```
/*************************************************
 //Description: Initialization and Creation of tasks for static scheduling in Free RTOS
  * 2 Periodic- Task 1 and Task 2
  * 1 Sporadic - Task 3
  * 1 Aperiodic - Task 4
  * Author : Sanjana Kalyanappagol
  *        Shreyas V
  * Date: May 4, 2017
  *
  * ************************************************
  */


#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "drivers/rgb.h"
#include "drivers/buttons.h"
#include "driverlib/ssi.h"
#include "utils/uartstdio.h"
#include "led_task.h"
#include "priorities.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "driverlib/adc.h"
#include <math.h>
#include "driverlib/gpio.c"
#include "inc/hw_gpio.h"
#include "qs-rgb.h"


//*****************************************************************************
//
// The stack size for the LED toggle task.
//
//*****************************************************************************
#define LEDTASKSTACKSIZE        128         // Stack size in words
#define outerdelay 6250 //delay for execution time
#define innerdelay 2000
```

```
#define Button_PERIPH SYSCTL_PERIPH_GPIOF
#define ButtonBase GPIO_PORTF_BASE
#define Button1 GPIO_PIN_4
#define Button2 GPIO_PIN_0
#define ButtonInt1 GPIO_INT_PIN_4
#define ButtonInt2 GPIO_INT_PIN_0


#define Button_PERIPH1 SYSCTL_PERIPH_GPIOC
#define ButtonBase1 GPIO_PORTC_BASE
#define SevenBase GPIO_PORTB_BASE




//*****************************************************************************
//
// Default LED toggle delay value. LED toggling frequency is twice this number.
//
//*****************************************************************************
char str[10]= "\0";
char morseout = '\0';
int rt=0,qz;
uint8_t pq=0;
int counter=0;
char str1[26][4]={".-","-...","-.-.","-..",".","..-.","--.",
             "....","..",".---","-.-",".-..","--","-.",
             "---",".--.","--.-",".-.","...","-","..-",
             "...-",".--","-..-","-.--","--.."};
volatile int g=1;
int check_prime(int);
extern void ConfigureUART();



//
// [G, R, B] range is 0 to 0xFFFF per color.
//
static uint32_t g_pui32Colors[3] = { 0x0000, 0x0000, 0x0000 };
volatile tAppState g_sAppState;

xTaskHandle xTask1;
xTaskHandle xTask2;
xTaskHandle xTask3;
xTaskHandle xTask4;
xSemaphoreHandle share;
xSemaphoreHandle morse;
xSemaphoreHandle sp;
xSemaphoreHandle g_pUARTSemaphore;

/**********************************************************
*Function to compare the string entered using morse code
*circuitry to decide which letter was entered by the user
 * *********************************************************
 */
char compare(char str[])
   {
      int i=0;
      pq=0;
      char letter;

      while(i<26)
      {
      qz=strcmp(str1[i],str);
         if(qz==0)
         {
            if(i==0)
            letter='a';
            else if(i==1)
            letter='b';
            else if(i==2)
            letter='c';
            else if(i==3)
            letter='d';
            else if(i==4)
            {
             strcpy(str,"");
             letter='e';
             }
             else if(i==5)
             letter='f';
             else if(i==6)
             letter='g';
             else if(i==7)
             letter='h';
             else if(i==8)
             letter='i';
             else if(i==9)
             letter='j';
             else if(i==10)
             letter='k';
             else if(i==11)
             letter='l';
             else if(i==12)
             letter='m';
             else if(i==13)
             letter='n';
             else if(i==14)
             letter='o';
             else if(i==15)
             letter='p';
             else if(i==16)
             letter='q';
             else if(i==17)
             letter='r';
             else if(i==18)
             letter='s';
             else if(i==19)
             letter='t';
             else if(i==20)
             letter='u';
             else if(i==21)
             letter= 'v';
             else if(i==22)
             letter='w';
             else if(i==23)
             letter='x';
             else if(i==24)
             letter='y';
             else
             letter='z';
           }
      i++;
   }
     return letter;
}

//HW interrupt for two on-board push buttons for the morse code
static void PortFIntHandler()
{

   g_sAppState.ui32Buttons = GPIOIntStatus(ButtonBase,true);

   switch(g_sAppState.ui32Buttons & ALL_BUTTONS)
   {

         case LEFT_BUTTON:
         //
         // Perform left button operation.
         //
          if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
            {
              UARTprintf("\nInterrupt occurred - DOT\n");
              xSemaphoreGive(g_pUARTSemaphore);
            }
          str[pq++]='.';  //store a dot in the string
          SysCtlDelay(9000000);
          GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
          break;

         case RIGHT_BUTTON:

         //
         // Perform the right button operation.
         //
         if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
           {
             UARTprintf("\nInterrupt occurred - DASH\n");
             xSemaphoreGive(g_pUARTSemaphore);
           }
         str[pq++]='-'; //store a dash in the string
        GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
        SysCtlDelay(9000000);
        break;
   }
         GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
}

/*InterruptHandler for push button used for end of string in morse code*/
static void PortCIntHandler()
{
   uint32_t status=0;
   static portBASE_TYPE xHigherPriorityTaskWoken;
   status = GPIOIntStatus(ButtonBase1,true);
   if(status)
   {
    if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
    {
      UARTprintf(" ENTER Interrupt Occured!!! \n");
      xSemaphoreGive(g_pUARTSemaphore);
    }
    SysCtlDelay(9000000);
    GPIOIntClear(ButtonBase1,status);
    xSemaphoreGiveFromISR(morse,&xHigherPriorityTaskWoken);
   }
}

/*Task 1 -Periodic Task- Temperature SEnsor configure with ADC*/

static void Task1(void *pvParameters)
{
   uint32_t ui32ADC0Value[4];
```

```
 volatile int task1_i,task1_j;
 volatile uint32_t ui32TempAvg, ui32TempValueC, ui32TempValueF;
 task1_i=0; task1_j=0;
 int flag=0;
 portTickType waketime1 =xTaskGetTickCount();


SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
//Temperature initialisation
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
ADCSequenceEnable(ADC0_BASE, 1);

while(1)
{
  // Update the LED buffer to turn off the currently working.

        g_pui32Colors[0] = 0x8000;
        g_pui32Colors[1] = 0x0000;
        g_pui32Colors[2] = 0x0000;

       // Configure the new LED settings.
        RGBColorSet(g_pui32Colors);
       xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY);
       UARTprintf("Task 1 starts at %dms\n",xTaskGetTickCount());
       xSemaphoreGive(g_pUARTSemaphore);
       g++;
       if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
           {
             UARTprintf("%d\n",g);
             xSemaphoreGive(g_pUARTSemaphore);
           }
      switch(g)
           {
              case 1: flag=0;
                     break;
              case 2: flag=1;
                            break;
              case 3 : flag=1;
                            break;
              default: flag=check_prime(g);
                            break;
            }
      if(flag)
```

```
         {
          xSemaphoreGive(sp);
          vTaskPrioritySet(xTask3, configMAX_PRIORITIES-1); //Set higher priority for Task3
         }

     flag=0;
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
       {
         UARTprintf("Task1 attempting to take mutex at %dms\n",xTaskGetTickCount());
         xSemaphoreGive(g_pUARTSemaphore);
       }
     if(xSemaphoreTake(share,portMAX_DELAY))
       {
         RGBEnable();
         if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
         {
           UARTprintf("Task1 mutex taken at %dms\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
         }
     //temperature code
      ADCIntClear(ADC0_BASE, 1);
      ADCProcessorTrigger(ADC0_BASE, 1);

     while(!ADCIntStatus(ADC0_BASE, 1, false))
     {
     }

     ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
     ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] +2)/4;
     ui32TempValueC = (((1475 - ((2475 * ui32TempAvg)) / 4096)/10)+7);
     ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
       UARTprintf("in celsius - %d \n",ui32TempValueC);
       UARTprintf("in fahrenheit - %d \n",ui32TempValueF);
       xSemaphoreGive(g_pUARTSemaphore);
     }

     GPIOPinWrite(SevenBase,GPIO_PIN_0,0xFF); //Seven Segment Display
     GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_3,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_6,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_7,0xFF);
     /*Delay loop to get required execution time*/
```

```
       for(task1_i=outerdelay*2;task1_i>0;task1_i--)
       {
           for(task1_j=innerdelay;task1_j>0;task1_j--)
           {

           }
       }
       if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
       {
           UARTprintf("Task1 Giving mutex at %dms\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
       }
       xSemaphoreGive(share);

       if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
       {
           UARTprintf("Task 1 ends at %d ms\n\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
       }
       vTaskDelayUntil(&waketime1, 7000);
    }
  }
}

/*Task 2-Periodic Task- SD Card Configured with SPI(SSI)*/
static void Task2(void *pvParameters)
{

   volatile int i,j;
   portTickType waketime2 =xTaskGetTickCount();
   while(1)
   {
     // Update the LED buffer to turn off the currently working.

       g_pui32Colors[0] = 0x0000;
       g_pui32Colors[1] = 0x8000;
       g_pui32Colors[2] = 0x0000;

      // Configure the new LED settings.

      RGBColorSet(g_pui32Colors);


       if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
       {
          UARTprintf("Task 2 starts at %dms\n",xTaskGetTickCount());
          xSemaphoreGive(g_pUARTSemaphore);
       }
    if(xSemaphoreTake(share,portMAX_DELAY))
    {
      RGBEnable();
      if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
         UARTprintf("Task2 mutex taken at %dms\n",xTaskGetTickCount());
         xSemaphoreGive(g_pUARTSemaphore);
      }

     GPIOPinWrite(SevenBase,GPIO_PIN_0,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_2,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_3,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_4,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);

     cmd_twentyfour(); //Function call for SD Card write

     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
       UARTprintf("\n\r");
       UARTprintf("Task 2 SD Write Successful\n");
       UARTprintf("\n\r");
       xSemaphoreGive(g_pUARTSemaphore);
     }

/*Delay loop to get required execution time*/
     for(i=outerdelay*2;i>0;i--)
     {

       for(j=innerdelay;j>0;j--)
       {

       }
     }
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
       UARTprintf("Task2 Giving mutex at %dms\n",xTaskGetTickCount());
       xSemaphoreGive(g_pUARTSemaphore);
     }

     xSemaphoreGive(share);
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
       UARTprintf("Task 2 ends at %d ms\n\n",xTaskGetTickCount());
      xSemaphoreGive(g_pUARTSemaphore);
     }

    }

    else
    {
      if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
      UARTprintf("Task2-unable to take mutex\n");
      xSemaphoreGive(g_pUARTSemaphore);
      }
    }
    vTaskDelayUntil(&waketime2, 8000);
    }
}
static void Task3(void *pvParameters)
{


  volatile int i,j;
  portTickType waketime3 =0;
 while(1)
  {

    if(xSemaphoreTake(sp,portMAX_DELAY))
      {
      if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
       UARTprintf("Task 3(SPORADIC) starts at %dms\n",xTaskGetTickCount());
       UARTprintf("g is prime %d\n",g);
       xSemaphoreGive(g_pUARTSemaphore);
      }
      GPIOPinWrite(SevenBase,GPIO_PIN_0,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_3,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
      GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
      GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);
      /*Delay loop to get required execution time*/
      for(i=outerdelay*1;i>0;i--)
      {
          for(j=innerdelay;j>0;j--)
          {
          }
       }
      if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
        UARTprintf("Task 3(SPORADIC) ends at %d ms\n\n",xTaskGetTickCount());
        xSemaphoreGive(g_pUARTSemaphore);
      }
    }
  }
}

static void Task4(void *pvParameters)
{

  while(1)
  {
    if(xSemaphoreTake(morse,portMAX_DELAY))
    {
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
       UARTprintf("Task 4(APERIODIC) starts %dms\n",xTaskGetTickCount());
       xSemaphoreGive(g_pUARTSemaphore);
      }

    morseout = compare(str); //function call to compare the morse string and return the letter it
represents

    if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
    {
        UARTprintf(" string is %c \n",morseout);
        xSemaphoreGive(g_pUARTSemaphore);
    }

    GPIOPinWrite(SevenBase,GPIO_PIN_0,0xFF); //Seven Segment Display
    GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
    GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
    GPIOPinWrite(SevenBase,GPIO_PIN_3,0xFF);
    GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
    GPIOPinWrite(SevenBase,GPIO_PIN_5,0x00);
    GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
    GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);

    if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
    {
      UARTprintf("Task 4(APERIODIC) ends at %d ms\n\n",xTaskGetTickCount());
      xSemaphoreGive(g_pUARTSemaphore);
    }
   }
  }
}
```

```
//*****************************************************************************
//
// Initializes of all tasks an configurations.
//
//*****************************************************************************
uint32_t LEDTaskInit(void)
{
    //
    // Initialize the GPIOs and Timers that drive the three LEDs.
    //

    RGBInit(1);
    RGBIntensitySet(0.3f);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    // Unlock PF0 so we can change it to a GPIO input
    // Unlock PF0 so we can change it to a GPIO input
    // Once we have enabled (unlocked) the commit register then re-lock it
    // to prevent further changes.  PF0 is muxed with NMI thus a special case.
    //
    HWREG(BUTTONS_GPIO_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(BUTTONS_GPIO_BASE + GPIO_O_CR) |= 0x01;
    HWREG(BUTTONS_GPIO_BASE + GPIO_O_LOCK) = 0;
    //
    // Enable interrupts to the processor.
    //
    ROM_IntMasterEnable();

    //Configure direction of the GPIO Pins for seven segment display
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_0);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_1);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_2);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_3);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_4);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_5);
    GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_6);
    //Configure direction of the GPIO Pins for push buttons used for morse code
    GPIOPinTypeGPIOInput(ButtonBase, Button1);
    GPIOPinTypeGPIOInput(ButtonBase, Button2);
    GPIOPinTypeGPIOInput(ButtonBase1, Button1);
    //Configure the current consumed by GPIO Pis
    GPIOPadConfigSet(ButtonBase ,Button1,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
    GPIOPadConfigSet(ButtonBase ,Button2,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
    GPIOPadConfigSet(ButtonBase1 ,Button1,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
    //Set the iNterrupt type for push buttons used in morsecode circuitry
    GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_4, GPIO_FALLING_EDGE);
    GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0, GPIO_FALLING_EDGE);
```

```
  GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_4, GPIO_RISING_EDGE);
  //Register th enames of the interrupt handler and enable the interrupt for GPIO Pins
  GPIOIntRegister(GPIO_PORTF_BASE,PortFIntHandler);
  GPIOIntRegister(GPIO_PORTC_BASE,PortCIntHandler);
  GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
  GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_0);
  GPIOIntEnable(GPIO_PORTC_BASE, GPIO_INT_PIN_4);

  //Create mutex and semaphore
  vSemaphoreCreateBinary(sp);
  share=xSemaphoreCreateMutex();
  vSemaphoreCreateBinary(morse);
  g_pUARTSemaphore = xSemaphoreCreateMutex();

  // Create the tasks.

  if(xTaskCreate(Task1, (signed portCHAR *)"LED1", LEDTASKSTACKSIZE, NULL,
          tskIDLE_PRIORITY + 3, &xTask1) != pdTRUE)
  {
    return(1);

  }

  // Success.
     if(xTaskCreate(Task2, (signed portCHAR *)"LED2", LEDTASKSTACKSIZE, NULL,
          tskIDLE_PRIORITY + 2, &xTask2) != pdTRUE)
  {
    return(1);

  }

  // Success.

 if(xTaskCreate(Task3, (signed portCHAR *)"LED3", LEDTASKSTACKSIZE, NULL,
          tskIDLE_PRIORITY + 1, &xTask3) != pdTRUE)
  {
    return(1);

  }
      if(xTaskCreate(Task4, (signed portCHAR *)"LED4", LEDTASKSTACKSIZE, NULL,
          tskIDLE_PRIORITY + 4, &xTask4) != pdTRUE)
  {
    return(1);
  }
  // Success.
  xSemaphoreTake(morse,portMAX_DELAY);
  return(0);
}
```

```
//Function to check if the number is prime
int check_prime(int g)
{
  int c;

  for ( c = 2 ; c <= (g/2); c++ )
  {
    if ( g%c == 0 )
         {
          return 0;
         }
  }
  if ( c== g/2)
   {
      return 1;
   }
 }
```

```
/**************************************************
 //Description: SD Card library functions
  * Author : Sanjana Kalyanappagol
  *         Shreyas V
  * Date: May 4, 2017
  *
  * *************************************************
  */

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
```

```
//*****************************************************************************
//

//!
//! This program uses the following peripherals and I/O signals.
//! - SSI0 peripheral
//! - GPIO Port A peripheral (for SSI0 pins)
//! - SSI0Clk - PA2
//! - SSI0Fss - PA3
//! - SSI0Rx  - PA4
//! - SSI0Tx  - PA5
//*******************************************************************************

uint8_t sdcard_init();
uint32_t cmd_zero();
uint32_t cmd_eight();
uint8_t cmd_feight();
void  cmd_fiftyfive();
;void  acmd(void);
void cmd_fiftynine();
void cmd_ten();
void cmd_twentyfour();
void cmd_seventeen();

int k;



uint8_t sdcard_init()
{
   int i=0;
   uint32_t response=0,r[3]={0x14,0,0};
   for(i=0;i<100000;i++){;}

   GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_3,0xFF);//Chip select high

   /*80 clock cycles before initializing the SD card*/
   for(i=0;i<10;i++)
   {
      SSIDataPut(SSI0_BASE, 0xFF);
   }
   while(SSIBusy(SSI0_BASE))
   {
   }
   GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_3,0x00); //Chip select low
   GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2);

   /*Function call for command zero : to wait for SD card to become idle*/
```

```
response = cmd_zero();
 /*Wait till its respone is zero*/
 if(response!=1)
 {
     return 0;
 }

 /*Function call for command eight*/
 cmd_eight();
 i=0;
 /*Send Dummy writes*/
 for(i=0;i<3;i++)
 {
     SSIDataPut(SSI0_BASE,0xFF);
 }
  /*Delay loop*/
  for(i=0;i<100000;i++)
  {

  }
  /*Function call for command fifty eight: To get details of the SD card like its ID*/
  cmd_feight();
  /*Send Dummy writes*/
  for(i=0;i<3;i++)
  {
     SSIDataPut(SSI0_BASE,0xFF);
  }

  response= 0xFF;i=0;

  /*Keep sending command fifty five and ACMD41 untill response is '0'*/
 while(response != 0)
  {
     //Function call for command fifty five: Always sent before ACMD
      cmd_fiftyfive();
      for(i=0;i<2;i++)
             {
                 SSIDataPut(SSI0_BASE,0xFF);
             }
      /*Function all for ACMD41 : Application command*/
       acmd();
       SSIDataPut(SSI0_BASE,0xFF);
       SSIDataGet(SSI0_BASE,&response);

 }

 for(i=0;i<100000;i++)
 {
   }
   for(i=0;i<3;i++)
   {
      SSIDataPut(SSI0_BASE,0xFF);
   }
      //Function call for command fifty nine to switch off the CRC byte
     cmd_fiftynine();
     for(i=0;i<3;i++)
     {
       SSIDataPut(SSI0_BASE,0xFF);
     }

     //Function call for command ten
     cmd_ten();
     for(i=0;i<3;i++)

     {
            SSIDataPut(SSI0_BASE,0xFF);
          }
      i=0;
}



/*-------------------------------------------------------------------------------------------
                         void cmd_zero()
 -------------------------------------------------------------------------------------------
 * I/P Arguments: none
 * Return value : Response from the sd card to command zero

 * description: Used for Reseting the SD card and to put it in SPI mode
--------------------------------------------------------------------------------------------*/
uint32_t cmd_zero()
{
   uint8_t cmd[7]={0x40,0,0,0,0,0x95,0xFF};// Data bytes of the cmd0
   uint32_t resp=0;
   int i=0;


   for(i=0;i<7;i++)                      // Write command
   {

      SSIDataPut(SSI0_BASE, cmd[i]);
   }

   i=0;
   while(SSIBusy(SSI0_BASE))
      {
      }

   while((resp!= 0x01))            // Wait for response
   {
      SSIDataPut(SSI0_BASE,0xFF);
      SSIDataGet(SSI0_BASE,&resp);
      i++;
   }


   return resp;                        // Return Response from the sd card
}

/*-----------------------------------------------------------------------------------------
                          void cmd_eight()
 -----------------------------------------------------------------------------------------
 * I/P Arguments: none
 * Return value : Response from the sd card to command eight

 * description: Used for Reading 40 bit status register of the SD card
-----------------------------------------------------------------------------------------*/
uint32_t cmd_eight()
{
    uint8_t cmd[6]={0x48,0,0,1,0xAA,0x87};// Data bytes of the command
    uint32_t resp=0;
   int i=0;

   for(i=0;i<6;i++)
   {
      SSIDataPut(SSI0_BASE, cmd[i]);                           // Write command to SD card
   }

   i=0;
   while(i<6)
   {
      SSIDataPut(SSI0_BASE,0xFF);
      SSIDataGet(SSI0_BASE,&resp);                     // Get 40 bit response from the SD card
      i++;
   }

   return resp;
}

/*-----------------------------------------------------------------------------------------
                          void cmd_feight()
 -----------------------------------------------------------------------------------------
 * I/P Arguments: none
 * Return value : Response from the sd card to command fifty eight

 * description: Used for Reading 40 bit status register of the SD card
---------------------------------------------------------------------------------------*/
 uint8_t cmd_feight()
 {
     uint8_t cmd[6]={0x7A,0,0,0,0,0x75};//Command bytes for CMD58
     uint32_t resp=0xFF;
   int i=0;


   for(i=0;i<6;i++)
   {
      SSIDataPut(SSI0_BASE, cmd[i]); //Write ccommand
   }

   i=0;

   while(i<11)
   {
      SSIDataPut(SSI0_BASE,0xFF);
      SSIDataGet(SSI0_BASE,&resp);
      i++;
   }

   return resp;
 }

 /*---------------------------------------------------------------------------------------
                              void cmd_fiftyfive()
  ---------------------------------------------------------------------------------------
  * I/P Arguments: none
  * Return value : none

  * description: Used before sending application specific commands(acmd). Its respose i used to
check if SD card is
  in idle mode or working mode
 ---------------------------------------------------------------------------------------*/


 void  cmd_fiftyfive()
  {
      uint8_t cmd[7]={0x77,0,0,0,0,0x65,0xFF}; //Command bytes for CMD55
      uint32_t resp=0xFF;
      int i=0;

    for(i=0;i<7;i++)
    {
       SSIDataPut(SSI0_BASE, cmd[i]); //Write command
    }

    i=0;

   return 0;
  }

 /*---------------------------------------------------------------------------------------
                              void acmd()
  ---------------------------------------------------------------------------------------
  * I/P Arguments: none
  * Return value : none

  * description: Used for putting SD card out of idle.
 ---------------------------------------------------------------------------------------*/
 void  acmd(void)
 {
    uint8_t cmd[7]={0x69,0x40,0,0,0,0x77,0xFF};//Command bytes for ACMD41
    uint32_t resp=1;
    int i=0;


    for(i=0;i<7;i++)
    {
       SSIDataPut(SSI0_BASE, cmd[i]); //Write command
    }

    i=0;
    return 0;
 }

 /*---------------------------------------------------------------------------------------
                            cmd_fiftynine()
   ---------------------------------------------------------------------------------------
   * I/P Arguments: none
   * Return value : none

   * description: To switch off the CRC Byte
  ---------------------------------------------------------------------------------------*/
 void cmd_fiftynine()
   {
      uint8_t cmd[7]={0x7B,0,0,0,0,0xFF,0xFF}; //Command bytes for CMD59
      uint32_t resp=0xFF;
      int i=0;

     for(i=0;i<7;i++)
   {
     SSIDataPut(SSI0_BASE, cmd[i]); //Write command
   }

   i=0;
   while(SSIBusy(SSI0_BASE)) //Check for SSI Busy
       {
       }

   for(i=0;i<7;i++)
      {
        SSIDataPut(SSI0_BASE,0xFF); //SEnd Dummy writes
      }

    return 0;

  }

 /*---------------------------------------------------------------------------------------
                   cmd_ten()
   ---------------------------------------------------------------------------------------
   * I/P Arguments: none
   * Return value : none

   * description: To set the sector length to 512 bytes
  ---------------------------------------------------------------------------------------*/

 void  cmd_ten()
   {
    uint8_t cmd[7]={0x50,0,0,0,0x02,0x00,0xFF}; //Command byted for CMD10
    uint32_t resp=0xFF;
   int i=0;

   for(i=0;i<7;i++)
   {
     SSIDataPut(SSI0_BASE, cmd[i]); //write command
   }

   i=0;

   }

 /*---------------------------------------------------------------------------------------
                  cmd_twentyfour()
   ---------------------------------------------------------------------------------------
   * I/P Arguments: none
   * Return value : none
```

```
   * description: To perform CRC write
  -------------------------------------------------------------------------------------*/
 void cmd_twentyfour()
    {
        uint8_t cmd[7]={0x58,0x00,0x24,0x68,0x00,0X00,0xFF}; //Command bytes for CMD24 with
Address
        uint32_t resp=0xFF;
        int i=0;

       for(i=0;i<7;i++)
       {
          SSIDataPut(SSI0_BASE, cmd[i]);//Write command
       }
       for(i=0;i<10000;i++)
       {

       }

       SSIDataPut(SSI0_BASE,0xFF);//Dummy write

      while(resp!=0) //Check for response '0' for CMD24
      {
        SSIDataPut(SSI0_BASE,0xFF);
        SSIDataGet(SSI0_BASE,&resp);
      }

      //Send Data packet for SD card write
       SSIDataPut(SSI0_BASE,0xFE);
          for(i=0;i<512;i++)
          {
             SSIDataPut(SSI0_BASE,0x03);
          }
          SSIDataPut(SSI0_BASE,0x00);
          SSIDataPut(SSI0_BASE,0x00);
          i=0;
          //Check for response
          while((resp== 0xFF || resp==0x00))
          {
          SSIDataPut(SSI0_BASE,0xFF);
          SSIDataGet(SSI0_BASE,&resp);
          i++;
          }

      return 0;

  }

 /*-------------------------------------------------------------------------------------
```

```
                     cmd_seventeen()
   -------------------------------------------------------------------------------------------
   * I/P Arguments: none
   * Return value : none

   * description: To perform CRC read
  -------------------------------------------------------------------------------------------*/
 void cmd_seventeen()
    {
       uint8_t cmd[7]={0x51,0x00,0x24,0x68,0x00,0X00,0xFF}; //Command bytes for CMD17 with
address to read
       uint32_t resp=0xFF;
       uint32_t rsp[50];
      int i=0;

      SSIDataPut(SSI0_BASE, 0xFF);

      for(i=0;i<7;i++)
      {
        SSIDataPut(SSI0_BASE, cmd[i]); //write command
      }

      //Wait for response
      while(resp!=0)
       {
        SSIDataPut(SSI0_BASE,0xFF);
        SSIDataGet(SSI0_BASE,&resp);
        }

     //Read the data packet
      SSIDataPut(SSI0_BASE,0xFE);
        for(i=0;i<512;i++)
        {
          SSIDataPut(SSI0_BASE,0xFF);
          SSIDataGet(SSI0_BASE,&rsp[i]);
        }
        SSIDataPut(SSI0_BASE,0x00);
        SSIDataPut(SSI0_BASE,0x00);

        //Wait for response
        while(resp== 0xFF || resp==0x00)
        {
        SSIDataPut(SSI0_BASE,0xFF);
        SSIDataGet(SSI0_BASE,&resp);
        }

     return 0;
    }
```

```
/*
 * tasks.h
 *
 *  Created on: Apr 24, 2017
 *      Author: Sanjana
 */

#ifndef _TASKS_H_
#define _TASKS_H_

#define configUSE_16_BIT_TICKS            0

typedef enum
{
   eNotWaitingNotification = 0,
   eWaitingNotification,
   eNotified
} eNotifyValue;

typedef struct tskTaskControlBlock
{
   volatile StackType_t    *pxTopOfStack;  /*< Points to the location of the last item placed on the
tasks stack.  THIS MUST BE THE FIRST MEMBER OF THE TCB STRUCT. */
   portTickType waketime;
   portTickType period;
   portTickType remaining_time;
   portTickType elapsed;
   portTickType slack;



   //double AET;
   portTickType WCET;
   xTaskHandle xTask;
   #if ( portUSING_MPU_WRAPPERS == 1 )
```

```
    xMPU_SETTINGS   xMPUSettings;      /*< The MPU settings are defined as part of the port layer.
THIS MUST BE THE SECOND MEMBER OF THE TCB STRUCT. */
    BaseType_t      xUsingStaticallyAllocatedStack; /* Set to pdTRUE if the stack is a statically
allocated array, and pdFALSE if the stack is dynamically allocated. */
  #endif

  xListItem             xGenericListItem;   /*< The list that the state list item of a task is reference from
denotes the state of that task (Ready, Blocked, Suspended ). */
  xListItem             xEventListItem;    /*< Used to reference a task from an event list. */
  unsigned portBASE_TYPE  uxPriority;        /*< The priority of the task.  0 is the lowest priority. */
  portSTACK_TYPE         *pxStack;          /*< Points to the start of the stack. */
  signed char            pcTaskName[ configMAX_TASK_NAME_LEN ];/*< Descriptive name given to
the task when created.  Facilitates debugging only. */ /*lint !e971 Unqualified char types are allowed
for strings and single characters only. */

  #if ( portSTACK_GROWTH > 0 )
    portSTACK_TYPE  *pxEndOfStack;     /*< Points to the end of the stack on architectures where
the stack grows up from low memory. */
  #endif

  #if ( portCRITICAL_NESTING_IN_TCB == 1 )
    unsigned portBASE_TYPE  uxCriticalNesting;  /*< Holds the critical section nesting depth for
ports that do not maintain their own count in the port layer. */
  #endif

  #if ( configUSE_TRACE_FACILITY == 1 )
    unsigned portBASE_TYPE  uxTCBNumber;       /*< Stores a number that increments each time a
TCB is created.  It allows debuggers to determine when a task has been deleted and then recreated.
*/
    unsigned portBASE_TYPE  uxTaskNumber;      /*< Stores a number specifically for use by third
party trace code. */
  #endif

  #if ( configUSE_MUTEXES == 1 )
    unsigned portBASE_TYPE  uxBasePriority;     /*< The priority last assigned to the task - used by
the priority inheritance mechanism. */
    UBaseType_t     uxMutexesHeld;
  #endif

  #if ( configUSE_APPLICATION_TASK_TAG == 1 )
    pdTASK_HOOK_CODE pxTaskTag;
  #endif

  #if( configNUM_THREAD_LOCAL_STORAGE_POINTERS > 0 )
    void *pvThreadLocalStoragePointers[ configNUM_THREAD_LOCAL_STORAGE_POINTERS ];
  #endif

  #if ( configGENERATE_RUN_TIME_STATS == 1 )
```

```
        unsigned long ulRunTimeCounter; /*< Stores the amount of time the task has spent in the
Running state. */
    #endif

    #if ( configUSE_NEWLIB_REENTRANT == 1 )
        /* Allocate a Newlib reent structure that is specific to this task.
        Note Newlib support has been included by popular demand, but is not
        used by the FreeRTOS maintainers themselves.  FreeRTOS is not
        responsible for resulting newlib operation.  User must be familiar with
        newlib and must provide system-wide implementations of the necessary
        stubs. Be warned that (at the time of writing) the current newlib design
        implements a system-wide malloc() that must be provided with locks. */
        struct  _reent xNewLib_reent;
    #endif

    #if ( configUSE_TASK_NOTIFICATIONS == 1 )
        volatile uint32_t ulNotifiedValue;
        volatile eNotifyValue eNotifyState;
    #endif

} tskTCB;

typedef tskTCB TCB_t;

extern tskTCB TCB,*TCB1,*TCB2,*TCB3;


#define prvGetTCBFromHandle( pxHandle ) ( ( ( pxHandle ) == NULL ) ? ( TCB_t * ) pxCurrentTCB : (
TCB_t * ) ( pxHandle ) )

/* The item value of the event list item is normally used to hold the priority
of the task to which it belongs (coded to allow it to be held in reverse
priority order).  However, it is occasionally borrowed for other purposes.  It
is important its value is not updated due to a task priority change while it is
being used for another purpose.  The following bit definition is used to inform
the scheduler that the value should not be changed - in which case it is the
responsibility of whichever module is using the value to ensure it gets set back
to its original value when it is released. */
#if configUSE_16_BIT_TICKS == 1
    #define taskEVENT_LIST_ITEM_VALUE_IN_USE    0x8000U
#else
    #define taskEVENT_LIST_ITEM_VALUE_IN_USE    0x80000000UL
#endif
#endif /* _TASKS_H_ */
/**************************************************
 //Description: Initialization and Creation of tasks for dynamic scheduling in Free RTOS
  * 2 Periodic- Task 1 and Task 2
  * 1 Sporadic - Task 3
 * 1 Aperiodic - Task 4
 * Author : Sanjana Kalyanappagol
 *        Shreyas V
 * Date: May 4, 2017
 *
 * ***************************************************
 */


#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "drivers/rgb.h"
#include "drivers/buttons.h"
#include "driverlib/ssi.h"
#include "utils/uartstdio.h"
#include "led_task.h"
#include "priorities.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "driverlib/adc.h"
#include <math.h>
#include "driverlib/gpio.c"
#include "inc/hw_gpio.h"
#include "qs-rgb.h"


//*****************************************************************************
//
// The stack size for the LED toggle task.
//
//*****************************************************************************
#define LEDTASKSTACKSIZE        128         // Stack size in words
#define outerdelay 6250 //delay for execution time
#define innerdelay 2000


#define Button_PERIPH SYSCTL_PERIPH_GPIOF
#define ButtonBase GPIO_PORTF_BASE
#define Button1 GPIO_PIN_4
#define Button2 GPIO_PIN_0
```

```
#define ButtonInt1 GPIO_INT_PIN_4
#define ButtonInt2 GPIO_INT_PIN_0


#define Button_PERIPH1 SYSCTL_PERIPH_GPIOC
#define ButtonBase1 GPIO_PORTC_BASE
#define SevenBase GPIO_PORTB_BASE




//*****************************************************************************
//
// Default LED toggle delay value. LED toggling frequency is twice this number.
//
//*****************************************************************************
char str[10]= "\0";
char morseout = '\0';
int rt=0,qz;
uint8_t pq=0;
int counter=0;
char str1[26][4]={".-","-...","-.-.","-..",".","..-.","--.",
            "....","..",".---","-.-",".-..","--","-.",
            "---",".--.","--.-",".-.","...","-","..-",
            "...-",".--","-..-","-.--","--.."};
volatile int g=1;
int check_prime(int);
extern void ConfigureUART();
extern int p;
extern TCB_t * volatile pxCurrentTCB;
UBaseType_t checkp;




//
// [G, R, B] range is 0 to 0xFFFF per color.
//
static uint32_t g_pui32Colors[3] = { 0x0000, 0x0000, 0x0000 };
volatile tAppState g_sAppState;

xTaskHandle xTask1;
xTaskHandle xTask2;
xTaskHandle xTask3;
xTaskHandle xTask4;
xSemaphoreHandle share;
xSemaphoreHandle morse;
xSemaphoreHandle sp;
xSemaphoreHandle g_pUARTSemaphore;
```

```
/********************************************************
*Function to compare the string entered using morse code
*circuitry to decide which letter was entered by the user
 * ******************************************************
 */
char compare(char str[])
   {
       int i=0;
       pq=0;
       char letter;

       while(i<26)
       {
       qz=strcmp(str1[i],str);
          if(qz==0)
          {
             if(i==0)
             letter='a';
             else if(i==1)
             letter='b';
             else if(i==2)
             letter='c';
             else if(i==3)
             letter='d';
             else if(i==4)
             {
              strcpy(str,"");
              letter='e';
              }
              else if(i==5)
              letter='f';
              else if(i==6)
              letter='g';
              else if(i==7)
              letter='h';
              else if(i==8)
              letter='i';
              else if(i==9)
              letter='j';
              else if(i==10)
              letter='k';
              else if(i==11)
              letter='l';
              else if(i==12)
              letter='m';
              else if(i==13)
              letter='n';
```

```
            else if(i==14)
            letter='o';
            else if(i==15)
            letter='p';
            else if(i==16)
            letter='q';
            else if(i==17)
            letter='r';
            else if(i==18)
            letter='s';
            else if(i==19)
            letter='t';
            else if(i==20)
            letter='u';
            else if(i==21)
            letter= 'v';
            else if(i==22)
            letter='w';
            else if(i==23)
            letter='x';
            else if(i==24)
            letter='y';
            else
            letter='z';
          }
      i++;
   }
     return letter;
}

//HW interrupt for two on-board push buttons for the morse code
static void PortFIntHandler()
{

   g_sAppState.ui32Buttons = GPIOIntStatus(ButtonBase,true);

   switch(g_sAppState.ui32Buttons & ALL_BUTTONS)
   {

         case LEFT_BUTTON:
         //
         // Perform left button operation.
         //
          if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
            {
              UARTprintf("\nInterrupt occurred - DOT\n");
              xSemaphoreGive(g_pUARTSemaphore);
             }
          str[pq++]='.';  //store a dot in the string
          SysCtlDelay(9000000);
          GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
          break;

         case RIGHT_BUTTON:

         //
         // Perform the right button operation.
         //
         if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
          {
            UARTprintf("\nInterrupt occurred - DASH\n");
            xSemaphoreGive(g_pUARTSemaphore);
          }
         str[pq++]='-'; //store a dash in the string
        GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
        SysCtlDelay(9000000);
        break;
   }
         GPIOIntClear(ButtonBase,g_sAppState.ui32Buttons);
}

/*InterruptHandler for push button used for end of string in morse code*/
static void PortCIntHandler()
{
   uint32_t status=0;
   static portBASE_TYPE xHigherPriorityTaskWoken;
   status = GPIOIntStatus(ButtonBase1,true);
   if(status)
   {
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
      UARTprintf(" ENTER Interrupt Occured!!! \n");
      xSemaphoreGive(g_pUARTSemaphore);
     }
    SysCtlDelay(9000000);
    GPIOIntClear(ButtonBase1,status);
    xSemaphoreGiveFromISR(morse,&xHigherPriorityTaskWoken);
   }
}

/*Task 1 -Periodic Task- Temperature SEnsor configure with ADC*/

static void Task1(void *pvParameters)
{
   uint32_t ui32ADC0Value[4];
   volatile int task1_i,task1_j;
```

```
 volatile uint32_t ui32TempAvg, ui32TempValueC, ui32TempValueF;
 task1_i=0; task1_j=0;
 int flag=0;

 tskTCB *TCB1 = prvGetTCBFromHandle (xTask1);
 tskTCB *TCB2 = prvGetTCBFromHandle (xTask2);
 tskTCB *TCB3 = prvGetTCBFromHandle (xTask3);
 TCB1 ->waketime = 0;
 TCB1->elapsed=0;
 TCB1->remaining_time=0;


SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
//Temperature initialisation
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
ADCSequenceEnable(ADC0_BASE, 1);

while(1)
{
  // Update the LED buffer to turn off the currently working.

         g_pui32Colors[0] = 0x8000;
         g_pui32Colors[1] = 0x0000;
         g_pui32Colors[2] = 0x0000;

        // Configure the new LED settings.
         RGBColorSet(g_pui32Colors);
        xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY);
        UARTprintf("Task 1 starts at %dms\n",xTaskGetTickCount());
        xSemaphoreGive(g_pUARTSemaphore);
        g++;
        if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
            {
              UARTprintf("%d\n",g);
              xSemaphoreGive(g_pUARTSemaphore);
            }
       switch(g)
            {
                case 1: flag=0;
                       break;
                case 2: flag=1;
                                 break;
                case 3 : flag=1;
```

```
                                  break;
                  default: flag=check_prime(g);
                                  break;
              }
         if(flag)
              {
                xSemaphoreGive(sp);
                vTaskPrioritySet(xTask3, configMAX_PRIORITIES-1); //Set higher priority for Task3
               }

        flag=0;
        if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
          {
              UARTprintf("Task1 attempting to take mutex at %dms\n",xTaskGetTickCount());
              xSemaphoreGive(g_pUARTSemaphore);
          }
        if(xSemaphoreTake(share,portMAX_DELAY))
          {
              RGBEnable();
              if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
              {
                  UARTprintf("Task1 mutex taken at %dms\n",xTaskGetTickCount());
                  xSemaphoreGive(g_pUARTSemaphore);
              }
        //temperature code
         ADCIntClear(ADC0_BASE, 1);
         ADCProcessorTrigger(ADC0_BASE, 1);

        while(!ADCIntStatus(ADC0_BASE, 1, false))
        {
        }

        ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] +2)/4;
        ui32TempValueC = (((1475 - ((2475 * ui32TempAvg)) / 4096)/10)+7);
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
        if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
        {
           UARTprintf("in celsius - %d \n",ui32TempValueC);
           UARTprintf("in fahrenheit - %d \n",ui32TempValueF);
           xSemaphoreGive(g_pUARTSemaphore);
       }

       GPIOPinWrite(SevenBase,GPIO_PIN_0,0xFF); //Seven Segment Display
       GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
       GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
       GPIOPinWrite(SevenBase,GPIO_PIN_3,0xFF);
```

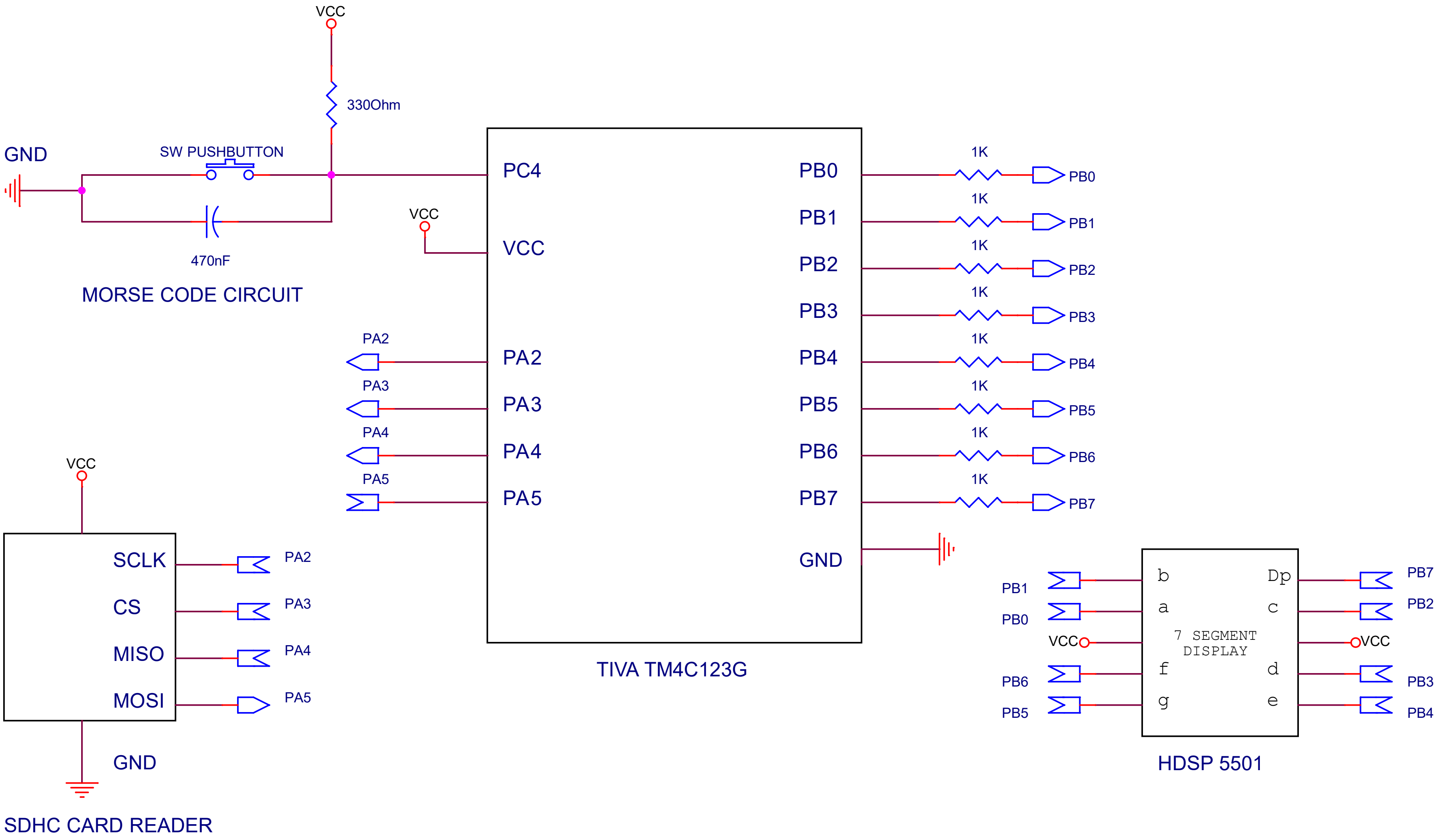
```
    GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
    GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
    GPIOPinWrite(SevenBase,GPIO_PIN_6,0xFF);
    GPIOPinWrite(SevenBase,GPIO_PIN_7,0xFF);
    /*Delay loop to get required execution time*/
    for(task1_i=outerdelay*2;task1_i>0;task1_i--)
    {
      for(task1_j=innerdelay;task1_j>0;task1_j--)
      {

      }
    }
    if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
    {
      UARTprintf("Task1 Giving mutex at %dms\n",xTaskGetTickCount());
      xSemaphoreGive(g_pUARTSemaphore);
    }
    xSemaphoreGive(share);

    if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
    {
      UARTprintf("Task 1 ends at %d ms\n\n",xTaskGetTickCount());
      xSemaphoreGive(g_pUARTSemaphore);
    }
                TCB1->elapsed= -xTaskGetTickCount();
    TCB1->remaining_time=TCB1->WCET-TCB1->elapsed;

    vTaskDelayUntil( &TCB1->waketime, (TCB1->period / portTICK_RATE_MS));
  }
 }
}

/*Task 2-Periodic Task- SD Card Configured with SPI(SSI)*/
static void Task2(void *pvParameters)
{

  volatile int i,j;

        tskTCB *TCB1 = prvGetTCBFromHandle (xTask1);
  tskTCB *TCB2 = prvGetTCBFromHandle (xTask2);
  tskTCB *TCB3 = prvGetTCBFromHandle (xTask3);
  TCB2 ->waketime = 0;
  TCB2->elapsed=0;
  TCB2->remaining_time=0;
  while(1)
  {
   // Update the LED buffer to turn off the currently working.
```

```
        g_pui32Colors[0] = 0x0000;
        g_pui32Colors[1] = 0x8000;
        g_pui32Colors[2] = 0x0000;

       // Configure the new LED settings.

       RGBColorSet(g_pui32Colors);


        if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
        {
           UARTprintf("Task 2 starts at %dms\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
        }

        if(xSemaphoreTake(share,portMAX_DELAY))
        {
           RGBEnable();
           if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
           {
               UARTprintf("Task2 mutex taken at %dms\n",xTaskGetTickCount());
               xSemaphoreGive(g_pUARTSemaphore);
           }

          GPIOPinWrite(SevenBase,GPIO_PIN_0,0x00);
          GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
          GPIOPinWrite(SevenBase,GPIO_PIN_2,0xFF);
          GPIOPinWrite(SevenBase,GPIO_PIN_3,0x00);
          GPIOPinWrite(SevenBase,GPIO_PIN_4,0x00);
          GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
          GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
          GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);

          cmd_twentyfour(); //Function call for SD Card write

          if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
          {
             UARTprintf("\n\r");
             UARTprintf("Task 2 SD Write Successful\n");
             UARTprintf("\n\r");
             xSemaphoreGive(g_pUARTSemaphore);
          }

/*Delay loop to get required execution time*/
         for(i=outerdelay*2;i>0;i--)
         {

            for(j=innerdelay;j>0;j--)
            {

            }
          }
          if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
          {
           UARTprintf("Task2 Giving mutex at %dms\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
          }

          xSemaphoreGive(share);
                    TCB2->elapsed= -xTaskGetTickCount();
          TCB2->remaining_time=TCB2->WCET-TCB2->elapsed;
          if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
          {
           UARTprintf("Task 2 ends at %d ms\n\n",xTaskGetTickCount());
           xSemaphoreGive(g_pUARTSemaphore);
          }

       }

       else
        {
           if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
           {
           UARTprintf("Task2-unable to take mutex\n");
           xSemaphoreGive(g_pUARTSemaphore);
           }
        }
        vTaskDelayUntil(&TCB2-> waketime, (TCB2->period / portTICK_RATE_MS));
        }
}
static void Task3(void *pvParameters)
{


  volatile int i,j;

            tskTCB *TCB1 = prvGetTCBFromHandle (xTask1);
  tskTCB *TCB2 = prvGetTCBFromHandle (xTask2);
  tskTCB *TCB3 = prvGetTCBFromHandle (xTask3);
  TCB3 ->waketime = 0;
  TCB3->elapsed=0;
  TCB3->remaining_time=0;
 while(1)
  {

        if(xSemaphoreTake(sp,portMAX_DELAY))
```

```
      {
      if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
       UARTprintf("Task 3(SPORADIC) starts at %dms\n",xTaskGetTickCount());
       UARTprintf("g is prime %d\n",g);
       xSemaphoreGive(g_pUARTSemaphore);
      }
      GPIOPinWrite(SevenBase,GPIO_PIN_0,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_3,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
      GPIOPinWrite(SevenBase,GPIO_PIN_5,0xFF);
      GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
      GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);

                      TCB3->elapsed= -xTaskGetTickCount();
      TCB3->remaining_time=TCB3->WCET-TCB3->elapsed;
      /*Delay loop to get required execution time*/
      for(i=outerdelay*1;i>0;i--)
      {
          for(j=innerdelay;j>0;j--)
          {
          }
       }
       if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
       {
         UARTprintf("Task 3(SPORADIC) ends at %d ms\n\n",xTaskGetTickCount());
         xSemaphoreGive(g_pUARTSemaphore);
       }
     }
  }
}

static void Task4(void *pvParameters)
{
  tskTCB *TCB1 = prvGetTCBFromHandle (xTask1);
  tskTCB *TCB2 = prvGetTCBFromHandle (xTask2);
  tskTCB *TCB3 = prvGetTCBFromHandle (xTask3);
  TCB4 ->waketime = 0;
  TCB4->elapsed=0;
  TCB4->remaining_time=0;
  while(1)
  {
    if(xSemaphoreTake(morse,portMAX_DELAY))
    {
     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
      {
       UARTprintf("Task 4(APERIODIC) starts %dms\n",xTaskGetTickCount());
       xSemaphoreGive(g_pUARTSemaphore);
      }
     TCB4->elapsed= -xTaskGetTickCount();
     TCB4->remaining_time=TCB4->WCET-TCB4->elapsed;
     morseout = compare(str); //function call to compare the morse string and return the letter it
represents

     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
          UARTprintf(" string is %c \n",morseout);
          xSemaphoreGive(g_pUARTSemaphore);
     }

     GPIOPinWrite(SevenBase,GPIO_PIN_0,0xFF); //Seven Segment Display
     GPIOPinWrite(SevenBase,GPIO_PIN_1,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_2,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_3,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_4,0xFF);
     GPIOPinWrite(SevenBase,GPIO_PIN_5,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_6,0x00);
     GPIOPinWrite(SevenBase,GPIO_PIN_7,0x00);

     if(xSemaphoreTake(g_pUARTSemaphore, portMAX_DELAY))
     {
       UARTprintf("Task 4(APERIODIC) ends at %d ms\n\n",xTaskGetTickCount());
       xSemaphoreGive(g_pUARTSemaphore);
     }
   }
}
}
//*****************************************************************************
//
// Initializes of all tasks an configurations.
//
//*****************************************************************************
uint32_t LEDTaskInit(void)
{
  //
  // Initialize the GPIOs and Timers that drive the three LEDs.
  //

  RGBInit(1);
  RGBIntensitySet(0.3f);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  // Unlock PF0 so we can change it to a GPIO input
  // Unlock PF0 so we can change it to a GPIO input
  // Once we have enabled (unlocked) the commit register then re-lock it
  // to prevent further changes.  PF0 is muxed with NMI thus a special case.
  //
  HWREG(BUTTONS_GPIO_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
  HWREG(BUTTONS_GPIO_BASE + GPIO_O_CR) |= 0x01;
  HWREG(BUTTONS_GPIO_BASE + GPIO_O_LOCK) = 0;
  //
  // Enable interrupts to the processor.
  //
  ROM_IntMasterEnable();

  //Configure direction of the GPIO Pins for seven segment display
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_0);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_1);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_2);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_3);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_4);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_5);
  GPIOPinTypeGPIOOutput(SevenBase, GPIO_PIN_6);
  //Configure direction of the GPIO Pins for push buttons used for morse code
  GPIOPinTypeGPIOInput(ButtonBase, Button1);
  GPIOPinTypeGPIOInput(ButtonBase, Button2);
  GPIOPinTypeGPIOInput(ButtonBase1, Button1);
  //Configure the current consumed by GPIO Pis
  GPIOPadConfigSet(ButtonBase ,Button1,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
  GPIOPadConfigSet(ButtonBase ,Button2,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
  GPIOPadConfigSet(ButtonBase1 ,Button1,GPIO_STRENGTH_12MA,GPIO_PIN_TYPE_STD_WPU);
  //Set the iNterrupt type for push buttons used in morsecode circuitry
  GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_4, GPIO_FALLING_EDGE);
  GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0, GPIO_FALLING_EDGE);
  GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_4, GPIO_RISING_EDGE);
  //Register th enames of the interrupt handler and enable the interrupt for GPIO Pins
  GPIOIntRegister(GPIO_PORTF_BASE,PortFIntHandler);
  GPIOIntRegister(GPIO_PORTC_BASE,PortCIntHandler);
  GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
  GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_0);
  GPIOIntEnable(GPIO_PORTC_BASE, GPIO_INT_PIN_4);

  //Create mutex and semaphore
  vSemaphoreCreateBinary(sp);
  share=xSemaphoreCreateMutex();
  vSemaphoreCreateBinary(morse);
  g_pUARTSemaphore = xSemaphoreCreateMutex();

  // Create the tasks.

  if(xTaskCreate(Task1, (signed portCHAR *)"LED1", LEDTASKSTACKSIZE, NULL,
              tskIDLE_PRIORITY + 3, &xTask1) != pdTRUE)
  {
     return(1);

  }

  // Success.
  //Set deadline and execution time for task 1
          TCB=prvGetTCBFromHandle (xTask1);
  TCB->period = 7000;
  TCB->xTask = xTask1;
  TCB->WCET = 2000;

      if(xTaskCreate(Task2, (signed portCHAR *)"LED2", LEDTASKSTACKSIZE, NULL,
              tskIDLE_PRIORITY + 2, &xTask2) != pdTRUE)
  {
     return(1);

  }

  // Success.
  //Set deadline and execution time for task 2
          TCB=prvGetTCBFromHandle (xTask2);
  TCB->period = 8000;
  TCB->xTask = xTask2;
  TCB->WCET = 2000;

if(xTaskCreate(Task3, (signed portCHAR *)"LED3", LEDTASKSTACKSIZE, NULL,
              tskIDLE_PRIORITY + 1, &xTask3) != pdTRUE)
  {
     return(1);

  }
  //Set deadline and execution time for task 3
          TCB=prvGetTCBFromHandle (xTask3);
  TCB->period = 4000;
  TCB->xTask = xTask3;
  TCB->WCET = 1000;
        if(xTaskCreate(Task4, (signed portCHAR *)"LED4", LEDTASKSTACKSIZE, NULL,
              tskIDLE_PRIORITY + 4, &xTask4) != pdTRUE)
  {
     return(1);
  }
          //Set deadline and execution time for task 4
          TCB=prvGetTCBFromHandle (xTask4);
  TCB->period = 4000;
  TCB->xTask = xTask4;
  TCB->WCET = 1000;
```

```
   // Success.
   xSemaphoreTake(morse,portMAX_DELAY);
   return(0);
}

//Function to check if the number is prime
int check_prime(int g)
{
  int c;

  for ( c = 2 ; c <= (g/2); c++ )
  {
    if ( g%c == 0 )
        {
          return 0;
        }
  }
  if ( c== g/2)
   {
      return 1;
   }
 }

*Description : Rearrange function and its calls for priority assignment in EDF algorithm for scheduling in task.c
 * Author: Sanjana Kalyanappaol
 *
 * Date: May 4 2017
 */
/*Function to rearrange the priority of the tasks*/
static int rearrangeLists(tskTCB *pxTCB)
{
   int i, flag;
   tskTCB *newTCB;
   flag=0;
   for(i=4;i>=0;i--)
   {
      if(listLIST_IS_EMPTY( &pxReadyTasksLists[i])!=pdTRUE)
      {
      newTCB=listGET_OWNER_OF_HEAD_ENTRY( &pxReadyTasksLists[i] );
      if((pxTCB->period + pxTCB->waketime) < (newTCB->period + newTCB->waketime) )
      {


         pxTCB->uxPriority=i+1;
         prvAddTaskToReadyList(pxTCB);

         flag=1;
         break;
      }
      else if((pxTCB->period + pxTCB->waketime) == (newTCB->period + newTCB->waketime) )
      {

         pxTCB->uxPriority=i;
         prvAddTaskToReadyList(pxTCB);
         flag=1;
         break;
      }
      else
      {
         while(1)
         {
            uxListRemove( &( newTCB->xGenericListItem ) );
            newTCB->uxPriority=i+1;
            prvAddTaskToReadyList(newTCB);
            if(listLIST_IS_EMPTY( &pxReadyTasksLists[i] ) != pdTRUE)
            {
                  newTCB=listGET_OWNER_OF_HEAD_ENTRY( &pxReadyTasksLists[i] );
            }
            else
            {
               break;
            }
         }
      }
      }
      if(flag==1)
      {
         taskYIELD_IF_USING_PREEMPTION();
      }
   }
   return 0;
}

/*Description : Rearrange function and its calls for priority assignment for LST algorithm in tasks.c
 * Author: Shreyas V
 *
 * Date: May 4 2017
 */
/*Function to rearrange the priority of the tasks8*/

static int rearrangeLists(tskTCB *pxTCB)
{
   int i, flag;
   tskTCB *newTCB;
   flag=0;
   for(i=3;i>=0;i--)
```

```
  {
    if(listLIST_IS_EMPTY( &pxReadyTasksLists[i])!=pdTRUE)
    {
    newTCB=listGET_OWNER_OF_HEAD_ENTRY( &pxReadyTasksLists[i] );
    if((pxTCB->period + pxTCB->waketime - pxTCB->remaining_time) < (newTCB->period + newTCB-
>waketime - newTCB->remaining_time) )
    {

      pxTCB->uxPriority=i+1;
      prvAddTaskToReadyList(pxTCB);

      flag=1;
      break;
    }
    else if((pxTCB->period + pxTCB->waketime  - pxTCB->remaining_time) == (newTCB->period +
newTCB->waketime - newTCB->remaining_time) )
    {

      pxTCB->uxPriority=i;
      prvAddTaskToReadyList(pxTCB);

      flag=1;

      break;
    }
    else
    {
      while(1)
      {
        uxListRemove( &( newTCB->xGenericListItem ) );

        newTCB->uxPriority=i+1;
        prvAddTaskToReadyList(newTCB);

        if(listLIST_IS_EMPTY( &pxReadyTasksLists[i] ) != pdTRUE)
        {
            newTCB=listGET_OWNER_OF_HEAD_ENTRY( &pxReadyTasksLists[i] );
        }
        else
        {
          break;
        }
      }

    }
    }
    if(flag==1)
    {
            taskYIELD_IF_USING_PREEMPTION();
        }
    }
    return 0;
}
```



Title
ESD FINAL PROJECT

Size	Document Number	Rev
A	Sanjana_Shreyas	<RevCode>

Date: Saturday, May 06, 2017 Sheet 1 of 1