Felix Daners Engineering

FreeRTOS Port for Renesas R32C/111 IAR Platform

Document Information

Document State	☐ Being Processed ☐ Submitted ☒ Accepted
Document Version	1.2
Creation Date	15. September 2009
Author	Felix Daners, Felix Daners Engineering
Customer	
Customer Responsible Person	
Project Name	
Customer Project Number	
Subject	FreeRTOS Port
Last Saved Date / File	15/09/2009 / FreeRTOS_IAR_R32C111_v1.2.doc
Number of Pages Following this Page	24

Distribution List

Customer	
Felix Daners Engineering	F. Daners

Document Approvals

Felix Daners Engineering	Date	Signature
	Date	Signature
Customer	Date	Signature
	Date	Signature

Felix Daners Engineering

Speerstrasse 32
CH-8200 Schaffhausen
T +41 (0)52 624 92 32
F +41 (0)52 624 92 31
E f.daners@swissworld.com

Document Disclaimer

While every reasonable precaution has been taken in the preparation of this document, neither the author nor Felix Daners Engineering assumes responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.
The information contained in this document is believed to be accurate. However, no guarantee is provided. Use this information at your own risk.

Document History

Version	Date	Changed / Reason for Change	Name
1.1	09/09/09	New Document	Felix Daners
1.2	15/09/09	corrected typos, Figure 4, Figure 6, assembler `portTimerB0Interrupt` (`FCLR B` deleted), description of demo application with more details	Felix Daners

Abstract

FreeRTOS™ is a portable, open source, mini Real Time Kernel - a free to download and royalty free RTOS that can be used in commercial applications. In this document a port to the R32C/111 IAR Platform is introduced.

FreeRTOS comes in three flavours:

The FreeRTOS source code is licensed by the GNU General Public License (GPL) with an exception. The exceptions permits the source code of applications that use FreeRTOS solely through the API published on the FreeRTOS.org WEB site to remain closed source, thus permitting the use of FreeRTOS in commercial applications without necessitating that the whole application be open sourced. The exception should only be used if you wish to combine FreeRTOS with a proprietary product and you comply with the terms stated in the exception itself.
OpenRTOS is a commercially licensed version of FreeRTOS.org. The OpenRTOS license does not contain any references to the GPL.
SafeRTOS is a derivative version of FreeRTOS.org that has been analyzed, documented and tested to meet the stringent requirements of the IEC 61508 safety standard. Complete safety lifecycle documentation artefacts have been created and independently audited to verify IEC 61508 SIL 3 conformance.

For further licensing information refer to the FreeRTOS WEB site www.freertos.org.

The contributed code included with this port is released under a BSD license.

Table of Content

List of Figures

References

[1] FreeRTOS, OpenRTOS V5.2.0 (www.freertos.org)

[2] R32C/111 Group Hardware Manual, RENESAS MCU M16C FAMILY / R32C/100 SERIES, Rev.1.00, Revision Date: Dec 12, 2008

[3] R32C IAR Assembler Reference Guide for the Renesas R32C/100 Microcomputer Family, First edition: April 2007

[4] IAR C/C++ Compiler Reference Guide for the Renesas R32C/100 Microcomputer Family, Second edition: January 2009

[5] R32C/100 SERIES SOFTWARE MANUAL RENESAS MCU M16C FAMILY / R32C/100 SERIES, Rev.1.00, Revision Date: Feb 20, 2009

[6] EVBR32C111-Carrier-USB USER MANUAL, © 2008 by Glyn GmbH & Co KG, Microcontroller Group

[7] software terminal emulator Tera Term, http://en.sourceforge.jp/projects/ttssh2

Compiler/Assembler Versions:

IAR Assembler for Renesas R32C 1.30A (1.30.1.4)

IAR C Compiler for Renesas R32C 1.30B (1.30.2.4)

IAR XLINK 4.61J (4.61.10.0)

Renesas R32C Simulator Debugger for HEW that comes with the C compiler package NC100 v101r00. To use this simulator , the timer interrupt for the OS timer needs to be set up as Vector 21, priority 4 and 1ms period.

Hardware Platform

The demo program runs on EVBR32C111-Carrier-USB by Glyn GmbH & Co KG, Microcontroller Group.

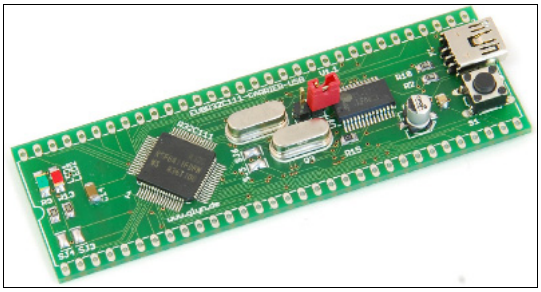

Figure 1 R32C/111-Starterkit (080928-91)

R32C/111-Starterkit (080928-91)

Distributor:

Elektor-Verlag GmbH
Systerfeldstrasse 25
52072 Aachen, Germany

Abbreviations

TCB Task Control Block

RTOS Real Time Operating System

1 Introduction

This port includes the following features:

- Separate Interrupt Stack (ISTACK), task stacks need not to save space for interrupt handlers.
- Interrupt nesting model that allows for interrupts running without RTOS interaction and a set of priority levels of RTOS handled interrupt service routines that also may nest.
- Delayed task switch until last interrupt nesting level is about to return.
- Simple compile time registration of custom interrupt service routines.
- Support for UART 0 to 7 on R32C/111 chips.

2 Task Stack Layout and Creation

Figure 2 shows the layout of a task stack. When a task is created, the function `xTaskCreate` calls `pxPortInitialiseStack`.

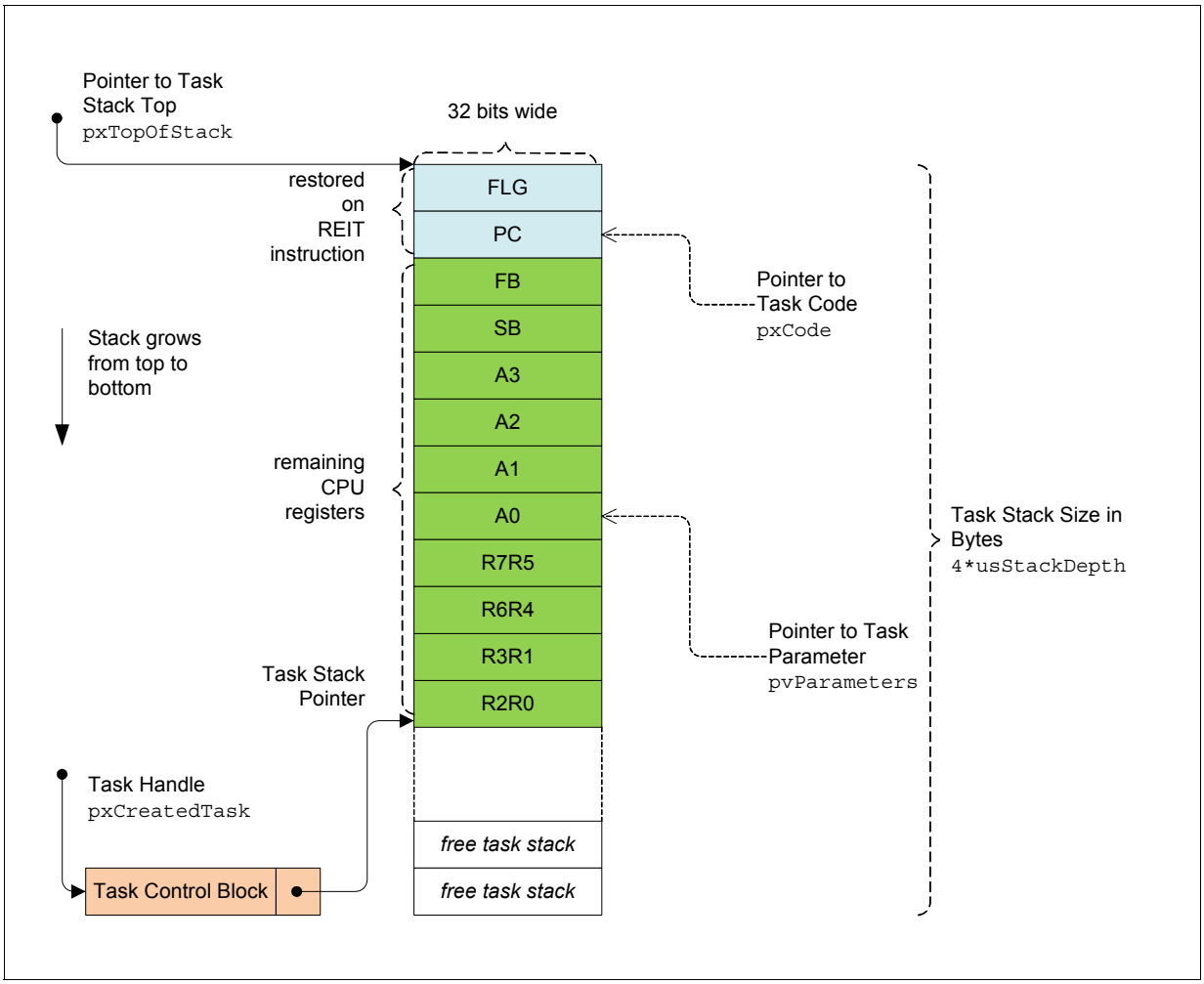

Figure 2 Task stack layout

`xTaskCreate` allocates memory for the task control block and the task stack. It then initializes both memory blocks and adds the newly created task control block to the appropriate task list.

```
portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack,
                                       pdTASK_CODE pxCode,
                                       pdTASK_PARAM pvParameters )
{
  /* select user stack, enable interrupt, set interrupt level to kernel */
  portSTACK_TYPE flag = 0x0040 | 0x0080 | (configKERNEL_INTERRUPT_PRIORITY<<12);

  pxTopOfStack--;

  /* | FLG          | */
  *pxTopOfStack-- = flag;

  /* | PC           | */
  *pxTopOfStack-- = (portSTACK_TYPE)((unsigned portLONG)pxCode );

  /* | FB                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)0xFBFBFBFB;

  /* | SB                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)0x3B3B3B3B;

  /* | A3                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)0xA3A3A3A3;

  /* | A2                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)0xA2A2A2A2;

  /* | A1                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)0xA1A1A1A1;

  /* | A0                                | */
  *pxTopOfStack-- = (portSTACK_TYPE)((unsigned portLONG)pvParameters);

  /* | R7R5                              | */
  *pxTopOfStack-- = (portSTACK_TYPE)0x77775555;

  /* | R6R4                              | */
  *pxTopOfStack-- = (portSTACK_TYPE)0x66664444;

  /* | R3R1                              | */
  *pxTopOfStack-- = (portSTACK_TYPE)0x33331111;

  /* | R2R0                              | */
  *pxTopOfStack   = (portSTACK_TYPE)0x22220000;

   return pxTopOfStack;
}
```

3 Task Switching Primitives

Task switching is done in five steps:

- Execute yield interrupt sequence. See `portYIELD()` and Figure 3.
- Save the context of the interrupted running task. See Figure 4.
- Select the new task to run, make it the new current task.
  Done by calling `vTaskSwitchContext()`.
- Restore the context of the new current task.
- Return from yield interrupt by executing a `REIT` instruction.

Task switching is initiated by calling `portYIELD()`. This macro uses an intrinsic function to fire the software interrupt vector `portYIELD_VECTOR`. `portYIELD_VECTOR` is defined as vector 1. On the R32C microcontroller, software interrupts are non-maskable.

```
#define portYIELD_VECTOR      1

/* this is non maskable! always yields even if I flag cleared */
#define portYIELD()                { __software_interrupt( portYIELD_VECTOR ); }
```

The R32C interrupt execution sequence is as follows:

- The CPU acknowledges an interrupt request to obtain the interrupt information (the interrupt number and the interrupt request level) from the interrupt controller. Then the IR bit of the corresponding interrupt is set to 0 (no interrupt requested).
- The flag register (FLG), prior to the interrupt sequence, is saved to a temporary register in the CPU.
- The following bits in the flag register (FLG) become 0:
  - The I flag (interrupt enable flag): interrupt disabled
  - The D flag (debug flag): single-step interrupt disabled
  - The U flag (stack pointer select flag): ISP selected
- The contents of the temporary register (1) in the CPU is saved to the stack area; or to the save flag register (SVF) in case of the fast interrupt.
- The contents of the program counter (PC) is saved to the stack; or to the save PC register (SVP) in case of the fast interrupt.
- The interrupt request level of the accepted interrupt is set in the IPL (processor interrupt priority level).
- The corresponding interrupt vector is read from the interrupt vector table.
- This interrupt vector is stored into the program counter (PC).

The handler for the interrupt that is called from `portYIELD()` expects the U flag to be cleared during interrupt execution sequence. Therefore software interrupts with vectors 128-255 are not allowed as yield interrupt.

```
#if portYIELD_VECTOR > 127
 #error "portYIELD_VECTOR must be < 128 (we expect FLG, PCH, PCM, PCL on ISTACK)"
#endif
```

From the interrupt execution sequence described above, at entry of yield interrupt handler we find the situation on task and interrupt stack as depicted in Figure 3.

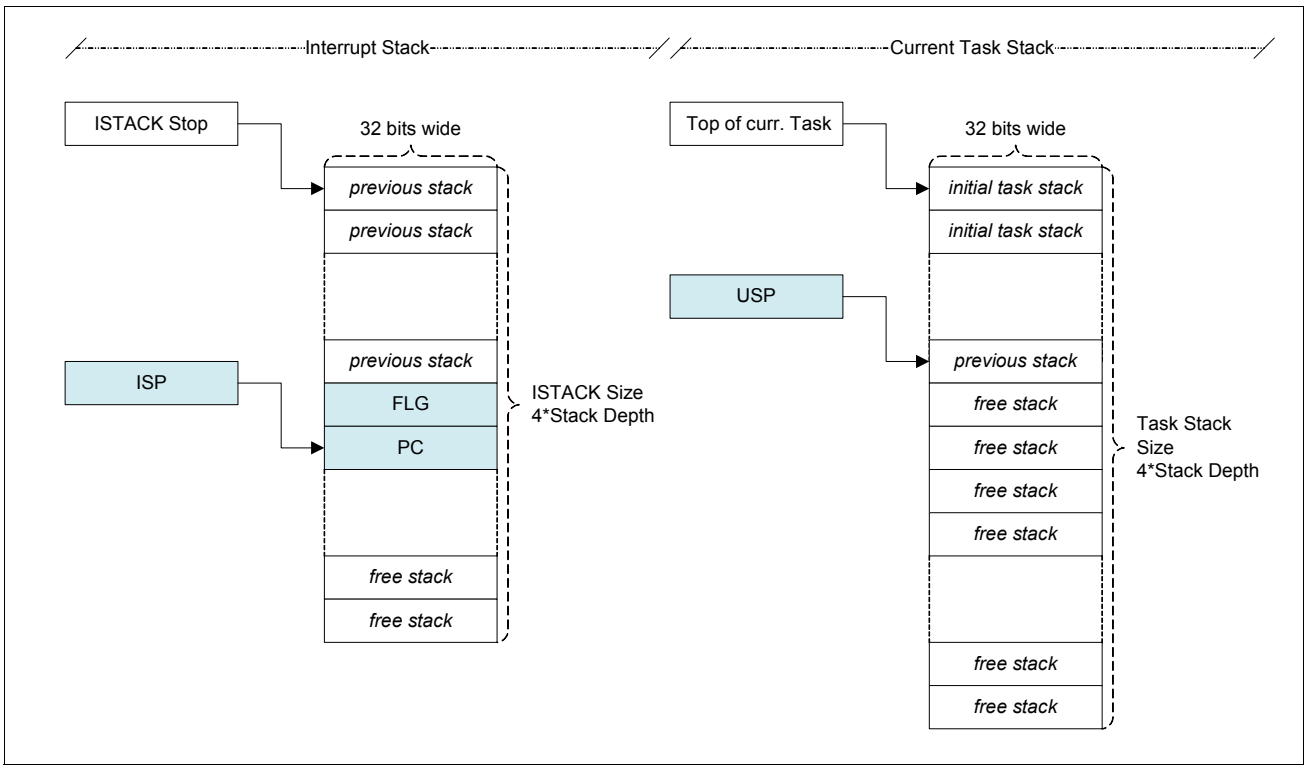

Figure 3 Interrupt- and User Stack Situation at Entry to Interrupt Service

The context of the current task is to be saved on the current task stack. This is done in four steps as also shown in Figure 4:

- The flag state and program counter saved on the interrupt stack during the interrupt execution sequence must be moved to the current task stack. This is most simply done by PUSH instructions with the U flag set because this implicitly allocates the required space from the task stack. (Marked with circle with number 1 in Figure 4)
- Save the remaining CPU registers on the task stack, using a PUSHM instruction. (2 in Figure 4)
- Now USP needs to be saved in the current task control block to be restored, when the this task is switched in again. (3 in Figure 4)
- The flag state and program counter on the interrupt stack are no longer used and need to be freed from the interrupt stack. (4 in Figure 4)
- Reactivate the interrupt stack, so interrupt service does not use task stack.

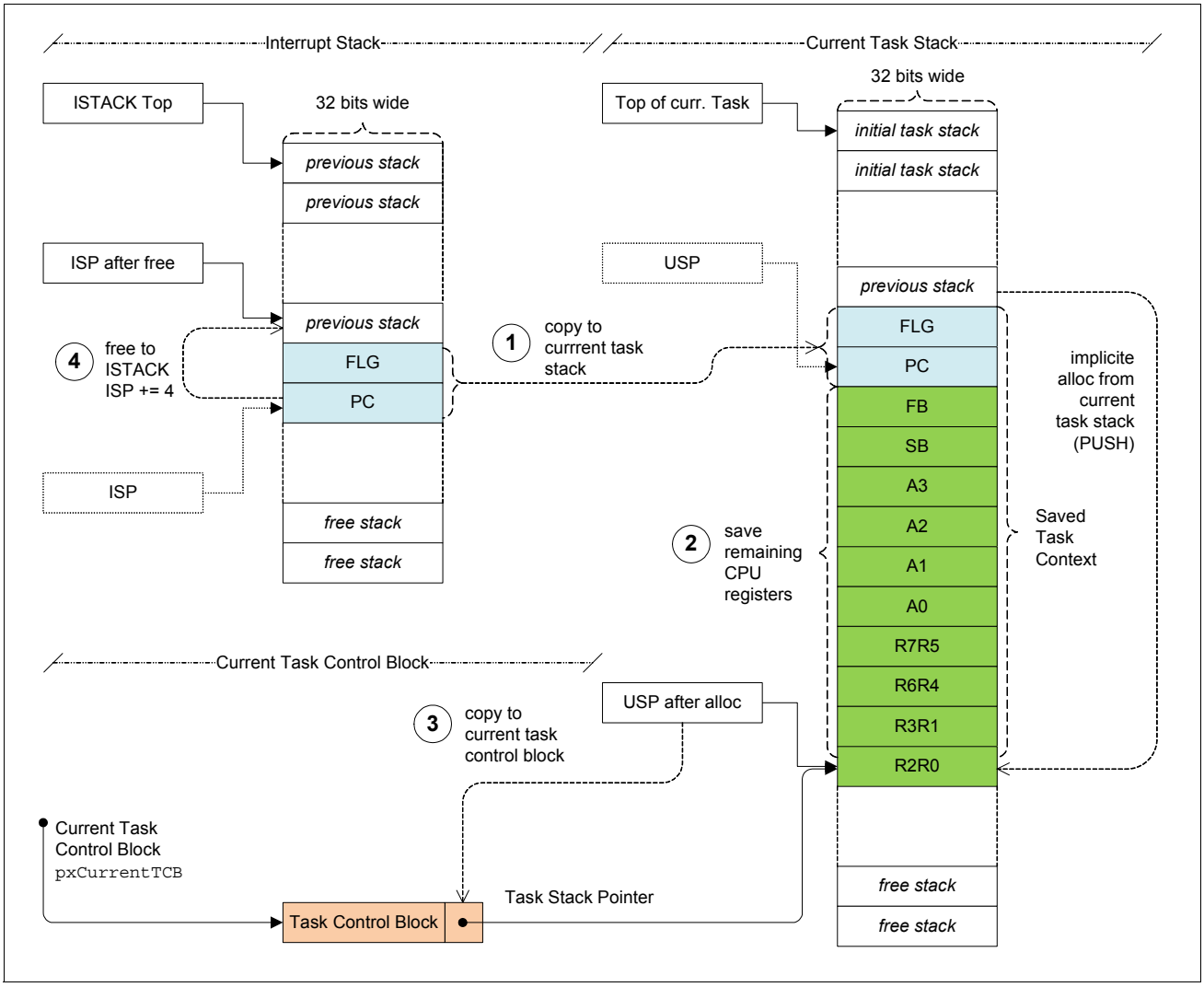

Figure 4 Task context saving

The advantage of this method is, that interrupt service routines can use their own stack (ISTACK) and do not rely on the stack of any switched out task. If the interrupt service routine used the stack of the switched out task, the interrupt stack space needs to be allocated on each task stack in addition to stack space the task requires for its own purpose. This is would not be a very economic memory usage. Find the assembler code below which implements step 1-4 shown in Figure 4.

```
        ; SAVE TASK CONTEXT ----------------------------------------------------
        ; see stack frame in port.c
save_context MACRO          ; Must copy saved registers from interrupt stack
                            ; to user stack and save remaining registers
                            ; we come here with U flag cleared          CYCLES
        PUSHC  FB           ; save FB to interrupt stack
        FSET   U            ; Save FB to task stack
        PUSHC  ISP          ; Load ISP to FB, using task stack as temp
        POPC   FB           ;

        ; now push PC, FB and FLG to task stack
        PUSH.L:G  8:8[FB]   ; | FLG |
        PUSH.L:G  4:8[FB]   ; | PC  |
        PUSH.L:G  0:8[FB]   ; | FB  |

        PUSHM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
                            ; now save the current user stack pointer in
                            ; the current task control block
                            ; the TCB is far memory
        MOV.L  pxCurrentTCB,A0;
        STC    SP,[A0];
        FCLR   U            ; activate the interrupt stack
        ADD.L  #12,SP       ; free from ISP the PC, FB and FLG
        ENDM                ;
                            ;
```

Selection of the task to switch in is not part of the port and is done by calling the RTOS function `vTaskSwitchContext()`.

Restoring the context of the new task is done by loading USP from the new current task control block followed by `POPM`.

```
        ; RESTORE TASK CONTEXT --------------------------------------------------
        ; see stack frame in port.c
restore_context MACRO       ; task context is on user stack
        FSET   U            ; activate the user stack
                            ; set user stack pointer to current task
                            ; the TCB is always far (32bit) memory
        MOV.L  pxCurrentTCB,A0;
        LDC    [A0],SP      ; restore the task context
        POPM   SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
        POPC   FB;
        ENDM                ;
                            ;
```

After the context is restored, a `REIT` with U flag set returns to the switched in task.

The code for the yield handler needs to consider two cases:

- The yield interrupt has been called from an interrupt service, this means the nesting level at entry is not equal to zero.
- The yield interrupt has fired from a running task. In this case the nesting level at entry equals zero.

When the nesting level is not zero at entry, the task selection is delayed until the last interrupt returns. The yield handler only sets a flag `intTaskSwitchPending`.

When the nesting level is zero, the yield interrupt needs to save the context of the interrupted task, call the task selection routine, restore the context of the new task and execute a `REIT` instruction.

Interrupt nesting is explained later in chapter 4.

```
        RSEG CODE24:CODE:REORDER:ROOT(0)

portYieldInterrupt:
        CMP.B #0,intNesting; do not reinable interrupts here,
                             ; we do not nest from this int.
                             ; so ++intNesting not required
        JNE portYieldInterrupt_0; If intNesting > 0, no task switch
                             ; task switch when lower priority int returns
        save_context;
        JSR.A vTaskSwitchContext;
        restore_context;
        REIT
portYieldInterrupt_0:
        MOV.L #-1,intTaskSwitchPending; intNesting is not 0,
                                     ; yield when ongoing
                                     ; interrupt terminates
        REIT;

        ; interrupt vector for YIELD
        COMMON INTVEC:HUGECONST:ROOT(0)
        ORG portYIELD_VECTOR*4
??portYieldInterrupt??INTVEC??:
        DC32    portYieldInterrupt;
```

4 Interrupt Nesting

To support interrupt nesting with minimal overhead, interrupt nesting and task selection is controlled with the help of two variables `intTaskSwitchPending` and `intNesting`. This allows for task selection to be delayed until the last of the nested ongoing interrupts complete.

```
        ; keep track of pending task switches
        ; cleared when interrupt prologue executed with intNesting==0 at entry
        ; set from tick timer interrupt when intNesting != 0 at entry
        ; set from yield interrupt when intNesting != 0 at entry
        ; set from user isr handler with the macro portEND_SWITCHING_ISR
        RSEG DATA16_N:NEARDATA:NOROOT(1)
        EVEN
intTaskSwitchPending:
        DS32 1


        ; keep track on the interrupt nesting
        ; Incremented in interrupt entry code
        ; Decremented in interrupt exit code
        RSEG DATA16_N:NEARDATA:NOROOT(1)
        EVEN
intNesting:
        DS8 1
```

Further the eight interrupt levels of the R32C are split into three groups.

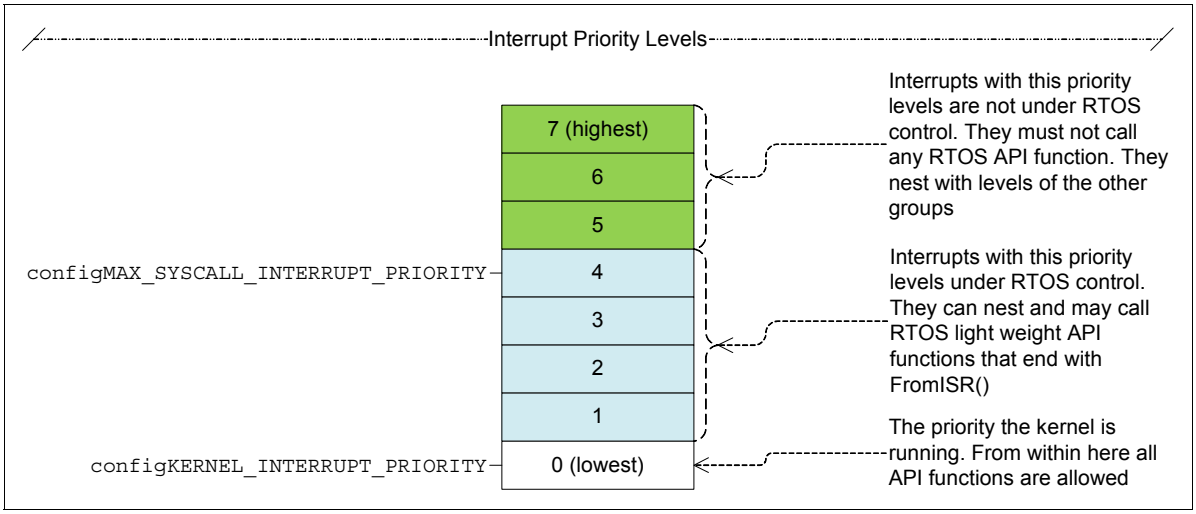

Figure 5 Interrupt priority levels

From this, we can define critical sections for the RTOS differently than just disabling all interrupts. We only need to disable interrupts that might call a RTOS API function. The compiler version used for this port has a bug in the `__get_interrupt_level` / `__set_interrupt_level` intrinsic functions that have effect when optimization is on. Some functions that could be written in C using these intrinsic functions have workaround code in the assembler file.

```
/* due to a bug in IAR R32C Compiler intrinsic __get_interrupt_level
   __set_interrupt_level functions, this is done in asm_func.s53 */
#ifndef __IAR_SYSTEMS_ASM__
  unsigned portBASE_TYPE portSetInterruptMaskFromISR(void);
  void portClearInterruptMaskFromISR(unsigned portBASE_TYPE saved);
#endif
#define portSET_INTERRUPT_MASK_FROM_ISR() portSetInterruptMaskFromISR()
#define portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedStatusRegister ) \
{ \
         portClearInterruptMaskFromISR(uxSavedStatusRegister);\
}
```

Following the workaround code in assembler for the macros above.

```
        ; CRITICAL SECTIONS FROM ISR                                    ------
        ; code called from interrupt ;  unsigned long f(void)
        RSEG CODE24:CODE:REORDER:NOROOT(0)
portSetInterruptMaskFromISR:
        STC   FLG,R2R0 ; Return current level in R2R0, including other flags
        LDIPL #configMAX_SYSCALL_INTERRUPT_PRIORITY;
        RTS;

        ; CRITICAL SECTIONS FROM ISR                                    ------
        ; code called from interrupt ;  void f(unsigned long)
        RSEG CODE24:CODE:REORDER:NOROOT(0)
portClearInterruptMaskFromISR:
        SHL.W    #-12,R0;
        AND.W    #7,R0  ; this should not be required, FLG does not define bit 15
        MULU.W   #jmpi_c-jmpi_b,R0; multiply by size of entry in code table
        ADD.W    #jmpi_b-jmpi_a,R0; add offset to code table
jmpi_a:
        JMPI.W   R0     ;
jmpi_b:
        LDIPL #0        ; 2 bytes - Offset 0
        RTS             ; 1 byte
jmpi_c:
        LDIPL #1        ;           Offset 3
        RTS
        LDIPL #2        ;           Offset 6
        RTS
        LDIPL #3        ;           Offset 9
        RTS
        LDIPL #4        ;           Offset 12
        RTS
        LDIPL #5        ;           Offset 15
        RTS
        LDIPL #6        ;           Offset 18
        RTS
        LDIPL #7        ;           Offset 21
        RTS
```

```
/* Critical section management. */
#define portDISABLE_INTERRUPTS()  {__set_interrupt_level(configMAX_SYSCALL_INTERRUPT_PRIORITY);}
#define portENABLE_INTERRUPTS()   {__set_interrupt_level(configKERNEL_INTERRUPT_PRIORITY);}
```

Another variant of the same concept is the `portENTER_CRITICAL` `portEXIT_CRITICAL` implementation. This port uses the nesting counter on the TCB for critical section nesting:

```
#define portCRITICAL_NESTING_IN_TCB (1)
#ifndef __IAR_SYSTEMS_ASM__
  extern void vTaskEnterCritical(void);
  extern void vTaskExitCritical(void);
#endif
#define portENTER_CRITICAL() do { vTaskEnterCritical(); } while (0)
#define portEXIT_CRITICAL()  do { vTaskExitCritical(); } while (0)
```

With this definition, interrupts from the highest priority group are never locked out. Interrupts from the group that call the light weight API functions (`FromISR(..)`) need prologue and epilogue around the handler code that keeps track of the interrupt nesting level and postpones task selection until the nesting level reaches zero. This is depicted in Figure 6. The assembler code for prologue and epilogue you find below.

```
        ; ENTRY CODE FOR USER INTERRUPT ----------------------------------------
        ; to be executed in each user interrupt
        ; check if nested interrupt. If nested, do not save task context
        ; if iterrupt has interrupted a task, save task context
user_interrupt_prologue MACRO
        LOCAL user_interrupt_prologue_0
        LOCAL user_interrupt_prologue_1
        INC.B intNesting;
        CMP.B #1,intNesting; intNesting was zero at entry, save task context
                           ; reset intTaskSwitchPending
        JEQ    user_interrupt_prologue_0
        PUSHC  FB;
        PUSHM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
                                    ; intNesting not zero,
                                    ; save context of interrupt being processed
        JMP   user_interrupt_prologue_1
user_interrupt_prologue_0:
        MOV.L #0,intTaskSwitchPending; Interrupts are not yet enabled, save
        save_context;
user_interrupt_prologue_1:
        MOV.L #intTaskSwitchPending,A0; Pass 32 bit address of intTaskSwitchPending
        FSET I; Interrupts above the one running allowed
        ENDM
```

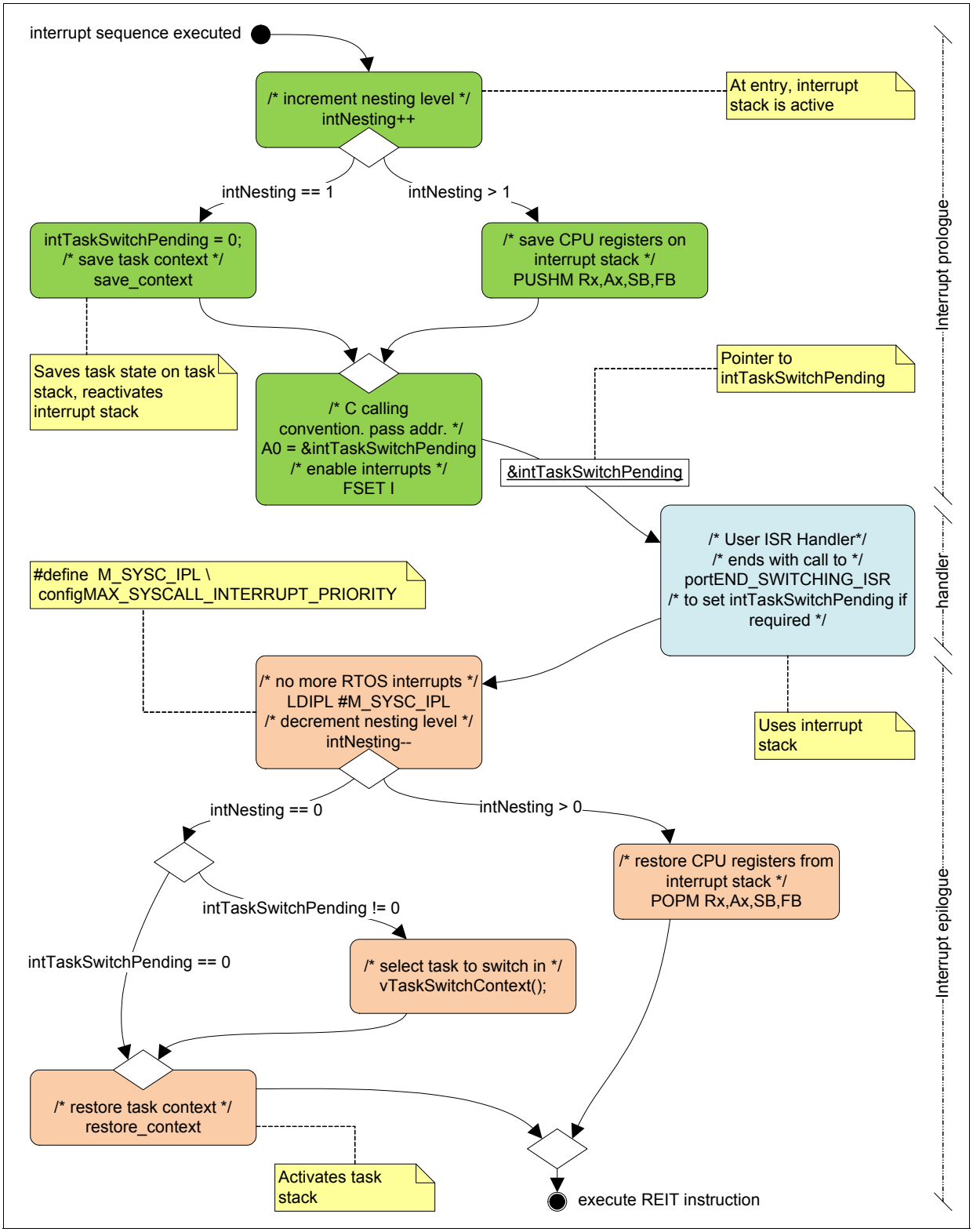

Figure 6 RTOS nested interrupt prologue and epilogue

```
        ; EXIT CODE FOR USER INTERRUPT ----------------------------------------
        ; to be executed in each user interrupt
        ; If nested, the previous interrupt context is on the Interrupt stack
        ; If not nested, the context of the task to activate needs to be
        ; restored on the user stack

user_interrupt_epilouge MACRO
        LOCAL user_interrupt_epilouge_0
        LOCAL user_interrupt_epilouge_1
        LOCAL user_interrupt_epilouge_2

        ; this macro is immedeately after the handler function returns.
        ; the handler function returns a value in R2R0
        ; return value in R2R0 : 0x0 no task switch required, otherwise
        ; do switch
        LDIPL #configMAX_SYSCALL_INTERRUPT_PRIORITY;  No more
                                                    ;  interrupts that use os
        CMP.B #0,intNesting; Just to make save. This never is the case
        JEQ    user_interrupt_epilouge_0
        DEC.B intNesting;
user_interrupt_epilouge_0:
        JNZ   user_interrupt_epilouge_1; Other interrupt is running
        CMP.L #0,intTaskSwitchPending;   Last interrupt check if task
                                    ;   switch is pending
        JEQ user_interrupt_epilouge_2; No task switch pending
        JSR  vTaskSwitchContext; Do the task switch
user_interrupt_epilouge_2:
        restore_context; Restore switched in task context
        REIT
user_interrupt_epilouge_1:
        POPM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0; Restore las interrupt context
        POPC  FB;
        REIT
        ENDM
```

5 System Tick Timer

The RTOS needs to track system time to implement task sleep delays, timeouts of blocking functions and pre-emptive multitasking. The system timer also needs to handle interrupt nesting. At every time tick the RTOS should select the next task (of same or higher task priority level). This simplifies the prologue and epilogue of the handler. For more details about interrupt nesting see chapter 4.

The following two functions install and uninstall timer R32C B0 as system tick timer. The following code is for 24 MHz counter clock.

```
/** set ms timer function */
/* This routine setups timer 1 and clears timer flag */
static void rtos_tick_timer_uninstall(void)
{
  tb0s = 0x00;      // stop timer B0 (count flag)
  tb0ic = 0;        // shut of interrupts from this timer
}

/** set ms timer function, assume count source clock is 24MHz */
/* This routine setups timer B0 and clears timer flag */
static void rtos_tick_timer_install(void)
{
  /* this function is called before interrupts are enabled */

  tb0mr = 0x40;     // XXXX XXXX
                    // |||| ||++- operation mode: 00: timer 01: event counter 10:one shot timer 11: PWM
                    // |||| |+--- pulse output at pin TA4out 0: OFF 1: ON
                    // |||| +---- gate function: 0: timer counts only when TA0in is "L", 1: .. is "H"
                    // |||+------ gate function  0: not available 1: available
                    // ||+------- must always be 0 in timer mode
                    // ++-------- count source select bits: 00:f1  01:f8  10:f32 11:fc32

  tb0 = (24000000ul/8u * 1 /*ms*/ /1000u )-1u;     // 1ms resolution @ 24MHz f8
  tb0s = 0x01;      // start timer B0 (count flag)
  tb0ic = configMAX_SYSCALL_INTERRUPT_PRIORITY;// interrupt priority level select bit 0...7
}
```

The interrupt handler function is written in assembler:

```
        ; SYSTEM TICK INTERRUPT HANDLER ----------------------------------------
        ; if interrupt nesting level is 0 at entry, do task switch
        ; if interrupt nesting level is greater 0 at entry, set
        ; intTaskSwitchPending and do increment system timer only
        RSEG CODE24:CODE:REORDER:ROOT(0)

portTimerB0Interrupt:
        INC.B intNesting;
        CMP.B #1,intNesting; yes do the task switch
        JNE   portTimerB0Interrupt_0
        FSET  I; Allow for other interrupts
        save_context;
        JSR.A vTaskIncrementTick;
        LDIPL #configMAX_SYSCALL_INTERRUPT_PRIORITY;
        MOV.B #0,intNesting; We end up here only when int nesting was zero
                             ; at entry
        JSR.A vTaskSwitchContext;
        restore_context;
        REIT
portTimerB0Interrupt_0: ; intNesting not 1, do not switch task,
                          ; do not save task context
        MOV.L #-1,intTaskSwitchPending;
        FSET I; Allow for other interrupts
        PUSHC  FB;
        PUSHM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
        JSR.A vTaskIncrementTick;
        LDIPL #configMAX_SYSCALL_INTERRUPT_PRIORITY;
        CMP.B #0,intNesting; Just to make save. This cannot be true
        JEQ    portTimerB0Interrupt_1;
        DEC.B intNesting;
portTimerB0Interrupt_1:
        POPM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
        POPC  FB;
        REIT


        ; interrupt vector for timer B0, system tick
        COMMON INTVEC:HUGECONST:ROOT(0)
        ORG 4*0x15
??portTimerB0Interrupt??INTVEC_15:
        DC32    portTimerB0Interrupt;
```

6 User Interrupt Handler

User Interrupt handlers that use the RTOS lightweight API (the functions that end with `FromISR(..)`) are defined at compile time using three macros. First, the name of the handler function is to be bound to an interrupt vector. This needs to be done in the project specific `FreeRTOSConfig.h` header file.
The macros to define are `USER_ISR_VECTOR_`*nn*, where *nn* is the hexadecimal vector number from the interval 0 to FF. Use a leading zero for vectors ‹ 10. Be careful when using interrupts with vectors › 0x7F. All vectors of peripheral circuits on the chip are below that limit, leaving only software interrupts for the higher vectors. The code has not yet been tested with vectors › 127 that do not clear the U flag.

```
// FreeRTOSConfig.h
// The name of the handler function for vector 0x14 is uart_1_rx_isr
#define USER_ISR_VECTOR_14 uart_1_rx_isr
```

This define enables a section in the port file `asm_func.s53` which defines prologue, epilogue and the vector table entry for the handler function. The assembler file has a section for each supported interrupt vector. Following the code that implements nesting prologue and epilogue as explained in chapter 4.

```
; asm_func.s53
#ifdef USER_ISR_VECTOR_14
       EXTERN USER_ISR_VECTOR_14
       PUBLIC ??USER_ISR_VECTOR??INTVEC_14

       RSEG CODE24:CODE:REORDER:ROOT(0)
userIsrVector14:
       user_interrupt_prologue;
       JSR.A USER_ISR_VECTOR_14; call the service routine
       user_interrupt_epilouge;
       REIT;

       COMMON INTVEC:HUGECONST:ROOT(0)
??USER_ISR_VECTOR??INTVEC_14:
       ORG 0x14*4
       DC32 userIsrVector14;

       #undef USER_ISR_VECTOR_14
#endif
```

Next, the prototype for the handler can be defined in any project file using another macro:

```
// my_isr.c
portINTERRUPT_HANDLER_PROTO(uart_1_rx_isr );
```

The implementation of the handler function is done in any project file as following snipped shows:

```
// my_isr.c
portINTERRUPT_HANDLER(uart_1_rx_isr )
{
    portBASE_TYPE xHigherPriorityTaskWoken;

    /* handler code */
    …
    …
    /* call a lightweigth API function */
    xQueueSendFromISR(hQueue, &m, &xHigherPriorityTaskWoken );
    portEND_SWITCHING_ISR( xHigherPriorityTaskWoken );
}
```

This shows the way the handler tells the interrupt epilogue if the task selection routine needs to be run.
The implementation is simple. When the handler is called from the port file `asm_func.s53` it passes a pointer to the variable `intTaskSwitchPending` (see chapter 4) from that same module in A0. This meets the specification for the R32C calling convention as defined in [4]. Because the handler calls are enclosed with appropriate prologue and epilogue, they do not need to save any processor registers and therefore can have the attribute `__task` (see [4]).

```
#define portINTERRUPT_HANDLER_PROTO( isrFunction ) \
        __task void isrFunction ( pdISR_PARAM pxSwitchRequired__ )
#define portINTERRUPT_HANDLER( isrFunction ) \
        __task void isrFunction ( pdISR_PARAM pxSwitchRequired__ )
```

The value returned is set from the macro `portEND_SWITCHING_ISR`. Do not assign a value to the parameter `pxSwitchRequired__` directly, more exact do not initialize this parameter with zero in your handler. If the handler is called while another lower priority interrupt is ongoing, the lower priority interrupt could already have requested a task switch. That is why the `|=` operator is used to modify the value in the macro `portEND_SWITCHING_ISR`.

```
#define portEND_SWITCHING_ISR( xSwitchRequired ) \
        if (1) {(*pxSwitchRequired__) |= xSwitchRequired;} else ((void)0)
```

7 Starting/Stopping the OS

Starting the RTOS requires some actions:

- Install the tick timer.
- Save the flag register.
- Push the processor registers to the active stack.
- Save the user stack pointer.
- Save interrupt stack pointer.
- Set `intNesting` to 0. (See Chapter 4)
- Do a `restore_context` as explained in »Task Switching Primitives« followed by a `REIT` instruction.

Interrupts with a priority equal or lower to `configMAX_SYSCALL_INTERRUPT_PRIORITY` (see chapter 4) should not be on when the RTOS is about to start. Interrupts are enabled when the context of the first task is restored. See chapter »Task Stack Layout and Creation« for details about the initial task context.

```
/*
 * Setup the hardware ready for the scheduler to take control.  This generally
 * sets up a tick interrupt and sets timers for the correct tick frequency.
 */
portBASE_TYPE xPortStartScheduler( void )
{
  rtos_tick_timer_install();

  /* this enables interrupts because the initialized stack frame contains
     the flag register status */
  portStartScheduler_asm(); /* returns, when portEndScheduler_asm is called */

  return pdFALSE;
}
```

The two stack pointer and the flag register registers to save have static space in the port assembler file `asm_func.s53`.

```
        ; LOCAL VARIABLES -----------------------------------------------------
        ; keep context prior to the scheduler started
        RSEG DATA16_N:NEARDATA:NOROOT(1)
        EVEN
preStartSchedulerUSP:
        DS32 1
        RSEG DATA16_N:NEARDATA:NOROOT(1)
        EVEN
preStartSchedulerFLAGS:
        DS32 1
        RSEG DATA16_N:NEARDATA:NOROOT(1)
        EVEN
preStartSchedulerISP:
        DS32 1
```

Following the code that saves the context and switches in the first task to run.

```
        ; OS START CODE, SAVE CONTEXT PRIOR OS UP, SWITCH IN FIRST TASK --------
        ; void portStartScheduler_asm(void)
        ; this code is used to activate the first task. It returns when
        ; void portEndScheduler_asm(void) is called
        RSEG CODE24:CODE:REORDER:NOROOT(0)
portStartScheduler_asm:
        PUSHC  FB;
        PUSHM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
        STC   FLG,preStartSchedulerFLAGS;
        FCLR  I; stop interrupts
        FSET  U;
        STC   SP,  preStartSchedulerUSP;
        STC   ISP, preStartSchedulerISP;
        MOV.B #0, intNesting
        restore_context;
        REIT;
```

The stop code is similar but in reverse order:

- Uninstall the tick timer.
- Do a `save_context` as explained in »Task Switching Primitives«.
- Restore ISP, USP and the FLG registers from prior the call to `portStartScheduler_asm`
- Restore the CPU registers from the active stack.
- Return from the subroutine.

```
/*
 * Undo any hardware/ISR setup that was performed by xPortStartScheduler() so
 * the hardware is left in its original condition after the scheduler stops
 * executing.
 */
void vPortEndScheduler( void )
{
  rtos_tick_timer_uninstall();
  portEndScheduler_asm();
}
```

Following the assembler code that switches out the task which called `vPortEndScheduler`.

```
        RSEG CODE24:CODE:REORDER:NOROOT(0)
portEndScheduler_asm:
        save_context; this activates the ISP
        FCLR  I; stop interrupts
        LDC   preStartSchedulerISP, ISP;
        FSET  U;
        LDC   preStartSchedulerUSP, SP;
        LDC   preStartSchedulerFLAGS, FLG;
        POPM  SB,A3,A2,A1,A0,R7R5,R6R4,R3R1,R2R0;
        POPC  FB;
        RTS;
```

8 The Demo Application

The demo application is compiled for [6], which is a low cost evaluation platform for the R32C/111 microcontroller. The MCU is powered from USB and connects with the PC with a serial port emulation over USB. To download the demo application, use the »FlashSta100.exe« application that comes with the kit on the CD.

The demo application runs three tasks and a timer interrupt:

- One that blinks the LED soldered to the PCB.
- One that receives messages sent from an interrupt routine (timer) and toggles P3.1 on pin 31 of the DIL64 pads. The interrupt toggles P3.2 on pin 32 of the the DIL64 pads.
- One that makes use of the FreeRTOS R32C UART driver and echoes characters received.

Open a terminal emulation window (for example [7] which can be downloaded for free from the internet) and connect to the serial port the PCB is connected to. When pressing the reset button, a welcome text is sent so the terminal (Figure 7). When text is entered, the com_task receives the characters and echoes them back to the terminal. From the terminal you may send a »break condition«. A »break condition« occurs when the receiver input is at the »space« level for longer than some duration of time, typically, for more than a character time. This is not necessarily an error, but appears to the receiver as a character of all zero bits with a framing error. The R32C UART driver allows to register a callback function that gets called when such a situation is detected. When using this callback one needs

to keep in mind that it is called from interrupt context, so restrictions to RTOS API functions called from interrupts apply for code executed in the callback routine.

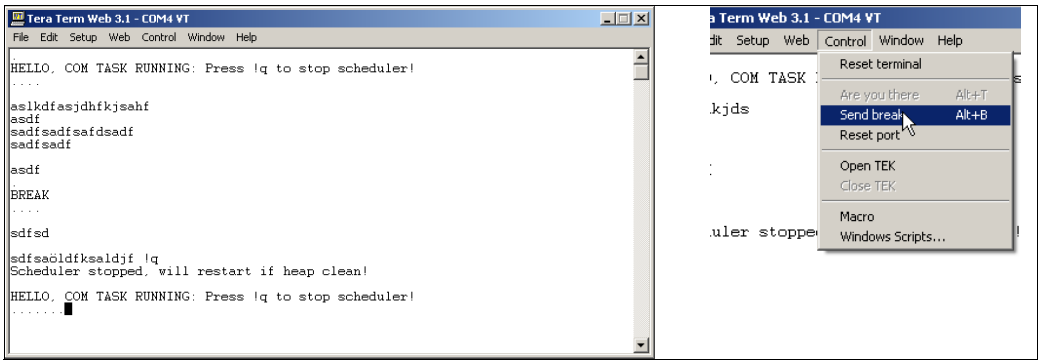

Figure 7 Terminal window

Using an oscilloscope one can measure the latency with which a task receives and handles a message sent from the timer interrupt. The time measured (see Figure 8) is just an example that gives an idea. Times required in a real application depend on some parameters. Their discussion is out of scope of this document.

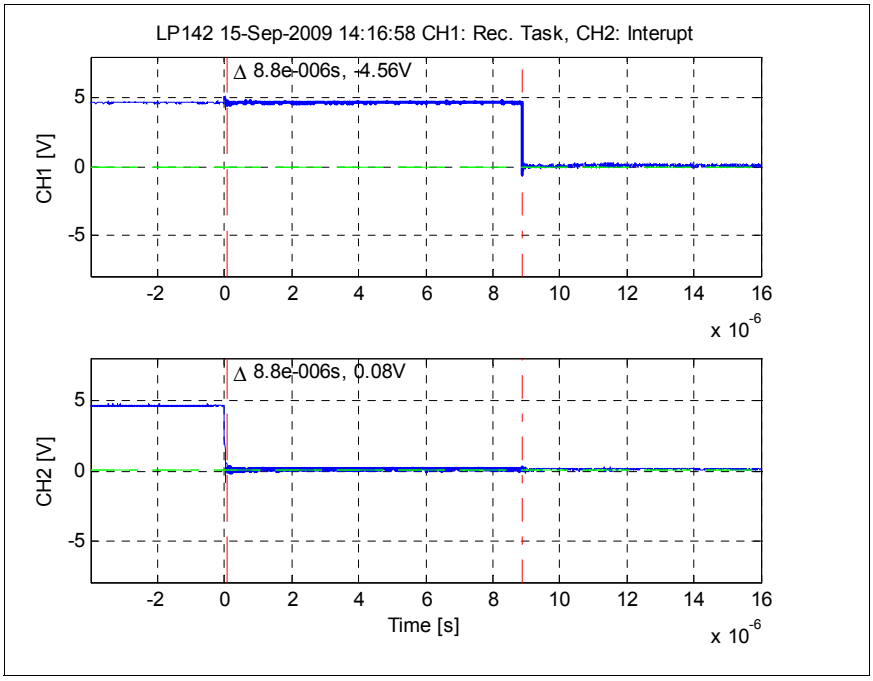

Figure 8 Latency of a task receiving a message from an interrupt

9 GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

* D. Preserve all the copyright notices of the Document.

* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

* H. Include an unaltered copy of this License.

* I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

* M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

* N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

* O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.