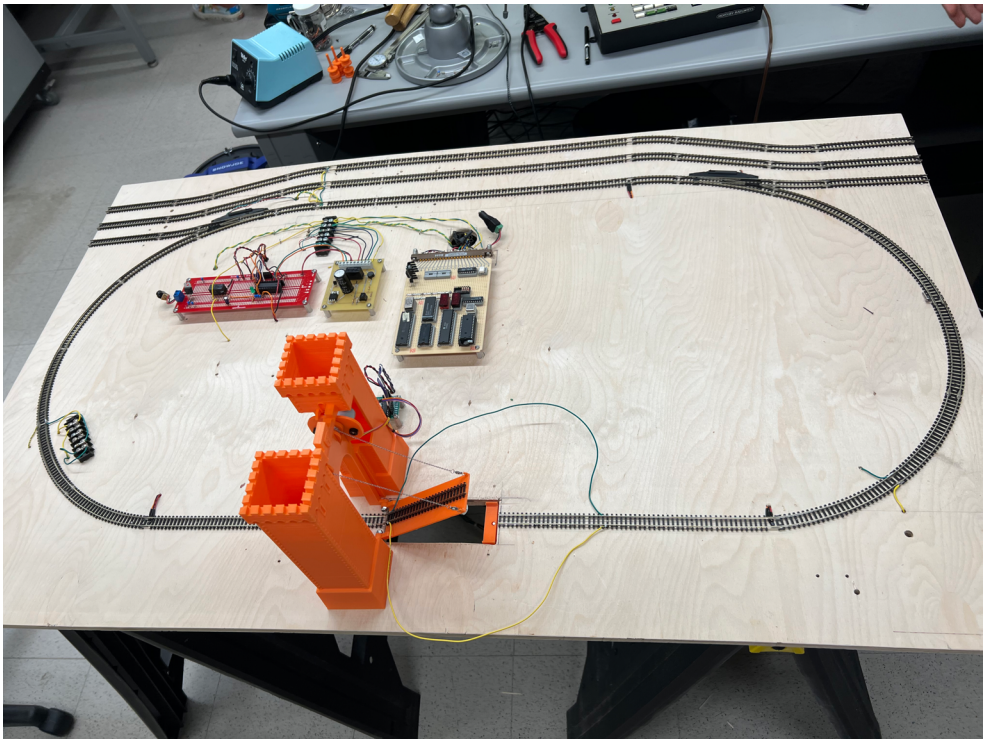
King's Castle

MAE 412 Group 8 Final Report

Jonathan Melkun, Justin Chae, Khoa Le



Department of Mechanical and Aerospace Engineering
Princeton University
5/15/2023

Contents

1 Introduction

As a final project for the MAE412 Microprocessors class, we have been tasked with creating a modular project board that interfaces with an existing model train test stand. On this project board, we are expected to create a sequence of events using model trains that involves some form of sensing and actuation. The metrics to measure success include robustness, hands-free, reliable operation, workmanship, documentation, project progress, and degree of difficulty/effort. The timeline for the final project includes 6 weeks of in-class lab time and 2 weeks of reading period concluded by a day of final projects demonstrations to the course instructors and fellow classmates.

2 Hornby Control

2.1 Function

The Hornby Zero 1 is a command control system to allow communication and control of locomotives along a track. Besides giving commands to specific trains to move, the Hornby Zero 1 also allowed for a train's direction, speed, and inertia to be controlled as well. The Horby system communicates with the trains by sending a square wave voltage to the tracks. Encoded within this square wave is a sequence of bits which is then read and decoded by the trains to execute the specific commands given by the control system.

2.2 Vector Board

In order to communicate with the Horby control system, we created a separate vector board computer, shown in Figure 2. The board consisted of a 6502 MPU as the main microprocessor with a 28C64B EEPROM, 6522 VIA, 6551 ACIA, 6116 RAM, and several other auxiliary parts such as a TIL 311 hex display, 1 MHz crystal oscillator, and a 16V8 GAL for our address decode logic. The ADL implemented on the GAL chip can be found in Figure 1, with the full diagram found in Appendix 18. The entire vector board circuit diagram can be found in Figure 3.

Once our vector board was completed, we were able to run the stock Echo 3.1 program taken from the course website and flashed onto our EEPROM. Once the chip was programmed, block control of the test stands and project board were enabled to allow for simultaneous control of multiple trains.

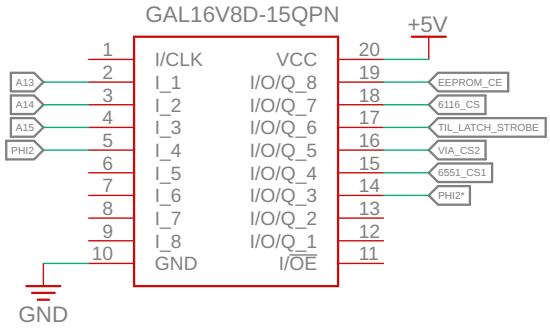


Figure 1: Diagram of the ADL from the GAL.

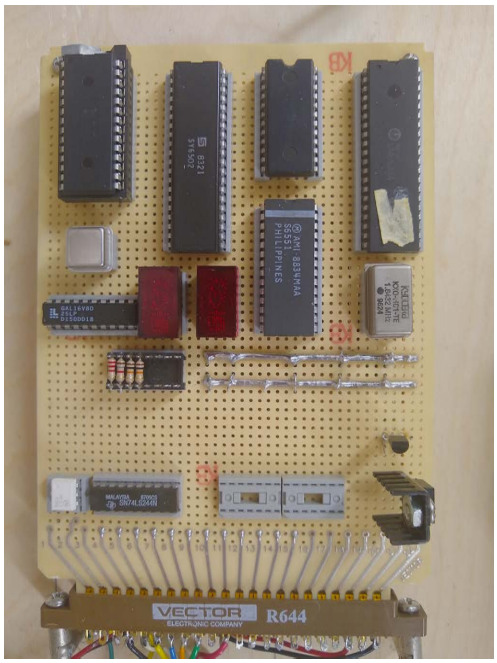


Figure 2: Photo of the chip layout on the vector board.

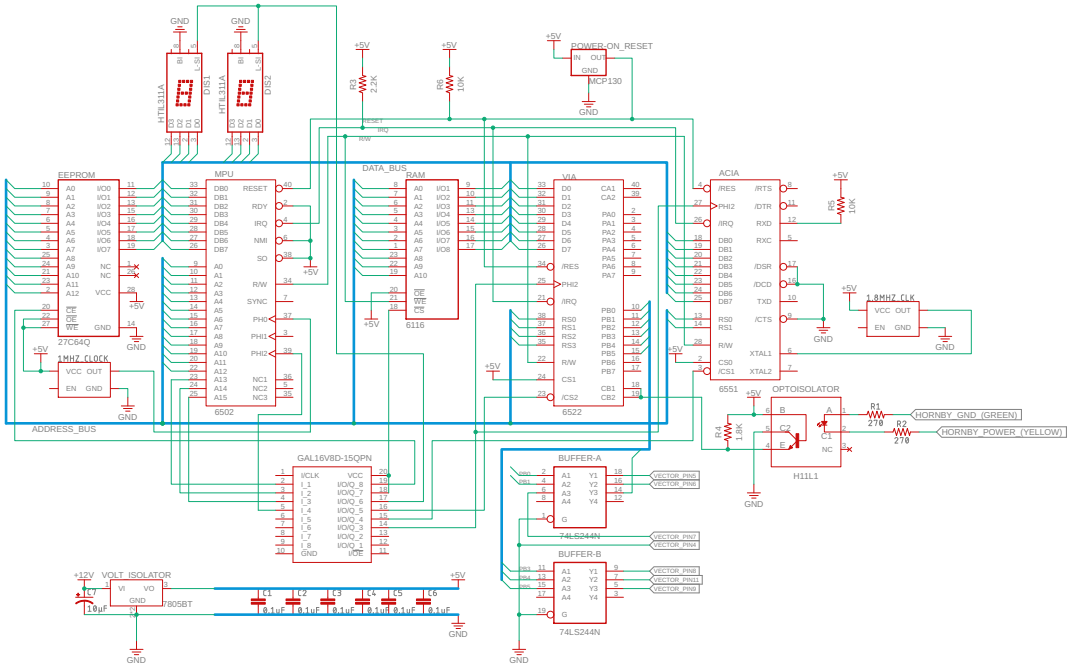


Figure 3: Wiring schematic of the vector board components.

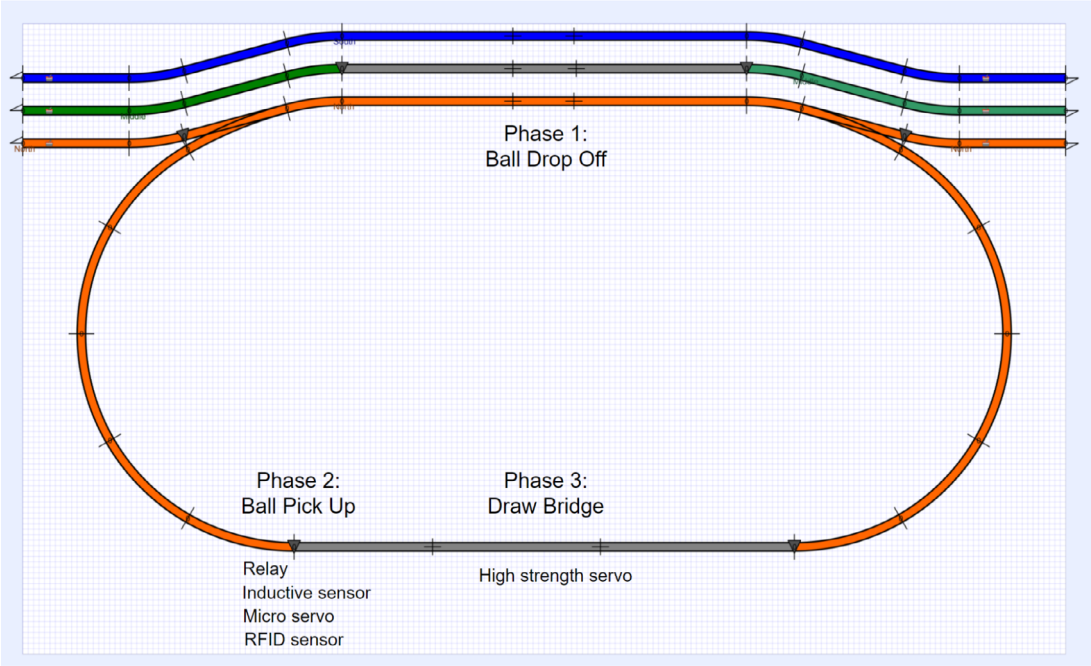


Figure 4: Preliminary layout of the track.

3 Planning

Originally, our project planned to have an elliptical shape track where a train enters from the right side on the North track, goes around the loop, and exits from the left side. To ensure our track layout works for the given shape and that we have enough track parts to complete the loop, we planned out the track layout using Anyrail. For sensing, as the train reaches the bottom of the loop, an RFID reader would detect if the train is present and a ball would be dropped into the train. Additionally, an inductive sensor would detect if the ball successfully dropped onto the train. Once a ball is detected, a servo will lower a drawbridge to connect the tracks and the train will continue along the loop. Once the train reaches the top of the loop, another RFID sensor would detect the train, switch the left turnout, and a passive mechanism would cause the train to deposit the ball off the train and the train would continue to the left board. This layout is as show in Figure 4. However, as we worked on the project, our design choices changed to increase robustness and consistency. At first, we employed an ultrasonic sensor to detect when the train would be close enough to cut track power, load the ball, and lower the drawbridge. However, we found that this sensor was not accurate enough to consistently stop the train in the correct position. We also tested a reflective sensor, but we also found this to be less consistent because of lighting conditions and because the sensor

was more complicated to implement for our board. In the end we settled on hall effect sensors to change track turnout, stop the train, lower the bridge, and raise the bridge.

We also created a ball loader and dumper mechanism as pictured in Figure 5 but decided not to use it because it would have been another point of possible mechanical failure for the system. Additionally, the loader would have been activated by the same sensor as the drawbridge so it would not have added much to our sensing and actuation intricacy.

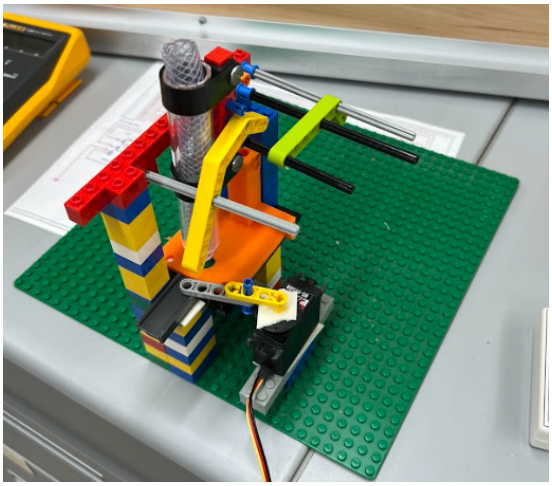


Figure 5: Ball loader mechanism powered by a servo motor.

In planning the layout of the board, we also elected to use the elliptical shape for consistency and simplicity. We drilled a hole in the board for our drawbridge instead of raising the track up and having an elevated bridge so there would not be problems dealing with gravity from the slope of the tracks as we stop and release the train. The drawbridge itself was designed in Creo and then 3D printed. We tested multiple iterations of the drawbridge before finding one that would cleanly connect the two tracks and sit flush against the board.

3.1 Project Board Construction

The board was assembled according to the same layout previously pictured in Figure 4 except without the ball loader. The three tracks were aligned to fit exactly with the tracks on the test stand. 2.5 inches were measured from the top of the board to the first track, and a jig was used to accurately space the other two tracks from the first one. Upon testing the alignment, one will see that the test stand tracks are at a slightly different elevation than the tracks from our board because of the board's curvature. To solve this problem, we use extra clamps in the center of the board and metal pieces to

Track		
2500, N Atlas Code 80, Flex 30".	4	(Minimum number of units: 3)
2510, N Atlas Code 80, Curve radius 9 13/16", angle 30º	10	
2511, N Atlas Code 80, Curve radius 9 13/16", angle 15º	12	
2513, N Atlas Code 80, Straight 4 7/8". (bulk)	10	
2700, N Atlas Code 80, Left turnout 4 7/8". 15º (remote)	1	
2701, N Atlas Code 80, Right turnout 4 7/8". 15º (remote)	1	
Track lengths		
2500, N Atlas Code 80, Flex 30".	78 27/32	
2510, N Atlas Code 80, Curve radius 9 13/16", angle 30º	51 1/4	
2511, N Atlas Code 80, Curve radius 9 13/16", angle 15º	30 3/4	
2513, N Atlas Code 80, Straight 4 7/8". (bulk)	48 31/32	
2700, N Atlas Code 80, Left turnout 4 7/8". 15º (remote)	9 27/32	
2701, N Atlas Code 80, Right turnout 4 7/8". 15º (remote)	9 27/32	
Total track length:	19'-1 17/32"	

Figure 6: List of track pieces used.

prop up the board at the ends. As for the other track pieces, we used curved tracks, flex track, and turnouts according to the bill of track materials below in Figure 6.

We chose to use flex track to account for any misalignments and length differences between the tracks on the right and left side of the board. Because flex tracks are long, easy to cut, and can be angled, we used it for the drawbridge section as well as connecting the tracks on opposite sides of the board together without running into issues or gaps that would derail the train. The turnouts were also necessary for guiding the train onto the main loop. These turnouts were tested by touching their power and ground wires to a power supply and seeing if the track physically switches.

The tracks were set according to the layout before. Holes were then drilled periodically between the tracks and the board where our nails would be set. We first loosely nailed the tracks down halfway in these planned positions in case they needed to be shifted. Once satisfied, we used the nail setter to drive in the nail flush with the track so that it would not interfere with the train. Flex track was measured on the distance between the side tracks, cut using a saw, then connected to the tracks. At the bottom of our board, we also drilled and sawed a rectangular hole in the middle of the bottom flextrack through the wood for our drawbridge. The drawbridge was then designed in Creo to fit the dimensions of the rectangular hole and was installed with screws connecting it to the board, and chains connecting the bridge platform to the pulley. The vector board, trickle charger, daughterboard, and stepper motor driver are all mounted on standoffs and screwed into the board. Holes were drilled into the board for wire management. Wires were also stapled to the board and those over 6 inches were twisted using a drill according to specifications. The hall effect sensors were set in place under the tracks before the drawbridge, after the drawbridge, and before the left turnout. Special care was taken so that the wire leads on these sensors did not touch the conductive track.

3.2 List of Components

- 28BYJ Stepper Motor + ULN2003 motor driver - The stepper motor is mounted on the top right of the drawbridge and is used to raise and lower the drawbridge by spinning a pulley of chains. The motor driver powers this stepper motor and communicates with the arduino for when to raise and lower the drawbridge.

- DRV5013 Digital-Latch Hall Effect Sensor (04E) - 3 of these are used. Each one polls at a thousandth of a second and outputs a high signal to the arduino when the train magnet runs over it. They are used to activate the draw bridge raising, lowering, track power, and track turnout.

- G5V-2-H1 Relay - 2 of these relays are on our board. One of them is on the trickle charge and is used for turning out the track. The other is on our daughter board in a relay control circuit (in appendix) which is used for turning on and off track power to stop the train.

- Voltage regulator - The regulator converts the 12V AC input from the wall socket and regulates it to a 5V DC output which is used to power the Arduino microcontroller and provide voltage to the trickle charge circuit

- Tracks - Straight, curved, flex tracks, and turnouts as explained previously. Some needed to be cut to the right length. Others had to be cleaned for good conductivity.

- ATMega328p Arduino microcontroller - The main microcontroller for our project. The Arduino takes inputs from all 3 hall effect sensors and outputs signals to control the drawbridge, trickle charge, and track turnout.

- Trickle charge circuit - contains a relay which switches the track turnout. Power, ground, track power, track ground is wired to it. The trickle charge has 3 inputs for wires from the turnout. We have wired the red wire to N.C. as the track is turned most of the time. The arduino is connected to direction, which switches the relay, and trigger, which switches the track.

- Chains - These metal chains are connected between the pulley and the drawbridge platform. They work to pull up the drawbridge and to lower it when the stepper is activated.

- 3D printed parts - we have printed 3D parts mainly for the drawbridge. These parts were created in CAD, printed, and sanded down to fit our needs. The drawbridge print is pictured in the appendix.

4 Drawbridge Mechanism

In order to ensure that the components would fit on the board as it was intended, the entire drawbridge was modeled in CAD. This allowed us to fine tune the dimensions and tolerances of the part. The parts were all 3D printed, which allowed quick iterations as they took no more than a couple hours to print. The drawbridge can be thought of in three parts: the bridge itself that the track was attached to, the pulley mechanism that raises and lowers the bridge, and the support structure that everything is mounted to. A castle shell was also printed to enclose the mechanism and give it a more aesthetic appeal. Figure 7 shows a representation of the final model in CAD.

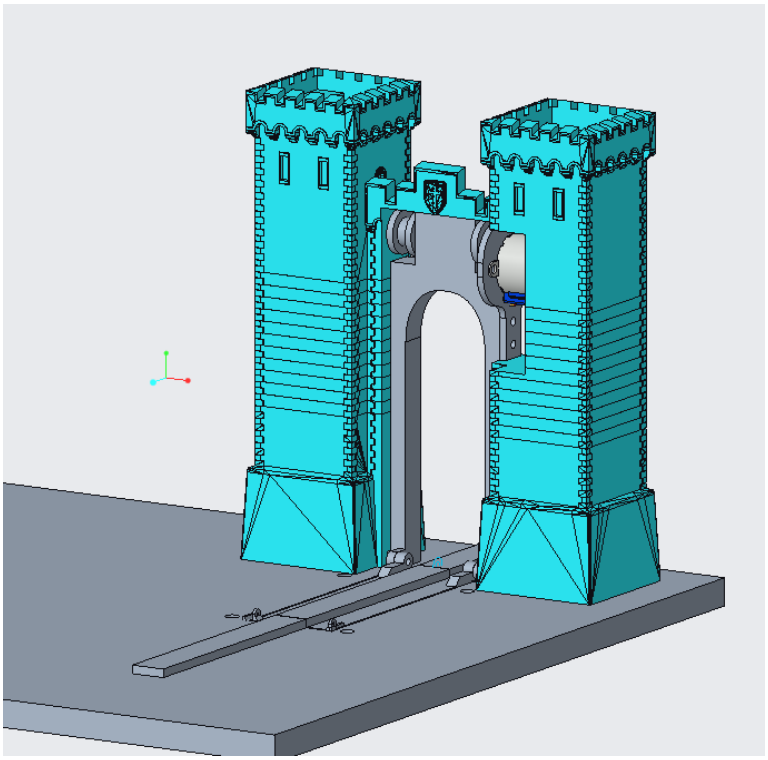


Figure 7: CAD model of the drawbridge.

4.1 Bridge Part

The bridge part is the part that the train drives over. When lowered, it covers the gap in the board and allows the train to cross. It is shown in Figure 8. A section of track was nailed to the bridge. The ends of the track were filed down to be slanted to allow for near-perfect interfacing with the other tracks as the bridge is lowered.

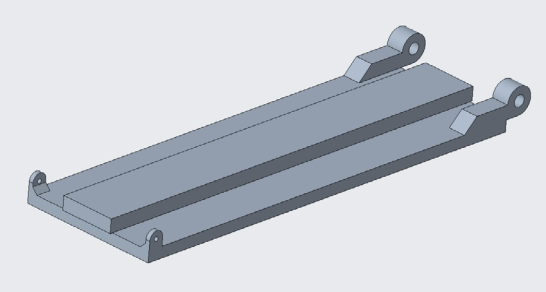


Figure 8: Drawbridge part. The rail is represented by the rectangle on top of the bridge.

4.2 Pulley

The pulley is a simple mechanism that allows for the rotational motion of the stepper motor to pull the end of the drawbridge up with chains. The chains are glued to the pulley and clipped on to the ends of the drawbridge with hooks that came with the chains. It is important to note that the pulley only provides force to raise the bridge. When the bridge is being lowered, the only thing pushing the bridge down is the force of gravity. As such, weights were added to the bottom of the bridge part to assist in the lowering of the bridge. A 1/4-20 screw was glued to the bottom to allow for adjustments by threading multiple nuts onto the screw until the desired results were acheived.

4.3 Mounting Geometry

To properly align the track and prevent the train from derailing, special mounting geometry was created to guide the bridge into its correct position. The bracket is shown in Figure 9. The bracket is screwed into the bottom of the project board by its two screw holes on the ends. The bracket features two sloped walls that act as a funnel that guide the lowering drawbridge into alignment. To further ensure that the drawbridge will be aligned properly, a magnet was glued onto the bridge part and the bracket. The hole for the magnet was incorporated into the model in Figure 9. This produces a satisfying snap that serves as a good indicator that the bridge has finished lowering and is ready for the train to cross.

4.4 Hinge

A crucial part of the drawbridge mechanism is the hinge mechanism. This allows the drawbridge to rotate up and down. However, early testing showed that this is a nontrivial task, as the placement of the center of rotation was an important aspect of the design. Early prototypes placed the center of rotation below the drawbridge, shown in Figure 10. This caused the tracks to intersect with each other as the drawbridge rotated. To

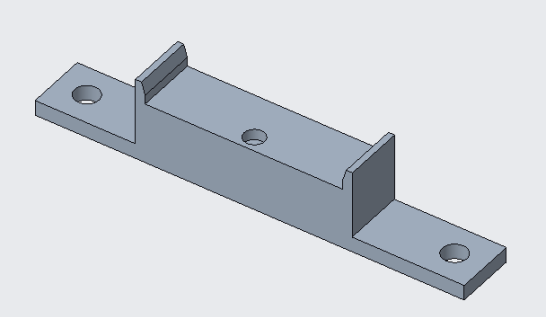


Figure 9: Mounting geometry.

avoid this, the center of rotation was placed above the drawbridge instead. This way, as the bridge rotates, the track on the bridge will be lifted up and over the other tracks. This motion can be seen in Figure 11.

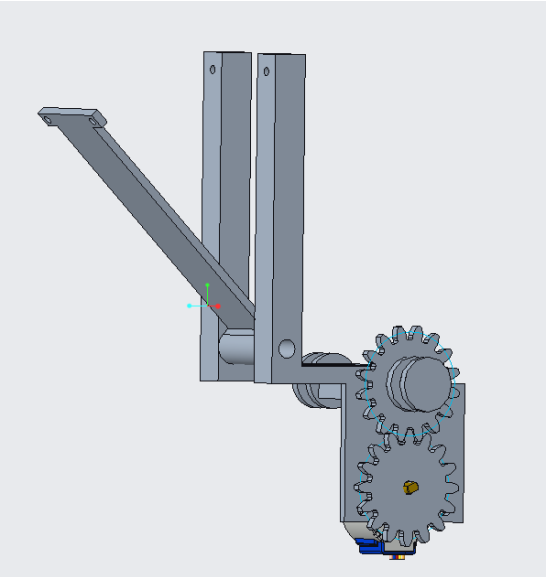


Figure 10: Early iteration of the drawbridge.

4.5 Final Model

After all parts were 3D printed, the components were secured to the project board with wood screws. The castle shell was made to be removable to allow for easier servicing of the parts. The model for the castle was based on an STL file from Thingiverse [1]. The model was modified in Tinkercad to make it taller, hollow it out to cut on material costs, and cut out rectangles to make room for the mechanism. The entire drawbridge mechanism physically assembled can be seen in Figure 12.

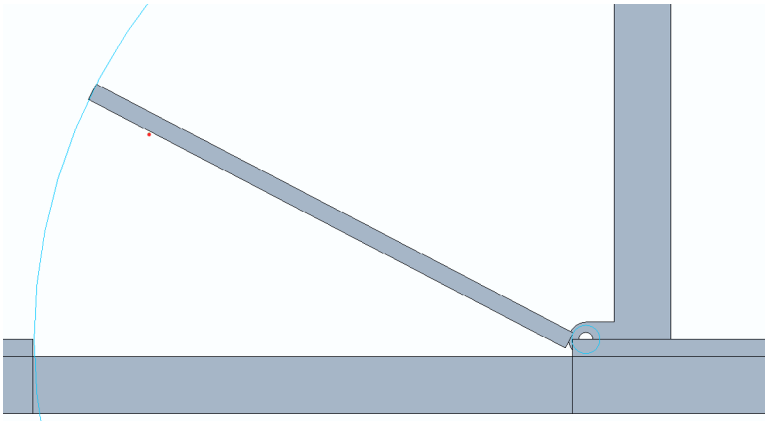


Figure 11: Image showing the rotation of the drawbridge.

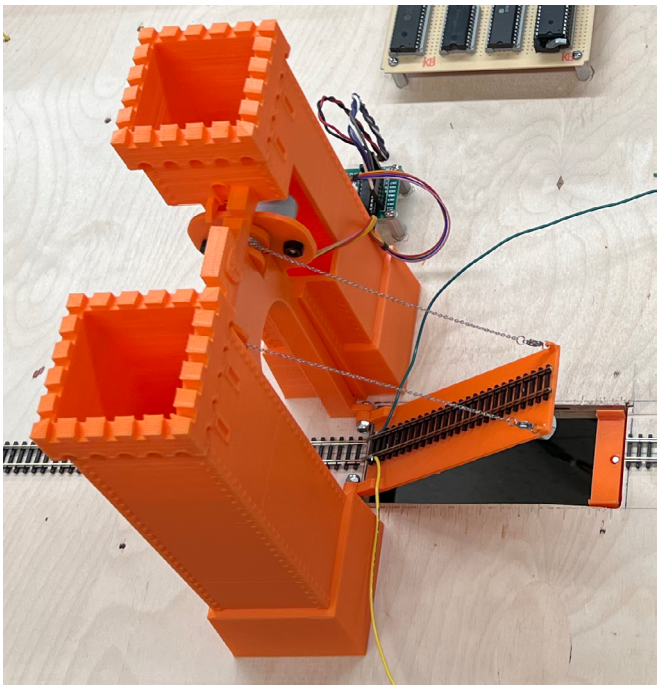


Figure 12: Image of the drawbridge on the project board.

5 Arduino Code

To simplify things, all of the sequencing was accomplished using the powerful Arduino microcontroller chip and its associated development tools. The only reason why one would need to interact with the vector board would be to perform interactive control, which could interface with the Hornby and reduce the speed of the train. This was a consideration early on in the project, but was scrapped because it was unnecessary. The train only needed to stop and start, which could be accomplished with a simple relay circuit that would cut power to the track, shown in Figure 13. Additionally, the sensors were connected to the Arduino, as well as the junction that allowed the train to enter and exit the track. Because of this, the entire Arduino sensing and sequencing circuit was wired on a separate daughter board, with no connections to the central computer driving the trains. The full electrical schematic can be seen in Figure 14.

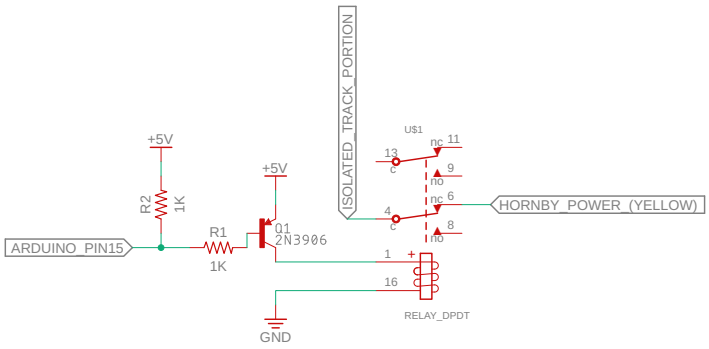


Figure 13: Schematic of relay circuit.

A general outline of the code can be found in Figure 15. The resting state of the Arduino is to check each of the hall effect sensor pins. In the diagram, these pins are 16, 17, and 18. If the Arduino does not read a low signal from these pins, it continues looping until it does. Once a train passes over a hall effect sensor, it produces a change in the magnetic field around the component, which is converted into a voltage reading that the Arduino can sense. When the magnet on the bottom of the train reaches a certain threshold distance, the output pin of the hall sensor goes from high to low. This is what the Arduino reads to tell that the train has passed over the sensor. Once the hall effect sensor goes low, when the Arduino polls this hall sensor in the next cycle it

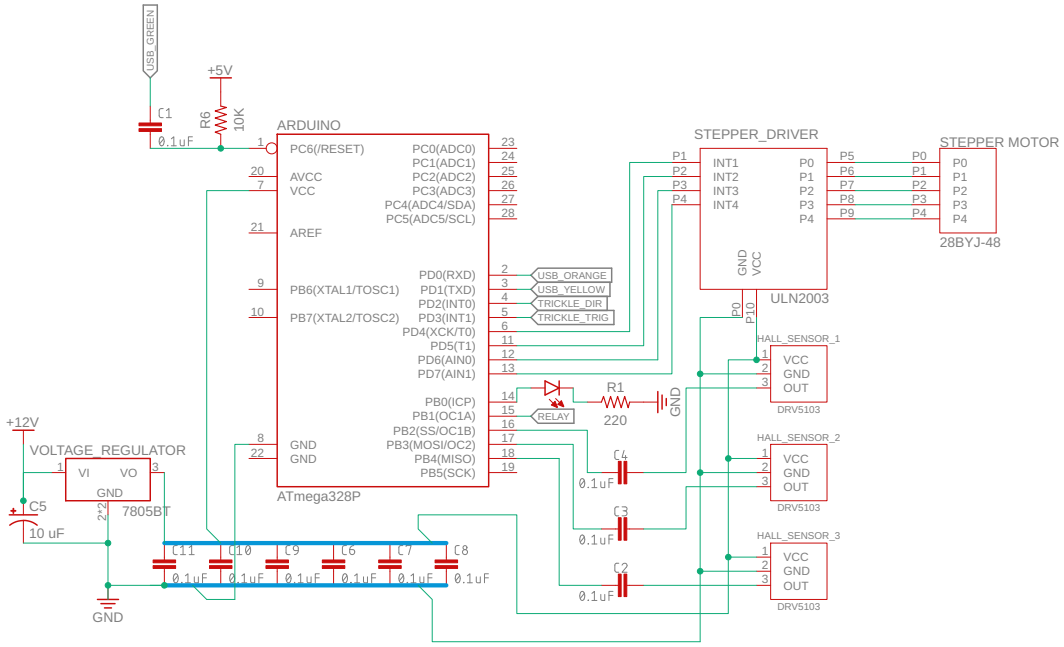


Figure 14: Electronics schematic of Arduino daughter board.

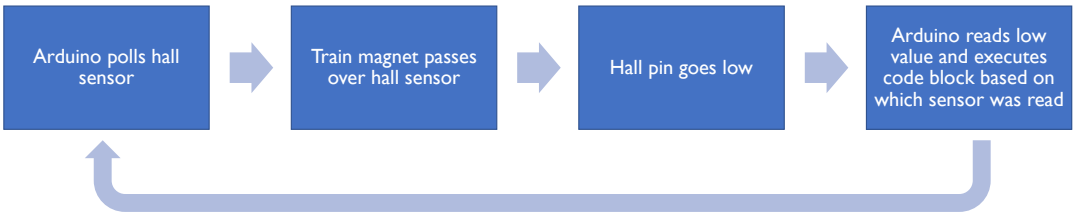


Figure 15: General outline of Arduino code.

immediately starts executing the corresponding code based on which hall sensor was tripped.

It is important to note that all of this happens simultaneously, as in the Arduino is still polling all three of the hall sensors while it is performing a task. There are no calls to the delay() function in the code. The delay function causes what is known as blocking, or when the processor hangs on a task before it can perform another task. This is acceptable for simple tasks like blinking an LED or only reading one hall sensor, but to build a more robust system blocking must be avoided. Since there were three hall effect sensors used, there is a risk that the Arduino would be busy performing one task, such as lowering/raising the drawbridge, while the train moves over one of the hall effect sensors. This would cause the Arduino to miss the reading because it was blocked by the raising bridge task, breaking the sequence. It is possible to design the system such that this never happens, but to account for unexpected possibilities the code was structured to use asynchronous timers instead of delays. This was implemented using the millisDelay library.

Timers were also used to prevent the sensors from being activated too many times. Ideally, we would want the train to only trigger the hall sensor once, then execute the code once. If there are multiple calls at once, it could overload the logic in the Arduino sequence or cause the Arduino to hang and miss a task. This would be especially troublesome before the drawbridge, where the Arduino must cut power to the track immediately. If it is late in this task, the train would move too far and collide with the bridge. Fortunately, as an additional layer of safety, the drawbridge in its up position prevents the train from falling off the project board, but this is a scenario that must be avoided if possible. As such, the hall effect sensors were limited to being triggered once every second, which was enough to only get one measurement reading out of the sensor. The Arduino is still polling the sensor constantly, but when it does receive a low value it starts an internal timer that prevents the code from being executed a second time in the next cycle of the processor. Once the timer is finished, the sensor is open to being activated again.

For visual reference, a diagram showing the locations of the hall sensors is shown in Figure 16.

5.1 Hall Sensor 1

The first hall sensor switches the junction to allow the train to enter the block. This halls sensor performs a dual purpose, as it allows the train to both enter and exit the block. To accomplish this, the Arduino code simply alternates the direction of the junction every time a train passes over it. A boolean variable stores the current direction

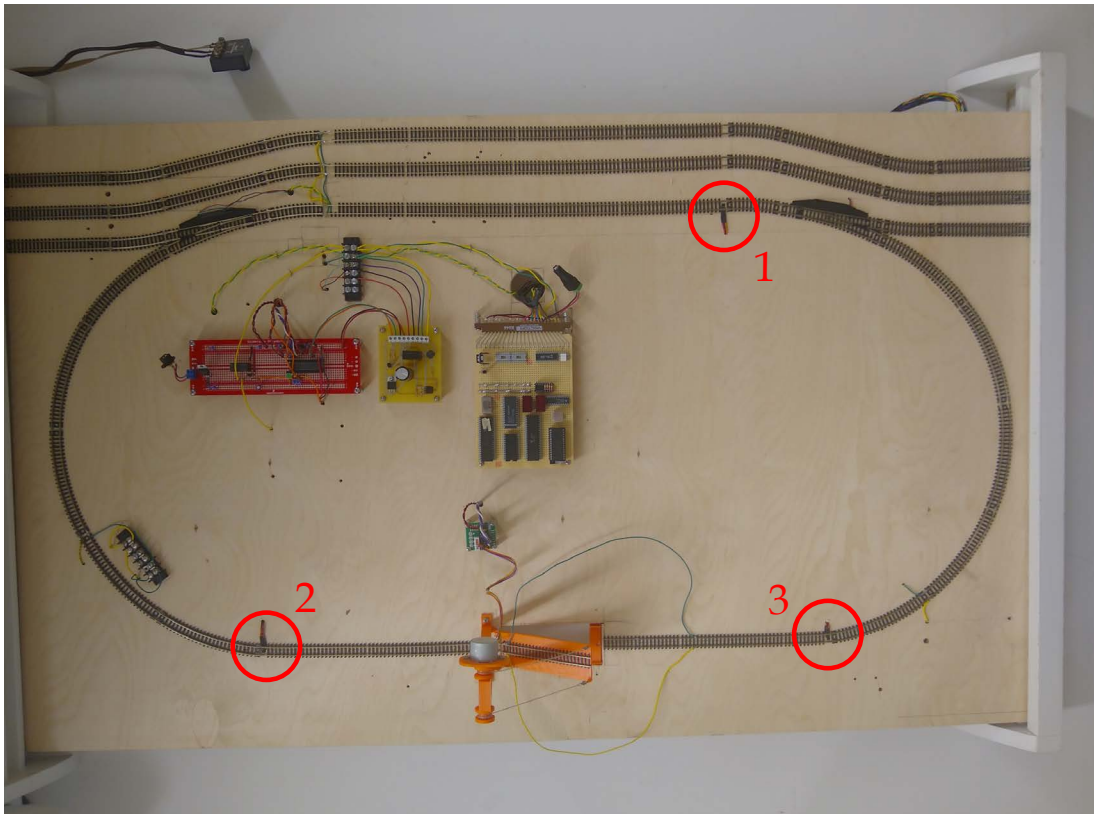


Figure 16: Diagram showing the three hall effect sensors in the board.

of the junction and is negated every time a train passes over the track. The Arduino then activates the relay in the trickle charge circuit by setting the trigger pin low for 100 microseconds This is enough time to allow the capacitor to discharge and throw the switch in the junction to the other side. After the 100 microseconds have passed, the trigger pin is set back to high to allow the capacitor to charge again. Because an insufficient amount of charge could cause the junction to partially flip, a timer was used to limit the amount of successive calls to the junction. Through testing, this timer was set at 9 seconds, so the junction could only be triggered once every 9 seconds but will still be constantly polled when the timer is not running.

5.2 Hall Sensor 2

Once the train reaches the second hall sensor, the system starts the bridge sequence. The first thing that the Arduino does in this sequence is to cut the power from the track by setting the relay pin to low. This stops the train to wait for the drawbridge to lower. The Arduino then tells the stepper motor to move 3000 steps, which is enough to lower the bridge from its up position to a flat position. At the same time, a timer is

started that waits for 8 seconds, which is enough time for the bridge to lower, and then sets the relay pin high again which starts the train moving again and crosses the bridge.

5.3 Hall Sensor 3

This hall effect sensor performs a single purpose. It simply functions to confirm that the train has crossed the bridge and once it does, it raises the bridge. This accounts for the train getting stuck before crossing the bridge, where a simple timer could raise the bridge before the train crosses. After the Arduino detects the hall sensor low value, it sets the stepper motor position back to its original upright position, resetting the system for the next train to come in.

5.4 Stepper Motor

The stepper motor is driven using the AccelStepper library [2], which allows the Arduino to control the stepper motor with a maximum speed and acceleration/deceleration. This increases the reliability of the system, as jerky movements could produce unpredictable results, especially with the difficulty of lining up the bridge. Setting limits in software also adds an additional layer of safety in that the system will not tear itself apart if the motor is told to move without raising up. For instance, if the hall sensor simply told the motor to advance 3000 steps, if the sensor was somehow tripped twice then the motor would overactuate and move the bridge beyond its intended range of motion, causing damage to the components. By only telling the motor positional values, it ensures that as long as the motor is positioned correctly on startup, it will never exceed the intended range of motion. The library also allows the motor to be driven independently of what the Arduino is doing. This makes it possible for the the Arduino to sense hall sensor 1 to switch the junction while the bridge is still being raised. This is done with the stepper.run() method, which polls the motor every loop and executes a step if it is due. Once the motor has reached its destination, the run method does not drive the motor further, until it receives a new move command.

5.5 Performance Metrics

Using the loopTimer library [3], the time for each loop was found to be about 80 microseconds per cycle, or approximately 12,500 Hz. This ensures that the Arduino does not miss a train passing over the hall effect sensor. The program also only uses 36% of the Arduino's onboard memory, which allows for room for expansion in the future. For debugging purposes, there are also multiple print statements that print messages according to what the Arduino is doing. For instance, once hall pin 2 is tripped and the relay pin is set to low, the Arduino prints "Stopping Train". This tells us that the

Arduino senses the train and is executing the intended instructions, so if the train does not stop we know that it's the hardware and not the software. The printing was done with the SafeString library [4] since the default Serial library includes calls to the delay() method which would be a bottleneck in keeping the loop running quickly. The code also implements a heartbeat LED that blinks once every second to show a visual indication that the Arduino has started up and is running the code. This ensures that the trains are not started on the track while the Arduino is still in its startup routine, which would cause it to miss measurements.

References

[1] Thingiverse.com. "Crusader Castle Gate by jansentee3d." (), [Online]. Available: https://www.thingiverse.com/thing:3332947 (visited on 05/16/2023).

[2] waspinator, *Waspinator/AccelStepper*, May 7, 2023. [Online]. Available: https://github.com/waspinator/AccelStepper (visited on 05/16/2023).

[3] spok7, *Spok7/LoopTimer*, Jun. 8, 2018. [Online]. Available: https://github.com/spok7/LoopTimer (visited on 05/16/2023).

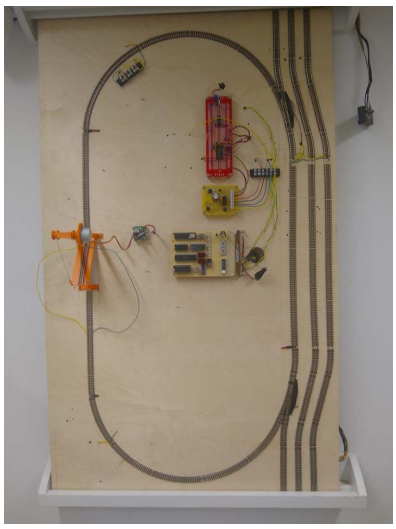
[4] PB2, *SafeString*, Apr. 7, 2023. [Online]. Available: https://github.com/PowerBroker2/SafeString (visited on 05/16/2023).

[5] "Arduino Serial I/O for the Real World Non-Blocking Text I/O using the SafeString library." (), [Online]. Available: https://www.forward.com.au/pfod/ArduinoProgramming/Serial_IO/index.html (visited on 05/16/2023).

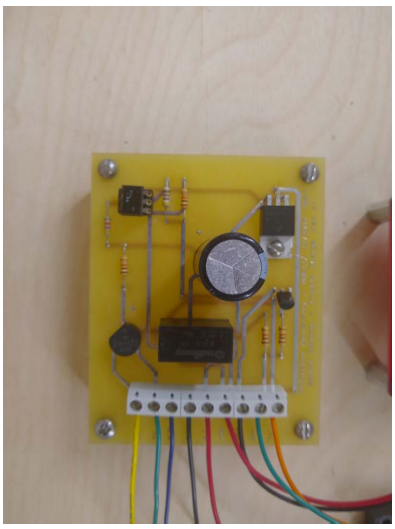
[6] "How to code Timers and Delays in Arduino." (), [Online]. Available: https://www.forward.com.au/pfod/ArduinoProgramming/TimingDelaysInArduino.html#using (visited on 05/16/2023).

[7] "Simple Multitasking Arduino on any board without using an RTOS." (), [Online]. Available: https://www.forward.com.au/pfod/ArduinoProgramming/RealTimeArduino/index.html (visited on 05/16/2023).

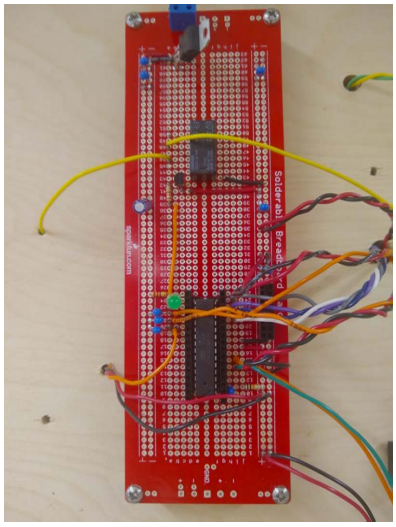
A Additional Photos



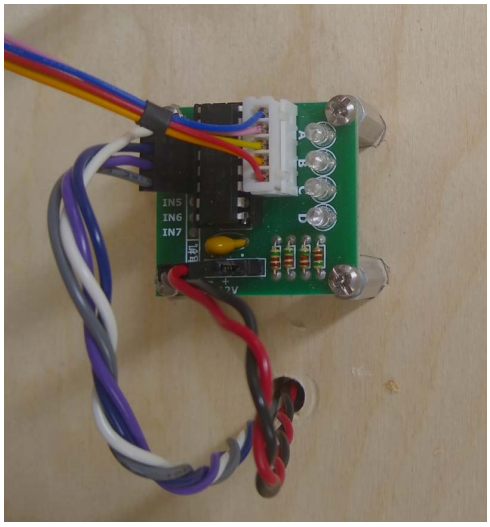
(a) Final Project Board



(b) Trickle Charger



(c) Arduino Daughter Board



(d) Stepper Driver

Figure 17: Various photos of the board and its components.

B Additional Diagrams

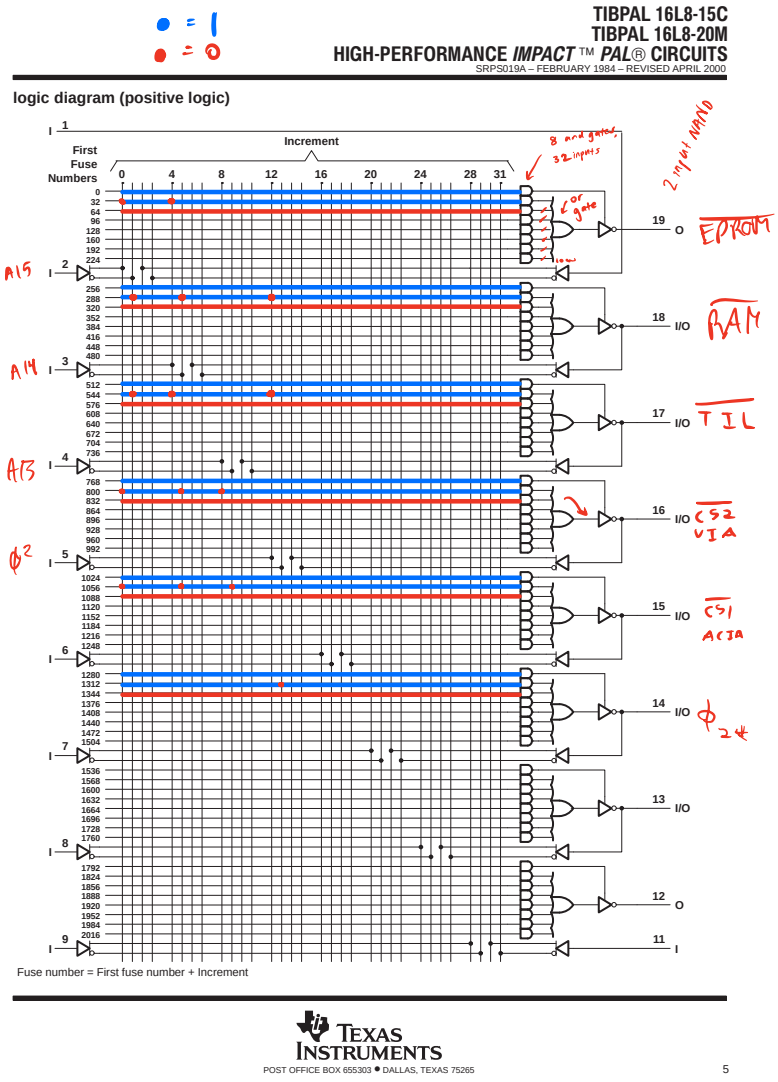


Figure 18: GAL worksheet.

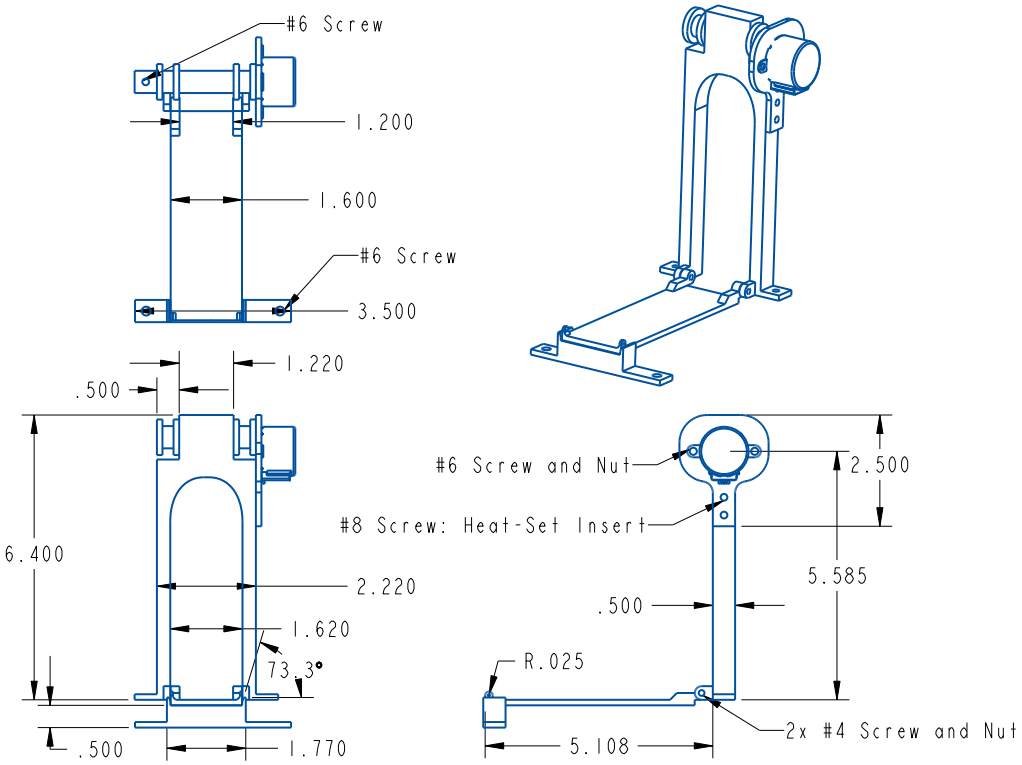


Figure 19: CAD drawing of the full bridge mechanism. Dimensions are for reference only, and are not intended for manufacturing purposes.

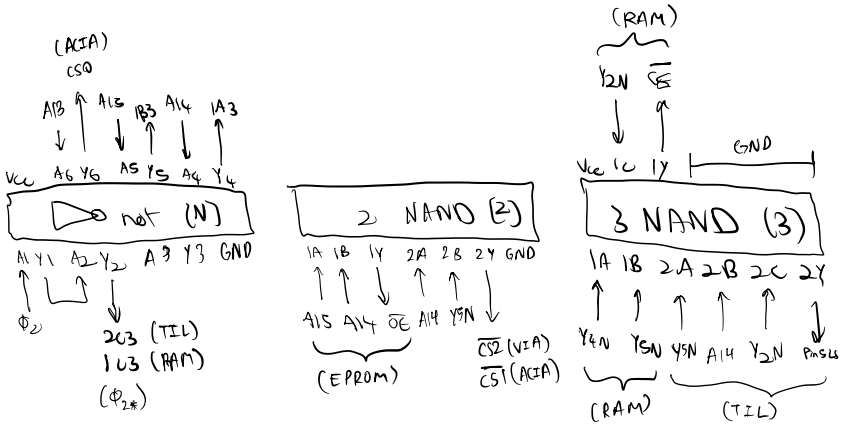


Figure 20: Sketch of the early ADL logic, implemented with 3 NAND gate chips instead of the GAL.

C Arduino Code

```
#include <SafeString.h>
#include <loopTimer.h>
#include <BufferedOutput.h>
#include <millisDelay.h>
#include <SafeStringReader.h>
#include <AccelStepper.h>
#define LEFT 1
#define STRAIGHT 0

// PIN DECLARATIONS
int dirPin      = 2;
int trigPin     = 3;
int stepperPin1 = 5;
int stepperPin2 = 6;
int stepperPin3 = 7;
int stepperPin4 = 8;
int ledPin      = 9;
int relayPin    = 10;
int hallPin1    = 11;
int hallPin2    = 12;
int hallPin3    = 13;

// TRICKLE CHARGE
bool dir = LEFT;

// STEPPER MOTOR
const int stepsPerRevolution  = 2038;  // number of steps per
   1 full revolution
const int stepperMaxSpeed     = 600;   // speed in RPM
const int stepperAcceleration = 300;
const int stepperDownPosition   = 3000;
AccelStepper stepper(4, stepperPin1, stepperPin3, stepperPin2,
    stepperPin4);

//BufferedOutput to avoid blocking in Serial.print
createBufferedOutput(bufferedOut, 80, DROP_UNTIL_EMPTY);
```

```
35
36 //Blocks the sequences from being triggered repeatedly
37 int JUNCTION_BLOCKING_TIME = 9000; //ms
38 int BRIDGE_BLOCKING_TIME = 1000;
39 int TRAIN_BRIDGE_WAIT = 8000;
40 int BRIDGE_UP_BLOCKING_TIME = 1000;
41
42 bool ledOn = false; // keep track of the led state
43 bool switchingStarted = false; // keep track of trickle charge
       state
44 millisDelay ledDelay;
45 millisDelay bridgeDelay;
46 millisDelay bridgeUpDelay;
47 millisDelay junctionTrigTimer;
48 millisDelay trainStartTimer;
49 millisDelay switchBlockOnDelay;
50
51 void setup() {
52   Serial.begin(9600);
53   bufferedOut.connect(Serial);  // connect bufferedOut to
       Serial
54   bufferedOut.println("Initializing");
55
56   stepper.setMaxSpeed(stepperMaxSpeed);
57   stepper.setAcceleration(stepperAcceleration);
58   pinMode(dirPin, OUTPUT);
59   pinMode(trigPin, OUTPUT);
60   pinMode(ledPin, OUTPUT);
61   pinMode(relayPin, OUTPUT);
62   pinMode(hallPin1, INPUT_PULLUP);
63   pinMode(hallPin2, INPUT_PULLUP);
64   pinMode(hallPin3, INPUT_PULLUP);
65
66   digitalWrite(relayPin, HIGH);
67   digitalWrite(trigPin, HIGH);
68   digitalWrite(dirPin, dir);
69   digitalWrite(trigPin, LOW);
70   delay(100);
```

```
71    digitalWrite(trigPin, HIGH);
72    ledDelay.start(1000); // start the ledDelay, toggle every
          1000ms, heartbeat
73  }
74
75  void blinkLed() {
76    if (ledDelay.justFinished()) {    // check if delay has timed
          out
77      ledDelay.repeat(); // start delay again without drift
78      ledOn = !ledOn;      // toggle the led
79      //bufferedOut.println("LED Toggling");
80      digitalWrite(ledPin, ledOn ? HIGH : LOW); // turn led on/
          off
81    } // else nothing to do this call just return, quickly
82  }
83
84  void switchJunction() {
85    if (!switchBlockOnDelay.isRunning()) {
86      if (digitalRead(hallPin1) == LOW) {
87        switchingStarted=true;
88        //bufferedOut.println("Starting Switch");
89        switchBlockOnDelay.start(JUNCTION_BLOCKING_TIME);
90      }
91    }
92    if (switchingStarted) { // start one now
93      switchingStarted = false;
94      dir = !dir;
95      digitalWrite(dirPin, dir);
96      digitalWrite(trigPin, LOW);
97      // start delay to turn off pin
98      junctionTrigTimer.start(100);
99    }
100   if (junctionTrigTimer.justFinished()) {
101     digitalWrite(trigPin, HIGH);
102     //bufferedOut.println("Stopping Switch");
103   }
104 }
105
```

```
106 void lowerBridge(){
107   if (!bridgeDelay.isRunning()){
108     if (digitalRead(hallPin2) == LOW) {
109       digitalWrite(relayPin, LOW);
110       //bufferedOut.println("Stopping Train");
111       stepper.moveTo(stepperDownPosition);
112       //bufferedOut.println("Lowering Bridge!");
113       trainStartTimer.start(TRAIN_BRIDGE_WAIT);
114       bridgeDelay.start(BRIDGE_BLOCKING_TIME);
115     }
116   }
117   if (trainStartTimer.justFinished()) {
118     digitalWrite(relayPin, HIGH);
119     //bufferedOut.println("Starting Train");
120   }
121 }
122
123 void raiseBridge(){
124   if (!bridgeUpDelay.isRunning()){
125     if (digitalRead(hallPin3) == LOW){
126       stepper.moveTo(0);
127       //bufferedOut.println("Raising Bridge!");
128       bridgeUpDelay.start(BRIDGE_UP_BLOCKING_TIME);
129   }
130   }
131 }
132
133 void loop() {
134   bufferedOut.nextByteOut(); // call at least once per loop to
        release chars
135   //loopTimer.check(bufferedOut); // send loop timer output to
        the bufferedOut
136   blinkLed();
137   switchBlockOnDelay.justFinished();
138   switchJunction();
139   bridgeDelay.justFinished();
140   lowerBridge();
141   bridgeUpDelay.justFinished();
```

```
142   raiseBridge();
143   stepper.run();
144 }
```