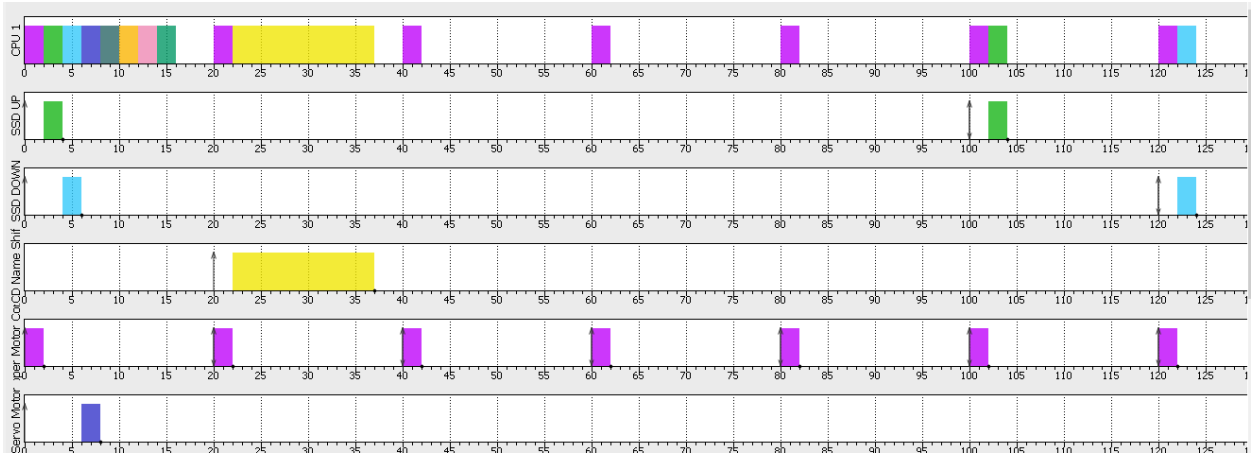
RTOS Tasks

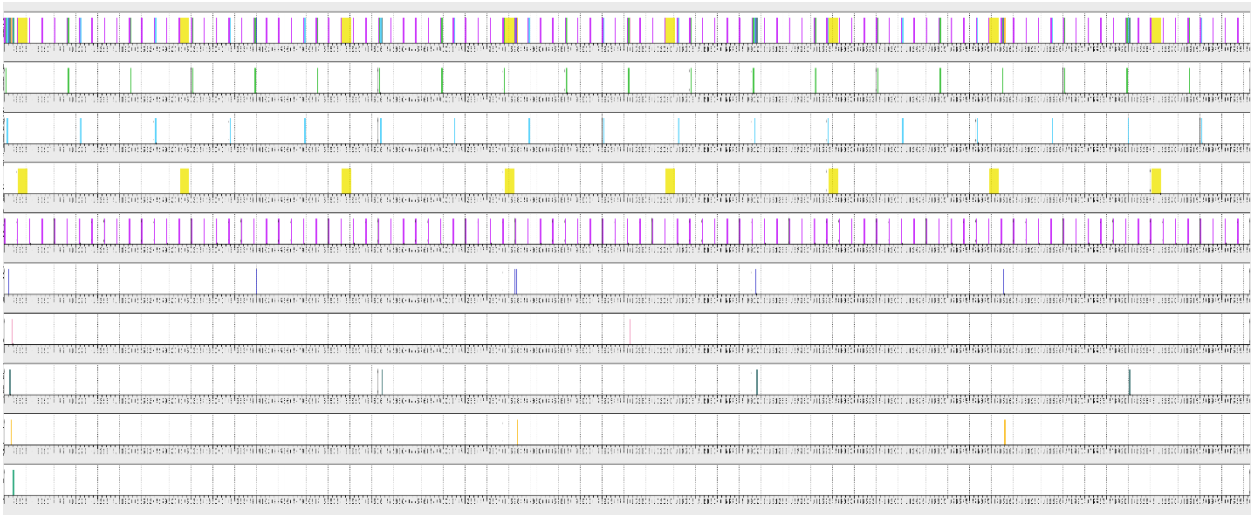
Prepared by:

Nour alaa eldin hanafy moharram

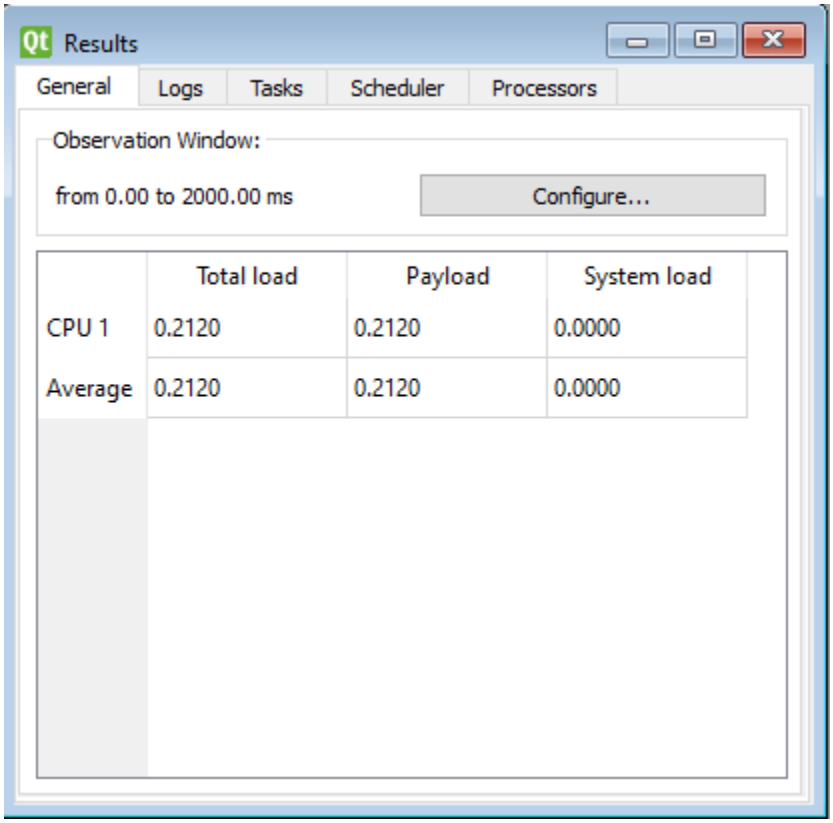
Task 1:

Timing diagram for 9 tasks:



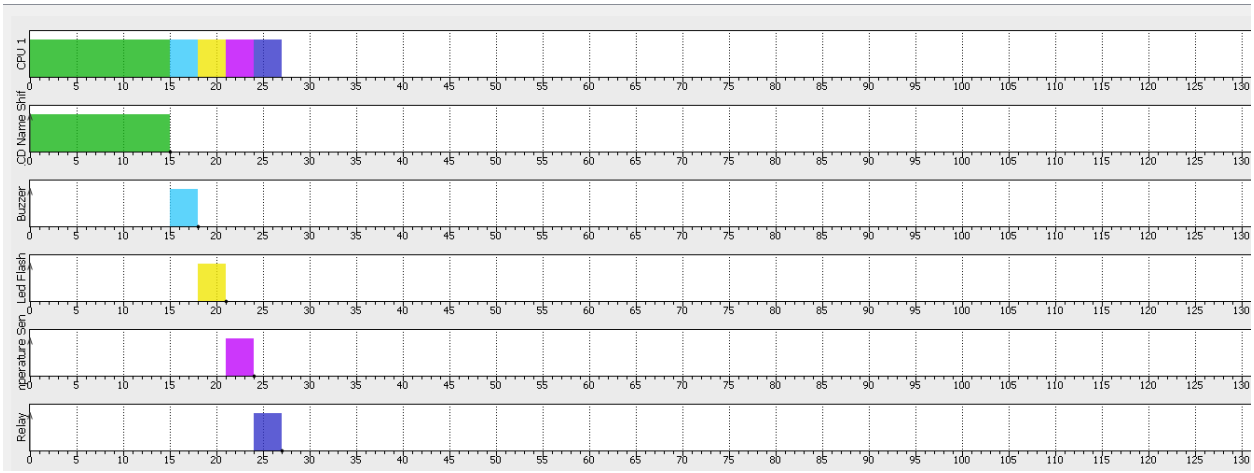


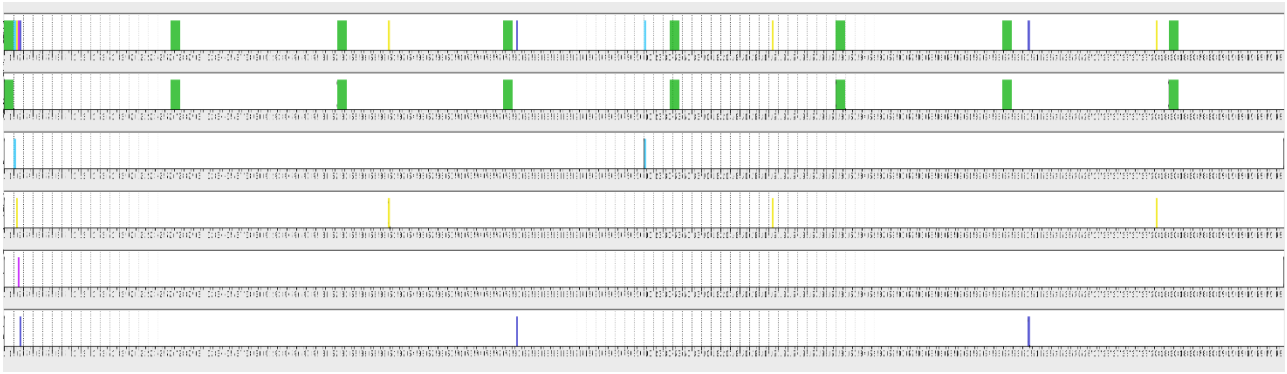
CPU Load for Task 1: (Total Executions time / tick time)*100=



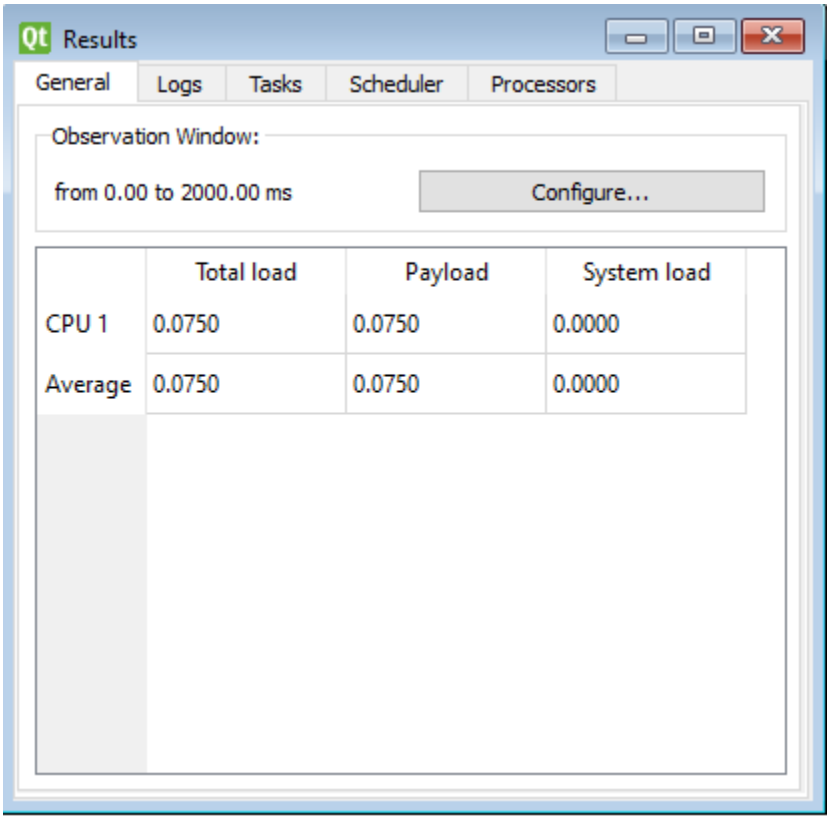
Task 2:

Timing Diagram:

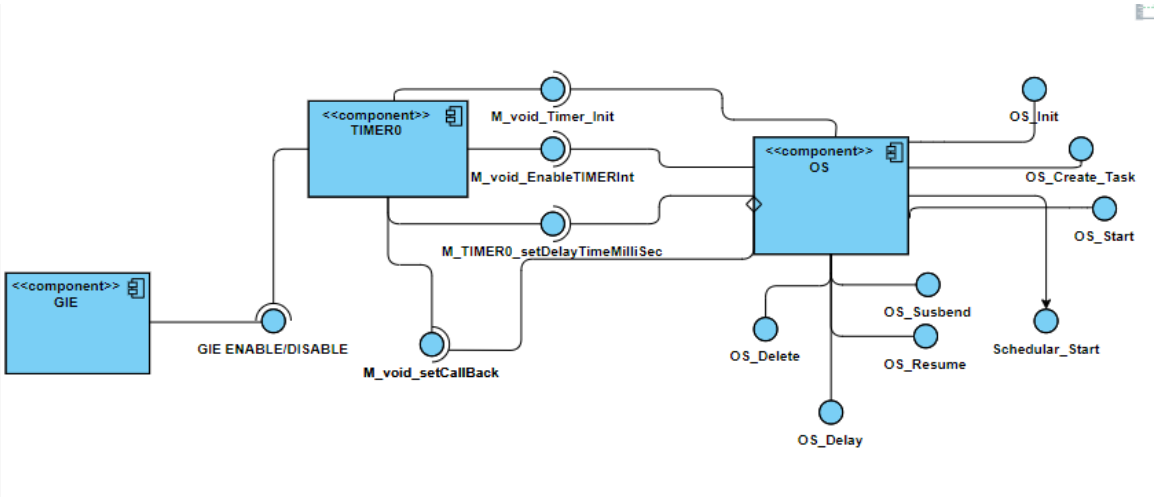




CPU Load for Task 2: (Total Executions time / tick time)*100=



Component Diagram:



Interfaces of OS_APIS:

1-

Syntax: OS_Init():

Description: This function will initialize the timer mode, enable the timer interrupt, enable the global interrupt and set the call back function.

Parameters:

- Inputs: void
- Outputs: void

2-

Syntax: OS_Create_Task():

Description: this function will save the created tasks in a specific queue in order to be ready for the scheduler process

Parameters:

- Inputs: pointer to the task data type (STRUCT)
- Outputs: void

3-

Syntax: Schedular_Start():

Description: this function will be called periodically at specific numbers of overflows to evaluate the tasks and determine which one will start execution or not.

Parameters:

- Inputs: void
- Outputs: void

4-

Syntax: OS_Start():

Description: this function is responsible for setting the tick time when the scheduler will be called to evaluate the tasks.

Parameters:

- Inputs: variable u16 that will carry the tick time required
- Outputs: void

5-

Syntax: OS_Suspend():

Description: this function change the status of the task from running to suspended in order to prevent it from execution.

Parameters:

- Inputs: u8 variable that identify the ID/Priority of the task
- Outputs: void

6-

Syntax: OS_Resume():

Description: this function will change the status of the task from suspended to ready to be able to resume it's execution according to its periodicity.

Parameters:

- Inputs: u8 variable that carry the ID/Priority of the required task.
- Outputs: void

7-

Syntax: OS_Delete():

Description: this function will delete a specific function.

Parameters:

- Inputs: u8 variable that carry the ID/Priority of the required task.

- Outputs: void

8-

Syntax: OS_Delay():

Description: this function will delay the periodicity of a specific function for a certain number of delays in addition to change its state from ready to suspend.

Parameters:

- Inputs:
    - u8 variable that carry the ID/Priority of the task
    - u32 variable that carry the number of delay ticks.
- Outputs: void

Task 3:

Determine the objective from the RTOS LABS:

- **Lab 1:**
    - Create two tasks each one toggle a led the tasks will execute periodically and without preemption.
- **Lab 2:**
    - Create two tasks each one toggle a led. The task with higher priority will be executed before the task with lower priority.
- **Lab 3:**

    - Create two tasks each one toggle a led. The task with higher priority will preempt the task with lower priority.
- **Lab 4:**
    - Create two tasks one that observe a push button and when pressed it will switch the direction of rotation of DC Motor.
- **Lab 5:**
    - Create two tasks one that observe a push button and then change the value of shared variable and the other task toggle led and change the value of the same variable.
- **Lab 6:**
    - The shared resource between the two tasks will be protected by using a flag.
- **Lab 7:**
    - The shared resource between the two tasks will be protected by disabling the scheduler for a certain time.
- **Lab 8:**
    - Use semaphore to guard the resource between the two tasks.
- **Lab 9:**
    - In case of two tasks share two resources with semaphores may cause a deadlock between the two tasks.
- **Lab 10:**

    - This lab observe the accident of priority inversion when a lower priority task with semaphore guarding a shared resource is executing and if a higher priority task tried to obtain this resource will go in a suspend state.
- **Lab 11:**
    - The solution for the priority inversion is to give the lower priority task that hold the semaphore its priority will be raise to the priority of the high priority that try to preempt it.
- **Lab 12:**
    - Semaphores can be used between tasks to establish a synchronization between them.

Task 4:

Description:

Basic explanation for the concepts of mailbox and priority celling.

Mailbox:

Message mailboxes

- A mailbox is a special memory location that one or more tasks can use to transfer data, or more generally for synchronization.
- The tasks rely on the kernel to allow them to
  - write to the mailbox via a post operation
  - Or read from it via a pend operation
- Direct access to any mailbox is not allowed
- A mailbox can only contain one message

Priority Celling:

An assumption is made that all the jobs in the system have a fixed priority. It does not fall into a deadlock state.

The chained blocking problem of the Priority Inheritance Protocol is resolved in the Priority Ceiling Protocol.

The properties of Priority Ceiling Protocols are:

1. Each of the resources in the system is assigned a priority ceiling.
2. The assigned priority ceiling is determined by the highest priority among all the jobs which may acquire the resource.

3. It makes use of more than one resource or semaphore variable, thus eliminating chain blocking.
4. A job is assigned a lock on a resource if no other job has acquired lock on that resource.
5. A job J, can acquire a lock only if the job's priority is strictly greater than the priority ceilings of all the locks held by other jobs.
6. If a high priority job has been blocked by a resource, then the job holding that resource gets the priority of the high priority task.
7. Once the resource is released, the priority is reset back to the original.
8. In the worst case, the highest priority job J_1 can be blocked by T lower priority tasks in the system when J_1 has to access T semaphores to finish its execution.