









Project 2 EDF Scheduler

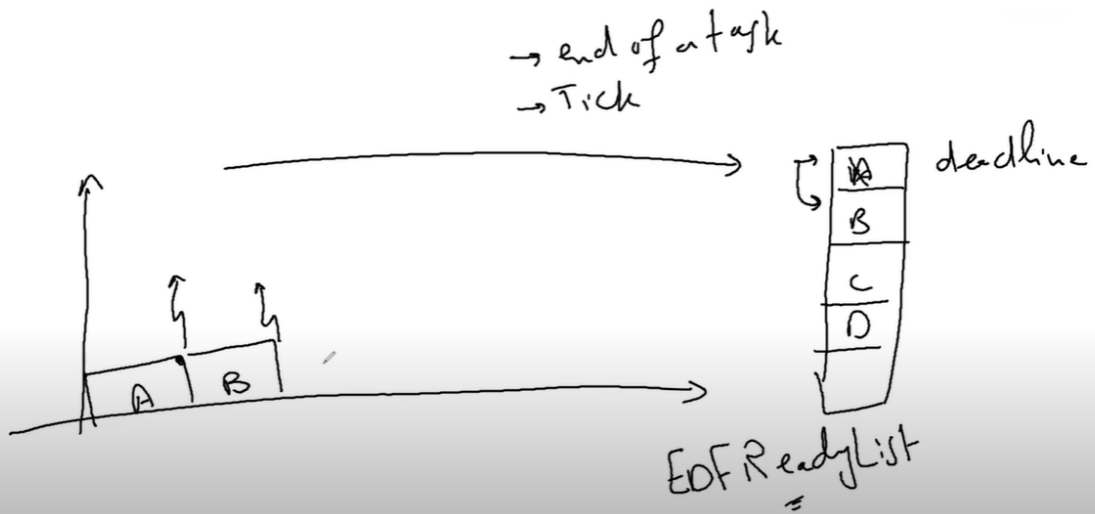
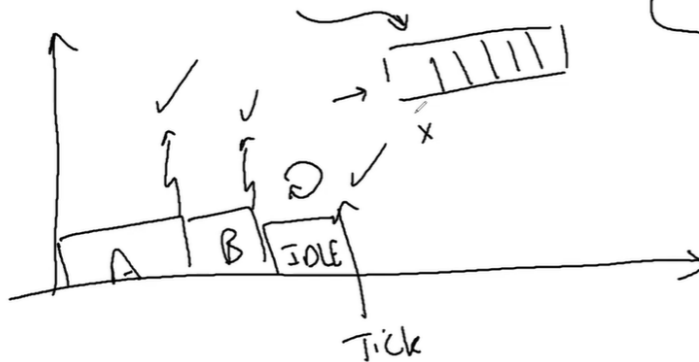
 Week	
 YouTube Link	https://classroom.udacity.com/nanodegrees/nd0900-fwd-t3/parts/c5e0f9f6-c4a0-462e-b4ca-619dfd99477a/modules/75f64c76-198c-405e-b1dd-0a5509fd2ca8/lessons/69b5bcfa-71a8-45c3-8973-2fc710f0f92e/concepts/0b3e4ee6-b7b0-4acd-b48b-f8e711144708 https://www.youtube.com/watch?v=d0AF1wZ2_yQ&t=9s
 Start Date	
 FileLink	
 Status	
 Notes	
 Other Sources Reference	
# Rem Esti Hrs	
 Property	

Project Analysis

EDF → Dynamic priority preemptive scheduler / freeRTOS

↳ priority → deadline

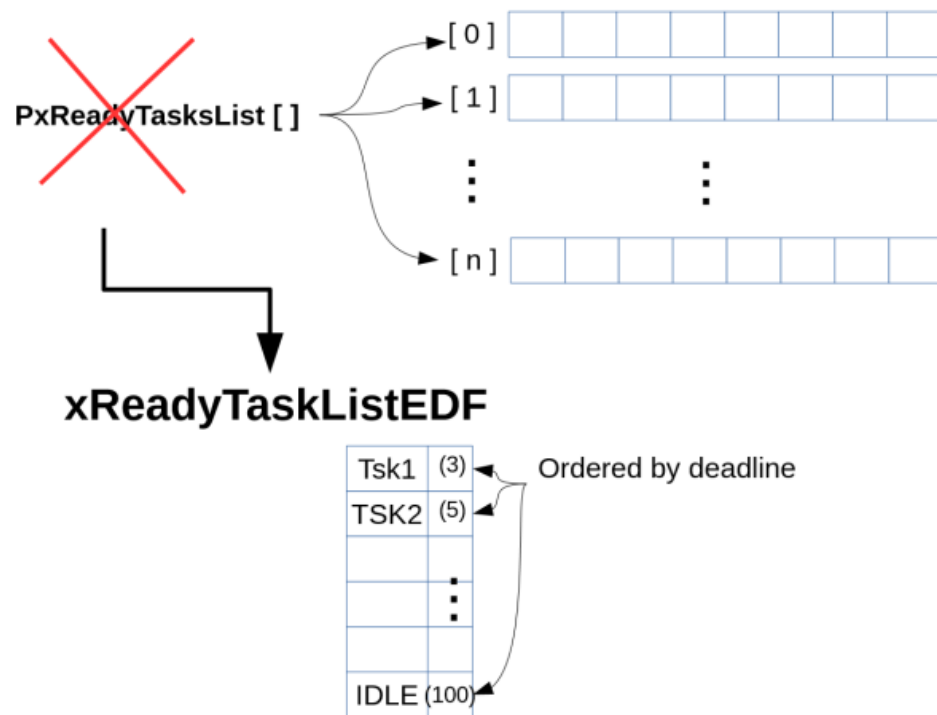
↳ preemptive
↳ fixed priority
↳ Round-Robin



Project Requirements:

- dynamic priority-based preemptive scheduling policy

- The general idea is to:
 - create a new Ready List (Figure 3.3), able to manage a dynamic task priority behavior:
 - it will contain tasks ordered by increasing deadline time, where positions in the list represent the tasks priorities.
 - The head of the list contains the task with the closest deadline



- The head of the list containing the running task.
- The rest of FreeRTOS architecture and structures, as the Waiting List and the clock mechanism are maintained with marginal changes.
- **Implementation assumptions :**
 - Periodic tasks only;
 - task deadline equal to task period;
 - only schedulable tasks set;
 - independent tasks only (no shared resources and no sync issues).

- When Task is moved from Delayed (Waiting List) to Ready List. DeadLine evaluation shall be calculated and insertion in Ready List shall be in the correct order based on deadline.
- Ex: Tick mechanism wakes tskZ removing it from Waiting List and adding it to Ready List, in the right position according to its new deadline
 - $Z_{deadline} = tick + Z_{period}$
- All changes that will be illustrated refer to tasks.c file
- Required configuration variables:
 - configUSE_EDF_SCHEDULER, is added to the FreeRTOS.h config file.
- Implement , the prvInitialiseTaskLists() method, that initialize all the task lists at the creation of the first task, is modified adding the initialization of xReadyTasksListEDF
- prvAddTaskToReadyList() method that adds a task to the Ready List → Very important as it's the one responsible about scheduling
- vListInsert() method is called to insert in xReadyTasksListEDF the task TCB pointer. The

item will be inserted into the list in a position determined by its item value

xGenericListItem

(descending item value order). So it is assumed that xGenericListItem contains the next task

deadline.

- when a task moves to the Ready List, the knowledge of its next deadline is needed in

order to insert it in the correct position.

- The deadline is calculated as: $TASK_{deadline} = tick_{cur} + TASK_{period}$, so every task needs to store its period value. A new variable is added in the tskTaskControlBlock structure (TCB):

The deadline is calculated as: $TASK_{deadline} = tick_{cur} + TASK_{period}$,

```

372  * the task. It is inserted at the end of the list.
373  */
374  */
375  #if configUSE_EDF_SCHEDULER == 0 /* E.C. : */
376  #define prvAddTaskToReadyList( pxTCB )
377  vListInsertEnd( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB
    )->xGenericListItem ) )
378  #else
379  #define prvAddTaskToReadyList( pxTCB ) /*xGenericListItem must contain the
    deadline value */ \
380  vListInsert( &(xReadyTasksListEDF), &( ( pxTCB )->xGenericListItem ) )

```

26 Chapter 3. EDF Scheduler

```

381 #endif

```

- `xTaskPeriodicCreate()` is a modified version of the standard method `xTaskGenericCreate()` shown in Cap.2, that receives the task period as additional input parameter and set the `xTaskPeriod` variable in the task TCB structure. Before adding the new task to the Ready List by calling `prvAddTaskToReadyList()`, the task's `xGenericListItem` is initialized to the value of the next task deadline.
- The **IDLE task management** is modified as well.
 - The initialization of the IDLE task happens in the `vTaskStartScheduler()` method, that:
 - starts the real time kernel tick processing
 - initialize all the scheduler structures.

Since FreeRTOS specifications want a task in execution at every instant, a correct management of the IDLE task is fundamental.

With the standard FreeRTOS scheduler, the IDLE task is a simple task initialized at the lowest priority.

In this way it would be scheduled only when no other tasks are in the ready state. With the

- EDF scheduler, the lowest priority behavior can be simulated by a task having the farrest deadline.
- Shall increment it in runtime as well Every time IDLE task executes (i.e. no other tasks are in the Ready List), it calls a method that increments its deadline in order to guarantee that IDLE task will remain in the last position of the Ready List (implemented)
- **vTaskStartScheduler()** method initializes the IDLE task and inserts it into the Ready List.
 - This MACRO listSET_LIST_ITEM_VALUE(&((pxNewTCB)->xGenericListItem), (pxNewTCB)->xTaskPeriod + currentTick); shall calculate deadline.
- taskSELECT HIGHEST PRIORITY TASK() method is replaced in order to assign to pxCurrentTCB the task at the first place of the new Ready List
- When the **tick** interrupt occurs, the ISR **vPortYeldFromTick()** is called. (not implmented)
 - Saves the running task context → portSAVECONTEXT()
 - xTaskIncrementTick()
 - if same tasks are wake up from the Waiting List (Not implmented)
 - performs a context switch → vTaskSwitchContext()
 - tskA's(new Task) GenericListlteam is set to tskA's(new Task) new deadline (currentTick + tskA period);
 - restores the context of the (tskA's) new running task.
 - *pxCurrentTCB pointer points to tskA;
 - portRESTORECONTEXT() restores the context
 - tskA is insered in xReadyTasksListEDF by calling the addTaskToReadyList() method

Missing Implementation for the project:

1- Increment IDLE Task Deadline to be the `max_align_t` value ever → (implementation in Function `static portTASK_FUNCTION(prvIdleTask, pvParameters)`) → DONE

2- **Evaluate Delayed List** by checking Current Tick. Then will decide if context Switch required if **YES**: → (implementation in Function `BaseType_t xTaskIncrementTick(void)`)

- Task will be removed from Delayed List → Remove MACRO implementation shall be adjusted for EDF
 - Remove from Event List → check also if this requires implementation adjustment
- Task will be added to ReadyList by **new DL Calculated**.
- Execute Context Switch based on DL information in ready list:
 - by setting `xSwitchRequired`

3-Check Condition for Context Switch shall not be Priority → shall be changed to deadline 1st → (implementation in Function `BaseType_t xTaskIncrementTick(void)`) or on return from it → May be implemented to get from the ReadyTask next one

Notes:

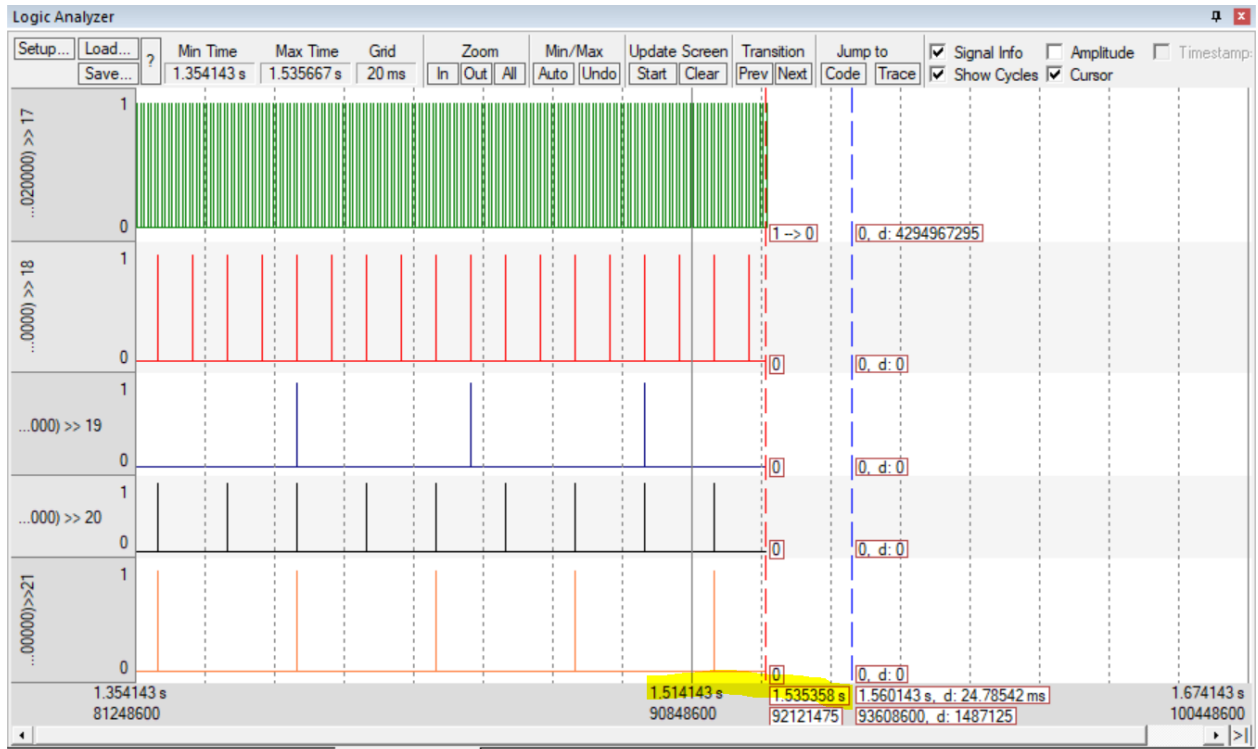
- **xGenericListItem** → **xStateListItem** name changed
- `xItemValue` → new calculated deadline
- `xItemValue` will be used by insert macro to decide location of Task in ReadyList

Project Completion Requirements:

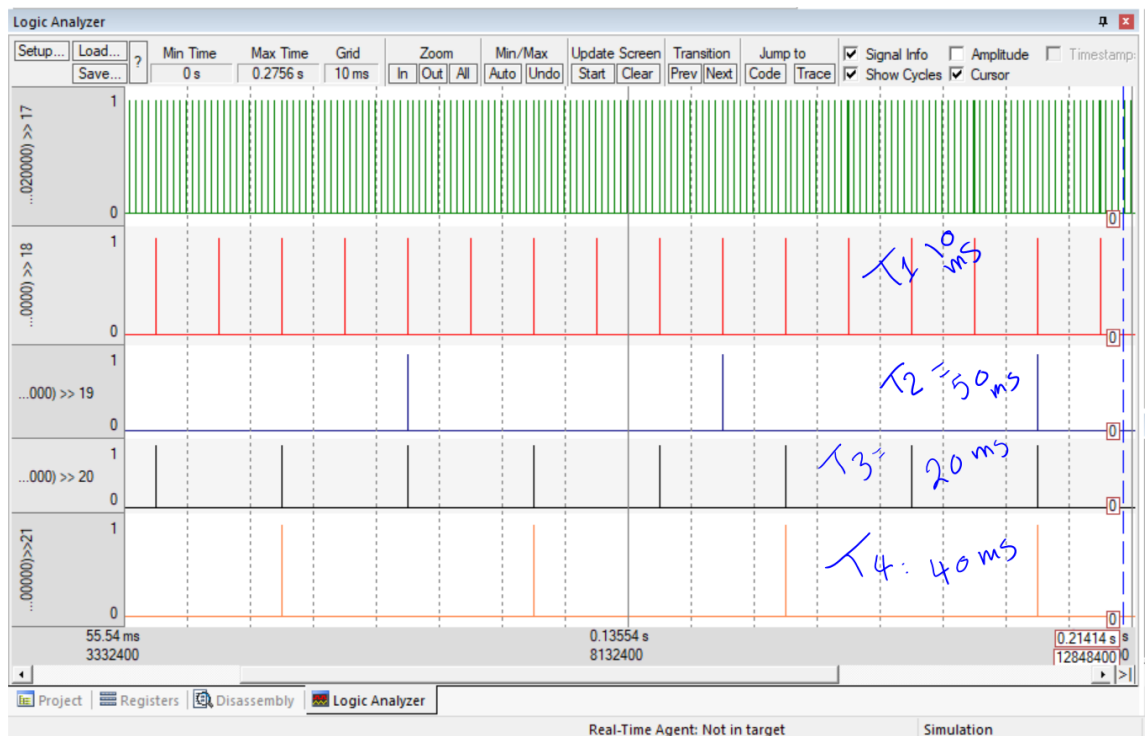
- Implement 4 tasks using EDF scheduler
 - TickHook → PIN1
 - Task1: (Periodicity: 10 , Priority: , ExecutionTime: 1 ms) → PIN2
 - Task2: (Periodicity: 50, Priority: , ExecutionTime: 8 ms) → PIN3

- Task3: (Periodicity: 20, Priority: , ExecutionTime: 2 ms) → PIN4
- Task4: (Periodicity: 40, Priority: , ExecutionTime: 5 ms) → PIN5
- 1. Verifying the system implementation with the following parameters:
 - Deadline
 - Periodicity
 - Priority
 - Execution Time → Add overload & Calculate by hooks
- Verify Schedulability with:
 - offline Simulator
 - Analytical analysis
 - Runtime → implement deadline Detector, & GetTime Counters and execution
 - Simple report of work and Comments
- Bonus → RTOS Runtime implementation update to consider Runtime analysis for EDF ReadyList

Actual RunTime Analysis (over 1.5 sec Period):

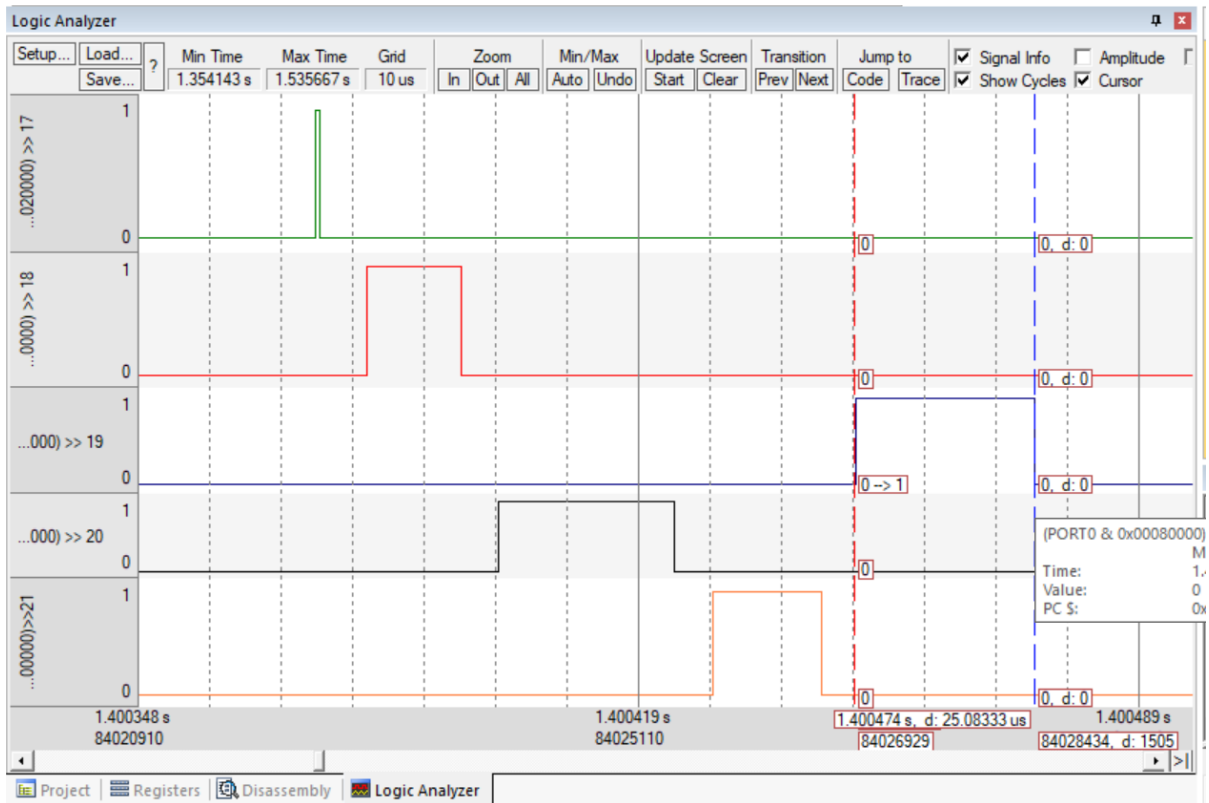


- Task Periodicity:



- Task1: (Actual Periodicity: 10) → PIN2
- Task2: (Actual Periodicity: 50) → PIN3
- Task3: (Actual Periodicity: 20) → PIN4
- Task4: (Actual Periodicity: 40) → PIN5
- **Hyper Period is 200 ms**
 - where all Tasks executions will be executed (Assuming same Task Start Offset).
 - by Simple measurements you can find Hyper Period is 200 ms

	A	B	C	D	E
14	Hyper Period				
15	Task name	Task 1	Task 2	Task 3	Task 4
16	Task Periodicity(ms)	10ms	50ms	20ms	40ms
17		Execution Time (us)			
18	0	13.3	24.6	T3	15.5
19	10	13.5			
20	20	13.5		25.15	
21	30	13.35			
22	40	13.9		25	15.78
23	50	13.38	24.9		
24	60	13.4		25.2	
25	70	13.6			
26	80	14		25	15.4
27	90	13.6			
28	100	T1	T2	T3	
29	110	T1			
30	120	T1		T3	T4
31	130	T1			
32	140	T1		T3	
33	150	T1	T2		
34	160	T1		T3	T4
35	170	T1			
36	180	T1		T3	
37	190	T1			
38	200	T1	T2	T3	T4
39	Max (us)	14	24.9	25.2	15.78



- **Worst case execution Time (using Keil Simulator):**

- Task1: 14 usec
- Task2: 24.9 usec
- Task3: 25.2 usec
- Task4: 15.78 usec

- **CPU Load Theoretical measurements**

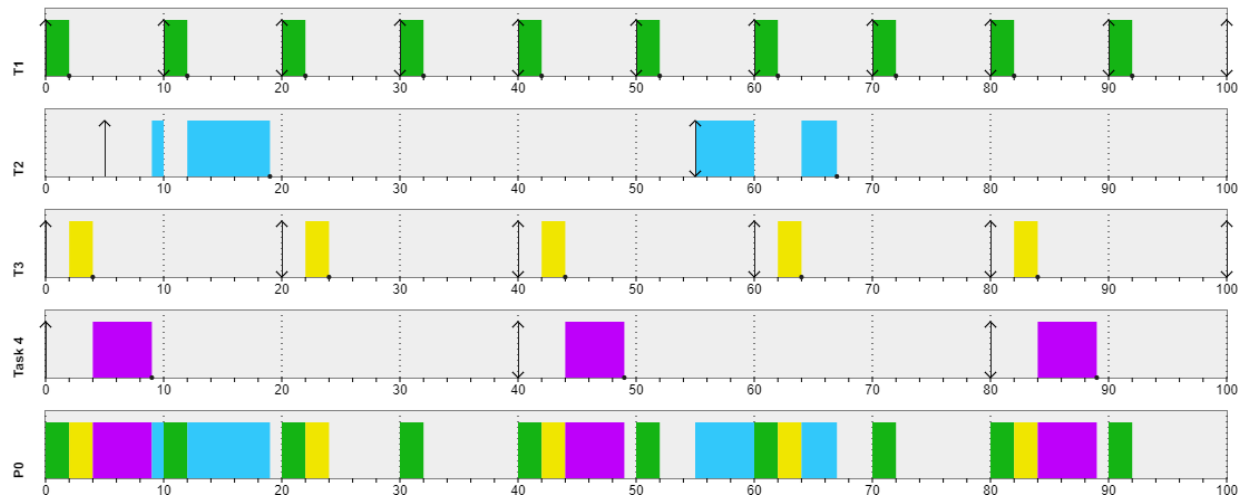
- CPU Load = 0.355 %

Task Name	Peridicity (ms)	Expected worst case Execution Time (usec)	Task execution Time during Hyper Period (200ms)
Task 1	10	14	0.28
Task 2	50	24.9	0.0996
Task 3	20	25.2	0.252
Task 4	40	15.78	0.0789

Total Execution During Hyper Period			0.7105
CPU load %			0.35525

Simulator Data (Offline Simulator):


- Simulator Estimation was very high



Results			
Start date (ms): <input type="text" value="0"/>		End date (ms): <input type="text" value="100"/>	
Gantt	General	Tasks	Processors
		Logs	Scheduler
Cpu	Total Load	Payload	System Load
P0	0.61	0.61	0.0
Average	0.61	0.61	0.0

Actual CPU Load Measurements (Using Keil Simulator):

- CPU Load = 0.341 %

 <code>cpu_load</code>	0.341816425	float
<code><Enter expression></code>		

