MY_OWN_RTOS

Implemented by : Anwar Ahmad Anwar Adly.

Supervised by Eng. : Keroles Khalil Shenouda.

Github

Linkedin

Contents

Overview.

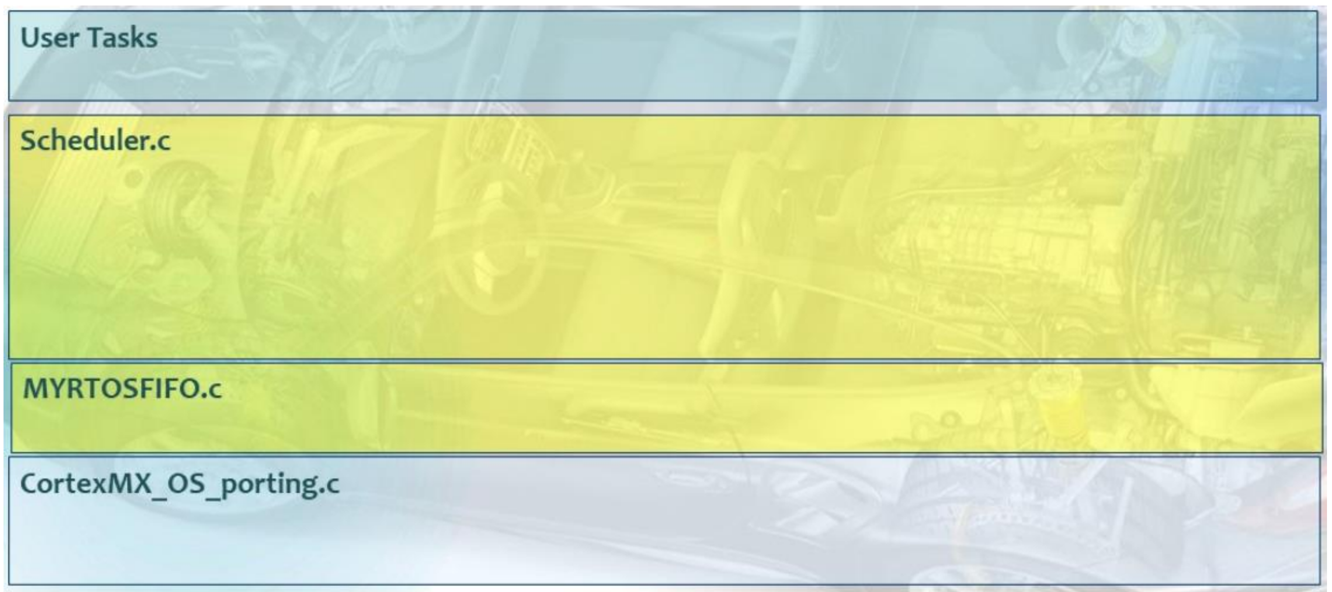
This project is around implementation RTOS from A to Z, It helped me to master C, Embedded C, ARM (Inline Assembly, Reset Sequence, SVC & PendSV, Stack Memory (How to design it into sections for main stack and each task then switching between Main Stack and Process Stack) and Switching context between 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).

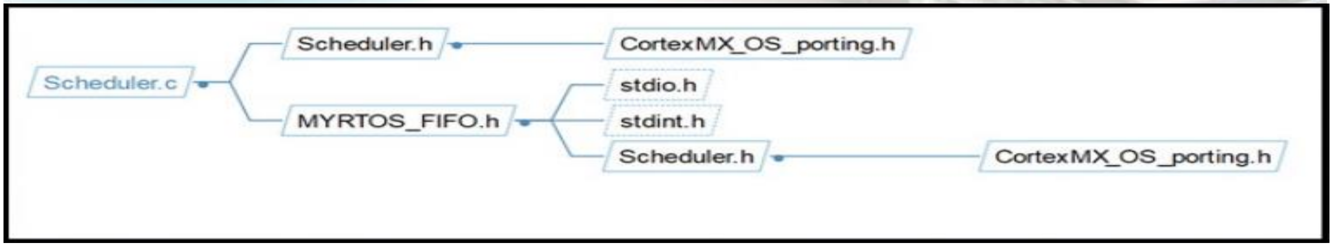
RTOS(Real-Time Operating System) : is a multitasking system that used to exchanging / sharing data between tasks, synchronizing tasks and schedule them, also it provides a services as task management, memory management, input / output management, filesystem management, system protection and networking, 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.

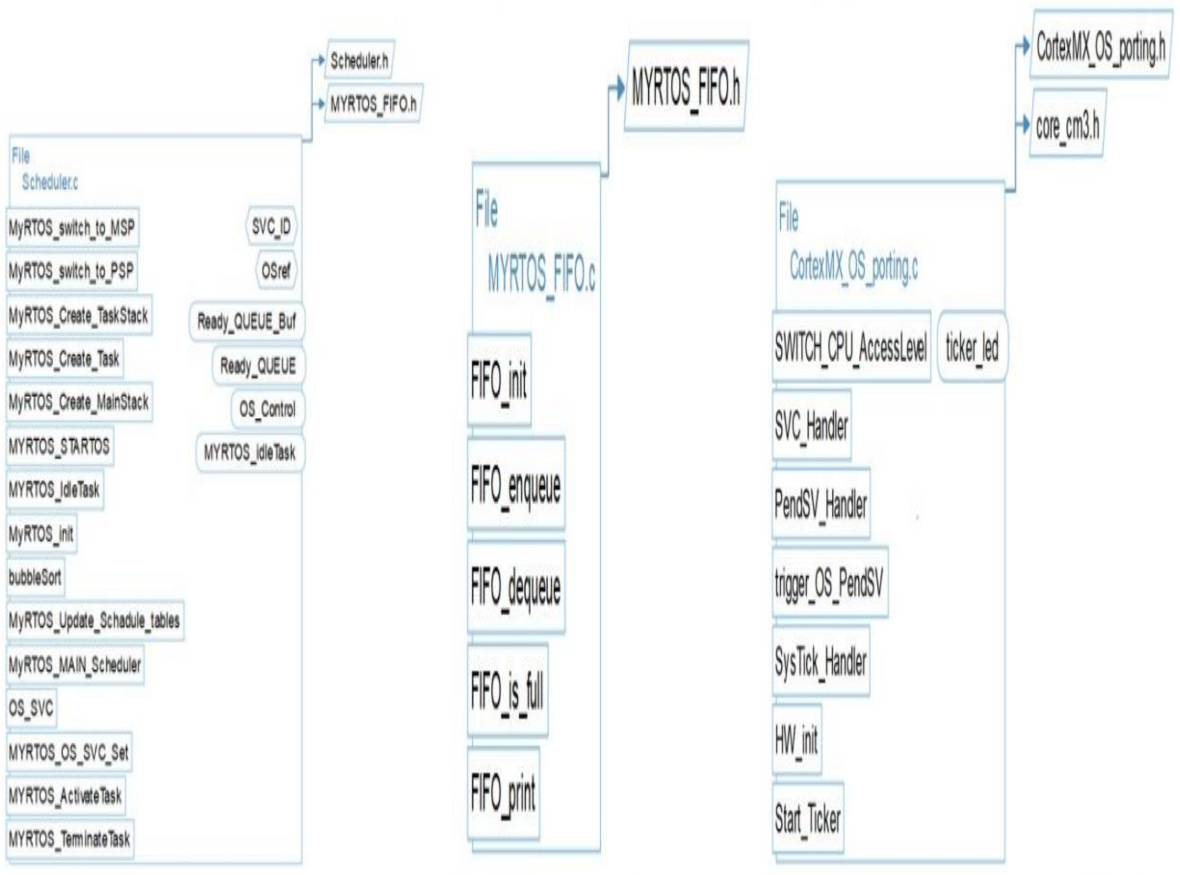
MY_OWN_RTOS characteristics :

- 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.

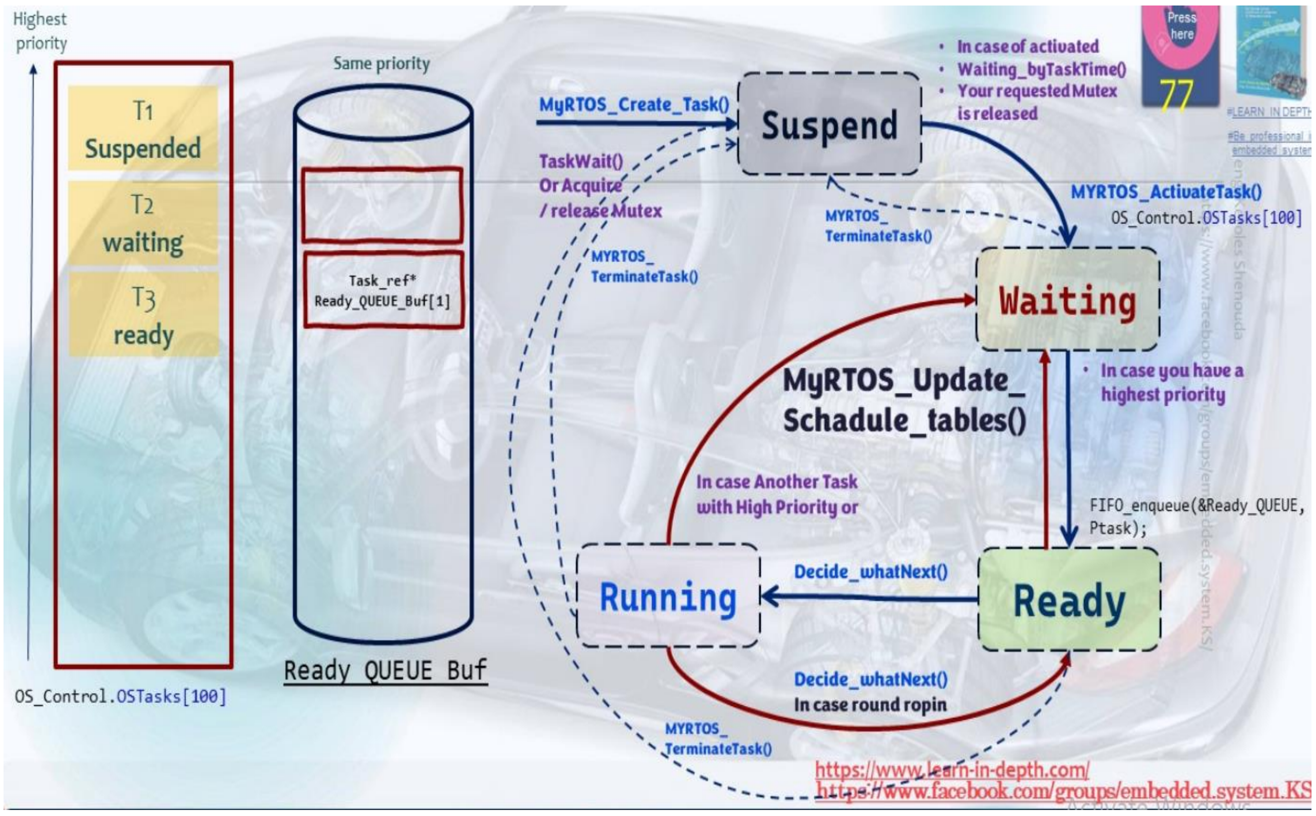
MyRTOS_Files.

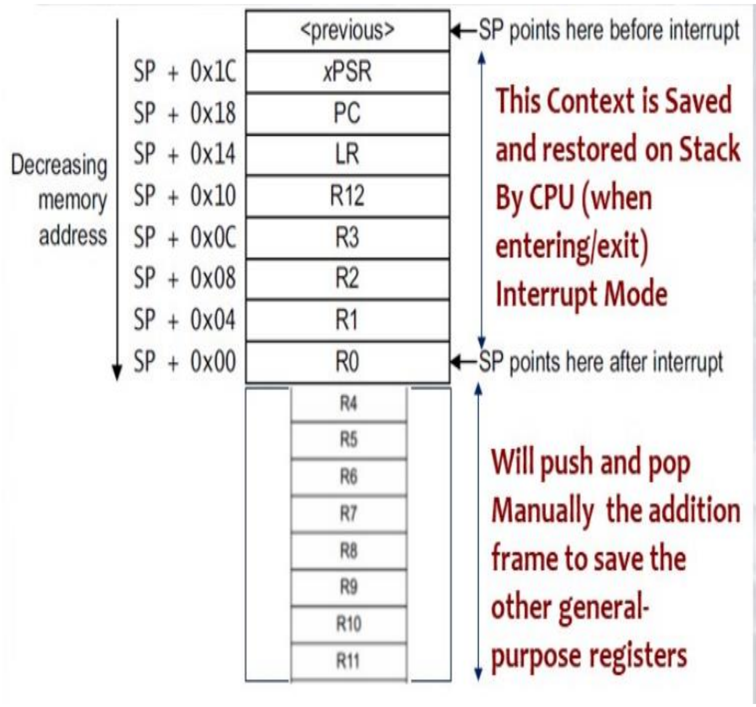


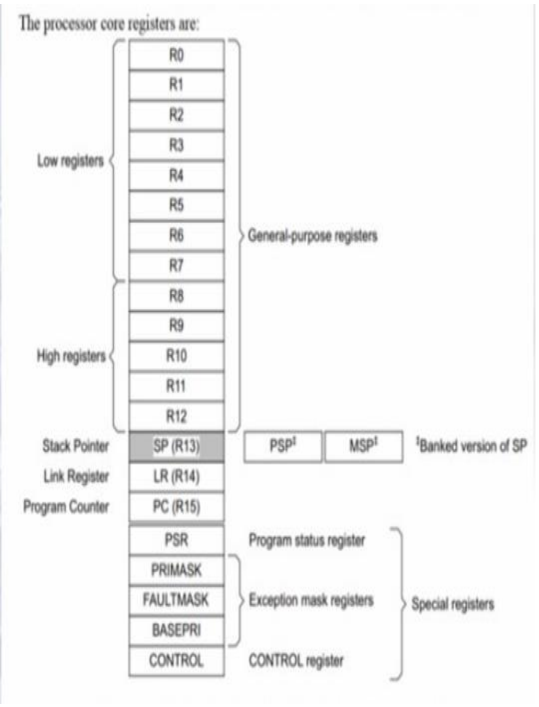




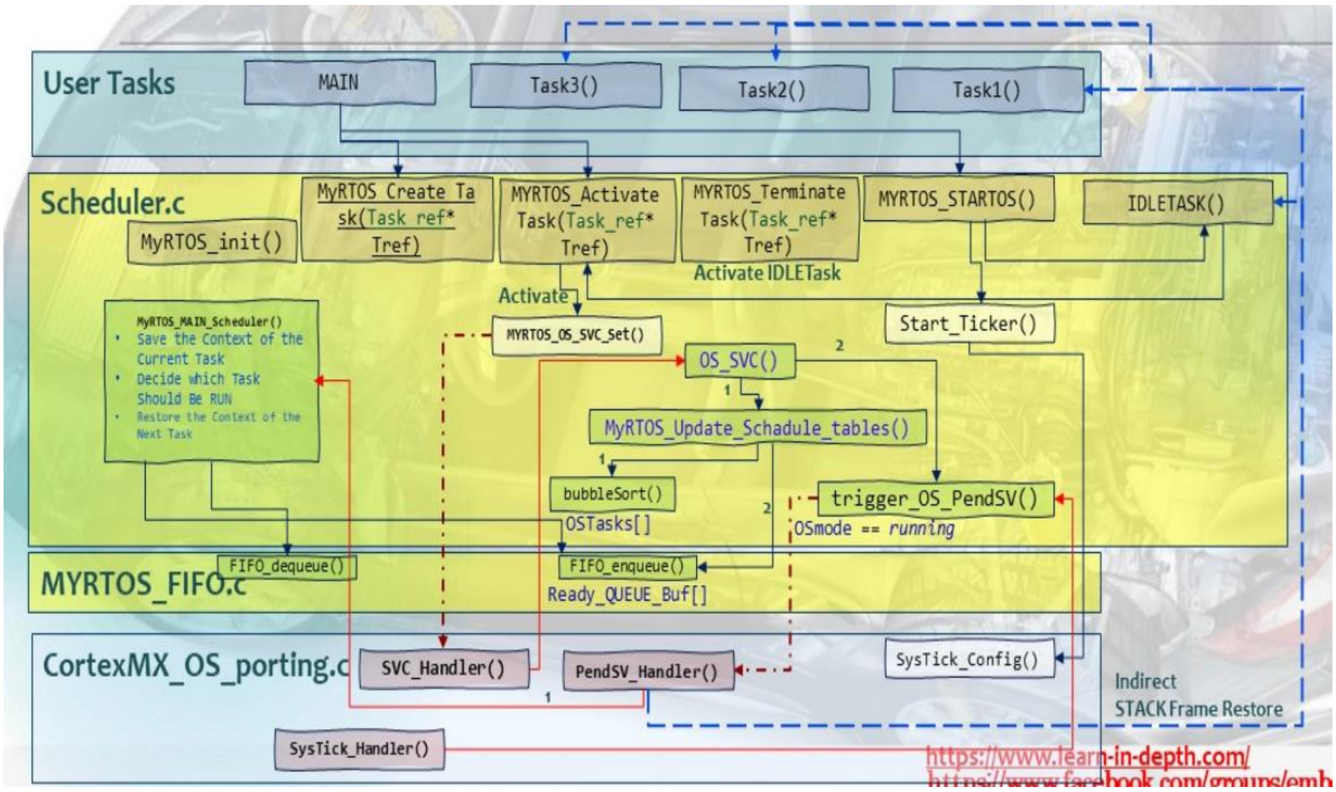
Task States & Context Switching.

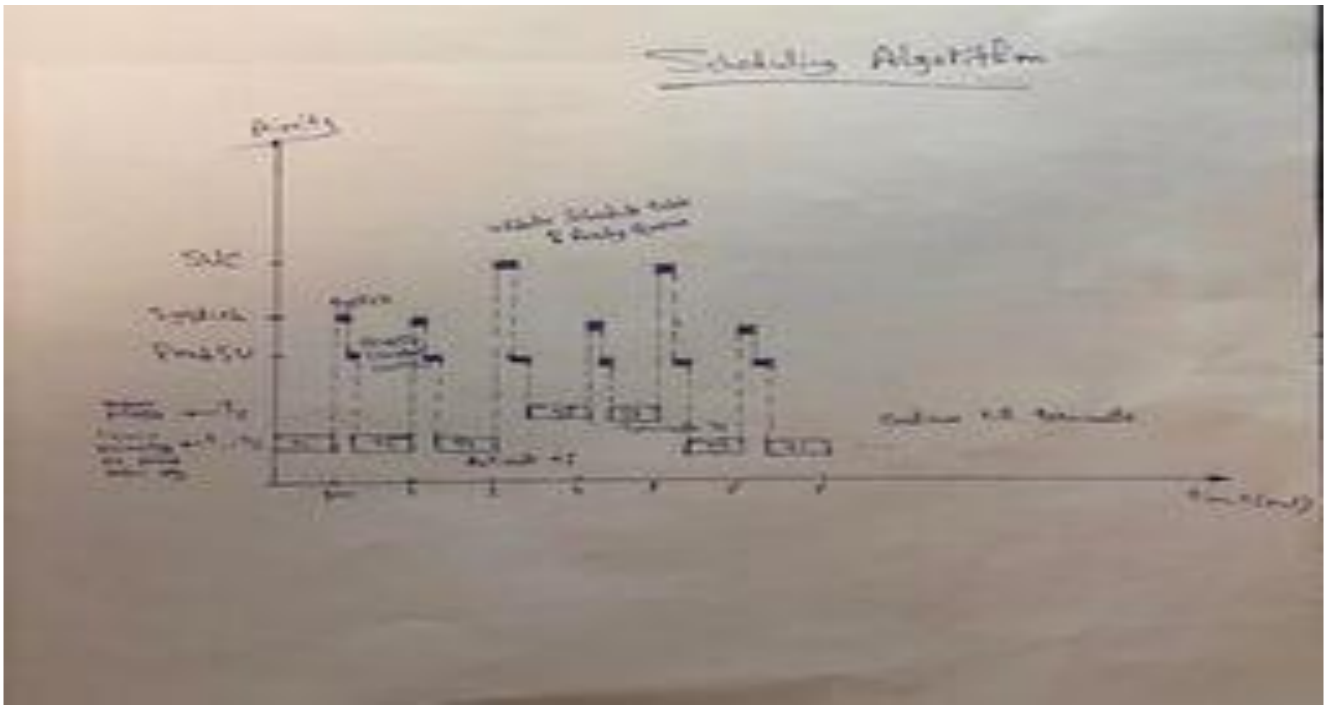



Decreasing memory address
<previous>
SP points here before interrupt
SP + 0x1C xPSR
SP + 0x18 PC
SP + 0x14 LR
SP + 0x10 R12
SP + 0x0C R3
SP + 0x08 R2
SP + 0x04 R1
SP + 0x00 R0
SP points here after interrupt
This Context is Saved and restored on Stack By CPU (when entering/exit) Interrupt Mode
R4
R5
R6
R7
R8
R9
R10
R11
Will push and pop Manually the addition frame to save the other general-purpose registers



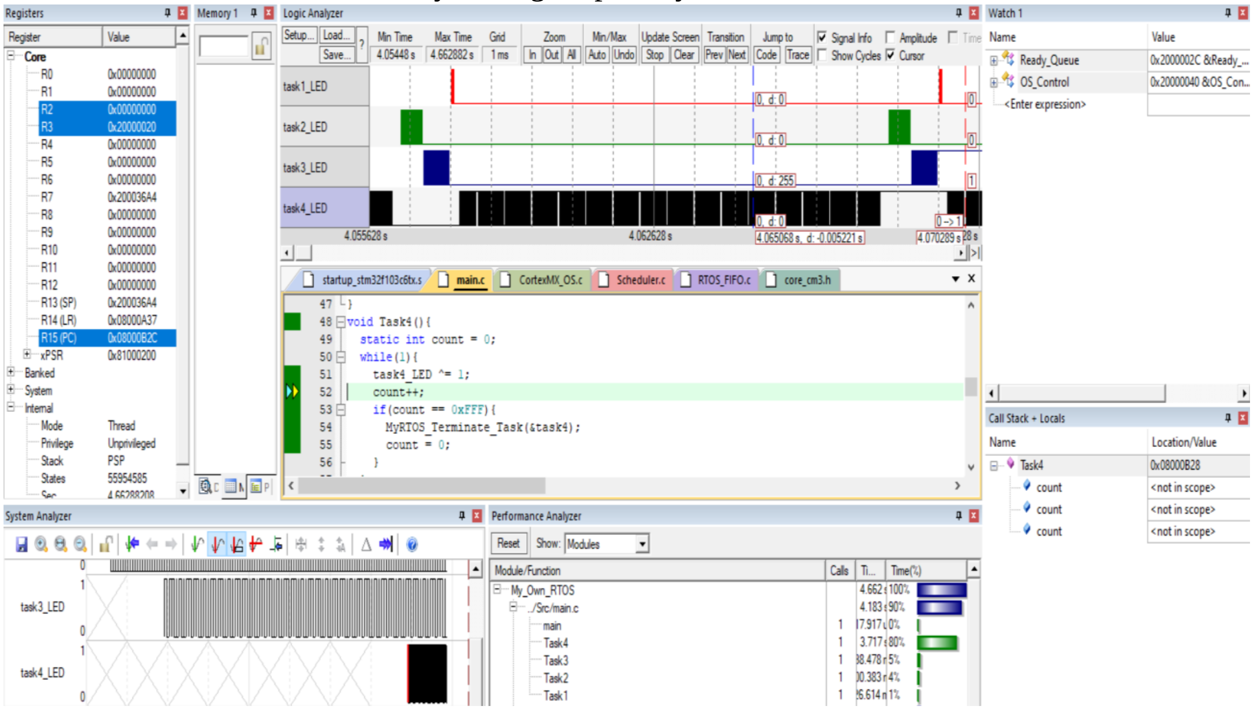
System Layers & Scheduling Algorithm.





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.



Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x20000020
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x200036A4
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200036A4
R14 (LR)	0x08000A37
R15 (PC)	0x08000B2C
xPSR	0x81000200
Mode	Thread
Privilege	Unprivileged
Stack	PSP
States	55954585

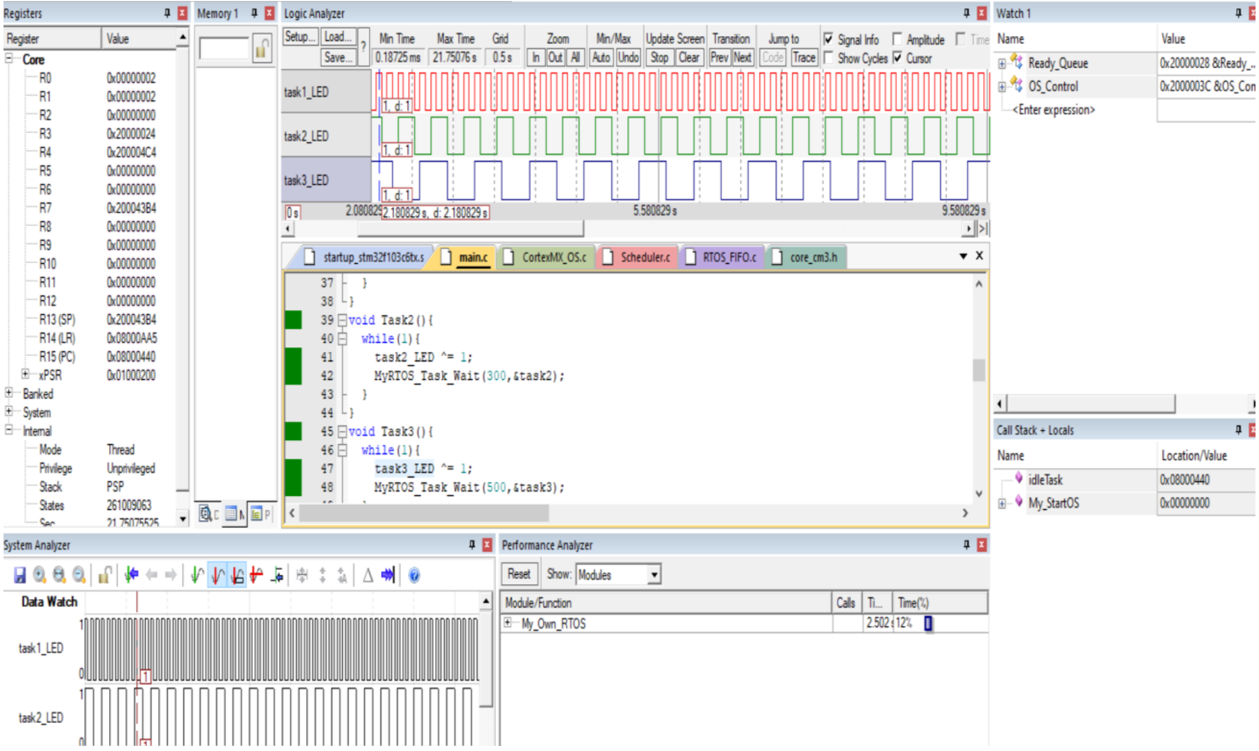
Name	Value
Ready_Queue	0x2000002C &Ready_...
OS_Control	0x20000040 &OS_Con...

```
}
void Task4(){
  static int count = 0;
  while(1){
    task4_LED ^= 1;
    count++;
    if(count == 0xFFF){
      MyRTOS_Terminate_Task(&task4);
      count = 0;
    }
```

Name	Location/Value
Task4	0x08000B28
count	<not in scope>
count	<not in scope>
count	<not in scope>

Module/Function	Calls	Ti...	Time(%)
My_Own_RTOS		4.662 s	100%
../Src/main.c		4.183 s	90%
main	1	17.917 u	0%
Task4	1	3.717 s	80%
Task3	1	38.478 m	5%
Task2	1	00.383 m	4%
Task1	1	26.614 m	1%

- 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.



Register	Value
R0	0x00000002
R1	0x00000002
R2	0x00000000
R3	0x20000024
R4	0x200004C4
R5	0x00000000
R6	0x00000000
R7	0x200043B4
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200043B4
R14 (LR)	0x08000AA5
R15 (PC)	0x08000440
xPSR	0x01000200
Mode	Thread
Privilege	Unprivileged
Stack	PSP
States	261009063

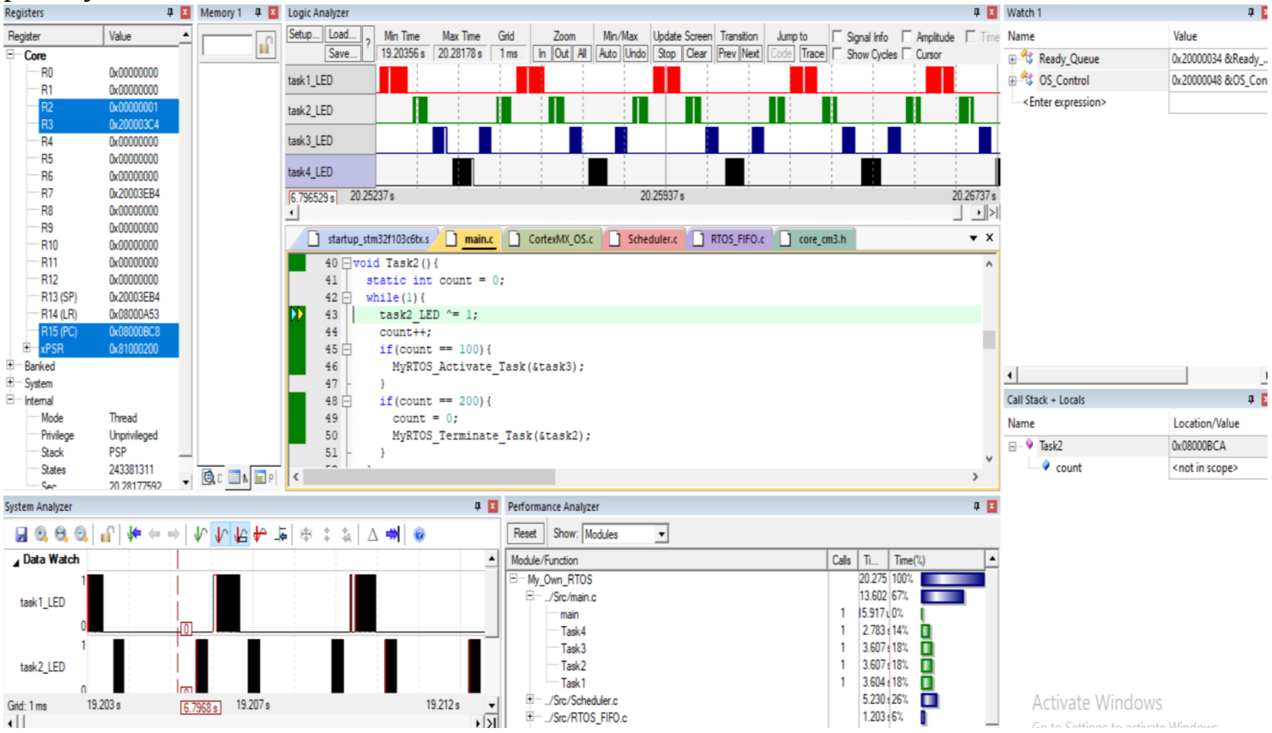
Name	Value
Ready_Queue	0x20000028 &Ready_...
OS_Control	0x2000003C &OS_Con

```
  }
}
void Task2(){
  while(1){
    task2_LED ^= 1;
    MyRTOS_Task_Wait(300,&task2);
  }
}
void Task3(){
  while(1){
    task3_LED ^= 1;
    MyRTOS_Task_Wait(500,&task3);
```

Name	Location/Value
idleTask	0x08000440
My_StartOS	0x00000000

Module/Function	Calls	Ti...	Time(%)
My_Own_RTOS		2.502 s	12%

- Task1 activates Task2 activates Task2 activates Task2 with a terminate time after 200ms, and activation time after 100ms from start execution, T4>T3>T2>T1 in priority.



Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000001
R3	0x200003C4
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x20003EB4
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20003EB4
R14 (LR)	0x08000A53
R15 (PC)	0x08000BC8
xPSR	0x81000200
Banked	
System	
Internal	
Mode	Thread
Privilege	Unprivileged
Stack	PSP
States	243381311
Sec	20.28177592

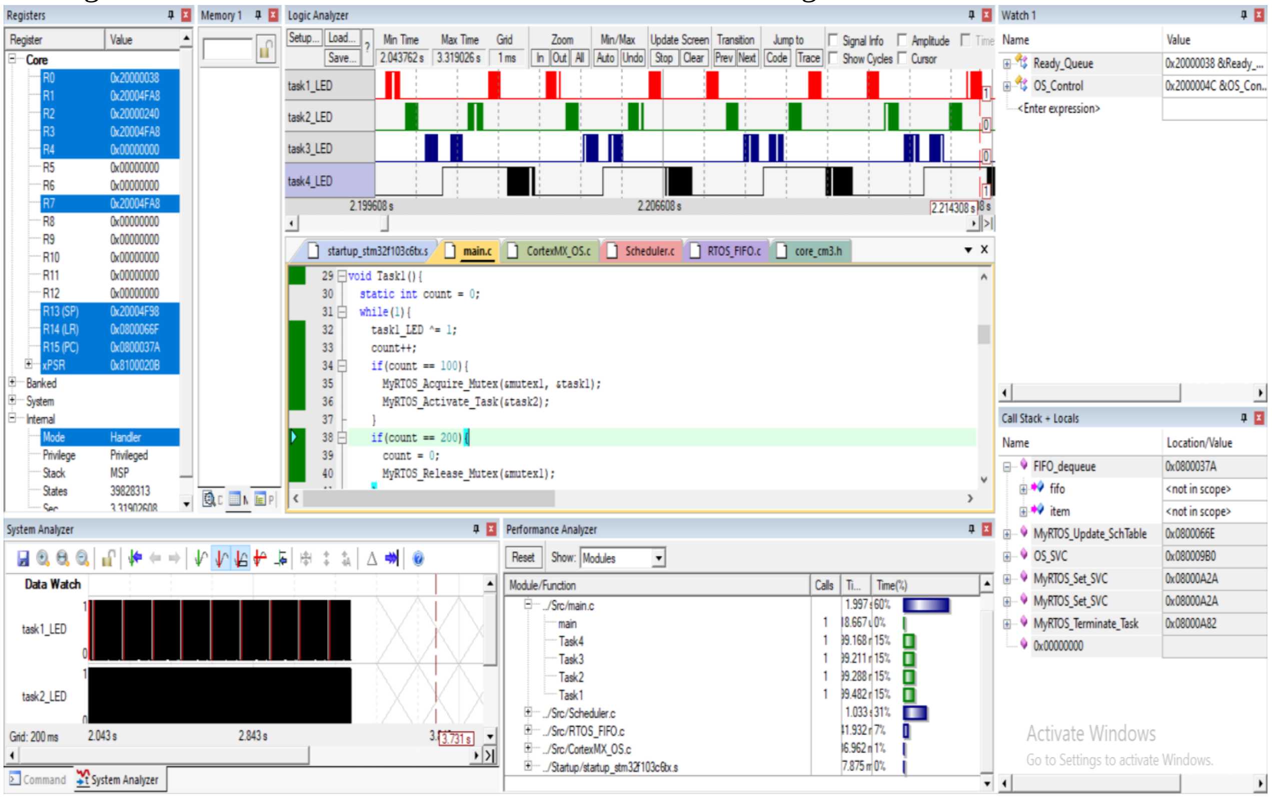
```c
void Task2(){
  static int count = 0;
  while(1){
    task2_LED ^= 1;
    count++;
    if(count == 100){
      MyRTOS_Activate_Task(&task3);
    }
    if(count == 200){
      count = 0;
      MyRTOS_Terminate_Task(&task2);
    }
```

Name	Value
Ready_Queue	0x20000034 &Ready_...
OS_Control	0x20000048 &OS_Con...
<Enter expression>	

Name	Location/Value
Task2	0x08000BCA
count	<not in scope>

Module/Function	Calls	Ti...	Time(%)
My_Own_RTOS		20.275	100%
./Src/main.c		13.602	67%
main	1	15.917	0%
Task4	1	2.783	14%
Task3	1	3.607	18%
Task2	1	3.607	18%
Task1	1	3.604	18%
./Src/Scheduler.c		5.230	26%
./Src/RTOS_FIFO.c		1.203	6%

- 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.



Register	Value
Core	
R0	0x20000038
R1	0x20004FA8
R2	0x20000240
R3	0x20004FA8
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x20004FA8
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20004F98
R14 (LR)	0x0800066F
R15 (PC)	0x0800037A
xPSR	0x8100020B
Banked	
System	
Internal	
Mode	Handler
Privilege	Privileged
Stack	MSP
States	39828313
Sec	3.31902608

```c
void Task1(){
  static int count = 0;
  while(1){
    task1_LED ^= 1;
    count++;
    if(count == 100){
      MyRTOS_Acquire_Mutex(&mutex1, &task1);
      MyRTOS_Activate_Task(&task2);
    }
    if(count == 200){
      count = 0;
      MyRTOS_Release_Mutex(&mutex1);
```

Name	Value
Ready_Queue	0x20000038 &Ready_...
OS_Control	0x2000004C &OS_Con...
<Enter expression>	

Name	Location/Value
FIFO_dequeue	0x0800037A
fifo	<not in scope>
item	<not in scope>
MyRTOS_Update_SchTable	0x0800066E
OS_SVC	0x080009B0
MyRTOS_Set_SVC	0x08000A2A
MyRTOS_Set_SVC	0x08000A2A
MyRTOS_Terminate_Task	0x08000A82
0x00000000	

Module/Function	Calls	Ti...	Time(%)
./Src/main.c		1.997	60%
main	1	18.667	0%
Task4	1	19.168	15%
Task3	1	19.211	15%
Task2	1	19.288	15%
Task1	1	19.482	15%
./Src/Scheduler.c		1.033	31%
./Src/RTOS_FIFO.c		11.932	7%
./Src/CortexMX_OS.c		16.962	1%
./Startup/startup_stm32f103c6tx.s		7.875	0%