SAFE**RTOS** APPLICATION NOTE: #34-172-AN-009

USING SAFE**RTOS** FROM ROM ON STELLARIS MICROCONTROLLERS

CONTENTS

REFERENCED DOCUMENTS

Ref #	Document	Description
1	34-172-MAN-1	SAFE**RTOS** User Manual
2	34-172-MAN-2	SAFE**RTOS** Safety Manual – available as part of the optional Design Assurance Pack.
3	IEC 61508	Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems

CHAPTER 1

SAFERTOS AND STELLARIS

1.1 INTRODUCTION

The LM3S9B96 is a Cortex M3 microcontroller from Texas Instruments that comes with a pre-built version of SAFE**RTOS** embedded in ROM. Applications running on the LM3S9B96 can link to and use the 'ROMed' copy of SAFE**RTOS** in commercial applications without paying any license fees.

The SAFE**RTOS** User Manual [Reference 1] and Safety Manual [Reference 2] provide detailed but generic information on the use of SAFE**RTOS**. This application note provides additional usage and configuration information that is specific only to using a Stellaris microcontroller that already has SAFE**RTOS** pre-programmed in ROM. The information in this application note supersedes that found in the generic manuals when using this platform.

The recommended way to get running with SAFE**RTOS** is to start with one of the pre-configured demo applications. This application note documents and explains how the demo applications are configured so the configuration can be copied or replicated for real applications.

1.1.1 Use in Safety Related Systems

Its small size and simplicity makes SAFE**RTOS** ideal for use in a wide range of embedded application areas, including those that require safety certification. SAFE**RTOS** has been independently certified to ease application certification, especially in industrial and medical applications. Users can buy a separate Design Assurance Pack which contains complete certification evidence for IEC61508 [Reference 3] SIL3 and FDA510(k). Refer to the SAFE**RTOS** User Manual and http://www.SafeRTOS.com for more information on using SAFE**RTOS** in safety related systems.

1.1.2 File Systems, USB Drivers, and TCP/IP Protocol Suites

http://www.SafeRTOS.com also provides information on the range of tightly integrated file systems, USB drivers and TCP/IP stacks that can be purchased separately for use with SAFE**RTOS**.

CHAPTER 2

INSTALLATION

2.1 ACCESSING SAFERTOS FROM ROM

2.1.1 SAFERTOS_API.h

The SAFE**RTOS** functionality for the LM3S9B96 is pre-built and resident in ROM so the SAFE**RTOS** source files do not need to be included in the application build. Instead a jump table is used to link the application code to the ROMed kernel code at run time. The jump table is contained in a header file called SafeRTOS_API.h. This header file must be included from any source file that wishes to call a SAFE**RTOS** API function. Including SafeRTOS_API.h allows the SAFE**RTOS** API to be used in the normal way.

2.2 PROJECT CONFIGURATION

2.2.1 Include Path

SafeRTOS_API.h is included within the StellarisWare directory structure. The same directory location contains several other dependent SAFE**RTOS** related header files and must be included in the compilers pre-processor include path (the compiler must be configured so it can find the SAFE**RTOS** header files).

2.2.2 Linker Configuration

The linker script used by the application must reserve the first 0x20C bytes of RAM for use by SAFE**RTOS**. See the provided demo applications for example linker scripts that can be used as a reference, or simply copied.

2.2.3 Vector Table

The interrupt vector table must be populated with the addresses of the SAFE**RTOS** interrupt handlers located in ROM. The addresses are defined in SAFERTOS_API.h. Table 2-1 contains the vectors that require populating with SAFE**RTOS** addresses and the constants to be used for each address. Listing 1 demonstrates how the constants can be used. Again, see the provided demo applications for example vector table definitions that can be used as a reference, or simply copied.

Table 2-1 Vector Table Definitions Required to Use SAFE**RTOS** From ROM

Vector	Required Definition
SVCall	vSafeRTOS_SVC_Handler_Address
PendSV	vSafeRTOS_PendSV_Handler_Address
SysTick	vSafeRTOS_SysTick_Handler_Address


```
/* Include the header file that contains the addresses of the SafeRTOS interrupt
handlers. */
#include "SafeRTOS_API.h"

//*****************************************************************************
//
// The vector table.
//
//*****************************************************************************
__attribute__ ((section(".isr_vector"))) /* (GCC syntax for demonstration only) */
void (* const g_pfnVectors[])(void) =
{
    (void *)&_vStackTop,                        // The initial stack pointer
    ResetISR,                                   // The reset handler
    NmiSR,                                      // The NMI handler
    FaultISR,                                   // The hard fault handler
    IntDefaultHandler,                          // The MPU fault handler
    IntDefaultHandler,                          // The bus fault handler
    IntDefaultHandler,                          // The usage fault handler
    0,                                          // Reserved
    0,                                          // Reserved
    0,                                          // Reserved
    0,                                          // Reserved
    ( void * ) vSafeRTOS_SVC_Handler_Address,   // SVCall handler
    IntDefaultHandler,                          // Debug monitor handler
    0,                                          // Reserved
    ( void * ) vSafeRTOS_PendSV_Handler_Address, // The PendSV handler
    ( void * ) vSafeRTOS_SysTick_Handler_Address,// The SysTick handler
    IntDefaultHandler,                          // GPIO Port A
    IntDefaultHandler,                          // GPIO Port B
    IntDefaultHandler,                          // GPIO Port C

    /* Rest of vector table definition goes below here. */
```

Listing 1 An example vector table definition

CHAPTER 3

INTERRUPT SERVICE ROUTINES

3.1 DEFINING AN INTERRUPT SERVICE ROUTINE (ISR)

Interrupt service routines do not require any special syntax and can be written as described in the compiler documentation.

An application defined interrupt handler can request a context switch by calling taskYIELD_FROM_ISR() as described in the SAFE**RTOS** User Manual [Reference 1], and demonstrated by Listing 2. An interrupt should do this if it unblocks a task, and the task that was unblocked has a priority above the currently executing task. Requesting a context switch within the interrupt will ensure the interrupt returns directly and immediately to the higher priority task.

```
void vXYZ_ISR( void )
{
portBASE_TYPE xYieldRequired = pdFALSE;

        /* Clear the interrupt. */

        /* Perform ISR work here. */

        /* A yield is required. */
        xYieldRequired = pdTRUE;

        /* Perform the yield. */
        taskYIELD_FROM_ISR( xYieldRequired );
}
```

Listing 2 The template for an ISR

3.2 INTERRUPT PRIORITIES AND NESTING

SAFE**RTOS** is pre-loaded into ROM and its configuration cannot be modified. In the ROMed version portSYSCALL_INTERRUPT_PRIORITY is set to 191. Therefore **interrupts that use the** SAFE**RTOS API must be assigned a priority equal to or below 191** (or 5, depending on which notation is used).

Important: The Cortex-M3 core uses numerically low priority numbers to represent HIGH priority interrupts, which can seem counter-intuitive and is easy to forget. If you wish to assign an interrupt a low priority do NOT assign it a priority of 0 (or other low numeric value) as this can result in the interrupt actually having the highest priority; and potentially make your system crash if this priority is above portSYSCALL_INTERRUPT_PRIORITY. Interrupts that use the SAFE**RTOS** API must be assigned a numeric priority of 191 or a numerically **higher** value to give them an interrupt priority equal to or lower than 191.

Only API functions that end in “FromISR” or “FROM_ISR” can be called from an interrupt service routine. Refer to the SAFE**RTOS** User Manual [Reference 1] for details of which API functions can be safely called from within interrupt service routines.

CHAPTER 4

DEMO APPLICATIONS

4.1 GETTING STARTED

One of the easiest ways to get running with SAFE**RTOS** in ROM is to start with a demo application that already has the compiler options, linker script, start up files and vector tables correctly configured. These are available from both Texas Instruments and WITTENSTEIN and are intended to not only get you up and running with the minimum of fuss, but also serve as a training aid and as a start point for your own application.

Amongst other things, the demo applications provide examples of:

- How to initialize the scheduler.
- How to use compiler extensions to ensure buffers used by tasks and queues are correctly aligned.
- How to create tasks.
- How to dimension and create queues.
- How to start the scheduler.
- How to configure the include paths.

4.2 TROUBLE SHOOTING

Finally, following a debug session the LM3S9B96 flash memory might be left in a locked state, depending on how the debug session was terminated. This will prevent the flash from being erased or re-programmed. To unlock the flash use the Texas Instruments Stellaris Flash Programming Utility that comes on the DK-LM3S9B9 CD, and is also available for download from the Texas Instruments WEB site.

CONTACT INFORMATION

User feedback is essential to the continued maintenance and development of SAFE**RTOS**. Please provide all software and documentation comments and suggestions to the most convenient contact point listed below.

Address:	WITTENSTEIN high integrity systems Brown's Court, Long Ashton Business Park Yanley Lane, Long Ashton Bristol, BS41 9LB England
Phone:	+44 (0)1275 395 600
Fax:	+44 (0)1275 393 630
Email:	support@HighIntegritySystems.com
Website	www.HighIntegritySystems.com