Ain Shams University

Faculty of Engineering – CHEP
CESS Program

Spring 2021

CSE345 Embedded Systems

Project

Time in different cities

Submitted by

Ahmed Abd El-Nasser Korany (17P8113)

Ahmed Essam Mohamed Ramzy (17P8229)

Abdelrahman Mahmoud Fangary (17P6006)

Moaz Mohamed Abd Elaziz Farrag (17P1023)

Mohamed Ahmed Hashem Mokbel (17P3067)

Table Of Contents

1. Introduction And Overview

This project is about developing and configuring a simple “Time in different cities” application. It's multitasking clock project where the current time in one of 10 cities in different countries can be selected from the PC keyboard. The time in the selected city is then displayed on the LCD together with the name of the city. The user selects the city where the time is to be displayed from a menu displayed on the PC screen.

2. Project Description

- When the program is started, the user by default is prompted to enter the time in London and this is displayed by default on the LCD and updated automatically every second. A menu is displayed on the PC that enables the user to select the city where the current time is to be displayed. The city name is displayed on the first row of the LCD, while the current time is displayed on the second row of the LCD.

- The **Aim** of the project is to show how multitasking can be used in a practical and useful clock project using FreeRTOS APIs.

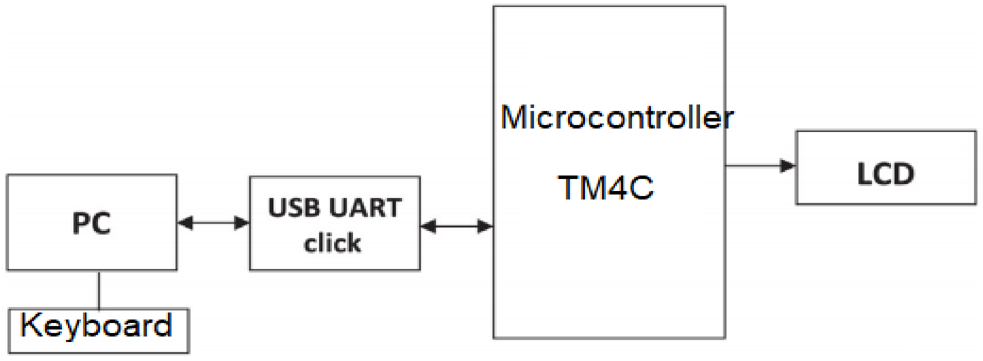
3. General Overview of Project Components and Listings

- The program consists of three tasks in addition to the Idle task and other functions that will be explained in section 5.
- The city names and their time differences from London are stored in two-dimensional character arrays called cities and timediff, respectively.

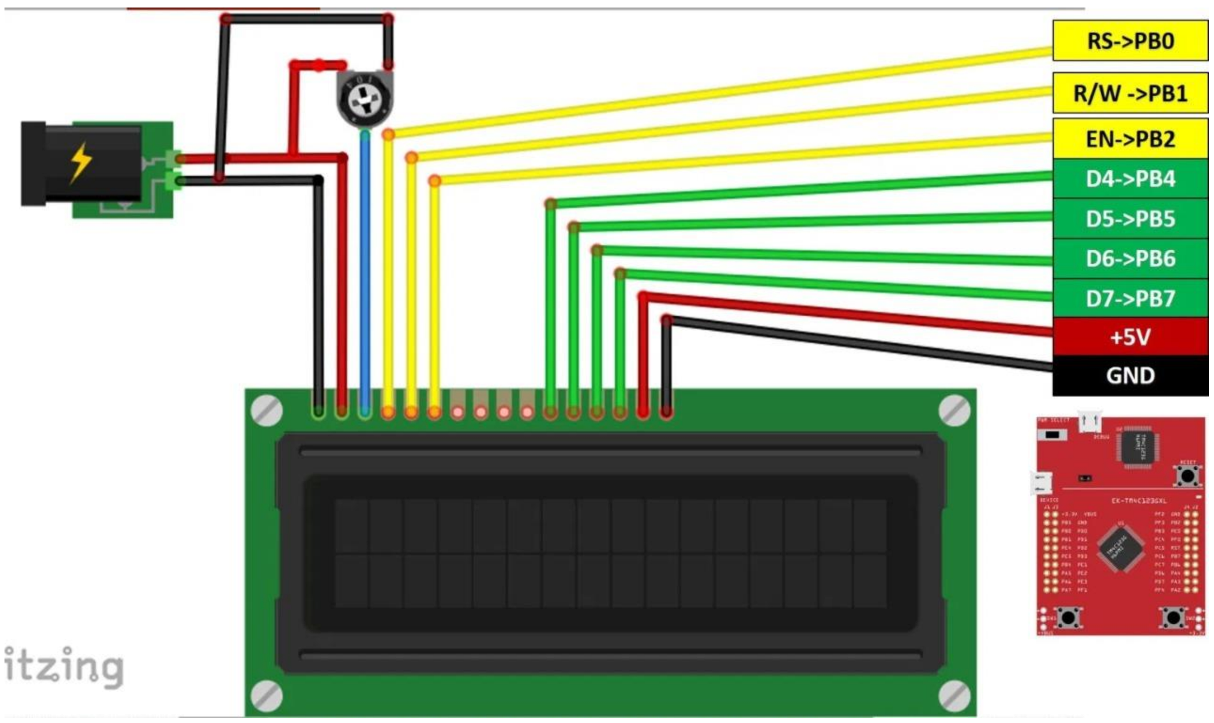
- Ten cities are defined with the following names and time differences from London.
- The cities are “London”, Paris”, “Madrid”, “Rome”, “Athens”, “Ankara”, “Istanbul”, “Cairo”, “Moscow”, “Tehran”.
- The time difference relative to London is 0, 1, 1, 1, 2, 2, 2, 2, 3, 4.
- The message that is printed on LCD consists of hours, minutes and seconds.

4. Diagrams and flow charts

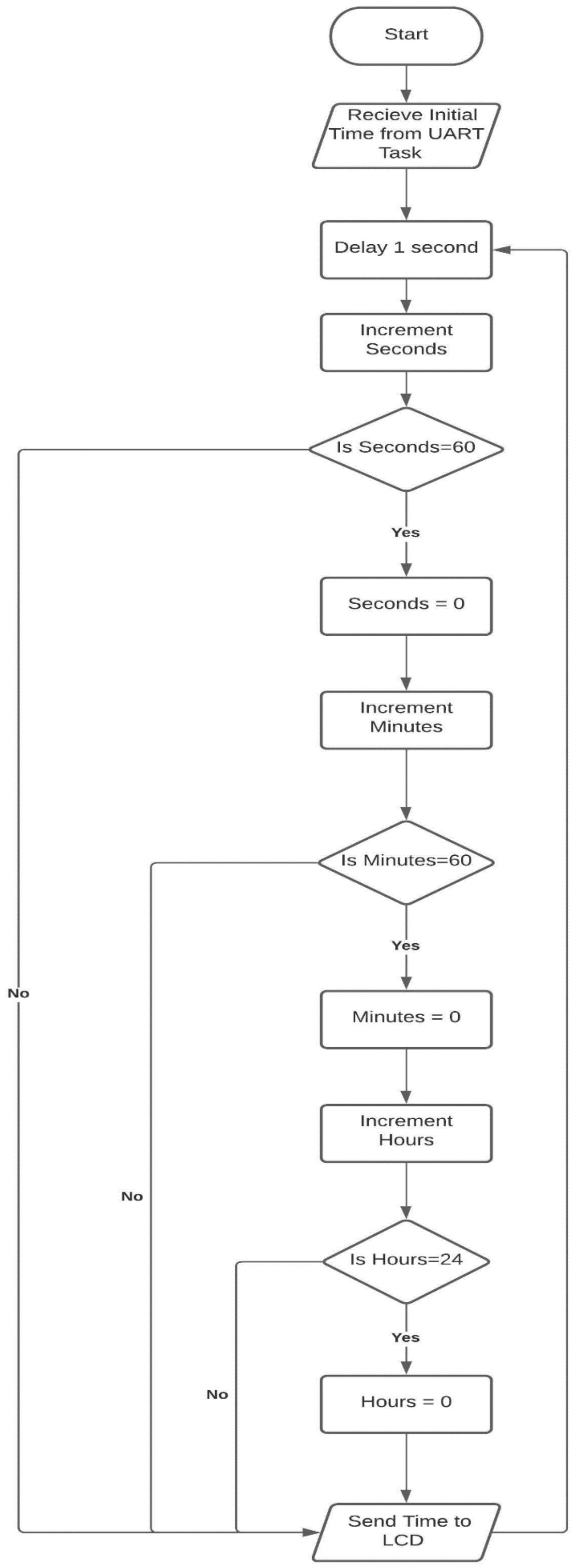
- Block Diagram



- LCD-interfacing-with-TM4C123-Tiva-C-Launcpad



- Flowchart


Start
Recieve Initial Time from UART Task
Delay 1 second
Increment Seconds
Is Seconds=60
Yes
Seconds = 0
Increment Minutes
Is Minutes=60
Yes
Minutes = 0
Increment Hours
Is Hours=24
Yes
Hours = 0
No
No
No
Send Time to LCD

5. Description of files and functions

Files:

- **Time_in_Cities.c:** Main core of the program and Application Data
- **LCD.h:** LCD Helper functions header files.
- **LCD.c:** LCD Helper functions implementation.
- **UART.c:** UART Helper functions header files.
- **UART.h:** UART Helper functions implementation.

Functions of Time_In_Cities.c File:

- **Void TimeTask():** First time**,** it receives the initial time from UARTTask via queue 2, then updates time every 1 second and send updated time to LCDTask via queue 1
- **Void UARTTask():** Asks the user to enter the time of London (initial time), sends the time to TimeTask via queue 2 to be updated every 1 second, then asks the user to select the city from list of cities to calculate time difference
- **void LCDTask():** LCD controller task function, it receives updated time for TimeTask and displays the selected city time on LCD.

Functions of UART.c File:

- **void ReadTime(char message_buffer[]):** This is a Time reading function to read initial time from user. it Terminates reading when user hits "Enter".
- **unsigned int ReadSelection():** This is City selection reading function, it reads integer number that represents the selected city. it Terminates reading when user hits "Enter".
- **Void UARTTransmit(uint8_t message_data):** Transmit character using UART
- **Char UARTRecieve():** Receive character using UART
- **void UARTPrintString (char *out_message):** Print a full string using the character transmission function, it terminates when reaches end of the string

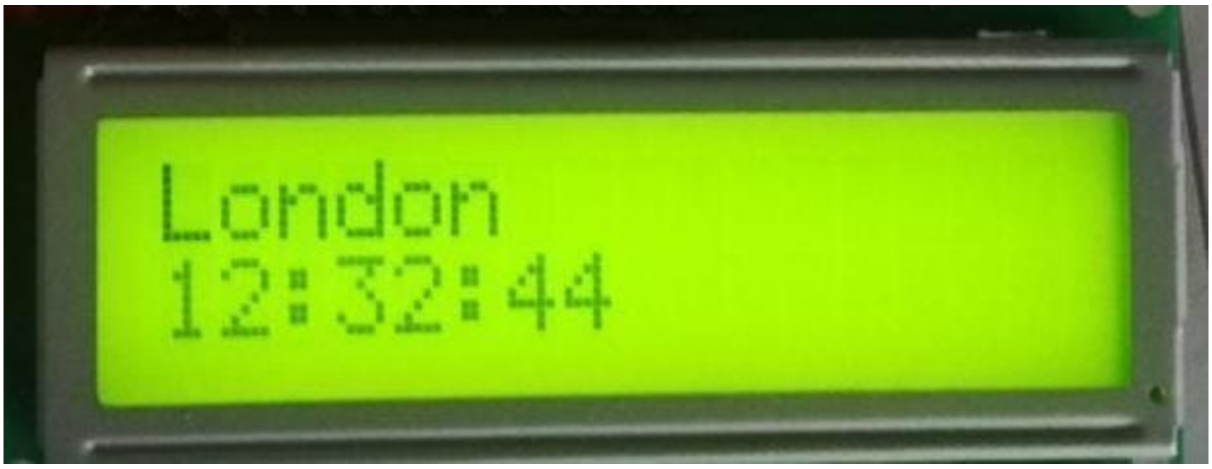
Functions of LCD.c File:

- **void LCD.Init():** LCD initialization.

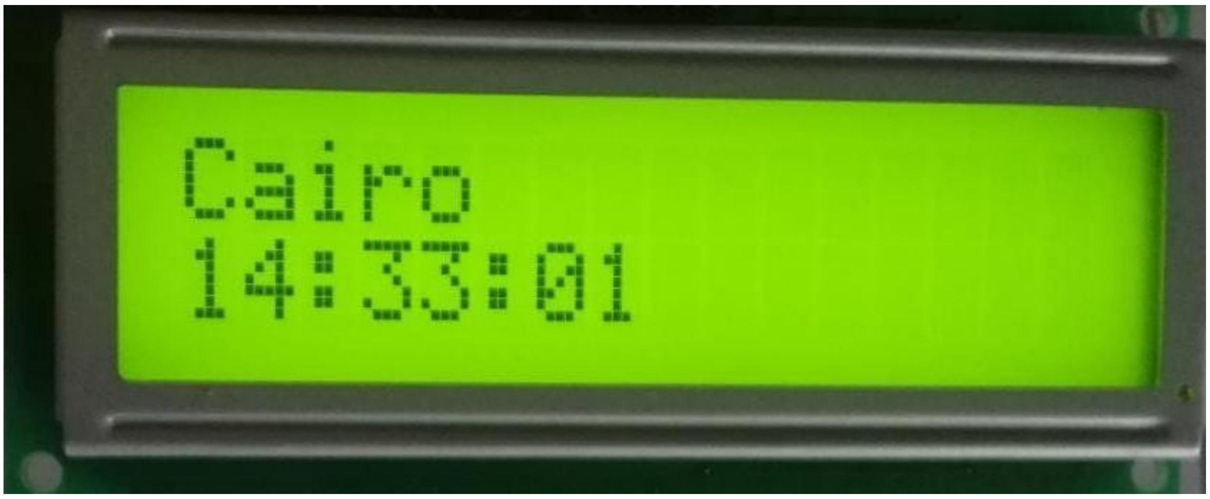
- **void LCDWriteNibble(unsigned char data, unsigned char control):** Data is sent to LCD as two nibbles of data accompanied with control commands via the EN and RS pins

- **void LCDCommand(unsigned char c):** Send a command to the LCD
- **DelayMS(int n) DelayUS(int n):** Mili and micro seconds delay functions.
- **void RemoveWhiteSpaces(char *string):**String white spaces removal
- **void LCDPrintChar(unsigned char data):** Print a character on LCD display.
- **void LCDPrintString(char *string):**Print a full string on LCD display by recursively calling LCDPrintChar.

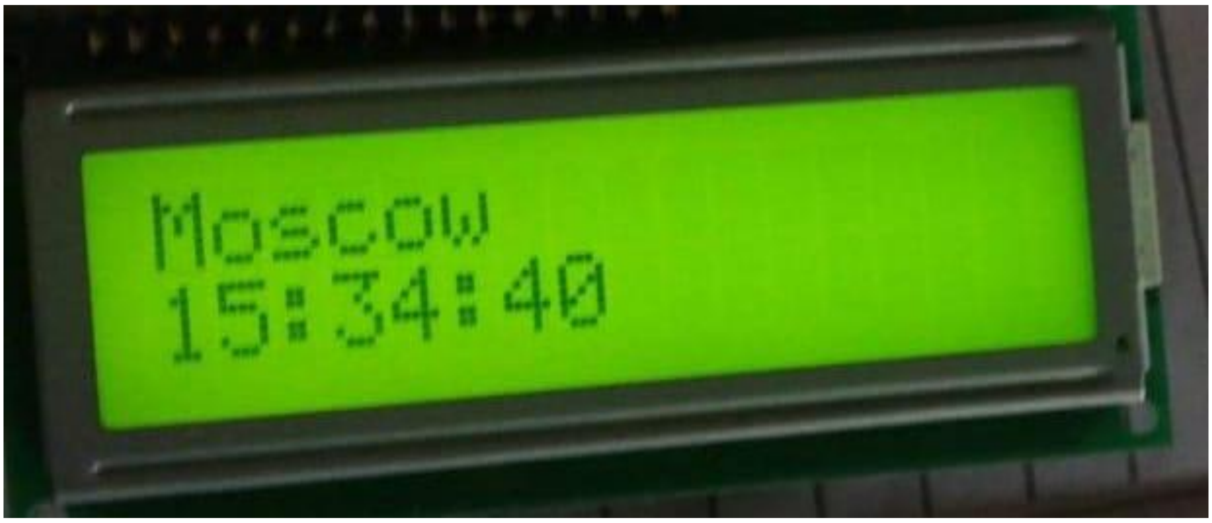
6. Screenshots sample



```
London
12:32:44
```



```
Cairo
14:33:01
```



```
Moscow
15:34:40
```

7. Codes

- *Time_in_Cities.c*

```
////////// ---------->>>>>>>>>> INCLUDES <<<<<<<<<<---------- //////////


// FreeRTOS.org includes
#include "FreeRTOS.h"
#include "FreeRTOSConfig.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

// C libraries includes
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>

// Stellaris includes
#include "TM4C123GH6PM.h"


////////////////////////////////////////////////////////////////////////////
/////////////////////////
////////////////////////////////////////////////////////////////////////////
/////////////////////////

////////// ---------->>>>>>>>>> AAPLICATION DATA <<<<<<<<<<----------
//////////


// Queue handlers
static QueueHandle_t queue1;
static QueueHandle_t queue2;

// Time message structure
// Time components (HH:MM:SS)
typedef struct Message {
    unsigned char hours;
    unsigned char minutes;
    unsigned char seconds;
} TimeMessage;

// World cities
char cities[][10] = {"London", "Paris", "Madrid", "Rome", "Athens",
"Ankara", "Istanbul", "Cairo", "Moscow", "Tehran"};

// Time Differences from London city
char time_diff[] = {0, 1, 1, 1, 2, 2, 2, 2, 3, 4};

// User city selection
// Global variable that represents user's selection from options menu
unsigned int selection;
```

```
////////// ---------->>>>>>>>>> TIME CONTROLLER TASK FUNCTION <<<<<<<<<<---
------- //////////


void TimeTask(void *pvParameters) {

    // The time message
    // To be recieved initially from the UART via queue2
    TimeMessage current_time;
    xQueueReceive(queue2, &current_time, portMAX_DELAY);

    for(;;) {

        // The idea is that time gets updated every 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));

        // Then we can increment the current time by one second
        current_time.seconds++;

        // Now, we need to mimic the clock
        // Every 60 seconds increments one minute
        // Every 60 minutes increments one Hour
        // Every day (24 hours) reset the clock
        // Note that we are using the 24 hours time format
        // And that's it :D
        if(current_time.seconds == 60) {

            current_time.seconds = 0;   // Reset the seconds to count a new
minute
            current_time.minutes++;     // Increment the current time by 1
minute

            // Keep incrementing minutes untill we reach one hour (60
minutes)
            if(current_time.minutes == 60) {
                current_time.minutes = 0;   // Reset the minutes to count a
new hour
                current_time.hours++;           // Increment the current
time by 1 hour

                // Keep incrementing hours untill we reach one day (24
hours)
                if(current_time.hours == 24) {
                    current_time.hours = 0; // Reset the hours to count a
new day
                }
            }
        }

        // The current time is updated on the LCD display every 1 second
        // Send current time to the LCD controller to display
        xQueueSend(queue1, &current_time, 0);

    }

}
```

```
///////////////////////////////////////////////////////////////////////////
/////////////////////////


////////// ---------->>>>>>>>>> LCD CONTROLLER TASK FUNCTION <<<<<<<<<<----
------ //////////


#include "LCD.h"


void LCDTask(void *pvParameters) {

    TimeMessage current_time; // Current time to display on the LCD
    char time_as_string[7];          // The time string in "char" type


    // Initialize the LCD
    // Communication via portB
    LCDInit();           // Init
    //LCDClear();            // Clear display
    selection = 0;  // Reset city selection to London

    // What this task in intended to do is simply three things
    // Acquire current time from the current controller task via queue1
    // Calculate time differences from london
    // and convert the time format into string format that is suitable to
be displayed
    // Print the selected city current time on the LCD display
    for (;;) {

        xQueueReceive(queue1, &current_time, portMAX_DELAY); // Get current
time

        // Note that initially the LCD displays the time in London
        // Which is given by the user at the beginning
        // After that it displays the selected city by user
        LCDPrintString(cities[selection]);

        // Adjust each time component and display it

        // Hours adjustment
        current_time.hours += time_diff[selection];
// calculate time difference
        if(current_time.hours > 23) { current_time.hours -= 24; }
// Cycle over one day (24 hours)
        memcpy(time_as_string, &current_time.hours, sizeof
current_time.hours); // Convert to string
        RemoveWhiteSpaces(time_as_string);
// Remove white Spaces
        // If (hours < 10 for example 9) -> (hours = 09);
        if(current_time.hours < 10) {
            time_as_string[1] = time_as_string[0];
// Move to second digit
            time_as_string[0] = '0';
// Insert leadig zero at first digit
            time_as_string[2] = '\0';
// Concatenate the null character
        }
        LCDPrintString(time_as_string);
// Display hours
```

```
        LCDPrintChar(':');
// Time separator

        // Minutes adjustment
        // Note that minutes are not affected by time differences
        memcpy(time_as_string, &current_time.minutes, sizeof
current_time.minutes); // Convert to string
        RemoveWhiteSpaces(time_as_string);
// Remove white Spaces
        // If (minutes < 10 for example 9) -> (minutes = 09)
        if(current_time.minutes < 10) {
            time_as_string[1] = time_as_string[0];
// Move to second digit
            time_as_string[0] = '0';
// Insert leadig zero at first digit
            time_as_string[2] = '\0';
// Concatenate the null character
        }
        LCDPrintString(time_as_string);
// Display hours
        LCDPrintChar(':');
// Time separator

        // Seconds adjustment
        // Note that seconds are not affected by time differences
        memcpy(time_as_string, &current_time.seconds, sizeof
current_time.seconds); // Convert to string
        RemoveWhiteSpaces(time_as_string);
// Remove white Spaces
        // If (seconds < 10 for example 9) -> (seconds = 09)
        if(current_time.seconds < 10) {
            time_as_string[1] = time_as_string[0];
// Move to second digit
            time_as_string[0] = '0';
// Insert leadig zero at first digit
            time_as_string[2] = '\0';
// Concatenate the null character
        }
        LCDPrintString(time_as_string);
// Display hours
        // No need for (:) the time separator at the end

    }
}
```

```
////////// ---------->>>>>>>>>> UART CONTROLLER TASK FUNCTION <<<<<<<<<<---
------- //////////

#include "UART.h"


void UARTTask(void *pvParameters) {

    // UART initialization
    UARTInit();

    // Acquire initial time from user (London)
    UARTPrintString("\n\rWELCOME TO TIME IN CITIES APPLICATION");
    UARTPrintString("\n\r=======================================");
    UARTPrintString("\rPlease enter current time in london with the format
(hh:mm:ss):");

    // Read initial time from user
    char message_buffer[10];    // Input message buffer
    TimeMessage init_time;      // Initial time message
    ReadTime(message_buffer);   // Read time
    // Now we have the initial time as meaningless characters
    // We need to extract time data from it
    // Note that each 2 characters represent a time component
    init_time.hours = 10*(message_buffer[0] - '0') + message_buffer[1] - 0;
// Convert hours
    init_time.minutes = 10*(message_buffer[3] - '0') + message_buffer[4] -
0;   // Convert minutes
    init_time.seconds = 10*(message_buffer[6] - '0') + message_buffer[7] -
0;   // Convert seconds

    // Send the init time to the Time controller task via queue2
    xQueueSend(queue2, &init_time, 0);

    // All this was initial
    // What this task should be doing all the time (looping)
    // is to display options menu and wait for user selection
    for (;;) {

        // Send the application control window to PC
        // Consisting of city options menu to select from
        UARTPrintString("\n\r\n\r\n\rPlease select a city to view its
time:\n\r");
        for(int i = 0; i < 10; i++) {
            UARTPrintString("\n\r");
            UARTTransmit(i + '0');
            UARTPrintString(". ");
            UARTPrintString(cities[i]);
        }
        UARTPrintString("\n\r\n\rSelection: ");

        // Read user's selected city
        selection = ReadSelection();

    }

}
```

```
///////////////////////////////////////////////////////////////////////////
////////////////////////

////////// ---------->>>>>>>>>> MAIN FUNCTION <<<<<<<<<<----------
//////////


// Initializes everything
// Including
// Invokes the scheduler
int main( void ) {

    // Define the queues
    queue1 = xQueueCreate(1, 8); // Queue1 with 1 max itemset of size 8
bytes
    queue2 = xQueueCreate(1, 8); // Queue2 with 1 max itemset of size 8
bytes

    // Tasks creation
    // All tasks are created with the following parameters:
    // Task function, Task name, Minimal stack size, Null parameters,
Highest priority, Null handler
    // Note that the 3 tasks have the same priority

    // Task 1 -> the Time controller task
    xTaskCreate(TimeTask, "Time Controller", 20, NULL,
configMAX_PRIORITIES, NULL);

    // Task 2 -> the LCD controller task
    xTaskCreate(LCDTask, "LCD Controller", 20, NULL, configMAX_PRIORITIES,
NULL);

    // Task 3 -> the UART controller task
    xTaskCreate(UARTTask, "UART Controller", 20, NULL,
configMAX_PRIORITIES, NULL);

    // Invoke the FreeRTOS scheduler
    vTaskStartScheduler();

    for( ;; ); // Should never reach here

}


///////////////////////////////////////////////////////////////////////////
/////////////////////////
///////////////////////////////////////////////////////////////////////////
/////////////////////////
```

- ***UART.h***

```
/////////////////////////////////////////////////////////////////////////////
/////////////////////////
/////////////////////////////////////////////////////////////////////////////
/////////////////////////

////////// ---------->>>>>>>>>> UART HELPING FUNCTIONS HEADER FILE
<<<<<<<<<<---------- //////////


// Stellaris includes
#include "TM4C123GH6PM.h"

// APIs to the UART functions
void UARTInit();                                                          //
UART module initialization over PortA
void UARTTransmit( uint8_t data_message );  // UART character transmission
void UARTPrintString(char *string);                         // UART full
message line transmission
char UARTRecieve();                                                 // UART
character recieving
void ReadTime(char message_buffer[]);               // Read initial time
unsigned int ReadSelection();                               // Read city
selection
```

- ***UART.c***

```
////////// ---------->>>>>>>>>> UART HELPING FUNCTIONS IMPLEMENTATION
<<<<<<<<<<---------- //////////


// UART header file inclusion
#include "UART.h"


// UART initialization
void UARTInit() {

    SYSCTL->RCGCUART |= 0x01;  // Clock gating enable to UART0
  SYSCTL->RCGCGPIO |= 0x01;  // Clock gating enable to GPIO PortA

  // UART0 configuration
  UART0->CTL = 0;            // Disable the UART0 module to configure
  UART0->IBRD = 104;  // For 9600 baud rate, integer = 104
  UART0->FBRD = 11;   // For 9600 baud rate, fractional = 11
  UART0->CC = 0;      // Use system clock
  UART0->LCRH = 0x60; // Configure data lenght as 8-bit, with no parity
bit, and disable FIFO
  UART0->CTL = 0x301; // Enable the UART0 module, alongside with Rx and Tx

  // PortA configuration to use as UART
    // Using Pin0 and Pin1 from the USB communication
  GPIOA->DEN = 0x03;                // Digital enable Pin0 and Pin1
    GPIOA->AMSEL = 0x03;            // Turn off analg function
  GPIOA->AFSEL = 0x03;          // Alter pins function, WHICH BY DEFAULT
ARE UART
  GPIOA->PCTL = 0x00000011; // Configure pins for UART use

}


// Character transmission using UART
void UARTTransmit( uint8_t message_data ) {

    while((UART0->FR & 0x20) != 0); // Wait for the buffer to be not full
(TXFF = 0)
    UART0->DR = message_data;               // Transmit a character

}
```

```
// Print a full string
// Uses the character transmission function
void UARTPrintString(char *out_message) {

    // Print untill null character (end of the string)
    while(*out_message) {
        UARTTransmit(*(out_message++));
    }

}


// Recieve a character through UART
char UARTRecieve() {

    char recieved_data;

    while((UART0->FR & 0x10) != 0); // Wait for a new input (RXFE = 0)
    recieved_data = UART0->DR;          // Recieve a character

    return (unsigned char)recieved_data;

}


// Time reading function
// To read initial time from user
// Terminates reading when user hits "Enter"
void ReadTime(char message_buffer[]) {

    unsigned char in_char;  // Character to read
    unsigned char i = 0;        // Message buffer iterator

    // Keep reading untill user hits "Enter"
    for(;;) {
        in_char = UARTRecieve();            // Recive characters from user
        UARTTransmit(in_char);              // Echo user's input on
terminal
        if( in_char == '\r') break;     // Check if "Enter" is hit to
terminate
        message_buffer[i] = in_char;    // Add the input char to message
        i++;
    }

    message_buffer[i] = '\0';   // Concatenate the null character

}
```

```
// City selection reading function
// Simply reads intiger number that represents the selected city
unsigned int ReadSelection() {

    unsigned int selection = 0; // The selection integer
    unsigned char in_char;          // The UART input

    for(;;) {
        in_char = UARTRecieve();                              // Recive
characters from user
        UARTTransmit(in_char);                                // Echo user's
input on terminal
        if( in_char == '\r') break;                       // Check if "Enter" is
hit to terminate
        in_char = in_char - '0';                              // Pure number
        selection = 10*selection + in_char; // Int numberic value from char
    }

    return selection;

}
```

- ***LCD.h***

```
////////// ---------->>>>>>>>>> LCD HELPING FUNCTIONS HEADER FILE
<<<<<<<<<<---------- //////////


// Stellaris includes
#include "TM4C123GH6PM.h"

// PortB pins mapping to LCD pins
#define LCD GPIOB   // LCD as symbolic name for GPIO PortB
#define RS 0x01     // RS pin connected to PortB Pin 0
#define RW 0x02     // RW pin connected to PortB Pin 1
#define EN 0x04     // EN pin connected to PortB Pin 2
// LCD Data pins (4:7) connected to GPIO PortB Data register pins (4:7)

// Some useful symbolic names for LCD command
// Instead of using meaningless command numbers
#define ClearDisplay            0x01
#define MoveCursorRight 0x06
#define CursorBlink       0x0F
#define FunctionSet4bit     0x28
#define SetFontSize5x7    0x20


// APIs to LCD functions
void LCDInit(void);
// LCD initialization
void LCDWriteNibble(unsigned char data, unsigned char control); // Writes
4-bits
void LCDCommand(unsigned char command);
// Send commands to LCD
void LCDPrintChar(unsigned char data);
// Prints ASCII character
void LCDPrintString (char *string);
// Prints full string
void RemoveWhiteSpaces(char *string);
// String white spaces removal
void DelayMS(int n);
// Delay for specified mili seconds
void DelayUS(int n);
// Delay for specified micro seconds
```

- ***LCD.c***

```
////////// ---------->>>>>>>>>> LCD HELPING FUNCTIONS IMPLEMENTATION
<<<<<<<<<<---------- //////////

// LCD header file inclusion
#include "LCD.h"


// LCD initialization
// Bassically initializes PortB to communicate with the LCD
// Then sends some init commands to it
void LCDInit(void) {

    // PortB initialization
    SYSCTL->RCGCGPIO |=(1<<1);  // Clock gating enable to GPIO PortB
    LCD->DEN |=0xFF;                         // Digital enable PortB pins
    LCD->DIR |=0xFF;                         // Set PortB pins as output

    // Lcd init commands
    LCDCommand(FunctionSet4bit);    // Select 4-bit mode (pins 4:7)
    LCDCommand(SetFontSize5x7);     // Font size 5x7 and 2 rows of LCD
    LCDCommand(MoveCursorRight);    // shift cursor right
    LCDCommand(ClearDisplay);       // Clear the display
    LCDCommand(CursorBlink);        // Enable the display by blinking the
cursor

}


// Data is sent to LCD as two nibbles of data
// accompanied with control commands via the EN and RS pins
void LCDWriteNibble(unsigned char data, unsigned char control) {

    data &= 0xF0;                                         // Extract data
upper nibble
  control &= 0x0F;                              // Extract control lower
nibble
  LCD->DATA = data | control;                // //  Set RS and RW to zero for
write operation
  LCD->DATA = data | control | EN;  // Provide Pulse to Enable pin to
perform write operation
  DelayUS(0);
  LCD->DATA = data;                              // Send data to LCD
  LCD->DATA = 0;                                     // Stop writing
data to LCD

}
```

```c
// Send a command to the LCD
// The same nibble idea applies
void LCDCommand(unsigned char command) {

    LCDWriteNibble(command & 0xF0, 0);  // Write upper nibble to LCD
  LCDWriteNibble(command << 4, 0);      // Write lower nibble to LCD

  if (command < 4)
        DelayMS(2);
  else
        DelayUS(40);

}


// Print a character on LCD display
void LCDPrintChar(unsigned char data) {

    LCDWriteNibble(data & 0xF0, RS);    // Writes upper nibble with RS = 1
to write data
    LCDWriteNibble(data << 4, RS);    // Writes lower nibble with RS = 1 to
write data
    DelayUS(40);

}


// Print a full string on LCD display
// by recursively calling LCDPrintChar
void LCDPrintString(char *string) {

    // Send each char of the string till the NULL terminator
    while(*string) {
        LCDPrintChar(*(string++));
    }

}


// String white spaces removal
// Its idea is simple
// Replace the white space with next character
// Propagate the white space
// Terminate the string with null character to exclude the white spaces
void RemoveWhiteSpaces(char *string)
{
    int count = 0;

    for (int i = 0; string[i]; i++) {
        if (string[i] != ' ') {
            string[count++] = string[i];
        }
    }
    string[count] = '\0';
}
```

```c
// Mili seconds delay
void DelayMS(int n)
{
    int i,j;
    for(i = 0; i < n; i++)
    for(j = 0; j < 3180; j++) {}
}

// Micro seconds delay
void DelayUS(int n)
{
    int i,j;
    for(i = 0; i < n; i++)
    for(j = 0; j < 3; j++)   {}

}
```