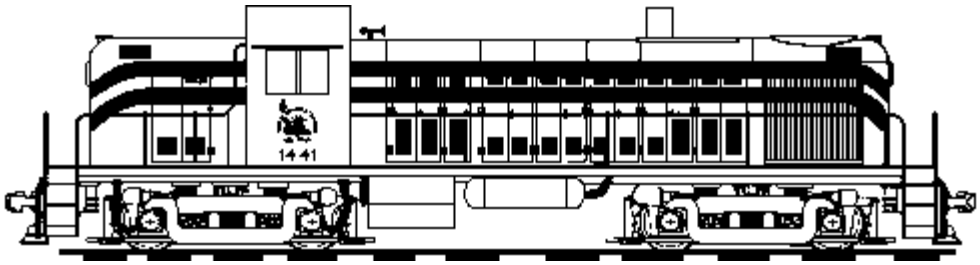
Train Race



Jason Kutch and Harris Yong
MAE 412
Professor M. Littman
May 2000

This paper represents our own work and was written in accordance with University regulations.

TABLE OF CONTENTS

1. INTRODUCTION TO TRAIN RACE

The objective of this project is to create an N-scale model train layout employing microcomputer control. All projects are to employ three basic operations: sense, actuate and sequence. Additional physical constraints define the layout of the project, but all components specific to our project, including the railroad tracks, daughter boards, and microprocessor are required to fit on a wooden board 48''×27'' in dimensions. The project board is to integrate seamlessly a pre-designed test stand.

The obvious features of the test stand are its three main tracks; a north and south track are both unidirectional, while a third track between these is bi-directional. Additionally, the test stand contains a computer interface board as well as the Hornby Zero-One™ railroad control system. The course (MAE 412)'s first half was devoted exclusively to the construction of a 6502 processor-based microprocessor unit which now serves as the interface between the project board and the Hornby system, thereby enabling the microcomputer to sense, respond to as well as initiate events on the test stand. As a background to our software, the computer executes the Lecky 2.0 routine, which allows multiple trains to operate on the board simultaneously without possibility of collision.

Our specific project, entitled “Train Race,” uses microcomputer control to make the speed of main alternating current trains directly dependent on the relative speeds of two direct current trains which run on independent railway loops and whose speeds are set by the user. Please see Appendix A for the general layout of the project board.

The project runs as follows: Initially, when the project is first powered, only trains on the AC railroads have the capability of moving. When the first AC train enters the board on the southbound track, it stops at the station defined by an optical proximity detector. Its arrival triggers DC trains, which run in sequence around their loops; the left DC train travels, followed by the right

DC train, both at speeds specified by potentiometer knobs housed on the project board. When each DC train is running, a software timing scheme runs and records the train "loop time" in arbitrary, which the microcomputer stores in memory locations. The stop point for each DC train is defined by a Hall Effect sensor, one of which is mounted below train level, in between the rails for each loop. When both DC trains complete their respective loops, the microprocessor compares the loop times of each train. By having the microprocessor interfacing with an Interactive Controller, codes sent by the microprocessor are translated into speeds and directions for trains on the AC circuit. If the left DC train completes its loop in less time than the right DC train, a code is sent from the microprocessor to the Interactive Controller, instructing the main track AC train to depart rapidly (speed 14). If the right DC train completes its loop more quickly, the waiting AC train departs more slowly (speed 10). Having run both the DC trains and having activated the AC circuit, one heat of Train Race is complete, and the system then waits for the next AC train, regardless of its number to enter the optical proximity detector station, and the same race sequence repeats indefinitely. Thus, the speed of any AC train operating on the system is defined by the relative speed of the two DC engines.

Train Race accomplishes the requirements of sensing, actuating, and sequencing with minimum hardware, straightforward software and with a high level of robustness.

2. OPERATION DIRECTIONS

Once power is supplied to the system and when an AC train is made to pass the optical proximity detector on the AC south AC track, Train Race operates autonomously. If there is no user input to change the relative speeds of the DC trains, Train Race continues to perform its functions without changing the AC circuit train speeds.

After each power up cycle, it is necessary to initialize the race by commanding an AC train to pass the optical proximity detector. The steps for this are as follows:

1. Start the program 'fmpu' at the prompt 'c:\>' on the Interactive Controller computer. The program should recognize computer 3 as being attached to the system. Load the data set of controller instructions from a file called 'race' into computer 3. Start the train block monitor routine.
2. Begin by bringing any train (1-8) at any speed (2-14) into the main track station located on the south track. The AC train will stop once it passes the optical proximity detector.
3. Operate DC trains within appropriate speed limits. The microcomputer's displays show a "stopwatch" for both DC train loops.
4. As soon as both trains have finished their respective loops, the display shows a 1 on the side of the winner.
5. The AC train will depart from the station, but there will be a short delay before the new speed is established.
6. Let the project run autonomously, or change the winner on the next race by adjusting the DC train speeds.

Table 1 summarizes the display codes for user convenience.

Display Code	Program State
00	System waiting for initial AC train to arrive at south track station.
Stopwatch	Individual DC loop train being timed
10	*Left* train won race; most recently departed AC train traveling at speed *14*
01	*Right* train won race; most recently departed AC train traveling at speed *10*

Table 1. Display codes.

There is no special requirement for the trains on the AC tracks. However, because the southbound track power is switched by a relay (discussed later), the rail connectors to the southbound tracks must be insulated. The DC trains must have a short enough "wheelbase" to be able to travel around the tight 8.0'' radius loops and must have a belly-mounted magnet such that it clears any obstructions while proceeding around the loop.

3. HARDWARE

The hardware involved with the project consists of the main microprocessor unit, along with DC current regulator circuits for DC train speed control, Hall sensors to detect DC train loop completion, and an optical sensor and track kill for AC train detection and stoppage, respectively, on the south track. The accessory circuitry was placed on 2 daughter boards, which were positioned on the project board with some proximity to the area that they controlled.

3.1 Microprocessor Unit

The core component of Train Race is a microcomputer based on the 6502 processor. This computer was built during the first half of the course and contains standard components including Versatile Interface Adapters (discussed later), Random Access Memory as well as a socket for an Erasable Programmable Read Only Memory chip. As directed, buffers are used for all lines entering and leaving from the microcomputer through an edge connector. An optical isolator prevents high voltages from the Hornby System from damaging the sensitive TTL circuitry.

A picture of the microprocessor unit mounted on the project board is shown in Figure 1. A schematic of the microprocessor unit is given in Appendix B.

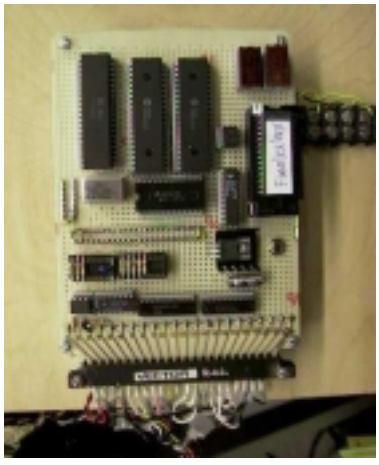


Figure 1. Photo of the microcomputer.

3.2 Address Decode Logic

The address logic for the 6502 microprocessor is decoded by a Generic Array Logic™ (GAL16V8) device. This device uses the three highest bits of the 16 address lines to activate particular devices. If writing to the device is necessary, φ_2 is included with the logic for timing purposes. That is, it is logically combined with the device's activation signals to ensure that the data is available at time of writing. The GAL also performs the logical inversion of the RESET line.

In decoding address logic, the GAL functions as a device allowing connections of specific inputs to hardwired logic gates. Essential connections are established by "burning" out those that are unwanted. The programming of a GAL for Train Race was accomplished with a .jed file containing lines of code specifying a '1' if a specific connection was to be removed and '0' if a connection should remain unaltered. The ADL for our computer was first planned on the diagram shown in Appendix C1. The appropriate .jed code was then read from this grid. Table 2 shows the relevant lines of code, with all other lines set to 0 except for the line immediately preceding the shown lines, which is set entirely to 1 in order to activate the GAL's output NOT gate.

LINE	CODE
0032	01110111011111111111111111111111
0288	10010111101111111111111111111111
0544	10011011101111111111111111111111
0800	01111011101111111111111111111111
1056	01111011011111111111111111111111
1312	11101111111111111111111111111111
1568	11111111111101111111111111111111

Table 2. Relevant lines of the .jed GAL code.

Appendix C3 shows the connectivity of our ADL with the appropriate components of the microcomputer.

3.3 Daughter Board #1

Daughter board #1 houses the components used to operate the AC southbound track. The two circuits on this board are those for the optical proximity sensor and for the southbound track power

kill. The southbound track is killed using a high TTL signal from PA0 at VIA 8000. When this line goes high, no current flows through the base of the 2N3906 transistor, and the relay opens the normally closed yellow AC connection.

The circuitry on daughter board #1 for the optical proximity detector provides the +5 VDC power and ground. An LM339 analog comparator compares the signal from the optical proximity detector with 2.5 V to send a TTL signal out to the computer (low = train is front of sensor). Various pull-up resistors and capacitors are used. A TIL149 transistor is also housed on this daughter board to trigger the relay. Figure 2 shows daughter board #1.

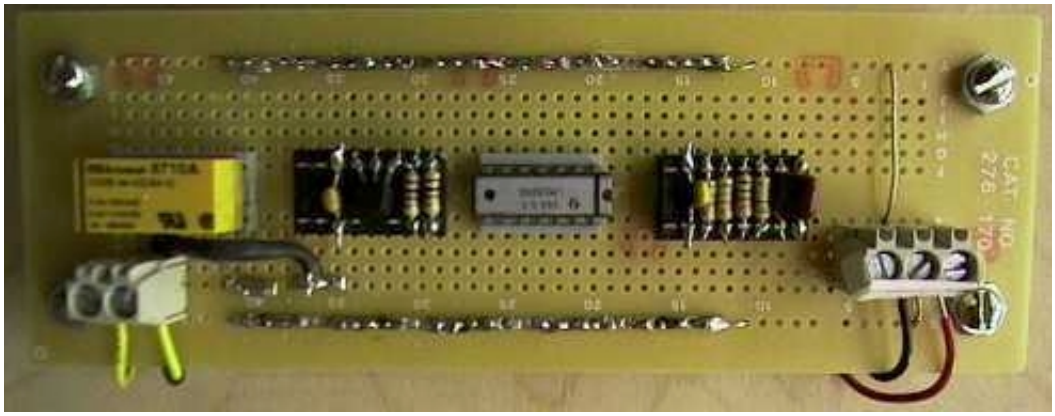


Figure 2. Photo of daughter board #1.

The wiring schematic is shown in Figure 3. For a larger image, please refer to Appendix D1.

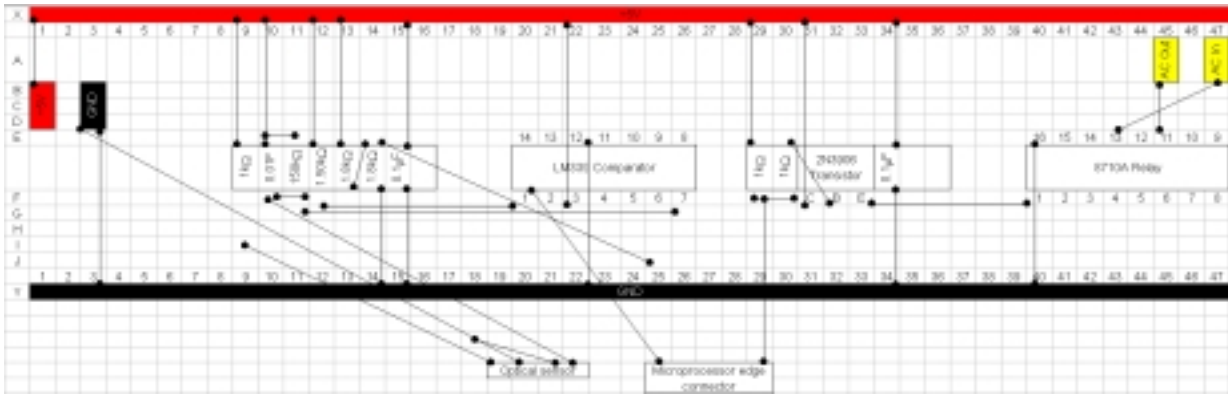


Figure 3.Layout and wiring of daughter board #1.

3.4 Daughter Board #2

Daughter board #2 contains most of the elements necessary to sense and actuate the two DC train loops. +5 VDC and ground drawn from the main computer, as are TTL control lines for both DC trains. The DC power is buffered using two 0.1 μF capacitors, and PB7 of VIA A000 is also routed to the daughter board for the purpose of cutting power draw during the data frames. AC power is fed to the motherboard and converted to +20 VDC and DC ground using a rectifier bridge circuit. A multiple input OR gate allows a DC train to run only when both the appropriate TTL signal and the PB7 lines are low. There are two 4N33 opto-isolators and two TIP 122 transistors, one each for each DC train. One rail of each DC track receives +20 VDC, while the other rail is connected to the collector of the transistor's collector. The user varies the train speeds by turning the appropriate potentiometer (mounted on the project board), which adjusts the base current to the transistor. A variety of pull-up resistors are used as required by the components.

The DC daughter board also contains the two 1 kΩ resistors required for the output leads of the two Hall Effect sensors that are used to detect the presence of the DC trains. As documented, the left DC train runs until it passes the Hall Effect sensor, after which the train is stopped to allow for the right train to run around the loop. When it reaches the Hall Effect sensor, the right train also stops, and the AC train runs at a speed determined by the relative loop times of the DC trains. Figure 4 shows daughter board #2.

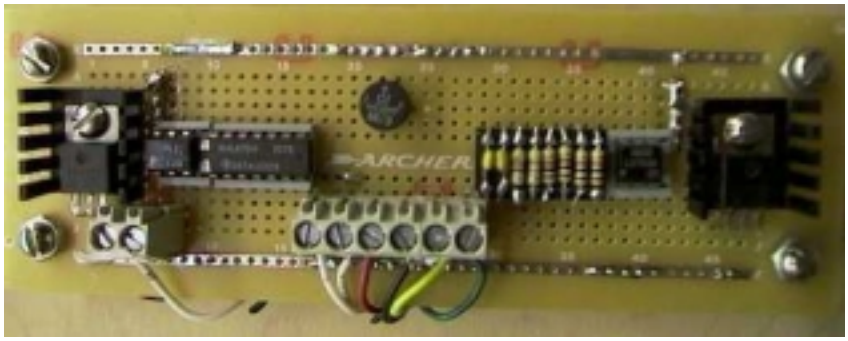


Figure 4. Photo of daughter board #2.

Figure 5 shows the layout and schematic of daughter board #2. Please see Appendix D2 for a larger image.

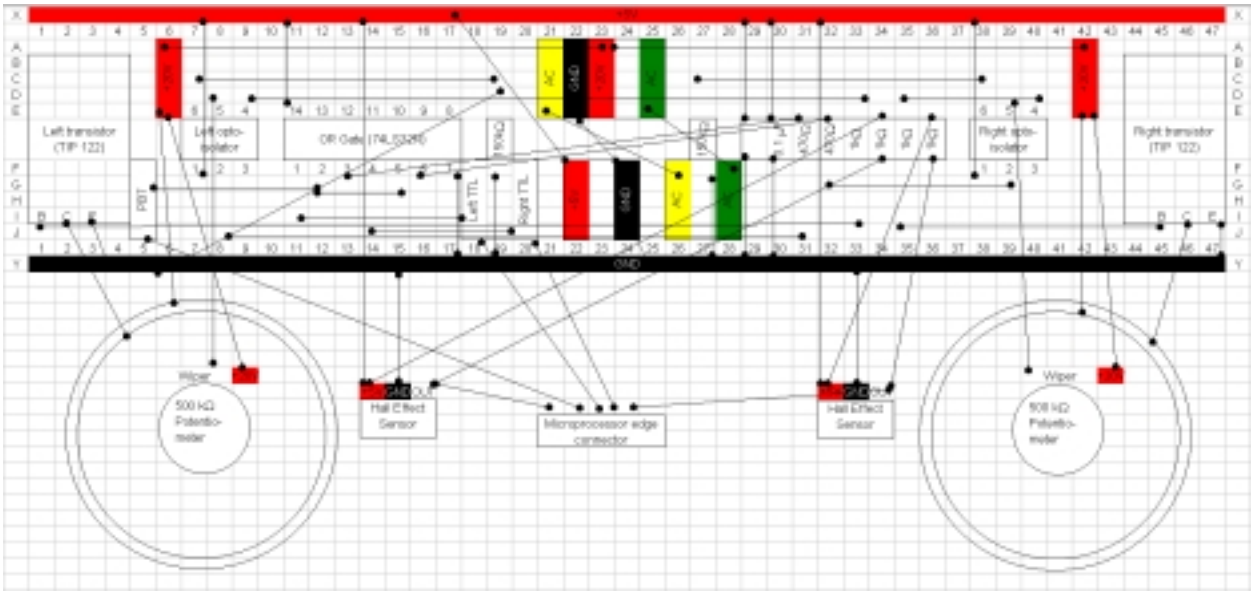


Figure 5. Layout and wiring of daughter board #2.

3.4 Miscellaneous Hardware

This section discusses the miscellaneous hardware mounted on the project board.

3.4.1 500 kΩ Potentiometers

As described earlier, a 500kΩ potentiometer is used to adjust the base current to the transistor controlling each DC train. These are mounted on the project board, with one lead to the 20 VDC supply off daughter board #2. The wiper of each potentiometer is connected to an opto-isolator. When the proper TTL signals are given, the base of the transistors receive current, and the appropriate DC rail circuit is closed to allow the train to run. Each potentiometer has a knob with a pointer as well as a dial to indicate the range of speeds that the DC trains can run safely.

3.4.2 Hall Effect Sensors

Hall Effect sensors are mounted between the rails of each DC track. They are used to sense the presence of a DC train. The detection of a DC train signals to the microprocessor that the train has completed its loop. The power for these sensors are taken off daughter board #2.

3.4.3 Terminal Blocks

Terminal blocks are used to connect and to distribute AC, DC and TTL signals across the project board. Because lines longer than 8 inches are required to be twisted with their ground, the use of many terminal blocks allows thin wires to be kept short; they are connected to terminal blocks, whose opposing ends house thicker wire that can be easily twisted with ground wires.

3.4.4 10 kΩ Resistor and 0.13 μF Capacitor

A 10kΩ resistor is connected across the yellow and green AC lines at a terminal block such that any remaining charge on the insulated southbound track after the relay cuts power to that track is dissipated. A 0.13 μF can be found at the same terminal block to dampen any electrical spikes that may result from switching the power on and off.

3.5 Hardware Conclusions

Train Race uses relatively few components and has been tested to be robust. It has also been verified that power draw through the AC circuit is acceptable at all times, particularly during the data frame. The addition of 150 kΩ resistors to ground at the base of the transistor that receives the light in the opto-isolators resulted in a faster power cut at the data frame.

4. SOFTWARE

The code for controlling Train Race was written in assembly language for the 6502 processor and compiled with a XASM65 cross-assembler. The compiled code was then burned onto a 2764A EPROM chip using GTEK hardware and software.

The software is written in as simple and straightforward a manner as is possible. While increased simplicity can be gained by using an indexed addressing scheme, this improvement was not implemented due to the robustness of the existing code.

Section 4.2.6 shows a flowchart of our software design. Appendix E contains the user section of the software in its entirety.

4.1 Interfacing to the Outside World

The VIA (versatile interface adapter) chips are the key to bringing in data to the computer and sending meaningful instructions to the Train Race components. Each VIA contains two bytes of accessible memory. Before these memory locations can be used for sensing and actuating, they require initialization of the direction of the data, which is specified by additional byte on eachVIA. In Train Race, VIA 8000 is used for data related to the project board, with address 8001 (Port A) related to south track station optical proximity sensor and relay control, and address 8000 (Port B) related to the control of the DC trains.

Tables 3 and 4show the data direction registers and their contents for ports A and B on VIA 8000.

Bit (MSB to LSB)	-	-	-	-	-	-	Optical Sensor	Relay Control
Contents	0	0	0	0	0	0	0	1

Table 3. Location 8003 (Port A data direction register)

Bit (MSB to LSB)	-	-	-	-	Right Control	Left Control	Right Hall	Left Hall

Contents	0	0	0	0	1	1	0	0

Table 4. Location 8002 (Port B data direction register)

The following code initialization code fragment fully initializes the state of the VIA data bytes for the remainder of the program execution.

```
LDA #$01
STA $8003          ; data direction for VIA 8000 (Port A)
LDA #$0C
STA $8002          ; data direction for VIA 8000 (Port B)
```

Briefly defining the terms in Tables 3 and 4, the "relay control" refers to the state of the south track power kill control circuit. Stopping a train on the south track is performed simply by throwing the normally connected switch inside the relay to an open state. The signal to throw the relay comes from a low TTL signal at the LSB (least significant bit) at port B.

"Optical sensor" refers to the state of the optical proximity detector at the south track station. When a reflective object such as the tape on every AC train is brought to the 1/8'' focal length of the optical proximity detector, the normally high sensor state line becomes low. Thus, watching for the arrival of a train involves monitoring the state of port A's bit 1 for a 0.

"Left Hall" refers to the state of the left DC track's Hall Effect sensor; monitoring for the loop completion of the left DC train involves watching the LSB of port B for a 0. "Left control" will stop any train on the left DC track if a 0 is placed on bit 1 of port B. The same sensing and actuating bit assignments apply to the right DC track.

4.2 Program Execution

4.2.1 Initialization

The user section begins with the standard initialization, defining the location of the program on the EPROM, took place. This code is immediately followed by initialization of the data direction registers for both ports A and B of VIA 8000, along with code that ensures that the DC trains are initially inactive.

4.2.2 Main Sequence

The central program sequence is a routine called CHK, from which all other routines have their origin. CHK ensures that the relay starts in the normally closed position (power being supplied to the south track) while continually monitoring the south track station optical proximity detector for an arriving train. If no train is located, the routine loops back on itself until a train is detected. If the optical sensor state turns low (due to a train's arrival), CHK branches to the WAIT subroutine.

The WAIT subroutine continues to read the state of the optical proximity detector. If WAIT detects that the south track optical sensor returns to the high state, it branches to a subroutine calls KILL which cuts power to the south track by throwing the relay to the open position. In operation, this sequence – CHK, WAIT, and KILL – effectively cuts power to the southbound track when the optical sensor detects the *back edge* of an AC train's reflector. While at fast speeds the train's inertia propels it far past the south track sensor upon track kill, at slow speeds the possibility exists for a train to stop with its reflector directly in front of the optical proximity detector if front edge triggering is used, causing the sensor to send a "train present" signal. This would mean that, when the south track train is instructed to depart upon the next heat of the race, the microprocessor would erroneously think that an AC train has just arrived and not send out the AC train after the DC race. By triggering the relay upon detection of the reflector's back edge, the possibility for this immediate detection upon departure is eliminated. This solution is more elegant and robust than inserting software delays since the exact stopping location for the AC train is not essential in Train Race.

4.2.3 DC Engine Race

Of general note, applicable to the entire program, is the concept of writing only to bits of ports A and B that are necessary to complete the immediate action. This writing is accomplished by using

the ORA and AND statements to set and clear bits, respectively. This scheme allows for fewer errors, as there is no danger of setting or clearing a bit that is still required for another operation.

The completion of subroutine KILL is followed immediately by the LTIME subroutine, whose exclusive duty is to reset the state of the left DC train timer. The scheme used for train timing is now discussed.

A simple solution for obtaining relative times for the two separate software loops might seem to be a counter that is incremented each time the LTIME is executed. In fact, if the loops execute on a sufficiently short time scale, this solution is appropriate and sufficient. However, because the clock frequency of the 6502 microprocessor is 1 MHz, LTIME loop times are on the order of microseconds, which is far faster than the times of the macroscopic train race events. The timing scheme used in Train Race to compare the speed of the two DC engines is based on monitoring the pulses on the main track power waveform shown in Figure 6, which is on the order of milliseconds.

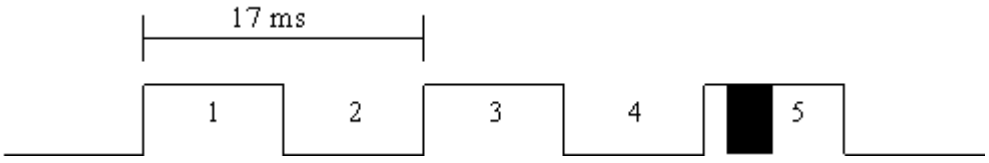


Figure 6. AC track waveform as generated by the Hornby System.

A pre-defined mailbox in the Arena software, FRANUM, allows the current programmer to monitor the current frame number on the main track waveform. Thus, a software counter uses the periodic cycling of FRANUM. The timing code for the left DC track is now presented and discussed.

```
LLOOK1 JSR LCHK          ; check state of train
       LDA FRANUM        ; timing loop - look for a frame number of 1
       CMP #$01
       BNE LLOOK1
       LDA #$01
       CLC
       ADC $02
       STA $02           ; increment counter when 1 is seen
       STA $4000
```

```
        LDA #$00
        ADC $01
        STA $01
LLOOK2  JSR LCHK          ; look for frame number of 2
        LDA FRANUM
        CMP #$02
        BNE LLOOK2
LL12    JSR LCHK           ; look for 1 again but do not increment counter
        LDA FRANUM
        CMP #$01
        BNE LL12
LL22    JSR LCHK           ; look for 2 again
        LDA FRANUM
        CMP #$02
        BNE LL22
        JMP LLOOK1         ; go back and do it again
```

Routine LLOOK1 (shown above) checks the current number on FRANUM and waits to see #$01 (frame number 1). When it does find a 1, it increments the low byte of a two-byte stopwatch for the left DC train. Figure 7 shows the zero-page memory locations for both the left and right DC train stopwatches.

Figure 7. Stopwatch memory locations for DC trains.

After the low byte of the stopwatch is stored to the display, the accumulator loads 0 and adds it the high byte of the stopwatch. While this initially seems to be a meaningless instruction, it is part of an elegant manner of holding a higher place in the stopwatch. When the low stopwatch byte becomes #$FF on the preceding ADC instruction, the carry bit is set. The complete ADC instruction, per the 6502 software design manual, is A + M + **C** $\rightarrow$ A. Thus, the high stopwatch byte will only increment when the low byte has reached its maximum, exactly like a real stopwatch.

To avoid continually adding to the stopwatch by continually detecting a 1 on FRANUM, the counter is diverted from incrementing on FRANUM = #$01 for 15 frames. This delay is accomplished by having a subroutine LLOOK2, which scans FRANUM until #$02 is found, and then calls subroutine LL12, which stands for "Left Look for 1 for the second time." When a frame number of 1 is detected, the program uses LL22 to search for the second occurrence of 2. Only when 2 is found for the second time does the program then search for 1, upon which the stopwatch is incremented once again.

The code is designed to combat the problem of DC trains stopping for periods of time along their race. A disastrous result could entail if only 1 byte of stopwatch were operated. The low byte of the counter for this train could very easily loop around back to, say, #$03, after reaching its maximum of #$FF. However, a critical amount of information is lost, namely that this train actually took longer than #$FF to complete the loop. If the other train legitimately completes its loop in #$45, the microprocessor would incorrectly determine that the former train won. Making the system completely robust to this sort of error is not possible without the addition of several sensors around the loop to monitor the train's progress. Additionally, adding more than one extra byte of stopwatch becomes difficult because only one carry bit exist, meaning that more sophistication is necessary to increment higher bytes. However, by slowing down the stopwatch by delaying incrementing to every second occurrence of frame 1, and adding a single additional stopwatch byte, we believe that the stopwatch is made sufficiently robust for practical purposes.

The stopwatch timing sequence of LLOOK1, LLOOK2, LL12, and LL22 periodically calls a separate routine called LCHK. This routine simply checks the state of the left DC track Hall sensor. If the sensor output is detected as a low, LCHK immediately calls LKILL, which sends a signal out to kill DC power to the left track, thereby stopping the left train. The power then switches to the

right DC track, and the software advances to RTIME, the analog of LTIME. Thus, the stopwatch timing sequence of routines is exited via LCHK.

Every routine above concerning the operation of the left track has an analog for the right track. The letter 'R' is simply substituted for the first 'L' in every subroutine name. The behavior of the left and right track trains are therefore identical. However, the end of RKILL allows the software to continue onto the COMP routine, which begins the determination of the winning DC engine.

4.2.4 Having the "Judges" Decide the Winner

By way of reminder, a win by the left DC train means that the train waiting at the south track station will depart at speed 14, while a win by the right DC train signals the AC train to depart at a slower speed 10. To allow the AC train to depart fast at speed 14, depart fast, a subroutine, FOUT, is executed; SOUT is executed for slow departure. These subroutines will be discussed shortly.

As previously mentioned, the completion of RKILL is the completion of the train race, and COMP is called to determine a winner. The subroutines COMP and GRLS (GReater than or LeSs than) constitute the necessary subroutines for determining which DC train completed its loop faster. The two routines are shown below

```
COMP    LDA $03          ; see which train won
        CMP $01
        BNE GRLS         ; go to greater/less than if high bytes not equal
        LDA $04          ; compare low bytes
        CMP $02
GRLS    BCS FOUT
        BCC SOUT
```

The scheme for determining the winner is as follows: if the right DC train's stopwatch high byte (location $03) is not equal to the left engine stopwatch high byte (location $01), the carry would be set if $03 > $01 and cleared if $03 < $01. Thus COMP branches to GRLS, where the appropriate signal to the AC train is determined. If the high stopwatch bytes for both trains are equal, then BNE GRLS is skipped, and the low bytes are compared. This comparison would perform the same

operation with respect to the carry bit. That is, if the right DC train's low stopwatch byte (location $04) is higher than that of the left DC train (location $02), then the left train has 1, the carry is set, and the AC train departs at high speed (subroutine FOUT).

4.2.5 A "Smart" Departure Routine

To add another layer of robustness to our project, we wanted any AC train that enters the south track station to depart at the correct speed as determined by the results of the race. Since setting the speed of any train on the Hornby system requires input of the associated train number, the program must know the number of the train that is waiting at the south track station. Fortunately, this information is readily available from the mailbox STRAIN. The task is to obtain this information and create a unique code for the Interactive Controller to tell that train to depart at the proper speed as defined by GRLS.

For the AC train to leave at the appropriate speed as a result of the train race, a binary instruction scheme is required; that is, train 1 could leave the station at either speed 14 or speed 10, train 2 could leave at either speed 14 or speed 10, and so on. Thus, the following coding scheme is used:

Waiting AC Train #	Winning DC Train	Code to Int. Con.
1	Left	01
1	Right	10
2	Left	02
2	Right	20
…	…	…
8	Left	08
8	Right	80
…	…	…

Table 5. Interactive controller coding scheme.

The meaning of the codes meant for the interactive controller will be explained in the next section; for now, it suffices to point out that a mapping scheme is created such that any possible

combination of a winning DC train and a waiting AC train is uniquely coded. The following two routines were used to implement the coding scheme.

```
INTF  LDA STRAIN      ; send info to correct train for FOUT
      AND #$0F
      STA $05
      RTS
INTS  LDA STRAIN      ; send info to correct train for SOUT
      ROL A
      ROL A
      ROL A
      ROL A
      AND #$F0
      STA $05
      RTS
```

These routines were called by either FOUT or SOUT as indicated in the comments. Basically, a fast departure requires the south track train number as the low four bits of Interactive Controller instruction, and a slow departure required the train number as the high four bits. Thus, INTF takes STRAIN, ensured that the four high bits are zeros and hen writes to location $05, the universal location for a code going to the Interactive Controller. INTS rotates STRAIN such that the four low bits become the four high bits, performs an AND instruction to clear the four low bits, and writes the result to location $05.

Finally, FOUT and SOUT, after calling INTF and INTS, respectively, uses the data written to $05 to send to the Interactive Controller. The subroutines indicate the winner by displaying a #$10 for a left win and #$01 for a right win. Finally, the overall sequence is completed by calling the original routine CHK, which activates the AC train waiting at the south track station.

4.2.5 Interactive Controller Codes

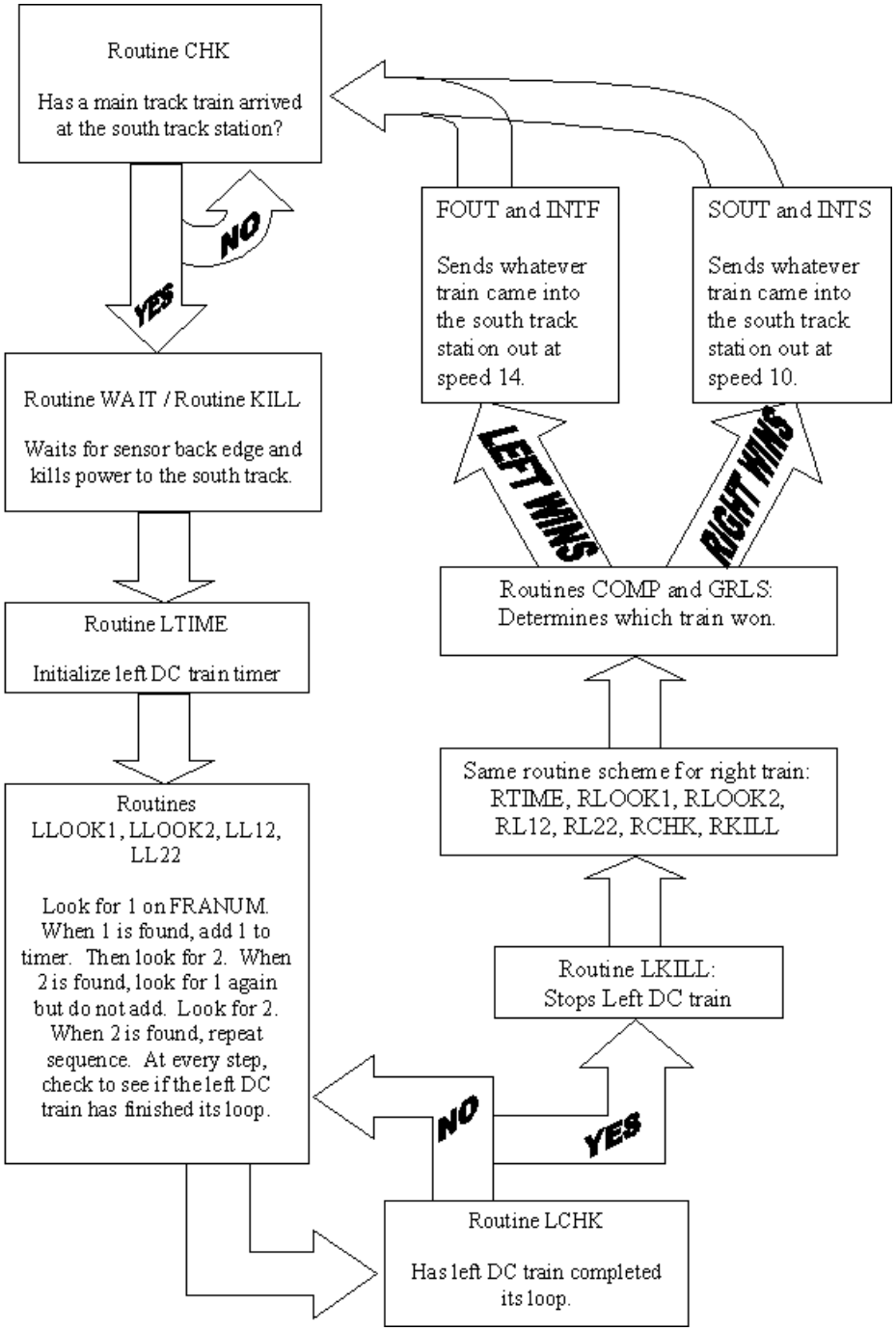
The Interactive Controller provides a mechanism by which the microprocessor could communicate directly with the Hornby control unit. A command code file called ‘race’ has been written and copied to all Interactive Controllers. The file consists of the following instructions

```
CODE  #$01        E1 S14 DF C03D01
      #$10        E1 S10 DF C03D01
```

```
#$02        E2 S14 DF C03D01
#$20        E2 S10 DF C03D01
. . .       . . .
#$08        E8 S14 DF C03D01
#$80        E8 S10 DF C03D01
```

Thus, the code that is stored at location $05 on the microprocessor is sent to the Interactive controller, which implements the coding scheme described in the previous section.

4.2.6 Operations Flowchart



4.3 Software Conclusions

The software for our project was successful insofar as its robustness of operation and ease of comprehension. This is not to say that improvements cannot be made. This section will outline a few of these potential improvements.

As a very simple improvement to implement, the program could cycle, upon completion of either the FOUT or SOUT routine, to a subroutine immediately preceding CHK, whose only responsibility would be to re-activate the south track. Currently, this action is performed by CHK. However, CHK continues to loop on itself thus unnecessarily continuing to activate the south track, an action that only needs to be performed once.

Furthermore, with some attention, the code related to the DC train timing could be made more elegant by implementing the concept of indexed indirect addressing. Suppose that address $06 contains the address of the low stopwatch byte. After the AC train arrives at the south track station, imagine a routine SETL being called. The essential code fragment is shown below.

```
SETL  LDA $02
      STA $06     ; contains location of left stopwatch low byte
```

Now, instead of having separate timing loops for each train, we would only have a single timing loop. Consider the first section of that loop:

```
LOOK1 JSR TCHK          ; check state of train
      LDA FRANUM        ; timing loop - look for a frame number of 1
      CMP #$01
      BNE LOOK1
      LDA #$00
      TAX
      LDA #$01
      CLC
      ADC ($06,X)
      STA ($06,X)       ; increment counter when 1 is seen
...
```

Now, imagine that after the left DC train has completed the loop, a routine SETR is called.

```
SETR  LDA $04
      STA $06     ; contains location of right stopwatch high byte
```

Thus, we have effectively turned the timing loop into a function, which takes as its argument an address containing the address to be updated as the stopwatch. We can see that, due to the execution of indexed addressing, location $07 would need to be set to zero in the initialization section of the program, such that the overall address accessed by the computer on the ADC and

STA commands would be $0006. Notice that a generalized procedure TCHK ("train check") would have to replace the current LCHK and RCHK. TCHK would first compare $06 to $02. If this comparison showed that the two were equal, TCHK would scan the left Hall sensor. To regain full functionality with this new scheme, indexed indirect addressing would need to be implemented for a high byte of stopwatch. With this indexed indirect scheme, the elegance of the program is increased as the timing loop is coded as a single function, which effectively receives input.

CONCLUSIONS

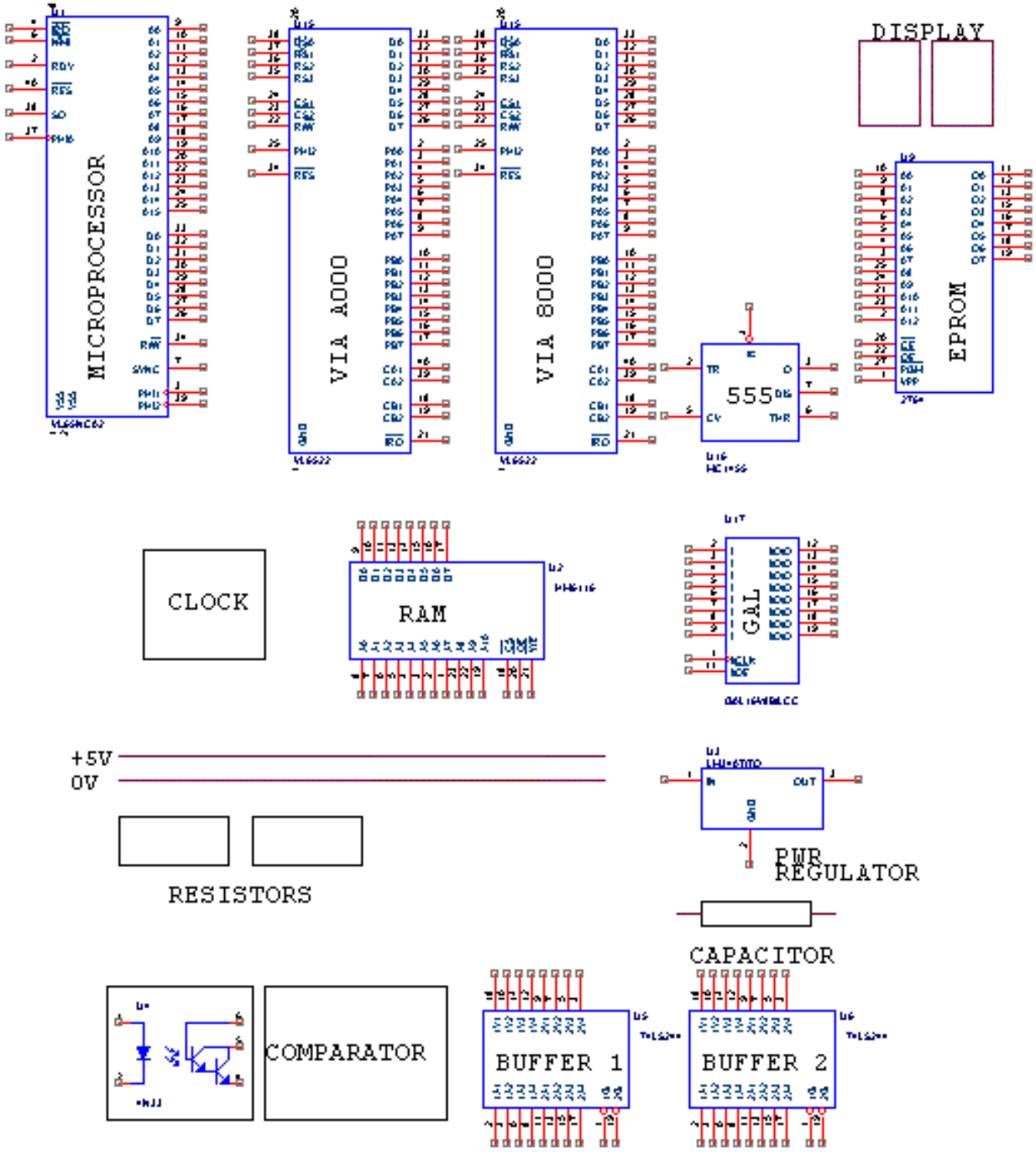
Train Race includes a variety of sensors and switches which the authors have learned to use effectively but is otherwise simple and robust. Of the constituents of Train Race, perhaps the more difficult aspects included coming up with a robust timing scheme for the two DC trains as well as a simple way of controlling the AC train by communicating with the Hornby System via interactive control.

Suggestions have been made in earlier sections as to how Train Race can be made more elegant, but Train Race successfully meets the requirements of using microprocessor control to sense, sequence and actuate model trains.

APPENDICES

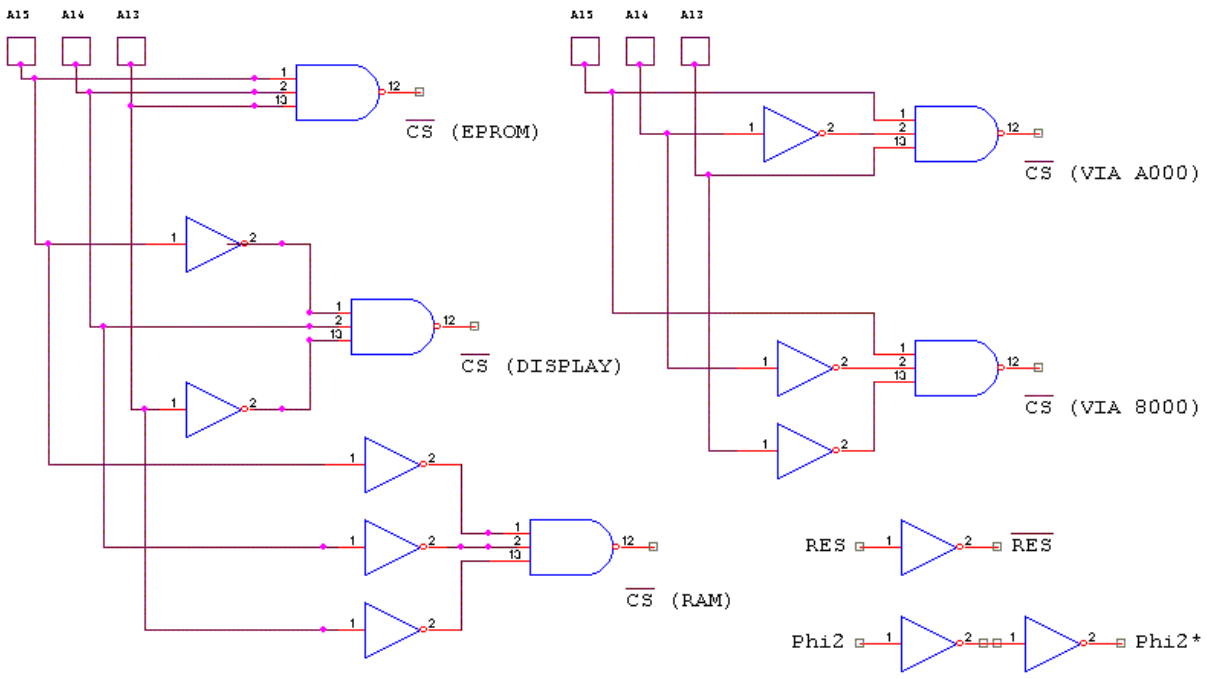
Appendix A: Train Race Project Board Layout

Appendix B: Microprocessor Layout

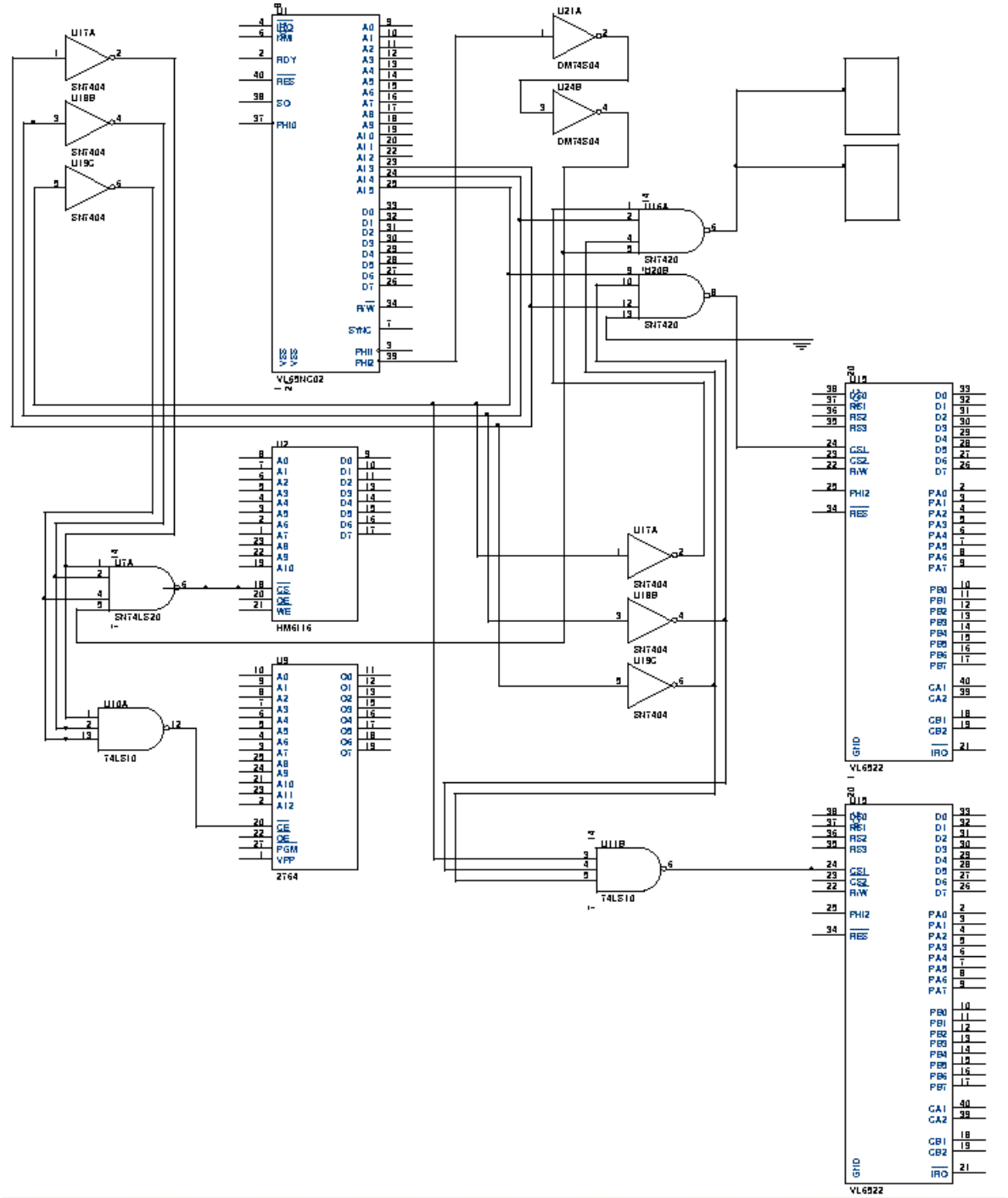


Appendix C1: Address Decode Logic Worksheet

Appendix C2: Address Decode Logic Schematic



Appendix C3: Schematic for ADL Interface to Computer



Appendix D1: Daughter Board #1 Layout and Schematic

Appendix D2: Daughter Board #2 Layout and Schematic

Appendix E: User Section of the Software

Note that this appendix does not contain additional code that was used for debugging in the early stages.

```
;   **** BEGINNING OF USER SECTION ****
            ORG $e000
tstart  sei
      cld
      ldx #$ff
      txs
      lda #$03       ; this is computer #3
      sta blkid
      lda #$00
      sta dflag
      lda #$00
      sta novrd
      sta sovrd
      jsr init
;
      LDA #$01        ; data direction for VIA 8000 (port A)
      STA $8003
      LDA #$0C        ; data direction for VIA 8000 (port B)
      STA $8002
      LDA #$0C        ; inactive DC trains
      ORA $8000
      STA $8000
CHK   LDA #$01
      ORA $8001
      STA $8001       ; Relay off -> power initially to track
      LDA $8001       ; monitors optical state, waits for 0
      ROR A
      AND #$01
      CMP #$00        ; there's a train
      BEQ WAIT
      JMP CHK
WAIT  LDA $8001       ; waits for sensor back edge
      ROR A
      AND #$01
      CMP #$01        ; train's back edge
      BEQ KILL
      JMP WAIT
KILL  LDA $8001       ; kills power to south track
      AND #$FE
      STA $8001
      LDA #$FB
      AND $8000
      STA $8000       ; activate left train only
LTIME LDA #$00        ; reset left loop timer
      STA $01
      STA $02
      JMP LLOOK1
LCHK  LDA $8000       ; check for left train at hall sensor
      AND #$01
```

```
      CMP #$00
      BEQ LKILL
      RTS
LKILL LDA #$04
      ORA $8000
      STA $8000       ; kill right train
      LDA #$F7
      AND $8000
      STA $8000       ; activate right train only
RTIME  LDA #$00         ; reset right loop timer
       STA $03
       STA $04
       JMP RLOOK1
RCHK  LDA $8000        ; check for right train at hall sensor
      AND #$02
      CMP #$00
      BEQ RKILL
      RTS
RKILL LDA #$08         ; kill right train
      ORA $8000
      STA $8000
      JMP COMP
LLOOK1 JSR LCHK         ; check state of train
       LDA FRANUM       ; timing loop - look for a frame number of 1
       CMP #$01
       BNE LLOOK1
       LDA #$01
       CLC
       ADC $02
       STA $02          ; increment counter when 1 is seen
       STA $4000
       LDA #$00
       ADC $01
       STA $01
LLOOK2 JSR LCHK         ; look for frame number of 2
       LDA FRANUM
       CMP #$02
       BNE LLOOK2
LL12   JSR LCHK          ; look for 1 again but do not increment counter
       LDA FRANUM
       CMP #$01
       BNE LL12
LL22   JSR LCHK          ; look for 2 again
       LDA FRANUM
       CMP #$02
       BNE LL22
       JMP LLOOK1        ; go back and do it again
RLOOK1 JSR RCHK          ; same as above for right DC train
       LDA FRANUM
       CMP #$01
       BNE RLOOK1
       LDA #$01
       CLC
       ADC $04
       STA $04
       STA $4000
       LDA #$00
```

```
       ADC $03
       STA $03
RLOOK2 JSR RCHK
       LDA FRANUM
       CMP #$02
       BNE RLOOK2
RL12   JSR RCHK
       LDA FRANUM
       CMP #$01
       BNE RL12
RL22   JSR RCHK
       LDA FRANUM
       CMP #$02
       BNE RL22
       JMP RLOOK1
COMP   LDA $03          ; see which train won
       CMP $01
       BNE GRLS         ; go to greater/less than if not equal
       LDA $04          ; compare low bytes
       CMP $02
GRLS   BCS FOUT
       BCC SOUT
FOUT  JSR INTF
      LDA $05        ; fast out left train wins
      STA WDATAH
      LDA #$FF
      STA DFLAG
      LDA #$10
      STA $4000
      JMP CHK
SOUT  JSR INTS
      LDA $05        ; slow out right train wins
      STA WDATAH
      LDA #$FF
      STA DFLAG
      LDA #$01
      STA $4000
      JMP CHK
INTF  LDA STRAIN      ; check state of strain and send info to correct train
for fast out
      AND #$0F
      STA $05
      RTS
INTS  LDA STRAIN      ; check state of strain and send info to correct train
for slow out
      ROL A
      ROL A
      ROL A
      ROL A
      AND #$F0
      STA $05
      RTS
;   **** END OF USER SECTION ****
```