FreeRTOS Scheduler Part 2

Norman McEntire
norman.mcentire@gmail.com

References

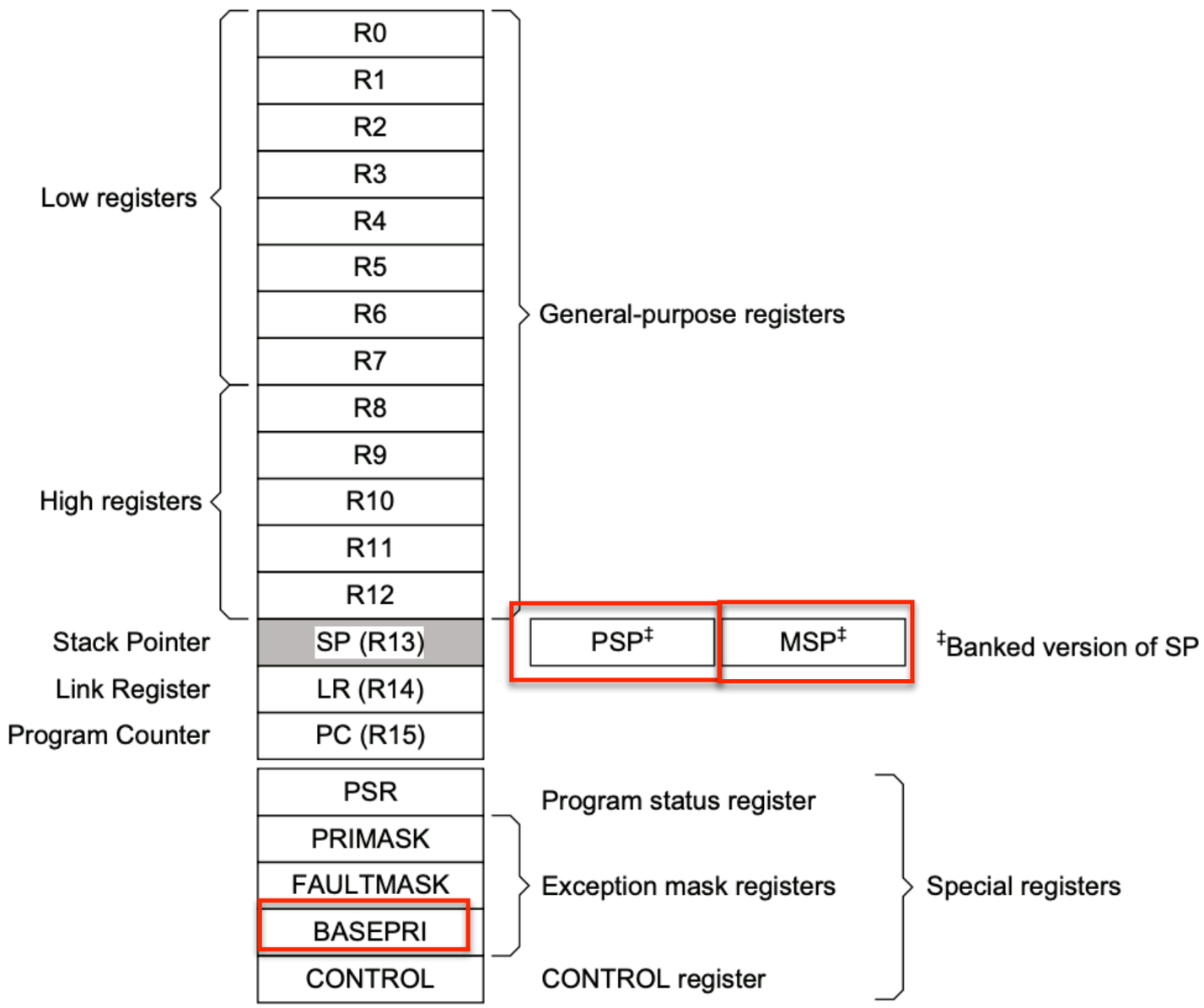
- STM32CubeIDE Project w/FreeRTOS
- http://infocenter.arm.com/help/topic/com.arm.doc.dui0553b/DUI0553.pdf

Cortex™-M4 Devices

Generic User Guide

ARM Cortext M4 Register Set

The processor core registers are:



Key Point:

* R13 has TWO registers
* MSP - Master Stack Pointer
* PSP - Program Stack Pointer

Vector Table

The Cortex-M4 Processor

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Figure 2-2 Vector table

Cortex-M4 Peripherals

Address	Core peripheral	Description
0xE000E008-0xE000E00F	SyStem Control Block	Table 4-12 on page 4-11
0xE000E010-0xE000E01F	System timer	Table 4-32 on page 4-33
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 4-2 on page 4-3
0xE000ED00-0xE000ED3F	System Control Block	Table 4-12 on page 4-11
0xE000ED90-0xE000ED93	MPU Type Register	Reads as zero, indicating MPU is not implemented[a]
0xE000ED90-0xE000EDB8	Memory Protection Unit	Table 4-38 on page 4-38
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 4-2 on page 4-3
0xE000EF30-0xE000EF44	Floating Point Unit	Table 4-49 on page 4-48

pxCurrentTCB (the current task)

```
/* The old tskTCB name is maintained above then typedefed to the new TCB_t name
below to enable the use of older kernel aware debuggers. */
typedef tskTCB TCB_t;

/*lint -save -e956 A manual analysis and inspection has been used to determine
which static variables must be declared volatile. */
PRIVILEGED_DATA TCB_t * volatile pxCurrentTCB = NULL;
```

```
#else /* portUSING_MPU_WRAPPERS */

    #define PRIVILEGED_FUNCTION
    #define PRIVILEGED_DATA
    #define FREERTOS_SYSTEM_CALL
    #define portUSING_MPU_WRAPPERS 0
```

Lists for Ready and Blocked Tasks

```
/* Lists for ready and blocked tasks. --------------------
xDelayedTaskList1 and xDelayedTaskList2 could be move to function scople but
doing so breaks some kernel aware debuggers and debuggers that rely on removing
the static qualifier. */
PRIVILEGED_DATA static List_t pxReadyTasksLists[ configMAX_PRIORITIES ] = { 0 };/*< Prioritised ready tasks. */
PRIVILEGED_DATA static List_t xDelayedTaskList1 = { 0 };                        /*< Delayed tasks. */
PRIVILEGED_DATA static List_t xDelayedTaskList2 = { 0 };                        /*< Delayed tasks (two lists are use
PRIVILEGED_DATA static List_t * volatile pxDelayedTaskList = NULL;              /*< Points to the delayed task list
PRIVILEGED_DATA static List_t * volatile pxOverflowDelayedTaskList = NULL;      /*< Points to the delayed task list
PRIVILEGED_DATA static List_t xPendingReadyList = { 0 };                        /*< Tasks that have been readied whi
```

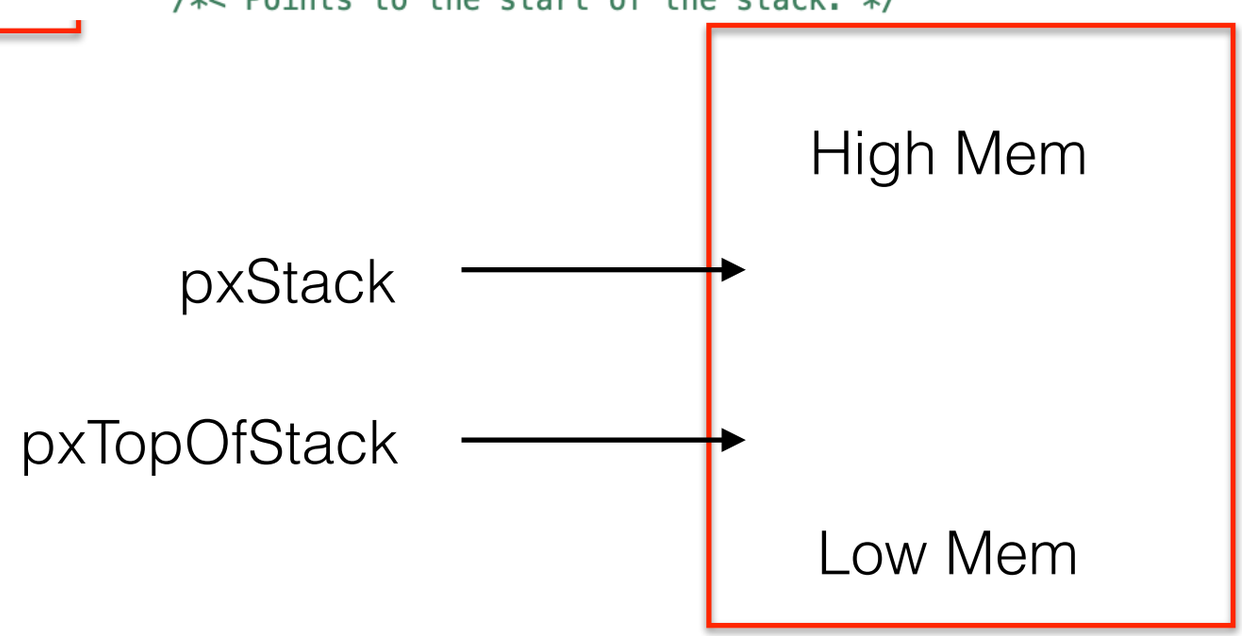
eTaskState

```
/* Task states returned by eTaskGetState. */
typedef enum
{
    eRunning = 0,   /* A task is querying the state of itself, so must be running. */
    eReady,         /* The task being queried is in a read or pending ready list. */
    eBlocked,       /* The task being queried is in the Blocked state. */
    eSuspended,     /* The task being queried is in the Suspended state, or is in the Blocked state with an infinite
    eDeleted,       /* The task being queried has been deleted, but its TCB has not yet been freed. */
    eInvalid        /* Used as an 'invalid state' value. */
} eTaskState;
```

typedef struct tskTaskControlBlock { .. }
tskTCB
Top of Stack and Start of Stack

```
/*
 * Task control block.  A task control block (TCB) is allocated for each task,
 * and stores task state information, including a pointer to the task's context
 * (the task's run time environment, including register values)
 */
typedef struct tskTaskControlBlock         /* The old naming convention is used to prevent breaking kernel aware de
{
    volatile StackType_t    *pxTopOfStack;  /*< Points to the location of the last item placed on the tasks stack.

    StackType_t             *pxStack;       /*< Points to the start of the stack. */
```



typedef struct tskTaskControlBlock { .. }
tskTCB
Lists

```
    ListItem_t          xStateListItem; /*< The list that the state list item of a task is reference from denotes th
    ListItem_t          xEventListItem;     /*< Used to reference a task from an event list. */
```

=> All tasks in same state are in same list

=> All tasks waiting on same event on same list

typedef struct tskTaskControlBlock { .. }
tskTCB
pcTaskName

```
264     char                    pcTaskName[ configMAX_TASK_NAME_LEN ];/*< Descriptive name given to the task when created.
```

typedef struct tskTaskControlBlock { .. }
tskTCB
uxPriority

```
UBaseType_t          uxPriority;          /*< The priority of the task.  0 is the lowest priority. */
```

0 is lowest priority

typedef struct tskTaskControlBlock { .. } tskTCB
configUSE_MUTEXES == 1

```
#if ( configUSE_MUTEXES == 1 )
    UBaseType_t     uxBasePriority;     /*< The priority last assigned to the task - used by the priority inheri
    UBaseType_t     uxMutexesHeld;
#endif
```

typedef struct tskTaskControlBlock { .. }
tskTCB

And a few others….just the ones were shown

vTaskStartScheduler
vTaskStartScheduler -> xPortStartScheduler()

```
2062
2063        /* Setting up the timer tick is hardware specific and thus in the
2064        portable interface. */
2065        if( xPortStartScheduler() != pdFALSE )
2066        {
2067            /* Should not reach here as if the scheduler is running the
2068            function will not return. */
2069        }
2070        else
2071        {
2072            /* Should only reach here if a task calls xTaskEndScheduler(). */
2073        }
```

xPortStartScheduler() - Part 1

```
286  */
287 BaseType_t xPortStartScheduler( void )
288 {
289     /* configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to 0.
290     See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html */
291     configASSERT( configMAX_SYSCALL_INTERRUPT_PRIORITY );
292
293     /* This port can be used on all revisions of the Cortex-M7 core other than
294     the r0p1 parts.  r0p1 parts should use the port from the
295     /source/portable/GCC/ARM_CM7/r0p1 directory. */
296     configASSERT( portCPUID != portCORTEX_M7_r0p1_ID );
297     configASSERT( portCPUID != portCORTEX_M7_r0p0_ID );
298
```

xPortStartScheduler() - Part 2

```
    /* Start the timer that generates the tick ISR.  Interrupts are disabled
    here already. */
    vPortSetupTimerInterrupt();
```

vPortSetupTimerInterrupt()

0xE000E010	SYST_CSR	RW	Privileged	a	*SysTick Control and Status Register*

0xE000E018	SYST_CVR	RW	Privileged	Unknown	*SysTick Current Value Register* on page 4-35

```
675 /*
676  * Setup the systick timer to generate the tick interrupts at the required
677  * frequency.
678  */
679 __attribute__(( weak )) void vPortSetupTimerInterrupt( void )
680 {
681     /* Calculate the constants required to configure the tick interrupt. */
682     #if( configUSE_TICKLESS_IDLE == 1 )
683     {
684         ulTimerCountsForOneTick = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ );
685         xMaximumPossibleSuppressedTicks = portMAX_24_BIT_NUMBER / ulTimerCountsForOneTick;
686         ulStoppedTimerCompensation = portMISSED_COUNTS_FACTOR / ( configCPU_CLOCK_HZ / configSYSTICK_CLOCK_HZ );
687     }
688     #endif /* configUSE_TICKLESS_IDLE */
689
690     /* Stop and clear the SysTick. */
691     portNVIC_SYSTICK_CTRL_REG = 0UL;
692     portNVIC_SYSTICK_CURRENT_VALUE_REG = 0UL;
693
694     /* Configure SysTick to interrupt at the requested rate. */
695     portNVIC_SYSTICK_LOAD_REG = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ ) - 1UL;
696     portNVIC_SYSTICK_CTRL_REG = ( portNVIC_SYSTICK_CLK_BIT | portNVIC_SYSTICK_INT_BIT | portNVIC_SYSTICK_ENABLE_BIT )
697 }
698 /*-----------------------------------------------------------*/
```

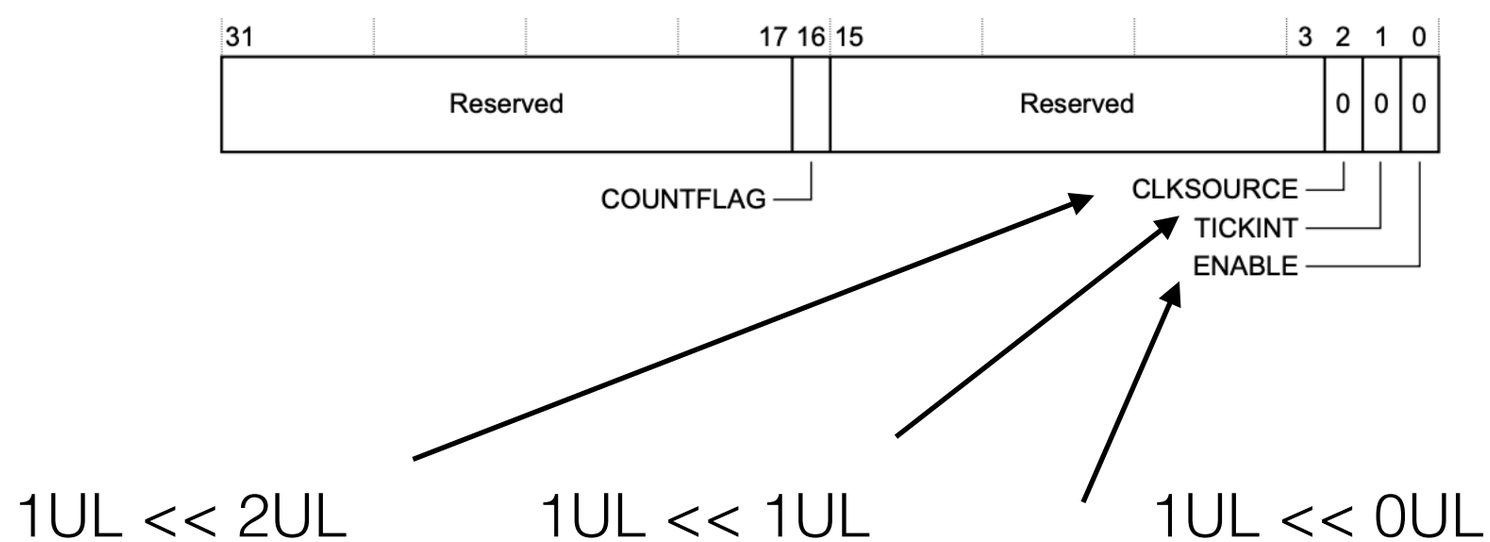
1UL << 2UL 1UL << 1UL 1UL << 0UL

0xE000E014	SYST_RVR	RW	Privileged	Unknown	*SysTick Reload Value Register* on page 4-34

SysTick Control and Status Register

4.4.1 SysTick Control and Status Register

The SysTick SYST_CSR register enables the SysTick features. The register resets to `0x00000000`, or to `0x00000004` if your device does not implement a reference clock. See the register summary in Table 4-32 for its attributes. The bit assignments are:



vPortEnableVFP()

```
371
372     /* Ensure the VFP is enabled - it should be anyway. */
373     vPortEnableVFP();
374
```

0xE000ED88	CPACR	RW	0x00000000	*Coprocessor Access Control Register*

```
700  /* This is a naked function. */
701  static void vPortEnableVFP( void )
702  {
703      __asm volatile
704      (
705          "   ldr.w r0, =0xE000ED88       \n" /* The FPU enable bits are in the CPACR. */
706          "   ldr r1, [r0]                \n"
707          "                               \n"
708          "   orr r1, r1, #( 0xf << 20 )  \n" /* Enable CP10 and CP11 coprocessors, then save back. */
709          "   str r1, [r0]                \n"
710          "   bx r14                      "
711      );
712  }
```

Assembly Language Instructions

ldr - load register
orr - OR register
str - store register
bx - branch indirect

Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors. See the register summary in *Cortex-M4F floating-point system registers* for its attributes. The bit assignments are:

31 30 29 28 27 26 25 24	23 22	21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved	CP11	CP10	Reserved

prvPortStartFirstTask() Part 1

```
    /* Start the first task. */
    prvPortStartFirstTask();
```

prvPortStartFirstTask() Part 2

```
static void prvPortStartFirstTask( void )
{
    /* Start the first task.  This also clears the bit that indicates the FPU is
    in use in case the FPU was used before the scheduler was started - which
    would otherwise result in the unnecessary leaving of space in the SVC stack
    for lazy saving of FPU registers. */
    __asm volatile(
                    " ldr r0, =0xE000ED08   \n" /* Use the NVIC offset register to locate the stack. */
                    " ldr r0, [r0]          \n"
                    " ldr r0, [r0]          \n"
                    " msr msp, r0           \n" /* Set the msp back to the start of the stack. */
                    " mov r0, #0            \n" /* Clear the bit that indicates the FPU is in use, see comment above.
                    " msr control, r0       \n"
                    " cpsie i               \n" /* Globally enable interrupts. */
                    " cpsie f               \n"
                    " dsb                   \n"
                    " isb                   \n"
                    " svc 0                 \n" /* System call to start first task. */
                    " nop                   \n"
                );
}
```

1. Load register with adds of offset

0xE000ED08	VTOR	RW	Privileged	0x00000000	*Vector Table Offset Register* on page 4-16

4. Load MSP

3. Load stack

2. Load register with offset address

cpsie i - change processor state, interrupts enable, PRIMASK
cpsie f - change processor state, interrupts enable, FAULTMASK
dsb - data sync barrier
isb - instruction sync barrier
svc - supervisor call
nop - no operation

Vector Table

The Cortex-M4 Processor

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Figure 2-2 Vector table

vPortSVCHandler

```
239 }
240 /*-----------------------------------------------------------*/
241
242 void vPortSVCHandler( void )
243 {
244     __asm volatile (
245                     "    ldr r3, pxCurrentTCBConst2      \n" /* Restore the context. */
246                     "    ldr r1, [r3]                    \n" /* Use pxCurrentTCBConst to get the pxCurrentTCB address.
247                     "    ldr r0, [r1]                    \n" /* The first item in pxCurrentTCB is the task top of stac
248                     "    ldmia r0!, {r4-r11, r14}        \n" /* Pop the registers that are not automatically saved on
249                     "    msr psp, r0                     \n" /* Restore the task stack pointer. */
250                     "    isb                             \n"
251                     "    mov r0, #0                      \n"
252                     "    msr basepri, r0                 \n"
253                     "    bx r14                          \n"
254                     "                                    \n"
255                     "    .align 4                        \n"
256                     "pxCurrentTCBConst2: .word pxCurrentTCB          \n"
257                 );
258 }
```

ldmia - load multiple increment after
(pop the register context for this task on the stack)

To be continued…