EE 472: Spring 2013

Michael William Pang Molina

Xin Chen

Youngjun Han

Dylan Armstrong

Table of Contents

[Introduction 2](#_Toc358589435)

[Design Procedure 2](#_Toc358589436)

[Description of Project 3](#_Toc358589437)

[Description of Functions 3](#_Toc358589438)

[Description of Operating System 4](#_Toc358589439)

[Error Analysis 5](#_Toc358589440)

[Time Estimates 6](#_Toc358589441)

[Diagrams 7](#_Toc358589442)

[High-level block diagram 7](#_Toc358589443)

[TrainCom 8](#_Toc358589444)

[SwitchControl 9](#_Toc358589445)

[Current Train 10](#_Toc358589446)

[SerialCom 11](#_Toc358589447)

[Schedule 12](#_Toc358589448)

[Pseudo Code 13](#_Toc358589449)

[Work Distribution 22](#_Toc358589450)

# Introduction

The focus of the final project of this course (projects 4 and projects 5 combined) was to continue to develop and refine the existing train control system that we had begun to develop in projects 2 and 3. The following list specifies the major revisions and additions for this project:

* Replace linked list scheduler with FreeRTOS’s scheduler.
* Implement a function counter to read the frequency of an external square wave. This function was to represent the passenger load of the train.
* Allow up to two trains (previously one train) to be present at the intersection at any given time.
* Replace the existing ‘gridlock’ with the possibility of an actual train blocking the intersection.
* Implement a web-based interface for the train control system.

Aside from the five major revisions listed above, there were several other pieces to this project, including a visual alarm indicating if a train has been waiting for a long time (6 or more cars of wait time) and revising the OLED to reflect the second train. The entire base program runs successfully as specified in the project 4 and project 5 reports.

In addition the hard requirements discussed above, we attempted to implement two extra credit additions to our project. First, we attempted to enhance the web-based interface, as outlined in the project 5 specification. Second, we attempted to implement analog voltage measurement using the ADC that is included on the Stellaris board. Two of our teammates (Leo Chen and Youngjun Han) put significant effort into these extra credit additions, however due to time constraints we were not able to fully implement either option. In the final submitted code, the work that was done towards these two additions is included, but it does not yet run fully in our project.

# Design Procedure

Similarly to our approach for the earlier projects, the design for this project made use of many of the design steps reccomended. The basic design process was as follows:

1. Draw all UML diagrams, starting with highest level diagrams
2. Understand the functionality of each task and revise UML diagrams as needed
3. Write pseudo-code for each task, separately
4. Review program logic and control scheme
5. Write code and work through all compile time errors
6. Test and debug code
7. Polish code

While our general design approach, outlined above, was similar to our approach for the earlier projects, the actual implementation of this process was notably different than for either of the before projects. First, since we didn’t really have to write new code to use FreeRTOS, we didn’t follow the design approach outlined above for this part. Instead, we just all began trying to understand what we were given (which was a big task by itself). Once we thought we knew enough about FreeRTOS, we began porting our project to demo FreeRTOS IAR project. After many hours of correcting compile time errors (most of which were due to conflicting / repeated header files), we just played with the code until it worked.

For the rest of the project revisions and additions, we followed the more traditional design approach. We (kind of) split the major task additions between members. Several people worked on the function generator and others worked on amending the logic for adding the second train. Once we had these two tasks working, we all worked together – intermittently – to complete and integrate all of the code. Finally, for the web-based interface, we all spent time wading through the FreeRTOS ethernet example code, then appended the example code to work for our project.

# Description of Project

## Description of Functions

**TrainCom():** This function is responsible for generating new trains. New trains are only allowed to be generated when there are fewer than two trains already present; this is verified by checking the numTrainsPresent variable, whose value is exactly equal to the number of trains that are present. When a train is generated, its size, payload, and to direction are determined, and this information is printed to the OLED screen.

**SwitchControl():** This function checks to see if the moving train has completely passed through the intersection yet. If the train has not yet passed, switchControl checks to see if there is a waiting train, and if there is, switchControl checks if the train has been waiting for 6 or more cars. If the train has completely passed (traversalTime = zero), then all the information for the waiting train is moved to the first train position, and the OLED screen is updated accordingly. This is done because the train1 must always hold the moving train’s information, and train2 is always the waiting train.

**CurrentTrain():** This function is responsible for flashing the OLED screen and for handling the audible alarms that indicate which direction the train is moving. This function only runs when there is a train currently moving through the intersection, as determined by SwitchControl().

**SerialCom():** This function is responsible for printing information – via the hardware serial port interface – to hyperterm on the local PC. This is accomplished simply by repeated calls to the UARTSend function.

**Schedule():** This function runs every 0.5 seconds (as dictated by using the FreeRTOS delay function). It keeps track of the global count and allows switchControl to run if at least one train is present.

**Main():** Main is responsible for running the setup functions, for initializing the data struct’s values, for creating all tasks using the FreeRTOS xTaskCreate function, and for starting the FreeRTOS task scheduler.

**Generate\_train\_stats():** This function is contained in the httpd-cgi.c file. This function generates the string that contains the train stats, which is inserted to the appropriate spot in the website (the tag ‘train-stats’ in the HTML file indicates where this string is inserted).

**randomInterger():** This function generates a (pseudo) random integer. It is used by trainCom() to generate the trainSize and the train ‘to direction’. Note that this function was not written by our group, but rather was taken from the random number example provided earlier in the quarter.

**frequencyCounter():** This function is called each time a train is generated. It grabs the value of the frequency (which is counted in the FrequencyIntGPIOa ISR) and calculates the payload value according to the given formula.

**FrequencyIntGPIOa():** This is the ISR for the frequency input pin. It increments the frequencyCount variable each time it is entered.

**IntGPIOe():** This is the button ISR. It sets variables according to which button is pressed.

**UARTSend():** Sends a string of unsigned char’s through the UART interface.

**buildString():** This function takes two pointers to arrays of characters. The contents of the second array are copied to the first array.

**pvrSetupHardware():** This function handles all of the hardware setup, including the system clock setup, enabling and configuring all peripherals, including the speaker, any GPIO pins that are used (for buttons and function counter pin), and the UART hardware.

## Description of Operating System

**TrainCom():** This function is used in the operating system and is created and implemented in this operating systems task scheduler where it executes it when the right conditions are met.

**SwitchControl():** This function is used in the operating system and is created and implemented in this operating systems task scheduler where it executes it when the right conditions are met.

**CurrentTrain():** This function is used in the operating system and is created and implemented in this operating systems task scheduler where it executes it when the right conditions are met.

**SerialCom():** This function is used in the operating system and is created and implemented in this operating systems task scheduler where it executes it when the right conditions are met.

**Schedule():** This function uses the operating system’s delay task to create a timer that counts by every 0.5 seconds.

**Main():** Main is creates all tasks using the FreeRTOS xTaskCreate function and starts the FreeRTOS task scheduler.

**Generate\_train\_stats():** This uses the operating system’s web server capabilities to write information from the system to an IP address.

The behavior of the system is supposed to act as a train manager. We used four buttons to create the starting point of a train and use a random direction that the train is headed towards. This random direction can either be north, east, south or west, and each have its unique way to warn the user a train is passing through the intersection. Also a maximum of two trains can be present in the intersection at once. The first one will pass through the intersection while the other will wait for the first one then continue from there. On an OLED we display the information of each train (to direction, from direction, size, wait time of the second train, passenger load, and global count). The passenger load is determined by the frequency of an inputted square signal into our system. We also output all the same information via web server and serial communication.

Our code runs and does exactly what we expect it to do, except for the extra credit additions, which we ran out of time to complete.

# Error Analysis

Our core project design works without issue (though the two extra credit additions that we attempted are not fully functional). Regarding the core project, one debugging issue that we had was implementing the buttons to chose the train from direction. In project 3 and 4, this was not an issue because once one train was completed, we did not allow trainCom to run again until the train had completely passed. However, with the addition of the second train in project 5, our logic for trainCom changed. Now, when there was only one train created, trainCom was still allowed to run, because we want to allow a second train to be created. The issue was that on a single button press, the ISR for the buttons would be entered multiple times. We believe that this can be attributed to the button signal bouncing around.

Since the trainCom flag was set to 1 in the ISR, this meant that trainCom would run multiple times for each button press. As stated before, this bug did not surface because once one train was created, trainCom simply could not run again. But now, when there were initially no trains present, and the button was pressed, occasionally two trains would be created since the ISR was entered multiple times due to button noise. To get around this issue, we created a time based flag that also must be present to enter trainCom. So now, to enter trainCom, both a button flag and the timer based flag must be true. The timer based flag can only be reset every two cycles of schedule (every 1 second), therefore trainCom cannot run more than once per second. This completely eliminated the button noise issue, and our code now runs properly.

# Time Estimates

|  |  |
| --- | --- |
| Subject | Time spent |
| Design | 4 hours |
| Coding | 8 hours |
| Test / Debug | 25 hours |
| Documentation | 5 hours |

# Diagrams

\*\*\*(Any of the above which were used in designing your code should be included. Combined, these elements should:

- Form a legible description, including both high-level structure and details, of your project

- clearly describe all functions of your project

- make it possible to navigate your source code if necessary)\*\*\*

## High-level block diagram



## TrainCom



## SwitchControl



## Current Train



## SerialCom



## Schedule



# Pseudo Code

**header.h**

#include header files for drivers libraries needed in this project;

declare error routine function prototype;

define new variable type bool;

declare function prototypes for the entire project;

define struct TCB;

create a struct for holding task trainCom’s data as trainComData;

create a struct for holding task switchControl’s data as switchControlData;

create a struct for holding task schedule’s data as scheduleData;

create a struct for holding task serialCom’s data as serialComData;

create a struct for holding task currentTrain’s data as currentTrainData;

//-------------------------------------------------------------------------//

**constantDefinitions.h**

#define statements of constant definitions for the entire project;

//-------------------------------------------------------------------------//

**main.c**

set mainINCLUDE\_WEB\_SERVER flag to 1 to include the web server

#include "header.c" and "constantDefinition.h"

#include FREERTOS library and hardware library header files

#include hardware library for ADC

#define constants for timer, uIP task, sprintf, task priorities, OLED queue size, and buffer size

extern vuIP\_Task();

static void prvSetupHardware();

extern void vSetupHighFrequencyTimer();

void vApplicationStackOverflowHook();

void vApplicationTickHook();

unsigned char timerState; // create a flag for timer interrupt

unsigned char State0 = 0; // create flags for button interrupt

unsigned char State1 = 0;

unsigned char State2 = 0;

unsigned char State3 = 0;

bool north = FALSE; // initialize global variables

bool west = FALSE;

bool east = FASLE;

bool south = FALSE;

bool trainPresent = FALSE

unsigned int trainSize = FALSE;

unsigned int traversalTime = 0;

unsigned int gridLockDelay = 0;

bool gridLock = FALSE;

unsigned int globalCount = 0;

unsigned int trainComFlag = 0;

unsigned int switchControlFlag = 0;

unsigned int serialComFlag = 0;

unsigned int currentTrainFlag = 0;

char gridLockMessage[]; // initialize global strings for gridlock alarm

char trainDirectionMessage[]; // and train info

char trainSizeMessage[]

char trainFromMessage[];

int main(void)

{

prvSetupHardware();

initialize trainCom TCB;

initialize serialCom TCB;

initialize switchControl TCB;

initialize currentTrain TCB;

initialize trainComData TCB; // initialize all the fields inside the structs

initialize serialComData TCB;

initialize switchControlData TCB;

initialize currentTrainData TCB;

initialize scheduleData TCB;

create trainCom semaphore;

create serialCom semaphore;

create currentTrain semaphore;

create switchControl semaphore;

create schedule semaphore;

take trainCom semaphore;

take serialCom semaphore;

take currentTrain semaphore;

take switchControl semaphore;

take schedule semaphore;

display the fixed text on OLED;

create vuIP\_task with priority of mainCHECK\_TASK\_PRIORITY - 1;

create tasks using xTaskCreate();

set high frequency interrupt to measure interrupt jitter time;

start the scheduler;

return 0;

}

void prvSetupHardware(void)

{

If (DEVICE\_IS\_REVA2)

turn the LDO voltage up to 2.75V

set the clocking to run directly from the crystal;

initialize the OLED display;

clear the default ISR handler and install IntTimer0 as the handler; // set up for timer

enable Timer 0;

configure Timer 0 and set the timebase to 0.5 second;

enable interrupts for Timer0 and activate it;

clear the default ISR handler and install IntGPIOe as the handler; // set up for buttons

enable GPIO port E, set pin 0,1,2,3 as inputs;

activate the pull-up on GPIO port E

Configure GPIO port E as triggering on falling edges

enable interrupts for GPIO port E

intEnable(INT\_GPIOE);

set PWM Divide Ratio to 1; // set up for speaker

set Device: PWM0 Enabled;

set GPIO Port: G Enabled;

tell Port G, Pin 1, to take input from PWM 0;

set a SPEAKER\_FREQUENCY Hz frequency as u1Period;

configure PWM0 in up-down count mode, no sync to clock;

set u1Period (SPEAKER\_FREQUENCY) as the period of PWM0;

set PWM0, output 1 to a duty cycle of DUTY\_CYCLE;

activate PWM0;

// set up for serialCom

clear the default ISR handler and install UARTIntHandler as the handler;

enable the peripherals;

enable processor interrupts;

set GPIO A0 and A1 as UART pins;

configure the UART for 115,200, 8-N-1 operation;

enable the UART interrupt;

// set up for frequency measuring pin

enable GPIO port B and set GPIO port B pin 0 to be the input;

enable the pull up for GPIO port B pin 0;

set as falling edge interrupt;

enable GPIOB pin 0 interrupt;

// set up for ADC

enable ADC port on the board;

set ADC pin 1 as input;

configure the pull down;

disable sample sequence 3;

configure sample sequence 3 to triggered by processor and has priority 0;

configure sample sequence step 0 in sequence 3 to chanel 0 and enable interrupt

enable sample sequence 3;

clear the interrupt status flag;

trigger sample sequence 3;

get sample value;

}

void frequencyCounter() // for passenger load simulation

{

frequency = frequency\*2;

if (frequency less than 1000)

payload = 0;

else if (frequency greater than 2000)

payload = 3000;

else calculate payload;

reset frequencyCount;

}

void vApplicationStackOverflowHook(void)

{

while(1) loop if the program reaches here;

}

void buildString(arrayPointer, newString array)

{

while (not the end of the string)

arraypointer value = new string value

}

void UARTIntHandler(void) // ISR for serialCom

{

unsigned long ulStatus;

Get the interrrupt status;

Clear the asserted interrupts;

while(UARTCharsAvail(UART0\_BASE))

read the next character from the UART and write it back to the UART;

}

voidUARTsend(const unsigned char \*pucBuffer, unsigned long ulCount)

{

while(ulCount--)

write the next character to the UART;

}

void IntGPIOe(void) // ISR for buttons

{

clear the interrupt to avoid continously looping;

Set State0,1,2, or 3 accordingly when a button is pressed;

// trigger button hardware interrupt

flip the polarity of the results above;

set trainComFlag = 1;

// when a button is pressed, we need to add trainCom to generate a train

give trainCom semaphore;

}

// ISR for analog square wave input

void FrequencyIntGPIOa(void){

Clear the asserted interrupt;

increment frequencyCount;

}

void schedule(void\* data)

{

recast the void pointer to schedule data pointer type

local varialbe int localTime and int digit;

while(1) {

increment global count;

reset digit = 0;

set localtime to current globalCount;

while(localtime is not zero) { // this part of the code display global count on screen

display local time from the lowest decimal place up to the highest;

}

print globalCount on screen;

if(number of train present is greater than 0){

give switchControl semaphore;

}

set trainCom flag every two clock cycle; // for debouncing

give serialCom semaphore;

call frequencyCounter() for measuring the passenger load;

call vTaskDelay() to delay 500 ms; // half second

}

}

//-------------------------------------------------------------------------//

**trainCom.c**

#include “header.h” and “constantDefinitions.h”;

#include "FreeRTOS.h" and "semphr.h"

extern trainCom semaphore

extern serialCom semaphore

extern payload and trainComFlag

void trainCom(void\* data)

{

re-cast the passed in void pointer to trainComData struct pointer type;

local variables int direction = 0; int TrainSize = 0;

while(1)

if trainCom semaphore is taken

if trainCom Flag is 1

set previousGlobalCount to current globalCount

if numTrainPresentP is less than 2

if (there is no train right now)

If (State0Ptr == 1) // up button pressed

Call builString to build train from north message;

Reset State0Ptr = 0; // reset interrupt

Else if (State1Ptr == 1) // down button pressed;

Call buildString to build train from south message;

Reset State1Ptr = 0;

Else if (State2Ptr ==1) // left button pressed;

Call buildString to build train to east message;

Reset State2Ptr = 0;

Else if (State3Ptr ==1) // right button pressed;

Call buildString to build train to west message;

Reset State3Ptr = 0;

TrainSize = randomInteger(2,9); // generate train size

Update trainSizeP = TrainSize;

Direction = randomInteger(0,3); // generate train to direction

If (direction == WEST)

westP = TRUE;

call buildString to build train to west message;

if (direction == NORTH)

northP = TRUE;

call buildString to build train to north message;

if (direction == EAST)

eastP = TRUE;

call buildString to build train to east message;

if (direction == SOUTH)

southP = TRUE;

call buildString to build train to south message;

call buildString to build train size message;

write TrainSize into trainSizeMessageP;

if (there is 1 train)

// create train 2

repeat the steps above to create another train;

print train 1 and train 2 information on the screen;

increment train present number;

give serialCom semaphore;

give switchControl semaphore;

reset trainComFlag;

}

//-------------------------------------------------------------------------//

**switchControl.c**

#include “header.h” && “constantDefinitions.h”

#include "FreeRTOS.h" and "semphr.h"

extern currenTrain semaphore

extern serialCom semaphore

extern switchControl semaphore

void switchControl(void\* data)

{

Re-cast the passed in void pointer to switchControlData struct pointer type

Set gridLockChecked = TRUE;

Int gridLockNum; // this random number is to determine whether there is a gridlock

if switchControl semaphore is taken

if there is a current train passing

decrement train traversal time;

if there are two train

create wait time message

if waitTime exceeds 71 count

display long wait message;

increment wait time;

give currentTrain semaphore;

else

decrement number of train present;

set wait time to zero;

if (there is only one train)

clear wait time string;

set all train1's information to train 2's information;

clear train 2 info;

display train 1 info on the screen;

display train 2 info on the screen;

else // no train waiting

reset train 1 and train 2 info;

display the empty info on the screen;

turn off the sound;

give serialCom semaphore;

}

//-------------------------------------------------------------------------//

**currentTrain.c**

#include "header.h" and "constantDefinitions.h"

#include "FreeRTOS.h" and "semphr.h"

extern currentTrain's semaphore;

void eastTrain(void); // helper function prototypes

void westTrain(void);

void northTrain(void);

void southTrain(void);

static char\* trainDirectionMesssage; // local variable pointers

static char\* trainSizeMessage; // three pointers pointing to char array

static char\* trainFrom; // for display message on OLED

static unsigned int remainingtraversalTime;

void currentTrain(void\* data) {

re-cast passed in void pointer to currentTrainData pointer type

trainDirectionMessage = trainDirectionMessageP;

trainSizeMessage = trainSizeMessageP

trainFrom = trainFromP;

remainningTraversalTime = traversalTimeP; // define local variable and variable pointers

if( currentTrain's semaphore is taken) {

if (there is a train passing)

if (train is from EAST) {

creates an integer array for west train sound pattern

if (eastSoundArray[remainingTraversalTime%26] ==1) // go through the array repeatedly

turn on the sound

else

turn off the sound

if (remainingTraversalTime %8 < 4)

// this algorithm gives a flashing rate of 2 sec ON 2 sec OFF

{

print trainDirectionMessage;

print trainSizeMessage;

print trainFrom;

} else

clear the screen;

} else if (the train is from west) {

creates an integer array for west train sound pattern

if (westSoundArray[remianingTraversalTime % 14] ==1)

// go through the array repeatedly

turn on the sound;

else

turn off the sound;

if (remainingTraversalTime % 4 <2) // a flashing rate of 1 sec ON 1 sec OFF

{

print trainDirectionMessage;

print trainSizeMessage;

print trainFrom;

} else

clear the screen;

} else if (the train is from south) {

creates an integer array for south train sound apttern

if (southSoundArray[remainingTraversalTime % 24 ==1) // traverse the array element

turn on the sound;

else

turn off the sound;

if (remainingTraversalTime % 4 < 2)

// a flashing rate of 1 sec ON 1 sec OFF

print trainDirectionMessage && trainSizeMessage && trainFromMessage;

else

clear the screen;

} else if (the train is from north) {

creates an integer array for north train sound pattern

if (northSoundArray[remainingTraversalTime % 20] ==1) {

// traverse the array element repeatedly

turn on the sound;

} else

turn off the sound;

if(remainingTraversalTime % 6 <3)

// a flashing rate of 1.5 sec ON ,1.5 sec OFF

print trainDirectionMessage && trainSizeMessage && trainFromMessage;

else

clear the screen;

}

}

}

//-------------------------------------------------------------------------//

**serialCom**

#include "header.h" and "constantDefinitions.h"

#include "FreeRTOS.h" and "semphr.h"

void serialCom(void\* data)

{

recast the void pointer to serialComData type

while(1) {

call UARTSend function to clear the screen;

call UARTSend function to display trainSizeMessage && trainTitle

&& trainFromMessage && trainToMessage && trainLoad

call UARTSend function to display global count;

}

}

# Work Distribution

|  |  |
| --- | --- |
| Task | Names |
| Design | Michael, Leo, Young, Dylan |
| UML Diagrams | Michael |
| pseudocode | Leo |
| trainCom | Michael |
| switchControl | Dylan |
| North/east/west train | Dylan, Michael |
| schedule | Young |
| Test / Debug | Michael, Leo, Young, Dylan |
| Documentation | Michael, Leo, Young, Dylan |