Automated Scoring System for Ticket to Ride

Team 43 -- Matthew McCracken and Andrew Douglas

ECE 445 Design Document -- Fall 2017

TA: Kexin Hui

1 Introduction

1.1 Objective

Ticket to Ride is a fun game to play with friends, but when it comes to counting your score, it can put a strain on your relationship. The scoring process includes tallying your scores obtained from laying down train cars on certain paths and trusting that your friends did it correctly. It also includes the longest path bonus, which can take a while to count and determine whose path is longer. Destination cards can be confusing to determine. Besides keeping track of the score, it is also tedious to place the individual train cars on the board to claim a route. This can take up a lot of time and slows the game down unnecessarily.

Our solution to this problem is to automate the scoring process of this game. Automation will greatly increase enjoyment, alleviating the most cumbersome and time consuming part of the whole experience. The automation will include indicating the current player's turn, showing current scores, and automatically calculating end-game scores including bonuses from destination cards and longest path. Instead of manually placing train cars on the board, LEDs will automatically be illuminated when a player captures a route. The color of the LEDs will indicate which player controls the route.

1.2 Background

From personal experience, tallying score is the worst part of Ticket to Ride. This sentiment is shared by Jacob Bryan (current TA for this course), who first suggested we implement the solution to this problem. There is feasible demand for a product like this, as there are automated versions of popular board games out there already, such as electronic Monopoly and The Game of Life. There was also a project in 2013 that automated the scoring process of Settlers of Catan that performed well - showing that there was a demand for automated scoring processes for board games like Ticket to Ride.

1.3 High-Level Requirements

- Train spaces should light up when the corresponding path is claimed.
- The game must be able to automatically calculate and display the players' score during the game, as well as automatically factor in longest path bonus and destination cards at the end of the game.
- The game should automatically end when a player's number of train cars dips to 2 or below.

2 Design

2.1 Block Diagram

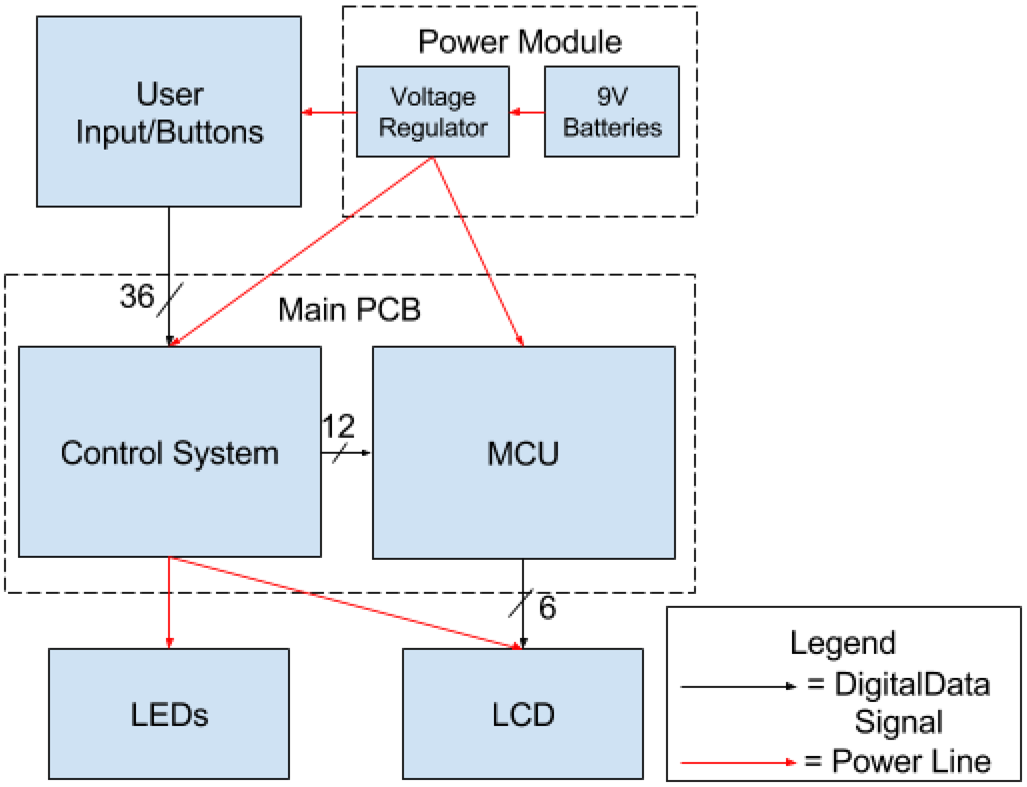

Figure 1: Block Diagram

The user input, control, and LED modules will take care of lighting the LEDs when a path is taken. The MCU will have dedicated memory to be able to store player scores as well as calculate scores in real time. The MCU will also keep track of number of train cars for each player, allowing for an automatic end to the game when the number of train cars goes below 2 for any player. The MCU accomplishes this by receiving a 12 bit digital data line from the control module that indicate what action each player took on their turn. These signals are then decoded by the MCU and used to update the game state. The LCD module is connected to the MCU and is used to give feedback to players about their current point totals. This screen will indicate when the game has ended and also show all of the players' point totals so that the winner is determined immediately.

2.2 Physical Design

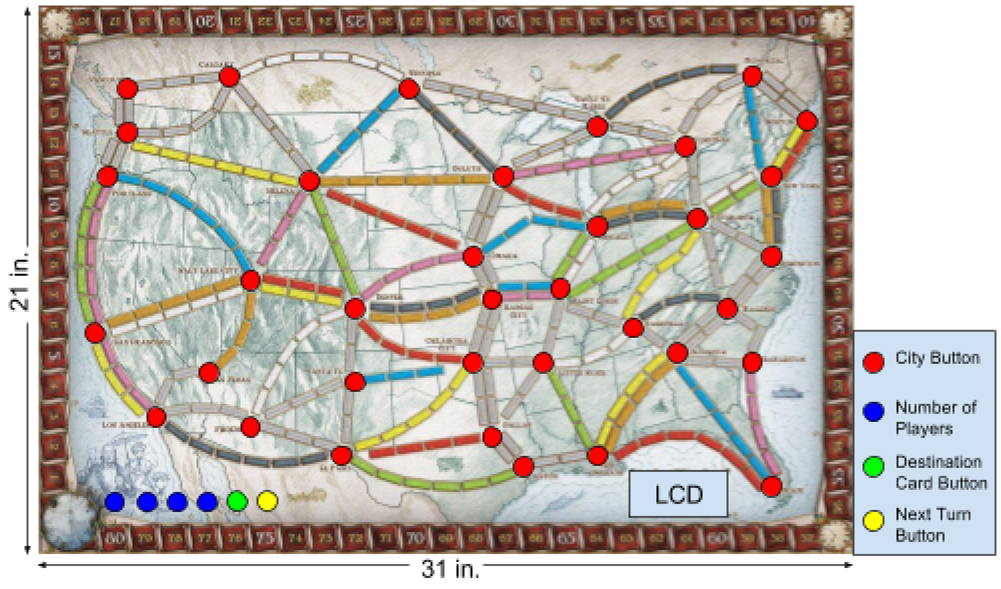

Figure 2: The Physical Board

Above is an image of the Ticket to Ride board game. Buttons will be placed at each of the different cities (the orange dots on the board) and will be used to indicate that a player is taking control of a rail line between two cities. The buttons will also allow a player to enter in the two endpoints listed on a destination card drawn. In addition to the buttons on the cities, there will be a few more buttons located in the lower left corner of the board allowing the users to indicate the number of players, pass the turn along to the next player, and indicate that they are entering a destination card rather than claiming a rail. Instead of placing the plastic trains to claim a route, LEDs will light up along each route to indicate that it has been claimed. The color of the LEDs on the path will be different depending on which player took control of the route. An LCD screen will be placed in the lower right corner of the board (where the scoring details are now located) and will display the current point total for each player and indicate which player is currently taking their turn. There will be a one inch deep wooden housing constructed that the game board will rest in - all of the wiring and electrical components can be placed in the housing to achieve a polished look in the finished product.

2.3 Block Design

2.3.1 User Interface Module

2.3.1a Functional Overview

The user interface module will output a 36 bit digital data signal to the control system module based on the specific buttons pressed by the user. Two city buttons will be pressed by a player when they want to claim a route. These two cities will correspond to two bits in the 36 bit data signal that will be set high. In addition to claiming routes, the buttons can be used to enter destination cards. This can be done by first pressing the “Destination Card Button” and then pressing the two relevant city buttons at the same time. There are also a few special buttons necessary to play the game, namely, buttons to indicate the number of players and a button that can be used to end a player's turn. The signals from each of the buttons are sent directly to the MCU for processing.

2.3.1b Requirements and Verifications

Requirements	Verifications
● Provide debounced signals to the control module representing the user input.	● Use an oscilloscope to observe behavior right after the button is pressed to see if the output is bouncing.
● Signals should be provided at 5V with a tolerance of ± 0.5V.	● Use a multimeter to determine input and output voltage of the module.

Table 1: Requirements and verifications for user interface module.

2.3.2 Control System Module

2.3.2a Functional Overview

The control module will take a 36 bit digital data signal input from the user interface module, as well as output a power line to the LED module and a 12 bit digital data signal to the MCU module. The control module will include logic to set control signals necessary for gameplay, as well as handle any button presses that occur in the user interface, such as number of players, end turn, and destination card input mode. When a set of buttons are pressed, the control module will determine if the move is valid. This can be done by maintaining one bit flip flops for each of the rail lines that indicate whether or not they have been claimed. If the move is not valid, the module sends a 'not valid' digital signal (12 bits of 1) to the MCU. If the move is valid, the control module will encode the path that was claimed, send the command to the MCU block (digital line, 12 bits), then turn on the LEDs associated with the path.

The encoded signal that will be sent to the MCU will be generated by a compression function. This is necessary because it would be impractical to send all 36 bits to the MCU, but a unique representation of each pair of cities is still necessary. In the 36 bit input vector, only two positions will have 1s at a time. These 1s correspond to the two cities between which a player would like to claim a route. Given this limitation, the compression function can simply output the number of zeros before the least significant 1 followed by the number of zeros between the least significant 1 and the most significant 1. Since the input is 36 bits long, the largest number of sequential zeros that could occur is 34. Six bits are needed to represent the number 34 and thus the output of the compression must be 12 bits in order to correctly handle all possible inputs. An example of the inputs and outputs to this function is shown in table 3, and the process that encodes the data is shown in figure 3.

2.3.2b Requirements and Verifications

Requirements	Verifications
• Compress 36-bit data from the User-Input module into 12-bit data to be sent to the MCU. • Input 5V so we can provide 5V to the correct LEDs (based on user input) with a tolerance of ± 0.5V in less than 0.25s. • Provide 25mA ± 5mA to the correct LEDs (based on user input) in less than 0.25s.	• Use oscilloscope to compare input-output voltages from logic gates and ensure the correct compression is output. See table 3. • Use multimeter to determine input and output voltage. Timing will be verified using a stopwatch to determine the delay between buttons pressed and LEDs turning on. • Use multimeter to determine input and output current. Timing will be verified using a stopwatch to determine the delay between buttons pressed and LEDs turning on.

Table 2: Requirements and verifications for control system module

2.3.2c Supporting Material

Input (36 bits)	**Output (12 bits)**
0...011	000000 000000
0...101	000000 000001
...	...
110...0	100010 000000

Table 3: Compression function I/O

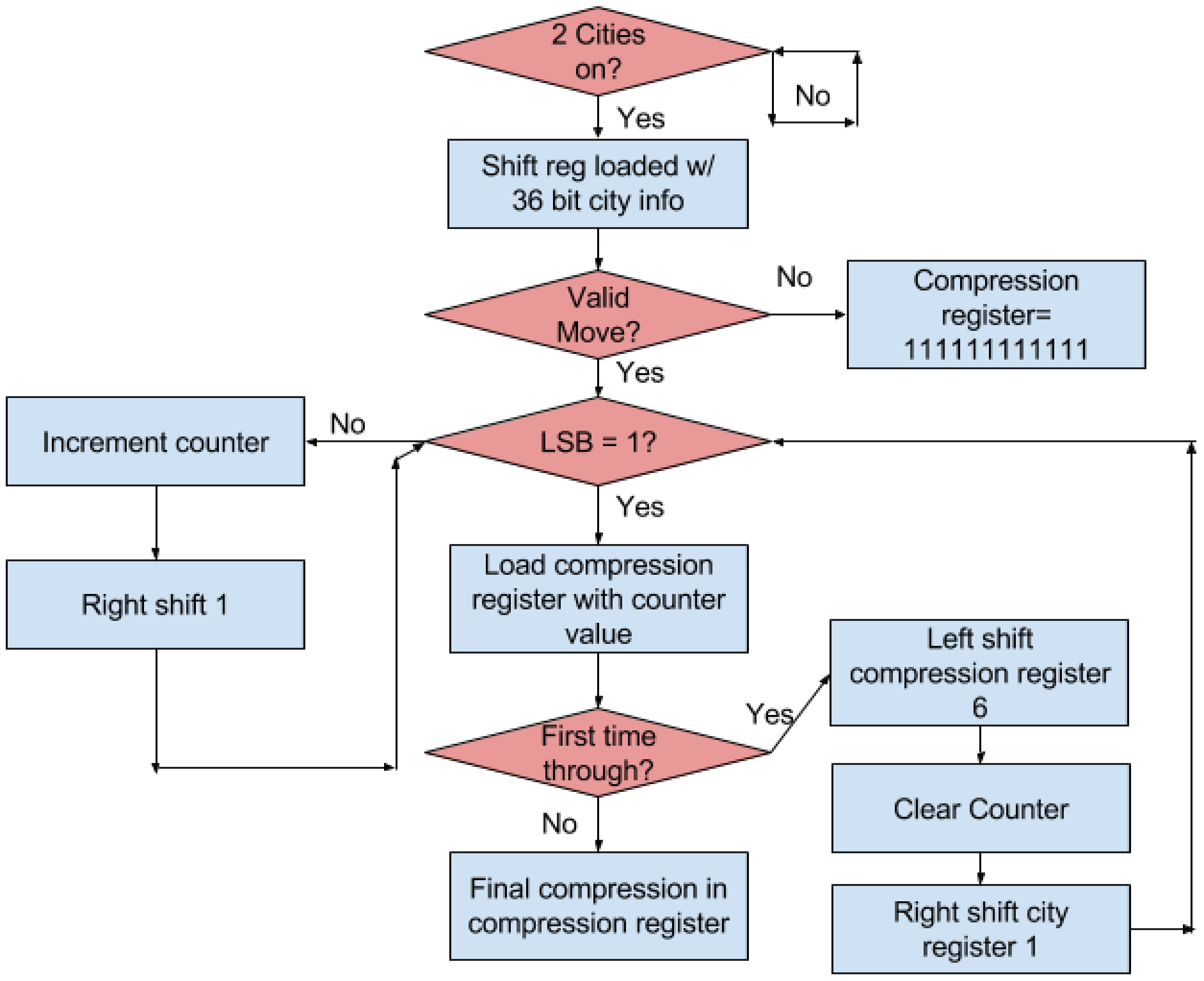
2 Cities on?
No
Yes
Shift reg loaded w/ 36 bit city info
Valid Move?
No
Compression register= 111111111111
Yes
Increment counter
No
LSB = 1?
Yes
Right shift 1
Load compression register with counter value
Left shift compression register 6
Yes
First time through?
Clear Counter
No
Final compression in compression register
Right shift city register 1

Figure 3: Flowchart of compression algorithm.

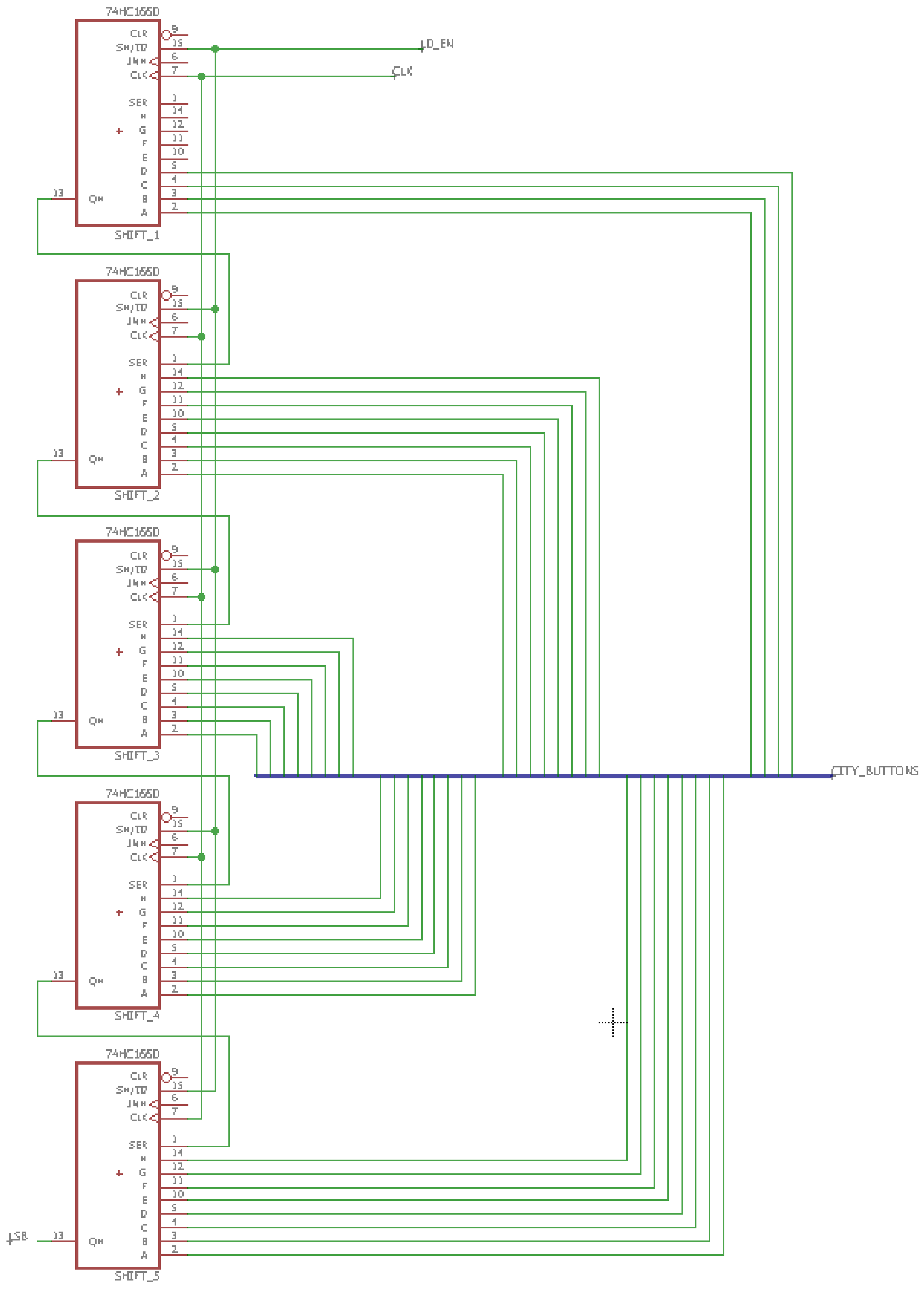

Figure 4: Circuit schematic of the first half of compression algorithm. Input comes from CITY_BUTTONS and output to next half is LSB.

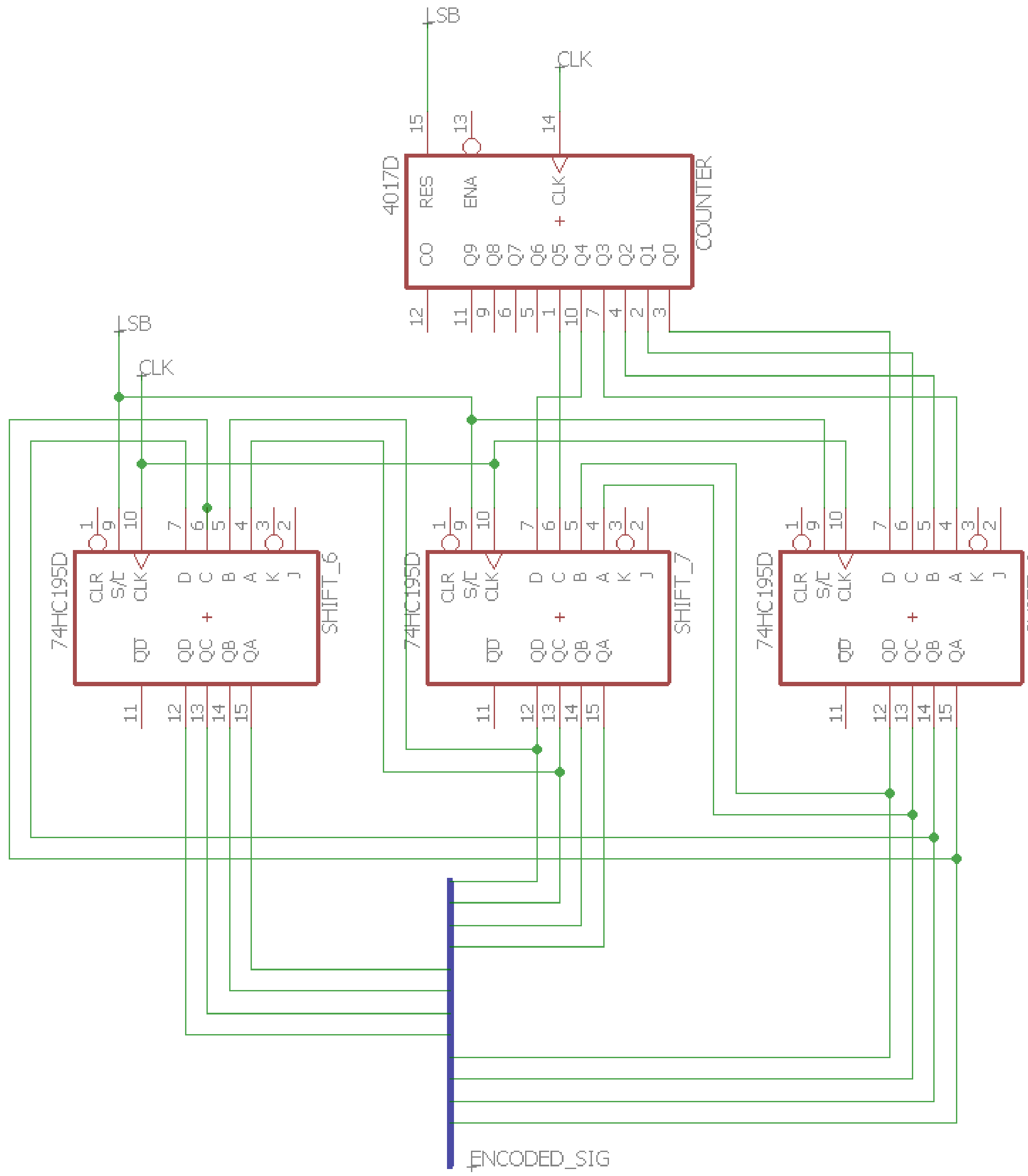

Figure 5: Circuit schematic of the second half of compression algorithm. The input from the first half is LSB and the output is ENCODED_SIG.

2.3.3 LED Module

2.3.3a Functional Overview

The LED module will be used to indicate the paths that each player has claimed. LEDs placed on the routes will also light up to indicate which player has claimed that route. Each player will have a different color, making it easy to see which routes they have just by looking for a specific color light. LEDs will also be used to give the users feedback about the choice they are currently making. For example, some cities are connected by a dual set of train lines. If a player is trying to claim one of these two lines (and neither has yet been claimed) the LEDs for one line will blink, indicating that the player needs to select between the two lines. The number of players buttons could then be used to select between the two lines. A message on the LCD screen would be used to indicate which button press corresponds to which route. In total, 308 LEDs will have to be used to implement this design, as there are 308 possible positions to place a train car.

Each LED should require around 25mA of current, with a tolerance of ± 5mA. This can be achieved using current-limiting resistors as to not burn them out. Each set of LEDs will have a 5V ± 0.5V input from the control system module.

Each individual path can be represented by a number of LEDs ranging from 1 to 6. The LEDs cannot be wired in series, as their operating voltage (approximately 2.2V) is too high to have 6 LEDs wired in series with a 5V supply. We will wire each series of LEDs in parallel with a different valued resistor for each configuration as to keep the LED current the same across the board. Configurations with more LEDs will require a smaller valued resistor to allow the correct current to be split across the multiple LEDs, a la KCL.

The color will be controlled via the MCU depending on the player number. There are five color options: red, green, blue, yellow, and white. The MCU will switch colors every turn depending on whose turn it is and always output a 3 bit digital color signal to the LEDs. The LEDs will store the color by using a flip flop for every color on each LED line. The flip flop will be enabled when both of the corresponding cities to that line of LEDs are high, allowing the current color from the MCU to be stored for that cycle and keep that color even when the player number changes.

2.3.3b Requirements and Verifications

Requirements	Verifications
● Receive 5V with a tolerance of ± 0.5V and limit current to 20mA using resistors. ● Display the paths taken in the correct colors, number of players, and the current turn.	● Use a multimeter to determine input voltages and currents for each line of LEDs. ● Every combination of button inputs will be tested to ensure that all LEDs light up accordingly, as well as display the correct number of players and current turn.

Table 4: Requirements and verifications for LED module

2.3.3c Supporting Material

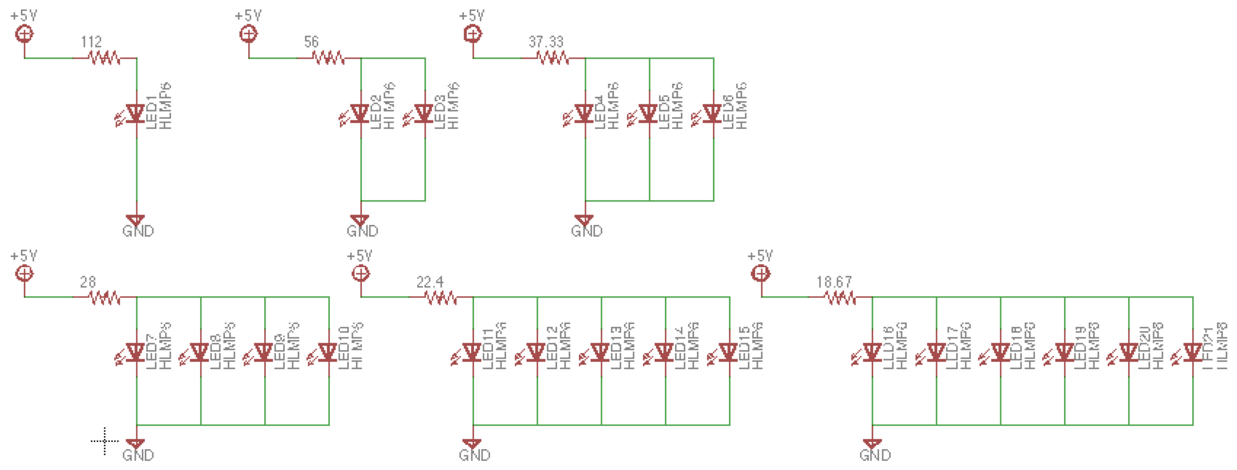

Figure 6: Sample LED lines of length 1 through 6 and how they should be wired.

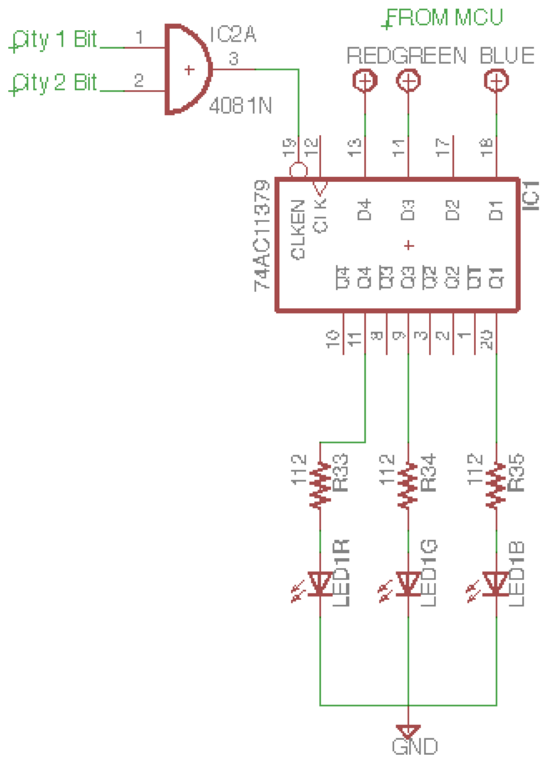

Figure 7: An LED line consisting single multi-colored LED, represented by 3 different colored LEDs. There will be a total of 78 lines with varying numbers of LEDs wired in parallel as in Figure 6.

2.3.4 MCU Module

2.3.4a Functional Overview

The MCU module will receive commands from the control module - in the form of a 12-bit digital data input - for what move was made by the current player, as well as number of players and current mode. The MCU block will keep track of cumulative user score and output a data signal to the LCD display. The MCU will also determine what color each path will be if claimed during a given turn. This is done by This module will also be responsible for calculating and assigning the longest route bonus as well as handling the destination cards that each player had entered in. The control system module will tell the MCU module what specific move was taken and the MCU will then have to update the game state to handle this action. The MCU will communicate with the LCD unit to display information to the players. If an error was received from the control unit, this will be decoded and displayed on the LCD in a human readable error message. Updates on the scores and current player turn can also be sent to the LCD for display.

2.3.4b Requirements and Verifications

Requirements	Verifications
● Decode 12-bit digital signals from the control module to calculate current scores.	● Every combination of cities will be tested to ensure that all scorekeeping is correct. See table 3.
● Successfully calculate score, including bonuses from destination cards and longest path.	● We will display values associated with the completion of routes and the longest path in real time so we can verify our results from the MCU.
● Output game-state information to the LCD screen.	● Every possible input will be executed as to verify the correct output from the MCU. This is a highly visual requirement so the verification will also be visual.
● Output the correct color assignments during a given turn	● A multimeter will be used to measure the color signals and ensure that the correct color is being output at all times. This is a requirement that can be verified visually as well as quantitatively.
● Receive 5V and 145mA from the power module voltage regulators to operate within the recommended range consistently.	● A multimeter will be used to measure input voltage and current to ensure the MCU is operating at the recommended levels.

Table 5: Requirements and verifications for MCU module.

2.3.5 LCD Module

2.3.5a Functional Overview

The LCD display will receive current score, turn, and status information from the MCU in the form of 6 bit digital data signals and output to show the real-time score calculations. The status information refers to whether or not a player is entering a destination card. The LCD display will make it easy for users to confirm that destination cards are being entered correctly. All pieces of information will be formatted in such a way that they will be human readable on the LCD screen. Power for the screen will be provided directly from the power module. These messages will display any errors to the users, such as an invalid city connection, and also make it easy to check everyone's current score.

2.3.5b Requirements and Verifications

Requirements	Verifications
● Receive input from MCU to display current scores, turn, and number of cars left.	● Use an oscilloscope to determine input data voltages. Correct usage of the LCD can also be observed on the screen itself.
● Input 5V ± 1V and 15mA ± 5mA from the power module.	● Use a multimeter to determine input voltage and current from power module to ensure operation within recommended bounds.

Table 6: Requirements and verifications for LCD module

2.3.6 Power Module

2.3.6a Functional Overview

The power module will consist of two 9V batteries in parallel and be used to send a power line to the whole board. Voltage regulators and current-regulating resistors will be used to stabilize power. Diodes will be used to prevent any damage to the electronics if the batteries are inserted backwards. It is important to keep power consumption to a minimum to maintain battery life, so we will use two 9V batteries in parallel. The need to plug into a wall outlet severely detracts from the experience of playing a board game so we will pay careful attention to maximize battery life.

2.3.6b Requirements and Verifications

Requirements	Verifications
● Output 5V with a tolerance of ± 0.5V from 9V alkaline batteries and voltage regulators to all modules but LED module.	● Use a multimeter to measure output voltage from voltage regulator to ensure that 9V is successfully being scaled to 5V.
● Output 145mA ± 10mA to the MCU.	● Use a multimeter to measure output current from power and input current to MCU.
● Output 10mA ± 5mA to the LCD screen.	● Use a multimeter to measure output current from power and input current to LCD.

Table 7: Requirements and verifications for power module

2.4 Tolerance Analysis

The control block is the block that poses the most risk to the completion of this project. The control unit needs to control LEDs, encode path data, and send it to the MCU for score calculation. It is the hardest module to implement solely because of the complexity of the hardware involved. There are a large number of routes that all have to produce a unique code that can be sent to the MCU. This unit will also be responsible for checking whether or not a move is valid. It will have to keep track of which routes are claimed and implement logic to check that the move a player has entered does not interfere with an existing route. Both of these components will require a significant amount of hardware design that can be time consuming and difficult to troubleshoot.

As the majority of our control system will be TTL logic, there is a high tolerance for error. Current is not much of a factor in TTL chips. Input voltage has a tolerance of about 1.5V, which is large compared to our 5V input signal. This nearly eliminates the need for noise reduction in our design. That being said, we will still try our best to hold to the voltage levels specified in this document and the datasheets for each component to ensure that nothing goes awry in our implementation.

3 Cost and Schedule

3.1 Cost Analysis

We can calculate the cost of labor using the formula suggested in the design document overview: number of partners x ($/hour) x 2.5 x hours to complete. Assuming a pay rate of 40 dollars an hour and approximately 10 hours of work each week (per person) we can estimate a rough labor cost of (2x40x2.5x100) or $20,000.

Part	Price / part	Quantity	Price	Running total
8 bit parallel-in serial-out shift register	$0.39	5	$1.95	$1.95
4 bit parallel-in parallel-out shift register	$2.49	3	$7.47	$9.42
10 bit counter	$0.51	1	$0.51	$9.93
Tactile Push Button Switch	$0.07	42	$2.94	$12.87
16x2 LCD Display	$9.95	1	$9.95	$22.82
32 Bit MCU MKV30F64VLH10	$5.30	1	$5.30	$28.12
Chanzon 5mm RGB LED	$0.09	308	$27.72	$55.84

Table 8: Pricing information.

Adding our labor cost to the cost of parts gives us a total cost estimate of $20,055.84.

3.2 Schedule

Week	Matt	Drew
10/9/17	Test control module implementation	Test LED module implementation
10/16/17	Begin first PCB design	Program MCU and check interaction with with control module and LCD
10/23/17	Order all necessary components (chips, LEDs, MCU, LCD)	Test that power module will
10/30/17	Prototype all modules on breadboard with PCB	Prototype all modules on breadboard with PCB
11/6/17	Finalize second PCB design	Begin building physical design prototype
11/13/17	Assemble all components into physical prototype	Verify functionality of all components and fix any bugs
11/20/17	Resolve any issues with functionality of overall design	Resolve any issues with functionality of overall design
11/27/17	Refine design based on comments in mock demo	Refine design based on comments in mock demo
12/4/17	Prepare demo and mock presentation	Prepare demo and mock presentation
12/11/17	Prepare presentation and final paper	Prepare presentation and final paper

Table 9: Schedule of next steps.

4 Ethics and Safety

We will uphold the IEEE code of ethics[1] during the development of our project. Any intellectual property concerns about building upon an existing board game will be avoided as we do not intend to sell the final product, however we will do our due diligence as to not plagiarize other works while designing our project. Our project has very minimal safety concerns, and could even improve upon the safety of the original board game. We will be eliminating the need for small game pieces and thus will be able to remove a choking hazard for small children. All of our electronics will be safely housed underneath the board in an insulated environment. We will take care that there are no exposed electrical components that could harm users. We will also put in a fail-safe to protect our circuit should the batteries be put in the wrong way, so there will be no chance of fire or frying our circuit.

References

[1] “IEEE Code of Ethics.” *IEEE*, www.ieee.org/about/corporate/governance/p7-8.html.