CS452 Assignment 0

Christoph Ulshoefer, student #20751216

January 10, 2018

1 Operation

1.a Loading up the program

Do the following in RedBoot to boot up the program:

1. load -b 0x00218000 -h 10.15.167.5 "ARM/csulshoe/main.elf"
2. go

In the first few seconds, the train set is initialized, i.e. all switches (except for two of the four three-way switches) are set to curved and GO is issued. Issuing commands and triggering sensors during this time should be possible. However, the time is only displayed after 10 seconds.

1.b Commands

- `tr <train> <speed>`
  Sets the train (1-80) to speed 0-15 (15 for reverse). The headlights should be toggled every time the speed of a train is updated internally.
- `rv <train>`
  Reverses the train with number 1-80. Note that the train decelerates naturally.
- `sw <switch> <direction>`
  Switches switch (valid switch numbers: 1-14, 153-156). On switching 153-156, the other half of the three-way switch is set to the opposite state to ensure trains never derail.
- `q`
  Issues STOP and quits.
- `GO`
  Issues GO
- `STOP`
  Issues STOP

2 Structure

2.a Buffers

Input/Output (ring) buffers for the train/terminal are used, as well as a separate buffer for pending turnout commands. The turnout buffer is necessary to ensure that only one turnout is set at a time. Additionally, a ring buffer is used for the list of recently triggered sensors, and for the list of trains that have received reverse commands.

2.b Main Polling Loop

1. Get the current time
2. Update the time on screen, if 0.1s has passed since the last update
3. Check if a reverse command has been issued in the last few seconds. If enough time is elapsed and the train has come to a stop, issue reverse. If some time has elapsed after reverse has been issued, reaccelerate
4. Process buffered COM1 content: If two bytes or more are in the buffer, read them in and reprint sensors, if a sensor value has been read. If the most recent sensor has been triggered again, ignore it. If all sensor bytes have been received, or if after 100ms we have not receive all sensor bytes, query all sensor modules again. Before querying the sensors, reprint the current status to the terminal. If an overrun error is encountered, dump the current sensor readings (we may have lost a byte in the middle and would show an incorrect sensor) and wait for the next sensor values. In practice, this method of dealing with overrun errors does not lose sensor values, and after the initialization phase, I was only able to create overrun errors by rapidly switching sensors many times a second, which would not occur during normal train operation.
5. Turn off solenoid, if necessary
6. Send switch commands, if we can do so and the above time constraints are met
7. If there is a byte in an input buffer, read in that byte. If our buffer is full, dump the buffer to let the developer know to pick a larger buffer size, and what caused the overflow. Since we have multiple 10s MB of space, the 80 KB used by I/O buffers (20000 elements each) are reasonable, and bug-free implementations should never in any way make the buffer overflow
8. If we have a byte to send and the FIFO/register transmit status and transmit busy is not set (an if we are talking to the train, CTS is asserted), we send a byte to the corresponding channel. We can send 6 bytes in total to COM2, spread over the entire loop, and 1 byte per iteration at most to COM1. Since the terminal supports a 40x higher baud rate, we only check twice whether we received a byte.
9. If we have received a byte from the terminal, interpret it, possibly issue commands and repaint the screen. Command length is limited to 10 characters

2.c Other data structures

The speed, direction, headlight and reverse state of trains is saved in a data structure. Each switch state is saved as well, and if a switch is already in a given state, the command is not issued to the actual hardware. Additionally, the switch state is used to ensure trains don't derail.

2.d Automated Testing and cross-compilation

To be able to separate ARM-specific code to ISA-independent code, and run automated tests on the latter during later kernel assignments, I ensured I can cross-compile ISA-independent code to x86. In this assignment, I have compiled some code to x86-64. Furthermore, the C++ testing framework Googletest has been used to test sensor update code and my buffer implementation. The corresponding test can be found in the `test` directory.

2.e Known bugs

- Copy-pasting and backspace/arrow keys and Ŝ/Q̂ do not work
- One train can only reverse after the previous reverse command for it has finished (Note that two trains can reverse simultaneously)

3 Hashes and files

(`make` builds the project)

```
9e6d56c3fd4d5e1c03ba2cc67d813634 ./.gitignore
02d8184c8c22d269f39b92be9923df26 ./.travis.yml
86f94eb7f277f669996ded83a9b3248d ./Makefile
4b69e0c0a19d62d686342245cc62558a ./README.md
974c7d0bcfb8ee105345b31b63d0bf9d ./include/glue/myio.h
6731d07eb89cbe097f4b1b16300da727 ./include/glue/mytimer.c
d08ed777f8684bb34d62704e2ac72ec5 ./include/glue/mytimer.h
3a089045d5fb3840bbc39eae4e938b24 ./include/labenv/bwio.c
d8a8796ba64293377e3cdd2a19ad59ba ./include/labenv/bwio.h
10d303e839309e6b696dc07910e7ef77 ./include/labenv/timer.c
2c36877367232f7a3d671f8e752b8886 ./include/labenv/timer.h
d6059762a375f61e278d8622ab53be37 ./include/labenv/timer_data.h
8eac53a0774ac6937893712ecc50c7a6 ./include/ts7200.h
2af81cc9373574862f34d6cb6d62480d ./main.c
04a99fa4e57230b2b2b7a525a1105097 ./main.ld
6cba0f6a4434346f780e350bc23e0659 ./src/a0terminal.c
296bec1611cd4bfa50c59b345002e7ab ./src/a0terminal.h
a5cf584411f6db99ab9f14f2356fb673 ./src/a0trackstate.c
d31a4e75c5feb06d489be868c8c9062f ./src/a0trackstate.h
0a22f4694653bf5a9ff1fd28554840f7 ./src/buffer.c
7bdcff6618ec7ad68906c0a5615aa337 ./src/buffer.h
c975481075298f6940ac9f6ea27c6026 ./src/stdlib.c
a08e7a0ddaeb2d84e994786b81bce592 ./src/stdlib.h
af9cdf9014bec08c6df74cead3ca105d ./src/track/README.txt
7707b57cb3a6c3703c0e3da3acbef763 ./src/track/parse_track.py
1e06c8505279ed9f4d10136465c81618 ./src/track/parts_tracka
a408ef1736b356024bf0dc2bc05d98b0 ./src/track/parts_trackb
a285cacf750b6e0f676dc0379638f622 ./src/track/track_data.c
684766078c7fbf292e194923789f78d4 ./src/track/track_data.h
0c5b6215fec059d476278453036947ff ./src/track/track_node.h
50e0b1150b39a2425cb2180c33e0e57f ./src/track/tracka
dab2764f1d7f07aa454bad2ec01158b8 ./src/track/trackb
000a214c2834c915e605751e799411a7 ./start-qemu.sh
6420f56d16b729f3b1f7df4ada222bae ./test-resources/assert.c
b427c24da6970a80c258742a8ce89a6a ./test-resources/assert.h
1f344718d59006633607b1e7e7c5f90d ./test/Makefile
1d5637c2d357f516ab04839882b14d62 ./test/test_buffer.cc
6380dcccd4770acc00c20d52658c2e19 ./test/test_buffer.h
bc2c6f76374c2e9c7ee3e73475392e97 ./valgrind.sh
d41d8cd98f00b204e9800998ecf8427e ./build/.gitkeep
```

4 Questions

4.a Clock missing updates/losing time

To not miss clock updates/lose time, we have to read and send the current time at least every 100ms. After every 1000th main loop iteration, I calculate the average time per loop. During various development stages, the time has varied from 0.44-0.56ms, depending on the amount of I/O done and the machine used, and whether it had a train set connected to it. If we only had to worry about updating the clock, we would be fine. However, sensors and turnouts also need to be reprinted. In practice, the worst-case time was not higher than 1500 clock ticks (3ms)
Assuming only five of the six opportunities for sending bytes to COM2 are used (i.e. one byte every 0.1ms), we still transmit $5 * 100/0.56 > 890$ bytes per 100 ms. In one clock iteration, at worst, we have two sensor updates and one switch update (the updates of command input etc are negligible). One switch update produces 10 bytes to be sent (repaint all switch states), one sensor update 150. Even in this unlikely event, we are still at only 1/3 of the upper limit for transmission above. Thus all clock updates should be transmitted in time. In addition, we transmit the time first. In practice, since the above worst-case scenario can never occur in two consecutive clock updates (since switches are switched at most every 150 ms), the average loop time is representative for the worst case loop time.
The clock itself does not lose time/miss updates because we query it much more often than the 100ms required, and the clock itself at 508 kHz, so even faster.

4.b Train hardware sensor query time

With full output, it takes 63-64ms for all sensor bytes to be updated on screen. Previously, with less output enabled, this value was as low as 61ms. I don't know if the Märklin hardware queries all sensor modules at the same time, or if it does so at the same time. Assuming they are queried in parallel and that transmission is the bottleneck, $61 - 11 * 1000/240 = 15.167$ms (with 10 bits per byte transmitted and 11 total bytes transmitted, i.e. 1 for querying and 10 for answering) is the lower bound for the sensor query response. The round-trip time from sending the byte to expecting the first response is thus 19-20ms