Department of Electrical Engineering
Spring Independent Work

Complete Automation and Monitoring for a Model Train Layout

by Arthur L. Coburn IV
Friday, May 15, 1987

Professor Michael Littman, advisor

pledge my honor that this work is my own in accordance with University regulations

Arthur L Coburn IV

DEDICATION

Without the support and love of my parents, this work would not have been possible.

I would also like to express my thanks to Mike Littman, who has been very helpful.

CONTENTS

ABSTRACT

The purpose of this project was to complete the automated interactive loop, so that a model train layout can run completely unattended. A Master Computer Program was written for an IBM-PC AT which not only carries out the interactive control process automatically, but also monitors the status of each block in the layout, reporting the identity of the trains currently inside each block and the state of the signal lights on either end of the block. This status reporting is done with the aid of an updated version of the ISERV routine. Finally, the Master Computer Program allows the user to enter any commands that could previously be entered from the Hornby keypad, but knowledge of the Hornby syntax is unnecessary. The final program is easy to use and robust, and was designed with the intent that it would be easy to update and improve in the future. This report is also written so that it can be used as a reference by students in the furute who wish to build upon this work.

INTRODUCTION

The model train layout is divided into sections or *blocks*, each with its own *block computer* which controls the project for that section (these computers are sometimes called *project* computers). An interrupt driven routine (called ISERV) is included in each block computer; it prevents train collisions, reads the bar codes on the trains, sets the signal lights correctly, and sends and receives data from the *master computer* (also called the *master controller*). This routine is invisible to the project designer, except through a small number of memory locations called *mailboxes*. A block computer cannot change the speed, direction or momentum of a train directly; it must ask the master controller to carry out the task. The master controller used to be a person who would visually monitor the layout. When a block computer requested attention, it would display a *request code* on its LEDs; the controller would look up the code in a table and execute the appropriate tasks. This process is called **interactive control**. This is now carried out automatically; a block computer signals the master computer that it needs attention and then sending a *command code* (or *request code*) to the master computer. The master computer then uses the code as an index to a table of commands; the commands tell the master computer which tasks to perform.

One central element in the layout is the *Hornby Zero-1* controller. It supplies power to the train engines and switches, and also sends data along the train tracks, telling the engines what speeds and directions they should go. This is possible because the track waveform is an AC square wave, alternating between power frames and data frames (see Littman, Wotiz, and Blaha, The Hornby Railways System of Microprocessor Control of Model Trains). The Hornby unit has a keypad for selecting engine numbers, a slider control to set speeds, and two push buttons to select the engines' directions. The Hornby unit can also send data to accessories, such as track switches. This ability has been adapted to allow the Hornby to send data to the various block computers around the layout. This is important for interactive control. When done manually, interactive control takes this form:

Manual Interactive Control

1) The block computer flashes a request code on its display.
2) The person monitoring the layout sees the request code and looks up in a table the appropriate commands to execute.
3) The person then sends a predefined acknowledgement code back to the block computer to verify the completion of the task.

Phil Dworsky and Mike Arena designed and built interfaces that allowed PCs to send and receive data from the layout, via an RS232 link. This was the first step toward total automation.

Dworsky, in 1982, designed a circuit that decoded the characters sent to it and simulated pushing corresponding keys on the Hornby keypad (see Phil Dworsky, A 300 Baud RS-232 Interface to the Hornby Zero-1 Model Train Controller, Fall, 1982). He discovered an important limitation of the Hornby unit--it cannot respond to more than ten keypresses a second. THIS IS SLOW! Thus he had to settle for a 300 baud communication link, and even with this, two dummy characters must be sent between each real command character in order to allow the Hornby to respond to each real character. This is discussed in greater detail later in the limitations section.

Arena then designed and built a circuit that collects return information from all the block computers and sends this via a 300 baud RS232 cable back to a PC (see Michael Arena, Interactive Control-Completing the Loop for Total Automation, May, 1985). The return information from a block consists of two 8-bit words of data. Since Dworsky's circuit originally used 7-bit words, his circuit had to be altered to accomodate the 8-bits to standardize the communication system. There were problems with noise in the Arena circuit that I built--these are discussed in the next section, on hardware.

Arena also wrote a program which would carry out interactive control automatically. However, there were limitations to his program: commands couldn't be typed from the keyboard at the same time the automatic interactive control routine was running; commands coudn't be edited after they were typed in originally; the user had to know the Hornby instruction

syntax. More importantly, though, Mike Littman and I wanted to add the capability of reading mailboxes in the block computers remotely and without user knowledge. This would then allow the program running on the PC to automatically generate status reports for each block in the layout.

The rest of this report desribes changes made to the hardware (Dworsky's and Arena's), changes made to the ISERV routine, the Master Computer Program, and the limitations of the system.

HARDWARE

The Arena Circuit Problem : The first step of my independent work was to build a copy of Mike Arena's circuit. There was one major difficulty in doing this. In the RIB (return information bus) data receiving circuit, a JK-flip-flop is used as a divide by two counter. This circuit is crucial in the timing of the data reception. It is also highly prone to noise. Arena's solution was to put a capacitor from the output of this flip-flop to ground, to try to despike any noise emanating from that point. This did not produce a completely reliable circuit. An occasional timing error would cause one bit to be read in as two, or two as one. For automatic interactive control, errors are intolerable. If a command code is transmitted incorrectly, then an incorrect acknowledgement will be returned, and the block computer will hang forever, waiting for an acknowledgement that will never come. Also, if the second byte of data is interpreted as a train number, speed, direction, etc. and an error occurs in transmission, the wrong thing will happen, and the data may even be illegal (i.e. out of range), causing the master computer program to suffer a major trauma.

The Solution : First, make sure that the circuit that converts the track waveform to computer levels works well and has a fairly large voltage swing (>3.5v). Then adjust the capacitor on the output of the JK-flip-flop so that the number of errors is minimized. Then connect a large (10µf or so) capacitor from the power pin of the flip-flop chip to the ground pin of the same chip. This despiking of the power supply seemed to

be the real solution to the problem. A program was run to test the new Arena circuit--in ten thousand sets of two bytes of return data, there was not a single error.

It was also discovered that the polarity of the track power and common wires can make a difference in the reliability of the data transmission. For some block computers that were tested, reversing the track wires where they connect to the Hornby made the difference between a very reliable and a very unreliable communication link.

Other notes: The Dworsky circuit must be altered so it will accept 8-bit words; the other parameters are all correct (1 stop bit, no parity, 300 baud).

Also, a 4N33 must be put in the proper slot in the *signal sensor board*; there is one located near each block underneath the layout. The 4N33 has been left out of many of these circuits (for reasons of cost, I imagine).

Also, there is no RIB cord connected to many of the signal sensor boards. They should be connected with the blue wire coming off pin 17 of the 44 pin edge connector and the white wire coming off pin 16. The RIB cables from all the projects in the layout should be connected together--all the blue wires should be connected together, and all the white wires connected together. This bus works like an interrupt line with a pull up resistor on the Arena circuit board.

CHANGES MADE TO ISERV

The next step was to alter the ISERV routine, which runs on every block computer, so that it could send status information back to the controlling computer. This was not that difficult; ISERV already had the capability of sending two bytes of return data back to the controller. A new protocol had to be set up which would allow the master computer to request that the status information of a certain block be sent; then the ISERV routine had to be changed so that only this block would return its status.

The OLD protocol was (as entered from the Hornby keypad):

0 x ->	meant	all block computers, set your bit if you require attention. (x = any number)
# 0 ->	meant	block number #, send your two bytes of return information.

The NEW protocol is (as entered from the Hornby keypad):

0 x <-	means	all block computers, set your bit if you require attention. (x = any number)
# 0 ->	means	block number #, send your two bytes of return information.
0 # ->	means	block number #, send your two bytes of *status information.*

The code 0 0 > is not used because the Hornby will not send this code.

The next matter was to decide which data in the block computers was most important. Professor Littman and I decided that, for the moment, the really important data is the state of the signal lights on each end of the block, and the numbers of the trains on the north and south tracks. The mailbox CONTRL in ISERV contains the state of each of the lights, and BCRASM happens to hold both the north and south train numbers (the north train number is the high four bits). These two mailboxes are therefore sent. It would be easy in the future to modify the ISERV routine to send back another set of mailboxes. The routine could be set up so that the first set of mailboxes are returned during one data cycle and the other set on the next data cycle, or so that a special global code (one that effects all the blocks simultaneously) would tell all the blocks which set of data to return when they are next polled.

The last matter was to know when to send this data. When the block computer receives data from the outside, there is a short section of code which checks to see if it is a reserved code (i.e. one of the above protocols). When the computer receives the 0 # > code, it sets a flag (STFLG); this signals the block computer to send the status data the next time the data frame comes around. The routine simply replaces the user defined WDATAL and WDATAH with CONTRL and BCRASM (respectively) and then sends these two bytes as usual (WDATAL is always sent back to the master computer first). Immediately afterwards, WDATAL and WDATAH are restored to their original values. The user never even knows.

THE MASTER COMPUTER PROGRAM

Arena had already written a basic automatic interactive program, but a new one needed to be written. The reasons are these:

1) The old program was not capable of sending or receiving status information.
2) The old program ran on a Macintosh; since then, PC ATs have been placed all around the lab.
3) The old program could not execute interactive control *and* allow the user to enter commands at the keyboard simultaneously.
4) The old program was not especially user-friendly.
5) The old program did not have well developed editing capabilities for the command files.

Thus it was decided that a new program would be written from scratch. Keeping in mind that much of the independent work done in the past built upon previous independent work, this program has been written so it is readable and expandable. Hopefully future students will be able to use it as a point of departure for future developments.

Turbo Pascal was the language of choice because it was available and seemed to be able to do everything necessary. It was important that Turbo had commands to do bit manipulations, and that it could deal well with serial communications.

Program Organization

The program is divided into two main parts; the first actually carries out the monitoring of the layout, the interactive control, and the manual control; the second is an editor, with accompanying DOS functions, to manipulate files of commands. These are the commands which are looked up and executed in the interactive control routine. The state of the program is stored in a variable called *PROGRAM_STATE*. It only has three possible values: *run, edit,* and *quit.*

The execution routine (*run_program*) follows this general format:

a) Initializes variables before execution begins.

1) Check keyboard for input; if there is some, see if a complete command has been entered. If so, execute it--if not, save the characters for next time. The routine that does this is ***manualexec.***
2) Get the status of the next active block and display. The active blocks are stored in *pollcomps.*
3) Check keyboard for input again. (***manualexec***).
4) See if there are any block computers that need attention (check *requestattncomps*). If there are, then

    i) Poll the first computer in the set (***pollblock***), get its return information (WDATAH and WDATAL) and execute the corresponding tasks (***executecommand***). ***executecommand*** will send an acknowledgement.

    If there are no computers on record as seeking attention, then

    ii) Send out a global poll request (*global_poll*). Each computer will set the bit corresponding to itself if it needs attention.
5) Jump back to part 1) unless *PROGRAM_STATE* no longer equals *run.*

The editing section executes the following functions from a menu format:

1) Save a command file to a disk.

2) Retrieve a command file from a disk.
3) Edit a command file that has been retrieved.
4) Display the entire contents of a command file.
5) Execute the train layout monitoring and interactive control routines (jump to ***run_program***).
6) Quit the entire program--return to the operating system.

The Logic of the Program

The program was developed a section at a time. The manual control section was the first to be developed. In order to allow the program to constantly monitor the layout, a routine was needed that would read in as many characters as had been entered from the keyboard and then return to the caller. This routine (***manualexec*** and its assosciated routines) executes any complete commands; if a command is only partially entered, then the left-over characters are stored in *KEYBOARD_BUFFER* until more characters are typed in. When new characters are entered, they are added to the *KEYBOARD_BUFFER*; if this completes a command, then it is executed and the characters are removed from the *KEYBOARD_BUFFER*; if a command is still not complete, the characters are left there.

This allows the program to constantly monitor the layout, because the user input routine does not wait for a command to be completed before it returns. If this were not true, then the program would just halt until the user finished typing in a command; all events on the layout would be ignored. This is a **re-entrant** input routine; it can be exited and re-entered and will be in the same state in which it was left.

If an error occurs when the user is entering a command at the keyboard, then the *ERROR* flag is set, an error message is displayed, and the *KEYBOARD_BUFFER* is cleared to prevent further errors. The next character entered clears the error message and resets the *ERROR* flag. This character is then ignored (this could be easily changed, depending on the user's preference).

A routine was developed next which would poll the block computers

and get their status information (called ***get_status_of_block***). A nice display was developed so that the lights and train numbers for all the blocks could be seen simultaneously. This was developed independently of the manual control routine, and then the two were combined. It is necessary to clear the RS232 input buffer before reading in data; this is accomplished by reading junk from the port immediately after writing to it. Otherwise the ***datathere*** routine will report data arriving in the input buffer when there really is none. One other important aspect of ***get_status_of_block*** and the other routines which poll the block computers is that they will not wait forever for return information; they will time out instead. This prevents the program from hanging.

Once the manual input routine was developed, it was connected to simple I/O routines so that it could be used to control the trains on the layout. These routines were further developed later, so that they update the current status of each train in the arrays *currentspeed, currentdir,* and *currentmomentum.* They also update the display for this information. It is also necessary to keep track of the current context for each block computer, i.e. the last set of trains controlled by the block, the last speed set by the block, and the last direction set by the block. The Hornby unit keeps track of each train's momentum individually so that the master computer does not have to keep track of the last momentum set by each computer. This context data is stored in the arrays *currenttrainset, computerspeed,* and *computerdir.*

Each block computer should not have to worry about the others around the layout. This is why the context of each block must be remembered. This allows each block computer to act as if it had sole control over the Hornby unit. If block two set engine 5 to speed 3 at one point, then if the next command is to set the current engine in reverse, then this engine will be engine number 5, even if another block sent a message to another engine in between. The program would slow down too much if, each time a block wanted to execute a command, the entire context for that block were transmitted to the Hornby. Therefore, the variable *CURR_CONTEXT* is used to keep track of which block last issued a command (the manual control routine is treated as block 0 for this purpose). If one block executes two commands in a row, with no other intervening commands, then the context for that

block does not need to be restored, so it isn't. If, however, a second block has the Hornby execute a command and then the first computer wants to send a command, then the context for the first block must be restored.

Scenario :

Computer#	Command executed	Characters sent to Hornby
2	Engine 3, speed4	< 3 > G
2	Reverse	
2	Engine 3, speed 4	< 3 > G
5	Engine 6, speed 12	< 6 > K
2	Reverse	< 3 > G ;

If, in the second series of commands, the context were not restored, then engine 6, not engine 3, would be set in reverse.

The automatic interactive control routines were then written. These commands are stored as a string of characters. The strings consist of command character/argument pairs. The command characters are the first letters in the words they correspond to; thus, in a command string, **E7** refers to engine 7; **S12** means speed 12; **DF** means direction forward; **M3** means a momentum of three; **C2D9** means send the data 9 to block computer 2. Each block computer has an array of commands which it can define; it refers to each command by its index in the array. When block 8 sends back WDATAH equal to $4C, this is used to look up a corresponding command string, which is then executed. WDATAL is used to transmit data which can be inserted into the command strings in place of fixed arguments. Since none of the arguments for any of the command characters can exceed 16, WDATAL can be treated as two nibbles, each of which can specify an argument value. Also, since the direction of a train has only two values and the momentum only four, the low nibble can be treated as two 2-bit words, if necessary. This allows the user to have up to three variable arguments in a single command string. Since, with only four bits, a 16 cannot be sent, train 16 is specified by sending a 0 (zero); a momentum of 4 can be specified using a two bit field by sending a zero also. The **insert_data_into_comstring** routine has been altered so that, for both

of these data types, the zero value is mapped into the previously unachievable value.

The editing procedure (***editcomstrings***) is designed to prompt the user with all the proper syntaxes and helpful reminders. The command strings are stored on disk and during the program's execution as they are typed in by the editor; this is so that the user can come back later and edit them. This means they are parsed once when they are first typed in (to insure correct syntax) and a second time when they are executed. The parsing routine does not take long, so this does not hurt the execution speed. Originally they were parsed into another string that could be transmitted directly to the Hornby; however, it was not easy to keep track of the data as it was sent in this form, so another routine, ***update_and_send***, reparses the already parsed string and calls the standard I/O routines. This is slightly round-about; this could be corrected in the future by making the ***updata_and_send*** routine parse the original, unparsed command string. If this is done, then the ***insert_data_into_comstring*** routine must be changed so that it inserts data into a string of the completely unparsed form.

Executecommand is the routine which is called when WDATAH and WDATAL are received after an interactive control request. If an error occurs here and is unrecoverable (mainly out-of-range errors) the the program prints a nice error message and quits. This is annoying, but there is no way to recover from such errors.

DOS routines were written to save a file of commands for any one of the blocks to disk and retrieve it later; however, there is no provision to allow the user to change directory from within the program; this could be added later.

See the Future Development section to see a complete list of suggested future improvements.

Future Development

I have tried to write this program so that it can be easily modified in the future. To do so, I have divided it up into separate logical blocks as much as possible. The Hornby communication protocol can be easily changed by modifying four short procedures (***sendtrainset,*** etc.). The routines that update the displays are grouped together and simple to modify. Here are a few suggested modifications:

1) Develop a nice display which would indicate the current context for each block computer. This could go on display page #2 (#0 is the lowest), which is currently not really used.
2) Add a directory printing routine to editing menu.
3) Add a function to the editing menu that allows the user to change the current directory.
4) Develop a system such that each block computer could send specialized data back to the master computer. This would then tell the master computer to print some message on the terminal, or update some sort of display which would specific for that block, i.e. for each block, this specialized data would have a different meaning.
5) Professor Littman suggested that we could have a totally realistic layout if we could enable the master computer to change the status of the lights on the ends of the blocks automatically. This would require more modifications to the ISERV routine. In the real world, if a train enters one block, then that block's light turns red, and the previous block's light turns amber. Also, there is a speed restriction on trains in the previous block.
6) Add a feature so that the block computers can toggle the direction of a train.

7) Add a feature that allows the block computers to get the current speed, direction, and momentum of each train from the master computer.

LIMITATIONS OF THE SYSTEM

The Timing of Communications to and from the Hornby

Because the Hornby cannot respond to more than ten characters per second, there is a significant limit to the speed at which the master computer can monitor the layout and send commands. Here is a worst case analysis of the speed achievable using 300 baud.

Call the act of a train passing the bar code reader the *event.* When this event occurs, the signal sensor board automatically reads the bar code of the train. We are interested in how long it will take the data from this event to reach the master computer. This will depend on how many blocks are active in the layout, and how often commands have to be sent to the Hornby. Assume there is only one block computer active, that it is not requesting attention, and that the user is not entering data from the keyboard. In the worst case, the block computer's status will be polled immediately before the train passes the bar code reader. The master computer will therefore not detect the occurence of the event yet. The master computer will then execute a ***global_poll*** , which sends three characters (3/10 sec). The Hornby receives this data, has to wait for the next data cycle (< .040 sec), and sends this data to the block computer. The block computer then waits for the next data cycle (~ .040 sec) and then sends the data back to Arena's circuit. These two bytes of data are then assembled and sent back via RS232 to the master computer (2/10 sec). Total time : 5/10 + ~ .08 sec ≈ .58 sec. Then the procedure is followed almost exactly again when then master computer polls that block for its status. Thus, it takes up to 2 x ~ .58 sec ≈ 1.16 sec for the master computer to determine that the event occured. In this amount of time, the

train can have moved two or more feet down the track.

This is very significant for N-scale trains, and the delay will be multiplied by the number of active blocks in the layout. However, there is no way to overcome this limitation unless the Hornby is replaced by a circuit that can respond faster to incoming data.

The Data Set of the Hornby

There is one other important limitation to the system. The Hornby uses BCD to send data over the tracks to accessories, in our case the block computers. There are three nibbles that are available as data for the block computers; nibbles 3, 4, and 5 (if you start counting at zero).

Data sent over the tracks from the Hornby to the blocks takes this form:

nibble number

#0	#1	#2	#3	#4	#5	#6	#7
0100	0000	0000	$b_0b_1b_2b_3$	$l_0l_1l_2l_3$	$000h_3$	pppp	0100 00
start	$00 means accessory frame		block id	------data----		parity	stop

Since BCD is used, the block id can only take on the values 0-9, even though four bits normally allows allow 16 possibilities, $0-$F. This limits the number of blocks that can be independently addressed by the Hornby and thus by the master computer. Also, the data can only take on the values $01-$09 and $80-$89 (note that the Hornby won't send a $00). If all 8 bits could be used fully, then 256 different values would be possible. Although this may be an excessive number of possible data messages, I do think it would be advantageous to be able to address more than 9 block computer individually.

CONCLUSION

The project works very well given then limitations imposed on it by the Hornby unit. It responds as quickly as possible to the changing status of the blocks. The entire system could be made absolutely remarkable if a circuit were designed which would take the place of the Hornby, mimicing the track wave form which it sends out, but built around a more powerful microprocessor. If this circuit were designed so that it could communicate with the master computer as fast as the Hornby talks to the blocks, then the speed of the overall system would be quite satisfactory, and probably rather impressive. This would only require a rate of 30 or so characters per second (300 baud with no spacers), and if 1200 baud could be used (again, with no spacers) then the limiting factor would become either the speed at which the Hornby can send data to the blocks or the speed and efficiency of the Master Computer Program.

There are still a few quirks in the system. One is that the modified Hornby unit does not always reset itself correctly when turned on. See the User's Guide section for instructions on how to insure that the Hornby resets itself correctly.

Another more important quirk is that the polarity of the track power makes a difference in the reliability of the data transmission from the block computers back to the IBM PC. With some block computers that were tested with the system, only one polarity would give reliable data transmission. Further, others did not have a bad polarity orientation. The best way to find which polarity is better for any given computer is by experimentation. There does not seem to be a preferred wiring scheme.

Finally, the button on the side of the modified Hornby must be left in while the Master Computer Program is running. If it is set to the extended position while the program is running, the program should be restarted; it may continue to run fine, but there is a chance that it will become permanently unsynchronized with the data transmission and fail.

Appendix A: A User's Guide to Interactive Control and the Master Computer Program

Setting Up the Interactive Control System

1) First connect the modified Hornby unit to the train layout by connecting the power terminals of the Hornby to the track. Note that for some block computers there is a PREFERRED orientation of the track wires; the way to find out if this is true for a given block computer is by experimentation. First hook up the whole system; if one orientation results in many incorrect data transmissions, the other orientation should be used.
2) Connect the (blue and white twisted wire) RIB cable to the jack in the back of the modified Hornby unit. There should be a mini-plug on the end of this RIB cable. This cable should emanate from the signal sensor board. Not every test stand has a RIB cable connected, and some test stands with this cable have bad signal sensor boards with burnt out or missing chips
3) Connect an RS232 cable from the back of the modified Hornby to the COM1 RS232 port on the IBM-PC AT.
4) Push in the switch on the side of the modified Hornby to the remote position. The red light on the front of the Hornby should come on when the unit is then plugged in.
5) Plug in the Hornby.
6) Occasionally the modified Hornby will initialize itself incorrectly, so that it shifts all RIB data one bit to the left. To test for this, type

   0 [the number of an active block computer] >

   If the block is clear, $6C should appear on the right display of the Arena circuit. If something else shows up, especially $D8,

then unplug the Hornby, wait one minute, and replug it in. This will almost always solve the problem. If you have to do this more than twice, something else is wrong. Also,if the MPU program has been started, exit and re-start it now.

7) Call up the MPU.COM program on the PC. Type MPU <ret> to execute the program.

Manual Control of the Trains from the Keyboard

<- and ->

To enter data from the keyboard, first use tho horizontal arrow keys to move under the appropriate heading.

trains

To *add* a train to the set of currently controlled trains, type the number (1-16) of the train to be added (in decimal) and then hit **return** (or enter). When you add a train, it is set to the current speed and direction. To *remove* a train from the set, type a **minus** (-) sign, followed by the train number and then **return**. To *clear* all the trains from control, hold down the **shift** key and type @. Then, when you add the next train, its old speed , direction and momentum will be automatically recalled.

speed

To *increase* the speed of the train(s) currently under control, type a **plus** (+) sign. You do not have to hit return. To *decrease* the speed of the train(s), type a **minus** (-) sign. Again, no return is necessary. You can also *set the speed* of the train(s) equal to an absolute number by typing in the number (0-14) and then hitting **return**.

direction

To set the direction of the train(s) to *forward*, type **f** or **F**. To set the direction to *reverse*, type **r** or **R**. To *toggle* the direction of the train(s), type **t** or **T**. If the train(s) are

currently going forward, then the will stop and then go the other direction.

momentum

The momentum of a train determines (artificially) how fast an engine can accelerate/decelerate. To *increase* the momentum of the train(s), type a **plus** (+) sign. To *decrease* the momentum of the train(s), type a **minus** (-) sign. To *set the momentum* of the train(s), type in the momentum (1-4) and hit **return.**

sending data to a block computer

To send data to a block computer, type **B.** This will change the manual control display temporarily. You will then be prompted to enter the *number of the block* that you want to receive the data. Hit **return** when you have entered the number. Then you will be prompted to enter the *data to be sent.* This should be in the ranges 1-9 or 80-89. Again, hit **return** when you have finished.

backspace

You can use backspace to delete an incorrect numeric entry. However, you must hit **return** to finish entering a number once you have started. You cannot use backspace to escape from entering a number.

Automatic Interactive Control

Overview

For interactive control, the basic procedure is that the block sends a code (the *command code*) and a byte of data back to the master computer, which is then used to look up a series of commands from a list in the memory of the master computer. These commands are then executed. Each block has its own set of commands. Each command can carry out single or multiple tasks. The block computers always send two bytes back to the

master computer; these are WDATAH and WDATAL. WDATAH is always interpreted as the *command code* . WDATAL is available to be used as variable data which can be inserted into the commands before they are executed. Thus, instead of having 14 commands to set a train's speed to the 14 different speeds, one command can be used to set the speed to whatever value is returned in WDATAL. WDATAL can even be used to send two data values simultaneously, for instance the speed and direction of a train. One possible way to do this is to have the high four bits of WDATAL be the speed to set the train to and the low bit indicate the direction in which to send the train.

Defining/Editing Commands

Start the MPU program. To get to the editing menu from the layout monitoring routine, hit **PF4**. When the program starts up, you are automatically taken to the editing menu. To start a new file of commands for a block, select item 3. Then type in the block number of your computer when prompted. Note that block must be in the range 1-9; you must select a number in this range for your project computer.

You will then be shown a new display, the command editing screen. You will first be prompted for the *command code* which you wish to define. Enter this, followed by **return**. You must enter both digits. Backspace and delete both work, as do the left and right arrow keys.

You will then be asked to enter (or define) the command string. The basic syntax of a command string is:

command char/argument, command char/argument, . . . acknowledgement

The display will help remind you of the sytax. Spaces are not allowed between the command character and the argument, but

any number of spaces can be entered after an argument and before the next command character. If you want your command code to change the speed of engine 3 to speed 12, enter

```
                    'E' is the command char for Engine
S12. . .            ('T' can be substituted for 'E')
                    'S' is the command char for Speed
```

However, you are not done yet! You must be sure that each command sends the proper acknowledgement back to your block computer so that it knows when the command has been executed. To send this acknowledgement, add this command onto the end of every command string you define:

```
                        'C' for block Computer
S12  C3D82              ('B' can be substituted for 'C')
                        'D' for Data
```

This says,'set engine 3 to speed 12, and then send the acknowledgement 82 back to block #3.' Acknowledgements must be in the range of 01-09 or 80-89.

To save your command string, type Y or return when asked, 'is this ok?' You can enter as many command strings as you like; when you are done, type N to the question, 'another?'

More Examples of Command Strings

To set the direction of a train using interactive control, enter a command such as

```
E7  DF . . .  (Acknow.)        'F' for Forward
      or                       'D' for Direction
E7  F . . .   (Acknow.)        (use 'R' for Reverse)
```

Both of these commands will set engine 7 to go forward. Note that the 'D' is optional here; it is necessary, however, when you use variable arguments (see the next section on variable arguemants).

To set the momentum of a train using interactive control, enter a command such as

M3	'M' for Momentum

This command would give engine 4 a momentum of 3.

To send a message from one block computer to another, you would type something like

C2D84	Again, 'C' for block Computer and 'D' for Data

The allowable values for the block computer number are 1-9, and the data can take on the values 01-09 and 80-89.

Note that very complex commands are possible, although perhaps not too useful. Here is an example-

R S6 E3 E4E5 F S13 M4

Interpretation: Engine 2 go backwards at speed 6; engines 3, 4, and 5 go forward at speed 13 with a momentum of 4.

Block Computer Context

The master computer keeps track of the train(s) currently being controlled by each block, and the speed and direction last set by each block. Once a block computer has taken control of an engine, it is not necessary for the engine to be specified in later commands. For example, if a block first wants send engine 2 in reverse, and then later wants to send the same engine forward, these two commands could be used-

1st command - E2 R (Acknowledgement)
2nd command - F (Acknowledgement)

A somewhat fancier scenario might go like this-

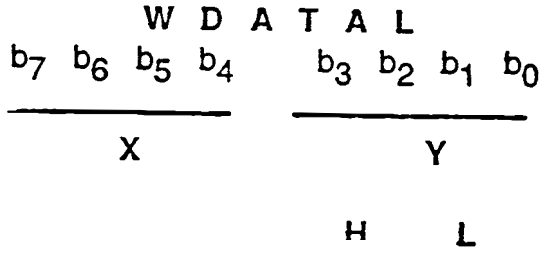
1st command - E5 S3 R (Acknow.)
2nd command - F (Acknow.)
3rd command - R S8 (Acknow.)

The first command would set engine 5 to speed 3 in reverse. The next command would send engine 5 forward, still at speed 3. The third command would send the engine backwards at speed 8. It will make the program run slightly faster if the engine is not re-specified every time.

Using Variable arguments with Interactive Control Commands

There is the option of using variable arguments with interactive control commands. This allows one command to be more general; for example, one command can be used to set a train to any speed. When the block computer sends its command code back to the master computer via the mailbox WDATAH, it also puts the desired argument value in WDATAL. The master computer then interprets this data correctly (as a train number, speed, etc.) and executes the command properly. A variable argument can be inserted anywhere a fixed argument goes; even the direction of an engine can be specified by a variable argument (with a little bit of necessary translation).

WDATAL is actually divided up into bit fields. This allows up to three parameters to be specified by WDATAL. An example is given just below. Here is a description of the bit fields in WDATAL and their names-



Example: If WDATAL = $46 (0100 0110_2), then

X = $04 ($0100_2$) Y = $06 ($0110_2$)

H = $01 ($01_2$) L = $02 ($10_2$)

These letters are entered in the same places that numeric arguments would be. For example, this command could be used to set train 4 to any speed:

E4 SY...(acknowledgement)

If this command were executed, and the block computer had loaded WDATAL with $09, then train 4 would be set to speed 9. If the following command were executed with WDATAL equal to $A0,

E8 SX...(acknowledgement)

then engine 8 would be set to speed 10 (A hex = 10 decimal). The speed of any train can be set using this command:

EX SY...(acknowledgement)

The high nibble (the high four bits) of WDATAL would specify the train number, and the low nibble would specify the speed. If WDATAL equalled $7C, then train 7 would be set to speed 12.

If WDATAL equalled $26, then train 2 would be set to speed 6. The momentum of a train can be set similarly. Setting the direction of a train can be done using variable arguments; if the bit field specified equals 0 then the train will be sent forward; if it equals 1 then the train will go in reverse. Example:

E2 S7 DH...(acknowledgement)
WDATAL equals $04

In this case, engine two would be set to speed 7 in reverse. Note that, in this situation, the letter D (for Direction) must be used, in order to tell the master computer how to interpret the data.

Saving to a disk

Once you have entered some command strings, you will want to save them permanently. To do so, get back to the main editing menu. Then select item 1. You will have to enter a filename and the block whose data you want to store. If there is already a file by the same name, you will be asked if you want to overwrite the file, enter a new filename, or quit from the file saving routine. There is one small problem-the program as it stands now only allows you to store or recall files from the current directory, and there is no way to change the directory from within the program. This will be amended soon.

Recalling a file from a disk

You can then retrieve your file from the disk using menu option 2. You have to supply the filename and the block for which to store the data. If the file is not in the current directory or does not exist, the program will yell at you and you won't be able to retrieve it.

Viewing a file

You can view all the commands in a command file at once by selecting menu item 4. The command file must be loaded (using the file retrieving routine) prior to this.

Running the Layout Monitor and Interactive Control Routine

To exit from the editing menu and actually execute the interactive control command that you have typed in, select menu item 5.

Quitting the program

Select menu item 6 if you want to quit the program. It will ask you a second time if you are sure you want to quit; you must answer yes in order to quit.

INTERACTIVE CONTROL reference sheet

from the block computer perspective

If a block computer wants the master controller/computer to carry out an operation, it must:

0) Clear the DFLAG mailbox ($3FF) at the very beginning of the program
1) Flash a *request code* (also called a *command code*) on the TIL 311 display
2) Wait for the DFLAG mailbox to clear to $00
3) Set the WDATAH mailbox equal to the *request code*
4) Set the WDATAL mailbox equal to the data for the requested command (not always necessary; see section on variable arguments)
5) Set the DFLAG mailbox equal to $FF
6) Wait until the proper acknowledgement appears in the RDATA

NOTE! You cannot count on a quick response from the master computer.
NOTE! You can **NOT** send $00 in both WDATAH and WDATAL. This will never reach the master computer-it gets stoppped by Arena's circuit.

FOR AUTOMATIC INTERACTIVE CONTROL

The project designer (or user) must run the MPU program and enter in the commands and the codes that these commands correspond to. The commands tell the MPU program what tasks to execute and what acknowledgement to send back to the block computer when the tasks have been completed.

AN EXAMPLE of Automatic Interactive Control:

Block computer #3 wants to have train 7 put in reverse

After stopping the train using a track kill circuit, the block computer does the following:

1) Displays the code $BD
2) Waits for the DFLAG mailbox to clear (i.e. = $00)
3) Loads the WDATAH mailbox with $BC
4) Loads the WDATAL mailbox with $00
5) Sets the DFLAG mailbox equal to $FF
6) Waits for the DFLAG mailbox to clear (this means WDATAL and WDATAH have been sent)
7) Waits for the code $89 to appear in the RDATA mailbox
8) (not always necessary) Clears the RDATA mailbox

After all of these operation, the track is re-powered. The train is now going in reverse.

The master computer program must be set up so that the command string for computer #3 corresponding to the request code $BD is

E7 R C3D89

E7 R tells train 7 to go in reverse. C3D89 tells the master computer to send the code 89 to computer #3. This is the acknowledgement that computer #3 is waiting for. See the Defining/Editing Commands section for complete syntax descriptions.

A more complex example:

E3E4 S2 F M3 C6D89

Engines 3 and 4, go forward at speed 2 with a momentum of 3. Then send the acknowledgement $89 to computer #6.

BIBLIOGRAPHY

All the sources listed in this report can be found in the famous blue book which Professor Littman compiles each year for the MAE 412 class.