LAB N° 3

Semaphore & Mutex

Lab Objectives

- Using semaphore API.
- Using Mutex

3.1 Specification of the application

We want to implement the functional structure (cf. figure 3.1) using FreeRTOS.

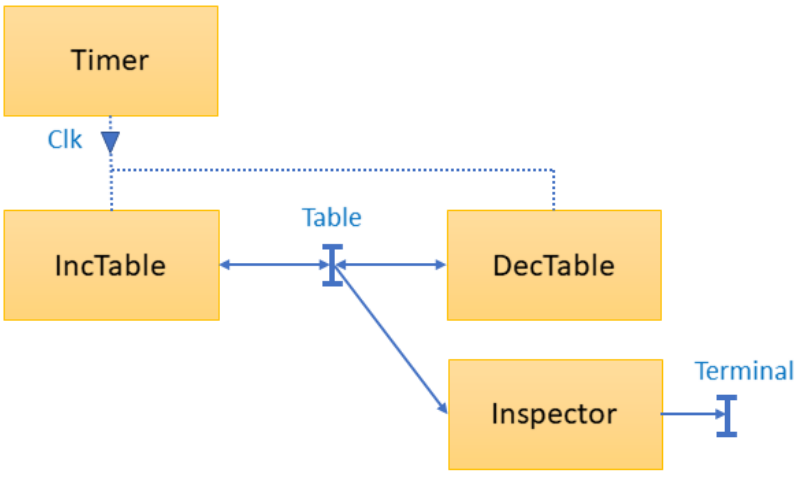

Figure 3.1 – *Functional description.*

The behavioral description of tasks is presented below in algorithm form. *Table* is an array of integer values that can represent a signal, such as a ramp. Its size is a constant (TABLE_SIZE) which can be modified during the tests. During initialization, it takes the

values [0,1,2,..., TABLE_SIZE-1]. Use TABLE_SIZE = 400.
Do not forget to use the *pdMS_ TO_ TICKS(iTime time)* macro to convert time in millisecond to tick number.

3.1.1 Timer task

This is a task that performs a *TaskDelay()* and generates the synchronization (semaphore) via *Clk*. The function *computeTime()* simulates a execution time in millisecond.

Task *Timer* **is**
Properties: Priority = 5
Out **:** Clk is event
Cycle :
waitForPeriod(250 ms);
computeTime(20 ms);
print("Task Timer : give sem");
notify(Clk);
end
end

Algorithm 3.1: Timer algorithm.

3.1.2 IncTable task

It is a temporary action activated by *Clk*. An *constNumber* is passed by the parameters of the task. Every 5 activations, the task increments by the *constNumber* for each element of *Table*. We simulate a computation time by the *computeTime()* pseudo function. Its functional behavior is as follows :

```
Task IncTable is
    Properties: Priority = 4
    In          : Clk is event
    In/Out      : Table is array[0 to TABLE_SIZE-1] of integer
    ActivationNumber := 0;
    Cycle Clk :
        if ActivationNumber = 0 then
            for index :=0 to TABLE_SIZE-1 do
                Table[index] := Table[index] + constNumber;
            end
            computeTime(50 ms);
            ActivationNumber := 4;
        else
            ActivationNumber := ActivationNumber - 1;
        end
    end
end
```

Algorithm 3.2: IncTable algorithm.

3.1.3 DecTable task

Its functional behavior is as follows :

```
Task DecTable is
    Properties: Priority = 4
    In          : Clk is event
    In/Out      : Table is array[0 to TABLE_SIZE-1] of integer
    Cycle Clk :
        for index :=0 to TABLE_SIZE-1 do
            Table[index] := Table[index] - 1;
        end
        computeTime(50 ms);
    end
end
```

Algorithm 3.3: DecTable algorithm.

3.1.4 Inspector task

It is a task which constantly checks the consistency of the *Table* and displays an error message when an inconsistency is found in the *Table* (exit on the program, use *exit(1)* function). For this, it takes the first value of the *Table* as a reference (*reference = Table[0]*) and checks each element of *Table* in accordance with its reference (*Table[index] = reference + index*). When the *Table* has been fully browsed, the cycle begins again (a new reference is

taken and the *Table* is checked again). Its functional behavior is as follows :

Task *Inspector* **is**
Properties: Priority = 4
In : Table is array[0 to TABLE_SIZE-1] of integer
Cycle :
print("Task Inspector is checking.");
reference := Table[0];
error := false;
for *index :=1* **to** *TABLE_SIZE-1* **do**
ComputeTime(100 us);
if $Table[index] \neq (reference + index)$ **then**
error := true;
end
end
print("Task Inspector ended its checking.");
if *error = true* **then**
print("Consistency error in the Table variable.");
exit();
end
end
end

Algorithm 3.4: Inspector algorithm.

3.2 First Task synchronization (Lab3-1)

Firstly, we will only implement the *Timer*, *DecTable* and *IncTable* tasks as well as the *Clk* and *Table* relationships. The *computeTime()* function can be implemented by the *COMPUTE_IN_TIME_MS()* or *COMPUTE_IN_TIME_US()* macros and the print() function by *DISPLAY()* macro.

3.2.1 Writing the application

1. Create the « lab3-1_one_sem_clk » lab from « esp32-vscode-project-template » GitHub repository.
2. Copy the provided « lab3-1_sdkconfig.defaults » file to the project folder and rename « sdkconfig.defaults ».
3. Copy the provided « my_helper_fct.h » file to the « main » folder.
4. Overwrite the « main.c » file by the provided code of the « lab3-1_main.c » file.
5. Write the program with the behavior of these 3 tasks and 1 semaphore (*xSemClk*) using the algorithms proposed above. All the tasks are created on the *Core_0*. Below is a creation reminder for a semaphore.

```
/* Creating Binary semaphore */
SemaphoreHandle_t xSemClk;
xSemClk = xSemaphoreCreateBinary();
/* Using semaphore */
xSemaphoreGive(xSemClk);
xSemaphoreTake(xSemClk, portMAX_DELAY);
```

6. Build the program without running it.

3.2.2 Scenarios with one clock semaphore

We will perform scenarios described in the Table 3.1 in order to identify problems and improve the program later. The task priority of *Timer* is 5 and run on *Core_ 0*.

Scenario	IncTable task	DecTable task
1	Prio(4),Core(0)	Prio(4),Core(0)
2	Prio(3),Core(0)	Prio(4),Core(0)
3	Prio(4),Core(0)	Prio(4),Core(1)
4	Prio(3),Core(0)	Prio(4),Core(1)

Table 3.1 – *Scenarios for task synchronization.*

Scenario 1 : Run the program, copy the console, trace in the figure 3.2 the behavior of the 3 tasks until 160 ticks and explain the problem.

Scenario 2 : Run the program, copy the console and explain the problem.

Scenario 3 : Run the program, copy the console and explain the problem.

Scenario 4 : Run the program, copy the console and explain the problem.

3.3 Task synchronization with 2 semaphores (Lab3-2)

We identified different issues in the previous section. We will perform again the scenarios described in the Table 3.1.

1. Duplicate the « lab3-1_one_sem_clk » folder to « lab3-2_two_sem_clk ».
2. Correct the clock program to send 2 separate semaphores (*xSemIncTab* and *xSemDecTab*) to *IncTable* and *DecTable* tasks.

Scenario 1 : Run the program, copy the console, trace in the figure 3.3 the behavior of the 3 tasks until 160 ticks and explain the behavior.

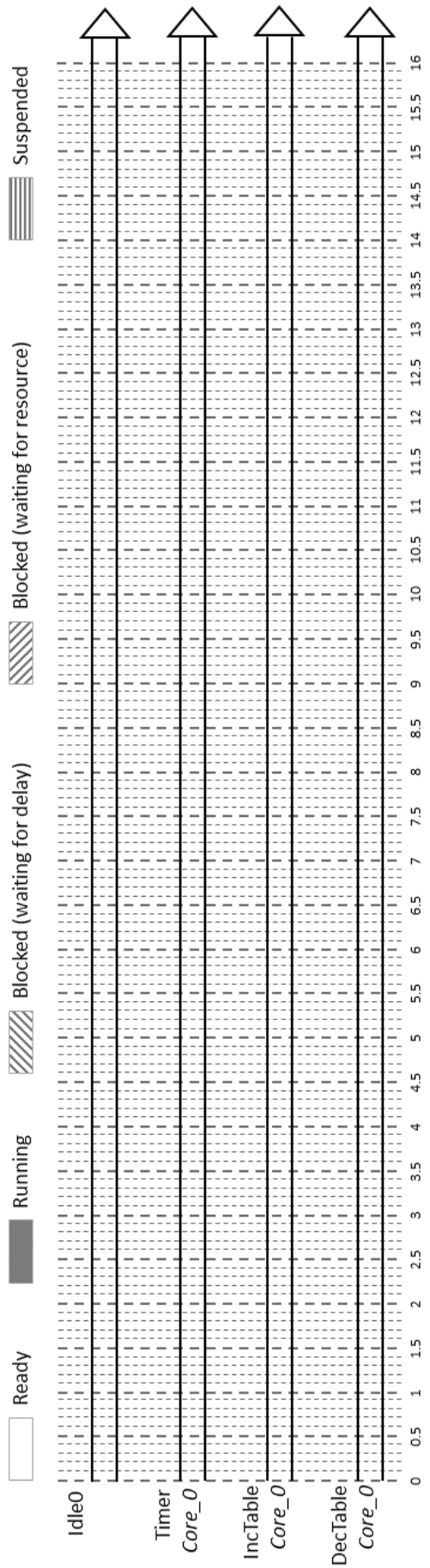

Figure 3.2 – *Scenario 1 : Priority(DecTable=4, IncTable=4), Core(DecTable=0, IncTable=0).*

Scenario 2 : Run the program, copy the console, trace in the figure 3.4 the behavior of the 3 tasks until 160 ticks and explain the behavior.

Scenario 3 : Run the program, copy the console, trace in the figure 3.5 the behavior of the 3 tasks until 160 ticks and explain the behavior.

Scenario 4 : Run the program, copy the console and explain the behavior.

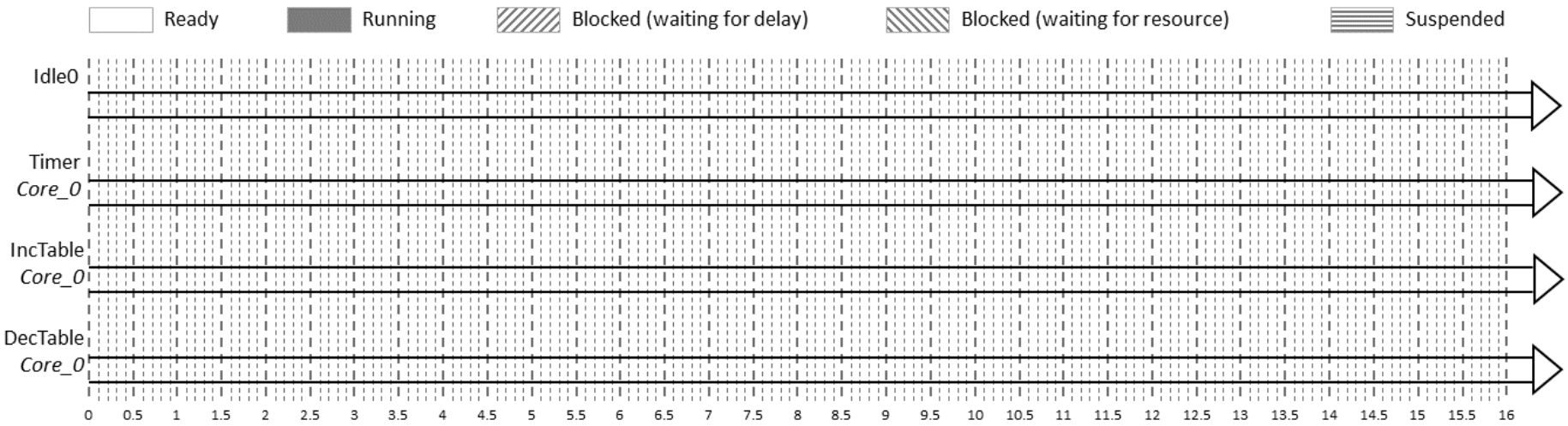

FIGURE 3.3 – *Scenario 1 : Priority(DecTable=4, IncTable=4), Core(DecTable=0, IncTable=0).*

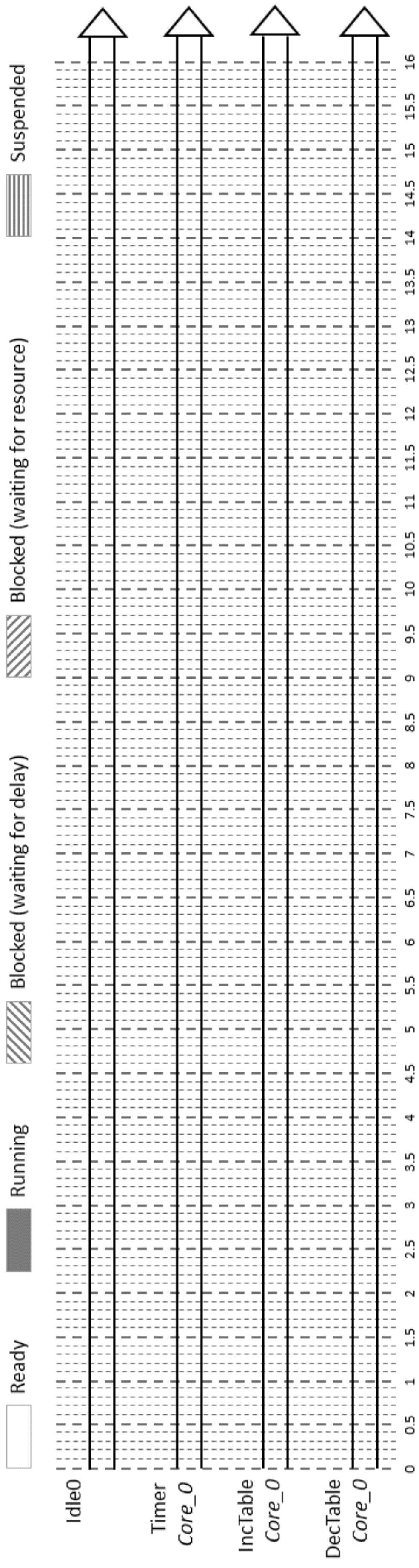

Figure 3.4 – *Scenario 2 : Priority(DecTable=4, IncTable=3), Core(DecTable=0, IncTable=0).*

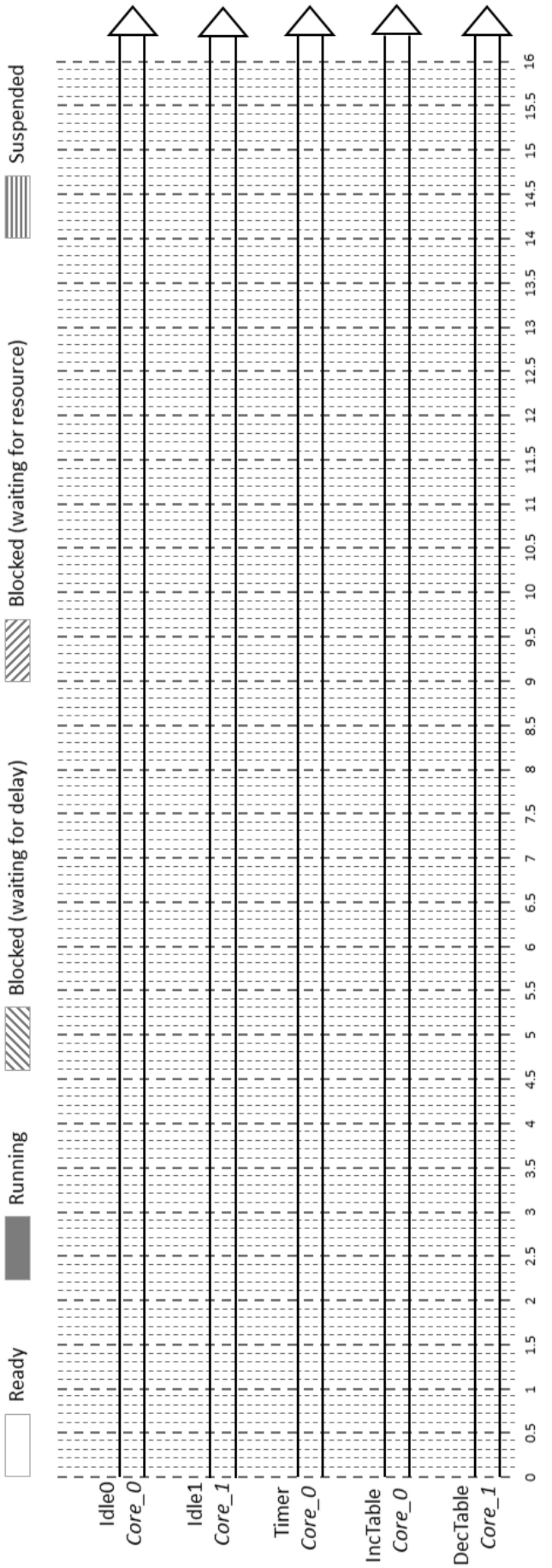

FIGURE 3.5 – *Scenario 3 : Priority(DecTable=4, IncTable=4), Core(DecTable=1, IncTable=0).*

3.4 Mutual Exclusion (Lab3-3)

We now add the *Inspector* task. We will perform the program with the task priorities described in the Table 3.2.

Timer task	IncTable task	DecTable task	Inspector task
Prio(5),Core(0)	Prio(3),Core(0)	Prio(3),Core(0)	Prio(4),Core(0)

Table 3.2 – *Task priorities with mutex.*

1. Duplicate the « lab3-2_two_sem_clk » folder to « lab3-3_mutex ».
2. Add the *Inspector* task.
3. Run the program, copy the console and explain the behavior. What is the problem ?
4. Correct the problem of initialization.
5. Run the program, copy the console and explain the behavior. What is the problem ?
6. Choose a new priority of the *Inspector* task to solve the problem.
7. Run the program, copy the console until 40 ticks, trace in the figure 3.6 the behavior of the 3 tasks until 40 ticks and explain the behavior.
8. Modify the *Inspector* task by adding a *Mutex* to manage access to the critical area. Below is a creation reminder for a Mutex.

```
/* Mutex */
SemaphoreHandle_t xSemMutex;
xSemMutex = xSemaphoreCreateMutex();
/* Using Mutex */
xSemaphoreGive(xSemMutex);
xSemaphoreTake(xSemMutex, portMAX_DELAY);
```

9. Run the program, copy the console, trace in the figure 3.7 the behavior of the 3 tasks until 40 ticks and explain the behavior and the effect of the Mutex.

10. Change the priority of *Inspector* task to 4. Run the program, copy the console until 40 ticks. What is the problem ?

11. We decide to change the *Inspector* task to *Core_ 1*. Run the program, copy the console until 40 ticks. What is the problem ?

12. We now decide to add a delay of 2 ticks after giving the mutex (using *vTaskDelay()* function) in the *Inspector* task. Run the program, copy the console, trace in the figure 3.8 the behavior of the 3 tasks and Mutex until 90 ticks and explain the behavior.