A FIRMWARE MODEL RAILROAD

INTRODUCTION

One of the most difficult aspects of scale model railroading is that of achieving realistic operation. Model locomotives have a tendency to be jerky, particularly at low speeds. In fact, it is virtually impossible to run most model locomotives at speeds of less than 15 scale miles per hour using conventional controllers.

The problem stems from the heavy frictional losses in the mechanism, particularly in the gear train. Because of these losses, motor current is high at all speeds and varies not so much with speed as with the angular position of the drive wheels. As a result, the locomotive tends to stall or jerk at low speeds with a rheostat or other types of controllers which have high output resistance. Better control can be obtained by using a controller with low output resistance, but speed fluctuations and stalling are still present with most models at low speeds.

A more recent technique[1] is to use full-voltage pulses of controlled width (possibly superimposed on DC) to obtain the desired average output. Because the pulse voltage is high enough to overcome the friction in the mechanism, stalling is no longer a problem. However, the average speed still varies with load, with the result that the locomotive will slow down suddenly when it starts up a grade or enters a sharp curve, unless the pulse width is adjusted to compensate.

[1]Fyffe, David, "Pure-pulse Transistor Throttle," *Model Railroader*, Vol. 32, #1 (January 1965), p. 63.

ACKNOWLEDGMENTS

The author wishes to thank Dr. Damian Gouleff and the Ontario Science Centre for making available the Centre's ELI MMD-1 microcomputer for the development of this controller.

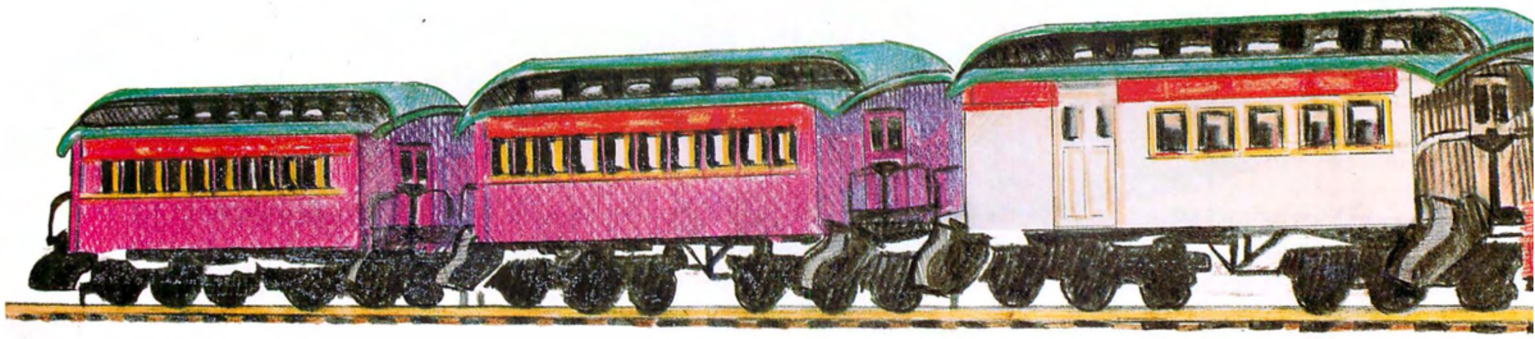
Servo techniques can be used to offset both frictional losses and load fluctuations, but a linear servo must be trimmed to compensate for each locomotive's motor resistance. The controller described here uses a combination of pulse and servo techniques to give excellent control for a wide variety of locomotives at speeds ranging down to less than 1 scale mile per hour.

Another objective in controller design is to simulate the enormous inertia of a locomotive. Many controllers do this by charging and discharging a large capacitor to obtain a slowly varying control voltage. In the present design, inertia simulation is done entirely by software, incrementing and decrementing a register to vary the speed slowly.

HARDWARE CONFIGURATION

The controller configured here uses an 8080 microprocessor with 512 bytes of ROM, a minimum of RAM (only about 16 bytes are actually used), one input port, and two output ports. The input port and one of the output ports are used in conjunction with a digital-to-analog converter (DAC), a bank of comparators, and a software analog-to-digital conversion routine to implement a cheap form of multi-channel analog input. The other output port is used to pulse the output amplifier.

The output amplifier is bipolar (complementary), allowing direction control to be done in software. As a result, the controller can be programmed so that if the reversing switch is changed while the train is running, instant reversal does not occur; instead the train will gradually slow



CONTROLLER

By Gifford Toole President, TRACE Toronto Region Association of Computer Enthusiasts

to a stop and then begin accelerating in the opposite direction.

Two of the analog inputs are used to measure the motor voltage, one for each direction, giving 8-bit resolution in either direction. A third analog input is used for the throttle setting, which is a voltage between 0 and 5 volts (derived from a pot). The spare bits of the input port are used for the direction switch and the brake.

Detailed schematics for the bipolar output amplifier, low-pass filter, and analog-to-digital conversion circuits are shown in Figures 2, 3 and 4.

The output is determined by the state of two bits of latched output Port 1. When bit 0 is one and bit 1 zero, transistor Q1 turns on, which in turn switches Q2 and Q3 on, giving full (nearly 12 volts) positive output. When bit 0 is zero and Bit 1 is one, Q1 is reverse-biased but Q4, Q5 and Q6 turn on, giving full negative output. When bits 0 and 1 are equal, all transistors are off and no output current results. Note that under no circumstances can both halves of the output amplifier be biased on simultaneously. Q7 and Q8 provide current limiting, turning on only if the average output current exceeds 1.4 amps or the instantaneous output current exceeds 3 amps. (Stated more exactly, Q7 and Q8 limit the magnitude of the output current I such that $.22\,I + .5\,I_{av} < V_{be}$.) The output clamping diodes limit inductive voltage transients from the motor when the transistors switch off.

The low-pass filter is really two simple low-pass filters, one inverting and the other non-inverting. This is done because the DAC produces an output of one polarity (+) only. When the amplifier output is positive, the non-inverting filter output is positive and is converted to an 8-bit (unsigned) number by the ADC routine, while the inverting filter output is negative and converts to zero. This technique avoids the offset errors which would crop up if the DAC output were offset to accommodate bipolar signals, and, at the same time, maintains 8-bit significance in the conversion.

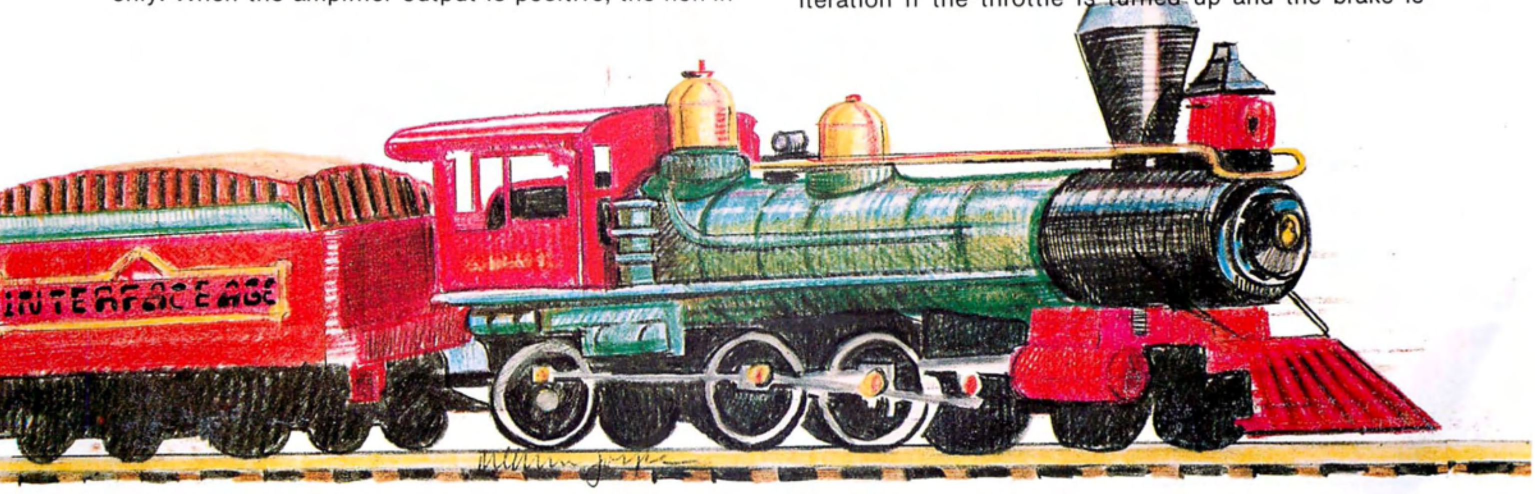
The software is described in Figure 5 as a set of APL functions. The APL notation is used because it affords a more concise, detailed description than is feasible with flowcharts.

SOFTWARE

The main program consists of an infinite loop which samples the motor voltage which, between pulses, is proportional to the speed, compares it to the desired speed, calculates how long the next pulse should be, delivers a pulse, then checks the control settings and adjusts the desired speed accordingly.

Analog-to-digital conversion is done by successive approximation (function ADC). The inputs from all but one of the comparators are ignored during any one conversion; a mask supplied as an argument to the ADC routine determines which bit is used. The next pulse width is calculated by a proportional-plus-integral control algorithm (next pulse).

The current desired speed is a 16-bit number (speed), kept in the HL register pair, of which the high-order eight bits are used in the speed comparison and pulse width calculation. The register is incremented on each iteration if the throttle is turned up and the brake is



released. If the brake is on, the register is decremented. Drag is simulated by applying a small decrement on each iteration. Negative numbers are used when the locomotive is running in reverse.

CONCLUSION

The firmware approach affords considerable flexibility in the design of a controller. Extra features can be added by simply changing the software (e.g., loss of steam pressure, running out of water or fuel, etc.). The unit can also be programmed to control two or more locomotives independently. The only additional hardware required consists of an output amplifier and three analog input channels.

```
      ∇ MAIN
[1]     SPEED← 0
[2]     PULSE← 0
[3]    LOOP:PULSE← PULSE NEXTPULSE SPEED
[4]     DELIVER PULSE
[5]     SPEED← SPEED + (THROT SPEED ¬ (DRAG SPEED) ¬ BRAKE SPEED)
        ÷ TIMECONST
[6]    →LOOP
      ∇

      ∇ P← PULSE NEXTPULSE SPEED
[1]     P← (0.5 × PULSE) + 2 × (SPEED - CURRENTSPEED)
      ∇

      ∇C← CURRENTSPEED
[1]     C← (ADC FORWARD) ¬ ADC REVERSE
      ∇

      ∇ R← ADC MASK;C
[1]     R← 128
[2]     C← 128
[3]    L:R OUTPUT 0
[4]     C← C ÷ 2
[5]     →EXIT IF C < 1
[6]     →LOW IF 0∧.= MASK INPUT 0
[7]     R← R - C
[8]     →L
[9]    LOW:R← R + C
[10]    →L
[11]   EXIT: →RET IF 0∧.= MASK INPUT 0
[12]    R← R ¬ 1
[13]   RET:
      ∇

      ∇ T← THROT SPEED;D
[1]     D← DIRECTION INPUT 0
[2]     →MAX IF D∧.= 0
[3]     SPEED← ¬ SPEED
[4]    MAX:T← (ADC THROTTLE) × (MAXSPEED + SPEED)
[5]     →RET IF D∨.≠0
[6]     T← ¬ T
[7]    RET:
      ∇

      ∇ D← DRAG SPEED
[1]     D← 0
[2]     →RET IF SPEED = 0
[3]     D← DRAGCONST
[4]     →RET IF SPEED > 0
[5]     D← ¬ DRAGCONST
[6]    RET:
      ∇

      ∇ DELIVER PULSE;D;C
[1]     →NEG IF PULSE < 0
[2]     D←FORWARD
[3]     →SH
[4]    NEG:D← REVERSE
[5]     PULSE← ¬ PULSE
[6]    SH:PULSE← 2 × PULSE
[7]     C← 0
[8]    L: →OFF IF C = PULSE
[9]     →ON IF C = 0
[10]   L1:C← C + 1
[11]    →L IF C≠256
[12]    →RET
[13]   ON:D OUTPUT 1
[14]    →L1
[15]   OFF:0 OUTPUT 1
[16]    →L1
[17]   RET:
      ∇
    ∇   B← BRAKESPEED
[1]     B← 0
[2]     →RET IF 0∧.= BRAKE INPUT 0
[3]     →RET IF SPEED = 0
[4]     B← BRAKECONST
[5]     →RET IF SPEED > 0
[6]     B← ¬ BRAKECONST
[7]    RET:
      ∇
```

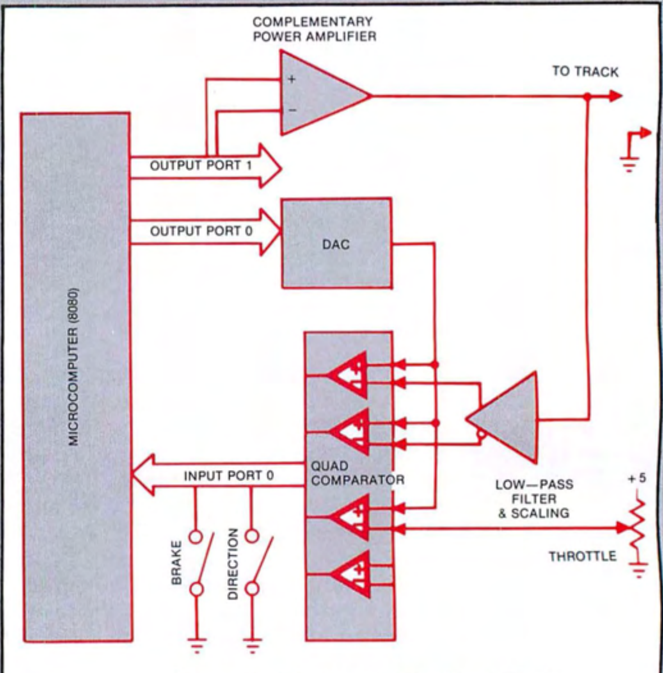


Figure 1. Hardware Configuration

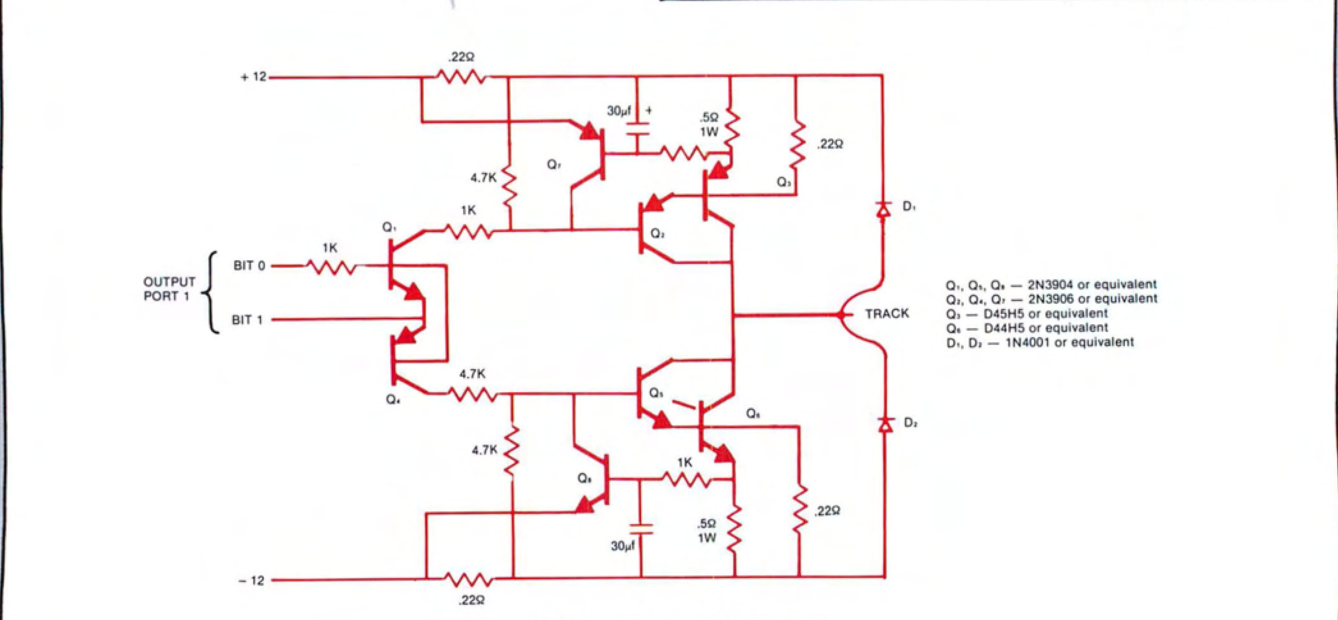


Figure 2. Output Amplifier

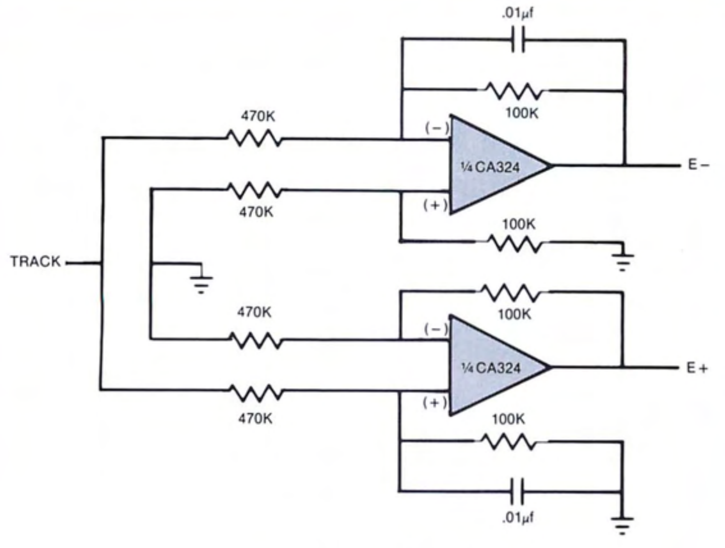


Figure 3. Low-Pass Filter

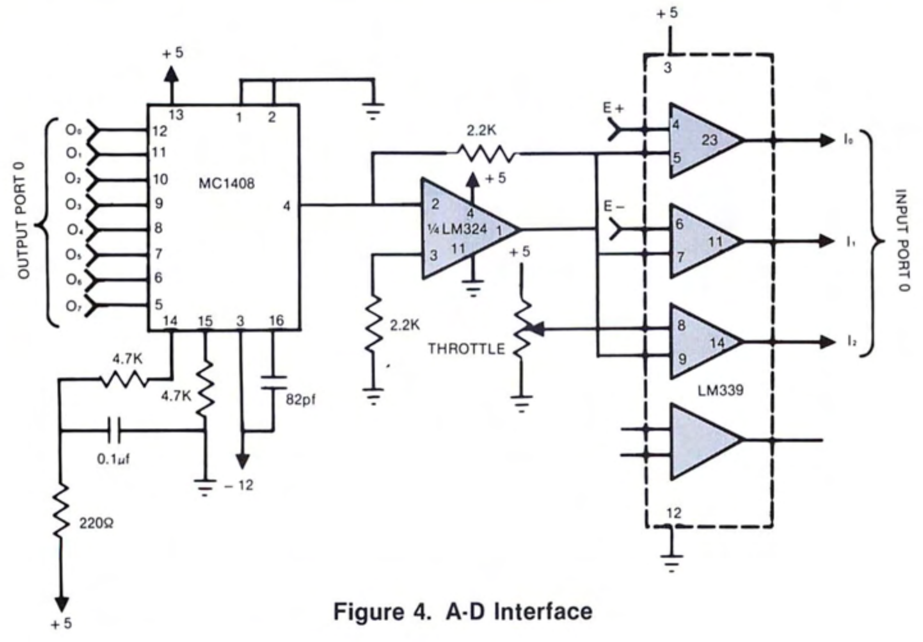


Figure 4. A-D Interface

1. The following constants are used

TIMECONST —	256
FORWARD —	0 0 0 0 0 0 0 1
REVERSE —	0 0 0 0 0 0 1 0
THROTTLE —	0 0 0 0 0 1 0 0
DIRECTION —	0 0 0 0 1 0 0 0
BRAKE —	0 0 0 1 0 0 0 0
DRAGCONST —	4
BRAKECONST —	64

2. The Function OUTPUT is assumed to output its left argument to the port whose number is given by the right argument.

3. The Function INPUT is assumed to input a byte from the port whose number is given by the right argument, and "AND" the eight bits with the left argument to produce a result.

Figure 5. Controller Algorithms