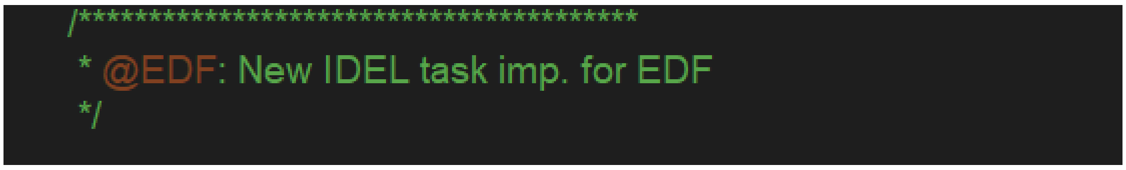
Topic: FreeRTOS EDF scheduler verification report
Date: 12-10-2022
Author: Mohamed Maher

1. ABSTRACT

Using project tasks and these constrains, we verified that our EDF scheduler implementation for FreeRTOS is working as expected based on comparing Keil simulation results for the project with SimSO simulator as shown in Fig1,2.

Hint: *To easily find our modifications in Tasks.c file, search for comment tag "@EDF"*



```
/*********************************************
 * @EDF: New IDEL task imp. for EDF
 */
```

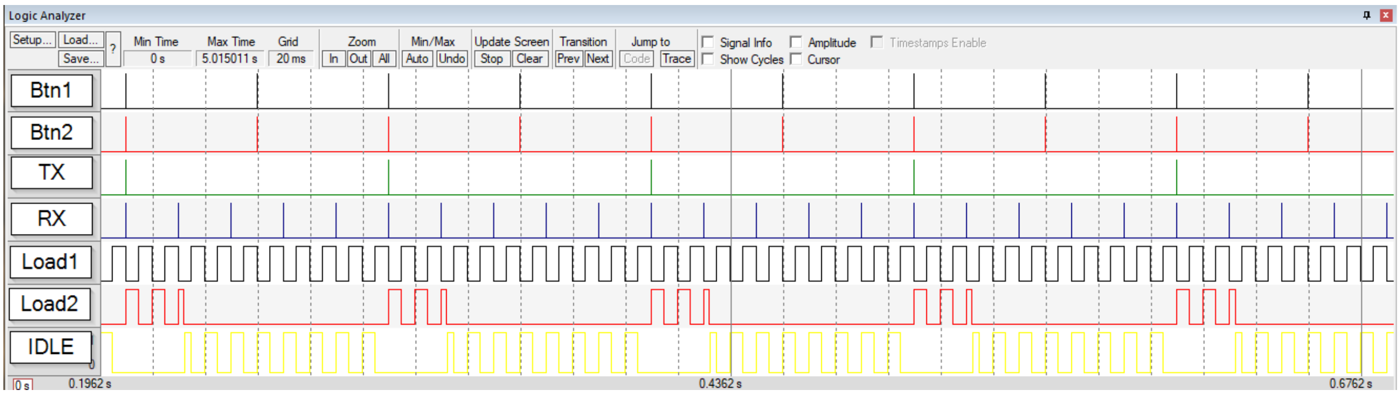


Figure 2: All tasks plot using trace probes

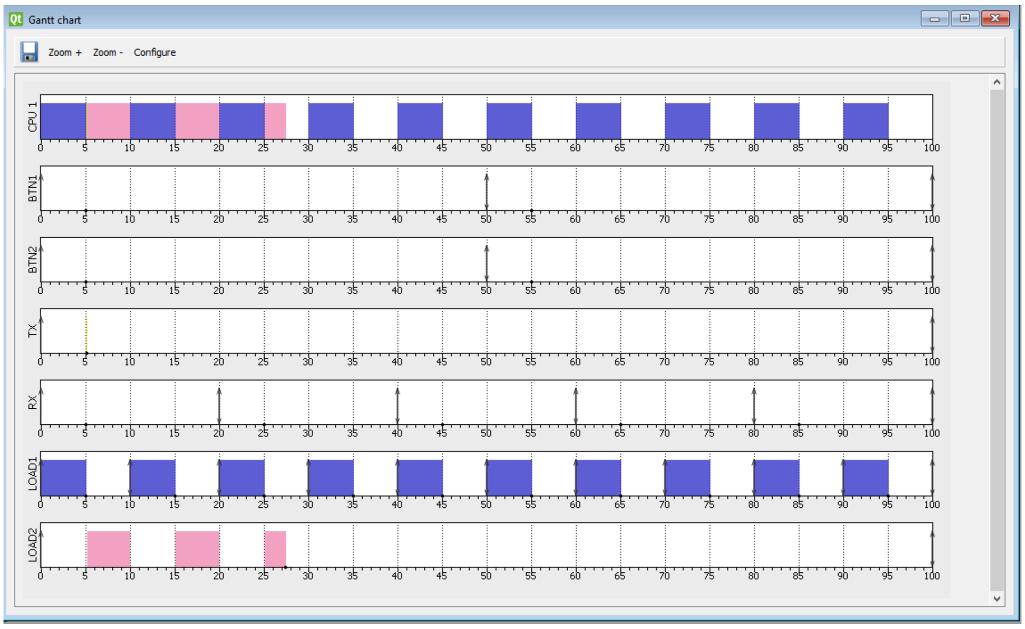
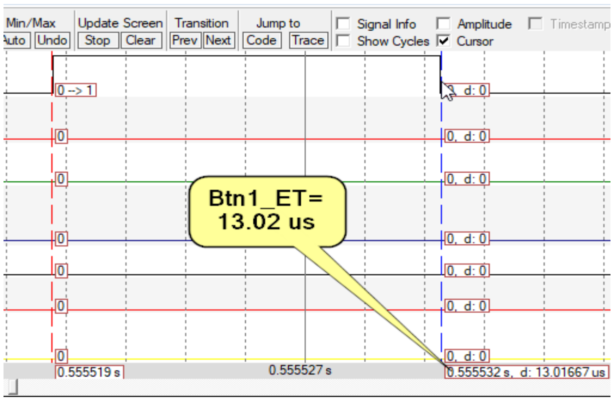
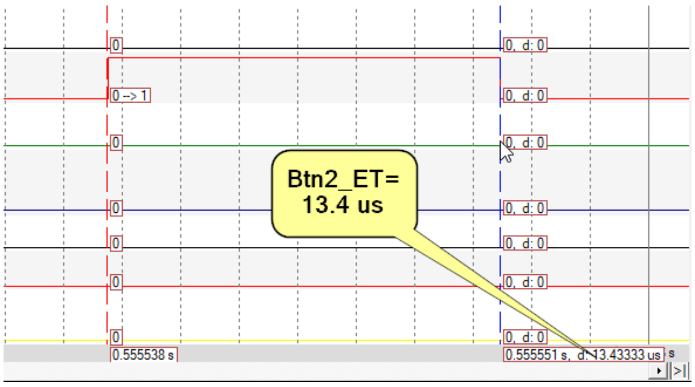


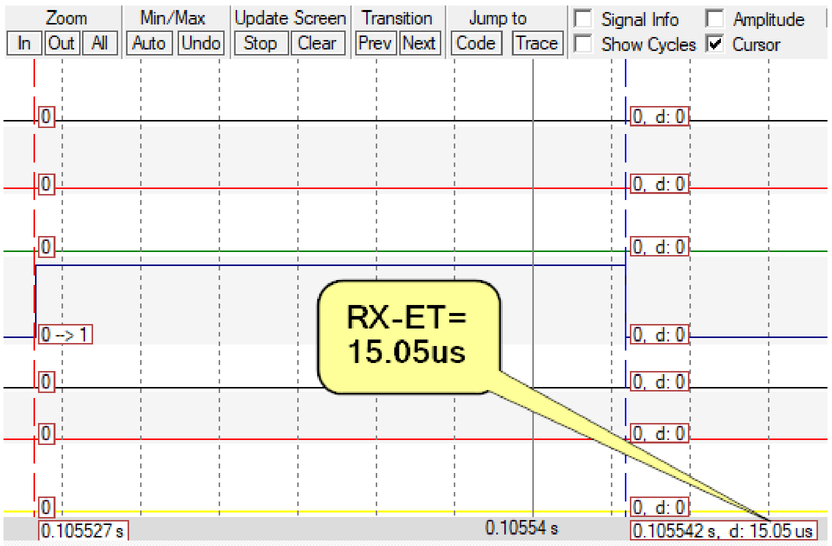
Figure 1Simso silmultor results

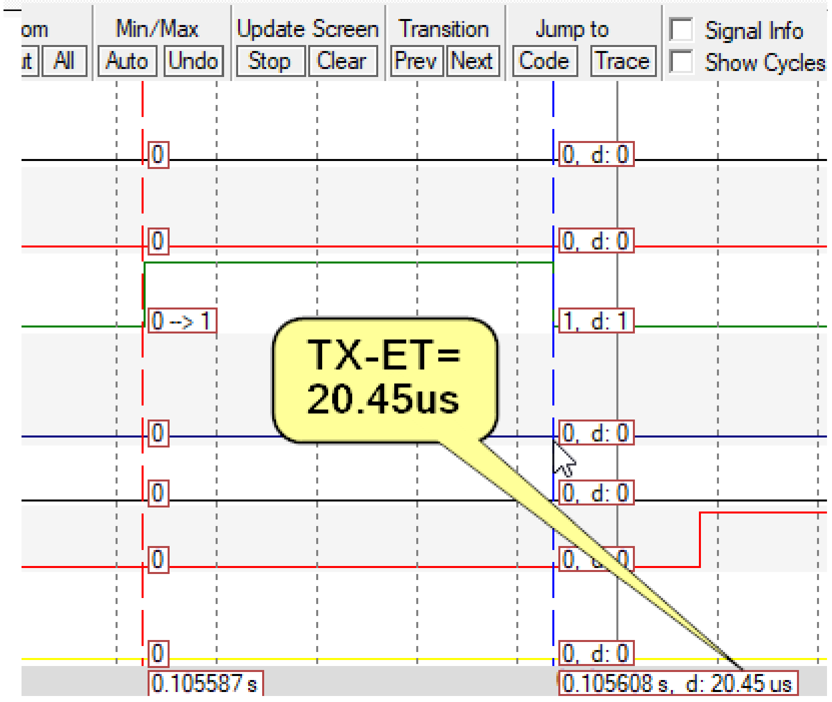
2. Measuring Tasks' Execution Time:

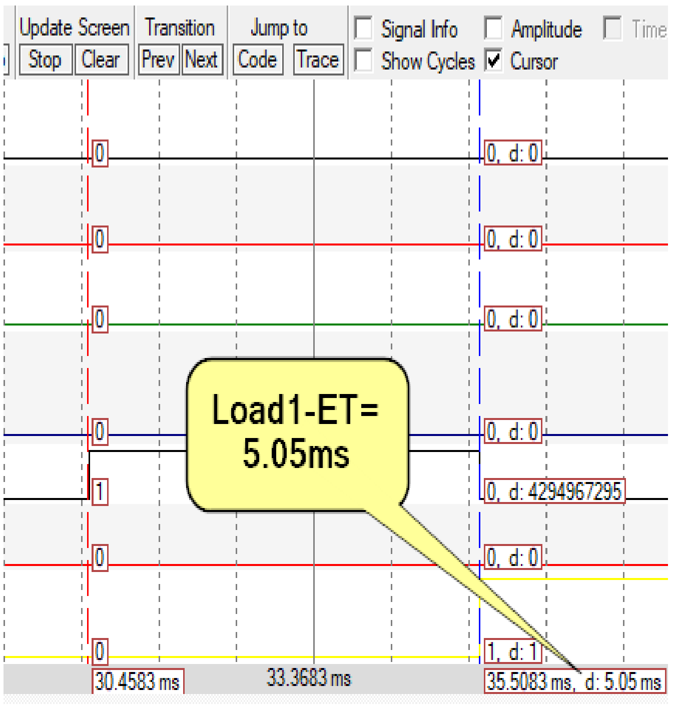
Using *traceTASK_SWITCHED_OUT()* and *traceTASK_SWITCHED_IN()* with GPIO we managed to measure every task execution time in run time as shown in following figures .

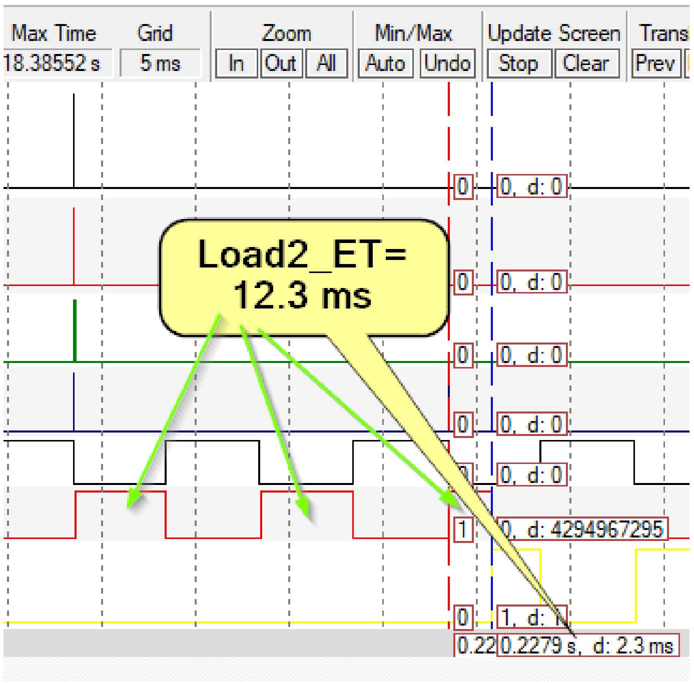












3. System Hyperperiod :

For a periodic system , system hyperperiod is the Least Common Multiple of all tasks periodicity.

So, System Hyperperiod = LCM {50,50,100,20,10,100}

= 100 mS

4. CPU Load

a. Using Analytical Method:

Cpu_load = sum (task execution time * n) / T

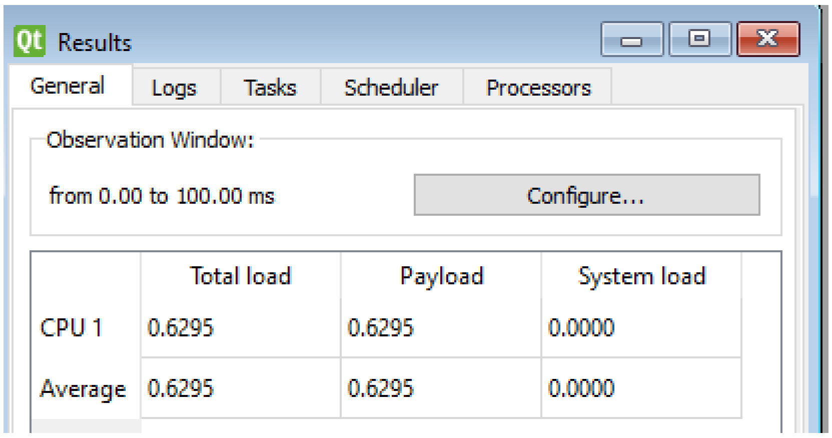
Where n: number of a task running during system time

T: System runtime period

Cpu_load = [(0.01302 * 2) + (0.0134 * 2) + (0.0205 * 1) +(0.01505 * 5) + (5.05 *10) + (12.3 * 1)] / 100 = 0.69

So our system load is around 63%

b. Using SimSo Simulator



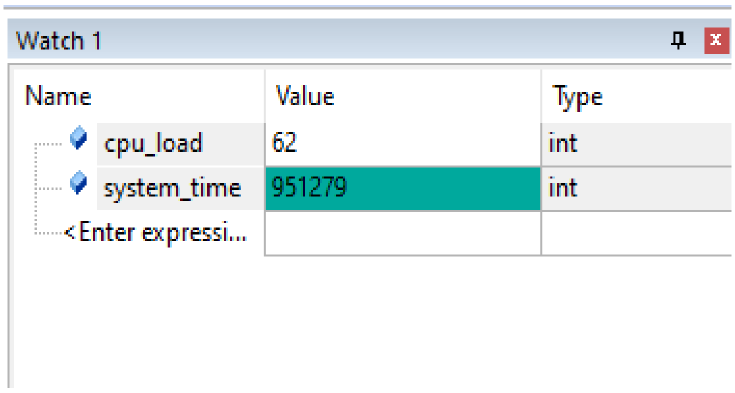
Observation Window:

from 0.00 to 100.00 ms

	Total load	Payload	System load
CPU 1	0.6295	0.6295	0.0000
Average	0.6295	0.6295	0.0000

c. Using timer 1 and trace macros

By using the same formula for calculating cpu load in our code with help of tasks tracing probes, we can see the cpu load in *keil simulator* as shown in next figure.



Name	Value	Type
cpu_load	62	int
system_time	951279	int
<Enter expressi...		

5. System schedulability

a. Using Rate Monotonic:

A system is schedulable for “RM” if it satisfies the following equation.

$$\sum_{k=1}^{n} \frac{Ci}{Ti} \leq U = n(2^{1/n} - 1)$$

Where n is the number of processes in the process set, Ci is the computation time of the process, Ti is the Time period for the process to run and U is the processor utilization.

**Ref: geeksforgeeks*

So, system U must be equal or less that URM

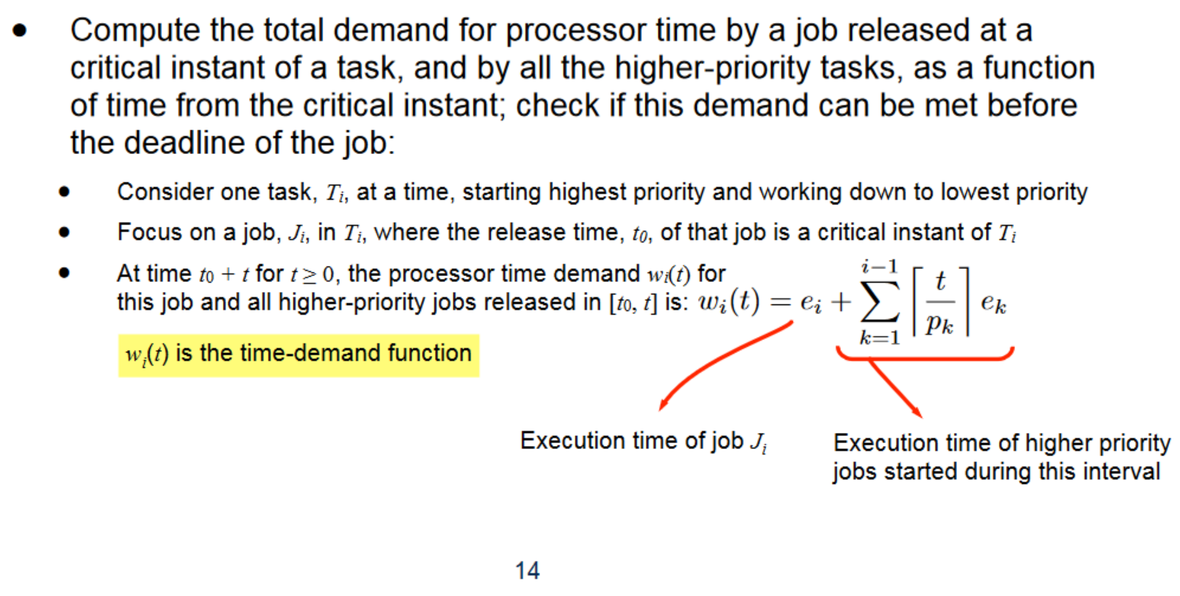
Let’s verify our system using this equation:

URM = 6 (2 ^ (1/6)   - 1) = 0.735

Our system total utilization = 0.62

So, our system will be schedulable for RM scheduler.

b. Using Time Demand:



- Compute the total demand for processor time by a job released at a critical instant of a task, and by all the higher-priority tasks, as a function of time from the critical instant; check if this demand can be met before the deadline of the job:
  - Consider one task, T_i, at a time, starting highest priority and working down to lowest priority
  - Focus on a job, J_i, in T_i, where the release time, t_0, of that job is a critical instant of T_i
  - At time $t_0 + t$ for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[t_0, t]$ is: $w_i(t) = e_i + \underbrace{\sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k}$

$w_i(t)$ is the time-demand function

Execution time of job J_i

Execution time of higher priority jobs started during this interval

14

Source: https://csperkins.org/teaching/2014-2015/adv-os/lecture02.pdf

Appling this formula to our system, we will start with earliest deadline tasks as it the highest priority.

- **Load1_Task [5.05, 10]**

  W(10) = 5.05 + 0 = 5.05 ms < 10 ms → **Task Passed**

- **UartRX_Task [0.01505, 20]**

  W(20) = 0.01505 + (20/10) 5.05 = 10.12ms < 20 ms* → **Task Passed**

- **Button_1_Task [0.01302, 50]**

  *W(50)= 0.01302 + [(50/20)*0.01505] + [(50/10)*5.05] = 25.05ms*

  W(50) < 50 ms → **Task Passed**

- **Button_2_Task [0.0134, 50]**

  W(50) = 0.0134 + [(50/50) 0.01302] + [(50/20)*0.01505] + [(50/10)*5.05]*

  W(50)= 25.31 < 50 ms → **Task Passed**

- **UartTX_Task [0.0205 , 100]**

  W(100) = 0.0205 + [(100/50)0.0134] + [(100/50) 0.01302] + [(100/20)*0.01505] + [(100/10)*5.05] = 50.5 < 100ms* → **Task Passed**

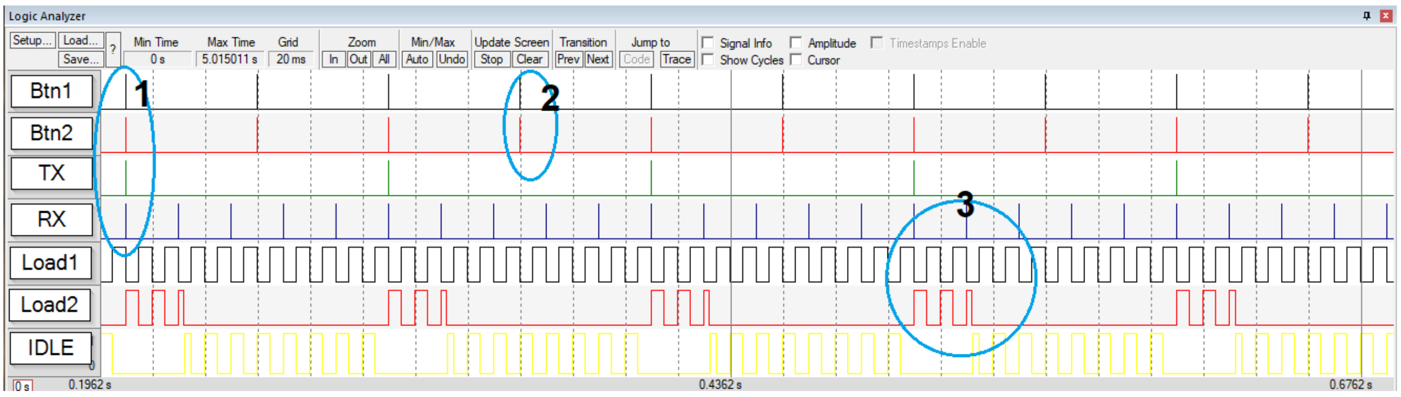
- **Load2_Task [12.3 , 100]**

  W(100)= 12.3 + [(100/100)*0.0205] + [(100/50)0.0134] + [(100/50)* 0.01302] + [(100/20)*0.01505] + [(100/10)*5.05]= 62.31 < 100ms* → **Task Passed**

  → **So, for a Time demand analysis our system is schedulable**.

6. Using Keil Simulator

Using Keil logic analyzer and task trace probes, we have 3 cases shown in the following figure which confirm that EDF is working as expected.



- Case #1:
  3 Tasks come at the same tick, scheduler runs the earliest task first. Btn1 then btn2 and finally uartTX Task.
- Case #2:
  2 Tasks with the same periodicity come at the same tick.
  Scheduler runs btn1 then btn2.
- Case #3:
  Load1_Task comes while Load2_Task in running , scheduler switch out load2 to run load1. Because it's earlier than load2.

That's all, Thanks for your time.