**Project 2 Report**

EE 472: Spring 2013

Dylan Armstrong

Xin Chen

Youngjun Han

Michael Molina

Table of Contents

[Design Procedure 2](#_Toc354949751)

[Time Estimates 3](#_Toc354949752)

[Empirically Measured Task Times 3](#_Toc354949753)

[Block Diagrams 4](#_Toc354949754)

[High-level block diagram 4](#_Toc354949755)

[TrainCom 5](#_Toc354949756)

[SwitchControl 6](#_Toc354949757)

[North/East/West Train 7](#_Toc354949758)

[Schedule 8](#_Toc354949759)

[Pseudo Code 9](#_Toc354949760)

[Header file 9](#_Toc354949761)

[void main() 9](#_Toc354949762)

[void traincom(void\* data) 9](#_Toc354949763)

[void switchControl(void\* data) 10](#_Toc354949764)

[void northTrain(void\* data) 11](#_Toc354949765)

[void eastTrain(void\* data) 12](#_Toc354949766)

[void westTrain(void\* data) 12](#_Toc354949767)

[void schedule(void\* data) 13](#_Toc354949768)

[Work Distribution 14](#_Toc354949769)

# Design Procedure

After carefully reading through the project specification, the first task we got our hands on is to draw the high level block diagram. This diagram is a flow chart that generalizes how six major tasks constitute a loop to achieve the program’s functionalities (as shown in page 4). Then for each of the six tasks, we drew an activity diagram that clearly illustrates the internal logics that specifies what the program should do in certain conditions (as shown in page 5-8). With these six activity diagrams, each member in the group was assigned to write the pseudo code for individual task before we did the actual coding on the computer.

The first task in the loop is trainCom. This task’s duty is to generate a train as long as the two fundamental information, direction and size. This is all trainCom has to do. Namely, in any other circumstances, this task will simply does nothing and pass to the next task, which is switchControl.

The second task, switchControl, focuses on dealing with the grid lock problem. The task generates a random number to determine if there is a gridlock only in the case when there is not any current train that is passing or any ongoing grid lock. If there is an ongoing gridlock, switchControl issues a warning that flashes on the screen. If currently there is a train passing through the intersection, switchControl will do nothing and pass to the next task.

The next three tasks in the loop are northTrain, westTrain, and eastTrain respectively. The logic and internal structure in each of these three tasks are quite similar. Each of these three tasks will only send a signal to the screen and sound in a specific pattern when the currently passing train is going towards the direction it represents. Otherwise, the three tasks do nothing but pass to the next task.

The final task, schedule, increases the global variable globalCount by 1. This task is important because it is the time reference for all the tasks in the program.

# Testing Process

To test and debug the train system we used the debugger tool in the IAR Embedded Workbench. The main error we had was in the schedule task where stuff would display at the wrong time or would remain shut off when we did not design it that way. To fix these errors we changed some conditions for if statements within the switch control function by stepping through the code line by line keeping track of all the global variables and making sure they behaved how we designed them for each situation.

Then to test the performance of each of our tasks we used an oscilloscope to measure the time each tasks takes. For different situations some tasks took longer or shorter so we measured the minimum and maximum time each took and recorded it in a table. To do this we used pin A7 to turn on before a task and off after the same task. Finally measuring the last task, schedule, we found that our system clock was approximately 506 ms.

# Time Estimates

|  |  |
| --- | --- |
| Subject | Time spent |
| Design | 6 hours |
| Coding | 2 hours |
| Test / Debug | 8 hours |
| Documentation | 2 hours |

# Empirically Measured Task Times

|  |  |  |
| --- | --- | --- |
| Task | Minimum Time | Maximum Time |
| trainCom | 6 μs | 100 μs |
| switchControl | 14.4 μs | 2.99 ms |
| northTrain | 22.6 μs | 4.98 ms |
| westTrain | 20.7 μs | 4.9 ms |
| eastTrain | 24.5 μs | 4.86 ms |
| Schedule | 506 ms | 506 ms |

# Block Diagrams

## High-level block diagram



## TrainCom



## SwitchControl



## North/East/West Train



## Schedule



# Pseudo Code

## Header file

Error giving statement

Declare Boolean type

Pre-calling the functions

Construct TCB

Construct data set for trainCom

Construct data set for switchControl

Construct data set for northTrain

Construct data set for eastTrain

Construct dataset for westTrain

Construct dataset for schedule

## void main()

Including header file

Initiate global variables

Initiate sound generator environment

Initiate OLED display environment

Assign functions and its data set pointers into TCB in order

void main()

while(true) {

loop through array of tasks

}

## void traincom(void\* data)

Extract data from TCB;

Initialize local variable for generated direction;

Initialize local variable for generated trainsSize;

If (no train is present && no gridlock){

Generate random number for trainSize;

Generate random number for direction;

Set global trainSize to localTrainSize;

Set global direction variable to correct direction; //0=west, 1=north, 2=east

Set global variable trainPresent to TRUE;

}

## void switchControl(void\* data)

Extract data from TCB;

Set gridlockChecked to TRUE;

Initialize local variable gridlockNum; //this will be the randomly generated number

Initialize the array to print trainSize to OLED;

Initialize local trainSize variable;

If (gridLockDelay = 0 && traversalTime = 0) {

// Then there is not currently a gridlock or a train passing through, so need //to generate a random number.

gridlockNum = random number between (-2,2)

if (gridLockNum is negative) //Then this a gridlock was generated{

Set global gridlock = TRUE;

Set global trainPresent = FALSE;

Calculate gridLockDelay time in terms of counts; //1 count = 0.5 seconds

Print “Gridlock Alarm” to OLED

Print correct train direction to OLED

Print current train size to OLED

} else //then we are letting the train pass {

calculate traversal time in terms of counts; //1 count = 0.5 seconds

Set global traversalTime to calculate time;

Set switchCount to traversalTIme; // so switchControl knows when the train has passed

}

} else if (gridlock = TRUE) // then we are currently in gridlock {

decrement gridlockDelay by 1;

if (gridlockDelay % 2 = 0) {

//gridlockDelay % 2 will always be either 0 or 1, so this alternates every loop //through

Print “gridlock alarm”;

} else {

Clear gridlock alarm message;

}

if (gridLockDelay = 0) {

// Then the gridlock has passed

Set global gridlock = FALSE;

Set global trainPresent = TRUE;

Clear “gridlock alarm” message;

} else { //since there is no gridlock and traversalTime != 0, there is currently a train passing

decrement switchCount by 1;

if (switchCount = 0) { //then the train has passed completely

Set global trainPresent = FALSE;

Set trainSize = 0;

Set traversalTime = 0;

Set all directions = FALSE;

Clear train information from screen;

Turn off sound;

}

}

}

## void northTrain(void\* data)

Extract data from TCB

Initialize soundArray //example: soundArray = {1,1,1,1,0,0,1,1,0,0}

Initialize trainSizeArray //this is used to print train size to OLED

Initialize localTrainSize

If(north = true && trainPresent = true && northCount <= traversalTime)

{

If( soundArray[westCountP % size of soundArray] == 1) /

//sound array holds a stream of 0’s and 1’s, which each index corresponding //to a single loop through the task. So for a long blast (2 seconds) there //would be four 1’s in a row, for a short blast (1 second) there are two 1’s in //row.

Play Sound

else{

don’t play sound

}

If(northCount % 4 < 2)

}

localTrainSize = trainSize (from TCB)

trainSizeArray[end] = localTrainSize + 48

display Direction

display trainSize

else {

Erase direction

Erase trainSize

}

//west count starts at 0 and counts up by one each loop through, until the train has passed

northCount++

} else {

northCount = 0 //reset northCount

}

## void eastTrain(void\* data)

Extract data from TCB

Initialize soundArray //example: soundArray = {1,1,1,1,0,0,1,1,0,0}

Initialize trainSizeArray //this is used to print train size to OLED

Initialize localTrainSize

If(east == true && trainPresent == true && eastCount <= traversalTime)

{

If( soundArray[westCountP % size of soundArray] == 1)

//sound array holds a stream of 0’s and 1’s, which each index corresponding to a single loop through the task. So for a long blast (2 seconds) there would be four 1’s in a row, for a short blast (1 second) there are two 1’s in row.

Play Sound

else{

don’t play sound

}

If(eastCount % 4 < 2)

localTrainSize = trainSize (from TCB)

trainSizeArray[end] = localTrainSize + 48

display Direction

display trainSize

else {

Erase direction

Erase trainSize

}

//east count starts at 0 and counts up by one each loop through, until the train has //passed

eastCount++

} else{

eastCount = 0 //reset eastCount

}

## void westTrain(void\* data)

Extract data from TCB

Initialize soundArray //example: soundArray = {1,1,1,1,0,0,1,1,0,0}

Initialize trainSizeArray //this is used to print train size to OLED

Initialize localTrainSize

If( west == true && trainPresent == true && westCount <= traversalTime)

{

If( soundArray[westCountP % size of soundArray] == 1)

//sound array holds a stream of 0’s and 1’s, which each index corresponding to a single loop through the task. So for a long blast (2 seconds) there would be four 1’s in a row, for a short blast (1 second) there are two 1’s in row.

Play Sound

else{

don’t play sound

}

If(westCount % 4 < 2)

localTrainSize = trainSize (from TCB)

trainSizeArray[end] = localTrainSize + 48

display Direction

display trainSize

else {

Erase direction

Erase trainSize

}

//west count starts at 0 and counts up by one each loop through, until the train has passed

westCount++

} else{

westCount = 0 //reset westCount

}

## void schedule(void\* data)

Extract data from TCB

int delay = 0

for(delay = (500ms); delay > 0; delay --){

}

globalCount++

# Work Distribution

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