/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2022 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 \*/

#define EVENT0 (1<<0)

#define EVENT1 (1<<1)

#define EVENT2 (1<<2)

#define EVENT\_ALLS ((EVENT2<<1) - 1)

/\* 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 ---------------------------------------------------------\*/

osThreadId Task1Handle;

osThreadId Task2Handle;

osThreadId Task3Handle;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

void StartTask02(void const \* argument);

void StartTask01(void const \* argument);

void StartTask03(void const \* argument);

/\* USER CODE BEGIN PFP \*/

EventGroupHandle\_t event\_hdl;

/\* 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();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* 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 Task1 \*/

osThreadDef(Task1, StartTask02, osPriorityLow, 0, 128);

Task1Handle = osThreadCreate(osThread(Task1), NULL);

/\* definition and creation of Task2 \*/

osThreadDef(Task2, StartTask01, osPriorityLow, 0, 128);

Task2Handle = osThreadCreate(osThread(Task2), NULL);

/\* definition and creation of Task3 \*/

osThreadDef(Task3, StartTask03, osPriorityLow, 0, 128);

Task3Handle = osThreadCreate(osThread(Task3), 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};

/\*\* 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\_DIV2;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL16;

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 GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : PA0 PA1 PA2 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2;

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);

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\* USER CODE BEGIN Header\_StartTask02 \*/

/\*\*

\* @brief Function implementing the Task1 thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_StartTask02 \*/

void StartTask02(void const \* argument)

{

/\* USER CODE BEGIN 5 \*/

/\* Infinite loop \*/

EventBits\_t even;

/\* Infinite loop \*/

for(;;)

{

even = xEventGroupWaitBits(event\_hdl,EVENT\_ALLS, pdTRUE, pdFAIL, portMAX\_DELAY);

if(even & EVENT0)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,GPIO\_PIN\_RESET);

}

}

// ham giup task sleep 1 ms

}

/\* USER CODE END 5 \*/

/\* USER CODE BEGIN Header\_StartTask01 \*/

/\*\*

\* @brief Function implementing the Task2 thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_StartTask01 \*/

void StartTask01(void const \* argument)

{

/\* USER CODE BEGIN 5 \*/

/\* Infinite loop \*/

EventBits\_t even;

/\* Infinite loop \*/

for(;;)

{

even = xEventGroupWaitBits(event\_hdl,EVENT\_ALLS, pdTRUE, pdFAIL, portMAX\_DELAY);

if(even & EVENT1)

{

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,GPIO\_PIN\_RESET);

}

}

}

/\* USER CODE BEGIN Header\_StartTask03 \*/

/\*\*

\* @brief Function implementing the Task3 thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_StartTask03 \*/

void StartTask03(void const \* argument)

{

/\* USER CODE BEGIN StartTask03 \*/

event\_hdl = xEventGroupCreate();

/\* Infinite loop \*/

for(;;)

{

xEventGroupSetBits(event\_hdl, EVENT0);

osDelay(2000);

xEventGroupSetBits(event\_hdl, EVENT1);

osDelay(2000);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,GPIO\_PIN\_SET);

osDelay(2000);

}

/\* USER CODE END StartTask03 \*/

}

/\*\*

\* @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.

\* @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 \*/