uCOS-II Report (2)

Yihe.Chen

yihect@gmail.com

内容

- Important Regs in Bf531
- Interrupt processing of Bf531
- VDSP 4.5 C Run-Time Mode for Bf531
- Porting uCOS-II to Bf531
- InterTask Communication and *Synchronization*
- Proposed Device Drivers Mode
- Q & A

Important Regs in Bf531

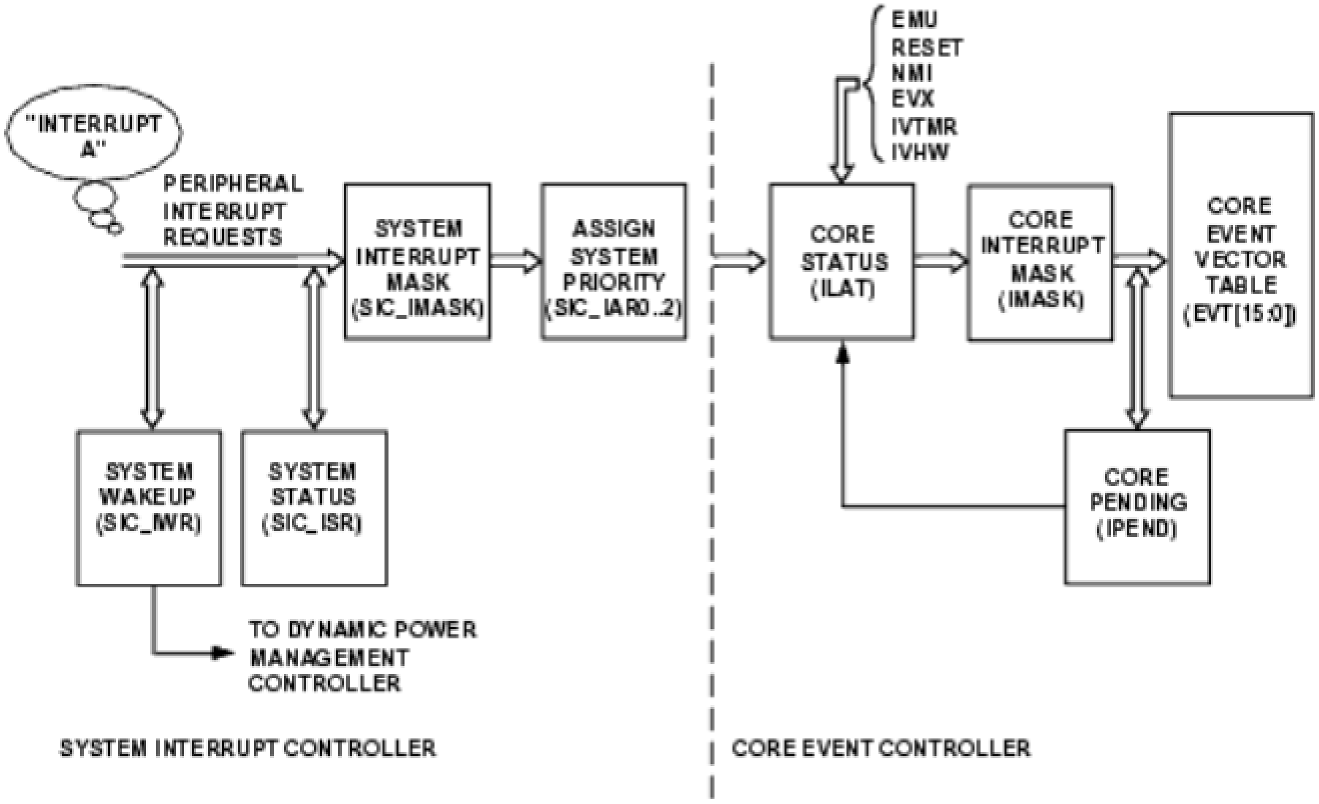
- SP & FP & PC（32位）
  - SP：指向栈顶元素。堆栈从高到低生长。
  - FP：指向最高Stack Frame 内存储有前一Stack Frame 的FP寄存器内容的栈单元
  - PC：指向下一条要执行的指令
- RETS（32位）
  - 当用CALL指令调用函数的时候，处理器自动把返回地址 保存到 RETS 中；函数执行完毕后的 RTS 指令将返回地址读到PC中

Important Regs in Bf531

- RETI（32位）
  - 当发生中断时，在执行中断处理程序之前，处理器将中断返回地址保存到 RETI 中；在中断处理程序完成时，执行 RTI 指令将中断返回地址装入到 PC 中继续执行

Interrupt processing of Bf531

- 中断处理块图



"INTERRUPT A"

PERIPHERAL INTERRUPT REQUESTS

SYSTEM INTERRUPT MASK (SIC_IMASK)

ASSIGN SYSTEM PRIORITY (SIC_IAR0..2)

SYSTEM WAKEUP (SIC_IWR)

SYSTEM STATUS (SIC_ISR)

TO DYNAMIC POWER MANAGEMENT CONTROLLER

SYSTEM INTERRUPT CONTROLLER

EMU
RESET
NMI
EVX
IVTMR
IVHW

CORE STATUS (ILAT)

CORE INTERRUPT MASK (IMASK)

CORE EVENT VECTOR TABLE (EVT[15:0])

CORE PENDING (IPEND)

CORE EVENT CONTROLLER

NOTE: NAMES IN PARENTHESES ARE MEMORY-MAPPED REGISTERS.

Interrupt processing of Bf531

- 全局中断的关闭和使能
  - CLI R0；

    --将 IMASK 保存到 R0中，并清零 IMASK 寄存器，这会关闭所有通用目的的中断（IVG5~IVG15）
  - STD R0

    --将R0 中的内容恢复到 IMASK 中，会使能原先允许的中断

Interrupt processing of Bf531

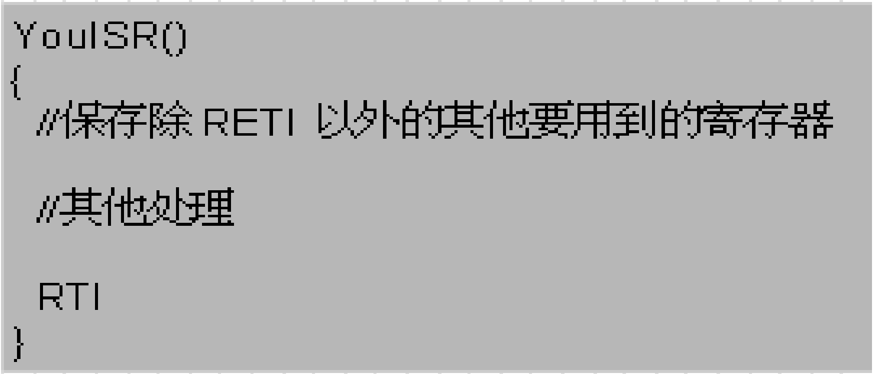
- 事件向量表（EVT）
  - 保存和各Core Event 所对应的中断向量；
  - 对于一个外部中断a来说，假如在 SIC_IAR 寄存器中将其映射到 Core Event IVG8上，则在中断发生的时候，处理器就从 EVT 中对应 IVG8 的位置处取出中断向量，并把其装入到 PC 寄存器中继续运行

Interrupt processing of Bf531

- 软中断（Software Interrupt）
  - 核心事件控制器(CEC)规定 将Core Event IVG14/IVG15 用做软中断
  - RAISE 指令用于发出软中断

Interrupt processing of Bf531

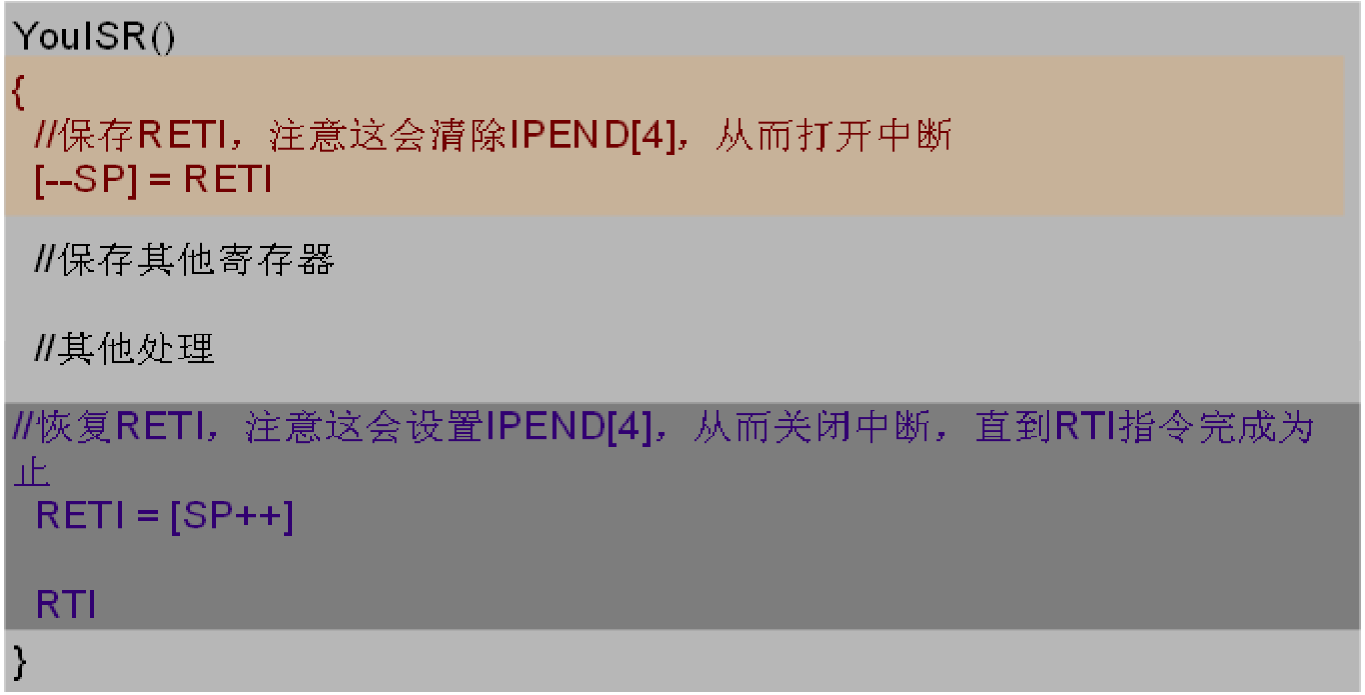
- 不可嵌套的中断
  - 不要求将RETI 保存在堆栈中，只需要保存ISR中需要用到的那些寄存器即可；返回时只需要用 RTI 指令



```
YouISR()
{
  //保存除 RETI 以外的其他要用到的寄存器

  //其他处理

  RTI
}
```

Interrupt processing of Bf531

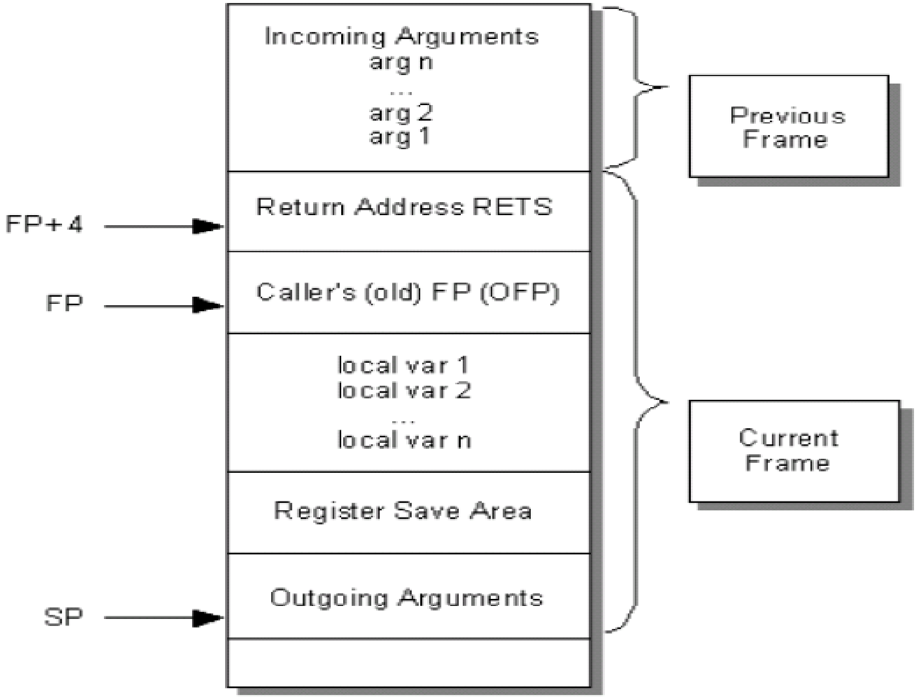
- 可嵌套的中断
  - 其 ISR 要先负责将RETI 保存到堆栈中，等到最后再从堆栈恢复RETI 寄存器



```
YouISR()
{
 //保存RETI，注意这会清除IPEND[4]，从而打开中断
 [--SP] = RETI

 //保存其他寄存器

 //其他处理

//恢复RETI，注意这会设置IPEND[4]，从而关闭中断，直到RTI指令完成为
止
 RETI = [SP++]

 RTI
}
```

VDSP 4.5 C Run-Time Mode for Bf531

- 堆栈的处理
  - Stack Frame：一小段堆栈单元，用于保存对应于当前正执行的C/C++函数的信息



VDSP 4.5 C Run-Time Mode for Bf531

- 堆栈的处理
  - 进入被调用函数时：

    a， Linking Stack Frames；

    b， Register Saving
  - 离开被调用函数时：

    a， Restore Registers；

    b， Unlinking Stack Frame

Porting uCOS-II to Bf531

- 被uCOS-II用到的处理器资源
  - Core Timer：

    a，用于给uCOS-II的运行提供时钟节拍；

    b，使用IVG6 Core Event
  - Software Trap：

    a，使用IVG14 Core Event 来进行任务级别的任务切换；

Porting uCOS-II to Bf531

- 任务上下文(Task Context)
  - 寄存器上下文

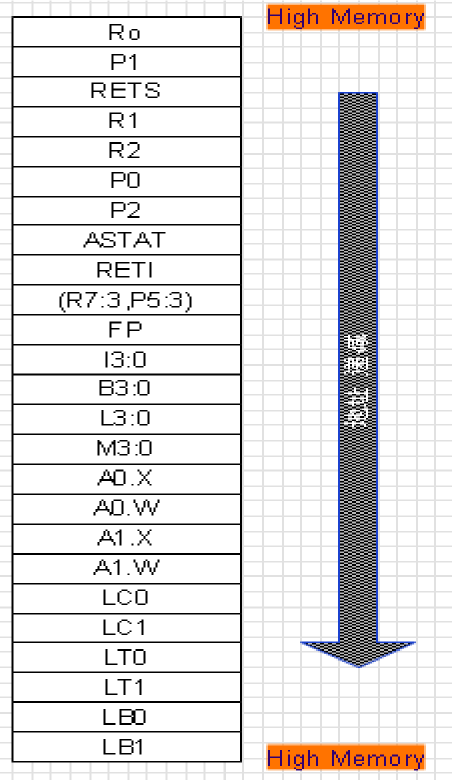
    a，当任务要放弃CPU时，需要把寄存器上下文保存到自己的堆栈中；

    b，当任务要被处理器调度执行时，需要把寄存器上下文从自己堆栈中恢复出来。

    所以，在a 和 b 步两者之间，就应该有一个协议来规定寄存器上下文在堆栈中是如何保存的

Porting uCOS-II to Bf531

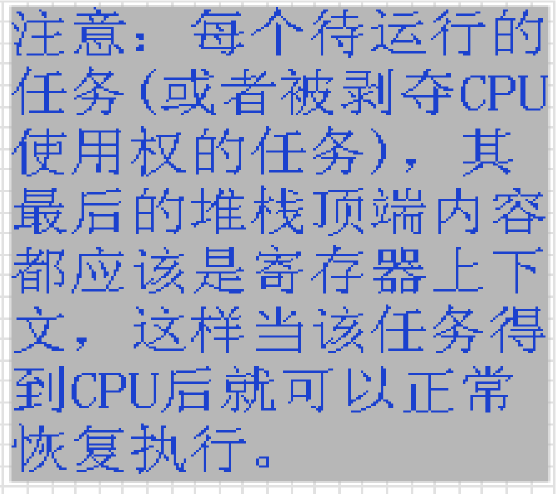
- 任务上下文(Task Context)
  - 寄存器上下文在堆栈中的保存



High Memory

Ro
P1
RETS
R1
R2
P0
P2
ASTAT
RETI
(R7:3,P5:3)
FP
I3:0
B3:0
L3:0
M3:0
A0.X
A0.W
A1.X
A1.W
LC0
LC1
LT0
LT1
LB0
LB1

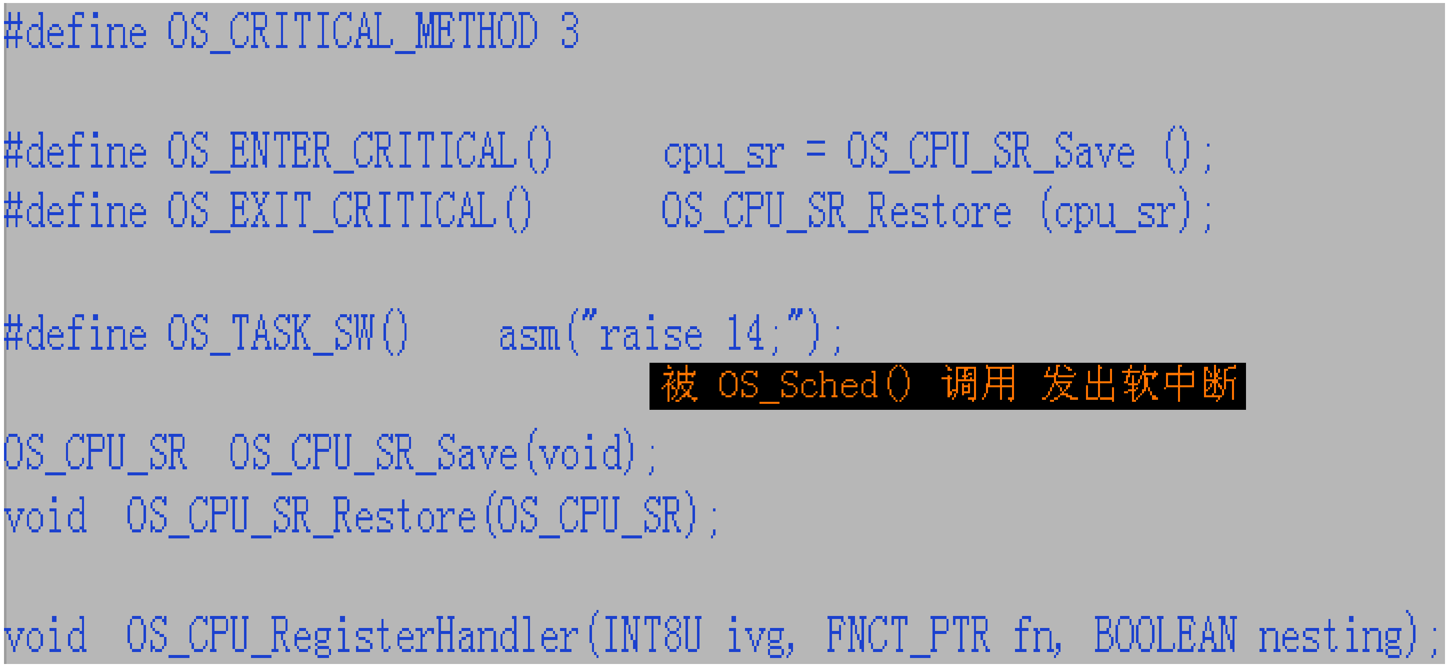
High Memory



注意：每个待运行的任务(或者被剥夺CPU使用权的任务)，其最后的堆栈顶端内容都应该是寄存器上下文，这样当该任务得到CPU后就可以正常恢复执行。

Porting uCOS-II to Bf531

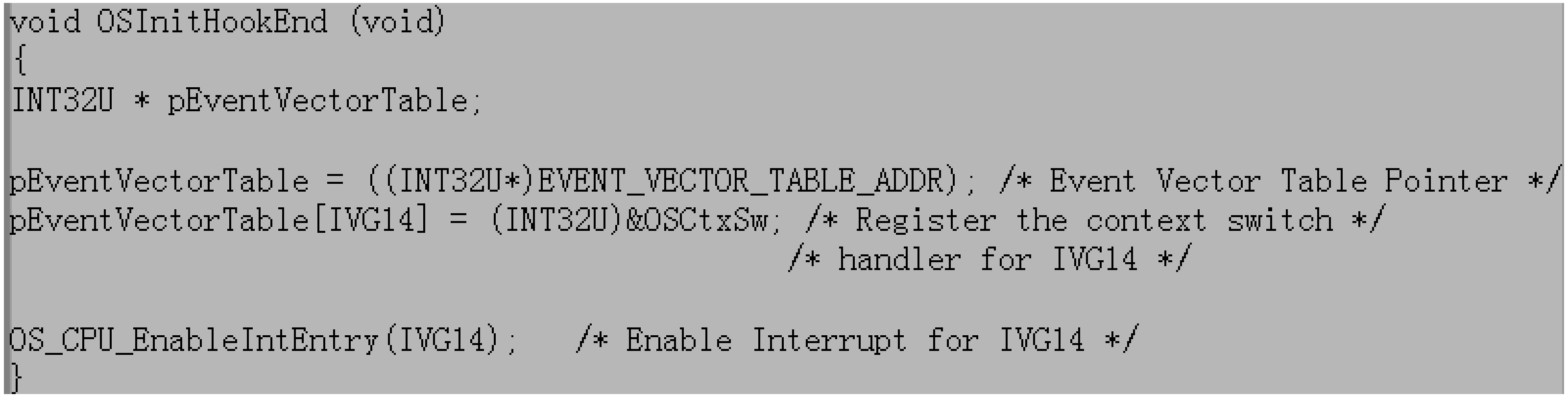
- os_cpu.h



```
#define OS_CRITICAL_METHOD 3

#define OS_ENTER_CRITICAL()      cpu_sr = OS_CPU_SR_Save ();
#define OS_EXIT_CRITICAL()       OS_CPU_SR_Restore (cpu_sr);

#define OS_TASK_SW()    asm("raise 14;");
                                  被 OS_Sched() 调用 发出软中断

OS_CPU_SR  OS_CPU_SR_Save(void);
void  OS_CPU_SR_Restore(OS_CPU_SR);

void  OS_CPU_RegisterHandler(INT8U ivg, FNCT_PTR fn, BOOLEAN nesting);
```

Porting uCOS-II to Bf531

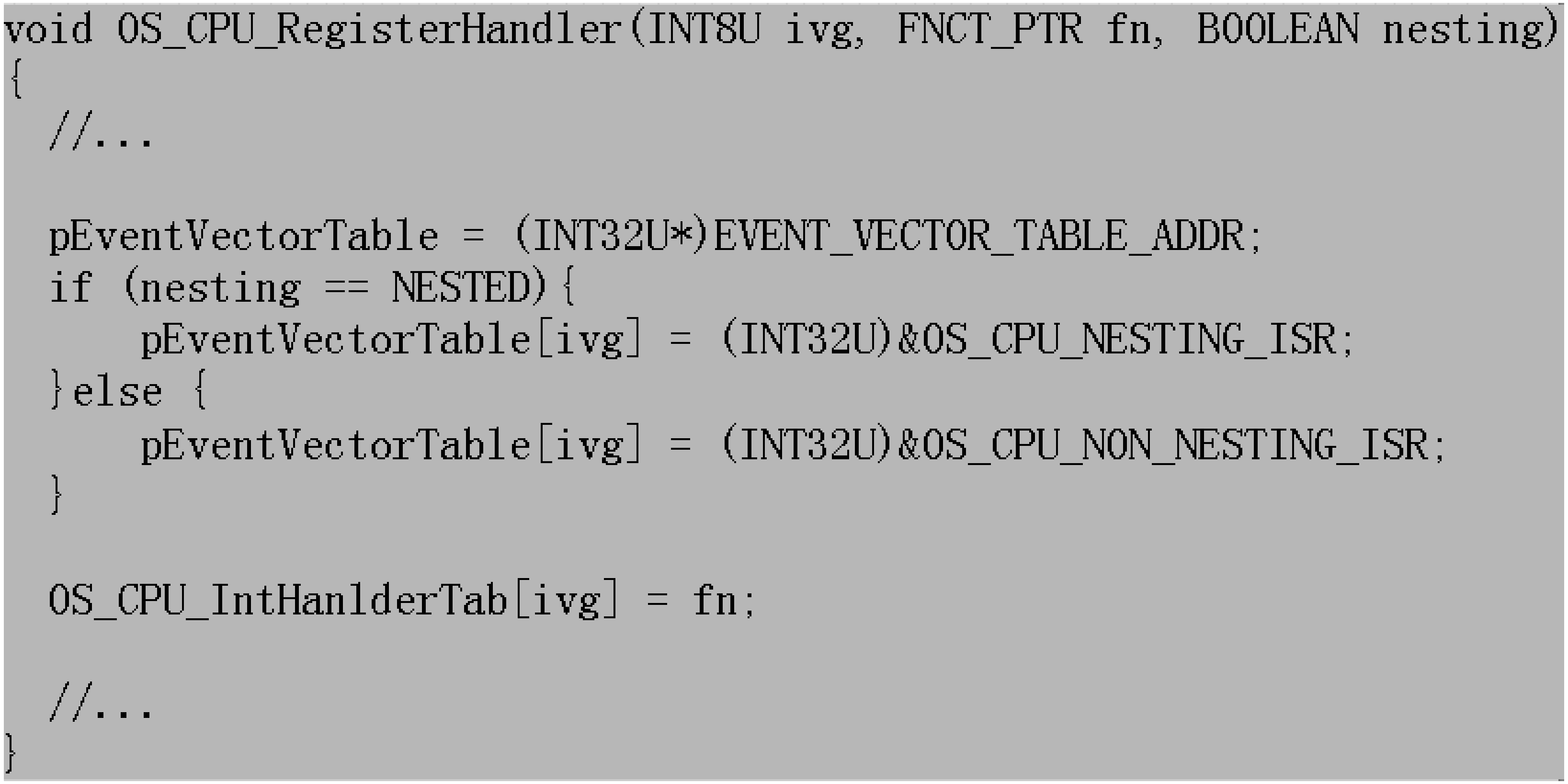
- os_cpu_c.c



```
void OSInitHookEnd (void)
{
INT32U * pEventVectorTable;

pEventVectorTable = ((INT32U*)EVENT_VECTOR_TABLE_ADDR); /* Event Vector Table Pointer */
pEventVectorTable[IVG14] = (INT32U)&OSCtxSw; /* Register the context switch */
                                            /* handler for IVG14 */

OS_CPU_EnableIntEntry(IVG14);   /* Enable Interrupt for IVG14 */
}
```

Porting uCOS-II to Bf531

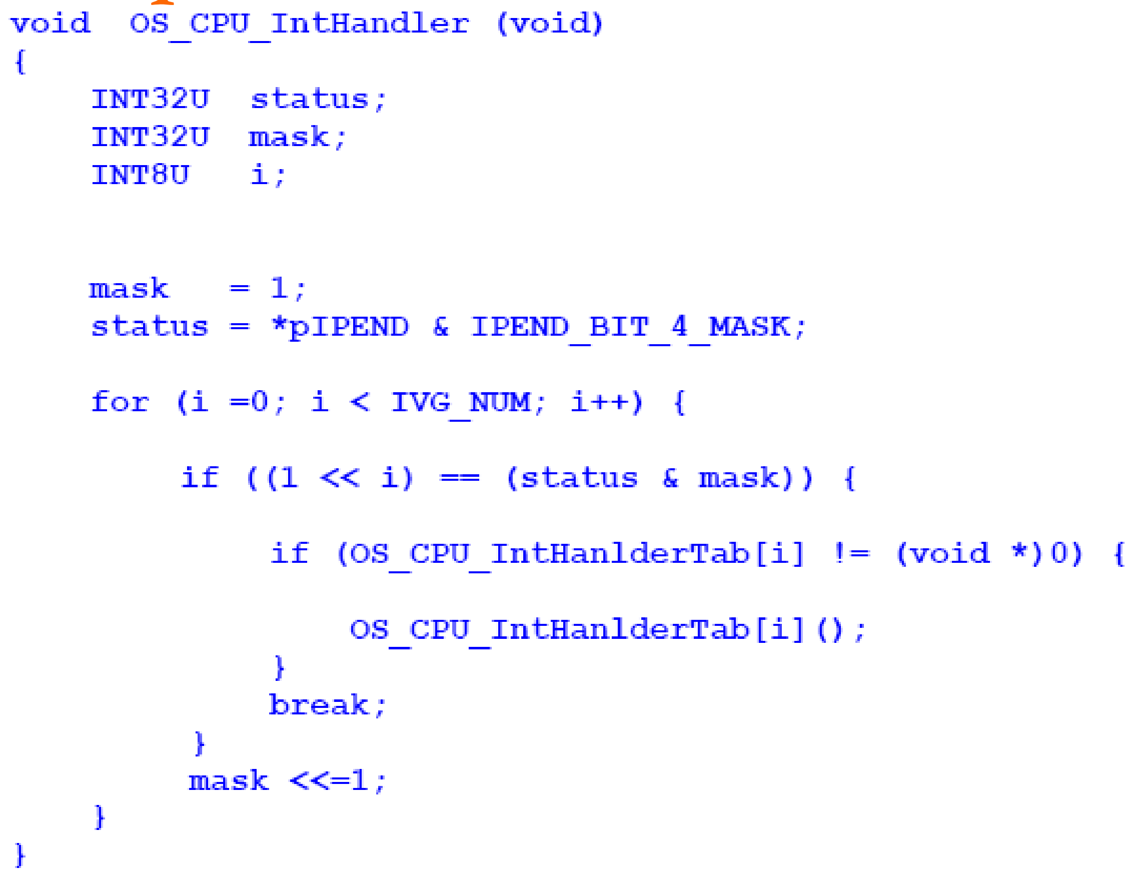
- os_cpu_c.c



```
void OS_CPU_RegisterHandler(INT8U ivg, FNCT_PTR fn, BOOLEAN nesting)
{
  //...

  pEventVectorTable = (INT32U*)EVENT_VECTOR_TABLE_ADDR;
  if (nesting == NESTED){
      pEventVectorTable[ivg] = (INT32U)&OS_CPU_NESTING_ISR;
  }else {
      pEventVectorTable[ivg] = (INT32U)&OS_CPU_NON_NESTING_ISR;
  }

  OS_CPU_IntHanlderTab[ivg] = fn;

  //...
}
```

Porting uCOS-II to Bf531

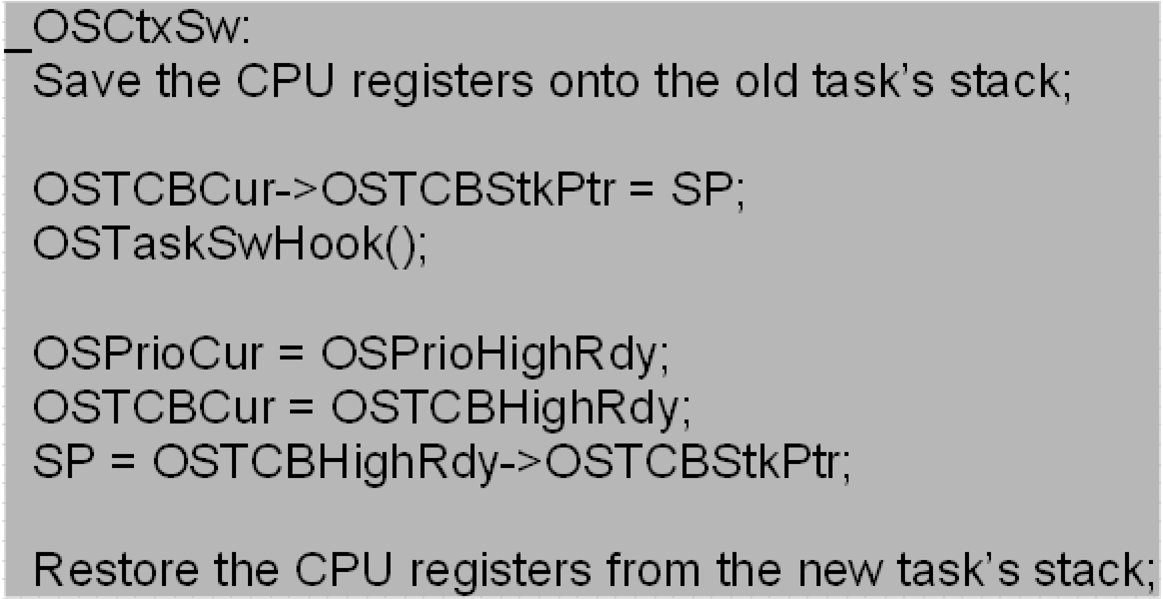
- os_cpu_c.c



```
void  OS_CPU_IntHandler (void)
{
    INT32U  status;
    INT32U  mask;
    INT8U   i;


    mask   = 1;
    status = *pIPEND & IPEND_BIT_4_MASK;

    for (i =0; i < IVG_NUM; i++) {

        if ((1 << i) == (status & mask)) {

            if (OS_CPU_IntHanlderTab[i] != (void *)0) {

                OS_CPU_IntHanlderTab[i]();
            }
            break;
        }
        mask <<=1;
    }
}
```

Porting uCOS-II to Bf531

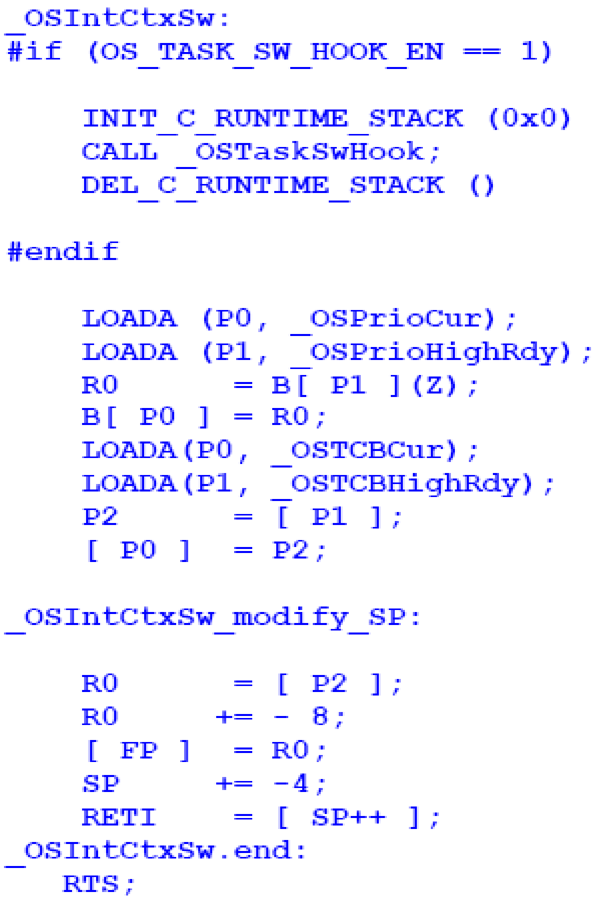
- os_cpu_a.asm



```
_OSCtxSw:
 Save the CPU registers onto the old task's stack;

 OSTCBCur->OSTCBStkPtr = SP;
 OSTaskSwHook();

 OSPrioCur = OSPrioHighRdy;
 OSTCBCur = OSTCBHighRdy;
 SP = OSTCBHighRdy->OSTCBStkPtr;

 Restore the CPU registers from the new task's stack;
```

Porting uCOS-II to Bf531

- os_cpu_a.asm



```
_OSIntCtxSw:
#if (OS_TASK_SW_HOOK_EN == 1)

    INIT_C_RUNTIME_STACK (0x0)
    CALL _OSTaskSwHook;
    DEL_C_RUNTIME_STACK ()

#endif

    LOADA (P0, _OSPrioCur);
    LOADA (P1, _OSPrioHighRdy);
    R0        = B[ P1 ](Z);
    B[ P0 ] = R0;
    LOADA(P0, _OSTCBCur);
    LOADA(P1, _OSTCBHighRdy);
    P2        = [ P1 ];
    [ P0 ]    = P2;

_OSIntCtxSw_modify_SP:

    R0        = [ P2 ];
    R0       += - 8;
    [ FP ]    = R0;
    SP       += -4;
    RETI      = [ SP++ ];
_OSIntCtxSw.end:
   RTS;
```

Porting uCOS-II to Bf531

- os_cpu_a.asm

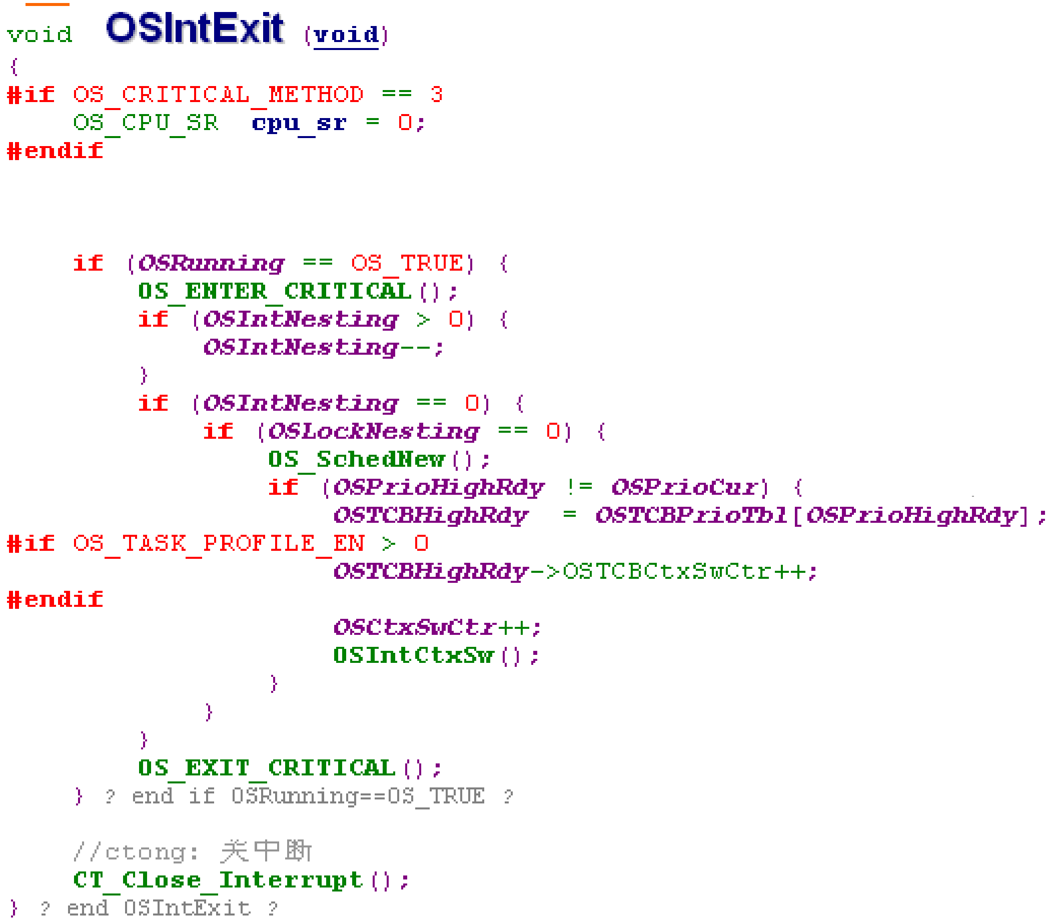
```
_OS_CPU_NESTING_ISR:

    [ -- SP ] = R0;
    [ -- SP ] = P1;
    [ -- SP ] = RETS;
    R0         = NESTED;
    CALL.X _OS_CPU_ISR_Entry;

    WORKAROUND_05000283()
    INIT_C_RUNTIME_STACK(0x0)

    CALL.X _OS_CPU_IntHandler;
    SP       += -4;
    RETI      = [ SP++ ];
    CALL.X _OSIntExit;
    DEL_C_RUNTIME_STACK()
    JUMP.X _OS_CPU_ISR_Exit;

_OS_CPU_NESTING_ISR.end:
    NOP;
```

```
_OS_CPU_NON_NESTING_ISR:

    [ -- SP ] = R0;
    [ -- SP ] = P1;
    [ -- SP ] = RETS;
    R0         = NOT_NESTED;
    CALL.X _OS_CPU_ISR_Entry;

    WORKAROUND_05000283()
    INIT_C_RUNTIME_STACK(0x0)

    CALL.X _OS_CPU_IntHandler
    CALL.X _OSIntExit;
    DEL_C_RUNTIME_STACK()
    JUMP.X _OS_CPU_ISR_Exit;


_OS_CPU_NON_NESTING_ISR.end:

    NOP;
```

```
#define DEL_C_RUNTIME_STACK()     \
    UNLINK;
```

Porting uCOS-I I to Bf531

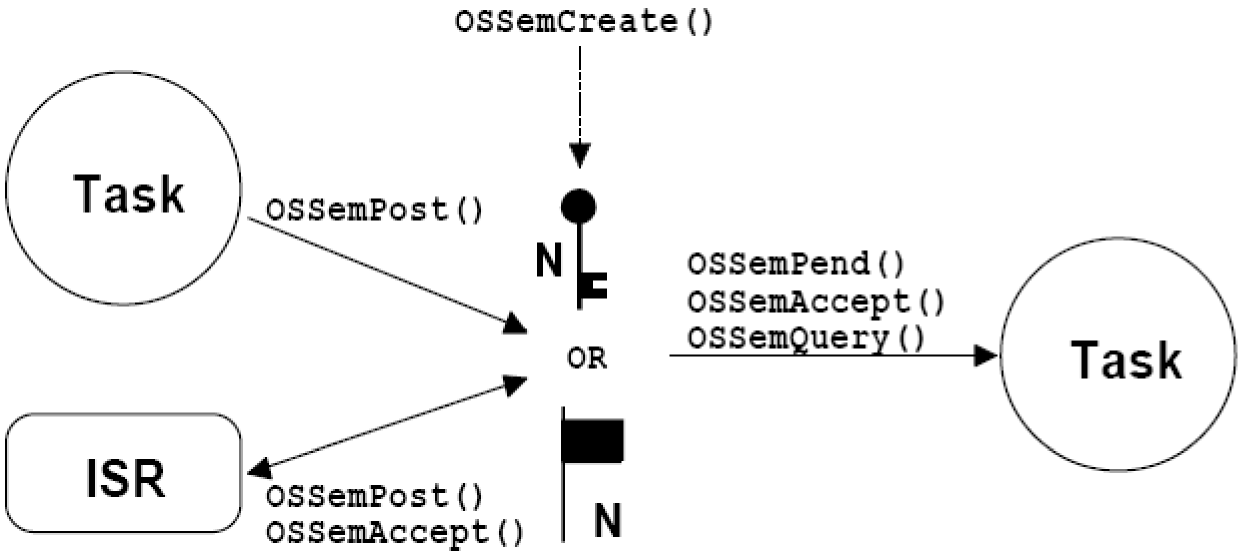
- Os_core.c



```
void  OSIntExit (void)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr = 0;
#endif


    if (OSRunning == OS_TRUE) {
        OS_ENTER_CRITICAL();
        if (OSIntNesting > 0) {
            OSIntNesting--;
        }
        if (OSIntNesting == 0) {
            if (OSLockNesting == 0) {
                OS_SchedNew();
                if (OSPrioHighRdy != OSPrioCur) {
                    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
#if OS_TASK_PROFILE_EN > 0
                    OSTCBHighRdy->OSTCBCtxSwCtr++;
#endif
                    OSCtxSwCtr++;
                    OSIntCtxSw();
                }
            }
        }
        OS_EXIT_CRITICAL();
    } ? end if OSRunning==OS_TRUE ?

    //ctong: 关中断
    CT_Close_Interrupt();
} ? end OSIntExit ?
```

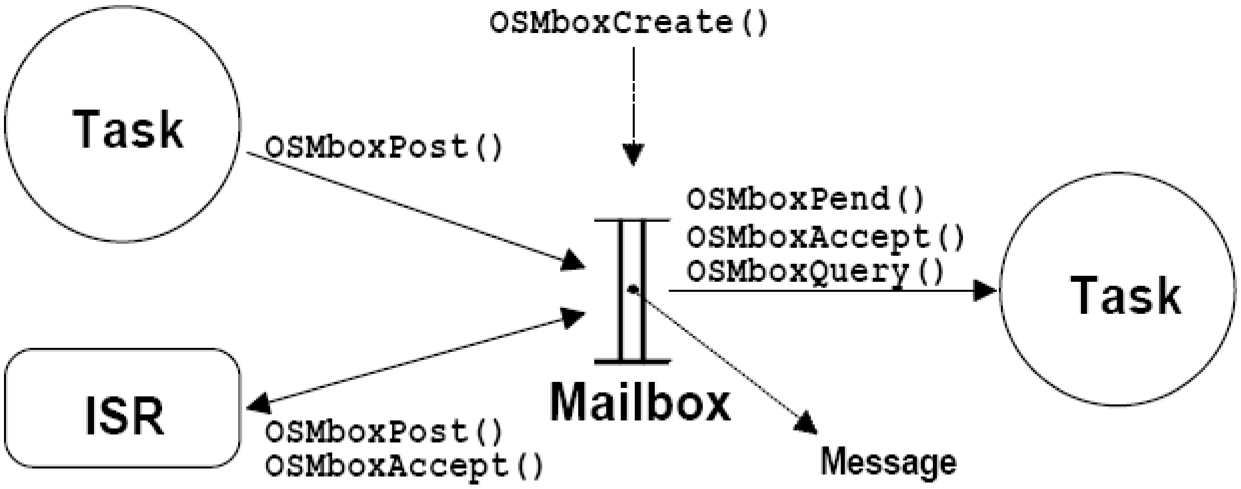
InterTask Communication and *Synchronization*

- Semaphore



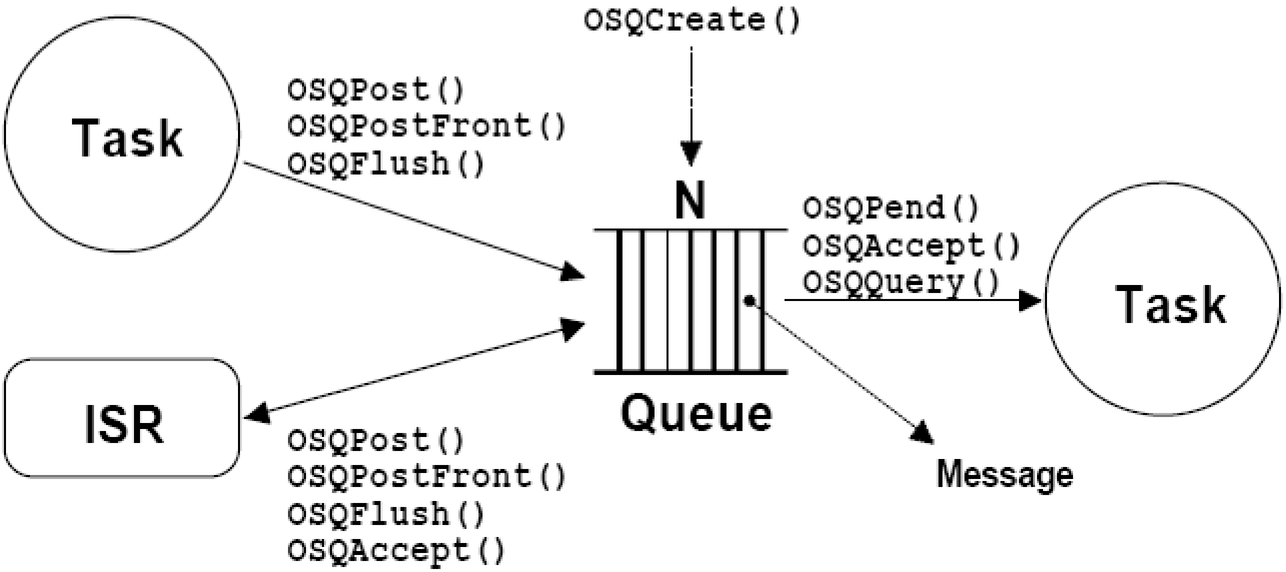
InterTask Communication and *Synchronization*

- Message Box



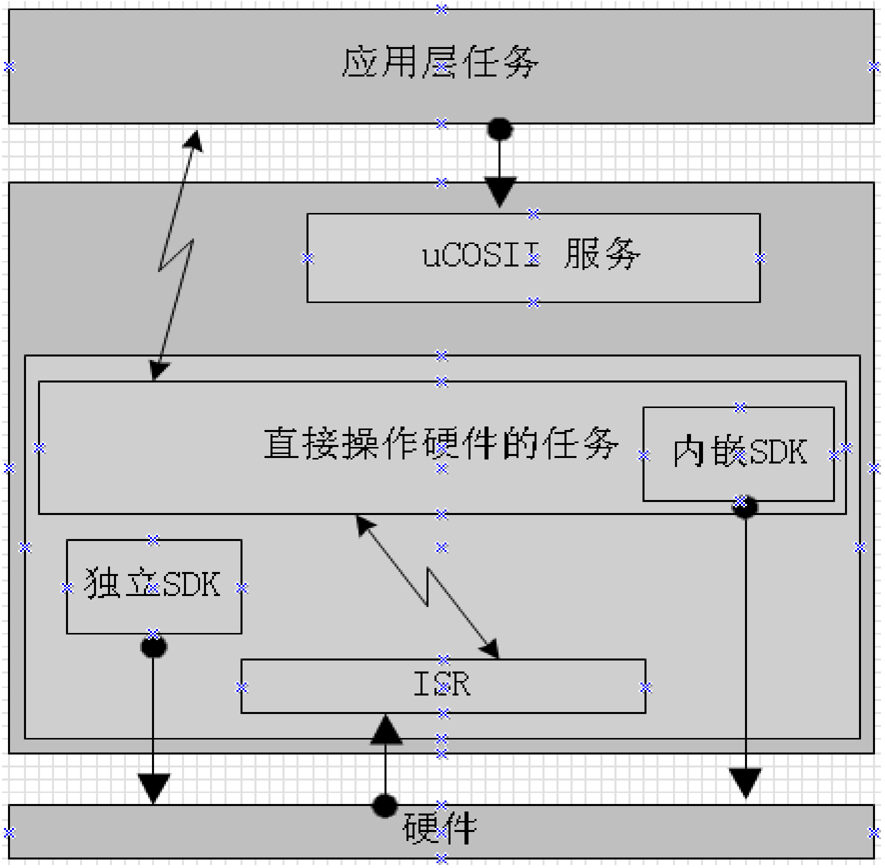
InterTask Communication and *Synchronization*

- Message Queue

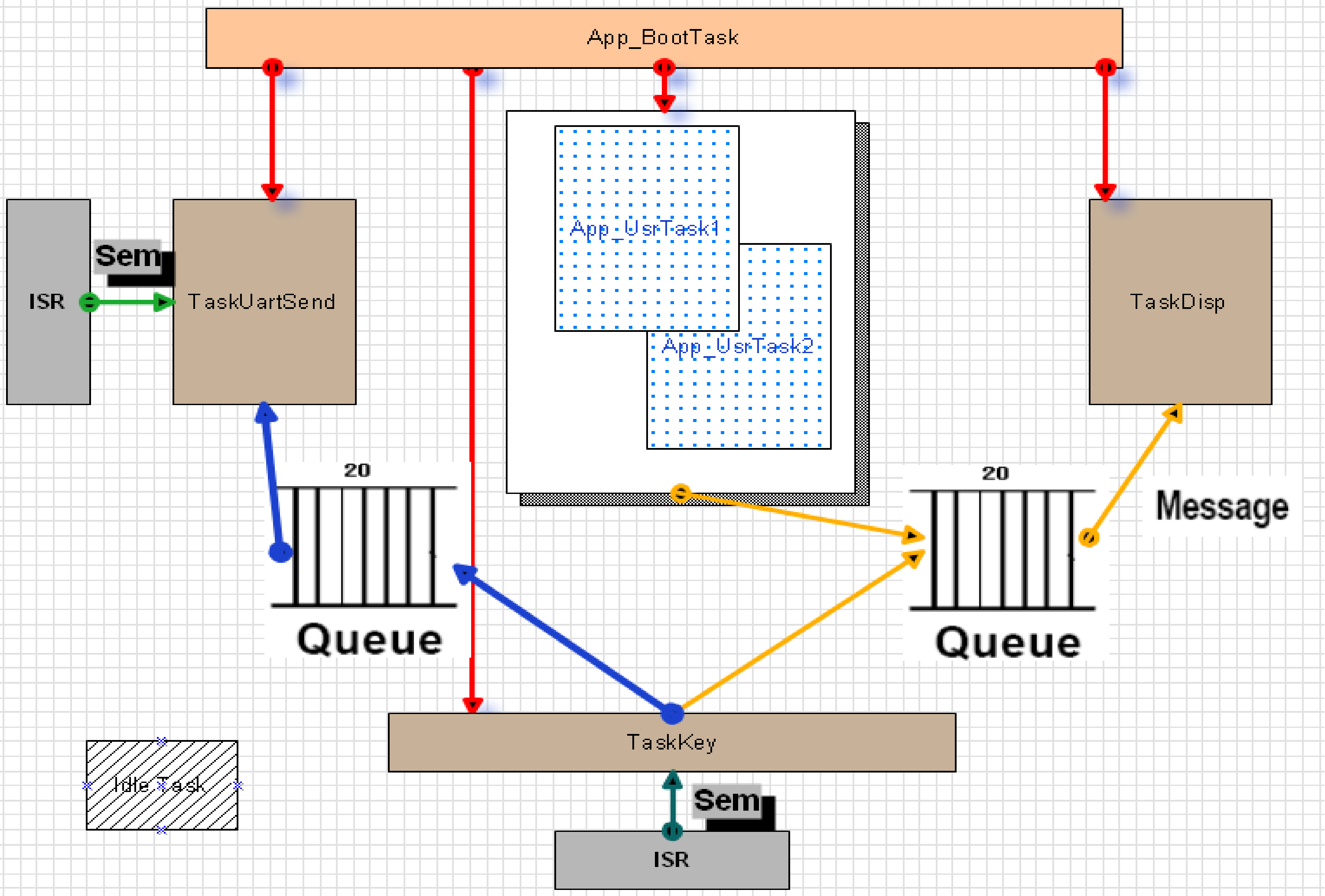


Proposed Device Drivers Mode

- Driver Mode



Proposed Device Drivers Mode



Q & A

THE END

Thanks a lot, any suggestion?