Application Report
SPNA177–April 2013

MPU Configuration

Trevor Jones

ABSTRACT

This application report provides an introduction to the MPU register and recommended usage for the TMS570 and RM48 devices.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/spna177.

Contents

List of Figures

List of Tables

1 MPU Register Set

The registers that control the MPU are part of the CP15 register set. Global MPU enable is in the system control register (see Table 1) and the registers to setup the MPU regions and part of the MPU Control and Configuration registers (see Table 2).

Table 1. System Control Register

CRn	Opcode 1	CRm	Opcode 2	
C6	0	C1	0	System Control Register

Table 2. MPU Control and Configuration Registers

CRn	Opcode 1	CRm	Opcode 2	
C6	0	C1	0	Region Base Register
			2	Region Size and Enable Register
			4	Region Access and Control Register
		C2	0	Memory Region Select Register

1.1 System Control Register

This register contains the MPU enable bit and other system control bits. It is important that when changing the MPU control bit (bit [0]) that the value of bits 1 to 31 be preserved.

The system control register is shown in Figure 1 and described in Table 3.

Figure 1. System Control Register

31	1	0
Reserved		M

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 3. System Control Register Field Descriptions

Bit	Field	Value	Description
31 -1	Reserved	0	Reserved
0	M		Enables the MPU:
		0	MPU disabled. This is the reset value.
		1	MPU enabled

1.2 MPU Control and Configuration Registers

There are four registers that are used to setup the individual MPU regions. The region base address (see Figure 2) which as the name suggests, is the address for the start of the MPU region.

The region base register is shown in Figure 2 and described in Table 4.

Figure 2. Region Base Register

31	5	4	0
Base Address		0000	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 4. Region Base Register Field Descriptions

Bit	Field	Value	Description
31 -5	Base Address		Base Address: Defines bits [31:5] of the base address of a region.
4-0	0000		Not Used (SBZ).

The region size and enable register (see Figure 3) determines the size of the region. This register also contains the region enable bit it should be the last register to be written to.

The region size and enable register is shown in Figure 3 and described in Table 5.

Figure 3. Region Size and Enable Register

31 16	15 8	7 6	5 1	0
0000000000000000	Sub-Region Disable	00	Size	E

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5. Region Size and Enable Register Field Descriptions

Bit	Field	Value	Description		
31-16	0000000000000000		Not Used (SBZ)		
15-8	Sub-Region Disable		Each bit position represents a sub-region, 0-7.		
			Bit [8] corresponds to sub-region 0		
			Bit [15] corresponds to sub-region 7		
			The meaning of each bit is:		
		0	Address range is part of this region		
		1	Address range is not part of this region		
7-6	00		Not Used (SBZ)		
5-1	Size		Defines the region size	b01101 = 16KB	b10111 = 16MB
			b00100 = 32 bytes	b01110 = 32KB	b11000 = 32MB
			b00101 = 64 bytes	b01111 = 64KB	b11001 = 64MB
			b00110 = 128 bytes	b10000 = 128KB	b11010 = 128MB
			b00111 = 256 bytes	b10001 = 256KB	b11011 = 256MB
			b01000 = 512 bytes	b10010 = 512KB	b11100 = 512MB
			b01001 = 1KB	b10011 = 1MB	b11101 = 1GB
			b01010 = 2KB	b10100 = 2MB	b11110 = 2GB
			b01011 = 4KB	b10101 = 4MB	b11111 = 4GB
			b01100 = 8KB	b10110 = 8MB	
0	E		Enables or disables a memory region:		
		0	Memory region disabled		
		1	Memory region enabled		

The region access control register (see Figure 4) determines the type memory region, if it is Normal, Cacheable or Sharable. The access privilege mode (user or privileged) and if the region contains executable code.

The region access control register is shown in Figure 4 and described in Table 6.

Figure 4. Region Access Control Register

31 13	12	11	10 8	7 6	5 3	2	1	0
00000000000000000000	XN	0	AP	00	TEX	S	C	B

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6. Region Access Control Register Field Descriptions

Bit	Field	Value	Description
31-13	0000000000000000000		Not used (SBZ)

Table 6. Region Access Control Register Field Descriptions (continued)

Bit	Field	Value	Description
12	XN		Execute never, determines if a region is executable:
		0	All instruction fetches enabled
		1	No instruction fetches enabled.
11	0		Not used (SBZ)
10-8	AP		Access Permissions
7-6	00		Not used (SBZ)
5-3	TEX		Type extension, defines the memory type of the region.
2	S		Share. Determines if the memory region is Shared or Non-shared:
		0	Non-shared
		1	Shared
1	C		Cacheable
0	B		Buffered (cache write back)

The three MPU register required to program a region are duplicated for each of the MPU regions. In order to program them we must first select the required region, this is done via the region select register (see Figure 5).

The region select register is shown in Figure 5 and described in Table 7.

Figure 5. Region Select Register

31	4	3	0
Base Address		Region	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7. Region Select Register Field Descriptions

Bit	Field	Value	Description
31-4	Base Address		Not Used (SBZ).
3-0	Region		Defines the register group to be selected.

1.3 MPU Region Access Control Settings

The access control register is basically split into two settings, the TEX, C and B bits that control the memory type and the AP field, which controls the access permissions.

1.3.1 Access Permissions

The AP field controls the access permissions for the memory region, and can limit the access to user mode, privileged mode and also limit access the read and write or read only. It can also be used to deny any access to the region.

Table 8. Access Permissions

AP Bit Values	Privileged Access	User Access	Description
000	None	None	No access to memory region.
001	Read/Write	None	Access in privileged mode only.
010	Read/write	Read-Only	User mode access restricted to read only.
011	Read/write	Read/Write	Full Access.
101	Read-Only	None	Access restricted the privileged read only, no user access allowed.
110	Read-Only	Read-Only	Access restricted to read-only for both user and privileged mode.

1.3.2 Memory Type

The TEX, C and B bits control the memory type, recommended settings for the TI devices.

Table 9. Memory Types

TEX	C	B	Description
0	0	0	Strongly Ordered – Recommended for System Frame
0	0	1	Device – Peripheral Region
1	0	0	Normal Memory for device without a cache
0	1	0	Normal Memory for device with cache (cache write through).

1.4 How to Enable and Disable the MPU

By default when the processor is reset the MPU is switch off. In order to enable the MPU the enable bit in the System Control Register (see Figure 1) must be set to one. The following code sequence is used to enable the MPU:

```
mrc p15, #0, r0, c1, c0, #0
orr r0, r0, #1
dsb
mcr p15, #0, r0, c1, c0, #0
isb
```

Before the MPU is switch on it is important that all memory operation have completed in order to achieve this Data Synchronization Barrier (DSB) instruction is used immediately before the MCR instruction that enables the MPU. Immediately after the MPU has been enabled is it important that the instruction pipe line is clear to force all new instruction fetches to use the MPU region settings. To do this a Instruction Synchronization Barrier (ISB) instruction.

The code sequence to disable the MPU is similar; we just need to set the MPU enable bit to zero:

```
mrc p15, #0, r0, c1, c0, #0
bic r0, r0, #1
dsb
mcr p15, #0, r0, c1, c0, #0
isb
```

Again, the DSB and ISB instructions must be used to make sure all memory operation are completed and the instruction pipe line is cleared.

1.5 How to Setup an MPU Region

In order to setup an MPU, the following sequence is required:

1. Select the region to region to be programmed using the Region Select Register (see Figure 5).
2. Set the region start address using the MPU Region Base Address Register (see Figure 2).
3. Set the region access attributes using the MPU Region Access Control Register (see Figure 4).
4. Set the MPU region size and enable the MPU region using the MPU Region Size and Enable Register (see Figure 3).

1.5.1 Region Setup While MPU is Disabled

When setting up an MPU region when the MPU itself is disabled does not require any special considerations. The following is an example to setup MPU region '0' to for a typical 2MB flash region starting at address 0x00000000, access is set to read-only in all modes:

```
mov     r0,  #0                     ; select region '0'
mcr     p15, #0, r0, c6, c2, #0
movw    r0,  #0x0000                ; region base address
movt    r0,  #0x0000
mcr     p15, #0, r0, c6, c1, #0
movw    r0,  #0x0608                ; access normal(no-cache), all modes, read-only
mcr     p15, #0, r0, c6, c1, #4
mov     r0,  #0x29                  ; size (2MB) and enable
mcr     p15, #0, r0, c6, c1, #2
```

1.5.2 Region Setup While MPU is Enabled

If we are updating the MPU settings for a region when the MPU is enabled we need to be careful not to disturb the MPU region the code is currently executing from. This can be achieved by using a higher priority MPU region to protect the executing code (see section 2.6). We also need to disable the region before we modify the settings:

```
mov   r0,  #0                     ; select region '0'
mcr   p15, #0, r0, c6, c2, #0
mrc   p15, #0, r0, c6, c1, #2     ; disable region by clearing enable bit
bic   r0, r0, #1
mcr   p15, #0, r0, c6, c1, #2
movw  r0,  #0x0000                ; region base address
movt  r0,  #0x0000
mcr   p15, #0, r0, c6, c1, #0
movw  r0,  #0x0608                ; access normal(no-cache), all modes, read-only
mcr   p15, #0, r0, c6, c1, #4
mov   r0,  #0x29                  ; size (2MB) and enable
mcr   p15, #0, r0, c6, c1, #2
```

If modifying the setting of a MPU region used to access data (RAM) , for a region we having just been writing data too, we must make sure that all write operations have completed before we disable the region. This is achieved by adding a Data Synchronization Barrier (DSB) instruction just before the MCR instruction that disables the region:

```
mov   r0,  #1                     ; select region '1'
mcr   p15, #0, r0, c6, c2, #0
mrc   p15, #0, r0, c6, c1, #2     ; disable region by clearing enable bit
bic   r0, r0, #1
dsb
mcr   p15, #0, r0, c6, c1, #2
movw  r0,  #0x0000                ; region base address
movt  r0,  #0x0800
mcr   p15, #0, r0, c6, c1, #0
movw  r0,  #0x0308                ; access normal(no-cache), all modes
mcr   p15, #0, r0, c6, c1, #4
mov   r0,  #0x0F                  ; size (256kb) and enable
mcr   p15, #0, r0, c6, c1, #2
```

Conversely if updating the settings for a region we will be immediately executing instructions from, an Instruction Synchronization Barrier (ISB) instructions should be executed immediately after the region is enabled to clear the instruction pipe-line.

```
mov   r0,  #0                     ; select region '0'
mcr   p15, #0, r0, c6, c2, #0
mrc   p15, #0, r0, c6, c1, #2     ; disable region by clearing enable bit
bic   r0, r0, #1
mcr   p15, #0, r0, c6, c1, #2
movw  r0,  #0x0000                ; region base address
movt  r0,  #0x0000
mcr   p15, #0, r0, c6, c1, #0
movw  r0,  #0x0608                ; access normal(no-cache), all modes, read-only
mcr   p15, #0, r0, c6, c1, #4
mov   r0,  #0x29                  ; size (2MB) and enable
mcr   p15, #0, r0, c6, c1, #2
isb
```

Although not strictly necessary, it is recommended that the MPU region setup code be placed inside a critical section e.g. that the interrupts are disabled. This ensures that in a complex system no other code can change the MPU selection register before the programming sequence has been completed.

1.5.3 Region Setup An Optimised Version

With carful construction of the code an optimized version of the code to load the MPU settings can be used. Assuming:

- The MPU register settings are stored in an array containing the settings consecutive MPU regions.
- Interrupts are disabled.
- The code is executing from a higher priority MPU region (that is, can not be corrupted by the regions being programmed), see Section 1.6.

You can use the following code:

```
; Load pointer to MPU settings array
movw     r0,   <array pointer>
movt     r0,   <array pointer>
ldr      r12,   [r0]

; Load first MPU region, (MPU select register value)
Mov      r4,   #3

ldmia    r12!, {r1-r3}                ; Load MPU region settings from array
mcr      p15, #0, r4, c6, c2, #0       ; Select region
mcr      p15, #0, r1, c6, c1, #0       ; Base Address
mcr      p15, #0, r3, c6, c1, #4       ; Access Attributes
mcr      p15, #0, r2, c6, c1, #2       ; Size and Enable

add      r4, r4, #1                    ; next region
ldmia    r12!, {r1-r3}                 ; Load next MPU region settings from array
mcr      p15, #0, r4, c6, c2, #0       ; Select region
mcr      p15, #0, r1, c6, c1, #0       ; Base Address
mcr      p15, #0, r3, c6, c1, #4       ; Access Attributes
mcr      p15, #0, r2, c6, c1, #2       ; Size and Enable

…

add      r4, r4, #1                    ; last region
ldmia    r12!, {r1-r3}                 ; Load last MPU region settings from array
mcr      p15, #0, r4, c6, c2, #0       ; Select region
```

```
mcr      p15, #0, r1, c6, c1, #0       ; Base Address
mcr      p15, #0, r3, c6, c1, #4       ; Access Attributes
mcr      p15, #0, r2, c6, c1, #2       ; Size and Enable
```

See a complete example in Appendix A.

1.5.4 Region Address and Size

When setting the region start address there is one very important consideration:

The start address MUST be aligned to a multiple of its size.

It is highly recommended that this alignment be checked at runtime.

Alternatively the address alignment can be forced at compile time using the ‘DATA_ALIGN’ pragma:

```
define mainREG_TEST_STACK_SIZE_WORDS 128
#define mainREG_TEST_STACK_ALIGNMENT ( mainREG_TEST_STACK_SIZE_WORDS *
                                                    sizeof( portSTACK_TYPE ) )
#pragma DATA_SECTION(xRegTest1Stack, ".stackReg1" )
#pragma DATA_ALIGN ( xRegTest1Stack , mainREG_TEST_STACK_ALIGNMENT );
static portSTACK_TYPE xRegTest1Stack[ mainREG_TEST_STACK_SIZE_WORDS ];
```

1.6 MPU Default Configuration

For a typical device with 12 MPU regions each region will have an associated priority. Region 0 is the region with the lowest priority and region 12 the highest priory. The higher priority regions override the settings of the lower priority ones.

Typically the MPU will be setup with a number of the regions statically defined that do not change during the execution of the application. While other MPU regions are used dynamically as the application requires (dynamic regions are typically assigned on a per task basis).

In order not to adversely affect the system it is recommended that the dynamic regions use the lowest possible priority MPU regions.

It is recommended that the highest priority MPU region (region 11) be used to protect the peripheral system frame. This ensures that the system frame can’t be corrupted by incorrect MPU regions. Disruption to the system frame may causes interrupts and other system registers not to function correctly.

An example MPU region layout:

Table 10. Example MPU Region Layout

Region No	Description
0	Flash – Default settings for the Flash (access in all modes)
1	Ram – Default settings for the RAM (privileged access only)
2	Peripherals - Default settings for the peripherals (devices access in all modes)
3	Dynamic Region
4	…
5	…
6	…
7	…
8	Dynamic Region
9	RAM Kernel – privileged access only
10	Flash Kernel – privileged access only
11	System Frame – privileged access only

Region 11, protects the system frame. Region 10 protects the OS kernel, the code that changes the dynamic MPU settings should reside in this region. Regions 0 to 2 defined the default fallback MPU settings for the Flash, RAM and Peripherals. Regions 3 to 8 are used for the application dynamic regions.

Appendix A MPU Examples

A.1 Using FreeRTOS

This appendix shows a complete MPU implementation example for an operating system using FreeRTOS. The code is split over three files, ‘portmacro.h’ contains the definition of the MPU settings constants and the structure types (xMPU_REGION_REGISTERS and xMPU_SETTINGS) used to store the dynamic MPU configuration. ‘Port.c’ contains the ‘C’ portion of the code, of interest is the function ‘prvSetupDefaultMPU’ that sets up the default MPU configuration. Finally, the is ‘portASM.asm’ that contains the assembly language routines, particularly the routine ‘portRESTORE_CONTEXT’ that programs the dynamic MPU regions every time there is a task switch.

The MPU settings for the Task Control Block (TCB), this is defined in the FreeRTOS file ‘task.c’:

```
# typedef struct tskTaskControlBlock
{
     /*< Points to the location of the last item placed on the tasks stack.
     THIS MUST BE THE FIRST MEMBER OF THE TCB STRUCT. */
     volatile portSTACK_TYPE *pxTopOfStack;

     /*< The MPU settings are defined as part of the port layer.
     THIS MUST BE THE SECOND MEMBER OF THE TCB STRUCT. */
     #if ( portUSING_MPU_WRAPPERS == 1 )
          xMPU_SETTINGS xMPUSettings;
     #endif

     …
     …

} tskTCB;
```

The variable ‘pxCurrentTCB’ points to the TCB of the current task, and the second entry in the TCB is an array contains the register settings for the dynamic MPU regions on a per task basis. ‘portRESTORE_CONTEXT’ load the MPU settings from the array ‘xMPUSettings’ on every task (context) switch.

The complete FreeRTOS code including this example can be downloaded from:

ftp://ftp.ti.com/pub/tms470/CORTEX_R4_MPU_TI_CCS.zip.

A.2 Portmacro.h

```
#ifndef __PORTMACRO_H__
#define __PORTMACRO_H__
/*-----------------------------------------------------------
 * Port specific definitions.
 *
 * The settings in this file configure FreeRTOS correctly for the
 * given hardware and compiler.
 *
 * These settings should not be altered.
 *-----------------------------------------------------------
 */
/* Type definitions. */
#define portCHAR char
#define portFLOAT float
#define portDOUBLE double
#define portLONG long
#define portSHORT short
#define portSTACK_TYPE unsigned long
#define portBASE_TYPE long
```

```
#if (configUSE_16_BIT_TICKS == 1)
    typedef unsigned portSHORT portTickType;
    #define portMAX_DELAY (portTickType) 0xFFFF
#else
    typedef unsigned portLONG portTickType;
    #define portMAX_DELAY (portTickType) 0xFFFFFFFFF
#endif


/* Architecture specifics.  */
#define portSTACK_GROWTH    (-1)
#define portTICK_RATE_MS    ((portTickType) 1000 / configTICK_RATE_HZ)

#define portBYTE_ALIGNMENT  8

/* Critical section handling. */
#pragma SWI_ALIAS(vPortEnterCritical, 2)
extern void vPortEnterCritical( void );

#pragma SWI_ALIAS(vPortExitCritical, 3)
extern void vPortExitCritical( void );

#pragma SWI_ALIAS(vPortDisableInterrupts, 5)
extern void vPortDisableInterrupts( void );

#pragma SWI_ALIAS(vPortEnableInterrupts, 6)
extern void vPortEnableInterrupts( void );

#define portENTER_CRITICAL() vPortEnterCritical()
#define portEXIT_CRITICAL() vPortExitCritical()
#define portDISABLE_INTERRUPTS() vPortDisableInterrupts()
#define portENABLE_INTERRUPTS() vPortEnableInterrupts()

/* Scheduler utilities. */

#pragma SWI_ALIAS(vPortYield, 0)
extern void vPortYield( void );

#define portYIELD()            vPortYield()
#define portSYS_SSIR1_REG     ( * ( ( volatile unsigned long * ) 0xFFFFFFB0 ) )
#define portSYS_SSIR1_SSKEY                                    ( 0x7500UL )
#define portYIELD_WITHIN_API() { portSYS_SSIR1_REG = portSYS_SSIR1_SSKEY; ( void )
portSYS_SSIR1_REG; }
#define portYIELD_FROM_ISR( x ) if( x != pdFALSE ){ portSYS_SSIR1_REG = portSYS_SSIR1_SSKEY; (
void ) portSYS_SSIR1_REG; }

/* Floating Point Support */
#pragma SWI_ALIAS(vPortTaskUsesFPU, 4)
extern void vPortTaskUsesFPU(void);

/* Task function macros as described on the FreeRTOS.org WEB site. */
#define portTASK_FUNCTION(vFunction, pvParameters) void vFunction(void *pvParameters)
#define portTASK_FUNCTION_PROTO(vFunction, pvParameters) void vFunction(void *pvParameters)

/* MPU specific constants. */
#define portUSING_MPU_WRAPPERS                        1
#define portPRIVILEGE_BIT                             ( 0x80000000UL )

#define portMPU_REGION_READ_WRITE                     ( 0x03UL << 8UL )
#define portMPU_REGION_PRIVILEGED_READ_ONLY           ( 0x05UL << 8UL )
#define portMPU_REGION_READ_ONLY                      ( 0x06UL << 8UL )
#define portMPU_REGION_PRIVILEGED_READ_WRITE          ( 0x01UL << 8UL )

#define portMPU_REGION_STRONGLY_ORDERED               ( 0x00UL )
#define portMPU_REGION_DEVICE                         ( 0x01UL )
```

```
#define portMPU_REGION_CACHEABLE                        ( 0x02UL )
#define portMPU_REGION_CACHEABLE_BUFFERABLE             ( 0x03UL )
#define portMPU_REGION_NORMAL                           ( 0x08UL )
#define portMPU_REGION_EXECUTE_NEVER                    ( 0x01UL << 12UL )

#define portMPU_REGION_ENABLE                           ( 0x01UL )

/* MPU region sizes */
#define portMPU_SIZE_32B                                ( 0x04UL << 1UL )
#define portMPU_SIZE_64B                                ( 0x05UL << 1UL )
#define portMPU_SIZE_128B                               ( 0x06UL << 1UL )
#define portMPU_SIZE_256B                               ( 0x07UL << 1UL )
#define portMPU_SIZE_512B                               ( 0x08UL << 1UL )
#define portMPU_SIZE_1KB                                ( 0x09UL << 1UL )
#define portMPU_SIZE_2KB                                ( 0x0AUL << 1UL )
#define portMPU_SIZE_4KB                                ( 0x0BUL << 1UL )
#define portMPU_SIZE_8KB                                ( 0x0CUL << 1UL )
#define portMPU_SIZE_16KB                               ( 0x0DUL << 1UL )
#define portMPU_SIZE_32KB                               ( 0x0EUL << 1UL )
#define portMPU_SIZE_64KB                               ( 0x0FUL << 1UL )
#define portMPU_SIZE_128KB                              ( 0x10UL << 1UL )
#define portMPU_SIZE_256KB                              ( 0x11UL << 1UL )
#define portMPU_SIZE_512KB                              ( 0x12UL << 1UL )
#define portMPU_SIZE_1MB                                ( 0x13UL << 1UL )
#define portMPU_SIZE_2MB                                ( 0x14UL << 1UL )
#define portMPU_SIZE_4MB                                ( 0x15UL << 1UL )
#define portMPU_SIZE_8MB                                ( 0x16UL << 1UL )
#define portMPU_SIZE_16MB                               ( 0x17UL << 1UL )
#define portMPU_SIZE_32MB                               ( 0x18UL << 1UL )
#define portMPU_SIZE_64MB                               ( 0x19UL << 1UL )
#define portMPU_SIZE_128MB                              ( 0x1AUL << 1UL )
#define portMPU_SIZE_256MB                              ( 0x1BUL << 1UL )
#define portMPU_SIZE_512MB                              ( 0x1CUL << 1UL )
#define portMPU_SIZE_1GB                                ( 0x1DUL << 1UL )
#define portMPU_SIZE_2GB                                ( 0x1EUL << 1UL )
#define portMPU_SIZE_4GB                                ( 0x1FUL << 1UL )

/* Default MPU regions */
#define portUNPRIVILEGED_FLASH_REGION                    ( 0UL )
#define portDEFAULT_RAM_REGION                           ( 1UL )
#define portGENERAL_PERIPHERALS_REGION                   ( 2UL )
#define portSTACK_REGION                                 ( 3UL )
#define portFIRST_CONFIGURABLE_REGION                    ( 4UL )
#define portLAST_CONFIGURABLE_REGION                     ( 8UL )
#define portPRIVILEGED_RAM_REGION                        ( 9UL )
#define portPRIVILEGED_FLASH_REGION                      ( 10UL )
#define portPRIVILEGED_SYSTEM_REGION                     ( 11UL )
#define portNUM_CONFIGURABLE_REGIONS                     ( ( portLAST_CONFIGURABLE_REGION - \
portFIRST_CONFIGURABLE_REGION ) + 1 )
/* Plus one to make space for the stack region.         */
#define portTOTAL_NUM_REGIONS                            ( portNUM_CONFIGURABLE_REGIONS + 1 )
#define portSWITCH_TO_USER_MODE()                        asm(" CPS #0x10");

typedef struct MPU_REGION_REGISTERS
{
          unsigned ulRegionBaseAddress;
          unsigned ulRegionSize;
          unsigned ulRegionAttribute;
} xMPU_REGION_REGISTERS;

/* Plus 1 to create space for the stack region. */
typedef struct MPU_SETTINGS
{
          xMPU_REGION_REGISTERS xRegion[ portTOTAL_NUM_REGIONS ];
} xMPU_SETTINGS;
```

```
#endif /* __PORTMACRO_H__ */
```

A.3 Port.c

```
#define MPU_WRAPPERS_INCLUDED_FROM_API_FILE

/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"

#undef MPU_WRAPPERS_INCLUDED_FROM_API_FILE

/*-----------------------------------------------------------*/

/* Registers required to configure the RTI. */
#define portRTI_GCTRL_REG                   ( * ( ( volatile unsigned long * ) 0xFFFFFC00 ) )
#define portRTI_TBCTRL_REG                  ( * ( ( volatile unsigned long * ) 0xFFFFFC04 ) )
#define portRTI_COMPCTRL_REG                ( * ( ( volatile unsigned long * ) 0xFFFFFC0C ) )
#define portRTI_CNT0_FRC0_REG               ( * ( ( volatile unsigned long * ) 0xFFFFFC10 ) )
#define portRTI_CNT0_UC0_REG                ( * ( ( volatile unsigned long * ) 0xFFFFFC14 ) )
#define portRTI_CNT0_CPUC0_REG              ( * ( ( volatile unsigned long * ) 0xFFFFFC18 ) )
#define portRTI_CNT0_COMP0_REG              ( * ( ( volatile unsigned long * ) 0xFFFFFC50 ) )
#define portRTI_CNT0_UDCP0_REG              ( * ( ( volatile unsigned long * ) 0xFFFFFC54 ) )
#define portRTI_SETINTENA_REG               ( * ( ( volatile unsigned long * ) 0xFFFFFC80 ) )
#define portRTI_CLEARINTENA_REG             ( * ( ( volatile unsigned long * ) 0xFFFFFC84 ) )
#define portRTI_INTFLAG_REG                 ( * ( ( volatile unsigned long * ) 0xFFFFFC88 ) )

/* Constants required to set up the initial stack of each task. */
#define portINITIAL_SPSR_IF_PRIVILEGED      ( ( portSTACK_TYPE ) 0x1F )
#define portINITIAL_SPSR_IF_UNPRIVILEGED    ( ( portSTACK_TYPE ) 0x10 )
#define portINITIAL_FPSCR                   ( ( portSTACK_TYPE ) 0x00 )
#define portINSTRUCTION_SIZE                ( ( portSTACK_TYPE ) 0x04 )
#define portTHUMB_MODE_BIT                  ( ( portSTACK_TYPE ) 0x20 )

/* The number of words on the stack frame between the saved Top Of Stack and
R0 (in which the parameters are passed. */
#define portSPACE_BETWEEN_TOS_AND_PARAMETERS ( 12 )

/*-----------------------------------------------------------*/

/* vPortStartFirstSTask() is defined in portASM.asm */
extern void vPortStartFirstTask( void );

/* MPU access routines defined in portASM.asm */
extern void prvMpuEnable(void);
extern void prvMpuDisable(void);
extern void prvMpuSetRegion(unsigned region, unsigned base, unsigned size, unsigned access);

/*-----------------------------------------------------------*/

#pragma SET_DATA_SECTION(".kernelBSS")

/* Count of the critical section nesting depth. */
unsigned portLONG ulCriticalNesting = 9999;

/*-----------------------------------------------------------*/


/* Saved as part of the task context. Set to pdFALSE if the task does not
require an FPU context. */
unsigned long ulTaskHasFPUContext = 0;

/*-----------------------------------------------------------*/

#pragma SET_CODE_SECTION(".kernelTEXT")
/*
```

```
* See header file for description.
*/
portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void
*pvParameters, portBASE_TYPE xRunPrivileged )
{
portSTACK_TYPE *pxOriginalTOS;

    pxOriginalTOS = pxTopOfStack;

    #if __TI_VFP_SUPPORT__
    {
        /* Ensure the stack is correctly aligned on exit. */
        pxTopOfStack--;
    }
    #endif
    /* Setup the initial stack of the task. The stack is set exactly as
    expected by the portRESTORE_CONTEXT() macro. */

    /* First on the stack is the return address - which is the start of the as
    the task has not executed yet. The offset is added to make the return
    address appear as it would within an IRQ ISR. */
    *pxTopOfStack = ( portSTACK_TYPE ) pxCode + portINSTRUCTION_SIZE;
    pxTopOfStack--;

    *pxTopOfStack = ( portSTACK_TYPE ) 0x00000000; /* R14 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) pxOriginalTOS; /* Stack used when task starts     goes in
R13. */
    pxTopOfStack--;

    #ifdef portPRELOAD_TASK_REGISTERS
    {
        *pxTopOfStack = ( portSTACK_TYPE ) 0x12121212;    /* R12 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x11111111;    /* R11 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x10101010;    /* R10 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x09090909;    /* R9 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x08080808;    /* R8 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x07070707;    /* R7 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x06060606;    /* R6 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x05050505;    /* R5 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x04040404;    /* R4 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x03030303;    /* R3 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x02020202;    /* R2 */
        pxTopOfStack--;
        *pxTopOfStack = ( portSTACK_TYPE ) 0x01010101;    /* R1 */
        pxTopOfStack--;
    }
    #else
    {
        pxTopOfStack -= portSPACE_BETWEEN_TOS_AND_PARAMETERS;
    }
    #else

/* Function parameters are passed in R0. */
*pxTopOfStack = ( portSTACK_TYPE ) pvParameters; /* R0 */
```

```
    pxTopOfStack--;

    /* Set the status register for system or user mode, with interrupts enabled. */
    if( xRunPrivileged == pdTRUE )
    {
        *pxTopOfStack = (portSTACK_TYPE) ((_get_CPSR() & ~0xFF) |
                                                    portINITIAL_SPSR_IF_PRIVILEGED);
    }
    else
    {
        *pxTopOfStack = (portSTACK_TYPE) ((_get_CPSR() & ~0xFF) |
    portINITIAL_SPSR_IF_UNPRIVILEGED);

    }

    if( ( ( unsigned long ) pxCode & 0x01UL ) != 0x00 )
    {
        /* The task will start in thumb mode. */
        *pxTopOfStack |= portTHUMB_MODE_BIT;

    }

    #ifdef __TI_VFP_SUPPORT__
    {
        pxTopOfStack--;

        /* The last thing on the stack is the tasks ulUsingFPU value,
        which by default is set to indicate that the stack frame does
        not include FPU registers. */
        *pxTopOfStack = pdFALSE;

    }
    #endif

    return pxTopOfStack;
    }

    /*-----------------------------------------------------------------------*/

    static unsigned long prvGetMPURegionSizeSetting( unsigned long ulActualSizeInBytes )
    {
    unsigned long ulRegionSize, ulReturnValue = 4;

        /* 32 is the smallest region size, 31 is the largest valid value for
        ulReturnValue. */
        for( ulRegionSize = 32UL; ulReturnValue < 31UL; ( ulRegionSize << 1UL ) )
        {
            if( ulActualSizeInBytes  <= ulRegionSize )
       {
          break;
       }
       else
       {
          ulReturnValue++;
       }
    }
       /* Shift the code by one before returning so it can be written directly
       into the the correct bit position of the attribute register. */
       return ulReturnValue << 1UL;
    }

    /*-----------------------------------------------------------------------*/

    void vPortStoreTaskMPUSettings( xMPU_SETTINGS *xMPUSettings, const struct xMEMORY_REGION * const
    xRegions, portSTACK_TYPE *pxBottomOfStack, unsigned short usStackDepth )
    {
```

```
long lIndex;
unsigned long ul;

   if( xRegions == NULL )

   {
      /* No MPU regions are specified so allow access to all of the RAM. */
      xMPUSettings->xRegion[0].ulRegionBaseAddress     = 0x08000000;
      xMPUSettings->xRegion[0].ulRegionSize            = portMPU_SIZE_256KB
                                                            | portMPU_REGION_ENABLE;
      xMPUSettings->xRegion[0].ulRegionAttribute       = portMPU_REGION_READ_WRITE
                                                            | portMPU_REGION_NORMAL;

   /* Invalidate all other regions. */
   for( ul = 1; ul < portNUM_CONFIGURABLE_REGIONS; ul++ )
   {
      xMPUSettings->xRegion[ ul ].ulRegionBaseAddress = 0x00000000UL;
      xMPUSettings->xRegion[ ul ].ulRegionSize = 0UL;
      xMPUSettings->xRegion[ ul ].ulRegionAttribute = 0UL;
   }
}
else
{
   /* This function is called automatically when the task is created - in
   which case the stack region parameters will be valid. At all other
   times the stack parameters will not be valid and it is assumed that the
   stack region has already been configured. */
   if( usStackDepth > 0 )
{
      /* Define the region that allows access to the stack. */
      xMPUSettings->xRegion[0].ulRegionBaseAddress = (unsigned)pxBottomOfStack;
      xMPUSettings->xRegion[0].ulRegionSize        = prvGetMPURegionSizeSetting(
       (unsigned long)usStackDepth * (unsigned long) sizeof(portSTACK_TYPE) )
                                                        | portMPU_REGION_ENABLE;
      xMPUSettings->xRegion[0].ulRegionAttribute   = portMPU_REGION_READ_WRITE
                                                        | portMPU_REGION_NORMAL;
}
lIndex = 0;

for( ul = 1; ul <=  portNUM_CONFIGURABLE_REGIONS; ul++ )
{
if( ( xRegions[ lIndex ] ).ulLengthInBytes > 0UL ))
          {
            /* Translate the generic region definition contained in
            xRegions into the R4 specific MPU settings that are then
            stored in xMPUSettings. */
            xMPUSettings-
>xRegion[ul].ulRegionBaseAddress = (unsigned long) Regions[lIndex].pvBaseAddress;
            xMPUSettings-
>xRegion[ul].ulRegionSize        = prvGetMPURegionSizeSetting( xRegions[ lIndex ].ulLengthInBytes
) | portMPU_REGION_ENABLE;
            xMPUSettings->xRegion[ul].ulRegionAttribute   = xRegions[ lIndex ].ulParameters;
        {
        else
        {
            /* Invalidate the region. */
            xMPUSettings->xRegion[ ul ].ulRegionBaseAddress = 0x00000000UL;
            xMPUSettings->xRegion[ ul ].ulRegionSize        = 0UL;
            xMPUSettings->xRegion[ ul ].ulRegionAttribute   = 0UL;
        }
        lIndex++;
      }
   }
}

/*-----------------------------------------------------------------------------*/
```

```
static void prvSetupDefaultMPU( void )
{
     /* make sure MPU is disabled */
     prvMpuDisable();

     /* First setup the entire flash for unprivileged read only access. */
     prvMpuSetRegion(portUNPRIVILEGED_FLASH_REGION, 0x00000000, portMPU_SIZE_4MB
                  | portMPU_REGION_ENABLE, portMPU_REGION_READ_ONLY
                  | portMPU_REGION_NORMAL);

     /* Setup the default region. This is where the kernel data
     is placed. */
     prvMpuSetRegion(portDEFAULT_RAM_REGION, 0x08000000, portMPU_SIZE_256KB
                  | portMPU_REGION_ENABLE, portMPU_REGION_PRIVILEGED_READ_WRITE
                  | portMPU_REGION_NORMAL);

     /* Default peripherals setup */
     prvMpuSetRegion(portGENERAL_PERIPHERALS_REGION, 0xFC000000, portMPU_SIZE_64MB
                  | portMPU_REGION_ENABLE, portMPU_REGION_READ_WRITE
                  | portMPU_REGION_EXECUTE_NEVER | portMPU_REGION_DEVICE);

     /* Setup the first 4K for privileged only access. This is where the kernel RAM is
     placed. */
     prvMpuSetRegion(portPRIVILEGED_RAM_REGION, 0x08000000, portMPU_SIZE_4KB
                  | portMPU_REGION_ENABLE, portMPU_REGION_PRIVILEGED_READ_ONLY
                  | portMPU_REGION_NORMAL);

     /* Setup the first 32K for privileged only access. This is where the kernel code     is
placed. */
     prvMpuSetRegion(portPRIVILEGED_FLASH_REGION, 0x00000000, portMPU_SIZE_32KB
                  | portMPU_REGION_ENABLE, portMPU_REGION_PRIVILEGED_READ_ONLY
                  | portMPU_REGION_NORMAL);

     /* Set System Frame to Priviledged Access Only */
     prvMpuSetRegion(portPRIVILEGED_SYSTEM_REGION, 0xFFF80000, portMPU_SIZE_512KB
                  | portMPU_REGION_ENABLE, portMPU_REGION_PRIVILEGED_READ_WRITE
                  | portMPU_REGION_EXECUTE_NEVER
                  | portMPU_REGION_STRONLY_ORDERED);
     /* Enable MPU */
     prvMpuEnable();
}
/*-----------------------------------------------------------*/

static void prvSetupTimerInterrupt(void)
{
     /* Disable timer 0. */
     portRTI_GCTRL_REG &= 0xFFFFFFFEUL;

     /* Use the internal counter. */
     portRTI_TBCTRL_REG = 0x00000000U;

     /* COMPSEL0 will use the RTIFRC0 counter. */
     portRTI_COMPCTRL_REG = 0x00000000U;

     /* Initialise the counter and the prescale counter registers. */
     portRTI_CNT0_UC0_REG = 0x00000000U;
     portRTI_CNT0_FRC0_REG = 0x00000000U;

     /* Set Prescalar for RTI clock. */
     portRTI_CNT0_CPUC0_REG = 0x00000001U;
     portRTI_CNT0_COMP0_REG = ( configCPU_CLOCK_HZ / 2 ) / configTICK_RATE_HZ;
     portRTI_CNT0_UDCP0_REG = ( configCPU_CLOCK_HZ / 2 ) / configTICK_RATE_HZ;

     /* Clear interrupts. */
     portRTI_INTFLAG_REG = 0x0007000FU;
```

```
	portRTI_CLEARINTENA_REG = 0x00070F0FU;

	/* Enable the compare 0 interrupt. */
	portRTI_SETINTENA_REG = 0x00000001U;
	portRTI_GCTRL_REG |= 0x00000001U;

}
/*-----------------------------------------------------------*/
/*
 * See header file for description.
 */
portBASE_TYPE xPortStartScheduler(void)
{
	/* Configure the regions in the MPU that are common to all tasks. */
	prvSetupDefaultMPU();

	/* Start the timer that generates the tick ISR. */
	prvSetupTimerInterrupt();

	/* Reset the critical section nesting count read to execute the first task. */
	ulCriticalNesting = 0;

	/* Start the first task. This is done from portASM.asm as ARM mode must be
	used. */

vPortStartFirstTask();
	/* Should not get here! */
	return pdFAIL;
}
/*-----------------------------------------------------------*/
/*
 * See header file for description.
 */
void vPortEndScheduler(void)
{
	/* It is unlikely that the port will require this function as there
	is nothing to return to. */
}
/*-----------------------------------------------------------*/

#if configUSE_PREEMPTION == 0

/* The cooperative scheduler requires a normal IRQ service routine to
 * simply increment the system tick. */
__interrupt void vPortNonPreemptiveTick( void )
{

	/* clear clock interrupt flag */
	RTI->INTFLAG = 0x00000001;

	/* Increment the tick count - this may make a delaying task ready
	to run - but a context switch is not performed. */
	vTaskIncrementTick();
}

	#else

/*
*************************************************************************
* The preemptive scheduler ISR is written in assembler and can be found
* in the portASM.asm file. This will only get used if portUSE_PREEMPTION
* is set to 1 in portmacro.h
*************************************************************************
*/
void vPortPreemptiveTick( void );
```

```
#endif
/*-----------------------------------------------------------*/
```

A.4 PortASM.asm

```
;/*-----------------------------------------------------------*/
;
; Save Task Context
;
portSAVE_CONTEXT .macro
        DSB

        ; Push R0 as we are going to use it
        STMDB  SP!, {R0}

        ; Set  R0 to point to the task stack pointer.
        STMDB  SP,{SP}^
        SUB    SP, SP, #4
        LDMIA  SP!,{R0}

        ; Push the return address onto the stack.
        STMDB  R0!, {LR}

        ; Now LR has been saved, it can be used instead of R0.
        MOV    LR, R0

        ; Pop R0 so it can be saved onto the task stack.
        LDMIA  SP!, {R0}

        ; Push all the system mode registers onto the task stack.
        STMDB  LR,{R0-LR}^
        SUB    LR, LR, #60

        ; Push the SPSR onto the task stack.
        MRS    R0, SPSR
        STMDB  LR!, {R0}

.if (__TI_VFP_SUPPORT__)
        ; Determine if the task maintains an FPU context.
        MOVW   R0, ulTaskHasFPUContext
        MOVT   R0, ulTaskHasFPUContext
        LDR    R0, [R0]

        ; Test the flag
        CMP    R0, #0

        ; If the task is not using a floating point context then skip the
        ; saving of the FPU registers.
        BEQ     PC+3
        FSTMDBD LR!, {D0-D15}
        FMRX    R1, FPSCR
        STMFD   LR!, {R1}

        ; Save the flag
        STMDB  LR!, {R0}
.endif

        ; Store the new top of stack for the task.
        MOVW   R0, pxCurrentTCB
        MOVT   R0, pxCurrentTCB
        LDR    R0, [R0]
        STR    LR, [R0]
       .endm

;/*-----------------------------------------------------------*/
```

```
;
; Restore Task Context
;
portRESTORE_CONTEXT .macro
       MOVW    R0, pxCurrentTCB
       MOVT    R0, pxCurrentTCB
       LDR     R0, [R0]

       ; task stack MPU region
       mov     r4, #3
       add     r12, r0, #4            ; point to regions in TCB
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable

       ; five dynamic MPU per task
       add     r4, r4, #1
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable
       add     r4, r4, #1
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable
       add     r4, r4, #1
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable
       add     r4, r4, #1
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable
       add     r4, r4, #1
       ldmia   r12!, {r1-r3}
       mcr     p15, #0, r4, c6, c2, #0 ; Select region
       mcr     p15, #0, r1, c6, c1, #0 ; Base Address
       mcr     p15, #0, r3, c6, c1, #4 ; Access Attributes
       mcr     p15, #0, r2, c6, c1, #2 ; Size and Enable

       LDR LR, [R0]

.if (__TI_VFP_SUPPORT__)
       ; The floating point context flag is the first thing on the stack.
       MOVW R0, ulTaskHasFPUContext
       MOVT R0, ulTaskHasFPUContext
       LDMFD LR!, {R1}
       STR R1, [R0]

     ; Test the flag
     CMP       R1, #0

     ; If the task is not using a floating point context then skip the
     ; VFP register loads.
     BEQ       PC+3

     ; Restore the floating point context.
```

```
        LDMFD     LR!, {R0}
        FLDMIAD   LR!, {D0-D15}
        FMXR      FPSCR, R0
    .endif

        ; Get the SPSR from the stack.
        LDMFD     LR!, {R0}
        MSR       SPSR_CSXF, R0

        ; Restore all system mode registers for the task.
        LDMFD     LR, {R0-R14}^

        ; Restore the return address.
        LDR       LR, [LR, #+60]
        DSB

        ; And return - correcting the offset in the LR to obtain the
        ; correct address.
        SUBS      PC, LR, #4
        .endm

    ;/*-----------------------------------------------------------*/
    ; Start the first task by restoring its context.

            .def vPortStartFirstTask

    vPortStartFirstTask:

            portRESTORE_CONTEXT

    ;/*-----------------------------------------------------------*/
    ; Yield to another task.

            .def vPortYieldProcessor

    swiPortYield:
        ; Restore stack and LR before calling vPortYieldProcessor
        Ldmfd    sp!, {r12,lr}

    vPortYieldProcessor:
        ; Within an IRQ ISR the link register has an offset from the true return
        ; address. SWI doesn't do this. Add the offset manually so the ISR
        ; return code can be used.
        ADD      LR, LR, #4

        ; First save the context of the current task.
        portSAVE_CONTEXT

        ; Select the next task to execute. */
        BL       vTaskSwitchContext

        ; Restore the context of the task selected to execute.

    ;/*-----------------------------------------------------------*/
    ; Yield to another task from within the FreeRTOS API

            .def vPortYeildWithinAPI

    vPortYeildWithinAPI:
            ; Save the context of the current task.

            portSAVE_CONTEXT
            ; Clear SSI flag.
            MOVW    R0, #0xFFF4
            MOVT    R0, #0xFFFF
            LDR     R0, [R0]
```

```
        ; Select the next task to execute. */
        BL      vTaskSwitchContext

        ; Restore the context of the task selected to execute.
        portRESTORE_CONTEXT

;/*-----------------------------------------------------------*/
; Preemptive Tick

        .def vPortPreemptiveTick

vPortPreemptiveTick:
        ; Save the context of the current task.
        portSAVE_CONTEXT

        ; Clear interrupt flag
        MOVW    R0, #0xFC88
        MOVT    R0, #0xFFFF
        MOV     R1, #1
        STR     R1, [R0]

        ; Increment the tick count, making any adjustments to the blocked lists
        ; that may be necessary.
        BL        vTaskIncrementTick

        ; Select the next task to execute.
        BL        vTaskSwitchContext

        ; Restore the context of the task selected to execute.
        portRESTORE_CONTEXT

;---------------------------------------------------------------------------
; SWI Handler, interface to Protected Mode Functions

        .def     vPortSWI

vPortSWI:
        stmfd   sp!, {r12,lr}
      .if .TMS470_LITTLE
        Mrs     r12, spsr
        ands    r12, r12, #0x20
        ldrbne  r12, [lr, #-2]
        ldrbeq  r12, [lr, #-4]
      .else
        Ldrb    r12, [lr, #-1]
      .endif
        Ldr     r14, table
        ldr     r12, [r14, r12, asl #2]
        blx     r12
        ldmfd sp!, {r12,pc}^

table:
        .word   jumpTable
jumpTable:
        .word   swiPortYield                ; 0 - vPortYieldProcessor
        .word   swiRaisePrivilege           ; 1 - Raise Priviledge
        .word   swiPortEnterCritical        ; 2 - vPortEnterCritical
        .word   swiPortExitCritical         ; 3 - vPortExitCritical
        .word   swiPortTaskUsesFPU          ; 4 - vPortTaskUsesFPU
        .word   swiPortDisableInterrupts    ; 5 - vPortDisableInterrupts
        .word   swiPortEnableInterrupts     ; 6 - vPortEnableInterrupts

;---------------------------------------------------------------------------
; vPortDisableInterrupts
swiPortDisableInterrupts:
```

```
        mrs     r0, SPSR
        orr     r0, r0, #0x80
        msr     SPSR_c, r0
        bx      r14

;-------------------------------------------------------------------------------
; vPortEnableInterrupts

swiPortEnableInterrupts:

mrs     r0, SPSR
        bic     r0, r0, #0x80
        msr     SPSR_c, r0
        bx      r14

;-------------------------------------------------------------------------------
; vPortTaskUsesFPU

swiPortTaskUsesFPU:
        movw    r12, ulTaskHasFPUContext
        movt    r12, ulTaskHasFPUContext
        mov     r0, #1
        str     r0, [r12]
        mov     r0, #0
        fmxr    FPSCR, r0
        bx      r14

;-------------------------------------------------------------------------------
; prvRaisePrivilege

; Must return zero in R0 if caller was in user mode

swiRaisePrivilege:
        mrs      r12, spsr
        ands     r0, r12, #0x0F ; return value
        orreq    r12, r12, #0x1F
        msreq    spsr_c, r12
        bx       r14

;-------------------------------------------------------------------------------
; vPortEnterCritical

swiPortEnterCritical:
        mrs      r0, SPSR
        orr      r0, r0, #0x80
        msr      SPSR_c, r0
        movw     r0, ulCriticalNesting
        movt     r0, ulCriticalNesting
        ldr      r12, [r0]
        add      r12, r12, #1
        str      r12, [r0]
        bx       r14

;-------------------------------------------------------------------------------
; vPortExitCritical

swiPortExitCritical:
        movw     r0, ulCriticalNesting
        movt     r0, ulCriticalNesting
        ldr      r12, [r0]
        cmp      r12, #0
        bxeq     r14
        subs     r12, r12, #1
        str      r12, [r0]
        bxne     r14
        mrs      r0, SPSR
```

```
        bic     r0, r0, #0x80
        msr     SPSR_c, r0
        bx      r14

;-------------------------------------------------------------------------------
; SetRegion

.def prvMpuSetRegion

; void _mpuSetRegion(unsigned region, unsigned base, unsigned size, unsigned access);

prvMpuSetRegion:
      and       r0, r0, #15 ; select region
      mcr       p15, #0, r0, c6, c2, #0
      mcr       p15, #0, r1, c6, c1, #0 ; Base Address
      mcr       p15, #0, r3, c6, c1, #4 ; Access Attributes
      mcr       p15, #0, r2, c6, c1, #2 ; Size and Enable
      bx        lr

;-------------------------------------------------------------------------------
; Enable Mpu

      .def prvMpuEnable

prvMpuEnable:
      mrc       p15, #0, r0, c1, c0, #0
      orr       r0, r0, #1
      dsb
      mcr       p15, #0, r0, c1, c0, #0
      isb
      bx        lr

;-------------------------------------------------------------------------------
; Disable Mpu

      .def prvMpuDisable
prvMpuDisable:
      mrc       p15, #0, r0, c1, c0, #0
      bic       r0, r0, #1
      dsb
      mcr       p15, #0, r0, c1, c0, #0
      isb
      bx        lr
;-------------------------------------------------------------------------------
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265