Use of MailBox in FreeRTOS

Targeted competences: Simple use of the mailbox in FreeRTOS

Hardware: STM32F7 Nucleo board

Framework: STM32CubeIDE v1.6.1 from STMircoelectronics

The aim of this document is to show how to use the mailbox concept in FreeRTOS in an example for a STM32 board.

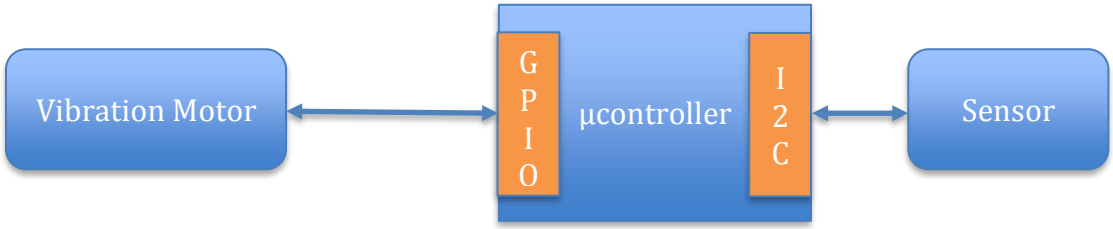


Figure 1: Global view of the system

We develop here a small code able to generate a control signal for a vibration motor inside a task and in function of values sent by another task by using a mailbox.

1. Microcontroller configuration

The first step is to configure the microcontroller. In our case, we use the NUCLEO-F767ZI platform based on a STM32F7 architecture. In order to configure this board we will use the CubeMx software. The first step after choosing the board is to name the project and then start to configure the microcontroller. First, we must use the freeRTOS by choosing him in the middleware tab as in figure 2.

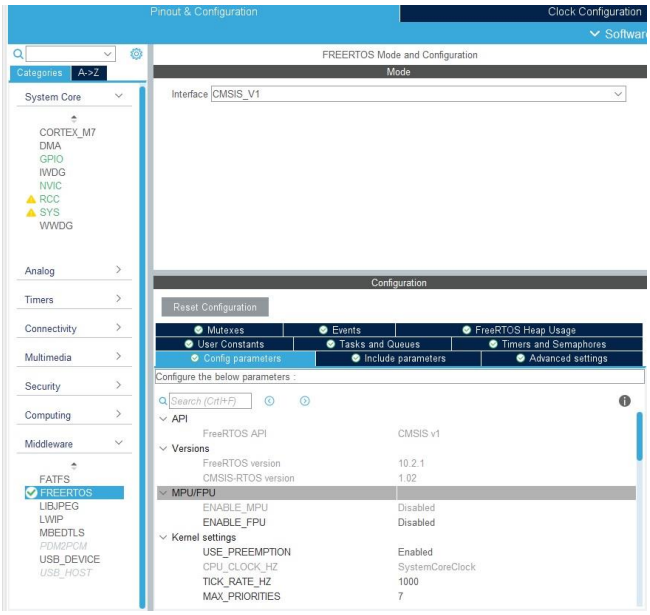


Figure 2:CubeIDE interface

In the FreeRTOS configuration tab, you have nothing to change; use the default configuration. In order to use tasks, we must create them into the Tasks and Queues tab.

Here, I add 2 tasks: one for the production of data (read the sensor data and send them via the mailbox) and one for the consumer of the sensor data. I use the default configuration for each task as you can see in the figure below. I called the tasks respectively ConsoTask that will use the StartConsoTask function and ProducTask that will use the StartProducTask. Each task has a normal priority level.

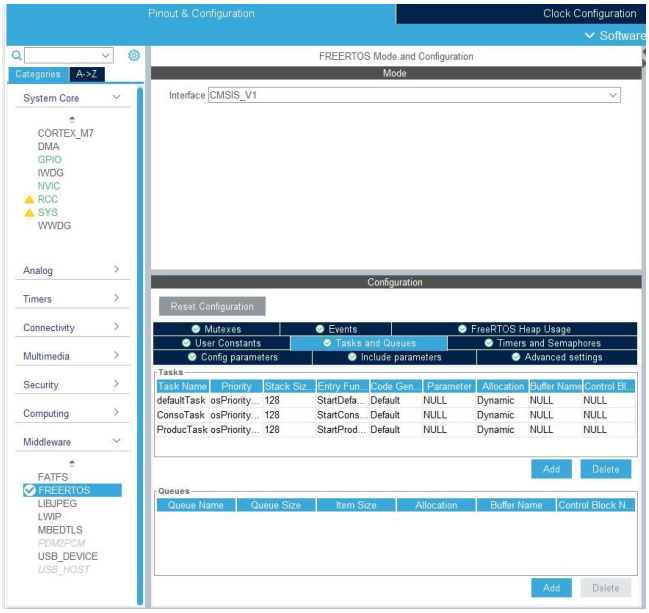


Figure 3: Task configuration tab

The next step is to configure the I^2C port in order to use it to connect the sensor to the microcontroller. As you can see in the figure below, I kept the classical (by default) configuration.

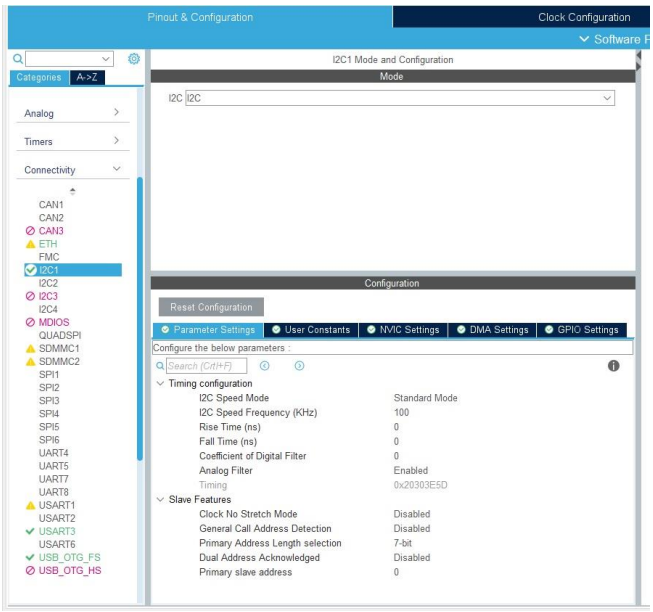


Figure 4: I^2C bus configuration

The last step is to configure the GPIO in output in order to pilot the vibration motor. Indeed, it is very easy to pilot him since we have just to set or reset an output pin to switch on or off the motor.

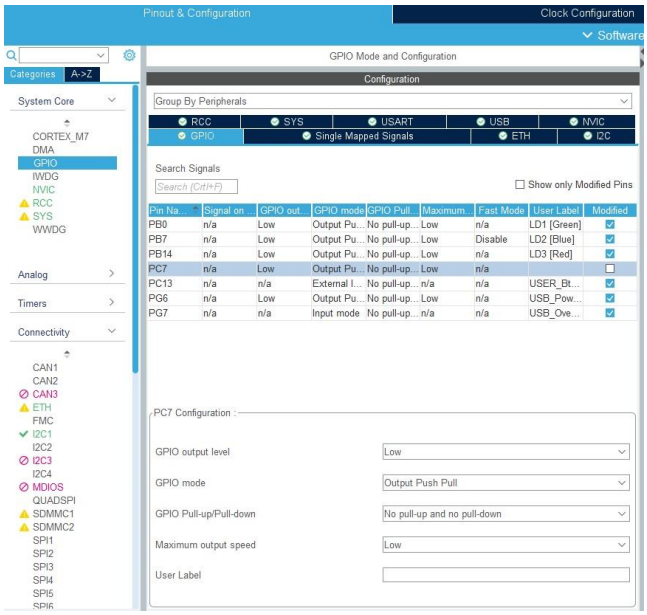


Figure 5: GPIO Configuration

Here, I also kept the default configuration then I connected the pin output to the motor control input and the VDD and GND pins of the moto to the VDD and GND pins of the board.

When this last step has been realized you have to generate the configuration code and the project by clicking on the Device configuration tool code generation.

Before using the sensor, we test the behavior of the mailbox with a simpler example. This example will read the state of the user button of the board and if this one is pushed then we switch on both the led 1 of the board and the vibration motor. The button state will be acquire by a first task and send via the mailbox to the other task. Therefore, the first task will read the user button state and will send it via a mailbox to the second task. This second task will compare the user button state with the push state and if these states match, then both the led 1 and the vibration motor will be switch on.

To do this, we must define some variables and the mailbox in order to send the user button state from one task to the other. You can see in the figure below these definitions.

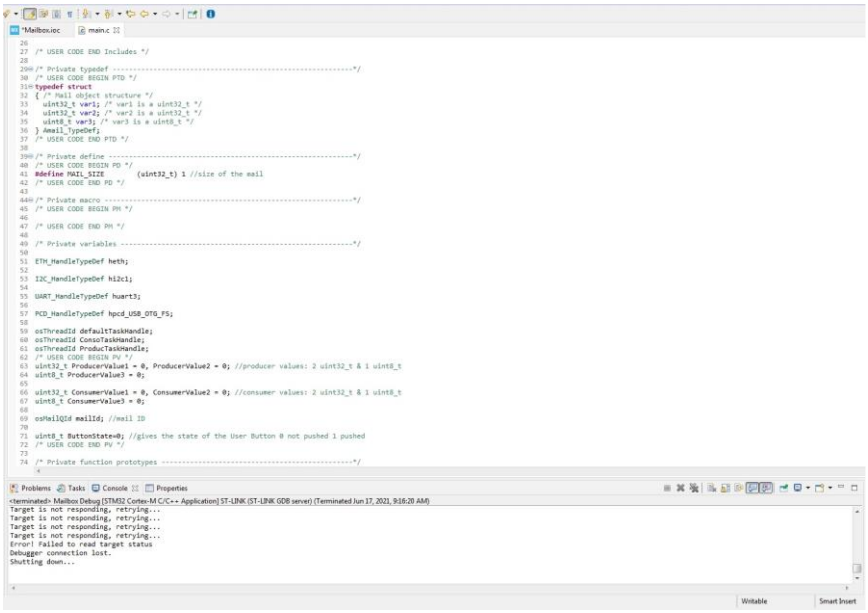


Figure 6: variable and mailbox definition

The first definition is for a data structure that store all the values of the mailbox. Here, 2 variables of 32 bit unsigned char and one 8 bit unsigned char are defined. Firstly, we will use the 8-bit variable to store the user button state, and then we will use the two other variables (32-bits) to store the values coming from the sensor. I define also the size of the mail; this parameter will be used later when we will create the mail queue. I define 4 more 32-bit unsigned char variables and 2 8-bit unsigned char variables to handle the different data. Finally, I define a mail ID variable whose the type is osMailQId (defined in the cmsis_os.h file).

The next step is to define and to create the mail queue and it is realized respectively by using the osMailQDef definition (line 147) and by using the osMailCreate function (line 149).

Next, we have to write the code of the production task in order to provide the data to the consumer task. The data will be store and send between the two tasks by using the mailbox. The figure below shows the first code that I wrote; this code is a simple example in order to test the functionality.

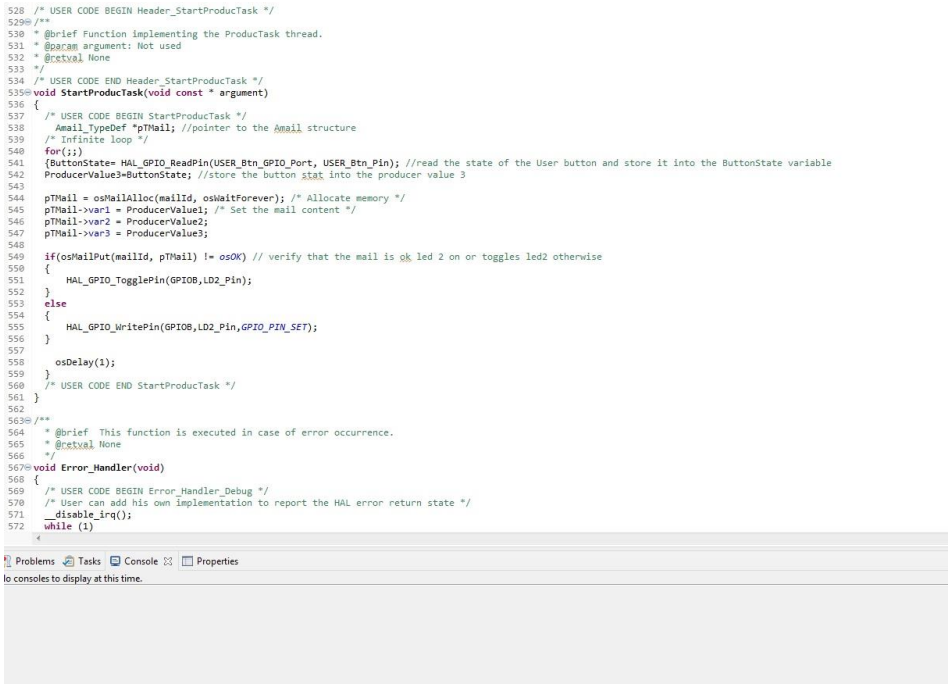


Figure 7: The producer task code

In this code, I read the state of the User button of the board and I store it into the ProducerValue3 variable. This variable is then store into the var3 field of the mailbox. If we cannot store the value into the mailbox then the led2 toggles, otherwise the led2 is switched on.

The second task to write is the consumer one. The figure below shows the code that I wrote in order to use the data comes from the mailbox.

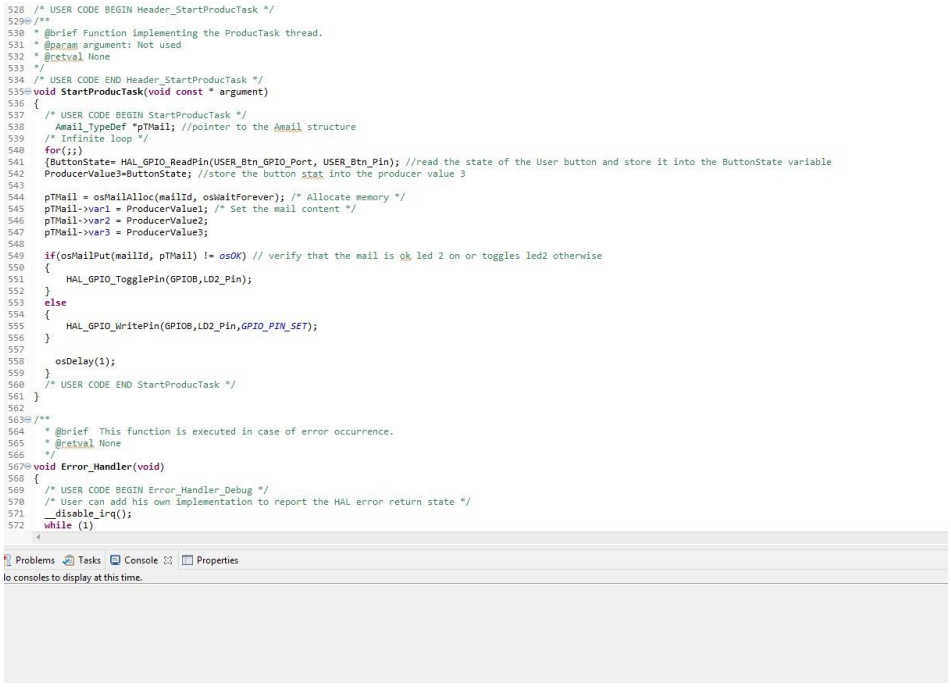


Figure 8: The consumer task code

If the User button state is equal to 0 (I also test the values of var1 and var2 fields but here they should be always equal to 0) then I switch off the led 1 and the vibration motor. Otherwise, both the led1 and the vibration moto is switched on.

You can test this code before use the more complex example using the sensor. To do that compile the code, download the execution code into the board and press the User Button.

After making this first simple example, I wrote another code more "complex" in order to send sensor values from one task to another one by using the mailbox. First, I defined new variables to manipulate sensor data as you can see in the figure below.

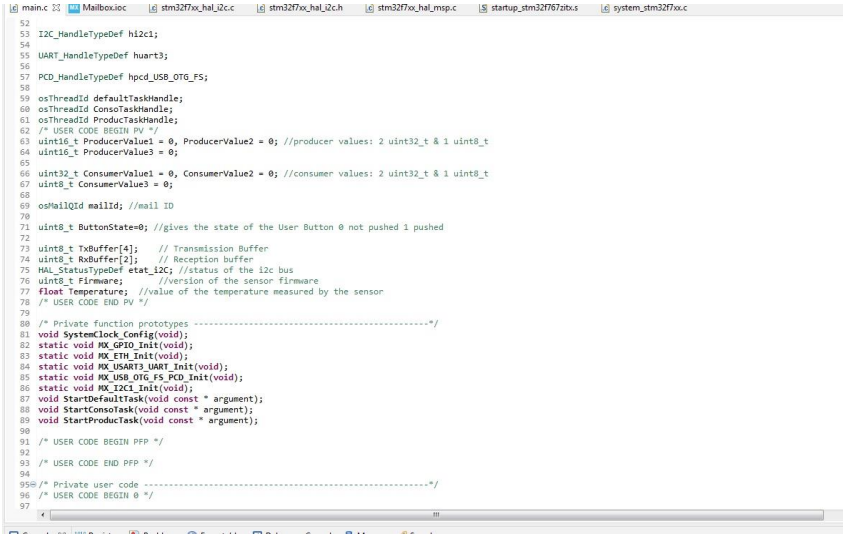
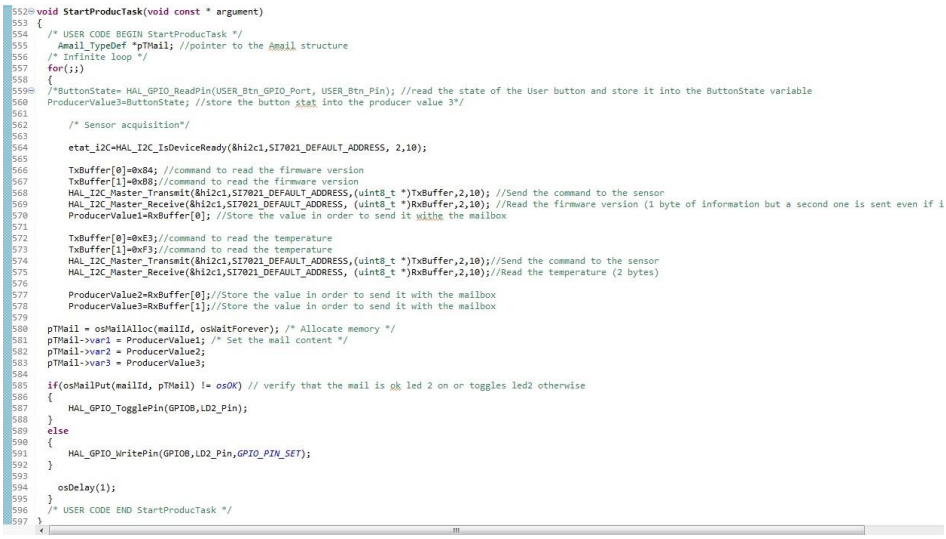


Figure 9: New variable definitions

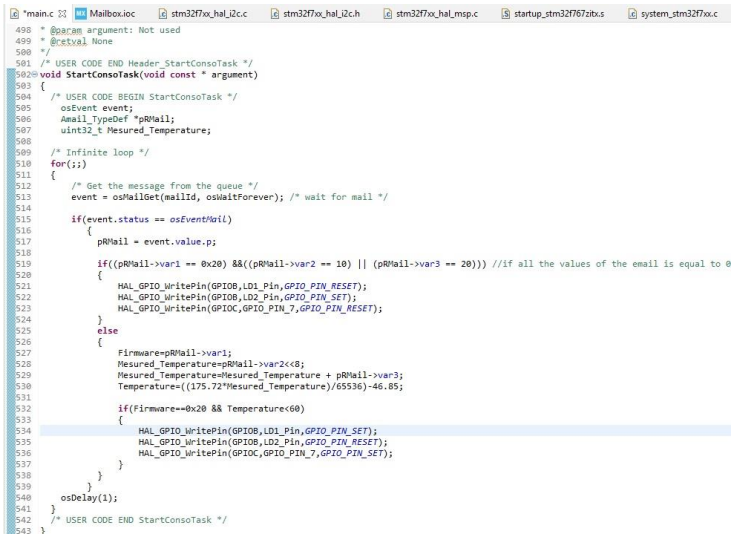
Now I modified the production task in order to add the sensor data acquisition. The sensor uses an I^2C bus connection to transmit its information to a master. In our case, the master is the µcontroller and it sends to the sensor some commands in order to obtain the wanted values. I used the SI7021 sensor that permits to obtain humidity and temperature values. To use this sensor, see its datasheet because here I only use the command to obtain the firmware version and the temperature value.
To obtain the different values of the sensor we have to use some predefined commands. To read the firmware version we must send the command 0x84 then 0xB8. To do that you must use the I^2C bus, transmit these values from the master (µcontroller) to the sensor, and then listen the sensor answer. Afterward, I want to read the temperature value from the sensor. In order to realize the read you must send to the sensor the following commands: 0xE3,0XF3. The answer of the sensor is stored into the mailbox in order to send these values to the consumer task. The figure below shows you the C code to do both the read of the firmware version and of the temperature value.



```
void StartProducTask(void const * argument)
{
  /* USER CODE BEGIN StartProducTask */
    Amail_TypeDef *pTMail; //pointer to the Amail structure
  /* Infinite loop */
  for(;;)
  {
  /*ButtonState= HAL_GPIO_ReadPin(USER_Btn_GPIO_Port, USER_Btn_Pin); //read the state of the User button and store it into the ButtonState variable
  ProducerValue3=ButtonState; //store the button stat into the producer value 3*/

      /* Sensor acquisition*/

      etat_i2C=HAL_I2C_IsDeviceReady(&hi2c1,SI7021_DEFAULT_ADDRESS, 2,10);

      TxBuffer[0]=0x84; //command to read the firmware version
      TxBuffer[1]=0xB8;//command to read the firmware version
      HAL_I2C_Master_Transmit(&hi2c1,SI7021_DEFAULT_ADDRESS,(uint8_t *)TxBuffer,2,10); //Send the command to the sensor
      HAL_I2C_Master_Receive(&hi2c1,SI7021_DEFAULT_ADDRESS, (uint8_t *)RxBuffer,2,10); //Read the firmware version (1 byte of information but a second one is sent even if i
      ProducerValue1=RxBuffer[0]; //Store the value in order to send it withe the mailbox

      TxBuffer[0]=0xE3;//command to read the temperature
      TxBuffer[1]=0xF3;//command to read the temperature
      HAL_I2C_Master_Transmit(&hi2c1,SI7021_DEFAULT_ADDRESS,(uint8_t *)TxBuffer,2,10);//Send the command to the sensor
      HAL_I2C_Master_Receive(&hi2c1,SI7021_DEFAULT_ADDRESS, (uint8_t *)RxBuffer,2,10);//Read the temperature (2 bytes)

      ProducerValue2=RxBuffer[0];//Store the value in order to send it with the mailbox
      ProducerValue3=RxBuffer[1];//Store the value in order to send it with the mailbox

  pTMail = osMailAlloc(mailId, osWaitForever); /* Allocate memory */
  pTMail->var1 = ProducerValue1; /* Set the mail content */
  pTMail->var2 = ProducerValue2;
  pTMail->var3 = ProducerValue3;

  if(osMailPut(mailId, pTMail) != osOK) // verify that the mail is ok led 2 on or toggles led2 otherwise
  {
      HAL_GPIO_TogglePin(GPIOB,LD2_Pin);
  }
  else
  {
      HAL_GPIO_WritePin(GPIOB,LD2_Pin,GPIO_PIN_SET);
  }

    osDelay(1);
  }
  /* USER CODE END StartProducTask */
}
```

Figure 10: Code to read the firmware version and the temperature value

It is important to notice that in order to read a value from the sensor you have to, first send to it a command by using the I^2C Transmit function then listen to the answer of the sensor by using the I^2C Receive function.
Then, we have to modify the consumer task code. Here, I write a code allows us to compute the temperature value by using the answer of the sensor. Indeed, the sensor returns us a value on 2 bytes so we must concatenate this value to be able to compute the real temperature value see the data sheet to know how compute this value). Afterward, I compare both the temperature and the firmware value in order to switch on the led 1 and the vibration motor. The code is shown below.



```
*main.c  Mailbox.ioc  stm32f7xx_hal_i2c.c  stm32f7xx_hal_i2c.h  stm32f7xx_hal_msp.c  startup_stm32f767zitx.s  system_stm32f7xx.c
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartConsoTask */
void StartConsoTask(void const * argument)
{
  /* USER CODE BEGIN StartConsoTask */
    osEvent event;
    Amail_TypeDef *pRMail;
    uint32_t Mesured_Temperature;

  /* Infinite loop */
  for(;;)
  {
      /* Get the message from the queue */
     event = osMailGet(mailId, osWaitForever); /* wait for mail */

     if(event.status == osEventMail)
        {
          pRMail = event.value.p;

          if((pRMail->var1 == 0x20) &&((pRMail->var2 == 10) || (pRMail->var3 == 20))) //if all the values of the email is equal to 0
          {
              HAL_GPIO_WritePin(GPIOB,LD1_Pin,GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOB,LD2_Pin,GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,GPIO_PIN_RESET);
          }
          else
          {
              Firmware=pRMail->var1;
              Mesured_Temperature=pRMail->var2<<8;
              Mesured_Temperature=Mesured_Temperature + pRMail->var3;
              Temperature=((175.72*Mesured_Temperature)/65536)-46.85;

              if(Firmware==0x20 && Temperature<60)
              {
                  HAL_GPIO_WritePin(GPIOB,LD1_Pin,GPIO_PIN_SET);
                  HAL_GPIO_WritePin(GPIOB,LD2_Pin,GPIO_PIN_RESET);
                  HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7,GPIO_PIN_SET);
              }
          }
        }
    osDelay(1);
  }
  /* USER CODE END StartConsoTask */
}
```

Figure 11: Consumer task code

A video showing the code test can be found here: https://youtu.be/5MFWuZELhwM