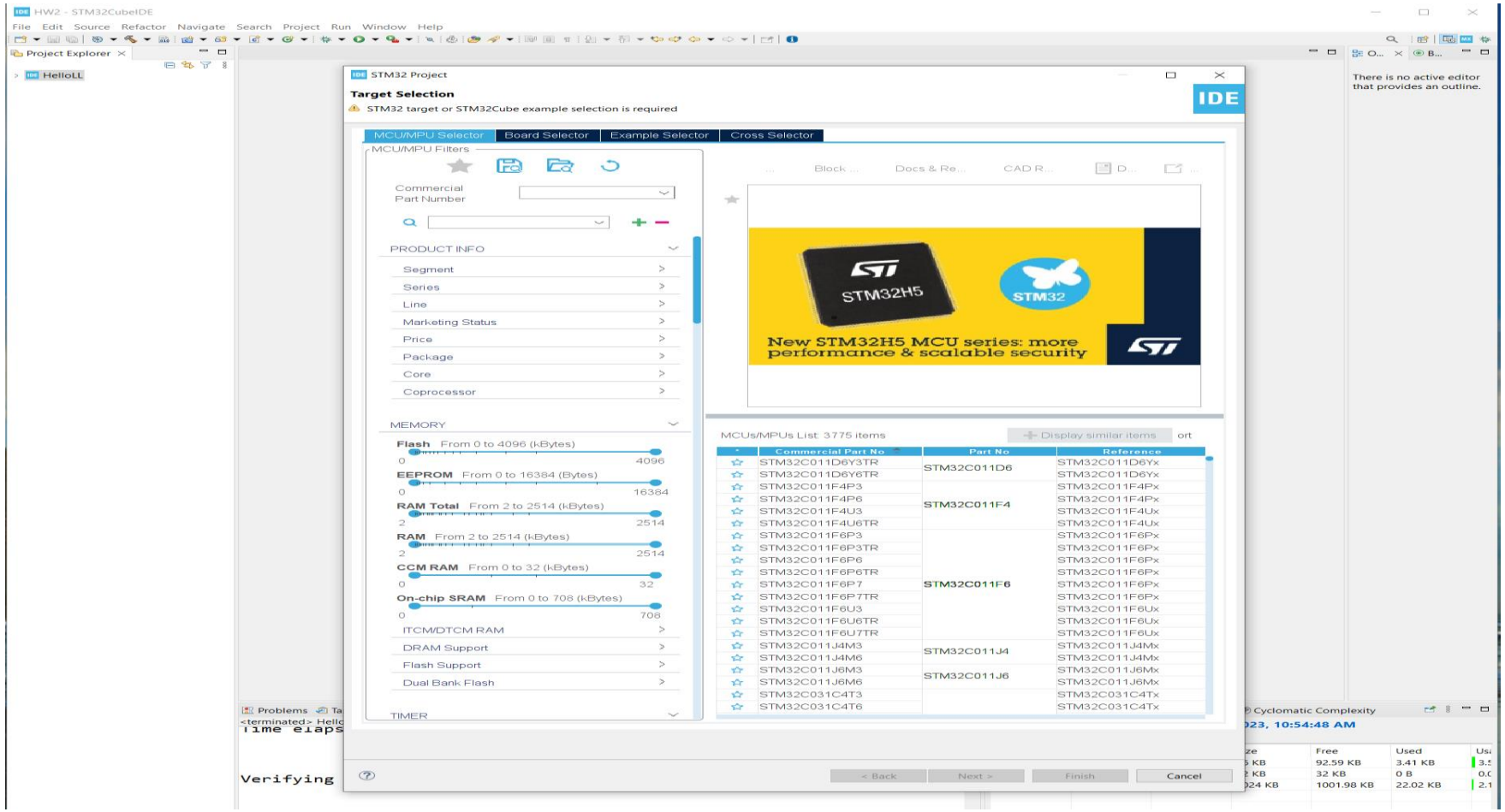
UCSD Embedded RTOS Final Assignment

By

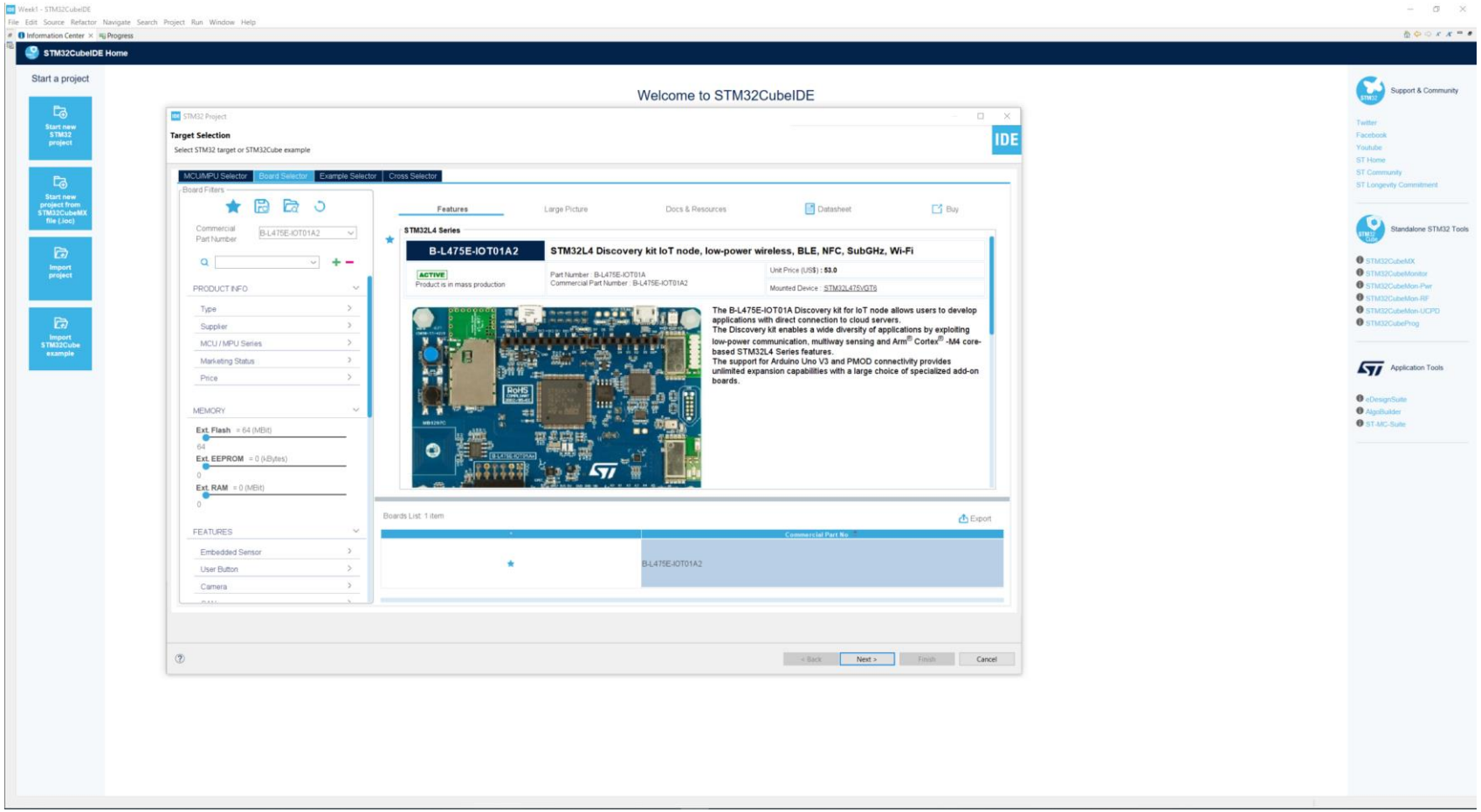
Hsuankai Chang

hsuankac@umich.edu

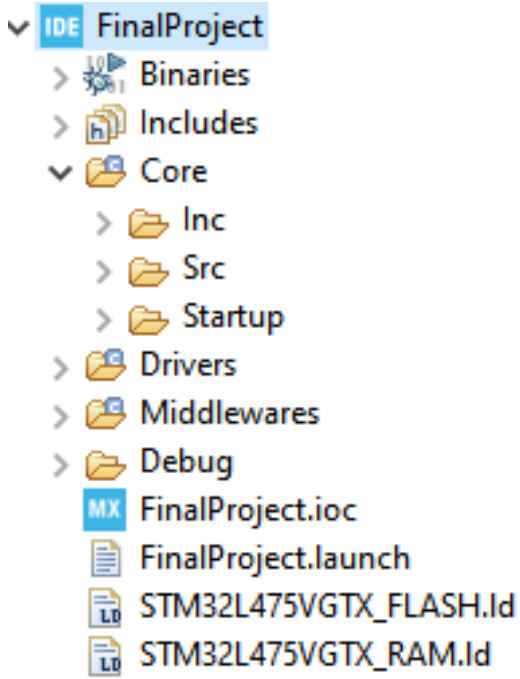
Step 1. Startup STM32CubeIDE and create new STM32 project



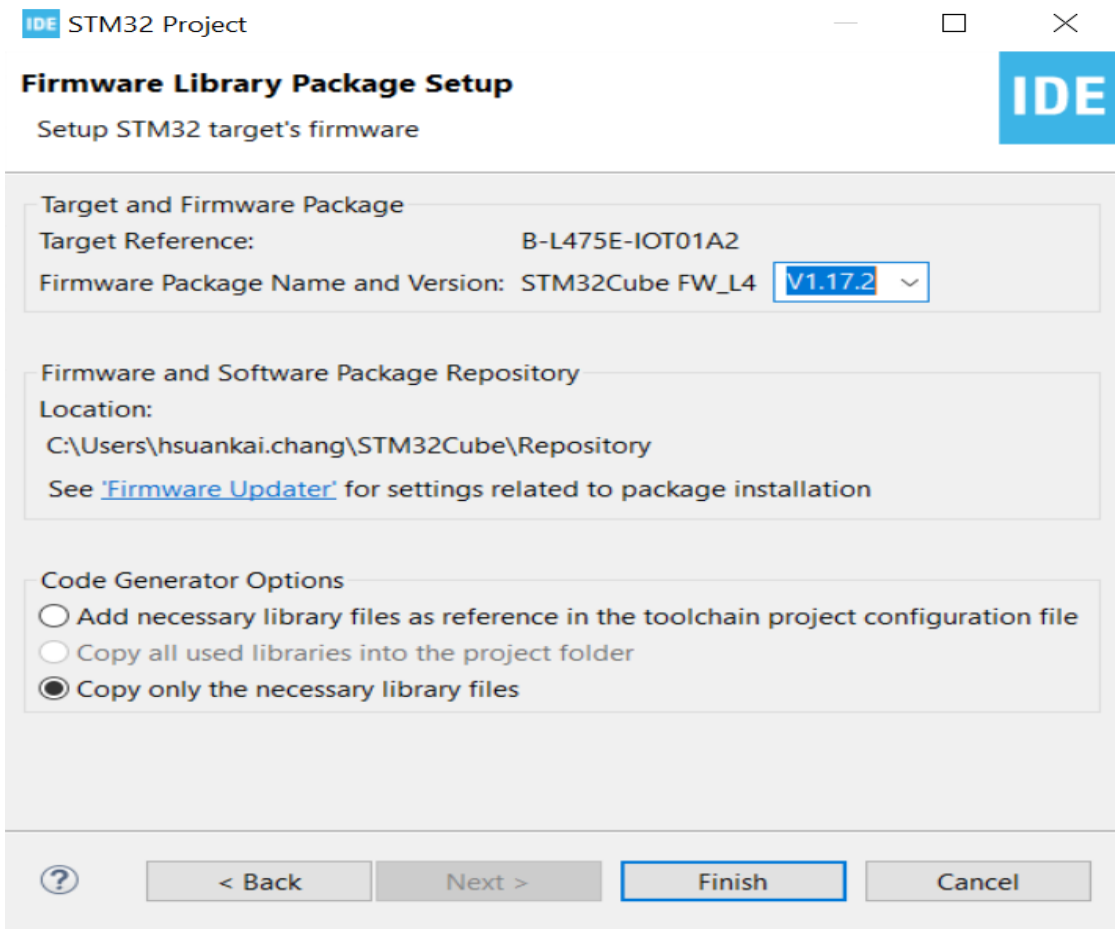
Step 2. Access board selector and type in the board you use, click Next



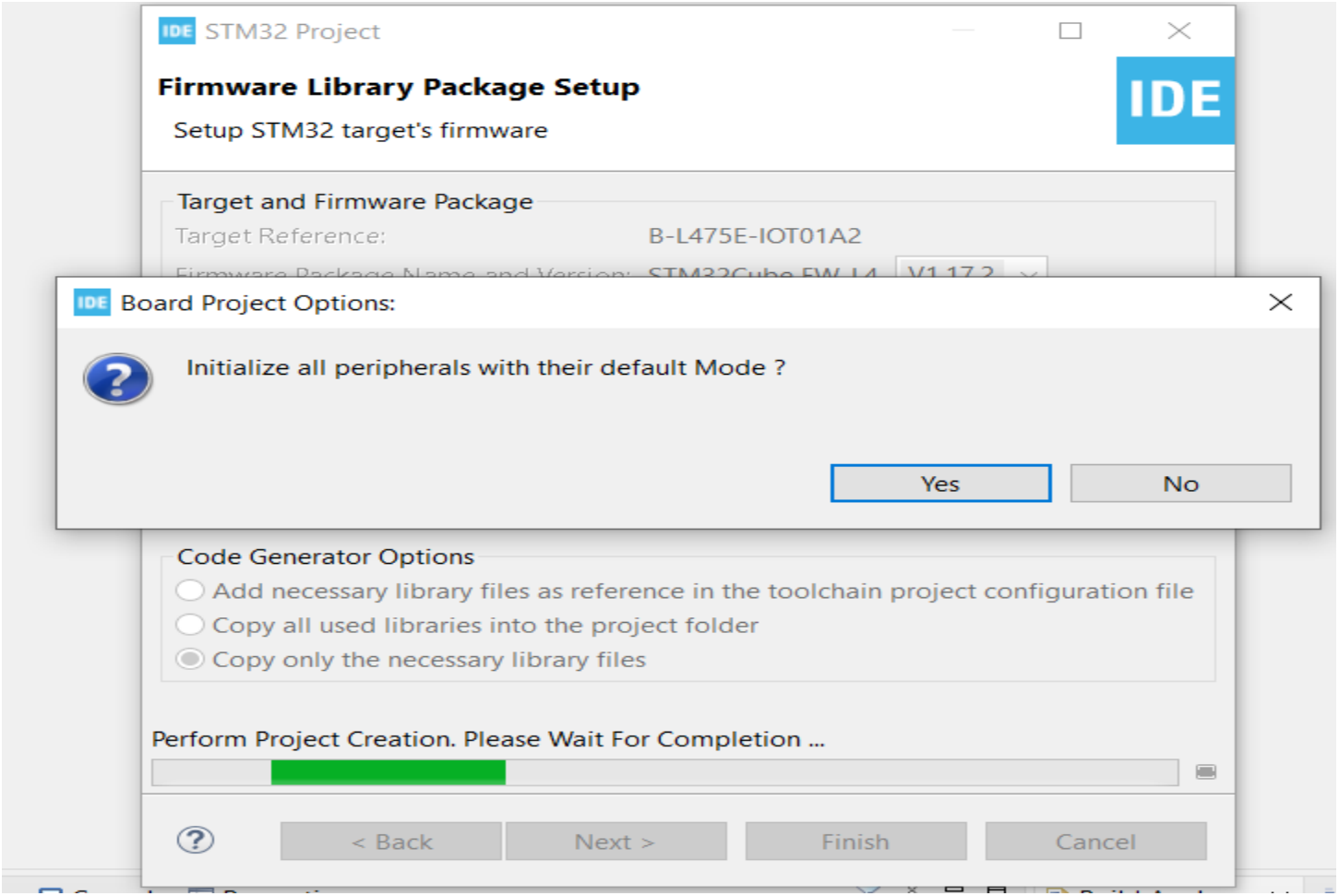
Step 3. Enter the project name then click Next



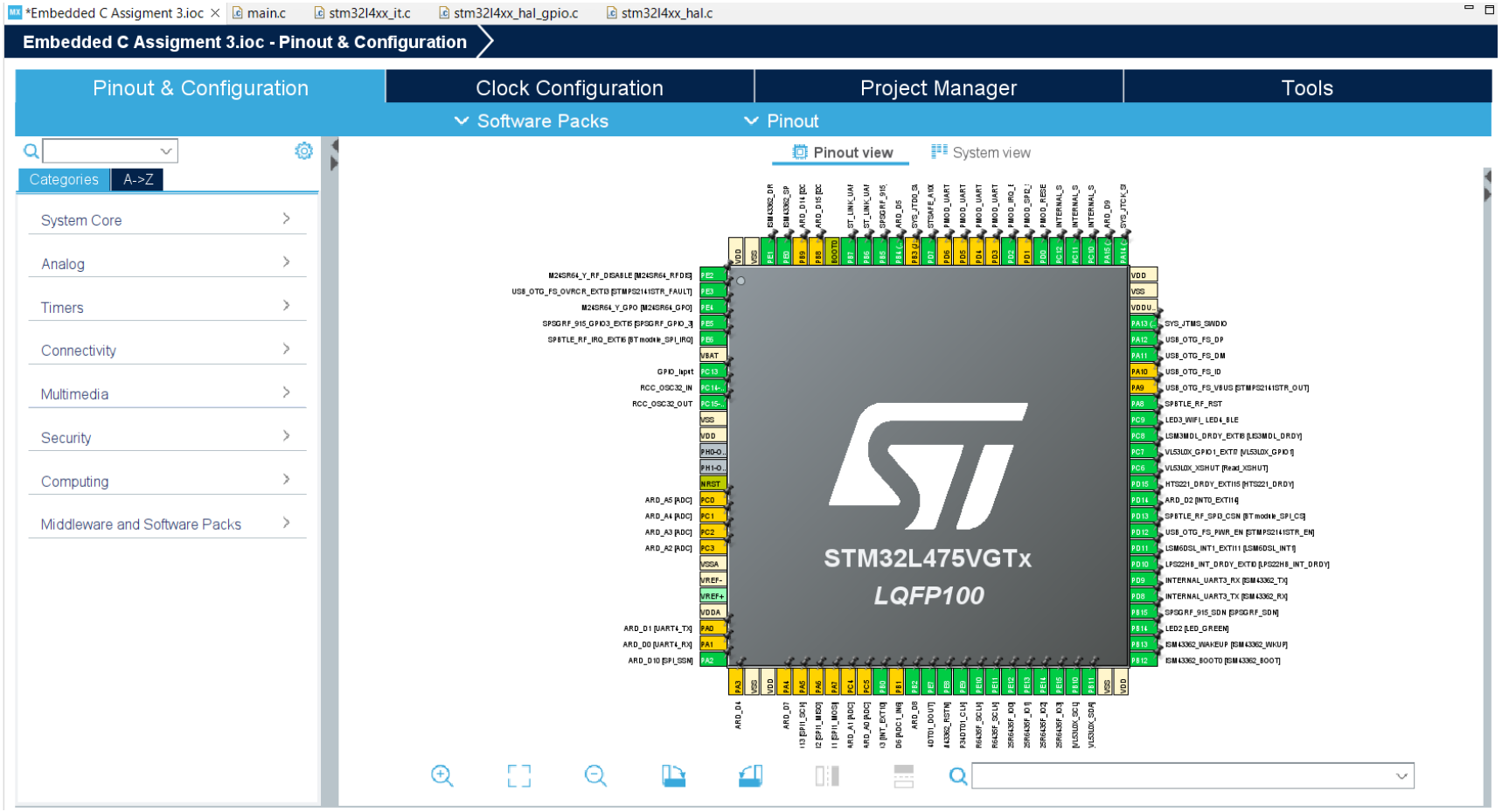
Step 4. See the firmware package name and version



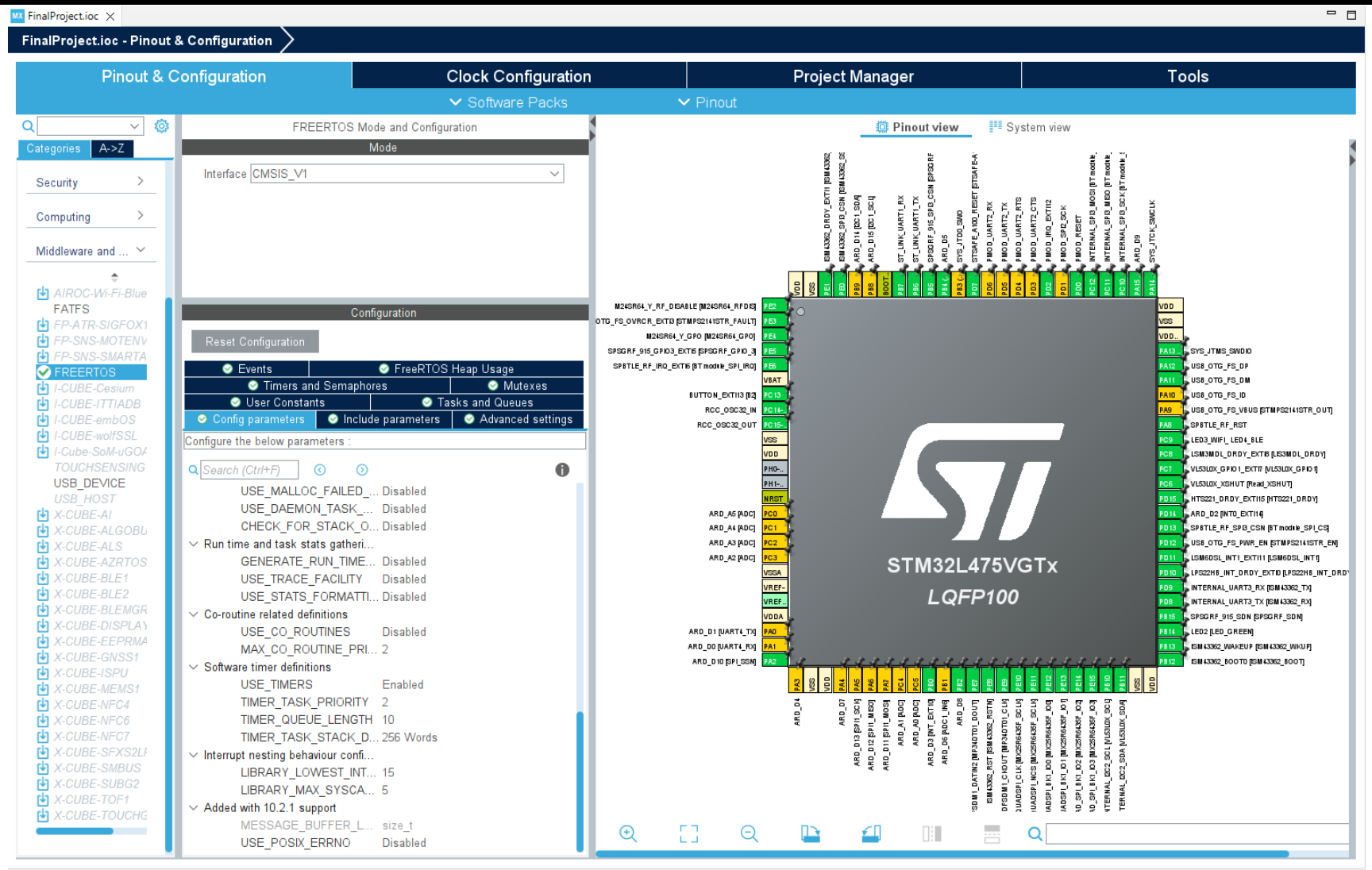
Step 5. Click yes to initialize all peripherals to default



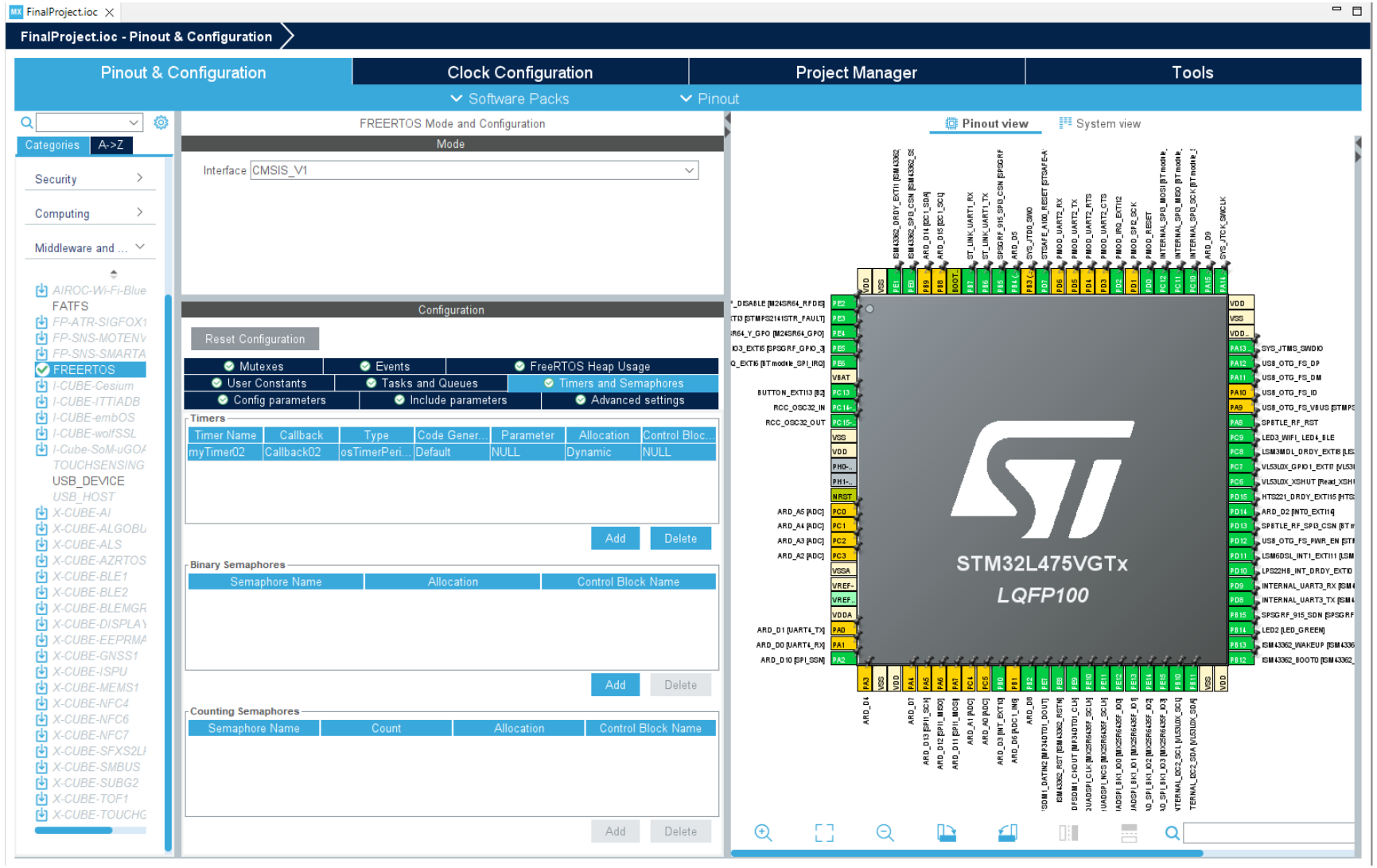
Step 6. When in .ioc file, click Pinout & Configurations.



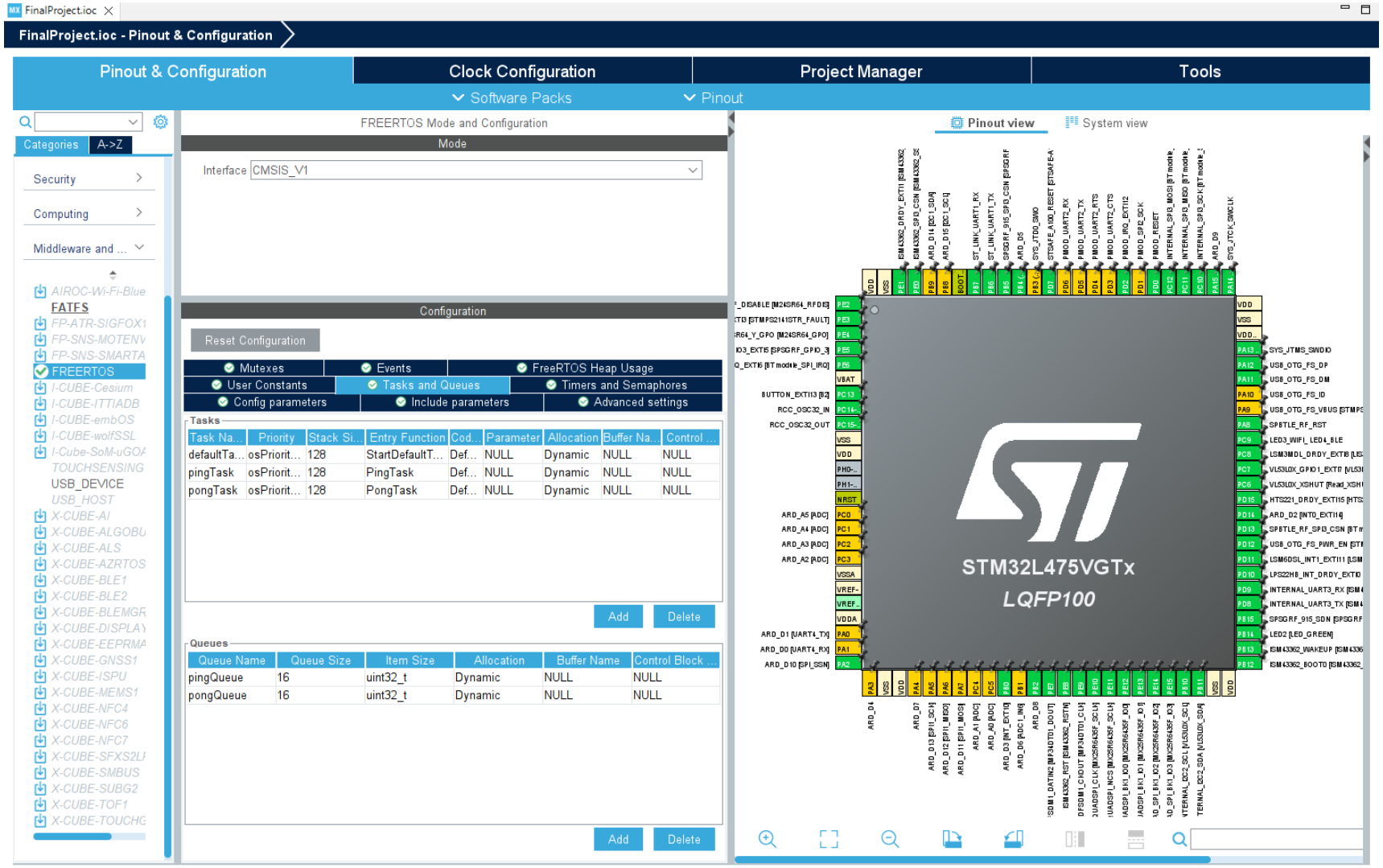
Step 7. I pick option 1 as my final project, and will demonstrate required features 1 – 4 in my following slides.
Enable CMSIS_V1 FreeRTOS, and enable software timers in the config parameters.



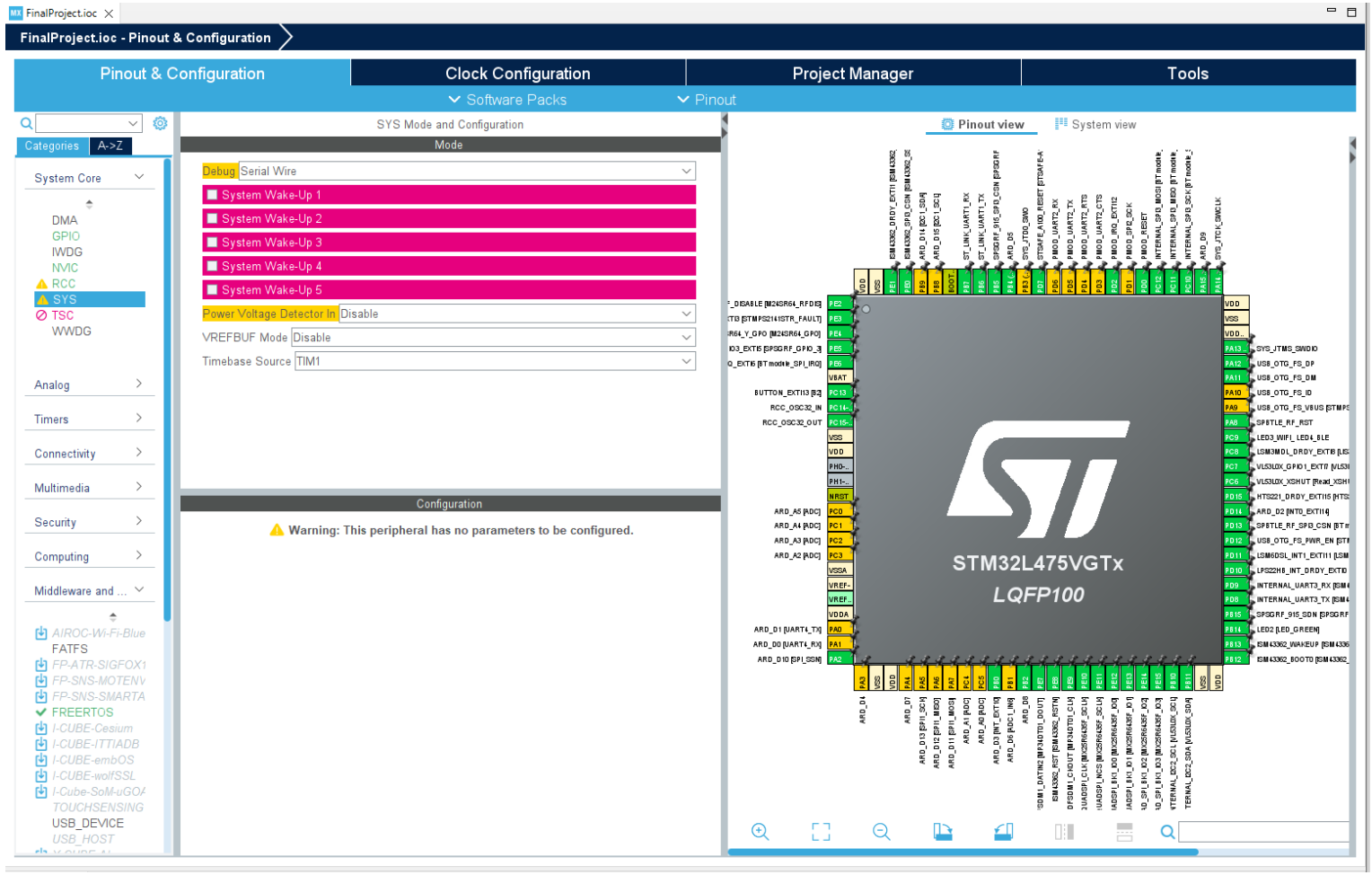
Step 8. Add one periodic timer task, for one shot timer, I manually create it using FreeRTOS API



Step 9. Besides default task, add two more tasks, one ping task and one pong task for feature 4



Step 10. Change Timebase source to use TIM1



Step 11. Code the main.c file, include more header files and TimerHandle_t variable for one shot timer

FinalProject.ioc | main.c

```c
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 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 */
#include <stdio.h>
#include <string.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
```

FinalProject.ioc | main.c

```c

I2C_HandleTypeDef hi2c2;

QSPI_HandleTypeDef hqspi;

SPI_HandleTypeDef hspi3;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart3;

PCD_HandleTypeDef hpcd_USB_OTG_FS;

osThreadId defaultTaskHandle;
osThreadId pingTaskHandle;
osThreadId pongTaskHandle;
osMessageQId pingQueueHandle;
osMessageQId pongQueueHandle;
osTimerId myTimer02Handle;
/* USER CODE BEGIN PV */
TimerHandle_t oneShotTimer;
/* USER CODE END PV */

```

Step 12. Timer, task and queue creation

FinalProject.ioc main.c

```c
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DFSDM1_Init();
MX_I2C2_Init();
MX_QUADSPI_Init();
MX_SPI3_Init();
MX_USART1_UART_Init();
MX_USART3_UART_Init();
MX_USB_OTG_FS_PCD_Init();
/* USER CODE BEGIN 2 */
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_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 */

/* Create the timer(s) */
/* definition and creation of myTimer02 */
osTimerDef(myTimer02, Callback02);
myTimer02Handle = osTimerCreate(osTimer(myTimer02), osTimerPeriodic, NULL);

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
oneShotTimer = xTimerCreate("One shot timer", pdMS_TO_TICKS(2000), pdFALSE, NULL, vTimerOneShotCallback);
osTimerStart(myTimer02Handle, 2000);
/* USER CODE END RTOS_TIMERS */

/* Create the queue(s) */
/* definition and creation of pingQueue */
osMessageQDef(pingQueue, 16, uint32_t);
pingQueueHandle = osMessageCreate(osMessageQ(pingQueue), NULL);

/* definition and creation of pongQueue */
osMessageQDef(pongQueue, 16, uint32_t);
pongQueueHandle = osMessageCreate(osMessageQ(pongQueue), NULL);

/* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
/* USER CODE END RTOS_QUEUES */
```

```c
  /* 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 pingTask */
osThreadDef(pingTask, PingTask, osPriorityIdle, 0, 128);
pingTaskHandle = osThreadCreate(osThread(pingTask), NULL);

/* definition and creation of pongTask */
osThreadDef(pongTask, PongTask, osPriorityIdle, 0, 128);
pongTaskHandle = osThreadCreate(osThread(pongTask), 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 */
}
```

Step 13. Feature 1, flashing LED2

```
747 void StartDefaultTask(void const * argument)
748 {
749   /* USER CODE BEGIN 5 */
750     /* Infinite loop */
751     for (;;) {
752         HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
753         osDelay(2000);
754     }
755   /* USER CODE END 5 */
756 }
757
```

Step 14. Feature 2, periodic timer and one-shot timer creation, periodic timer start, and periodic timer callback function

```c
/* Create the timer(s) */
/* definition and creation of myTimer02 */
osTimerDef(myTimer02, Callback02);
myTimer02Handle = osTimerCreate(osTimer(myTimer02), osTimerPeriodic, NULL);

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
oneShotTimer = xTimerCreate("One shot timer", pdMS_TO_TICKS(2000), pdFALSE, NULL, vTimerOneShotCallback);
osTimerStart(myTimer02Handle, 2000);
/* USER CODE END RTOS_TIMERS */
```

```c
/* Callback02 function */
void Callback02(void const * argument)
{
  /* USER CODE BEGIN Callback02 */
    static int countAutoReload = 1;
    char buf[100];
    snprintf(buf, sizeof(buf), "autoReloadTimer triggered: %d\r\n", countAutoReload);
    HAL_UART_Transmit(&huart1, (uint8_t*) buf, strlen(buf), 1000);
    // ARD_D5
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_4);
    countAutoReload++;
  /* USER CODE END Callback02 */
}
```

Step 15. Feature 3, one shot timer code. The reason why I did not use CMSIS API to create one shot timer is, I find out osTimerStart can not be used in ISR, so change to use FreeRTOS API

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == BUTTON_EXTI13_Pin)
    {
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;
        if(xTimerStartFromISR(oneShotTimer, &xHigherPriorityTaskWoken) != pdPASS)
        {
            char buf[100];
            snprintf(buf, sizeof(buf), "oneShotTimer start failed\r\n");
            HAL_UART_Transmit(&huart1, (uint8_t*) buf, strlen(buf), 1000);
        }

        if( xHigherPriorityTaskWoken != pdFALSE )
        {
            portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
        }
    }
}

void vTimerOneShotCallback( TimerHandle_t xTimer )
{
    static int countOneShot = 1;
    char buf[100];
    snprintf(buf, sizeof(buf), "oneShotTimer triggered: %d\r\n", countOneShot);
    HAL_UART_Transmit(&huart1, (uint8_t*) buf, strlen(buf), 1000);
    // ARD_D8
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_2);
    countOneShot++;
}
```

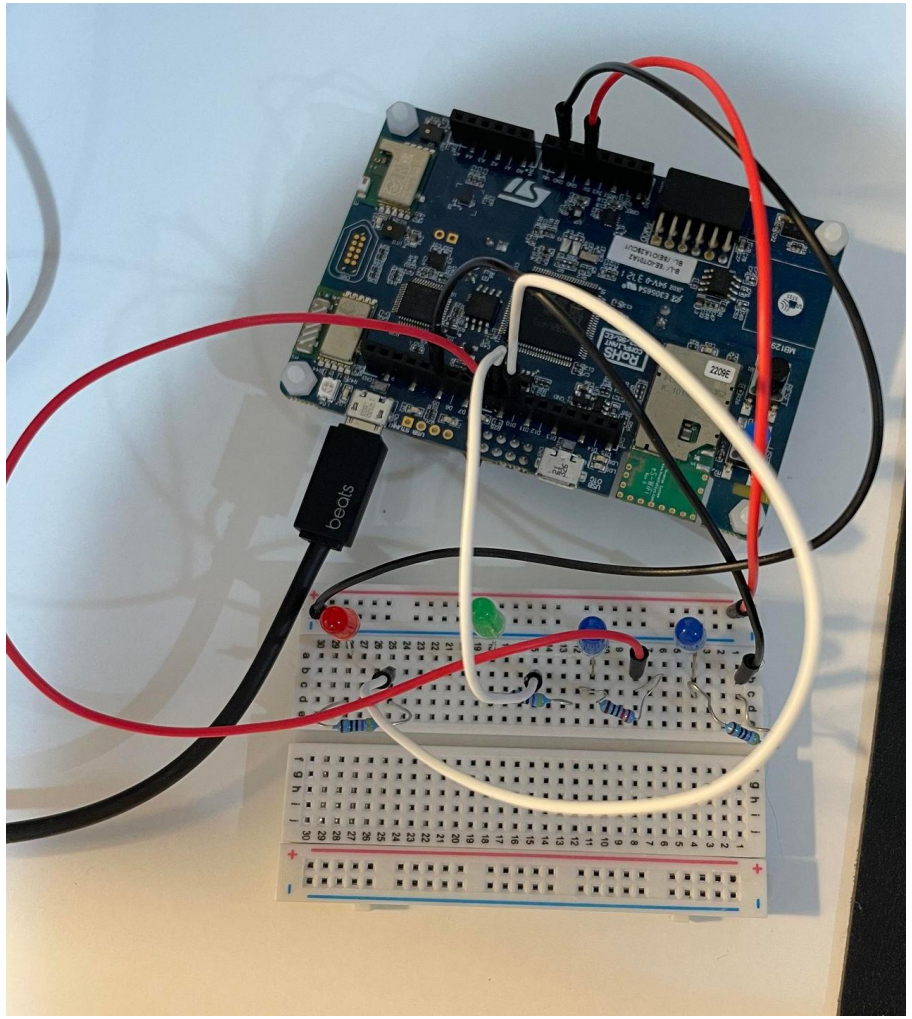
Step 15. Feature 4, ping pong task

```c
osMessageQDef(pingQueue, 16, uint32_t);
pingQueueHandle = osMessageCreate(osMessageQ(pingQueue), NULL);

/* definition and creation of pongQueue */
osMessageQDef(pongQueue, 16, uint32_t);
pongQueueHandle = osMessageCreate(osMessageQ(pongQueue), NULL);

```

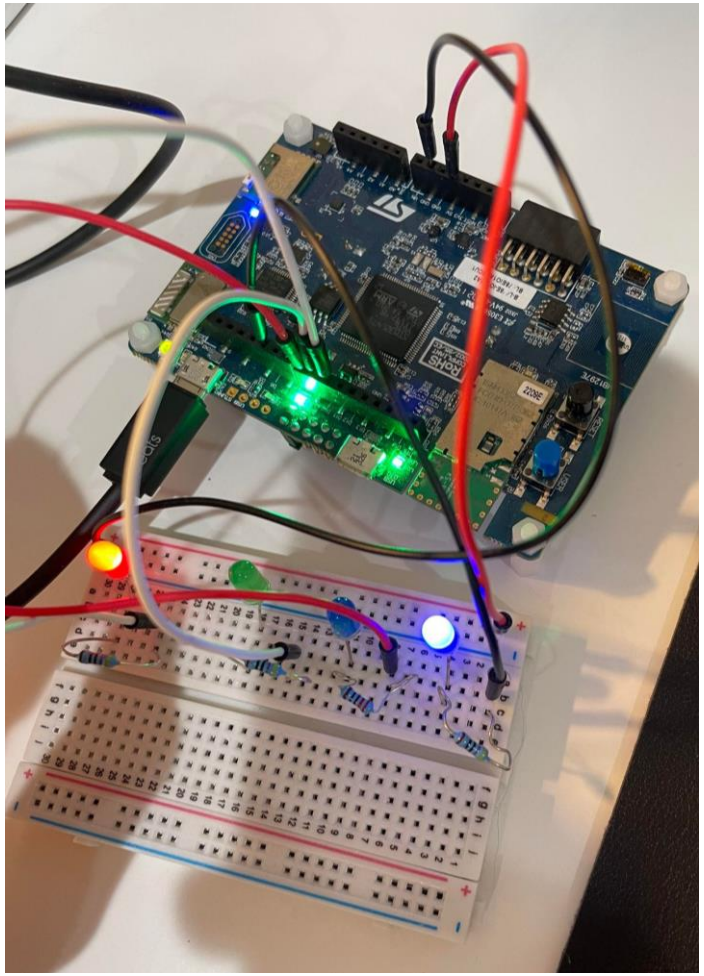
```c
void PingTask(void const * argument)
{
  /* USER CODE BEGIN PingTask */
  static int pingcmd = 0;
  osEvent pingEvt;
  char buf[100];
  /* Infinite loop */
  for(;;)
  {
      osMessagePut(pingQueueHandle, pingcmd, osWaitForever);
      pingcmd++;
      pingEvt = osMessageGet(pongQueueHandle, osWaitForever);
      if(pingEvt.status == osEventMessage){
          snprintf(buf, sizeof(buf), "pong command count: %d\r\n", (int)(pingEvt.value.p));
          HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
          // ARD_D10
          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_15);
      }
      osDelay(2000);
  }
  /* USER CODE END PingTask */
}
```

```c
/* USER CODE BEGIN Header_PongTask */
/**
* @brief Function implementing the pongTask thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_PongTask */
void PongTask(void const * argument)
{
  /* USER CODE BEGIN PongTask */
  static int pongcmd = 0;
  osEvent pongEvt;
  char buf[100];
  /* Infinite loop */
  for(;;)
  {
      pongEvt = osMessageGet(pingQueueHandle, osWaitForever);
      if(pongEvt.status == osEventMessage){
          snprintf(buf, sizeof(buf), "ping command count: %d\r\n", (int)(pongEvt.value.p));
          HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
          // ARD_D9
          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
          osMessagePut(pongQueueHandle, pongcmd, osWaitForever);
          pongcmd++;
      }
      osDelay(2000);
  }
  /* USER CODE END PongTask */
}
```

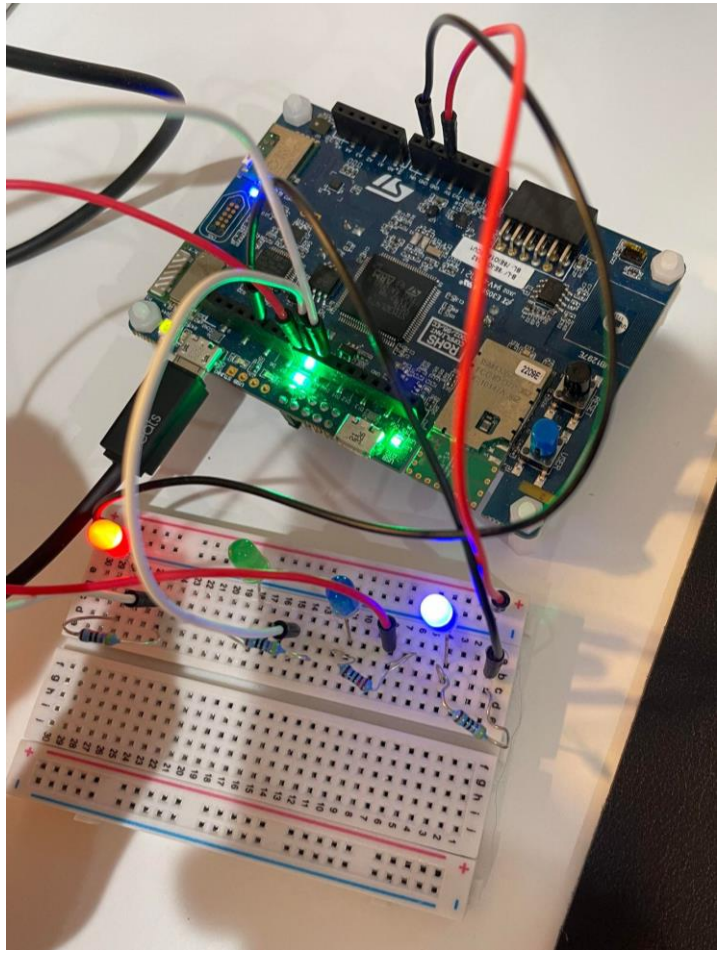
Step 16. Connect the hardware, from right to left is ARD_D5, ARD_D8, ARD_D9, ARD_D10

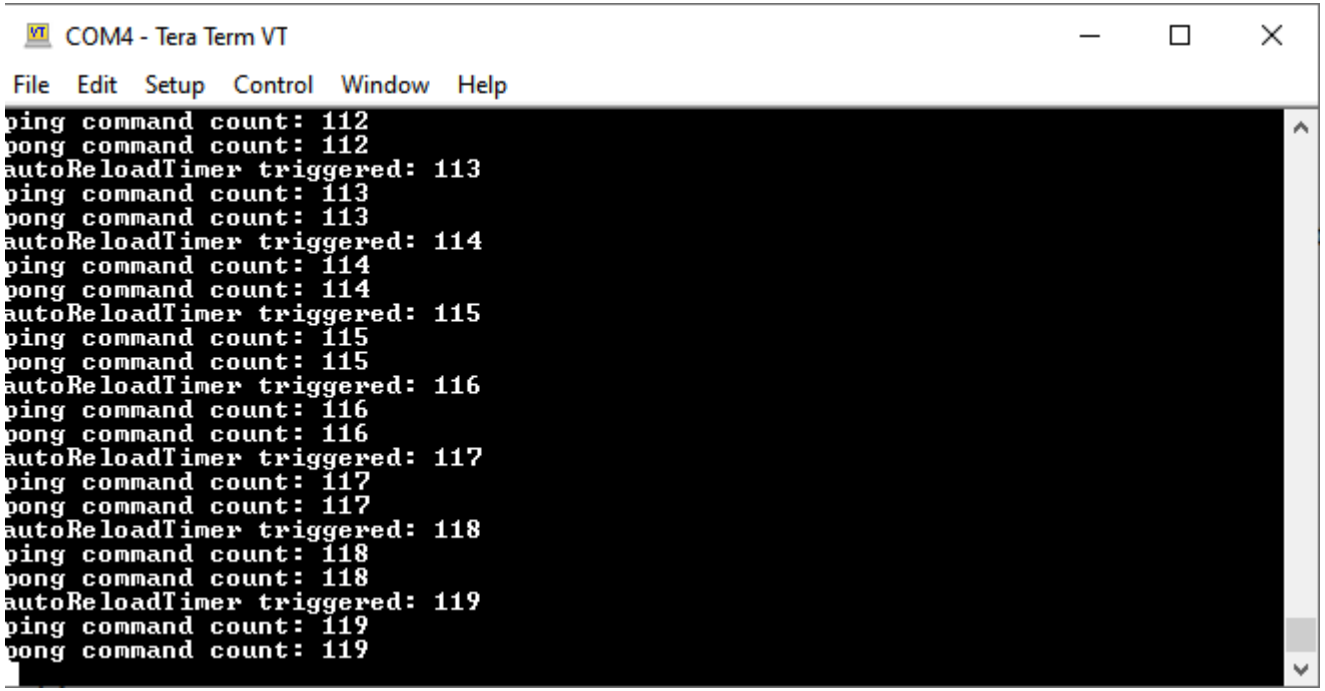


Step 17. Feature 1 test successful, LED2 pin flash at a rate of 2 seconds

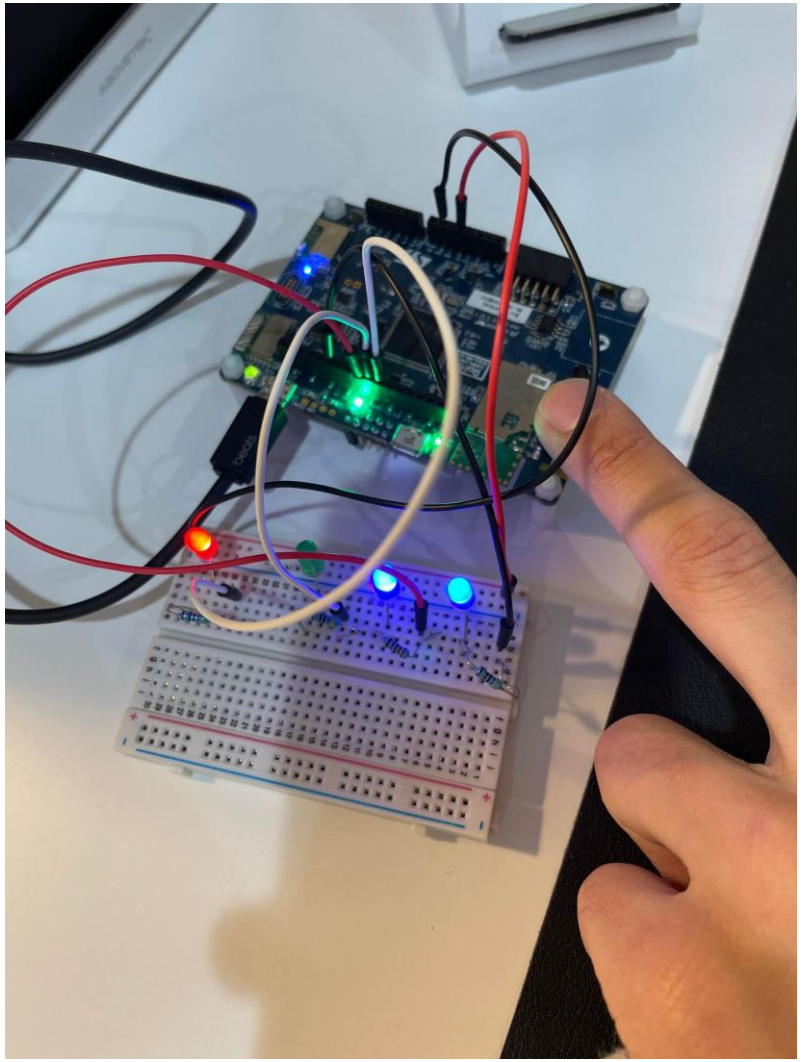


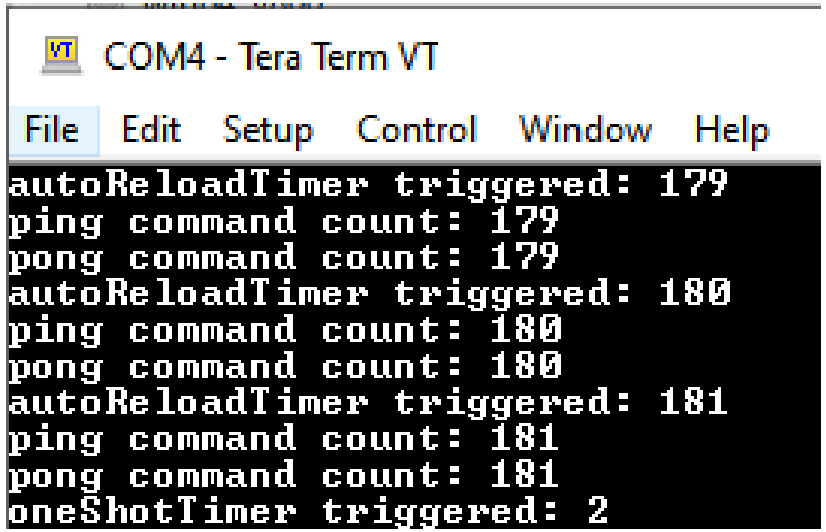
Step 18. Feature 2 test successful, ARD_D5 LED flash at a rate of 2 seconds, timer call back function triggered



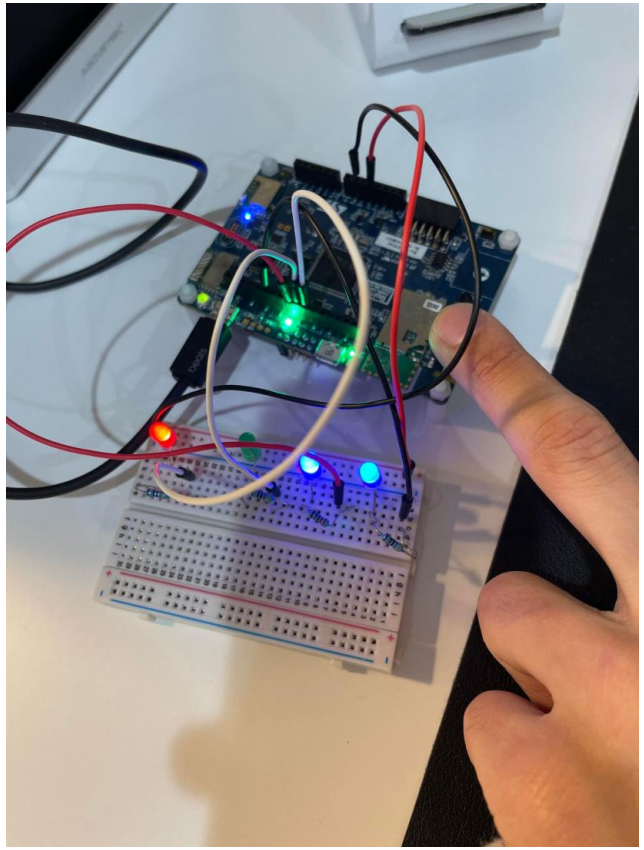


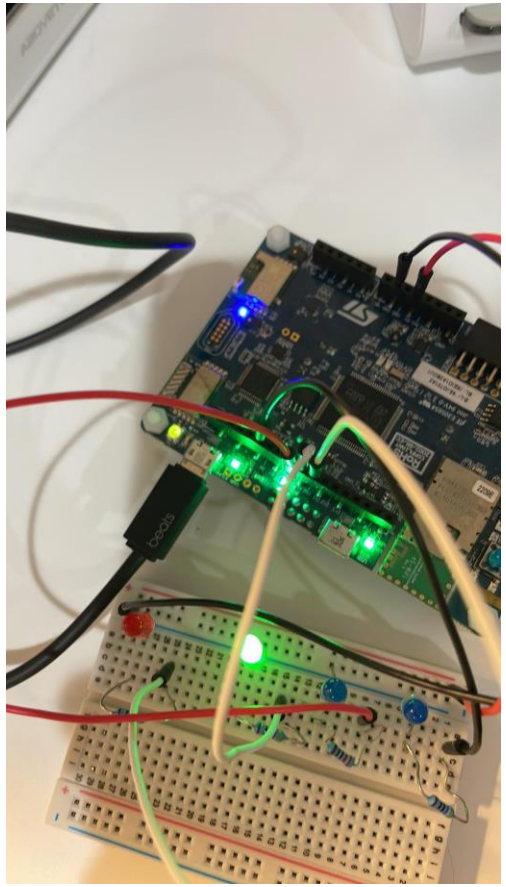
Step 19. one shot timer test successful, call back function triggered, ARD_D8 LED flash with time out 2 seconds

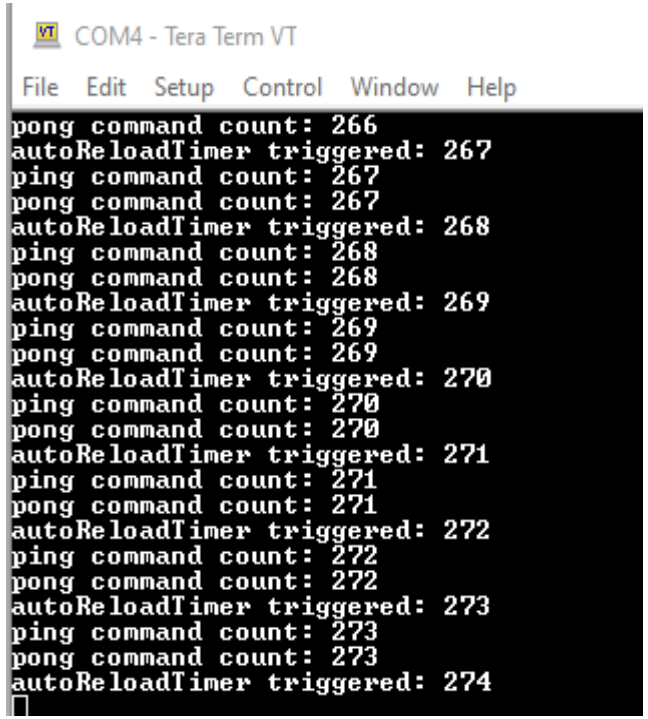




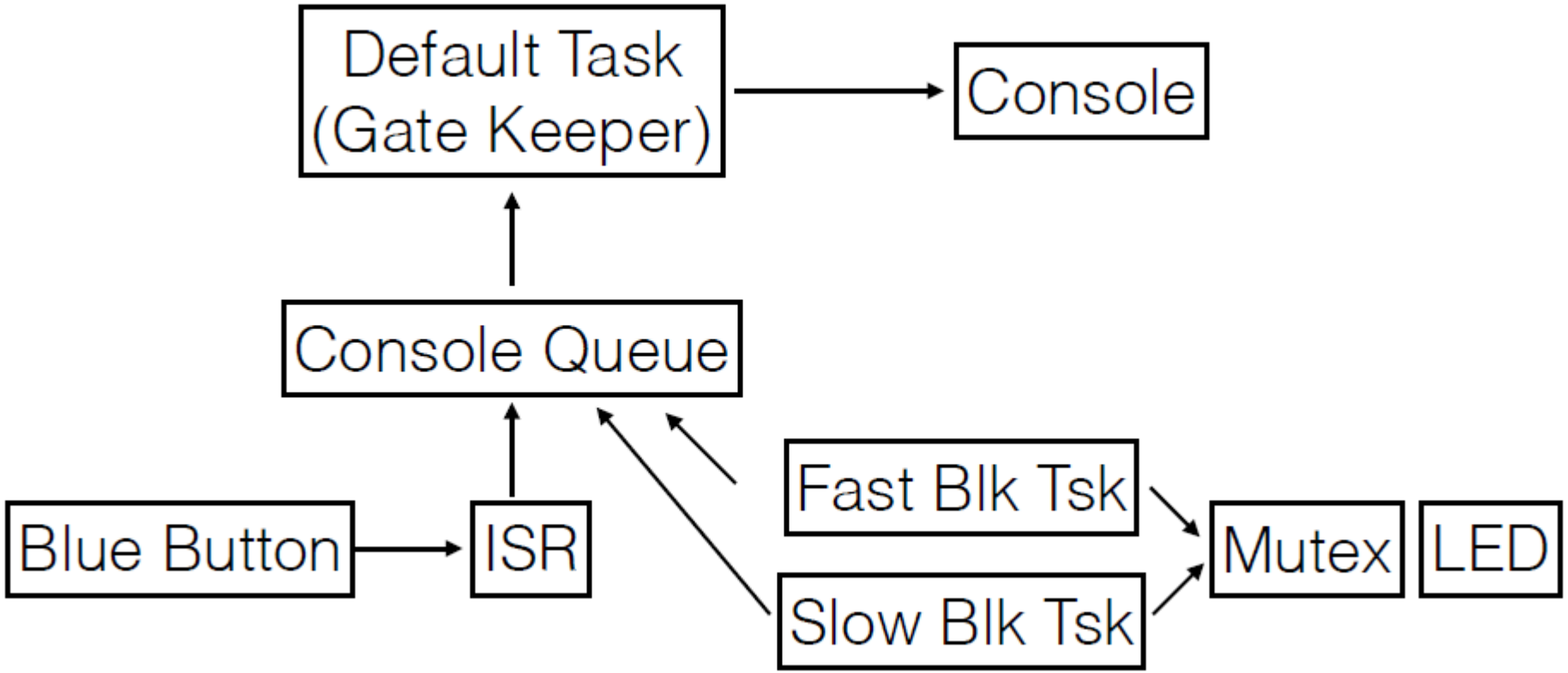
Step 20. Feature 4 test successful, ping pong task send command to queue to toggle LED (ARD_D9, ARD_D10 pin)



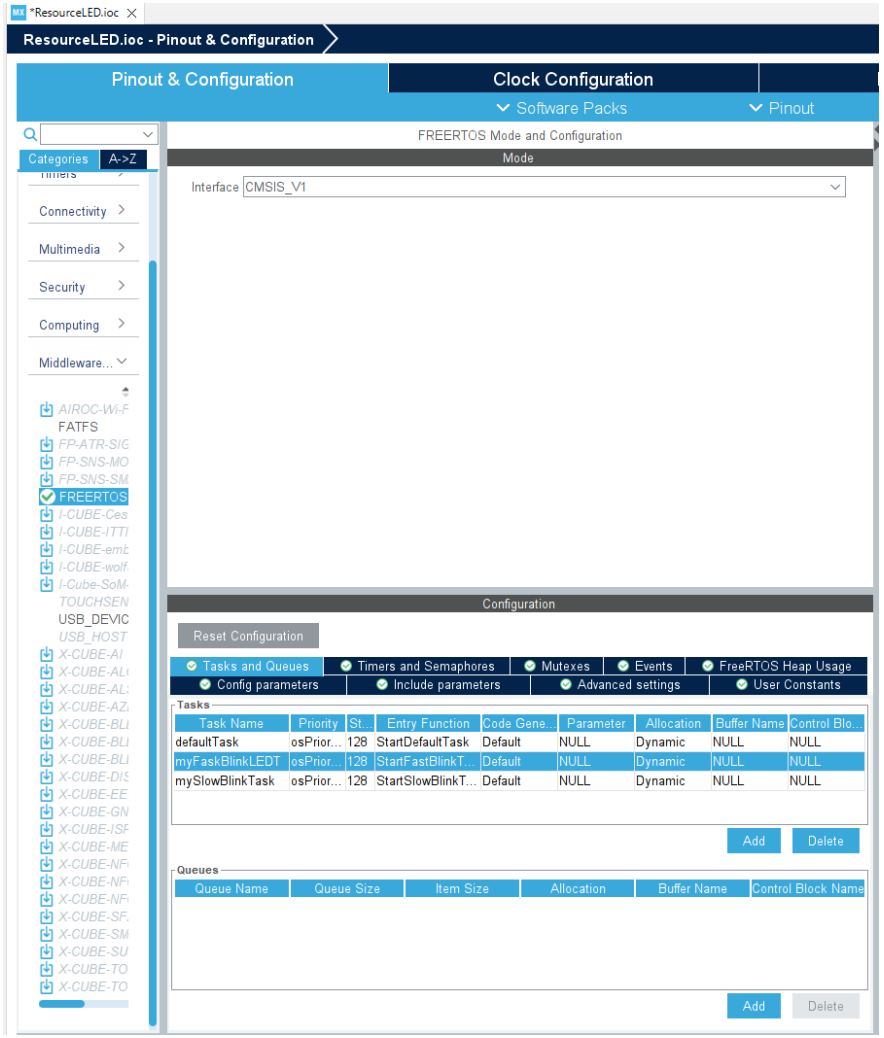




Step 20. Resource LED bonus project. I will create a new project and follow the code demo like below



Step 21. Add two more tasks, fast and slow blink tasks



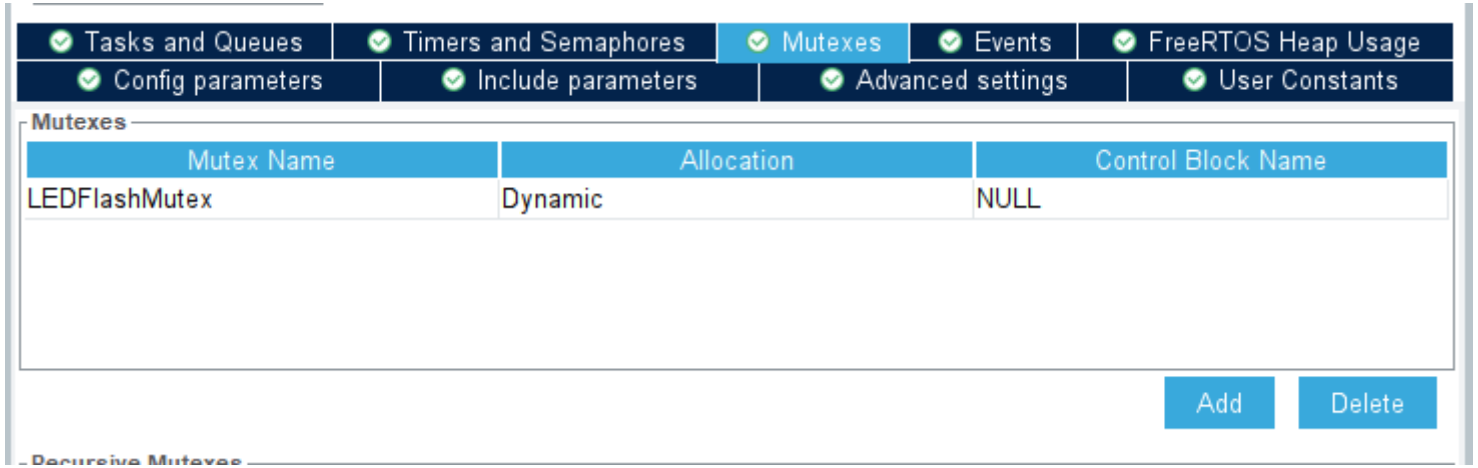
Step 22. Add a queue

Queues

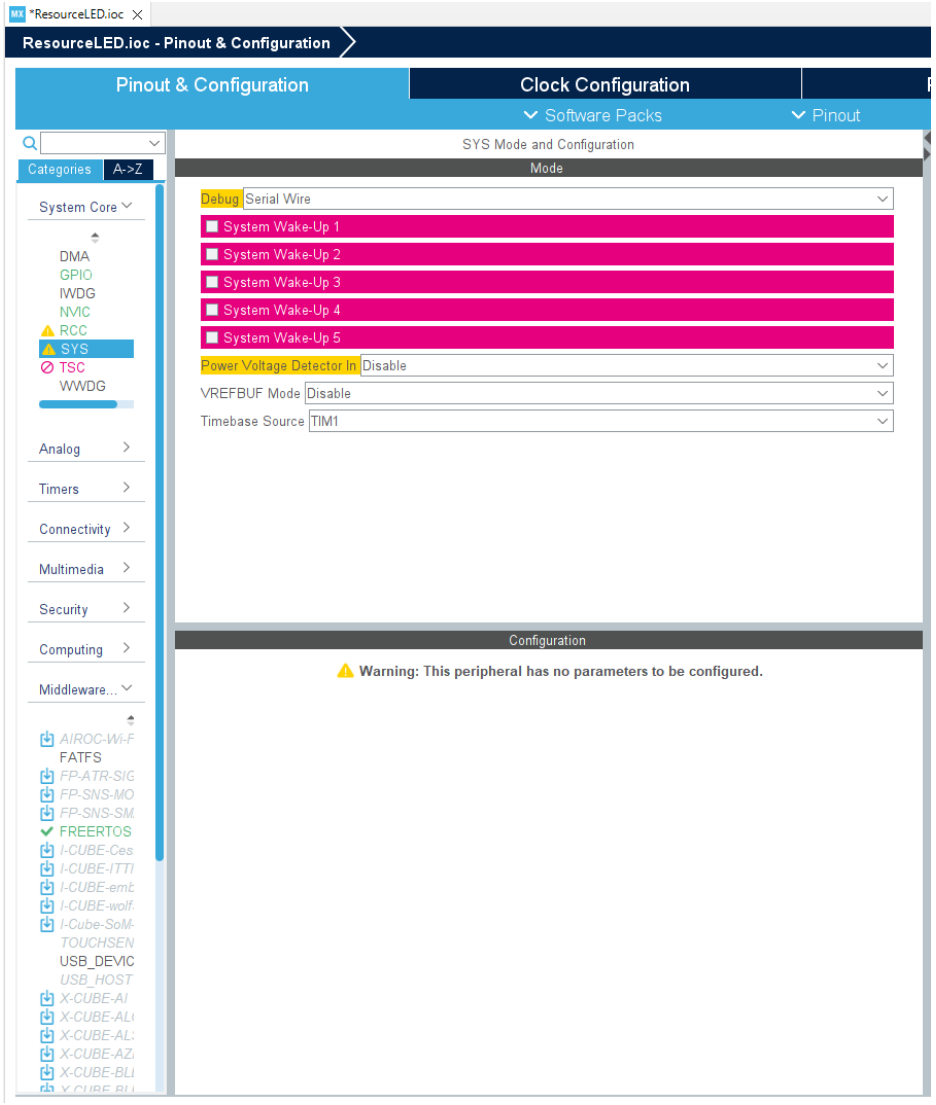
Queue Name	Queue Size	Item Size	Allocation	Buffer Name	Control Block Name
ConsoleQueue	16	uint16_t	Dynamic	NULL	NULL

Add Delete

Step 23. Add a mutex



Step 24. Select Timbase to TIM1



Step 25. Code the default task

ResourceLED.ioc | main.c | stm32l4xx_it.c | stm32l4xx_hal_gpio.c | startup_stm32l475vgtx.s

```c
/**
  * @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 */
    char buf[100];
  /* Infinite loop */
  for(;;)
  {
    // Wait to receive command
    osEvent event = osMessageGet(ConsoleQueueHandle, osWaitForever);
    // Format message for console
    snprintf(buf, sizeof(buf), "event.value.v: %lu\r\n", event.value.v);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
  }
  /* USER CODE END 5 */
}
```

Step 26. Fast LED blink task

```
741 void StartFastBlinkTask(void const * argument)
742 {
743   /* USER CODE BEGIN StartFastBlinkTask */
744   /* Infinite loop */
745   for(;;)
746   {
747     osDelay(1000);
748     // Grab mutex
749     osMutexWait(LEDFlashMutexHandle, osWaitForever);
750     osMessagePut(ConsoleQueueHandle, CMD_FAST_LED_FLASH, osWaitForever);
751     HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, SET);
752     osDelay(250);
753     HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
754     osMutexRelease(LEDFlashMutexHandle);
755   }
756   /* USER CODE END StartFastBlinkTask */
757 }
```

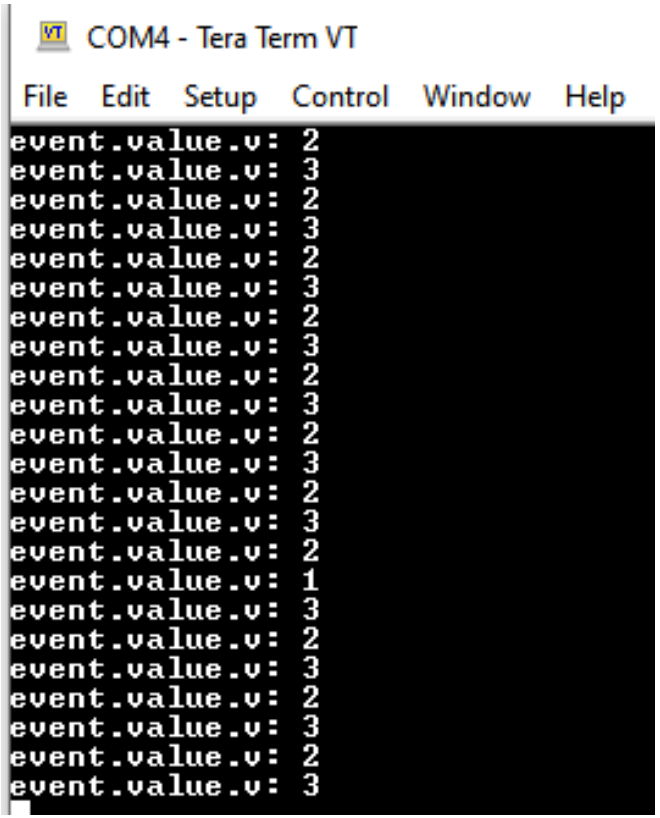
Step 27. slow LED blink test

```
void StartSlowBlinkTask(void const * argument)
{
  /* USER CODE BEGIN StartSlowBlinkTask */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    // Grab mutex
    osMutexWait(LEDFlashMutexHandle, osWaitForever);
    osMessagePut(ConsoleQueueHandle, CMD_SLOW_LED_FLASH, osWaitForever);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, SET);
    osDelay(1000);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
    osMutexRelease(LEDFlashMutexHandle);
  }
  /* USER CODE END StartSlowBlinkTask */
}
```

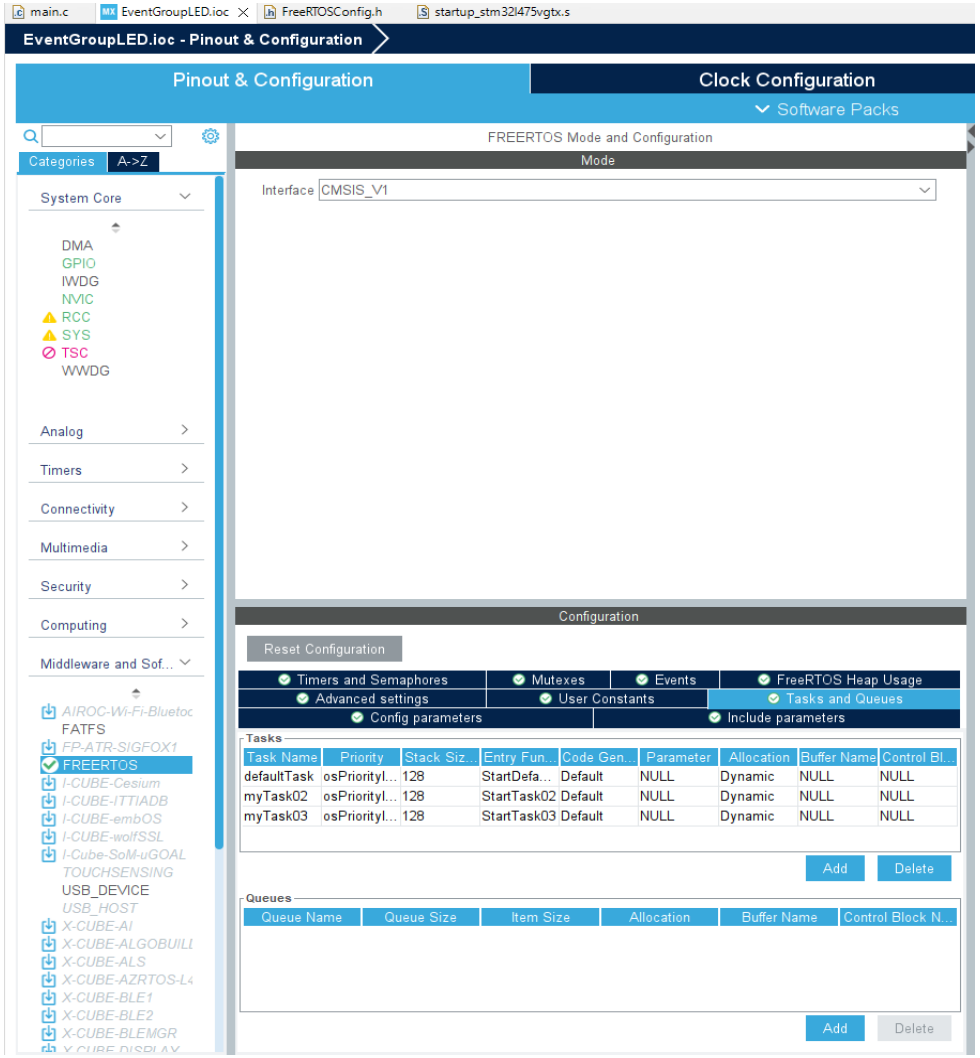
Step 28. GPIO call back

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    osMessagePut(ConsoleQueueHandle, CMD_BUTTON_PRESS, osWaitForever);
}
/* USER CODE END 4 */
```

Step 29. Build and run the code, test is successful



Step 30. Bonus 2, Event group LED project, create 2 more tasks



Step 31. Create even group handle and flags

main.c | EventGroupLED.ioc | FreeRTOSConfig.h | startup_stm32l47

```c
I2C_HandleTypeDef hi2c2;

QSPI_HandleTypeDef hqspi;

SPI_HandleTypeDef hspi3;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart3;

PCD_HandleTypeDef hpcd_USB_OTG_FS;

osThreadId defaultTaskHandle;
osThreadId myTask02Handle;
osThreadId myTask03Handle;
/* USER CODE BEGIN PV */
#define mainDEFAULT_BIT     (1UL << 0UL)
#define mainTASK_BIT_1      (1UL << 1UL)

EventGroupHandle_t  xEventGroup;
/* USER CODE END PV */
```

main.c | FreeRTOSConfig.h | stm32l4xx_it.c

```c
  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_DFSDM1_Init();
  MX_I2C2_Init();
  MX_QUADSPI_Init();
  MX_SPI3_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */
  xEventGroup = xEventGroupCreate();
  /* USER CODE END 2 */
```

Step 32. Set flags in default task and task02

main.c | EventGroupLED.ioc | FreeRTOSConfig.h | startup_stm32l475vgtx.s

```c
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1000);
    xEventGroupSetBits(xEventGroup, mainDEFAULT_BIT);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask02 */
/**
* @brief Function implementing the myTask02 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(3000);
    xEventGroupSetBits(xEventGroup, mainTASK_BIT_1);
  }
  /* USER CODE END StartTask02 */
}
```

Step 33. Blink LED2 in task3, build and run the code, test is successful

```
739 void StartTask03(void const * argument)
740 {
741   /* USER CODE BEGIN StartTask03 */
742   /* Infinite loop */
743   for(;;)
744   {
745       xEventGroupWaitBits(xEventGroup,
746                           mainDEFAULT_BIT | mainTASK_BIT_1,
747                           pdTRUE,
748                           pdTRUE,
749                           portMAX_DELAY);
750       HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, SET);
751       osDelay(1000);
752       HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
753   }
754   /* USER CODE END StartTask03 */
755 }
```