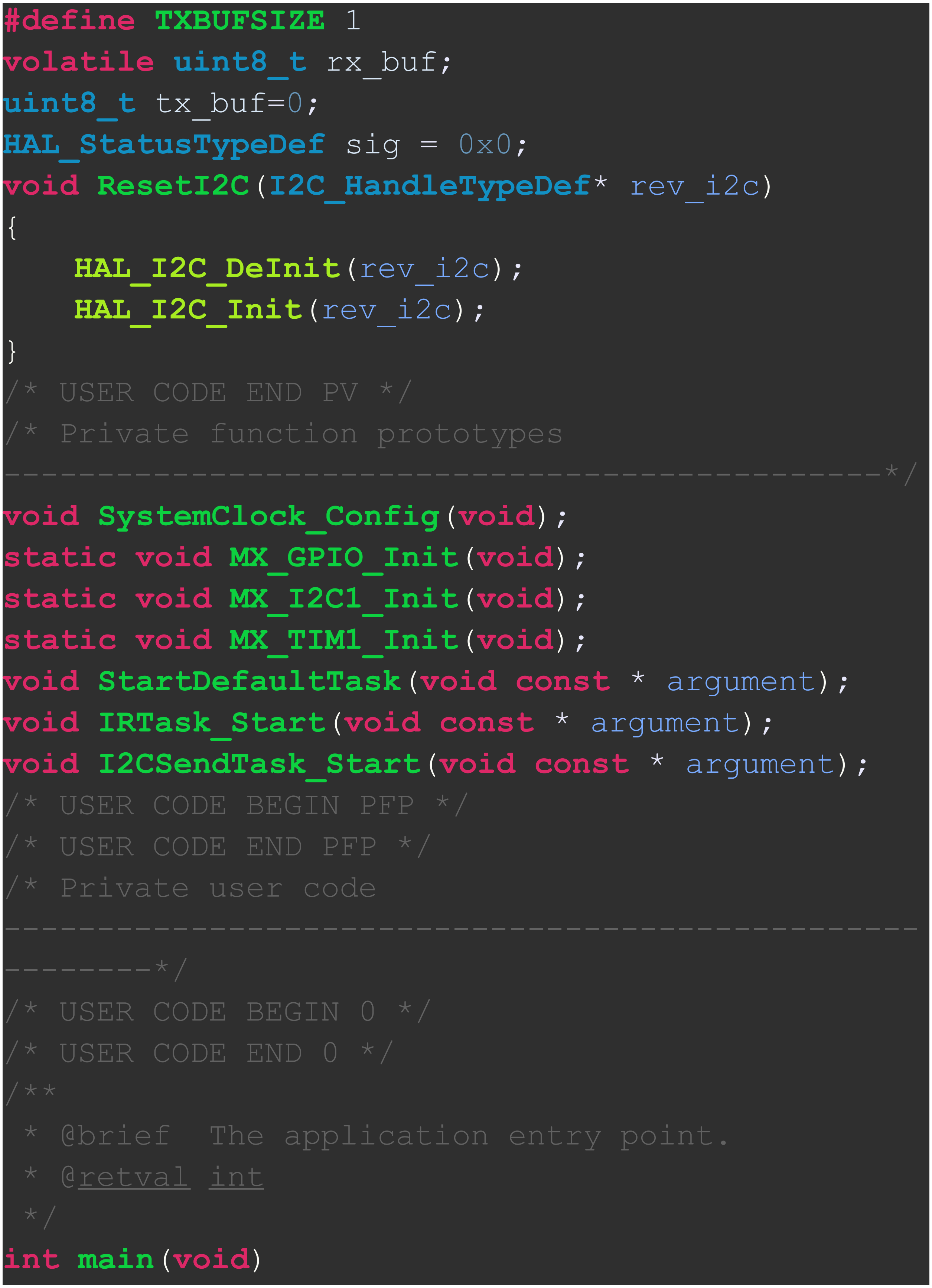
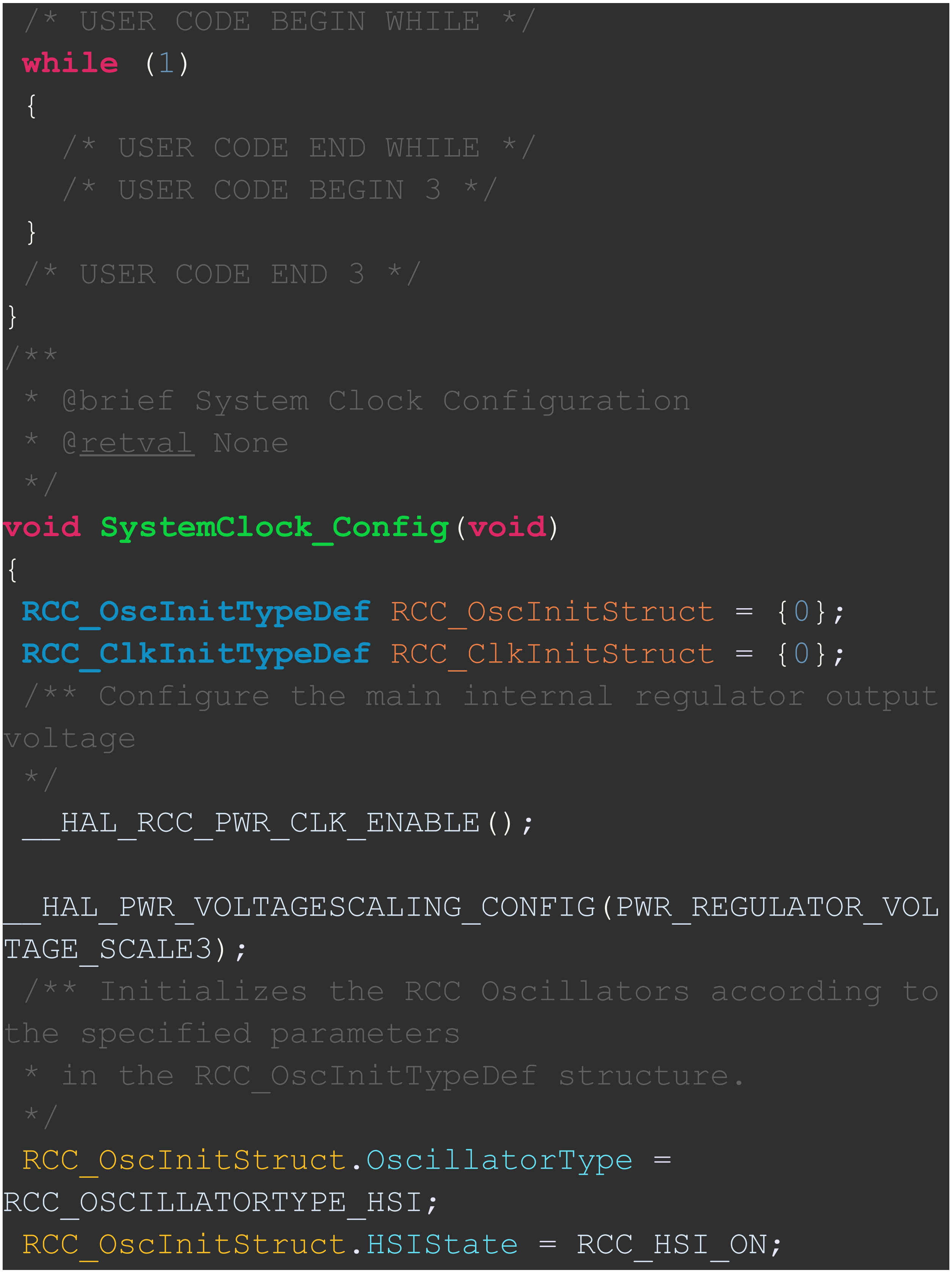
```c
/* USER CODE BEGIN Header */
/**

*************************************************
*****************************
 * @file           : main.c
 * @brief          : Main program body

*************************************************
*****************************
 * @attention
 *
 * Copyright (c) 2024 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can
be found in the LICENSE file
 * in the root directory of this software
component.
 * If no LICENSE file comes with this software,
it is provided AS-IS.
 *

*************************************************
*****************************
 */
/* USER CODE END Header */
/* Includes
-------------------------------------------------
-----------------*/
#include "main.h"
```

```
#include "cmsis_os.h"
/* Private includes
-------------------------------------------------
---------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef
-------------------------------------------------
----------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define
-------------------------------------------------
-----------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro
-------------------------------------------------
------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables
-------------------------------------------------
--------*/
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;
osThreadId defaultTaskHandle;
osThreadId IrReadTaskHandle;
osThreadId I2CSendTaskHandle;
osMutexId I2CMutexHandle;
/* USER CODE BEGIN PV */
```

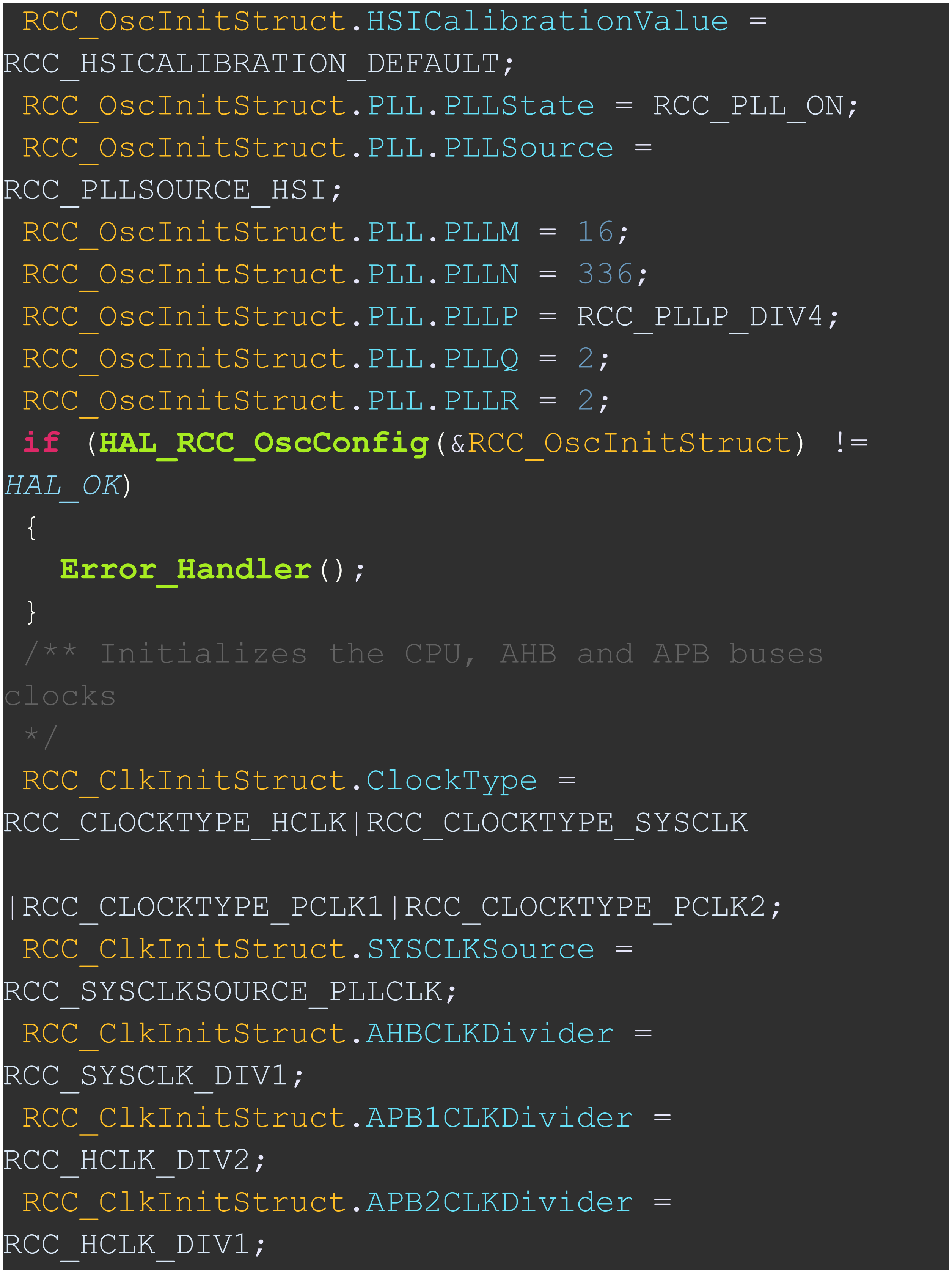


```c
#define TXBUFSIZE 1
volatile uint8_t rx_buf;
uint8_t tx_buf=0;
HAL_StatusTypeDef sig = 0x0;
void ResetI2C(I2C_HandleTypeDef* rev_i2c)
{
    HAL_I2C_DeInit(rev_i2c);
    HAL_I2C_Init(rev_i2c);
}
/* USER CODE END PV */
/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM1_Init(void);
void StartDefaultTask(void const * argument);
void IRTask_Start(void const * argument);
void I2CSendTask_Start(void const * argument);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code
-------------------------------------------------
--------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
```

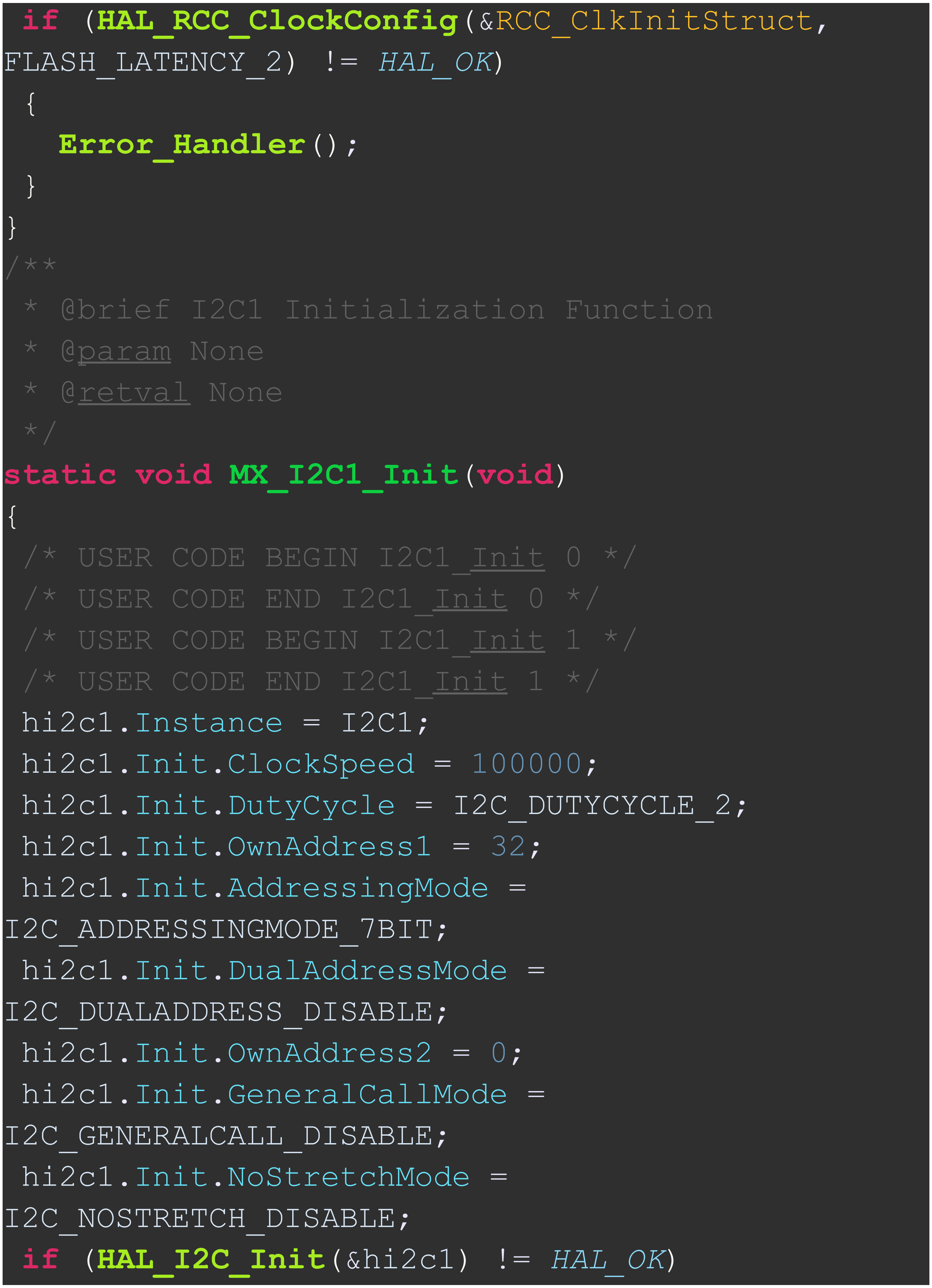
```
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */
 /* MCU Configuration--------------------------------------------------------*/
 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();
 /* USER CODE BEGIN Init */
 /* USER CODE END Init */
 /* Configure the system clock */
 SystemClock_Config();
 /* USER CODE BEGIN SysInit */
 /* USER CODE END SysInit */
 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_I2C1_Init();
 MX_TIM1_Init();
 /* USER CODE BEGIN 2 */
 /* USER CODE END 2 */
 /* Create the mutex(es) */
 /* definition and creation of I2CMutex */
 osMutexDef(I2CMutex);
 I2CMutexHandle = osMutexCreate(osMutex(I2CMutex));
 /* USER CODE BEGIN RTOS_MUTEX */
 /* add mutexes, ... */
 /* USER CODE END RTOS_MUTEX */
 /* USER CODE BEGIN RTOS_SEMAPHORES */
 /* add semaphores, ... */
 /* USER CODE END RTOS_SEMAPHORES */
 /* USER CODE BEGIN RTOS_TIMERS */
 /* start timers, add new ones, ... */
 /* USER CODE END RTOS_TIMERS */
 /* USER CODE BEGIN RTOS_QUEUES */
 /* add queues, ... */
 /* USER CODE END RTOS_QUEUES */
 /* Create the thread(s) */
 /* definition and creation of defaultTask */
 osThreadDef(defaultTask, StartDefaultTask,
osPriorityNormal, 0, 128);
 defaultTaskHandle =
osThreadCreate(osThread(defaultTask), NULL);
 /* definition and creation of IrReadTask */
 osThreadDef(IrReadTask, IRTask_Start,
osPriorityNormal, 0, 128);
 IrReadTaskHandle =
osThreadCreate(osThread(IrReadTask), NULL);
 /* definition and creation of I2CSendTask */
 osThreadDef(I2CSendTask, I2CSendTask_Start,
osPriorityNormal, 0, 128);
 I2CSendTaskHandle =
osThreadCreate(osThread(I2CSendTask), NULL);
 /* USER CODE BEGIN RTOS_THREADS */
 /* add threads, ... */
 /* USER CODE END RTOS_THREADS */
 /* Start scheduler */
 osKernelStart();
 /* We should never get here as control is now
taken by the scheduler */
 /* Infinite loop */
```



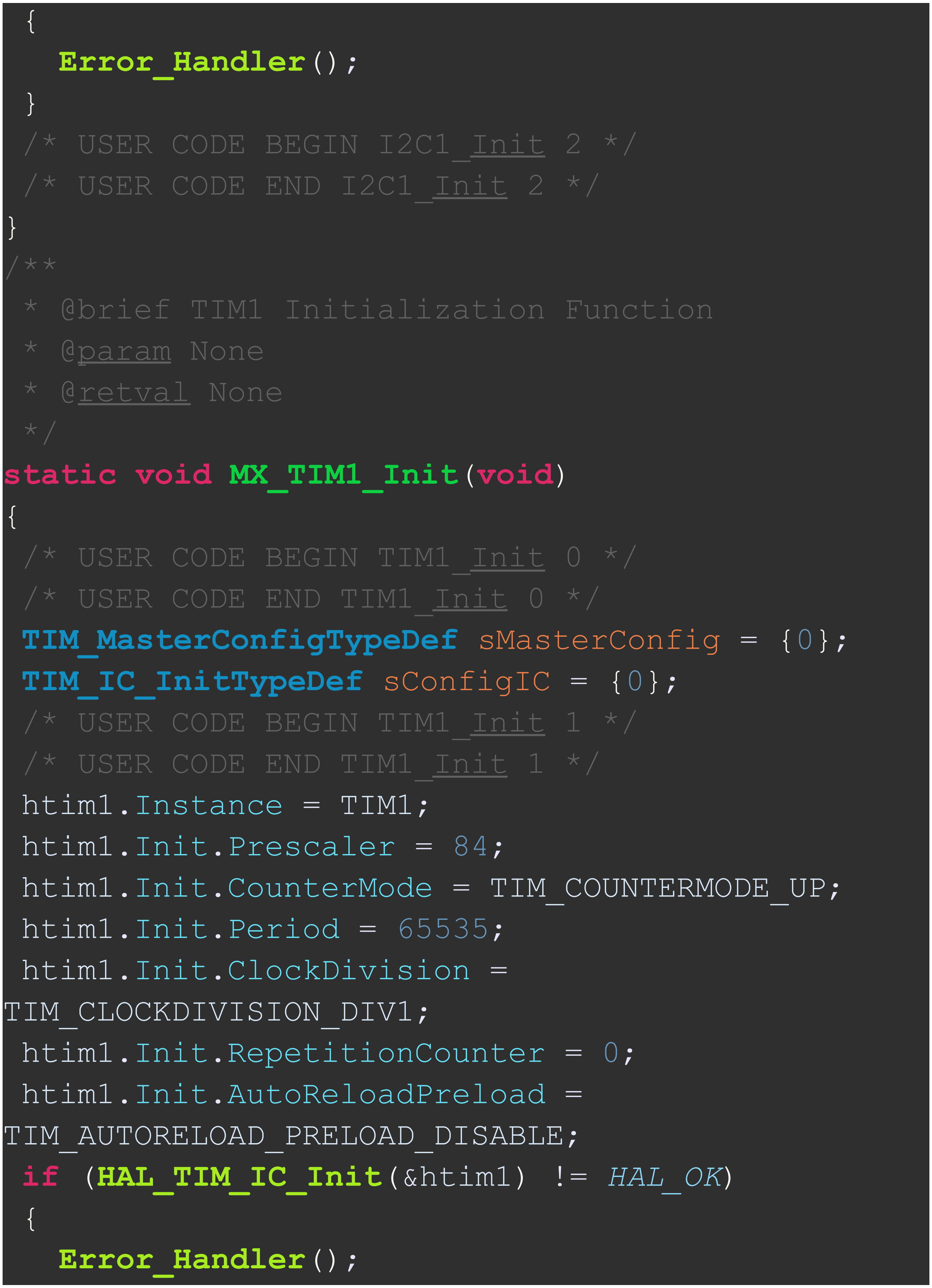
```
 /* USER CODE BEGIN WHILE */
 while (1)
 {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Configure the main internal regulator output voltage
 */
 __HAL_RCC_PWR_CLK_ENABLE();
 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```



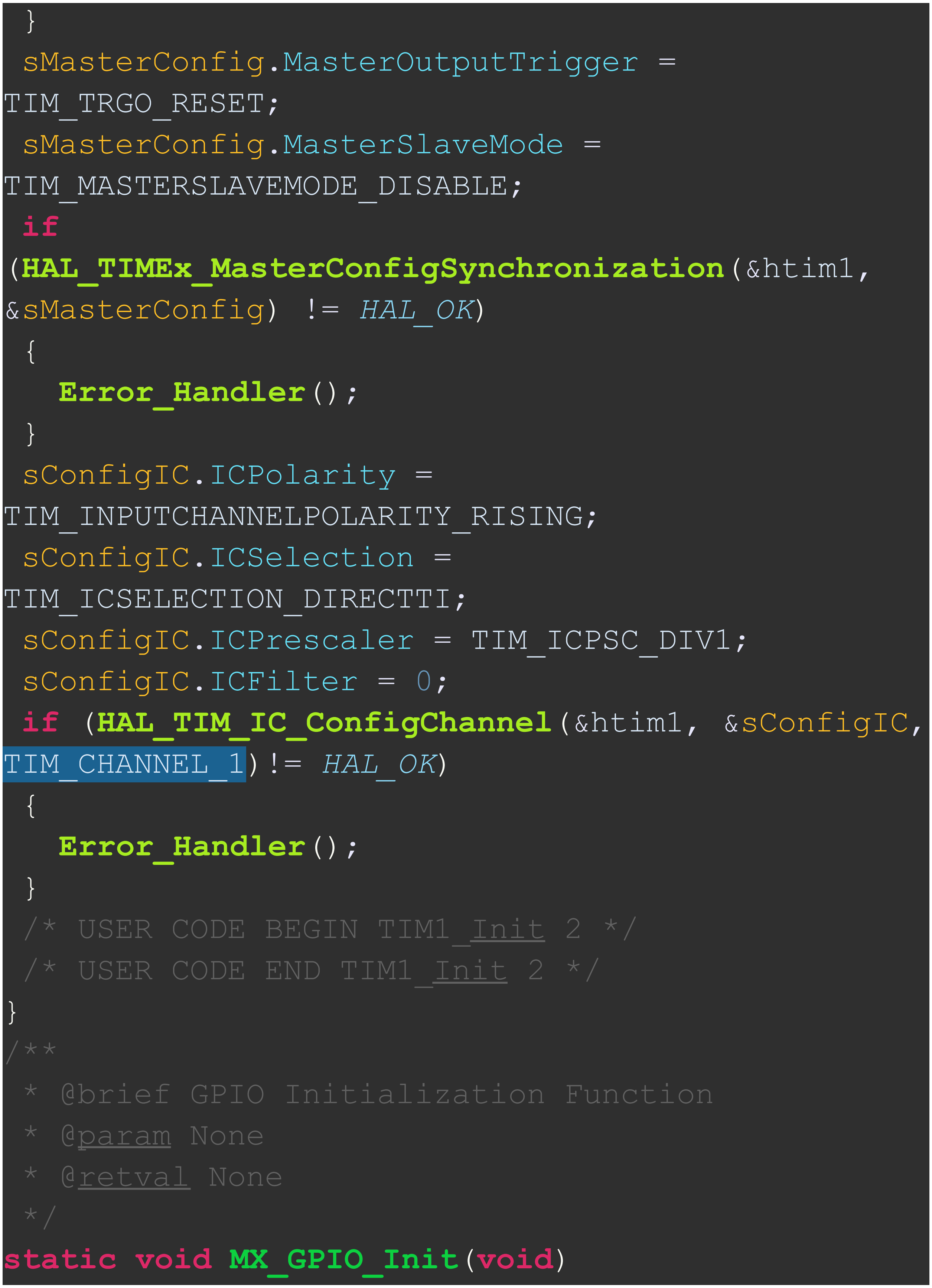
```
 RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource =
RCC_PLLSOURCE_HSI;
 RCC_OscInitStruct.PLL.PLLM = 16;
 RCC_OscInitStruct.PLL.PLLN = 336;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
 RCC_OscInitStruct.PLL.PLLQ = 2;
 RCC_OscInitStruct.PLL.PLLR = 2;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) !=
HAL_OK)
 {
   Error_Handler();
 }
 /** Initializes the CPU, AHB and APB buses
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_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider =
RCC_HCLK_DIV1;
```



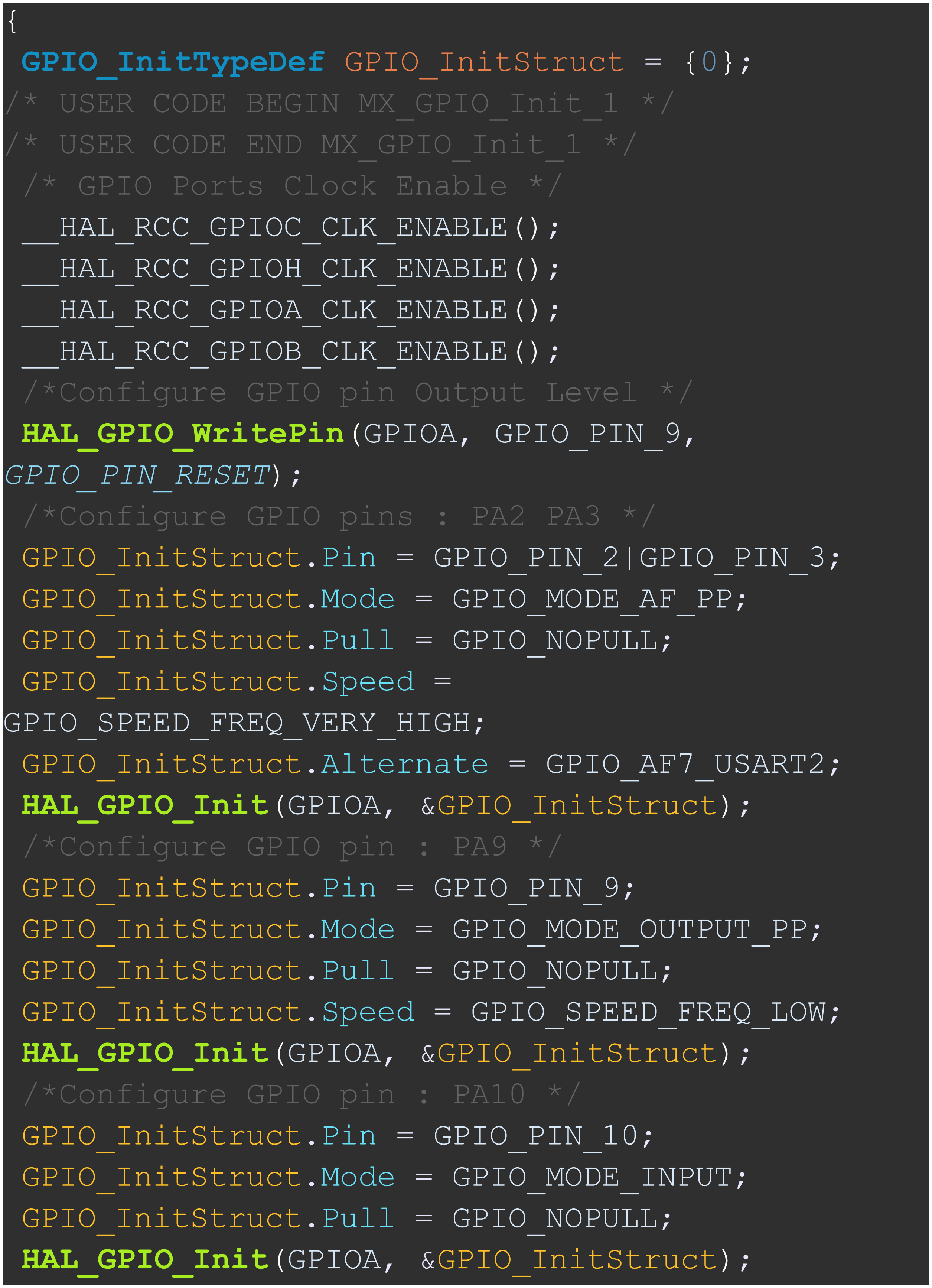
```
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void)
{
 /* USER CODE BEGIN I2C1_Init 0 */
 /* USER CODE END I2C1_Init 0 */
 /* USER CODE BEGIN I2C1_Init 1 */
 /* USER CODE END I2C1_Init 1 */
 hi2c1.Instance = I2C1;
 hi2c1.Init.ClockSpeed = 100000;
 hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
 hi2c1.Init.OwnAddress1 = 32;
 hi2c1.Init.AddressingMode =
I2C_ADDRESSINGMODE_7BIT;
 hi2c1.Init.DualAddressMode =
I2C_DUALADDRESS_DISABLE;
 hi2c1.Init.OwnAddress2 = 0;
 hi2c1.Init.GeneralCallMode =
I2C_GENERALCALL_DISABLE;
 hi2c1.Init.NoStretchMode =
I2C_NOSTRETCH_DISABLE;
 if (HAL_I2C_Init(&hi2c1) != HAL_OK)
```



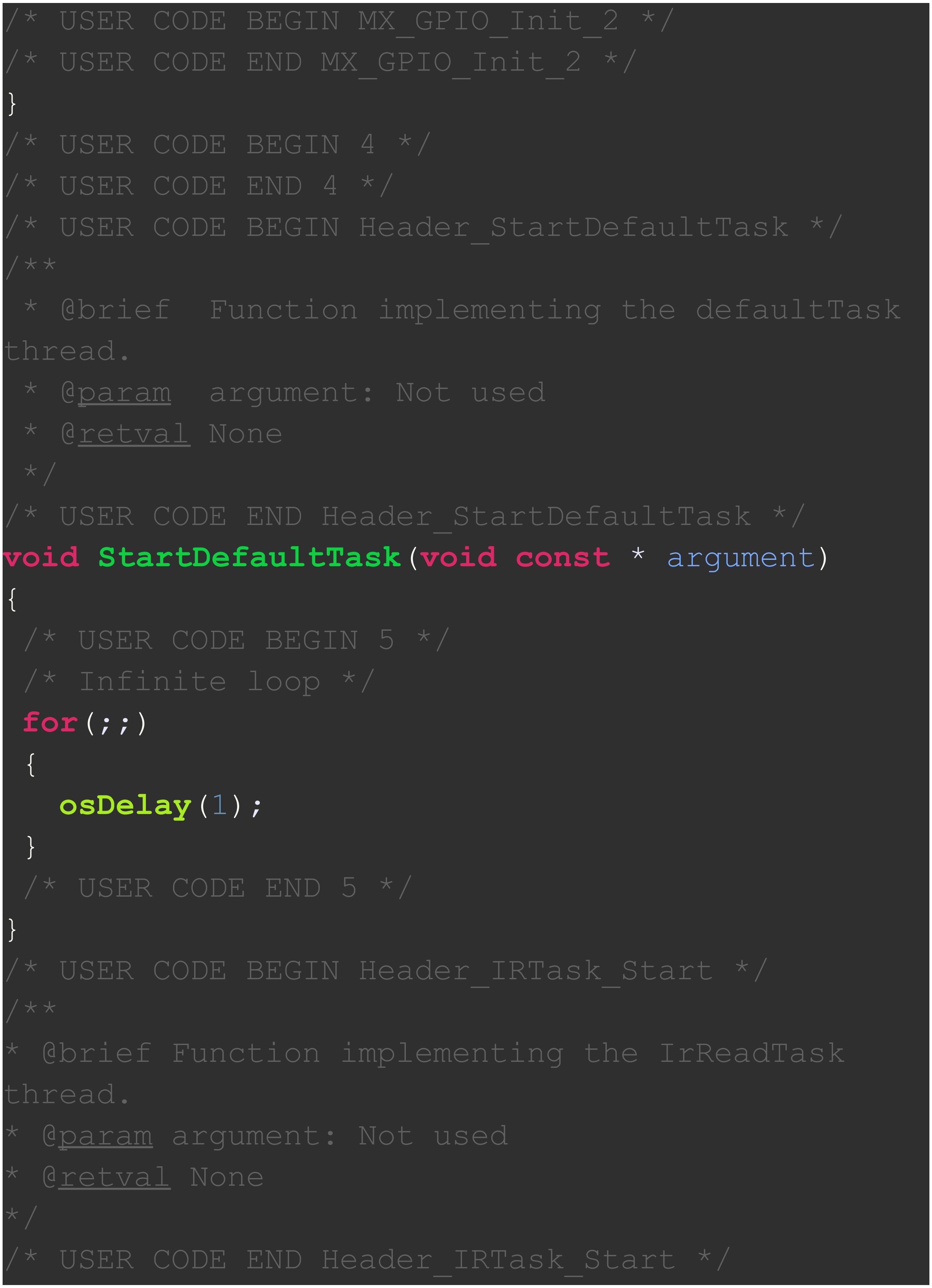
```
 {
   Error_Handler();
 }
 /* USER CODE BEGIN I2C1_Init 2 */
 /* USER CODE END I2C1_Init 2 */
}
/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{
 /* USER CODE BEGIN TIM1_Init 0 */
 /* USER CODE END TIM1_Init 0 */
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 TIM_IC_InitTypeDef sConfigIC = {0};
 /* USER CODE BEGIN TIM1_Init 1 */
 /* USER CODE END TIM1_Init 1 */
 htim1.Instance = TIM1;
 htim1.Init.Prescaler = 84;
 htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim1.Init.Period = 65535;
 htim1.Init.ClockDivision =
TIM_CLOCKDIVISION_DIV1;
 htim1.Init.RepetitionCounter = 0;
 htim1.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
 {
   Error_Handler();
```



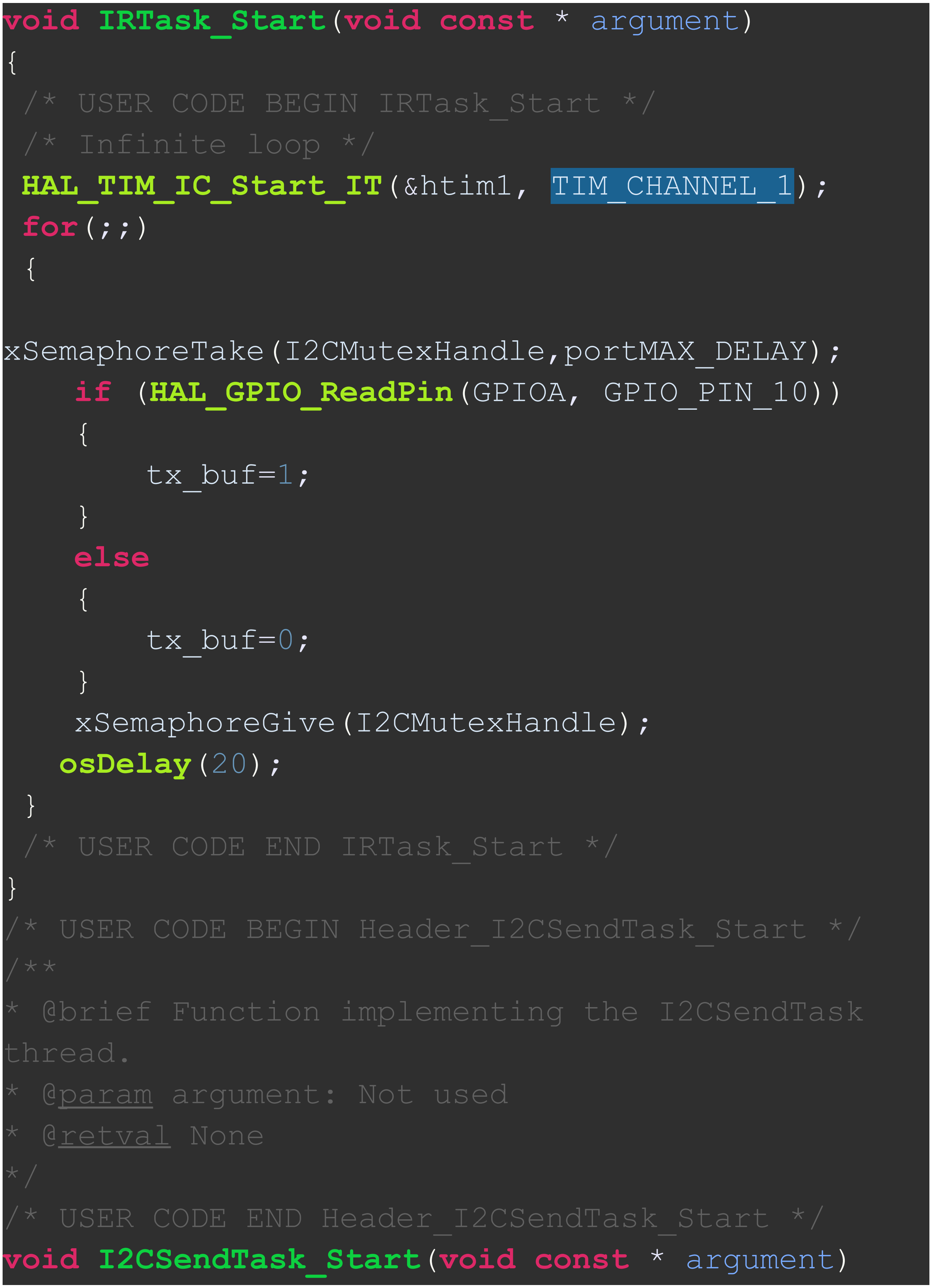
```
 }
 sMasterConfig.MasterOutputTrigger =
TIM_TRGO_RESET;
 sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
 if
(HAL_TIMEx_MasterConfigSynchronization(&htim1,
&sMasterConfig) != HAL_OK)
 {
   Error_Handler();
 }
 sConfigIC.ICPolarity =
TIM_INPUTCHANNELPOLARITY_RISING;
 sConfigIC.ICSelection =
TIM_ICSELECTION_DIRECTTI;
 sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
 sConfigIC.ICFilter = 0;
 if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC,
TIM_CHANNEL_1) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN TIM1_Init 2 */
 /* USER CODE END TIM1_Init 2 */
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
```



```
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOC_CLK_ENABLE();
 __HAL_RCC_GPIOH_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9,
GPIO_PIN_RESET);
 /*Configure GPIO pins : PA2 PA3 */
 GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed =
GPIO_SPEED_FREQ_VERY_HIGH;
 GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pin : PA9 */
 GPIO_InitStruct.Pin = GPIO_PIN_9;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pin : PA10 */
 GPIO_InitStruct.Pin = GPIO_PIN_10;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

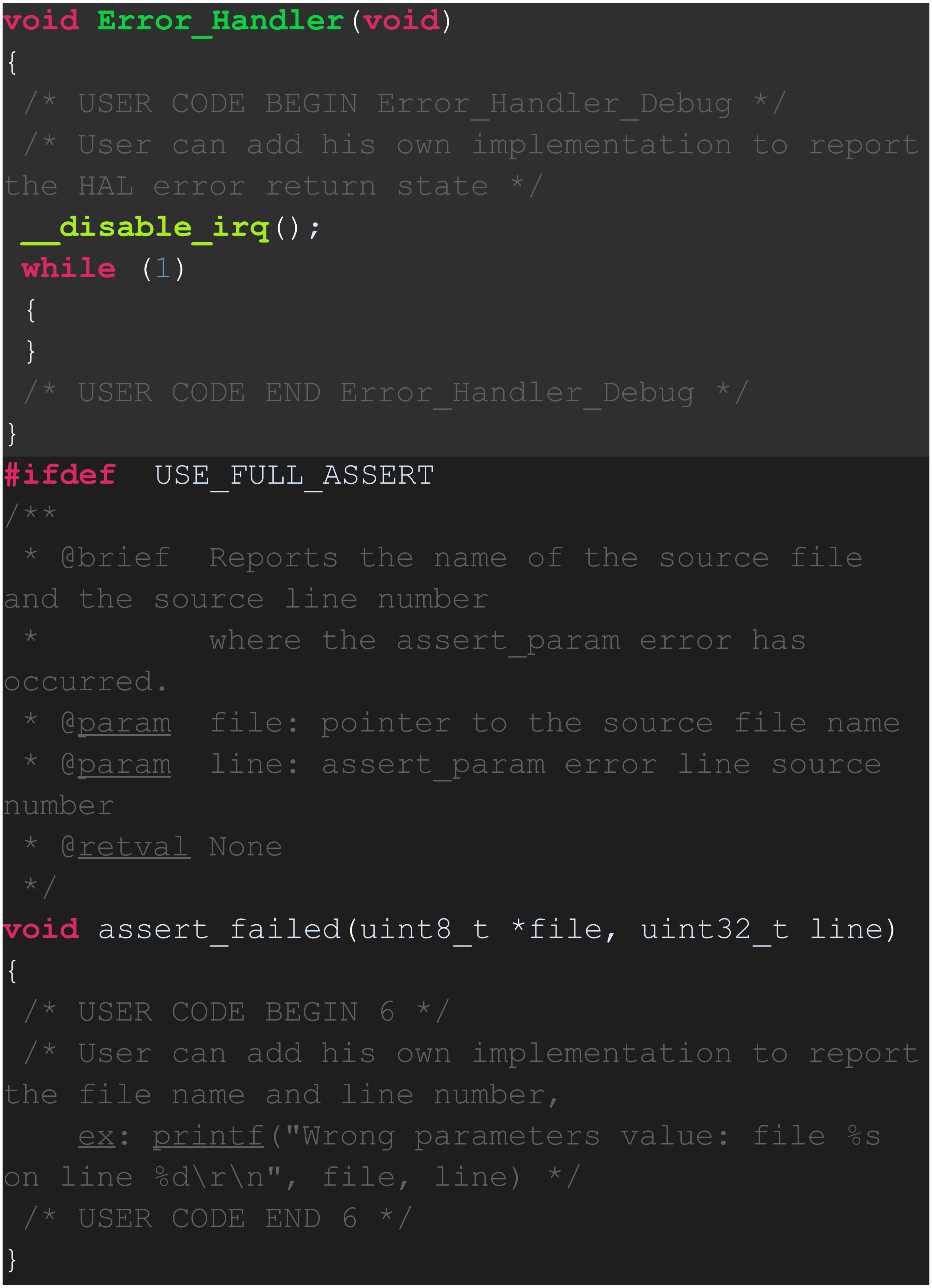


```
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/* USER CODE BEGIN Header_StartDefaultTask */
/**
 * @brief  Function implementing the defaultTask thread.
 * @param  argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
 /* USER CODE BEGIN 5 */
 /* Infinite loop */
 for(;;)
 {
   osDelay(1);
 }
 /* USER CODE END 5 */
}
/* USER CODE BEGIN Header_IRTask_Start */
/**
* @brief Function implementing the IrReadTask thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_IRTask_Start */
```



```
void IRTask_Start(void const * argument)
{
 /* USER CODE BEGIN IRTask_Start */
 /* Infinite loop */
 HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);
 for(;;)
 {

xSemaphoreTake(I2CMutexHandle,portMAX_DELAY);
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10))
    {
        tx_buf=1;
    }
    else
    {
        tx_buf=0;
    }
    xSemaphoreGive(I2CMutexHandle);
   osDelay(20);
 }
 /* USER CODE END IRTask_Start */
}
/* USER CODE BEGIN Header_I2CSendTask_Start */
/**
* @brief Function implementing the I2CSendTask
thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_I2CSendTask_Start */
void I2CSendTask_Start(void const * argument)
```

```
{
 /* USER CODE BEGIN I2CSendTask_Start */
 /* Infinite loop */
 for(;;)
 {
     if ((sig = HAL_I2C_Slave_Receive(&hi2c1,
(uint8_t*)&rx_buf, TXBUFSIZE,
0xFFFFFFF))!=HAL_OK)
     {
         ResetI2C(&hi2c1);
     }
     if ((sig == HAL_OK))
     {
         if((sig = HAL_I2C_Slave_Transmit(&hi2c1,
(uint8_t*)&tx_buf,TXBUFSIZE, 0x01))!= HAL_OK)
         {
             ResetI2C(&hi2c1);
         }
         else if(rx_buf == 0x02)
         {
             rx_buf = 0;
             xSemaphoreTake(I2CMutexHandle,
portMAX_DELAY);

HAL_I2C_Slave_Transmit(&hi2c1,(uint8_t*)&tx_buf,T
XBUFSIZE, 0x01);
             xSemaphoreGive(I2CMutexHandle);
         }
     }
   osDelay(20);
 }
 /* USER CODE END I2CSendTask_Start */
}
/**
 * @brief  Period elapsed callback in non
blocking mode
 * @note   This function is called  when TIM2
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 == TIM2) {
   HAL_IncTick();
 }
 /* USER CODE BEGIN Callback 1 */
 /* USER CODE END Callback 1 */
}
/**
 * @brief  This function is executed in case of
error occurrence.
 * @retval None
 */
```



```
void Error_Handler(void)
{
 /* USER CODE BEGIN Error_Handler_Debug */
 /* User can add his own implementation to report
the HAL error return state */
 __disable_irq();
 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,
    ex: printf("Wrong parameters value: file %s
on line %d\r\n", file, line) */
 /* USER CODE END 6 */
}
```

```
#endif /* USE_FULL_ASSERT */
```