**Learning the Development Environment - The Last Step**

**EE 472 Lab 4&5**

**University of Washington**

**Department of Electrical Engineering**

**Xindi Liu, Xinyi Chang, Xuanlin Zhu**

**August 23, 2014**

**TABLE OF CONTENTS**

**ABSTRACT--------------------------------------------------------------------------------------------3**

**INTRODUCTION------------------------------------------------------------------------------------3**

**DISCUSSION OF THE LAB----------------------------------------------------------------------3**

**PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS------------35**

**ANALYSIS OF ERRORS----------------------------------------------------------------------- 40**

**ANALYSIS OF POTENTIAL NOT WORKING PROBLEM---------------------------41**

**TEST PLAN----------------------------------------------------------------------------------------41**

**TEST SPECIFICATION------------------------------------------------------------------------41**

**TEST CASES--------------------------------------------------------------------------------------43**

**SUMMARY----------------------------------------------------------------------------------------45**

**CONCLUSION-----------------------------------------------------------------------------------45**

**ESTIMATE HOURS PER PERSON--------------------------------------------------------45**

**APPENDIX---------------------------------------------------------------------------------------46**

**ABSTRACT**

This lab is last step of developing a smart train system with IAR embedded workbench and C language. For the last two projects, we moved our tasks to a real-time operating system (FreeRTOS), used analog digital converter (ADC), and simple web server. In this project, we identify the improvement, code in C language, debug and test the system. This report includes design specification, software implementation, discussion of the results, and analysis of any errors during the project and test of the system. We test the system by testing part of the system and adding more functions and see if the output meets the spec. And our system works as described in the spec.

**INTRODUCTION**

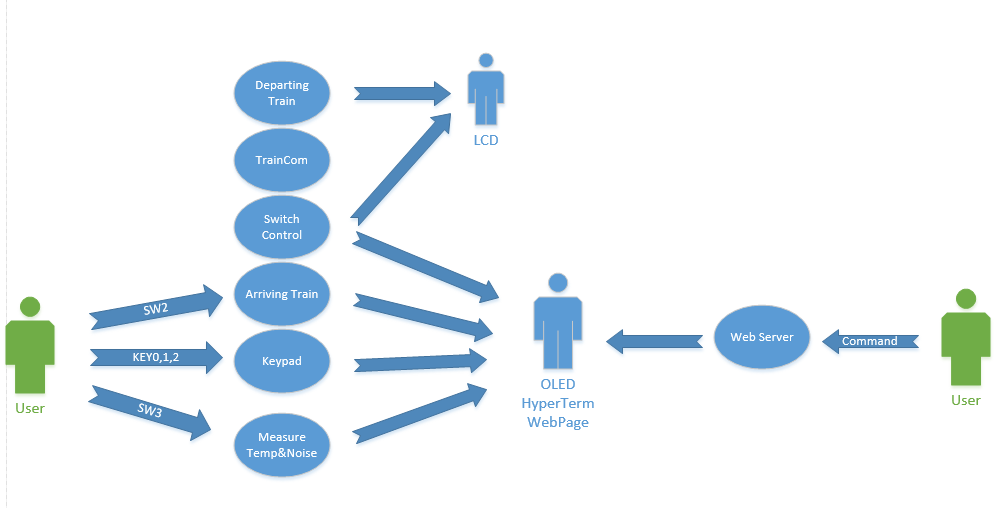
This lab is the last step to develop a smart train system. For this train system, except for the features we already had in the previous system, we used real-time operating system (FreeRTOS), analog digital converter, remote communications system, simple web server, handler, and network interface. In this lab, we designed, developed and integrated software and hardware components following the lab specifications. The system is composed with 13 tasks and we continued to build the program with pointers in C language. And we kept working with IAR Embedded Workbench tool to build and run program with EKI-LM3S8962 chip.

**DESIGN SPECIFICATION**

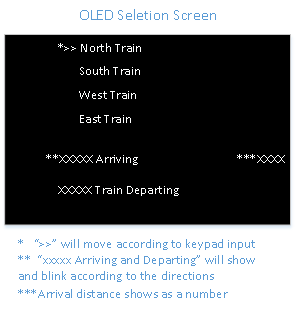
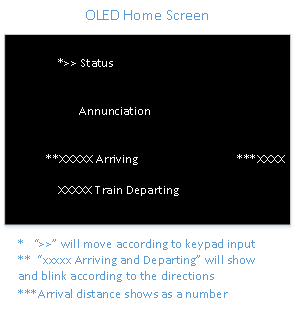
This project is the last step of developing a smart train system when the train encounter an intersection.

Input includes SW2 on the Stellaris board as the source of the interrupt to increase the number of coming train SW3 on the Stellaris board as the source of the interrupt to measure the wheel temperature, a keypad to control OLED display, a series of random number to generate the train with different arrival time, arriving direction, size, departing direction, time of gridlock, and success of burst. Also we added inputs include a sine wave range from 0 to 3V with frequency from 35 Hz to 3.75 kHz, square wave according to its remaining distance and voltage represent to wheel temperature.

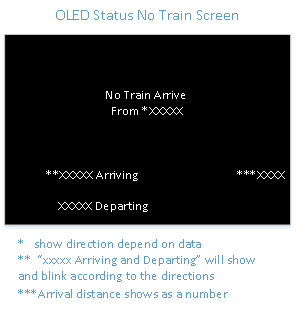
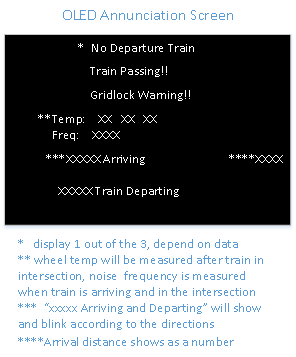
Output includes a LCD display, an OLED display, Hyperterm, webpage and blast utilizing the PWM peripheral. The LCD display shows whether the burst succeed and thermonuclear when encounter gridlock. If train is departing, it shows number of train present, train direction, intersection available, train size and traversal time. The OLED display has two mode, train status mode and annunciation mode. At train status mode, the OLED display should display “North Train”, “South Train”, “East Train”, and “West Train”, For each selected train, it should display its identification, amount of train arriving, whether the train is at intersection, train arriving direction, train departing direction, four lock status, train size, and its estimated traversal time. At annunciation mode, display alarm information and wheel temperature. Web page also display the information of current train. LCD, OLED display and Hyperterm and web page should appear as illustrated in the following front panel diagram.



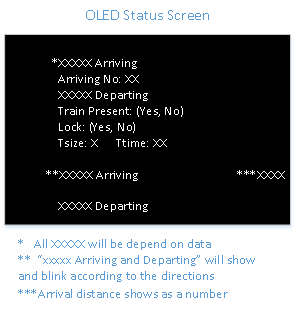
**Figure 1.1 User case**



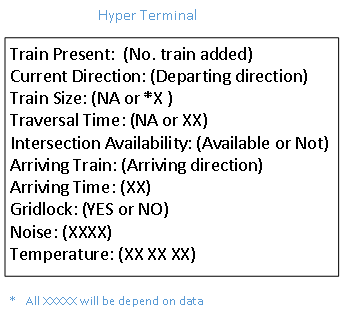
**Figure 1.2.1 OLED Home Screen Figure 1.2.2 OLED Train status selection Screen**



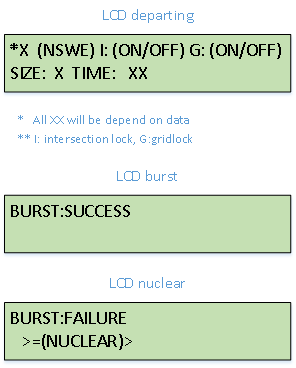
**Figure 1.2.3 OLED Annunciation Screen Figure 1.2.4 OLED train status when no train**



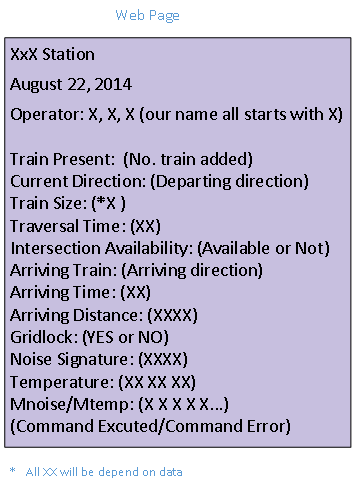
**Figure 1.2.5 OLED train status**



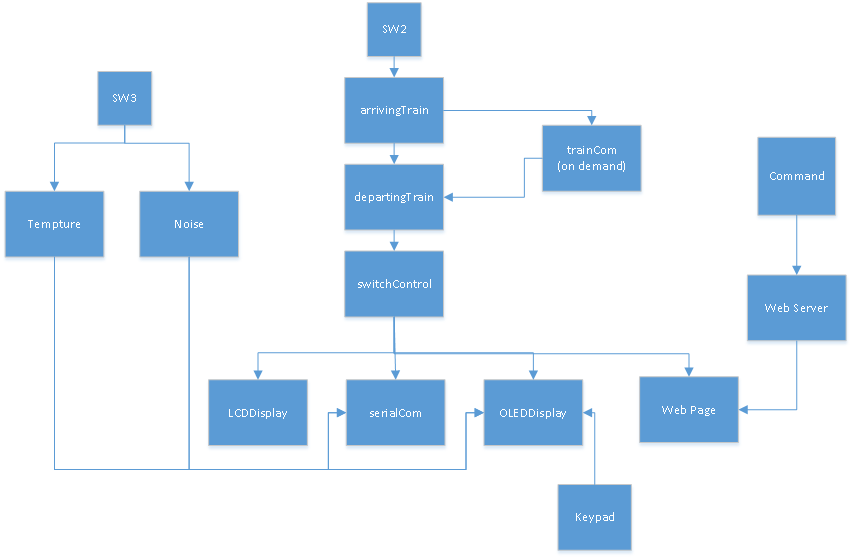
**Figure 1.2.6 Hyperterm display**



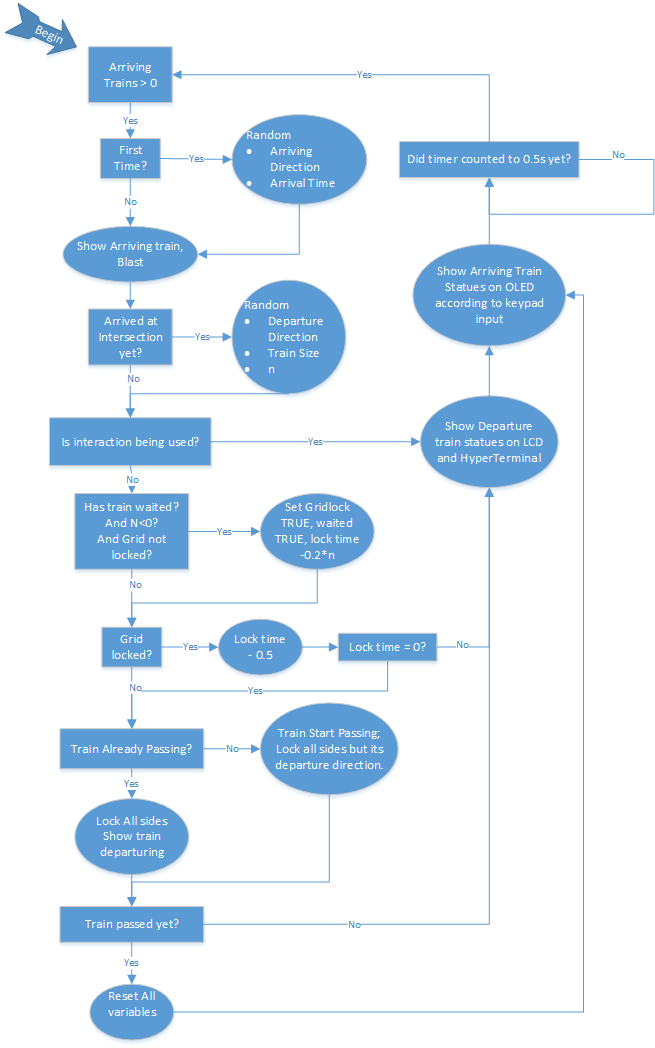
**Figure 1.2.7 LCD display**



**Figure 1.2.8 Web Page display**



**Figure 1.3 Block diagram**



**Figure 1.4 Control and Data Flow Diagram**

**SOFTWARE IMPLEMENTATION**

We are asked to use FreeRTOS to manage schedule, so we used xQueueCreate to create the tasks and schedule them according to its priority. The priority is based on the order of Lab 3 linklist. We run task arrvingTrain first, so we make this priority as 9; we run lcdDisplay and oledDisplay and the new tasks last, so we make these priority as 1. Stack depth is decided with command, usTaskGetStackHighWaterMark(). Then we use vTaskStartScheduler to run these tasks as we scheduled. When the tasks are not needed or need to be added back, we suspend and resume the task in arrvingTrain since it has priority 9.

pseudo code:

int main:

startup

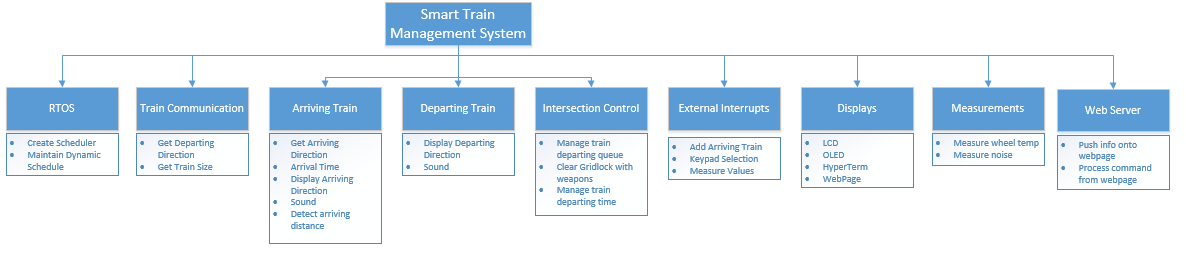
initialize

Create new tasks and add them to the list of tasks that are ready to run

Starts the RTOS scheduler

return 0;

end



**Figure 2.1 Functional Decomposition**

The program is composed by 13 major tasks: arrvingTrain, trainCom, departingTrain, switchControl, lcdDisplay, oledDisplay, localKeypad, serialCom, temperatureMeasurement, noiseCapture, noiseProcessing, command, and remoteCom.

**arrvingTrain:**

Increment the *trainArrving* each time SW2 interrupt the train system. If *trainArrving* is 0, do nothing. If *trainArrving* is greater than 0, put this train in an arrival queue, random generate to binary number indicating train incoming direction, generate random number n and wait 0.4n minutes to let the train arrives at the intersection. When the train is 1000m away from the intersection, the train output a signal indicating its remaining distance. The period of signal and remaining distance has a linear correlation, period = 53\*remaining distance -5175. We use a timer with constant frequency interrupt to measure the output signal frequency and calculate the remaining distance and shows the value on OLED display. When the train is 400m away from the intersection, we set a *checkTrain* flag to TRUE, indicating the scheduler should schedule *trainCom*. When the train arrives at the intersection, we reset all variables.

Before the train arrives at the intersection, if incoming direction is north, flash “North Train” at a 1.5 second rate on OLED display and blasts with a cycle of 2-2 2-1; if incoming direction is south, flash”South Train” at a 1.5 second rate on OLED display and blasts with a cycle of 2-2 2-3; if incoming direction is east, flash “East Train” at a 2 second rate on OLED display and blasts with a cycle of 3-2 2-1; if incoming direction is west, flash “West Train” at a 1 sec rate and blast with a cycle of 1-2 2-1.

pseudo code:

void arrvingTrain(void\* train)

if trainArrving >0

increase count

if firstTime

generate random direction that is not same as previous direction

generate random number n and set arrvialTime as 24n

set previous direction as current direction

set firstTime false

end

if remaining distance is less than 1000

generate a frequency according to its current remaining distance

display the remaining distance at OLED display

end

if the measured remaining distance is 10% less than the last time we measured

record the remaining distance in arrvingTrainDistance queue

end

if remaining distance is less than 400

set checkTrain flag to true;

end

flash direction on OLED display and blast according to its direction

if count to arrvialTime: reset everything.

end

end

We use Timer3IntHandler to measure the output frequency. Timer3 has a constant frequency 8000 Hz. So each time the the output is not 0, we increase the count so that we can measure the period of the output signal and remaining distance. We found the linear correlation between count and remaining distance as count \* 2.35 + 97.

void Timer3IntHandler(void)

capture a noise data

if the output voltage is a 1

increase count

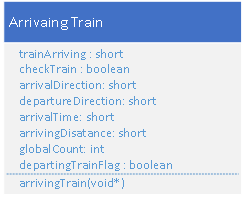
else

remaining distance = count \* 2.35 + 97

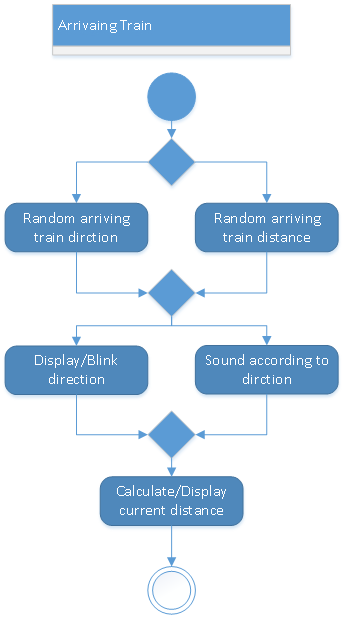
reset count

end

end



**Figure 2.2.1 Class diagram of Arriving Train**



**Figure 2.2.2 Activity diagram of Arriving Train**

**trainCom:**

This task manages the first train in the arrival queue, determines its departure direction and train size by generate random number and record these train information in queue trainDeparting, increase trainPresent, and decrease *trainArriving*. Then, set *checkTrain* flag to FALSE, indicating the scheduler to delete the *trainCom* task and suspend trainCom from scheduler..

void trainCom(void\* train)

random generate departure direction that is different from arrival direction

random generate train size between 1 and 9

record the size and departure direction in trainDeparting

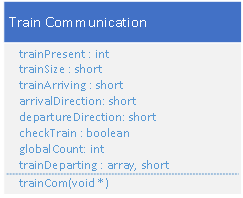
increase trainPresent

decrease trainArrving

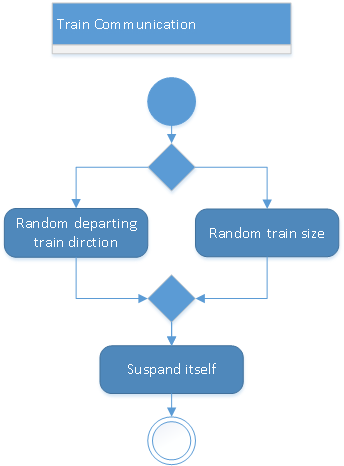
set *checkTrain* to false

suspend trainCom

end



**Figure 2.2.3 Class diagram of Train Communication**



**Figure 2.2.4 Activity diagram of Train Communication**

**departingTrain:**

If *departingTrainFlag* is false, do nothing. If *departureDirection* is north, blast with a cycle of 1-2 2-1. If *departureDirection* south, blast with a cycle of 1-3 2-1. If *departureDirection* is east, blast with a cycle of 1-4 2-1. If *departureDirection* is west, blast with a cycle of 1-5 2-1.

pseudo code:

void departingTrain()

if departingTrainFlag is true

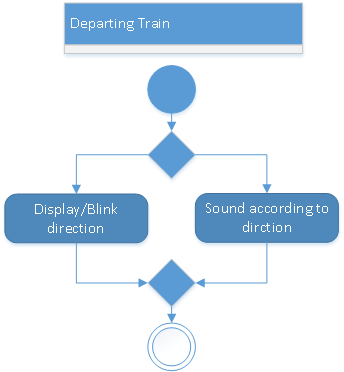
flash direction and blast according to departure direction, do nothing if departure direction is 0

end

end



**Figure 2.2.5 Class diagram of Departing Train**



**Figure 2.2.6 Activity diagram of Departing Train**

**switchControl:**

If *trainPresent* is 0, do nothing. If trainPresent is greater than 0 and current train size is 0(which means the last train has just passed), read the train size and departing direction from trainDeparting queue. Then, generate a random number n between -2 and 2. If n>0(indicating there is gridlock), wait 0.2n minutes before the train crosses the intersection. If it has gridlock, the system will automatically fires one phasor burst with 50% probability to clear gridlock. If phasor burst failed, the system will automatically fire a thermonuclear that has 100% probability to clear gridlock. If n<=0(indicating there is no gridlock) or the train has waited for 0.2n minutes or gridlock has been cleared, the train start to pass the intersection. Display informations on LCD and OLED displays. Then the train needs 0.1\**trainSize* minutes to pass the intersection. Then, reset *direction,* and *trainSize* and decrease *trainPresent*.

pseudo code:

void switchControl(void\* train)

if trainPresent is greater than 0

if train size is 0

read train size and departing direction from trainDeparting queue

set traversal time to be train size \* 6

generate random number n

if n<0

set gridlock to be true

set delayValue to be -12n

set delayTime = delayValue

generate random number m

if m >0

set delayValue to be 4

set burstFail to be false

reset m

else if m<0

set delayValue to be 6

set burstFail to be true

m = 0;

end

end

if gridlock

decrease delayValue

if delayValue reach 0

set gridlock to be false

end

else

intersectionLock is released

` reset all variables

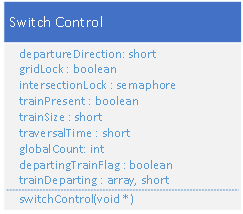
delete the train information from trainDeparting queue

end

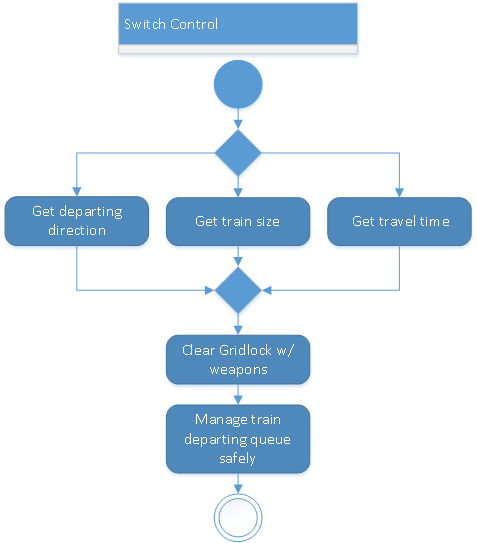
end

end

end



**Figure 2.2.7 Class diagram of Switch Control**



**Figure 2.2.8 Activity diagram of Switch Control**

**lcdDislay:**

If *trainPresent* is OFF, do nothing. If *trainPresent* is greater than 0 and has gridlock, display information of burst and thermonuclear. If there is no gridlock, display number of train present, departure direction and intersection lock at first row and display train size and traversal time at the second row.

pseudo code:

void LCDDisplay(void\* train)

if train present is greater than 0 and has gridlock

if burst fail

print “BURST: FAILURE”

print “>=(nuclear)>”

else

print ”BURST: SUCCESS”

end

else if train present is greater than 0 and train is departing

print number of trainPresent

print departure direction

print status of intersection

print spaces to print the next line

print “Size:”

print train size

print “Time:”

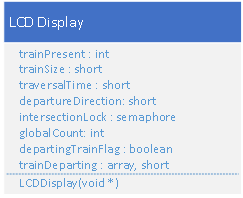
print traversal time

else

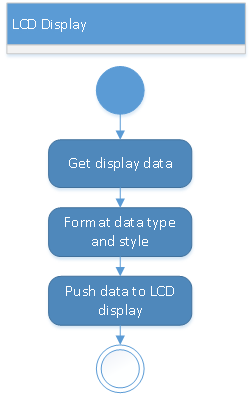
clear lcd

end

end



**Figure 2.2.9 Class diagram of LCD display**



**Figure 2.2.10 Activity diagram of LCD display**

**oledDisplay:**

According to command on website, if press D one time, show nothing, and if press D one more time, show information.

According to selection with keypad, if in Train Status mode, display “North Train”, “South Train”, “East Train”, and “West Train”. For each selected train, it should display its identification, amount of train arriving, whether the train is at intersection, train arriving direction, train departing direction, four lock status, train size, and its estimated traversal time. If in annunciation mode, show temperature and warning if temperature increases 20% or more.

pseudo code:

void oledDisplay(void\* train),

switch mode

case 0:

print “status” and “annunciation”

print cursor in front the choice the user selected

case 1:

if statusSelect is same as arrival direction

print out information of that train

else

print no train from that direction

end

case 2:

if gridlock and departure direction is not 0: print gridlock warning

else if gridlock is not 0 and departure direction is not 0: print train passing

else if departure direction is 0: print “No departure train”

end

show measured temperature

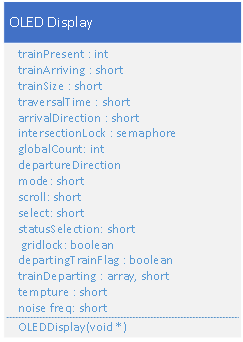
if one of the temperature is 20% greater than the recent temperature

print ”Too hot”

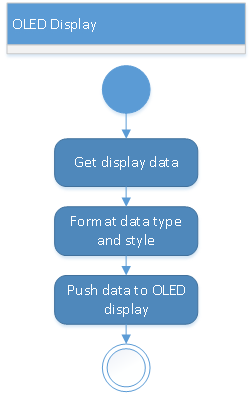
end

end

end



**Figure 2.2.11 Class diagram of OLED display**



**Figure 2.2.12 Activity diagram of OLED display**

**localKeypad:**

Use three functions to implement keypad. mode() controls the Mode Select screen selection; statusSelection() controls which train status the user want to see; scrollNselection() controls the scroll key and selection key. In localKepad(), those three functions run first one by one and if the user press home, we reset all variables including mode back to 0, and clear the screen.

pseudo code:

void localKeypad(void\* train)

call modeSelect(); call trainStatus(); call scrollNselect;

if press key 0

reset mode, select, annunciation, scroll and statusSelection

clear screen

end

If Mode Select screen(*mode* = 0) is shown, and the user choose the first choice, which is train status, *mode* changes to 1, and if the user choose the second choice, which is annunciation*, mode* changes to 2.

pseudo code:

void modeSelect()

if mode is 0

if select train status

set mode to 1

reset select and scroll

else if select annonciation

set mode to 2

reset select and scroll

end

end

end

If train status(*mode* = 1) is shown, and the user select north train, we change the *statusSelection* to 1, indicating the OLEDdisplay need to show information of north train. Same if the user choose south train, west train or east train.

pseudo code:

if select train status

set statusSelection to be the direction the user select

end

end

If the user press scroll key or selection key, we increase variable *scroll* or set *select* to 1.

pseudo code:

if press key 1

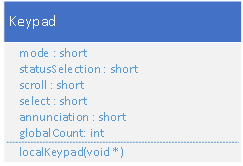
increase scroll

if press key 2

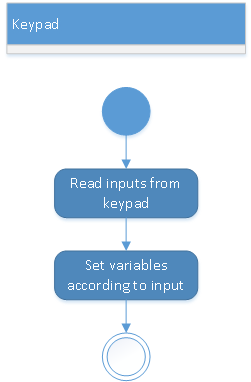
set select to be 1

else

else



**Figure 2.2.13 Class diagram of Keypad**



**Figure 2.2.14 Activity diagram of Keypad**

**serialCom:**

Send all the information need to be displayed to Hyperterm.

pseudo code:

void serialCom(void \*train)

if exist an arriving train

print “train present” and number of arriving train

print “Current Direction” and departure direction

print “Train size:” and train size

print “Traversal Time” and traversal time

print “Intersection availability”

if intersection lock is locked: print ”Not available”

else print “available”

print “Arriving train” and arriving direction

print “Arrival time” and arrival time

print “Gridlock”

if gridlock: print “Yes”; or print “No”

print ”Noise”

if noiseTransBuf is not NULL

print the recent measured noise frequency

else

print “NA”

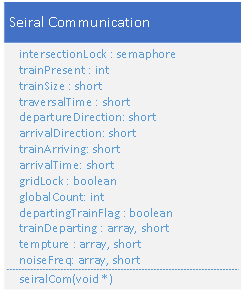
end

print “Temperature”

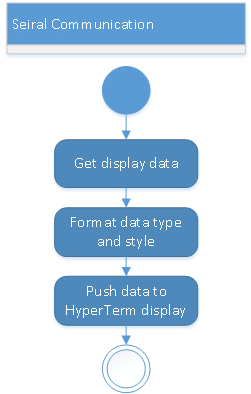
print the recent measured temperature

end

end



**Figure 2.2.15 Class diagram of Serial Communication**



**Figure 2.2.16 Activity diagram of Communication**

**temperatureMeasurement:**

read input voltage and convert to temperature

pseudo code:

void vTemperatureMeasurement(void\* train)

if press button, departingTrainFlag is true and user command measuring via internet

trigger ADC conversion

wait for conversion to be completed

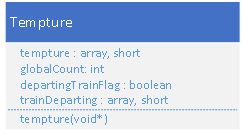
Read ADC value

convert the measured value in milivolt

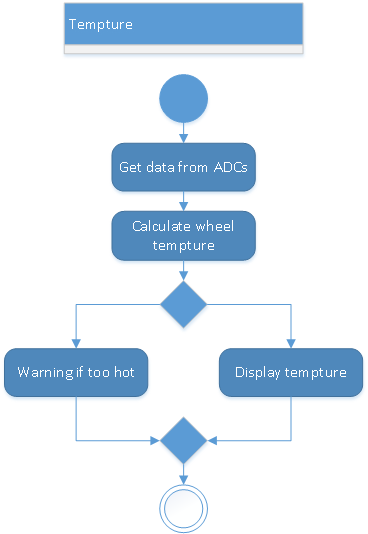
save the value in temperatureBuf queue

end

end



**Figure 2.2.17 Class diagram of Temperature Measurement**



**Figure 2.2.18 Activity diagram of Temperature Measurement**

**noiseCapture:**

We use a timer as an interrupt to capture 256 sample data and store the data in noiseCaptureBuf. We use Timer3 with frequency of 8000 Hz as an interrupt to measure the sample data so the sample frequency is 8000 Hz. Each time we have an interrupt, the task capture a data and put it in noiseCaptureBuf.

pseudo code:

void vNoiseCapture(void\* train)

if index is less than 256, interrupt from timer3 and user command measuring via internet

trigger ADC conversion

wait for conversion to be completed

Read ADC value

convert the value to range -31 to 32 and store it in noiseCaptureBuf[index]

increase index

if index reaches 256

reset index to 0

schedule noiseProcessing task

end

end

**noiseProcessing:**

read sample data and use fft function to calculate the frequency of the given sample data. Calculate the frequency every fifth times we capture a set of new data.

pseudo code:

void vNoiseProcessing(void\* train)

if the task is scheduled

if this is the fifth data set

create an imagBuf with size 256 and all zeros

calculate the peak index of the given sample data

calculate the noise frequency based on peak index

save the frequency in noiseTransBuf

end

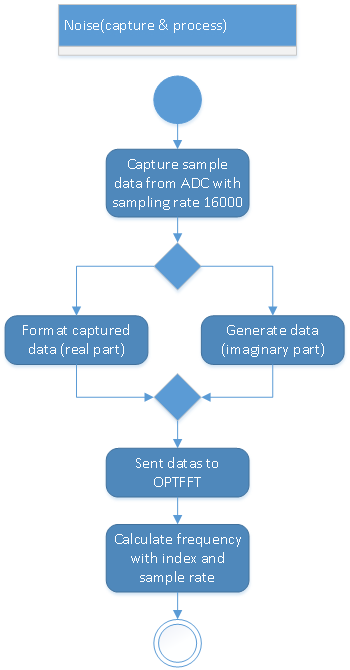
record the number of data set

end

end



**Figure 2.2.19 Class diagram of Noise**



**Figure 2.2.20 Activity diagram of Noise**

**command:**

capture the command from the internet and react as the command

pseudo code:

void vCommand(void\* train)

if the command is S: start measurement

else if command is P: stop measurement

else if command is D:

count how many time the user input D

if it is odd times: stop OLED display

else if it is even times: resume OLED display

end

else if command include M

if command also includes noise: show noiseTransBuf

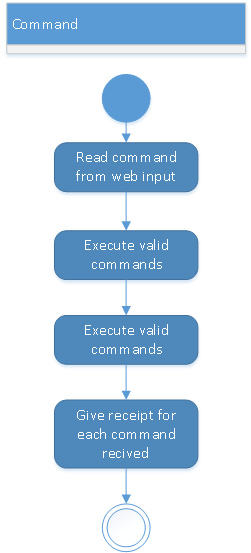
else if command also includes temp: show temperatureBuf

else: need to show error on internet

else if command is none of above: need to show error message on internet



**Figure 2.2.21 Class diagram of Command**



**Figure 2.2.22 Activity diagram of Command**

**remoteCom:**

Decide the messages that should be displayed on the website and display messages with sprintf.

pseudo code:

unsigned short vRemoteCom( void \*train )

assign departureDirection to departD

assign intersection availability to intersection

assign arrivalDirection to arrivalD

assign gridlock to grid

if the user input command Mnoise

assign noiseTransBuf to a temperate queue

else if the user input command Mtemp

assign temperatureBuf to a temperate queue

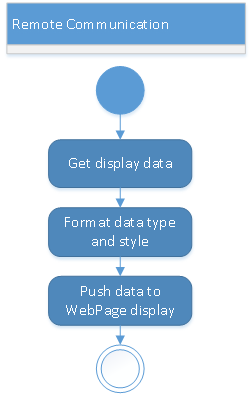
else: the temperature queue is all zero

show the messages on website

end



**Figure 2.2.23 Class diagram of Remote Communication**



**Figure 2.2.24 Activity diagram of RemoteCommunication**

**PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS**

When the system start working, the Mode Select Page displays on the OLED. Two modes can be selected on this page as Figure 3.1 shows below, Status and Annunciation. At any time key0 is pressed on the keypad, the OLED will back to Mode Select page.

As there is no arriving train at beginning, LCD and Hyperterm shows nothing.



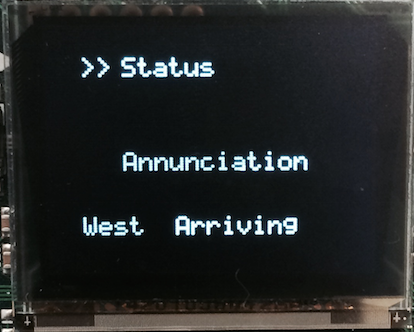
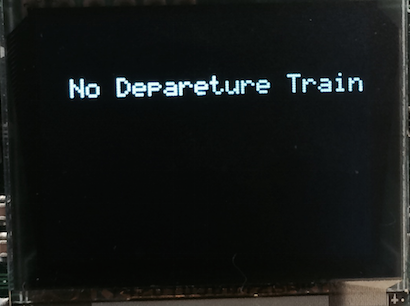
**Figure 3.1 Mode Select Page of OLED**

When press SW2, there will be one train from random direction arriving, and the number of arriving train increased by 1 each time when SW2 is pressed.

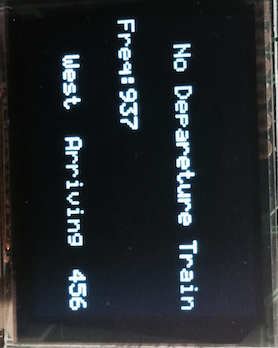
We pressed SW2, the Figure 3.2 below shows that there is a west train arriving and “West Arriving” flashs at a 1.0 second rate and the annunciation consists of 1 long blast of 2 seconds each followed by 2 short blasts of 1 second each. “West Arriving” keeps flashing and annunciation until the arrival time count-down to zero.

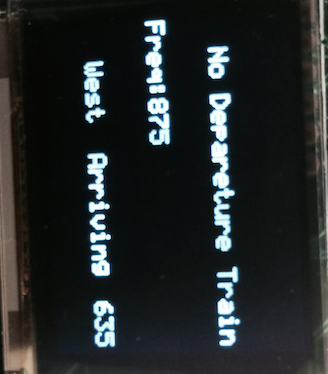
Key1 is used to scroll the arrow, and Key2 to used to select the mode. We also need to command “S”(indicates start mode) on the remote display to start the measurement, and those measurement can be stopped when we command “P”(indicates stop mode) on remote display.The measurement includes noise measurement and temperature measurement. The noise keeps measuring unless we command “P”(indicates stop mode). The temperature will be measured when we press SW3 and there is a train at the intersection.

We press Key1 to scroll the arrow to “Annunciation” and press Key2 to select, and Annunciation mode page is show on OLED as Figure 3.2 below. As there is no departure train right now, the OLED shows the message and “West Arriving” keeps flashing as Figure 3.3 below.

**Figure 3.2 Screenshot of OLED Figure 3.3 Annunciation mode on OLED**

We command “S” on the remote display to start the measurement, therefore frequency is now showing on OLED as Figure 3.4. As we generate the sine wave with frequency 1k by function generator, the frequency shows on OLED should around 1000. The train are equipped with a transmitter that is automatically activated within 1km from the intersection, and the distance to intersection will display on OLED when the train within 1km from the intersection. Figure 3.4 and 3.5 below shows the measurement of noise frequency and the arriving distance(the distance is decreasing and the train is closer and closer to the intersection).

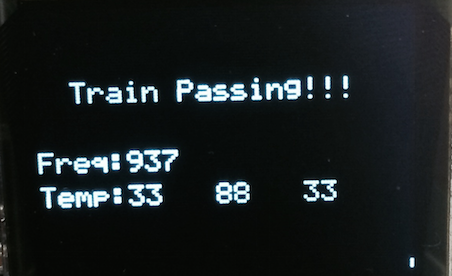
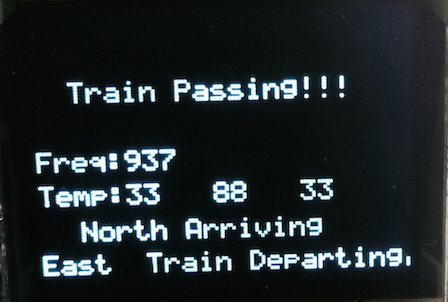


**Figure 3.4 Screenshot of OLED Figure 3.5 Screenshot of OLED**

When the arrival time and arriving distance count-down to zero, the train will start departing.The program generates a random from 1- 4 to indicate departing direction, the departing direction should not same as arriving direction. We press SW2 again, there will be another arriving train. And we keep press SW2, the number of arriving train keep increasing.

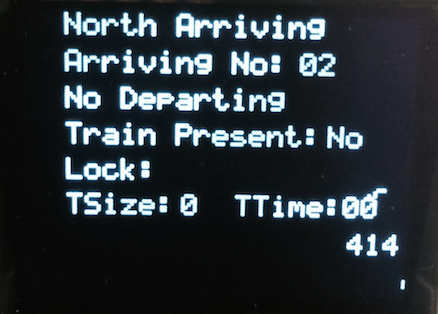
The Figure 3.6 and Figure 3.7 below shows that there is a north train arriving and “North Arriving” flashes at a 1.5 second rate, also there is a east train departure and “East Train Departing” flash at a 2.0 second rate. As there are two trains arriving and departing at same time, only departing train’s annunciation can be heard and the east departing train annunciation consists of 3 long blasts of 2 seconds each followed by 2 short blasts of 1 second each and each blast is separated by 1 second. Also, as there is an East Train departing without gridlock, the “Train Passing” message shows on OLED.

### When we press SW2, as there is a train the intersection, the temperature can be measured. The temperature is changed by altering the potentiometer. When any of the three currently measured values exceeds the largest of the three most recent previously measured values by more than 20%, the “Too Hot” warning will show on OLED.



**Figure 3.6 Screenshot of OLED Figure 3.7 Screenshot of OLED**

We then change to “Train Status” mode by pressing keys on keypad. And we select “North Train”, then train status for the selected train including Identification, Train Arriving / Present, Direction Arriving, Direction Departing,Lock Status,Train Size, Estimated Traversal Time should display as Figure 3.8 below. As there is no train departure, train present is no, Tsize and TTime is 00.



**Figure 3.8 Display of Train Status**

When there is no train departing, the LCD is clear, and if there is a departing train, LCD show the departing train message.

If there is no gridlock, LCD directly shows the departure train information as Figure 3.9 below. The semaphore first shows on and then changes to off when the train has passed as the intersection need to be locked to prevent other train enters the intersection.



**Figure 3.9 Display of Departure Train Information on LCD**

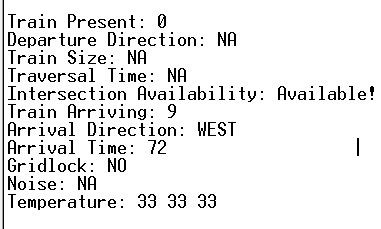
If there is a gridlock, at first signs of gridlock, a burst is used to clear the gridlock of intersection. The bust is a 50-50 chance succeed. If the burst succeed, LCD shows succeed information as Figure 3.10 below and then show the departure train information when the gridloc cleared. If the burst not succeed, LCD shows failure information as Figure 3.11 below, and a nuclear is automatically coming to clear the gridlock.



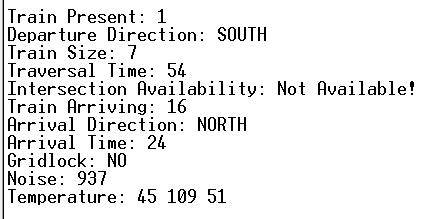
**Figure 3.10 Burst Succeed Information on LCD**

**Figure 3. 11 Burst Failure and Nuclear Information on LCD**

On the Hyperterm, specified status, warning, alarm information also measurement data shows as Figure 3.12 and 3.13 below.



**Figure 3.12 Screenshot of Hyperterm (No departing train)**



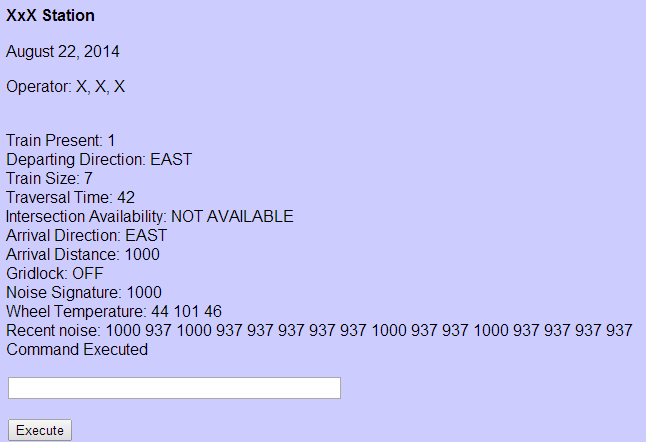
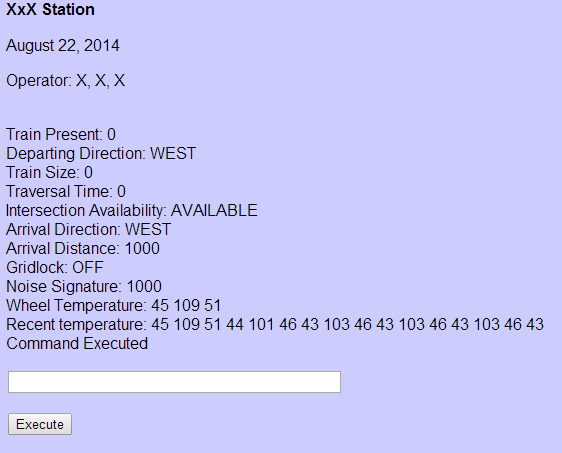
**Figure 3.13 Screenshot of Hyperterm(With departing train)**

On remote display, the name of the station, current date, operator’s name, also data, status, and warning information shows as Figure 3.14 below. And it continually update the displayed data at a 5 second rate.



**Figure 3.14 Remote Display**

If we command “Mnoise”, 16 most recent noise data will show as Figure 3.15 below and if we command “Mtemp”, 16 most recent temperature data will show as Figure 3.16 below. When the command is correct and accepted, “Command Executed” information will shows on remote display, otherwise, “Command Error” warning will shows.

**Figure 3.15 Mnoise Command for Display Figure 3.16 Mtemp Command for Display**

All the display shows that our result matches the expected output!

**Empirically Measured Individual Task Execution Time**

|  |  |
| --- | --- |
| arrivingTrain | 2.21 us (no train arriving)  5.16 ms (when a train arriving) |
| trainCom | 3.32 ms (checkTrain) |
| departingTrain | 2.26 us (train not present)  3.1 ms (train present) |
| switchControl | 6.75 ms |
| LCDDisplay | 13.3 ms |
| OLEDDisplay | Mode select Page ----- 5.7 ms  Annunciation Page ---- 13.8 ms  Train Status Page ---- 12.2 ms  Train Selection Page ---13.6 ms |
| localKeypad | 18.4 us |
| serialCom | 44.5 ms |
| noiseCapture | 30.2 ms |
| noiseProcessing | 11.0 ms |
| Tempture | 2.6 ms |
| remoteCom | 7.3 ms |
| command | 12.2 ms |

**Table 1. Measured task execution time**

**ANALYSIS OF ANY ERRORS**

The LCD doesn’t show information of burst and nuclear. We use debugger to find if the system runs those statements, and it turns out the logic works fine. The statements should print characters on the LCD display but it show nothing. Then we debug line by line and found out that we give RS a 0\*00 instead of 0\*FF.

**ANALYSIS OF WHY THE PROJECT MAY NOT HAVE WORKED**

The LCD flashes during display. We tried to change the task delay rate and the flash frequency increases as we decrease the delay rate. We think it might be caused by FreeRTOS and it clear the screen at a rate that we can see.

Noise frequency is not calculated correctly. It only works when the input frequency is 1 kHz, 2 kHz, 3 kHz and 3.5 kHz. During demo, we realized that it is caused by interrupt. So we should use other way to realize this feature.

**TEST PLAN**

For this project, we need to make sure that we can compile the program and then download and run the program on the target platform correctly.

We first need to test that each task works properly. For random number generation function, we need to make sure that it can randomly generate numbers in the range we want. For vArrvingTrain and vDepartingTrain tasks we need to test that when the proper train arriving or departing, it flashes at proper rate and the annunciation is also correct. For vArrvingTrain we should also test that the signal we transmit is correct and the arriving distance is count-down when the train closer to intersection.For vSwitchControl task, we should test that the gridlock time and traversal time we compute are correct, “Gridlock Warning” and “Train Passing” warning are showing at proper time, also semaphore is on and off at correct time and in correct sequence. Besides that, for vSwitchControl task, we should also test that the burst or nuclear can be used to clear the gridlock. For vTrainCom task we should test that departing train is add to departing queue after an arriving train arrived. For vTemperatureMeasurement task we should test that temperature can be measured and “Too Hot” warning will show at proper tiem. For vNoiseCapture and vNoiseProcessing task we should test that the frequency we compute is almost same as the frequency on the function generator.

After that we need to test the keypad, SW2 and SW3 can properly worked and then we then need to test vLcdDisplay, vOledDisplay, vSerialCom tasks to make sure that all information are correctly shows on LCD, OLED and Hyperterm. We should finally test vCommand and vRemote task to make sure the remote display shows correct data and can receive proper command.

**TEST SPECIFICATION**

1. Test random number generation. For train size, the random number is from 1 to 9. For arriving direction and departure direction, the random number is generated from 1 to 4. Also we need to test the randN in arrivingTrain task is from 1 to 3, and n in switchControl task is from -2 to 2.

2. Test the flash rate and annunciation of arriving train. The North Train should flash at 1.5 second rate and the annunciation consists of 2 long blasts of 2 seconds each followed by 2 short blasts of 1 second each. The South Train should flash at 1.5 second rate and the annunciation consists of 2 long blasts of 2 seconds each followed by 2 long blasts of 2 second each. The East should flash at a 2.0 second rate and the annunciation should consist of 3 long blasts of 2 seconds each followed by 2 short blasts of 1 second each. The West Train should flash at 1.0 second rate and the annunciation should consist of 1 long blast of 2 seconds each followed by 2 short blasts of 1 second each.

3. Test the flash rate and annunciation of departing train. The North Train should flash at 1.5 second rate and the annunciation consists of 1 long blasts of 2 seconds each followed by 2 short blasts of 1 second each. The South Train should flash at 1.5 second rate and the annunciation consists of 1 long blasts of 3 seconds each followed by 2 short blasts of 1 second each. The East should flash at a 2.0 second rate and the annunciation should consist of 1 long blasts of 4 seconds each followed by 2 short blasts of 1 second each. The West Train should flash at 1.0 second rate and the annunciation should consist of 1 long blast of 5 seconds each followed by 2 short blasts of 1 second each.

4. Test the arriving distance. The arriving distance displayS within 1km from intersection, and the distance keeps counting-down until 0, after that a train will start departing.

5. Test the gridlock, burst, nuclear and traversal time. When n<0, there is a gridlock, Gridlock Warning should be shown. When there is gridlock, a burst is used to clear the gridlock. If m > 0, the burst failed, and a nuclear is automatically used to clear the gridlock. If m < 0, the burst succeed, the gridlock cleared and train start passing. When the train start passing,the traversal time equal to trainSize \* 6.

6. Test the setting of semaphore. When train start passing, the semaphore should set to on to lock the intersection. And after the train has passed, the locks should be unlocked, and semaphore should set to off.

7. Test departing queue. When an arriving train arrived at intersection, a departing train will add to departing queue and departing trains will departure in sequence.

8. Test temperature measurement, when any of the three currently measured values exceeds the largest of the three most recent previously measured values by more than 20%, “Too Hot” warning will show.

9. Test noise measurement, the frequency we compute should almost same as the frequency of sine wave we generate by function generation.

10. Test SW2. When SW2 has been pressed, the number of arriving train should increase by 1.

11. Test SW3, when SW3 has been pressed, temperature should be measured.

12. Test Keypad. When Key0 has been pressed, the Mode Select page should be displayed on OLED. When Key1 has been pressed, the arrow points to an option should point to another option. When Key2 has been pressed, the option has been selected and the display should jump to directed page.

13. Test the LCD display. When trainPresent < 0, nothing is displayed on LCD. If the trainPresent > 0, on the top row of the display shows value of trainPresent, departure train direction, and the state of semaphore on the second row of the display shows the current trainSize, and traversalTime.

14. Test the Hyperterm display. The hyperterm should display information of train present, departure direction, train size, traversal time, intersection availability, train arriving, arrival direction, arrival time, gridlock, noise signature and wheel temperature.

15. Test the OLED display. The Mode Select page shows two modes, option of status and annunciation should be shown. In the Train Status mode, option of North Train, South Train, East Train and West Train should display. And if there is an arriving train in the direction you select, Train status for the selected train including Identification, Train Arriving / Present, Direction Arriving, Direction Departing,Lock Status,Train Size, Estimated Traversal Time should display. If there is no train in the direction you select, the information of no train coming will also display. In the Annunciation mode, the information of gridlock, noise frequency and wheel temperature should be display.

16. Test remote display. Remote display should show the name of the station, current date, the operator’s name and data, status, and warning information including train present, departure direction, train size, traversal time, intersection availability, train arriving, arrival direction, arrival time, gridlock, noise signature and wheel temperature.

17. Test command. The S command indicates START mode. The P command indicates STOP mode. The D command enables or disables the OLED display. The M command requests the return of the most recent value(s) of the specified data. Command execution and error should also show.

**TEST CASES**

1. To test the random number generation function, we write the function in a separate file and run the function to use OLED to display the number it generated. For train size, we set int low to 1, int high to 9, the program generates several numbers and display them OLED. We then check if those numbers are all in the range of 1 to 9. For arriving direction and departure direction, we set int low to 1, int high to 4, and we then check if those numbers are all in the range of 1 to 4. For n in switchControl task, we set int low to -2, int high to 2, and we then check if those numbers are all in the range of -2 to 2.

2. To test the flash rate and annunciation of arriving train. We first set the arrivalDirection to 0 instead of random number to let the system always generate the arriving train to the North, therefore test the exact flash rate and annunciation of North Train. We then set the arrivalDirection to 1 to test South Train, 2 to test East Train, and 3 to test West Train.

3. To test the flash rate and annunciation of departing train. We first set the departureDirection to 0 instead of random number to let the system always generate the departure train to the North, therefore test the exact flash rate and annunciation of North Train. We then set the departureDirection to 1 to test South Train, 2 to test East Train , and 3 to test West Train.

4. To test the arriving distance, we set the arrival time to a constant number and then test that the arriving distance starts at 1000 and keep counting down to zero.

5. To test the gridlock time and traversal time. We set the the trainSize to a specific number from 1 to 9. We added a timer on the OLED to show the traversal time to test that the train passes at proper time.

We first set the random number n in switchControl Task to a positive number 1 or 2, therefore there is no grid lock for the train. As there is no grid lock, the train start passirng at beginning. The traversalTime = 6 \* trainSize, and when the counter count to the traversal time, the train should have passed the intersection.

We then set the random number n in switchControl task to a negative number -1 or -2, therefore there is a gridlock. The gridlock warning should shows at beginning. We set the random number m to a positive number 1 or 2 to test the case burst succeed and set the random number m to a negative -1 or -2 to test the case burst failed and nuclear is used to clear the gridlock.

When the train start passing, when the counter count to the traversal time, the train should have been passed the intersection.

6. To test the setting semaphore, we look at the LCD display, which clearly displays the status of semaphore.

7. To test the departing queue, we add several arriving train and print the departing queue out the see the direction and train size.

8. To test the temperature measurement, we first set the potentiometer to the smallest value and test the temperature, then set the potentiometer to the largest value to test the temperature. “Too Hot” warning should show at this time.

9. To test the noise measurement, we keep measuring the frequency and print out the data. We compare the data we print with the frequency we set on function generator.

10. To test SW2, we let the number of arriving train display on OLED, and then test whether the SW2 works correctly. When we press SW2, the number of arriving train should increase and only increase by 1. We press the SW2 with different speed to make sure the key is working.

11. Same as SW3, we change the potentiometer and press SW. Then we look at the temperature data, the temperature data should change each time we press the button. Press the SW3 with different speed to make sure the key is working.

12. To test Keypad, we just press the key and looked at the OLED to make sure that all three keys work functionally correctly as Test Spec.

12. Test Keypad. When Key0 has been pressed, the Mode Select page should be displayed on OLED. When Key1 has been pressed, the arrow points to an option should point to another.

13. & 14. & 15. & 16To test all displays on LCD, OLED, Hyperterm, and remote displau we down load the program, and run the program. We compare all the displays with our specification to make sure those display are correct. And we then compare all four screens to make sure that information are consistent.

17. To test command we input different command on remote display to make sure that each command works properly.

**SUMMARY**

This is the last step of developing a smart train system using IAR embedded system and C language. In this lab, we make a further improvement of the system based on the spec and we debug and test the system. The inputs is SW 2, SW 3 on the Stellaris board, key 0, 1, 2 on keypad, random numbers voltage input and time interrupt generated by the system. The outputs are blasts according to train direction and information shown on LCD display, OLED display webpage and Hyperterm. The messages include train status (arriving direction, arrival time, train size, arrival direction, departure direction, arriving time, traversal time) and annunciation (gridlock, train passing, intersection lock, temperature). To test the system, we tested basic functions of the system to make the system run for one time with given specific parameters. Then add task and functions to see if they meet the requirement. And our project works as asked by the spec.

**CONCLUSION**

Through this project, we make an improvement on the previous smart train system. We move our tasks to a real-time operating system, FreeRTOS, used analog digital converter (ADC), and design a webpage that shows all the information.

For further development, the engineers can simulate more intersections and make it as a real rail net of the country. Then the train size, arriving direction, departing direction, etc should be given by the user instead of random generated number. And the system should find the best way to control the trains.

**ESTIMATE HOURS EACH PERSON**

|  |  |
| --- | --- |
| Design | 20 |
| Coding | 38 |
| Test/Debug | 50 |
| Documentation | 8 |

**APPENDICES**

**Appendix A: main.c**

// \*================================================================

// Smart Train Management System

// Xinyi Chang, Xindi Liu, Xuanlin Zhu

// 2014 - 08 - 22

// \*================================================================

**Appendix B: lab4header.h**

// \*================================================================

// emulate the Boolean type

// From EE 472 WEB SITE

// \*================================================================

enum myBool { FALSE = 0, TRUE = 1 };

typedef enum myBool Bool;

// \*================================================================

// emulate the lock type

// \*================================================================

enum myLock { OFF = 0, ON = 1 };

typedef enum myLock Lock;

// \*================================================================

// TCB structure

// \*================================================================

struct taskSct

{

void (\*myTCB)(void\*);

void\* taskDataPtr;

};

typedef struct taskSct TCB;

struct arrivingTrainDatax

{

unsigned short\* trainArriving;

unsigned short\* trainPresent;

Bool\* checkTrain;

unsigned short\* arrivalTime;

unsigned short\* arrivalDirection;

unsigned int\* globalCount;

unsigned short\* departureDirection;

unsigned short\* arrivingTrainDistance;

Bool\* departingTrainFlag;

};

typedef struct arrivingTrainDatax arrivingTrainData;

struct trainComDatax

{

unsigned short\* trainArriving;

Bool\* departingTrainFlag;

unsigned short\* trainSize;

unsigned short\* arrivalDirection;

unsigned short\* departureDirection;

Bool\* checkTrain;

unsigned int\* globalCount;

unsigned short\* trainDeparting;

unsigned short\* trainPresent;

unsigned short\* traversalTime;

};

typedef struct trainComDatax trainComData;

struct departingTrainDatax

{

Bool\* departingTrainFlag;

unsigned short\* departureDirection;

unsigned int\* globalCount;

unsigned short\* trainDeparting;

unsigned short\* trainPresent;

};

typedef struct departingTrainDatax departingTrainData;

struct switchControlDatax

{

unsigned short\* departureDirection;

Lock\* interLock; //intersection lock

Bool\* gridlock;

Bool\* departingTrainFlag;

unsigned short\* trainSize;

unsigned short\* traversalTime;

unsigned int\* globalCount;

unsigned int\* temperatureBuf;

unsigned short\* trainDeparting;

unsigned short\* trainPresent;

};

typedef struct switchControlDatax switchControlData;

struct lcdDisplayDatax

{

Lock\* interLock; //intersection lock

Bool\* departingTrainFlag;

unsigned short\* trainSize;

unsigned short\* traversalTime;

unsigned short\* departureDirection;

unsigned int\* globalCount;

unsigned short\* trainPresent;

Bool\* gridlock;

};

typedef struct lcdDisplayDatax lcdDisplayData;

struct oledDisplayDatax

{

Bool\* departingTrainFlag;

unsigned short\* trainArriving;

unsigned short\* trainSize;

unsigned short\* traversalTime;

unsigned short\* arrivalDirection;

Lock\* interLock; //intersection lock

unsigned int\* globalCount;

unsigned short\* departureDirection;

unsigned short\* mode;

unsigned short\* scroll;

unsigned short\* select;

unsigned short\* statusSelection;

Bool\* gridlock;

unsigned int\* temperatureBuf;

unsigned short\* arrivingTrainDistance;

unsigned short\* trainDeparting;

unsigned short\* trainPresent;

};

typedef struct oledDisplayDatax oledDisplayData;

struct localKeypadDatax

{

unsigned short\* mode;

unsigned short\* statusSelection;

unsigned short\* scroll;

unsigned short\* select;

unsigned short\* annunciation;

unsigned int\* globalCount;

};

typedef struct localKeypadDatax localKeypadData;

struct serialComDatax

{

Lock\* interLock; //intersection lock

Bool\* departingTrainFlag;

unsigned short\* trainSize;

unsigned short\* traversalTime;

unsigned short\* departureDirection;

unsigned short\* arrivalDirection;

unsigned short\* trainArriving;

unsigned short\* arrivalTime;

Bool\* gridlock;

unsigned int\* globalCount;

unsigned int\* temperatureBuf;

unsigned short\* trainDeparting;

unsigned short\* trainPresent;

unsigned int\* noiseTransBuf;

};

typedef struct serialComDatax serialComData;

struct temperatureDatax

{

Bool\* departingTrainFlag;

unsigned int\* temperatureBuf;

Bool\* globalStartMeasure;

};

typedef struct temperatureDatax temperatureData;

struct noiseCaptureDatax

{

signed int\* noiseCaptureBuf;

Bool\* globalStartMeasure;

};

typedef struct noiseCaptureDatax noiseCaptureData;

struct noiseProcessingDatax

{

signed int\* noiseCaptureBuf;

unsigned int\* noiseTransBuf;

int\* noiseFrequency;

};

typedef struct noiseProcessingDatax noiseProcessingData;

struct remoteCommunicationDatax

{

Bool\* departingTrainFlag;

unsigned short\* trainSize;

unsigned short\* traversalTime;

unsigned short\* departureDirection;

unsigned short\* arrivalDirection;

unsigned short\* trainArriving;

unsigned short\* arrivalTime;

Bool\* gridlock;

unsigned int\* temperatureBuf;

unsigned short\* trainPresent;

unsigned short\* arrivingTrainDistance;

unsigned short\* noiseTransBuf;

unsigned short\* showRecentData;

};

typedef struct remoteCommunicationDatax remoteCommunicationData;

struct commandDatax

{

char\* readCommand;

Bool\* globalStartMeasure;

unsigned short\* showRecentData;

};

typedef struct commandDatax commandData;

// ++++++++++++++++++++++++++++++++++++++++++++++++++

// initialize variables

// ++++++++++++++++++++++++++++++++++++++++++++++++++

//unsigned portBASE\_TYPE task1;

//unsigned portBASE\_TYPE task2;

//unsigned portBASE\_TYPE task3;

//unsigned portBASE\_TYPE task4;

//unsigned portBASE\_TYPE task5;

//unsigned portBASE\_TYPE task6;

//unsigned portBASE\_TYPE task7;

//unsigned portBASE\_TYPE task8;

//unsigned portBASE\_TYPE task9;

//unsigned portBASE\_TYPE task10;

//unsigned portBASE\_TYPE task11;

Bool burstFail = TRUE;

unsigned short showRecentData = 0;

extern char readCommand[32];

Bool globalStartMeasure = FALSE;

Bool globalStartDisplay = TRUE;

int commandAccept = 0;

int noiseFrequency = 0;

Bool execute = FALSE;

signed int noiseCaptureBuf[256];

int noiseTransBuf[16];

Bool captureNoise = FALSE;

Bool processNoise = FALSE;

unsigned short test11 = 1;

Bool addingTrain = FALSE; //debounce sw2

Bool measuringTemp = FALSE; //debouce sw3

Bool measure = FALSE;

Bool tcGo = FALSE;

Bool startSerial = FALSE;

Bool RunMinor = TRUE;

// command and control - direction

unsigned short arrivalDirection = 4;

unsigned short departureDirection = 4;

// command and control - annunciation

Bool checkTrain = FALSE;

// Status and annunciation management

Bool departingTrainFlag = FALSE;

unsigned short trainSize = 0;

unsigned short traversalTime = 0;

unsigned short arrivalTime = 0;

unsigned short trainArriving = 0;

unsigned short arrivingWhistle = 0;

unsigned short departingWhistle = 0;

// Keypad

Bool OLEDChanged = TRUE;

unsigned short mode = 0; //mode select = 0, train status = 1, annunciation = 2

unsigned short statusSelection = 0;

unsigned short scroll = 0;

unsigned short select = 0;

unsigned short annunciation = 0;

unsigned short key0 = 1; //home--A4

unsigned short key1 = 1; //scroll--D4

unsigned short key2 = 1; //select--D5

// Alarm

Bool gridlock = FALSE;

unsigned int globalCount = 0;

//new

unsigned short arrivingTrainDistance[8];

unsigned long temperatureBuf[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

Bool wheelTemperature = FALSE;

Lock interLock = OFF; //intersection lock

xSemaphoreHandle intersectionLock;

unsigned int remainDistance = 1000;

// ADC

unsigned long ulValue3;

unsigned long ulValue0;

unsigned long ulValue1;

unsigned long ulValue2;

unsigned short trainDeparting[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //train departing queue

unsigned short trainPresent = 0;

arrivingTrainData myArrivingTrain;

trainComData myTrainCom;

departingTrainData myDepartingTrain;

switchControlData mySwitchControl;

lcdDisplayData myLcdDisplay;

oledDisplayData myOledDisplay;

localKeypadData myLocalKeypad;

serialComData mySerialCom;

temperatureData myTemperature;

noiseCaptureData myNoiseCapture;

noiseProcessingData myNoiseProcessing;

remoteCommunicationData myRemote;

commandData myCommand;

TCB arrivingTD;

TCB trainCD;

TCB departingTD;

TCB SWCD;

TCB LCDD;

TCB OLEDD;

TCB keypadD;

TCB serialCD;

TCB temperatureD;

TCB noiseCD;

TCB noisePD;

TCB ramoteCD;

TCB commandD;

// \*================================================================

// define methods

// \*================================================================

int randomInteger(int low, int high);

/\*

void arrivingTrain(void\* train);

void trainCom(void\* train);

void departingTrain(void\* train);

void switchControl(void\* train);

void LCDDisplay(void\* train);

void OLEDDisplay(void\* train);

void localKeypad(void\* train);

void serialCom(void\* train);

void temperatureMeasurement(void\* train);

\*/

void lcdSR();

void lcdClear();

void printOn();

void printOff();

void printSize();

void printTime();

void printNorth();

void printSouth();

void printWest();

void printEast();

void printSpace();

void printNumber(short number);

void IntTimer0(void);

void IntGPIOf(void);

void insert(TCB\* node);

void UARTSend(const unsigned char \*pucBuffer, unsigned long ulCount);

void delay(unsigned long aValue);

void startUp();

void initialize();

void deleteFunction(TCB\* node);

void modeSelect();

void trainStatus();

void scrollNselect();

void printGridLock();

void printinterLock();

signed int optfft(signed int real[256], signed int imag[256]);

void printO();

void printX();

void printBurst();

void printNuclear();

# define TIME\_BASE 3846 // define 3846 delay in system as 0.5s delay

**Appendix C: main.c**

/\*

FreeRTOS V7.0.1 - Copyright (C) 2011 Real Time Engineers Ltd.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* FreeRTOS tutorial books are available in pdf and paperback. \*

\* Complete, revised, and edited pdf reference manuals are also \*

\* available. \*

\* \*

\* Purchasing FreeRTOS documentation will not only help you, by \*

\* ensuring you get running as quickly as possible and with an \*

\* in-depth knowledge of how to use FreeRTOS, it will also help \*

\* the FreeRTOS project to continue with its mission of providing \*

\* professional grade, cross platform, de facto standard solutions \*

\* for microcontrollers - completely free of charge! \*

\* \*

\* >>> See http://www.FreeRTOS.org/Documentation for details. <<< \*

\* \*

\* Thank you for using FreeRTOS, and thank you for your support! \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This file is part of the FreeRTOS distribution and has been modified to

demonstrate three simple tasks running.

FreeRTOS is free software; you can redistribute it and/or modify it under

the terms of the GNU General Public License (version 2) as published by the

Free Software Foundation AND MODIFIED BY the FreeRTOS exception.

>>>NOTE<<< The modification to the GPL is included to allow you to

distribute a combined work that includes FreeRTOS without being obliged to

provide the source code for proprietary components outside of the FreeRTOS

kernel. FreeRTOS is distributed in the hope that it will be useful, but

WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY

or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for

more details. You should have received a copy of the GNU General Public

License and the FreeRTOS license exception along with FreeRTOS; if not it

can be viewed here: http://www.freertos.org/a00114.html and also obtained

by writing to Richard Barry, contact details for whom are available on the

FreeRTOS WEB site.

1 tab == 4 spaces!

http://www.FreeRTOS.org - Documentation, latest information, license and

contact details.

http://www.SafeRTOS.com - A version that is certified for use in safety

critical systems.

http://www.OpenRTOS.com - Commercial support, development, porting,

licensing and training services.

\*/

/\*

\* Creates all the application tasks, then starts the scheduler. The WEB

\* documentation provides more details of the standard demo application tasks.

\* In addition to the standard demo tasks, the following tasks and tests are

\* defined and/or created within this file:

\*

\* "OLED" task - the OLED task is a 'gatekeeper' task. It is the only task that

\* is permitted to access the display directly. Other tasks wishing to write a

\* message to the OLED send the message on a queue to the OLED task instead of

\* accessing the OLED themselves. The OLED task just blocks on the queue waiting

\* for messages - waking and displaying the messages as they arrive.

\*

\* "Check" hook - This only executes every five seconds from the tick hook.

\* Its main function is to check that all the standard demo tasks are still

\* operational. Should any unexpected behaviour within a demo task be discovered

\* the tick hook will write an error to the OLED (via the OLED task). If all the

\* demo tasks are executing with their expected behaviour then the check task

\* writes PASS to the OLED (again via the OLED task), as described above.

\*

\* "uIP" task - This is the task that handles the uIP stack. All TCP/IP

\* processing is performed in this task.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Please ensure to read http://www.freertos.org/portlm3sx965.html

\* which provides information on configuring and running this demo for the

\* various Luminary Micro EKs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set the following option to 1 to include the WEB server in the build. By

default the WEB server is excluded to keep the compiled code size under the 32K

limit imposed by the KickStart version of the IAR compiler. The graphics

libraries take up a lot of ROM space, hence including the graphics libraries

and the TCP/IP stack together cannot be accommodated with the 32K size limit. \*/

// set this value to non 0 to include the web server

#define mainINCLUDE\_WEB\_SERVER 1

/\* Standard includes. \*/

#include <stdio.h>

//self add

#include <limits.h>

#include <string.h>

#include <math.h>

/\* Scheduler includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "semphr.h"

/\* Hardware library includes. \*/

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_sysctl.h"

#include "driverlib/sysctl.h"

#include "driverlib/gpio.h"

#include "grlib.h"

#include "drivers/rit128x96x4.h"

#include "osram128x64x4.h"

#include "formike128x128x16.h"

//self add

#include "driverlib/debug.h"

#include "driverlib/pwm.h"

#include "driverlib/interrupt.h"

#include "driverlib/timer.h"

#include "driverlib/uart.h"

#include "driverlib/adc.h"

#include "inc/hw\_ints.h"

/\* Demo app includes. \*/

#include "lcd\_message.h"

#include "bitmap.h"

//struct

#include "lab4header.h"

#include "uip.h"

#include "psock.h"

#include "httpd.h"

#include "httpd-cgi.h"

#include "httpd-fs.h"

/\*-----------------------------------------------------------\*/

/\*

The time between cycles of the 'check' functionality (defined within the

tick hook.

\*/

#define mainCHECK\_DELAY ( ( portTickType ) 5000 / portTICK\_RATE\_MS )

// Size of the stack allocated to the uIP task.

#define mainBASIC\_WEB\_STACK\_SIZE ( configMINIMAL\_STACK\_SIZE \* 3 )

// The OLED task uses the sprintf function so requires a little more stack too.

#define mainOLED\_TASK\_STACK\_SIZE ( configMINIMAL\_STACK\_SIZE + 50 )

// Task priorities.

#define mainQUEUE\_POLL\_PRIORITY ( tskIDLE\_PRIORITY + 2 )

#define mainCHECK\_TASK\_PRIORITY ( tskIDLE\_PRIORITY + 3 )

#define mainSEM\_TEST\_PRIORITY ( tskIDLE\_PRIORITY + 1 )

#define mainBLOCK\_Q\_PRIORITY ( tskIDLE\_PRIORITY + 2 )

#define mainCREATOR\_TASK\_PRIORITY ( tskIDLE\_PRIORITY + 3 )

#define mainINTEGER\_TASK\_PRIORITY ( tskIDLE\_PRIORITY )

#define mainGEN\_QUEUE\_TASK\_PRIORITY ( tskIDLE\_PRIORITY )

// The maximum number of messages that can be waiting for display at any one time.

#define mainOLED\_QUEUE\_SIZE ( 20 )

// Dimensions the buffer into which the jitter time is written.

#define mainMAX\_MSG\_LEN 25

/\*

The period of the system clock in nano seconds. This is used to calculate

the jitter time in nano seconds.

\*/

#define mainNS\_PER\_CLOCK ( ( unsigned portLONG ) ( ( 1.0 / ( double ) configCPU\_CLOCK\_HZ ) \* 1000000000.0 ) )

// Constants used when writing strings to the display.

#define mainCHARACTER\_HEIGHT ( 9 )

#define mainMAX\_ROWS\_128 ( mainCHARACTER\_HEIGHT \* 14 )

#define mainMAX\_ROWS\_96 ( mainCHARACTER\_HEIGHT \* 10 )

#define mainMAX\_ROWS\_64 ( mainCHARACTER\_HEIGHT \* 7 )

#define mainFULL\_SCALE ( 15 )

#define ulSSI\_FREQUENCY ( 3500000UL )

/\*-----------------------------------------------------------\*/

/\*

\* The task that handles the uIP stack. All TCP/IP processing is performed in

\* this task.

\*/

extern void vuIP\_Task( void \*pvParameters );

/\*

\* The display is written two by more than one task so is controlled by a

\* 'gatekeeper' task. This is the only task that is actually permitted to

\* access the display directly. Other tasks wanting to display a message send

\* the message to the gatekeeper.

\*/

static void vOLEDTask( void \*pvParameters );

/\*

\* Configure the hardware .

\*/

static void prvSetupHardware( void );

/\*

\* Configures the high frequency timers - those used to measure the timing

\* jitter while the real time kernel is executing.

\*/

//extern void vSetupHighFrequencyTimer( void );

/\*

\* Hook functions that can get called by the kernel.

\*/

void vApplicationStackOverflowHook( xTaskHandle \*pxTask, signed portCHAR \*pcTaskName );

void vApplicationTickHook( void );

/\*-----------------------------------------------------------\*/

/\*

The queue used to send messages to the OLED task.

\*/

xQueueHandle xOLEDQueue;

/\*-----------------------------------------------------------\*/

void vArrvingTrain(void \*train);

void vDepartingTrain(void \*train);

void vTrainCom(void \*train);

void vSwitchControl(void \*train);

void vTemperatureMeasurement(void \*train);

void vLcdDisplay(void \*train);

void vOledDisplay(void \*train);

void vLocalKeypad(void \*train);

void vSerialCom(void \*train);

void vNoiseCapture(void \*train);

void vNoiseProcessing(void \*train);

unsigned short vRemoteCom( void \*train );

void vCommand(void\* train);

xTaskHandle AT; // ArrvingTrain

xTaskHandle DT; // DepartingTrain

xTaskHandle TC; // TrainCom

xTaskHandle SC; // SwitchControl

xTaskHandle TM; // TeperatureMeasurement

xTaskHandle LD; // LcdDisplay

xTaskHandle OD; // OledDisplay

xTaskHandle LK; // LocalKeypad

xTaskHandle SC; // SerialCom

xTaskHandle NC; // NoiseCapture

xTaskHandle NP; // NoiseProcessing

xTaskHandle RC; //remote com

xTaskHandle CO; //COMMAND

int taskdelay = 500;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Please ensure to read http://www.freertos.org/portlm3sx965.html

\* which provides information on configuring and running this demo for the

\* various Luminary Micro EKs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main( void )

{

startUp();

initialize();

// |--------------------------------------------------------|

// | Schedule Task |

// |--------------------------------------------------------|

//

// this is the startup task

// set up the hardware

intersectionLock = xSemaphoreCreateCounting(1,1);

prvSetupHardware();

#if mainINCLUDE\_WEB\_SERVER != 0

{

/\*

Create the uIP task if running on a processor that includes a MAC and PHY.

\*/

if( SysCtlPeripheralPresent( SYSCTL\_PERIPH\_ETH ) )

{

xTaskCreate( vuIP\_Task, ( signed portCHAR \* ) "uIP", mainBASIC\_WEB\_STACK\_SIZE, NULL, mainCHECK\_TASK\_PRIORITY - 1, NULL );

}

}

#endif

// create the tasks

xOLEDQueue = xQueueCreate( mainOLED\_QUEUE\_SIZE, sizeof( xOLEDMessage ) );

xTaskCreate(vArrvingTrain, "ArrvingTrain", 120, &myArrivingTrain, 9, &AT); //1

xTaskCreate(vTrainCom, "TrainCom", 90, &myTrainCom, 8, &TC); //2

xTaskCreate(vDepartingTrain, "DepartingTrain", 80, &myDepartingTrain, 7, &DT);//3

xTaskCreate(vSwitchControl, "SwitchControl", 100, &mySwitchControl, 6, &SC); //4

xTaskCreate(vTemperatureMeasurement, "TemperatureMeasurement", 80, &myTemperature, 5, &TM);//8

xTaskCreate(vLocalKeypad, "LocalKeypad",80, &myLocalKeypad, 4, &LK); //7

xTaskCreate(vLcdDisplay, "LcdDisplay", 80, &myLcdDisplay, 1, &LD); //5

xTaskCreate(vOledDisplay, "OledDisplay", 170, &myOledDisplay, 1, &OD); //6

xTaskCreate(vSerialCom, "SerialCom", 300, &mySerialCom, 3, &SC); //11

xTaskCreate(vNoiseCapture, "NoiseCapture", 80, &myNoiseCapture, 1, &NC); //9

xTaskCreate(vNoiseProcessing, "NoiseProcessing", 620, &myNoiseProcessing, 1, &NP); //10

xTaskCreate(vCommand, "Command", 90, &myCommand, 1, &CO);

vTaskStartScheduler();

}

void vApplicationStackOverflowHook( xTaskHandle \*pxTask, signed portCHAR \*pcTaskName )

{

( void ) pxTask;

( void ) pcTaskName;

while( 1 );

}

/\*-----------------------------------------------------------\*/

void prvSetupHardware( void )

{

/\*

If running on Rev A2 silicon, turn the LDO voltage up to 2.75V. This is

a workaround to allow the PLL to operate reliably.

\*/

if( DEVICE\_IS\_REVA2 )

{

SysCtlLDOSet( SYSCTL\_LDO\_2\_75V );

}

// Set the clocking to run from the PLL at 50 MHz

SysCtlClockSet( SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_8MHZ );

/\*

Enable Port F for Ethernet LEDs

LED0 Bit 3 Output

LED1 Bit 2 Output

\*/

SysCtlPeripheralEnable( SYSCTL\_PERIPH\_GPIOF );

GPIODirModeSet( GPIO\_PORTF\_BASE, (GPIO\_PIN\_2 | GPIO\_PIN\_3), GPIO\_DIR\_MODE\_HW );

GPIOPadConfigSet( GPIO\_PORTF\_BASE, (GPIO\_PIN\_2 | GPIO\_PIN\_3 ), GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD );

}

/\*-----------------------------------------------------------\*/

void vApplicationTickHook( void )

{

static xOLEDMessage xMessage = { "PASS" };

static unsigned portLONG ulTicksSinceLastDisplay = 0;

portBASE\_TYPE xHigherPriorityTaskWoken = pdFALSE;

/\*

Called from every tick interrupt. Have enough ticks passed to make it

time to perform our health status check again?

\*/

ulTicksSinceLastDisplay++;

if( ulTicksSinceLastDisplay >= mainCHECK\_DELAY )

{

ulTicksSinceLastDisplay = 0;

}

}

void startUp()

{

SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_8MHZ);

// Initialize the OLED display.

RIT128x96x4Init(1000000);

//

// Enable the peripherals used by this example.

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOC);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOG);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Timer0 Setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// // Set the clocking to run directly from the crystal

//

// SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

//

// //

// // Configure the two 32-bit periodic timers.

// //

// TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

//

// TimerLoadSet(TIMER0\_BASE, TIMER\_A, SysCtlClockGet() / 2);

//

//

// //

// // Setup the interrupts for the timer timeouts.

// //

// IntEnable(INT\_TIMER0A);

//

// TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

//

//

// //

// // Enable the timers.

// //

// TimerEnable(TIMER0\_BASE, TIMER\_A);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Timer1 Setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set the clocking to run directly from the crysta

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

//

// Configure the two 32-bit periodic timers.

//

TimerConfigure(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

TimerLoadSet(TIMER1\_BASE, TIMER\_A, 2 \* SysCtlClockGet());

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Timer2 Setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set the clocking to run directly from the crysta

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER2);

//

// Configure the two 32-bit periodic timers.

//

TimerConfigure(TIMER2\_BASE, TIMER\_CFG\_PERIODIC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Timer3 Setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set the clocking to run directly from the crysta

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER3);

//

// Configure the two 32-bit periodic timers.

//

TimerConfigure(TIMER3\_BASE, TIMER\_CFG\_PERIODIC);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ISR sw2 setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOInput(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_STRENGTH\_2MA,

GPIO\_PIN\_TYPE\_STD\_WPU);

GPIOIntTypeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_FALLING\_EDGE);

GPIOPinIntEnable(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

IntEnable(INT\_GPIOF);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ISR sw3 setup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);

GPIOPinTypeGPIOInput(GPIO\_PORTE\_BASE, GPIO\_PIN\_0);

GPIOPadConfigSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_0, GPIO\_STRENGTH\_2MA,

GPIO\_PIN\_TYPE\_STD\_WPU);

GPIOIntTypeSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);

GPIOPinIntEnable(GPIO\_PORTE\_BASE, GPIO\_PIN\_0);

IntEnable(INT\_GPIOE);

// |--------------------------------------------------------|

// | pwmgen.c - PWM signal generation example. |

// | |

// | Copyright (c) 2005-2012 Texas Instruments Incorporated.|

// | All rights reserved. |

// |--------------------------------------------------------|

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM0);

SysCtlPWMClockSet(SYSCTL\_PWMDIV\_1);

unsigned long ulPeriod;

//

// Set GPIO F0 and G1 as PWM pins. They are used to output the PWM0 and

// PWM1 signals.

//

GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinTypePWM(GPIO\_PORTG\_BASE, GPIO\_PIN\_1);

//

// Compute the PWM period based on the system clock.

//

ulPeriod = SysCtlClockGet() / 70;/////////////////////////////////////////////////sound turned off

//

// Set the PWM period to 100 (A) Hz.

//

PWMGenConfigure(PWM0\_BASE, PWM\_GEN\_0,

PWM\_GEN\_MODE\_UP\_DOWN | PWM\_GEN\_MODE\_NO\_SYNC);

PWMGenPeriodSet(PWM0\_BASE, PWM\_GEN\_0, ulPeriod);

//

// Set PWM0 to a duty cycle of 25% and PWM1 to a duty cycle of 75%.

//

PWMPulseWidthSet(PWM0\_BASE, PWM\_OUT\_0, ulPeriod / 4);

PWMPulseWidthSet(PWM0\_BASE, PWM\_OUT\_1, ulPeriod \* 3 / 4);

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

//===========================================================

// LCD Setup

//===========================================================

//LCD display control

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_7); //d7

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_6); //d6

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_5); //d5

GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_4); //d4

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_BASE, GPIO\_PIN\_7); //e

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_BASE, GPIO\_PIN\_6); //rs

// Initialize the LCD display.

delay(10000);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); // RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7 n

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6 f

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7 n

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6 f

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

//on/off

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6 d

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5 c

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4 b

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

lcdClear();

///===========================================================

// HYPERTERM setup (UTAR)

//============================================================

//

// Enable the peripherals used by this example.

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//

// Enable processor interrupts.

//

IntMasterEnable();

//

// Set GPIO A0 and A1 as UART pins.

//

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//

// Configure the UART for 115,200, 8-N-1 operation.

//

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |

UART\_CONFIG\_PAR\_NONE));

//

// Enable the UART interrupt.

//

IntEnable(INT\_UART0);

UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

//============================================================

// KEYPAD setup

//============================================================

GPIOPinTypeGPIOInput(GPIO\_PORTA\_BASE, GPIO\_PIN\_4); // home -- 0

GPIOPinTypeGPIOInput(GPIO\_PORTB\_BASE, GPIO\_PIN\_4); // scroll --1

GPIOPinTypeGPIOInput(GPIO\_PORTB\_BASE, GPIO\_PIN\_5); // select -- 2

//initialize

GPIOPadConfigSet(GPIO\_PORTA\_BASE, GPIO\_PIN\_4, GPIO\_STRENGTH\_2MA,

GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_4, GPIO\_STRENGTH\_2MA,

GPIO\_PIN\_TYPE\_STD\_WPD);

GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_5, GPIO\_STRENGTH\_2MA,

GPIO\_PIN\_TYPE\_STD\_WPD);

//============================================================

// Arriving train setup

//============================================================

GPIOPinTypeGPIOOutput(GPIO\_PORTD\_BASE, GPIO\_PIN\_4); // output frequency

GPIOPinTypeGPIOInput(GPIO\_PORTD\_BASE, GPIO\_PIN\_5); // wireless input

//=============================================================

//ADC SETUP for tempture

//=============================================================

//

// Enable the first sample sequencer to capture the value of channel 0 when

// the processor trigger occurs.

//

//ADCSequenceDisable(ADC0\_BASE, 3);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

ADCSequenceConfigure(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceConfigure(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceConfigure(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceStepConfigure(ADC0\_BASE, 0, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH0);

ADCSequenceStepConfigure(ADC0\_BASE, 1, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH1);

ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH2);

ADCSequenceEnable(ADC0\_BASE, 0);

ADCSequenceEnable(ADC0\_BASE, 1);

ADCSequenceEnable(ADC0\_BASE, 2);

//for noise

ADCSequenceConfigure(ADC0\_BASE, 3, ADC\_TRIGGER\_PROCESSOR, 0);

ADCSequenceStepConfigure(ADC0\_BASE, 3, 0, ADC\_CTL\_IE | ADC\_CTL\_END | ADC\_CTL\_CH3);

ADCSequenceEnable(ADC0\_BASE, 3);

return;

}

void initialize()

{

// |--------------------------------------------------------|

// | Build Task Queue |

// |--------------------------------------------------------|

// declear variables and set pointers

arrivingTD.taskDataPtr = &myArrivingTrain;

myArrivingTrain.trainArriving = &trainArriving;

myArrivingTrain.checkTrain = &checkTrain;

myArrivingTrain.arrivalDirection = &arrivalDirection;

myArrivingTrain.departureDirection = &departureDirection;

myArrivingTrain.globalCount = &globalCount;

myArrivingTrain.arrivalTime = &arrivalTime;

myArrivingTrain.arrivingTrainDistance = arrivingTrainDistance;

myArrivingTrain.departingTrainFlag = &departingTrainFlag;

myArrivingTrain.trainPresent = &trainPresent;

trainCD.taskDataPtr = &myTrainCom;

myTrainCom.trainArriving = &trainArriving;

myTrainCom.departingTrainFlag = &departingTrainFlag;

myTrainCom.trainSize = &trainSize;

myTrainCom.arrivalDirection = &arrivalDirection;

myTrainCom.departureDirection = &departureDirection;

myTrainCom.checkTrain = &checkTrain;

myTrainCom.globalCount = &globalCount;

myTrainCom.trainDeparting = trainDeparting;

myTrainCom.trainPresent = &trainPresent;

departingTD.taskDataPtr = &myDepartingTrain;

myDepartingTrain.departingTrainFlag = &departingTrainFlag;

myDepartingTrain.departureDirection = &departureDirection;

myDepartingTrain.globalCount = &globalCount;

myDepartingTrain.trainDeparting = trainDeparting;

myDepartingTrain.trainPresent = &trainPresent;

SWCD.taskDataPtr = &mySwitchControl;

mySwitchControl.departureDirection = &departureDirection;

mySwitchControl.departingTrainFlag = &departingTrainFlag;

mySwitchControl.trainSize = &trainSize;

mySwitchControl.interLock = &interLock;

mySwitchControl.traversalTime = &traversalTime;

mySwitchControl.gridlock = &gridlock;

mySwitchControl.globalCount = &globalCount;

mySwitchControl.trainDeparting = trainDeparting;

mySwitchControl.trainPresent = &trainPresent;

LCDD.taskDataPtr = &myLcdDisplay;

myLcdDisplay.departingTrainFlag = &departingTrainFlag;

myLcdDisplay.trainSize = &trainSize;

myLcdDisplay.traversalTime = &traversalTime;

myLcdDisplay.departureDirection = &departureDirection;

myLcdDisplay.interLock = &interLock;

myLcdDisplay.gridlock = &gridlock;

myLcdDisplay.globalCount = &globalCount;

myLcdDisplay.trainPresent = &trainPresent;

OLEDD.taskDataPtr = &myOledDisplay;

myOledDisplay.departingTrainFlag = &departingTrainFlag;

myOledDisplay.trainSize = &trainSize;

myOledDisplay.interLock = &interLock;

myOledDisplay.traversalTime = &traversalTime;

myOledDisplay.trainArriving = &trainArriving;

myOledDisplay.arrivalDirection = &arrivalDirection;

myOledDisplay.globalCount = &globalCount;

myOledDisplay.departureDirection = &departureDirection;

myOledDisplay.mode = &mode;

myOledDisplay.scroll = &scroll;

myOledDisplay.select = &select;

myOledDisplay.statusSelection = &statusSelection;

myOledDisplay.gridlock = &gridlock;

myOledDisplay.arrivingTrainDistance = arrivingTrainDistance;

myOledDisplay.temperatureBuf= temperatureBuf;

myOledDisplay.trainDeparting = trainDeparting;

myOledDisplay.trainPresent = &trainPresent;

keypadD.taskDataPtr = &myLocalKeypad;

myLocalKeypad.mode = &mode;

myLocalKeypad.statusSelection = &statusSelection;

myLocalKeypad.scroll = &scroll;

myLocalKeypad.select = &select;

myLocalKeypad.annunciation = &annunciation;

myLocalKeypad.globalCount = &globalCount;

serialCD.taskDataPtr = &mySerialCom;

mySerialCom.departingTrainFlag = &departingTrainFlag;

mySerialCom.trainSize = &trainSize;

mySerialCom.traversalTime = &traversalTime;

mySerialCom.departureDirection = &departureDirection;

mySerialCom.interLock = &interLock;

mySerialCom.gridlock = &gridlock;

mySerialCom.trainArriving = &trainArriving;

mySerialCom.globalCount = &globalCount;

mySerialCom.arrivalTime = &arrivalTime;

mySerialCom.arrivalDirection = &arrivalDirection;

mySerialCom.trainDeparting = trainDeparting;

mySerialCom.trainPresent = &trainPresent;

temperatureD.taskDataPtr = &myTemperature;

myTemperature.departingTrainFlag = &departingTrainFlag;

myTemperature.temperatureBuf = temperatureBuf;

myTemperature.globalStartMeasure = &globalStartMeasure;

noiseCD.taskDataPtr = &myNoiseCapture;

myNoiseCapture.noiseCaptureBuf = &noiseCaptureBuf;

myNoiseCapture.globalStartMeasure = &globalStartMeasure;

noisePD.taskDataPtr = &myNoiseProcessing;

myNoiseProcessing.noiseCaptureBuf = &noiseCaptureBuf;

myNoiseProcessing.noiseTransBuf = &noiseTransBuf;

myNoiseProcessing.noiseFrequency = &noiseFrequency;

ramoteCD.taskDataPtr = &myRemote;

myRemote.departingTrainFlag = &departingTrainFlag;

myRemote.trainSize = &trainSize;

myRemote.traversalTime = &traversalTime;

myRemote.departureDirection = &departureDirection;

myRemote.arrivalDirection = &arrivalDirection;

myRemote.trainArriving = &trainArriving;

myRemote.arrivalTime = &arrivalTime;

myRemote.gridlock = &gridlock;

myRemote.temperatureBuf = temperatureBuf;

myRemote.trainPresent = &trainPresent;

myRemote.arrivingTrainDistance = arrivingTrainDistance;

myRemote.noiseTransBuf = noiseTransBuf;

myRemote.showRecentData = &showRecentData;

commandD.taskDataPtr = &myCommand;

myCommand.readCommand = readCommand;

myCommand.globalStartMeasure = &globalStartMeasure;

myCommand.showRecentData = &showRecentData;

return;

}

void vArrvingTrain(void\* train)

{

while (1)

{

//task1 = uxTaskGetStackHighWaterMark(NULL);

arrivingTrainData \*arrivingT = (arrivingTrainData\*)train;

static Bool firstTime = TRUE; // the train first time appeared(away from intersection)

int randD0;

int randD1;

int randN;

static int count = 0;

static int prevDirection = 0;

static double speed;

//vCommand(train);

//RIT128x96x4StringDraw(readCommand, 80, 20, 15);

// press the switch then a new train arriving

if (\*(arrivingT->trainArriving) > 0) {

count ++;

if(firstTime)

{

randD0 = randomInteger(0, 1); // random number d;

randD1 = randomInteger(0, 1);

while (prevDirection == randD0\*2 + randD1) {

randD0 = randomInteger(0, 1);

randD1 = randomInteger(0, 1);

}

randN = randomInteger(1,3); // random number n, cannot be negative

\*(arrivingT->arrivalDirection) = randD0\*2 + randD1;

prevDirection = randD0 \* 2 + randD1;

firstTime = FALSE;

\*(arrivingT->arrivalTime) = 24 \* randN;

speed = 2000.0 /(double) \*(arrivingT->arrivalTime);

\*(arrivingT->arrivingTrainDistance) = 1000;

for (int i = 1; i < 7; i ++)

{

\*(arrivingT->arrivingTrainDistance + i) = 0;

}

}

// generate frequency within 1km

if(1900 > speed \* count \* 0.5 && speed \* count \* 0.5 > 1000)

{

IntEnable(INT\_TIMER2A);

TimerLoadSet(TIMER2\_BASE, TIMER\_A, (unsigned long)(SysCtlClockGet() / (2000000 / ( 53 \* (2000 - speed \* count \* 0.5) - 5175))));

TimerIntEnable(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerEnable(TIMER2\_BASE, TIMER\_A);

char dis[80];

sprintf(dis, "%d", (\*arrivingT->arrivingTrainDistance));

if(globalStartDisplay)

{

if((\*arrivingT->arrivingTrainDistance)>999 )

{

RIT128x96x4StringDraw(dis, 102, 73, 15);

} else if ((\*arrivingT->arrivingTrainDistance)>99 )

{

RIT128x96x4StringDraw(" ", 102, 73, 15);

RIT128x96x4StringDraw(dis, 108, 73, 15);

} else if ((\*arrivingT->arrivingTrainDistance)>9 )

{

RIT128x96x4StringDraw(" ", 102, 73, 15);

RIT128x96x4StringDraw(dis, 114, 73, 15);

} else {

RIT128x96x4StringDraw(" ", 102, 73, 15);

RIT128x96x4StringDraw(dis, 120, 73, 15);

}

}

} else if (speed \* count \* 0.5 > 1900 && globalStartDisplay){

RIT128x96x4StringDraw("100", 108, 73, 15);

}

// receive frequency

// active timer3

IntEnable(INT\_TIMER3A);

TimerLoadSet(TIMER3\_BASE, TIMER\_A, SysCtlClockGet() / 16000);

TimerIntEnable(TIMER3\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerEnable(TIMER3\_BASE, TIMER\_A);

if (remainDistance < 0.9 \* (\*(arrivingT->arrivingTrainDistance)))

{

for (int i = 7 ; i > 0; i -- )

{

\*(arrivingT->arrivingTrainDistance+i) = \*(arrivingT->arrivingTrainDistance+(i-1));

}

\*(arrivingT->arrivingTrainDistance) = remainDistance;

}

switch (\*(arrivingT->arrivalDirection)) {

case 0:

switch (\*(arrivingT->globalCount) % 6) {

case 0:

case 1:

case 2:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("North Arriving", 15, 73, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 13, 73, 15);

break;

}

switch (\*(arrivingT->globalCount) % 20) {

case 0:

case 1:

case 2:

case 3:

case 6:

case 7:

case 8:

case 9:

case 12:

case 13:

case 16:

case 17:

if(!\*(arrivingT->departingTrainFlag))// blast priorityb

{

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

default:

if(!\*(arrivingT->departingTrainFlag))// blast priorityb

{

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

}

break;

case 1:

switch (\*(arrivingT->globalCount) % 6) {

case 0:

case 1:

case 2:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("South Arriving", 15, 73, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 13, 73, 15);

break;

}

switch (\*(arrivingT->globalCount) % 28) {

case 0:

case 1:

case 2:

case 3:

case 6:

case 7:

case 8:

case 9:

case 12:

case 13:

case 14:

case 15:

case 16:

case 17:

case 20:

case 21:

case 22:

case 23:

case 24:

case 25:

if(!\*(arrivingT->departingTrainFlag))

{

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

default:

if(!\*(arrivingT->departingTrainFlag))// blast priorityb

{

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

}

break;

case 2:

switch (\*(arrivingT->globalCount) % 8) {

case 0:

case 1:

case 2:

case 3:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("East Arriving", 15, 73, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 13, 73, 15);

break;

}

switch (\*(arrivingT->globalCount) % 26) {

case 0:

case 1:

case 2:

case 3:

case 6:

case 7:

case 8:

case 9:

case 12:

case 13:

case 14:

case 15:

case 18:

case 19:

case 22:

case 23:

if(!\*(arrivingT->departingTrainFlag))

{

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

default:

if(!\*(arrivingT->departingTrainFlag))// blast priorityb

{

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

}

break;

case 3:

switch (\*(arrivingT->globalCount) % 4) {

case 0:

case 1:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("West Arriving", 15, 73, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 13, 73, 15);

break;

}

switch (\*(arrivingT->globalCount) % 14) {

case 0:

case 1:

case 2:

case 3:

case 6:

case 7:

case 10:

case 11:

if(!\*(arrivingT->departingTrainFlag))

{

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

default:

if(!\*(arrivingT->departingTrainFlag))// blast priority

{

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

}

break;

}

break;

}

if (\*(arrivingT->arrivingTrainDistance) < 400)

{

\*(arrivingT->checkTrain) = TRUE;

vTaskResume(TC);

}

if (0 == count\*0.5 - \*(arrivingT->arrivalTime))

{

firstTime = TRUE;

tcGo = TRUE;

count = 0;

\*(arrivingT->arrivalTime) = 0;

TimerIntDisable(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerDisable(TIMER2\_BASE, TIMER\_A);

TimerIntDisable(TIMER3\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerDisable(TIMER3\_BASE, TIMER\_A);

remainDistance = 1000;

\*(arrivingT->trainArriving) = \*(arrivingT->trainArriving) - 1;

\*(arrivingT->trainPresent) = \*(arrivingT->trainPresent) + 1;

\*(arrivingT->arrivingTrainDistance) = 0;

RIT128x96x4StringDraw(" ", 108, 73, 15);

RIT128x96x4StringDraw(" ", 13, 73, 15);

}

} else {

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_4, 0x00);

}

vTaskDelay(taskdelay \* 2);

}

return;

}

// |--------------------------------------------------------|

// | trainCom task |

// |--------------------------------------------------------|

void vTrainCom(void\* train)

{

while (1)

{

//task2 = uxTaskGetStackHighWaterMark(NULL);

// re-cast the task argument as pointer to the data structure type

trainComData \*tCom = (trainComData\*)train;

if (tcGo && \*(tCom->checkTrain)){

// generate random direction and train size

int randDirc;

tcGo = FALSE;

randDirc = randomInteger(0,3);

while ((\*(tCom->arrivalDirection) == randDirc) ){

randDirc = randomInteger(0,3);

}

static int randSize;

randSize = randomInteger(1,9);

int index = 0;

while (\*(tCom->trainDeparting + (index + 1) )!= 0)

{

index = index + 2;

}

\*(tCom->trainDeparting + index) = randDirc;

\*(tCom->trainDeparting + (index + 1)) = randSize;

\*(tCom->checkTrain) = FALSE;

vTaskSuspend(TC);

}

vTaskDelay(taskdelay);

}

return;

}

void vDepartingTrain(void\* train)

{

while (1)

{

//task3 = uxTaskGetStackHighWaterMark(NULL);

static int countD = 0;

departingTrainData \*departingT = (departingTrainData\*)train;

if (\*(departingT->departingTrainFlag)) {

switch (\*(departingT->departureDirection)) {

case 0:

switch (countD % 6) {

case 0:

case 1:

case 2:

if (globalStartDisplay){

RIT128x96x4StringDraw("North Train Departing", 0, 85, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 0, 85, 15);

break;

}

switch (countD % 14) {

case 0:

case 1:

case 2:

case 3:

case 6:

case 7:

case 10:

case 11:

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

default:

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

}

countD++;

break;

case 1:

switch ( countD % 6) {

case 0:

case 1:

case 2:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("South Train Departing", 0, 85, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 0, 85, 15);

break;

}

switch (countD % 16) {

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 8:

case 9:

case 12:

case 13:

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

default:

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

}

countD++;

break;

case 2:

switch (countD % 8) {

case 0:

case 1:

case 2:

case 3:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("East Train Departing", 0, 85, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 0, 85, 15);

break;

}

switch (countD % 18) {

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 10:

case 11:

case 14:

case 15:

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

default:

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

}

countD++;

break;

case 3:

switch (countD % 4) {

case 0:

case 1:

if (globalStartDisplay)

{

RIT128x96x4StringDraw("West Train Departing", 0, 85, 15);

}

break;

default:

RIT128x96x4StringDraw(" ", 0, 85, 15);

break;

}

switch (countD % 20) {

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

case 9:

case 12:

case 13:

case 16:

case 17:

// Enable the PWM0 and PWM1 output signals.

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, true);

// Enable the PWM generator.

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

default:

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

break;

}

countD++;

break;

}

}

vTaskDelay(taskdelay);

}

return;

}

// |--------------------------------------------------------|

// | switchControl task |

// |--------------------------------------------------------|

void vSwitchControl(void\* train)

{

while(1)

{

//task4 = uxTaskGetStackHighWaterMark(NULL);

globalCount++;

switchControlData \*SWData = (switchControlData\*)train;

static double totalTime = 0;

static int n = 0;// for gridlock

static int m = 0;// for gridlock burst

static double delayValue = 0;

static double delayTime = 0;

//static Bool waited = FALSE; //never waited for the gridlock before

//static Bool firstTime = TRUE;

if(\*(SWData->trainPresent) > 0){ //waiting at intersection and gridlock

totalTime += 0.5;

if (\*(SWData->trainSize) == 0){ // \*(SWData->trainSize)==0

n = randomInteger(-2,2);

\*(SWData->departingTrainFlag) = TRUE;

\*(SWData->departureDirection) = \*(SWData->trainDeparting);

\*(SWData->trainSize) = \*(SWData->trainDeparting + 1);

\*(SWData->traversalTime) = \*(SWData->trainSize) \* 6;

if(n<0){

\*(SWData->gridlock) = TRUE;

delayValue = -12 \* n ; // 0.2 \* n minutes

delayTime = delayValue;

m = randomInteger(-2,2); // burst

}

if (m > 0) // burst success

{

delayValue = 3;

burstFail = FALSE;

m = 0;

} else if (m < 0){ // brust fail, use nuclear weapon

delayValue = 6;

burstFail = TRUE;

m = 0;

}

}

if(\*(SWData->gridlock))

{

delayValue -= 0.5; // waited for one minor cycle

if(0 == delayValue)

{

\*(SWData->gridlock) = FALSE;

}

} else // train go

{

xSemaphoreGive(intersectionLock);

\*(SWData->interLock) = ON;

// if train passed, reset all data

if (totalTime > (\*(SWData->traversalTime) + delayTime))

{

\*(SWData->departingTrainFlag) = FALSE;

\*(SWData->interLock) = OFF;

\*(SWData->gridlock) = FALSE;

\*(SWData->traversalTime) = 0;

\*(SWData->trainSize) = 0;

totalTime = 0;

delayTime = 0;

delayValue = 0;

\*(SWData->trainPresent) = \*(SWData->trainPresent) - 1;

xSemaphoreTake(intersectionLock, 10);

// stop the sound

PWMOutputState(PWM0\_BASE, PWM\_OUT\_0\_BIT | PWM\_OUT\_1\_BIT, false);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

RIT128x96x4StringDraw(" ", 0, 85, 15);

for (int i = 2; i < 15; i+=2){

\*(SWData->trainDeparting + i - 2) = \*(SWData->trainDeparting + i);

\*(SWData->trainDeparting + i - 1) = \*(SWData->trainDeparting + i + 1);

}

\*(SWData->trainDeparting + 14) = 0;

\*(SWData->trainDeparting + 15) = 0;

}

}

}

vTaskDelay(taskdelay);

}

return;

}

void vLcdDisplay(void\* train)

{

while(1)

{

//task5 = uxTaskGetStackHighWaterMark(NULL);

lcdDisplayData \*lcdData = (lcdDisplayData\*)train;

static int move = 0;

if (\*(lcdData->trainPresent) > 0 && gridlock)

{

if(burstFail)

{

printBurst();

printX();

for(int i = 0; i < (27 + move); i++)

{ //shift to next line

printSpace();

}

printNuclear();

for(int i = 0; i < (28 - move); i++){

printSpace();

}

move += 2;

} else if(!burstFail)

{

printBurst();

printO();

for(int i = 0; i < 67; i++)

{ //shift to next line

printSpace();

}

}

} else if(\*(lcdData->trainPresent) > 0 && \*(lcdData->departingTrainFlag))

{

move = 0;

// first line

printSpace();

printNumber(\*(lcdData->trainPresent)); //1

printSpace(); //1

switch(\*(lcdData->departureDirection)){ //1

case 0:

printNorth();

break;

case 1:

printSouth();

break;

case 2:

printEast();

break;

case 3:

printWest();

break;

}

printSpace(); //1

printinterLock(); //2

if (\*(lcdData->interLock) == ON) //4

{

printOn();

printSpace();

printSpace();

}else

{

printOff();

printSpace();

}

printGridLock(); //2

if (\*(lcdData->gridlock)) //4

{

printOn();

printSpace();

printSpace();

}else

{

printOff();

printSpace();

}

for(int i = 0; i < 23; i++){ //shift to next line

printSpace();

}

// Second line

printSpace(); //1

printSize(); //5

printNumber(\*(lcdData->trainSize)); //1

printSpace(); //1

printTime(); //5

printNumber((\*(lcdData->traversalTime)/10)%10);//1

printNumber(\*(lcdData->traversalTime)%10);//1

for(int i = 0; i < 25; i++){

printSpace();

}

} else //if (!\*(lcdData->departingTrainFlag))

{

lcdClear();

}

vTaskDelay(taskdelay \* 3);

}

return;

}

void printOn()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printOff()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printGridLock()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printinterLock()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printNorth()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printSouth()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printWest()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printEast()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printSize()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printTime()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void lcdClear()

{

//clear

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void lcdSR()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7 S/C

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6 L/R

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0x00); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printSpace()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printNumber(short number)

{

switch(number) {

case 0:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 1:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 2:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 3:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 4:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 5:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 6:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 7:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 8:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

case 9:

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

break;

}

}

void printBurst()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printO()

{

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20);

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printX()

{

delay(20); //F

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //A

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //I

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //L

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //U

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //R

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //E

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

void printNuclear()

{

delay(20); //> 0011 1110

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //= 0011 1101

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //( 0010 1000

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //N 0100 1110

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //U 0101 0101

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //C

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //L

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //E

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //A

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //R

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); // )

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0x00); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

delay(20); //>

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0x00); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0x00); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0xFF); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7, 0xFF); //7

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_6, 0xFF); //6

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_5, 0xFF); //5

GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4, 0x00); //4

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_6, 0xFF); //RS

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0xFF); //e

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7, 0x00); //e

}

// |--------------------------------------------------------|

// | OLEDDisplay task |

// |--------------------------------------------------------|

void vOledDisplay(void \*train)

{

while(1)

{

//task6 = uxTaskGetStackHighWaterMark(NULL);

// re-cast the task argument as pointer to the data structure type

oledDisplayData \*disData = (oledDisplayData\*)train;

if(OLEDChanged && globalStartDisplay)

{

// Screen 1 (home)

switch (\*disData->mode)

{

case 0: //home

RIT128x96x4StringDraw("Status", 30, 12, 15);

RIT128x96x4StringDraw("Annunciation", 30, 48, 15);

switch ((\*disData->scroll) % 2)

{

case 0:

RIT128x96x4StringDraw(">>", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

break;

case 1:

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(">>", 15, 48, 15);

break;

}

break;

case 1: //status

switch (\*disData->statusSelection)

{

case 0:

RIT128x96x4StringDraw("North Train", 30, 12, 15);

RIT128x96x4StringDraw("South Train", 30, 24, 15);

RIT128x96x4StringDraw("East Train", 30, 36, 15);

RIT128x96x4StringDraw("West Train ", 30, 48, 15);

switch ((\*disData->scroll) % 4)

{

case 0:

RIT128x96x4StringDraw(">>", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 24, 15);

RIT128x96x4StringDraw(" ", 15, 36, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

break;

case 1:

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(">>", 15, 24, 15);

RIT128x96x4StringDraw(" ", 15, 36, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

break;

case 2:

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 24, 15);

RIT128x96x4StringDraw(">>", 15, 36, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

break;

case 3:

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 24, 15);

RIT128x96x4StringDraw(" ", 15, 36, 15);

RIT128x96x4StringDraw(">>", 15, 48, 15);

break;

}//end cursor

break;

case 1:

if(\*disData->arrivalDirection == 0)

{

RIT128x96x4StringDraw("North Arriving ", 15, 0, 15);

RIT128x96x4StringDraw("Arriving No: ", 15, 12, 15);

char TrainA0[2];

char TrainA1[2];

TrainA0[0] = (int)(\*(disData->trainArriving)/10 )+ '0';

TrainA0[1] = '\0';

TrainA1[0] = (int)(\*(disData->trainArriving)%10) + '0';

TrainA1[1] = '\0';

RIT128x96x4StringDraw(TrainA0, 90, 12, 15);

RIT128x96x4StringDraw(TrainA1, 96, 12, 15);

if (!\*(disData->departingTrainFlag))

{

RIT128x96x4StringDraw("No Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 0)

{

RIT128x96x4StringDraw("North Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 1)

{

RIT128x96x4StringDraw("South Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 2)

{

RIT128x96x4StringDraw("East Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 3)

{

RIT128x96x4StringDraw("West Departing ", 15, 24, 15);

}

RIT128x96x4StringDraw("Train Present: ", 15, 36, 15);

if(\*(disData->trainPresent) > 0){

RIT128x96x4StringDraw("Yes", 100, 36, 15);

} else{

RIT128x96x4StringDraw("No ", 100, 36, 15);

}

RIT128x96x4StringDraw("Lock: ", 15, 48, 15);

if(\*(disData->interLock) == ON)

{

RIT128x96x4StringDraw("On", 46, 48, 15);

}

RIT128x96x4StringDraw("TSize: ", 15, 60, 15);

char TrainSZ[2];

TrainSZ[0] = \*(disData->trainSize) + '0';

TrainSZ[1] = '\0';

RIT128x96x4StringDraw(TrainSZ, 52, 60, 15);

RIT128x96x4StringDraw("TTime: ", 70, 60, 15);

char TTime0[2];

char TTime1[2];

TTime0[0] = (int)(\*(disData->traversalTime)/10) + '0';

TTime0[1] = '\0';

TTime1[0] = (int)(\*(disData->traversalTime)%10) + '0';

TTime1[1] = '\0';

RIT128x96x4StringDraw(TTime0, 106, 60, 15);

RIT128x96x4StringDraw(TTime1, 112, 60, 15);

}

else

{

RIT128x96x4StringDraw(" ", 0, 0, 15);

RIT128x96x4StringDraw(" ", 0, 12, 15);

RIT128x96x4StringDraw(" ", 0, 48, 15);

RIT128x96x4StringDraw(" ", 0, 60, 15);

RIT128x96x4StringDraw(" No Train Arrive ", 0, 24, 15);

RIT128x96x4StringDraw(" From North ", 0, 36, 15);

}

break;

case 2:

if(\*disData->arrivalDirection == 1)

{

RIT128x96x4StringDraw("South Arriving ", 15, 0, 15);

RIT128x96x4StringDraw("Arriving No: ", 15, 12, 15);

char TrainA0[2];

char TrainA1[2];

TrainA0[0] = (int)(\*(disData->trainArriving)/10 )+ '0';

TrainA0[1] = '\0';

TrainA1[0] = (int)(\*(disData->trainArriving)%10) + '0';

TrainA1[1] = '\0';

RIT128x96x4StringDraw(TrainA0, 90, 12, 15);

RIT128x96x4StringDraw(TrainA1, 96, 12, 15);

if (!\*(disData->departingTrainFlag))

{

RIT128x96x4StringDraw("No Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 0)

{

RIT128x96x4StringDraw("North Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 1)

{

RIT128x96x4StringDraw("South Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 2)

{

RIT128x96x4StringDraw("East Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 3)

{

RIT128x96x4StringDraw("West Departing ", 15, 24, 15);

}

RIT128x96x4StringDraw("Train Present: ", 15, 36, 15);

if(\*(disData->trainPresent) > 0){

RIT128x96x4StringDraw("Yes", 100, 36, 15);

} else{

RIT128x96x4StringDraw("No ", 100, 36, 15);

}

RIT128x96x4StringDraw("Lock: ", 15, 48, 15);

if(\*(disData->interLock) == ON)

{

RIT128x96x4StringDraw("On", 46, 48, 15);

}

RIT128x96x4StringDraw("TSize: ", 15, 60, 15);

char TrainSZ[2];

TrainSZ[0] = \*(disData->trainSize) + '0';

TrainSZ[1] = '\0';

RIT128x96x4StringDraw(TrainSZ, 52, 60, 15);

RIT128x96x4StringDraw("TTime: ", 70, 60, 15);

char TTime0[2];

char TTime1[2];

TTime0[0] = (int)(\*(disData->traversalTime)/10) + '0';

TTime0[1] = '\0';

TTime1[0] = (int)(\*(disData->traversalTime)%10) + '0';

TTime1[1] = '\0';

RIT128x96x4StringDraw(TTime0, 106, 60, 15);

RIT128x96x4StringDraw(TTime1, 112, 60, 15);

}

else

{

RIT128x96x4StringDraw(" ", 0, 0, 15);

RIT128x96x4StringDraw(" ", 0, 12, 15);

RIT128x96x4StringDraw(" ", 0, 48, 15);

RIT128x96x4StringDraw(" ", 0, 60, 15);

RIT128x96x4StringDraw(" No Train Arrive ", 0, 24, 15);

RIT128x96x4StringDraw(" From South ", 0, 36, 15);

}

break;

case 3:

if(\*disData->arrivalDirection == 2)

{

RIT128x96x4StringDraw("East Arriving ", 15, 0, 15);

RIT128x96x4StringDraw("Arriving No: ", 15, 12, 15);

char TrainA0[2];

char TrainA1[2];

TrainA0[0] = (int)(\*(disData->trainArriving)/10 )+ '0';

TrainA0[1] = '\0';

TrainA1[0] = (int)(\*(disData->trainArriving)%10) + '0';

TrainA1[1] = '\0';

RIT128x96x4StringDraw(TrainA0, 90, 12, 15);

RIT128x96x4StringDraw(TrainA1, 96, 12, 15);

if (!\*(disData->departingTrainFlag))

{

RIT128x96x4StringDraw("No Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 0)

{

RIT128x96x4StringDraw("North Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 1)

{

RIT128x96x4StringDraw("South Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 2)

{

RIT128x96x4StringDraw("East Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 3)

{

RIT128x96x4StringDraw("West Departing ", 15, 24, 15);

}

RIT128x96x4StringDraw("Train Present: ", 15, 36, 15);

if(\*(disData->trainPresent) > 0){

RIT128x96x4StringDraw("Yes", 100, 36, 15);

} else{

RIT128x96x4StringDraw("No ", 100, 36, 15);

}

RIT128x96x4StringDraw("Lock: ", 15, 48, 15);

if(\*(disData->interLock) == ON)

{

RIT128x96x4StringDraw("On", 46, 48, 15);

}

RIT128x96x4StringDraw("TSize: ", 15, 60, 15);

char TrainSZ[2];

TrainSZ[0] = \*(disData->trainSize) + '0';

TrainSZ[1] = '\0';

RIT128x96x4StringDraw(TrainSZ, 52, 60, 15);

RIT128x96x4StringDraw("TTime: ", 70, 60, 15);

char TTime0[2];

char TTime1[2];

TTime0[0] = (int)(\*(disData->traversalTime)/10) + '0';

TTime0[1] = '\0';

TTime1[0] = (int)(\*(disData->traversalTime)%10) + '0';

TTime1[1] = '\0';

RIT128x96x4StringDraw(TTime0, 106, 60, 15);

RIT128x96x4StringDraw(TTime1, 112, 60, 15);

}

else

{

RIT128x96x4StringDraw(" ", 0, 0, 15);

RIT128x96x4StringDraw(" ", 0, 12, 15);

RIT128x96x4StringDraw(" ", 0, 48, 15);

RIT128x96x4StringDraw(" ", 0, 60, 15);

RIT128x96x4StringDraw(" No Train Arrive ", 0, 24, 15);

RIT128x96x4StringDraw(" From East ", 0, 36, 15);

}

break;

case 4:

if(\*disData->arrivalDirection == 3)

{

RIT128x96x4StringDraw("West Arriving ", 15, 0, 15);

RIT128x96x4StringDraw("Arriving No: ", 15, 12, 15);

char TrainA0[2];

char TrainA1[2];

TrainA0[0] = (int)(\*(disData->trainArriving)/10 )+ '0';

TrainA0[1] = '\0';

TrainA1[0] = (int)(\*(disData->trainArriving)%10) + '0';

TrainA1[1] = '\0';

RIT128x96x4StringDraw(TrainA0, 90, 12, 15);

RIT128x96x4StringDraw(TrainA1, 96, 12, 15);

if (!\*(disData->departingTrainFlag))

{

RIT128x96x4StringDraw("No Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 0)

{

RIT128x96x4StringDraw("North Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 1)

{

RIT128x96x4StringDraw("South Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 2)

{

RIT128x96x4StringDraw("East Departing ", 15, 24, 15);

} else if (\*disData->departureDirection == 3)

{

RIT128x96x4StringDraw("West Departing ", 15, 24, 15);

}

RIT128x96x4StringDraw("Train Present: ", 15, 36, 15);

if(\*(disData->trainPresent) > 0){

RIT128x96x4StringDraw("Yes", 100, 36, 15);

} else{

RIT128x96x4StringDraw("No ", 100, 36, 15);

}

RIT128x96x4StringDraw("Lock: ", 15, 48, 15);

if(\*(disData->interLock) == ON)

{

RIT128x96x4StringDraw("On", 46, 48, 15);

}

RIT128x96x4StringDraw("TSize: ", 15, 60, 15);

char TrainSZ[2];

TrainSZ[0] = \*(disData->trainSize) + '0';

TrainSZ[1] = '\0';

RIT128x96x4StringDraw(TrainSZ, 52, 60, 15);

RIT128x96x4StringDraw("TTime: ", 70, 60, 15);

char TTime0[2];

char TTime1[2];

TTime0[0] = (int)(\*(disData->traversalTime)/10) + '0';

TTime0[1] = '\0';

TTime1[0] = (int)(\*(disData->traversalTime)%10) + '0';

TTime1[1] = '\0';

RIT128x96x4StringDraw(TTime0, 106, 60, 15);

RIT128x96x4StringDraw(TTime1, 112, 60, 15);

}

else

{

RIT128x96x4StringDraw(" ", 0, 0, 15);

RIT128x96x4StringDraw(" ", 0, 12, 15);

RIT128x96x4StringDraw(" ", 0, 48, 15);

RIT128x96x4StringDraw(" ", 0, 60, 15);

RIT128x96x4StringDraw(" No Train Arrive ", 0, 24, 15);

RIT128x96x4StringDraw(" From West ", 0, 36, 15);

}

break;

}//end train status

break;

case 2: //annunciation

if(\*(disData->gridlock) && (\*(disData->departingTrainFlag))) {

RIT128x96x4StringDraw("Gridlock Warning!!!", 12, 24, 15);

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

} else if (!\*(disData->gridlock) && (\*(disData->departingTrainFlag))){

RIT128x96x4StringDraw("Train Passing!!! ", 12, 24, 15);

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

} else if(!\*(disData->departingTrainFlag)){

RIT128x96x4StringDraw("No Depareture Train", 12, 24, 15);

RIT128x96x4StringDraw(" ", 15, 12, 15);

RIT128x96x4StringDraw(" ", 15, 48, 15);

}

int temperatureReading0 = 32 \* (\*(disData->temperatureBuf)) / 100 + 33;

double pretempReading0 = 0.384\* (double)(\*(disData->temperatureBuf + 3)) + 33.0 ;

char temp0[80];

sprintf(temp0, "%d", temperatureReading0);

char temp1[80];

int temperatureReading1 = 32 \* (\*(disData->temperatureBuf + 1)) / 100 + 33;

double pretempReading1 = 0.384\* (double)(\*(disData->temperatureBuf + 4)) + 33.0 ;

sprintf(temp1, "%d", temperatureReading1);

char temp2[80];

int temperatureReading2 = 32 \* (\*(disData->temperatureBuf + 2)) / 100 + 33;

double pretempReading2 = 0.384\* (double)(\*(disData->temperatureBuf + 5)) + 33.0 ;

sprintf(temp2 , "%d", temperatureReading2);

if(\*(disData->trainArriving) > 0 || \*(disData->departingTrainFlag))

{

char freq0[80];

sprintf(freq0, "%d", noiseFrequency);

RIT128x96x4StringDraw("Freq:",0,48,15);

RIT128x96x4StringDraw(freq0, 30, 48, 15);

} else {

RIT128x96x4StringDraw(" " ,0,48,15);

}

if(\*(disData->departingTrainFlag))

{

RIT128x96x4StringDraw("Temp:",0,60,15);

RIT128x96x4StringDraw(temp0 ,30,60,15);

RIT128x96x4StringDraw(temp1 ,60,60,15);

RIT128x96x4StringDraw(temp2 ,90,60,15);

} else {

RIT128x96x4StringDraw(" " ,0,60,15);

}

if ((temperatureReading0 > pretempReading0

|| temperatureReading1> pretempReading1

||temperatureReading2>pretempReading2)

&& (\*(disData->temperatureBuf + 3) != 0 )

&& (\*(disData->temperatureBuf + 4) != 0)

&& (\*(disData->temperatureBuf + 5) != 0))

{

RIT128x96x4StringDraw("TOO HOT!!",0,0,15); // alarm

} else

{

RIT128x96x4StringDraw(" ",0,0,15); // alarm

}

break;

} //end mode

vTaskDelay(taskdelay);

}

}

return;

}

void vLocalKeypad(void\* train)

{

while(1)

{

//task7 = uxTaskGetStackHighWaterMark(NULL);

localKeypadData \*keypadData = (localKeypadData\*)train;

modeSelect();

trainStatus();

scrollNselect();

// Home

if(0x10 == GPIOPinRead(GPIO\_PORTA\_BASE,GPIO\_PIN\_4) && key0)

{

OLEDChanged = TRUE;

key0 = 0;

\*keypadData->mode = 0;

\*keypadData->select = 0;

\*keypadData->annunciation = FALSE;

\*keypadData->scroll = 0;

\*keypadData->statusSelection = 0;

for (int i = 0; i < 101; i+=5)

{

for (int j = 0; j < 130; j+=5)

{

RIT128x96x4StringDraw(" ", j, i, 15);

}

}

}

if(0x00 == GPIOPinRead(GPIO\_PORTA\_BASE,GPIO\_PIN\_4) && !key0){

key0 = 1;

}

vTaskDelay(taskdelay);

}

return;

}

void vTemperatureMeasurement(void\* train)

{

while(1)

{

//task8 = uxTaskGetStackHighWaterMark(NULL);

temperatureData \*tempData = (temperatureData\*) train;

// int maxTemp;

if (measure&& \*(tempData->departingTrainFlag) && \*(tempData->globalStartMeasure)) {

ADCIntClear(ADC0\_BASE, 0);

// Trigger the ADC conversion.

ADCProcessorTrigger(ADC0\_BASE, 0);

// Wait for conversion to be completed.

while(!ADCIntStatus(ADC0\_BASE, 0, false)) {}

// Read ADC Value.

ADCSequenceDataGet(ADC0\_BASE, 0, &ulValue0);

delay(5);

ADCIntClear(ADC0\_BASE, 1);

// Trigger the ADC conversion.

ADCProcessorTrigger(ADC0\_BASE, 1);

// Wait for conversion to be completed.

while(!ADCIntStatus(ADC0\_BASE, 1, false)) {}

// Read ADC Value.

ADCSequenceDataGet(ADC0\_BASE, 1, &ulValue1);

delay(5);

ADCIntClear(ADC0\_BASE, 2);

// Trigger the ADC conversion.

ADCProcessorTrigger(ADC0\_BASE, 2);

// Wait for conversion to be completed.

while(!ADCIntStatus(ADC0\_BASE, 2, false)) {}

// Read ADC Value.

ADCSequenceDataGet(ADC0\_BASE, 2, &ulValue2);

for (int i = 15; i > 2; i--){

\*(tempData->temperatureBuf + i) = \*(tempData->temperatureBuf + (i - 3));

}

\*(tempData->temperatureBuf) = ulValue0 \* 341 / 122; //convert to millivolts

\*(tempData->temperatureBuf + 1) = ulValue1 \* 341 /122;

\*(tempData->temperatureBuf + 2) = ulValue2 \* 341 /122;

measure = FALSE;

}

vTaskDelay(taskdelay);

}

return;

}

void vNoiseCapture(void\* train)

{

while(1)

{

//task9 = uxTaskGetStackHighWaterMark(NULL);

noiseCaptureData \*captureData = (noiseCaptureData\*) train;

static int c = 0;

if(c < 256 && captureNoise && \*(captureData->globalStartMeasure))

{

captureNoise = FALSE;

ADCIntClear(ADC0\_BASE, 3);

// Trigger the ADC conversion.

ADCProcessorTrigger(ADC0\_BASE, 3);

// Wait for conversion to be completed.

while(!ADCIntStatus(ADC0\_BASE, 3, false)) {}

// Read ADC Value.

ADCSequenceDataGet(ADC0\_BASE, 3, &ulValue3);

//noiseCaptureBuf [c]= ulValue3;

noiseCaptureBuf [c]= (signed short)(0.063\* ulValue3 - 32);

c++;

}

if (c == 256){

c = 0;

processNoise = TRUE;

}

}

return;

}

void vNoiseProcessing(void\* train)

{

while(1)

{

//task10 = uxTaskGetStackHighWaterMark(NULL);

noiseProcessingData \*processingData = (noiseProcessingData\*) train;

static signed int peakIndex = 0;

//static int freq = 0;

static int read = 0;

if (processNoise){

if (read%5 == 0){

processNoise = FALSE;

signed int imagBuf[256];

for (int i = 0; i < 256; i ++)

{

imagBuf[i] = 0;

}

peakIndex = optfft(noiseCaptureBuf, imagBuf);

noiseFrequency = peakIndex \* 16000 / 256 ;

for (int i = 15; i > 0; i--){

\*(processingData->noiseTransBuf + i) = \*(processingData->noiseTransBuf + (i - 1));

}

\*(processingData->noiseTransBuf) = noiseFrequency;

measure = FALSE;

for(int i = 0; i < 256; i++)

{

noiseCaptureBuf[i] = 0;

}

}

read++;

}

vTaskDelay(500);

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* optfft.c \*/

/\* \*/

/\* An optimized version of the fft function using only 16-bit integer math. \*/

/\* \*/

/\* Optimized by Brent Plump \*/

/\* Based heavily on code by Jinhun Joung \*/

/\* \*/

/\* - Works only for input arrays of 256 length. \*/

/\* - Requires two arrays of 16-bit ints. The first contains the samples, the \*/

/\* second contains all zeros. The samples range from -31 to 32 \*/

/\* - Returns the index of the peak frequency \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "optfft.h"

#define ABS(x) (((x)<0)?(-(x)):(x))

#define CEILING(x) (((x)>511)?511:(x))

signed int optfft(signed int real[256], signed int imag[256]) {

signed int i, i1, j, l, l1, l2, t1, t2, u;

#include "tables.c"

/\* Bit reversal. \*/

/\*Do the bit reversal \*/

l2 = 128;

i=0;

for(l=0;l<255;l++) {

if(l < i) {

j=real[l];real[l]=real[i];real[i]=j;

}

l1 = l2;

while (l1 <= i){

i -= l1;

l1 >>= 1;

}

i += l1;

}

/\* Compute the FFT \*/

u = 0;

l2 = 1;

for(l=0;l<8;l++){

l1 = l2;

l2 <<= 1;

for(j=0;j<l1;j++){

for(i=j;i<256;i+=l2){

i1 = i + l1;

t1 = (u1[u]\*real[i1] - u2[u]\*imag[i1])/32;

t2 = (u1[u]\*imag[i1] + u2[u]\*real[i1])/32;

real[i1] = real[i]-t1;

imag[i1] = imag[i]-t2;

real[i] += t1;

imag[i] += t2;

}

u++;

}

}

/\* Find the highest amplitude value \*/

/\* start at index 1 because 0 can hold high values \*/

j=1;

l=0;

for ( i=1; i<(128); i++ ) {

l1 = square[CEILING(ABS(real[i]))]+square[CEILING(ABS(imag[i]))];

if (l1 > l) {

j = i;

l = l1;

}

}

return (j);

}

void vSerialCom(void\* train)

{

while(1)

{

//task11 = uxTaskGetStackHighWaterMark(NULL);

///============================================================

// HYPERTERM setup (UTAR)

//============================================================

//

// Enable the peripherals used by this example.

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//

// Enable processor interrupts.

//

IntMasterEnable();

//

// Set GPIO A0 and A1 as UART pins.

//

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//

// Configure the UART for 115,200, 8-N-1 operation.

//

UARTConfigSetExpClk(UART0\_BASE, SysCtlClockGet(), 115200,

(UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |

UART\_CONFIG\_PAR\_NONE));

//

// Enable the UART interrupt.

//

IntEnable(INT\_UART0);

UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

serialComData \*serialData = (serialComData\*)train;

if(startSerial) {

////////////////////////////

UARTSend((unsigned char \*)"Train Present: " , 16);

unsigned char TP[2];

TP[0] = (\*(serialData->trainPresent)) + 48;

if (\*(serialData->trainPresent) < 10) {

UARTSend((unsigned char \*)TP, 1);

} else

{

int temp4 = (\*(serialData->trainArriving));

char twoDigit4[3];

int i\_4 = 1;

while ( temp4 != 0) {

int digt4 = temp4 % 10;

twoDigit4[i\_4] = digt4 + 48;

temp4 = temp4 / 10;

i\_4--;

}

UARTSend((unsigned char \*)twoDigit4, 2);

}

UARTSend((unsigned char \*)"\n\r", 2);

// print current direction and train size

UARTSend((unsigned char \*) "Departure Direction: ", 22);

if((\*serialData->departingTrainFlag)){

switch(\*(serialData->departureDirection)) {

case 0:

UARTSend((unsigned char \*)"NORTH\n\r", 8);

UARTSend((unsigned char \*)"Train Size: ", 13);

unsigned char TS[1];

TS[1] = (\*(serialData->trainSize)) + 48;

UARTSend((unsigned char \*)TS, 1);

UARTSend((unsigned char \*)"\n\r", 2);

break;

case 1:

UARTSend((unsigned char \*)"SOUTH\n\r", 8);

UARTSend((unsigned char \*)"Train Size: ", 13);

unsigned char TS2[1];

TS2[1] = (\*(serialData->trainSize)) + 48;

UARTSend((unsigned char \*)TS2, 1);

UARTSend((unsigned char \*)"\n\r", 2);

break;

case 2:

UARTSend((unsigned char \*)"EAST\n\r", 7);

UARTSend((unsigned char \*)"Train Size: ", 13);

unsigned char TS3[1];

TS3[1] = (\*(serialData->trainSize)) + 48;

UARTSend((unsigned char \*)TS3, 1);

UARTSend((unsigned char \*)"\n\r", 2);

break;

case 3:

UARTSend((unsigned char \*)"WEST\n\r", 7);

UARTSend((unsigned char \*)"Train Size: ", 13);

unsigned char TS4[1];

TS4[1] = (\*(serialData->trainSize)) + 48;

UARTSend((unsigned char \*)TS4, 1);

UARTSend((unsigned char \*)"\n\r", 2);

break;

}

} else

{

UARTSend((unsigned char \*)"NA\n\r", 4);

UARTSend((unsigned char \*)"Train Size: ", 13);

UARTSend((unsigned char \*)"NA\n\r", 4);

}

// PRINT TRAVERSAL TIME

UARTSend((unsigned char \*)"Traversal Time: " , 17);

int temp2 = (\*(serialData->traversalTime));

char twoDigit2[3];

int i\_2 = 1;

while ( temp2 != 0) {

int digt2 = temp2 % 10;

twoDigit2[i\_2] = digt2 + 48;

temp2 = temp2 / 10;

i\_2 --;

}

if (\*(serialData->traversalTime) == 0) {

UARTSend((unsigned char \*)"NA\n\r", 4);

} else {

UARTSend((unsigned char \*)twoDigit2, 2);

UARTSend((unsigned char \*)"\n\r", 2);

}

UARTSend((unsigned char \*)"Intersection Availability: " , 28);

if(\*serialData->departingTrainFlag){

UARTSend((unsigned char \*)"Not Available!\n\r", 16);

} else {

UARTSend((unsigned char \*)"Available!\n\r", 12);

}

// print train arriving

UARTSend((unsigned char \*)"Train Arriving: " , 17);

unsigned char TA[2];

TA[0] = (\*(serialData->trainArriving)) + 48;

if (\*(serialData->trainArriving) < 10) {

UARTSend((unsigned char \*)TA, 1);

} else

{

int temp = (\*(serialData->trainArriving));

char twoDigit[3];

int i = 1;

while ( temp != 0) {

int digt = temp % 10;

twoDigit[i] = digt + 48;

temp = temp / 10;

i--;

}

UARTSend((unsigned char \*)twoDigit, 2);

}

UARTSend((unsigned char \*)"\n\r", 2);

UARTSend((unsigned char \*)"Arrival Direction: ", 20);

switch(\*(serialData->arrivalDirection)) {

case 4:

UARTSend((unsigned char \*)"NA\n\r", 4);

break;

case 0:

UARTSend((unsigned char \*)"NORTH\n\r", 8);

break;

case 1:

UARTSend((unsigned char \*)"SOUTH\n\r", 8);

break;

case 2:

UARTSend((unsigned char \*)"EAST\n\r", 7);

break;

case 3:

UARTSend((unsigned char \*)"WEST\n\r", 7);

break;

}

UARTSend((unsigned char \*)"Arrival Time: ", 15);

if(\*(serialData->arrivalTime) != 0)

{

int temp3 = (\*(serialData->arrivalTime));

char twoDigit3[3];

int i\_3 = 1;

while ( temp3 != 0) {

int digt3 = temp3 % 10;

twoDigit3[i\_3] = digt3 + 48;

temp3 = temp3 / 10;

i\_3 --;

}

UARTSend((unsigned char \*)twoDigit3, 2);

UARTSend((unsigned char \*)"\n\r", 2);

} else

{

UARTSend((unsigned char \*)"NA\n\r", 4);

}

UARTSend((unsigned char \*)"Gridlock: ", 12);

if (\*(serialData->gridlock))

{

UARTSend((unsigned char \*)"YES", 4);

//UARTSend((unsigned char \*)"\n\r", 2);

} else {

UARTSend((unsigned char \*)"NO", 3);

}

UARTSend((unsigned char \*)"\n\r", 2);

// PRINT NOISE

UARTSend((unsigned char \*)"Noise: ", 8);

if (noiseTransBuf[0] != NULL && noiseTransBuf[0] != 0)

{

int noisePrint = noiseTransBuf[0];

if (noisePrint > 999)

{

char printNoise[5];

int i\_4 = 3;

while ( noisePrint != 0) {

int digt4 = noisePrint % 10;

printNoise[i\_4] = digt4 + 48;

noisePrint = noisePrint / 10;

i\_4 --;

}

UARTSend((unsigned char \*)printNoise, 4);

UARTSend((unsigned char \*)"\n\r", 2);

} else

{

char printNoise1[4];

int i\_T4 = 2;

while ( noisePrint != 0) {

int digtT4 = noisePrint % 10;

printNoise1[i\_T4] = digtT4 + 48;

noisePrint = noisePrint / 10;

i\_T4 --;

}

UARTSend((unsigned char \*)printNoise1, 3);

UARTSend((unsigned char \*)"\n\r", 2);

}

} else

{

UARTSend((unsigned char \*)"NA\n\r", 4);

}

// print tempertature

UARTSend((unsigned char \*)"Temperature: ", 14);

int temperatureReading0 = 32 \* (temperatureBuf[0]) / 100 + 33;

int temperatureReading1 = 32 \* (temperatureBuf[1]) / 100 + 33;

int temperatureReading2 = 32 \* (temperatureBuf[2]) / 100 + 33;

// print temperature0

if (temperatureReading0 > 99)

{

char printTemp0[4];

int i\_5 = 2;

while ( temperatureReading0 != 0) {

int digt5 = temperatureReading0 % 10;

printTemp0[i\_5] = digt5 + 48;

temperatureReading0 = temperatureReading0 / 10;

i\_5 --;

}

UARTSend((unsigned char \*)printTemp0, 3);

UARTSend((unsigned char \*)" ", 2);

} else

{

int temp6 = temperatureReading0;

char twoDigit6[3];

int i\_6 = 1;

while ( temp6 != 0) {

int digt6 = temp6 % 10;

twoDigit6[i\_6] = digt6 + 48;

temp6 = temp6 / 10;

i\_6 --;

}

UARTSend((unsigned char \*)twoDigit6, 2);

UARTSend((unsigned char \*)" ", 2);

}

// print temperature1

if (temperatureReading1 > 99)

{

char printTemp1[4];

int i\_7 = 2;

while ( temperatureReading1 != 0) {

int digt7 = temperatureReading1 % 10;

printTemp1[i\_7] = digt7 + 48;

temperatureReading1 = temperatureReading1 / 10;

i\_7 --;

}

UARTSend((unsigned char \*)printTemp1, 3);

UARTSend((unsigned char \*)" ", 2);

} else

{

int temp8 = temperatureReading1;

char twoDigit8[3];

int i\_8 = 1;

while ( temp8 != 0) {

int digt8 = temp8 % 10;

twoDigit8[i\_8] = digt8 + 48;

temp8 = temp8 / 10;

i\_8 --;

}

UARTSend((unsigned char \*)twoDigit8, 2);

UARTSend((unsigned char \*)" ", 2);

}

// print temperature2

if (temperatureReading2 > 99)

{

char printTemp2[4];

int i\_9 = 2;

while ( temperatureReading2 != 0) {

int digt9 = temperatureReading2 % 10;

printTemp2[i\_9] = digt9 + 48;

temperatureReading2 = temperatureReading2 / 10;

i\_9 --;

}

UARTSend((unsigned char \*)printTemp2, 3);

} else

{

int tempT1 = temperatureReading2;

char twoDigitT1[3];

int i\_T1 = 1;

while ( tempT1 != 0) {

int digtT1 = tempT1 % 10;

twoDigitT1[i\_T1] = digtT1 + 48;

tempT1 = tempT1 / 10;

i\_T1 --;

}

UARTSend((unsigned char \*)twoDigitT1, 2);

}

UARTSend((unsigned char \*)"\n\r", 2);

UARTSend((unsigned char \*)"\n\r", 2);

}

vTaskDelay(taskdelay);

}

return;

}

void vCommand(void\* train)

{

while(1)

{

//task1 = uxTaskGetStackHighWaterMark(NULL);

static int countDisplay = 0;

commandData \*comData = (commandData\*)train;

if (strstr(readCommand,"S"))

{

\*(comData->globalStartMeasure) = TRUE;

commandAccept = 1;

} else if (strstr(readCommand,"P"))

{

\*(comData->globalStartMeasure) = FALSE;

commandAccept = 1;

} else if (strstr(readCommand, "D"))

{

readCommand[0] = NULL;

countDisplay ++;

OLEDChanged = TRUE;

execute = TRUE;

if (countDisplay % 2 == 1)

{

globalStartDisplay = FALSE;

for (int i = 0; i < 101; i+=5)

{

for (int j = 0; j < 130; j+=5)

{

RIT128x96x4StringDraw(" ", j, i, 15);

}

}

}else

{

globalStartDisplay = TRUE;

}

} else if (strstr(readCommand, "M")){

if (strstr(readCommand, "temp")){

// show temperatureBuf

\*(comData->showRecentData) = 1;

commandAccept = 1;

} else if (strstr(readCommand, "noise")){

//show noiseBuf

\*(comData->showRecentData) = 2;

commandAccept = 1;

} else {

// show error

commandAccept = 2;

}

} else if (readCommand[0] == NULL)

{

commandAccept = 0;

} else

{

commandAccept = 2;

}

//readCommand[0] = NULL;

//vTaskDelay(taskdelay);

}

}

// |--------------------------------------------------------|

// | delay Function |

// | Fome EE 472 WEB SITE |

// | Assignment 1, project1a-2014.c |

// |--------------------------------------------------------|

void delay(unsigned long aValue)

{

volatile unsigned long i = 0;

volatile unsigned int j = 0;

for (i = aValue; i > 0; i--)

{

for (j = 0; j < 100; j++);

}

return;

}

int randomInteger(int low, int high)

{

static int seed = 8464863;

double randNum = 0.0;

int multiplier = 2743;

int addOn = 5923;

double max = INT\_MAX + 1.0;

int retVal = 0;

if (low > high)

retVal = randomInteger(high, low);

else

{

seed = seed\*multiplier + addOn;

randNum = seed;

if (randNum <0)

{

randNum = randNum + max;

}

randNum = randNum/max;

retVal = ((int)((high-low+1)\*randNum))+low;

}

seed += 357;

return retVal;

}

void GPIOFIntHandler(void) { ///sw2

// Setup the interrupts for the timer timeouts.

IntEnable(INT\_TIMER1A);

// Enable the timers.

//

TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerEnable(TIMER1\_BASE, TIMER\_A);

// CLEAR GPIO

GPIOPinIntClear(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

addingTrain = TRUE;

}

void GPIOEIntHandler(void) { /// sw3

// Setup the interrupts for the timer timeouts.

IntEnable(INT\_TIMER1A);

// Enable the timers.

//

TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerEnable(TIMER1\_BASE, TIMER\_A);

// CLEAR GPIO

GPIOPinIntClear(GPIO\_PORTE\_BASE, GPIO\_PIN\_0);

measuringTemp = TRUE;

}

void Timer1IntHandler(void)

{

TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

if (addingTrain)

{

trainArriving++;

startSerial = TRUE;

addingTrain = FALSE;

}

if (measuringTemp)

{

measure = TRUE;

}

TimerIntDisable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

TimerDisable(TIMER1\_BASE, TIMER\_A);

}

void Timer2IntHandler(void)

{

TimerIntClear(TIMER2\_BASE, TIMER\_TIMA\_TIMEOUT);

if(test11)

{

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_4, 0x10);

} else if (!test11)

{

GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_4, 0x00);

}

test11 = 1 - test11;

}

void Timer3IntHandler(void)

{

static int countDistance = 0;

TimerIntClear(TIMER3\_BASE, TIMER\_TIMA\_TIMEOUT);

captureNoise = TRUE;

if (0x20 == GPIOPinRead(GPIO\_PORTD\_BASE,GPIO\_PIN\_5) )

{

countDistance ++;

} else if (0x00 == GPIOPinRead(GPIO\_PORTD\_BASE,GPIO\_PIN\_5) && countDistance > 0 ){

remainDistance = (int)(countDistance \* 2.35 + 97);

countDistance = 0;

}

}

void

UARTIntHandler(void)

{

unsigned long ulStatus;

//

// Get the interrrupt status.

//

ulStatus = UARTIntStatus(UART0\_BASE, true);

//

// Clear the asserted interrupts.

//

UARTIntClear(UART0\_BASE, ulStatus);

//

// Loop while there are characters in the receive FIFO.

//

while(UARTCharsAvail(UART0\_BASE))

{

//

// Read the next character from the UART and write it back to the UART.

//

UARTCharPutNonBlocking(UART0\_BASE, UARTCharGetNonBlocking(UART0\_BASE));

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Send a string to the UART.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void

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

{

//

// Loop while there are more characters to send.

//

while(ulCount--)

{

//

// Write the next character to the UART.

//

UARTCharPutNonBlocking(UART0\_BASE, \*pucBuffer++);

for(int i = 0; i < 3000; i++);

//delay(5);

}

}

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

// keypad functions:

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

// Mode Select

void modeSelect(void)

{

if(mode == 0)

{

if(scroll % 2 == 0 && 1 == select)

{

mode = 1;

select = 0;

scroll = 0;

}

else if (scroll % 2 == 1 && 1 == select)

{

mode = 2;

select = 0;

scroll = 0;

}

}

}

// Train Status

void trainStatus(void)

{

if (1 == mode && 1 == select){

statusSelection = (scroll % 4) + 1;

}

}

//Scrolling and Selection

void scrollNselect(void)

{

if(0x10 == GPIOPinRead(GPIO\_PORTB\_BASE,GPIO\_PIN\_4) && key1)

{

scroll++;

key1 = 0;

OLEDChanged = TRUE;

}

if(!key1 && (0x00 == GPIOPinRead(GPIO\_PORTB\_BASE,GPIO\_PIN\_4)))

{

key1 = 1;

}

//select

if(0x20 == GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_5) && key2)

{

select = 1;

key2 = 0;

OLEDChanged = TRUE;

}

if(!key2 && 0x00 == GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_5) )

{

key2 = 1;

}

}

////////////////////////////////////////////////////////////////////////////////////

unsigned short vRemoteCom( void \*train )

{

remoteCommunicationData \*remoteData = (remoteCommunicationData\*) train;

// print departure direction

char departD[6];

switch (departureDirection)

{

case 0 :

strcpy (departD,"NORTH");

break;

case 1:

strcpy (departD,"SOUTH");

break;

case 2:

strcpy (departD,"EAST");

break;

case 3:

strcpy (departD,"WEST");

break;

case 4:

strcpy (departD,"NA");

break;

}

// print intersection status

char intersection[14];

if (departingTrainFlag)

{

strcpy (intersection,"NOT AVAILABLE");

} else{

strcpy (intersection,"AVAILABLE");

}

// print arrival direction

char arrivalD[6];

switch (arrivalDirection)

{

case 0 :

strcpy (arrivalD,"NORTH");

break;

case 1:

strcpy (arrivalD,"SOUTH");

break;

case 2:

strcpy (arrivalD,"EAST");

break;

case 3:

strcpy (arrivalD,"WEST");

break;

case 4:

strcpy (arrivalD,"NA");

}

// print gridlock status

char grid[4];

if (gridlock)

{

strcpy (grid,"ON");

} else

{

strcpy (grid,"OFF");

}

int temperatureReading0 = 32 \* (temperatureBuf[0]) / 100 + 33;

int temperatureReading1 = 32 \* (temperatureBuf[1]) / 100 + 33;

int temperatureReading2 = 32 \* (temperatureBuf[2]) / 100 + 33;

// show recent data

char dataName[25] = "Mnoise/Mtemp: ";

int data[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

switch (showRecentData)

{

case 1:

strcpy(dataName, "Recent temperature:");

if (globalStartMeasure)

{

for(int i = 0; i < 16; i ++){

data[i] = 32 \* temperatureBuf[i] / 100 + 33;

}

}

break;

case 2:

strcpy(dataName, "Recent noise:");

if(globalStartMeasure)

{

for(int i = 0; i < 16; i ++){

data[i] = noiseTransBuf[i];

}

}

break;

}

// show command execute

char showExecute[20];

if (commandAccept == 1)

{

strcpy(showExecute, "Command Executed");

commandAccept = 0;

} else if(commandAccept == 2)

{

strcpy(showExecute, "Command Error!!!");

commandAccept = 0;

} else

{

strcpy(showExecute, " ");

if(execute)

{

strcpy(showExecute, "Command Executed");

execute = FALSE;

}

}

sprintf(uip\_appdata,

"<br>Train Present: %d"\

"<br>Departing Direction: %s"\

"<br>Train Size: %d"\

"<br>Traversal Time: %d"\

"<br>Intersection Availability: %s"\

"<br>Arrival Direction: %s"\

"<br>Arrival Distance: %d"\

"<br>Gridlock: %s"\

"<br>Noise Signature: %d"\

"<br>Wheel Temperature: %d %d %d"\

"<br>%s %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d"\

"<br> %s"\

"<p>"\

"<input type=\"text\" name=\"Command\" value=\"\" size=\"50\">",

trainPresent, departD, trainSize, traversalTime,

intersection,arrivalD, arrivingTrainDistance[0], grid, noiseTransBuf[0],

temperatureReading0,temperatureReading1,temperatureReading2, dataName, data[0],

data[1],data[2],data[3],data[4],data[5],data[6],data[7], data[8],

data[9],data[10],data[11],data[12],data[13],data[14],data[15],

showExecute);

return strlen( uip\_appdata );

}

/\*---------------------------------------------------------------------------\*/

/////////////////////////////////////////////////////////////////////////////////////

void vOLEDTask( void \*pvParameters )

{

xOLEDMessage xMessage;

unsigned portLONG ulY, ulMaxY;

static portCHAR cMessage[ mainMAX\_MSG\_LEN ];

extern volatile unsigned portLONG ulMaxJitter;

unsigned portBASE\_TYPE uxUnusedStackOnEntry;

const unsigned portCHAR \*pucImage;

// Functions to access the OLED.

void ( \*vOLEDInit )( unsigned portLONG ) = NULL;

void ( \*vOLEDStringDraw )( const portCHAR \*, unsigned portLONG, unsigned portLONG, unsigned portCHAR ) = NULL;

void ( \*vOLEDImageDraw )( const unsigned portCHAR \*, unsigned portLONG, unsigned portLONG, unsigned portLONG, unsigned portLONG ) = NULL;

void ( \*vOLEDClear )( void ) = NULL;

vOLEDInit = RIT128x96x4Init;

vOLEDStringDraw = RIT128x96x4StringDraw;

vOLEDImageDraw = RIT128x96x4ImageDraw;

vOLEDClear = RIT128x96x4Clear;

ulMaxY = mainMAX\_ROWS\_96;

pucImage = pucBasicBitmap;

// Just for demo purposes.

uxUnusedStackOnEntry = uxTaskGetStackHighWaterMark( NULL );

ulY = ulMaxY;

/\* Initialise the OLED \*/

vOLEDInit( ulSSI\_FREQUENCY );

while( 1 )

{

// Wait for a message to arrive that requires displaying.

xQueueReceive( xOLEDQueue, &xMessage, portMAX\_DELAY );

// Write the message on the next available row.

ulY += mainCHARACTER\_HEIGHT;

if( ulY >= ulMaxY )

{

ulY = mainCHARACTER\_HEIGHT;

vOLEDClear();

}

// Display the message

sprintf( cMessage, "%s", xMessage.pcMessage);

vOLEDStringDraw( cMessage, 0, ulY, mainFULL\_SCALE );

}

}