CS 452 Assignment 0

<Name>
<Student ID>
<Date>

1 Overview

This program controls a Märklin 6051 train set. Commands can be issued to start, stop, and reverse trains, adjust their speed, and change switches on the track. At the same time, the program will process sensor data from the track and keep a display of the most recently triggered sensors.

2 Operating Instructions

The executable file can be found under <Path to file>. To run the program issue the following commands in Redboot:

1. load -b 0x00218000 -h 129.97.167.12 ”ARM/<Path to file>”
2. go

The program will then initialize the train set and the display, and then will accept the following commands to control the train set:

- `tr train_number train_speed`

  - Set any train in motion at the desired speeds (from 0-14, where 0 is stop). To turn on the train lights, add 16 to the desired speed.

- `rv train_number`

  - Reverse the desired train.

- `sw switch_number switch_direction`

  - Throw the given switch to straight (S) or curved (C).

Note that the program assumes that only trains 35, 37, 39, and 41 will be used.

3 Program Description

3.1 Main Polling Loop

Each pass of the polling loop will do the following:

1. Read the current time from the 40 bit timer register. This timer was used because it counts up, while the others count down. If any trains have received a reverse command, it will compare the current time with the projected time that the train will be stopped, and if it is elapsed, issue the reverse command and the subsequent speed command.

2. If one millisecond has elapsed since the last iteration of the loop it will update the clock display.

3. If there is data to send to the train and the train controller is ready to receive a byte, it will send a byte.

4. If there is output to the terminal and the UART transmit buffer is not full, it will send a byte to the terminal.

5. If the train controller is sending sensor data, read in a byte and process it, updating the display of triggered sensors if necessary.

6. If there is input from the terminal, read a byte and echo it back to the terminal. If the character is a new line, it will parse the command and if it is valid, issue the corresponding command to the train controller.

7. If all data bytes from the previous sensor data request have been received, it will issue a new request for data from all sensors.

Reasoning: The loop is fast enough that on each pass of the loop the program will be able to input/output one character to/from the terminal and the train set, as well as doing all processing of input. Input/output is buffered so that nothing gets lost and is fast enough that the buffers will never get full.

3.2 Parsing

The command parsing algorithm used simply does a switch on the first letter of the command received since there are only three valid commands. In the interest of keeping it fast, it uses many hard coded values to determine if the

command is valid and instead of using a printf function, will simply translate a2i/i2a directly.
The track sensor data parsing simply uses bitwise operators to determine which sensors have changed and a ring buffer that holds string representations of the last 10 triggered sensors.

3.3 Data structures

The input/output buffers are implemented as ring buffers, which are allocated on the stack and maintain pointers to the first valid char in the buffer and the last valid char. They are first in, first out (and are therefore called queues in the code). This allows the program to use a relatively small area of memory for each buffer, that will simply wrap to the beginning when it reaches the end and will not run out of space.
There is also a structure for keeping track of the speed of each train, whether or not they have received a reverse command, and the time that the train will be stopped after receiving the reverse command, so that in the event of a reverse, the program can wait the appropriate amount of time before issuing the reverse and subsequent speed up command to the proper train. It is likewise very fast.

3.4 Known Bugs

- The program is not quite fast enough to handle input copy/pasted into the terminal. It is fast enough to handle most of it and all other cases, however.
- The arrow keys are not handled from the terminal and will produce undesirable results when used.

4 Files Submitted (with md5 hashes)

- 8af75849b7e5551e37ea210a1a2646f2 <Path to file>
- 0af431f2447f8006aef5150536e08b9e <Path to file>
- ffae889eb8b702f77fe3cf33b5b34427 <Path to file>
- f4d74a8a815cfd4262412d232ebceff0 <Path to file>

- 37348c3b11990539c3549d0edc1fe936 <Path to file>
- e22edcaf56bda11e4ac128ca6891aa86 <Path to file>
- 34db12e3a32e3731d195eb7e0c825a4d <Path to file>
- 691589f59671558d9d5f14e20045b0fb <Path to file>
- 9218d7ba3117c872eb5ce2d7deb2ae2d <Path to file>

5 Question Responses

i. I know that my clock does not miss updates or lose time because the clock only needs to update every 100 milliseconds, and one iteration of my main loop runs in the worst case on average between 0.4-0.5 milliseconds (found out by timing the main loop).

ii. The train hardware takes ~17ms to send the first byte of a sensor query, and ~61ms to send all 10 bytes.