Real-Time Vending Machine Design Document

CENG 3020
Design Project 2

Ahmed Haroon - T00562613
Cem Doganay - T00582940
James H. Howe - T00562601

THOMPSON RIVERS
UNIVERSITY

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the design and the rationale behind the design of the real-time vending machine. It should sufficiently describe the design to any reader that wishes to review or replicate the design of the product.

1.2 Scope

This document is focused on the decisions made regarding the design of the product. It covers the parameters that were used for the basis of the design and it discusses the interfaces with which the user will interact with. The real time design decisions are then discussed. The risks and limitations of this project and design are then listed. This document concludes with summarizing the design and discussing possible improvement that could be made in the future.

1.3 Project Overview

This project emulates the behavior of a real-time vending machine which allows the users to purchase different types of chocolate. It allows the user to pay either $1 to $2 at a time and will allow a maximum balance of $15. It will refund money when requested by the user or when there is a significant lapse of time between the payment.

2 Design

2.1 Parameters

- Number of Types of Chocolates: It was decided that the machine would service 4 different types of chocolate items with different prices. This is because of the hardware limitations that we had which allowed us to add only 4 buttons for the user.
- Types of coins: It was decided that only 2 types of coins would be allowed. The $1 and $2 coins. This is because we are limited to showing the current balance using only 4 leds. This is discussed more in the interfaces section.

- Lapsed Time: We decided that the lapse time would be 60 seconds. This seemed as a reasonable amount of time to consider that the user wants to cancel the transaction.

2.2 User Interface

The user interface will consist of a number of LEDs and buttons. The system has only 4 LEDs available (due to hardware considerations) to show the amount of money that the user has paid (current balance). Due to this limitation, the interface will show the binary representations of the balance. The maximum amount for the balance is also limited to $15 due to the fact that only 4 LEDs are being used to represent the balance. 2 other LEDs on the board are being used as well to indicate to the user when the system is expecting money to be inputted and when it expects the user to select the chocolate item.

5 buttons are also being used for the user input. When the system is in the payment phase, only 3 buttons work where one indicates $1 payment, one indicates $2 payment and the other is for the user to finish payment. The system does not have options for payments that are not whole numbers (such as 50 cents) due to the limitation of using only 4 LEDs to represent the balance in binary. When the system is in the chocolate selection phase, 4 buttons are used by the user to indicate which chocolate they want to buy. The user will be able to buy chocolates that cost $1, $2, $3 and $4. The 5th button will be used by the user to finish the transaction.

2.3 Real Time Design Decisions & Rationale

For this project, the system is supposed to facilitate the transaction between the vending machine and the user. We have identified two phases that the user will go through when using the system:

- Payment Phase: This is when the user is inputting money into the machine for purchasing the chocolate items.
- Selection Phase: This is when the user is selecting the chocolate items that they want.

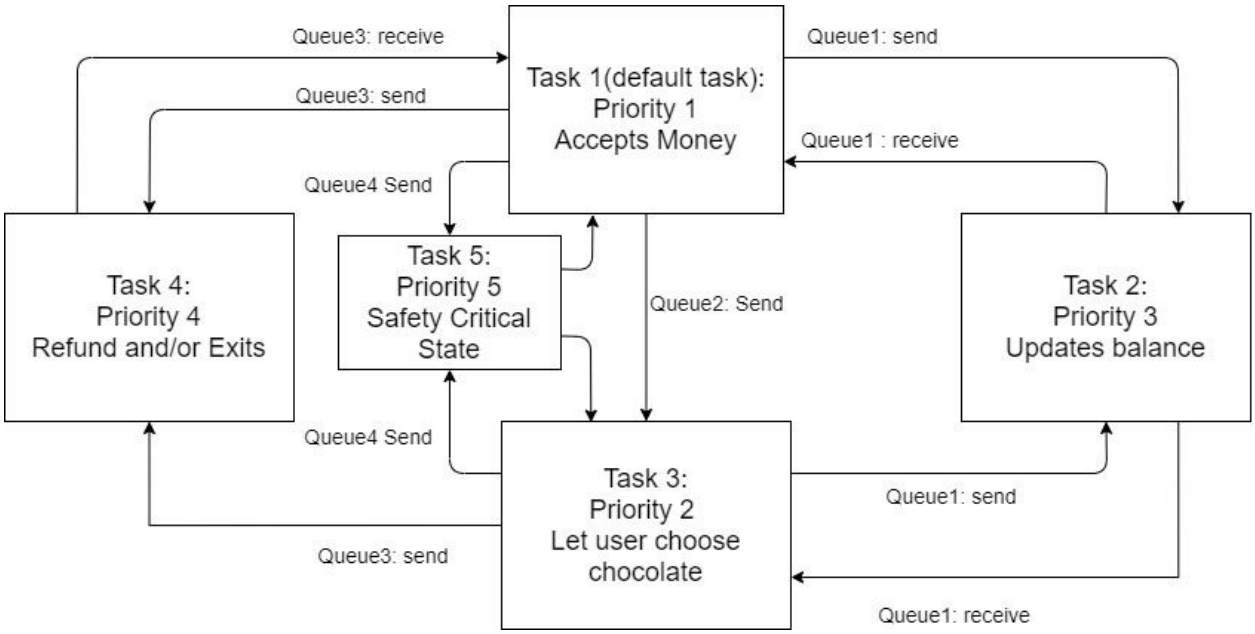

Figure 1 - Design flow of the real-time vending machine system

Real-Time decisions include having 5 tasks and 4 queues. When the system first starts, all the queues are empty so all the tasks which try to receive something from any queue are blocked in the beginning, leaving only task 1 ready. The purpose for each of the tasks and the queues (referred from figure 1) are explained below:

- Task1: This task is what basically implements the payment phase of the system. Blue LED is switched on to indicate to the user that the payment phase is ongoing. It reads the input by the user and then updates the balance. It has the lowest priority and it does not depend on any queue to run. It does, however, send to all the queues to trigger the start of other tasks for different situations. Task 2 is triggered to everytime the user enters payment and the balance needs to be updated. Task 3 is triggered when the user has finished paying. Task 4 is triggered when the time between 2 consecutive payments is longer than the lapsed time (specified in previous section). Task 5 is triggered constantly to check if the system needs to be in the safety critical state.

- Task 2: This task receives from queue 1 so this task is triggered whenever another task sends to queue 1. It then updates the current balance, displays that current balance on the LEDs and then returns back to the task that triggered it. It's priority is greater than tasks 1 and 3.

- Task 3: This task is responsible for facilitating the selection phase of the system. It is triggered whenever task 1 sends to queue 2. The user input that this task reads is what type of chocolate the user is requesting. The task then updates the balance accordingly. Everytime the system registers a user input in this task, it triggers task 2 in order to update the balance. It then triggers task 4 when the user has indicated that they have finished with the transaction. The priority of this task is only higher than task 1 so that once the selection phase starts, the selection phase should not start again. Task 5 is triggered constantly to check if the system needs to be in the safety critical state.

- Task 4: This task is responsible for ending the transaction. If there is still some balance left, it refunds it to make the balance zero and displays that on the LEDs. It then triggers task 1 to start the next transaction. Its priority is greater than tasks 1, 2 and 3.

- Task 5: This is the critical-state which is triggered when the inner temperature of the machine is higher than the range specified in the requirements document for an extended period of time. The purpose of this task is to prevent the user from digesting potentially spoiled food. During the specified condition, it turns the blue and red LEDs on to indicate that the products may be spoiled. The system then stays in that task until someone from maintenance checks that the products are safe and restarts the system. This task has the highest priority so that no other task can pre-empt it.

3 Risks, Limitations and Other Considerations

3.1 Limitations

- Hardware Limitations: The team did not have access to the desired amount of electrical hardware which forced a significant level of cutback on some features and the complete rejection of other features that would be useful for this product.

- Software Limitations: There were a number of problems that we faced regarding the installation of the software which resulted in delays of the implementation of the design. No other software was also known to the team in regards to implementing real-time systems.

- Limitations on Collaborations: The team was restricted to communicating online only which hindered the team's ability to meet and work effectively.

3.2 Health and Safety Risks Considered

- Power cut: In case of an unexpected power cut, the air conditioner unit will lose its power and the interior chocolates might become spoiled depending on the ambient temperature.
- Spoiled food: If the chocolates are kept at a temperature that is not ideal then there is a possibility of the chocolates sustaining damage.
- Tipping of Vending Machine: If the vending machine is not mounted on the floor, it might fall onto a person in case of user-caused disruption or an earthquake.
- Liquid related risks: The vending machine is designed only for indoor operations and is not waterproof. Liquids might damage the vending machine and could harm the ones near it.
- Vandalism and thievery: Vending machines do not have a resilient glass and it might be possible to break the glass and access the products. Also, the vending machine could be stolen if it is not mounted on the floor properly.

3.3 Other Considerations

- Economical: The operating system does not require updates as the vending machine does not connect to the internet, reducing costs to maintain.
- Environmental: The lighting used in the vending machine will be LEDs reducing power consumption compared to fluorescent lights.
- Cultural: The currency accepted for this vending machine will only be Canadian coins by default. Different hardware is required for other currencies.
- Societal: Easy instructions describing how to purchase products will be included in the exterior graphical design for confused individuals.

3.4 Ethical Aspects

- If the basket does not detect the purchased chocolate within five seconds after dispensing, the vending machine will dispense the same chocolate. This

feature addresses the problem of if the purchased chocolate gets stuck somewhere in the vending machine when payment has already occured.

3.5 Other Standards

- The company that acquires one of our vending machines has to follow guidelines of their provincial governments' vending machine policy. For example, the BC government uses "Healthier Choices in Vending Machines in B.C. Public Buildings Policy."
- Federal Governments "Refrigerated vending machines, Energy performance standard – March 2017" has been followed during the design of the vending machine.
- MISRA coding standard guidelines have been followed which the core FreeRTOS source files conform to.
- GNU coding standards have been followed in order to improve readability and understandability of the code.
- ISO 9000-3 Software Engineering Standard has been followed as discussed in the Software Quality Attributes section of the SRS Document.

4 Conclusion and Future Work

The purpose of this project is to emulate the behavior of a vending machine using real-time systems. To do this, the design team has incorporated tasks and queues which were responsible for emulating the different operations of a vending machine. This vending machine would service only 4 types of chocolates and does not accept the total amount paid to be greater than $15. This is due to hardware, software and collaborative limitations which were imposed on the project. There is a safety critical condition which prevents users from getting possibly spoiled items. Risks, limitations and other considerations were discussed in terms of vending machines in general. For future work, we would like to improve the user interface to show the prices of the items along with the current balance. The safety-critical task currently is never used due to the fact that we do not have the hardware to implement that task so that is something we could add in the future. Also, incorporating a number pad instead of buttons would be an improvement in the input interface.

5 References

Healthier Choices in Vending Machines in BC Public Buildings. (2014). Retrieved April 4, 2020, from https://www2.gov.bc.ca/assets/gov/health/managing-your-health/healthy-eating/vending-policy-2014.pdf

Refrigerated vending machines – March 2017. (2019, February 4). Retrieved April 2, 2020, from https://www.nrcan.gc.ca/refrigerated-vending-machines-march-2017/19428

Activities MISRA C. (n.d.). Retrieved April 3, 2020, from https://www.misra.org.uk/Activities/MISRAC/tabid/160/Default.aspx

GNU Coding Standards. (n.d.). Retrieved April 2, 2020, from https://www.gnu.org/prep/standards/standards.html

ISO 9000-3:1997. (2004, February 11). Retrieved April 4, 2020, from https://www.iso.org/standard/26364.html

Appendix A: Additional Figures

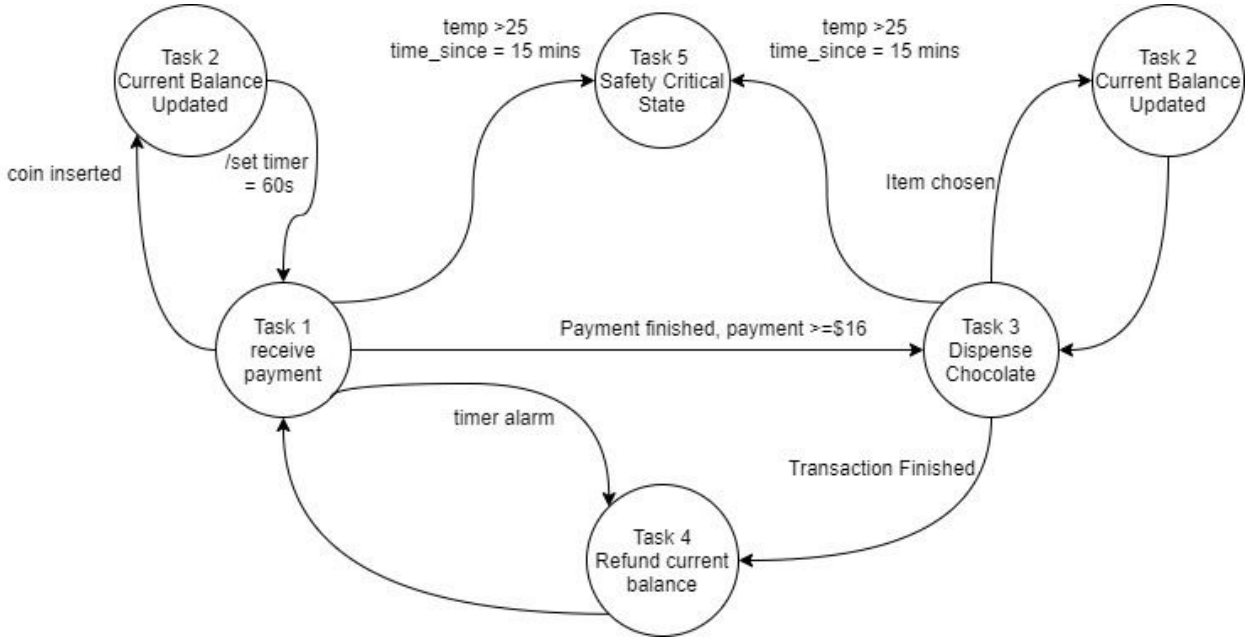

Figure 2 - Finite State Machine of the System

Appendix B: Code Used in "main.c" File

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"


/* Private variables ---------------------------------------------------------*/
osThreadId Task1Handle;
osThreadId Task2Handle;
osThreadId Task3Handle;
osThreadId Task4Handle;
osThreadId Task5Handle;
osMessageQId Queue1Handle;
osMessageQId Queue2Handle;
osMessageQId Queue3Handle;
osMessageQId Queue4Handle;

int balance = 0;
int money_input = 0;

TickType_t badTempStartTime;
TickType_t timer;


/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void v_Task1(void const * argument);
void v_Task2(void const * argument);
void v_Task3(void const * argument);
void v_Task4(void const * argument);
void v_Task5(void const * argument);

int main(void)
{

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

  /* Configure the system clock */
  SystemClock_Config();


  /* Initialize all configured peripherals */
  MX_GPIO_Init();

   /* definition and creation of Queue1 */
  osMessageQDef(Queue1, 1, uint16_t);
  Queue1Handle = osMessageCreate(osMessageQ(Queue1), NULL);

  /* definition and creation of Queue2 */
  osMessageQDef(Queue2, 1, uint16_t);
```

```
  Queue2Handle = osMessageCreate(osMessageQ(Queue2), NULL);

  /* definition and creation of Queue3 */
  osMessageQDef(Queue3, 1, uint16_t);
  Queue3Handle = osMessageCreate(osMessageQ(Queue3), NULL);

  /* definition and creation of Queue4 */
  osMessageQDef(Queue4, 1, uint16_t);
  Queue4Handle = osMessageCreate(osMessageQ(Queue4), NULL);

    /* Create the thread(s) */
  /* definition and creation of Task1 */
  osThreadDef(Task1, v_Task1, osPriorityLow, 0, 128);
  Task1Handle = osThreadCreate(osThread(Task1), NULL);

  /* definition and creation of Task2 */
  osThreadDef(Task2, v_Task2, osPriorityNormal, 0, 128);
  Task2Handle = osThreadCreate(osThread(Task2), NULL);

  /* definition and creation of Task3 */
  osThreadDef(Task3, v_Task3, osPriorityBelowNormal, 0, 128);
  Task3Handle = osThreadCreate(osThread(Task3), NULL);

  /* definition and creation of Task4 */
  osThreadDef(Task4, v_Task4, osPriorityAboveNormal, 0, 128);
  Task4Handle = osThreadCreate(osThread(Task4), NULL);

  /* definition and creation of Task5 */
  osThreadDef(Task5, v_Task5, osPriorityHigh, 0, 128);
  Task5Handle = osThreadCreate(osThread(Task5), NULL);

  /* Start scheduler */
  osKernelStart();

  }

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** 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_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
```

```
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** 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_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC0 PC1 PC2 PC3 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PB12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : PB13 PB14 PB15 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : PD8 PD9 */
  GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}


void v_Task1(void const * argument)
{
        TickType_t start = xTaskGetTickCount();
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); //signals payment phase

        for (;;)
        {
                money_input = 0;

                //xQueueSendToBack(Queue4Handle, Task1Handle, portMAX_DELAY); critical task

                if (balance >= 15)
                {
                        xQueueSendToBack(Queue2Handle, Task1Handle, portMAX_DELAY);
                }

                if ((xTaskGetTickCount() - start) >= pdMS_TO_TICKS(15000))
                        {
                        xQueueSendToBack(Queue3Handle, Task1Handle, portMAX_DELAY);
                        start = xTaskGetTickCount();
                        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
                }

                if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_9)) //1 dollar button
                {
                        money_input = 1;
                        xQueueSendToBack(Queue1Handle, Task1Handle, portMAX_DELAY);
                        start = xTaskGetTickCount();
                }

                if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_8)) //2 dollar button
                {
                        money_input = 2;
                        xQueueSendToBack(Queue1Handle, Task1Handle, portMAX_DELAY);
                        start = xTaskGetTickCount();
                }
```

```
                if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13)) // next button
                {
                        xQueueSendToBack(Queue2Handle, Task1Handle, portMAX_DELAY);
                        TickType_t start = xTaskGetTickCount();
                        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
                }
        }
}


void v_Task3(void const * argument)
{

        xQueueReceive(Queue2Handle, Task3Handle, portMAX_DELAY);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET); // signals selection phase
        HAL_Delay(200);

          /* Infinite loop */

        for (;;)
        {

           money_input = 0;
           // xQueueSendToBack(Queue4Handle, Task3Handle, portMAX_DELAY); safety critical task



            if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_9)) //button for $1 chocolate
            {
                        money_input = -1;
                        xQueueSendToBack(Queue1Handle, Task3Handle, portMAX_DELAY);
            }

            else if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_8)) //button for $2 chocolate
            {
                        money_input = -2;
                        xQueueSendToBack(Queue1Handle, Task3Handle, portMAX_DELAY);
            }

            else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15)) //button $3 chocolate
            {
                        money_input = -3;
                        xQueueSendToBack(Queue1Handle, Task3Handle, portMAX_DELAY);
            }

            else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_14)) //button $4 chocolate
            {
                        money_input = -4;
                        xQueueSendToBack(Queue1Handle, Task3Handle, portMAX_DELAY);
            }

             else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13)) //done transaction button
             {
                        xQueueSendToBack(Queue3Handle, Task3Handle, portMAX_DELAY);
```

```
                        xQueueReceive(Queue2Handle, Task3Handle, portMAX_DELAY);
                        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
            }

        }
}



void v_Task2(void const * argument)
{
{
   int n = 0;


   for (;;)
   {

      xQueueReceive(Queue1Handle, Task2Handle, portMAX_DELAY);

      if (balance + money_input >= 0)
      {
         balance = balance + money_input;
      }
      money_input = 0;


      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); // update!
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_RESET); // update!
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET); // update!
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // update!

      n = balance;

      // function to convert decimal to binary

       int binaryNum[4] = {0,0,0,0};  // array to store binary number


      // counter for binary array
      int i = 0;
      while (n > 0) {

         // storing remainder in binary array
         binaryNum[i] = n % 2;
         n = n / 2;
         i++;
      }

      // printing binary array in reverse order
      if (binaryNum[0] == 1)
      {
         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); // update!
          HAL_Delay(200);
```

```
        }

        if (binaryNum[1] == 1)
        {
           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_SET);
            HAL_Delay(200);
        }

        if (binaryNum[2] == 1)
        {
           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_SET); // update!
            HAL_Delay(200);
        }

        if (binaryNum[3] == 1)
        {
           HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET); // update!
            HAL_Delay(200);
        }

    }

}
}



void v_Task4(void const * argument)
{
          for(;;)
          {

             xQueueReceive(Queue3Handle, Task4Handle, portMAX_DELAY);

             balance = 0;

              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); // update!
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_RESET); // update!
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET); // update!
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // update!
          }
}

void v_Task5(void const * argument) //safety critical task NOT IMPLEMENTED
{
          int temp = 0;

          for (;;)
          {

                    xQueueReceive(Queue4Handle, Task5Handle, portMAX_DELAY);

                    if (badTempStartTime != 0 && ((xTaskGetTickCount() - badTempStartTime) >
                              pdMS_TO_TICKS(900000)))
```

```
                        {
                                for (;;)
                                {
                                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, GPIO_PIN_SET);
                                HAL_Delay(500);
                                HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, GPIO_PIN_RESET);
                                HAL_Delay(500);
                                }
                        }


                /*if (temp != getTemp())       Safety critical section
                {
                        if (getTemp() >= 25)
                        {
                                badTempStartTime = xTaskGetTickCount();
                        }

                        else
                        {
                                badTempStartTime = 0;
                        }
                } */

        }
}

void Error_Handler(void)
{ }

void assert_failed(uint8_t *file, uint32_t line)
{ }


 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
   if (htim->Instance == TIM1) {
   HAL_IncTick();
  }
}


#ifdef  USE_FULL_ASSERT

#endif
```