**E E 472 SUM 2014**

**LAB5**

**Final Project**

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

**Members: Bowen Xue, Bohan Wang, Ruiheng Wang**

Signatures:

Bowen Xue                                  Bohan Wang                                     Ruiheng Wang

**Abstract**

This lab assignment constitutes the final phase of the Smart Train Management System. Like last project, the system utilizes the ARM microprocessor, which communicates with other auxiliary devices---OLED display, LCD and keypad---through GPIO pins. An analog-to-digital converter (ADC) is applied to mimic infrared sensor to monitor wheel temperature. Additionally, the biggest change on software side is the use of FreeRTOS, a real-time operating system with powerful scheduling and task management functions. It will schedule and run all tasks in the system. The system now has the ability to handle multiple incoming trains and monitor the distance to the intersection by keeping track of the frequency of emitting signal, configured also by GPIO pins. The user will interact with the system through the keypad and observe desired information or warnings on OLED, LCD and PC.  The group tests the system by examining the functionalities of all tasks and running them under FreeRTOS environment. The lab fails the FreeRTOS running initially, but later the bug was fixed and all tasks were able to run on the operating system.

**Introduction**

For this lab the group designs the train management system based on last lab, including the ARM microprocessor and peripherals: PWM generator, OLED, LCD and ADC. The microprocessor is able to send information to PC through serial communication and Hyperterm program. The biggest improvement is the using of FreeRTOS, which handles task creation, scheduling and delay.  Like last time, the system will be trigger by the SW2 button on the board and thus generate an interrupt, which signals the incoming of a train.  It will handle trains coming from a random direction and route them to a different direction. Now the system will hold the train waiting at the intersection through intersection lock, implemented by a counting semaphore, if a train is passing thorough the crossroad. Annunciations and display are much like last time where PWM generator will blast sounds with different patterns indicating different directions. The system will use ADC as the input of the wheel temperature, and signal the OLED if the temperature is too high. The group implements hardware timer to determine the major cycle and do the counting of the system time base. Later on, the system is also capable of monitoring engine by a noise processing module. Finally the user is able to control the system via a remote computer by issuing commands and receiving messages.

**Discussion of the lab**

**Design Spec**

The system is intended to route passenger trains through an intersection. Basic functionalities have been preserved from the preliminary design.  Upon interrupted by the user, a train is arriving to the intersection. It will display arrival direction, train size and traversal time. Meanwhile the annunciation subsystem will blast sounds according to the direction (details listed in the table below). Additionally, the system will monitor the incoming distance when the train is less than 1.5km away from the crossroad. The changing of distance will be measured on the oscilloscope by the changing of signal frequency. When the train is within 400 meters, a checkTrain add-task flag will be set to let the scheduler execute the *TrainCom* task. The *TrainCom* task will increment the *trainPresent* variable and adds the train to the *trainDeparting* queue. This task will determine the outbound direction as well as the train size by random number generator. After execution, it will be removed from the scheduler.

*SwitchControl* task will be scheduled thereafter, which handles directional switch, intersection locks and gridlock. Instead of four separate locks, this task will use a counting semaphore to dynamically determine the outbound direction based on the arrival direction. At the same time the *DepartingTrain* task will be scheduled, the train will be delayed for the traversal time if no gridlock and the semaphore will be released when the train has passed. The departing train will be the first one in the *trainDeparting* queue which stores the train outbound direction and train size. The *DepartingTrain* task, much like last time, will be scheduled on call, blast sound patterns of direction and signal the ending of OLED flashing message. A new task, *temperatureMeasurement,* will be incorporated. It uses ADC to digitize the DC input and to mimic the temperature sensor on wheels. The output of ADC will be ranged from 0V to 0.325 V. After the measurement is amplified, the readings will be converted to real temperature data and stored in a buffer. Three temperature readings will be measured and stored on demand, and the task will signal a warning message to the OLED if the highest of the three is 20% higher than the highest of last three ones.

Like previous labs, users will interact with the system through keypad, which includes select, scroll and home button. A menu interface is displayed on the OLED indicating the desired information to be displayed. The *OLEDDisplay* task supports two modes: Train Status and Annunciation. In the first mode, four options of four directions are displayed, and each of which contains train present info, arriving distance, arriving direction and departing direction. In the second mode, the OLED will display alarm information, temperature alarm and arriving distance. In addition, all information will be formatted and displayed on LCD and PC through *SerialCommucniation* task. The system uses hardware timer TIMER0 to synchronize all tasks. A global counter is incremented at time interrupt to maintain synchronization. The following table lists the details of annunciation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Train Direction | North | South | East | West |
| Flashing Rate | 1.5s | 1.5s | 2s | 1s |
| Blasting | Two long blasts of two seconds each followed by two 1s short blasts | Two 2s-long blasts each followed by two 1s short blasts | Three 2s long blasts each followed by two 1s short blasts | One 2s long blasts each followed by two 1s short blasts |

                                Table1. Flashing and Blasting Pattern for Arriving Train

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Train Direction | North | South | East | West |
| Flashing Rate | 1.5s | 1.5s | 2s | 1s |
| Blasting | One 2s long blasts each followed by two 1s short blasts | One 3s long blasts each followed by two 1s short blasts | One 4s long blasts each followed by two 1s short blasts | One 5s long blasts each followed by two 1s short blasts |

                                Table2. Flashing and Blasting Pattern for Departing Train

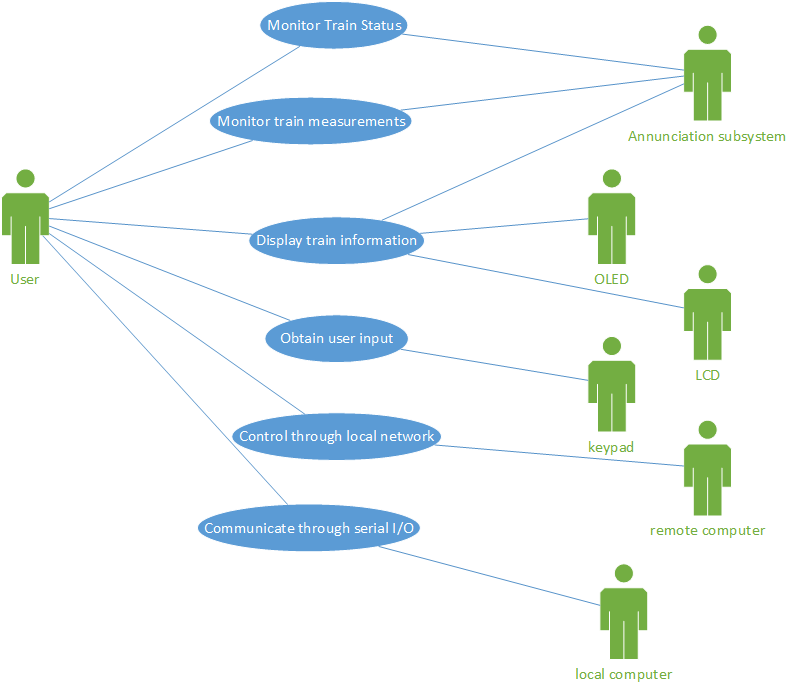
|  |  |  |  |
| --- | --- | --- | --- |
| Button | 0 | 1 | 2 |
| Function | Return to the home page | Scroll the four directions | Select the specific direction |

                                Table3. Keypad Functions

|  |  |
| --- | --- |
| Command or response | Meaning |
| S | START all measurement tasks |
| P | STOP all measurement tasks |
| D | turn ON or OFF the OLED |
| M <payload> | returns the recent measurement of temperature or noise signature |
| A <command> | confirms the receive of the specific command |
| E | error message indicating invalid command |

Table4. Commands and Responses

**Use cases**

****

Monitor train status

This case handles all relevant status information for all directions. User is able to monitor the current direction and departing direction of a given train.

Exceptions: train arrival and departing subsystems inoperable, or malfunctioning distance, temperature or noise measurements.

Monitor train measurements

User is able to keep track of several key measurements for any train passing the intersection. Three parameters are measured: arriving distance, starting from 1.5km away from intersection; wheel temperature, and engine noise signature. Users are able to control the measuring tasks and observe output from either a PC or an OLED.

Exceptions: malfunctioned sensors; no reading due to inoperable train arrival system.

Display information

Users are able to observe formatted real-time train info on OLED and LCD. The OLED displays all train information and warnings for all time, with a simple UI that the user choose the desired information. LCD will display information as long as a train is departing.

Exceptions: malfunctioned LCD or OLED; incorrect display due to bad or delayed internal communication.

Obtain user input

Users are able to interact with OLED UI by a keypad. Supporting functions include scrolling, selection and return to main menu.

Exceptions: malfunctioned keypad; communication failure of sending signals to OLED.

Send information

Users are able to observe formatted train information and measurements on a PC connected by serial port. All information are updated accordingly.

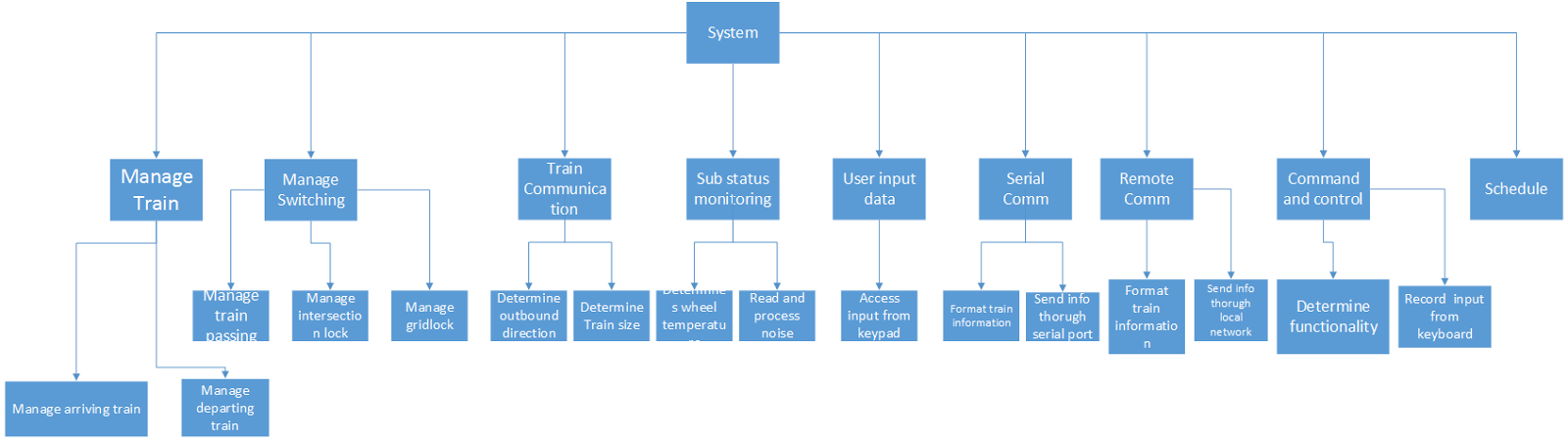
Exceptions: bad communication between microprocessor and computer; malfunctioned sensors.

Control through a remote computer

Users are able to remotely control the train system by issuing a series of commands on a web browser. Supporting commands include start and stop of all measurement tasks, turning OLED on and off and request the most recent measurements. In addition, the web browser is continuously refreshing to display the most up-to-date train information.

Exceptions: bad network connection; malfunctioned sensors.

**Functinal Decomposition**

****

The above is the major functional block decomposition, including *Manage Trains, Manage Switching, Train Communications, Subsystem Status Monitoring, User Input Data, Serial*

*Communications, Remote Communications, Command and Control, Display, and Schedule.*

Manage Trains: Train management with functions similar to the use case description.

Manage Switching: Determines the departing direction based on the arrival direction, and control all intersection lock signals and block any passing if a gridlock occurs.

Train Communications: Communicate arriving system and departing system.

Subsystem status monitoring: Monitor all train measurements and format any warning signals for output.

User input data: Record input from the keypad and match display with a given input.

Serial communications: Receive train information from all tasks and transmit them to a local computer through serial port.

Remote communication: Gather train information and transmit them through a local area network for display.

Command and control: Record user input from keyboard on remote computer, and send commands to microprocessor for control purpose.

Display: Display real-time train info on OLED and LCD

Scheduler: Schedule all tasks within the FreeRTOS given priorities and task handlers.

**Front Panel Diagrams**

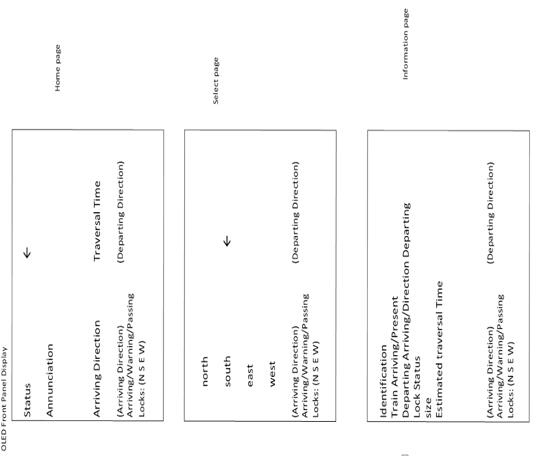




Fig 5

**Block Diagram**

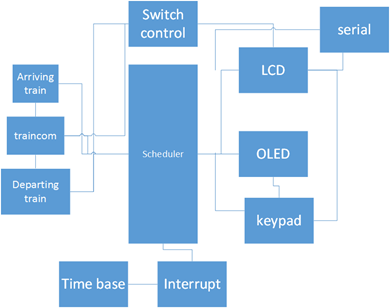
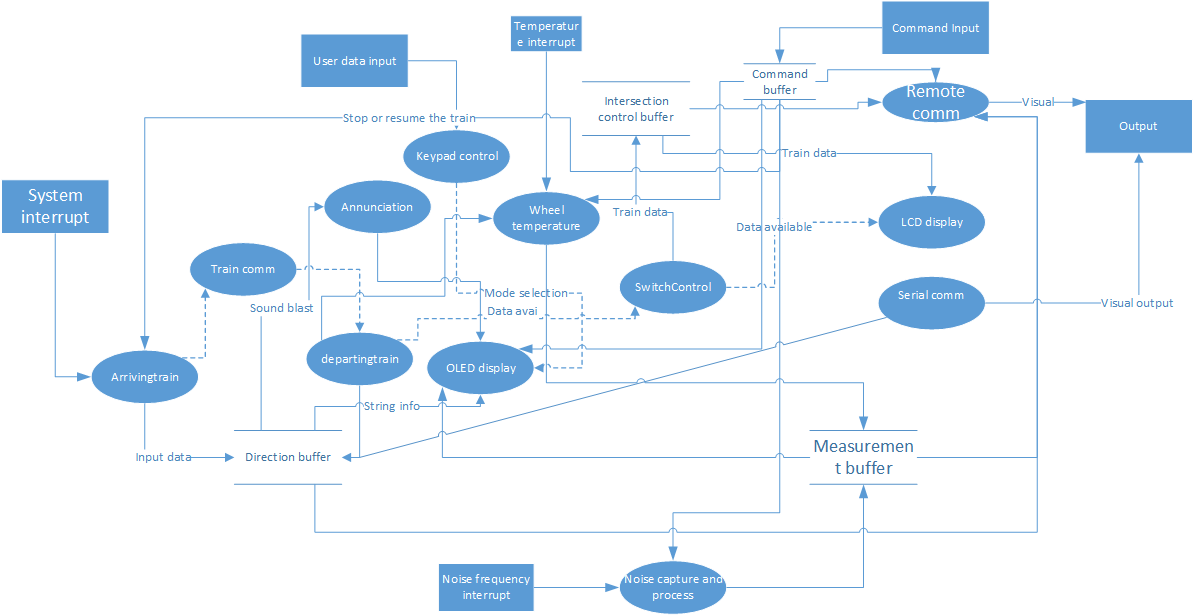
****

Fig 6

The software flow is pretty much similar to previous phases.

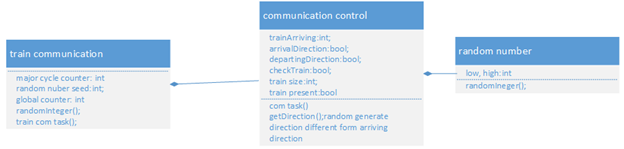
**Control and Data Flow Diagram**

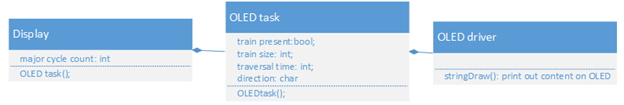
****

The above is the control and data flow of the final project. The system interrupt, SW2 will initiate the system. Additionally, command line input will control arriving train task to stop the train from entering the intersection as well as interrupt all measurement tasks. Remote communication task will be the same as serial local communication, but will display data on a remote computer.

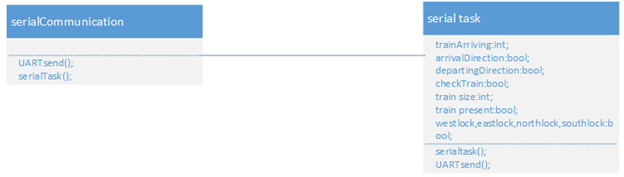
**Class Diagram**

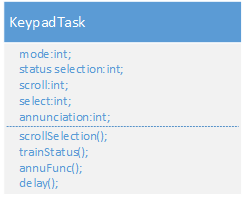
Diagrams below indicate the fields and operations of each task in this system.

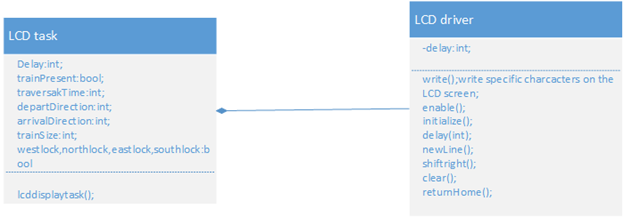
****

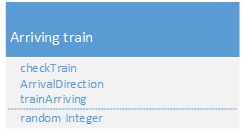
****

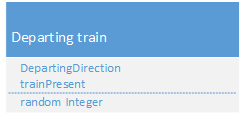
****

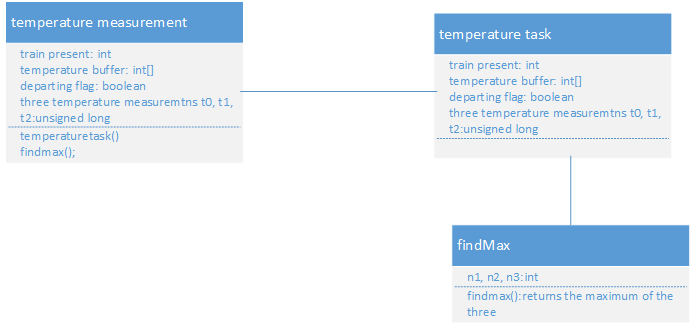
****

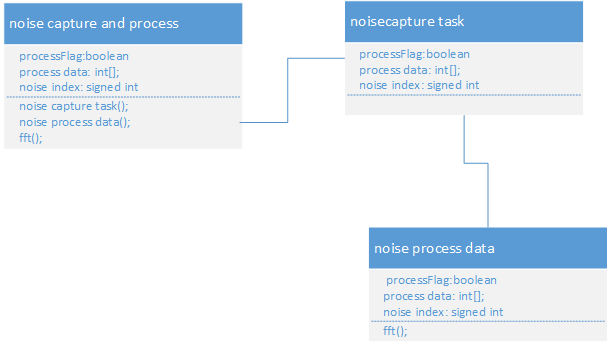
****

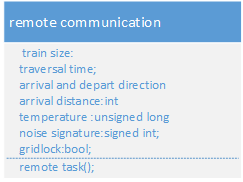
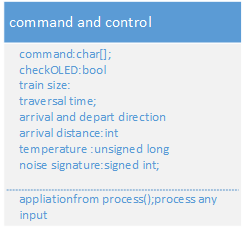
****

****

****

****

****

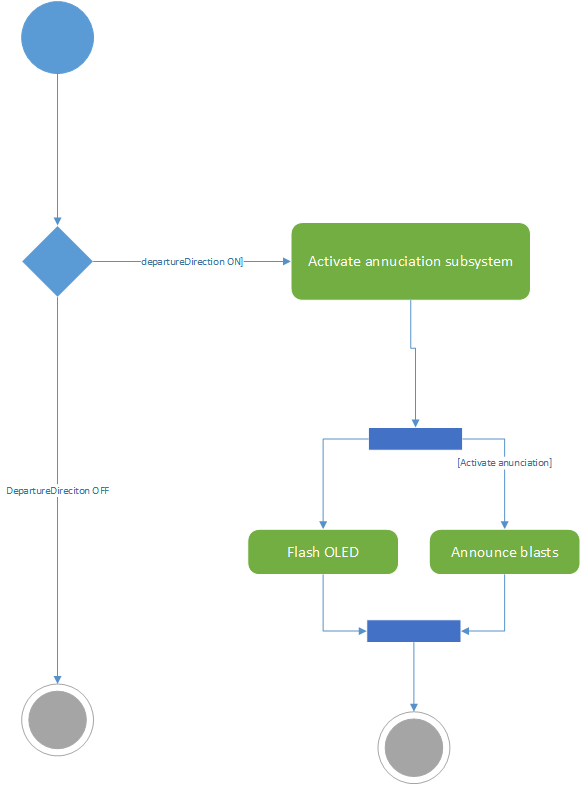
****

**Activity Diagrams**



**Fig.7 Activity Diagram**

The above diagram shows how the arriving train task acts when a train is arriving to the intersection.



**Fig.7 Activity Diagram**

The above shows how the system manages the train departure when the train leave the intersection.



**Fig.7 Activity Diagram**

The above diagram shows activities of the train communication. When the train is present, it will randomly generate the outbound directon and train size.



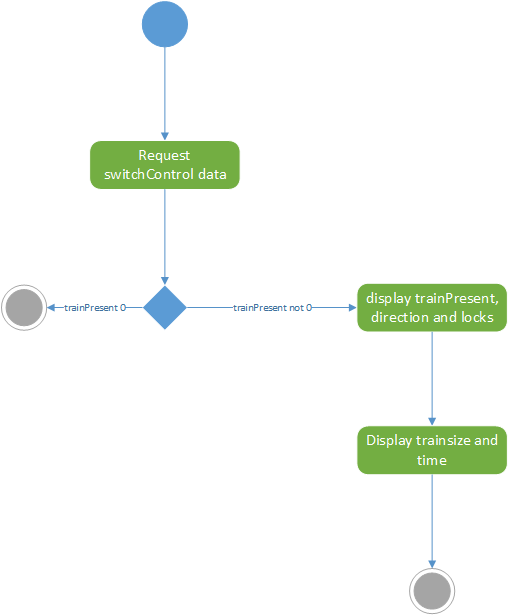
**Fig.7 Activity Diagram**

The above diagram shows how the system routes the train. For this phase, it is similar to the previous in that it handles gridlock by issuing warning message and waiting time for the next train to pass. It also manages the signal(“locks” in this task)to determine if trains outside of the intersection are allowed to pass. This task will determine the traversal time and delay.



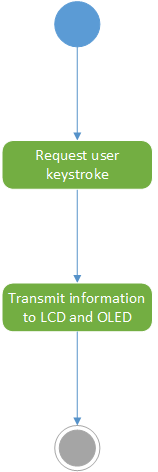
**Fig.7 Activity Diagram**

The OLED task is the user interface. It receives the selection from keypad and display train information or warning message.



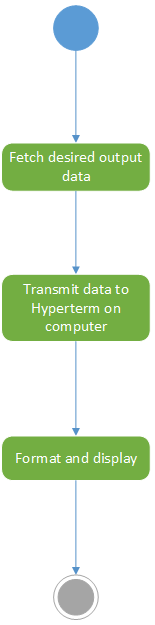
**Fig.7 Activity Diagram**

This LCD task is a new one for this phase. It receives message from the switchControl task and displays information concerning the routing.



**Fig.7 Activity Diagram**

This task serves as the input of the system. Users are able to select the train direction and the content of information to be displayed on OLED.

**Fig.7 Activity Diagram**

This task will summarize all the information , format them and output them to the computer terminal using the serial port.



Command task activity diagram.



Remote communication diagram.



Noise capture and process activity diagram

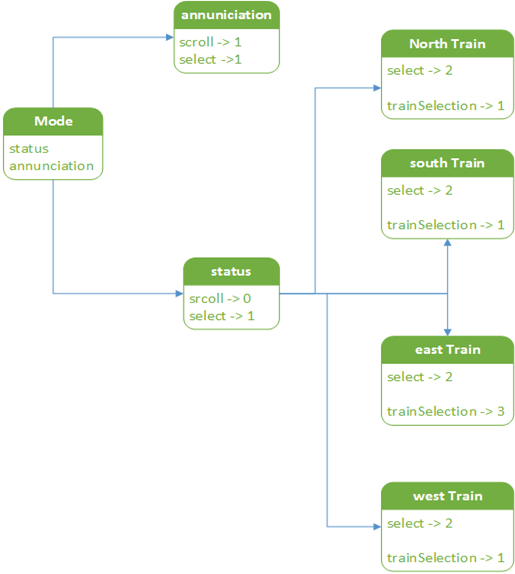


Wheel temperature measurement diagram.

**State Chart**

****

The above diagram shows how a train passes through the intersection.

****

The above is the state chart for the keypad.

**Sequence Diagram**

The following diagram shows the sequential order of each tasks when one train passes the intersection. Due to the complexity of the system, the diagram is only able to give a rough sketch of the chronicle order. User will trigger the start of arrivingISR which immediately starts the arriving train task. Switch control is later activated to route the system and meanwhile the train communication is added to communication between train arriving and departing systems. The user will also trigger the noise capture task, temperature measurement when departing and input data to the keypad and command.

All tasks responsible for display will be running all the time.



**Task control blocks**

The following task control blocks are defined in the final project.

struct ArrivingTrainData  
{  
 unsigned short\* trainArriving;  
 bool\* checkTrain;  
 unsigned short\* arrivalDirection;  
 unsigned short\* arrivingTrainDistance;//designer's choice  
 unsigned short\* trainPresent;  
 bool\* trainDepartingFlag;  
};  
typedef struct ArrivingTrainData ArrivingTrainData;  
ArrivingTrainData arrivingTrainData;  
  
struct TrainCom  
{  
 unsigned short\* trainArriving;  
 unsigned short\* trainPresent;  
 unsigned short\* trainSize;  
 unsigned short\* arrivalDirection;  
 unsigned short\* departureDirection;  
 bool\* checkTrain;  
 int\* trainDepartingQueue;  
};  
typedef struct TrainCom TrainCom;  
TrainCom trainComData;  
  
struct SwitchControl  
{  
 unsigned short\* departureDirection;  
 unsigned short\* departingTrain;  
 bool\* gridlock;  
 unsigned short\* trainPresent;  
 unsigned short\* trainSize;  
 unsigned short\* traversalTime;  
 bool\* trainDepartingFlag;  
 xSemaphoreHandle\* intersectionLock;  
 int\* trainDepartingQueue;    
 unsigned short\* arrivalDireciton;  
};  
typedef struct SwitchControl SwitchControl;  
SwitchControl switchControlData;  
  
struct DepartingTrainData  
{  
 unsigned short\* trainPresent;  
 unsigned short\* departureDirection;  
 int\* trainDepartingQueue;  
 bool\* trainDepartingFlag;  
};  
typedef struct DepartingTrainData DepartingTrainData;  
DepartingTrainData departingTrainData;  
  
struct LcdDisplayData  
{  
 unsigned short\* trainPresent;  
 unsigned short\* trainSize;  
 unsigned short\* traversalTime;  
 unsigned short\* departureDirection;  
 unsigned short\* arrivalDirection;  
 bool\* trainDepartingFlag;  
};  
typedef struct LcdDisplayData LcdDisplayData;  
LcdDisplayData lcdDisplayData;  
  
struct Oled  
{  
 unsigned short\* trainPresent;  
 unsigned short\* trainArriving;  
 unsigned short\* trainSize;  
 unsigned short\* traversalTime;  
 unsigned short\* arrivalDirection;  
 unsigned short\* departureDirection;  
 unsigned short\* mode;  
 unsigned short\* statusSelection;  
 unsigned short\* select;  
 unsigned short\* scroll;  
 unsigned short\* annunciation;   
 unsigned long\* temperatureBuf;  
 unsigned short\* arrivingTrainDistance;  
 bool\* trainDepartingFlag;  
 bool\* gridlock;  
 signed int\* noiseIndex;  
 signed int\* processData;  
 bool\* phasor;  
 bool\* photon;   
};  
typedef struct Oled OledDisplay;  
OledDisplay oledData;  
     
struct LocalKeypadData  
{  
 unsigned short\* mode;  
 unsigned short\* statusSelection;  
 unsigned short\* scroll;  
 unsigned short\* select;  
 unsigned short\* annunciation;  
};  
typedef struct LocalKeypadData LocalKeypadData;  
LocalKeypadData keypadData;  
  
struct TempData  
{  
 unsigned short\* trainPresent;  
 unsigned long\* temperatureBuf;  
 bool\* trainDepartingFlag;  
 unsigned long\* temp0;  
 unsigned long\* temp1;  
 unsigned long\* temp2;  
   
};  
typedef struct TempData TempData;  
TempData tempData;  
  
struct Serial  
{  
 unsigned short\* trainPresent;  
 unsigned short\* trainArriving;  
 unsigned short\* trainSize;  
 unsigned short\* traversalTime;  
 unsigned short\* arrivalDirection;  
 unsigned short\* departureDirection;  
 xSemaphoreHandle\* intersectionLock;  
 unsigned long\* temperatureBuf;  
 bool\* gridlock;  
 unsigned short\* arrivingTrainDistance;  
};  
typedef struct Serial SerialCom;  
SerialCom serialData;  
  
struct NoiseCaptureData  
{  
 bool\* processFlag;  
};  
typedef struct NoiseCaptureData NoiseCaptureData;  
NoiseCaptureData noiseCaptureData;  
  
struct NoiseProcessData  
{  
 signed int\* processData;  
 bool\* processFlag;  
 signed int\* noiseIndex;  
};  
typedef struct NoiseProcessData NoiseProcessData;  
NoiseProcessData noiseProcessData;  
  
struct MeasureDistanceData  
{  
 int\* meter;  
};  
typedef struct MeasureDistanceData MeasureDistanceData;  
MeasureDistanceData measureDistanceData;  
  
struct AttackData  
{  
 bool\* phasor;  
 bool\* photon;   
 int\* meter;  
};  
typedef struct AttackData AttackData;  
AttackData attackData;  
  
struct Command  
{  
 unsigned short\* trainPresent;  
 unsigned short\* trainArriving;  
 unsigned short\* arrivingTrainDistance;  
 unsigned short\* trainSize;  
 unsigned short\* traversalTime;  
 unsigned short\* arrivalDirection;  
 unsigned short\* departureDirection;  
 xSemaphoreHandle\* intersectionLock;  
 unsigned long\* temperatureBuf;  
 bool\* gridlock;   
 char\* response;  
 signed int\* noiseIndex;  
 signed int\* processData;  
 unsigned long\* temp0;  
 unsigned long\* temp1;  
 unsigned long\* temp2;  
};  
typedef struct Command CommandData;  
CommandData commandData;

**Functional Prototype**

The following functions are defined in the main file.

//arrivingTrain Task

void arrivingTrainTask(void\*);

//train communication with intersection

void trainComTask(void\*);

//departing

void departingTrainTask(void\*);

//manageSwitch

void switchControlTask(void\*);

//lcdDisplay

void lcdDisplayTask(void\*);

//oled

void oledTask(void\*);

//keypad

void keypadTask(void\*);

//serialCom

void serialTask(void\*);

void noiseCaptureTask(void\* data);

void noiseProcessTask(void\* data);

void measureDistanceTask(void\* data);

void attackTask(void\* data);

void remoteTask(void\* data);

void temperatureMeasureTask(void\* tempData);

**pseudo codes for the major tasks**

main:

setup the FreeROTS server;

Void main(void) {

        Set up the necessary hardware peripherals and interrupt handlers;

        Declare variables and task control blocks;

        Store the address of all the variables within each struct to pointers;

        Point the address of each task to a pointer;

        Create tasks using RTOS task create function,

        Start the scheduler;

}

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

trainCom:

void trainComTask(void\* trainComDataPtr)

{

        declare the counter and direction variable

        randomly set the train size;

        get the inbound direction;

        randomly set the outbound direction different from the arrival direction

        increment the global counter;

        delete the task from the task queue;

}

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

switch:

void switchControl(void\* switchControlPtr){

        declare the counter that calculate the pass time and warning display time ;

        declare the integer that will determine the train size;

        if (train is present){

                    if (first time the train is present) {

                                generate the random interger n;

                    }

                    declare and define the time to manage the gridlock;

                    if (n is negative)

                    {

                                display warning message;

                    }

                    else //n positive;

                    {

                                increment counter of passing time;

                    }

                    if (is in gridlock ){

                                if (gridlock has ended)

                                {

                                            clear the warning and reset the gridlock flag;

                                            increment the passing counter;

                                }else {

                                            increment the counter to calculate the warning time;

                                }

                    }

                    if (locks in all direction are false){

                                set the locks according to the inbound direction;

                    }

                    display info on OLED AND LCD;

                    if (train has passed){

                                open the locks according to the departing direction;

                    }

        }

        fetch the trainsize variable;

        if (train has passed)

        {

                    reset the trainPresent and the traversalTime variable and all direction;

        }

}

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

arriving （for north, east and west)

void arrivingTrain(void\* dataPtr){

        declare the counter and data pointer;

        if (train is arriving){

                    if (first time the train enters the intersection){

                                store the direction to data pointer;

randomly set the direction;

enable PWM generator and blast sounds

                                display the train arriving direction and passing time;

                    } else {

                                display empty string;

                    }

                    If the train 1.5km away from intersection

                                Set frequency reading parameters;

                                Calculate distance

                    If within 400m

                                Set the checktrain flag and schedule the train com task

        }

        Disable PWM generator;

        Reset all parameters;

}

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

departing（for north, east and west)

void departingTrain(void\* dataPtr){

        declare the counter and data pointer;

        request the departing direction from the traincomtask;

        if (train is present){

                    if (first time the train enters the intersection){

                                get a copy of the current global counter as a local timer;

                                store the direction to data pointer;

                                display the train arriving direction and passing time;

                                switch(Based on the outbound direction){

                                            case 1: north

                                            case 2: south

                                            case 3: east;

                                            case 4: west;

                                                        flash OLED and blast sounds

                                }

        }

}

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

OLEDdisplay:

void oledTask(){

        declare the pointer and counter;

        set up the counter

        detects the falling edge and set the flag;

        //handles the top level selection menu;

        if (mode is 0){

                    display status and annuciation options on OLED;

                    if statements setting the cursor positions on OLED;

        }

        if (mode is 1){

                    erase mode cursors;

        }

        //status selction

        if (mode is 1 and annuciation is 0){

                    display the next-level status menu;

                    switch between four trains based on the scroll variable;

        }

        erase the current display from OLED;

        //display the information for a given train;

        clear the previous display;

        dipslay the arriving train based on OLED;

        display the train status, arriving or present;

        display the arrival direction;

        display the departure direction;

        display the lock information;

        display the size information;

        //announce

        if (train present and in annunciation mode)

                    display warning message if gridlock;

                    else display safe to pass informkation

        reinitialize all local parameters;

}

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

LCDtask

import external LCD driver for displaying charcters, numbers and operations

void lcddisplay(void\* dataptr){

        declare pointers and variables;

        write characters and cursors following the front panel diagram;

}

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

keypad

import

void keypadtask(){

        declare task pointer and variable;

        call helper functions for mode selection, scroll selection and train status display;

        initialize all the local copy of task data;

        set the variable controling the keystroke for three different menus

        if (pins are all 0){

                    clear all variables and OLED;

        }

        //mode selection

        if (mode is 0)

                    if (select is 1)

                                if (scroll is odd)

                                            deselect annuciation

        //scroll selection

        if (key is up){

                    increment scroll variable;

                    reset the bouncing variable;

        }

        make the selection by setting the select variable

}

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

serialcom task:

import the UART send project transmitting data to computer;

void serialcom(){

        declare variables and pointers;

        display the arriving direction and the arrival time;

        display the departing direction in character string;

        display the trainsize;

        display the travseraltime;

        display the intersection lock status;

        display the gridlock information;

}

Temperature measurement

Void temptask(void\*)

{

        Recast the data pointer to void and declare variables;

        Set up ADC process trigger and obtain data for three channels

        Convert and compute the temperature;

        Store temperature to the buffer;

        Compare two recent readings

        If satiefies criteria

                    Signal the warning

}

Void noiseprocess task()

{

        Enable ADC;

        Calculate time to sample the voltage;

        Perform FFT;

        Find the maximum index and store it

}

**Software Implementation**

The operation system is changed into FreeRTOS which is a real time operating system. There are 15 tasks in total.

**Arriving Train**

Arriving train task is responsible for generating a train whenever the user pressed the button, in our case it is SW2 on stellaris board. The debouncer is achieved by adding a delay after the button is pressed. Then the train produced unique sound pattern specific for each train. The north train has the pattern, 2-2-1-1, and flashing rate 1.5s; south train has sound pattern 2-2-3-3, with flashing rate 1.5s; east train has sound pattern 2-2-2-1-1, with flashing rate 2s; west train has sound pattern 2-1-1, with flashing rate 1s(Above patterns can be found on the design spec part).As train gets closer to the intersection, it generates signal with different frequency. It is modeled as a linear relation in this system. As frequency gets higher, distance to intersection becomes smaller. The signal with different frequency is generated by PWM, and it is governed by the relation: 400 + 10 \* frequencyCounter), where frequencyCounter represents how many times the frequency will be increased. The maximum frequency is 590. The reading part is done by using a GPIO pin, which is configured that when it detects a falling edge, an interrupt handler is entered and another counter in the handlers will increment itself. Since our minor cycle is 0.1 second, the frequency is calculated by using 10 x that counter. The distance is calculated by the formula: (-5 \* frequencyRead) + 3000. All the train have to start at 1500m, their speed is constant at 100m/s. Therefore the total time to reach the intersection is 15s. The frequency generating and reading part is activated at 1000m, which is done by using the distance formula and a if statement. At 400m, trainCom task is resumed into the scheduler. Arriving direction is obtained by using two random number, s0,and s1, and generator is obtained from the website. When s0==s1==0, it is a north train; when s1== 0,s0==1, then it is a south train; when s1==1, s0==0, it is a east train; when s1==s0==1, it is a west train. The user is able to add multiple trains, but only one train is allowed to travel on the trail.

**TrainCom**

TrainComTask is responsible for adding the train into departing queue. After the train has arrived at the intersection, it is added to  departing queue which is able to handle upto nine trains. The queue has a size of 20, but every two elements are grouped together. The first element indicates departing direction, and the second element indicates the train size. The last group i.e. element 19th and 20th are reserved. Inbound train cannot have the same direction as outbound train, and a recursion call is added to accomplish this determine the final direction different than the inbound direction.

**Switch Control**

Switch Control is responsible for managing the track switch. Train present variable records the number of departing train, its maximum is 9. When a train is passing the intersection, gridlock may be added, and it is modeled by a random number, when the number is smaller than 0, a gridlock is added, otherwise traversal time is determined only by the train size. Random number is obtained by using the function on the website. When a gridlock is detected on the track, the disruptor fire will be issued automatically, the change of success to clear the gridlock is determined a random number. If disruptor fire doesn’t succeed, a thermonuclear weapon is used to facilitate the process. The previous two weapon is continuously used until the gridlock is cleared. When the train has passed the intersection, train present should be decremented. If available, the next train will be picked up from the departing queue and repeat the previous action.

Departing train task is responsible for producing the unique sound pattern for specific train. The north train is 2-1-1, the south train is 3-1-1, the east train is 4-1-1, and the west train is 5-1-1(details in the design spec).  Under the case which an inbound train with sound pattern A and an outbound train with sound pattern B, the outbound train takes higher priority.

**LCD Display**

Lcd display task is responsible for displaying the information of a departing train. The information on the first row includes the number of train present, the departing direction of the currently departing train, and the state of the semaphore lock. The information on the second row includes the size of the currently departing train, and its traversal time. Lcd display task is supported by a lcd driver which includes a calling set, for example, write\_A, write\_B, write\_C, write\_0, write\_1, write\_2… for more information, please refer to the code. Lcd driver is developed based on the API specified by the document on the website under lcd display. Initialization is done before its usage, the mode of 4 data each time is used to save the GPIO pin, therefore after each command, an extra delay is required.

**OLED**

OLED task is the only place where text is displayed on the OLED . The original string draw function might be too slow if the text is long, as the result, word might be distorted. Therefore the embedded OLED Task is used to assist to draw the string on the OLED . One limitation of the OLED Task function is that the x and y locations cannot be specified, the screen will row back after 9 lines have been filled up. Therefore the basic idea to draw the string without shifting or distortion is to have each section separate, and the each section must also have exactly nine lines. The oled screen is divided into two section, the cutting line falls at line 80. The part of screen below that line is reserved for flashing. As long as there is an incoming train, theOLED would blank regardless whether there is a departing train. When the system is first turned on, user is able to choose status or annunciation. Under status, there are four trains available for user to select. The direction on the screen indicates inbound direction. Under the case that there are one arriving train and departing train going and leaving the intersection at the same time, arriving train takes higher priority, since the departing information has been displayed on the lcd. If the user select the direction without a train, the screen won’t do anything. If it is the case, the user needs to go back to the home page and start again. If the user select the right four information can be displayed, train’s ID, Train arriving / train departing, train arriving/ train present, and distance to intersection. Since train present and lcd can indicate if the train is departing or not. Under the condition if there is an arriving train and a departing train, the arriving train, on the second line, takes higher priority than the departing train. OLED Task also takes the responsibility to show the photon and phasor information. If the annunciation is chosen, gridlock, temperature safety, noise signature and distance to intersection will be displayed.

**KeyPad**

Local keypad takes the responsibility of accepting user’s input. There are three button in the system, the home return button, scroll and select button. The keypad in the system is a 4x4, 16 input keypad, and eight pins corresponds to four rows and four columns. Only the first three keys on the first row is used. To save the GPIO pin, the pin 1 is connected to 3.3V, and the other three pins are connected to GPIO pin to detect the input. If the button is pressed once, it would be nice if the action is only taken once. Therefore three static variable are used to detect that rising edge  and they are reset if the they encounter a falling edge. The keypad has four functions. Under mode 0, “status”, ”annunciation ” are to be displayed. If the on of them is selected, the mode will be changed into 1, and if status is chosen, the screen will go to “North””South””East””West”, and they corresponds to status Selection 1,2,3,4. If the “annunciation” is chosen, gridlock, temperature safety, noise signature and distance to intersection will be displayed.

**Temperature Measurement**

Temperature measurement task takes the responsibility to measure the voltage and convert them into data and then store them into array. Resistors and voltage divider is used to model the temperature. Every time the button is pressed, three measurements will be taken and the highest data will be used to compare with the highest data in the next take. If the next one is 1.2 greater than the current data, a warning is issued, which will be displayed on the OLED . ADC is triggered by software, since it is convenient and totally asynchronous. Temperature will be taken based on user’s demand. In this case, it is conveyed by a button on the stellaris board. This tasks uses ADC0, ADC1 and ADC2, ADC1 is shared with distance sensor, which will be discussed later.

**SerialCom**

Serial Com task takes the responsibility to retrieve the all the data from the system and output them onto the PC, it is accomplished by UART and hyperterm program.

**Noise Signature Processing**

NoiseCapture tasks is issued based on user’s demand, it samples voltage at equal duration 256 times. Then data is processed by optfft which is provided on the website to assists to compute the FFT of the signal. Since optfft takes data from 32 to -31, then an offset is added and scaled to fit into the range. ADC3 is used here, and a timer is chosen to be the trigger. Data is stored in the ADC3’s handler.

**Distance Measurement**

Measure distance Task takes the responsibility to take the voltage generated by the infrared sensor, and convert them into meter. Since the sensor is only able to measure the distance approximately 5cm to 80cm. Then we use the ratio  1cm: 1m to model the situation.

**Command**

This task takes the input from the webserver and will be scheduled when a command is received or information is formatted and ready to be transmitted. It will access all relevant data from the system and prepare them for transmission. The design is roughly based  input process function in the RTOS micro-IP task file. The original file is able to take the input from a textbox and output it to OLED. Now the input is accepted in the textbox, checked if it is matched with the commands in the table above, and stored for transmittion. S and P commands toggle the interrupts of measurement tasks,i.e. temperature measurement, noise processing and approaching frequency measurement, and control the suspension of these tasks. D command turns on or turns off the OLED upon issued. M command requests the most recent measurement of temperature and noise. Besides the above commands, the browser will display responses according to the input. If the webserver has received a valid command, it will display an “A<command>” response, excluding M command which will be displayed as itself. If the input is invalid, an error message E will be displayed.

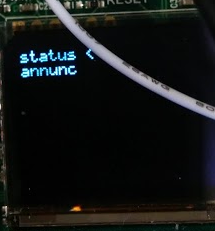
**Remote Communications**

This task handles the setup of network, web server and the data format. The network interface is roughly based on the I/O page of the RTOS. The IP address is set in IP task file, matched with the local area network. Since the interface can be broken down into static part and dynamic part, the group treat them in different ways. The first part is editted in HTML file, and the rest is formatted in common gate interface(CGI) file. The CGI file handles all dynamic display, or the data transmission RTOS web server.

**Obstacle Management Subsystem**

Attack task takes the responsibility to get rid of the obstacle. When the object falls within 3cm, which is 30 m in the train system, a phasor is activated and the user can use a button to start it. When the object falls within 0.5cm, which corresponds to 5m in the train system, the photon weapon is activated, and the user can press the same button to start it.

**Presentation, Discussion and Analysis of Results**

****

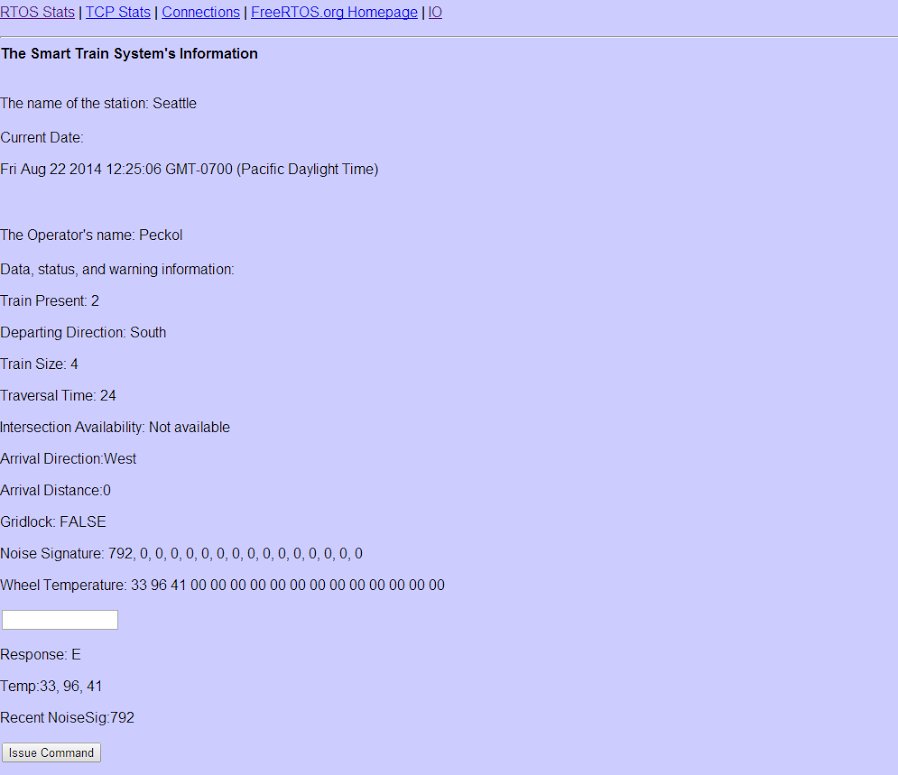
**Fig.1 System Start           Fig. 2 Train Coming**

****

**Fig. 3 Schoole and Select Departure Direction           Fig. 4 Train Information**

****

**Fig. 5  Temperature Display Fig.6  LCD Display**



**Fig. 7 Web Server Output**

Fig. 1- 4 show the OLED disply of the smart train management system. Based on the test cases, the OLED display stays at the Fig.1 when the system starts operate. If there is a traim coming, the arrival direction should be shown in the bottom line as Fig.2. If the user choice the status mode, the page will turn to selection page as Fig.3. In this page, the user can scroole and select any departure direction as she/he wants. At this time, the departure direction is south and after selecting the direction, the train’s information shows on the OLED borad as Fig.4. If the user chooses the annunciation mode, the intersection ability, grid lock, temperature infomation will be showed as Fig.5. The Fig.6. shows the display on LCD board. Only if when there is the train coming, the LCD board will display the arrival train’s information (present, arrival direction -> depature direction, Lock status, train size and the traversla time). Otherwise, there will no display on the LCD board. Fig.7 shows the output of the web server. The top halfpart shows the train system’s and present train’s information. The bottom part shows the command interaction. If the user input request to the text box and click the “issue commen” button, the result will be responsed.

**ErrorHandling**

**Error Analysis**

At the beginning of the design, one of the biggest error is the task running and scheduling in the RTOS environment. When running in RTOS, data pointers are not available on the IAR watchdog, so it is difficult to determine whether data have been passed into tasks. Pointers often point to void or error locations when the group tried run only a few tasks. The group later found out that variables and data pointer must be defined as static inside task so that their values are visible.

At first, noise signature cannot detect certain frequency. It is later discovered that the optfft function only accept inputs of range from -31 to 32. After the correction, the result is pretty close.

If the text to be displayed onto the oled uses Rit128x97x48stringDraw function, the screen may have unexpected result and the texts are distorted. It is corrected by using the vOledTask provided by the FreeRtos system. Since it stores the texts into a queue, therefore each text would have enough time to display.

There was a time that train cannot reach the intersection, switchControl and trainCom task were checked but no error was found. In the end, it was found that the wire that connected the PWM approaching frequency and the GPIO pin reading the frequency was broken, therefore the train could never move, since its frequency was never changed.

For the remote connection, there was a time when data cannot be transmitted to the webserver. It is later discovered that the CGI generator did not access the remote task data pointer as an argument. Later connection was restored and the server is able to display real-time information.

**Analysis of why project didn’t work**

The project is generally a success. Although external output seems fine, several design specs were not satisfied. Both the remote communication and the command tasks were not wrapped as RTOS “tasks”, since they are individual modules in the existing files. In this case, although RTOS has access to server and takes in commands all the time, both tasks are not scheduled, meaning that they are running outside of the RTOS environment. Also functionalities of these two tasks were blended in the design, so it is difficult for further improvements or modifications. A better way would be to organize codes from various sources and group them in separate files.  Additionally, temperature measurements are carried out with no accurate time gap, 600 microseconds in the spec. Generally, these drawbacks don’t degrade performance greatly, but the design is somewhat problematic.

**Testing**

**Test plan**

In order to test the final phase of this system, the group approach the testing more carefully than last time. Since this phase is based on the last one, the group first ensure that functions and modules from the last time work well. Later the group modify the tasks according to the new spec and test their functionality. New tasks are also tested separately by discarding any random functionalities and observing if the output matches the expectation. The group tested the OLED and switch controls by fixing the direction and setting the gridlock manually. Hardware designs are tested by revisiting the user manuals of keypad and OLED and measuring the voltage level through a multimeter. Similarly, all measurement tasks are tested by considering all possible output and adjusting input to match them. All communication tasks are tested by continuously observing display to see if any changes on input will be reflected accordingly.

The next is to integrate all tasks the check if there are any intercommunication problems. Since the FreeRTOS environment is more complex than tasks queues, the group went through demo projects to understand new features and important functionalities of RTOS. Next, all tasks are added to RTOS one at a time to test their functionalities. More tasks are added with different priorities to ensure multitask scheduling. Finally, the group incorporate all tasks and examine any final problems.

**Test spec**

The testing of the system is to be determined by various restrictions on the spec and the system.

arrivingtrain and departingTrain: These two tasks are tested based on the annunciation and the display on OLED.The spec is reproduced below for convenience

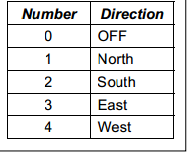
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Train Direction | North | South | East | West |
| Flashing Rate | 1.5s | 1.5s | 2s | 1s |
| Blasting | 2-2, 2-1 | 2-2, 2-3 | 3-2, 2-1 | 1-2, 2-1 |

                            Table. Flashing and Blasting Pattern for Arriving Train

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Train Direction | North | South | East | West |
| Flashing Rate | 1.5s | 1.5s | 2s | 1s |
| Blasting | 1-1, 2-1 | 1-2, 2-1 | 1-3, 2-1 | 1-4, 2-1 |

                            Table. Flashing and Blasting Pattern for Departing Train

TrainCom: This task is tested by generating random numbers to mimic the train size and outbound direction. The direction and the corresponding numbers are listed below. The group isolates the trainCom task and outputs the parameters to the OLED to see if the train size and direction are valid.



Switch Control:this task must have valid gridlock handling part, 0.2s \* randomnum n, and the traversal handler, 0.1\* train size min. Also it must handle all the intersection locks before and after the train passing the intersection, and must output train information to the OLED.

LCDdisplay: This task contains the hardware configuration and the software developing. For the hardware setup, all the connections and ports must be matched according to the LCD manual. GPIO ports are also configured to connect it to the microprocessor. The LCD must display the train present status, direction and four locks on the first row and the train size as well as the traversal time on the second row.

OLED: This task contains two parts. First it must display train info when train is arriving. Secondly, it must interact with the user as one pressing the keypad to either scroll or select an entry of the menu. The OLED will display different menu based on the front panel diagram.

Keypad: This task controls the input of the system. Each key must match its functionalities to the design. Select key will give input data and enters a different display, and scroll key will move the cursor down and comes back to the top.

Wheel temperature task: this task must record three data upon users’ request during train departure. Three measurements should be obtained in sequence with less than 600us. The task must store the 16 most recent temperature, with three readings as a group. It must signal a warning message if any of the current group exceeds the largest of the previously measured data by more than 20%.

Command and control: This task must able to record a series of commands aforementioned. It must determine if any input is valid and format a response prepared for transmission.

Noise capture and process: These two tasks must sample a sine wave isochronously. These samples are processed via FFT, and the index of the peak value in the frequency spectra must be determined.

Serial communication: this task will output the data to the computer through the serial port. It formats all information in the following manner:

Train Present <number of trains>

Current Direction <direction of train using intersection >

Train Size <size of train using intersection>

Traversal Time <traversal time for train using intersection>

Intersection Availability < state of intersection availability>

Train Arriving <incoming train>

Arrival Time <incoming train arrival time>

Gridlock <gridlock state>

Noise Signature <measured value>

Wheel Temperature <measured values>

Remote communication: this task must able to configure a local area network and set up a small web server. It will create the network interface on a web browser. It also must format all data to be displayed and continuously update the data by refreshing the webpage. The format should be like the following:

·         The name of the station

·         Current date

·         The operator’s name

·         Data, status, and warning information

Train Present <number of trains>

Departing Direction <direction of train using intersection >

Train Size <size of train using intersection>

Traversal Time <traversal time for train using intersection>

Intersection Availability < state of intersection availability>

Arrival Direction <direction>

Arrival Distance <incoming train distance>

Gridlock <gridlock state>

Noise Signature <measured value>

Wheel Temperature <measured values>

In all, for trainArriving task, variable with name trainArriving is set to be true. Trian communication has check train to be true. DepartingTrain has trainPresent and departure direction to be 1. Keypad is tested by toggling teh gpio associated with keypad LCD is tested by repeatingly calling the task. OLED is tested by setting train resent o t be true, trainSeize to be 3, traversal time 18 arrival direction 2, departure direction 3. Serial Com is tested by setting all the variable to be 1, and repeatedly calling the function.  Switch is tested by setting train present to be true.

**Test Cases**

The group tests waiting time, traversal time and the hardware timer by timing the interval or the sound pattern with a stopwatch on cellphone.

Arriving train is tested by setting the trainPresent to be 1, then buy hardcoding the arrivaldirection to be 1, 2, 3,  in order to listen the sound pattern in the table above. The group also use cell phone to time the intervals more accurately. For arrival signal, the group test the frequency generation by connecting the GPIO pin to oscilloscope to observe if frequency is increasing as the train approaches. The group also observe frequency counter, how many times the frequency increases by observing in IAR watchdog. If the frequency generation is valid, the group then test the distance reading on watchdog based on the calculated speed 100m/s. Another important distance 400m, where the train communication task is scheduled, is tested by examining the value of the flag.

TrainCom: this task is tested by hardcoding arrival direction to determine if the departure direction is valid. Since this task is able to handle multiple trains, the group also examine the departing queue whether the departing information and train size are stored.

DepartingTrain: the departure train is tested by setting trainPresent to be 1, hardcoding departureDirection to be 1,2,3,4 and listening to the sound pattern in the table above.

LCDdisplay:It  is tested by writing a small display message and sending it through the GPIO ports.By writing sample characters in different positions on the LCD, it can be determined the correct position of the expected output. The next step is to add LCD and switchControl to the task queue and see if the output information of switch task can be seen on LCD.

Serial communcation is tested by setting all variable to be 1 and then read value from screen through the Hyperterm.

Switch control is tested by setting trainPresent to be true for testing the situation when the train just entered the intersection. It is also tested when the train is about to leave the intersection. Semaphore is also tested by

OLED is tested with keypad to see if the right button can result into a right reading.  Specifically, by pressing button 0, the OLED will go back to home screen, only displaying “status” and “annuciation”. Similarly pressing select button will enter the next-level menu, or display the train information if selecting a train. Pressing the scroll button will move the cursor down the next entry, or to the top if the cursor reaches the bottom.

Temperature measurement task: first of all the group set the train is present and test if the voltage readings are correct and if the task outputs reading upon interrupt on a multimeter. Then the group test if all readings are stored in the buffer by examining values in watchdog. If the task has passed all tests, the group will ensure that it will determine the maximum of the three readings and signal a warning if the criterion meets.

Noise capture and processing: first of all the group test if the task is able to sample a sine wave isochronously. Later on the group examine the FFT result and determine if the returned index corresponds to the peak frequency.

Remote connection: this task is tested by first setting up the server and displaying the default page, and then importing various fixed data. If the display matches the actual value, the group will import the data struct and determine if the task is able to communicate with command task, and also update data in time.

Command task: this task is tested by typing commands on the network interface and examine if the output matches the design expectation.

**Summary**

This phase is generally a success. The group develops the system using IAR embedded workbench to program the TI Stellaris implementation of the ARM cortex microprocessor under the FreeROTS environment. The system utilizes the random number generator to resemble the size of incoming train, arriving direction as well as departing direction. In addition to the OLED, the system outputs switching information on the LCD and formatted train information on computer through serial port and on a remote computer through LAN. The system also allows the user to choose between status mode and annunciation mode through keypad and observe different display on OLED. It supports several commands transmitted through LAN to control the system as well. Same as the last phase, sound blasts and string flashing are preserved for different direction, but departing annunciation is different from arriving system. The system uses hardware timer with more accurate delay between two trains. As for the scheduler, FreeRTOS has an internal scheduler that handles schedules based on priority and dynamic scheduling. The group uses some improved functions from last lab and tests the new tasks separately and incorporates all tasks together for final testing

**Conclusion**

In general this lab is the collective work by all lab members. The group designs the final phase of the smart train management system. The train management system routes all incoming trains from a random direction to a desired direction at an intersection. It consists of ARM Cortex­M3 microcomputer and peripherals including an OLED,LCD, keypad and PWM generator. The biggest improvement of this phase is the use of FreeRTOS, with more powerful scheduling and multitasking. This phase also includes multiple measurement tasks that monitor engine noise, wheel temperature and distance. The design is generally a success, but with a few dissatisfactions. Keypad setup and connection is one of the frustrating parts, including the appropriate choices of resistors and voltage level, as well as the pin assignment. Additionally, the web server is setup outside of the RTOS, so even though access to server is guaranteed it does not satisfy the design spec. In this case the command task cannot be dynamically scheduled. The group also spends a substantial amount of time on debugging pointer and intercommunciation errors, which can be avoided by developing and testing individual tasks more thoroughly.

**Work Contributions**

|  |  |
| --- | --- |
| Operating System and Integration,Start Up | Bowen Xue |
| Schedule | Bowen Xue |
| Timer with Basic Hardware Timer | Bowen Xue |
| ISR-Train Arriving/Arriving Train/Departing Train | Bowen Xue |
| Train Communication | Bowen Xue |
| Switch Control | Bohan Wang |
| OLED | Bowen Xue |
| LCD | Bowen Xue |
| Local Keypad | Bowen Xue |
| Temperature | Ruiheng Wang |
| Local Communications | Ruiheng Wang |
| Remote Communications | Ruiheng Wang/Bohan Wang |
| Command Task | Bohan Wang |
| Noise Signature | Bowen Xue |
| Extra Creadits | Bowen Xue |
| Testing | Ruiheng Wang/Bowen Xue/Bohan Wang |
| Lab Report Writing | Ruiheng Wang/Bowen Xue/Bohan Wang |

**Appendix**

//Main

#define mainINCLUDE\_WEB\_SERVER 1

/\* Standard includes. \*/

#include <stdio.h>

/\* Scheduler includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "semphr.h"

/\* extra library\*/

#include "inc/hw\_types.h"

#include "driverlib/debug.h"

#include "driverlib/sysctl.h"

#include "drivers/rit128x96x4.h"

#include "inc/hw\_gpio.h"

#include "inc/hw\_ints.h"

#include "inc/hw\_memmap.h"

#include "driverlib/gpio.h"

#include "driverlib/interrupt.h"

#include "driverlib/timer.h"

#include "driverlib/pwm.h"

#include "driverlib/uart.h"

#include "driverlib/adc.h"

#include <stdlib.h>

#include <limits.h>

#include <optfft.h>

#include "semphr.h"

/\* Hardware library includes. \*/

#include "hw\_memmap.h"

#include "hw\_types.h"

#include "hw\_sysctl.h"

#include "sysctl.h"

#include "gpio.h"

#include "grlib.h"

#include "rit128x96x4.h"

#include "osram128x64x4.h"

#include "formike128x128x16.h"

/\* Demo app includes. \*/

#include "lcd\_message.h"

#include "bitmap.h"

#define GRIDLOCK -120

#define DELAY 2

#define ON 1

#define OFF 0

#define FS 7000

#define INCLUDE\_vTaskSuspend 1

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

typedef enum myBool bool;

//data struct +++ 0nly arrivig train

struct ArrivingTrainData

{

 unsigned short\* trainArriving;

 bool\* checkTrain;

 unsigned short\* arrivalDirection;

 unsigned short\* arrivingTrainDistance;//designer's choice

 unsigned short\* trainPresent;

 bool\* trainDepartingFlag;

};

typedef struct ArrivingTrainData ArrivingTrainData;

ArrivingTrainData arrivingTrainData;

struct TrainCom

{

 unsigned short\* trainArriving;

 unsigned short\* trainPresent;

 unsigned short\* trainSize;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 bool\* checkTrain;

 int\* trainDepartingQueue;

};

typedef struct TrainCom TrainCom;

TrainCom trainComData;

struct SwitchControl

{

 unsigned short\* departureDirection;

 unsigned short\* departingTrain;

 bool\* gridlock;

 unsigned short\* trainPresent;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 bool\* trainDepartingFlag;

 xSemaphoreHandle\* intersectionLock;

 int\* trainDepartingQueue;

 unsigned short\* arrivalDireciton;

};

typedef struct SwitchControl SwitchControl;

SwitchControl switchControlData;

struct DepartingTrainData

{

 unsigned short\* trainPresent;

 unsigned short\* departureDirection;

 int\* trainDepartingQueue;

 bool\* trainDepartingFlag;

};

typedef struct DepartingTrainData DepartingTrainData;

DepartingTrainData departingTrainData;

struct LcdDisplayData

{

 unsigned short\* trainPresent;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* departureDirection;

 unsigned short\* arrivalDirection;

 bool\* trainDepartingFlag;

};

typedef struct LcdDisplayData LcdDisplayData;

LcdDisplayData lcdDisplayData;

struct Oled

{

 unsigned short\* trainPresent;

 unsigned short\* trainArriving;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 unsigned short\* mode;

 unsigned short\* statusSelection;

 unsigned short\* select;

 unsigned short\* scroll;

 unsigned short\* annunciation;

 unsigned long\* temperatureBuf;

 unsigned short\* arrivingTrainDistance;

 bool\* trainDepartingFlag;

 bool\* gridlock;

 signed int\* noiseIndex;

 signed int\* processData;

 bool\* phasor;

 bool\* photon;

};

typedef struct Oled OledDisplay;

OledDisplay oledData;

struct LocalKeypadData

{

 unsigned short\* mode;

 unsigned short\* statusSelection;

 unsigned short\* scroll;

 unsigned short\* select;

 unsigned short\* annunciation;

};

typedef struct LocalKeypadData LocalKeypadData;

LocalKeypadData keypadData;

struct TempData

{

 unsigned short\* trainPresent;

 unsigned long\* temperatureBuf;

 bool\* trainDepartingFlag;

 unsigned long\* temp0;

 unsigned long\* temp1;

 unsigned long\* temp2;

};

typedef struct TempData TempData;

TempData tempData;

struct Serial

{

 unsigned short\* trainPresent;

 unsigned short\* trainArriving;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 xSemaphoreHandle\* intersectionLock;

 unsigned long\* temperatureBuf;

 bool\* gridlock;

 unsigned short\* arrivingTrainDistance;

};

typedef struct Serial SerialCom;

SerialCom serialData;

struct NoiseCaptureData

{

 bool\* processFlag;

};

typedef struct NoiseCaptureData NoiseCaptureData;

NoiseCaptureData noiseCaptureData;

struct NoiseProcessData

{

 signed int\* processData;

 bool\* processFlag;

 signed int\* noiseIndex;

};

typedef struct NoiseProcessData NoiseProcessData;

NoiseProcessData noiseProcessData;

struct MeasureDistanceData

{

 int\* meter;

};

typedef struct MeasureDistanceData MeasureDistanceData;

MeasureDistanceData measureDistanceData;

struct AttackData

{

 bool\* phasor;

 bool\* photon;

 int\* meter;

};

typedef struct AttackData AttackData;

AttackData attackData;

struct Command

{

 unsigned short\* trainPresent;

 unsigned short\* trainArriving;

 unsigned short\* arrivingTrainDistance;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 xSemaphoreHandle\* intersectionLock;

 unsigned long\* temperatureBuf;

 bool\* gridlock;

 char\* response;

 signed int\* noiseIndex;

 signed int\* processData;

 unsigned long\* temp0;

 unsigned long\* temp1;

 unsigned long\* temp2;

};

typedef struct Command CommandData;

CommandData commandData;

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

/\*

 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 )      //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 ( 45 )                   //3  oledSize

// 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 )  //8000000UL

/\* These are global variable\*/

unsigned short trainArrivingG =0;// TOT

int freCount =0;

unsigned short nn= 0;

unsigned long frequencyDetection= 0;

unsigned long arrivingCounter = 0;

int frequencyRead = 400;

int t =0;

int r =0;

unsigned long ulValue0 = 0;

unsigned long ulValue1 =0;

unsigned long ulValue2=0;

unsigned short arrivalDirectionG = 0;  //0

unsigned short departureDirectionG = 0;  //0

bool checkTrainG = FALSE;

unsigned short trainPresentG = 0; //FALSE

bool trainDepartingFlagG = FALSE;//FALSE;

unsigned short trainSizeG = 0 ;  //0

unsigned short traversalTimeG = 0; //0

bool gridlockG = FALSE;

   //keypad

unsigned short modeG = 0;

unsigned short trainStatusSelectionG = 0;;

unsigned short scrollG = 0;

unsigned short selectG = 0;

unsigned short annunciationG = 0;

unsigned int globalCounter = 0;

int seed = 1;

int index = 0;

bool annunciationFlag = FALSE;

short modeL = 0;

short statusSelectionL = 0;

short scrollL = 0;

short selectL = 0;

short annunciationL = 0;

int debouncerA = 1;

int debouncerB = 1;

int debouncerC = 1;

unsigned int delayCounterF1 = 0;

unsigned int delayCounterE0 = 0;

bool processFlagG = FALSE;

int fftIndex = 0;

int delayCounterE2 = 0;

int delayCounterE3= 0;

bool startCollection = FALSE;

bool removeIt = FALSE;

signed int processDataG[16] = {0,0,0,0,

                              0,0,0,0,

                              0,0,0,0,

                              0,0,0,0};

int trainDepartingQueueG[20] = {0,0,0,0,0,

                               0,0,0,0,0,

                               0,0,0,0,0,

                               0,0,0,0,0}; // max 20 train in the system

unsigned long temperatureBufG[16] = {0,0,0,0,

                                     0,0,0,0,

                                     0,0,0,0,

                                     0,0,0,0};

unsigned short arrivingTrainDistanceG[8]= {0,0,0,0,

                                          0,0,0,0};

signed int real[256];

signed int realCompute[256];

signed int imag[256];

signed int noiseIndexG = 0;

int meterG = 80;

bool afterCollection = FALSE;

unsigned long voltageADC = 0;

unsigned long temp0 = 0;

unsigned long temp1=0;

unsigned long temp2=0;

bool phasorG = FALSE;

bool photonG = FALSE;

char responseG[10]="Hello";

//\*end \*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

/\*

\* 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 );

/\*

 three dummy tasks of different priorities that simply run, announce

 themselves, then sleep

\*/

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

/\*

 The queue used to send messages to the OLED task.

\*/

xQueueHandle xOLEDQueue;

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

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

\* 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.

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

#ifdef DEBUG

void

\_\_error\_\_(char \*pcFilename, unsigned long ulLine)

{

}

#endif

/\*                   handlers                   \*/

void delay(unsigned long aValue);

void Timer1IntHandler(void)

{

   TimerIntClear(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

   globalCounter++;

 arrivingCounter++;

}

void GPIOAIntHandler(void)

{

 GPIOPinIntClear(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);

 frequencyDetection++;

}

void GPIOFIntHandler(void)

{

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

 trainArrivingG++;

 delayCounterF1 = globalCounter;

 IntDisable(INT\_GPIOF);

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

}

void UARTIntHandler(void)

{

   unsigned long ulStatus;

   ulStatus = UARTIntStatus(UART0\_BASE, true);

   UARTIntClear(UART0\_BASE, ulStatus);

   while(UARTCharsAvail(UART0\_BASE))

   {

     UARTCharPutNonBlocking(UART0\_BASE, UARTCharGetNonBlocking(UART0\_BASE));

   }

}

void GPIOEIntHandler(void)

{

 if(0x01 == GPIOPinIntStatus(GPIO\_PORTE\_BASE, false))

 {

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

   t++;

   IntDisable(INT\_GPIOE);

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

 }

 if(0x04 == GPIOPinIntStatus(GPIO\_PORTE\_BASE, false))

 {

   GPIOPinIntClear(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

   delayCounterE2 = globalCounter;

   removeIt= TRUE;

 }

 if(0x08 ==GPIOPinIntStatus(GPIO\_PORTE\_BASE, false))

 {

   GPIOPinIntClear(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

   delayCounterE3 = globalCounter;

   startCollection = TRUE;

 //  IntDisable(INT\_GPIOE);

 //  GPIOPinIntDisable(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

 }

}

void ADC3Handler(void)

{

 ADCIntClear(ADC\_BASE, 3);

 if(fftIndex<256 && startCollection)

 {

   ADCSequenceDataGet(ADC\_BASE, 3 , real+fftIndex);

   fftIndex++;

 }

 else if(fftIndex == 256)

 {

   startCollection = FALSE;

   processFlagG = TRUE;

   TimerDisable(TIMER0\_BASE, TIMER\_A);

   TimerControlTrigger(TIMER0\_BASE,TIMER\_A, false);

   fftIndex=0;

 }

}

/\*                  myProto                    \*/

int randomInteger(int low, int high);

int getDirection (int , int, int);

void trainComTask(void\* trainComDataPtr);

void switchControlTask(void\* switchControlDataPtr);

void departingTrainTask(void\* departingTrain);

void clear(void);

void frequencyWrite();

void arrivingTrainTask(void\* arrivingTrainData);

void lcdDisplayTask(void\* lcdDisplay);

void oledTask(void\* oled);

void scrollSelection(void);

void annuFunc(void);

void trainStatus(void);

void trainStatus(void);

void modeSelect(void);

void keypadTask(void\* dataPtr);

void temperatureMeasureTask(void\* tempData);

unsigned long findMax(unsigned long temp0, unsigned long temp1, unsigned long temp2);

void serialTask(void\* serial);

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

void noiseCaptureTask(void\* data);

void noiseProcessTask(void\* data);

void measureDistanceTask(void\* data);

void attackTask(void\* data);

//              lcd            //

void enable(void);

void initialize(void);

void write\_A(void);

void write\_S(void);

void write\_I(void);

void write\_Z(void);

void write\_E(void);

void write\_P(void);

void write\_R(void);

void write\_N(void);

void write\_T(void);

void write\_W(void);

void write\_M(void);

void write\_O(void);

void write\_U(void);

void write\_H(void);

void write\_1(void);

void write\_2(void);

void write\_3(void);

void write\_4(void);

void write\_5(void);

void write\_6(void);

void write\_7(void);

void write\_8(void);

void write\_9(void);

void write\_0(void);

void newLine(void);

void shiftRight(void);

void clear(void);

void returnHome(void);

void screenLeft();

void cursorOff();

void write\_dash();

void write\_greater();

//\*\*\*\*\*\*\*\*\*\*\*semaphor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

xSemaphoreHandle intersectionLockG;

xTaskHandle trainComHandle;

xTaskHandle lcdHandle;

int c = 0;

int main( void )

{

   prvSetupHardware();

   /\*

       Create the queue used by the OLED task.  Messages for display on the OLED

       are received via this queue.

   \*/

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

   /\*

       Exclude some tasks if using the kickstart version to ensure we stay within

       the 32K code size limit.

   \*/

   #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

//   SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN |

//                  SYSCTL\_XTAL\_8MHZ);

   RIT128x96x4Init(1000000);

   //\*\*\*\*\*\*\*start up, hardware timer\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER1);

   IntMasterEnable();

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

   TimerLoadSet(TIMER1\_BASE, TIMER\_A, SysCtlClockGet()/10);

   IntEnable(INT\_TIMER1A);

   TimerIntEnable(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

   TimerEnable(TIMER1\_BASE, TIMER\_A);

   //\*\*\*\*\*\*end timer \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*configure the sound\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   unsigned long ulPeriod;

   SysCtlPWMClockSet(SYSCTL\_PWMDIV\_1);

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM0);

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOG);

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

   ulPeriod = SysCtlClockGet() / 440;

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

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

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

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

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

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

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

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*approaching frequency\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   unsigned long period;

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB);

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

   period = SysCtlClockGet() / 400; //start at 400 Hz

   PWMGenConfigure(PWM\_BASE, PWM\_GEN\_1,

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

   PWMGenPeriodSet(PWM\_BASE, PWM\_GEN\_1, period);

   PWMPulseWidthSet(PWM\_BASE, PWM\_OUT\_2, period / 2);

   PWMOutputState(PWM\_BASE, PWM\_OUT\_2\_BIT, TRUE);

   PWMGenEnable(PWM\_BASE, PWM\_GEN\_1);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*frequency read\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

   GPIOPinTypeGPIOInput(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);  //  read5--> change to read frequency

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

                    GPIO\_PIN\_TYPE\_STD\_WPD);

   GPIOIntTypeSet(GPIO\_PORTA\_BASE, GPIO\_PIN\_4, GPIO\_FALLING\_EDGE);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*frequency read end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*lcd\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOC);

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

   GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_4);  //D4

   GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_5);  //D5

   GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_6);  //D6

   GPIOPinTypeGPIOOutput(GPIO\_PORTC\_BASE, GPIO\_PIN\_7);  //D7

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

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

   initialize();

   delay(DELAY);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*lcd end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\* keypad\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   GPIOPinTypeGPIOInput(GPIO\_PORTB\_BASE, GPIO\_PIN\_4); // 35  five

   GPIOPinTypeGPIOInput(GPIO\_PORTB\_BASE, GPIO\_PIN\_5); //38   six

   GPIOPinTypeGPIOInput(GPIO\_PORTB\_BASE, GPIO\_PIN\_6); //37    seven

   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);

   GPIOPadConfigSet(GPIO\_PORTB\_BASE, GPIO\_PIN\_6, GPIO\_STRENGTH\_2MA,

                    GPIO\_PIN\_TYPE\_STD\_WPD);

   //\*\*\*\*\*\*\*\*\*\*\*\*end keypad\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*temperature take button\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);

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

   GPIOPinTypeGPIOInput(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

   GPIOPinTypeGPIOInput(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

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

                    GPIO\_PIN\_TYPE\_STD\_WPU);

   GPIOPadConfigSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_2, GPIO\_STRENGTH\_2MA,

                    GPIO\_PIN\_TYPE\_STD\_WPU);

   GPIOPadConfigSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA,

                    GPIO\_PIN\_TYPE\_STD\_WPU);

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

   GPIOIntTypeSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_2, GPIO\_FALLING\_EDGE);

   GPIOIntTypeSet(GPIO\_PORTE\_BASE, GPIO\_PIN\_3, GPIO\_FALLING\_EDGE);

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

   GPIOPinIntEnable(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

   GPIOPinIntEnable(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

   IntEnable(INT\_GPIOE);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*end temperature take\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

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

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

   ADCSequenceEnable(ADC0\_BASE, 0);

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

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

   ADCSequenceEnable(ADC0\_BASE, 1);

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

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

   ADCSequenceEnable(ADC0\_BASE, 2);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*end ADC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*sw2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   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);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end sw2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UART\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

   IntMasterEnable();

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

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

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

                        UART\_CONFIG\_PAR\_NONE));

   IntEnable(INT\_UART0);

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

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end UART\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADC timer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC);

   SysCtlADCSpeedSet(SYSCTL\_ADCSPEED\_500KSPS);  //125 250 500 1M

   ADCSequenceConfigure(ADC\_BASE, 3, ADC\_TRIGGER\_TIMER, 0);

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

   SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

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

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

   TimerControlTrigger(TIMER0\_BASE, TIMER\_A, false);

   ADCSequenceEnable(ADC\_BASE, 3);

   ADCIntEnable(ADC\_BASE, 3);

   IntEnable(INT\_ADC3);

   //\*\*\*\*\*\*\*\*\*\*\*\*\*end ADC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   //\*\*\*\*\*\*\*\*\*\*\*semaphor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   intersectionLockG = xSemaphoreCreateCounting(1,1);

   //data initialization

   arrivingTrainData.trainArriving = &trainArrivingG;

   arrivingTrainData.checkTrain = &checkTrainG;

   arrivingTrainData.arrivalDirection = &arrivalDirectionG;

   arrivingTrainData.arrivingTrainDistance = arrivingTrainDistanceG;

   arrivingTrainData.trainPresent = &trainPresentG;

   arrivingTrainData.trainDepartingFlag = &trainDepartingFlagG;

   //\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

   trainComData.arrivalDirection = &arrivalDirectionG;

   trainComData.checkTrain = &checkTrainG;

   trainComData.departureDirection = &departureDirectionG;

   trainComData.trainArriving = &trainArrivingG;

   trainComData.trainPresent= &trainPresentG;

   trainComData.trainSize = &trainSizeG;

   trainComData.trainDepartingQueue= trainDepartingQueueG;

   switchControlData.departureDirection = &departureDirectionG;

   switchControlData.gridlock = &gridlockG;

   switchControlData.trainPresent = &trainPresentG;

   switchControlData.trainSize = &trainSizeG;

   switchControlData.traversalTime = &traversalTimeG;

   switchControlData.trainDepartingQueue = trainDepartingQueueG;

   switchControlData.trainDepartingFlag = &trainDepartingFlagG;

   switchControlData.intersectionLock = &intersectionLockG;

   switchControlData.arrivalDireciton = &arrivalDirectionG;

   departingTrainData.departureDirection = &departureDirectionG;

   departingTrainData.trainPresent = &trainPresentG;

   departingTrainData.trainDepartingFlag = &trainDepartingFlagG;

   departingTrainData.trainDepartingQueue = trainDepartingQueueG;

   lcdDisplayData.trainSize = &trainSizeG;

   lcdDisplayData.trainDepartingFlag = &trainDepartingFlagG;

   lcdDisplayData.arrivalDirection = &arrivalDirectionG;

   lcdDisplayData.trainPresent = &trainPresentG;

   lcdDisplayData.departureDirection = &departureDirectionG;

   lcdDisplayData.traversalTime = &traversalTimeG;

   oledData.annunciation = &annunciationG;

   oledData.arrivalDirection = &arrivalDirectionG;

   oledData.departureDirection = &departureDirectionG;

   oledData.mode = &modeG;

   oledData.scroll = &scrollG;

   oledData.select = &selectG;

   oledData.statusSelection = &trainStatusSelectionG;

   oledData.trainArriving = &trainArrivingG;

   oledData.trainPresent = &trainPresentG;

   oledData.trainSize = &trainSizeG;

   oledData.traversalTime = &traversalTimeG;

   oledData.gridlock = &gridlockG;

   oledData.temperatureBuf = temperatureBufG;

   oledData.arrivingTrainDistance = arrivingTrainDistanceG;

   oledData.trainDepartingFlag = &trainDepartingFlagG;

 oledData.noiseIndex = &noiseIndexG;

   oledData.processData = processDataG;

   oledData.phasor = &phasorG;

   oledData.photon = &photonG;

   keypadData.mode = &modeG;

   keypadData.statusSelection = &trainStatusSelectionG;

   keypadData.scroll = &scrollG;

   keypadData.select = &selectG;

   keypadData.annunciation = &annunciationG;

   tempData.temperatureBuf = temperatureBufG;

   tempData.trainPresent = &trainPresentG;

   tempData.trainDepartingFlag = &trainDepartingFlagG;

   tempData.temp0=&temp0;

   tempData.temp1=&temp1;

   tempData.temp2=&temp2;

   serialData.gridlock = &gridlockG;

   serialData.trainPresent = &trainPresentG;

   serialData.trainArriving = &trainArrivingG;

   serialData.trainSize = &trainSizeG;

   serialData.traversalTime = &traversalTimeG;

   serialData.arrivalDirection = &arrivalDirectionG;

   serialData.departureDirection = &departureDirectionG;

   serialData.intersectionLock = &intersectionLockG;

   serialData.temperatureBuf = temperatureBufG;

   serialData.arrivingTrainDistance=arrivingTrainDistanceG;

   noiseCaptureData.processFlag = &processFlagG;

   noiseProcessData.processFlag = &processFlagG;

   noiseProcessData.processData = processDataG;

   noiseProcessData.noiseIndex= &noiseIndexG;

   measureDistanceData.meter= &meterG;

   attackData.phasor = &phasorG;

   attackData.photon = &photonG;

   attackData.meter = &meterG;

   commandData.gridlock = &gridlockG;

   commandData.trainPresent = &trainPresentG;

   commandData.trainArriving = &trainArrivingG;

   commandData.trainSize = &trainSizeG;

   commandData.traversalTime = &traversalTimeG;

   commandData.arrivalDirection = &arrivalDirectionG;

   commandData.departureDirection = &departureDirectionG;

   commandData.intersectionLock = &intersectionLockG;

   commandData.temperatureBuf = temperatureBufG;

   commandData.arrivingTrainDistance = arrivingTrainDistanceG;

   commandData.response = responseG;

   commandData.noiseIndex = &noiseIndexG;

   commandData.processData = processDataG;

   commandData.temp0 = &temp0;

   commandData.temp1 = &temp1;

   commandData.temp2 = &temp2;

   /\* Start the tasks \*/

   xTaskCreate( vOLEDTask, ( signed portCHAR \* ) "OLED", mainOLED\_TASK\_STACK\_SIZE, NULL, tskIDLE\_PRIORITY, NULL );

   xTaskCreate(arrivingTrainTask, "TOTArriving" , 100, (void\*)&arrivingTrainData, 8, NULL);

   xTaskCreate(oledTask, "TOTOled" , 1500, (void\*)&oledData, 7, NULL);

   xTaskCreate(keypadTask, "TOTKeypad" , 100, (void\*)&keypadData, 1, NULL);

   xTaskCreate(trainComTask, "TOTTrainCom" , 100, (void\*)&trainComData, 7, &trainComHandle);

   xTaskCreate(switchControlTask, "TOTSwitch" , 100, (void\*)&switchControlData, 6, NULL);

   xTaskCreate(departingTrainTask, "TOTDeparting" , 100, (void\*)&departingTrainData, 5, NULL);

   xTaskCreate(lcdDisplayTask, "TOTLcd" , 100, (void\*)&lcdDisplayData, 2, &lcdHandle);

   xTaskCreate(temperatureMeasureTask, "TOTTemp" , 100, (void\*)&tempData, 6, NULL);

   xTaskCreate(serialTask, "TOTSerial" , 500, (void\*)&serialData, 5, NULL);

   xTaskCreate(noiseCaptureTask, "TOTCapture" , 1000, (void\*)&noiseCaptureData, 7, NULL);

   xTaskCreate(noiseProcessTask, "TOTProcess" , 1000, (void\*)&noiseProcessData, 6, NULL);

   xTaskCreate(measureDistanceTask,"TOTDistance", 100, (void\*)&measureDistanceData, 5, NULL);

   xTaskCreate(attackTask, "TOTAttack", 100, (void\*)&attackData, 5, NULL);

   vTaskSuspend( trainComHandle);

   vTaskStartScheduler();

   /\* Will only get here if there was insufficient memory to create the idle task. \*/

   return 0;

}

/\*

 three dummy tasks

\*/

/\*

 the OLED Task

\*/

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 );

   }

}

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

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 portLONGulTicksSinceLastDisplay = 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;

   }

}

/\*                               myFunction                              \*/

int randomInteger(int low, int high)

{

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;

}

return retVal;

}

void frequencyWrite()

{

 static unsigned long preCounter =0;

 static int frequencyCounter = 1;

 static int ulPeriod;

 //freqency generating, and output through GPIO

 if( (400 + 10 \* frequencyCounter) <= 590) //280

 {

   if(!((arrivingCounter - preCounter)%5)) // frequency increments itself very 0.5 second, it will change 18 times

   {

     GPIOPinTypePWM(GPIO\_PORTB\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

     PWMGenDisable(PWM\_BASE, PWM\_GEN\_1);

     PWMOutputState(PWM\_BASE, PWM\_OUT\_2\_BIT, 0);

     ulPeriod = SysCtlClockGet() / (int)frequencyWrite;

     PWMGenPeriodSet(PWM\_BASE, PWM\_GEN\_1, SysCtlClockGet() / (400 + 10 \* frequencyCounter));

     PWMPulseWidthSet(PWM\_BASE, PWM\_OUT\_2, SysCtlClockGet() / (400 + 10 \* frequencyCounter) /2 );

     PWMOutputState(PWM\_BASE, PWM\_OUT\_2\_BIT, 1);

     PWMGenEnable(PWM\_BASE, PWM\_GEN\_1);

     frequencyCounter++;

     preCounter = arrivingCounter;

   }

 }

 else

 {

   IntDisable(INT\_GPIOA);

   GPIOPinIntDisable(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);

   frequencyCounter = 1;

   preCounter = 0;

 }

}

void arrivingTrainTask(void\* arrivingTrainData)

{

  static ArrivingTrainData\* arrivingPtr = NULL;

  arrivingPtr = (ArrivingTrainData\*)arrivingTrainData;

// ArrivingTrainData\* arrivingPtr = &arrivingTrainData;

  xOLEDMessage xMessage;

 while(1)

 {

  //     trainhere

   if(\*(arrivingPtr->trainArriving))

   {

     static unsigned int firstTime = 1;

     static unsigned int firstTimeCheckTrain = 1;

     static int distance = 0;

     static int preDistance = 1500;

     static int offset = 0;

     static int pwmDisable = 1;

     static int onceA = 1;

     static int onceB = 0;

    // static int timeValue = 0;

     if(firstTime)

     {

       short s1 = randomInteger(0,1);

       short s0 = randomInteger(0,1);

       if(s1 == 0 && s0 ==0) //north

       {

         \*arrivingPtr->arrivalDirection  = 1;

       }

       else if(s1 == 0 && s0 == 1) // south

       {

         \*arrivingPtr->arrivalDirection  = 2;

       }

       else if(s1 == 1 && s0 == 0) //east

       {

         \*arrivingPtr->arrivalDirection  = 3;

       }

       else if(s1 == 1 && s0 == 1)  //west

       {

         \*arrivingPtr->arrivalDirection  = 4;

       }

       offset = arrivingCounter;

       firstTime = 0;

       PWMOutputState(PWM\_BASE, PWM\_OUT\_2\_BIT, true);

       PWMGenEnable(PWM\_BASE, PWM\_GEN\_1);

       GPIOPinIntEnable(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);

       IntEnable(INT\_GPIOA);

     }

     if(!(\*arrivingPtr->trainDepartingFlag)) //departing train takes higher prority

     {

       switch(\*arrivingPtr->arrivalDirection){

       case 1: //north train

         if( 0 == (arrivingCounter - offset)%110 || 30 == (arrivingCounter - offset)%110 || 60 == (arrivingCounter - offset)%110 || 90 == (arrivingCounter - offset)%110)

         {

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

         }

         else if( 20 == (arrivingCounter - offset)%110|| 50 == (arrivingCounter - offset)%110 || 80 == (arrivingCounter - offset)%110 || 100 == (arrivingCounter - offset)%110)

         {

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

         }

         break;

       case 2: //south

         if(0 == (arrivingCounter - offset)%130 || 30 == (arrivingCounter - offset)%130 || 60 == (arrivingCounter - offset)%130 || 90 == (arrivingCounter - offset)%130)

         {

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

         }

         else if(20 == (arrivingCounter - offset)%130|| 50 == (arrivingCounter - offset)%130 || 80 == (arrivingCounter - offset)%130 || 120 == (arrivingCounter - offset)%130)

         {

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

         }

         break;

       case 3: //east

         if(0 == (arrivingCounter - offset)%120 || 40 == (arrivingCounter - offset)%120 || 70== (arrivingCounter - offset)%120 || 100 == (arrivingCounter - offset)%120)

         {

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

         }

         else if(30== (arrivingCounter - offset)%120 || 60 == (arrivingCounter - offset)% 120 || 90 == (arrivingCounter - offset)%120 || 110 == (arrivingCounter - offset)%120)

         {

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

         }

         break;

       case 4: //west

         if(0 == (arrivingCounter - offset)%100 || 20 == (arrivingCounter - offset)%100 || 50 == (arrivingCounter - offset)%100 || 80 == (arrivingCounter - offset)%100 )

         {

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

         }

         else if(10 == (arrivingCounter - offset)%100 || 40== (arrivingCounter - offset)%100 || 70 == (arrivingCounter - offset)%100 || 90 == (arrivingCounter - offset)%100)

         {

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

         }

         break;

       default: break;

       }

     }

     if(arrivingCounter - offset >= 50) //50 is the delayTime

     {

       frequencyRead = frequencyDetection \* 10;

       frequencyWrite();

       //...frequency  read

       frequencyDetection = 0;  // reset to start a new reading

       distance = (int)(-5 \* frequencyRead) + 3000;

       if((preDistance - distance)\*10 >  preDistance )

       {

         if(distance > 100)

         {

           preDistance = distance;

         }

         \*(arrivingPtr->arrivingTrainDistance + index%8) = (int)preDistance;

         index++;

       }

       if(preDistance < 400 && firstTimeCheckTrain)

       {

         \*arrivingPtr->checkTrain = TRUE;

         firstTimeCheckTrain = 0;

       }

     }

     if(preDistance< 100 && pwmDisable)

     {

       PWMGenDisable(PWM\_BASE, PWM\_GEN\_1);

       PWMOutputState(PWM\_BASE, PWM\_OUT\_2\_BIT, 0);

       pwmDisable = 0;

     }

     //\*\*\*reset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

     if(arrivingCounter - offset >= 150) // it has passed

     {

       vTaskResume(trainComHandle);

       (\*arrivingPtr->trainArriving)--;

       (\*arrivingPtr->trainPresent)++;

       offset = 0;

       firstTime = 1; // start a new train

       distance = 0;

       index = 0;

       firstTimeCheckTrain = 1;

       frequencyRead = 400;

       frequencyDetection = 0;

       preDistance = 3000;

       pwmDisable = 1;

       PWMOutputState(PWM0\_BASE, PWM\_OUT\_1\_BIT , false); // no sound

       arrivingCounter = 0;

       int j = 0;

       for(j = 0; j<8 ;j++)

       {

         \*(arrivingPtr->arrivingTrainDistance+j) = 0;

       }

     }

   }

   if(globalCounter - delayCounterF1 == 15 && 0 != trainArrivingG)

   {

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

     IntEnable(INT\_GPIOF);

   }

   vTaskDelay(100);

 }

}

void trainComTask(void\* trainComDataPtr)

{

 TrainCom\* trainComData = (TrainCom\*) trainComDataPtr;

 while(1)

 {

   if(\*trainComData->checkTrain)

   {

     int size = 0;

     size =  randomInteger(1,9);

     int outDirection = 0;

     outDirection = getDirection(0, 3, \*trainComData->arrivalDirection);

     int j = 0;

     while(\*(trainComData->trainDepartingQueue+j))

     {

       j++;

     }

     //put the new data at the end

     \*(trainComData->trainDepartingQueue+j) =  outDirection;

     j++;

     \*(trainComData->trainDepartingQueue+j) = size;

     //take the old data at the head

     \*trainComData->departureDirection = \*(trainComData->trainDepartingQueue+ 0);

     \*trainComData->trainSize = \*(trainComData->trainDepartingQueue + 1); //always takes the first

     \*trainComData->checkTrain = FALSE;

   }

   vTaskSuspend(trainComHandle);

   vTaskDelay(100);

 }

}

//to get a different outbound direction than inbound direction

int getDirection(int low, int high, int direction)

{

 int outDirection = randomInteger(0,3);

 if(outDirection == direction)

 {

   outDirection = getDirection(0, 3, direction);

 }

 return outDirection;

}

void switchControlTask(void\* switchControlDataPtr)

{

 SwitchControl\* switchPtr = (SwitchControl\*) switchControlDataPtr;

 static int n = 0;

 static unsigned long offset = 0;

 static int firstTime = 1;

 static int gridWait = 0;

 static int unlockGrid = 0;

 static int timeElapsed = 0;

 static int passTime =0;

 while(1)

 {

   if (\*(switchPtr -> trainPresent) )

   {

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

     if(xSemaphoreTake(intersectionLockG,( portTickType ) 20 ) ==pdTRUE)

     {

       n = randomInteger(-3, 3);

       nn = n;

       offset = globalCounter;

       if(n < 0)

       {

         gridWait = GRIDLOCK \* n;

         \*switchPtr->gridlock = TRUE;

       }

       else

       {

         gridWait = 0;

       }

       unlockGrid = globalCounter + gridWait;

       passTime = globalCounter + gridWait + 10\*(\*(switchPtr -> traversalTime));

       firstTime = 0;

     }

     \*switchPtr->trainDepartingFlag = TRUE;

     if( \*switchPtr->gridlock )

     {

       int disruptorFire = randomInteger(-3, 3);

       timeElapsed = globalCounter -offset;

       if(disruptorFire >= 0 )

       {

         \*switchPtr->gridlock = FALSE;

         unlockGrid= 0;

         passTime = offset + timeElapsed + 10\*(\*(switchPtr -> traversalTime));

         timeElapsed = 0;

       }

       else

       {

         int thermonuclear =randomInteger(-3, 3);

         if(thermonuclear> 0 )

         {

           \*switchPtr->gridlock = FALSE;

           unlockGrid= 0;

           passTime = offset + timeElapsed + 10\*(\*(switchPtr -> traversalTime));

           timeElapsed = 0;

         }

       }

     }

     // unlock the grid

     if(globalCounter >= unlockGrid && n < 0)

     {

       \*switchPtr->gridlock = FALSE;

       unlockGrid = 0;

     }

     // resetting

     if (globalCounter > passTime)

     {

       //when the train has passed do a complete reset

       \*switchPtr -> departureDirection = 0;

       \*switchPtr->trainSize = 0;

       \*switchPtr->trainDepartingFlag = FALSE;

       (\*switchPtr->trainPresent)--;

       \*(switchPtr -> traversalTime) = 0;

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

       passTime = 0;

       offset =0;

       clear();  //clear lcd

       delay(2\*DELAY);

       firstTime = 1;

       n = 0;

       globalCounter = 0;

       //update the queue

       int j = 0;

       while(\*(switchPtr->trainDepartingQueue+j)&& j<20)

       {

         \*(switchPtr->trainDepartingQueue+j)= \*(switchPtr->trainDepartingQueue+j+2);

         \*(switchPtr->trainDepartingQueue+j+1)= \*(switchPtr->trainDepartingQueue+j+3);

         j+=2;

       }

       if(\*switchPtr->trainDepartingQueue != 0)

       {

         \*switchPtr->departureDirection = \*switchPtr->trainDepartingQueue;

         \*switchPtr->trainSize = \*(switchPtr->trainDepartingQueue+1);

       }

       if(\*switchPtr->trainPresent == 0 )

       {

         \*switchPtr->arrivalDireciton= 0;

       }

       // release the semaphore

       xSemaphoreGive(intersectionLockG);

     }

   }

   vTaskDelay(100);

 }

}

void departingTrainTask(void\* departingTrain)

{

 DepartingTrainData\* departingPtr = (DepartingTrainData\*) departingTrain;

 static unsigned long offsetDeparting = 0;

 static int firstTime = 1;

 while(1)

 {

   if (\*departingPtr->trainDepartingFlag)

   {

     if(firstTime)

     {

       offsetDeparting = globalCounter;

       firstTime = 0;

     }

     switch(\*departingPtr->departureDirection){

      case 1: //north train

        if( 0 == (globalCounter - offsetDeparting)%90 || 20 == (globalCounter - offsetDeparting)%90  || 40 == (globalCounter - offsetDeparting)%90 || 70 == (globalCounter - offsetDeparting)%90)

        {

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

        }

        else if( 10 == (globalCounter - offsetDeparting)%90 || 30 == (globalCounter - offsetDeparting)% 90 || 60 == (globalCounter - offsetDeparting)%90 || 80 == (globalCounter - offsetDeparting)%90)

        {

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

        }

        break;

      case 2: //south

        if(0 == (globalCounter - offsetDeparting)% 100 || 20 == (globalCounter - offsetDeparting)%100 || 50 == (globalCounter - offsetDeparting)%100 || 80 == (globalCounter - offsetDeparting)%100)

        {

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

        }

        else if(10 == (globalCounter - offsetDeparting)%100 || 40 == (globalCounter - offsetDeparting)%100 || 70 == (globalCounter - offsetDeparting)%100 || 90 == (globalCounter - offsetDeparting)%100)

        {

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

        }

        break;

      case 3: //east

        if(0 == (globalCounter - offsetDeparting)%110 || 20 == (globalCounter - offsetDeparting) % 110 || 60== (globalCounter - offsetDeparting)%110 || 90== (globalCounter - offsetDeparting)%110)  //east

        {

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

        }

        else if( 10== (globalCounter - offsetDeparting)%110 || 50 == (globalCounter - offsetDeparting)% 110 || 80 == (globalCounter - offsetDeparting)%110 || 100== (globalCounter - offsetDeparting)%110)

        {

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

        }

        break;

      case 4: //west

        if(0 == (globalCounter - offsetDeparting) % 120 || 20 == (globalCounter - offsetDeparting)%120 || 70 == (globalCounter - offsetDeparting)%120 || 100 == (globalCounter - offsetDeparting)%120)

        {

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

        }

        else if(10 == (globalCounter - offsetDeparting) % 120 || 60 == (globalCounter - offsetDeparting) % 120 || 90 == (globalCounter - offsetDeparting) % 120 || 110 == (globalCounter - offsetDeparting)%120)

        {

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

        }

        break;

      default:

        break;

      }

    }

   //condition to reset the

   if(!(\*departingPtr->trainDepartingFlag) && offsetDeparting) //no trainpresent

   {

     offsetDeparting = 0;

     firstTime = 1;

   }

   vTaskDelay(100);

 }

}

void lcdDisplayTask(void\* lcdDisplay)

{

 LcdDisplayData\* lcdPtr = (LcdDisplayData\*) lcdDisplay;

 static int firstTime = 1;

 while(1)

 {

   if(\*lcdPtr->trainDepartingFlag)

   {

     if(firstTime)

     {

       // cursorOff();

       delay(DELAY);

       write\_P();

       delay(DELAY);

       write\_R();

       delay(DELAY);

       write\_S();

       delay(DELAY);

       write\_T();

       delay(DELAY);

       shiftRight();

       delay(DELAY);

       switch(\*lcdPtr->trainPresent){

       case 1: write\_1(); delay(DELAY); break;

       case 2: write\_2(); delay(DELAY); break;

       case 3: write\_3(); delay(DELAY); break;

       case 4: write\_4(); delay(DELAY); break;

       case 5: write\_5(); delay(DELAY); break;

       case 6: write\_6(); delay(DELAY); break;

       case 7: write\_7(); delay(DELAY); break;

       case 8: write\_8(); delay(DELAY); break;

       case 9: write\_9(); delay(DELAY); break;

       default: break;

       }

       shiftRight();

       delay(DELAY);

       switch (\*lcdPtr->arrivalDirection){

       case 1: write\_N(); delay(DELAY); break;

       case 2: write\_S(); delay(DELAY); break;

       case 3: write\_E(); delay(DELAY); break;

       case 4: write\_W(); delay(DELAY); break;

       default: break;

       }

       write\_dash();

       delay(DELAY);

       write\_greater();

       delay(DELAY);

       switch (\*lcdPtr->departureDirection){

       case 1: write\_N(); delay(DELAY); break;

       case 2: write\_S(); delay(DELAY); break;

       case 3: write\_E(); delay(DELAY); break;

       case 4: write\_W(); delay(DELAY); break;

       }

       shiftRight();

       delay(DELAY);

       returnHome();

       delay(5\*DELAY);

       newLine();

       delay(5\*DELAY);

       write\_S();

       delay(DELAY);

       write\_I();

       delay(DELAY);

       write\_Z();

       delay(DELAY);

       write\_E();

       delay(DELAY);

       shiftRight();

       delay(DELAY);

       switch(\*lcdPtr->trainSize){

       case 1:  write\_1(); delay(DELAY); break;

       case 2:  write\_2(); delay(DELAY); break;

       case 3:  write\_3(); delay(DELAY); break;

       case 4:  write\_4(); delay(DELAY); break;

       case 5:  write\_5(); delay(DELAY); break;

       case 6:  write\_6(); delay(DELAY); break;

       case 7:  write\_7(); delay(DELAY); break;

       case 8:  write\_8(); delay(DELAY); break;

       case 9:  write\_9(); delay(DELAY); break;

       default: write\_0(); delay(DELAY); break;

       }

       shiftRight();

       delay(DELAY);

       write\_T();

       delay(DELAY);

       write\_I();

       delay(DELAY);

       write\_M();

       delay(DELAY);

       write\_E();

       delay(DELAY);

       shiftRight();

       delay(DELAY);

       shiftRight();

delay(DELAY);

       switch(\*lcdPtr->traversalTime/10){

       case 1:  write\_1(); delay(DELAY); break;

       case 2:  write\_2(); delay(DELAY); break;

       case 3:  write\_3(); delay(DELAY); break;

       case 4:  write\_4(); delay(DELAY); break;

       case 5:  write\_5(); delay(DELAY); break;

       case 6:  write\_6(); delay(DELAY); break;

       case 7:  write\_7(); delay(DELAY); break;

       case 8:  write\_8(); delay(DELAY); break;

       case 9:  write\_9(); delay(DELAY); break;

       default: write\_0(); delay(DELAY); break;

       }

       switch(\*lcdPtr->traversalTime%10){

       case 1:  write\_1(); delay(DELAY); break;

       case 2:  write\_2(); delay(DELAY); break;

       case 3:  write\_3(); delay(DELAY); break;

       case 4:  write\_4(); delay(DELAY); break;

       case 5:  write\_5(); delay(DELAY); break;

       case 6:  write\_6(); delay(DELAY); break;

       case 7:  write\_7(); delay(DELAY); break;

       case 8:  write\_8(); delay(DELAY); break;

       case 9:  write\_9(); delay(DELAY); break;

       default: write\_0(); delay(DELAY); break;

       }

       firstTime = 0;

     }

   }

   else

   {

     firstTime = 1;

   }

   vTaskDelay(100);

 }

}

//                lcd  driver                   //

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;

}

void enable()

{

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

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

}

void initialize( void )

{

 //initialize

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 |GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6, 0x00); //function set

 delay(1000); // >40ms

 //function set

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 |GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 delay(DELAY);

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4 , 0x20);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 , 0xC0);

 enable();

 delay(DELAY);

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4 , 0x20);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 , 0xC0);

 enable();

 delay(DELAY);

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_6 | GPIO\_PIN\_7, 0x00);  // function set

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_6 | GPIO\_PIN\_7 , 0xF0);  // function set

 enable();

 delay(DELAY);

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_6 | GPIO\_PIN\_7 , 0x00);  // function set

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_6 | GPIO\_PIN\_7 , 0x10);  // function set

 enable();

 delay(100);

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4 , 0x00);  // function set

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4 , 0x60);

 enable();

}

void

write\_S()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

}

void

write\_I()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x90);

 enable();

}

void

write\_Z()

**{**

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xA0);

 enable();

}

void

write\_E()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

}

void

write\_P()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 enable();

}

void

write\_R()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x20);

 enable();

}

void

write\_N()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xE0);

 enable();;

}

void

write\_T()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

}

void

write\_W()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x70);

 enable();

}

void

write\_M()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xD0);

 enable();

}

void

write\_O()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xF0);

 enable();

}

void

write\_U()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

}

void

write\_H()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x80);

 enable();

}

void

write\_0()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 enable();

}

void

write\_1()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x10);

 enable();

}

void

write\_2()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x20);

 enable();

}

void

write\_3()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

}

void

write\_4()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

}

void

write\_5()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x50);

 enable();

}

void

write\_6()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x60);

 enable();

}

eSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xvoid

write\_7()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x70);

 enable();

}

void

write\_8()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x80);

 enable();

}

void

write\_9()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x90);

 enable();

}

void shiftRight()  // one

{

 //RS RW  =0  0001  11xx

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x10);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 , 0x40);

 enable();

}

void newLine()

{

 int j =0;

 for(j = 0; j< 40 ; j++)

 {

   shiftRight();

   delay(DELAY);

 }

}

void clear()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x10);

 enable();

}

void returnHome()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x20);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void write\_A()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x40);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x10);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void screenLeft()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x10);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x80);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void cursorOff()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x00);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xC0);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void write\_dash()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x20);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xD0);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void write\_greater()

{

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

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0x30);

 enable();

 GPIOPinWrite(GPIO\_PORTC\_BASE, GPIO\_PIN\_7 | GPIO\_PIN\_6 | GPIO\_PIN\_5 | GPIO\_PIN\_4, 0xE0);

 enable();

 GPIOPinWrite(GPIO\_PORTD\_BASE, GPIO\_PIN\_5,0x00); //RS

}

void oledTask(void\* oled)

{

 OledDisplay\* oledPtr = (OledDisplay\*) oled;

 static int offset = 0;

 static int offsetFirstTime =1;

 xOLEDMessage xMessage;

 char str[6];

 while(1)

 {

   if(offsetFirstTime)

   {

     offset = globalCounter;

     offsetFirstTime = 0;

   }

   if(\*oledPtr->mode == 0)

   {

     if(scrollL%2 == 0 )

     {

       xMessage.pcMessage = "status <-       ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "annunc          ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "                ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueuOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     else

     {

       xMessage.pcMessage = "status          ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "annunc <-       ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "                ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

   }

   if( (\*oledPtr->mode == 1) && (\*oledPtr->annunciation == 0))

   {

     if(!(\*oledPtr->statusSelection))

     {

        switch (\*oledPtr->scroll % 4){

         case 0: xMessage.pcMessage = "north  <-";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "south    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "east     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "west     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 break;

         case 1: xMessage.pcMessage = "north    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "south  <-";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "east     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "west     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 break;

         case 2:  xMessage.pcMessage ="north    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "south    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "east   <-";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "west     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 break;

         case 3: xMessage.pcMessage = "north    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "south    ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "east     ";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage = "west   <-";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xMessage.pcMessage ="";

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 xQueueSend( xOLEDQueue, &xMessage, 0 );

                 break;

         default: break;

       }

     }

   }

   if(\*oledPtr->statusSelection )

   {

     // arriving train has higher priority on oled

     if(\*oledPtr->statusSelection == \*oledPtr->arrivalDirection )

     { //ID

       switch(\*oledPtr->arrivalDirection){

       case 1:  xMessage.pcMessage = "ID: north  train";

                xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

       case 2:  xMessage.pcMessage = "ID: south  train";

                xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

       case 3:  xMessage.pcMessage = "ID: east  train";

                xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

       case 4:  xMessage.pcMessage = "ID: west  train";

                xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

       default: xMessage.pcMessage = "";

                xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

       }

       //train arriving present

       if((\*oledPtr->trainArriving))//\*oledPtr->trainDepartingFlag

       {

         xMessage.pcMessage = "Train Arriving";

         xQueueSend( xOLEDQueue, &xMessage, 0 );

       }

       else

       {

         xMessage.pcMessage = "Train Present ";

         xQueueSend( xOLEDQueue, &xMessage, 0 );

       }

       //direction of arriving or departing train

       if(\*oledPtr->trainArriving)

       {

         switch(\*oledPtr->arrivalDirection){

         case 1:  xMessage.pcMessage = "arriving N";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 2:  xMessage.pcMessage = "arriving S";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 3:  xMessage.pcMessage = "arriving E";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 4:  xMessage.pcMessage = "arriving W";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         default: xMessage.pcMessage = "           ";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         }

       }

       else

       {

         switch(\*oledPtr->departureDirection){

         case 1:  xMessage.pcMessage = "Departing N";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 2:  xMessage.pcMessage = "Departing S";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 3:  xMessage.pcMessage = "Departing E";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         case 4:  xMessage.pcMessage = "Departing W";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         default: xMessage.pcMessage = "                ";

                  xQueueSend( xOLEDQueue, &xMessage, 0 ); break;

         }

       }

       xMessage.pcMessage ="" ;

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage="             ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage ="" ;

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

   }

   //flashing when a train is arriving

   if(\*oledPtr->trainArriving)

   {

     switch(\*oledPtr->arrivalDirection){

     case 1: if((arrivingCounter%30) == 15)

             {

               xMessage.pcMessage = "";

 xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xMessage.pcMessage = "North Train";

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xMessage.pcMessage = "";

               xQueueSend( xOLEDQueue, &xMessage, 0 );

             }

             else if((arrivingCounter%30) == 0)

             {

               xMessage.pcMessage = "";

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xMessage.pcMessage = "               ";

               xQueueSend( xOLEDQueue, &xMessage, 0 );

               xMessage.pcMessage = "";

               xQueueSend( xOLEDQueue, &xMessage, 0 );

             }

             break;

     case 2: if((arrivingCounter%30) == 15)

           {

xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "South Train";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           else if((arrivingCounter%30) == 0)

           {

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "               ";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           break;

     case 3: if((arrivingCounter%40) == 20)

           {

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "East Train";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           else if((arrivingCounter%40) == 0)

           {

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "               ";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           break;

     case 4: if((arrivingCounter%20) == 10)

           {

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "West Train";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           else if((arrivingCounter%20) == 0)

           {

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "              ";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

             xMessage.pcMessage = "";

             xQueueSend( xOLEDQueue, &xMessage, 0 );

           }

           break;

     default: break;

     }

   }

   else

   {

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "              ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

   }

if(\*oledPtr->annunciation )

   {

     if(\*oledPtr->trainPresent)  // one line

     {

       if(\*oledPtr->gridlock)

       {

         xMessage.pcMessage = "gridLock         ";

         xQueueSend( xOLEDQueue, &xMessage, 0 );

       }

       else

       {

         xMessage.pcMessage = "safe            ";

         xQueueSend( xOLEDQueue, &xMessage, 0 );

       }

     }

     else

     {

       xMessage.pcMessage = "not avaliable  ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     if(!annunciationFlag)  //second line

     {

       xMessage.pcMessage = "safe temperature";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     else

     {

       xMessage.pcMessage = "high temperature ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     //third line

     if(\*oledPtr->noiseIndex) //if there is data, thus index is non-zero, display the most recent one

     {

       static char fftFrequency[6];

       fftFrequency[0] = '0'+\*(oledPtr->processData+((\*oledPtr->noiseIndex)%16)-1)/1000  ;

       fftFrequency[1] = (\*(oledPtr->processData+((\*oledPtr->noiseIndex)%16)-1)/100)%10 + '0';

       fftFrequency[2] = (\*(oledPtr->processData+((\*oledPtr->noiseIndex)%16)-1)/10)%10 +'0';

       fftFrequency[3] = \*(oledPtr->processData+((\*oledPtr->noiseIndex)%16)-1)%10 +'0';

       fftFrequency[4] = '\0';

       xMessage.pcMessage = fftFrequency;

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     else

     {

       xMessage.pcMessage = "NA          ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

   }

   if((((\*oledPtr->statusSelection == \*oledPtr->arrivalDirection) && \*oledPtr->statusSelection !=0)|| \*oledPtr->annunciation ))  //fourth Line

   {

     unsigned short currentIndex = 0;

     if(index != 0)

     {

       if(index%8 == 0)

       {

         currentIndex = 7;

       }

       else

       {

         currentIndex = (index%8) - 1;

       }

     }

     if(\*(oledPtr->arrivingTrainDistance+currentIndex) > 100)

     {

       str[0] = \*(oledPtr->arrivingTrainDistance+currentIndex)/1000 + '0';

       str[1] = (\*(oledPtr->arrivingTrainDistance+currentIndex)/100)%10 + '0';

       str[2] = (\*(oledPtr->arrivingTrainDistance+currentIndex)/10)%10 +'0';

       str[3] = \*(oledPtr->arrivingTrainDistance+currentIndex)%10 +'0';

       str[4] = '\0';

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = str;

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

     else

     {

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "     ";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xMessage.pcMessage = "";

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

       xQueueSend( xOLEDQueue, &xMessage, 0 );

     }

   }

   if(\*oledPtr->phasor && !(\*oledPtr->photon))

   {

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = " ' ' ' ' ' ' ' ' ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = " ' ' 'phasor ' ' ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = " ' ' ' ' ' ' ' ' ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

   }

   if(\*oledPtr->photon && !(\*oledPtr->phasor ))

   {

     xMessage.pcMessage ="";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = " | | | | | | | | ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = " | | | |photon | ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

 xMessage.pcMessage = " | | | | | | | | ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

   }  //addhere

   if((\*oledPtr->phasor || \*oledPtr->photon ) && (globalCounter-delayCounterE2 == 7 ))

   {

     xMessage.pcMessage ="";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "                 ";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xMessage.pcMessage = "";

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 );

     xQueueSend( xOLEDQueue, &xMessage, 0 ); //end

     \*oledPtr->photon = FALSE;

     \*oledPtr->phasor = FALSE;

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

   }

   //reset

   if(FALSE == \*oledPtr->trainDepartingFlag && offset)

   {

     offset = 0;

     offsetFirstTime = 1;

   }

   vTaskDelay(100);

 }

}

void keypadTask(void\* dataPtr)

{

 LocalKeypadData\* keyPtr = (LocalKeypadData\*) dataPtr;

 while(1)

 {

   scrollSelection();

   modeSelect();

   annuFunc();

   trainStatus();

   if(GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_4) == 0x10 && debouncerC)

   {

     debouncerC = 0;

     modeL = 0;

     selectL = 0;

     annunciationL = 0;

     scrollL =0;

     statusSelectionL = 0;

   }

   if(!debouncerC && (GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_4) == 0x10))

   {

     debouncerC = 1;

   }

 \*keyPtr->annunciation = annunciationL;

   \*keyPtr->scroll = scrollL;

   \*keyPtr->select = selectL;

   \*keyPtr->mode = modeL;

   \*keyPtr->statusSelection = statusSelectionL;

   vTaskDelay(100);

 }

}

void modeSelect(void)

{

 if(modeL == 0)

 {

   if(1 == selectL)

   {

     if(scrollL % 2 == 0)

     {

       annunciationL = OFF;

     }

     else

     {

       annunciationL = ON;

     }

   }

 }

}

void trainStatus(void)

{

 if(modeL == 1 && annunciationL == OFF)

 {

   if(2 == selectL)

   {

     switch(scrollL % 4 + 1) {   // 1 is offse

     case 1: statusSelectionL = 1; //north

             break;

     case 2: statusSelectionL = 2; //south

             break;

     case 3: statusSelectionL = 3; //east

             break;

     case 4: statusSelectionL = 4; //west

 break;

     default:

             break;

     }

   }

 }

}

void annuFunc(void)

{

 if(modeL == OFF)

 {

   if(scrollL%2 == 1 && 1 == selectL )

   {

     annunciationL = ON;

     modeL = ON;

   }

   if(scrollL%2 == 0 && 1 == selectL )

   {

     annunciationL = OFF;

     modeL = ON;

   }

 }

}

void scrollSelection(void)

{

 if((GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_5) == 0x20) && debouncerA)

 {

   scrollL++;

   debouncerA  = 0;

 }

 if(!debouncerA && (GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_5) == 0x20))

 {

   debouncerA = 1;

 }

 //select

 if((GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_6) == 0x40) && debouncerB)

 {

   if(selectL == 0)

   {

    selectL = 1; //on

   }

   else if(selectL == 1)

   {

    selectL = 2; //on

   }

   debouncerB = 0;

 }

 if(!debouncerB && (GPIOPinRead(GPIO\_PORTB\_BASE, GPIO\_PIN\_6) == 0x40) )

 {

   debouncerB = 1;

 }

}

void temperatureMeasureTask(void\* tempData)

{

 TempData\* tempPtr =(TempData\*) tempData;

 static int index = 0;

 static int preValue = 1000;

 static int curValue = 0;

 while(1)

 {

   if(\*tempPtr->trainDepartingFlag)

   {

     if(t-r == 1)

     {

       ADCProcessorTrigger(ADC0\_BASE, 0);

       while(!ADCIntStatus(ADC0\_BASE, 0, false))

       {

       }

       ADCSequenceDataGet(ADC0\_BASE, 0, &ulValue0);

       ADCProcessorTrigger(ADC0\_BASE, 1);

       while(!ADCIntStatus(ADC0\_BASE, 1, false))

       {

       }

       ADCSequenceDataGet(ADC0\_BASE, 1, &ulValue1);

       ADCProcessorTrigger(ADC0\_BASE, 2);

       while(!ADCIntStatus(ADC0\_BASE, 2, false))

       {

       }

       ADCSequenceDataGet(ADC0\_BASE, 2, &ulValue2);

       unsigned long volt0 = 29 \* ulValue0;

       unsigned long volt1 = 29 \* ulValue1;

       unsigned long volt2 = 29 \* ulValue2;

       temp0 = volt0 \* 32 / 10000 + 33;

       temp1 = volt1 \* 32 / 10000 + 33;

       temp2 = volt2 \* 32 / 10000 + 33;

       \*(tempPtr->temperatureBuf + index%16) = temp0;

       index++;

       \*(tempPtr->temperatureBuf + index%16) = temp1;

       index++;

       \*(tempPtr->temperatureBuf + index%16) = temp2;

       index++;

       r = t;

       curValue = findMax(temp0, temp1, temp2);

       if(curValue\*10 >= 12\* preValue)  // curValue\*10 > 12\* preValue

       {

         annunciationFlag = TRUE;

       }

       preValue = curValue;

     }

   }

   else

   {

     index = 0;

     preValue = 1000;

     curValue = 0;

   }

   if(globalCounter - delayCounterE0 ==10 )

   {

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

   }

   vTaskDelay(100);

 }

}

unsigned long findMax(unsigned long temp0, unsigned long temp1, unsigned long temp2)

{

   unsigned long max = temp0;

   if (temp1 > temp0 && temp1 > temp2)

     max = temp1;

   else if(temp2 > temp1 && temp2 > temp0)

     max = temp2;

   return max;

}

void serialTask(void\* serial)

{

 SerialCom\* serialPtr = (SerialCom\*) serial;

 while(1)

 {

   UARTSend((unsigned char \*) "Train Present: ", 15);

   char traNum[3];

   traNum[0] = '0' + \*(serialPtr->trainArriving) / 10;

   traNum[1] = '0' + \*(serialPtr->trainArriving) % 10;

   traNum[2] = '\0';

   UARTSend((unsigned char \*) traNum, 2);

   UARTSend((unsigned char \*) "\n\r", 2);

   //Departure Direction

   UARTSend((unsigned char \*) "Depart Direction: ", 18);

   switch(\*serialPtr->departureDirection)

   {

   case 1:

     UARTSend((unsigned char \*) "North", 6);

     break;

   case 2:

     UARTSend((unsigned char \*) "South", 6);

     break;

   case 3:

     UARTSend((unsigned char \*) "East", 5);

     break;

   case 4:

     UARTSend((unsigned char \*) "West", 5);

     break;

   case 0:

     UARTSend((unsigned char \*) "NA", 3);

     break;

   }

   UARTSend((unsigned char \*) "\n\r", 3);

   //Train Size

   UARTSend((unsigned char \*) "Train Size: ", 13);

   char traSize[2];

   traSize[0] = '0' + \*(serialPtr->trainSize);

   traSize[1] = '\0';

   UARTSend((unsigned char \*) traSize, 1);

   UARTSend((unsigned char \*) "\n\r", 2);

   //Traversal Time

   UARTSend((unsigned char \*) "Traversal Time: ", 17);

   char traTime[3];

   traTime[0] = '0' + \*(serialPtr->traversalTime)/10;

   traTime[1] = '0' + \*(serialPtr->traversalTime)%10;

   traTime[2] = '\0';

   UARTSend((unsigned char \*) traTime, 2);

   UARTSend((unsigned char \*) "\n\r", 2);

   //Intersection Availability(Lock Status)

   UARTSend((unsigned char \*) "Intersection: ", 14);

   if( xSemaphoreTake(\*serialPtr->intersectionLock, ( portTickType ) 10))

   {

     UARTSend((unsigned char \*) "PASS", 5);

     xSemaphoreGive(\*serialPtr->intersectionLock);

   }

   else

   {

     UARTSend((unsigned char \*) "In Use", 7);

   }

   UARTSend((unsigned char \*) "\n\r", 2);

   //Number of arriving train

   UARTSend((unsigned char \*) "# of arriving: ", 15);

   char traTime2[3];

   traTime2[0] = '0' + \*(serialPtr->trainArriving)/10;

   traTime2[1] = '0' + \*(serialPtr->trainArriving)%10;

   traTime2[2] = '\0';

   UARTSend((unsigned char \*) traTime2, 2);

   UARTSend((unsigned char \*) "\n\r", 2);

  //Arrival direction

   UARTSend((unsigned char \*) "Arrival Directin: ", 17);

   switch(\*serialPtr->arrivalDirection)

   {

   case 1:

     UARTSend((unsigned char \*) "North", 5);

     break;

   case 2:

     UARTSend((unsigned char \*) "South", 5);

     break;

   case 3:

     UARTSend((unsigned char \*) "East", 4);

     break;

   case 4:

     UARTSend((unsigned char \*) "West", 4);

     break;

   case 0:

     UARTSend((unsigned char \*) "NA", 2);

     break;

   }

   UARTSend((unsigned char \*) "\n\r", 2);

   //Arrival Time

   UARTSend((unsigned char \*) "Arrival Time: ", 14);

   char arrTime[3];

   arrTime[0] = '0' + 6 \* nn / 10;

   arrTime[1] = '0' + 6 \* nn % 10;

   arrTime[2] = '\0';

   UARTSend((unsigned char \*) arrTime, 2);

   UARTSend((unsigned char \*) "\n\r", 2);

   //GridLock

   UARTSend((unsigned char \*) "GridLock: ", 10);

   switch(\*serialPtr->gridlock)

   {

   case 1:

     UARTSend((unsigned char \*) "ON", 3);

     break;

   case 0:

     UARTSend((unsigned char \*) "OFF", 4);

     break;

   }

   UARTSend((unsigned char \*) "\n\r", 2);

   //  temperature buf

   UARTSend((unsigned char \*) "temperature: ", 13);

   int i=0;

   for(i=0; i<16;i++)

   {

     char firstDigit[4];

     firstDigit[0]= '0'+(\*(serialPtr->temperatureBuf+i))/10;

     firstDigit[1]= '0'+(\*(serialPtr->temperatureBuf+i))%10;

     firstDigit[2]=' ';

     firstDigit[3]='\0';

     UARTSend((unsigned char \*) firstDigit , 4);

   }

   UARTSend((unsigned char \*) "\n\r", 2);

   UARTSend((unsigned char \*) "Arrival Time: ", 14);

   char arrTimea[3];

   if(\*serialPtr->trainArriving)

   {

     arrTimea[0] = '1';

     arrTimea[1] = '5';

     arrTimea[2] = '\0';

   }

   else

   {

     arrTimea[0] = '0';

     arrTimea[1] = '0';

     arrTimea[2] = '\0';

   }

   UARTSend((unsigned char \*) arrTimea, 3);

   UARTSend((unsigned char \*) "\n\r", 3);

   UARTSend((unsigned char \*) "distance: ", 10);

   i = 0;

   while( i<8 )  //stophere

   {

     char str2[7];

     str2[0] = '0' + (\*(serialPtr -> arrivingTrainDistance + i))/1000  ;

     str2[1] = '0' + (\*(serialPtr -> arrivingTrainDistance + i))/100%10;

     str2[2] = '0' + (\*(serialPtr -> arrivingTrainDistance + i))/10%10;

     str2[3] = '0' + (\*(serialPtr -> arrivingTrainDistance + i))%10;

     str2[4] = '\0';

     UARTSend((unsigned char \*) str2 , 5);

     UARTSend((unsigned char \*) " " , 1);

     i++;

   }

   UARTSend((unsigned char \*) "\n\r", 2);

   vTaskDelay(100);

 }

}

void

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

{

   while(ulCount--)

   {

     UARTCharPutNonBlocking(UART0\_BASE, \*pucBuffer++);

   }

   delay(30);

}

void noiseCaptureTask(void\* data)

{

 NoiseCaptureData\* capPtr = (NoiseCaptureData\*) data;

 while(1)

 {

   if(startCollection && globalCounter- delayCounterE3 == 10)

   {

     TimerControlTrigger(TIMER0\_BASE, TIMER\_A, true);

     TimerEnable(TIMER0\_BASE, TIMER\_A);

**}**

   vTaskDelay(100);

 }

}

signed int maxAmplitudeIndex;

void noiseProcessTask(void\* data)

{

 NoiseProcessData\* proPtr = (NoiseProcessData\*) data;

 int index =0;

 while(1)

 {

   if(\*proPtr->processFlag)

   {

     \*proPtr->processFlag = FALSE;

     int j = 0;

     for(j=0; j<256; j++)

     {

       imag[j] = 0;

       realCompute[j] = (real[j]/16 - 16);

     }

    maxAmplitudeIndex = optfft(realCompute, imag);

    index = (\*proPtr->noiseIndex)%16;

    \*(proPtr->processData+index) =  maxAmplitudeIndex \* FS / 256;

    (\*proPtr->noiseIndex)++;

   }

   vTaskDelay(100);

 }

}

void measureDistanceTask(void\* data)

{

 MeasureDistanceData\* distancePtr = (MeasureDistanceData\*) data;

 xOLEDMessage xMessage;

 while(1)

 {

   ADCProcessorTrigger(ADC0\_BASE, 1);

   while(!ADCIntStatus(ADC0\_BASE, 1, false))

   {

   }

   ADCSequenceDataGet(ADC0\_BASE, 1, &voltageADC);

   \*distancePtr->meter = (int)(voltageADC / -12.0 + 86);

   vTaskDelay(100);

 }

}

void attackTask(void\* data)

{

 AttackData\* attackPtr = (AttackData\*) data;

 static bool phasorPrepare = FALSE;

 static bool photonPrepare = FALSE;

 while(1)

 {

   if(\*attackPtr->meter > 5 && \*attackPtr->meter<= 30)

   {

     phasorPrepare = TRUE;

     photonPrepare = FALSE;

   }

   if(\*attackPtr->meter <= 5)

   {

     phasorPrepare = FALSE;

     photonPrepare = TRUE;

   }

   if(removeIt && globalCounter -delayCounterE2 > 5)

   {

     \*attackPtr->phasor = phasorPrepare;

     \*attackPtr->photon = photonPrepare;

     removeIt = FALSE;

     phasorPrepare= FALSE;

     photonPrepare= FALSE;

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

   }

   vTaskDelay(100);

 }

}

//ewarm

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

//

// startup\_ewarm.c - Boot code for Stellaris.

//

// Copyright (c) 2006-2007 Luminary Micro, Inc.  All rights reserved.

//

// Software License Agreement

//

// Luminary Micro, Inc. (LMI) is supplying this software for use solely and

// exclusively on LMI's microcontroller products.

//

// The software is owned by LMI and/or its suppliers, and is protected under

// applicable copyright laws.  All rights are reserved.  Any use in violation

// of the foregoing restrictions may subject the user to criminal sanctions

// under applicable laws, as well as to civil liability for the breach of the

// terms and conditions of this license.

//

// THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED

// OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

// LMI SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR

// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

//

// This is part of revision 100 of the Stellaris Ethernet

// Applications Library.

//

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

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

//

// Enable the IAR extensions for this source file.

//

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

#pragma language=extended

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

//

// Forward declaration of the default fault handlers.

//

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

static void NmiSR(void);

static void FaultISR(void);

static void IntDefaultHandler(void);

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

//

// External declaration for the interrupt handler used by the application.

//

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

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

//

// The entry point for the application.

//

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

extern void \_\_iar\_program\_start(void);

extern void xPortPendSVHandler(void);

extern void xPortSysTickHandler(void);

extern void vPortSVCHandler(void);

extern void vT2InterruptHandler( void );

extern void vT3InterruptHandler( void );

extern void vEMAC\_ISR( void );

//extern void Timer0IntHandler( void );

extern void GPIOAIntHandler(void);

extern void Timer1IntHandler(void);

extern void GPIOEIntHandler(void);

extern void GPIOFIntHandler(void);

extern void UARTIntHandler(void);

//extern void Timer0IntHandler(void);

extern void ADC3Handler(void);

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

//

// Reserve space for the system stack.

//

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

#ifndef STACK\_SIZE

#define STACK\_SIZE                              120

#endif

static unsigned long pulStack[STACK\_SIZE] @ ".noinit";

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

//

// A union that describes the entries of the vector table.  The union is needed

// since the first entry is the stack pointer and the remainder are function

// pointers.

//

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

typedef union

{

   void (\*pfnHandler)(void);

   unsigned long ulPtr;

}

uVectorEntry;

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

//

// The minimal vector table for a Cortex-M3.  Note that the proper constructs

// must be placed on this to ensure that it ends up at physical address

// 0x0000.0000.

//

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

\_\_root const uVectorEntry \_\_vector\_table[] @ ".intvec" =

{

   { .ulPtr = (unsigned long)pulStack + sizeof(pulStack) },

                                           // The initial stack pointer

   \_\_iar\_program\_start,                    // The reset handler

   NmiSR,                                  // The NMI handler

   FaultISR,                               // The hard fault handler

   IntDefaultHandler,                      // The MPU fault handler

   IntDefaultHandler,                      // The bus fault handler

   IntDefaultHandler,                      // The usage fault handler

   0,                                      // Reserved

   0,                                      // Reserved

   0,                                      // Reserved

   0,                                      // Reserved

   vPortSVCHandler,                        // SVCall handler

   IntDefaultHandler,                      // Debug monitor handler

   0,                                      // Reserved

   xPortPendSVHandler,                     // The PendSV handler

   xPortSysTickHandler,                    // The SysTick handler

   GPIOAIntHandler,                      // GPIO Port A

   IntDefaultHandler,                      // GPIO Port B

   IntDefaultHandler,                      // GPIO Port C

   IntDefaultHandler,                      // GPIO Port D

   GPIOEIntHandler,                      // GPIO Port E

   UARTIntHandler,                      // UART0 Rx and Tx

   IntDefaultHandler,                      // UART1 Rx and Tx

   IntDefaultHandler,                      // SSI Rx and Tx

   IntDefaultHandler,                      // I2C Master and Slave

   IntDefaultHandler,                      // PWM Fault

   IntDefaultHandler,                      // PWM Generator 0

   IntDefaultHandler,                      // PWM Generator 1

   IntDefaultHandler,                      // PWM Generator 2

   IntDefaultHandler,                      // Quadrature Encoder

   IntDefaultHandler,                      // ADC Sequence 0

   IntDefaultHandler,                      // ADC Sequence 1

   IntDefaultHandler,                      // ADC Sequence 2

   ADC3Handler,                      // ADC Sequence 3

   IntDefaultHandler,                      // Watchdog timer

   IntDefaultHandler,//Timer0IntHandler,                       // Timer 0 subtimer A

   IntDefaultHandler,                      // Timer 0 subtimer B

   Timer1IntHandler,                      // Timer 1 subtimer A

   IntDefaultHandler,                      // Timer 1 subtimer B

   vT2InterruptHandler,                      // Timer 2 subtimer A

   IntDefaultHandler,                      // Timer 2 subtimer B

   IntDefaultHandler,                      // Analog Comparator 0

   IntDefaultHandler,                      // Analog Comparator 1

   IntDefaultHandler,                      // Analog Comparator 2

   IntDefaultHandler,                      // System Control (PLL, OSC, BO)

   IntDefaultHandler,                      // FLASH Control

   GPIOFIntHandler,                      // GPIO Port F

   IntDefaultHandler,                      // GPIO Port G

   IntDefaultHandler,                      // GPIO Port H

   IntDefaultHandler,                      // UART2 Rx and Tx

   IntDefaultHandler,                      // SSI1 Rx and Tx

   vT3InterruptHandler,                    // Timer 3 subtimer A

   IntDefaultHandler,                      // Timer 3 subtimer B

   IntDefaultHandler,                      // I2C1 Master and Slave

   IntDefaultHandler,                      // Quadrature Encoder 1

   IntDefaultHandler,                      // CAN0

   IntDefaultHandler,                      // CAN1

   IntDefaultHandler,                      // CAN2

   vEMAC\_ISR,                              // Ethernet

   IntDefaultHandler,                      // Hibernate

   IntDefaultHandler,                      // USB0

   IntDefaultHandler,                      // PWM Generator 3

   IntDefaultHandler,                      // uDMA Software Transfer

   IntDefaultHandler                       // uDMA Error

};

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

//

// This is the code that gets called when the processor receives a NMI.  This

// simply enters an infinite loop, preserving the system state for examination

// by a debugger.

//

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

static void

NmiSR(void)

{

   //

   // Enter an infinite loop.

   //

   while(1)

   {

   }

}

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

//

// This is the code that gets called when the processor receives a fault

// interrupt.  This simply enters an infinite loop, preserving the system state

// for examination by a debugger.

//

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

static void

FaultISR(void)

{

   //

   // Enter an infinite loop.

   //

   while(1)

   {

   }

}

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

//

// This is the code that gets called when the processor receives an unexpected

// interrupt.  This simply enters an infinite loop, preserving the system state

// for examination by a debugger.

//

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

static void

IntDefaultHandler(void)

{

   //

   // Go into an infinite loop.

   //

   while(1)

   {

   }

}

//httpd-cgi

/\*\*

\* \addtogroup httpd

\* @{

\*/

/\*\*

\* \file

\*         Web server script interface

\* \author

\*         Adam Dunkels <adam@sics.se>

\*

\*/

/\*

\* Copyright (c) 2001-2006, Adam Dunkels.

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\* 1. Redistributions of source code must retain the above copyright

\*    notice, this list of conditions and the following disclaimer.

\* 2. Redistributions in binary form must reproduce the above copyright

\*    notice, this list of conditions and the following disclaimer in the

\*    documentation and/or other materials provided with the distribution.

\* 3. The name of the author may not be used to endorse or promote

\*    products derived from this software without specific prior

\*    written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS

\* OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY

\* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE

\* GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

\* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

\* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

\* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*

\* This file is part of the uIP TCP/IP stack.

\*

\* $Id: httpd-cgi.c,v 1.2 2006/06/11 21:46:37 adam Exp $

\*

\*/

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "semphr.h"

#include "uip.h"

#include "psock.h"

#include "httpd.h"

#include "httpd-cgi.h"

#include "httpd-fs.h"

#include <stdio.h>

#include <string.h>

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

typedef enum myBool bool;

struct Command

{

 unsigned short\* trainPresent;

 unsigned short\* trainArriving;

 unsigned short\* arrivingTrainDistance;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 xSemaphoreHandle\* intersectionLock;

 unsigned long\* temperatureBuf;

 bool\* gridlock;

 char\* response;

 signed int\* noiseIndex;

 signed int\* processData;

 unsigned long\* temp0;

 unsigned long\* temp1;

 unsigned long\* temp2;

};

typedef struct Command CommandData;

extern CommandData commandData;

HTTPD\_CGI\_CALL(file, "file-stats", file\_stats);

HTTPD\_CGI\_CALL(tcp, "tcp-connections", tcp\_stats);

HTTPD\_CGI\_CALL(net, "net-stats", net\_stats);

HTTPD\_CGI\_CALL(rtos, "rtos-stats", rtos\_stats );

HTTPD\_CGI\_CALL(io, "led-io", led\_io );

static const struct httpd\_cgi\_call \*calls[] = { &file, &tcp, &net, &rtos, &io, NULL };

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

static

PT\_THREAD(nullfunction(struct httpd\_state \*s, char \*ptr))

{

 PSOCK\_BEGIN(&s->sout);

 PSOCK\_END(&s->sout);

}

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

httpd\_cgifunction

httpd\_cgi(char \*name)

{

 const struct httpd\_cgi\_call \*\*f;

 /\* Find the matching name in the table, return the function. \*/

 for(f = calls; \*f != NULL; ++f) {

   if(strncmp((\*f)->name, name, strlen((\*f)->name)) == 0) {

     return (\*f)->function;

   }

 }

 return nullfunction;

}

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

static unsigned short

generate\_file\_stats(void \*arg)

{

 char \*f = (char \*)arg;

 return snprintf((char \*)uip\_appdata, UIP\_APPDATA\_SIZE, "%5u", httpd\_fs\_count(f));

}

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

static

PT\_THREAD(file\_stats(struct httpd\_state \*s, char \*ptr))

{

 PSOCK\_BEGIN(&s->sout);

 PSOCK\_GENERATOR\_SEND(&s->sout, generate\_file\_stats, strchr(ptr, ' ') + 1);

 PSOCK\_END(&s->sout);

}

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

static const char closed[] =   /\*  "CLOSED",\*/

{0x43, 0x4c, 0x4f, 0x53, 0x45, 0x44, 0};

static const char syn\_rcvd[] = /\*  "SYN-RCVD",\*/

{0x53, 0x59, 0x4e, 0x2d, 0x52, 0x43, 0x56,

0x44,  0};

static const char syn\_sent[] = /\*  "SYN-SENT",\*/

{0x53, 0x59, 0x4e, 0x2d, 0x53, 0x45, 0x4e,

0x54,  0};

static const char established[] = /\*  "ESTABLISHED",\*/

{0x45, 0x53, 0x54, 0x41, 0x42, 0x4c, 0x49, 0x53, 0x48,

0x45, 0x44, 0};

static const char fin\_wait\_1[] = /\*  "FIN-WAIT-1",\*/

{0x46, 0x49, 0x4e, 0x2d, 0x57, 0x41, 0x49,

0x54, 0x2d, 0x31, 0};

static const char fin\_wait\_2[] = /\*  "FIN-WAIT-2",\*/

{0x46, 0x49, 0x4e, 0x2d, 0x57, 0x41, 0x49,

0x54, 0x2d, 0x32, 0};

static const char closing[] = /\*  "CLOSING",\*/

{0x43, 0x4c, 0x4f, 0x53, 0x49,

0x4e, 0x47, 0};

static const char time\_wait[] = /\*  "TIME-WAIT,"\*/

{0x54, 0x49, 0x4d, 0x45, 0x2d, 0x57, 0x41,

0x49, 0x54, 0};

static const char last\_ack[] = /\*  "LAST-ACK"\*/

{0x4c, 0x41, 0x53, 0x54, 0x2d, 0x41, 0x43,

0x4b, 0};

static const char \*states[] = {

 closed,

 syn\_rcvd,

 syn\_sent,

 established,

 fin\_wait\_1,

 fin\_wait\_2,

 closing,

 time\_wait,

 last\_ack};

static unsigned short

generate\_tcp\_stats(void \*arg)

{

 struct uip\_conn \*conn;

 struct httpd\_state \*s = (struct httpd\_state \*)arg;

 conn = &uip\_conns[s->count];

 return snprintf((char \*)uip\_appdata, UIP\_APPDATA\_SIZE,

        "<tr><td>%d</td><td>%u.%u.%u.%u:%u</td><td>%s</td><td>%u</td><td>%u</td><td>%c %c</td></tr>\r\n",

        htons(conn->lport),

        htons(conn->ripaddr[0]) >> 8,

        htons(conn->ripaddr[0]) & 0xff,

        htons(conn->ripaddr[1]) >> 8,

        htons(conn->ripaddr[1]) & 0xff,

        htons(conn->rport),

        states[conn->tcpstateflags & UIP\_TS\_MASK],

        conn->nrtx,

        conn->timer,

        (uip\_outstanding(conn))? '\*':' ',

        (uip\_stopped(conn))? '!':' ');

}

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

static

PT\_THREAD(tcp\_stats(struct httpd\_state \*s, char \*ptr))

{

 PSOCK\_BEGIN(&s->sout);

 for(s->count = 0; s->count < UIP\_CONNS; ++s->count) {

   if((uip\_conns[s->count].tcpstateflags & UIP\_TS\_MASK) != UIP\_CLOSED) {

     PSOCK\_GENERATOR\_SEND(&s->sout, generate\_tcp\_stats, s);

   }

 }

 PSOCK\_END(&s->sout);

}

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

static unsigned short

generate\_net\_stats(void \*arg)

{

 struct httpd\_state \*s = (struct httpd\_state \*)arg;

 return snprintf((char \*)uip\_appdata, UIP\_APPDATA\_SIZE,

         "%5u\n", ((uip\_stats\_t \*)&uip\_stat)[s->count]);

}

static

PT\_THREAD(net\_stats(struct httpd\_state \*s, char \*ptr))

{

 PSOCK\_BEGIN(&s->sout);

#if UIP\_STATISTICS

 for(s->count = 0; s->count < sizeof(uip\_stat) / sizeof(uip\_stats\_t);

     ++s->count) {

   PSOCK\_GENERATOR\_SEND(&s->sout, generate\_net\_stats, s);

 }

#endif /\* UIP\_STATISTICS \*/

 PSOCK\_END(&s->sout);

}

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

extern void vTaskList( signed char \*pcWriteBuffer );

static char cCountBuf[ 32 ];

long lRefreshCount = 0;

static unsigned short

generate\_rtos\_stats(void \*arg)

{

   lRefreshCount++;

   sprintf( cCountBuf, "<p><br>Refresh count = %d", lRefreshCount );

       vTaskList( uip\_appdata );

   strcat( uip\_appdata, cCountBuf );

   return strlen( uip\_appdata );

}

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

static

PT\_THREAD(rtos\_stats(struct httpd\_state \*s, char \*ptr))

{

 PSOCK\_BEGIN(&s->sout);

 PSOCK\_GENERATOR\_SEND(&s->sout, generate\_rtos\_stats, NULL);

 PSOCK\_END(&s->sout);

}

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

/\*

char \*pcStatus;

extern unsigned long uxParTestGetLED( unsigned long uxLED );

static unsigned short generate\_io\_state( void \*arg )

{

   if( uxParTestGetLED( 0 ) )

   {

         pcStatus = "checked";

       }

   else

   {

       pcStatus = "";

   }

   sprintf( uip\_appdata,

       "<input type=\"checkbox\" name=\"LED0\" value=\"1\" %s>LED"\

       "<p>"\

       "<input type=\"text\" name=\"LCD\" value=\"Enter LCD text\" size=\"16\">",

       pcStatus );

   return strlen( uip\_appdata );

}

\*/

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

//char \*pcStatus;  //0

//extern unsigned long uxParTestGetLED( unsigned long uxLED );

char \*ifGridlock;

char \*resp;

char \*arrdirec;

char \*departdirec;

signed int recentNoise = 0;

char\* intersection;

char temperatureAll[50];

static unsigned short generate\_io\_state(void \*arg)

{

       CommandData\* aPtr = (CommandData\*) arg;

       if (\*aPtr->gridlock)

       {

         ifGridlock = "TRUE";

       }

       else

       {

          ifGridlock = "FALSE";

       }

       switch(\*aPtr->arrivalDirection)

       {

         case 1:

           arrdirec = "North";

           break;

         case 2:

           arrdirec = "South";

           break;

         case 3:

           arrdirec = "East";

         case 4:

           arrdirec = "West";

           break;

         default:

           arrdirec = "";

           break;

       }

       switch(\*aPtr->departureDirection)

       {

         case 1:

           departdirec = "North";

           break;

         case 2:

           departdirec = "South";

           break;

         case 3:

           departdirec = "East";

         case 4:

           departdirec = "West";

           break;

         default:

           departdirec = "";

           break;

       }

       if(xSemaphoreTake(\*aPtr->intersectionLock, ( portTickType ) 10)==pdTRUE)

       {

         intersection = " Pass";

         xSemaphoreGive(\*aPtr->intersectionLock);

       }

       else

       {

         intersection = " Not available";

       }

       //values from temperature buffer

       signed int a0, a1, a2,a3, a4, a5, a6,a7,a8,a9,a10, a11, a12, a13, a14, a15;

       a0 = \*(aPtr->processData + 0);

       a1 = \*(aPtr->processData + 1);

       a2 = \*(aPtr->processData + 2);

       a3 = \*(aPtr->processData + 3);

       a4 = \*(aPtr->processData + 4);

       a5 = \*(aPtr->processData + 5);

       a6 = \*(aPtr->processData + 6);

       a7 = \*(aPtr->processData + 7);

       a8 = \*(aPtr->processData + 8);

       a9 = \*(aPtr->processData + 9);

       a10 = \*(aPtr->processData + 10);

       a11 = \*(aPtr->processData + 11);

       a12 = \*(aPtr->processData + 12);

       a13 = \*(aPtr->processData + 13);

       a14 = \*(aPtr->processData + 14);

       a15= \*(aPtr->processData + 15);

       resp = aPtr->response;

       int i =0;

       int j= 0;

       for(i =0; i<48; i++)

       {

         temperatureAll[i] = \*(aPtr->temperatureBuf+j)/10+'0';

         i++;

         temperatureAll[i] = \*(aPtr->temperatureBuf+j)%10+'0';

         i++;

         temperatureAll[i] = ' ';

         j++;

       }

       temperatureAll[i]= '\0';

       int mostRecentNoise = 0;

       int index = \*aPtr->noiseIndex;

       if(index != 0)

       {

         mostRecentNoise = \*(aPtr->processData + index - 1);

       }

       else

       {

         mostRecentNoise = \*(aPtr->processData + index + 15);

       }

       sprintf( uip\_appdata,

               "Train Present: %d"\

               "<p>"\

       "Departing Direction: %s"\

       "<p>"\

               "Train Size: %d"\

               "<p>"\

               "Traversal Time: %d"\

               "<p>"\

               "Intersection Availability:%s"\

               "<p>"

                "Arrival Direction:%s"\

               "<p>"\

                "Arrival Distance:%d"\

                "<p>"\

                "Gridlock: %s"\

                "<p>"\

                 "Noise Signature: %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d "\

                "<p>"\

                "Wheel Temperature: %s"\

                "<p>"\

                 "<input type=\"text\" name=\"LCD\" value=\"\" size=\"16\">"\

                 "<p>"\

                "Response: %s"\

                 "<p>"\

                 "Temp:%d, %d, %d"\

                 "<p>"\

                 "Recent NoiseSig:%d",

       \*aPtr->trainPresent,departdirec, \*aPtr->trainSize, \*aPtr->traversalTime,

               intersection, arrdirec,

                \*aPtr->arrivingTrainDistance,ifGridlock, a0, a1, a2, a3, a4, a5, a6,a7,a8,a9,a10, a11, a12, a13, a14, a15,

                temperatureAll,resp, \*aPtr->temp0, \*aPtr->temp1, \*aPtr->temp2, mostRecentNoise);

       return strlen( uip\_appdata );

}

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

static PT\_THREAD(led\_io(struct httpd\_state \*s, char \*ptr))

{

 //RemoteData\* aPtr = (RemoteData\*) arg;

 PSOCK\_BEGIN(&s->sout);

 PSOCK\_GENERATOR\_SEND(&s->sout, generate\_io\_state, &commandData);

 PSOCK\_END(&s->sout);

}

/\*\* @} \*/

/\*

\* Copyright (c) 2001, Swedish Institute of Computer Science.

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\* 1. Redistributions of source code must retain the above copyright

\*    notice, this list of conditions and the following disclaimer.

\* 2. Redistributions in binary form must reproduce the above copyright

\*    notice, this list of conditions and the following disclaimer in the

\*    documentation and/or other materials provided with the distribution.

\* 3. Neither the name of the Institute nor the names of its contributors

\*    may be used to endorse or promote products derived from this software

\*    without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

\* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED.  IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE

\* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS

\* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

\* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

\* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

\* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

\* SUCH DAMAGE.

\*

\* This file is part of the lwIP TCP/IP stack.

\*

\* Author: Adam Dunkels <adam@sics.se>

\*

\* $Id: httpd-fs.c,v 1.1 2006/06/07 09:13:08 adam Exp $

\*/

#include "httpd.h"

#include "httpd-fs.h"

#include "httpd-fsdata.h"

#ifndef NULL

#define NULL 0

#endif /\* NULL \*/

#include "httpd-fsdata.c"

#if HTTPD\_FS\_STATISTICS

static u16\_t count[HTTPD\_FS\_NUMFILES];

#endif /\* HTTPD\_FS\_STATISTICS \*/

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

static u8\_t

httpd\_fs\_strcmp(const char \*str1, const char \*str2)

{

 u8\_t i;

 i = 0;

loop:

 if(str2[i] == 0 ||

    str1[i] == '\r' ||

    str1[i] == '\n') {

   return 0;

 }

 if(str1[i] != str2[i]) {

   return 1;

 }

 ++i;

 goto loop;

}

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

int

httpd\_fs\_open(const char \*name, struct httpd\_fs\_file \*file)

{

#if HTTPD\_FS\_STATISTICS

 u16\_t i = 0;

#endif /\* HTTPD\_FS\_STATISTICS \*/

 struct httpd\_fsdata\_file\_noconst \*f;

 for(f = (struct httpd\_fsdata\_file\_noconst \*)HTTPD\_FS\_ROOT;

     f != NULL;

     f = (struct httpd\_fsdata\_file\_noconst \*)f->next) {

   if(httpd\_fs\_strcmp(name, f->name) == 0) {

     file->data = f->data;

     file->len = f->len;

#if HTTPD\_FS\_STATISTICS

     ++count[i];

#endif /\* HTTPD\_FS\_STATISTICS \*/

     return 1;

   }

#if HTTPD\_FS\_STATISTICS

   ++i;

#endif /\* HTTPD\_FS\_STATISTICS \*/

 }

 return 0;

}

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

void

httpd\_fs\_init(void)

{

#if HTTPD\_FS\_STATISTICS

 u16\_t i;

 for(i = 0; i < HTTPD\_FS\_NUMFILES; i++) {

   count[i] = 0;

 }

#endif /\* HTTPD\_FS\_STATISTICS \*/

}

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

#if HTTPD\_FS\_STATISTICS

u16\_t httpd\_fs\_count

(char \*name)

{

 struct httpd\_fsdata\_file\_noconst \*f;

 u16\_t i;

 i = 0;

 for(f = (struct httpd\_fsdata\_file\_noconst \*)HTTPD\_FS\_ROOT;

     f != NULL;

     f = (struct httpd\_fsdata\_file\_noconst \*)f->next) {

   if(httpd\_fs\_strcmp(name, f->name) == 0) {

     return count[i];

   }

   ++i;

 }

 return 0;

}

#endif /\* HTTPD\_FS\_STATISTICS \*/

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

//uip

/\*

This

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.

   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.

\*/

/\* Standard includes. \*/

#include <string.h>

/\* Scheduler includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "semphr.h"

/\* uip includes. \*/

#include "hw\_types.h"

#include "uip.h"

#include "uip\_arp.h"

#include "httpd.h"

#include "timer.h"

#include "clock-arch.h"

#include "hw\_ethernet.h"

#include "ethernet.h"

#include "hw\_memmap.h"

#include "lmi\_flash.h"

#include "sysctl.h"

/\* Demo includes. \*/

#include "emac.h"

#include "partest.h"

#include "lcd\_message.h"

#include <string.h>

/\* Scheduler includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "semphr.h"

/\* uip includes. \*/

#include "hw\_types.h"

#include "uip.h"

#include "uip\_arp.h"

#include "httpd.h"

#include "timer.h"

#include "clock-arch.h"

#include "hw\_ethernet.h"

#include "ethernet.h"

#include "hw\_memmap.h"

#include "lmi\_flash.h"

#include "sysctl.h"

/\* Demo includes. \*/

#include "emac.h"

#include "partest.h"

#include "lcd\_message.h"

#include "inc/hw\_gpio.h"

#include "inc/hw\_ints.h"

#include "inc/hw\_memmap.h"

#include "driverlib/gpio.h"

#include "driverlib/interrupt.h"

#include "driverlib/timer.h"

#include "driverlib/pwm.h"

#include "driverlib/uart.h"

#include "driverlib/adc.h"

#include <stdlib.h>

#include <limits.h>

#include <optfft.h>

#include "semphr.h"

/\* Hardware library includes. \*/

//#include "hw\_memmap.h"

struct timer {

 clock\_time\_t start;

 clock\_time\_t interval;

};

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

typedef enum myBool bool;

struct Command

{

 unsigned short\* trainPresent;

 unsigned short\* trainArriving;

 unsigned short\* arrivingTrainDistance;

 unsigned short\* trainSize;

 unsigned short\* traversalTime;

 unsigned short\* arrivalDirection;

 unsigned short\* departureDirection;

 unsigned short\* intersectionLock;

 unsigned long\* temperatureBuf;

 bool\* gridlock;

 char\* response;

 signed int\* noiseIndex;

 signed int\* processData;

 unsigned long\* temp0;

 unsigned long\* temp1;

 unsigned long\* temp2;

 signed int recentNoise;

};

typedef struct Command CommandData;

extern CommandData commandData;

extern void RIT128x96x4DisplayOn(void);

extern void RIT128x96x4DisplayOff(void);

extern unsigned long temp0, temp1, temp2;

bool checkOLED = TRUE;

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

/\* IP address configuration. \*/

/\*

#define uipIP\_ADDR0     172

#define uipIP\_ADDR1     25

#define uipIP\_ADDR2     218

#define uipIP\_ADDR3     19

\*/

#define uipIP\_ADDR0     128

#define uipIP\_ADDR1     95

#define uipIP\_ADDR2     141

#define uipIP\_ADDR3     220

/\* How long to wait before attempting to connect the MAC again. \*/

#define uipINIT\_WAIT    100

/\* Shortcut to the header within the Rx buffer. \*/

#define xHeader ((struct uip\_eth\_hdr \*) &uip\_buf[ 0 ])

/\* Standard constant. \*/

#define uipTOTAL\_FRAME\_HEADER\_SIZE  54

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

/\*

\* Send the uIP buffer to the MAC.

\*/

static void prvENET\_Send(void);

/\*

\* Setup the MAC address in the MAC itself, and in the uIP stack.

\*/

static void prvSetMACAddress( void );

/\*

\* Port functions required by the uIP stack.

\*/

void clock\_init( void );

clock\_time\_t clock\_time( void );

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

/\* The semaphore used by the ISR to wake the uIP task. \*/

extern xSemaphoreHandle xEMACSemaphore;

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

void clock\_init(void)

{

   /\* This is done when the scheduler starts. \*/

}

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

/\* Define clock functions here to avoid header file name clash between uIP

and the Luminary Micro driver library. \*/

clock\_time\_t clock\_time( void )

{

   return xTaskGetTickCount();

}

extern void timer\_set(struct timer \*t, clock\_time\_t interval);

extern int timer\_expired(struct timer \*t);

extern void timer\_reset(struct timer \*t);

void vuIP\_Task( void \*pvParameters )

{

portBASE\_TYPE i;

uip\_ipaddr\_t xIPAddr;

struct timer periodic\_timer, arp\_timer;

extern void ( vEMAC\_ISR )( void );

   /\* Enable/Reset the Ethernet Controller \*/

   SysCtlPeripheralEnable( SYSCTL\_PERIPH\_ETH );

   SysCtlPeripheralReset( SYSCTL\_PERIPH\_ETH );

   /\* Create the semaphore used by the ISR to wake this task. \*/

   vSemaphoreCreateBinary( xEMACSemaphore );

   /\* Initialise the uIP stack. \*/

   timer\_set( &periodic\_timer, configTICK\_RATE\_HZ / 2 );

   timer\_set( &arp\_timer, configTICK\_RATE\_HZ \* 10 );

   uip\_init();

   uip\_ipaddr( xIPAddr, uipIP\_ADDR0, uipIP\_ADDR1, uipIP\_ADDR2, uipIP\_ADDR3 );

   uip\_sethostaddr( xIPAddr );

   httpd\_init();

   while( vInitEMAC() != pdPASS )

   {

       vTaskDelay( uipINIT\_WAIT );

   }

   prvSetMACAddress();

   for( ;; )

   {

       /\* Is there received data ready to be processed? \*/

       uip\_len = uiGetEMACRxData( uip\_buf );

       if( uip\_len > 0 )

       {

           /\* Standard uIP loop taken from the uIP manual. \*/

           if( xHeader->type == htons( UIP\_ETHTYPE\_IP ) )

           {

               uip\_arp\_ipin();

               uip\_input();

               /\* If the above function invocation resulted in data that

               should be sent out on the network, the global variable

               uip\_len is set to a value > 0. \*/

               if( uip\_len > 0 )

               {

                   uip\_arp\_out();

                   prvENET\_Send();

               }

           }

           else if( xHeader->type == htons( UIP\_ETHTYPE\_ARP ) )

           {

               uip\_arp\_arpin();

               /\* If the above function invocation resulted in data that

               should be sent out on the network, the global variable

               uip\_len is set to a value > 0. \*/

               if( uip\_len > 0 )

               {

                   prvENET\_Send();

               }

           }

       }

       else

       {

           if( timer\_expired( &periodic\_timer ) )

           {

               timer\_reset( &periodic\_timer );

               for( i = 0; i < UIP\_CONNS; i++ )

               {

                   uip\_periodic( i );

                   /\* If the above function invocation resulted in data that

                   should be sent out on the network, the global variable

                   uip\_len is set to a value > 0. \*/

                   if( uip\_len > 0 )

                   {

                       uip\_arp\_out();

                       prvENET\_Send();

                   }

               }

               /\* Call the ARP timer function every 10 seconds. \*/

               if( timer\_expired( &arp\_timer ) )

               {

                   timer\_reset( &arp\_timer );

                   uip\_arp\_timer();

               }

           }

           else

           {

               /\* We did not receive a packet, and there was no periodic

               processing to perform.  Block for a fixed period.  If a packet

               is received during this period we will be woken by the ISR

               giving us the Semaphore. \*/

               xSemaphoreTake( xEMACSemaphore, configTICK\_RATE\_HZ / 2 );

           }

       }

   }

}

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

static void prvENET\_Send(void)

{

   vInitialiseSend();

   vIncrementTxLength( uip\_len );

   vSendBufferToMAC();

}

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

static void prvSetMACAddress( void )

{

unsigned portLONG ulUser0, ulUser1;

unsigned char pucMACArray[8];

struct uip\_eth\_addr xAddr;

   /\* Get the device MAC address from flash \*/

   FlashUserGet(&ulUser0, &ulUser1);

   /\* Convert the MAC address from flash into sequence of bytes. \*/

   pucMACArray[0] = ((ulUser0 >>  0) & 0xff);

   pucMACArray[1] = ((ulUser0 >>  8) & 0xff);

   pucMACArray[2] = ((ulUser0 >> 16) & 0xff);

   pucMACArray[3] = ((ulUser1 >>  0) & 0xff);

   pucMACArray[4] = ((ulUser1 >>  8) & 0xff);

   pucMACArray[5] = ((ulUser1 >> 16) & 0xff);

   /\* Program the MAC address. \*/

   EthernetMACAddrSet(ETH\_BASE, pucMACArray);

   xAddr.addr[ 0 ] = pucMACArray[0];

   xAddr.addr[ 1 ] = pucMACArray[1];

   xAddr.addr[ 2 ] = pucMACArray[2];

   xAddr.addr[ 3 ] = pucMACArray[3];

   xAddr.addr[ 4 ] = pucMACArray[4];

   xAddr.addr[ 5 ] = pucMACArray[5];

   uip\_setethaddr( xAddr );

}

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

void vApplicationProcessFormInput( portCHAR \*pcInputString, portBASE\_TYPE xInputLength, void \*arg )

{

CommandData\* aPtr = (CommandData\*) arg;

char \*c, \*pcText;

static portCHAR cMessageForDisplay[ 32 ];

extern xQueueHandle xOLEDQueue;

xOLEDMessage xOLEDMessage;

signed int noise = 0;

/\* Process the form input sent by the IO page of the served HTML. \*/

   c = strstr( pcInputString, "?" );

   if( c )

   {

       /\* Turn LED's on or off in accordance with the check box status. \*/

     /\*

       if( strstr( c, "LED0=1" ) != NULL )

       {

           vParTestSetLED( 0, 1 );

       }

       else

       {

           vParTestSetLED( 0, 0 );

       }

       \*/

       /\* Find the start of the text to be displayed on the LCD. \*/

       pcText = strstr( c, "LCD=" );

       pcText += strlen( "LCD=" );

       /\* Terminate the file name for further processing within uIP. \*/

       \*c = 0x00;

       /\* Terminate the LCD string. \*/

       c = strstr( pcText, " " );

       if( c != NULL )

       {

           \*c = 0x00;

       }

       /\* Add required spaces. \*/

       while( ( c = strstr( pcText, "+" ) ) != NULL )

       {

           \*c = ' ';

       }

       /\* Write the message to the LCD. \*/

   strcpy( cMessageForDisplay, pcText );

       //Handles the data display and storage

       if (!(strcmp(cMessageForDisplay, "D")))

       {

         //acknowledges D

         strcpy(aPtr->response, "AD");

         if (checkOLED)

         {

          RIT128x96x4DisplayOff();

          checkOLED = FALSE;

         }

         else

         {

           RIT128x96x4DisplayOn();

           checkOLED = TRUE;

         }

       }

        else if (!(strcmp(cMessageForDisplay,"MT")))

       {

               strcpy(aPtr->response, "MT");

               \*aPtr->temp0 = temp0;

               \*aPtr->temp1 = temp1;

               \*aPtr->temp2 = temp2;

       }

       else if(!(strcmp(cMessageForDisplay,"MN")))

       {

           //display recent noise

           //acknowledges noise M

           strcpy(aPtr->response, "MN");

           signed int index = \*aPtr->noiseIndex;

           if(index == 0)

           {

              noise = \*(aPtr->processData + index + 15 );

           }

           else

           {

              noise= \*(aPtr->processData + index - 1);

           }

           aPtr->recentNoise = noise;

        }

       else if (!(strcmp(cMessageForDisplay,"S")))

       {

           //enable all interrupts of measurement tasks

           //acknowledges START

           strcpy(aPtr->response, "AS");

           //Temperature and noise detection

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

           GPIOPinIntEnable(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

           GPIOPinIntEnable(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

           IntEnable(INT\_GPIOE);

           //vTaskSuspend(temperatureMeasurementTask);

           //vTaskSuspend(noiseCaptureTask);

           // frequency detection

           GPIOPinIntEnable(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);

           IntEnable(INT\_GPIOA);

       }

       else if (!(strcmp(cMessageForDisplay, "P")))

       {

           //disable all interrupts of measurement tasks

           //acknowledges STOP

           strcpy(aPtr->response, "AP");

           IntDisable(INT\_GPIOE);

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

           GPIOPinIntDisable(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

           GPIOPinIntDisable(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);

           // frequency detection

           IntDisable(INT\_GPIOA);

           GPIOPinIntDisable(GPIO\_PORTA\_BASE, GPIO\_PIN\_4);

       }

       else

       {//responde with error message

         strcpy(aPtr->response,"E");

       }

   }

}