THE HORNBY ZERO-ONE SYSTEM
for Microcomputer Control of Model Trains

M. G. Littman, R. Wotiz and G. Blaha
(Department of Mechanical and Aerospace Engineering
Princeton University, Princeton, NJ 08544, USA)

The Hornby Zero-One system, which is based on the use of dedicated four-bit microcomputers, allows for the control of 16 locomotives and 100 accessories. A discussion of how a hobbyist microcomputer can be used to operate the system also is provided.

IMPORTANT! This article describes projects undertaken in the USA. No assurance can be given regarding the availability of the equipment described, nor its operation from the different mains voltages and frequencies used in other countries such as the UK.

Hornby Hobbies have asked us to point out that the authors have no connection with Hornby and that Hornby cannot enter into any correspondence arising from this article.

The Hornby *Zero-One* system for the control of model trains provides an excellent example of how microprocessors may be used effectively for performing complex tasks involving electromechanical devices. The system uses four-bit microcomputers for independent control of up to 16 locomotives and 100 accessories on a layout.

Besides allowing for the simultaneous operation of multiple devices, it offers a number of attractive control features. For example, it is possible to simulate various levels of inertia in the control of individual locomotives. Each train can be set thus to respond to speed changes in a manner that is consistent with the load that is being pulled. It is possible also to arrange for a number of locomotives to operate in tandem. This means that several locomotives can be used for a single train.

The most important feature offered by the Hornby system, however, is the elimination of the extensive wiring that is needed with conventional model layouts. With the Hornby system, the tracks are used to transmit both power and control information from the master controller/power-unit to each of the devices on the layout. Accessories (turnouts, signals, *etc.*) are controlled by railside receivers, while locomotives are controlled by on-board receivers.

The basic components are shown in Figures 1 and 2. The control system relies heavily on the use of specialised four-bit microcomputers (Texas Instruments TMS1000), which are incorporated into the master controller and into the system receivers. Given the sophistication of the system and the fact that so many computers are used, it is remarkable indeed that a minimal system, including controller and two receivers, can be purchased for as little as $350.

This article is in two parts. The first will describe the methods of control and inter-computer communications. The second will discuss how a hobbyist microcomputer, such as the Rockwell AIM 65, can be used to replace the master controller in order to allow for the programmable operation of the network.

How Zero-One Works

An oscillograph of the track waveform produced by the master controller is shown in Figure 3. The signal is bipolar and consists of two distinct components: a low-frequency portion

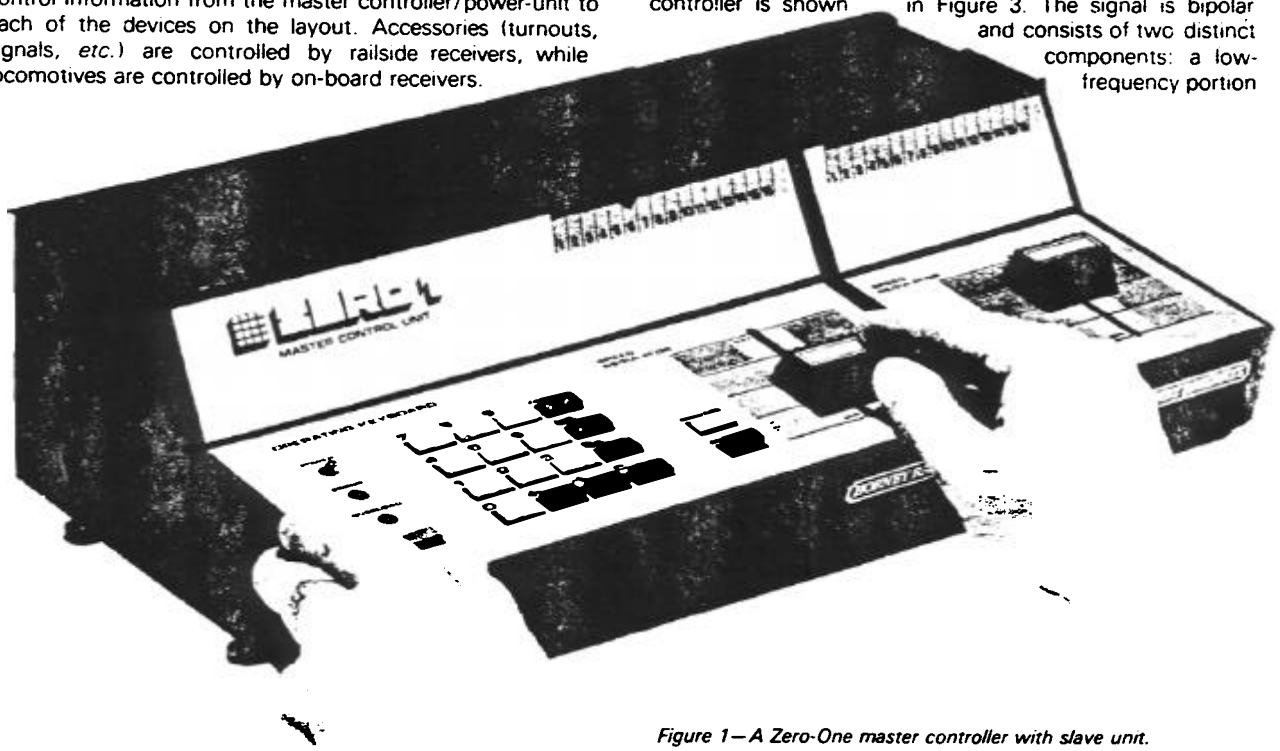

Figure 1—A Zero-One master controller with slave unit.

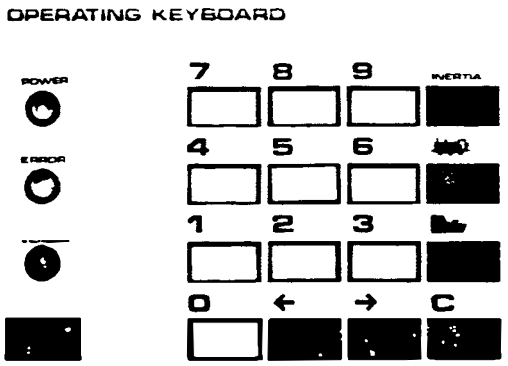

Figure 2—Close-up of the standard Zero-One keyboard.

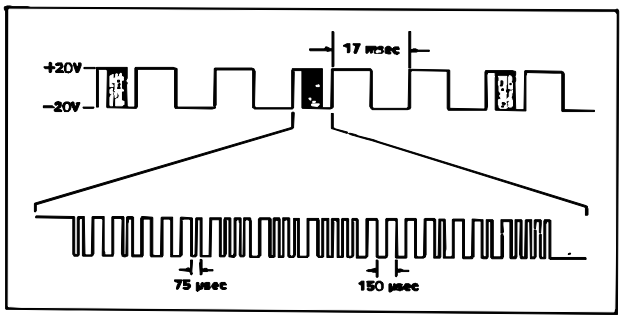

Figure 3—Track waveform with expanded view of control data frame.

at 60 Hz and a high-frequency portion at 6.6 or 13.3 kHz. The low- and high-frequency portions alternate in time. The low-frequency signal is used to provide power to devices, while the high-frequency signal is used in the transfer of control information to the two types of receivers. The control information contains instructions for the receiver computers as to the type and extent of power that is to be applied to the various devices on the layout.

Power for the operation of accessories and locomotives,

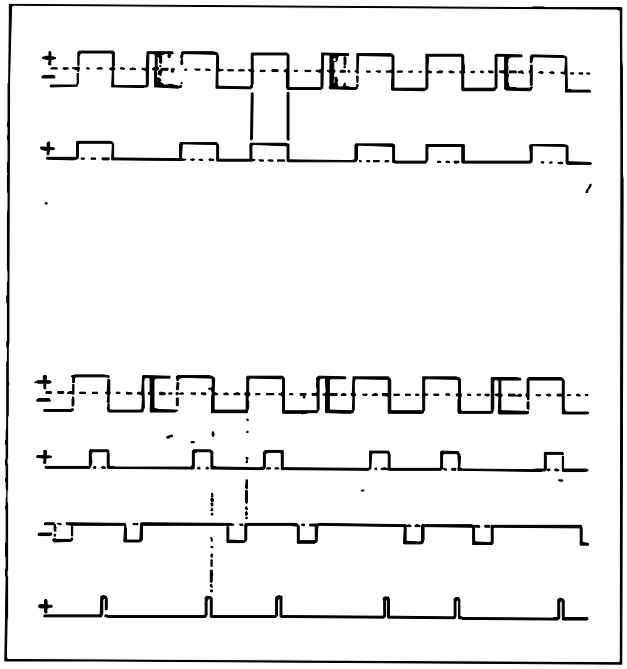

Figure 4—(a) Pulsed power for accessory operation.
(b) Pulsed power for locomotive operation at three different settings: half speed forward, half speed reverse, 1/14th speed forward.

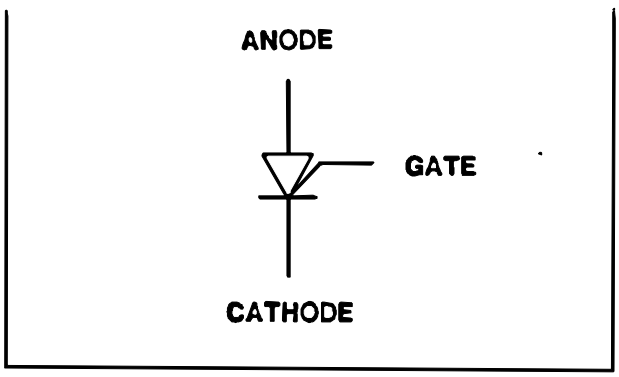

Figure 5—Symbol for a Silicon Controlled Rectifier (SCR).

which as noted above is derived from the low-frequency portion of the track waveform, is applied in a pulsed manner as shown in Figure 4. The pulse repetition rate is sufficiently high to ensure that motors, lamps and solenoids on the layout operate without jitter.

Accessories receive pulses of a single polarity having maximum possible duration as indicated in Figure 4a. The accessory state (left or right, red or green, *etc.*) is set by the application of these power pulses to either of two control lines on the given accessory (one control line for each state).

Locomotive operation is slightly more complicated. Locomotives receive pulses of either polarity having any of 14 durations. The locomotive speed is set by the duration of the power pulses—the shorter the pulse the slower the locomotive runs. The locomotive direction is determined by the polarity of the power pulses. Examples of locomotive powering for half forward, half reverse and 1/14th forward speeds are shown in Figure 4b.

Besides allowing for the operation of many devices, the power distribution system used here serves to reduce electrical noise on the tracks during control information transfer. This is because devices are not powered during the high-frequency portion of the track wave-form and the pulsed-powered devices generate little noise during their 'power-off' periods. The time sequencing of power and control data thus provides an elegant solution to the potentially bothersome problem of electrical interference.

As noted earlier, the application of pulsed power to a given device is effected by its associated receiver computer. The task of this computer is greatly simplified by the use of silicon

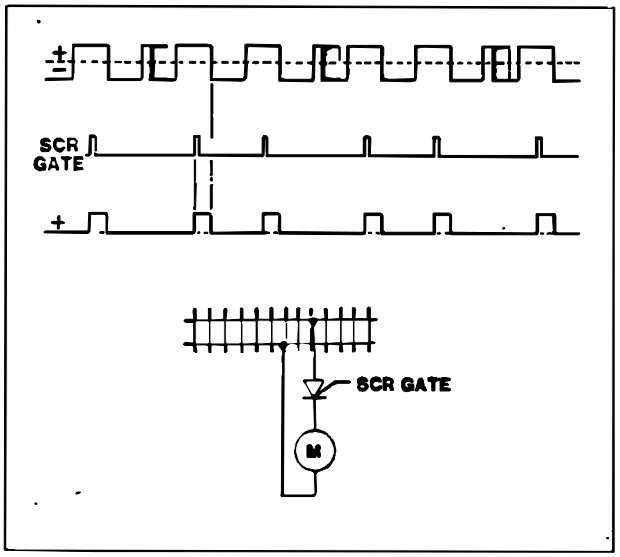

Figure 6—Example of SCR gate control for locomotive travelling at half speed forward. Below is a schematic of SCR-controlled motor. (The motor here is wired for unidirectional operation only.)

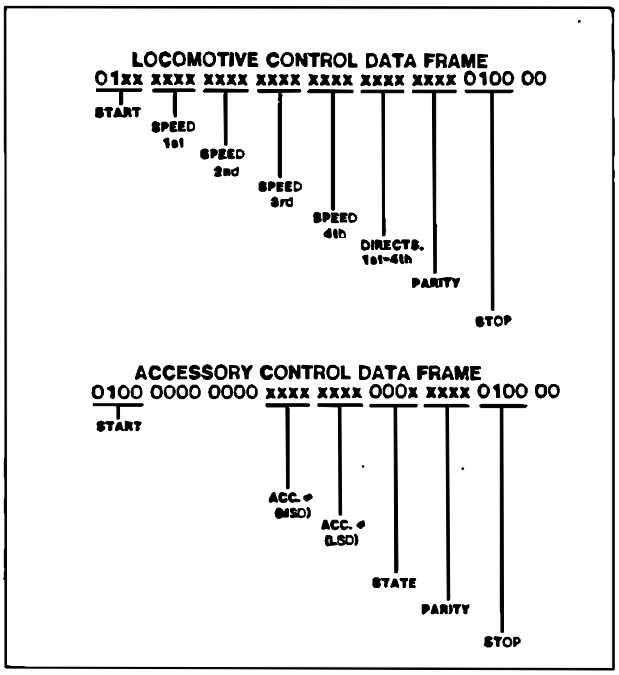

Figure 7—Bit assignments for locomotive and accessory control data frames.

controlled rectifiers (SCR). An SCR is an electronic switch that is turned on by the application of a forward bias pulse to its gate input (see Figure 5). Once it is on, the SCR functions like a diode, that is, it allows current to flow in one direction only. The SCR is turned off by a reversal of polarity. In its "off" state the SCR prevents current from flowing in either direction. The primary function of the receiver computer is thus to monitor the track waveform and apply 'on' pulses to the SCR at the proper moments. An example of computer-generated SCR control signals for a locomotive travelling at half speed in the forward direction is shown in Figure 6.

Control information to locomotive or accessory receivers is sent serially in a burst of bits. The bursts are interspersed between power cycles. A given burst contains 34 bits of information with assignments as shown in Figure 7. Since four-bit microprocessors are used, data has been grouped into eight four-bit 'nibbles'. (Bit numbers 33 and 34 are always trailing zeros.) Within each nibble the least significant bit (LSB) is sent first. The first group of four data bits in the stream is the START nibble. The last group of four bits is the STOP nibble. The seventh group of four bits is the PARITY nibble which is used for error checking. The remaining nibbles

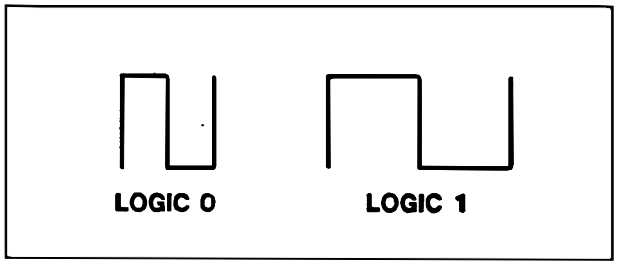

Figure 8—Frequency Shift Keying (FSK) scheme for serial data communications.

are used for communication with either locomotives or accessories.

We will consider locomotive control first. For locomotive control, the 34-bit burst (data frame) contains codes for the speed and direction of four locomotives. In order to transfer information to all 16 locomotives in the system, four separate datas frames must be sent. To distinguish between the data frames, identification numbers are assigned as part of the START nibble. Within a given frame, the second, third, fourth and fifth nibbles are used for speed control. Here, each nibble contains the speed of one of the four locomotives. The speed can assume any value between 0001_2 and 1112_2, where 0001_2 is the "off" code and 1111_2 is the "full power" code. Note that 0000_2 is not an allowed speed. This is because the 0000_2 code in nibbles two and three is a flag indicating that the data frame is for accessory control. The sixth nibble is used for direction control. Here, each bit contains the direction of one of the four locomotives. Logic 1 indicates forward and logic 0 indicates reverse.

For accessory control, a given frame of 34 data bits contains information to set the state of any one of the 100 possible devices on the system. Each accessory device is assigned an identification number between 0 and 99. The fourth and fifth nibble in the data stream are used for encoding the identification number of the selected accessory. Here the binary code decimal (BCD) system is used. Only the most significant bit (MSB) of the sixth nibble is implemented. This bit determines the state of the selected accessory.

The reliability of data transmission in both the locomotive and accessory data streams is enhanced because of the PARITY nibble which is used for error checking. The value of the PARITY nibble is preset by the Hornby controller on transmission of data such that successive addition of the eight nibbles in the data stream gives a 1111_2 sum. (The sum is actually a succession of ADC instructions—the carry bit is not cleared between instructions.) If the sum of the eight nibbles as calculated by the receivers does not equal 1111_2 then an error in data reception has occurred and the information is ignored.

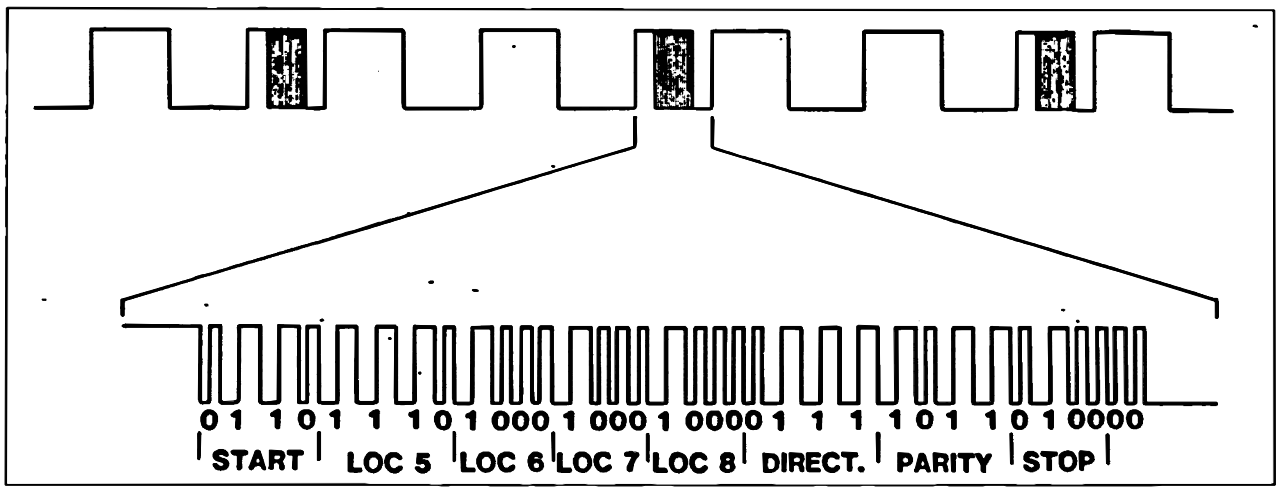

Figure 9—Detailed breakdown of bits and coding for locomotive control data frame number 01_2.

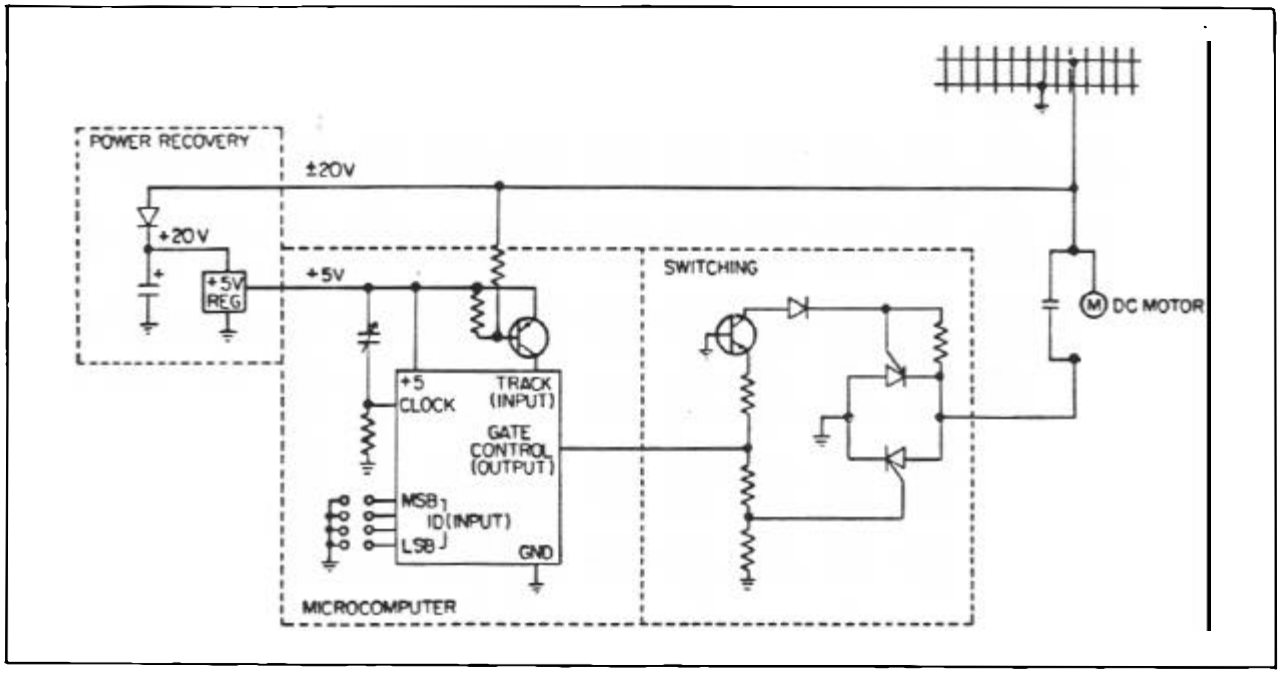

Figure 10—Schematic of locomotive receiver.

The Hornby controller in its normal mode alternates power cycles with locomotive control frames. The control sequence involves writing first to locomotives 1-4, next to locomotives 5-8, then to locomotives 9-12, and finally to locomotives 13-16. Once all 16 locomotives have been addressed, the controller repeats the sequence. Control information to accessory devices, on the other hand, is sent when keyed in on the controller keypad and only then. In principle, it is not necessary to communicate with a given locomotive after it has been set unless the speed or direction of that locomotive is to be changed. However, because of the possibility that the on-board computer periodically may lose its power due to poor electrical contact, the control information to all locomotives is sent repeatedly. In the worst case, if power were interrupted due to a faulty connection, only a hundred milliseconds or so would elapse before new control information would be sent.

To encode the 34-bit data stream onto the tracks the system of frequency shift keying (FSK) is used. A diagram of the FSK method is given in Figure 8. Each bit is coded as a square wave—a short period indicates logic 0, a long period indicates logic 1. The short and long periods of the FSK method are similar to the dits and dahs used in Morse code. A complete locomotive control frame is shown in Figure 9. Here locomotive 5 is to be set at reverse speed 0111_2 while locomotives 6, 7, and 8 are to be set at their minimum (off) forward speeds of 0001_2.

As a final topic, we examine the locomotive receiver to understand how the control information is implemented. A schematic diagram of the locomotive receiver is shown in Figure 10. The receiver consists of three subcircuits: power recovery, computer, and electronic switches. Power recovery is straightforward. The bipolar track voltage is half-wave rectified and filtered by the diode and capacitor. A five volt regulator is used to power the computer. (Unlike the motors, lamps and solenoids on the system, the computer receives continuous power.) The microcomputer monitors track voltage using one line of the input data port. Four other input lines are jumpered for selecting the locomotive identification number (16 possible values). One output line is used to control power to the motor through either of the two SCRs. (One of the SCRs is used for forward motion, while the other is used for reverse motion.) The microcomputer is used so effectively that the hardware needed to implement the system is minimal.

Using a Hobbyist Microcomputer

In this section we explain how the Hornby controller can be replaced with a hobbyist microcomputer. To accomplish this, a single line from an output data port is used in conjunction with a circuit to convert a TTL logic signal into a bipolar track signal and a program to generate a proper serial data stream. The circuit that was designed for this purpose is appropriate for use with almost any microcomputer, however, the program that was written for running the circuit is appropriate only for the Rockwell AIM 65 microcomputer with 4K RAM and a 1 MHz clock.

The circuit shown in Figure 11 converts a TTL logic signal into the required ±20 volt track signal. The circuit is short-circuit protected by use of the 7815 or 7915 voltage regulators, which shut down if too much current is drawn. The circuit has a current-handling capacity of 2 amps, which is sufficient for running about four locomotives. (The capacity of the circuit can be increased by using higher rated transformers and pass transistors, and reducing the value of the current limiting resistors R4 and R5.) In essence, the circuit is a bipolar power supply with a TTL controlled push-pull switching network. The choice of polarity is determined by the state of the analog comparator IC1. If the input IC1 is less than +2.5 volts then Q1, Q2, and Q3 are turned on and Q4, Q5, and Q6 are turned off. If the input to the analog comparator is greater than +2.5 volts, the opposite situation is the case. Diodes D1 and D2 protect against high-voltage electrical transients.

In order to drive this circuit, the program in Table I was developed. This program emulates the locomotive control aspect of the Hornby controller. Program operation is intended to be similar to the operation of the Hornby controller. To set locomotive 3 at speed 1110_2 in the forward direction, for example, one types L2<CR>followed by F (for foward) and D_{16} (speed). (Note that the locomotive identification codes and speeds are numbered differently here than before. To get the Hornby values add 1. Thus L2 refers to locomotive 3 and D_{16} refers to speed $E_{16} = 1110_2$.) Subsequent speeds and directions can be keyed in without retyping the locomotive identification string. R is used for reverse. Numbers between 0_{16} and E_{16} (Hornby's 1_{16} and F_{16}) are used to distinguish the 15 possible speeds. More than one locomotive can be specified by keying in L*i* L*j* L*k* . . . <CR>, where *i*, *j*, *k* are numbers between 0_{16} and E_{16}. This feature is useful for multiple heading of locomotives (two or more pulling in tandem).

TABLE 1

```
*=$010C
        JMP ST          ; F1 key starts program
                        ; program entry
ORB=$A000               ; output register B
T1L=$A004               ; timer 1 - low byte
T1H=$A005               ; timer 1 - high byte
T2L=$A008               ; timer 2 - low byte
T2H=$A009               ; timer 2 - high byte

MAIN PROGRAM:

*=$0D00
T0      .BYT 8          ; delay for zero FSK code
T1      .BYT 50         ; delay for one FSK code
T2      .BYT 6          ; delay between end of power and begin of data
T3      .BYT 30         ; delay for 60 Hz generation
ST      JSR INIT        ; initialization
        JSR DISP        ; display
TOP     JSR OUT
        CMP #'L         ; was L typed?
        BEQ LOC         ; yes, go to LOC
        CMP #'F         ; F?
        BNE NOF         ; no
        LDA #1
        STA DIR         ; set DIR to 1
        JMP CHG         ; change it
NOF     CMP #'R         ; R?
        BNE NOR         ; no
        LDA #0
        STA DIR         ; set DIR to 0
        JMP CHG         ; change it
NOR     CMP #$0D        ; <cr>?
        BNE NOCR        ; no
        JMP TOP
NOCR    JSR $EA7D       ; A reg. contains speed -- convert to hex
        BCS ERR         ; error
        CLC
        ADC #1
        STA SPD         ; update speed parameter
        JMP CHG         ; change FR list
ERR     LDA #' '        ; error entry
        JSR $E9BC       ; output space
        LDA #'?
        JSR $E9BC       ; output question mark
        JSR $EA24
ERR2    JSR OUT         ; wait for clear instruction
        CMP #'C         ; C?
        BNE ERR2        ; loop until clear
        JSR DISP        ; display
        JMP TOP         ; start over
CHG     JSR CHNG        ; execute change

        JMP TOP         ; start over
LOC     JSR $E9BC       ; echo L
        JSR OUT
        PHA
        JSR $E9BC       ; echo loco number
        PLA
        JSR $EA7D       ; convert loco number to hex
        BCS ERR
        LDY YSAV        ; update parameters
        TAX
        LDA ADSP,X
        STA ENTSP,Y
        LDA ADDIR,X
        STA ENTDIR,Y
        INY
        STY YSAV
        JSR OUT
        CMP #'L         ; a second loco?
        BEQ LOC         ; yes
        CMP #$0D        ; <cr>?
        BEQ COPY        ; yes
        LDY #0
        STY YSAV
        CMP #'C         ; C?
        BNE ERR         ; no
        JMP TOP         ; start over

COPY    LDY YSAV        ; copy speeds and directions
        STY CNT
COPY1   DEY
        LDA ENTSP,Y
        STA ACTSP,Y
        LDA ENTDIR,Y
        STA ACTDIR,Y
        CPY #0
        BNE COPY1
        STY YSAV
        JSR CHNG
        JMP TOP


INITIALIZATION SUBROUTINE:  initial setup

INIT    LDX #0
        LDA #$02        ; these are start bits
LOOP2   STA FRID        ; FRID = frame identification
        STA FR,X        ; FR = list of bits for signal
LOOP1   INX
        LDA #$01        ; 01 = zero speed for trains
        STA FR,X
        TXA
        AND #$03        ; loop which gives all 4 trains zero speed
        BNE LOOP1
        INX
        LDA #$0F        ; F = 1111 = forward direction for all 4
                        ;   trains
        STA FR,X
        INX             ; skip parity spot
        LDA #$02        ; stop bits
        STA FR,X
        INX
        LDA FRID        ; FRID has the following 4 values: 02, 06
                        ;   0A, 0E
        CLC
        ADC #4          ; this increments FRID to the next value
        CMP #$0F        ; tests if all frames are completed
        BMI LOOP2
        JSR PARITY      ; determines parity value for each of the 4
                        ;   frames
        LDA #1
        STA DIR         ; initializes current direction and speed
        STA SPD         ;   values
        STA $A002       ; DDRB, PB0 = output
        STA ORB
        LDA #0
        STA CNT         ; count
        STA YSAV        ; save Y
        STA T1L         ; reset timer 1
        STA T2L         ; reset timer 2
        LDA #$40
        STA $A00B
        LDA T3
        STA T1H
        RTS

CHANGE SUBROUTINE:  makes changes to FR list as necessary

CHNG    LDX CNT         ; if CNT = 0 there is nothing to change
        BEQ CHRET
CHNEXT  DEX
        LDA ACTDIR,X
        STA SAV         ; SAV = address of active direction bits
        SEC
        SBC ACTSP,X     ; get correct position of speed bit to be
                        ;   changed
        TAY
        LDA #$10        ; load bit mask
ROT1    LSR A
        DEY             ; shift bit mask into correct position
        BNE ROT1
        LDY DIR         ; DIR = current direction
        BEQ ZEROD       ; ZEROD = zero direction = reverse
        LDY SAV
        ORA FR,Y        ; DIR = 1 = forward
        JMP STDIR       ; mask is in accumulator; FR,Y = actual
                        ;   direction bit
ZEROD   EOR #$0F        ; get "0" in correct position of direction
                        ;   bits
        LDY SAV
        AND FR,Y        ; gets correct bits into accumulator
STDIR   STA FR,Y
        LDY ACTSP,X     ; ACTSP = active speed; addresses of active
                        ;   speeds byte
        LDA SPD         ; SPD = desired speed
        STA FR,Y        ; puts correct speed in FR list
        CPX #0          ; are all changes made?
        BNE CHNEXT      ; if not done, go to top of change and
                        ;   continue
        JSR PARITY      ; figures out new parity after changes
CHRET   JSR DISP        ; display
        RTS             ; return to main routine

PARITY SUBROUTINE:  calculates and stores correct parity values

PARITY  LDX #6          ; X = index for FR list; 6 = parity position
                        ;   in list
NEXTP   CLC
        LDA #0
        ADC FR-6,X
        JSR FBADC       ; FBADC = four bit add-with-carry
        ADC FR-5,X
        JSR FBADC
        ADC FR-4,X
        JSR FBADC
        ADC FR-3,X
        JSR FBADC
        ADC FR-2,X
        JSR FBADC
        ADC FR-1,X
        JSR FBADC
        ADC #2
        JSR FBADC
        EOR #$0F        ; complement result
        AND #$0F        ; strip high nibble
        STA FR,X        ; stores correct parity value
        TXA
        CLC
        ADC #8          ; advance to next parity position in FR list
        TAX
        CMP #31         ; test if parity for all 4 frames has been
                        ;   calculated
        BMI NEXTP
        RTS
FBADC   CMP #$10        ; emulates Four-Bit ADC instruction
        BMI FBRET
        SBC #$0F
        CLC
FBRET   RTS
```

TABLE 1 (continued)

OUT SUBROUTINE: outputs data frame using PB0 of 6522 VIA

```
OUT         LDX #0
NXTFR       JSR WT1
            DEC ORB          ; output logic 0 to track
            LDA T1L
            LDA #0
            STA T2L
            LDA T2
            STA T2H
            LDA #8
            STA BYTENO       ; BYTENO = 8
NXTBYT      LDA FR,X
            LDY #4           ; only want last 4 bits of each byte
LP1         LSR A            ; LSB → carry
            PHA              ; save accumulator
            JSR WT2
            INC ORB          ; output logic 1 to track
            BCS SEND1
            LDA T0           ; set delay
            JMP SENDIT
SEND1       LDA T1
SENDIT      STA T2L
            LDA #0
            STA T2H
            JSR WT2
            DEC ORB          ; output logic 0
            LDA #0
            STA T2H
            PLA
            DEY
            BNE LP1
            INX
            DEC BYTENO       ; BYTENO = BYTENO - 1
            BNE NXTBYT       ; do it again
            JSR WT2          ; output trailing zeros
            INC ORB          ; output logic 1
            LDA T0
            STA T2L
            LDA #0
            STA T2H
            JSR WT2
            DEC ORB          ; output logic 0
            LDA #0
            STA T2H
            JSR WT2
            INC ORB          ; output logic 1
            JSR WT1
            DEC ORB          ; output logic 0
            LDA T1L
            JSR WT1
            INC ORB          ; output logic 1
            LDA T1L
            JSR WT1
            DEC ORB          ; output logic 0
            LDA T1L
            JSR WT1
            INC ORB          ; output logic 1
            LDA T1L
            CPX #31
            BPL NONXTF
            JMP NXTFR

NONXTF      JSR $E907        ; RCHEK - scan keyboard for character
            CPY #0
            BPL RTS1
            JMP OUT
RTS1        RTS

WT1         LDA #$40
            JMP W2
WT2         LDA #$20
W2          BIT $A00D        ; time out?
            BEQ W2
            RTS              ; yes
```

DISPLAY SUBROUTINE

```
DISP        JSR $EA24        ; CRCK
            LDA DIR          ; direction
            JSR $EA51        ; NOUT
            LDA #' '
            JSR $E9BC        ; OUTALL
            LDA SPD          ; SPD = speed
            JSR $EA51        ; NOUT
            JSR $EA24        ; CRCK
            RTS
```

VARIABLES

```
CNT         *=*+1            ; count
FRID        *=*+1            ; frame id
SAV         *=*+1
YSAV        *=*+1            ; y reg temp storage
DIR         *=*+1            ; direction
SPD         *=*+1            ; speed
BYTENO      *=*+1            ; byte number
ENTDIR      *=*+16           ; entered directions
ENTSP       *=*+16           ; entered speeds
ACTDIR      *=*+16           ; active directions
ACTSP       *=*+16           ; active speeds
ADDIR       .BYT 5,5,5,5,13  ; addresses of direction bits in FR list
            .BYT 13,13,13,21,21,21
            .BYT 21,29,29,29,29
ADSP        .BYT 1,2,3,4,9   ; addresses of speed bits in FR list
            .BYT 10,11,12,17,18
            .BYT 19,20,25,26,27,28
FR          *=*+32           ; FR is list of 32 nibbles (4 frames)
.END
```

AIM 65 MONITOR ROUTINES

address	name	function
EA7D	HEX	Converts A register from ASCII to hex. Number is placed in low nibble, high nibble set to zero
E9BC	OUTALL	Outputs ASCII character in A to display.
EA24	CRCK	Forces output of characters in print buffer.
EA51	NOUT	Converts low nibble in A to ASCII and outputs to display.
E907	RCHEK	Scan keyboard for input.

AIM 65 VERSATILE INTERFACE ADAPTER (VIA) -- 6522

address	function
A000	port B output data register
A002	port B direction register: 0 = input, 1 = output
A004	timer 1 - low byte
A005	timer 1 - high byte (starts timer when written)
A008	timer 2 - low byte
A009	timer 2 - high byte (starts timer when written)
A00B	auxiliary control register
A00D	flag register

AIM 65 PROCEDURES (w/ 4K RAM)

EDITOR -- Begin source at 0200. Avoid any unnecessary spaces. There is absolutely no room for comments.

ASSEMBLER -- Symbol table at 0D00-0FFF. Because of limited space, object code must be assembled to tape. Core image can be loaded from tape using "L" command.

To run program, press "F1" key.

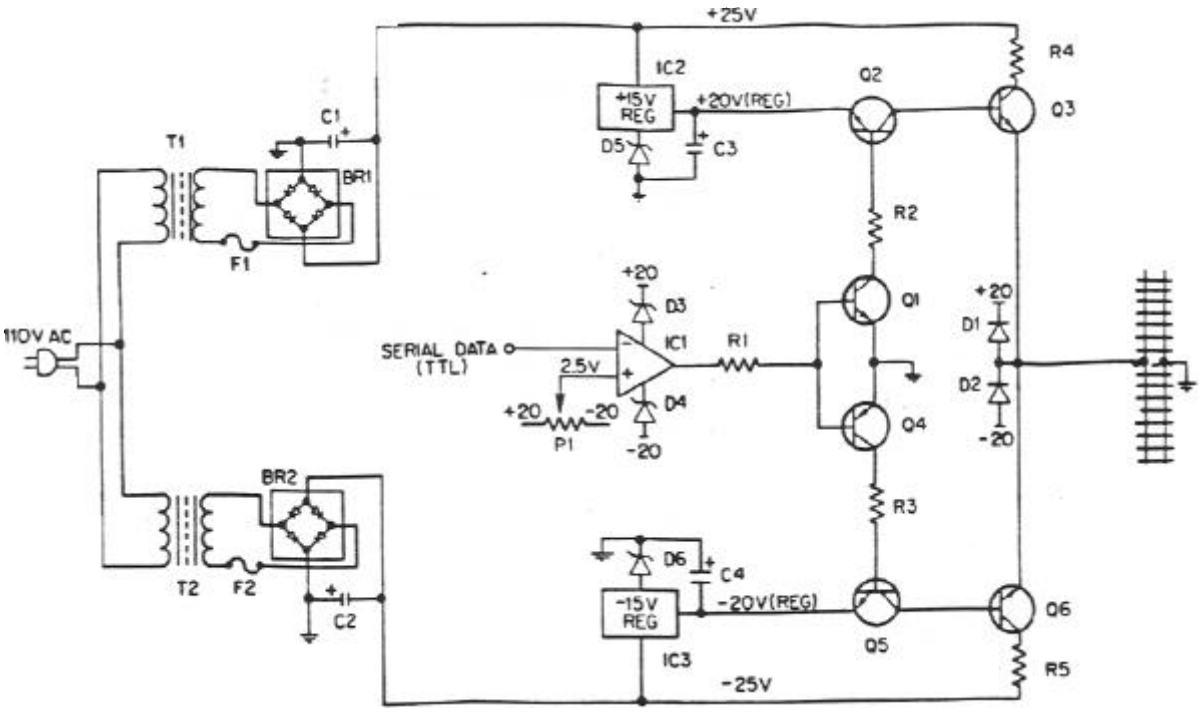

Figure 11— Schematic of Microcomputer-to-Track interface.

Note that the last value set for a given locomotive speed and direction is remembered, even while a different locomotive is being addressed through keyboard operation. It is possible, thus, to have locomotive 6 going forward at speed 5_{16}, while locomotive 9 is travelling in reverse at speed C_{16}, and so on.

If you modify this program to run on other computers, care must be exercised with regard to the timing of the data stream. This program makes use of the Rockwell AIM 65 VIA timer to achieve the proper timing. In general, the program is not directly transferrable to other machines even if a 6522 VIA timer is available.

This work is an outgrowth of an undergraduate laboratory course in microcomputer control. Students in this course design and build dedicated 6502-based microcomputers for the control of projects associated with an N-scale (1:180) layout. Custom accessory receivers are developed, and then programmed (using EPROMs), for specific tasks. Over the last two years several projects have been built. The work described here has been directed at integrating the course projects into a computer-operated network. This work has been supported by the Department of Mechanical and Aerospace Engineering, Princeton University. We thank A. Lo, J. Bittner, Jr., E. Conover, and N. Wyatt for helpful suggestions.

Parts List for Figure 11

T1, T2	25 VAC, 2 amp transformer
BR1, BR2	2 amp, 50 PIV bridge rectifier
F1, F2	2 amp SLO-BLOW fuse
IC1	741C operational amplifier
IC2	7815 (TO-220) voltage regulator
IC3	7915 (TO-220) voltage regulator
D1, D2	1 amp, 200 PIV diode
D3, D4, D5, D6	1 watt, 5.1 volt Zener diode
C1, C2	2200 μF, 50 WVDC electrolytic capacitor
C3, C4	0.1 μF, 50 WVDC disc ceramic capacitor
R1	1/4 watt, 5600Ω resistor
R2, R3	1 watt, 330Ω resistor
R4, R5	10 watt, 5Ω resistor
Q1	2N3904 transistor
Q4	2N3906 transistor
Q2, Q6	GE69 power transistor (TO-220)
Q3, Q5	GE66 power transistor (TO-220)
HS1, HS2	TO-220 heat sinks for Q3 and Q6
P1	1/4 watt, 50 KΩ potentiometer

Note: All parts may be obtained from Radio Shack.

APOLOGY: REGULAR SERIES

We regret that, owing to circumstances beyond our control, we have been unable to include in this issue the third in Henry Best's series of articles *Towards Full Automation* and the second in Nigel Taylor's series *Interface and Interlock.* We hope to include these in the next issue.