EDF PROJECT REPORT

1. Using analytical methods calculate the following for the given set of tasks:

Calculate the system hyperperiod

Our system has 6 tasks as follow

Task	Period	Execution time	Deadline
Button_1_Monitor	50	0.0233	50
Button_2_Monitor	50	0.0233	50
Periodic_Transmitter	100	0.16	100
Uart_Receiver	20	WCE = 4.92	20
Load_1_Simulation	10	5.01	10
Load_2_Simulation	100	12.075	100

These execution times was extracted from real time simulation from both trace hooks counter calculations and logic analyzer of (GPIO) pins.

According to periodicity we can see that all periods can be a fraction of the largest periodic task .

Largest periodic function is 100 ms.

Button_1_Monitor and Button_2_Monitor occur 2 times in 100 ms.

Uart_Receiver occur 5 times in 100 ms.

Load_1_Simulation occur 10 times in 100 ms.

So the hyperperiod is 100 ms in which all tasks have to execute at this time.

Calculate the CPU load:

	Occurrence in 100 ms	Execution time	Total execution time in hyper period	Percentage
Button_1_Monitor	2	0.0233	0.0466	0%
Button_2_Monitor	2	0.0233	0.0466	0%
Periodic_Transmitter	1	0.16	0.16	0%
Uart_Receiver	5	WCE = 4.9 ms	24.5	24.5%
Load_1_Simulation	10	5	50	50%
Load_2_Simulation	1	12	12	12%
Total			In WCE = 87	87%

Note* this calculation depends on the worst case execution time (WCET) as the function Uart_Receiver not always takes 4.9 ms at the worst case the function will wait until all queue "3" messages will be sent ,and if there is no messages in QUEUE it will not wait to send all messages and it will return immediate ……

During real time simulation the average CPU load is 73%.

Check system schedulability using URM and time demand analysis techniques (Assuming the given set of tasks are scheduled using a fixed priority rate -monotonic scheduler)

1- By URM

The total utilization (U) = Sum(Execution time / Period)
= (0.0233/50) + (0.0233/50) + (0.16/100) + (4.9/20) + (5/10) + (12/100)
= 0.000466 + 0,000466 + 0.0016+0.245+0.5+0.12 = 0.86753 (it is < 1)

URM (Ulub) = 6(2^(1/6) - 1) = 0.734
According to the calculation the total utilization time (0.86799) > URM (0.734) so the system (may or may not be schedulable) and cannot be grantee as schedulable and we need more analysis

2- By Time demand analysis

Schedulable system is grantee when Time provided > Time demand

Time Provided is the hyperperiod = 100 ms

The time demand for each task according to priority "However priority depends on deadline"

1- Load_1_Simulation

W(10)(Load_1_Simulation) = 5 + 0 = 5 ms… < 10 ms so the system is feasible for Load_1_Simulation task

2- Uart_Receiver

W(20)(Uart_Receiver) = 4.9 + (20/10)*5 = 14.9ms …. <20ms so the system is feasible for Uart_Receiver task

3- Button_1_Monitor

W(50)(Button_1_Monitor) = 0.0233+(50/50)*0.0233+ (50/10)*5+(50/20)*4.9 = 0.0466 + 5*5 + 2.5-> 3*4.9 = 39.466 ms…. < 50 ms so the system is feasible for Button_1_Monitor task

4- Button_2_Monitor

W(50)(Button_2_Monitor) = 0.0233+(50/50)*0.0233+ (50/10)*5+(50/20)*4.9 = 0.0466 + 5*5 + 2.5-> 3*4.9 = 39.466 ms…. < 50 ms so the system is feasible for Button_2_Monitor task

5- Periodic_Transmitter

W(100)(Periodic_Transmitter) = 0.16 + (100/100)*12 + (100/50)*0.0233 + (100/50)*0.0233 + (100/10)*5 + (100/20)*4.9 = 0.16 + 12 + 0.0466 + 0.0466 + 50 + 24.5 = 86.753ms…. < 100 ms so the system is feasible for Periodic_Transmitter task

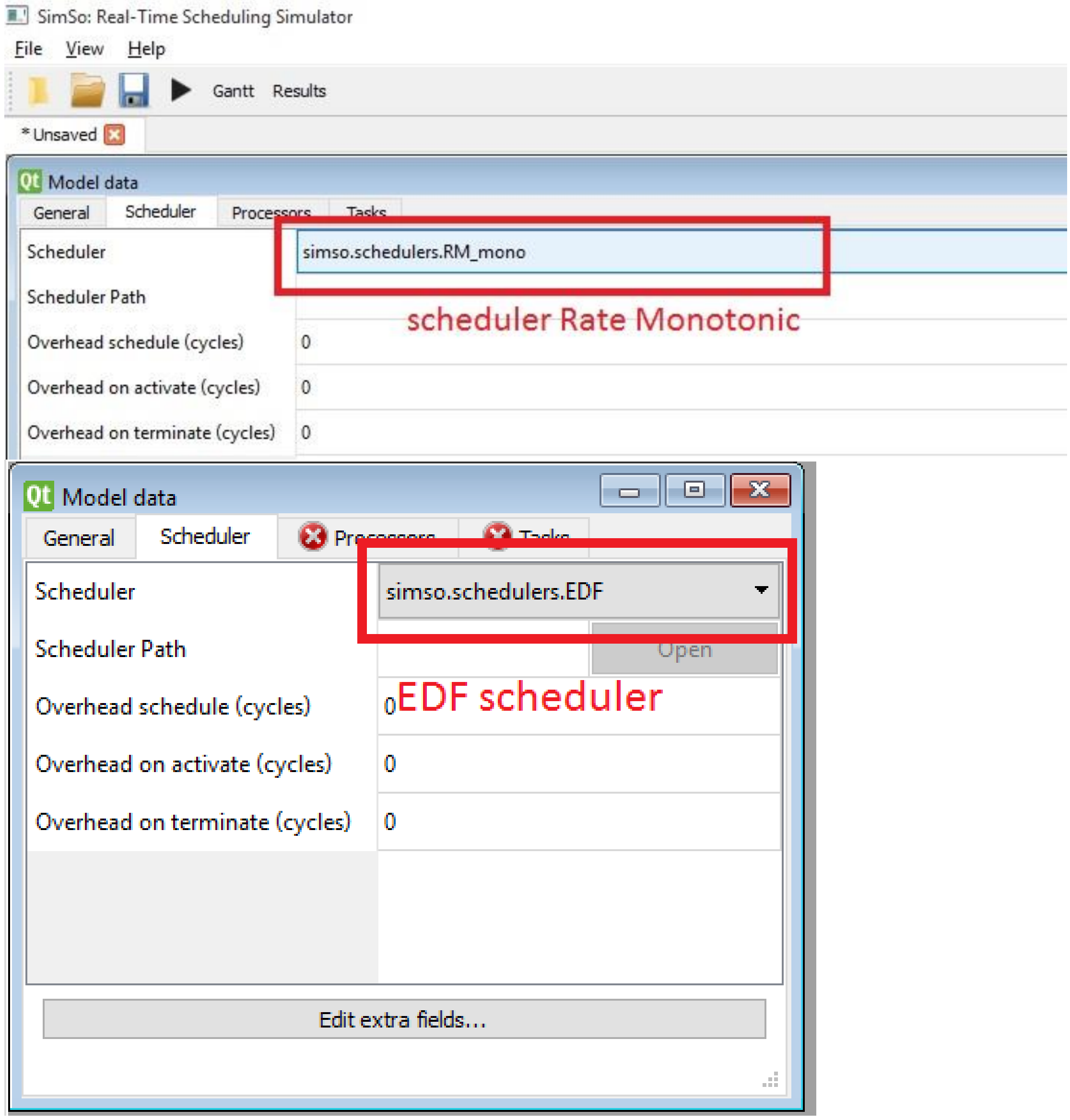
6- Load_2_Simulation

W(100)(Load_2_Simulation) = 12 + (100/100)*0.16 + (100/50)*0.0233 + (100/50)*0.0233 + (100/10)*5 + (100/20)*4.9 = 12+ 0.16 + 0.0466 + 0.0466 + 50 + 24.5 = 86.753 ms…. < 100 ms so the system is feasible for Load_2_Simulation task

In conclusion by analytic method the system is schedulable

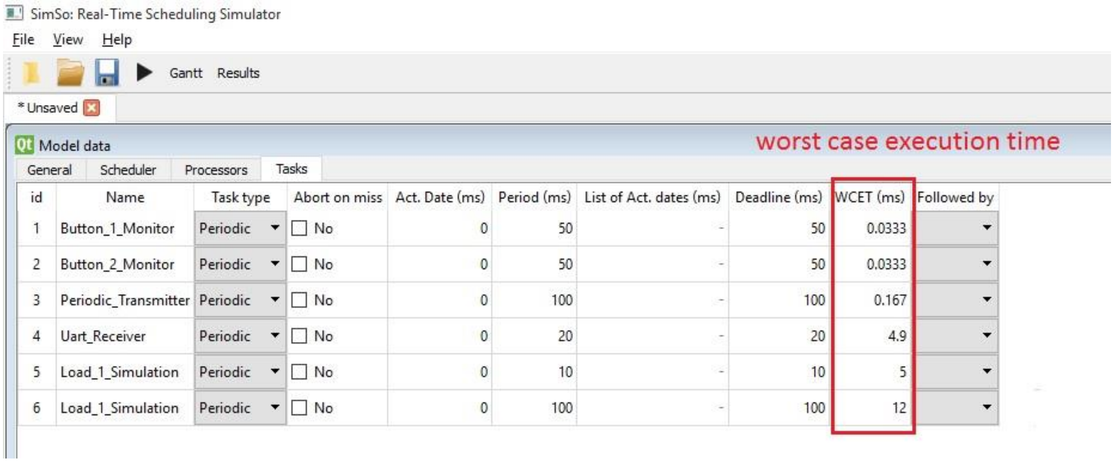
2. Using Simso offline simulator, simulate the given set of tasks assuming: Fixed priority rate monotonic scheduler

1- Configuration



The scheduler we choose either Rate Monotonic or EDF both schedulers give the same results.

2- Tasks

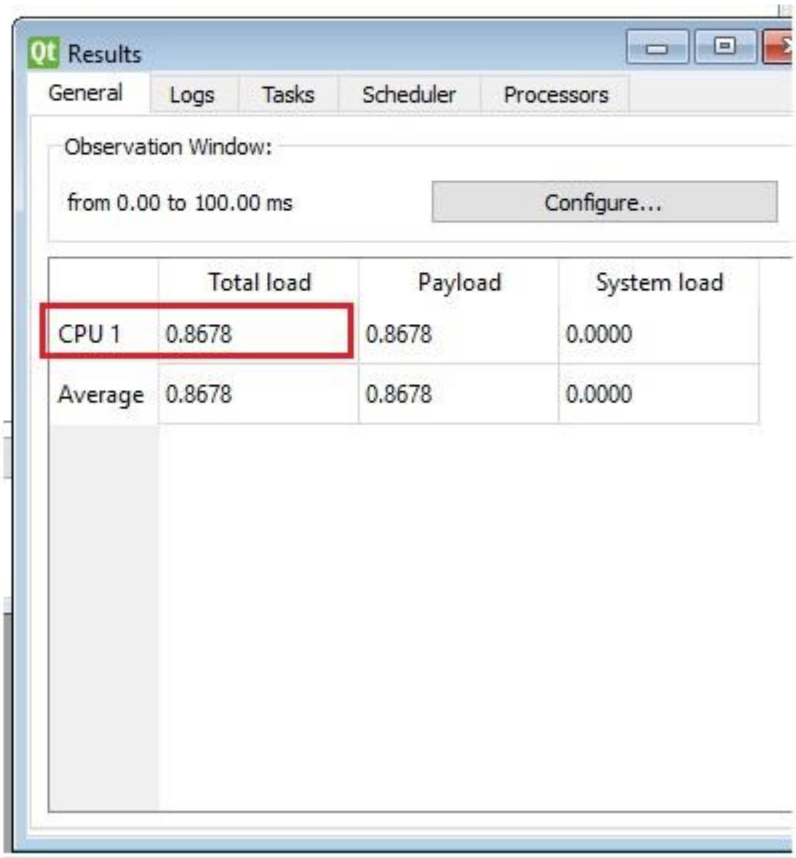


id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)	Followed by
1	Button_1_Monitor	Periodic	No	0	50	-	50	0.0333	
2	Button_2_Monitor	Periodic	No	0	50	-	50	0.0333	
3	Periodic_Transmitter	Periodic	No	0	100	-	100	0.167	
4	Uart_Receiver	Periodic	No	0	20	-	20	4.9	
5	Load_1_Simulation	Periodic	No	0	10	-	10	5	
6	Load_1_Simulation	Periodic	No	0	100	-	100	12	

We entered the task parameters as shown and we entered worst execution time "WCET" according to real time simulation (data extracted from keil simulation from logic analyser mainly and other methods so please refer to section >> 3. Using Keil simulator in run-time and the given set of tasks)

3- Result

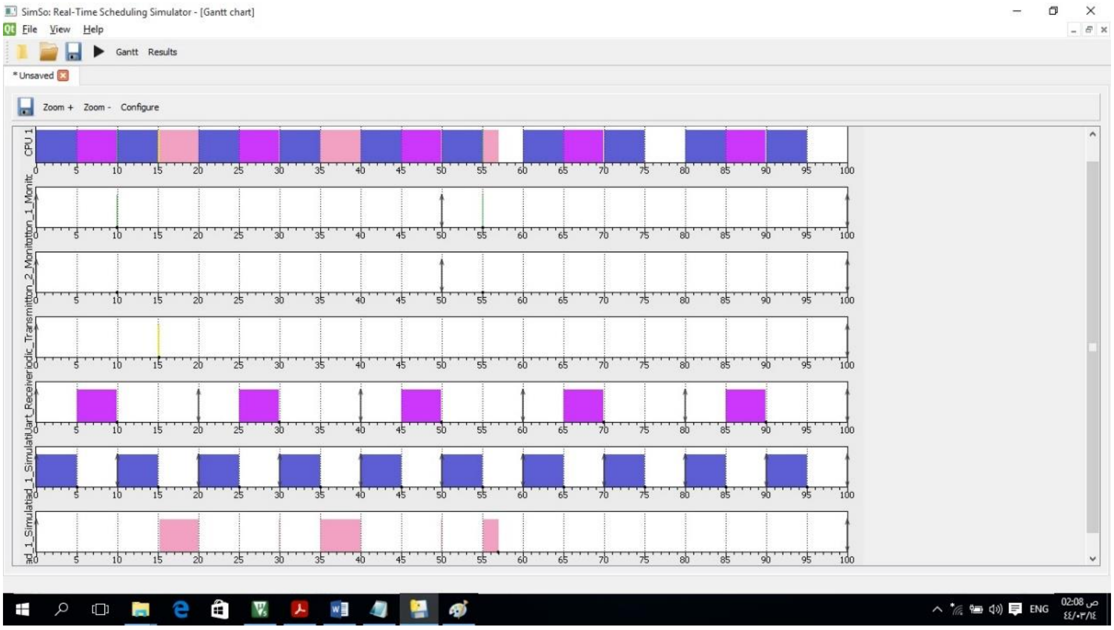
a- CPU load



	Total load	Payload	System load
CPU 1	0.8678	0.8678	0.0000
Average	0.8678	0.8678	0.0000

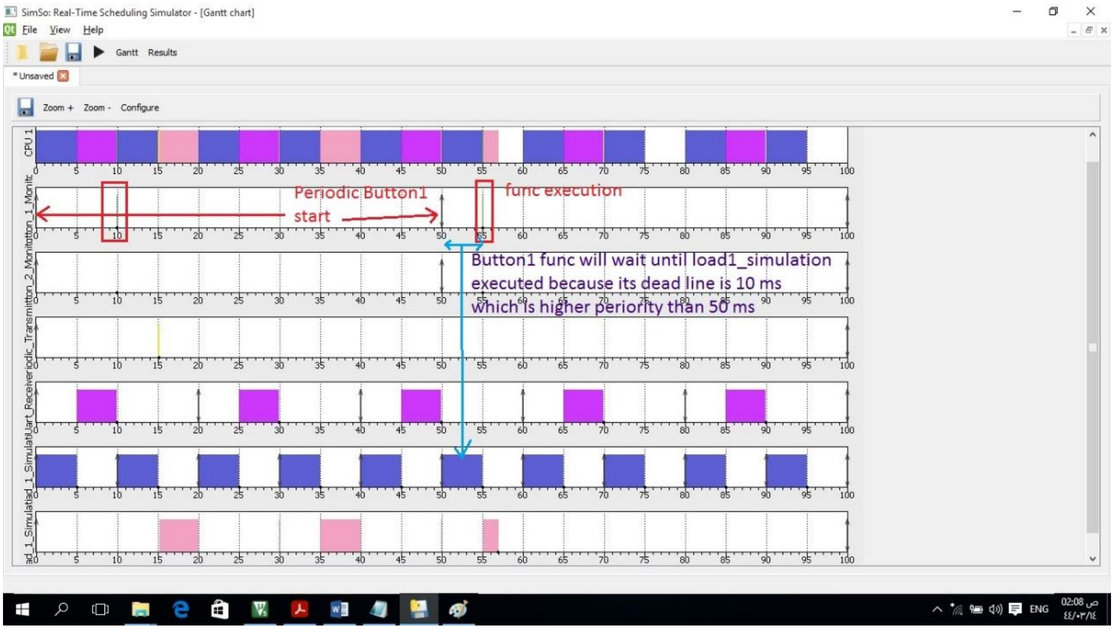
As we can see that CPU load is in the range of 86.8% in WCET of all tasks.

b- Chart simulation "Gantt"



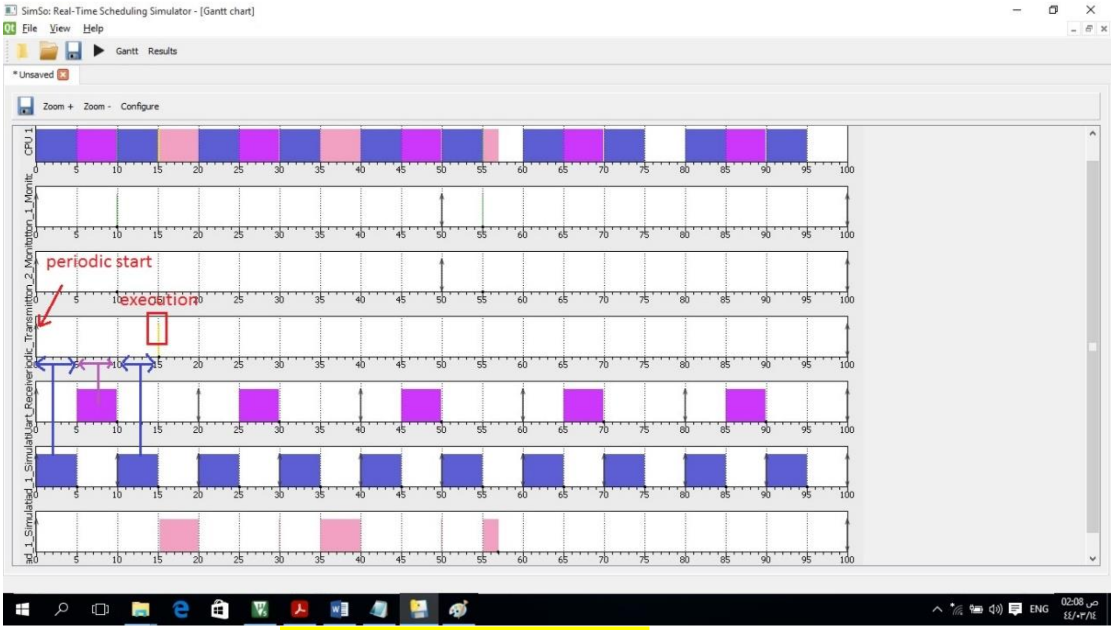
As we can see all tasks don't break their deadlines

*As regard Button1 and Button2 tasks they are very short period occurring 2 times in 100 ms scale and didn't breaks its dead line



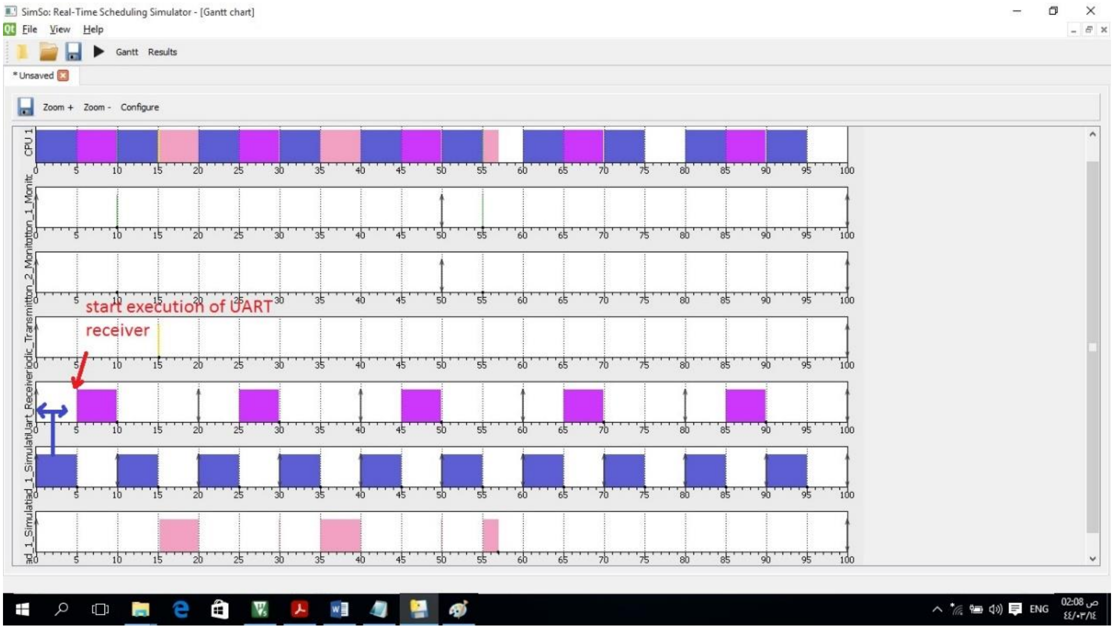
From the chart its execution will be delayed until load1simulation task end its work which will take 5 ms because its dead line is 10 ms < 50 ms of the button task deadline.

*as regard periodic_transmitter it occurs at period of 100 ms with WCET 0.167 ms and didn't missed the dead line

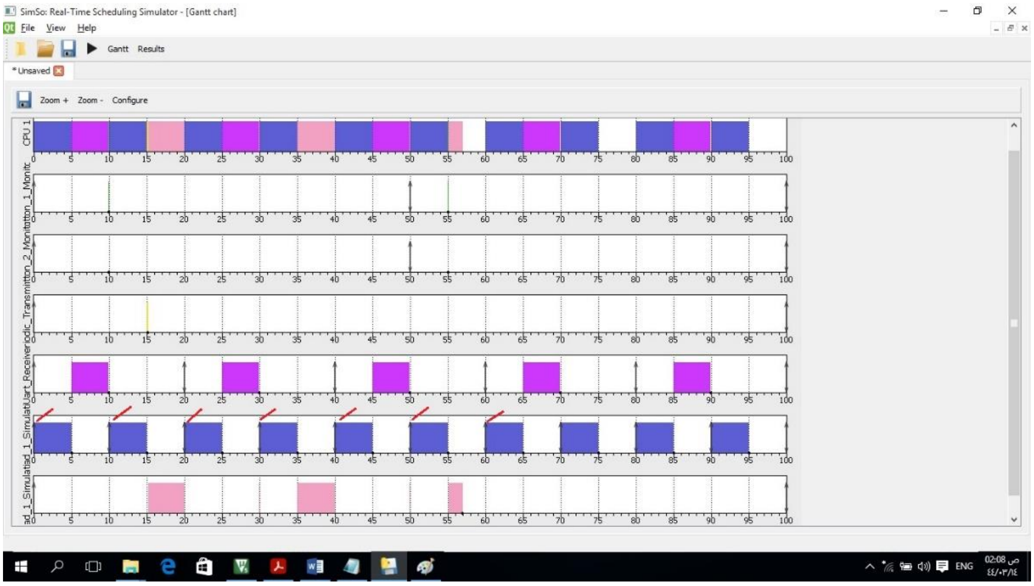


As we can see the periodic transmitter func execution will be delayed for 15 ms this is because its deadline is 100 ms seconds so it will be delayed until load1 func (10 ms deadline) executed and finished then UART receiver func (20 ms dedline) executed and fished, again load1 func .

*as regard UART receiver func (20 ms deadline) it will be executed immediately (5 ms) after load1 func (10 ms deadline) because the priority depends on deadline and it didn't break its deadline

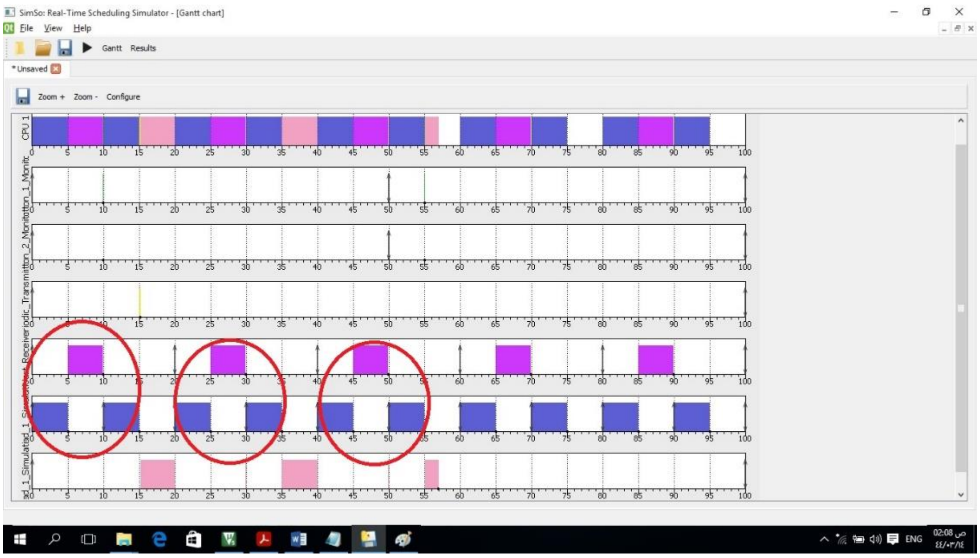


*as regard load1 func (10 ms deadline) it will preempt any other function and will delay all functions until its execution and its very clear preemption of the load2 func (100 ms deadline) at different time frame .



As we can see no delay in execution of the function every time because it is the least deadline

*as regard load2 func (100 ms deadline and execution time of 12 ms) it will be delayed and preempted by all other functions and it will finish its execution time before deadline



In conclusion according to SIMSO simulation program all tasks didn't break their deadlines and the nearest deadline will preempt other tasks with high cpu load (86.8%) and the system is schedulable.

3. Using Keil simulator in run-time and the given set of tasks:

1- Implementing trace and Tick macro hooks and GPIO

In the FreeRTOSConfig.h file

```
main.c | serial.c | GPIO.h | GPIO.c | tasks.c | GPIO_cfg.c | FreeRTOSConfig.h | task.h | list.c | list.h | poi
 * See http://www.freertos.org/a00110.html
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION      1
#define configUSE_IDLE_HOOK       0
#define configUSE_TICK_HOOK       1
#define configCPU_CLOCK_HZ        ( ( unsigned long ) 60000000 )  /* =12.0MHz xtal multiplied by 5 usin
#define configTICK_RATE_HZ        ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES      ( 4 )
#define configMINIMAL_STACK_SIZE  ( ( unsigned short ) 90 )
#define configTOTAL_HEAP_SIZE     ( ( size_t ) 13 * 1024 )
#define configMAX_TASK_NAME_LEN   ( 8 )
#define configUSE_TRACE_FACILITY   1  ////////////////////////////////////////
#define configUSE_16_BIT_TICKS     0
#define configIDLE_SHOULD_YIELD    0

#define IDLE_PERIOD                200   // this is the idle period and must be larger than any other
```

We will set the macros configUSE_TICK_HOOK and configUSE_TRACE_FACILITY to 1 to enable idle hook and traceability

```
////////////////////////////////////////////////////////
//  Task Tags and trace hooks
#define configUSE_APPLICATION_TASK_TAG     1

#define traceTASK_SWITCHED_IN()       do\
                                        {switch((int)pxCurrentTCB->pxTaskTag){\
                                          case Button_1_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_1_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_1_Monitor_TimeStampIn] = T1TC;\
                                            break;\
                                          case Button_2_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_2_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_2_Monitor_TimeStampIn] = T1TC;\
                                            break;\
```

Also we will set the (configUSE_APPLICATION_TASK_TAG) to 1 to give tasks name tags to trace each task

```
#define traceTASK_SWITCHED_IN()       do\
                                        {switch((int)pxCurrentTCB->pxTaskTag){\
                                          case Button_1_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_1_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_1_Monitor_TimeStampIn] = T1TC;\
                                            break;\
                                          case Button_2_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_2_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_2_Monitor_TimeStampIn] = T1TC;\
                                            break;\
                                          case Periodic_Transmitter_Tag:\
                                            GPIO_write(TaskMonitor,Periodic_Transmitter_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Periodic_Transmitter_TimeStampIn] = T1TC;\
                                            break;\
                                          case Uart_Receiver_Tag:\
                                            GPIO_write(TaskMonitor,Uart_Receiver_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Uart_Receiver_TimeStampIn] = T1TC;\
                                            break;\
                                          case Load_1_Simulation_Tag:\
                                            GPIO_write(TaskMonitor,Load_1_Simulation_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Load_1_Simulation_TimeStampIn] = T1TC;\
                                            break;\
                                          case Load_2_Simulation_Tag:\
                                            GPIO_write(TaskMonitor,Load_2_Simulation_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Load_2_Simulation_TimeStampIn] = T1TC;\
                                            break;\
                                          case Idle_Tag:\
                                            GPIO_write(TaskMonitor,Idle_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Idle_TimeStampIn] = T1TC;\
                                            break;\
                                              }\
                                            }while(0)
```

```
#define traceTASK_SWITCHED_OUT()     do\
                                        {switch((int)pxCurrentTCB->pxTaskTag){\
                                          case Button_1_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_1_Monitor_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Button_1_Monitor_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Button_1_Monitor_EcutionTime] = TaskTraceArr[Button_1_Monitor_TimeStampOut]-TaskTraceArr[Butt
                                            TaskTraceArr[Button_1_Monitor_TotalEcutionTime] += TaskTraceArr[Button_1_Monitor_EcutionTime];\
                                            break;\
                                          case Button_2_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_2_Monitor_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Button_2_Monitor_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Button_2_Monitor_EcutionTime] = TaskTraceArr[Button_2_Monitor_TimeStampOut]-TaskTraceArr[Butt
                                            TaskTraceArr[Button_2_Monitor_TotalEcutionTime] += TaskTraceArr[Button_2_Monitor_EcutionTime];\
                                            break;\
                                          case Periodic_Transmitter_Tag:\
                                            GPIO_write(TaskMonitor,Periodic_Transmitter_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Periodic_Transmitter_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Periodic_Transmitter_EcutionTime] = TaskTraceArr[Periodic_Transmitter_TimeStampOut]-TaskTrace
                                            TaskTraceArr[Periodic_Transmitter_TotalEcutionTime] += TaskTraceArr[Periodic_Transmitter_EcutionTime];\
                                            break;\
                                          case Uart_Receiver_Tag:\
                                            GPIO_write(TaskMonitor,Uart_Receiver_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Uart_Receiver_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Uart_Receiver_EcutionTime] = TaskTraceArr[Uart_Receiver_TimeStampOut]-TaskTraceArr[Uart_Recei
                                            TaskTraceArr[Uart_Receiver_TotalEcutionTime] += TaskTraceArr[Uart_Receiver_EcutionTime];\
                                            break;\
                                          case Load_1_Simulation_Tag:\
                                            GPIO_write(TaskMonitor,Load_1_Simulation_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Load_1_Simulation_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Load_1_Simulation_EcutionTime] = TaskTraceArr[Load_1_Simulation_TimeStampOut]-TaskTraceArr[Lc
                                            TaskTraceArr[Load_1_Simulation_TotalEcutionTime] += TaskTraceArr[Load_1_Simulation_EcutionTime];\
                                            break;\
                                          case Load_2_Simulation_Tag:\
                                            GPIO_write(TaskMonitor,Load_2_Simulation_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Load_2_Simulation_TimeStampOut] = T1TC;\
```

We defined both trace TASK_SWITCHED_IN() & traceTASK_SWITCHED_OUT() . this function like macros are prone for errors and it is better to define it as a function call however we want to make the least time delay to accurately calculate the performance of the system

```
#define traceTASK_SWITCHED_IN()      do\
                                        {switch((int)pxCurrentTCB->pxTaskTag){\
                                          case Button_1_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_1_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_1_Monitor_TimeStampIn] = T1TC;\
                                            break;\
                                          case Button_2_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_2_Monitor_PIN,PIN_IS_HIGH);\
                                            TaskTraceArr[Button_2_Monitor_TimeStampIn] = T1TC;\
                                            break;\
                                          case Periodic_Transmitter_Tag:\
```

going through task tags
if the function tag is ...
make gpio specifc pin high
store the current timer1 counter
inside the trace array

```
#define traceTASK_SWITCHED_OUT()     do\
                                        {switch((int)pxCurrentTCB->pxTaskTag){\
                                          case Button_1_Monitor_Tag:\
                                            GPIO_write(TaskMonitor,Button_1_Monitor_PIN,PIN_IS_LOW);\
                                            TaskTraceArr[Button_1_Monitor_TimeStampOut] = T1TC;\
                                            TaskTraceArr[Button_1_Monitor_EcutionTime] = TaskTraceArr[Button_1_Monitor_TimeStampOut]-TaskTraceArr[Button_
                                            TaskTraceArr[Button_1_Monitor_TotalEcutionTime] += TaskTraceArr[Button_1_Monitor_EcutionTime];\
                                            break;\
                                          case Button_2_Monitor_Tag:\
```

going though task tags
if current taks tag is ..
make specifc pin low
read current timer1 counts
calculate difference
accumulate this calacultions

Inside FreeRTOSConfig.h after we have to include my "TraceDef.h" which have defintions .. TaskTraceArr[] present inside main.c

In Main.c file

```c
////////////////////////////////////////////
uint32_t TaskTraceArr[MAX_TRACE_NUMB];

uint32_t Button_1_Monitor_ExcutionTime ,Button_2_Monitor_ExcutionTime,Periodic_Transmitter_ExcutionTime,
          Uart_Receiver_ExcutionTime , Load_1_Simulation_ExcutionTime , Load_2_Simulation_ExcutionTime;

uint8_t CPU_Load,Button_1_Monitor_LoadPercent,Button_2_Monitor_LoadPercent,Periodic_Transmitter_LoadPercent,
        Uart_Receiver_LoadPercent,Load_1_Simulation_LoadPercent,Load_2_Simulation_LoadPercent;

///////////////////////////////////////////////
#define WCETcounterThreshold  20    // wait 20 cycle befor start sampling periodic excution time
int WCET_counter =0;
int WCET_Monitor = 0;   // to monitor max periodic tasks load for selecting item = worst case excustion time
///////////////////////////////
```

Defining TaskTraceArr[] , Button_1_Monitor_ExcutionTime ,Button_2_Monitor_ExcutionTime, ………
And CPU_Load,Button_1_Monitor_LoadPercent, Button_2_Monitor_LoadPercent, ……..
Variables to monitor them during run-time of the system both we will obtain absolute and percentage values

Also we define WCET_Monitor variable which stands for worst case execution time to monitor specific task WCET according to hash defined macros inside the "TraceDef.h"

```c
//
void Button_1_Monitor(void* pvparams){
  vTaskSetApplicationTaskTag( NULL, (void*) Button_1_Monitor_Tag );     // giving tag number to the task


```

We will give each task its specific tag number

```c
// used to calculate worest case excution time and periodic excution time
#if (configTaskExcutionTimePeriodic ==1)
{
  #ifdef  Task1_WCET
  if(WCET_counter >= WCETcounterThreshold){
    if(WCET_Monitor < TaskTraceArr[Button_1_Monitor_TotalEcutionTime]){
      WCET_Monitor = TaskTraceArr[Button_1_Monitor_TotalEcutionTime];
      WCET_counter = WCETcounterThreshold;
      }}
  WCET_counter++;
  #endif
  TaskTraceArr[Button_1_Monitor_TotalEcutionTime] =0;
  }

#endif

  }
}
```

To calculate WCET for specific task you have to set " configTaskExcutionTimePeriodic" and " Task1_WCET" macros to 1 inside "TraceDef.h" file
As you can see we will not sample WCET except after counting n number of task execution because during start execution of simulator it will pass very large number to WCET_Monitor variable which is not correct

Inside "TraceDef.h"

```c
#define MAX_TRACE_NUMB   35

#define TaskMonitor    PORT_0
```

defining numb of elements in trace array

we will monitor tasks through PORT0

```c
////////////////////////
// INPUTS BUTTONS
#define Button1_in   PIN0
#define Button2_in   PIN1
////////////////////////
```

Defining input pins

```c
// OUTPUT MONITORING
#define Button_1_Monitor_PIN         PIN2
#define Button_2_Monitor_PIN         PIN3
#define Periodic_Transmitter_PIN     PIN4
#define Uart_Receiver_PIN            PIN5
#define Load_1_Simulation_PIN        PIN6
#define Load_2_Simulation_PIN        PIN7
#define Idle_PIN                     PIN8
#define Tick_PIN                     PIN9
```

defining output pins

```c
//////////////////////////////////////////////
// Task Tags that will be traced during running
#define Idle_Tag                     0
#define Button_1_Monitor_Tag         1
#define Button_2_Monitor_Tag         2
#define Periodic_Transmitter_Tag     3
#define Uart_Receiver_Tag            4
#define Load_1_Simulation_Tag        5
#define Load_2_Simulation_Tag        6
```

defining tag number for each task

```c
//////////////////////////////////////////////////////////
// Array elements whcih will trace the time of tasks excution
// present in TaskTraceArr[MAX_TRACE_NUMB]
// Task1
#define Button_1_Monitor_TimeStampIn              0
#define Button_1_Monitor_TimeStampOut             1
#define Button_1_Monitor_EcutionTime              2
#define Button_1_Monitor_TotalEcutionTime         3
// Task2
#define Button_2_Monitor_TimeStampIn              4
#define Button_2_Monitor_TimeStampOut             5
#define Button_2_Monitor_EcutionTime              6
#define Button_2_Monitor_TotalEcutionTime         7
// Task3
#define Periodic_Transmitter_TimeStampIn          8
#define Periodic_Transmitter_TimeStampOut         9
#define Periodic_Transmitter_EcutionTime          10
#define Periodic_Transmitter_TotalEcutionTime     11
//Task4
#define Uart_Receiver_TimeStampIn                 12
#define Uart_Receiver_TimeStampOut                13
#define Uart_Receiver_EcutionTime                 14
#define Uart_Receiver_TotalEcutionTime            15
```

Defining elements of the trace array

```
///////////////////////////////////////////////////////
#define configTaskExcutionTimePeriodic  1

//////////////////////////////////////////////////////////
// uncomment only (one) definition to monitor the variable  WCET_Monitor
// according to task number the code to get the worst case excustion time will added
// to enable this variable you must define configTaskExcutionTimePeriodic = 1

//#define Task1_WCET
//#define Task2_WCET
#define Task3_WCET
//#define Task4_WCET
//#define Task5_WCET
//#define Task6_WCET
//#define Task7_WCET
//#define Task8_WCET
```

To calculate task specific WCET to be monitored by value of WCET_Monitor .

//

For accurate results we will set TIMER1 devision to 100 instead of 1000

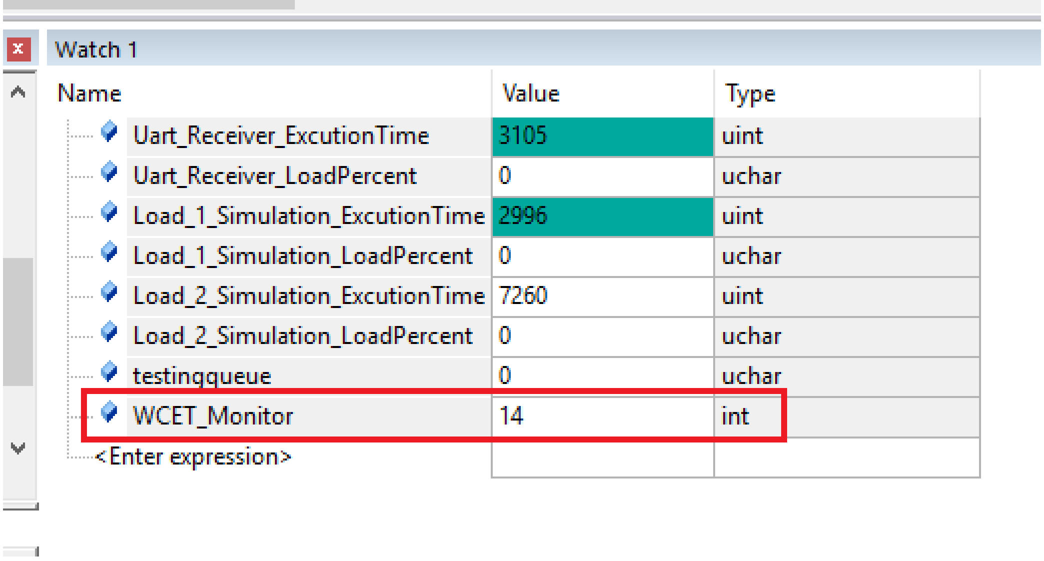
```
/* Function to initialize and start timer 1 */
static void configTimer1(void)
{
    T1PR = 100;          ///////////////////////
    T1TCR |= 0x1;        // enable
}
```

Then inside "TraceDef.h"

```
///////////////////////////////////////////////////////
#define configTaskExcutionTimePeriodic  1

//////////////////////////////////////////////////////////
// uncomment only (one) definition to monitor the variable  WCET_Monitor
// according to task number the code to get the worst case excustion time will added
// to enable this variable you must define configTaskExcutionTimePeriodic = 1

#define Task1_WCET
//#define Task2_WCET
//#define Task3_WCET
//#define Task4_WCET
//#define Task5_WCET
//#define Task6_WCET
//#define Task7_WCET
```

Set configTaskExcutionTimePeriodic to 1 and one by one define Taskn_WCET

Then in Watch Variable window add the WCET_Monitor variable to read the required task WCET



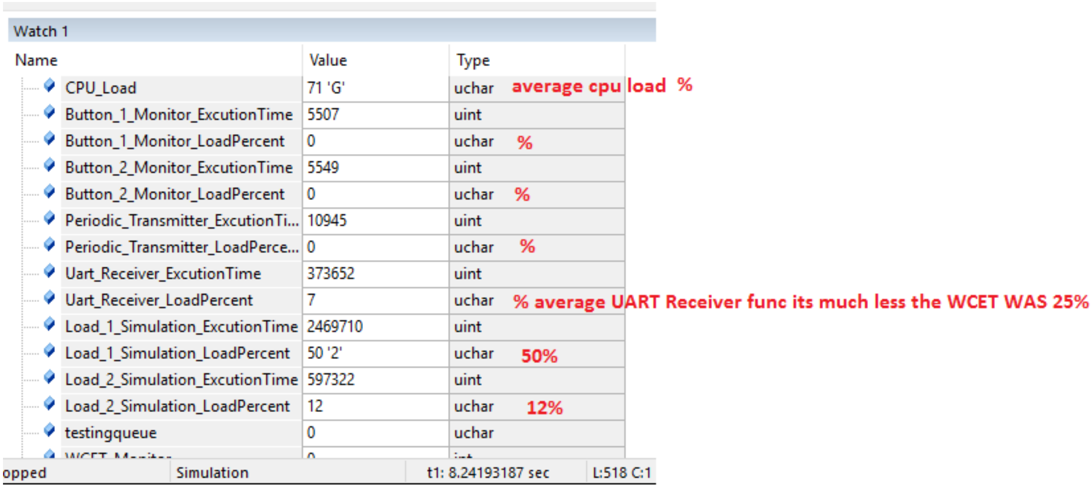
In this case we displayed WCET OF Button1 task which is 14 and to convert it to ms we will divide it by 600 (14/600) = 0.0233333 >> 0.023 ms or 23 us

Results

Tasks	WCET Read	Result
Button_1_Monitor	14	0.0233
Button_2_Monitor	14	0.0233
Periodic_Transmitter	96	0.16
Uart_Receiver	2953	4.92
Load_1_Simulation	3006	5.01
Load_2_Simulation	7246	12.075

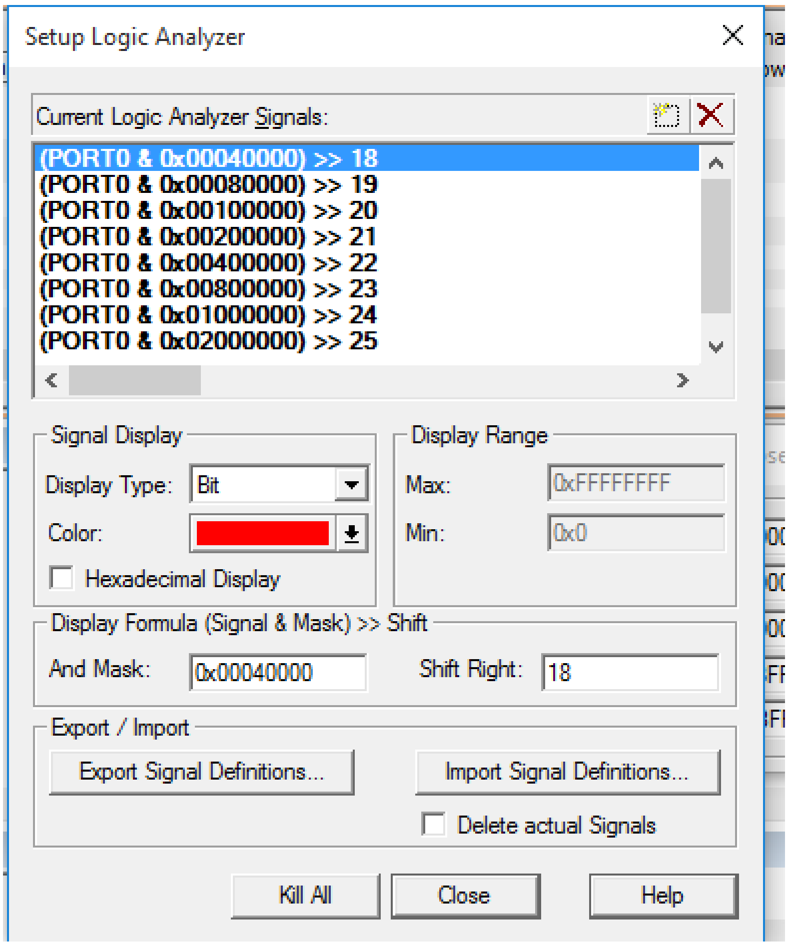
Average results percentage

```


/////////////////////////////////////////////////////
//#define configTaskExcutionTimePeriodic   1

/////////////////////////////////////////////////////
```
Undefine configTaskExcutionTimePeriodic then run simulation



These are average percentage of tasks utilization time from trace macros

2- Logic analyzer interpretation

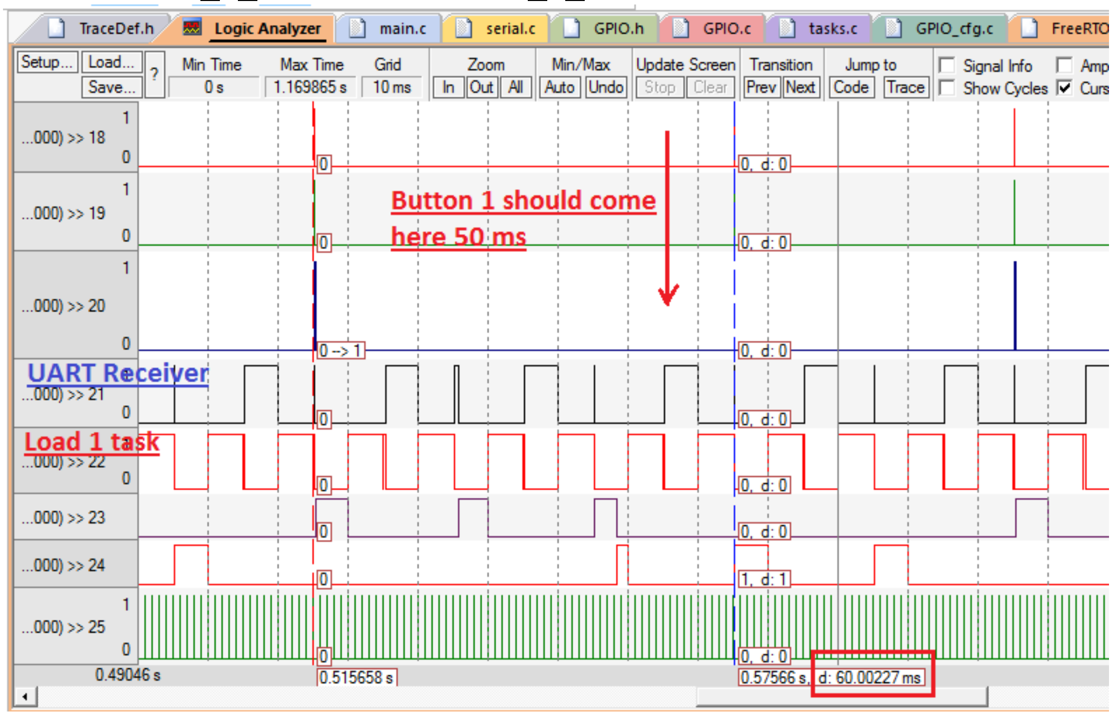


Button_1_Monitor	pin >> 18
Button_2_Monitor	pin >> 19
Periodic_Transmitter	pin >> 20
Uart_Receiver	pin >> 21
Load_1_Simulation	pin >> 22
Load_2_Simulation	pin >> 23
Idle	pin >> 24
Tick	pin >> 25

*Tick

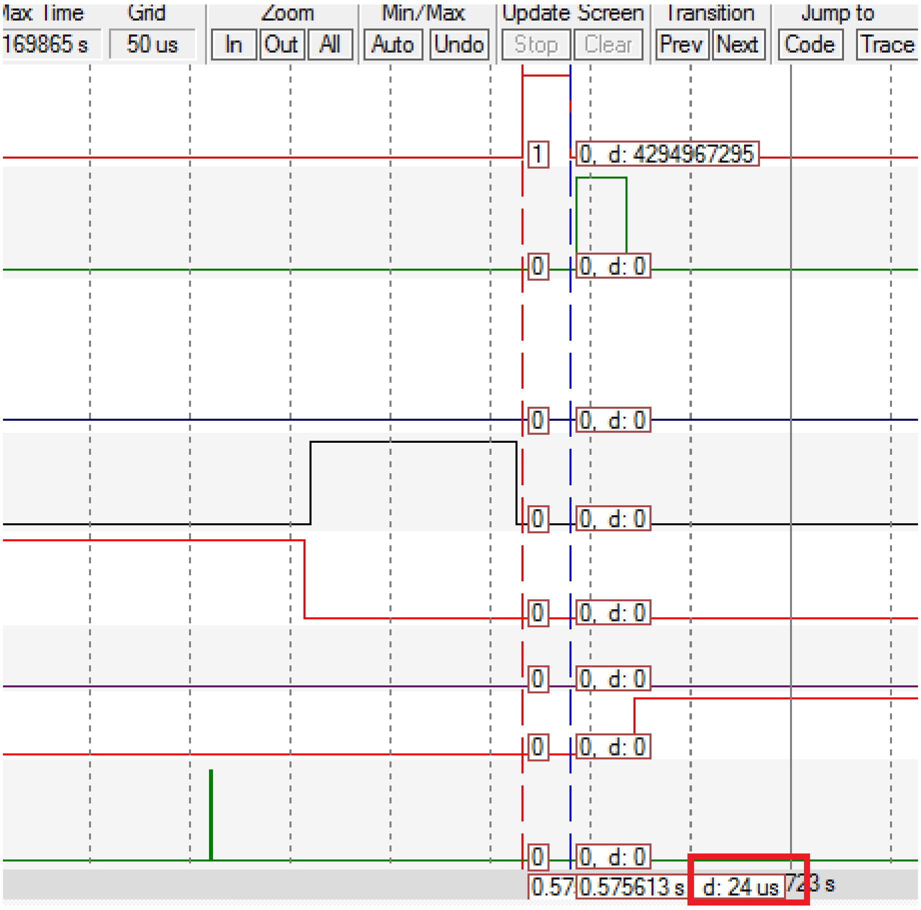
By logic analyzer the ticks will be fired every 1 ms

Button_1_Monitor & Button_2_Monitor



Button_1_Monitor & Button_2_Monitor should come every 50 ms . you can see the are delayed to 60 ms why??

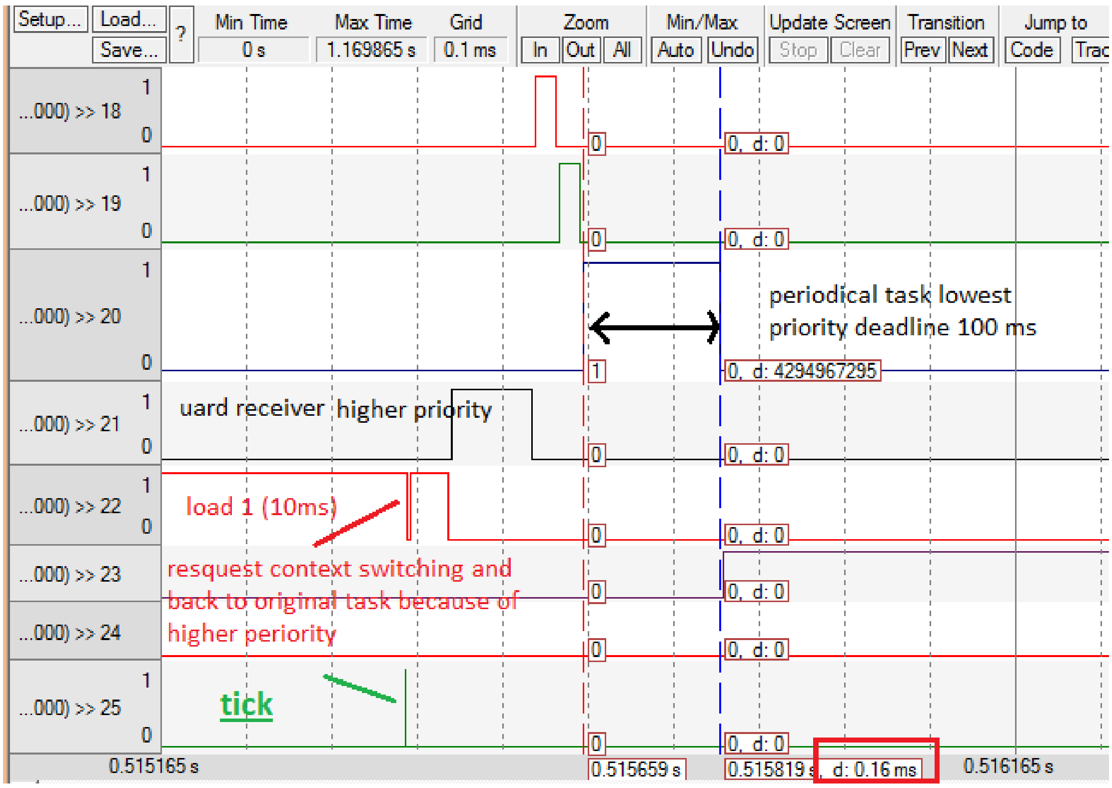
Because load1 simulation task (10 ms deadline) blue arrow and UART receiver task (20 ms deadline) Green arrow executed before it because their priorities are higher delaying it by 10 ms



The execution time of Button 1 and 2 functions are 0.025 ms or 25 us

///

Periodic transmitter function

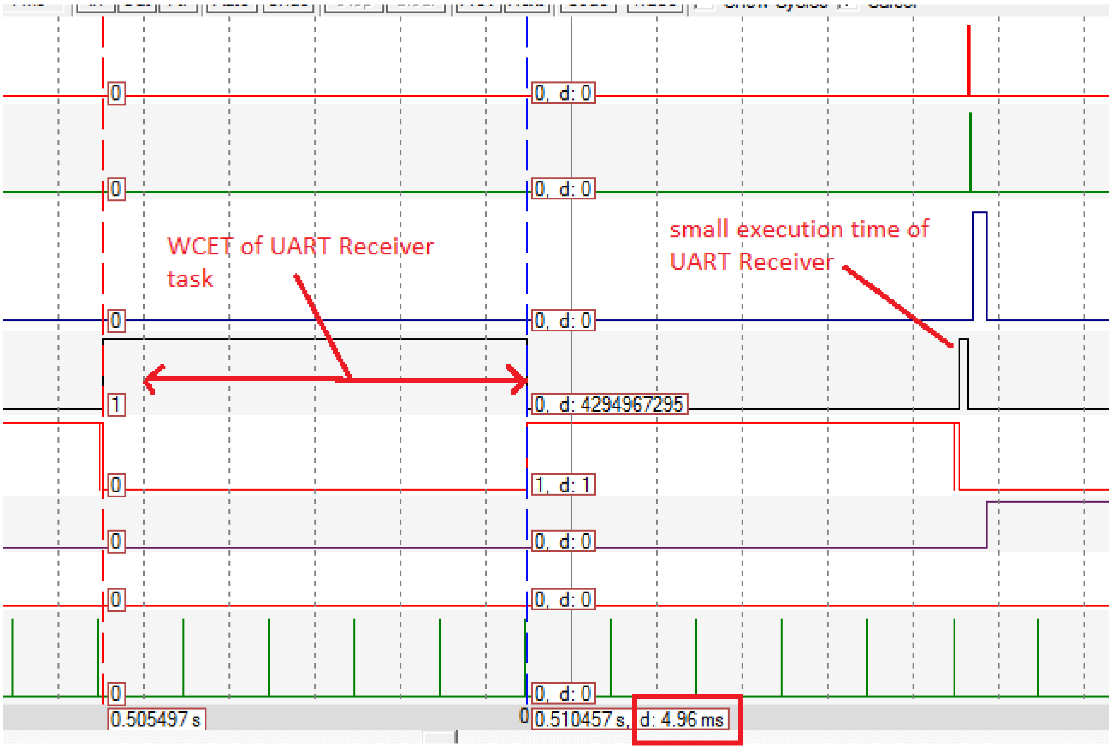


After the tick and insertion of the new tasks which were in delayed list the system added them to EDF Ready list and request for context switching and because the load1 task were running and didn't finished its execution and its deadline is 10 ms, context switching will get back to load1 task making this spike . then after this the UART receiver task will be executed because it is higher priority (20 ms deadline) and after its execution the other tasks (button 1 and 2) then at the last periodically transmitter task execute.

Its execution time is 0.16 ms

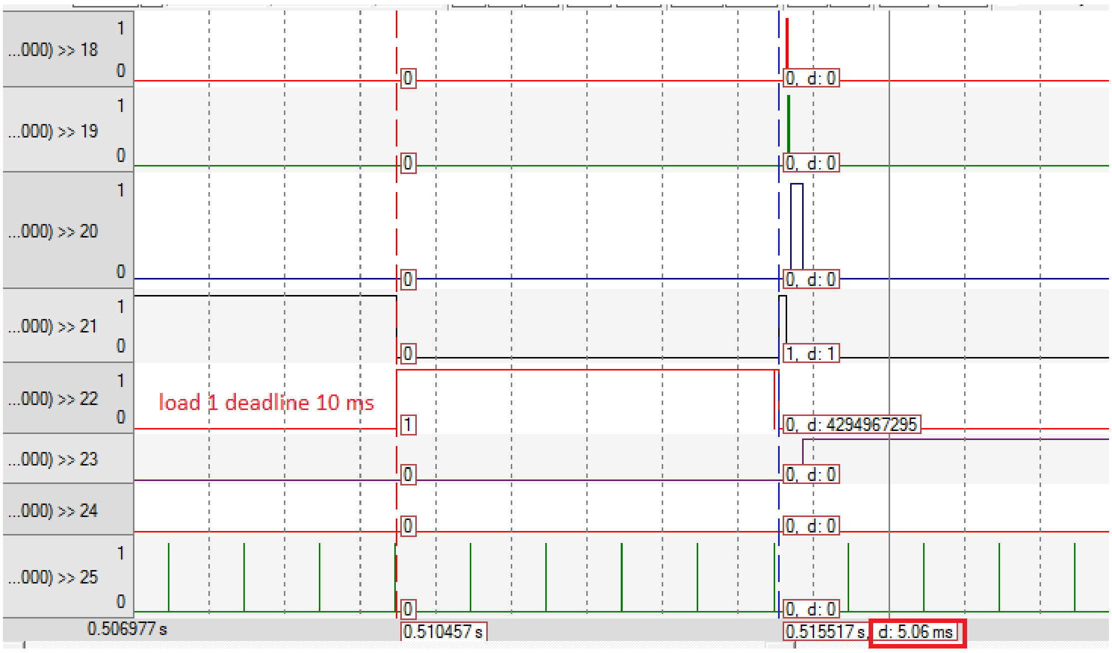
///

Uart_Receiver



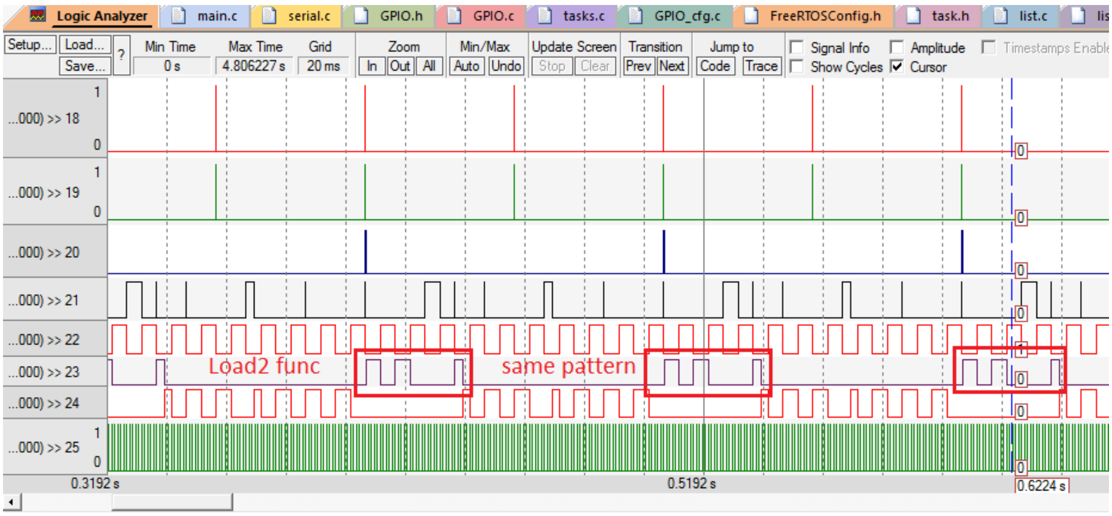
The worst case execution time of Uart_Receiver is 4.96 ms this is because at some point the queue contains many messages to be sent and it will wait UART module to send those strings.

Load_1_simulation



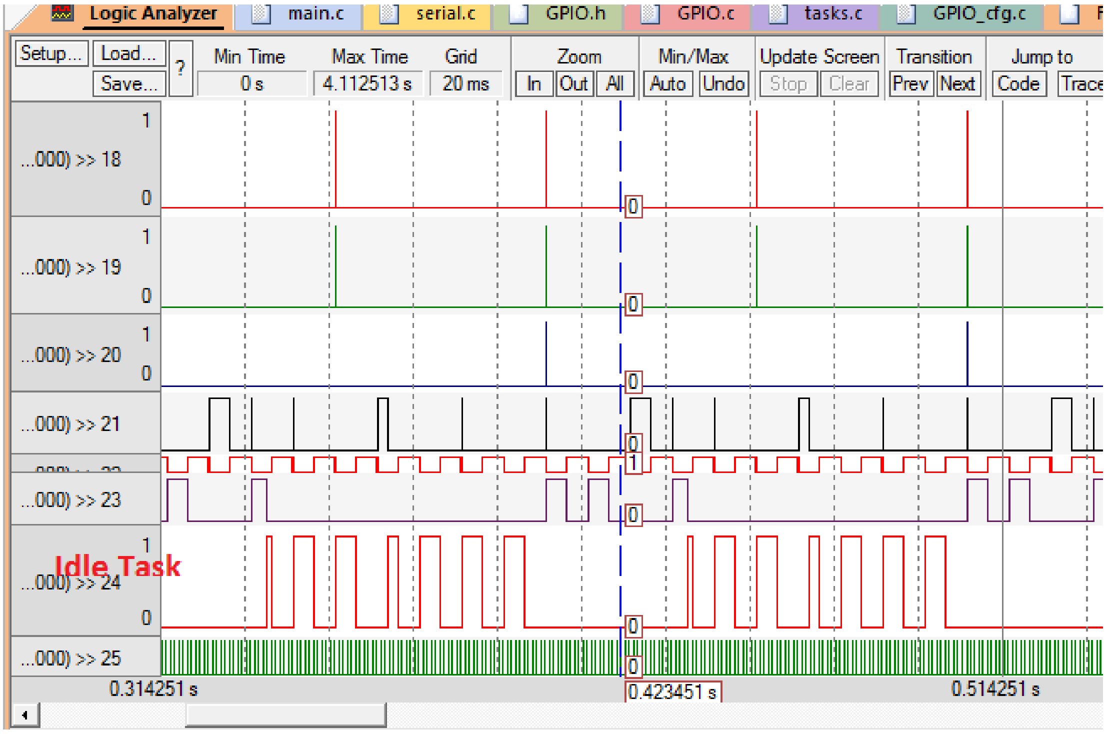
execution time of load 1 is 5 ms

Load_2_simulation



It is preempted by all other tasks has the same pattern every 100 ms = hyper period it is about 12 ms

IDLE task



It is preempted by any other tasks

3- Using FreeRTOS Statistics

Implementation

In FreeRTOS.h

```
#define configMAX_TASK_NAME_LEN    ( 8 )
#define configUSE_TRACE_FACILITY   1    ////
#define configUSE_16_BIT_TICKS     0
#define configIDLE_SHOULD_YIELD    0
```

Set configUSE_TRACE_FACILITY to 1

```
///////////////////////////////////////////////////////////////////////
// Definition
#define configGENERATE_RUN_TIME_STATS          1   ////////////////////////
#define configUSE_STATS_FORMATTING_FUNCTIONS   1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()
#define portGET_RUN_TIME_COUNTER_VALUE() T1TC      // this macro will return
```

Add those macros

In Tasks.c

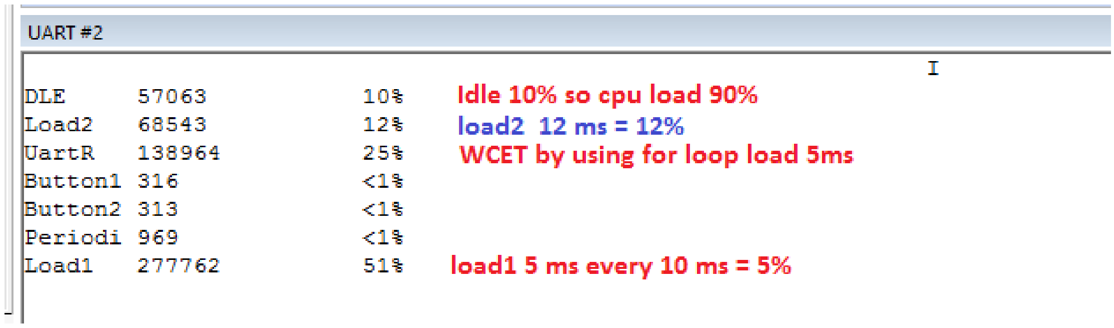
```
#if ( configUSE_TRACE_FACILITY == 1 )

  UBaseType_t uxTaskGetSystemState( TaskStatus_t * const pxTaskStatusArray, const UBa
  {

  UBaseType_t uxTask = 0, uxQueue = configMAX_PRIORITIES;

    vTaskSuspendAll();
    {
      /* Is there a space in the array for each task in the system? */
      if( uxArraySize >= uxCurrentNumberOfTasks )
      {
        /* Fill in an TaskStatus_t structure with information on each
```

Inside uxTaskGetSystemState function

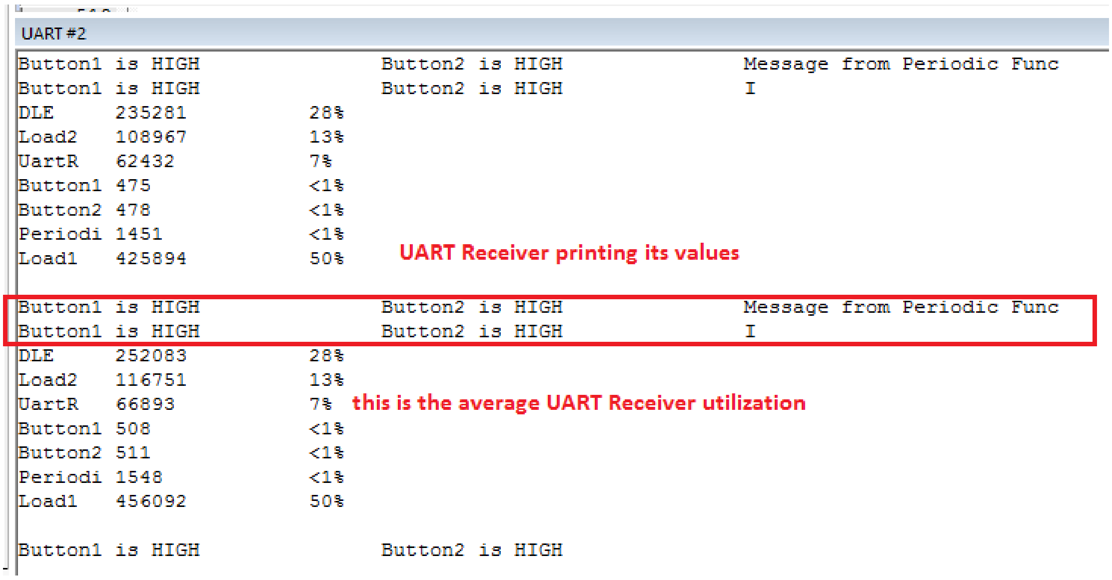
```
/////////////////////////////////////////////////////////////////////////////////////////
#if (configUSE_EDF_SCHEDULER == 1)
{

    uxTask += prvListTasksWithinSingleList( &( pxTaskStatusArray[ uxTask ] ),( List_t * ) &( xReadyTasksListEDF ), eReady );
}
#else
```

We will change to this by adding
(List_t *) &(xReadyTasksListEDF) instead of
&(pxReadyTasksLists[uxQueue])

Results



You can see that the UART load in WCET is 25%



When we just make the UART Receiver func do its job the average utilization will be 7% those results are consistent with our implementation

In conclusion the EDF implementation meets the requirements of no task will break its deadline and the system overall is schedulable with low margin as in WCET the cpu load will be 90% and we have only 10% margine to play around. All results from our implementation are consistent with FreeRTOS statistics function even more reliable than it.