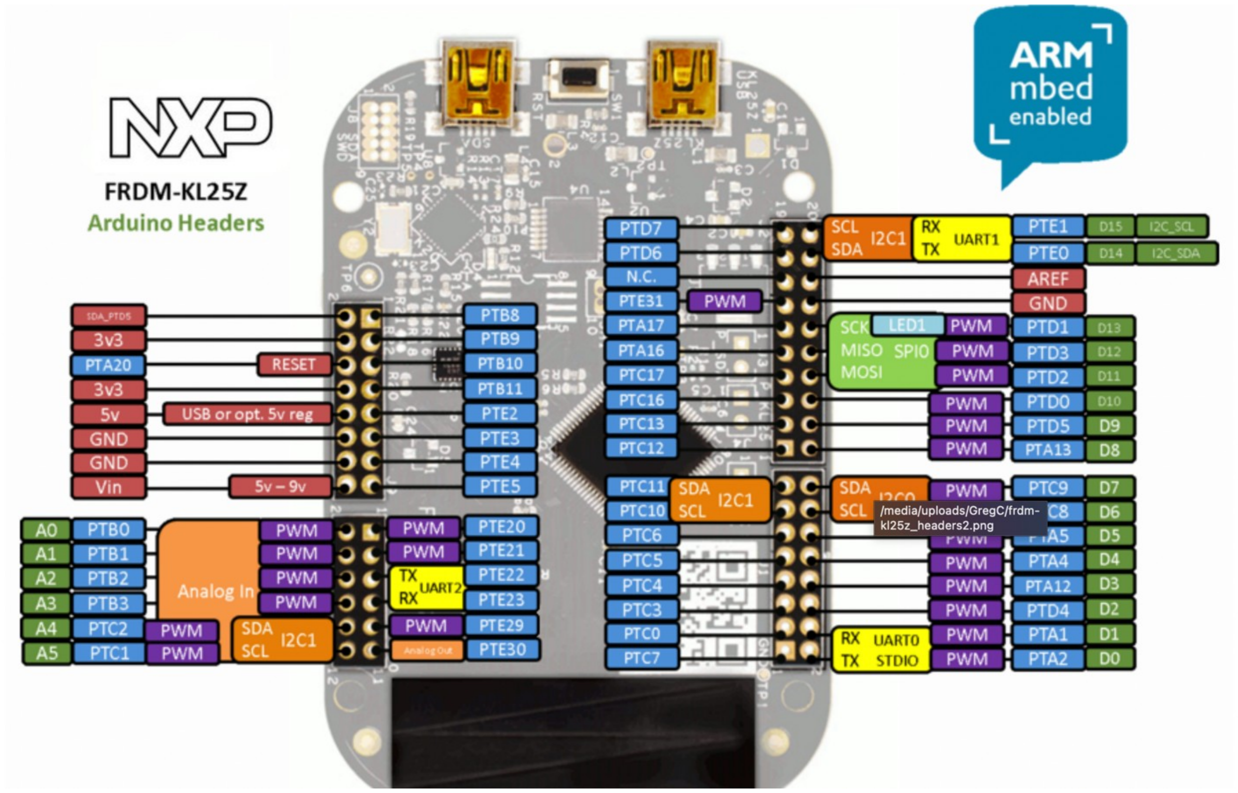
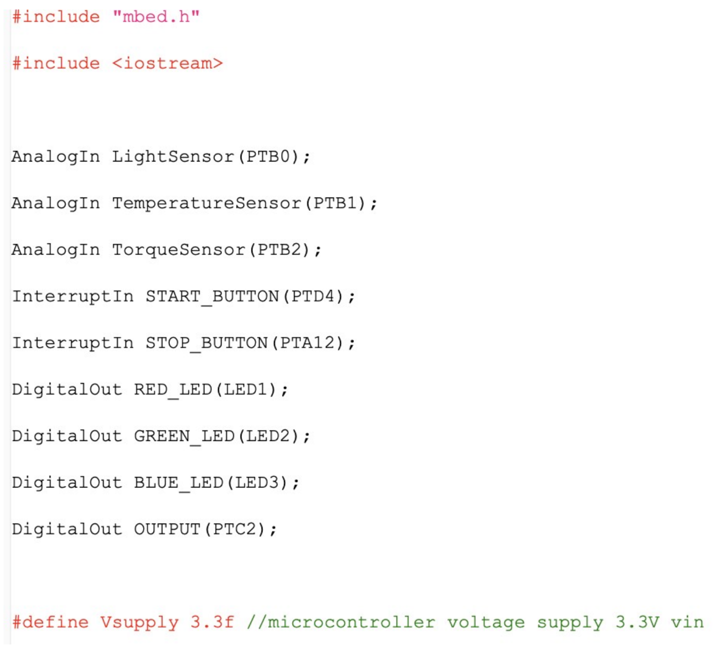


Figure 2

DESCRIPTION OF SOFTWARE

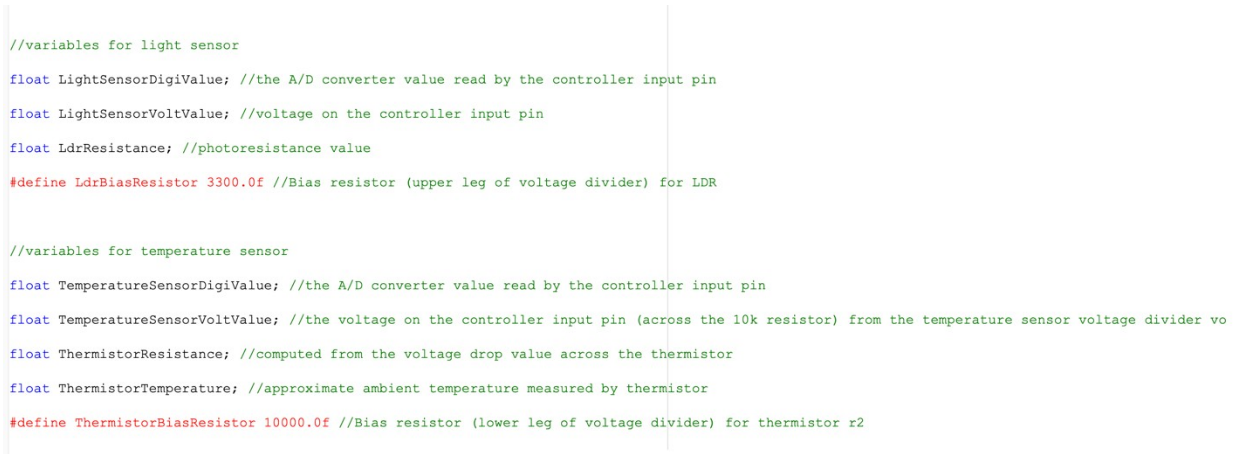
The software section of this project included using a template blinking light program and modifying it. In figure 3 there is a list of all components the components for input and output. The voltage is also defined in this program as 3.3f.



```
#include "mbed.h"
#include <iostream>

AnalogIn LightSensor(PTB0);
AnalogIn TemperatureSensor(PTB1);
AnalogIn TorqueSensor(PTB2);
InterruptIn START_BUTTON(PTD4);
InterruptIn STOP_BUTTON(PTA12);
DigitalOut RED_LED(LED1);
DigitalOut GREEN_LED(LED2);
DigitalOut BLUE_LED(LED3);
DigitalOut OUTPUT(PTC2);

#define Vsupply 3.3f //microcontroller voltage supply 3.3V vin
```

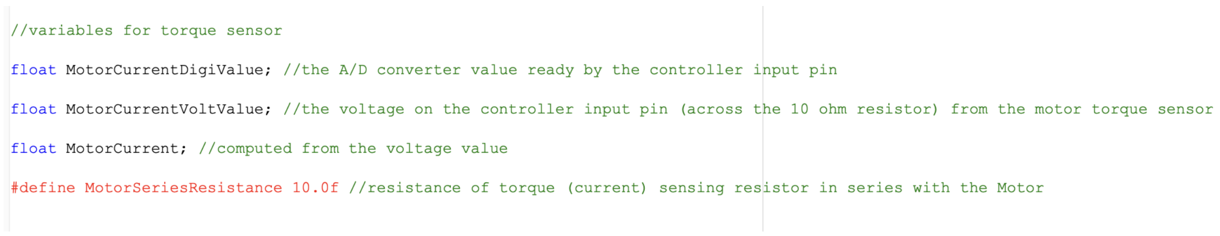
Figure 3

The LDR, thermistor and motor torque sensors also have all their variables defined along with the resistors used for their reference. Figure 4 shows how there's a Digi value and volt value for both the LDR and the thermistor. The Motor has the same values to define as shown in figure 5.



```
//variables for light sensor
float LightSensorDigiValue; //the A/D converter value read by the controller input pin
float LightSensorVoltValue; //voltage on the controller input pin
float LdrResistance; //photoresistance value
#define LdrBiasResistor 3300.0f //Bias resistor (upper leg of voltage divider) for LDR

//variables for temperature sensor
float TemperatureSensorDigiValue; //the A/D converter value read by the controller input pin
float TemperatureSensorVoltValue; //the voltage on the controller input pin (across the 10k resistor) from the temperature sensor voltage divider vo
float ThermistorResistance; //computed from the voltage drop value across the thermistor
float ThermistorTemperature; //approximate ambient temperature measured by thermistor
#define ThermistorBiasResistor 10000.0f //Bias resistor (lower leg of voltage divider) for thermistor r2
```

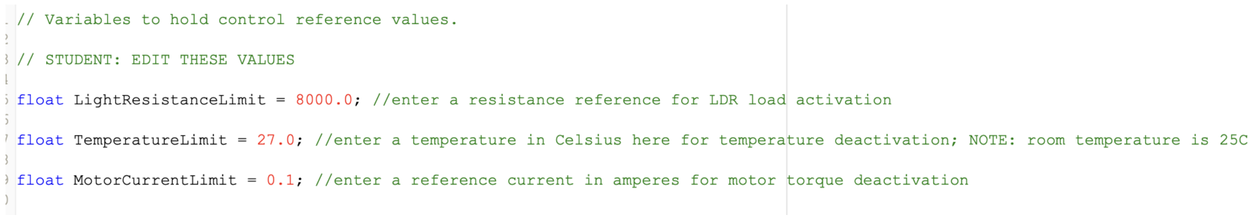
Figure 4



```
//variables for torque sensor
float MotorCurrentDigiValue; //the A/D converter value ready by the controller input pin
float MotorCurrentVoltValue; //the voltage on the controller input pin (across the 10 ohm resistor) from the motor torque sensor
float MotorCurrent; //computed from the voltage value
#define MotorSeriesResistance 10.0f //resistance of torque (current) sensing resistor in series with the Motor
```

Figure 5

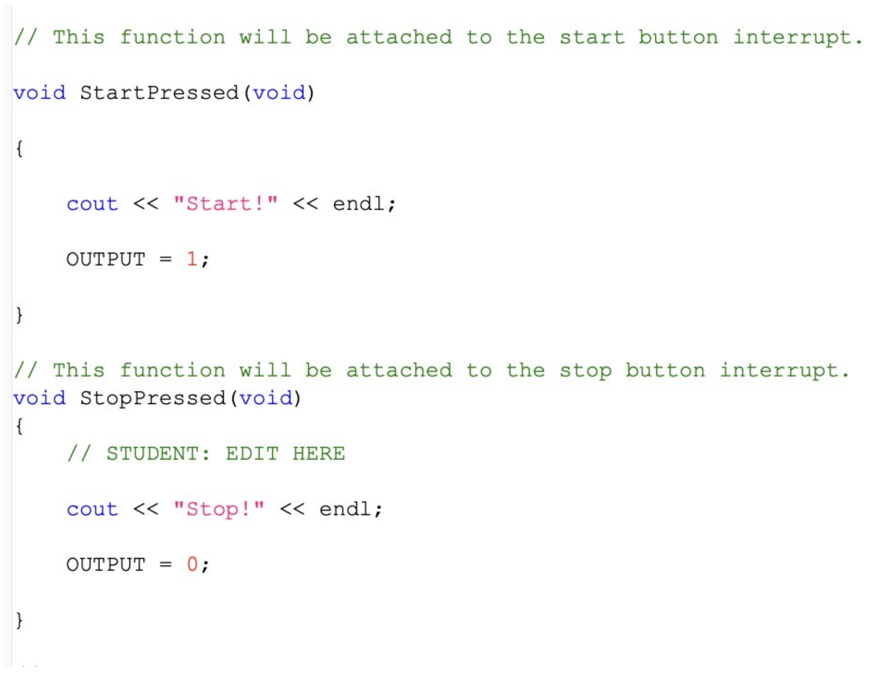
The values for reference are defined after taking some measurements for the resistance limits in figure 6.



```
// Variables to hold control reference values.
// STUDENT: EDIT THESE VALUES
float LightResistanceLimit = 8000.0; //enter a resistance reference for LDR load activation
float TemperatureLimit = 27.0; //enter a temperature in Celsius here for temperature deactivation; NOTE: room temperature is 25C
float MotorCurrentLimit = 0.1; //enter a reference current in amperes for motor torque deactivation
```

Figure 6.

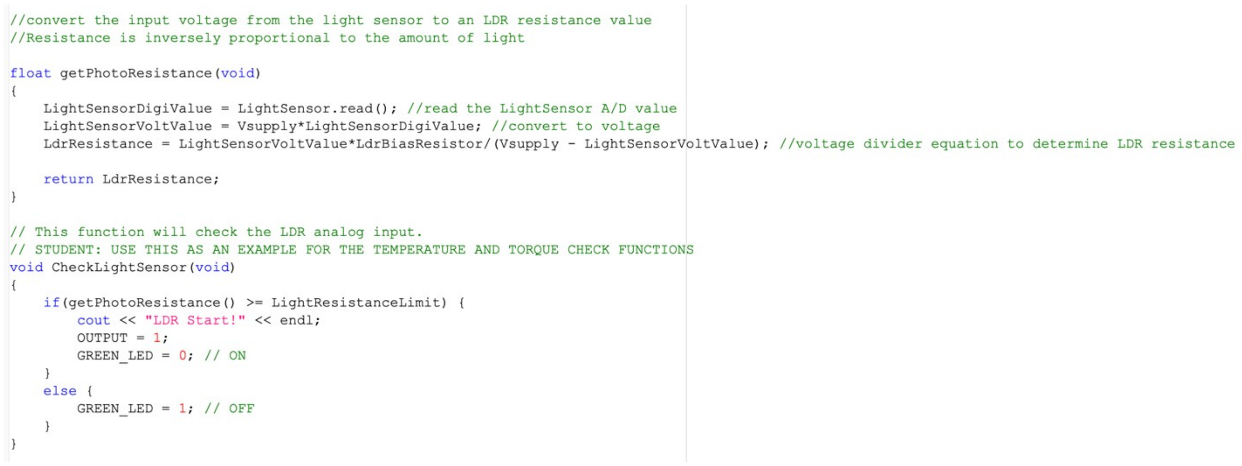
The start button logic is already defined, so the stop function is created based on that code in figure 7.



```
// This function will be attached to the start button interrupt.
void StartPressed(void)
{
    cout << "Start!" << endl;
    OUTPUT = 1;
}
// This function will be attached to the stop button interrupt.
void StopPressed(void)
{
    // STUDENT: EDIT HERE
    cout << "Stop!" << endl;
    OUTPUT = 0;
}
```

Figure 7

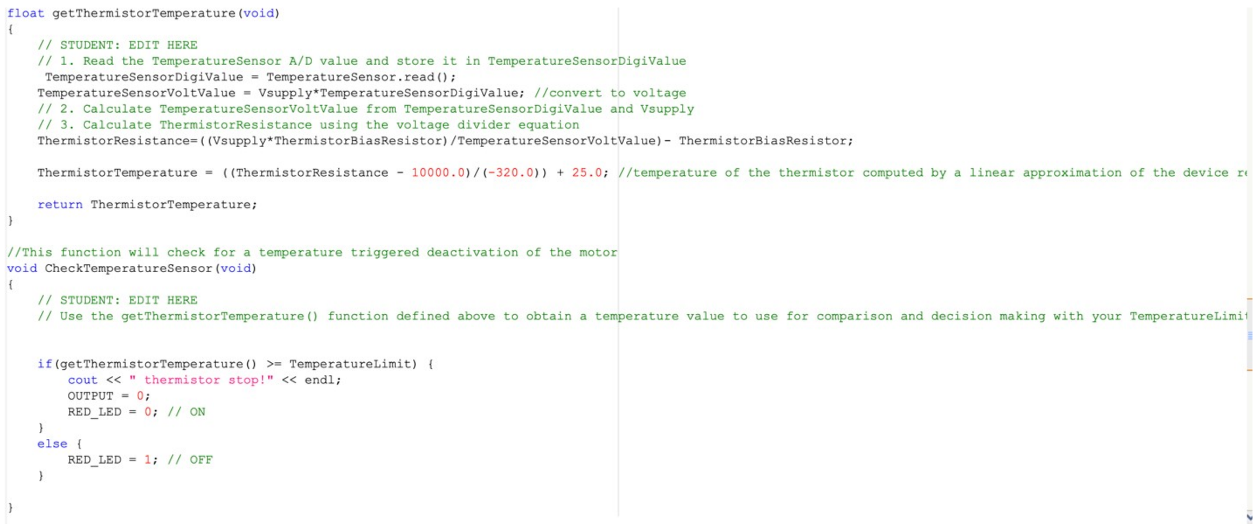
The get photoresistance function is used to change voltage from the LDR to resistance. The resistance is found by multiplying the volt value times the bias resistor then dividing that by the voltage supply minus the same volt value. The checklightsensor function is used to get the analog input and is defined in figure 8.



```
//convert the input voltage from the light sensor to an LDR resistance value
//Resistance is inversely proportional to the amount of light

float getPhotoResistance(void)
{
    LightSensorDigiValue = LightSensor.read(); //read the LightSensor A/D value
    LightSensorVoltValue = Vsupply*LightSensorDigiValue; //convert to voltage
    LdrResistance = LightSensorVoltValue*LdrBiasResistor/(Vsupply - LightSensorVoltValue); //voltage divider equation to determine LDR resistance

    return LdrResistance;
}

// This function will check the LDR analog input.
// STUDENT: USE THIS AS AN EXAMPLE FOR THE TEMPERATURE AND TORQUE CHECK FUNCTIONS
void CheckLightSensor(void)
{
    if(getPhotoResistance() >= LightResistanceLimit) {
        cout << "LDR Start!" << endl;
        OUTPUT = 1;
        GREEN_LED = 0; // ON
    }
    else {
        GREEN_LED = 1; // OFF
    }
}
```

Figure 8

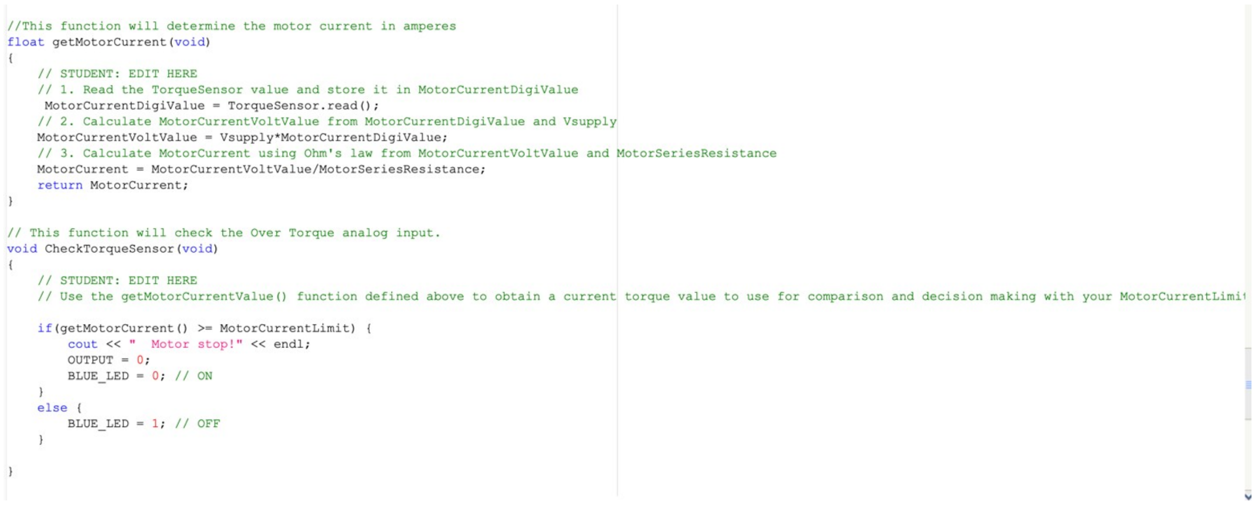
A function for getting thermistor temperature is created to convert the voltage to a Celsius value. It starts by reading in the temperature sensor data and multiplying it by the 3.3 voltage supply to get the volt value. Using the voltage divider equation, the thermistor resistance is found by multiplying the bias resistor by the voltage supply and dividing the values by the volt value of the temperature sensor. The thermistor resistor value is subtracted from this equation. The entire temperature of the thermistor is found by subtracting the 10k resistor from the thermistor resistance and dividing it by -320, then adding 25. This is illustrated in figure 9.



```
float getThermistorTemperature(void)
{
    // STUDENT: EDIT HERE
    // 1. Read the TemperatureSensor A/D value and store it in TemperatureSensorDigiValue
     TemperatureSensorDigiValue = TemperatureSensor.read();
    TemperatureSensorVoltValue = Vsupply*TemperatureSensorDigiValue; //convert to voltage
    // 2. Calculate TemperatureSensorVoltValue from TemperatureSensorDigiValue and Vsupply
    // 3. Calculate ThermistorResistance using the voltage divider equation
    ThermistorResistance=((Vsupply*ThermistorBiasResistor)/TemperatureSensorVoltValue)- ThermistorBiasResistor;

    ThermistorTemperature = ((ThermistorResistance - 10000.0)/(-320.0)) + 25.0; //temperature of the thermistor computed by a linear approximation of the device r

    return ThermistorTemperature;
}

//This function will check for a temperature triggered deactivation of the motor
void CheckTemperatureSensor(void)
{
    // STUDENT: EDIT HERE
    // Use the getThermistorTemperature() function defined above to obtain a temperature value to use for comparison and decision making with your TemperatureLimi


    if(getThermistorTemperature() >= TemperatureLimit) {
        cout << " thermistor stop!" << endl;
        OUTPUT = 0;
        RED_LED = 0; // ON
    }
    else {
        RED_LED = 1; // OFF
    }

}
```

Figure 9

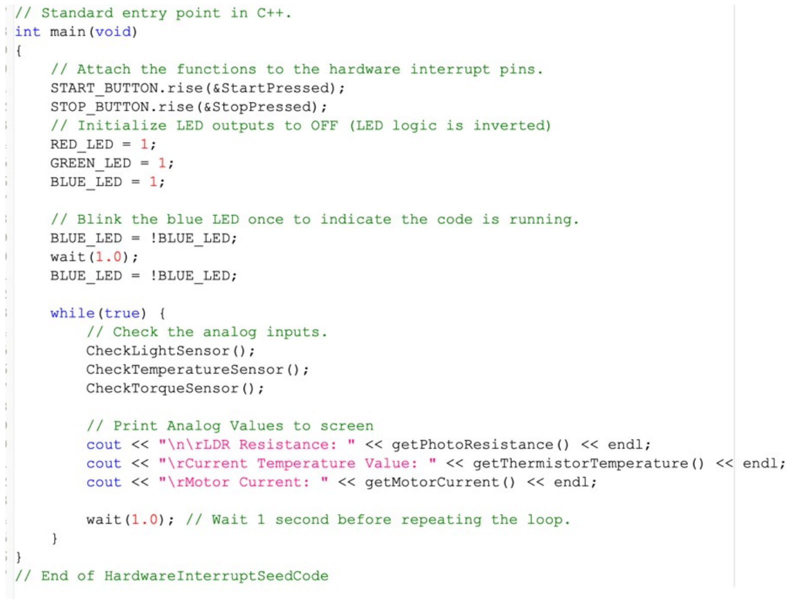
Next the motor current needs to be found by first reading in its digivalue and multiplying it by the voltage supply to get its volt value. To find the current the equation V=IR is used and arranged for I= V/R. the resistance was already defined earlier to be 10f, so the calculated voltvalue is divided by this to get the current. There also needs to be a torque function to check for excess torque defined as checktorquesensor. An if statement is used to compare the running value of the motor current to the set current limit. If this limit of .1 amperes is exceeded, then a blue led will turn on and the output to the motor will be stopped. These functions can be shown in figure 10.



```
//This function will determine the motor current in amperes
float getMotorCurrent(void)
{
    // STUDENT: EDIT HERE
    // 1. Read the TorqueSensor value and store it in MotorCurrentDigiValue
     MotorCurrentDigiValue = TorqueSensor.read();
    // 2. Calculate MotorCurrentVoltValue from MotorCurrentDigiValue and Vsupply
    MotorCurrentVoltValue = Vsupply*MotorCurrentDigiValue;
    // 3. Calculate MotorCurrent using Ohm's law from MotorCurrentVoltValue and MotorSeriesResistance
    MotorCurrent = MotorCurrentVoltValue/MotorSeriesResistance;
    return MotorCurrent;
}

// This function will check the Over Torque analog input.
void CheckTorqueSensor(void)
{
    // STUDENT: EDIT HERE
    // Use the getMotorCurrentValue() function defined above to obtain a current torque value to use for comparison and decision making with your MotorCurrentLimi

    if(getMotorCurrent() >= MotorCurrentLimit) {
        cout << "  Motor stop!" << endl;
        OUTPUT = 0;
        BLUE_LED = 0; // ON
    }
    else {
        BLUE_LED = 1; // OFF
    }

}
```

Figure 10

The last code component involves using a main program function that initializes the start stop buttons, and the state of the led's. As the program is running the analog outputs are printed out continuously every second as defined in figure 11.



```
// Standard entry point in C++.
int main(void)
{
    // Attach the functions to the hardware interrupt pins.
    START_BUTTON.rise(&StartPressed);
    STOP_BUTTON.rise(&StopPressed);
    // Initialize LED outputs to OFF (LED logic is inverted)
    RED_LED = 1;
    GREEN_LED = 1;
    BLUE_LED = 1;

    // Blink the blue LED once to indicate the code is running.
    BLUE_LED = !BLUE_LED;
    wait(1.0);
    BLUE_LED = !BLUE_LED;

    while(true) {
        // Check the analog inputs.
        CheckLightSensor();
        CheckTemperatureSensor();
        CheckTorqueSensor();

        // Print Analog Values to screen
        cout << "\n\rLDR Resistance: " << getPhotoResistance() << endl;
        cout << "\rCurrent Temperature Value: " << getThermistorTemperature() << endl;
        cout << "\rMotor Current: " << getMotorCurrent() << endl;

        wait(1.0); // Wait 1 second before repeating the loop.
    }
}
// End of HardwareInterruptSeedCode
```

Figure 11

DESCRIPTION OF OPERATION

To operate the program either the start button or the ldr is shaded. When the ldr is shaded this increases the voltage and causes the circuit motor to turn on. As either the torque of the motor is increased, or the thermistor is heated long enough the power will turn off. Three important measurements are taken to find time delays of the start button the stop button and the LDR. The digital output of the microcontroller is compared to all of these components to find the time delay. Figure 1 shows the oscilloscope setup and figure 12 shows the time delay, the same process is shown in figure 13 for the oscilloscope setup with figure 14 being the time delay reading. The final 2 figures 15 and 16 are used to show the set up for the LDR and the resistor, along with their time delay.

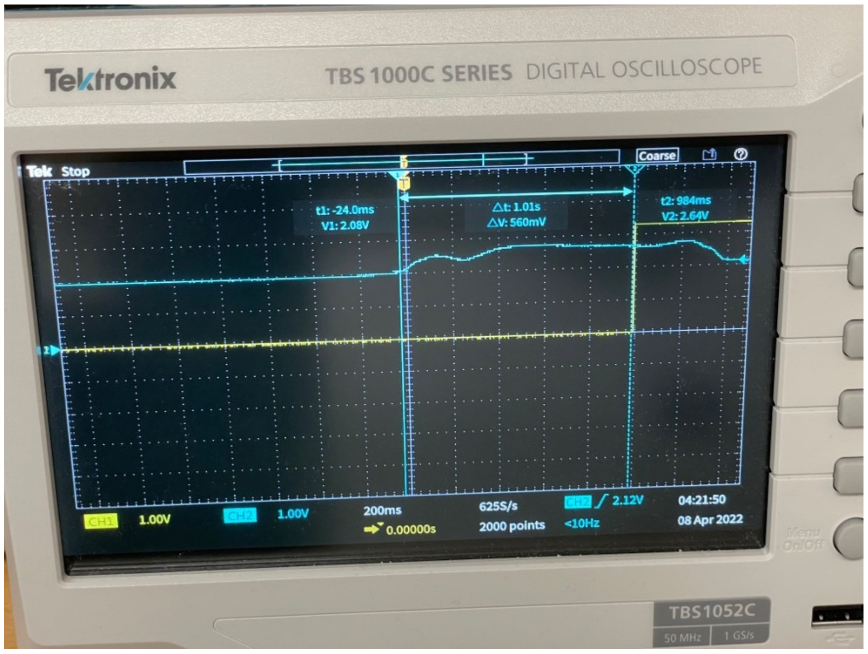


Figure 16

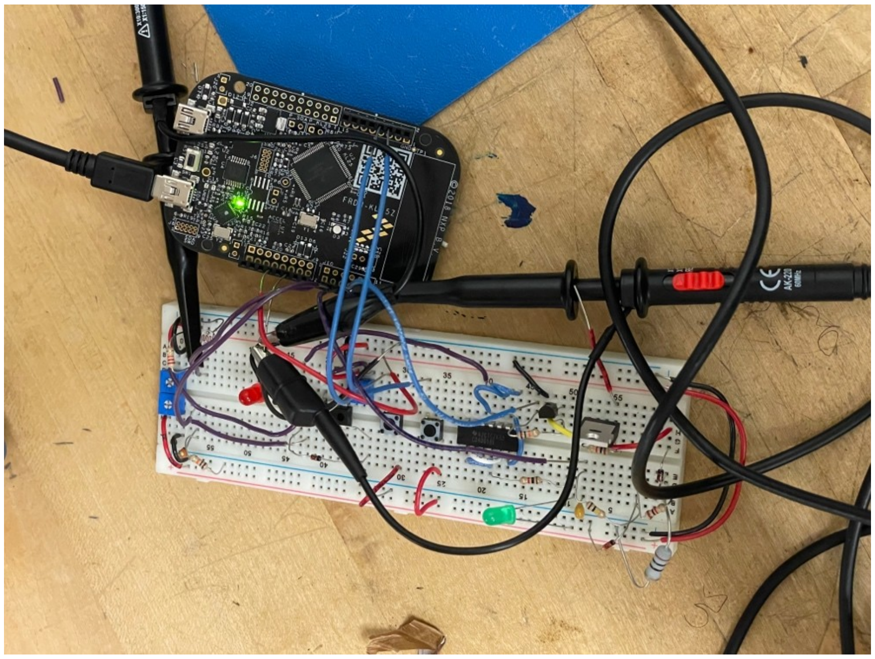


Figure 15

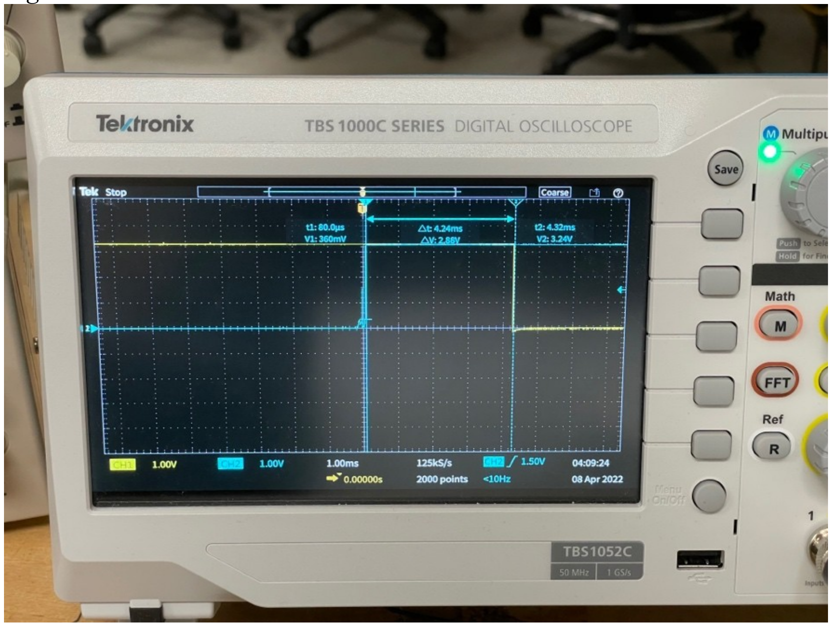


figure 14

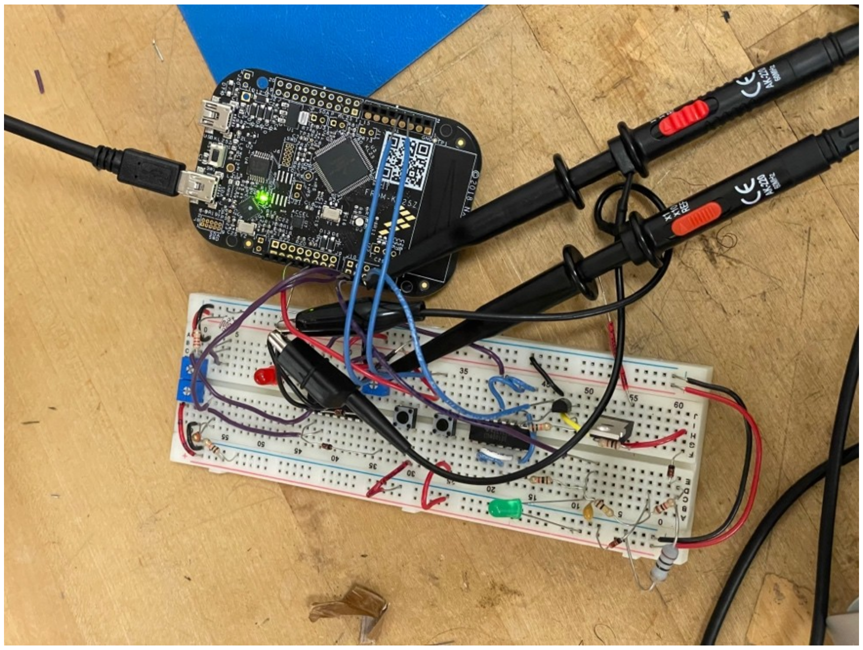


Figure 13

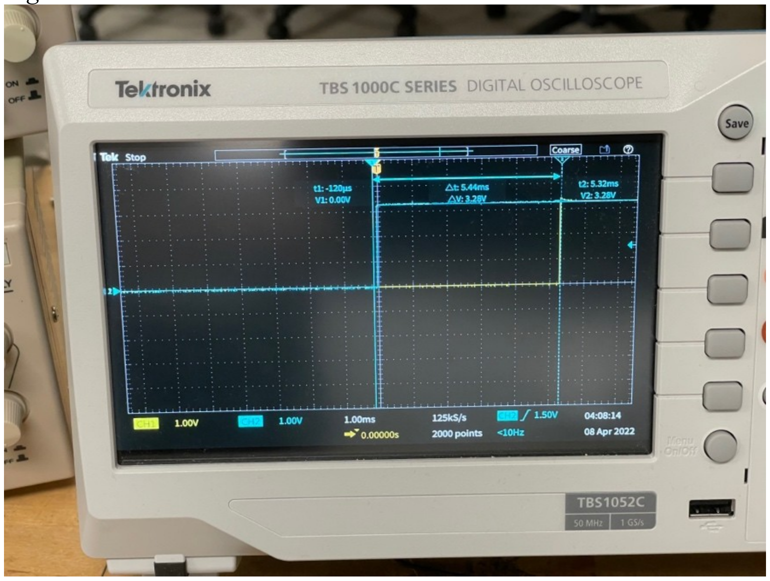


Figure 12