Midterm

ENGG4420: Real-Time Systems Design

Instructor:
Dr.Radu Muresan

Group 17: Wed-8:30 Section

Bilal Ayyache: 0988616
Robert Mackenzie Beggs: 0819747

Start Date: October 14th, 2020
End Date: November 8th, 2020

Contents

List of Figures

1 Introduction

1.1 Problem Description

Effective task scheduling is arguably the foremost challenge in implementing a model in a real time environment as temporal, precedence, and resource constraints must be considered. Coupled with the actuality that proper real time system design requires a constant expression of time, from which behaviours can be constantly sampled, design becomes challenging as these qualities hold equal importance. A conceptual example of these ideals implemented can be seen in a real-time embedded smart oven system with remote control capability. Given a list of design requirements and assumptions, a smart oven system was implemented, during which the temporal, precedence and resource constraints were considered.

1.2 System Requirements

To effectively implement a model in a real time paradigm, temporal, precedence and resource constraints must be considered for a given task. To illustrate the importance of these concepts, specified diagrams of the system are presented in which the task interaction and time constraints are highlighted. The presented block diagram emphasizes and justifies design choices including the use of semaphores, mutex, queues, flags as well as interrupt services routines, while the timing diagram highlights task interaction. The system is capable of preheating to a given temperature, as well as alternating between a set of given temperatures to match a given recipe, seeking user intervention when necessary via use of an alarm. This was accomplished via use of uC/OS-III.

1.3 Constraints

The constraints associated with real time systems are outlined in the Appendix.

2 System Design Task Diagram

2.1 System Overview

Design for real time operating systems are often split into two phases; design and implementation. The system, as specified in the requirements, was first designed on paper, as pictured in Figure 5, then implemented in uC/OS-III, as depicted in Figure 1. The above system can be further delineated into two systems. System 1 describes the real time elements of the system while System 2 encapsulates the sequential elements of the system. System 1, as described in Figure 2, depicts the continuous tuning of the temperature via the dependent relationship that exists between the controller and the sensor task. The sensor task continually monitors the current temperature articulated to the controller while the controller task monitors whether the temperature communicated by the sensor matches the desired temperature. In the case that the desired temperature is equal to the current temperature measured by the sensor, the state switches from a 'running' to a 'wait' state. The initial temperature, $70°C$, and desired temperature communicated from storage. System 2, as described in Figure 3, depicts the heating process. A clear delineation is provided in which manual intervention is required whenever a heating task completes, at which time the next can commence. User intervention is marked by a button press i.e 'continue' as pictured in Figure 3 . Additionally, upon completion of each heating task, an alarm sounds. There are two categories of heating tasks; preheat tasks which represent the initial heating, and heat oven tasks which represent the continual heating during the baking process.

2.2 Task Classification

Temporal constraints are one characteristic of real time systems, that increase the complexity of the design process. Given the circumstance in which a deadline coupled with a task is missed, the downstream effects are considered. The severity of the impact, produces the associated categorization of hard, firm or soft. Hard traditionally denotes catastrophic system failure, and correspond to failure of sensor, actuating or control activities. Firm activities involve much less critical consequences, while soft tasks relate to user interaction [3]. The smart oven system implementation periodically relies on user intervention via the 'manual assistance task' and the 'continue task'; these were classified as a soft task as it involves handling input data from the user and does not have disastrous implications should it fail. Likewise, all 'GUI tasks' are considered to be soft tasks for the same reason; this includes the 'Clock Task' as well as the 'oven display task'. The 'controller task' is categorized as a hard task as it actuates the temperature control and is the component which interacts closely with the environment [3]. Failure would mean that the temperature is not altered on time or at all; associated the safety concerns, such as the oven temperature rising past safe limits, justify this classification. As stated in Section 2.1, the 'Sensor Task' interacts closely with the 'controller task'; it is responsible for data acquisition, and detection of system status. For these reasons, the 'Sensor Task' is classified as a hard task; similarly to the 'controller task' failure of the 'sensor task' would result in possible safety concerns. Though the 'alarm task' alerts the user to intervene and may result in ruined food, from a system perspective, it is not catastrophic failure and was classified as a soft task. Finally, the two heating tasks, preheat and heat were classified as firm tasks due to their association and interaction with the hard tasks; 'controller' and 'sensor'; failure to properly communicate the desired temperature to the controller long term could result in negative consequences to the health of the system, however they are not singularly critical and can be skipped for a cycle without consequence. All 'soft' tasks have the same low priority; the firm tasks are given higher, equivalent priority and the two hard tasks share the highest priority in order to ensure system health.

3 System Software Implementation

3.1 Compilation

The system compiles with no errors, and takes approximately 2 seconds to build, as seen on the CDT Build Console in Figure 6. Furthermore, the STM32 microcontroller display, shows the current oven temperature and the desired oven temperature; this can be seen in Figure 7. Finally, during runtime, the terminal displays the preheat temperature range, the value entered by the user, an alert from the alarm when preheating is complete, instructions to the user, as well as clock time. This interaction is displayed in Figure 8.

3.2 Main

The main function is used to configure and otherwise arrange the structure of the subsequent code, via the creation of objects and requisite library initialization; for example, 'HAL_init' is used to initialize the HAL library. Likewise, 'BSP' characterized functions such as 'BSP_ClkInit', 'BSP_InitInit' and 'BSP_OS_TickInit' are used to initialize the main clock, interrupt vector table and kernel tick timer respectively. Similary, 'Mem_Init', 'CPU_IntDis', 'CPU_Init', 'Math_Init' are required to load the memeory management module, configure interrupts for the CPU, initialize services related to the CPU and load the module containing functions for performing mathematics. Finally, 'OSInit' is used to initalize uc/OS-III [1]. If no errors are discovered following the initialization, 'OSQCreate' takes a pointer to the queue control block, which is predefined, and reserved in advanced as a global variable, a pointer to the string which contains the name to of a queue, an integer dictating the max size of the associated message queue, and a pointer a variable used to hold an error message, and creates a message queue [2]. This function is used to create queues for to regulate communication, plant input, set point, GUI, and operator input; in this implementation, this corresponds to queues which regulate oven temperature. Following creation of these queues, if no error is thrown, control is passed to uC/OS-III via 'OS_Start'.

3.3 Oven Preheat Function

The purpose of oven preheat is to heat the oven to a desired temperature. Given a pointer to a queue, timeout in clock ticks, an argument which specifies blocking or non blocking, the message size limit, a pointer to a timestamp, and a pointer to an error code, 'OSQPend' returns a message from the identified queue [2]. Utilizing the the functionality of 'OSQPend', the desired temperature, as well as the current temperature and the preheat flag are obtained and then are immediately converted from strings to double-precision floating-point numbers. The status of preheat flag is then examined; if it is equal to 1, the control process proceeds. In the case that it is not equal to 1, the oven has been successfully been preheated and the task finishes. Following the initial status observation of the preheat flag is complete the desired temperature is compared to the current oven temperature. If the current temperature is higher then that of the desired temperature the oven, the oven temperature is decremented by a constant, preselected value; otherwise it is incremented by that same value. Following this sequence, 'OSQPost' is given the the pointer to the queue associated with the oven temperature as well as the message, the size of the message, the config options, and a pointer to variable which holds an error message; this sends a message regarding the temperature of the oven to a task via the specified message queue [2]. This process is repeated indefinitely, with a 200 ms delay between cycles, having the effect of adjusting the temperature until it reaches the desired temperature.

3.4 Oven Heat Function

The purpose of oven heat is make sure the oven adheres to a given temperature. Through utilization of the same functions described in Section 3.3. Given a pointer to the oven temperature queue, 'OSQPend' returns messages from the designated queue [2]. This function is used to receive information about the oven temperature as well as the desired oven temperature. This information is received as strings and converted to floating point numbers. At this time, if the oven temperature is not equal to the desired temperature the control flow of the program continues to additional cases. Should the initial comparison statement evaluate to true, the oven temperature is evaluated relative to the desired temperature; if it is greater the temperature is the temperature is decremented, otherwise, it is incremented. Following these comparisons, 'OSQPost' is used to send messages to the temperature task via the oven temperature queue. This process is repeated perpetually with a 200 ms delay between evaluations, the overall byproduct of which is that the temperature is adjusted.

3.5 Oven Communication Function

The purpose of oven communication is to control the communications for the entirety of the system. At the start of the process, the state of the system is evaluated. If it is the preheat state, the user is prompted to enter a preheat temperature range, and their response is recorded via 'OSQPend'. If the value entered by the user falls within the oven's operating limits it is converted to an integer and control is passed to the control unit. In the case that it is time to change the heat, such is the case when following a recipe, the user is prompted in the same way, and the temperature as well as time are entered via the Comm Queue through use of 'OSQPend'; the same input checks are performed. Following this process, the sequence outlined in Figure 3 is employed, using a combination of 'OSQPost' to send messages to the appropriate tasks, and 'OSTimeDlyHMSM' as a method to configure relative priority in a non real time sequence [2]. As communicated in Figure 3, program status is evaluated to infer the progress of a given cycle; this is accomplished via the use of flags. Flags relating to the iteration, and function and evaluated; their status indicates the need to adjust oven temperature, adjust the oven display, communicate with the sensor, clock, controller or request manual intervention. This was also implemented through utilization of 'OSQPost' and 'OSTimeDlyHMSM'. This entire process is repeated indefinitely, as when a given recipe has completed, control flow is passed to an 'end' task.

3.6 Oven Controller Function

The oven controller function regulates the temperature of the oven and actuates change in temperature. An initial comparison is performed during which the appropriate heat task is selected (preheat or heat). At this point, the desired temperature and the current temperature are acquired via use of 'OSQPend', and the current temperature is incremented or decremented according to its relationship to the desired temperature. At this time an evaluation is done to see if the preheating stage has completed; if it has an affirmation flag is sent via 'OSQPost', else a negative flag is sent. Finally, the new oven temperature is actuated via 'OSQPost'.

3.7 Oven Display Function

The oven display function utilizes 'BSP_LCD' functions in combination with 'OSQPend' to continually update the oven display.

3.8 Oven Clock Function

The oven clock function receives a message from 'OSQPend' communicating the state of the program. Given that a recipe is active, the function receives the current clock time from 'OSQPend' and adjusts it. After a short delay achieved via 'OSTimeDlyHMSM', the updated time is posted via 'OSQPost'.

3.9 Manual Assistance

The purpose of the manual assistance task is to alert the user to intervene; this means initiating a button press task to continue. A flag carrying the state of the program is received via 'OSQPend'. A series of evaluations then occur. Depending on the status of the program indicated by the flag, an appropriate message is communicated to the user.

3.10 CONT/END

The purpose of this pair of functions is to evaluate the status of the program to determine whether it should halt the control flow or continue. This is accomplished via receiving a status flag from 'OSQPend' and then performing a comparison; if it is time to end, a message is given to the user and control flow is halted. Else, control flow proceeds.

3.11 Oven Clock Alarm

Oven clock alarm alerts the user to the fact a heating stage has completed via a message. This is accomplished via evaluating a flag received from 'OSQPend' and performing a comparison.

3.12 AppTaskCreate

Tasks are defined in 'AppTaskCreate' for use subsequently via 'OSTaskCreate'. Given a pointer to a task's 'OS_TCB', the name of a pointer, a pointer that defines the task, a pointer to a function which receives additional parameters, an integer indication task priority, a pointer to the task address, the desired stack size, the max amount of messages receivable via the message queue and a pointer to a variable that recieves error codes, 'OSTaskCreate' creates a task that is capable of being accepted by uC/OS-III's multitasking environment [2].

3.13 Button Task

The purpose of the button task is to confirm user action following the manual assistance task. Given a GPIO peripheral, and specific port bit, 'HAL_GPIO_ReadPin' returns the input port value which is evaluated against the GPIO pin set. If the two values are equal, the counter tracking button presses is incremented [1]. This is essential as it tracks the progress of the tasks listed in Section 2. The present control mode is then altered; if it is manual it switches to auto and vice versa. This cycle is repeated according to the task diagram specified in Section 2, with a delay of 200 ms delay between each cycle, as long as DEF remains true.

3.14 Comm Task

Comm task is an agnostic task used to receive messages from a given queue from queues given the appropriate input through use of 'OSQPend'; during the task, which is repeated indefinitely, the buffer is dynamically adjusted according to size that communications queue specifies.

3.15 GUI Task

If the correct circumstances are met, the display is altered via 'BSP_LCD' functions. Messages from the clock task and heat task are received from 'OSQpend', and the parameters on the LCD are adjusted considering the system status. The GUI is updated indefinitely, with a delay of 300 clock ticks between cycles.

4 Simulation Results

4.1 Timing Diagrams

Timing diagrams denote the transitions between possible states through which a task can enter. The bare tasks required for operation are running, ready and waiting. 'Running' signifies the act of executing on the processor, 'ready' signifies that a task is in the ready queue but is not able to yet be processed due to the processing of other tasks, and 'waiting' represents that the task is paused until a specific event is accomplished by another task; semaphores are typically used to coordinate signals needed to alter between 'wait' and 'ready' states [3]. As pictured in Figure 4, the communication task, concurrently executed with the 'preheat' and 'alarm' tasks, moves from a 'ready' to 'exe', running state, at the start of the preheat process, and returns to a ready state after the alarm task executes, indicating that the was oven preheated to the desired temperature; the preheat task completes as the alarm task ends. The oven display then moves from a ready state to a running state in order to show the time on the timer as well as the current temperature of the oven; it moves back to a ready state upon completion of the timer. The timing diagrams of the oven display, heat, controller and sensor tasks are identical as they work in tandem, this part of the cycle is repeated as necessary in a given recipe. As described in detail in Section 2, the heat task sends the new desired temperature to the controller task whilst the sensor task reads the current temperature; if the desired temperature has not been achieved, the cycle repeats. At the point at which the temperature has been maintained for a given amount of time, the clock task moves to a ready state, denoting that time has been reached, at which point the manual system task moves from a ready state to a ready state signifying the necessity for user intervention.

5 Conclusion

The unique challenges posed by building a system that functions within the paradigm of real time operation are typically overcome by division of the problem into two distinct phases; design and development. The design phase of project assumes liability for efficiently coping with the temporal precedence and resource constraints of a given project via categorization of task priority, as well as architecture planning through block and timing diagrams. The development phase of the project requires effective implementation of the elements articulated in the design phase, strictly adhering to the presented temporal, resource and precedence constraints. Given the problem of designing a smart oven system, functional black diagrams of the system and timing task diagrams created during the design phase were presented. The smart oven was developed such that it has the ability to perform the task of preheating in addition to preheating. Finally, a manual assistance task was created such that it responded to active commands. The challenges of implementing a real time system via uC/OS-III were highlighted. The additional challenge of incorporation of periphery devices into a real time paradigm is left for future experimentation.

References

[1] 2020. URL: `http://www.disca.upv.es/aperles/arm_cortex_m3/llibre/st/STM32F439xx_User_Manual/group__gpio__exported__functions__group2.html`.

[2] 2020. URL: `https://doc.micrium.com/display/osiiidoc`.

[3] Giorgio C Buttazzo. *Hard real-time computing systems*. 3rd ed. Springer, 2011.

Appendices

A Background

A.1 Constraints

A.1.1 Temporal

The typical temporal considerations of a given process can be summarized as the delay, deadline and duration characteristics. In order to effectively describe periodic tasks a set of assumptions are considered strictly then relaxed. The assumptions are as follows, given a set of tasks:

- Tasks that are periodic in nature are assigned a consistent period
- Execution time will be worst case for every given task
- The deadline is the same for each given task
- There are no precedence or resource constraints to be considered; each task can be considered independent [3]

For a set of tasks, there are various temporal scheduling algorithms to consider for use. A commonality between all is that optimality can be ensured by examining whether a set of tasks can be scheduled during a time in which higher priority tasks will be released; if this is possible, it can be scheduled simultaneously with all other tasks [3]. Task are classified as being hard, firm or soft according to the consequence to the system health of missing said task; this provides a natural primary step for articulating priority.

A.1.2 Precedence

Graph theory is used to depict hierarchical prerequisites and dependencies in a set of tasks. The resultant precedence graph is necessary to described subsets that can be computed in parallel, and which cannot [3].

A.1.3 Resource

The foremost concern in regards to resource constraints is maintaining mutual exclusion, such that at any given time, only a single task may access a given resource; supporting mutual exclusion guarantees a degree of data consistency. Furthermore, synchronization is used to manage access to shared resources

A.2 Objects

A.2.1 Semaphores

One object that is instrumental in ensuring mutual exclusion via synchronization are semaphores. Semaphores are capable of synchronizing interrupt service routines (ISR) to a given task. To synchronize tasks that use shared resources and encourage mutual exclusion, a subset of semaphores entitled binary semaphores are utilized. Another subset of semaphores, counting semaphores, are used to maintain a tally of how often a given resource is accessed. Vanilla semaphores, such as the ones described, suffer from priority inversion; a problem in which tasks with lower priority status supersede those with higher priority [3].

A.2.2 Queues

In real time environments, the traditional first in first out (FIFO) queue is not considered viable as priority is not taken into account. Priority queues, as the name implies, consider the priority of a given task to determine its place in a queue.

B Figures

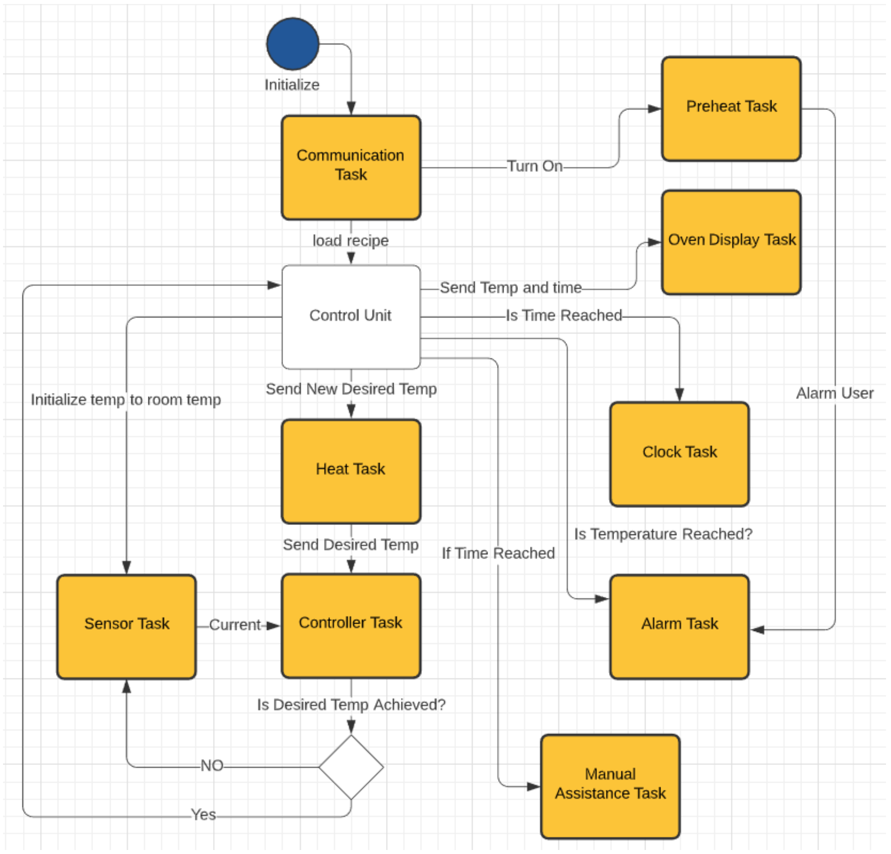


Figure 1: Block Diagram describing the control process of the system

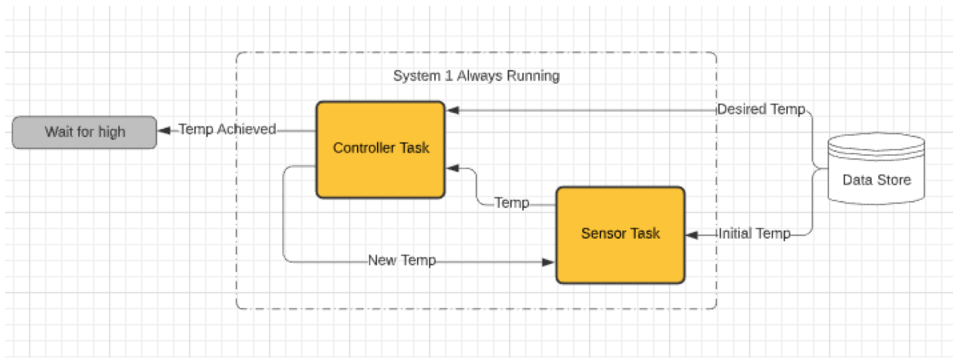


Figure 2: System 1 encapsulates the real time elements of the system

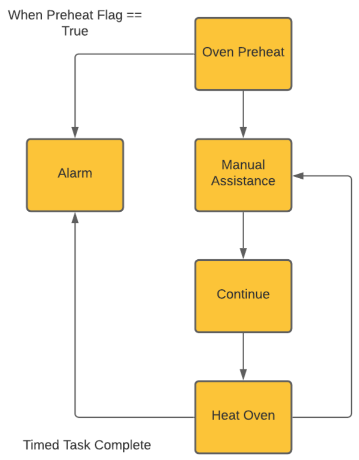


Figure 3: System 2 encapsulates the sequential elements of the system

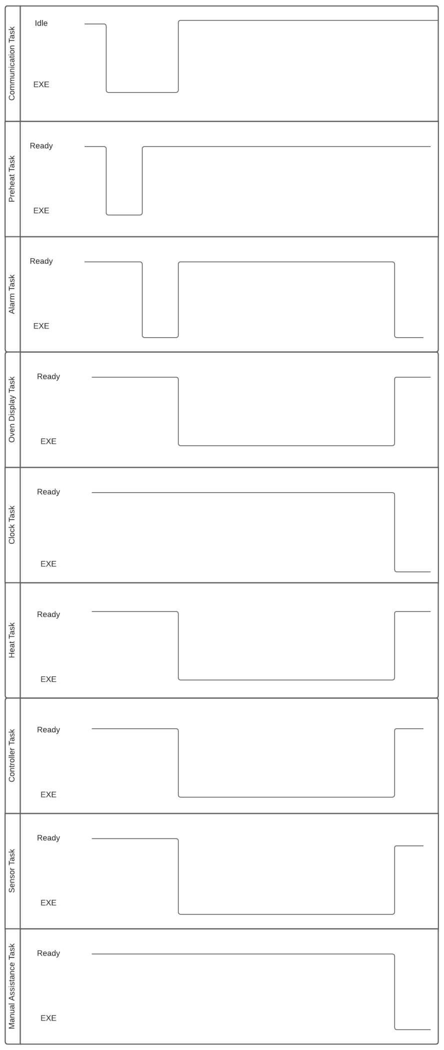


Figure 4: The timing diagram associated with system tasks. This timing diagram shows the heating process executing a single time.

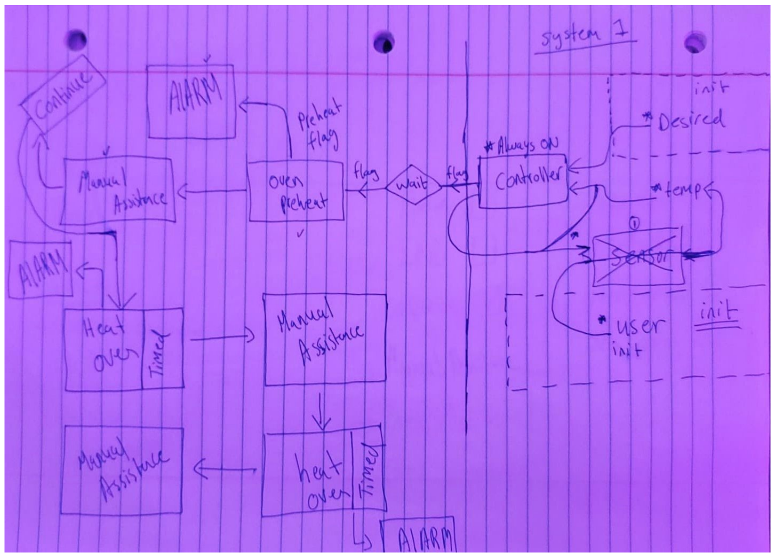
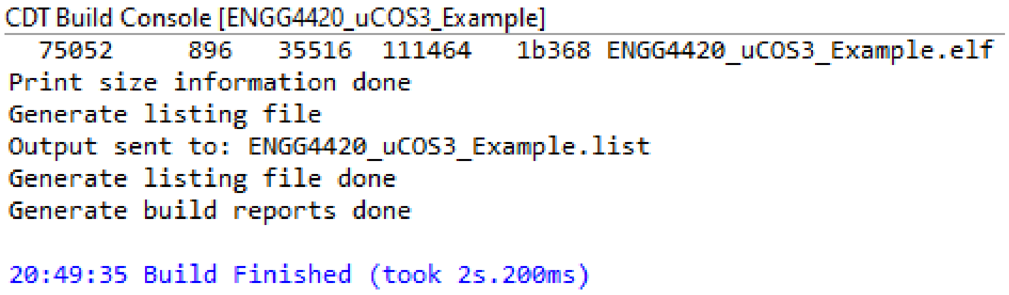


Figure 5: The first stages of the design process; hand drawn block diagrams.



CDT Build Console [ENGG4420_uCOS3_Example]

```
  75052     896   35516  111464   1b368 ENGG4420_uCOS3_Example.elf
Print size information done
Generate listing file
Output sent to: ENGG4420_uCOS3_Example.list
Generate listing file done
Generate build reports done

20:49:35 Build Finished (took 2s.200ms)
```

Figure 6: Illustrating successful compilation on the CDT Build Console.

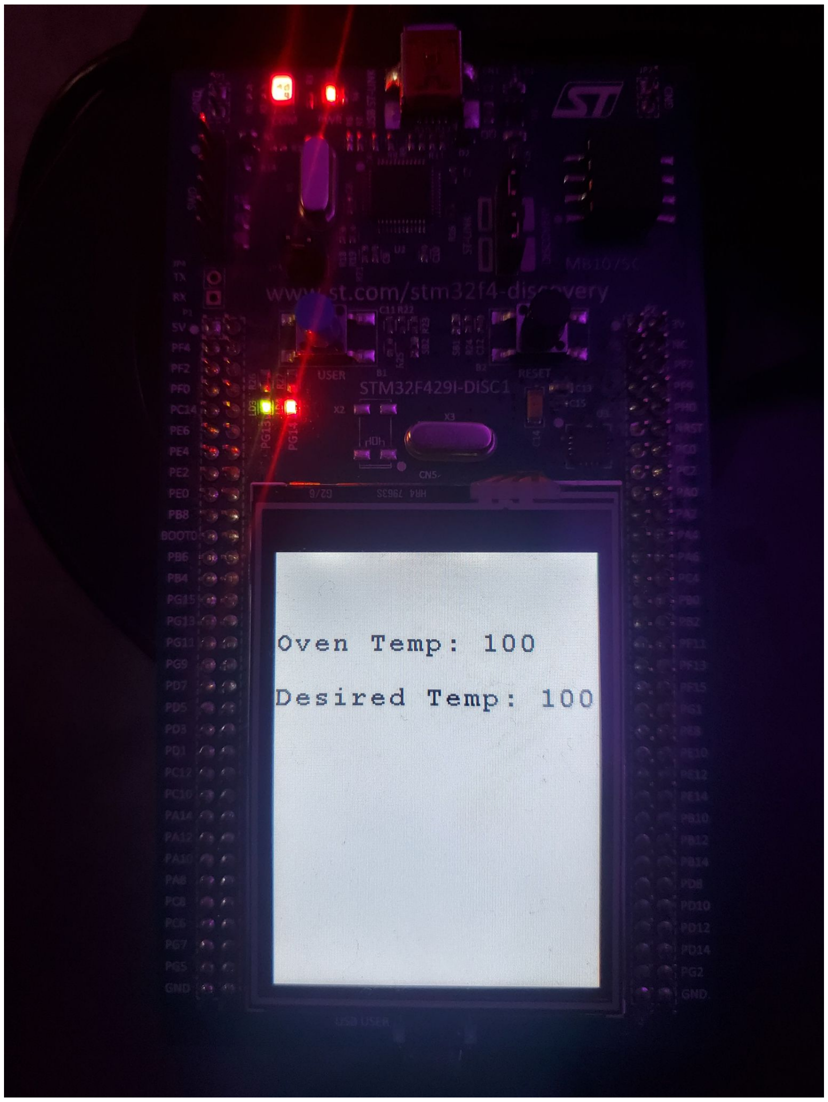
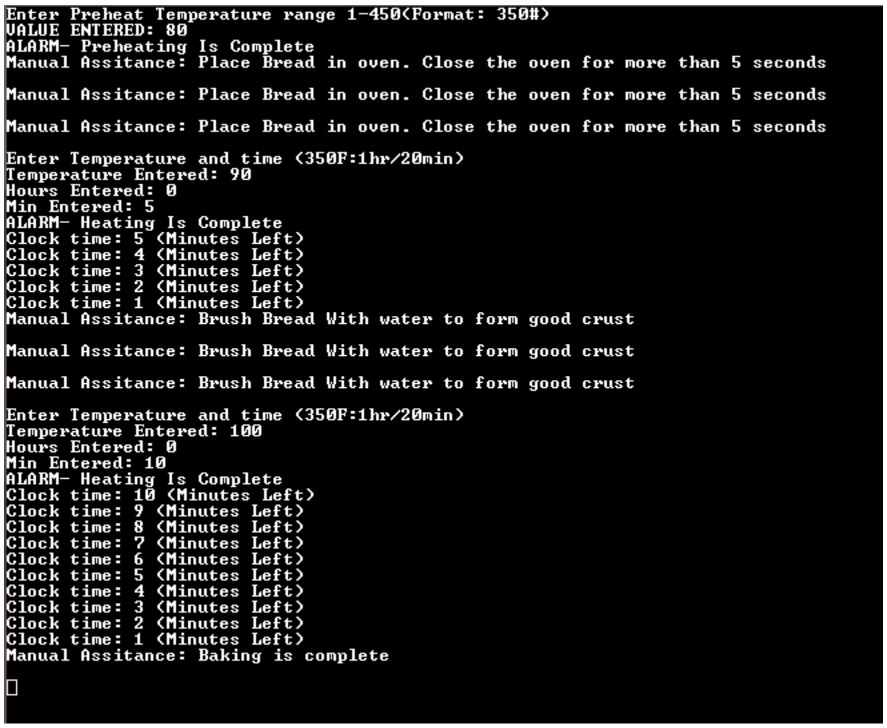


Figure 7: Displaying temperature on the STM32 microcontroller display



```
Enter Preheat Temperature range 1-450(Format: 350#)
VALUE ENTERED: 80
ALARM- Preheating Is Complete
Manual Assitance: Place Bread in oven. Close the oven for more than 5 seconds

Manual Assitance: Place Bread in oven. Close the oven for more than 5 seconds

Manual Assitance: Place Bread in oven. Close the oven for more than 5 seconds

Enter Temperature and time (350F:1hr/20min)
Temperature Entered: 90
Hours Entered: 0
Min Entered: 5
ALARM- Heating Is Complete
Clock time: 5 (Minutes Left)
Clock time: 4 (Minutes Left)
Clock time: 3 (Minutes Left)
Clock time: 2 (Minutes Left)
Clock time: 1 (Minutes Left)
Manual Assitance: Brush Bread With water to form good crust

Manual Assitance: Brush Bread With water to form good crust

Manual Assitance: Brush Bread With water to form good crust

Enter Temperature and time (350F:1hr/20min)
Temperature Entered: 100
Hours Entered: 0
Min Entered: 10
ALARM- Heating Is Complete
Clock time: 10 (Minutes Left)
Clock time: 9 (Minutes Left)
Clock time: 8 (Minutes Left)
Clock time: 7 (Minutes Left)
Clock time: 6 (Minutes Left)
Clock time: 5 (Minutes Left)
Clock time: 4 (Minutes Left)
Clock time: 3 (Minutes Left)
Clock time: 2 (Minutes Left)
Clock time: 1 (Minutes Left)
Manual Assitance: Baking is complete
```

Figure 8: Showing the output that is displayed on the terminal while the system runs

C Code

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "usb_device.h"
#include "gpio.h"

/* user Includes */
#include "includes.h"


#define APP_CFG_TASK1_PRIO 19u
#define APP_CFG_TASK2_PRIO 20u
#define APP_CFG_TASK1_STK_SIZE 256u
#define APP_CFG_TASK2_STK_SIZE 256u




/* Private variables ---------------------------------------------------------*/
uint8_t RxData[256];
uint32_t data_received = 0;
uint8_t manualAssistant[50];

uint32_t ManualAssistantCounter = 0;
uint32_t NumOfManualRequired = 3;
uint32_t OvenTempDisplay = 0;
uint32_t OvenDesiredDisplay= 0;

static  OS_TCB   StartupTaskTCB;
static  CPU_STK  StartupTaskStk[APP_CFG_STARTUP_TASK_STK_SIZE];

static  OS_TCB   CommTaskTCB;
static  CPU_STK  CommTaskStk[APP_CFG_COMM_TASK_STK_SIZE];

static  OS_TCB   BtnTaskTCB;
static  CPU_STK  BtnTaskStk[APP_CFG_BTN_TASK_STK_SIZE];

static  OS_TCB   GuiTaskTCB;
static  CPU_STK  GuiTaskStk[APP_CFG_GUI_TASK_STK_SIZE];
static OS_TCB Task1TCB;
static CPU_STK Task1Stk[APP_CFG_TASK1_STK_SIZE];
static OS_TCB Task2TCB;
static CPU_STK Task2Stk[APP_CFG_TASK2_STK_SIZE];



OS_Q   CommQ;
OS_SEM  PrintSem;



/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);

/* user private function prototypes */
static  void  AppTaskCreate         (void);
static  void  StartupTask (void  *p_arg);
static  void  CommTask (void  *p_arg);
static  void  BtnTask (void  *p_arg);
static  void  GuiTask (void  *p_arg);

static void Task1 (void *p_arg);
static void Task2 (void *p_arg);

int myprintf(const char *format, ...);
static void showText(uint16_t line, char* text);


//----------------------------------------------------------------------------//
//----------------------------------------------------------------------------//
//----------------------------------------------------------------------------//

/*MIDTERM FUNCTIONS AND CALLS*/
#define INIT_TEMP 70
#define TEMP_INC 1
#define SAMPLE_TIME 200u
//Priority
#define oven_Communication_prio 8u
#define oven_Display_prio 9u
```

```
#define oven_Sensor_prio 10u
#define oven_Clock_prio 11u
#define oven_Controller_prio 12u
#define oven_Manual_Assistance_prio 13u
#define oven_Preheat_prio 14u
#define oven_Heat_prio 15u
#define oven_Continue_prio 16u
#define oven_END_prio 17u
#define oven_Alarm_prio 18u
//Stack Size
#define oven_Communication_size 256u
#define oven_Display_size 256u
#define oven_Sensor_size 256u
#define oven_Clock_size 256u
#define oven_Controller_size 256u
#define oven_Manual_Assistance_size 256u
#define oven_Preheat_size 256u
#define oven_Heat_size 256u
#define oven_Continue_size 256u
#define oven_END_size 256u
#define oven_Alarm_size 256u
//Declaration
static  OS_TCB   oven_Communication_TCB;
static  CPU_STK  oven_Communication_STK[oven_Communication_size];

static  OS_TCB   oven_Display_TCB;
static  CPU_STK  oven_Display_STK[oven_Display_size];

static  OS_TCB   oven_Sensor_TCB;
static  CPU_STK  oven_Sensor_STK[oven_Sensor_size];

static  OS_TCB   oven_Clock_TCB;
static  CPU_STK  oven_Clock_STK[oven_Clock_size];

static  OS_TCB   oven_Controller_TCB;
static  CPU_STK  oven_Controller_STK[oven_Controller_size];

static  OS_TCB   oven_Manual_Assistance_TCB;
static  CPU_STK  oven_Manual_Assistance_STK[oven_Manual_Assistance_size];

static  OS_TCB   oven_Heat_TCB;
static  CPU_STK  oven_Heat_STK[oven_Heat_size];

static  OS_TCB   oven_Preheat_TCB;
static  CPU_STK  oven_Preheat_STK[oven_Preheat_size];

static  OS_TCB   oven_Continue_TCB;
static  CPU_STK  oven_Continue_STK[oven_Continue_size];

static  OS_TCB   oven_END_TCB;
static  CPU_STK  oven_END_STK[oven_END_size];

static  OS_TCB   oven_Alarm_TCB;
static  CPU_STK  oven_Alarm_STK[oven_Alarm_size];

static void oven_Communication(void *p_arg);
static void oven_Display(void *p_arg);
static void oven_Sensor(void *p_arg);
static void oven_Clock(void *p_arg);
static void oven_Controller(void *p_arg);
static void oven_Manual_Assistance(void *p_arg);
static void oven_Preheat(void *p_arg);
static void oven_Heat(void *p_arg);
static void oven_Continue(void *p_arg);
static void oven_END(void *p_arg);
static void oven_Alarm(void *p_arg);

void buildControlUnit(int functionNum);
//queus for control system
OS_Q oven_Communication_Q;
OS_Q oven_Display_Q;
OS_Q oven_Sensor_Q;
OS_Q oven_Clock_Q;
OS_Q oven_Controller_Q;
OS_Q oven_Manual_Assistance_Q;
OS_Q oven_Preheat_Q;
OS_Q oven_Heat_Q;
OS_Q oven_Continue_Q;
OS_Q oven_END_Q;
OS_Q oven_Alarm_Q;
//Queus for tasks
OS_Q oven_temp;            //OVEN TEMP- REALTIME
OS_Q oven_desired;          //DESIRED TEMP- REALTIME
OS_Q controller_flag;       //CONTROLLER REACHED DESIRED TEMP
OS_Q preHeat_flag;          //PREHEATING COMPLETE
OS_Q oven_alarm_preheat;    //TURN ALARM ON AFTER PREHEAT
OS_Q press_button;
OS_Q userInput;
OS_Q heat_flag;
OS_Q controller_heat_flag;
OS_Q oven_alarm_heat;

OS_Q clock_on;
OS_Q clock_flag;

OS_Q oven_Manual_Assistance2_Q;


#if APP_CFG_MATH_TASK
/* ------------ FLOATING POINT TEST TASK -------------- */
static  OS_TCB       App_TaskEq0FpTCB;
static  CPU_STK      App_TaskEq0FpStk[APP_CFG_TASK_EQ_STK_SIZE];
static  void  App_TaskEq0Fp         (void  *p_arg);             /* Floating Point Equation 0 task.
#define  APP_TASK_EQ_0_ITERATION_NBR              16u
#endif

/**
  * @brief  The application entry point.
  * @retval None
  */
int main(void)
{

  OS_ERR  os_err;

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  BSP_ClkInit();                                              /* Initialize the main clock
  BSP_IntInit();                                              /* Initialize the interrupt vector t
```

```
BSP_OS_TickInit();                                         /* Initialize kernel tick timer

Mem_Init();                                                /* Initialize Memory Managment Modul
CPU_IntDis();                                              /* Disable all Interrupts
CPU_Init();                                                /* Initialize the uC/CPU services
Math_Init();                                               /* Initialize Mathematical Module

OSInit(&os_err);                                           /* Initialize uC/OS-III
if (os_err != OS_ERR_NONE) {
    while (1);
}

App_OS_SetAllHooks();                                      /* Set all applications hooks

                                                          /* Create COMM Queue         */


OSSemCreate(&PrintSem,
            "PrintSemaphore",
            (OS_SEM_CTR)1,
            &os_err);
//--------------------------------------
//--------------------------------------
//--------------------------------------
//--------------------------------------
//midterm queues and semaphores
//Control Unit
OSQCreate(&CommQ,
          "Comm Queue",
           10,
          &os_err);

OSQCreate(&oven_Communication_Q,
          "oven_Communication_Q",
           1,
          &os_err);
OSQCreate(&oven_Display_Q,
          "oven_Display_Q",
           1,
          &os_err);
OSQCreate(&oven_Sensor_Q,
          "oven_Sensor_Q",
           1,
          &os_err);
OSQCreate(&oven_Clock_Q,
          "oven_Clock_Q",
           1,
          &os_err);
OSQCreate(&oven_Controller_Q,
          "oven_Controller_Q",
           1,
          &os_err);
OSQCreate(&oven_Manual_Assistance_Q,
          "oven_Manual_Assistance_Q",
           1,
          &os_err);
OSQCreate(&oven_Preheat_Q,
          "oven_Preheat_Q",
           1,
          &os_err);
OSQCreate(&oven_Heat_Q,
          "oven_Heat_Q",
           1,
          &os_err);
OSQCreate(&oven_Continue_Q,
          "oven_Continue_Q",
           1,
          &os_err);
OSQCreate(&oven_END_Q,
          "oven_END_Q",
           1,
          &os_err);
OSQCreate(&oven_Alarm_Q,
          "oven_Alarm_Q",
           1,
          &os_err);
//------------------------
OSQCreate(&oven_temp,
          "oven_temp",
           1,
          &os_err);

OSQCreate(&oven_desired,
          "oven_desired",
           1,
          &os_err);

OSQCreate(&controller_flag,
          "controller_flag",
           1,
          &os_err);
OSQCreate(&controller_heat_flag,
          "controller_flag",
           1,
          &os_err);

OSQCreate(&preHeat_flag,
          "preHeat_flag",
           1,
          &os_err);

OSQCreate(&oven_alarm_preheat,
          "oven_alarm_preheat",
           1,
          &os_err);

OSQCreate(&press_button,
          "press_button",
           1,
          &os_err);

OSQCreate(&userInput,
          "userInput",
           1,
          &os_err);

OSQCreate(&heat_flag,
          "heat_flag",
           1,
          &os_err);

OSQCreate(&oven_alarm_heat,
          "oven_alarm_heat",
           1,
          &os_err);

OSQCreate(&clock_on,
          "oven_alarm_heat",
           1,
          &os_err);
OSQCreate(&clock_flag,
          "oven_alarm_heat",
           1,
          &os_err);
OSQCreate(&oven_Manual_Assistance2_Q,
          "oven_Manual_Assistance2_Q",
           1,
          &os_err);



//--------------------------------------
//--------------------------------------
//--------------------------------------
//--------------------------------------
OSTaskCreate(&StartupTaskTCB,                            /* Create the startup task
             "Startup Task",
              StartupTask,
              0u,
              APP_CFG_STARTUP_TASK_PRIO,
             &StartupTaskStk[0u],
              StartupTaskStk[APP_CFG_STARTUP_TASK_STK_SIZE / 10u],
              APP_CFG_STARTUP_TASK_STK_SIZE,
              0u,
              0u,
              0u,
             (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
             &os_err);
if (os_err != OS_ERR_NONE) {
    while (1);
}

OSStart(&os_err);                                        /* Start multitasking (i.e. give con

while (DEF_ON) {}                                         /* Should Never Get Here.
```

```
}


static void oven_Communication (void *p_arg)
{
  (void)p_arg;
  OS_ERR os_err;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  int firstRun = 1; //initialization flag

  int keepControllerOn = 5; //VALUE TO KEEP ON
  int keepAlarmOn = 11; //VALUE TO KEEP ON
  int continueOn = 9; //VALUE TO KEEP ON

  int functionNum = 1;  //inputted by user
  int desiredTemp = 80; //inputted by user


  int roomTemperature = 70; //InitialTemperature

  void *ovenTempMessage;
  int oven_temp_value = 0;  //Oven Temperature

  void *commPreheatMessage;
  int commPreheat = 2;

  void *commheatMessage;
  int commheat = 2;

  void *commClockMessage;
  int commClock = 0;

  void *commAssistant2Message;
  int commAssistant2 = 0;

  void *buttonPressedMessage;
  int buttonPressedValue = 0;
```

```
//Userinput Variables
char wordSaved[2] = "";

void *p_msg;
char manualAssistant[20] = "";
int manualAssistantReady = 1;

int changePreheatTemp = 1;
int changeheatTemp = 0;

int clockHours = 0;
int clockmin = 0;
int clockTime = 0;




while(1){
  takeInput:

  if(functionNum == 1){
    if(changePreheatTemp){
      changePreheatTemp = 0;
      while (DEF_TRUE){
        printf("Enter Preheat Temperature range 1-450(Format: 350#)\n\r");
        while(manualAssistantReady){
          p_msg = OSQPend(&CommQ,
                        0,
                        OS_OPT_PEND_BLOCKING,
                        &msg_size,
                        &ts,
                        &os_err);
          strcpy(wordSaved,(char *)p_msg);

          if(strcmp(wordSaved,"#") == 0){ manualAssistantReady = 0; }
          else{
            if(msg_size > 0){
              strcat(manualAssistant,(char *)p_msg);
              msg_size = 0;
            }
          }
        }
        printf("VALUE ENTERED: %d\n\r", atoi(manualAssistant));
        if(atoi(manualAssistant) > 0 && atoi(manualAssistant) <= 450){
          desiredTemp = atoi(manualAssistant);
          functionNum = 7;
           goto controlUnit;
        }
      }
    }else if(changeheatTemp){
      printf("Enter Temperature and time (350F:1hr/20min)\n\r");
      while (DEF_TRUE){
                  strcpy(manualAssistant,"");
                  manualAssistantReady = 1;
                  while(manualAssistantReady){
                    p_msg = OSQPend(&CommQ,
                                0,
                                OS_OPT_PEND_BLOCKING,
                                &msg_size,
                                &ts,
                                &os_err);
                    strcpy(wordSaved,(char *)p_msg);

                    if(strcmp(wordSaved,":") == 0){ manualAssistantReady = 0; }
                    else{
                      if(msg_size > 0){
                        strcat(manualAssistant,(char *)p_msg);
                        msg_size = 0;
                      }
                    }
                  }
                  //Sentence Created Here
                  printf("Temperature Entered: %d\n\r", atoi(manualAssistant));
                  if(atoi(manualAssistant) > 0 && atoi(manualAssistant) <= 450){
                    desiredTemp = atoi(manualAssistant);
                  }
                  //------------------------------------------------------------
                  //------------------------------------------------------------
                  //------------------------------------------------------------
                  //------------------------------------------------------------
                  strcpy(manualAssistant,"");
                  manualAssistantReady = 1;
```

```
while(manualAssistantReady){
  p_msg = OSQPend(&CommQ,
              0,
              OS_OPT_PEND_BLOCKING,
              &msg_size,
              &ts,
              &os_err);
  strcpy(wordSaved,(char *)p_msg);

  if(strcmp(wordSaved,"/") == 0){ manualAssistantReady = 0; }
  else{
    if(msg_size > 0){
      strcat(manualAssistant,(char *)p_msg);
      msg_size = 0;
    }
  }
}
//Sentence Created Here
printf("Hours Entered: %d\n\r", atoi(manualAssistant));
if(atoi(manualAssistant) > 0 && atoi(manualAssistant) <= 450){
  clockHours = atoi(manualAssistant);
}
//------------------------------------------------------------
//------------------------------------------------------------
//------------------------------------------------------------
//------------------------------------------------------------
strcpy(manualAssistant,"");
manualAssistantReady = 1;
while(manualAssistantReady){
  p_msg = OSQPend(&CommQ,
              0,
              OS_OPT_PEND_BLOCKING,
              &msg_size,
              &ts,
              &os_err);
  strcpy(wordSaved,(char *)p_msg);

  if(strcmp(wordSaved,"#") == 0){ manualAssistantReady = 0; }
  else{
    if(msg_size > 0){
      strcat(manualAssistant,(char *)p_msg);
```

```
                                msg_size = 0;
                              }
                            }
                          }
                          //Sentence Created Here
                          printf("Min Entered: %d\n\r", atoi(manualAssistant));
                          if(atoi(manualAssistant) > 0 && atoi(manualAssistant) <= 450){
                            clockmin = atoi(manualAssistant);
                            changeheatTemp = 0;
                            functionNum = 8;
                            goto controlUnit;
                          }
                          //------------------------------------------------------------
                          //------------------------------------------------------------
                          //------------------------------------------------------------
                          //------------------------------------------------------------
              }
            }
          }

          while(1){
            OvenDesiredDisplay = desiredTemp;
            controlUnit:
            //ALWAYS RUNNING TASKS
            //KEEP CONTROLLER ON
            OSQPost(&oven_Controller_Q,
                                                  (int *) &keepControllerOn,
                                                  sizeof((void *)&keepControllerOn),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
            OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                                  &os_err);

            // KEEP ALARM READY
            OSQPost(&oven_Alarm_Q,
                                                  (int *) &keepAlarmOn,
                                                  sizeof((void *)&keepAlarmOn),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
            OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                                  &os_err);
  //KEEP SENDING DESIRED VALUE
  OSQPost(&oven_desired,
                                                  (int *) &desiredTemp,
                                                  sizeof((void *)&desiredTemp),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
  OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                                  &os_err);
  //ALWAYS CHECK INPUT ON CMD IF CONTINUE
  OSQPost(&oven_Continue_Q,
                                                  (int *) &continueOn,
                                                  sizeof((void *)&continueOn),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
  OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                                  &os_err);

  if(firstRun == 1){
    firstRun = 0;
    OSQPost(&oven_temp,
                                                  (int *) &roomTemperature,
                                                  sizeof((void *)&roomTemperature),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                                  &os_err);
  }
  //----------------------------------------------
  //----------------------------------------------
  //OVEN DISPLAY
  if(functionNum == 2){
    OSQPost(&oven_Display_Q,
                                                  (int *) &functionNum,
                                                  sizeof((void *)&functionNum),
                                                  OS_OPT_POST_FIFO,
                                                  &os_err);
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                  OS_OPT_TIME_HMSM_STRICT,
                                              &os_err);
      //----------------------------------------------
      //----------------------------------------------
      //OVEN SENSOR
      }else if(functionNum == 3){
        //DO NOT USE THIS FUNCTION
        OSQPost(&oven_Sensor_Q,
                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                              OS_OPT_TIME_HMSM_STRICT,
                                              &os_err);
      //----------------------------------------------
      //----------------------------------------------
      //OVEN CLOCK
      }else if(functionNum == 4){
        OSQPost(&oven_Clock_Q,
              (int *) &functionNum,
              sizeof((void *)&functionNum),
              OS_OPT_POST_FIFO,
              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);

        clockTime = (clockHours*60)+clockmin;

        OSQPost(&clock_on,
                                              (int *) &clockTime,
                                              sizeof((void *)&clockTime),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);

        commClockMessage = OSQPend(&clock_flag,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
```

```
                      &msg_size,
                      &ts,
                      &os_err);
    commClock = *((int *)commClockMessage);

    if(commClock == 1){
      functionNum = 6;
    }
  //-----------------------------------------------
  //-----------------------------------------------
  //OVEN CONTROLLER
  }else if(functionNum == 5){
    //DONT GO IN HERE
    OSQPost(&oven_Controller_Q,
          (int *) &functionNum,
          sizeof((void *)&functionNum),
          OS_OPT_POST_FIFO,
          &os_err);
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
          OS_OPT_TIME_HMSM_STRICT,
          &os_err);
  //-----------------------------------------------
  //-----------------------------------------------
  //MANUAL ASSISTANCE
  }else if(functionNum == 6){


    OSQPost(&oven_Manual_Assistance_Q,
          (int *) &functionNum,
          sizeof((void *)&functionNum),
          OS_OPT_POST_FIFO,
          &os_err);

    OSTimeDlyHMSM(0u, 0u, 5u, 0,
          OS_OPT_TIME_HMSM_STRICT,
          &os_err);

    buttonPressedMessage = OSQPend(&press_button,
                0,
                OS_OPT_PEND_NON_BLOCKING, //  wait for queue
                &msg_size,
                &ts,
                &os_err);

    buttonPressedValue = *((int *)buttonPressedMessage);

    if(ManualAssistantCounter == NumOfManualRequired-1){
        printf("Oven ShutDown\n\r");
        desiredTemp = 0;
        functionNum = 12;
      }


    //TAKE NEW INPUT
    if(buttonPressedValue == 1){
        changeheatTemp = 1;
        functionNum = 1;
        ++ManualAssistantCounter;
        goto takeInput;
      // functionNum = 12;
    }
  //----------------------------------------------
  //----------------------------------------------
  //OVEN PREHEAT
  }else if(functionNum == 7){
    OSQPost(&oven_Preheat_Q,
          (int *) &functionNum,
          sizeof((void *)&functionNum),
          OS_OPT_POST_FIFO,
          &os_err);
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
          OS_OPT_TIME_HMSM_STRICT,
          &os_err);

    commPreheatMessage = OSQPend(&preHeat_flag,
                0,
                OS_OPT_PEND_NON_BLOCKING, //  wait for queue
                &msg_size,
                &ts,
                &os_err);

    commPreheat = *((int *)commPreheatMessage);
    if(commPreheat == 1){
      OSQPost(&oven_alarm_preheat,
          (int *) &commPreheat,
          sizeof((void *)&commPreheat),
          OS_OPT_POST_FIFO,
          &os_err);
      functionNum = 6;
    }
  //----------------------------------------------
  //----------------------------------------------
  //OVEN_HEAT
  }else if(functionNum == 8){
    OSQPost(&oven_Heat_Q,
          (int *) &functionNum,
          sizeof((void *)&functionNum),
          OS_OPT_POST_FIFO,
          &os_err);
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
          OS_OPT_TIME_HMSM_STRICT,
          &os_err);

    commheatMessage = OSQPend(&heat_flag,
                0,
                OS_OPT_PEND_NON_BLOCKING, //  wait for queue
                &msg_size,
                &ts,
                &os_err);

    commheat = *((int *)commheatMessage);

    if(commheat == 1){
      OSQPost(&oven_alarm_heat,
          (int *) &commheat,
          sizeof((void *)&commheat),
          OS_OPT_POST_FIFO,
          &os_err);
      functionNum = 4;
    }

  //----------------------------------------------
```

```
      //---------------------------------------------
      //OVEN_CONTINUE
      }else if(functionNum == 9){
        OSQPost(&oven_Continue_Q,
              (int *) &functionNum,
              sizeof((void *)&functionNum),
              OS_OPT_POST_FIFO,
              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);
      //---------------------------------------------
      //---------------------------------------------
      //OVEN END
      }else if(functionNum == 10){
        OSQPost(&oven_END_Q,
              (int *) &functionNum,
              sizeof((void *)&functionNum),
              OS_OPT_POST_FIFO,
              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);
      //---------------------------------------------
      //---------------------------------------------
      //OVEN ALARM
      }else if(functionNum == 11){
        //DO NOT USE- SHOULD BE ALWAYS ON
        OSQPost(&oven_Alarm_Q,
              (int *) &functionNum,
              sizeof((void *)&functionNum),
              OS_OPT_POST_FIFO,
              &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);
      }else if(functionNum == 12){

      }
    }
  }
```

```
}

//Controller Complete
static void oven_Controller (void *p_arg)
{

  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 5;
  int taskSwitch = 0;

  void *ovenTempMessage;
  void *desiredTempMessage;

  int desired_temp = 0;
  int oven_temp_value = 0;
  int controllerFlag = 0;

  int equalFlag = 0;

  int preheatStage = 1;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Controller_Q,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){
      // printf("controller On\n\r");
      desiredTempMessage = OSQPend(&oven_desired,
                             0,
                             OS_OPT_PEND_NON_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
      ovenTempMessage = OSQPend(&oven_temp,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);

      desired_temp = *((int *)desiredTempMessage);
      oven_temp_value = *((int *)ovenTempMessage);

      if(oven_temp_value > desired_temp){
        equalFlag = 0;
        oven_temp_value = oven_temp_value - TEMP_INC;
      }else if(oven_temp_value < desired_temp){
        equalFlag = 0;
        oven_temp_value = oven_temp_value + TEMP_INC;
      }

      if(oven_temp_value == desired_temp && equalFlag == 0){
        controllerFlag = 1;
        equalFlag = 1;

        if(preheatStage == 1){
          preheatStage = 0;
          //printf("Preheat Complete\n\r");
          OSQPost(&controller_flag,
                                                (int *) &controllerFlag,
                                                sizeof((void *)&controllerFlag),
                                                OS_OPT_POST_FIFO,
                                                &os_err);
        }else{
          OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                OS_OPT_TIME_HMSM_STRICT,
                                                &os_err);
          OSQPost(&controller_heat_flag,
                                                (int *) &controllerFlag,
                                                sizeof((void *)&controllerFlag),
                                                OS_OPT_POST_FIFO,
                                                &os_err);
        }
```

```
      }

      OSQPost(&oven_temp,
                                                (int *) &oven_temp_value,
                                                sizeof((void *)&oven_temp_value),
                                                OS_OPT_POST_FIFO,
                                                &os_err);

      OvenTempDisplay = oven_temp_value;
    }else{
      //FUNCTION NOT RUNNING
    }

  }

}

static void oven_Preheat(void *p_arg)
{
  (void)p_arg;
  OS_ERR os_err;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 7;
  int taskSwitch = 0;

  void *controllerFlagMessage;
  int controllerFlag = 0;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Preheat_Q,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
    taskSwitch = *((int *)taskSwitchMessage);

    if(taskSwitch == taskNumber){
      //FUNCTION RUNNING
      controllerFlagMessage = OSQPend(&controller_flag,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
      controllerFlag = *((int *)controllerFlagMessage);

      if(controllerFlag == 1){
        OSQPost(&preHeat_flag,
                                        (int *) &controllerFlag,
                                        sizeof((void *)&controllerFlag),
                                        OS_OPT_POST_FIFO,
                                        &os_err);
      }
    }else{
      //FUNCTION NOT RUNNING
    }
    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                        OS_OPT_TIME_HMSM_STRICT,
                                        &os_err);
  }
}

static void oven_Heat (void *p_arg)
{
  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;

  void *controllerFlagMessage;
  int controllerFlag = 0;

  int taskNumber = 8;
  int taskSwitch = 0;

    while(1){
```

```
    taskSwitchMessage = OSQPend(&oven_Heat_Q,
                                 0,
                                 OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
    taskSwitch = *((int *)taskSwitchMessage);

    if(taskSwitch == taskNumber){
      //FUNCTION RUNNING
      controllerFlagMessage = OSQPend(&controller_heat_flag,
                                 0,
                                 OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
      controllerFlag = *((int *)controllerFlagMessage);

      if(controllerFlag == 1){
        OSQPost(&heat_flag,
                                                    (int *) &controllerFlag,
                                                    sizeof((void *)&controllerFlag),
                                                    OS_OPT_POST_FIFO,
                                                    &os_err);


      }




    }else{
      //FUNCTION NOT RUNNING
    }

    OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
                                                    OS_OPT_TIME_HMSM_STRICT,
                                                    &os_err);
  }
}
```

```
static void oven_Display (void *p_arg)
{
  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_I2C3_Init();
  MX_LTDC_Init();
  MX_SPI5_Init();

  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LCD_BACKGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_LayerDefaultInit(LCD_FOREGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_SelectLayer(LCD_FOREGROUND_LAYER);
  BSP_LCD_DisplayOn();
  BSP_LCD_Clear(LCD_COLOR_WHITE);

  void *taskSwitchMessage;
  int taskNumber = 2;
  int taskSwitch = 0;
  while(1){
    taskSwitchMessage = OSQPend(&oven_Display_Q,
                            0,
                            OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                           &msg_size,
                           &ts,
                           &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){

      BSP_LCD_SetTextColor(LCD_COLOR_BLUE);
            BSP_LCD_DisplayStringAtLine(1, (uint8_t *)"   Oven Temperature: ");
            BSP_LCD_SetTextColor(LCD_COLOR_GREEN);
            BSP_LCD_DisplayStringAtLine(2, (uint8_t *)"    ENGG4420");
            BSP_LCD_SetTextColor(LCD_COLOR_GREEN);
            BSP_LCD_DisplayStringAtLine(3, (uint8_t *)"   Oven");
            BSP_LCD_SetTextColor(LCD_COLOR_RED);
            BSP_LCD_DisplayStringAtLine(5, (uint8_t *)"   (uC/OS-III) ");
      OSTimeDlyHMSM(0u, 0u, 0u, 10u,
                                          OS_OPT_TIME_HMSM_STRICT,
                                          &os_err);

    }else{
      //FUNCTION NOT RUNNING
    }
  }
}

static void oven_Clock (void *p_arg)
{

  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 4;
  int taskSwitch = 0;

  void *clockTimeMessage;
  int clockTime = 0;

  int clockDone = 1;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Clock_Q,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){
      clockTimeMessage = OSQPend(&clock_on,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
      clockTime = *((int *)clockTimeMessage);

      while(clockTime != 0){
        printf("Clock time: %d (Minutes Left)\n\r", clockTime);
        clockTime = clockTime - 1;

        OSTimeDlyHMSM(0u, 0u, 1u, 0u,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);
      }

      OSQPost(&clock_flag,
                                                    (int *) &clockDone,
                                                    sizeof((void *)&clockDone),
                                                    OS_OPT_POST_FIFO,
                                                    &os_err);
        OSTimeDlyHMSM(0u, 0u, 0u, SAMPLE_TIME,
              OS_OPT_TIME_HMSM_STRICT,
              &os_err);
    }else{
      //FUNCTION NOT RUNNING
    }
  }

}

static void oven_Manual_Assistance (void *p_arg)
{

  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  void *taskSwitch2Message;
  int i = 0;

  int taskNumber = 6;
  int taskSwitch = 0;
  int taskSwitch2 = 0;


  int firstRun = 0;
  int onTrigger = 1;

  char Sentence1[70] = "Place Bread in oven. Close the oven for more than 5 seconds\n\r" ;
  char Sentence2[50] = "Brush Bread With water to form good crust\n\r" ;
  char Sentence3[50] = "Baking is complete\n\r" ;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Manual_Assistance_Q,
                                 0,
                                 OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
    taskSwitch = *((int *)taskSwitchMessage);



    if(taskSwitch == taskNumber){

      if(ManualAssistantCounter == 0){
        printf("Manual Assitance: %s\n\r", Sentence1);
      }else if(ManualAssistantCounter == 1){
        printf("Manual Assitance: %s\n\r", Sentence2);
      }else if(ManualAssistantCounter == 2){
        printf("Manual Assitance: %s\n\r", Sentence3);
        //TURN OFF OVEN
      }

      //wait for continue print every 5 seconds
    }else{
      //FUNCTION NOT RUNNING
    }
  }

}
```

```
static void oven_Continue (void *p_arg)
{

  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 9;
  int taskSwitch = 0;

  int buttonValue = 1;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Continue_Q,
                                 0,
                                 OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){
      if(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN) == GPIO_PIN_SET){
                OSQPost(&press_button,
                                            (int *) &buttonValue,
                                            sizeof((void *)&buttonValue),
                                            OS_OPT_POST_FIFO,
                                            &os_err);
              while(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN)==GPIO_PIN_SET);
    }

    }else{
      //FUNCTION NOT RUNNING
    }
  }
}

static void oven_END (void *p_arg)
{
  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 10;
  int taskSwitch = 0;
  while(1){
    taskSwitchMessage = OSQPend(&oven_END_Q,
                               0,
                               OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                              &msg_size,
                              &ts,
                              &os_err);
    taskSwitch = *((int *)taskSwitchMessage);

    if(taskSwitch == taskNumber){
      printf("End function Running\n\r");
    }else{
      //FUNCTION NOT RUNNING
    }
  }

}

static void oven_Alarm (void *p_arg)
{
  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 11;
  int taskSwitch = 0;

  void *preheatDoneMessage;
  int preheatDone = 0;

  void *heatDoneMessage;
  int heatDone = 0;
  while(1){
    taskSwitchMessage = OSQPend(&oven_Alarm_Q,
                                 0,
                                 OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){
      preheatDoneMessage = OSQPend(&oven_alarm_preheat,
                                 0,
                                 OS_OPT_PEND_NON_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
      preheatDone = *((int *)preheatDoneMessage);

      heatDoneMessage = OSQPend(&oven_alarm_heat,
                                 0,
                                 OS_OPT_PEND_NON_BLOCKING, // doesn't wait for queue
                                &msg_size,
                                &ts,
                                &os_err);
      heatDone = *((int *)heatDoneMessage);

      if(preheatDone == 1){
        printf("ALARM- Preheating Is Complete\n\r");
      }
      if(heatDone == 1){
        printf("ALARM- Heating Is Complete\n\r");
      }

    }else{
      //FUNCTION NOT RUNNING
    }
  }
}
/*-----------------------------------------------------------------------------------------------------------
//HELPER FUNCTIONS
void buildControlUnit(int functionNum){
```

```
OS_ERR os_err;
OS_MSG_SIZE  msg_size;
      CPU_TS       ts;

OSQPost(&oven_Display_Q,
                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Sensor_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Clock_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Controller_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Manual_Assistance_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Preheat_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);
OSQPost(&oven_Heat_Q,

                                              (int *) &functionNum,
                                              sizeof((void *)&functionNum),
                                              OS_OPT_POST_FIFO,
                                              &os_err);

OSQPost(&oven_Continue_Q,

                                              (int *) &functionNum,
                                               sizeof((void *)&functionNum),
                                               OS_OPT_POST_FIFO,
                                               &os_err);
  OSQPost(&oven_END_Q,
                                               (int *) &functionNum,
                                               sizeof((void *)&functionNum),
                                               OS_OPT_POST_FIFO,
                                               &os_err);
  OSQPost(&oven_Alarm_Q,
                                               (int *) &functionNum,
                                               sizeof((void *)&functionNum),
                                               OS_OPT_POST_FIFO,
                                               &os_err);
}
static void oven_Sensor (void *p_arg)
{
  OS_ERR os_err;
  (void)p_arg;
  OS_MSG_SIZE  msg_size;
        CPU_TS       ts;

  void *taskSwitchMessage;
  int taskNumber = 3;
  int taskSwitch = 0;

  void *ovenTempMessage;
  int oven_temp_value = 0;

  while(1){
    taskSwitchMessage = OSQPend(&oven_Sensor_Q,
                             0,
                             OS_OPT_PEND_BLOCKING, // doesn't wait for queue
                            &msg_size,
                            &ts,
                            &os_err);
    taskSwitch = *((int *)taskSwitchMessage);
    if(taskSwitch == taskNumber){
      //NO NEED TO DO ANYTHING HERE
    }else{
      //FUNCTION NOT RUNNING
    }
  }
}



/*********************************************************************************************
*                                            STARTUP TASK
*
* Description : This is an example of a startup task.  As mentioned in the book's text, you MUST
*               initialize the ticker only once multitasking has started.
* Arguments   : p_arg   is the argument passed to 'StartupTask()' by 'OSTaskCreate()'.
* Returns     : none
* Notes       : 1) The first line of code is used to prevent a compiler warning because 'p_arg' is
*                  used.  The compiler should not generate any code for this statement.
*********************************************************************************************
*/

static  void  StartupTask (void *p_arg)
{
  OS_ERR  os_err;
  (void)p_arg;

  OS_TRACE_INIT();                                        /* Initialize the uC/OS-III Trace re

  BSP_OS_TickEnable();                                    /* Enable the tick timer and interru
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USB_DEVICE_Init();

  BSP_LED_Init();

#if OS_CFG_STAT_TASK_EN > 0u
  OSStatTaskCPUUsageInit(&os_err);                        /* Compute CPU capacity with no task
#endif

#ifdef CPU_CFG_INT_DIS_MEAS_EN
  CPU_IntDisMeasMaxCurReset();
#endif

  AppTaskCreate();                                        /* Create Application tasks
  while (DEF_TRUE) {                                          /* Task body, always written as an i
      BSP_LED_Toggle(0);
      OSTimeDlyHMSM(0u, 0u, 1u, 0u,
                    OS_OPT_TIME_HMSM_STRICT,
                    &os_err);

  }
}

/*********************************************************************************************
*                                          AppTaskCreate()
*
* Description : Create application tasks.
* Argument(s) : none
* Return(s)   : none
* Caller(s)   : AppTaskStart()
* Note(s)     : none.
*********************************************************************************************
*/

static  void  AppTaskCreate (void)
{
    OS_ERR  os_err;
    //----------------------------------------------------------------------------
    //----------------------------------------------------------------------------
    //MIDTERM TASK CREATION
    OSTaskCreate(&GuiTaskTCB,                                /* Create the comm task
                 "GUI Task",
                  GuiTask,
                  0u,
                  APP_CFG_GUI_TASK_PRIO,
                 &GuiTaskStk[0u],
                  GuiTaskStk[APP_CFG_GUI_TASK_STK_SIZE / 10u],
                  APP_CFG_GUI_TASK_STK_SIZE,
                  0u,
                  0u,
                  0u,
                 (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                 &os_err);
    if (os_err != OS_ERR_NONE) {
        while (1);
    }

    OSTaskCreate(&oven_Communication_TCB, /* Create the task1 */
                 "oven_Communication",
                 oven_Communication,
                 0u,
                 oven_Communication_prio,
                 &oven_Communication_STK[0u],
                 oven_Communication_STK[oven_Communication_size / 10u],
                 oven_Communication_size,
                       0u,
                       0u,
                       0u,
                       (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                       &os_err);

    OSTaskCreate(&oven_Display_TCB, /* Create the task1 */
                 "oven_Display",
                 oven_Display,
                 0u,
                 oven_Display_prio,
                 &oven_Display_STK[0u],
                 oven_Display_STK[oven_Display_size / 10u],
                 oven_Display_size,
                       0u,
                       0u,
                       0u,
                       (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                       &os_err);

    OSTaskCreate(&oven_Sensor_TCB, /* Create the task1 */
                 "oven_Sensor",
                 oven_Sensor,
                 0u,
                 oven_Sensor_prio,
                 &oven_Sensor_STK[0u],
                 oven_Sensor_STK[oven_Sensor_size / 10u],
                 oven_Sensor_size,
                       0u,
                       0u,
                       0u,
                    (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                    &os_err);

    OSTaskCreate(&oven_Clock_TCB, /* Create the task1 */
                "oven_Clock",
                oven_Clock,
                0u,
                oven_Clock_prio,
                &oven_Clock_STK[0u],
                oven_Clock_STK[oven_Clock_size / 10u],
                oven_Clock_size,
                    0u,
                    0u,
                    0u,
                    (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                    &os_err);


    OSTaskCreate(&oven_Controller_TCB, /* Create the task1 */
                "oven_Controller",
                oven_Controller,
                0u,
                oven_Controller_prio,
                &oven_Controller_STK[0u],
                oven_Controller_STK[oven_Controller_size / 10u],
                oven_Controller_size,
                    0u,
                    0u,
                    0u,
                    (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                    &os_err);

    OSTaskCreate(&oven_Manual_Assistance_TCB, /* Create the task1 */
                "oven_Manual_Assistance",
                oven_Manual_Assistance,
                0u,
                oven_Manual_Assistance_prio,
                &oven_Manual_Assistance_STK[0u],
                oven_Manual_Assistance_STK[oven_Manual_Assistance_size / 10u],
                oven_Manual_Assistance_size,
                    0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

    OSTaskCreate(&oven_Heat_TCB, /* Create the task1 */
            "oven_Heat",
            oven_Heat,
            0u,
            oven_Heat_prio,
            &oven_Heat_STK[0u],
            oven_Heat_STK[oven_Heat_size / 10u],
            oven_Heat_size,
                  0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

    OSTaskCreate(&oven_Preheat_TCB, /* Create the task1 */
            "oven_Preheat",
            oven_Preheat,
            0u,
            oven_Preheat_prio,
            &oven_Preheat_STK[0u],
            oven_Preheat_STK[oven_Preheat_size / 10u],
            oven_Preheat_size,
                  0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

    OSTaskCreate(&oven_Continue_TCB, /* Create the task1 */
            "oven_Continue",
            oven_Continue,
            0u,
            oven_Continue_prio,
            &oven_Continue_STK[0u],
            oven_Continue_STK[oven_Continue_size / 10u],
            oven_Continue_size,
                  0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

OSTaskCreate(&oven_END_TCB, /* Create the task1 */
            "oven_END",
            oven_END,
            0u,
            oven_END_prio,
            &oven_END_STK[0u],
            oven_END_STK[oven_END_size / 10u],
            oven_END_size,
                  0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

OSTaskCreate(&oven_Alarm_TCB, /* Create the task1 */
            "oven_Alarm",
            oven_Alarm,
            0u,
            oven_Alarm_prio,
            &oven_Alarm_STK[0u],
            oven_Alarm_STK[oven_Alarm_size / 10u],
            oven_Alarm_size,
                  0u,
                  0u,
                  0u,
                  (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                  &os_err);

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// OSTaskCreate(&Task1TCB, /* Create the task1 */
//             "Task1",
//             Task1,
//             0u,
//             APP_CFG_TASK1_PRIO,
//              &Task1Stk[0u],
//              Task1Stk[APP_CFG_TASK1_STK_SIZE / 10u],
//              APP_CFG_TASK1_STK_SIZE,
      //              0u,
      //              0u,
      //              0u,
      //              (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
      //              &os_err);

// OSTaskCreate(&Task2TCB, /* Create the task2 */
//                "Task2",
//                Task2,
//                0u,
//                APP_CFG_TASK2_PRIO,
//                &Task2Stk[0u],
//                Task2Stk[APP_CFG_TASK2_STK_SIZE / 10u],
//                APP_CFG_TASK2_STK_SIZE,
//                0u,
//                0u,
//                0u,
//                (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
//                &os_err);

// OSTaskCreate(&CommTaskTCB,                               /* Create the comm task
//               "Comm Task",
//                CommTask,
//                0u,
//                APP_CFG_COMM_TASK_PRIO,
//               &CommTaskStk[0u],
//                CommTaskStk[APP_CFG_COMM_TASK_STK_SIZE / 10u],
//                APP_CFG_COMM_TASK_STK_SIZE,
//                2u,
//                0u,
//                0u,
//               (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
//               &os_err);
// if (os_err != OS_ERR_NONE) {
//     while (1);
// }

// OSTaskCreate(&BtnTaskTCB,                               /* Create the comm task
    //                "Button Task",
    //                 BtnTask,
    //                 0u,
    //                 APP_CFG_BTN_TASK_PRIO,
    //                &BtnTaskStk[0u],
    //                 BtnTaskStk[APP_CFG_BTN_TASK_STK_SIZE / 10u],
    //                 APP_CFG_BTN_TASK_STK_SIZE,
    //                 0u,
    //                 0u,
    //                 0u,
    //                (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
    //                &os_err);
    // if (os_err != OS_ERR_NONE) {
    //     while (1);
    // }




#if APP_CFG_MATH_TASK
    /* ------------- CREATE FLOATING POINT TASK ----------- */
    OSTaskCreate((OS_TCB      *)&App_TaskEq0FpTCB,
            (CPU_CHAR    *)"FP  Equation 1",
            (OS_TASK_PTR  ) App_TaskEq0Fp,
            (void        *) 0,
            (OS_PRIO      ) APP_CFG_TASK_EQ_PRIO,
            (CPU_STK     *)&App_TaskEq0FpStk[0],
            (CPU_STK_SIZE ) App_TaskEq0FpStk[APP_CFG_TASK_EQ_STK_SIZE / 10u],
            (CPU_STK_SIZE ) APP_CFG_TASK_EQ_STK_SIZE,
            (OS_MSG_QTY   ) 0u,
            (OS_TICK      ) 0u,
            (void        *) 0,
            (OS_OPT       )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR | OS_OPT_TASK_SAVE_FP),
            (OS_ERR      *)&os_err);
    if (os_err != OS_ERR_NONE) {
        while (1);
    }
  #endif
}
```

```
/*
**********************************************************************************************
*                                            COMM TASK
*
* Description : This is an example of a Comm task.  As mentioned in the book's text, you MUST
*               initialize the ticker only once multitasking has started.
* Arguments   : p_arg   is the argument passed to 'StartupTask()' by 'OSTaskCreate()'.
* Returns     : none
* Notes       : 1) The first line of code is used to prevent a compiler warning because 'p_arg' is
*                  used.  The compiler should not generate any code for this statement.
**********************************************************************************************
*/

static  void  CommTask (void *p_arg)
{
  OS_ERR  os_err;
  void        *p_msg;
  OS_MSG_SIZE  msg_size;
  CPU_TS       ts;

  manualAssistant[0]='\0';

  (void)p_arg;

#if OS_CFG_STAT_TASK_EN > 0u
  OSStatTaskCPUUsageInit(&os_err);                          /* Compute CPU capacity with no task

#endif

#ifdef CPU_CFG_INT_DIS_MEAS_EN
  CPU_IntDisMeasMaxCurReset();
#endif

  while (DEF_TRUE) {                                        /* Task body, always written as an i
    p_msg = OSQPend(&CommQ,
                     0,
                     OS_OPT_PEND_BLOCKING,
                    &msg_size,
                    &ts,
                    &os_err);

        if(msg_size >0){
                memcpy(manualAssistant,p_msg, msg_size);
                msg_size = 0;
    }
    myprintf("...CommTask received ...%s\n\r", (char *)p_msg);
  }
}

/*****************************************************************************************************
*                                             BUTTON TASK
*
* Description : This is an example of a Button task.
* Arguments   : p_arg   is the argument passed to 'BtnTask()' by 'OSTaskCreate()'.
* Returns     : none
* Notes       : 1) The first line of code is used to prevent a compiler warning because 'p_arg' is
*                  used.  The compiler should not generate any code for this statement.
*****************************************************************************************************
*/

static  void  BtnTask (void *p_arg)
{
  OS_ERR  os_err;
  unsigned int count_press = 0;

  (void)p_arg;

#if OS_CFG_STAT_TASK_EN > 0u
  OSStatTaskCPUUsageInit(&os_err);                              /* Compute CPU capacity with no task
#endif

#ifdef CPU_CFG_INT_DIS_MEAS_EN
  CPU_IntDisMeasMaxCurReset();
#endif

  while (DEF_TRUE) {                                            /* Task body, always written as an i
        OSTimeDlyHMSM(0u, 0u, 0u, 10u,
                                        OS_OPT_TIME_HMSM_STRICT,
                                        &os_err);
    if(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN) == GPIO_PIN_SET){

                myprintf("BtnTask running: Button Pressed : %i times \n\r", count_press);
                count_press++;
                while(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN)==GPIO_PIN_SET);
    }
  }
}

/**************************************************************************************************
*                                            GUI TASK
*
* Description : This is the Graphical User Interfae task
* Arguments   : p_arg   is the argument passed to 'GuiTask()' by 'OSTaskCreate()'.
* Returns     : none
* Notes       : 1) The first line of code is used to prevent a compiler warning because 'p_arg' is
*                  used.  The compiler should not generate any code for this statement.
**************************************************************************************************
*/
void GuiTask(void *p_arg)
{
  /* USER CODE BEGIN GuiTask */
  OS_ERR  os_err;


  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_I2C3_Init();
  MX_LTDC_Init();
  MX_SPI5_Init();

  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(LCD_BACKGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_LayerDefaultInit(LCD_FOREGROUND_LAYER, LCD_FRAME_BUFFER);
  BSP_LCD_SelectLayer(LCD_FOREGROUND_LAYER);
  BSP_LCD_DisplayOn();
  BSP_LCD_Clear(LCD_COLOR_WHITE);

  char plantOutputStr[30];
  int plantOutputLine = 3;

  char ovenTemp[30];
  int ovenLine = 3;

  char ovenDesired[30];
  int ovenDLine = 5;

  while(1)
  {
    sprintf(ovenTemp, "Oven Temp: %d", OvenTempDisplay);
    showText(ovenLine, ovenTemp);

    sprintf(ovenDesired, "Desired Temp: %d", OvenDesiredDisplay);
    showText(ovenDLine, ovenDesired);



          OSTimeDlyHMSM(0u, 0u, 0u, 10u,
                                              OS_OPT_TIME_HMSM_STRICT,
                                              &os_err);
  }
}
```

```
/*********************
 *
 *
 * ADD MORE TASK FUNCTIONS HERE !
 *
 *
 *********************/
static void showText(uint16_t line, char* text) {
        BSP_LCD_ClearStringLine(line);
        BSP_LCD_DisplayStringAtLine((uint16_t) line, text);
}



/*************** ADDITIONAL TASKS AND HARDWARE SETUP ********************************************

#if APP_CFG_MATH_TASK
/*
**************************************************************************************************
*                                              App_TaskEqOFp()
*
* Description : This task finds the root of the following equation.
*                f(x) =  e^-x(3.2 sin(x) - 0.5 cos(x)) using the bisection mehtod
*
* Argument(s) : p_arg   is the argument passed to 'App_TaskEqOFp' by 'OSTaskCreate()'.
*
* Return(s)   : none.
*
* Note(s)     : none.
**************************************************************************************************
```

```
*/

void  App_TaskEq0Fp (void  *p_arg)
{
    CPU_FP32    a;
    CPU_FP32    b;
    CPU_FP32    c;
    CPU_FP32    eps;
    CPU_FP32    f_a;
    CPU_FP32    f_c;
    CPU_FP32    delta;
    CPU_INT08U  iteration;
    RAND_NBR    wait_cycles;

    OS_ERR  os_err;

    while (DEF_TRUE) {
        eps       = 0.00001;
        a         = 3.0;
        b         = 4.0;
        delta     = a - b;
        iteration = 0u;
        if (delta < 0) {
            delta = delta * -1.0;
        }

        OSTimeDlyHMSM(0u, 0u, 0u, 10u,
                      OS_OPT_TIME_HMSM_STRICT,
                      &os_err);
        while (((2.00 * eps) < delta) ||
               (iteration    > 20u  )) {
            c   = (a + b) / 2.00;
            f_a = (exp((-1.0) * a) * (3.2 * sin(a) - 0.5 * cos(a)));
            f_c = (exp((-1.0) * c) * (3.2 * sin(c) - 0.5 * cos(c)));

            if (((f_a > 0.0) && (f_c < 0.0)) ||
                ((f_a < 0.0) && (f_c > 0.0))) {
                b = c;
            } else if (((f_a > 0.0) && (f_c > 0.0)) ||
                       ((f_a < 0.0) && (f_c < 0.0))) {
                a = c;
            } else {
                break;
            }

            delta = a - b;
            if (delta < 0) {
               delta = delta * -1.0;
            }
            iteration++;

            wait_cycles = Math_Rand();
            wait_cycles = wait_cycles % 1000;

            while (wait_cycles > 0u) {
                wait_cycles--;
            }

            if (iteration > APP_TASK_EQ_0_ITERATION_NBR) {
                //APP_TRACE_INFO(("App_TaskEq0Fp() max # iteration reached\n"));
                break;
            }
        }

        APP_TRACE_INFO(("Eq0 Task Running ....\n"));

        if (iteration == APP_TASK_EQ_0_ITERATION_NBR) {
            APP_TRACE_INFO(("Root = %f; f(c) = %f; #iterations : %d\n", c, f_c, iteration));
        }
    }
}
#endif

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
```

```
    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

#if 0
  /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
```

```
    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
#endif
}

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM1 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM1) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**
  * @}
  */
#define USE_VCP 1
#define USE_MYMUTEX 0
#if USE_VCP
#ifdef __GNUC__
/* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
int __io_putchar(int ch);
int _write(int file,char *ptr, int len)
{
 int DataIdx;
#if USE_MYMUTEX
 static        OS_MUTEX           MyMutex;
 OS_ERR  os_err;
 CPU_TS  ts;
                // use mutex to protect VCP access
                OSMutexPend((OS_MUTEX *)&MyMutex,
                                  (OS_TICK   )0,
                                  (OS_OPT    )OS_OPT_PEND_BLOCKING,
                                  (CPU_TS   *)&ts,
                                  (OS_ERR   *)&os_err);
#endif
 for(DataIdx= 0; DataIdx< len; DataIdx++)
 {
 __io_putchar(*ptr++);
 }
#if USE_MYMUTEX
                OSMutexPost((OS_MUTEX *)&MyMutex,
                                  (OS_OPT    )OS_OPT_POST_NONE,
                                  (OS_ERR   *)&os_err);
#endif
 return len;
}
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
        while(CDC_Transmit_HS((uint8_t *)&ch, 1) != USBD_OK);
        return ch;
}

#else
#ifdef __GNUC__
int _write(int32_t file, uint8_t *ptr, int32_t len)
{
        /* Implement your write code here, this is used by puts and printf for example */
        /* return len; */
        int i;
        for(i=0; i<len; i++)
                ITM_SendChar(*ptr++);
        return len;
}
#endif
volatile int32_t ITM_RxBuffer = ITM_RXBUFFER_EMPTY;
```

```
int fputc(int ch, FILE *f) {
  return (ITM_SendChar(ch));
}

int fgetc(FILE *f) {                /* blocking */
  while (ITM_CheckChar() != 1);
  return (ITM_ReceiveChar());
}

#endif

int myprintf(const char *format, ...)
{
        OS_ERR  os_err;
        CPU_TS  ts;
    OSSemPend(&PrintSem,
              0,
              OS_OPT_PEND_BLOCKING,
              &ts,
              &os_err);
        va_list alist;
        va_start(alist, format);
        vprintf(format, alist);
    va_end(alist);
    OSSemPost(&PrintSem,
              OS_OPT_POST_1,
              &os_err);
    return 0;
}

/******END OF FILE****/
```