Interactive Control

Completing the Loop for Total Automation

by
Michael J. Arena
May 29, 1985
MAE 412
Professor Michael Littman
ABSTRACT

A software and hardware interface between the block computers on the main line and the HORNBY Zero-i controller. The block computers are already able to receive information from the HORNBY controller via the interrupt service routine. So this project enables the total automation of the main line without the need for a person to monitor the system.
INTRODUCTION

The automation loop involves many intermediate steps. The block computers send their sixteen bits of return information to their signal sensor boards. These boards are all attached to a single bus called the Return Information Bus. During the data frame of the HORNBY power cycle, the signal sensor board sends the information to the 6502 based project I designed. My project then sends the information via an RS-232 300 baud interface to the Macintosh computer. The Macintosh then processes the information and determines the appropriate actions to be taken which results in a string of characters. The string is then sent over an RS-232 300 baud interface to Phil Dworsky's project which interprets the characters in the string and the simulates the keypushes on the HORNBY controller unit. The controller then sends information out to the tracks to control the locomotives and talk to the block computers. And finally, the block computers read information off of the tracks via Lecky's interrupt service routine.
HARDWARE

The board I developed uses a 6502 microcomputer to manage shift registers for the Return Information Bus and an ACIA to manage the RS-232 port.

Shift Registers and Logic

A comment about the hardware in general: because of the timing restrictions and noise problems, the exact chip numbers and types should be used if the project is to be copied. Do not substitute LS for S chips, etc. because the timing will not remain within specifications.

The address decoding logic is drawn in Figure #1 on the following page. All chips become enabled with an active low signal. The RAM and LEDs do not directly accept $\phi_2$ as an input (as does the VIA, and the ACIA) so the output of the LS138 is OR'ed with NOT $\phi_2$. When $\phi_2$ goes from low to high (and NOT $\phi_2$ goes from high to low which enables the chips), this becomes a reference for the write timing for these devices.

The data that the HORNBY controller sends during a data frame consists of 34 bits of information. Each bit is frequency shift keyed so each bit is a full square-wave cycle (see Special Note below). I use one of the JK flip flops (configured as a T flip flop) in the 7476 as a divide-by-2 counter since one bit of return information is sent for every two bits of track information. I use two 8 bit shift registers (7491A) and the other JK flip flop (configured as a D flip flop) to hold the
seventeenth bit. Figure #2 on the following page shows the track signal and the corresponding clock signal for shifting the registers. A high-to-low transition clocks the shift registers and flip flop. The XOR gate (LS86) that goes into the clock flip flop is used to account for different track polarities. PA0 is set low if the transition of the first bit in the data frame goes from high to low and set high for the other polarity. Therefore, the first transition of the track signal always creates a high-to-low transition into the clock the flip flop. At the end of the first bit of data, there will be another high to low transition so the flip flop will toggle to low thus triggering the shift registers.

The output of the clock flip flop is NAND'ed with PB7 which is high during a data frame and low otherwise. So the shift registers can only be triggered by the track while in a data frame. Finally, the output is NAND'ed with PA1 which is normally high and is pulsed low when the software is reading in the shift registers after the data frame.

Special Note: The drawing for the track signal in Figure #2 shows 35 low-to-high transitions and 34 high-to-low transitions. Since the clock flip flop is clocked on the first transition (low-to-high which is converted to high-to-low by the XOR gate), it should get 35 pulses but the output of the clock signal shows only 34 transitions. Professor Littman believes that it is possible that the last cycle bit of the data frame does not exist so that would account for the output.
Logic Diagrams

Figure 1

Figure 2

NOTE: This last cycle may not actually exist
ACIA

This operation is much simpler. The logic signal from the 6522 goes into an MC1488 which makes it the correct voltage levels for RS-232 communication. The MC1488 needs a negative power supply so an ICL7660 is used which converts +5 volts into -5 volts. The MC1488 also needs a positive voltage greater than +5 volts so the unregulated +9 volts from the supply is used. For receiving RS-232, I copied the circuit that the AIM-65 uses. If you look on the Schematic Diagram, you can see that the input line goes between two diodes; one from ground to the line and the other from the line to +5 volts. This properly converts the usual ±12 volts.

SPECIAL

1) There are two regulators for the project. One drives all the logic components and the other drives only the LEDs. This regulator’s output is denoted by +5L on the Schematic Diagram.

2) Because the LEDs draw so much current, a large 9 volt power supply that can supply at least one amp should be used.

3) The flip flops are very sensitive to spikes so pull up resistors and capacitors were used for all lines requiring a logic "1" (i.e. +5 volts)

4) When powering up the components of the system, there is a preferred order. First power the HORNBY controller, then all the block computers, then my project computer, and finally start the Macintosh program running. This helps minimize garbage coming
up on the Return Information Bus.

5) The blue and white wires coming out of the 44 pin connector on my project are the Return Information Bus. The blue wire is GROUND and should be connected to pin 17 on the signal sensor board's 44 pin connector. The white wire should connect to pin 16 of the signal sensor board. NOTE: Most signal sensor boards are missing the 4N33 that drives the Return Information Bus so this may need to be installed. See the layout of the signal sensor board in the train lab for more details.
SOFTWARE

software for the interface is composed of two parts: a program in the EPROM of my project computer, and a BASIC program that runs on the Macintosh.

Interactive Control Program
in EPROM

listing with a line by line description of operation can be found at the end of this report. The general functioning of the program is as follows. During frame #1, the sixteen bits in the shift registers are read. Then two flags are checked to see if special processing needs to be done. The first one is a flag that says that the information just read in was the return information from a poll request of all computers. Therefore, set the lowest bit of the 16 bits to a "1" and send it. The MacIntosh checks that the lowest bit is set otherwise it assumes that the information just received was garbage. The next flag that is checked if the first wasn’t true is one of the Lecky mailboxes INTMPU. This signal says that the following data frame will contain return information from a poll request. Therefore, set the poll request flag for next time, throw away the 16 bits just read in, and wait for frame 4. The reason for throwing away the information just read, is that it takes 53 milliseconds to send two bytes of data at 300 baud. Since each frame spans approximately 8.3 milliseconds (60 Hz), the 5 frames span 41.5 milliseconds. Therefore, if the information was sent, the
program would miss the data frame with the return information by the time it returned from the sending routine. If neither of the flags were set then both bytes are checked for being zero. If they are both zero then they are not displayed or sent. Finally, the program waits for frame 4 where it clears the flip flops and prepares for the following data frame.

**Hornby Controller Program**

The listing for this program can also be found at the end of the report. The program has three main parts: the first controls manual operation, the second controls automatic operation, and the third handles editing of command tables. When the program starts, it asks you to edit and/or load the command tables and then hit return. This allows you to set up the system before beginning operation. You can save or load command tables by selecting the FILE menu and the appropriate field. The list of files on the disk is displayed and you can select on by clicking it twice. (In the case of SAVE, if you want to save the table under a new name then hit cancel and a new prompt box will appear with a field to fill in the name from the keyboard.)

When you hit RETURN, you enter the MANUAL mode of operation where the user directly controls the trains instead of the block computers. To get to the AUTOMATIC mode of operation, select the MODE menu and the Automatic field and then click one of the "buttons" on the screen. To get from AUTOMATIC mode to MANUAL mode simply select the Manual field of the MODE menu.
Manual Mode

The screen dump on the following page shows the basic layout for the manual mode screen and the editing screen (each has a few differences). Each little box on the screen is a BASIC BUTTON. Clicking one of these "buttons" with the mouse is equivalent to pressing the buttons on the HORNBY controller. There are some buttons that are different or missing from the HORNBY layout. There is no PANIC, CLEAR, or SLAVE keys. But I have added INCrement Speed, DECrement Speed, and TOGGLE direction for the convenience of the user. The program stores the direction and speeds of each train in arrays, so hitting TOGGLE will cause a lookup of the direction of the ActiveLoco and flip that bit in the array. It then determines whether to send a FORWARD or REVERSE character. Also, the program keeps track of the ActiveLoco by remembering the number of the last LOCO entered. NOTE: Even though the user can control more than one loco with the "slide", the program only remembers the last one.

Automatic Mode

The basic idea behind automatic operation is this: each computer has an ID number (1-9). The return information consists of two bytes. Each byte is an index into an array of 256 command strings. Each command string contains a sequence of keypushes of the HORNBY keypad to be executed.

In Automatic mode, the MacIntosh can handle up to 9 block computers. It first sends out a poll request to all computers.
Editing table for
Computer # 1

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Inertia</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>LOCO</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>Toggle</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>INC Spd</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Reverse</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>DEC Spd</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>Forward</td>
<td>1</td>
</tr>
</tbody>
</table>

Current string for command BD:  <1>;M

New string:  <1>;M
Then it waits for two bytes of return information. It then checks to make sure that the lowest bit was set by my computer. Then it determines which computers need servicing. It then polls each computer in turn for its return information. It then waits for two bytes of information from the Current Computer being polled. Then it uses the high byte as an index into the Current Computer's command table. The string returned is then executed (if not empty) by deciphering the the string, setting appropriate values for ActiveLoco and the arrays Direction, Speed, CompCurrLoco. ActiveLoco, Direction, and Speed were described above. CompCurrLoco is the number of the loco that a particular computer is controlling. The reason this is needed is demonstrated by the following example:

There are two computers on the main line, #3 and #5. Both have return information. The Mac polls #3 first. This computer wants to set LOCO 7 to Speed 2. Then the Mac polls #5. It wants to set LOCO 1 in REVERSE at Speed 14. The Mac then polls both computers again. Computer #3 has return information. The Mac polls #3 and receives the command to INCrement Speed. The Mac checks to see whether the ActiveLoco is the same as the #3 Computer's Current Loco (in the array CompCurrLoco) Since the ActiveLoco is number 1 (set by computer #5), the Mac first sends out a string setting the ActiveLoco to 7 and then determines that the new Speed should be 3 and sends that string out.

This situation allows the computers to act independently of other computers on the line since they don't have to know about current locos and such. However, a problem could arise if two computers are controlling the same LOCO. One computer could set it to REVERSE and the other to FORWARD so neither project would
function properly. But you could modify this program to save speed and direction in more arrays for each computer if this becomes a real necessity.
EDITOR

The reasons behind using tables of command strings instead of each user writing his own subroutine in BASIC are numerous. First of all, since everyone uses a table with 255 command strings (00 is not an allowed command), this helps standardize system. Also, user written subroutines might affect the operation of other subroutines. It also facilitates the uses of multiple user systems. Also, the user does not have to know anything about BASIC programming or what characters to send, etc. They only need to know how to use a HORNBY keypad.

When you select the EDIT menu, the screen depicted earlier is displayed. The program then prompts you for the computer number of the table you want to edit. Select a new command byte by hitting the COMMAND button. You are prompted for a 2 digit number. The old value of the string is displayed (which will probably look incomprehensible since it is a string of characters which each key translates. See table of keys and characters at end of report.) Enter a new sequence of keys by clicking the "keys" of the HORNBY layout. The speed slide is replaced by 15 distinct speed settings. If you make a mistake, use the DELETE button to erase the last character of the New string. When finished entering the string, hit OK to save this new string or CANCEL to forget it. NOTE: While entering a new string, the DONE and COMMAND buttons are disabled so that you do not accidentally forget to save a newly entered string. Clicking either OK or CANCEL re-enables them. Hit DONE when you are ready to return to the normal operation of the program.
SPECIAL

1) A problem can occur if the program is in Automatic mode and the track is shorted. The MacIntosh will have sent a poll request and then wait for two bytes of information. Since the block computers never received the request the Mac will wait forever. But there are two ways to fix this. On the HORNBY controller, hit CLEAR and then enter a poll request by hitting "0 1 ->". If this doesn't work then you might reset my project then repeat the above procedure. A second method can be done by switching to Manual mode on the Mac and then back to Automatic mode (after CLEARring the HORNBY). This will send a new poll request. Again, you might have to reset my project.

2) Also, the Mac might hang if the return information sent by a block computer is all zeros. My project never sends return information to the Mac if both bytes are zero. My only suggestion to prevent this is to enforce the rule that at least one byte of return information has to be non-zero. In fact, I suggest that the normal format for return information should be to set the high byte to zero and the low byte to the desired command. If, however, you need to execute a very long command string at one time then you could send two commands. The high byte would contain the code for the first half of the command string and the low byte would be the code for the continuation of the command string. See note 4) below.

3) As a convention for naming command tables stored on disk, I use the descriptive name of the project followed by the
computer number it is to be loaded into. For example, the two
that are on the "Mike Arena" disk are "Busloader4" and
"Turntable!". So the project that controls the turntable would
be loaded as the first table in the array of nine tables.

4) Also as a convention, I decided that each command string
stored in the command tables could be 20 characters long. Each
character is equivalent to one keypush. This makes the size of
the table when saved to disk 5k. The two tables saved on disk
are in this format so if you need to expand the size of a string
then I suggest linking two bytes together rather than changing
the size of the strings. The Mac executes the high byte's string
first and then the low byte's string so you could consider this
one string of length 40.

5) Since the array of tables in the Mac program is 9x256x20,
this takes up 45k of RAM, so the program can only be run on a Mac
with 512k.

6) While in manual mode, ignore the buttons OK, CANCEL,
DONE, COMMAND, and DELETE. They only have meaning in the editor.

7) When the Mac executes a command string, it deciphers the
string and prints on the screen what operations are being
performed. But then I never bothered to clean up the screen for
the next command so this could be fixed in the program.
CONCLUSION

The project works very reliably and performs all the functions that I set out to do at the beginning of the semester. The automatic system has been tested with only one block computer at a time, but I assume that it will work the same with multiple computers.

Since the main software runs on a MacIntosh, it will be very easy to modify the program to suit the needs of future projects. New "keys" could be added to the HORNBY layout to incorporate more functions. I hope that this project will prove to be extremely useful to future students in the course.

Michael J. Arena
List of HORNBY keys and their equivalent characters in the program

(This is from Phil Dworsky's report except for my extra keys)

<table>
<thead>
<tr>
<th>Speed</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>G</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>7</td>
<td>L</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
</tr>
<tr>
<td>9</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>N</td>
</tr>
<tr>
<td>11</td>
<td>J</td>
</tr>
<tr>
<td>12</td>
<td>K</td>
</tr>
<tr>
<td>13</td>
<td>I</td>
</tr>
<tr>
<td>14</td>
<td>H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HORNBY Key</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>INERTIA</td>
<td>:</td>
</tr>
<tr>
<td>REVERSE</td>
<td>;</td>
</tr>
<tr>
<td>FORWARD</td>
<td>?</td>
</tr>
<tr>
<td>LOCO</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>=</td>
</tr>
<tr>
<td>-&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>INC Spd</td>
<td>+</td>
</tr>
<tr>
<td>DEC Spd</td>
<td>-</td>
</tr>
<tr>
<td>TOGGLE</td>
<td>~</td>
</tr>
</tbody>
</table>
Interactive Control Program

in EPROM

Start

SEI ; The next 4 instructions are the usual
CLD ; initialization instructions
LDX #FF
TXS
LDA #01 ; Set computer number to 1
STA 03FA ; Mailbox for computer number
STA 0200 ; Location 0200 holds the bit for PA0
           ; which is either set or cleared depend-
           ; ing on the polarity of the track.
           ; This is explained in the writeup
JSR FB00 ; Jump to initialization routine
LDX #F7 ; Mask for direction of Port A on VIA
STX A003 ; Sets all but PA3 to outputs
JSR InitACIA ; Initialize the RS232 port
LDA #00 ; Location 0205 is a flag to signal that
STA 0205 ; the next data frame will contain the
           ; return data from a poll request
LDA 03F5 ; Load the PLRTY mailbox
BEQ MainLoop ; If zero then start main loop
LDA #00 ; Else zero out location 0200
STA 0200

MainLoop

LDA 0200 ; Load bit for PA0
ORA #02 ; Set PA1 high and PA2 low
STA A001 ; Store in Port A's data register
ORA #06 ; Bring PA2 back high. Pulsing PA2
           ; clears the JK flipflops before reading
           ; return information
STA A001

WaitFor

LDA 03FA ; Load frame number
CMP #01 ; Is it frame #1?
BNE WaitFor ; If not then loop
JSR GetByte ; Get first byte of return info.
STA 0202 ; Returned in A register. Location 0202
           ; is the high byte of return information
JSR GetByte ; Get next byte
STA 0201 ; Store low byte of return information
LDA 0205 ; Load poll request flag
BNE PollData ; If set then got to special routine
LDA 03F8 ; Load INTMPU mailbox. This mailbox is
           ; set if Hornby just sent a poll request
BEQ CheckByte ; Not a poll request
LDA #FF ; It was a poll request so set poll
STA 0205 ; request flag and
BNE AfterSend ; Jump past section that transmits info
BEQ AfterSend ; since we must catch next data frame

CheckByte

LDA 0201 ; Load low byte of return info
ORA 0202 ; Or with bits of high byte
BEQ AfterSend ; If both zero then don't display or send

Display+Send

LDA 0201 ; Else display low byte

195
STA 2000 ; at lower two LEDs
LDA 0202 ; Display high byte
STA 4000 ; at middle LEDs
JSR SendInfo ; And transmit both bytes over RS232

AfterSend
LDA 03FB ; Load this computer's RDATA mailbox
STA 6000 ; and display at higher LEDs

WaitFor4
LDA 03FA ; Load frame number
CMP #04 ; Is it frame #4?
BEQ MainLoop ; Yes, so got to main loop
BNE WaitFor4 ; No, so try again

This routine reads 8 bits from the shift registers and puts them in A

GetByte
LDX #00 ; Clear all registers
LDY #00
LDA #00

GetLoop
CLC ; Clear carry bit that gets shifted into A
TXA ; X temporarily holds byte. Transfer to A
ROL A ; then shift A left
TAX ; Save in X
LDA A001 ; Load top bit of shift registers into A
AND #08 ; Clear all bits but PA3
BNE zzz ; If bit was high then leave low bit of X
; cleared since incoming bits are inverted
TXA ; Else load temporary byte
ORA #01 ; Set low bit
TAX ; And save again

zzz
LDA 0200 ; Load bit PA0
ORA #04 ; Pulse bit PA1 low which shifts the
STA A001 ; shift registers one bit to the left
ORA #06 ; Bring it back high
STA A001
INY ; Increment number of bits we've read
CPY #08 ; Have we read 8 bits?
BNE GetLoop ; No, so get next bit
TXA ; Yes, so put byte in register A
RTS ; And return

; This routine initializes the ACIA to 300 baud, No parity, 8 bits,
; 1 stop bit, and enables transmitter and receiver operation even though
; the receiver is not used

InitACIA
LDA #0B ; Mask for
STA C002 ; Command register
LDA #16 ; Mask for
STA C003 ; Control Register
RTS ; And return

; This routine sets the low bit of the low byte of return information
; to a 1 so that the Macintosh knows that this is valid return data from
; a poll request.

PollData
LDA 0201 ; Load low byte of return info
ORA #01 ; Set low bit
STA 0201 ; Save it
STA 6000 ; Flash low byte to LEDs
NOP ; Next 3 NOPs were to zero out an
NOP ; instruction. Just ignore them.
NOP
LDA #00 ; Load zero to clear
STA 0205 ; poll request flag
STA 03F8 ; and INTMPU mailbox if set
JMP Display+Send ; Display poll info then send it

; This routine sends the low byte then the high byte of return info
; over the RS232 line.

SendInfo
LDA C001 ; Load Status register of ACIA
AND #10 ; Mask out all but Transmit Data Register
          ; Empty bit
BEQ SendInfo ; Can't send new info yet
LDA 0201 ; Load low byte of return info
STA C000 ; Store in Transmit Data Register

Send1
LDA C001 ; Wait as above
AND #10
BEQ Send1
LDA 0202 ; Load high byte of return info
STA C000 ; And send it

Send2
LDA C001 ; Wait as above
AND #10
BEQ Send2
RTS ; And return
REM --- Hornby Controller Program
REM --- by Mitie Arena

REM --- "Table" is a 2 dimensional array of strings
REM --- Each of the 8 possible computers running on the system
REM --- can have 255 (i.e. one byte) command strings
REM --- Each command string can be MAXLEN characters long
DIM table$(9,256)
MAXLEN = 20 'NOTE: All files must have records of the
' same length so this constant should probably
' not be changed.

REM --- NOTE: The first element is not used in the following arrays
DIM Speed(17)
DIM Direction(17)
DIM ComeciLoco(10)
DIM request(10)
ActiveLoco = 1 'Current loco being controlled by Hornby

REM --- These are the characters that Phil Dworsky's project
REM --- accepts as the keys and slide on the Hornby unit
OUTPORT$ = "7410652953xx++-7HJKUSNLDDEGB5CA"
filename$ = ""
poll$ = "0 1 x" 'Used to poll all computers
manual = 1
automatic = 0
Node = manual 'This determines whether you are in manual or automatic mode

GOSUB init
LOCATE 4,5: PRINT "Load tables and/or edit tables then";
LOCATE 5,5: PRINT "Hit return when ready to start";

st$ = INKEY$
IF st$ = "" THEN GOTO st
GOTO ManualMode

REM --- This routine waits for a software button to be pushed and grabs any
REM --- return information from the RS232
GetButton:
dialogId = DIALOG(0)
IF LOC(1) = 0 THEN GOTO gb1
r1$ = INPUT$(1,1)
r2$ = INPUT$(1,1)
LOCATE 16,15
TEXTFACE 1
PRINT HEX$(ASC(r2$));
PRINT "",HEX$(ASC(r1$));
TEXTFACE 0
gb1:
IF dialogId <> 1 THEN GOTO GetButton
```
ButtonId = DIALOG()
IF buttonId = 79 THEN key$ = ""; GOTO gb2
IF buttonId = 53 THEN GOTO GetButton
key$ = MID$(tr, buttonId, 1)

LOCATE 14,15; PRINT key$;
RETURN

ManualMode:
WINDOW 4,,(10,20)-(750,450),2
GOSUB SetupEditScreen
LOCATE 1,1; PRINT "To get to AUTOMATIC mode, set MODE";
LOCATE 2,1; PRINT "to automatic then hit any 'button' on";
LOCATE 3,1; PRINT "the screen:";
LOCATE 14,7; PRINT "Transmit:";
LOCATE 15,3; PRINT "Received:";

ManualLoop:
GOSUB GetButton
IF Mode = manual THEN GOTO manual: "If Mode was changed then branch to automatic section
WINDOW CLOSE 4
MENU 4,0,1
temp$ = INPUT$(LOC(1,1)) "Clear receive buffer before continuing
GOTO PollAll

manual:
IF key$ <> "" THEN GOTO men0:
   REM --- Toggle direction of train
dir = direction(ActiveLoco)
IF dir = 0 THEN GOTO tog1
   Direction(ActiveLoco) = 0
   send$ = "; "
   GOTO togend
tog1:
   Direction(ActiveLoco) = 1
   send$ = "; "
togend:
   GOSUB Transmit
   GOTO ManualLoop

men0:
IF key$ <> "+" AND key$ <> "-" THEN GOTO men1
   REM --- Increment ActiveLoco's speed but not past 14 or
   REM --- Decrement speed but not below 0.
   REM --- Increment and Decrement mean increase or decrease speed
   REM --- independent of Direction
   sp = Speed(ActiveLoco)
   IF sp = 14 AND key$ = "+" THEN GOTO ManualLoop
   IF sp = 0 AND key$ = "-" THEN GOTO ManualLoop
   IF key$ = "+" THEN sp = sp + 1 ELSE sp = sp - 1
```
speed(ActiveLoco) = sp
send$ = MID$(butval$, (33-sp), 1) + " 
GOSUB Transmit
GOTO ManualLoop

man1:
  IF key$ <> "<" THEN GOTO man2
  REM --- if new loco is specified then we want to change ActiveLoco
  tnum = 1: tkey$ = ""
  GOSUB GetButton
  IF key$ <> "1" THEN GOTO loc1
  REM --- Might have two digit train number
  tkey$ = "1 
  GOSUB GetButton
  IF key$ >= "0" AND key$ <= "6" THEN tnum = 10 ELSE tnum = 5
loc1:
  IF tnum = 5 THEN cval = 1
  IF tnum = 10 THEN cval = 10 + VAL(key$)
  IF tnum = 1 THEN cval = VAL(key$)
  IF cval <> ActiveLoco THEN ActiveLoco = cval
  REM --- This next statement is a kludge. It handles the case where the user
  REM --- input 'LOCO 1 LOCO ...' If this statement was not here then the program
  REM --- would not remember the correct ActiveLoco
  IF key$ = "<" THEN send$ = "< 1 ": GOSUB transmit: GOTO man1
  send$ = "< + tkey$ + key$ + " 
  GOSUB Transmit
  GOTO ManualLoop

man2:
  IF key$ <> "; AND key$ <> "?" THEN GOTO man3
  REM --- reset Direction for ActiveLoco
  IF key$ = "; THEN Direction(ActiveLoco) = 0 ELSE Direction(ActiveLoco) = 1
  send$ = key$ + " 
  GOSUB Transmit
  GOTO ManualLoop

man3:
  REM --- Character must be either a speed control character, an arrow,
  REM --- a digit, or INERTIA character
  send$ = key$ + " 
  GOSUB Transmit
  GOTO ManualLoop

REM --- Wait for two bytes to appear on RS232
Poll:
PollLoop1:
  IF LOC(1) = 0 THEN GOTO PollLoop1
  r1$ = INPUT$(1,1)
PollLoop2:
  IF LOC(1) = 0 THEN GOTO PollLoop2
r29 - INPUT$(1,1)
RETURN

nALL:
LOCATE 1,16:PRINT "Polling all computers"
PRINT #1,poll$;
GOSUB Poll

'Get two bytes of return information
'NOTE: Bit 0 should be set to 1 by my project computer
'as a signal that this byte and the next one are the actual
'return information. Theoretically, all zeroes could be returned
'if no computer was ready to send information. It also
'serves as a protection device. If the Mac does receive a
'non-zero byte but the first bit is not a 1 then my project
'computer must have missed the poll request so the Mac polls
'again.

n1 = ASC(r1$)
IF (n1 AND 1) = 0 THEN GOTO PollALL
n2 = ASC(r2$)
LOCATE 3,3:PRINT "Received: ";
TEXTFACE 1:LOCATE 3,10:PRINT HEX$(n2)" "HEX$(n1): TEXTFACE 0

IF Mode = manual THEN GOTO ManualMode
REM ---- now set determine which computers need servicing
Offset = 0
CurrMask = 64
FOR i = 1 TO 7
    IF (n2 AND CurrMask) > 0 THEN request(i) = 1 ELSE request(i) = 0
    CurrMask = CurrMask / 2
NEXT i

CurrMask = 128
FOR i = 8 TO 9
    IF (n1 AND CurrMask) > 0 THEN request(i) = 1 ELSE request(i) = 0
    CurrMask = CurrMask / 2
NEXT i

CurrCompNum = 1
REM ---- Cycle through array of requests until there are no more
HandleRequests:
    IF CurrCompNum > 9 THEN GOTO PollALL
    IF request(CurrCompNum) = 0 THEN CurrCompNum = CurrCompNum + 1: GOTO HandleRequests
REM ---- Now poll individual computer
LOCATE 1,16:PRINT "Polling computer ",CurrCompNum;
send$ = STR$(CurrCompNum)+" 0 > ";
temp$ = INPUT$(LOC(1),1) 'Clear out buffer if any junk in it
GOSUB Transmit
GOSUB Poll
n1 = ASC(r1$)
n2 = ASC(r2$)
LOCATE 3,3:PRINT "Received: ";
TEXTFACE 1: LOCATE 3,10: PRINT HEX$(n2) HEX$(n1): TEXTFACE 0

CurrCommand = n2  'Do high byte first
IF CurrCommand = 0 THEN GOTO NextByte
"%3 = table$(CurrCompNum-1,CurrCommand)
LOCATE 5,3: PRINT "Executing command ";
TEXTFACE 1: PRINT HEX$(CurrCommand);","%3,SPACE$(20); TEXTFACE 0
CurrLine = 7
LOCATE CurrLine,6
GOSUB HandleCommand

NextByte:
CurrCommand = n1  'Do low byte
IF CurrCommand = 0 THEN GOTO HandleRequests
"%3 = table$(CurrCompNum-1,CurrCommand)
LOCATE 5,3: PRINT "Executing command ";
TEXTFACE 1: PRINT HEX$(CurrCommand);","%3,SPACE$(20); TEXTFACE 0
CurrLine = 7
LOCATE CurrLine,6
GOSUB HandleCommand
CurrCompNum = CurrCompNum + 1
GOTO HandleRequests

HandleCommand:
REM --- The following does not necessarily have to be added but I am assuming REM --- that most project computers don't know that other projects are also REM --- controlling the trains. So, project #1 might think that the Hornby REM --- is currently controlling its train but project #5 might have changed the REM --- current train while it had control. Therefore, if the currently ActiveLoco REM --- is differant from the train number in CompCurrLoco then the new loco is REM --- selected before the TX$ is sent.
REM --- First it checks whether to do anything at all
IF LEN(TX$): 0 THEN PRINT "Command string empty."; RETURN
IF LEFT$(TX$,1) = "" THEN PRINT "Command string empty."; RETURN
IF ActiveLoco = CompCurrLoco(CurrCompNum) THEN GOTO ContinueHandling
cci = CompCurrLoco(CurrCompNum)
ActiveLoco = cci
IF cci > 9 THEN extra$ = "1" : ofs = 10 ELSE extra$ = "" : ofs = 0
digit$ = MKI$(cci - ofs)
PRINT "Sending Initial train selection: LOCO ";extra$;",";digit$;" ->";
CurrLine = CurrLine + 1
LOCATE CurrLine,6
send$ = "<" + extra$ + digit$ + "">"
GOSUB Transmit

REM --- NOTE: This section looks very similar to the manual section.

ContinueHandling:
position = 1
c$ = MID$(TX$,position,1)
WHILE position <= 20 AND c$ <> " " AND c$ <> ""
IF c$ <> ">>" THEN GOTO cont0
   REM --- Toggle direction of train
   dir = direction(ActiveLoco)
   PRINT "Setting direction of train ";ActiveLoco," to ";
   IF dir = 0 THEN GOTO toggle1
   Direction(ActiveLoco) = 0
   send$ = "; "
   PRINT "REVERSE";
   GOTO toggleend
toggle1:
   Direction(ActiveLoco) = 1
   send$ = "? "
   PRINT "FORWARD";
toggleend:
   CurrLine = CurrLine + 1
   LOCATE CurrLine,5
   GOSUB Transmit
   GOTO ContEnd
cont0:
IF c$ <> "++" AND c$ <> "--" THEN GOTO cont1
   REM --- Increment ActiveLoco's speed but not past 14 or
   REM --- Decrement speed but not below 0.
   REM --- Increment and Decrement mean increase or decrease speed
   REM --- independent of Direction
   sp = Speed(ActiveLoco)
   IF sp = 14 AND c$ = "+" THEN GOTO ContEnd
   IF sp = 0 AND c$ = "-" THEN GOTO ContEnd
   IF c$ = "+" THEN sp = sp + 1 ELSE sp = sp - 1
   Speed(ActiveLoco) = sp
   IF c$ = "++" THEN PRINT "Incrementing speed of LOCO ";ActiveLoco," to ";sp;SPACE$(20);
   IF c$ = "--" THEN PRINT "Decrementing speed of LOCO ";ActiveLoco," to ";sp;SPACE$(20);
   CurrLine = CurrLine + 1
   LOCATE CurrLine,6
   send$ = MID$(butval$,(33-sp),1) + " "
   GOSUB Transmit
   GOTO ContEnd
cont1:
IF c$ <> "<<" THEN GOTO cont2
   REM --- if new loco is specified then we want to change ActiveLoco
   mult = 1
   position = position + 1
   IF position > 20 THEN LOCATE 12,1 : PRINT "Malformed string!!"; GOTO ContEnd
   c$ = MID$(TX$ ,position ,1)
IF c$ <> "1" THEN GOTO loc01
   REM --- Might have two digit train number
   IF position+1 > 20 THEN GOTO loc01
   tempc$ = MID$(TX$ ,position+1 ,1)
IF tempc$ >= "0" AND tempc$ <= "6" THEN mult = 10

local:

. PRINT "LOCO ;c$; ";
  cval = VAL(c$)
  IF mult = 10 THEN cval = (cval*10) + VAL(tempc$)
  IF cval < ActiveLoco THEN ActiveLoco = cval : CompCurrLoco(CurrCompNum) = cval
  REM --- We have grabbed loco character and next one so only send those two
  REM --- If we did read tempc$, then just leave it for next iteration
  REM --- since we didn't actually grab it, we just looked at it
  send$ = "< " + c$ + " "
  GOSUB Transmit
  GOTO ContEnd
cont2:

IF c$ <> "," AND c$ <> "?" THEN GOTO cont3
REM --- reset Direction for ActiveLoco
IF c$ = "," THEN Direction(ActiveLoco) = 0 ELSE Direction(ActiveLoco) = 1
IF c$ = "?" THEN PRINT "REVERSE"; ELSE PRINT "FORWARD";
  CurrLine = CurrLine + 1
  LOCATE Currline,6
  send$ = c$ + " "
  GOSUB Transmit
  GOTO ContEnd
cont3:

REM --- Character must be either a speed control character, an arrow,
REM --- a digit, or INERTIA character
IF c$ <> ";" AND c$ <> ":" THEN cont31
  IF c$ = ";" THEN PRINT ";", ELSE PRINT ";->";
  CurrLine = CurrLine + 1
  LOCATE Currline,6
  GOTO cont3end
cont31:

IF c$ < "0" OR c$ > "9" THEN GOTO cont32
  PRINT c$ " ";
  GOTO cont3end
cont32:

IF c$ <> "=" THEN GOTO cont33
  PRINT "Inertia ";
  GOTO cont3end
cont33:

  PRINT "Set LOCO ";ActiveLoco:" to speed ";33-INST(butval$,c$);
  CurrLine = CurrLine + 1
  LOCATE Currline,6
cont3end:

  send$ = c$ + " ">
  GOSUB Transmit
ContEnd:

  position = position + 1
IF position = 20 THEN ON = MID$(TEXT,position,1)

RETURN

_REM --- Transmit send$ over RS232
Transmit:
  i = LEN(send$)
  FOR i = 1 TO 1
    PRINT #1, MID$(send$, i, 1);
  NEXT i
  RETURN

_REM --- Initialize the menus
init:
  TEXTFONT 2
  TEXTSIZE 12
  TEXTMODE 0

REM --- Setup menus and variables
MENU 1,0,1,"Quit"
MENU 1,1,1,"Yes"

MENU 2,0,1,"Edit"
MENU 2,1,1,"Table"

MENU 3,0,1,"File"
MENU 3,1,1,"Load"
MENU 3,2,1,"Save"

MENU 4,0,1,"Mode"
MENU 4,1,1,"Manual"
MENU 4,2,1,"Automatic"

MENU 5,0,0,"
ON MENU GOSUB HandleMenu
MENU ON

FOR i = 1 TO 9
  request(i) = 0
  CompCurrLoco(i) =
  Direction(i) = 1
  Speed(i) = 0
NEXT i
FOR i = 10 TO 16
  Direction(i) = 1
  Speed(i) = 0
NEXT i
REM --- These parameters are burned into my EPROM
OPEN "COM1:300,n,8,1" AS #1 LEN=2000
RETURN

HandleMenu:
MENU OFF 'Disable all menus while handling menus
m = MENU(0)
menuItem = MENU(1)
IF m = 1 THEN END 'got QUIT command
IF m = 3 THEN GOTO DoFile 'got FILE command
IF m = 2 THEN menuItem 'if not MODE selection then go to EDIT section
IF menuItem = 1 THEN Mode = manual ELSE Mode = automatic
MENU 4,0,1
GOTO MenuReturn

menu1:
WINDOW 2,,(10,20)-(750,450),2
tempMode = Mode : Mode = automatic 'Temporarily set Mode so that SetupEdit Screen
GOSUB SetupEditScreen 'will draw all the buttons
Mode = tempMode
GOSUB DoEdit
WINDOW CLOSE 2
MENU 1,0,1

MenuReturn:
MENU ON
RETURN

REM --- First determine whether to LOAD or SAVE
DoFile:
IF menuItem = 2 THEN GOTO SaveFile

REM --- Uses BASIC's "files" command to bring up the file menu
REM --- Simply saves 256 strings of length MAXLEN. User is prompted
REM --- for which table to save
LoadFile:
filename$ = FILES$(1)
IF filename$ = "" THEN MENU 2,0,1: GOTO MenuReturn
OPEN filename$ AS #2 LEN = MAXLEN
FIELD #2, MAXLEN AS buffer$
GOSUB GetCompNum
FOR i = 0 TO 255
  GET #2,i+1
  table$(bikid,i) = buffer$
NEXT i
CLOSE #2
MENU 2,0,1
REM --- First the "files" box appears. Select a file and then hit "OPEN"
REM --- If you want to create a new file then hit "CANCEL"
REM --- A new "files" box appears where you can type in the name
REM --- If you don't want to create a file either then hit "CANCEL"

SaveFile:
filename$ = FILES$(1)
IF filename$ <> "" THEN GOTO DoSave
filename$ = FILES$(0)
IF filename$ = "" THEN MENU 2,0,1: GOTO MenuReturn

DoSave:
GOSUB GetCompNum
OPEN filename$ AS #2 LEN = MAXLEN
FIELD #2, MAXLEN AS buffer$
FOR i = 0 TO 255
  LSET buffer$ = table$(bliid,i)
  PUT #2,i+1
NEXT i
CLOSE #2
MENU 2,0,1;
GOTO MenuReturn

REM --- Draws layout of Hornby controller keypad and speed slide
SetupEditScreen:
RESTORE
LOCATE 7,33: PRINT "SPEED";
FOR i=1 TO 12
  READ x,y,nam$
  BUTTON i,i,nam$, (x,y)-(x+30,y+15)
NEXT i
FOR i=13 TO 16
  READ x,y,nam$
  BUTTON i,i,nam$, (x,y)-(x+60,y+15)
NEXT i
FOR i=19 TO 33
  READ x,y,nam$
  BUTTON i,1,nam$, (x,y)-(x+20,y+12)
NEXT i
FOR i=34 TO 38
  READ x,y,nam$
  BUTTON i,1,nam$, (x,y)-(x+70,y+25)
NEXT i
READ x,y,nam$
BUTTON 39,1,nam$, (x,y)-(x+60,y+15)
RETURN
REMich - Gets the number of the array entry to be edited

GetCommand:

```
WINDOW 3.,(150,50)-(350,100),2
CommLoop:
    LOCATE 1,2
    PRINT "Enter 2 digit command in HEX";
    LOCATE 2,10 : INPUT comm$
    comm$ = UCASE$(comm$)
    IF LEN(comm$) <> 2 THEN CommLoop
    id$ = LEFT$(comm$,1)
    rd$ = RIGHT$(comm$,1)
    ln = ASC(id$) - 46
    IF ln > 9 AND ln < 16 THEN GOTO CommLoop
    IF ln > 16 THEN ln = ln - 7
    IF ln < 0 OR ln > 15 THEN GOTO CommLoop
    rn = ASC(rd$) - 46
    IF rn > 9 AND rn < 16 THEN GOTO CommLoop
    IF rn > 16 THEN rn = rn - 7
    IF rn < 0 OR rn > 15 THEN GOTO CommLoop
    CommNum = 16*ln + rn
    IF CommNum = 0 THEN GOTO CommLoop
    WINDOW CLOSE 3
    RETURN
```

REM ---- Gets the number of the table for either "EDIT", "LOAD", or "SAVE"

GetCompNum:

```
WINDOW 3.,(150,50)-(370,100),2
CompLoop:
    LOCATE 1,2
    PRINT "Enter computer's BLKID (1-9)";
    LOCATE 2,10 : INPUT blikid
    IF blikid < 1 OR blikid > 9 THEN CompLoop
    blikid = blikid - 1
    WINDOW CLOSE 3
    RETURN
```

REM ---- Processes all the buttons
REM ---- Command strings can be MAXLEN characters long
REM ---- When finished entering "keystroke" use "OK" to save the newly
REM ---- entered string in the table or use "CANCEL" to enable the
REM ---- "DONE" and "COMMAND" buttons. (NOTE: while entering a string,
REM ---- the "DONE" and "COMMAND" buttons are disabled so that you can't
REM ---- quit editing or select a new command byte until you finished
REM ---- with the present command.) Use the "DELETE" button while you
REM ---- are entering "keys" in the string to erase the last character
REM ---- entered. Use the "COMMAND" button to select a new table entry.
REM ---- Use "DONE" to quit editing.
DoEdit:
CommNum = 1: comm$ = "01" 'default to second element of array
'00 is not an allowed command

\$SUB SetCommNum

TEXTSIZE 18
LOCATE 1,1: PRINT "Editing table for ";
LOCATE 2,1: PRINT "Computer ",blkid+1;
TEXTSIZE 12
LOCATE 14,1: PRINT "Current string for command ";
TEXTFACE i: LOCATE 14,21: PRINT comm$": "; TEXTFACE 0
LOCATE 14,28: PRINT table$(blkid,CommNum);
LOCATE 16,1: PRINT "New string ";
LOCATE 16,12
newcomm$ = ""
numkeys = 0

REM ---

EditLoop:
dialogid=DIALOG(0)
IF dialogid> THEN EditLoop
buttonid=DIALOG(i)
IF buttonid > 33 AND buttonid < 39 THEN ProcessCommand
BUTTON 37,0 'Turn off "DONE" and "COMMAND" buttons
BUTTON 36,0
IF numkeys >= MAX THEN GOTO EditLoop
numkeys = numkeys + 1
IF buttonid = 39 THEN key$ = "" : GOTO ed1
key$ = MID$(butval$,buttonid,1) 'Find corresponding byte to send Hornby

ed1:
PRINT key$;
newcomm$ = newcomm$ + key$
GOTO EditLoop

REM --- button hit was not a "key" so it must be an editing command

ProcessCommand:
IF buttonid > 34 THEN label1 'Branch if not "OK"
    table$(blkid,CommNum) = newcomm$
- LOCATE 14,28: PRINT table$(blkid,CommNum);
LOCATE 16,12: PRINT table$(blkid,CommNum);
- BUTTON 36,1
  BUTTON 37,1 :
  GOTO EditLoop
- label1:
  IF buttonid > 35 THEN label2 'Branch if not "CANCEL"
    BUTTON 36,1
    BUTTON 37,1
    GOTO EditLoop
- label2:
IF buttonID > 36 THEN label3 'Branch if not "DONE"
RETURN
label3: 
IF buttonID > 37 THEN label4 'Branch if not "COMMAND"
    Gosub GetCommand
    Locate 14,1: Print "Current string for command:"
    TextFace 1: Locate 14,21: Print comm$: ": TextFace 0
    Locate 14,28: Print table$(blkid,CommNum); Print space$(25);
    Locate 16,1: Print "New string:"
    Locate 16,12: Print space$(25);
    Locate 16,12
    newcomm$ = "": numkeys = 0
    Goto EditLoop
label4: 
IF numkeys = 0 THEN Goto EditLoop 'Nothing to "DELETE"
    numkeys = numkeys - 1
    Locate 16,12
    IF numkeys > 0 THEN label5
    newcomm$ = "": Print space$(25);
    Locate 16,12
    Goto EditLoop
label5: 
    newcomm$ = left$(newcomm$,numkeys)
    Print newcomm$;
    Goto EditLoop

REM --- x,y coordinate of button, title of button
    Data 10,110,7
    Data 10,130,4
    Data 10,150,1
    Data 10,170,0
    Data 50,110,8
    Data 50,130,5
    Data 50,150,2
    Data 50,170,<- 
    Data 90,110,9
    Data 90,130,6
    Data 90,150,3
    Data 90,170,-> 
    Data 130,110,Inertia
    Data 130,130,Loco
    Data 130,150,Inc Spd
    Data 130,170,Dec Spd
    Data 210,150,Reverse
    Data 210,170,Forward
    Data 300,10,14
DATA 300,22,13
DATA 300,34,12
DATA 300,46,11
DATA 300,58,10
DATA 300,70,9
DATA 300,82,8
DATA 300,94,7
DATA 300,106,6
DATA 300,118,5
DATA 300,130,4
DATA 300,142,3
DATA 300,154,2
DATA 300,166,1
DATA 300,178,0
DATA 370,10,0k
DATA 370,40,Cancel
DATA 370,70,Done
DATA 370,100,Command
DATA 370,130,Delete
DATA 210,130,Toggle

REM --- End of program