FreeRTOS Scheduler Part 1

Norman McEntire
norman.mcentire@gmail.com

Special Thanks

- Chris (Thread mode vs Handler mode)
- Ernie (Scheduler, ARM Assembly Code)
- Young (Scheduler, ARM Assembly Code)

References

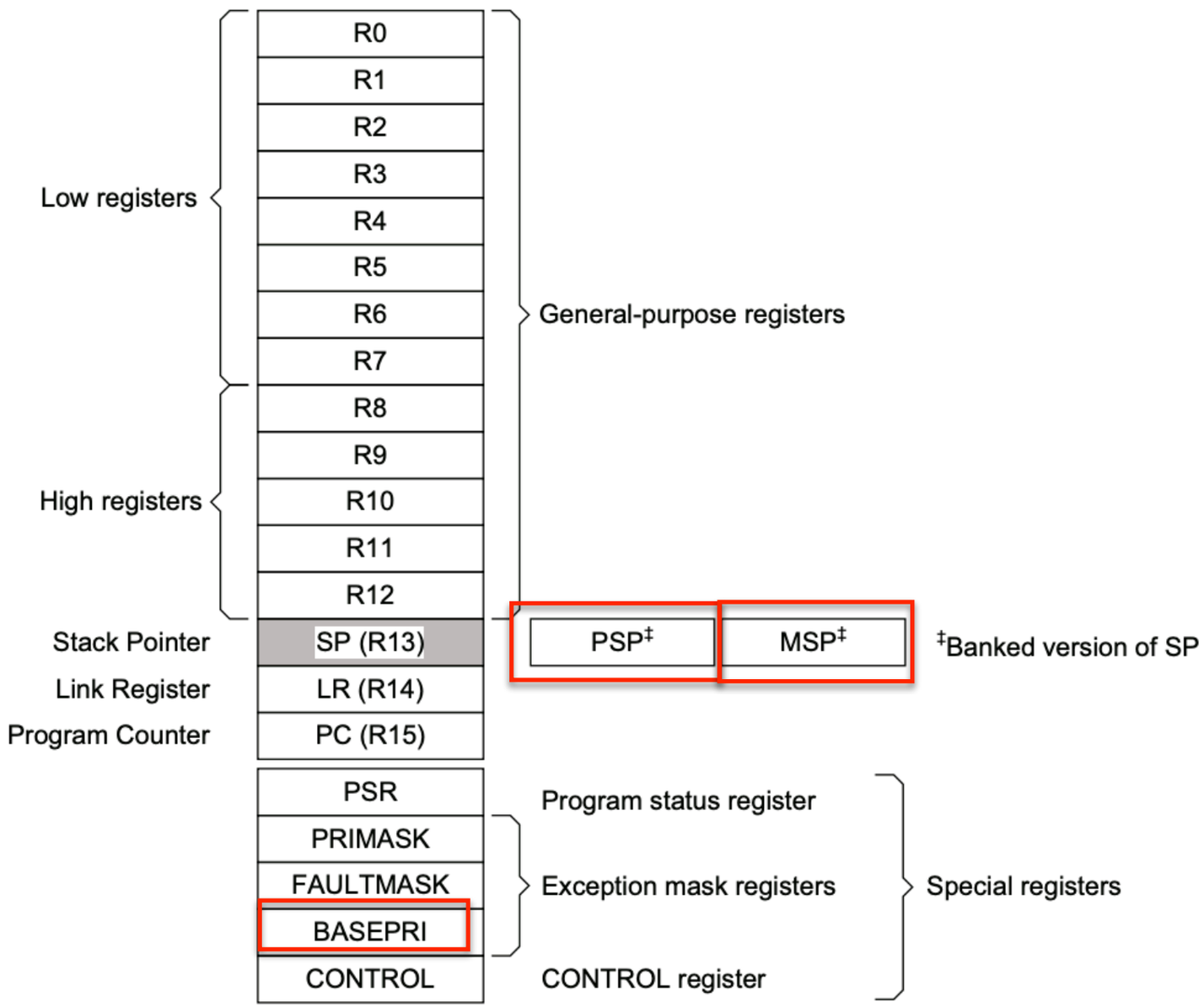
- STM32CubeIDE Project w/FreeRTOS
- http://infocenter.arm.com/help/topic/com.arm.doc.dui0553b/DUI0553.pdf

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

ARM Cortext M4
Base Priority Register

Base Priority Mask Register

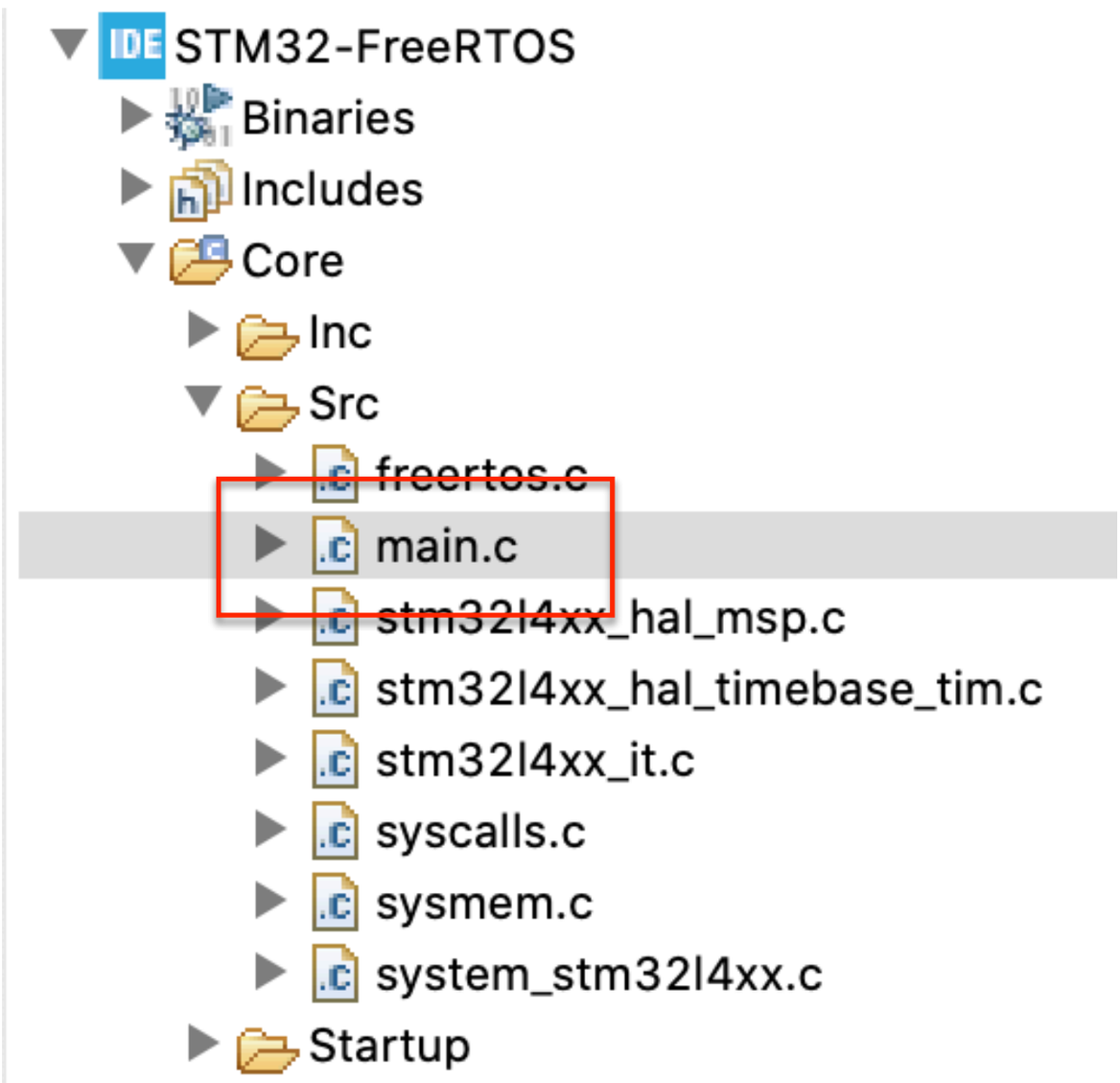
The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in Table 2-2 on page 2-3 for its attributes. The bit assignments are:

31 … 8	7 … 0
Reserved	BASEPRI

Table 2-9 BASEPRI register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI[a]	Priority mask bits: `0x00` = no effect Nonzero = defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

Code Tour - main.c



Code Tour - main.c
main() -> osKernelStart()

```
/* Create the thread(s) */
/* definition and creation of defaultTask */
osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

/* definition and creation of myTask02 */
osThreadDef(myTask02, StartTask02, osPriorityLow, 0, 128);
myTask02Handle = osThreadCreate(osThread(myTask02), NULL);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* Start scheduler */
osKernelStart();
```

CMSIS RTOS Functions

Code Tour - cmsis_os.c
osKernelStart() -> vTaskStartScheduler()

port.c | main.c | cmsis_os.c

```c
 * @note   MUST REMAIN UNCHANGED: \b osKernelInitialize shall be consistent in every CMSIS-RT
 */
osStatus osKernelInitialize (void);

/**
 * @brief  Start the RTOS Kernel with executing the specified thread.
 * @param  thread_def    thread definition referenced with \ref osThread.
 * @param  argument      pointer that is passed to the thread function as start argument.
 * @retval status code that indicates the execution status of the function
 * @note   MUST REMAIN UNCHANGED: \b osKernelStart shall be consistent in every CMSIS-RTOS.
 */
osStatus osKernelStart (void)
{
  vTaskStartScheduler();

  return osOK;
}
```

FreeRTOS Function

Code Tour - tasks.c
vTaskStartScheduler - Part 1

port.c | main.c | cmsis_os.c | tasks.c

```c
#endif /* ( ( INCLUDE_xTaskResumeFromISR == 1 ) && ( IN
/*-----------------------------------------------------

void vTaskStartScheduler( void )
{
BaseType_t xReturn;

    /* Add the idle task at the lowest priority. */
    #if( configSUPPORT_STATIC_ALLOCATION == 1 )
    {
```

FreeRTOS Config Option

Code Tour - tasks.c
vTaskStartScheduler - Part 2
Create Idle Task

FreeRTOS Function

```
1998    #else
1999    {
2000        /* The Idle task is being created using dynamically allocated RAM. */
2001        xReturn = xTaskCreate(  prvIdleTask,
2002                                configIDLE_TASK_NAME,
2003                                configMINIMAL_STACK_SIZE,
2004                                ( void * ) NULL,
2005                                portPRIVILEGE_BIT, /* In effect ( tskIDLE_PRIORITY | portPRIVILEGE_BIT ), bu
2006                                &xIdleTaskHandle ); /*lint !e961 MISRA exception, justified as it is not a r
2007    }
2008    #endif /* configSUPPORT_STATIC_ALLOCATION */
```

FreeRTOS Idle Task
(when no other task is running)

Code Tour - tasks.c
vTaskStartScheduler - Part 3
Create Timer Task (if configured)

FreeRTOS Config Option

```
#if ( configUSE_TIMERS == 1 )
{
    if( xReturn == pdPASS )
    {
        xReturn = xTimerCreateTimerTask();
    }
    else
    {
        mtCOVERAGE_TEST_MARKER();
    }
}
#endif /* configUSE_TIMERS */
```

Code Tour - tasks.c
vTaskStartScheduler - Part 4
Disable Interrupts

```
2033
2034        /* Interrupts are turned off here, to ensure a tick does not occur
2035        before or during the call to xPortStartScheduler().  The stacks of
2036        the created tasks contain a status word with interrupts switched on
2037        so interrupts will automatically get re-enabled when the first task
2038        starts to run. */
2039        portDISABLE_INTERRUPTS();
2040
```

FreeRTOS Port Specific Code
(changes based on uC Port)

Code Tour - portmacro.h portDISABLE_INTERRUPTS()

port.c | main.c | cmsis_os.c | tasks.c | portmacro.h

```
90  #define portNVIC_INT_CTRL_REG       ( * ( ( volatile uint32_t * ) 0xe000ed04
91  #define portNVIC_PENDSVSET_BIT      ( 1UL << 28UL )
92  #define portEND_SWITCHING_ISR( xSwitchRequired ) if( xSwitchRequired != pdFA
93  #define portYIELD_FROM_ISR( x ) portEND_SWITCHING_ISR( x )
94  /*-----------------------------------------------------------*/
95
96  /* Critical section management. */
97  extern void vPortEnterCritical( void );
98  extern void vPortExitCritical( void );
99  #define portSET_INTERRUPT_MASK_FROM_ISR()       ulPortRaiseBASEPRI()
100 #define portCLEAR_INTERRUPT_MASK_FROM_ISR(x)    vPortSetBASEPRI(x)
101 #define portDISABLE_INTERRUPTS()                vPortRaiseBASEPRI()
102 #define portENABLE_INTERRUPTS()                 vPortSetBASEPRI(0)
103 #define portENTER_CRITICAL()                    vPortEnterCritical()
104 #define portEXIT_CRITICAL()                     vPortExitCritical()
105
```

Code Tour - portmacro.h vPortRaiseBASEPRI - Part 1

This is assembly language code

Code has "side effects". Tells compiler do not touch!

```
portFORCE_INLINE static void vPortRaiseBASEPRI( void )
{
uint32_t ulNewBASEPRI;

    __asm volatile
    (
        "   mov %0, %1                                          \n" \
        "   msr basepri, %0                                     \n" \
        "   isb                                                 \n" \
        "   dsb                                                 \n" \
        :"=r" (ulNewBASEPRI) : "i" ( configMAX_SYSCALL_INTERRUPT_PRIORITY ) : "memory"
    );
}
```

output operand (ulNewBASEPRI)

Input operand configMAX…

Assembly Instructions

mov - move to general register
msr - move to special register
ism - instruction set barrier
dsb - data sync barrier

__asm volatile ("":::"memory")

Code Tour - portmacro.h vPortRaiseBASEPRI - Part 2

```
/* The highest interrupt priority that can be used by any interrupt service
 routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
 INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
 PRIORITY THAN THIS! (higher priorities are lower numeric values. */
 #define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5

/* Interrupt priorities used by the kernel port layer itself.  These are generic
 to all Cortex-M ports, and do not rely on any particular library functions. */
 #define configKERNEL_INTERRUPT_PRIORITY        ( configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )
/* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
 See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
 #define configMAX_SYSCALL_INTERRUPT_PRIORITY    ( configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS) )
```

```
/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
 /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
 #define configPRIO_BITS         __NVIC_PRIO_BITS
#else
 #define configPRIO_BITS         4
#endif
```

```
/**
  * @brief Configuration of the Cortex-M4 Processor and Core Peripherals
   */
#define __CM4_REV                 0x0001  /*!< Cortex-M4 revision r0p1                        */
#define __MPU_PRESENT             1       /*!< STM32L4XX provides an MPU                      */
#define __NVIC_PRIO_BITS          4       /*!< STM32L4XX uses 4 Bits for the Priority Levels */
#define __Vendor_SysTickConfig    0       /*!< Set to 1 if different SysTick Config is used  */
#define __FPU_PRESENT             1       /*!< FPU present                                    */
```

ARM Cortext-M4 Base Priority Register

Base Priority Mask Register

The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in Table 2-2 on page 2-3 for its attributes. The bit assignments are:

31 … 8	7 … 0
Reserved	BASEPRI

Table 2-9 BASEPRI register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI[a]	Priority mask bits: `0x00` = no effect Nonzero = defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

Code Tour - tasks.c vTaskStartScheduler - Part 4 configUSE_NEWLIB_REENTRANT

```
#if ( configUSE_NEWLIB_REENTRANT == 1 )
{
    /* Switch Newlib's _impure_ptr variable to point to the _reent
    structure specific to the task that will run first. */
    _impure_ptr = &( pxCurrentTCB->xNewLib_reent );
}
#endif /* configUSE_NEWLIB_REENTRANT */
```

Code Tour - tasks.c
vTaskStartScheduler - Part 5
portMAX_DELAY, xTickCount

```
2048
2049        xNextTaskUnblockTime = portMAX_DELAY;
2050        xSchedulerRunning = pdTRUE;
2051        xTickCount = ( TickType_t ) configINITIAL_TICK_COUNT;
2052
```

```
867 #ifndef configINITIAL_TICK_COUNT
868     #define configINITIAL_TICK_COUNT 0
869 #endif
```

```
59 #if( configUSE_16_BIT_TICKS == 1 )
60     typedef uint16_t TickType_t;
61     #define portMAX_DELAY ( TickType_t ) 0xffff
62 #else
63     typedef uint32_t TickType_t;
64     #define portMAX_DELAY ( TickType_t ) 0xffffffffUL
65
66     /* 32-bit tick type on a 32-bit architecture, so reads of the tick count do
67     not need to be guarded with a critical section. */
68     #define portTICK_TYPE_IS_ATOMIC 1
69 #endif
```

Code Tour - tasks.c
vTaskStartScheduler - Part 6
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        }
```

To be continued…