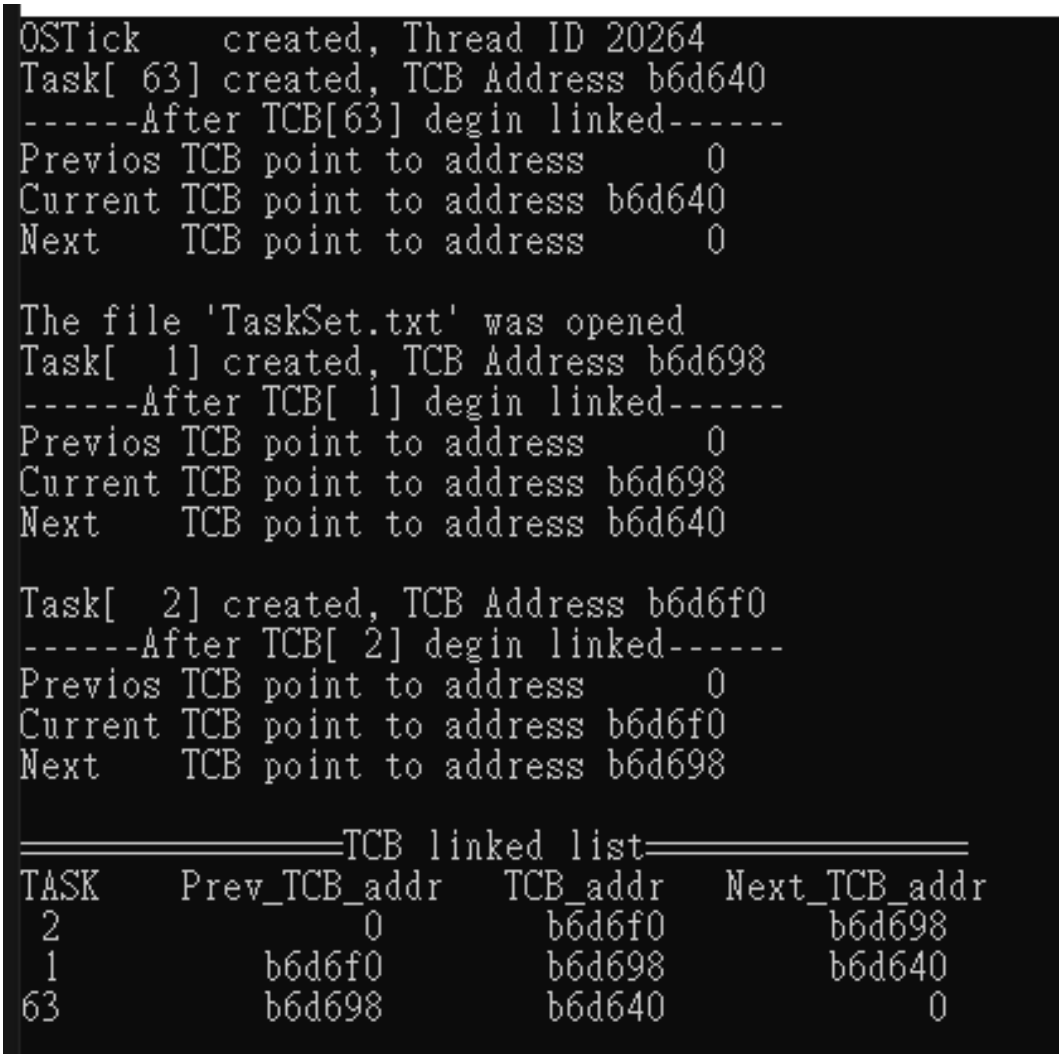
RTOS_M11102155_PA1

Name : 楊鏡業

Student ID : M11102155

[Part 1] Task Control Block Linked List

Screenshot Result :



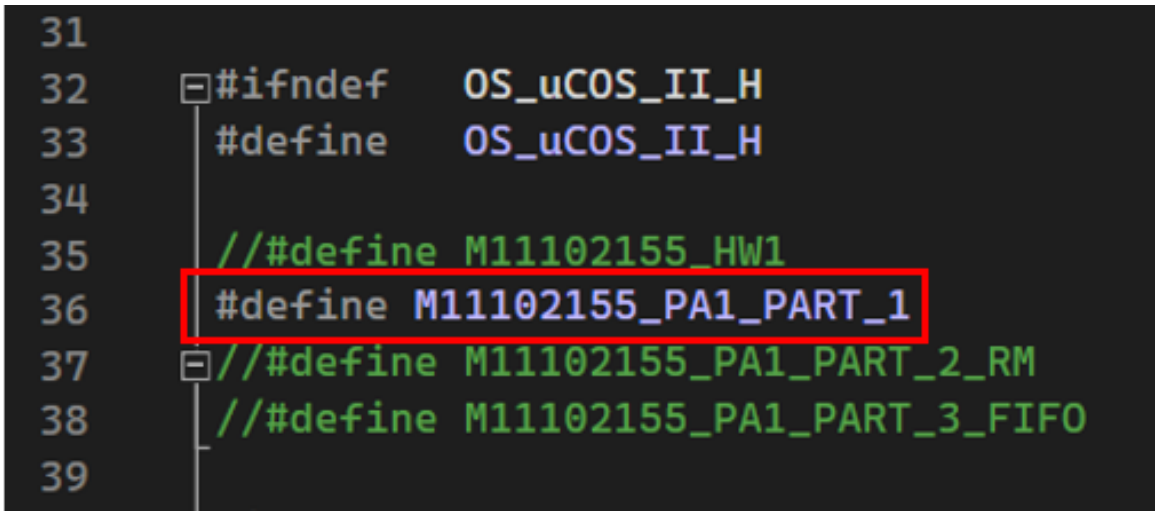
Code Review :

The code I modify will wrapped between

```
#ifdef M11102155_PA1_PART_1

    # modify code !!!

#endif /* M11102155_PA1_PART_1 */
```

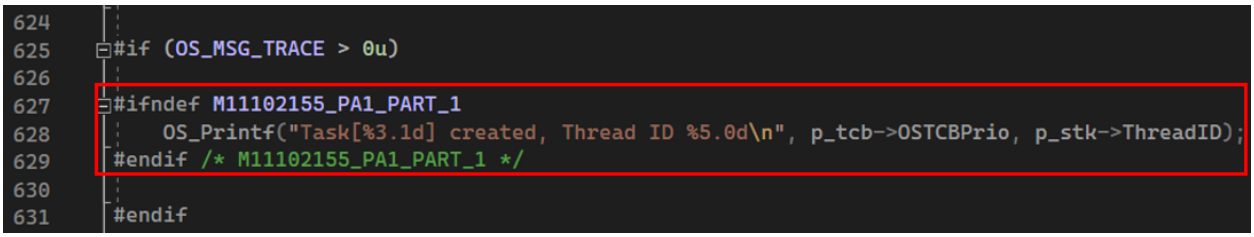
usos_ii.h :

For using #ifdef M11102155_PA1_PART_1 #endif in all code segament.



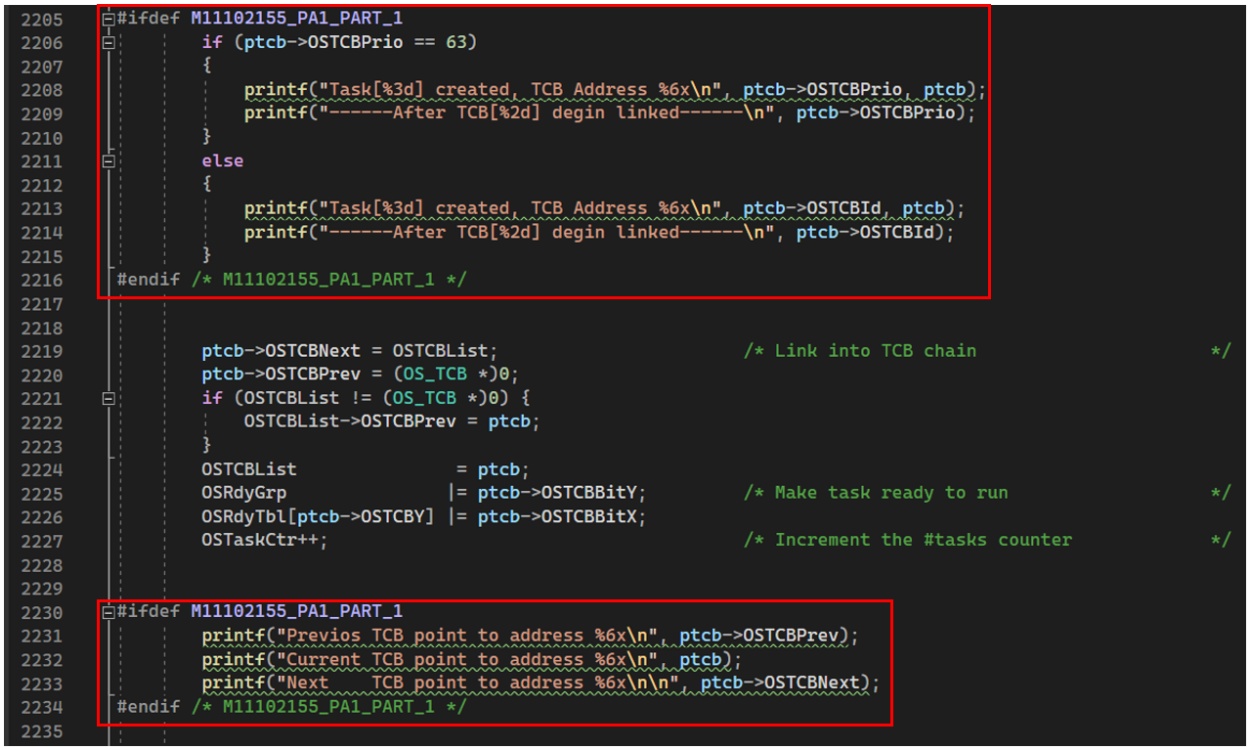
os_cpu_c.c :

In “ OSTCBInitHook() ”, close print the information of create task.



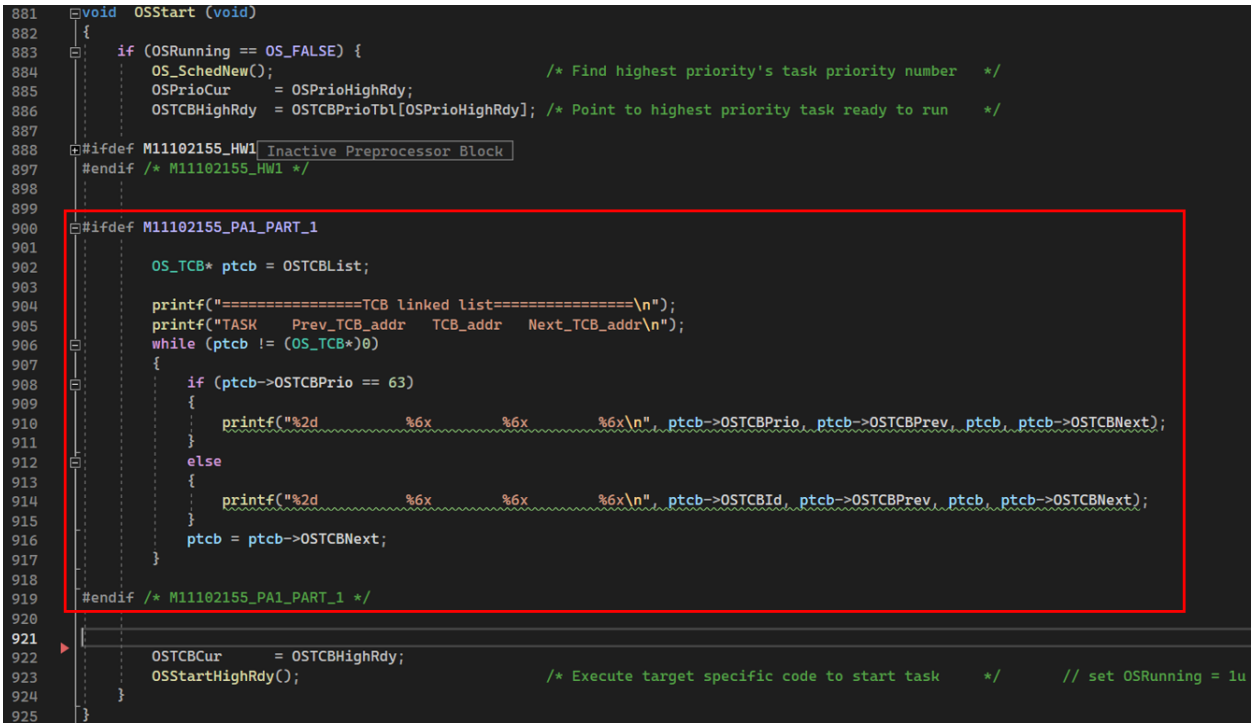
os_core.c :

In “ OS_TCBInit() “, show the double linked list result of TCB List after insert the new task’s TCB into the list. I separate the print code by the operations of linking linked list of TCB List.



```
2205  #ifdef M11102155_PA1_PART_1
2206          if (ptcb->OSTCBPrio == 63)
2207          {
2208              printf("Task[%3d] created, TCB Address %6x\n", ptcb->OSTCBPrio, ptcb);
2209              printf("------After TCB[%2d] degin linked------\n", ptcb->OSTCBPrio);
2210          }
2211          else
2212          {
2213              printf("Task[%3d] created, TCB Address %6x\n", ptcb->OSTCBId, ptcb);
2214              printf("------After TCB[%2d] degin linked------\n", ptcb->OSTCBId);
2215          }
2216  #endif /* M11102155_PA1_PART_1 */
2217
2218
2219          ptcb->OSTCBNext = OSTCBList;                    /* Link into TCB chain                  */
2220          ptcb->OSTCBPrev = (OS_TCB *)0;
2221          if (OSTCBList != (OS_TCB *)0) {
2222              OSTCBList->OSTCBPrev = ptcb;
2223          }
2224          OSTCBList               = ptcb;
2225          OSRdyGrp               |= ptcb->OSTCBBitY;      /* Make task ready to run               */
2226          OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2227          OSTaskCtr++;                                    /* Increment the #tasks counter         */
2228
2229
2230  #ifdef M11102155_PA1_PART_1
2231          printf("Previos TCB point to address %6x\n", ptcb->OSTCBPrev);
2232          printf("Current TCB point to address %6x\n", ptcb);
2233          printf("Next    TCB point to address %6x\n\n", ptcb->OSTCBNext);
2234  #endif /* M11102155_PA1_PART_1 */
2235
```

In “ OSStart() ”, show the final result of the TCB List in double linked list after creating all tasks and insert them into TCB List.



```
881  void  OSStart (void)
882  {
883      if (OSRunning == OS_FALSE) {
884          OS_SchedNew();                           /* Find highest priority's task priority number   */
885          OSPrioCur     = OSPrioHighRdy;
886          OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
887
888  #ifdef M11102155_HW1  Inactive Preprocessor Block
897  #endif /* M11102155_HW1 */
898
899
900  #ifdef M11102155_PA1_PART_1
901
902          OS_TCB* ptcb = OSTCBList;
903
904          printf("================TCB linked list================\n");
905          printf("TASK    Prev_TCB_addr   TCB_addr   Next_TCB_addr\n");
906          while (ptcb != (OS_TCB*)0)
907          {
908              if (ptcb->OSTCBPrio == 63)
909              {
910                  printf("%2d          %6x         %6x         %6x\n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
911              }
912              else
913              {
914                  printf("%2d          %6x         %6x         %6x\n", ptcb->OSTCBId, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
915              }
916              ptcb = ptcb->OSTCBNext;
917          }
918
919  #endif /* M11102155_PA1_PART_1 */
920
921
922          OSTCBCur      = OSTCBHighRdy;
923          OSStartHighRdy();                        /* Execute target specific code to start task     */       // set OSRunning = 1u
924      }
925  }
```

[Part 2] RM Scheduler Implementation

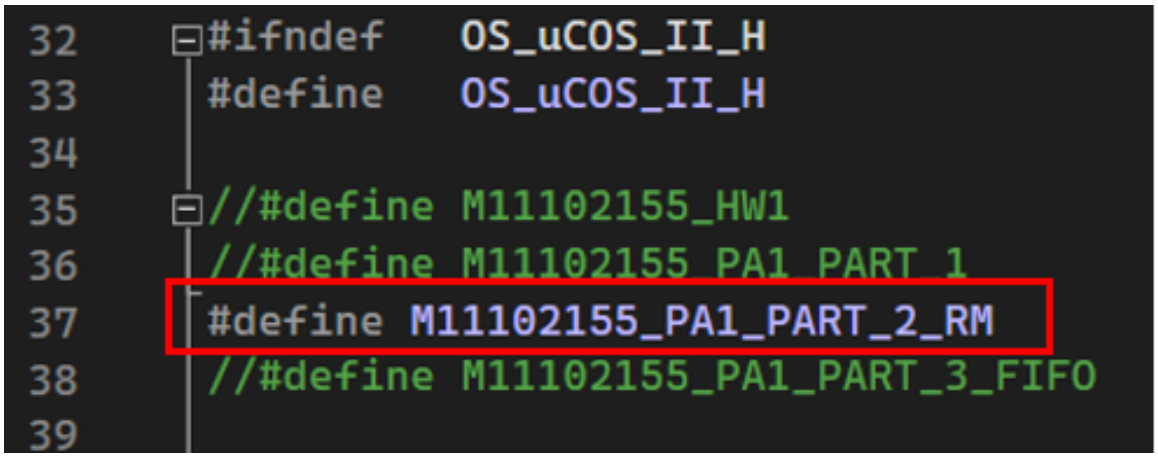
Code Review :

The code I modify will wrapped between

```
#ifdef M11102155_PA1_PART_2_RM

    # modify code !!!

#endif /* M11102155_PA1_PART_2_RM */
```

usos_ii.h :

For using #ifdef M11102155_PA1_PART_2_RM #endif in all code segament.



In structure “ OS_TCB ”, i insert some parameters to record the inforamtions when doing RMS.

```
619
620  typedef struct os_tcb {
621      OS_STK          *OSTCBStkPtr;           /* Pointer to current top of stack                          */
622
623  #if OS_TASK_CREATE_EXT_EN > 0u
624      void            *OSTCBExtPtr;           /* Pointer to user definable data for TCB extension         */
625      OS_STK          *OSTCBStkBottom;        /* Pointer to bottom of stack                               */
626      INT32U           OSTCBStkSize;          /* Size of task stack (in number of stack elements)         */
627      INT16U           OSTCBOpt;              /* Task options as passed by OSTaskCreateExt()              */
628      INT16U           OSTCBId;               /* Task ID (0..65535)                                       */
629  #endif
630
631      struct os_tcb   *OSTCBNext;             /* Pointer to next     TCB in the TCB list                  */
632      struct os_tcb   *OSTCBPrev;             /* Pointer to previous TCB in the TCB list                  */
633
634  #if OS_TASK_CREATE_EXT_EN > 0u
635  #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
636      OS_TLS           OSTCBTLSTbl[OS_TLS_TBL_SIZE];
637  #endif
638  #endif
639
640  #if (OS_EVENT_EN)
641      OS_EVENT        *OSTCBEventPtr;         /* Pointer to          event control block                  */
642  #endif
643
644  #if (OS_EVENT_EN) && (OS_EVENT_MULTI_EN > 0u)
645      OS_EVENT       **OSTCBEventMultiPtr;    /* Pointer to multiple event control blocks                 */
646  #endif
647
648  #if ((OS_Q_EN > 0u) && (OS_MAX_QS > 0u)) || (OS_MBOX_EN > 0u)
649      void            *OSTCBMsg;              /* Message received from OSMboxPost() or OSQPost()          */
650  #endif
651
652  #if (OS_FLAG_EN > 0u) && (OS_MAX_FLAGS > 0u)
653  #if OS_TASK_DEL_EN > 0u
654      OS_FLAG_NODE    *OSTCBFlagNode;         /* Pointer to event flag node                               */
655  #endif
656      OS_FLAGS         OSTCBFlagsRdy;         /* Event flags that made task ready to run                  */
657  #endif
658
659      INT32U           OSTCBDly;              /* Nbr ticks to delay task or, timeout waiting for event    */
660      INT8U            OSTCBStat;             /* Task      status                                         */
661      INT8U            OSTCBStatPend;         /* Task PEND status                                         */
662      INT8U            OSTCBPrio;             /* Task priority (0 == highest)                             */
663
664      INT8U            OSTCBX;                /* Bit position in group  corresponding to task priority    */
665      INT8U            OSTCBY;                /* Index into ready table corresponding to task priority    */
666      OS_PRIO          OSTCBBitX;             /* Bit mask to access bit position in ready table           */
667      OS_PRIO          OSTCBBitY;             /* Bit mask to access bit position in ready group           */
668
669  #if OS_TASK_DEL_EN > 0u
670      INT8U            OSTCBDelReq;           /* Indicates whether a task needs to delete itself          */
671  #endif
672
673  #if OS_TASK_PROFILE_EN > 0u
674      INT32U           OSTCBCtxSwCtr;         /* Number of time the task was switched in                  */
675      INT32U           OSTCBCyclesTot;        /* Total number of clock cycles the task has been running   */
```

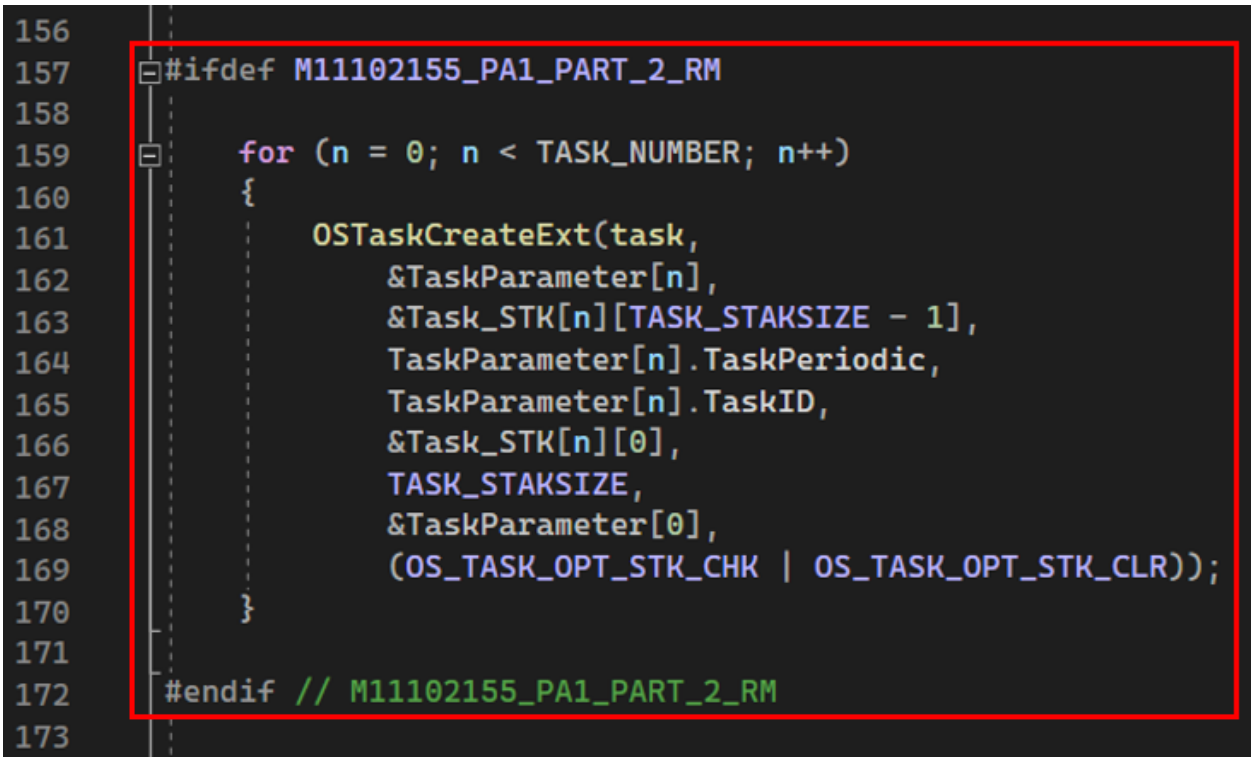
```
676     INT32U           OSTCBCyclesStart;      /* Snapshot of cycle counter at start of task resumption   */
677     OS_STK          *OSTCBStkBase;          /* Pointer to the beginning of the task stack              */
678     INT32U           OSTCBStkUsed;          /* Number of bytes used from the stack                     */
679 #endif
680
681 #if OS_TASK_NAME_EN > 0u
682     INT8U           *OSTCBTaskName;
683 #endif
684
685 #if OS_TASK_REG_TBL_SIZE > 0u
686     INT32U           OSTCBRegTbl[OS_TASK_REG_TBL_SIZE];
687 #endif
688
689 #ifdef M11102155_PA1_PART_2_RM
690
691     INT16U num_times_job;                // Records the number of times (assume j) this task occurs periodically.
692     INT16U num_recent_execute_time;      // Records the time (in ticks) that the task has executed at time j.
693     INT16U total_execute_time;           // Records the worst-case execution time of the task.
694     INT16U arrive_time;                  // Records the arrival time of the task at time j.
695     INT16U period;                       // Record period of the task.
696     INT16U deadline_time;                // Records the deadline of the task at time j.
697     INT16U response_time;                // Record the response time of the task at time j.
698
699
700     /*
701        - e.g. : task 5's execution time is 3 ticks and occur periodically for every 8 ticks, ask the recod of third time task 5 occur.
702            - num_times_job          =  3 (third time occur)
703            - num_recent_execute_time =  0 (havn't execute yet)
704            - total_execute_time      =  3 (worst total execute time)
705            - arrive_time             = 24 (task 5's third time arrive time is at tick 24)
706            - deadline_time           = 32 (task 5's forth time arrive time is at tick 24 + 8 = 32)
707
708        - assume after task execute 2 ticks and preemtive by other HPT
709            - num_times_job          =  3 (third time occur)
710            - num_recent_execute_time =  2 (execute 2 ticks)
711            - total_execute_time      =  3 (worst total execute time)
712            - arrive_time             = 24 (task 5's third time arrive time is at tick 24)
713            - deadline_time           = 32 (task 5's forth time arrive time is at tick 24 + 8 = 32)
714     */
715
716 #endif /* M11102155_PA1_PART_2_RM */
717
718 } OS_TCB;
719
```

main.c :

Function define :

```
69  #ifdef M11102155_PA1_PART_2_RM
70  static void task(void* p_arg);
71  #endif /* M11102155_PA1_PART_2_RM */
72
73  #ifdef M11102155_PA1_PART_1 Inactive Preprocessor Block
76  #endif /* M11102155_PA1_PART_1 */
77
78  static  void  StartupTask (void  *p_arg);
```

Create the task with number of input task, and set the task’s period as task’s priority :



```
156
157 #ifdef M11102155_PA1_PART_2_RM
158
159     for (n = 0; n < TASK_NUMBER; n++)
160     {
161         OSTaskCreateExt(task,
162             &TaskParameter[n],
163             &Task_STK[n][TASK_STAKSIZE - 1],
164             TaskParameter[n].TaskPeriodic,
165             TaskParameter[n].TaskID,
166             &Task_STK[n][0],
167             TASK_STAKSIZE,
168             &TaskParameter[0],
169             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
170     }
171
172 #endif // M11102155_PA1_PART_2_RM
173
```

In “ task() ”, first initial the count of the task has executed and the deadline of the first job, then start to periodically do task.

In the inner while loop the task will keep executing until occur interrupt “ Tick ISR ” at the end of each tick, and the count of “ num_recent_execute_time “, means the count of task has executed in current job, will increase by one and consider whether the amount is greater than the total executed time (task finish executing this job) or not. If is, it will break the while loop and do the delay, the delay time will be compute in the OSTimeDly() function. On the other hand, if is not, task will keep doing this job until finish.

Here is a special case, in Task Set 2 at tick 11, the current job of task2 is done but Tick interrupt occur earlier than OSTimeDly(), the task 3 preempt task 2 before task 2 do delay, so task 2 will do delay after task 3 finish it’s job, here i conquer this issue by checking whether this case happen or not in Tick ISR. After adding the condition, this case’s task (e.g. task 2) will be complete (delay) during Tick ISR before preempted by task 3. Futhermore I reset the “ num_recent_execute_time “ before the task be ready, so when task 2 become ready it will still stuck at inner loop and doing the test, consider this job is done or not, won’t repeat doing the OSTimeDly().

```
228
229  #ifdef M11102155_PA1_PART_2_RM
230
231  static void task(void* p_arg)
232  {
233      task_para_set* task_date;
234      task_date = p_arg;
235
236      // Initial #executed time(num_recent_execute_time) and the deadline of the task.
237      OS_ENTER_CRITICAL();
238      OSTCBCur->num_recent_execute_time = 0;
239      OSTCBCur->deadline_time = OSTCBCur->arrive_time + OSTCBCur->period;
240      OS_EXIT_CRITICAL();
241
242      while (1)
243      {
244          // For every task keep executing unitl finish or preemptive
245          while (OSTCBCur->num_recent_execute_time < OSTCBCur->total_execute_time);    // TimeTick interrupt happen before while loop end !!!
246
247          OSTimeDly(0);
248      }
249  }
250
251  #endif /* M11102155_PA1_PART_2_RM */
```

os_time.c :

In “ OSTimeDly() “, i assume the input argument is zero, cause i’ll compute the delay time by using the informations store in TCB. At the beginning it will compute two amount, first is the actual delay time, ticks, it will be compute by the task’s deadline time minus the current tick time. And the second is the response time of the task, cause when the task enter this function, it means the task is already finish the job.

```
55  void  OSTimeDly (INT32U ticks)
56  {
57      INT8U      y;
58  #if OS_CRITICAL_METHOD == 3u                 /* Allocate storage for CPU status register           */
59      OS_CPU_SR  cpu_sr = 0u;
60  #endif
61
62  #ifdef M11102155_PA1_PART_2_RM
63      // Compute the delay time before task arrive again.
64      OS_ENTER_CRITICAL();
65      ticks = OSTCBCur->deadline_time - OSTime;
66      OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
67      OS_EXIT_CRITICAL();
68  #endif /* M11102155_PA1_PART_2_RM */
69
70
71
72      if (OSIntNesting > 0u) {                 /* See if trying to call from an ISR                 */
73          return;
74      }
75      if (OSLockNesting > 0u) {                /* See if called with scheduler locked               */
76          return;
77      }
78      if (ticks > 0u) {                        /* 0 means no delay!                                 */
79          OS_ENTER_CRITICAL();
80          y             =  OSTCBCur->OSTCBY;   /* Delay current task                                */
81          OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
82          OS_TRACE_TASK_SUSPENDED(OSTCBCur);
83          if (OSRdyTbl[y] == 0u) {
84              OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
85          }
86          OSTCBCur->OSTCBDly = ticks;          /* Load ticks in TCB                                 */
87          OS_TRACE_TASK_DLY(ticks);
88          OS_EXIT_CRITICAL();
89          OS_Sched();                          /* Find next task to run!                            */
90      }
91  }
```

os_core.c :

In os_core.c i divided into 3 parts :

1. TCB initialization when creating the tasks.
2. Task's job is complete and call the OSTimeDly() then enter the OS_Sched() for finding the next high priority task to execute.
3. For all Tick Interrupt occur in every end of the time tick, it will consider the task's job is complete or preemt by other high priority task, and also check whether exist job violate (overflow).

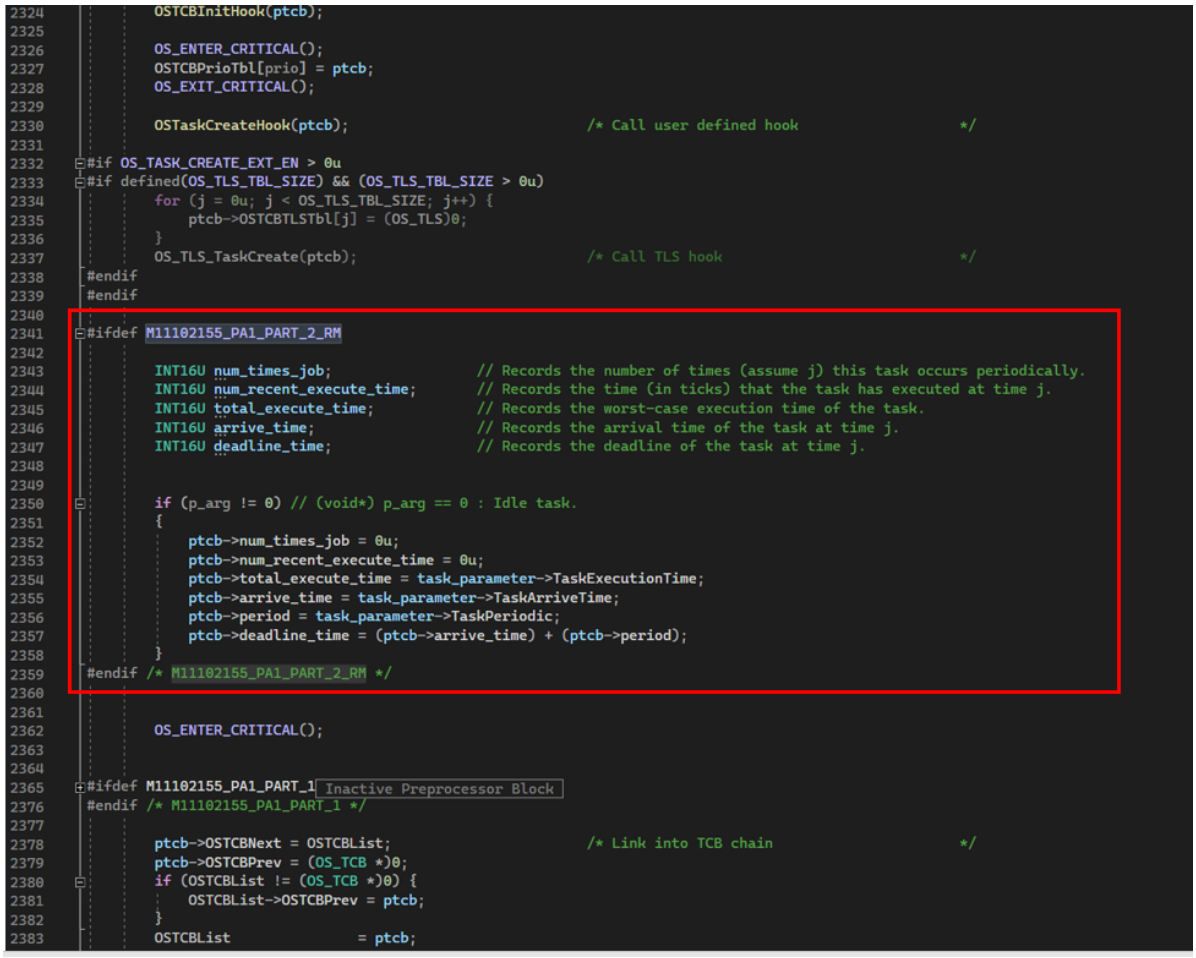
Start from here :

1. In " OS_TCBInit() ", performing the TCB initialization when createing the task. First transfer the pointer to the task's parameter then deal with the tasks with different arrive time in the beginning, set the arrive time as delay, call attention to, if " p_arg == 0 " means it's the Ilde task. Second, initial the parameter for recording each task

job’s scheduling informations. Lastly, we set the task with “ arrive time = 0 ” and Idle task ready.

```
2198
2199    INT8U  OS_TCBInit (INT8U    prio,
2200                       OS_STK  *ptos,
2201                       OS_STK  *pbos,
2202                       INT16U   id,
2203                       INT32U   stk_size,
2204                       void    *pext,
2205                       INT16U   opt,
2206                       void*   p_arg)
2207    {
2208        OS_TCB    *ptcb;
2209    #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
2210        OS_CPU_SR  cpu_sr = 0u;
2211    #endif
2212    #if OS_TASK_REG_TBL_SIZE > 0u
2213        INT8U      i;
2214    #endif
2215    #if OS_TASK_CREATE_EXT_EN > 0u
2216    #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
2217        INT8U      j;
2218    #endif
2219    #endif
2220
2221    #ifdef M11102155_PA1_PART_2_RM
2222        task_para_set* task_parameter = (task_para_set*)p_arg;
2223        //if (p_arg != 0)      // (void*) p_arg == 0 : Idle task.
2224        //{
2225        //    printf("Task ID = %2d,  arr_time = %2d,  exe_time = %2d,  period_time = %2d,   prio = %2d\n", task_parameter ->TaskID,
2226        // task_parameter ->TaskArriveTime, task_parameter->TaskExecutionTime, task_parameter->TaskPeriodic, task_parameter->TaskPriority);
2227        //}
2228    #endif /* M11102155_PA1_PART_2_RM */
2229
2230
2231
2232        OS_ENTER_CRITICAL();
2233        ptcb = OSTCBFreeList;                             /* Get a free TCB from the free TCB list    */
2234        if (ptcb != (OS_TCB *)0) {
2235            OSTCBFreeList           = ptcb->OSTCBNext;    /* Update pointer to free TCB list          */
2236            OS_EXIT_CRITICAL();
2237            ptcb->OSTCBStkPtr       = ptos;               /* Load Stack pointer in TCB                */
2238            ptcb->OSTCBPrio         = prio;               /* Load task priority into TCB              */
2239            ptcb->OSTCBStat         = OS_STAT_RDY;        /* Task is ready to run                     */
2240            ptcb->OSTCBStatPend     = OS_STAT_PEND_OK;    /* Clear pend status                        */
```

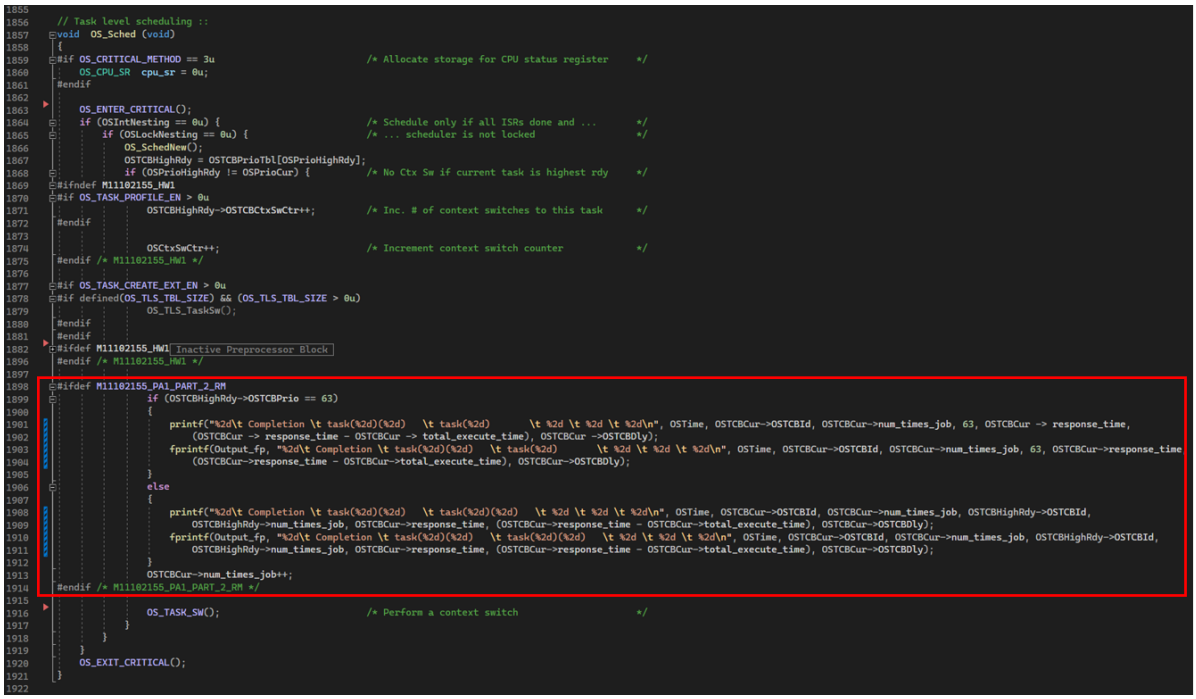
```
2241
2242
2243    #ifndef M11102155_PA1_PART_2_RM
2244            ptcb->OSTCBDly           = 0u;                    /* Task is not delayed                    */
2245    #else
2246            if (p_arg != 0)
2247            {
2248                //ptcb->OSTCBStat = OS_STAT_SUSPEND;
2249                ptcb->OSTCBDly = (INT32U)(task_parameter->TaskArriveTime);    //  Cuase task have arrive time, we set the task delay
2250                                                                              //  #arrive time to meet the task's arrive time.
2251
2252                //printf("TASK %d delay init %d\n", task_parameter->TaskID, task_parameter->TaskArriveTime);
2253            }
2254            else
2255            {
2256                ptcb->OSTCBDly = 0u;
2257            }
2258    #endif /* M11102155_PA1_PART_2_RM */
2259
2260
2261
2262    #if OS_TASK_CREATE_EXT_EN > 0u
2263            ptcb->OSTCBExtPtr        = pext;                  /* Store pointer to TCB extension         */
2264            ptcb->OSTCBStkSize       = stk_size;              /* Store stack size                       */
2265            ptcb->OSTCBStkBottom     = pbos;                  /* Store pointer to bottom of stack       */
2266            ptcb->OSTCBOpt           = opt;                   /* Store task options                     */
2267            ptcb->OSTCBId            = id;                    /* Store task ID                          */
2268    #else
2269            pext                     = pext;                  /* Prevent compiler warning if not used   */
2270            stk_size                 = stk_size;
2271            pbos                     = pbos;
2272            opt                      = opt;
2273            id                       = id;
2274    #endif
2275
2276    #if OS_TASK_DEL_EN > 0u
2277            ptcb->OSTCBDelReq        = OS_ERR_NONE;
2278    #endif
2279
2280    #if OS_LOWEST_PRIO <= 63u                                    /* Pre-compute X, Y                       */
2281            ptcb->OSTCBY             = (INT8U)(prio >> 3u);
2282            ptcb->OSTCBX             = (INT8U)(prio & 0x07u);
2283    #else                                                        /* Pre-compute X, Y                       */
2284            ptcb->OSTCBY             = (INT8U)((INT8U)(prio >> 4u) & 0xFFu);
2285            ptcb->OSTCBX             = (INT8U) (prio & 0x0Fu);
2286    #endif
2287                                                                 /* Pre-compute BitX and BitY              */
2288            ptcb->OSTCBBitY          = (OS_PRIO)(1uL << ptcb->OSTCBY);
2289            ptcb->OSTCBBitX          = (OS_PRIO)(1uL << ptcb->OSTCBX);
2290
2291    #if (OS_EVENT_EN)
2292            ptcb->OSTCBEventPtr      = (OS_EVENT  *)0;        /* Task is not pending on an  event       */
2293    #if (OS_EVENT_MULTI_EN > 0u)
```



```
2324            OSTCBInitHook(ptcb);
2325
2326            OS_ENTER_CRITICAL();
2327            OSTCBPrioTbl[prio] = ptcb;
2328            OS_EXIT_CRITICAL();
2329
2330            OSTaskCreateHook(ptcb);                         /* Call user defined hook                   */
2331
2332    #if OS_TASK_CREATE_EXT_EN > 0u
2333    #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
2334            for (j = 0u; j < OS_TLS_TBL_SIZE; j++) {
2335                ptcb->OSTCBTLSTbl[j] = (OS_TLS)0;
2336            }
2337            OS_TLS_TaskCreate(ptcb);                        /* Call TLS hook                            */
2338    #endif
2339    #endif
2340
2341    #ifdef M11102155_PA1_PART_2_RM
2342
2343            INT16U num_times_job;               // Records the number of times (assume j) this task occurs periodically.
2344            INT16U num_recent_execute_time;     // Records the time (in ticks) that the task has executed at time j.
2345            INT16U total_execute_time;          // Records the worst-case execution time of the task.
2346            INT16U arrive_time;                 // Records the arrival time of the task at time j.
2347            INT16U deadline_time;               // Records the deadline of the task at time j.
2348
2349
2350            if (p_arg != 0) // (void*) p_arg == 0 : Idle task.
2351            {
2352                ptcb->num_times_job = 0u;
2353                ptcb->num_recent_execute_time = 0u;
2354                ptcb->total_execute_time = task_parameter->TaskExecutionTime;
2355                ptcb->arrive_time = task_parameter->TaskArriveTime;
2356                ptcb->period = task_parameter->TaskPeriodic;
2357                ptcb->deadline_time = (ptcb->arrive_time) + (ptcb->period);
2358            }
2359    #endif /* M11102155_PA1_PART_2_RM */
2360
2361
2362            OS_ENTER_CRITICAL();
2363
2364
2365    #ifdef M11102155_PA1_PART_1 Inactive Preprocessor Block
2376    #endif /* M11102155_PA1_PART_1 */
2377
2378            ptcb->OSTCBNext = OSTCBList;                    /* Link into TCB chain                      */
2379            ptcb->OSTCBPrev = (OS_TCB *)0;
2380            if (OSTCBList != (OS_TCB *)0) {
2381                OSTCBList->OSTCBPrev = ptcb;
2382            }
2383            OSTCBList               = ptcb;
```

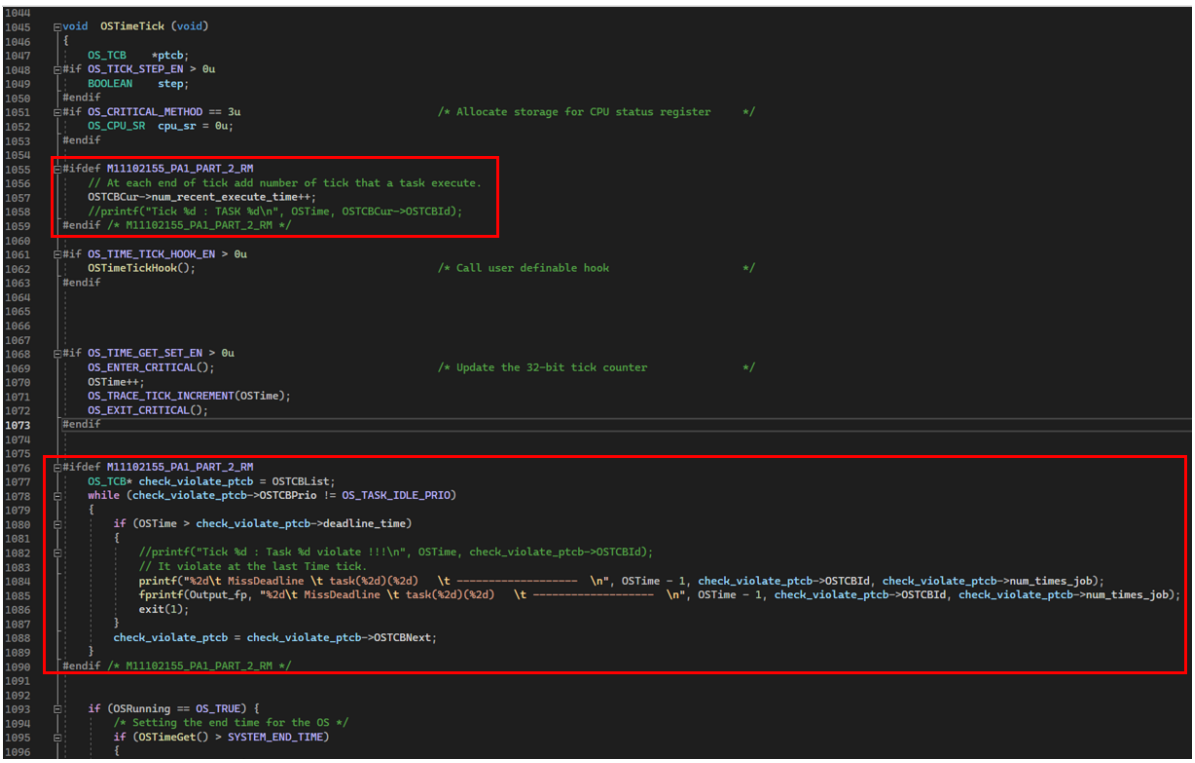
```
2384
2385 #ifndef M11102155_PA1_PART_2_RM
2386
2387         OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
2388         OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2389
2390 #else
2391         if (p_arg != 0)
2392         {
2393             if (task_parameter->TaskArriveTime == 0)
2394             {
2395                 OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
2396                 OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2397             }
2398         }
2399         else
2400         {
2401             OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
2402             OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2403         }
2404
2405
2406
2407 #endif /* M11102155_PA1_PART_2_RM */
2408
2409
2410
2411         OSTaskCtr++;                                    /* Increment the #tasks counter          */
2412
2413
2414 #ifdef M11102155_PA1_PART_1 Inactive Preprocessor Block
2418 #endif /* M11102155_PA1_PART_1 */
2419
2420
2421
2422
2423         OS_TRACE_TASK_READY(ptcb);
2424         OS_EXIT_CRITICAL();
2425         return (OS_ERR_NONE);
2426     }
2427     OS_EXIT_CRITICAL();
2428     return (OS_ERR_TASK_NO_MORE_TCB);
2429 }
2430
```

2. In “ OS_Sched() ”, the job is complete and call OSTimeDly() then find the next high priority task the execute and print the informaiton onto CRT. After print the informationm, write the information into Output file then the number of task’s job will increase by one. Eventually, do the task level context switch and keep executing the high priority task.



```
// Task level scheduling ::
void  OS_Sched (void)
{
#if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
    OS_CPU_SR  cpu_sr = 0u;
#endif

    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0u) {                         /* Schedule only if all ISRs done and ...       */
        if (OSLockNesting == 0u) {                    /* ... scheduler is not locked                  */
            OS_SchedNew();
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
            if (OSPrioHighRdy != OSPrioCur) {         /* No Ctx Sw if current task is highest rdy     */
#ifndef M11102155_HW1
#if OS_TASK_PROFILE_EN > 0u
                OSTCBHighRdy->OSTCBCtxSwCtr++;        /* Inc. # of context switches to this task      */
#endif

                OSCtxSwCtr++;                         /* Increment context switch counter             */
#endif /* M11102155_HW1 */

#if OS_TASK_CREATE_EXT_EN > 0u
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
                OS_TLS_TaskSw();
#endif
#endif
#ifdef M11102155_HW1 Inactive Preprocessor Block
#endif /* M11102155_HW1 */

#ifdef M11102155_PA1_PART_2_RM
                if (OSTCBHighRdy->OSTCBPrio == 63)
                {
                    printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur -> response_time,
                        (OSTCBCur -> response_time - OSTCBCur -> total_execute_time), OSTCBCur ->OSTCBDly);
                    fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time
                        (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                }
                else
                {
                    printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId,
                        OSTCBHighRdy->num_times_job, OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                    fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId,
                        OSTCBHighRdy->num_times_job, OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                }
                OSTCBCur->num_times_job++;
#endif /* M11102155_PA1_PART_2_RM */

                OS_TASK_SW();                         /* Perform a context switch                     */
            }
        }
    }
    OS_EXIT_CRITICAL();
}
```

2. In “ OSTimeTick() “, first increase the “ OSTCBCur→ num_recent_execute_time ”, means the count of task has executed in current job, by one, and consider whether exist task’s job violate(overflow) in the next OSTime(so we must do OSTime++ before the test). If there exist a job that violate, print the informaiton of this job has violate at previous time tick onto CRT and into Output file then the whole system will stop.



```
1044
1045  void  OSTimeTick (void)
1046  {
1047      OS_TCB     *ptcb;
1048  #if OS_TICK_STEP_EN > 0u
1049      BOOLEAN    step;
1050  #endif
1051  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
1052      OS_CPU_SR  cpu_sr = 0u;
1053  #endif
1054
1055  #ifdef M11102155_PA1_PART_2_RM
1056      // At each end of tick add number of tick that a task execute.
1057      OSTCBCur->num_recent_execute_time++;
1058      //printf("Tick %d : TASK %d\n", OSTime, OSTCBCur->OSTCBId);
1059  #endif /* M11102155_PA1_PART_2_RM */
1060
1061  #if OS_TIME_TICK_HOOK_EN > 0u
1062      OSTimeTickHook();                                 /* Call user definable hook                   */
1063  #endif
1064
1065
1066
1067
1068  #if OS_TIME_GET_SET_EN > 0u
1069      OS_ENTER_CRITICAL();                              /* Update the 32-bit tick counter             */
1070      OSTime++;
1071      OS_TRACE_TICK_INCREMENT(OSTime);
1072      OS_EXIT_CRITICAL();
1073  #endif
1074
1075
1076  #ifdef M11102155_PA1_PART_2_RM
1077      OS_TCB* check_violate_ptcb = OSTCBList;
1078      while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
1079      {
1080          if (OSTime > check_violate_ptcb->deadline_time)
1081          {
1082              //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
1083              // It violate at the last Time tick.
1084              printf("%2d\t MissDeadline \t task(%2d)(%2d)   \t --------------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1085              fprintf(Output_fp, "%2d\t MissDeadline \t task(%2d)(%2d)   \t --------------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1086              exit(1);
1087          }
1088          check_violate_ptcb = check_violate_ptcb->OSTCBNext;
1089      }
1090  #endif /* M11102155_PA1_PART_2_RM */
1091
1092
1093      if (OSRunning == OS_TRUE) {
1094          /* Setting the end time for the OS */
1095          if (OSTimeGet() > SYSTEM_END_TIME)
1096          {
```

If there doesn’t exist a job violate, then will consider whether the exist jobs want to become ready(end delay) from all waiting jobs, after setting the job that finish waiting ready it will also reset it’s arrive time, number of time job has executed and the deadline.

```
#endif
            ptcb = OSTCBList;                                    /* Point at first TCB in TCB list               */
            while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {       /* Go through all TCBs in TCB list              */
                OS_ENTER_CRITICAL();
                if (ptcb->OSTCBDly != 0u) {                      /* No, Delayed or waiting for event with TO     */
                    ptcb->OSTCBDly--;                            /* Decrement nbr of ticks to end of delay       */
                    if (ptcb->OSTCBDly == 0u) {                  /* Check for timeout                            */

                        if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
                            ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
                            ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
                        } else {
                            ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
                        }

                        if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
                            OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
                            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

#ifdef M11102155_PA1_PART_2_RM
                            OS_ENTER_CRITICAL();
                            ptcb->arrive_time = OSTime;
                            ptcb->num_recent_execute_time = 0;
                            ptcb->deadline_time = ptcb->arrive_time + ptcb->period;
                            OS_EXIT_CRITICAL();
#endif /* M11102155_PA1_PART_2_RM */

                            OS_TRACE_TASK_READY(ptcb);
                        }
                    }
                }
                ptcb = ptcb->OSTCBNext;                          /* Point at next TCB in TCB list                */
                OS_EXIT_CRITICAL();
            }
        }
    }
```

After performing OSTimeTick(), we will do OSIntExit() to find whether exist higher priority task before leave interrupt. If exist then execute the task with higher priority, else back to remain task.

In “ OSIntExit() “, apart from performing the preemptive case, we also consider overcomming the special case we mentioned at the “ task() ” in “ main.c ” part. If the job is complete before preempt, it will print the job is complete then delay the job, else print the job is preempt by the job with higher priority. After all perform OSIntCtxsw(), switch the higher priority to execute.

```
void  OSIntExit (void)
{
#if OS_CRITICAL_METHOD == 3u                         /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr = 0u;
#endif

    if (OSRunning == OS_TRUE) {
        OS_ENTER_CRITICAL();
        if (OSIntNesting > 0u) {                     /* Prevent OSIntNesting from wrapping       */
            OSIntNesting--;
        }
        if (OSIntNesting == 0u) {                    /* Reschedule only if all ISRs complete ... */
            if (OSLockNesting == 0u) {               /* ... and not locked.                      */
                OS_SchedNew();
                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
                if (OSPrioHighRdy != OSPrioCur) {    /* No Ctx Sw if current task is highest rdy */

#ifndef M11102155_HW1 Active Preprocessor Block
 #endif /* M11102155_HW1 */

#if OS_TASK_CREATE_EXT_EN > 0u
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
                    OS_TLS_TaskSw();
 #endif
 #endif

                    OS_TRACE_ISR_EXIT_TO_SCHEDULER();

#ifdef M11102155_HW1
                    // Cause the only interrupt here is time up such that idle task is interrupt by timetick
                    printf("%2d\t task(%2d)      \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, 63, OSTCBHighRdy -> OSTCBId, OSTCBHighRdy -> OSTCBCtxSwCtr, OSCtxSwCtr);
                    fprintf(Output_fp, "%2d\t task(%2d)      \t\t task(%2d)(%2d)\t\t %2d\n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
                    OSCtxSwCtr++;
#endif /* M11102155_HW1 */
```

```
#ifdef M11102155_PA1_PART_2_RM
                    if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
                    {
                        // Compute response time.
                        OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
                        if (OSTCBHighRdy->OSTCBPrio == 63)
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)        \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)        \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                        }
                        else
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job,
                                OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->deadline_time - OSTime);
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job,
                                OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->deadline_time - OSTime);
                        }
                        OSTCBCur->num_times_job++;

                        // Do delay !!!
                        INT8U      y;
                        INT32U ticks = OSTCBCur->deadline_time - OSTime;
                        if (ticks > 0u) {                       /* 0 means no delay!                                  */
                            OS_ENTER_CRITICAL();
                            y = OSTCBCur->OSTCBY;               /* Delay current task                                 */
                            OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
                            OS_TRACE_TASK_SUSPENDED(OSTCBCur);
                            if (OSRdyTbl[y] == 0u) {
                                OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
                            }
                            OSTCBCur->OSTCBDly = ticks;         /* Load ticks in TCB                                  */
                            OS_TRACE_TASK_DLY(ticks);
                            OS_EXIT_CRITICAL();
                        }
                        //printf("     task %d ---- ISR level delay : %2d\n", OSTCBCur->OSTCBId, ticks);
                    }
                    else
                    {
                        if (OSTCBHighRdy->OSTCBPrio == 63)
                        {
                            printf("%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
                            fprintf(Output_fp, "%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
                        }
                        else
                        {
                            if (OSTCBCur->OSTCBPrio == 63)
                            {
                                printf("%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                                fprintf(Output_fp, "%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                            }
                            else
                            {
                                printf("%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)(%2d)  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                                fprintf(Output_fp, "%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)(%2d)  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                            }
                        }
                    }
#endif /* M11102155_PA1_PART_2_RM */
```

```
                    OSIntCtxSw();                       /* Perform interrupt level ctx switch      */
                } else {
                    OS_TRACE_ISR_EXIT();
                }
            } else {
                OS_TRACE_ISR_EXIT();
            }
        } else {
            OS_TRACE_ISR_EXIT();
        }

        OS_EXIT_CRITICAL();
    }
}
```

Deadling missing situation handling under RM :

In my solution, i traverse the TCB List except Idle task at every Tick ISR and consider whether exist a task's job that the next Time tick is greater than it's deadline time recorded in TCB or not. If yes, means a task's job is violate, else all task's job is safe from there deadline recently.

I add the checking part in " OSTimeTick() " :

```
1045  void  OSTimeTick (void)
1046  {
1047      OS_TCB     *ptcb;
1048  #if OS_TICK_STEP_EN > 0u
1049      BOOLEAN     step;
1050  #endif
1051  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
1052      OS_CPU_SR  cpu_sr = 0u;
1053  #endif
1054
1055  #ifdef M11102155_PA1_PART_2_RM
1056      // At each end of tick add number of tick that a task execute.
1057      OSTCBCur->num_recent_execute_time++;
1058      //printf("Tick %d : TASK %d\n", OSTime, OSTCBCur->OSTCBId);
1059  #endif /* M11102155_PA1_PART_2_RM */
1060
1061  #if OS_TIME_TICK_HOOK_EN > 0u
1062      OSTimeTickHook();                                 /* Call user definable hook                    */
1063  #endif
1064
1065
1066
1067
1068  #if OS_TIME_GET_SET_EN > 0u
1069      OS_ENTER_CRITICAL();                              /* Update the 32-bit tick counter              */
1070      OSTime++;
1071      OS_TRACE_TICK_INCREMENT(OSTime);
1072      OS_EXIT_CRITICAL();
1073  #endif
1074
1075
1076  #ifdef M11102155_PA1_PART_2_RM
1077      OS_TCB* check_violate_ptcb = OSTCBList;
1078      while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
1079      {
1080          if (OSTime > check_violate_ptcb->deadline_time)
1081          {
1082              //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
1083              // It violate at the last Time tick.
1084              printf("%2d\t MissDeadline \t task(%2d)(%2d)   \t --------------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1085              fprintf(Output_fp, "%2d\t MissDeadline \t task(%2d)(%2d)   \t --------------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1086              exit(1);
1087          }
1088          check_violate_ptcb = check_violate_ptcb->OSTCBNext;
1089      }
1090  #endif /* M11102155_PA1_PART_2_RM */
1091
1092
1093      if (OSRunning == OS_TRUE) {
1094          /* Setting the end time for the OS */
1095          if (OSTimeGet() > SYSTEM_END_TIME)
```

Each job’s deadline will be initial using “ arrive time + period ” when Job is creating (inside TCB initial function, OS_TCBInit()) and update when the task arrive again (finish delay and become ready, inside OSTimeTick()), both modify function are reside in os_core.c.

deadline initial in “ OS_TCBInit() ” :

```
2338  #ifdef M11102155_PA1_PART_2_RM
2339
2340          INT16U num_times_job;                 // Records the number of times (assume j) this task occurs periodically.
2341          INT16U num_recent_execute_time;       // Records the time (in ticks) that the task has executed at time j.
2342          INT16U total_execute_time;            // Records the worst-case execution time of the task.
2343          INT16U arrive_time;                   // Records the arrival time of the task at time j.
2344          INT16U deadline_time;                 // Records the deadline of the task at time j.
2345
2346
2347          if (p_arg != 0) // (void*) p_arg == 0 : Idle task.
2348          {
2349              ptcb->num_times_job = 0u;
2350              ptcb->num_recent_execute_time = 0u;
2351              ptcb->total_execute_time = task_parameter->TaskExecutionTime;
2352              ptcb->arrive_time = task_parameter->TaskArriveTime;
2353              ptcb->period = task_parameter->TaskPeriodic;
2354              ptcb->deadline_time = (ptcb->arrive_time) + (ptcb->period);
2355          }
2356  #endif /* M11102155_PA1_PART_2_RM */
```

deadline update in “ OSTimeTick() ” :

```
1128            }
1129        #endif
1130            ptcb = OSTCBList;                                      /* Point at first TCB in TCB list               */
1131            while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {         /* Go through all TCBs in TCB list              */
1132                OS_ENTER_CRITICAL();
1133                if (ptcb->OSTCBDly != 0u) {                        /* No, Delayed or waiting for event with TO     */
1134                    ptcb->OSTCBDly--;                              /* Decrement nbr of ticks to end of delay       */
1135                    if (ptcb->OSTCBDly == 0u) {                    /* Check for timeout                            */
1136
1137                        if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
1138                            ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
1139                            ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
1140                        } else {
1141                            ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
1142                        }
1143
1144                        if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
1145                            OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
1146                            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
1147
1148        #ifdef M11102155_PA1_PART_2_RM
1149                            OS_ENTER_CRITICAL();
1150                            ptcb->arrive_time = OSTime;
1151                            ptcb->num_recent_execute_time = 0;
1152                            ptcb->deadline_time = ptcb->arrive_time + ptcb->period;
1153                            OS_EXIT_CRITICAL();
1154        #endif /* M11102155_PA1_PART_2_RM */
1155
1156                            OS_TRACE_TASK_READY(ptcb);
1157                        }
1158                    }
1159                }
1160                ptcb = ptcb->OSTCBNext;                            /* Point at next TCB in TCB list                */
1161                OS_EXIT_CRITICAL();
1162            }
1163        }
1164    }
1165
```

[Part 3] FIFO Scheduler Implementation

Comparing RM with FIFO :

RM is preemptive and FIFO is non-preemptive, and the priority of the job is different, cause RM is static priority, the job of each task won't change when executing. On the other hand, FIFO is dynamic priority, the priority of each job is according to the arrive time of each job, as the job arrive firstly, it's priority is higher than other jobs that lately arrive.

In inplementation i consider not using changing the prioirty in FIFO, I just create a circular FIFO queue and insert the jobs into queue when they arrive. The scheduling is to dequeue a job from circular FIFO queue.

Modify Code Review :

The code I modify will wrapped between

```
#ifdef M11102155_PA1_PART_3_FIFO

    # modify code !!!

#endif /* M11102155_PA1_PART_3_FIFO*/
```

The code I continue to use from Part 2

```
#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA1_PART_3_FIFO)

    # same as Part 2 !!!

#endif /* M11102155_PA1_PART_2_RM | M11102155_PA1_PART_3_FIFO*/
```

ucos_ii.h :

For using #ifdef M11102155_PA1_PART_3_FIFO #endif in all code segament.

```
34
35  //#define M11102155_HW1
36  //#define M11102155_PA1_PART_1
37  //#define M11102155_PA1_PART_2_RM
38  #define M11102155_PA1_PART_3_FIFO
39
40  #ifdef __cplusplus
41  extern "C" {
42  #endif
```

In “ ucos_ii.h ”, I create a circular queue for storing the FIFO waiting list.

```
122
123 #ifdef M11102155_PA1_PART_3_FIFO
124
125 typedef struct FIFO_Q_INFO
126 {
127     INT16U front;
128     INT16U end;
129     INT16U num_item;
130     INT16U size;
131
132 }FIFO_Q_INFO;
133
134 FIFO_Q_INFO* fifo_q_info;
135 INT16U* fifo_queue;
136
137 #endif /* M11102155_PA1_PART_3_FIFO */
```

app_hooks.c :

In “ app_hooks.c ”, I write a funciton for creating the FIFO queue.

```
176
177
178
179  #ifdef M11102155_PA1_PART_3_FIFO
180
181  void FIFOQInit()
182  {
183      fifo_queue = (INT16U*)malloc((TASK_NUMBER + 1) * sizeof(INT16U));
184      fifo_q_info = (FIFO_Q_INFO*)malloc(1 * sizeof(FIFO_Q_INFO));
185
186      if (fifo_queue == NULL | fifo_q_info == NULL)
187      {
188          printf("FIFO QUEUE malloc failed !!! \n");
189      }
190      else
191      {
192          printf("FIFO Queue malloc sucess !!!\n");
193      }
194      fifo_q_info->front = 0;
195      fifo_q_info->end = 0;
196      fifo_q_info->num_item = 0;
197      fifo_q_info->size = (TASK_NUMBER + 1);
198
199      printf("END OF FIFO QUEUE INIT !!!\n");
200
201  }
202
203  #endif /* M11102155_PA1_PART_3_FIFO */
204
205
206
207
```

main.c :

In “ main.c ”, First create the circular FIFO queue, then create the tasks, here i use the task id as its priority, and store the task id into the circular FIFO queue, when i dequeue then i can get the task’s priority and find the TCB from using TCBPrioTbl[].

```
96   int  main (void)
97   {
98   #if OS_TASK_NAME_EN > 0u
99       CPU_INT08U  os_err;
100  #endif
101
102
103      CPU_IntInit();
104
105      Mem_Init();                                          /* Initialize Memory Managment Module                */
106      CPU_IntDis();                                        /* Disable all Interrupts                            */
107      CPU_Init();                                          /* Initialize the uC/CPU services                    */
108
109      OSInit();                                            /* Initialize uC/OS-II                               */
110
111
112
113      /* Initialize Output File */
114      OutFileInit();
115
116      /* Input File */
117      InputFile();
118
119
120  #ifdef M11102155_PA1_PART_3_FIFO
121      // Init FIFO QUEUE.
122      FIFOQInit();
123  #endif /* M11102155_PA1_PART_3_FIFO */
124
125
126      /* Dynamic Create the Stack size */
127      Task_STK = malloc(TASK_NUMBER * sizeof(int*));
128
129      /* for each pointer, allocate storage for an array of int */
130      int n;
131      for (n = 0; n < TASK_NUMBER; n++)
132      {
133          Task_STK[n] = malloc(TASK_STAKSIZE * sizeof(int));
134      }
```

```
180
181
182      // Set task ID as priority.
183  #ifdef M11102155_PA1_PART_3_FIFO
184
185      for (n = 0; n < TASK_NUMBER; n++)
186      {
187          OSTaskCreateExt(task,
188              &TaskParameter[n],
189              &Task_STK[n][TASK_STAKSIZE - 1],
190              TaskParameter[n].TaskID,
191              TaskParameter[n].TaskID,
192              &Task_STK[n][0],
193              TASK_STAKSIZE,
194              &TaskParameter[0],
195              (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
196      }
197
198  #endif /* M11102155_PA1_PART_3_FIFO */
```

os_time.c :

In “ os_time.c ”, we cancel the operation of setting the ready table.

```
54
55  void  OSTimeDly (INT32U ticks)
56  {
57      INT8U      y;
58  #if OS_CRITICAL_METHOD == 3u                    /* Allocate storage for CPU status register           */
59      OS_CPU_SR  cpu_sr = 0u;
60  #endif
61
62  #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA1_PART_3_FIFO)
63      // Compute the delay time before task arrive again.
64      OS_ENTER_CRITICAL();
65      ticks = OSTCBCur->deadline_time - OSTime;
66      OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
67      OS_EXIT_CRITICAL();
68  #endif /* M11102155_PA1_PART_2_RM | M11102155_PA1_PART_3_FIFO */
69
70
71
72      if (OSIntNesting > 0u) {                    /* See if trying to call from an ISR                  */
73          return;
74      }
75      if (OSLockNesting > 0u) {                   /* See if called with scheduler locked                */
76          return;
77      }
78      if (ticks > 0u) {                           /* 0 means no delay!                                  */
79          OS_ENTER_CRITICAL();
80
81  #ifndef M11102155_PA1_PART_3_FIFO
82          y             =  OSTCBCur->OSTCBY;      /* Delay current task                                 */
83          OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
84          //OS_TRACE_TASK_SUSPENDED(OSTCBCur);
85          if (OSRdyTbl[y] == 0u) {
86              OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
87          }
88  #endif /* M11102155_PA1_PART_3_FIFO */
89
90          OS_TRACE_TASK_SUSPENDED(OSTCBCur);
91          OSTCBCur->OSTCBDly = ticks;             /* Load ticks in TCB                                  */
92          OS_TRACE_TASK_DLY(ticks);
93          OS_EXIT_CRITICAL();
94          OS_Sched();                             /* Find next task to run!                             */
95      }
96  }
97
```

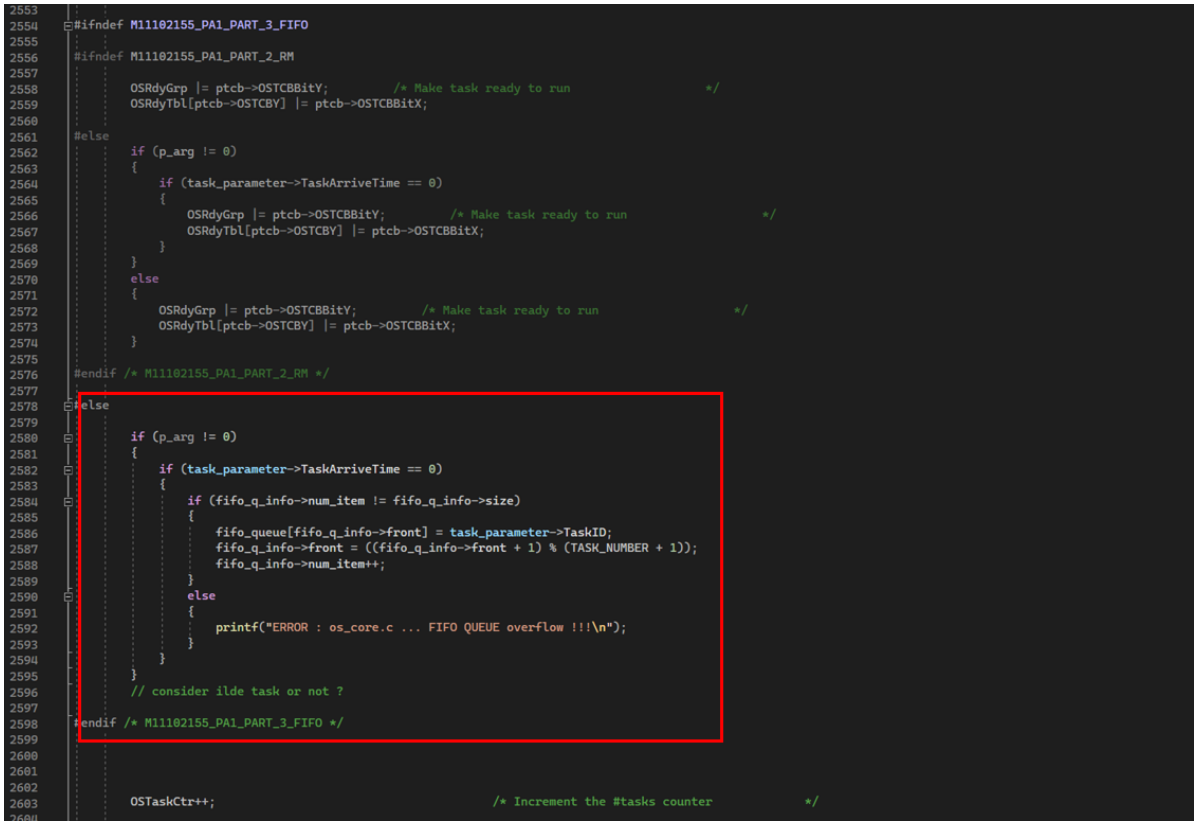
os_core.c :

In os_core.c i divided into 3 parts :

1. TCB initialization when creating the tasks and selecting the first job
2. Task's job is complete and call the OSTimeDly() then enter the OS_Sched() for finding the next high priority task to execute.
3. For all Tick Interrupt occur in every end of the time tick, it will consider the task's job is complete or preemt by other high priority task, and also check whether exist job violate (overflow).

Start from here :

1. TCB initialization in " OSTCBInit() ", I just insert the task into circuler FIFO queue when it's arrive time equals to zero and cancel set the ready table, cause FIFO won't using it.



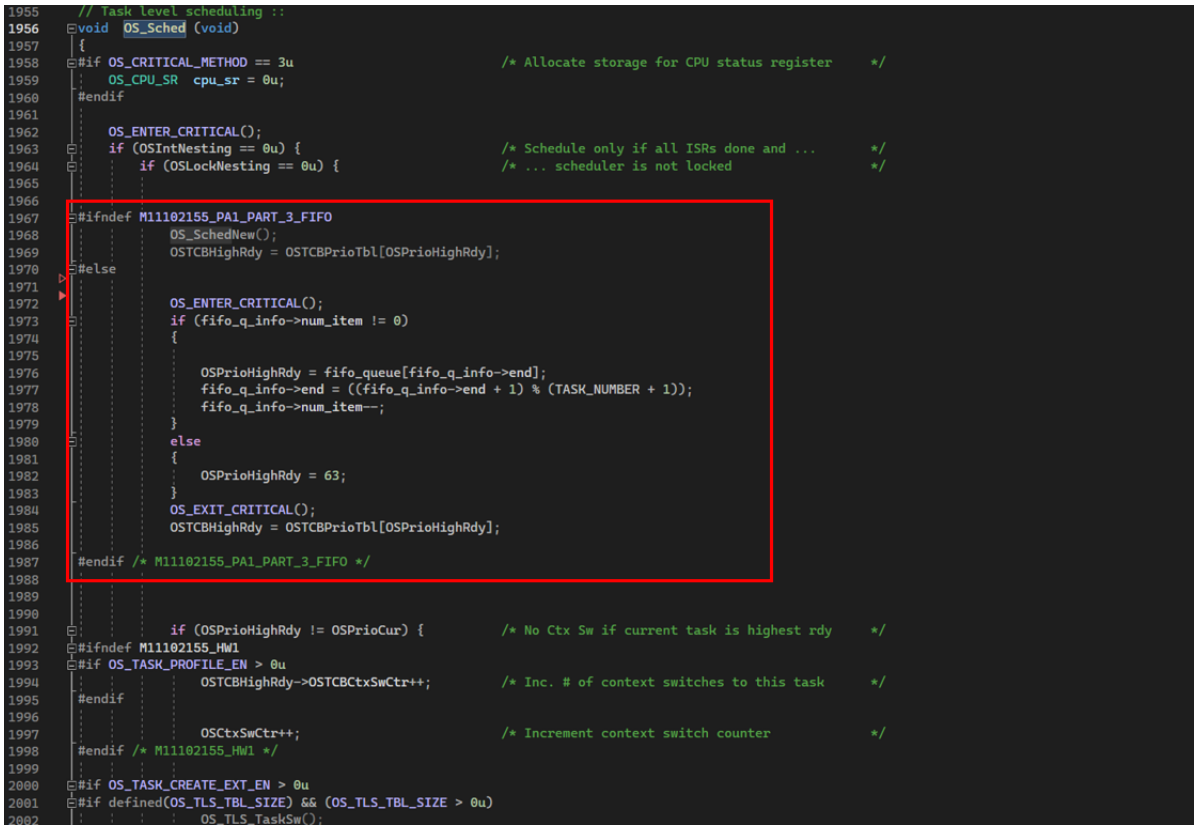
```
#ifndef M11102155_PA1_PART_3_FIFO

#ifndef M11102155_PA1_PART_2_RM

        OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

#else
        if (p_arg != 0)
        {
            if (task_parameter->TaskArriveTime == 0)
            {
                OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
                OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
            }
        }
        else
        {
            OSRdyGrp |= ptcb->OSTCBBitY;         /* Make task ready to run                */
            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        }

#endif /* M11102155_PA1_PART_2_RM */

#else

        if (p_arg != 0)
        {
            if (task_parameter->TaskArriveTime == 0)
            {
                if (fifo_q_info->num_item != fifo_q_info->size)
                {
                    fifo_queue[fifo_q_info->front] = task_parameter->TaskID;
                    fifo_q_info->front = ((fifo_q_info->front + 1) % (TASK_NUMBER + 1));
                    fifo_q_info->num_item++;
                }
                else
                {
                    printf("ERROR : os_core.c ... FIFO QUEUE overflow !!!\n");
                }
            }
        }
        // consider ilde task or not ?

#endif /* M11102155_PA1_PART_3_FIFO */



        OSTaskCtr++;                                /* Increment the #tasks counter          */
```

For selecting the first job in “ OSStart() ”, just dequeue from the circular FIFO queue, if there is no item in the queue then run Idle task, otherwise get the job that earlest insert into the queue.

```
1011
1012  void  OSStart (void)
1013  {
1014      if (OSRunning == OS_FALSE) {
1015  #ifndef M11102155_PA1_PART_3_FIFO
1016          OS_SchedNew();                               /* Find highest priority's task priority number   */
1017          OSPrioCur     = OSPrioHighRdy;
1018          OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
1019  #ifdef M11102155_HW1
1020          // Open the Output file.
1021          if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) != 0)
1022          {
1023              printf("Error :: Output File Open Error !!! ");
1024              exit(0);
1025          }
1026          printf("%2d\t **********   \t\t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBHighRdy-> OSTCBId, OSTCBHighRdy-> OSTCBCtxSwCtr, OSCtxSwCtr);
1027          fprintf(Output_fp, "%2d\t **********   \t\t\t task(%2d)(%2d)\t\t %2d\n", OSTime, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr, OSCtxSwCtr);
1028  #endif /* M11102155_HW1 */
1029  #ifdef M11102155_PA1_PART_1
1030          OS_TCB* ptcb = OSTCBList;
1031          printf("===============TCB linked list===============\n");
1032          printf("TASK    Prev_TCB_addr   TCB_addr   Next_TCB_addr\n");
1033          while (ptcb != (OS_TCB*)0)
1034          {
1035              if (ptcb->OSTCBPrio == 63)
1036              {
1037                  printf("%2d          %6x        %6x        %6x\n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
1038              }
1039              else
1040              {
1041                  printf("%2d          %6x        %6x        %6x\n", ptcb->OSTCBId, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
1042              }
1043              ptcb = ptcb->OSTCBNext;
1044          }
1045  #endif /* M11102155_PA1_PART_1 */
1046          OSTCBCur      = OSTCBHighRdy;
1047          OSStartHighRdy();                            /* Execute target specific code to start task     */       // set OSRunning = 1u
1048  #else
1049
1050          OS_ENTER_CRITICAL();
1051          if (fifo_q_info->num_item != 0)
1052          {
1053              OSPrioHighRdy = fifo_queue[fifo_q_info->end];
1054              fifo_q_info->end = ((fifo_q_info->end + 1) % (TASK_NUMBER + 1));
1055              fifo_q_info->num_item--;
1056          }
1057          else
1058          {
1059              OSPrioHighRdy = 63;
1060          }
1061          OS_EXIT_CRITICAL();
1062
1063          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run    */
1064          OSTCBCur = OSTCBHighRdy;
1065          OSStartHighRdy();
1066
1067  #endif /* M11102155_PA1_PART_3_FIFO 1*/
1068
1069      }
1070  }
```

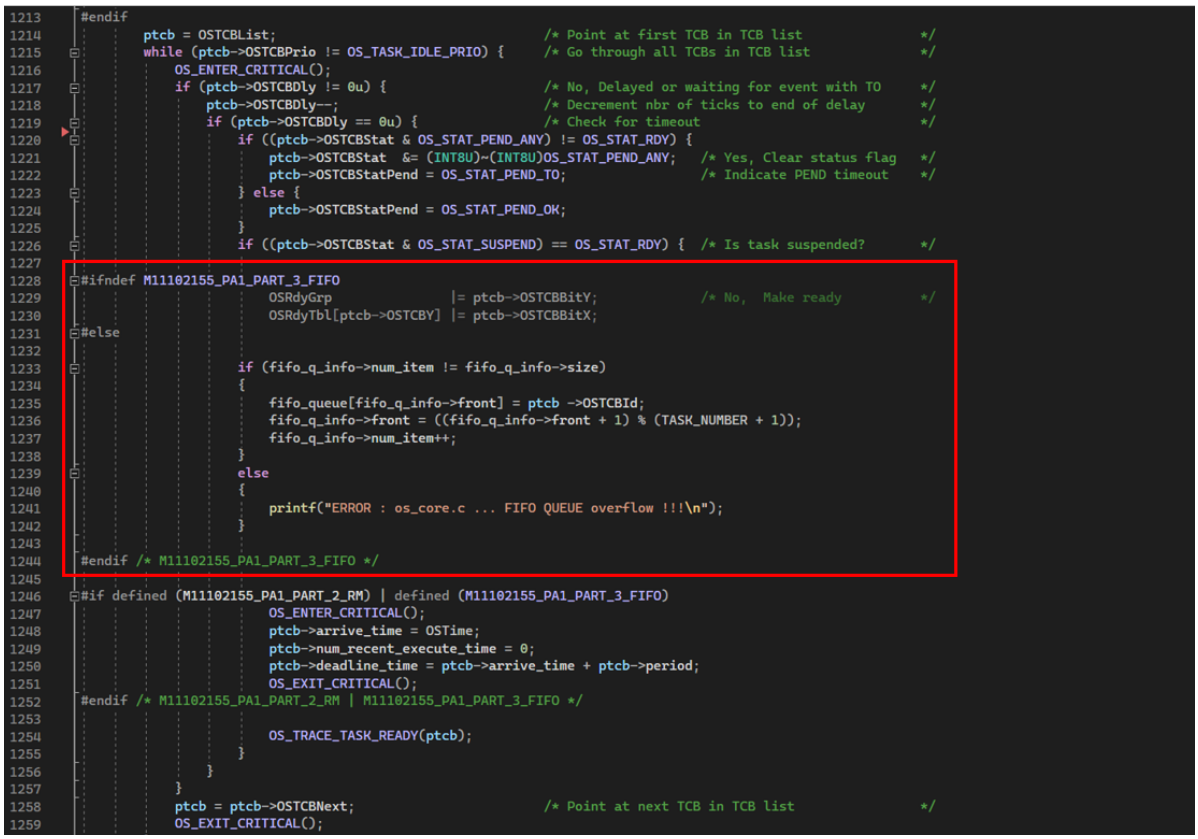
*** For all scheduling part, cancel operation of setting the ready table, by using dequeue and insert of circular queue ***

2. In “ OS_Sched() ”, for finding the next job to execurte, just dequeue the circular FIFO queue. If there is no waiting job in queue, just switch to Idle task.



```
1955    // Task level scheduling ::
1956    void  OS_Sched (void)
1957    {
1958    #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
1959        OS_CPU_SR  cpu_sr = 0u;
1960    #endif
1961
1962        OS_ENTER_CRITICAL();
1963        if (OSIntNesting == 0u) {                         /* Schedule only if all ISRs done and ...       */
1964            if (OSLockNesting == 0u) {                    /* ... scheduler is not locked                  */
1965
1966
1967    #ifndef M11102155_PA1_PART_3_FIFO
1968                OS_SchedNew();
1969                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
1970    #else
1971
1972                OS_ENTER_CRITICAL();
1973                if (fifo_q_info->num_item != 0)
1974                {
1975
1976                    OSPrioHighRdy = fifo_queue[fifo_q_info->end];
1977                    fifo_q_info->end = ((fifo_q_info->end + 1) % (TASK_NUMBER + 1));
1978                    fifo_q_info->num_item--;
1979                }
1980                else
1981                {
1982                    OSPrioHighRdy = 63;
1983                }
1984                OS_EXIT_CRITICAL();
1985                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
1986
1987    #endif /* M11102155_PA1_PART_3_FIFO */
1988
1989
1990
1991                if (OSPrioHighRdy != OSPrioCur) {         /* No Ctx Sw if current task is highest rdy     */
1992    #ifndef M11102155_HW1
1993    #if OS_TASK_PROFILE_EN > 0u
1994                    OSTCBHighRdy->OSTCBCtxSwCtr++;        /* Inc. # of context switches to this task      */
1995    #endif
1996
1997                    OSCtxSwCtr++;                         /* Increment context switch counter             */
1998    #endif /* M11102155_HW1 */
1999
2000    #if OS_TASK_CREATE_EXT_EN > 0u
2001    #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
2002                    OS_TLS_TaskSw();
```

3. In “ OSTimeTick() “, If there find the job is getting ready (finish delay), just insert it into circular FIFO queue (FIFO waiting list) instead of doing operation on ready table.



```
#endif
        ptcb = OSTCBList;                                  /* Point at first TCB in TCB list               */
        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {     /* Go through all TCBs in TCB list              */
            OS_ENTER_CRITICAL();
            if (ptcb->OSTCBDly != 0u) {                    /* No, Delayed or waiting for event with TO     */
                ptcb->OSTCBDly--;                          /* Decrement nbr of ticks to end of delay       */
                if (ptcb->OSTCBDly == 0u) {                /* Check for timeout                            */
                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
                        ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
                    } else {
                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
                    }
                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */
#ifndef M11102155_PA1_PART_3_FIFO
                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
#else

                        if (fifo_q_info->num_item != fifo_q_info->size)
                        {
                            fifo_queue[fifo_q_info->front] = ptcb ->OSTCBId;
                            fifo_q_info->front = ((fifo_q_info->front + 1) % (TASK_NUMBER + 1));
                            fifo_q_info->num_item++;
                        }
                        else
                        {
                            printf("ERROR : os_core.c ... FIFO QUEUE overflow !!!\n");
                        }

#endif /* M11102155_PA1_PART_3_FIFO */

#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA1_PART_3_FIFO)
                        OS_ENTER_CRITICAL();
                        ptcb->arrive_time = OSTime;
                        ptcb->num_recent_execute_time = 0;
                        ptcb->deadline_time = ptcb->arrive_time + ptcb->period;
                        OS_EXIT_CRITICAL();
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA1_PART_3_FIFO */

                        OS_TRACE_TASK_READY(ptcb);
                    }
                }
            }
            ptcb = ptcb->OSTCBNext;                        /* Point at next TCB in TCB list                */
            OS_EXIT_CRITICAL();
```

In “ OSIntExit() “, we will consider whether the circular FIFO queue is empty, and the previous task is Idle task or not. If the circular FIFO queue is empty and the task is finish, we meet the issue we talk about previuosly, then the next task will be Idle task and after arrive the task will still stuck at the inner while loop of “ task() ”. If the queue isn’t empty we will consider the presious task is Idle task or not, if so, just dequeue a job from the queue.

```
694    // Interrupt (Kernel) level scheduling ::
695  void  OSIntExit (void)
696  {
697  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
698      OS_CPU_SR  cpu_sr = 0u;
699  #endif
700
701      if (OSRunning == OS_TRUE) {
702          OS_ENTER_CRITICAL();
703          if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping       */
704              OSIntNesting--;
705          }
706          if (OSIntNesting == 0u) {                     /* Reschedule only if all ISRs complete ... */
707              if (OSLockNesting == 0u) {                /* ... and not locked.                      */
708
709  #ifndef M11102155_PA1_PART_3_FIFO
710                  OS_SchedNew();
711                  OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
712  #else
713
714                  OS_ENTER_CRITICAL();
715                  if (fifo_q_info->num_item == 0)         // there is no task in the waiting queue, execute the Idle task
716                  {
717                      if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
718                      {
719                          OSPrioHighRdy = 63;
720                          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
721                      }
722                  }
723                  else
724                  {
725                      if (OSTCBCur->OSTCBId == OS_TASK_IDLE_ID)
726                      {
727                          OSPrioHighRdy = fifo_queue[fifo_q_info->end];
728                          fifo_q_info->end = ((fifo_q_info->end + 1) % (TASK_NUMBER + 1));
729                          fifo_q_info->num_item--;
730                          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
731                      }
732
733                  }
734                  OS_EXIT_CRITICAL();
735
736  #endif /* M11102155_PA1_PART_3_FIFO */
737
738                  if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy */
739
740  #ifndef M11102155_HW1 Active Preprocessor Block
745  #endif /* M11102155_HW1 */
```

```
760
761  #ifdef M11102155_PA1_PART_2_RM Inactive Preprocessor Block
820  #endif /* M11102155_PA1_PART_2_RM */
821
822
823  #ifdef M11102155_PA1_PART_3_FIFO
824                  if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
825                  {
826                      // Compute response time.
827                      OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
828
829                      // Do delay !!!
830                      INT32U ticks = OSTCBCur->deadline_time - OSTime;
831                      if (ticks > 0u) {                       /* 0 means no delay!                        */
832                          OS_ENTER_CRITICAL();
833                          OS_TRACE_TASK_SUSPENDED(OSTCBCur);
834                          OSTCBCur->OSTCBDly = ticks;         /* Load ticks in TCB                        */
835                          OS_TRACE_TASK_DLY(ticks);
836                          OS_EXIT_CRITICAL();
837                      }
838
839                      if (OSTCBHighRdy->OSTCBPrio == 63)
840                      {
841                          printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
842                              (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
843                          fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
844                              (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
845                          OSTCBCur->num_times_job++;
846                      }
847                  }
848                  else
849                  {
850                      if (OSTCBHighRdy->OSTCBPrio == 63)
851                      {
852                          printf("%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
853                          fprintf(Output_fp, "%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
854                      }
855                      else
856                      {
857                          if (OSTCBCur->OSTCBPrio == 63)
858                          {
859                              printf("%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
860                              fprintf(Output_fp, "%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
861                          }
862                      }
863                  }
864  #endif /* M11102155_PA1_PART_3_FIFO */
865
866                  OSIntCtxSw();                       /* Perform interrupt level ctx switch       */
867              } else {
868                  OS_TRACE_ISR_EXIT();
869              }
870          } else {
871              OS_TRACE_ISR_EXIT();
```