**3. Actuator**

**3.1 Methodology: H-Bridge**

(questions to be considered: which transistors to pick, comparison with PWM method)

H-Bridge is used in the actuator circuit in the system. An H-Bridge can be used to drive TEC, which is represented as RL in Figure 3.1, bidirectionally. When A, D are closed, the current will flow from left to right across TEC, while C and B are closed, the current will flow from right to left across TEC. Based on the testings on TEC in week 3, it is known that changing the direction of the current through TEC allows to achieve either heating up or cooling down it.

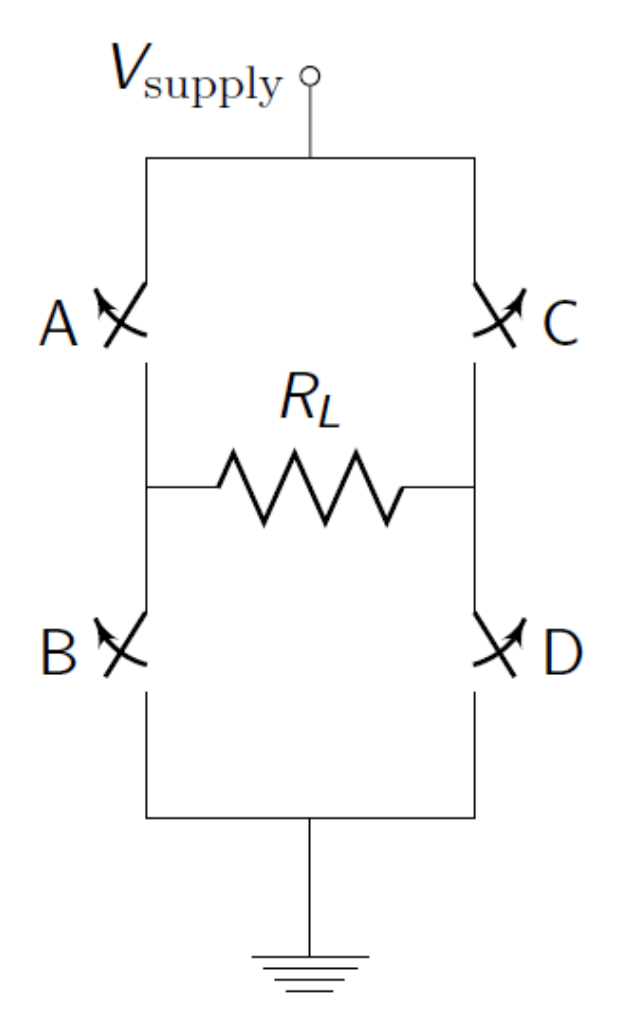


Figure 3.1 Basic structure of a H-bridge

At the beginning stage of designing H-Bridge circuit, four NPN transistors were used as A,B,C,D switches in Figure 3.1. We were provided by three kinds of NPN transistors: TIP41C, MJE200 and BC548B. The collectors (C) of transistors of A and C are connected to the power supply and their emitters (E) are connected to collectors (C) of B and D. The emitters (E) of B and D are connected to a common ground. One input voltage controlling the status of A and D and the other one controlling the status of C and B. They are connected to resistors, which are concatenated with the bases of these transistors (as what is shown in Figure 3.2). Based on previous workshops, it is known we need around 1A to heat up or cooling down TEC fluently. According to specifications of these NPN transistors, BC548B is not powerful enough to drive TEC because the maximum collector current it can have is 100 mA. The other two should work fine for the circuit.

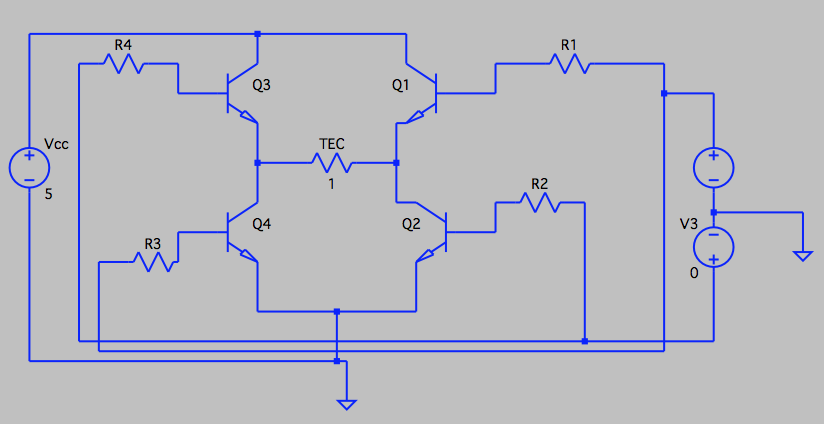


Figure 3.2 H-Bridge with four NPN transistors

Therefore, A,B,C and D are replaced by TIP41Cs. Varying the resistance of the four resistors connected to the bases of the transistors from 500 Ohm to 100 Ohm, with one input voltage turned on with 5V, we got the following result from the simulation.

| Resistance (Ohms) | I (TEC) | I (Arduino) |
| --- | --- | --- |
| 100 | 1.499A | 65.4mA |
| 200 | 1.081A | 35.5mA |
| 300 | 871.5mA | 24.58mA |
| 400 | 736.9mA | 18.87mA |
| 500 | 641.7mA | 15.342mA |

Table 3.1 A summary of the current across TEC and Arduino with different size of resistors connecting to transistors bases.

| Resistance(Ohm) | I(TEC) | I(Arduino) |
| --- | --- | --- |
| 10k | 1.3478776A | 458.55777µA |
| 15k | 1.1843177A | 321.26353µA |
| 18k | 1.110014A | 273.64725µA |
| 22k | 1.0284261A | 229.24428µA |

According to the simulation result, when resistance is 200 ohms the current across TEC is around 1A. However, the current through Arduino is 35.5mA, which is larger than the Arduino’s current limit 20mA. Therefore, the resistance should be between 300 ohms and 400 ohms. 380 ohm resistor is chosen as the resistor in this circuit. In this case, only 760mA current will running through TEC. After connecting the circuit to a breadboard, and using a dummy load (1 ohm resistor) as the TEC, the measured voltage across the dummy load (which is equal to the current across the load) varied from 0.5V to 0.7V. Consider we might use PWM (analogWrite() function from Arduino), it would be better that the circuit can generate current larger than this value so that we can have a wider range of current available for driving TEC. Therefore Darlington structure is introduced to further amplify the current so that we can have enough current and not exceeding the current limit of Arduino. BC548B is chosen for T1 (in Figure 3.3) and

Figure 3.3 Darlington Structure

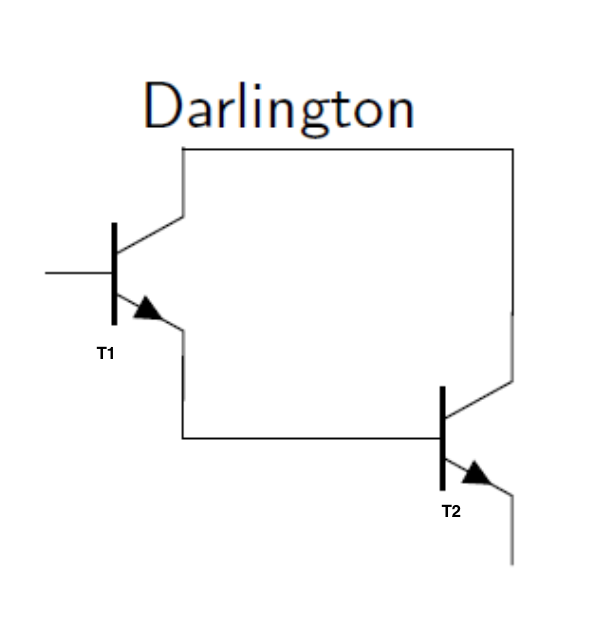


Table 3.2 A summary of the current across TEC and Arduino with different size of resistors connecting to transistors bases.

TIP41C is chosen for T2. This time, we do not need to worry about that BC548B is not powerful enough since TIP41C following it will amplify the current with a forward current gain around 75 (refer to the specification of TIP41C). The resistor need to be increased or the current across TEC will be too large and damage TEC. Therefore, varying the resistance value among the resistors that are available to us (10k, 15k, 18k, 22k), simulation result are shown in the Table 3.2. Based on the data achieved from previous testing on H-bridge without Darlington transistors, there was always a small discrepancy between the actual result and that from simulation.Therefore we decided to use 10k resistors connecting to the bases of the NPN transistors in the circuit.

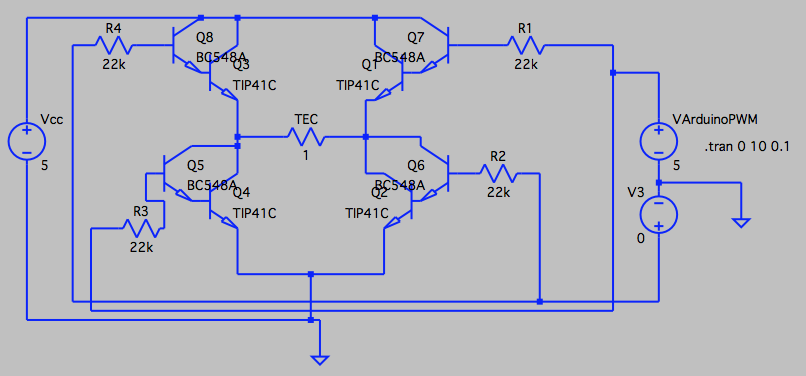
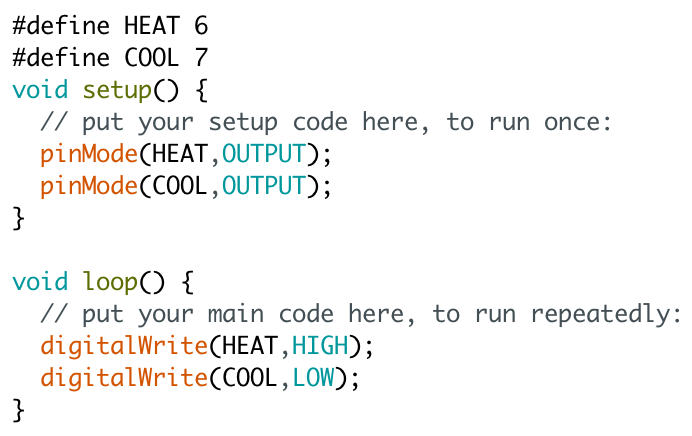


Figure 3.6 Final design for H-Bridge

**3.2 Result**

Two sets of tests were done: one run of testing was done on a breadboard and the other sets of testing was done after all the components were soldered to a veroboard. Same as the testing done before, a 1Ohm dummy load was used as a substitution of TEC.

The results from these testing were consistent with the result from the LTSpice simulation. There was 1.3V voltage drop across the dummy load when one input voltage was turned on with 5V (from the desk bench DC power supply). By alternatively turning on the two input voltages, a change in direction of the voltage drop could be observed.

After tests on breadboard and veroboard were done, we connected the H-Bridge to TEC and let Arduino’s digital output become the voltage inputs. A simple test program for powering the H-Bridge was written. It took around 30 seconds to heat up the TEC from room temperature (25 degrees) to 45 degrees while it took around 2 minutes to cool down from 45 degrees to 5 degrees. In general, heating up TEC took less time than cooling down it. The voltage across two ports of TEC varies from 1.4V to 1.7V, which is larger than the simulation result (across the dummy load). This might be due to the internal resistance of TEC is greater than 1Ohm.

Figure 3.7 A simple program for heating up TEC

The power dissipated in the system is estimated to be 5V \* 1.3A = 6.5

**3.3 Discussion**

Limitations of this H-Bridge circuit:

| A | B | C | D |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |

Since there are two inputs to this H-Bridge, there will be totally four combinations of inputs.

Table 3.3 All four possible combinations of inputs to H-Bridge



Figure 3.7 (a)(b) segments of code in the controller program for heating up and cooling down

The first combination will be dangerous because it will create a short circuit and damage the components. Our way of preventing this short circuit happening is to program the controller so that only one input will be turned on each time (as what is shown on the left). For example, when current temperature is lower than the set temperature, the input for heating TEC is turned on, and the other one is off. When temperature goes over the target temperature, instead of turning on the cooling input, both inputs are turned off, considering the time between Arduino executing the if statement and turning on the other input will result in a short period of short circuit result in malfunction of the H-Bridge.

Comparison with other choice of actuator circuit (Push-Pull amplifier):

Another common method for driving a load bidirectionally is the push-pull amplifier.

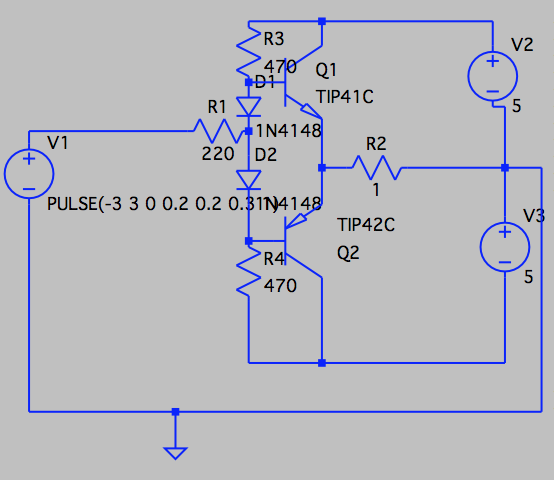


Figure 3.8 Push-pull amplifier

In this design, Q1(NPN) “pushes” current from V2 through TEC to GND, while Q2(PNP) “pulls” current from GND through TEC to V3. V1 is the control signal from Arduino. R2 is TEC. The diodes were added to cancel the crossover distortion.

Comparing an H-Bridge actuator and a push-pull amplifier one, there are a few reasons for us to choose H-Bridge as our driving circuit.

1. Arduino can only generate 0V or 5V. Even with PWM pins and analogRead() function, it can only generate positive voltages. Bipolar control signal can be achieved by adding an op-amp to shift the range of the signal. However, H-Bridge does not have this problem.
2. H-Bridge can be easily driven by PWM pins while a LPF (low pass filter) circuit is probably required to convert the square wave to a DC voltage.

In conclusion, H-Bridge uses less components than push-pull amplifier and it is more easily driven by a micro-controller (Arduino).

**4. Control Signal**

**4.1 Methodology: Bang-Bang Control**

Sensor

(Thermistor)

User Input

(Potentiometer)

Microcontroller

(Arduino)

if absolute difference is greater than 1

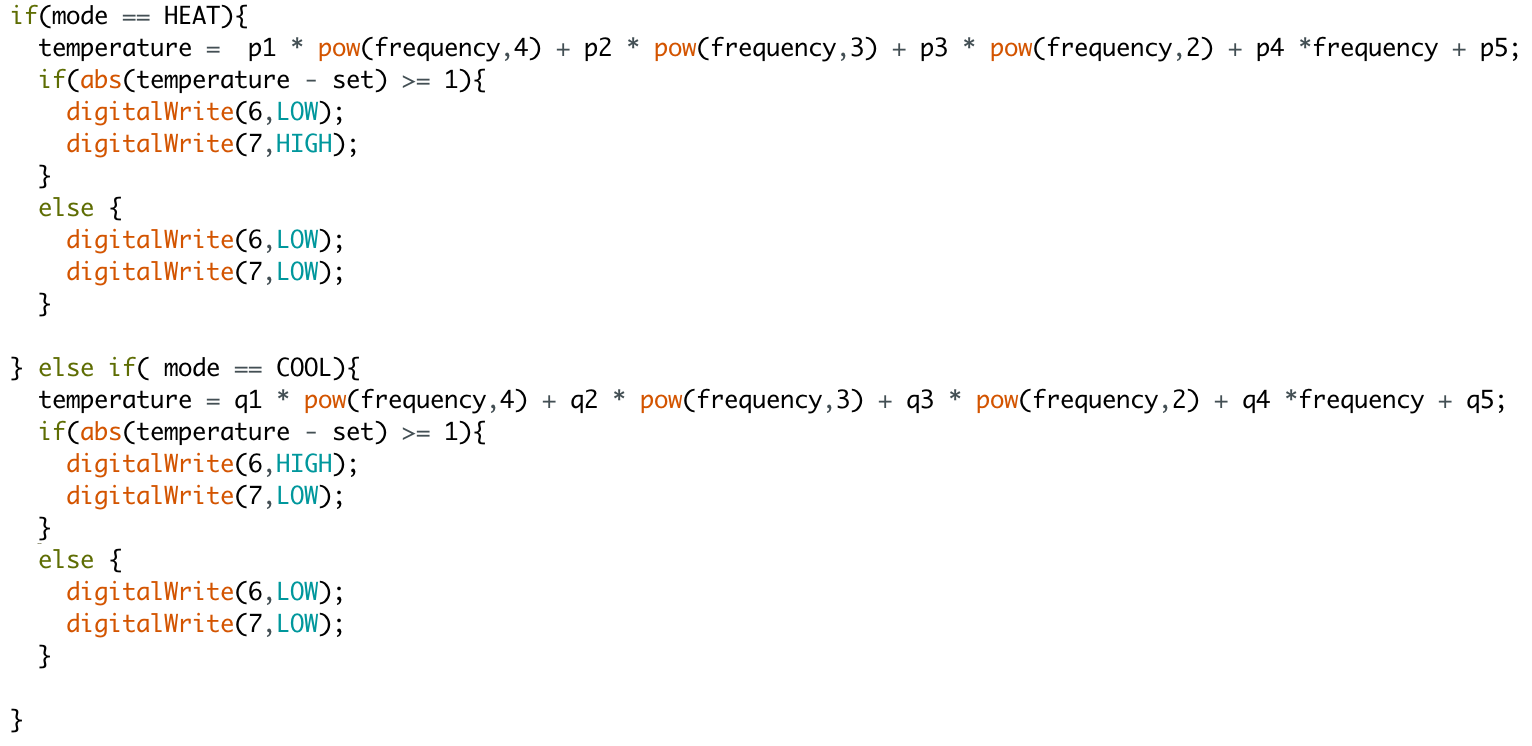
TEC

Heating up or cooling down

Figure 4.1 A block diagram showing the control system

Bang Bang control was used in our system. The principle behind this control is really simple. when the difference between the set value and the real measured temperature is greater than zero, one of the output pins will be turned on depending on which mode (heating or cooling) the controller is in. Since we have two different calibration curves for cooling TEC and heating TEC, a push button was set in the circuit for the user to set the mode of the controller. When a temperature higher than the real temperature was set, the “HEAT” mode should be selected as well so that controller can follow the heating curve to heat up TEC.

Figure 4.2 A segment of code showing the controlling system



**4.2 Result**

Setting the target temperature to be 45 degree at room temperature (26 degree), the result is shown in the graph below.

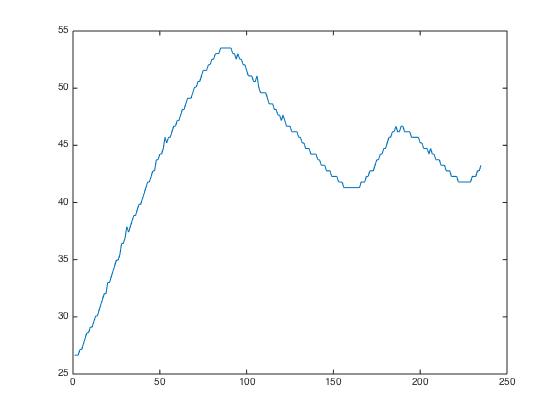


Figure 4.3 time vs temperature heating from 26 degree to 45 degree

This graph is based on the temperature measure every half second. Therefore the time step of the x-axis is half second. According to the graph, it took around 25 seconds (50 \* 0.5 second) to heat the TEC from 26 degree to 50 degree. However, it overshoots to 53 degree afterwards. Because bang bang control always drives the circuit with full power, it is easy for the system to overshoot the target temperature. It took more than two minutes (250 \* 0.5 second = 125 seconds) for the system to settle down to the target temperature.

**4.3 Discussion**

Limitations of Bang Bang control:

As Bang Bang control always drive the circuit with full power, it consumes much power than using PWM with PID control. Also, this issue brought us another problem that the system always overshoots the target and takes longer time to settle down to the target temperature than expected. These problem can be solved by using PID control. Since we don’t have enough PWM pins on Arduino UNO (LCD display and frequency reading functions use all these PWM pins), we are not able to apply PID control on our system. In Figure 4.4, there is a piece of code which is suppose to



Figure 4.4 A segment of code doing PID control

do PID control. By using millis() function to record the time elapsed and calculating the integration of errors, as well as the derivative of the error, the output’s value will be dependent on the amount of the error, the integration of errors and the derivative of the errors. Also, three parameters kp,ki,kd are introduced to control the weight of each of these values. By doing this, the circuit will not always driven by full power so that energy can be saved. Also, it can solve the problem of overshooting the target.