**REPORT LAB**

**EMBEDDED SYSTEM - CO3054**

*Group:* **CC02**

*Student:* **Dương Gia An – 1952163**

Contents

[**I.** **INTRODUCTION TO ESP32 AND ESP-IDF** 2](#_Toc117868766)

[1. After install ESP-IDF extension on VS code, I create a project with HelloWorld Template. 2](#_Toc117868767)

[2. Build the Project and connect ESP32 (COM3). 2](#_Toc117868768)

[3. Flash (UART) project code to ESP32 by Press BOOT button on ESP32 while Flashing. 3](#_Toc117868769)

[4. Press Monitor Device to see ESP execute flashed code. 5](#_Toc117868770)

[**II.** **ESP32 GPIO AND FREERTOS TASK** 6](#_Toc117868771)

[Code in file **main.c** 6](#_Toc117868772)

[Link Github: CO3054\_ES\_LAB/LAB1 at main · kinggiaan/CO3054\_ES\_LAB (github.com) 7](#_Toc117868773)

[***Explain:*** 7](#_Toc117868774)

[***Result:*** 8](#_Toc117868775)

[***Does the ESP-IDF need the vTaskStartScheduler() routine?*** 9](#_Toc117868776)

1. **LAB 3 - Preemptive with Slicing**
2. Prioritized Pre-emptive Scheduling with Time Slicing.  
   Graphical user interface, text, application

   Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, website

Description automatically generated

Code:

|  |
| --- |
| #include <stdio.h>  #include "sdkconfig.h"  #include "freertos/FreeRTOS.h"  #include "freertos/task.h"  #include "esp\_system.h"  #include "esp\_spi\_flash.h"  #include "freertos/FreeRTOSConfig.h"  **void** **vTaskFunction**(**void** \*pvParameters)  {  **char** \*pcTaskName;  **const** TickType\_t xDelay250ms = pdMS\_TO\_TICKS(**250**);  /\*The string to print out is passed in via the parameter .  Cast this to a character pointer . \*/  pcTaskName = (**char**\*) pvParameters;  /\*As per most tasks, this task is implemented in  an infinite loop . \*/  **for** (;;)  {  /\*Print out the name of this task . \*/  printf(pcTaskName);  printf("Hello**\n**");  /\*Delay for a period . This time a call to vTaskDelay ()  is used which places the task into the Blocked state  until the delay period has expired . The parameter takes  a time specified in " ticks ", and the pdMS\_TO\_TICKS () macro  is used (where the xDelay250ms constant is declared) to  convert 250 milliseconds into an equivalent time in ticks .\*/  vTaskDelay(xDelay250ms);  }  vTaskDelete(NULL);  }  **void** **vTaskFunction1**(**void** \*pvParameters)  {  **char** \*pcTaskName;  **const** TickType\_t xDelay500ms = pdMS\_TO\_TICKS(**500**);  /\*The string to print out is passed in via the parameter .  Cast this to a character pointer . \*/  pcTaskName = (**char**\*) pvParameters;  /\*As per most tasks, this task is implemented in  an infinite loop . \*/  **for** (;;)  {  /\*Print out the name of this task . \*/  printf(pcTaskName);  printf("Hello**\t**Yo yo**\n**");  /\*Delay for a period . This time a call to vTaskDelay ()  is used which places the task into the Blocked state  until the delay period has expired . The parameter takes  a time specified in " ticks ", and the pdMS\_TO\_TICKS () macro  is used (where the xDelay250ms constant is declared) to  convert 250 milliseconds into an equivalent time in ticks .\*/  vTaskDelay(xDelay500ms);  }  vTaskDelete(NULL);  }  **static**  **const** **char** \*pcTextForTask1 = "Task 1 is running **\r\n**";  **static**  **const** **char** \*pcTextForTask2 = "Task 2 is running **\r\n**";  **static**  **const** **char** \*pcTextForTask0 = "Task 0 is running **\r\n**";  **void** **app\_main**(**void**)  {  printf("===============**\n**");  xTaskCreate(vTaskFunction, " Task 0 Ilde", **2048**,  (**void**\*) pcTextForTask0, **0**, NULL);  /\*Create the first task at priority 1.  The priority is the second to last parameter . \*/  xTaskCreate(vTaskFunction1, " Task 1", **2048**,  (**void**\*) pcTextForTask1, **1**, NULL);  /\*Create the second task at priority 2, which is higher than a priority of 1.  The priority is the second to last parameter . \*/  xTaskCreate(vTaskFunction, " Task 2", **2048**,  (**void**\*) pcTextForTask2, **0**, NULL);    /\*Start the scheduler so the tasks start executing . \*/  //vTaskStartScheduler();  /\*Will not reach here . \*/  } |

Text

Description automatically generated with medium confidence

**Explain result:**

1. **Prioritized Pre-emptive Scheduling (without Time Slicing)**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, website

Description automatically generated**

**Code:**

|  |
| --- |
| **void** **app\_main**(**void**)  {  printf("===============**\n**");  xTaskCreate(vTaskFunction, " Task 0 Ilde", **2048**,  (**void** \*)pcTextForTask0, **0**, NULL);  /\* Create the first task at priority 1.  The priority is the second to last parameter . \*/    /\* Create the second task at priority 2 ,  which is higher than a priority of 1.  The priority is the second to last parameter . \*/  xTaskCreate(vTaskFunction, " Task 2", **2048**,  (**void** \*)pcTextForTask2, **0**, NULL);  //Delay add Task 1 after 1000ms  **vTaskDelay(1000 / portTICK\_PERIOD\_MS);**  xTaskCreate(vTaskFunction1, " Task 1", **2048**,  (**void** \*)pcTextForTask1, **1**, NULL);    /\* Start the scheduler so the tasks start executing . \*/  //vTaskStartScheduler();  /\* Will not reach here . \*/  } |

**Result:**

**Text

Description automatically generated with medium confidence**

**Explain Result:**

1. **Graphical user interface, text, application

   Description automatically generated  
   Text

   Description automatically generated**

**Code:**

|  |
| --- |
| #include <stdio.h>  #include "sdkconfig.h"  #include "freertos/FreeRTOS.h"  #include "freertos/task.h"  #include "esp\_system.h"  #include "esp\_spi\_flash.h"  #include "freertos/FreeRTOSConfig.h"  **void** **vTaskFunction**(**void** \*pvParameters)  {  **char** \*pcTaskName;  **const** TickType\_t xDelay250ms = pdMS\_TO\_TICKS(**250**);  TickType\_t count1000ms = pdMS\_TO\_TICKS(**0**);  /\* The string to print out is passed in via the parameter .  Cast this to a character pointer . \*/  pcTaskName = (**char** \*)pvParameters;  /\* As per most tasks , this task is implemented in  an infinite loop . \*/  **for** (;;)  {  /\* Print out the name of this task . \*/  printf(pcTaskName);  printf("Hello**\n**");  /\* Delay for a period . This time a call to vTaskDelay ()  is used which places the task into the Blocked state  until the delay period has expired . The parameter takes  a time specified in " ticks " , and the pdMS\_TO\_TICKS () macro  is used ( where the xDelay250ms constant is declared ) to  convert 250 milliseconds into an equivalent time in ticks .\*/  vTaskDelay(xDelay250ms);  **if** (count1000ms == pdMS\_TO\_TICKS(**1000**)){  taskYIELD();  }  **else** count1000ms += pdMS\_TO\_TICKS(**250**);  }  vTaskDelete(NULL);  }  **void** **vTaskFunction1**(**void** \*pvParameters)  {  **char** \*pcTaskName;  **const** TickType\_t xDelay500ms = pdMS\_TO\_TICKS(**500**);  /\* The string to print out is passed in via the parameter .  Cast this to a character pointer . \*/  pcTaskName = (**char** \*)pvParameters;  /\* As per most tasks , this task is implemented in  an infinite loop . \*/  **for** (;;)  {  /\* Print out the name of this task . \*/  printf(pcTaskName);  printf("Hello**\t**Yo yo**\n**");  /\* Delay for a period . This time a call to vTaskDelay ()  is used which places the task into the Blocked state  until the delay period has expired . The parameter takes  a time specified in " ticks " , and the pdMS\_TO\_TICKS () macro  is used ( where the xDelay250ms constant is declared ) to  convert 250 milliseconds into an equivalent time in ticks .\*/  vTaskDelay(xDelay500ms);  }  vTaskDelete(NULL);  }  **static** **const** **char** \*pcTextForTask1 = "Task 1 is running **\r\n**";  **static** **const** **char** \*pcTextForTask2 = "Task 2 is running **\r\n**";  **static** **const** **char** \*pcTextForTask0 = "Task 0 is running **\r\n**";  **void** **app\_main**(**void**)  {  printf("===============**\n**");  xTaskCreate(vTaskFunction, " Task 0 Ilde", **2048**,  (**void** \*)pcTextForTask0, **0**, NULL);  /\* Create the first task at priority 1.  The priority is the second to last parameter . \*/    /\* Create the second task at priority 2 ,  which is higher than a priority of 1.  The priority is the second to last parameter . \*/    //Delay add Task 1 and Task 2 after 1000ms  vTaskDelay(**1000** / portTICK\_PERIOD\_MS);  xTaskCreate(vTaskFunction1, " Task 1", **2048**,  (**void** \*)pcTextForTask1, **1**, NULL);  xTaskCreate(vTaskFunction, " Task 2", **2048**,  (**void** \*)pcTextForTask2, **0**, NULL);  /\* Start the scheduler so the tasks start executing . \*/  //vTaskStartScheduler();  /\* Will not reach here . \*/  } |

**Result:**

**Text

Description automatically generated**

1. **Extra Exercise**

|  |
| --- |
| 1. #include <stdio.h> 2. #include "sdkconfig.h" 3. #include "freertos/FreeRTOS.h" 4. #include "freertos/task.h" 5. #include "esp\_system.h" 6. #include "esp\_spi\_flash.h" 7. #include "freertos/FreeRTOSConfig.h" 8. **void** **vTaskFunction\_Idle**(**void** \*pvParameters) 9. { 10. **char** \*pcTaskName; 11. **const** TickType\_t xDelay250ms = pdMS\_TO\_TICKS(**250**); 12. TickType\_t count1000ms = pdMS\_TO\_TICKS(**0**); 13. /\* The string to print out is passed in via the parameter . 14. Cast this to a character pointer . \*/ 15. pcTaskName = (**char** \*)pvParameters; 16. /\* As per most tasks , this task is implemented in 17. an infinite loop . \*/ 18. **for** (;;) 19. { 20. /\* Print out the name of this task . \*/ 21. printf(pcTaskName);   // Hook IDLE TASK   1. /\* Print chip information \*/ 2. **esp\_chip\_info\_t** chip\_info; 3. esp\_chip\_info(&chip\_info); 4. printf("This is %s chip with %d CPU core(s), WiFi%s%s, ", 5. CONFIG\_IDF\_TARGET, 6. chip\_info.cores, 7. (chip\_info.features & CHIP\_FEATURE\_BT) ? "/BT" : "", 8. (chip\_info.features & CHIP\_FEATURE\_BLE) ? "/BLE" : ""); 9. printf("silicon revision %d, ", chip\_info.revision); 10. printf("%dMB %s flash**\n**", spi\_flash\_get\_chip\_size() / (**1024** \* **1024**), 11. (chip\_info.features & CHIP\_FEATURE\_EMB\_FLASH) ? "embedded" : "external"); 12. printf("Minimum free heap size: %d bytes**\n**", esp\_get\_minimum\_free\_heap\_size()); 13. /\* Delay for a period . This time a call to vTaskDelay () 14. is used which places the task into the Blocked state 15. until the delay period has expired . The parameter takes 16. a time specified in " ticks " , and the pdMS\_TO\_TICKS () macro 17. is used ( where the xDelay250ms constant is declared ) to 18. convert 250 milliseconds into an equivalent time in ticks .\*/ 19. vTaskDelay(xDelay250ms); 20. **if** (count1000ms == pdMS\_TO\_TICKS(**1000**)){ 21. taskYIELD(); 22. } 23. **else** count1000ms += pdMS\_TO\_TICKS(**250**); 24. } 25. vTaskDelete(NULL); 26. } 27. **void** **vTaskFunction2**(**void** \*pvParameters) 28. { 29. **char** \*pcTaskName; 30. **const** TickType\_t xDelay250ms = pdMS\_TO\_TICKS(**250**); 31. /\* The string to print out is passed in via the parameter . 32. Cast this to a character pointer . \*/ 33. pcTaskName = (**char** \*)pvParameters; 34. /\* As per most tasks , this task is implemented in 35. an infinite loop . \*/ 36. **for** (;;) 37. { 38. /\* Print out the name of this task . \*/ 39. printf(pcTaskName); 40. printf("Hello **\n**"); 41. /\* Delay for a period . This time a call to vTaskDelay () 42. is used which places the task into the Blocked state 43. until the delay period has expired . The parameter takes 44. a time specified in " ticks " , and the pdMS\_TO\_TICKS () macro 45. is used ( where the xDelay250ms constant is declared ) to 46. convert 250 milliseconds into an equivalent time in ticks .\*/ 47. vTaskDelay(xDelay250ms); 49. } 50. vTaskDelete(NULL); 51. } 52. **void** **vTaskFunction1**(**void** \*pvParameters) 53. { 54. **char** \*pcTaskName; 55. **const** TickType\_t xDelay500ms = pdMS\_TO\_TICKS(**500**); 56. /\* The string to print out is passed in via the parameter . 57. Cast this to a character pointer . \*/ 58. pcTaskName = (**char** \*)pvParameters; 59. /\* As per most tasks , this task is implemented in 60. an infinite loop . \*/ 61. **for** (;;) 62. { 63. /\* Print out the name of this task . \*/ 64. printf(pcTaskName); 65. printf("Hello**\t**Yo yo**\n**"); 66. /\* Delay for a period . This time a call to vTaskDelay () 67. is used which places the task into the Blocked state 68. until the delay period has expired . The parameter takes 69. a time specified in " ticks " , and the pdMS\_TO\_TICKS () macro 70. is used ( where the xDelay250ms constant is declared ) to 71. convert 500 milliseconds into an equivalent time in ticks .\*/ 72. vTaskDelay(xDelay500ms); 73. } 74. vTaskDelete(NULL); 75. } 76. **static** **const** **char** \*pcTextForTask1 = "Task 1 is running **\r\n**"; 77. **static** **const** **char** \*pcTextForTask2 = "Task 2 is running **\r\n**"; 78. **static** **const** **char** \*pcTextForTask0 = "Task 0 is running **\r\n**"; 79. **void** **app\_main**(**void**) 80. { 82. printf("===============**\n**"); 83. xTaskCreate(vTaskFunction\_Idle, " Task 0 Ilde", **2048**, 84. (**void** \*)pcTextForTask0, **0**, NULL); 85. /\* Create the first task at priority 1. 86. The priority is the second to last parameter . \*/ 88. /\* Create the second task at priority 2 , 89. which is higher than a priority of 1. 90. The priority is the second to last parameter . \*/ 92. //Delay add Task 1 after 1000ms 93. vTaskDelay(**1000** / portTICK\_PERIOD\_MS); 94. xTaskCreate(vTaskFunction1, " Task 1", **2048**, 95. (**void** \*)pcTextForTask1, **1**, NULL); 96. xTaskCreate(vTaskFunction2, " Task 2", **2048**, 97. (**void** \*)pcTextForTask2, **0**, NULL); 98. /\* Start the scheduler so the tasks start executing . \*/ 99. //vTaskStartScheduler(); 100. /\* Will not reach here . \*/ 101. } |

**Result:  
Text

Description automatically generated**

1. **ESP32 GPIO AND FREERTOS TASK**

## Code in file **main.c**

1. #include <stdio.h>
2. #include "sdkconfig.h"
3. #include "freertos/FreeRTOS.h"
4. #include "freertos/task.h"
5. #include "esp\_system.h"
6. #include "esp\_spi\_flash.h"

9. **void** print\_id(**void** \*pvParameter){
10. **while**(1){
11. **printf**("DUONG GIA AN : %d \n",1952163);
12. vTaskDelay(1000/portTICK\_PERIOD\_MS);
14. }
15. vTaskDelete(NULL);
17. }
19. **void** blinky(**void** \*pvParameter){
20. **while**(1){
21. **printf**("Press Button\n");
22. **int**  rd = **rand**() % (5000 + 1 - 0) + 0;
23. vTaskDelay(rd /portTICK\_PERIOD\_MS);
24. }
25. vTaskDelete(NULL);
26. }

29. **void** app\_main(){
30. xTaskCreate(&print\_id, "print\_id", 2048, NULL, 0, NULL);
31. xTaskCreate(&blinky, "blinky", 2048,NULL,0,NULL );
33. **for** (**int** i = 20; i >= 0; i--) {
34. **printf**("Remaing %d seconds...\n", i);
35. vTaskDelay(1000 / portTICK\_PERIOD\_MS);
36. }
37. **printf**("Restarting now.\n");
38. vTaskDelay(5000 / portTICK\_PERIOD\_MS);
39. **fflush**(stdout);
40. esp\_restart();
41. }

## Link Github: [CO3054\_ES\_LAB/LAB1 at main · kinggiaan/CO3054\_ES\_LAB (github.com)](https://github.com/kinggiaan/CO3054_ES_LAB/tree/main/LAB1)

## ***Explain:***

* + Cyclic task: void **print\_id()** is task that print my student ID every 2 seconds.
  + Acylic task: void **Blinky()** is alternated for button in GPIO in ESP32. I change to a random time to press button from 0 – 5000ms.
  + **app\_main()** will print time stamp every 1 second and restart ESP after 20 seconds.

The *priority* and *usStackDepth* of task cyclic/acylic is the same as 0 and 2048 (mean 2048\*4 bytes will be allocated for these tasks).

## ***Result:***

Graphical user interface, text

Description automatically generated

## ***Does the ESP-IDF need the vTaskStartScheduler() routine?***

No, because ESP-IDF will call vTaskStartScheduler() automatically.

|  |
| --- |
| Unlike Vanilla FreeRTOS, users must not call [**vTaskStartScheduler()**](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html#_CPPv419vTaskStartSchedulerv). Instead, ESP-IDF FreeRTOS is started automatically. The entry point is a user defined void app\_main(void) function.   * Typically, users would spawn the rest of their applications task from app\_main. * The app\_main function is allowed to return at any point (i.e., before the application terminates). * The app\_main function is called from the main task.   The main task is one of multiple tasks that are automatically spawned by ESP-IDF during startup.[[1]](#footnote-1) |

1. [FreeRTOS - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html) [↑](#footnote-ref-1)