GitLab Repository: https://git.uwaterloo.ca/e52wang/cs452a0

How to run

Existing directory of files: /u9/e52wang/cs452/cs452a0
Existing full pathname of image: /u9/e52wang/cs452/cs452a0/iotest.img

Upload image to raspberry pi and restart pi.

OR

git clone https://git.uwaterloo.ca/e52wang/cs452a0
cd cs452a0
git checkout *commit SHA*
make

Upload image to raspberry pi and restart pi.

Program description

Set up
- Configure and set up UART for console and Marklin line
  - Note: Corrected Marklin baud ival and fval to 1250 and 1 respectively using formula in BCM documentation
- Send go command 96 and sensor reset-mode command 192 to Marklin
- Set all switches to hard-coded default values
- Set all trains' previous speeds to 0 in case rv is the first command to the train
- Print all initial values to console including default switch values and '>' for user input line
- Set up buffers indices for two circular buffers for sending bytes to console and marklin
- Set up two regular buffers for user input and sensor data

Main loop
- If 100ms has passed since last time update, update time on screen
- Check if there are unsent bytes in the console and Marklin circular buffer; if so, push the next byte from buffer to the CR of the respective line if it's available for data
- If we have sent a sensor query and have not received all 10 bytes, check if there is a new byte in the Marklin CR; if so, run onReceiveSensorByte()
  - if it is the final (10^{th}) byte of the sensor query, process the bits in all 10 bytes and if any sensor has been triggered, update the console sensor display
  - if it is any of the previous bytes, push the byte into the sensor buffer
- Check if the program has received a byte from console; if so, process the character
  - If it is a return keystroke, attempt to execute the function stored in the input buffer and clear the input buffer as well as input display in console
    - Execute function: do character checking to see if the characters entered by the user is a valid command and if the values they entered are reasonable; if so, execute the command by sending the appropriate values to Marklin, if not, display an error message on console
    - Note: The program assumes that valid train numbers are between 0 and 99 and valid switch numbers are between 0 and 999
  - If it is a regular character, push into input buffer
    - Note: The program currently do not accept backspace or arrow keys

Implementation

- Timer
  - In kmain(), set matchValue variable to the CLO register of system timer as the starting counter value
  - In each iteration of mainloop(), read CLO register of system timer
    - Note: Assume we won't need the upper 32 bits since it is unlikely that we run the program for anywhere near 2^32/10^6 = 4295 seconds
  - If the current counter value is greater than or equal to the previous matchValue, update the console displayed timer and update matchValue to current counter value
- Polling push bytes to console or Marklin
  - For each line, implement a circular buffer to store bytes to be sent and a variable to store the last byte index that has been already been sent
  - Each time a new byte needs to be sent to console or Marklin, add it to the respective buffer
  - In each iteration of mainloop(), check if the last byte that's been sent is the last byte in the circular buffer; if not, attempt to send the byte; if succeeded, update the last byte sent variable
- Post-complete Marklin command delay
  - To indicate the end of a complete Marklin command in the circular buffer, the program pushes a byte of value of 255 into the buffer
  - Note: the program assumes that we would not need to send 255 to Marklin for an actual command; if this changes later, we would need to change this value
  - In the polling put function for Marklin, when a byte 255 is encountered in the Marklin buffer, we store the current time and do not send another Marklin byte until 150ms has passed from the stored time
- Reverse command
  - The program stores the current speed of every train when a tr command is sent
  - When a reverse command is sent, push a 0 speed command for the train, then a byte of value 255 to indicate the end of a command as described previously, followed by a byte of value 254 into the Marklin circular buffer, then a speed command of the current speed stored, finally another byte of value 255 to indicate the end of another command
  - Note: the program assumes that we would not need to send 254 to Marklin for an actual command; if this changes later, we would need to change this value
  - In the polling put function for Marklin, when a byte 254 is encountered in the Marklin buffer, we store the current time and do not send another Marklin byte until 2.5s has passed from the stored time
  - The program also ensures to not continue to send sensor poll requests during this process to avoid bytes buildup during the wait

Important Data Structure

A circular buffer of size 5000 for bytes to be sent to console

- I use a circular buffer to ensure that we don't need to continue to allocate more memory to avoid running out of space as more output to console is generated
- The large buffer size is to accommodate the number of initial setup output to console

A circular buffer of size 200 for bytes to be sent to Marklin

- I use a circular buffer to ensure that we don't need to continue to allocate more memory to avoid running out of space as more output to Marklin is generated
- The large buffer size is to accommodate the initial switch change commands to Marklin

A regular buffer of size 20 for input bytes from user
- Note: we assume the user wouldn't input in a function of more than 20 characters

A regular buffer of size 10 for sensor bytes received from Marklin

A regular buffer of size 12 to store the 12 most recent sensors triggered

Questions

How do you know that your clock does not miss updates or lose time?

I know that my clock does not miss updates or lose time since it only needs to update every 100ms but the maximum main loop cycle runtime measures only 300 clock counts or 0.3ms.

How long does the train hardware take to reply to a sensor query?

(Time is measured from when a polling request is sent to Marklin and when the final (10^{th}) byte is received)
In the first 500 sensor polls, an average of 61559 clock count of 61.559ms passes
The maximum time measured for a sensor poll is 61589 clock counts or 61.589ms
The minimum time measured for a sensor poll is 61533 clock counts of 61.533ms

To check the measured sensor query time on the console, enter command 't' and then the return key and sensor polling timing results will be printed on the console. Printed data includes the average measured polling time, the number of sensor polling that has been executed, and the max and min polling time measured.