MY_OWN_RTOS

Implemented by : Anwar Ahmad Anwar Adly.

Supervised by Eng. : Keroles Khalil Shenouda.

Github

Linkedin

Contents

Overview.

This project is around implementation RTOS from A to Z, It helped me to master C, Embedded C, ARM (Inline Assembly, Reset Sequence, SVC & PendSV, Stack Memory (How to design it into sections for main stack and each task then switching between Main Stack and Process Stack) and Switching context between Tasks) and also mastering RTOS Concepts (Time Management, Performance (CPU Utilization) Management, Task States, Scheduling Algorithms through Preemptive with Priority Algorithm and Round Robin with Priority Algorithm, Mutex & Semaphore, Priority Inversion and Deadlock Conditions).

RTOS(Real-Time Operating System) : is a multitasking system that used to exchanging / sharing data between tasks, synchronizing tasks and schedule them, also it provides a services as task management, memory management, input / output management, filesystem management, system protection and networking, And it comes to solve problem of superloop (while(1)) that comes in when starting adding so many tasks that start some of them missing deadlines or preventing other features from working.

MY_OWN_RTOS characteristics :

- Reliability : as an application will be available without fail for long periods without human intervention.

- Predictability : as the system is mor deterministic and performs it's functionality meeting the time requirements.

- Performance : as it perform fast enough to fulfill the time requirements, so we use wait for event instruction "wfe" in Idle_Task that used when there are no tasks in the system to higher the performance.

- Scalability : as it's files doesn't depend on only one microcontroller architecture, only the CortexMX_Porting file depend on Cortex_M family and only to use in another system just change this file.
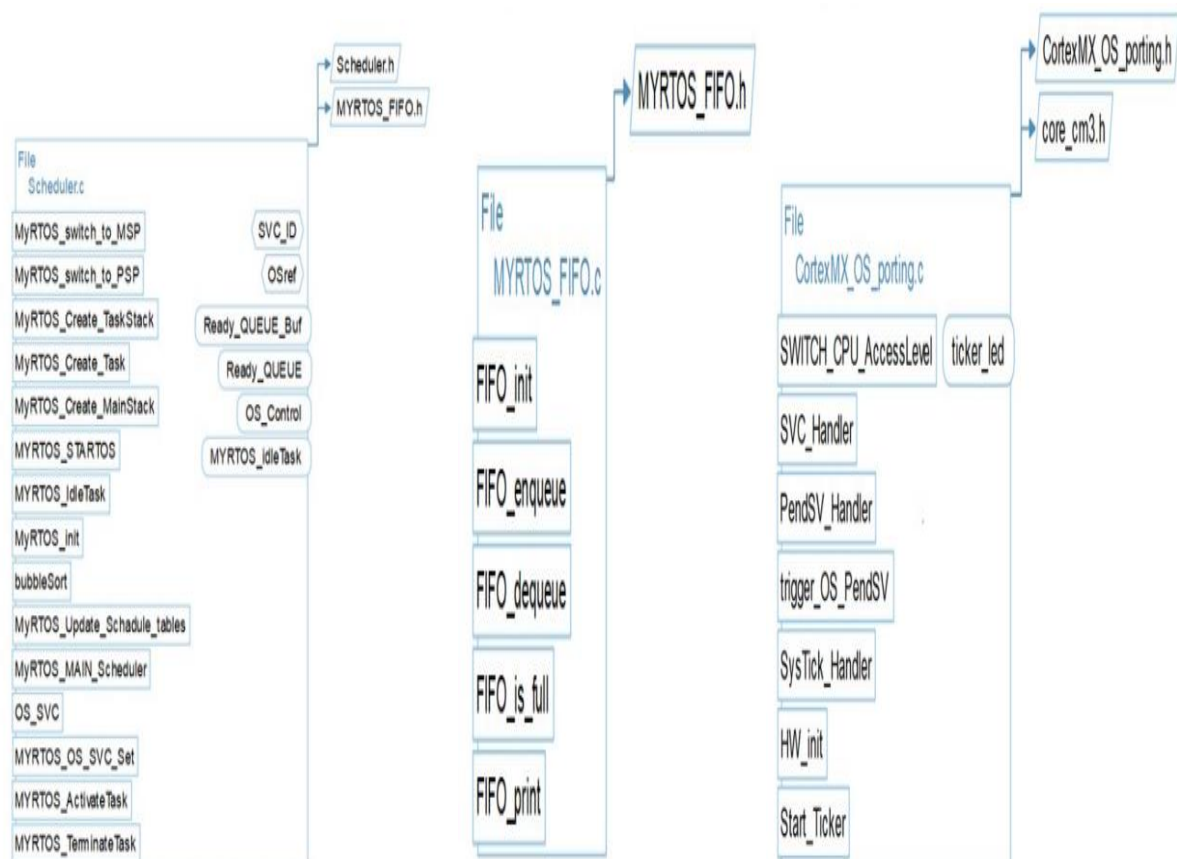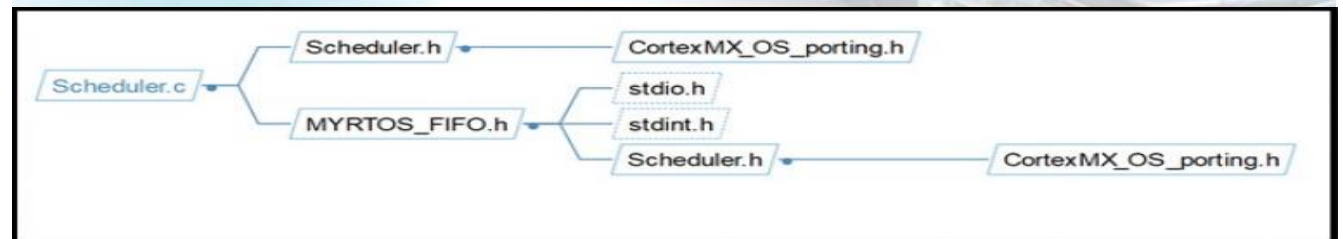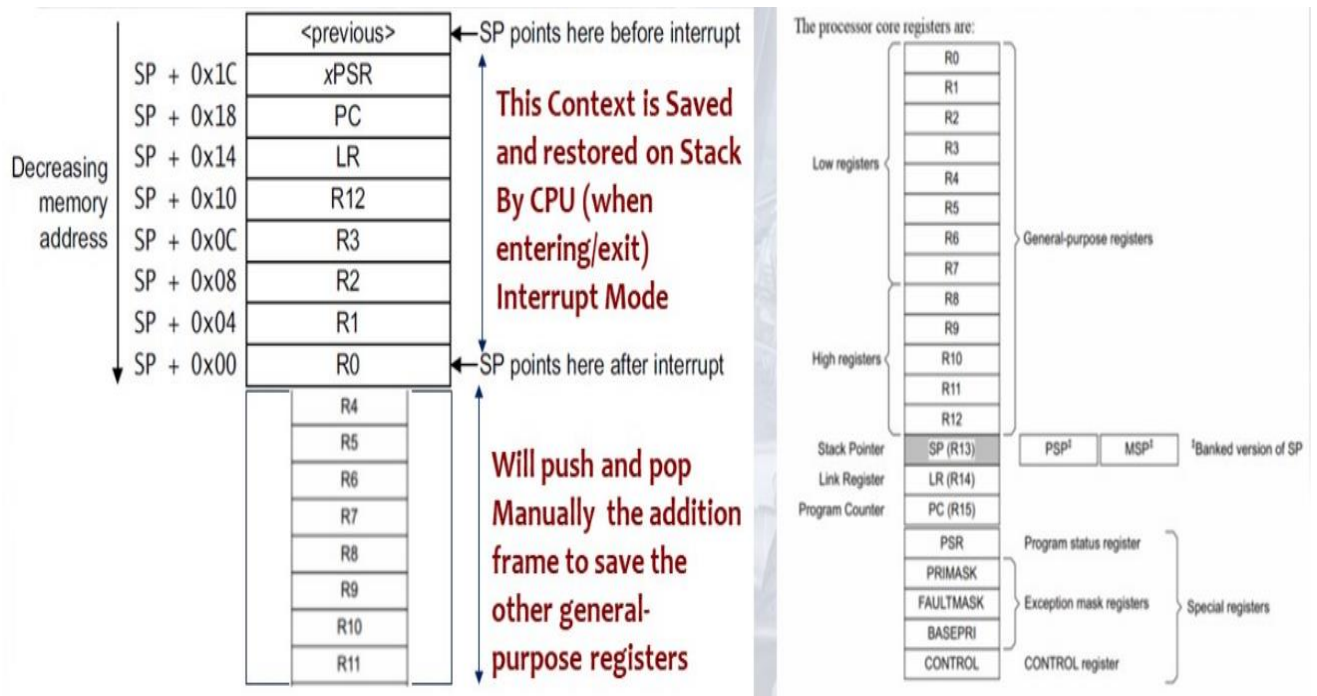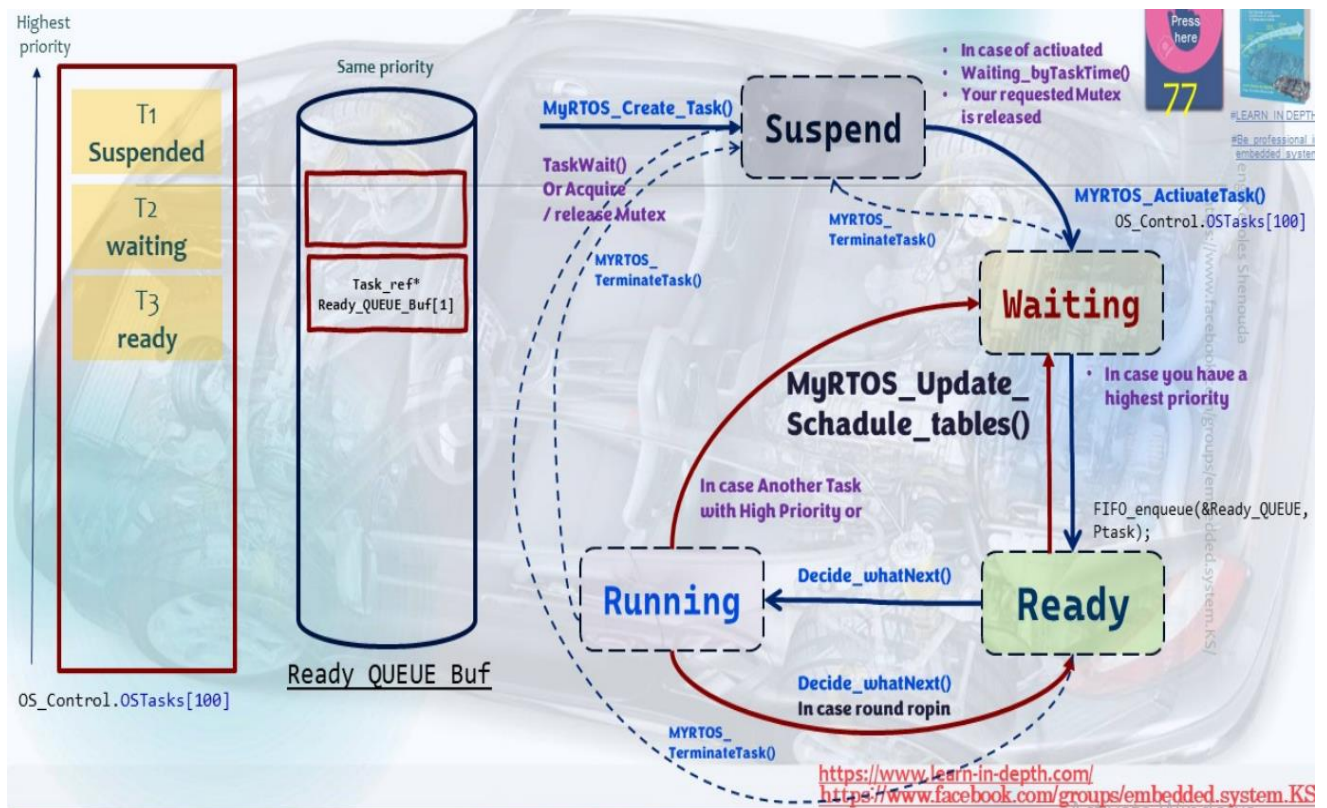
MyRTOS_Files.

User Tasks

Scheduler.c

MYRTOSFIFO.c

CortexMX_OS_porting.c

Scheduler.c ──┬── Scheduler.h ── CortexMX_OS_porting.h
              └── MYRTOS_FIFO.h ──┬── stdio.h
                                  ├── stdint.h
                                  └── Scheduler.h ── CortexMX_OS_porting.h

---

File Scheduler.c → Scheduler.h, MYRTOS_FIFO.h

MyRTOS_switch_to_MSP	SVC_ID
MyRTOS_switch_to_PSP	OSref
MyRTOS_Create_TaskStack	Ready_QUEUE_Buf
MyRTOS_Create_Task	Ready_QUEUE
MyRTOS_Create_MainStack	OS_Control
MYRTOS_STARTOS	MYRTOS_idleTask
MYRTOS_IdleTask	
MyRTOS_init	
bubbleSort	
MyRTOS_Update_Schadule_tables	
MyRTOS_MAIN_Scheduler	
OS_SVC	
MYRTOS_OS_SVC_Set	
MYRTOS_ActivateTask	
MYRTOS_TerminateTask	

File MYRTOS_FIFO.c → MYRTOS_FIFO.h

- FIFO_init
- FIFO_enqueue
- FIFO_dequeue
- FIFO_is_full
- FIFO_print

File CortexMX_OS_porting.c → CortexMX_OS_porting.h, core_cm3.h

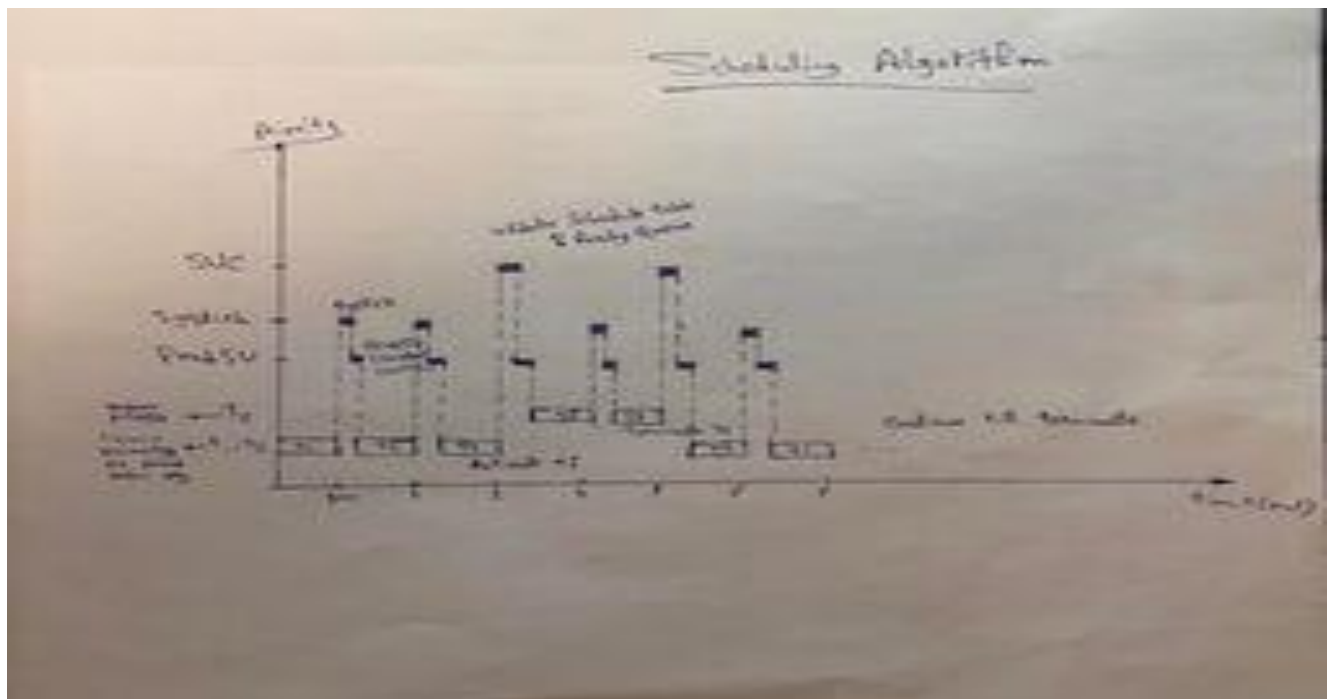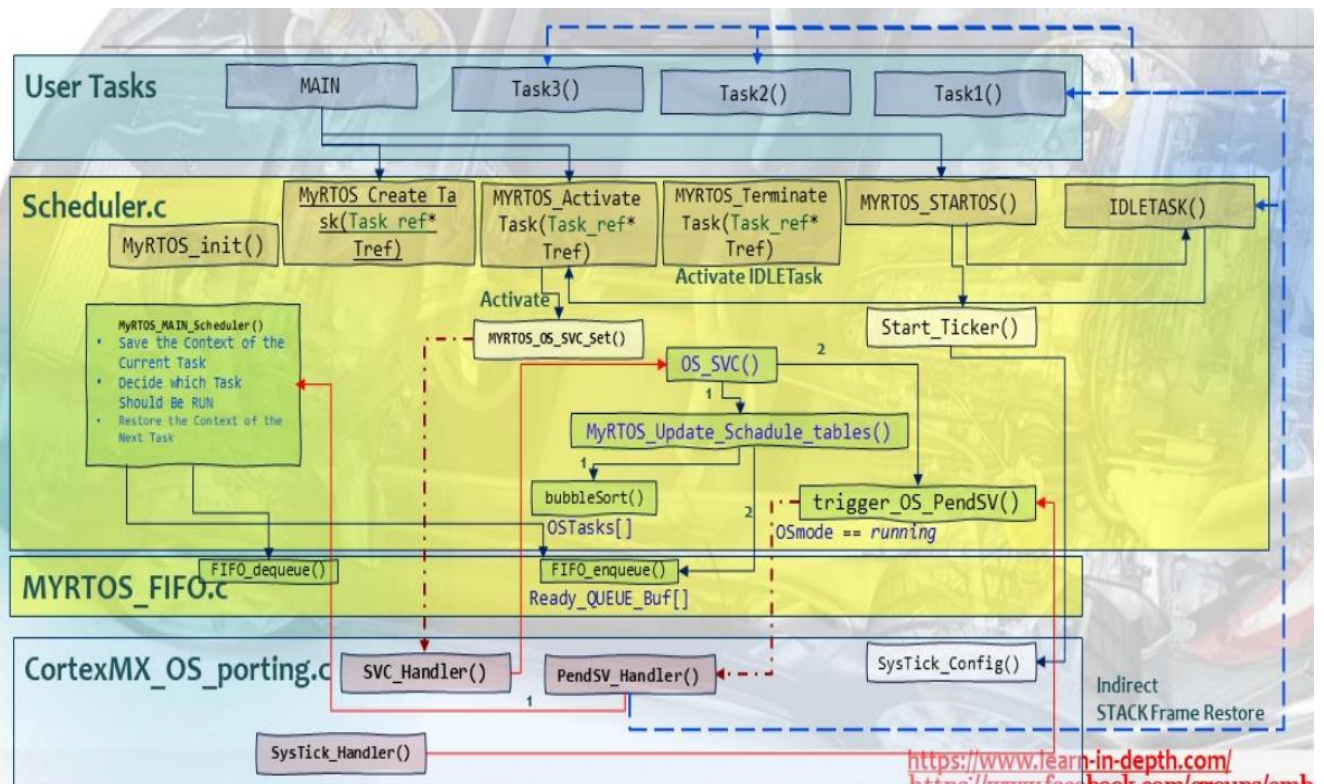|	
SWITCH_CPU_AccessLevel	ticker_led
SVC_Handler	
PendSV_Handler	
trigger_OS_PendSV	
SysTick_Handler	
HW_init	
Start_Ticker	

Task States & Context Switching.

# System Layers & Scheduling Algorithm.

Case Studies.

- Task1 activates Task4 which is the higher priority than Task2 and Task3, So Task1 start execution and activates Task4 start execution till terminates then Task2 and Task3 start execution because they are higher priority than Task1.
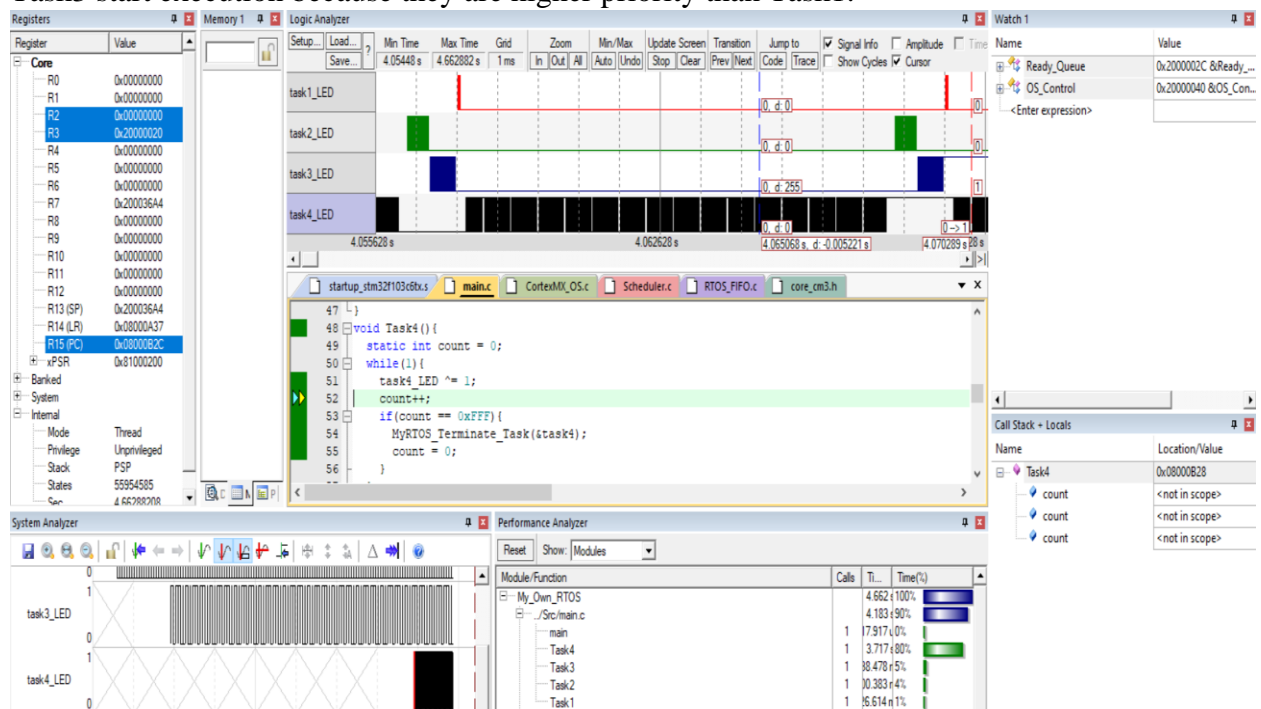
- Task1, Task2 and Task3 have the same priority so executing round-robin algorithm and each Task has a waiting time, 100ms, 300ms and 500ms respectively.
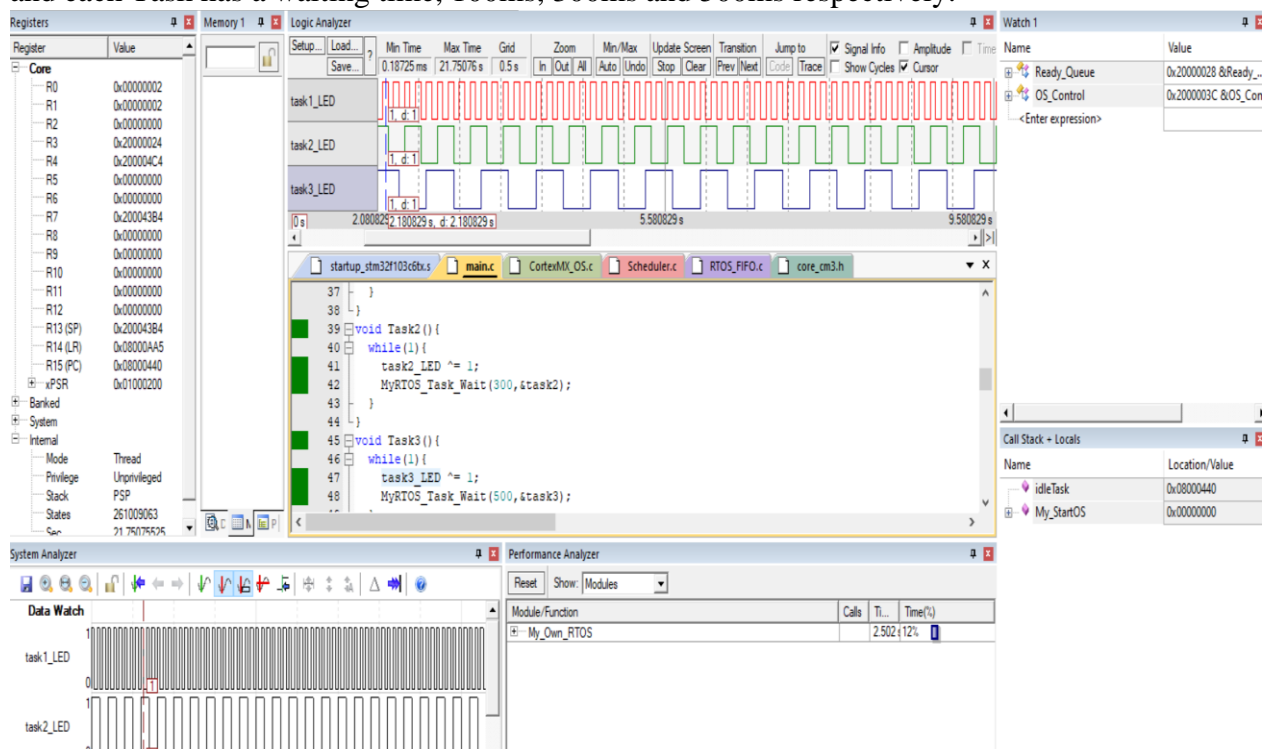
- Task1 activates Task2 activates Task3 activates Task4 with a terminate time after 200ms, and activation time after 100ms from start execution, T4>T3>T2>T1 in priority.

- Task4 higher priority than Task1 and shared the same resource, Task1 start executing till Task4 start and need to acquire mutex but it already acquired by Task1, so it enter waiting state till Task1 release mutex and Task4 start executing.