LAB N° 1

Task & Scheduling

Lab Objectives

- Creating, suspending and deleting a task.
- Understanding the scheduling of tasks and priority effect.
- Multi-cores scheduling
- Using Idle task.
- Task handler

1.1 Task scheduling on one core (Lab1-1)

The entry point of an application is the *app_main()* function. This function is actually a task of priority 1. We will usually create the other application tasks from the *app_main()* task. We will create an application with 2 tasks (including *app_main()* task), then 3 tasks running on 1 core in order to understand the behavior of the scheduler.

1.1.1 Preparation

Answer the following questions :

- Study the parameters of the following function : *xTaskCreate()* and *vTaskDelete()*. Web help

- Study the following function : *uxTaskPriorityGet()*. Web help

- Study the following function : *vTaskDelay()*. Web help

- What do they do the following macros in the provided « my_helper_fct.h » file ? *DISPLAY()* and *DISPLAYB()*.

- *Task 1* is an instance of the *vTaskFunction()* in the provided code of the « lab1-1_main.c » file. Is that true ?

- Open the provided « lab1-1_sdkconfig.defaults » file. What does the *CONFIG_FREERTOS_UNICORE* parameter correspond to ? Web help

- What is the tick period in ms ? see the *CONFIG_FREERTOS_HZ* parameter and Web help

1.1.2 Understanding the task scheduling

1. Create a new folder named « part3_freertos ».
2. In the « part3_freertos » folder, create the « lab1-1_1_core_sched » lab from « esp32-vscode-project-template » GitHub repository.
3. Overwrite the « main.c » file by the provided code of the « lab1-1_main.c » file.
4. Copy the provided « my_helper_fct.h » file to the « main » folder.
5. Copy the provided « lab1-1_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
6. Build and run the program.
7. Trace in the figure 1.1 the behavior of the *app_main()* and *Task 1* tasks until 160 ticks.
8. Why the *app_main()* task does not run ?

On devrait pas voit le exit app_main car created task priority > app_main
app_main ne display pas car TASK1 prio mais tjr running donc ne redonne

9. What is the simulated computation time ?

   1tick = 10ms => entre 28 et 56 = 28 => 280ms

10. Create a new instance *Task 2* of the *vTaskFunction()* with the same priority of *Task 1*, i.e. priority=5.

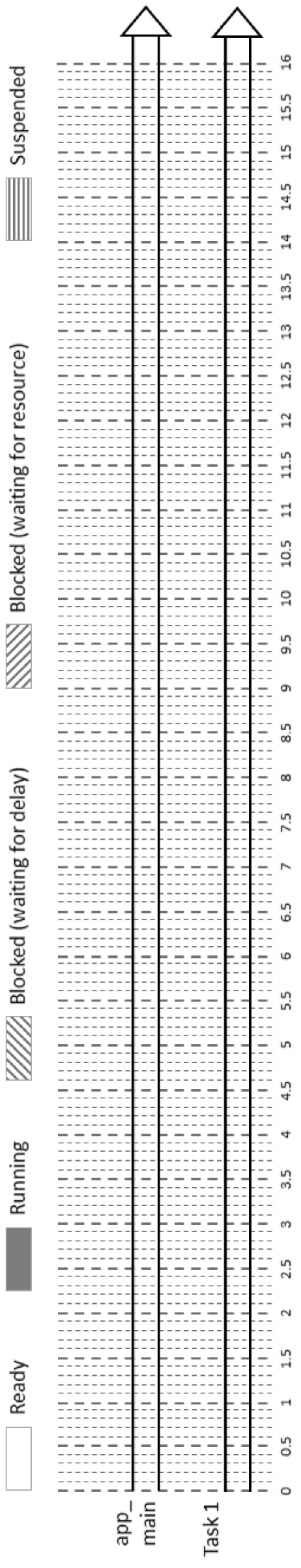

Figure 1.1 – *Scheduling of 2 tasks.*

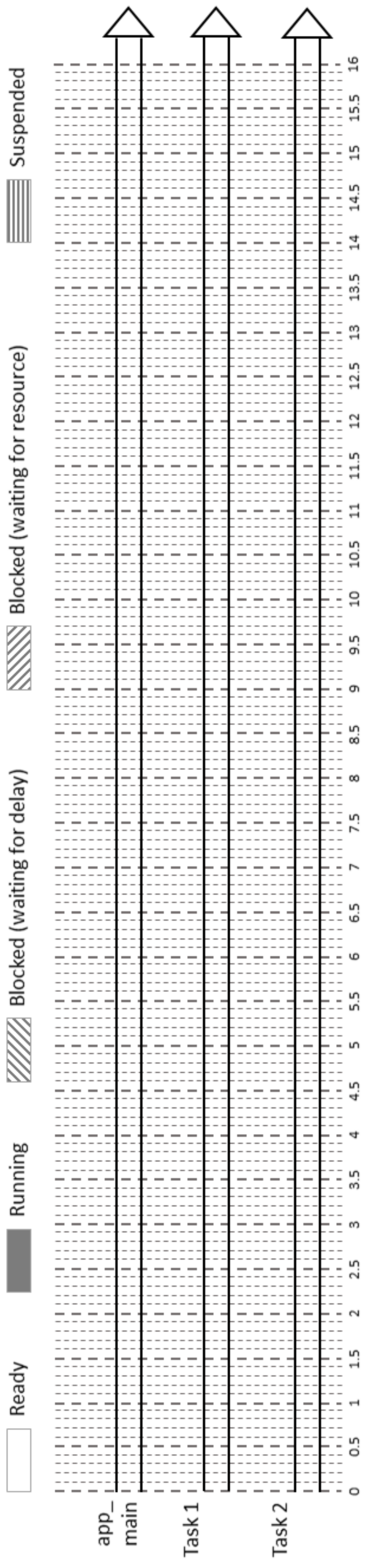

Figure 1.2 – *Scheduling of 3 tasks.*

11. Run the program. Comment on the behavior.

    Passe jamais sur TASK2 car TAS1 tjr running
    ne start même pas
    why => pas réellement crée, pas le tps, car TASK1 tjr en run et app_main s'exécute pas
    => soluce : suspendre le scheduler

12. We are now going to suspend the scheduler with 2 functions when creating the tasks.

```
void app_main(void) {
  ...
  vTaskSuspendAll();
  // Task creation ...
  ...
  xTaskResumeAll();
  ...
```

13. Run the program.
14. For this scenario, trace in the figure 1.2 the behavior of the *app_main()*, *Task 1* and *Task 2* tasks until 160 ticks.
15. Comment on the behavior. Why the *app_main()* task does not run ?

    because same reason
    => trace : 10ms (voir photo) because round robin (on change tous les ticks)

16. Change the priority=6 for the Task 2. Run the program and comment on the behavior for this scenario.

    TASK2 always running TASK1 not even starting

17. We are now going to add delay on *vTaskFunction()* after the simulation of computation time.

```
void vTaskFunction(void *pvParameters) {
...
for (;; ) {
  DISPLAY("Run computation of %s", pcTaskName);

  /* Delay for simulating a computation */
  for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++){
  }

  // Add Delay of 300 ms
  DISPLAY("Delay of %s", pcTaskName);
  vTaskDelay(300 / portTICK_PERIOD_MS);
  ...
```

18. Trace in the figure 1.3 the behavior of the *app_main()*, *Task 1* and *Task 2* tasks until 160 ticks.

1.1.3 Using trace information (Optional work)

It is possible to get information about the tasks of the application using the *vTaskList()* function (Web help).

1. Copy the code provided in the « lab1-1_add_vtasklist.c » file to print trace information as below. Note that size of the buffer is approximately 40 bytes per task.

```
/* Buffer to extract trace information */
static char buffer[40*8];

void app_main(void) {
  ...
  /* Print task list */
  vTaskList(buffer);
  printf("--------------------------------------------------------\n"
     );
  printf("task\t\tstate\tprio\tstack\ttsk id\tcore id\n");
  printf("--------------------------------------------------------\n"
     );
  printf(buffer);

  DISPLAY("==== Exit APP_MAIN ====");
  ...
```

2. The 2 first parameters are compulsory to use the *vTaskList()* function. The last parameter is used to print the code id. We active the 3 parameters. 2 solutions :

   - Using the menuconfig command and go to *Component config/FreeRTOS/Enable FreeRTOS trace facility, Enable FreeRTOS stats formatting functions ans Enable display of xCoreID in vTaskList.*
   - Add these lines below in the « sdkconfig.defaults » file (provided in *lab1-1_config_trace_information.txt* file)and delete the « sdkconfig » to take into account the modifications.

   ```
   # Trace facility to extract trace information
   CONFIG_FREERTOS_USE_TRACE_FACILITY=y
   CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS=y
   CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID=y
   ```

3. Run the program. Copy the trace information. The *stack* column is the amount of free stack space in bytes there has been since the task was created.

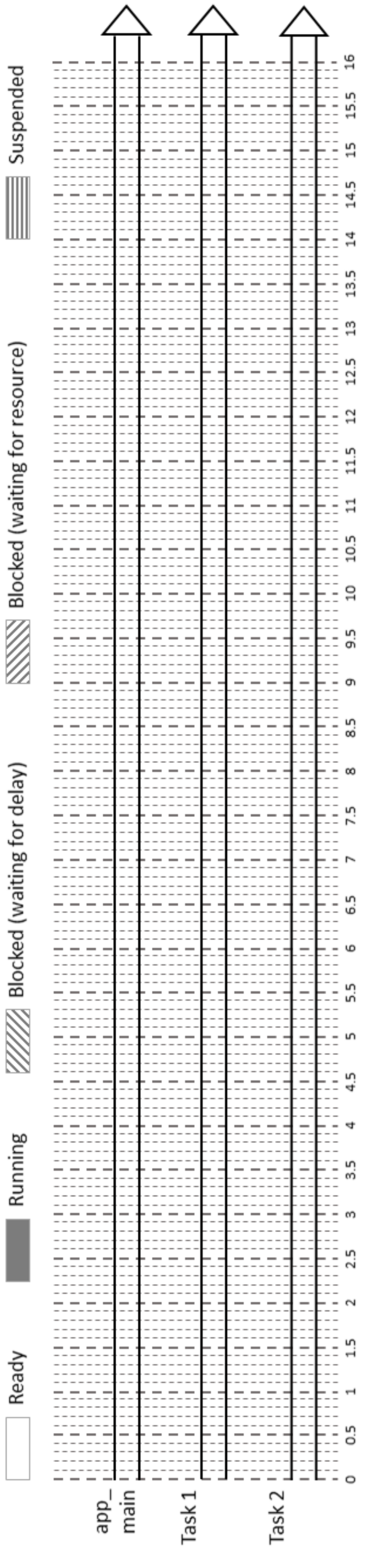

FIGURE 1.3 – *Scheduling of 3 tasks with task delay.*

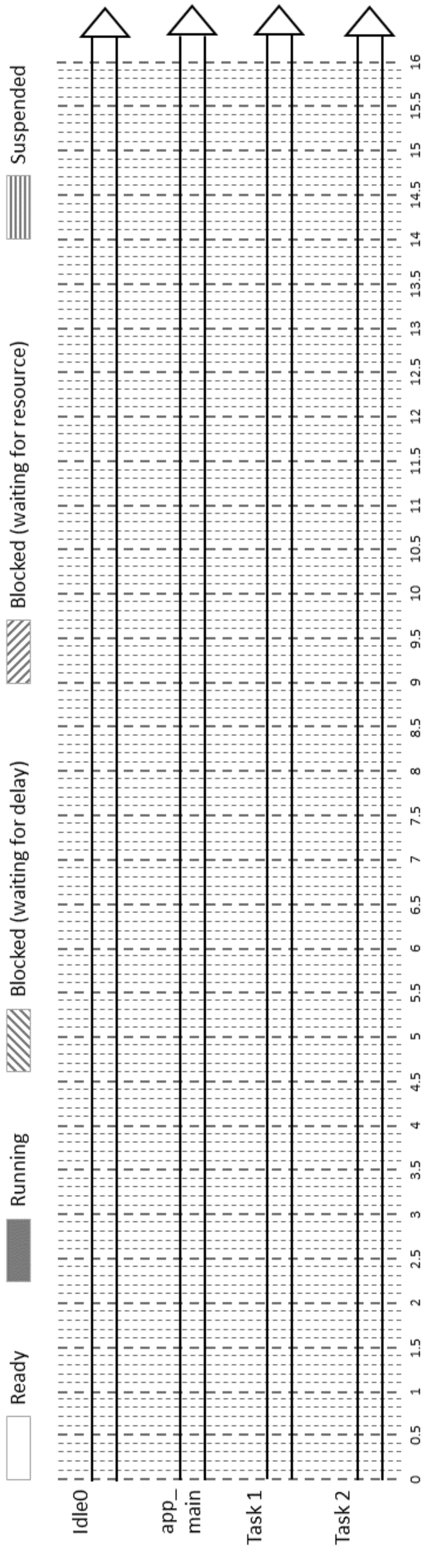

FIGURE 1.4 – *Scheduling of 3 tasks with task delay.*

4. How many tasks are there ?

5. What is the highest priority task ?

6. What does the *main* task correspond to ?

7. What is the role of the *ipc0* task ? Web help

8. What is the role of the *esp_timer* task ? Web help

9. What is the role of the *Tmr Svc* task ? Web help 1 and Web help 2

10. What is the role of the *IDLE0* task ? What is its priority ? Web help

11. The default stack size is 4000 bytes. Change the constant *STACK_SIZE*=1400 bytes. What is the problem ? Change the size to the nearest hundredth to avoid the problem.

1.1.4 Idle and Tick Task Hooks (Optionnal work)

An idle task hook is a function that is called during each cycle of the idle task. To use them, we will modify the configuration.

1. Add the two hook functions provided in the « lab1-1_add_idle_hook.c » file.

```
bool vApplicationIdleHook ( void ){
  DISPLAY ("IDLE");
  return true;
}
```

```
void vApplicationTickHook ( void ){
}
```

2. Why are there 2 hook functions ? What is the difference ? Web help 1 and Web help 2

3. The first parameter is compulsory to use the hooks functions. The second parameter adjusts the stack size depending on the behavior of the hook functions. 2 solutions :
   - Using the menuconfig command and go to *Component config/FreeRTOS/Use FreeRTOS legacy hooks.*
   - Add these lines below in the « sdkconfig.defaults » file (provided in *lab1-1_config_idle_hook.txt* file) and delete the « sdkconfig » to take into account the modifications.

     ```
     # allows add hook functions
     CONFIG_FREERTOS_LEGACY_HOOKS=y
     CONFIG_FREERTOS_IDLE_TASK_STACKSIZE=4096
     ```

4. Run the program. What do you see for the *Idle* and *app_main* tasks ?

5. Modify the task delay of vTaskFunction() function to 400ms instead of 100ms. What do you see for *Idle* and *app_main* tasks ? Conclusion ?

6. Trace in the figure 1.4 the behavior of tasks until 160 ticks.

1.2 Task scheduling on two cores (Lab1-2)

the ESP32 has 1 or 2 cores. It is possible to choose on which core the task should be mapped or to let the scheduler choose. At the end of this Lab, we will see another solution to use the *idle* task without using the configuration but an API.

1. Create the « lab1-2_2_cores_sched » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab1-2_main.c » file.
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab1-2_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».

5. Study the following function : *xTaskCreatePinnedToCore()*. Web help

1.2.1 Task scheduling scenarios

We are going to test different scheduling scenarios according to the cores on which the tasks will be executed. In order to facilitate the different scenarios to be executed, we will use the preprocessor macros as below :

```
//#define DIFF_PRIORITY

#define PINNED_TO_CORE 0x00
//   0x00: Task 1: Core 0, Task 2: Core 0
//   0x01: Task 1: Core 0, Task 2: Core 1
//   0x10: Task 1: Core 1, Task 2: Core 0
//   0x11: Task 1: Core 1, Task 2: Core 1

//#define IDLE_HOOKS

//#define TASK_DELAY
```

You must run the 4 scenarios, copy the trace till 160 ticks (1600 ms) and comment it.

- Uncomment *DIFF_PRIORITY* to have the priority(Task 1) = 5 and priority(Task 2) = 6.
- Scenario n° 1 : Task 1 : Core 0, Task 2 : Core 0

Deux tasks sur 1 coeur, TASK2 seulement s'exécute car prio

- Scenario n° 2 : Task 1 : Core 0, Task 2 : Core 1

Les deux s'exécutent, prio pas d'importance, commence et finit en même tps

- Scenario n° 3 : Task 1 : Core 1, Task 2 : Core 0
  inversé par rapport a scénario 2 car TASK2 sur core0

- Scenario n° 4 : Task 1 : Core 1, Task 2 : Core 1.
  Seulement TASK2 sur core 1 car prio mais apparition app_main task on core 0 because its free

  => With TASK3 : still only task2 on core1 but now task3 and app_main task are running round robin because same prio, compilation app_main 10ticks task3 28 ticks

For the last scenario n° 5, we are going to add a task on *core 0* using task mapping of scenario n° 4.

1. Add a *Task 3* on core 0 with priority = 1.
2. Trace in the figure 1.5 the behavior of the tasks until 160 ticks.

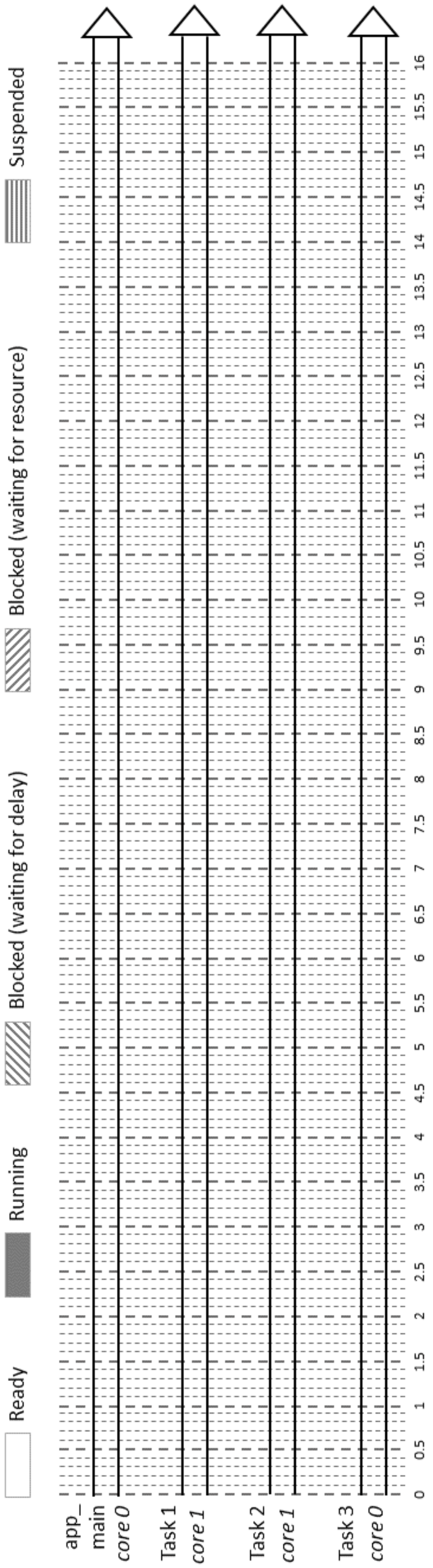

Figure 1.5 – *Scheduling on 2 cores with 4 tasks.*

1.2.2 Idle Task Hook from API (Optional work)

The disadvantage of configuring the hook functions from the configuration menu is the declaration of the 2 functions while generally the *vApplicationIdleHook()* function is the only useful function. Thus, it is possible to use an API which allows adding only the *vApplicationIdleHook()* function.

1. Uncomment the *IDLE_HOOKS* preprocessor macro. We use the previous scenario n° 5.
2. Study the following function : *esp_register_freertos_idle_hook_for_cpu()*. Web help
3. Run the program and comment the behavior. What do you notice about the execution of the *IDLE1* task ?
4. Uncomment the *TASK_DELAY* preprocessor macro.
5. Run the program and comment the new behavior, in particular the *IDLE1* task.

1.3 Approximated/Exactly periodic task (Lab1-3)

In some applications it is important to have tasks with an exact period. We will compare the use of the *TaskDelay()* function which allows you to wait for a certain time and the *TaskDelayUntil()* function which allows you to adjust the delay according to the time spent.

1. Duplicate the « lab1-2_2_cores_sched » folder to « lab1-3_periodic_task ».
2. Check that you have uncommented the *DIFF_PRIORITY*, *IDLE_HOOKS* and *TASK_DELAY* preprocessor macros.
3. Set the *PINNED_TO_CORE* macro to $0x00$ to run all tasks on *PRO_CPU* (Core 0).
4. Change the *mainDELAY_LOOP_COUNT* constant to *0xFFFFF*. What is the simulated computation time ?
5. Change the behavior of the *vTaskFunction()* task as below.

```
#ifdef TASK_DELAY
  /* Approximated/Periodic Delay */
  #ifdef PERIODIC_TASK_DELAY
    DISPLAY("Periodic Delay of %s", pcTaskName);
    vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(300));
```

```
      #else
        DISPLAY("Approximated Delay of %s", pcTaskName);
        vTaskDelay(pdMS_TO_TICKS(300));
      #endif
    #endif
```

6. What does it do the following macro *pdMS_ TO_ TICKS()* ?

7. Run the program and comment the behavior. What is the simulated compute time ? What is the interval/period of Tasks 1,2 and 3 ? To compute time, use the tick number from the display of « Run computation of Task X » (X = 1,2 or 3) comment.

T2 = 45 - 58 = 13+30 = 43 (boucle for 13ticks le reste 30) periode
pareil pour les autres

8. Study the *vTaskDelayUntil()* function. Web help

9. Uncomment the *PERIODIC_TASK_DELAY* preprocessor macro.
10. Build the project and correct the compilation error if necessary.
11. Run the program at least up to 400 ticks and comment the behavior. What is the interval/period of Tasks 1,2 and 3 ? What is the problem ? Correct it.

period de 30 pour T2 et T1
sauf pour T3 : 194 374
118 : IDLE0 a 30

12. Change the *mainDELAY_ LOOP_ COUNT* constant to *0x7FFFF*. What is the interval/period of Tasks 1,2 and 3 ? Comment the new behavior.

augmenter tps de blocage, T1 T2 T3 respecte 30

1.4 Task handler and dynamic priority (Lab1-4, Optional work)

During the execution of the application, it is sometimes useful to modify the priority of a task or any other actions. To do this, it is necessary to get the handler of the task on which we have to operate these actions.

1. Create the « lab1-4_task_handler_priority » lab from « esp32-vscode-project-template » GitHub repository.
2. Overwrite the « main.c » file by the provided code of the « lab1-4_main.c » file.
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Copy the provided « lab1-4_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
5. Study the *uxTaskPriorityGet()* and *vTaskPrioritySet()* functions. Web help
6. Study the *vTaskGetRunTimeStats()* function. What parameters should be configured in *sdkconfig* file or from graphical menu config ? Web help
7. Explain the use of the handler. In what tasks ? Why ?
8. Run the program. What is the problem ?
9. How are you going to correct the problem ?
10. Run the program after code modification and comment the scenarios.
11. Comment also the statistics.