FreeRTOS Lab 1

TA: Sarah p76131369@gs.ncku.edu.tw

Outline

- Requirements
- Port
- Task
- Queue
- Button Issues
- Grading

Requirements

- Demo Video Example
- Must using MultiTask, with Inter-Task Communication (ITC) mechanism
- Two Tasks: **LED-task** and **Button-task**

Requirements

- **LED-task** will have two states (S1, S2)

  S1: First, the Red LED lights up for 1 second, followed by the Orange LED lighting up for 1 second (with the Red LED turned off), then the Green LED lights up for 1 second (with both the Red and Orange LEDs turned off). This sequence repeats, cycling through the Red, Orange, and Green LEDs.

  S2: Only Orange LED is blinking (2 second ON, 2 second OFF, ...).

Requirements

- **Button-task**: If the button is pressed, the LED-task will switch to another state (S1→S2).
  - Debounce handling
  - Edge detection handling

Port

- User Manual, Page 18

6.3 LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the PC and the ST-LINK/V2-A.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
- User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
- User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
- User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.
- USB LD7: green LED indicates when V_{BUS} is present on CN5 and is connected to PA9 of the STM32F407VGT6.
- USB LD8: red LED indicates an over-current from V_{BUS} of CN5 and is connected to the I/O PD5 of the STM32F407VGT6.

6.4 Push buttons

- B1 USER: User and Wake-Up buttons are connected to the I/O PA0 of the STM32F407VG.
- B2 RESET: Push button connected to NRST is used to RESET the STM32F407VG.

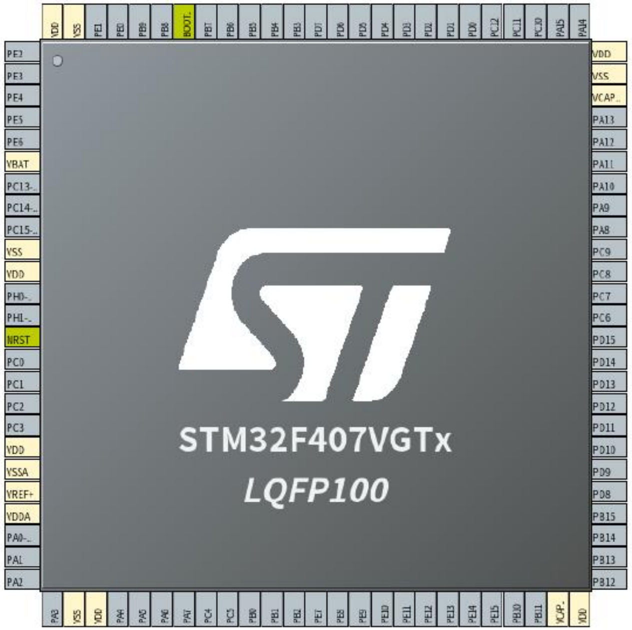
Port

- Pin Assignments

  PD12 → Green LED PD13 → Orange LED
  PD14 → Red LED PA0 → Blue Button

- Configure the LED
  - Left-click on PD12, set it to **GPIO_Output** (it will turn green)
  - Right-click on PD12, select **Enter User Label**, and name it GREEN_LED (or any preferred name)
  - Repeat the process for PD14 (RED_LED) and PD13 (ORANGE_LED)
- Configure the User Button
  - Left-click on PA0, set it to **GPIO_Input** (it will turn green)
  - Right-click on PA0, select **Enter User Label**, and name it BLUE_BUTTON (or any preferred name)
- Important Note
  - Label names can be customized but avoid spaces
  - Press Ctrl + S to save your settings
  - Remember to comment out the handlers discussed in Lab 0

- Diagram in the .ioc file



Port

GPIO_Input

GPIO_Output

Port

- Save the .ioc file to automatically generate code in Project/Core/Src
  - main.h

```c
#define BLUE_BUTTON_Pin GPIO_PIN_0
#define BLUE_BUTTON_GPIO_Port GPIOA
/* ... */
#define GREEN_LED_Pin GPIO_PIN_12
#define GREEN_LED_GPIO_Port GPIOD
#define ORANGE_LED_Pin GPIO_PIN_13
#define ORANGE_LED_GPIO_Port GPIOD
#define RED_LED_Pin GPIO_PIN_14
#define RED_LED_GPIO_Port GPIOD
```

  - main.c

```c
static void MX_GPIO_Init(void)
{
  /* ... */
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GREEN_LED_Pin|ORANGE_LED_Pin|RED_LED_Pin|GPIO_PIN_15
                          |Audio_RST_Pin, GPIO_PIN_RESET);
  /* ... */
  /*Configure GPIO pin : BLUE_BUTTON_Pin */
  GPIO_InitStruct.Pin = BLUE_BUTTON_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BLUE_BUTTON_GPIO_Port, &GPIO_InitStruct);
  /* ... */
  /*Configure GPIO pins : GREEN_LED_Pin ORANGE_LED_Pin RED_LED_Pin PD15
                           Audio_RST_Pin */
  GPIO_InitStruct.Pin = GREEN_LED_Pin|ORANGE_LED_Pin|RED_LED_Pin|GPIO_PIN_15
                          |Audio_RST_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
  /* ... */
}
```

Port

- HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinStatePinState)
- HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
- HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

```c
void Example_GPIO_Control(void) {
    /* Initialize LED (assuming it's connected to GPIOD, Pin 12) */
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET); // Turn off LED initially

    while (1) {
        // Read button state (assuming button is connected to GPIOA, Pin 0)
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET) {
            // If button is pressed, toggle the LED
            HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
            HAL_Delay(200);
        }
    }
}
```

Task

- **Task Function**
  - A task is a small program that runs in an infinite loop and does not exit
  - Reaching the end of a task function may cause unexpected behavior
  - A task function must not contain a return statement

```
void ATaskFunction(void *pvParameters) {
    int counter = 0; // Each task has its own local variable copy
    while(1) {
        counter++;
        vTaskDelay(1000);
    }

    // Should never reach here
}
```

Task

- ## Creating Task

```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                        const char * const pcName,
                        const configSTACK_DEPTH_TYPE usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask )
```

pvTaskCode	Pointer to the task entry function
pcName	A descriptive name for the task
usStackDepth	The number of words (not bytes!) to allocate for use as the task's stack
pvParameters	A value that will passed into the created task as the task's parameter
uxPriority	The priority at which the created task will execute
pxCreatedTask	Used to pass a handle to the created task out of the xTaskCreate() function pxCreatedTask is optional and can be set to NULL

```
xTaskCreate(LED_Handler, "LEDTask_APP", 128, NULL, 1, NULL);
```

Task

- **Task Priority**

  Each task is assigned a priority from 0 to (configMAX_PRIORITIES - 1), where configMAX_PRIORITIES Is defined within FreeRTOSConfig.h

```c
#define configUSE_IDLE_HOOK             0
#define configUSE_TICK_HOOK             0
#define configCPU_CLOCK_HZ              ( SystemCoreClock )
#define configTICK_RATE_HZ              ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES            ( 5 )
#define configMINIMAL_STACK_SIZE        ( ( unsigned short ) 130 )
#define configTOTAL_HEAP_SIZE           ( ( size_t ) ( 75 * 1024 ) )
#define configMAX_TASK_NAME_LEN         ( 10 )
#define configUSE_TRACE_FACILITY        1
#define configUSE_16_BIT_TICKS          0
#define configIDLE_SHOULD_YIELD         1
#define configUSE_MUTEXES               1
#define configQUEUE_REGISTRY_SIZE       8
```

Task

- **vTaskStartScheduler**
  - Start the RTOS scheduler, giving the kernel control over task execution
  - The **idle task** and optionally the **timer daemon task** are created automatically when the scheduler starts
  - It only returns if there is not enough heap memory to create these tasks
  - RTOS demo projects typically call it in the **main function** of main.c

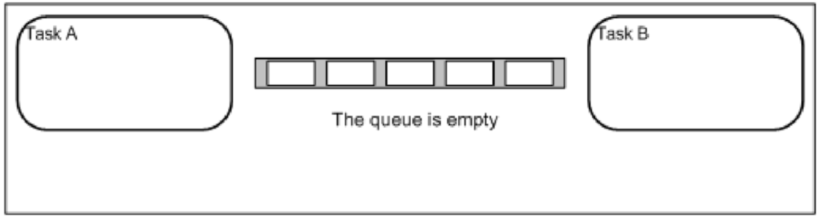
Task

- **vTaskStartScheduler**

```c
int main( void )
{
    xTaskCreate( vTaskCode,
                 "NAME",
                 STACK_SIZE,
                 NULL,
                 tskIDLE_PRIORITY,
                 NULL );

    // Start the real-time scheduler.
    vTaskStartScheduler();

    // Will not get here unless there is insufficient RAM.
    return 0;
}
```

Queue

- The primary form of inter-task communications
- Can be used to send/receive messages between
  - Tasks
  - ISRs and Tasks
- Messages are sent through queues by copy
  - Send: copy to the Queue
  - Receive: copy from the Queue



- FreeRTOS does not define the message structure, allow users to define
  - Message can be pointers or the actual data items/structures
  - For large data, a message can be just a pointer to the data item to avoid large copying overhead

Queue

- **xQueueCreate**

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize );
```

  - uxQueueLength

    The maximum number of items the queue can hold at any one time
  - uxItemSize

    The size, in bytes, required to hold each item in the queue

```
MsgQueue = xQueueCreate(10, sizeof(uint32_t));
```

Queue

- **xQueueSend**

```
BaseType_t xQueueSend(
                        QueueHandle_t xQueue,
                        const void * pvItemToQueue,
                        TickType_t xTicksToWait
                      );
```

  - xQueue
    The handle to the queue on which the item is to be posted
  - pvItemToQueue
    The size, in bytes, required to hold each item in the queue
  - xTicksToWait
    The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full

Queue

- **xQueueSend**

```c
void SenderTask(void *pvParameters) {
    uint32_t msg = 42;
    while (1) {
        xQueueSend(MsgQueue, &msg, portMAX_DELAY); // Send message
        taskYIELD(); // Immediately give CPU time to another task
    }
}
```

Queue

- **xQueueReceive**

```
BaseType_t xQueueReceive(
                          QueueHandle_t xQueue,
                          void *pvBuffer,
                          TickType_t xTicksToWait
);
```

  - xQueue
    The handle to the queue on which the item is to be received
  - pvBuffer
    Pointer to the buffer into which the received item will be copied
  - xTicksToWait
    The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call

Queue

- **xQueueReceive**

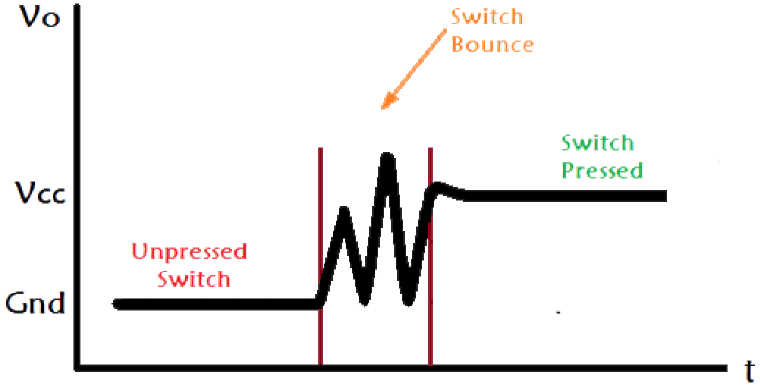
```c
// LED Task: Receives button events and toggles the LED
void LedTask(void *pvParameters) {
    uint32_t receivedValue;

    while (1) {
        // Wait to receive data from the queue
        if (xQueueReceive(MsgQueue, &receivedValue, portMAX_DELAY) == pdPASS) {
            // Toggle LED when button press is received
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
        }
    }
}
```

Queue

- **Blocking on Queue**
  - Sending to a full queue → Task blocks until space is available or timeout occurs.
  - Receiving from an empty queue → Task blocks until data arrives or timeout occurs.
  - If multiple tasks are blocked on the same queue, the highest-priority task is unblocked first.

Button Issues

- Problem
  - Mechanical switches bounce when pressed, causing multiple rapid detections instead of a single press
  - This can lead to unintended multiple task executions



Button Issues

- Solution
  - Use software debounce by adding a delay after detecting a button press
  - Software debounce example
    - vTaskDelay: Delays the task, allowing other tasks to run

```
void Button_Handler(void *pvParameters){
    uint32_t buttonState = 0;
    uint32_t LEDState = 0;

    while(1){
        if(HAL_GPIO_ReadPin(BLUE_BUTTON_GPIO_Port, GPIO_PIN_0)){

            vTaskDelay(10);

            while(HAL_GPIO_ReadPin(BLUE_BUTTON_GPIO_Port,GPIO_PIN_0)){;}

            /* ... */
        }
    }
}
```

Grading

- Including **Demo** and **Report**
- **Demo (5%)**
  - At least one LED state is correct (1%)
  - The LED state changes after pressing the button (2%)
  - The LED state after pressing the button matches the lab requirements (2%)
  - The demo can be done in class, or you can upload a demo video
- **Report (3%)**
  - Write according to the format given on Moodle
  - Please submit a PDF file with the file name as studentID_labNo.pdf
    for example: P76131234_lab1.pdf
- Both the **Demo** and **Report** must be uploaded on Moodle before 23:59 on 3/27; submissions after the deadline will not be graded