

Application Note

FreeRTOS on Hercules Devices



Veena Kamath

Abstract

FREERTOS is a popular open-source real-time operating system used in embedded systems. It provides APIs for tasks, semaphores, mutexes, timers, and so forth.

HALCoGen is a GUI-based driver generating tool for the Hercules family of devices. HALCoGen also supports FREERTOS for various devices in the Hercules family. It enables users to generate the FREERTOS code, along with other drivers.

This application report provides an overview on FreeRTOS. This document is not a FreeRTOS user's guide, and only describes how to use FreeRTOS on Hercules devices with HALCoGen.

Table of Contents

1 FreeRTOS Source Files	2
2 FreeRTOS Support in HALCoGen	2
3 MPU Settings as an Add-on for Safety Use Cases	4
3.1 MPU Wrapper Functions	4
3.2 MPU Regions for Tasks	4
4 Porting FREERTOS on a Different Part Number	6
5 Revision History	11

List of Figures

Figure 2-1. Select Device	2
Figure 2-2. OS Tab	3
Figure 4-1. Create a HALCoGen Project	6
Figure 4-2. Configure PLL	7
Figure 4-3. New Project	8
Figure 4-4. VIM Configuration	9
Figure 4-5. VIM ISR Assignments	9
Figure 4-6. Enable PortSWI	10

List of Tables

Table 3-1. Device Families	4
----------------------------	---

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

1 FreeRTOS Source Files

FreeRTOS source files can be divided into platform-dependent and platform-independent files. It supports various families, including TI's Hercules microcontroller.

The FreeRTOS\Source folder contains the drivers for each submodule, such as tasks, semaphores, queues, and so forth. These drivers are platform-independent.

The FreeRTOS\Source\include folder contains the header files, including a header file named `mpu_wrappers.h`. For the platforms that support MPU, these header files redefine the kernel functions to the corresponding MPU wrapper functions. More details on MPU wrapper functions are provided in the following sections.

The FreeRTOS\Source\portable\XXX folder contains platform-dependent code. For every platform, FreeRTOS provides a `.c` and an assembly file for port functions, and a header file named `portmacro.h` for the platform-dependent macros.

Along with these source files, a FreeRTOS project requires the following files:

- `FreeRTOSConfig.h` — Contains macros for customizing the FreeRTOS drivers. Sample file provided as part of demos.
- `mpu_wrappers.c` — Contains the MPU wrapper functions for kernel functions. For FreeRTOS v9 and later, a sample file is provided as part of FreeRTOS\Source\portable\Common folder.

2 FreeRTOS Support in HALCoGen

To generate FreeRTOS code along with HALCoGen drivers, select `DeviceName_FREERTOS` as Device while creating the HALCoGen project, as shown in [Figure 2-1](#). FreeRTOS drivers are not supported for every device variant in a device family. This document provides steps on how to customize the drivers for a different device variant.

The FreeRTOS source files are generated with a prefix `"os_"`.

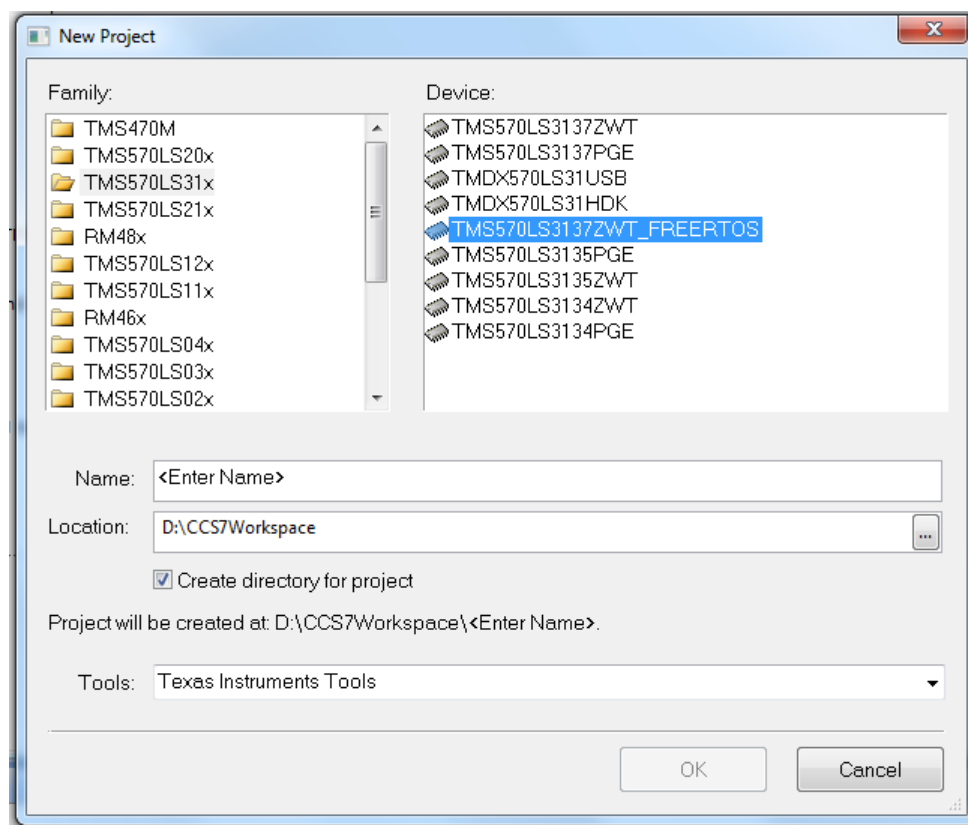


Figure 2-1. Select Device

Navigate to the OS tab to configure various options for the OS files, as shown in [Figure 2-2](#).

Start Page TMS570LS3137ZWT_FREERTOS **OS** PINMUX GIO SCI LIN SCI2 MIBSPI1

General

Configuration
Configuration options will set macros in FreeRTOSConfig.h

☒ Use Task Preemption ☐ Use Mutexes ☒ Use Verbose Stack Checking
☐ Use Idle Hook ☐ Use Recursive Mutexes ☐ Use Timers
☐ Use Tick Hook ☐ Use Counting Semaphores ☐ Generate Runtime Statistics
☐ Use Co-Routines ☒ Idle Task Should Yield ☐ Use Malloc Failed Hook
☐ Use Trace Facility ☐ Use Stack Overflow Hook

Task Configuration

RTI Clock (Hz): 90000000 Tick Rate (Hz): 1000
Max Priorities: 5 Total Heap Size: 8192
Task Name Length: 16 Min Stack Size: 128

Coroutine Configuration

Coroutine Priorities: 2

Timers Configuration

Timer Task Priority: 0 Queue Length: 0 Stack Size: 0

Figure 2-2. OS Tab

These configurations are translated to create the FreeRTOSConfig.h, and are provided as part of the generated drivers.

3 MPU Settings as an Add-on for Safety Use Cases

For devices with a Memory Protection Unit (MPU), FreeRTOS provides hooks for MPU configurations. MPU lets the user partition the device memory into various regions and configure the access rights for each regions. TI's Hercules microcontrollers are Arm®Cortex®-R based and support MPU.

3.1 MPU Wrapper Functions

There are two modes of CPU: privileged mode and user mode (names may differ depending on the device). User mode is a restricted mode, and is typically used to execute the user tasks. Privileged mode is used to execute the kernel code and for system initialization. The kernel code and data are typically placed in a privileged memory section, so that these are not accessed directly by user tasks. MPU wrapper functions provide wrappers for kernel functions so that they are executed in privileged mode.

Sample MPU wrapper function:

```
void MPU_vTaskDelete ( TaskHandle_t xTaskToDelete )
{
    BaseType_t xRunningPrivileged = prvRaisePrivilege();
    vTaskDelete ( xTaskToDelete );
    portRESET_PRIVILEGE( xRunningPrivileged );
}
```

3.2 MPU Regions for Tasks

The FreeRTOS task driver provides support to have n number of memory regions assigned to each task. The memory regions and its access rights may vary, depending on the tasks.

In a FreeRTOS environment, the MPU regions can be categorized into task-specific regions and common regions. Task-specific regions are configured every time a task switch occurs. The common regions are configured at the time the scheduler is initialized.

The first four regions and the last region (regions 0-3 and n-1) are configured when the scheduler is started and are common for all the tasks. This includes the flash, RAM, and peripherals memories. The other regions are task-specific, and can be configured during the task creation. There is a dedicated MPU region for the task stack.

[Table 3-1](#) lists the device families with the number of MPU regions, and how they are split into common and task-specific regions.

Table 3-1. Device Families

Device Family	No. of MPU Regions	Common Regions	Task-Specific Regions
TMS570LS31x RM48x TMS570LS12x RM46x	12	Region 0: Unprivileged flash region Region 1: Privileged flash region Region 2: Privileged RAM region Region 3: Peripherals region Region 11: Privileged system region	Region 4: Task stack Regions 5-10: User-defined
TMS570LS04x RM42x	8	Region 0: Unprivileged flash region Region 1: Privileged flash region Region 2: Privileged RAM region Region 3: Peripherals region Region 7: Privileged system region	Region 4: Task stack Regions 5,6: User-defined

Table 3-1. Device Families (continued)

Device Family	No. of MPU Regions	Common Regions	Task-Specific Regions
TMS570LC43x RM57x	16	Region 0: Unprivileged flash region Region 1: Privileged flash region Region 2: Privileged RAM region Region 3: Peripherals region Region 15: Privileged system region	The number of task-configurable regions can be changed in the HALCoGen GUI. Default = 3 Region 11: Task stack Regions 12-14: User-defined The remaining regions can be configured in the HALCoGen GUI, and are common for all tasks.

Note

Privileged flash and RAM regions include the privileged code and data section used by the kernel.

All the MPU configurations done through the HALCoGen GUI are reset when the scheduler is initialized. With TMS570LC43x and RM57x devices, there is an option to set the number of regions used by the FreeRTOS. The regions not used by FreeRTOS can be configured through the HALCoGen GUI. These are not reset by the FreeRTOS code.

To modify the default MPU regions set by FreeRTOS, modify the prvSetupDefaultMPU function available on the os_port.c file.

A task can be created using two FreeRTOS APIs:

- **xTaskCreate** — This disables all user-defined task-specific regions, and enables access to the entire RAM region as part of the task stack region. The task created using this API is called a non-restricted task, as this has access to the entire RAM region.
- **xTaskCreateRestricted** — This lets the user define the regions they need access to. The task stack region is set with the actual stack size. The task created using this API is called a restricted task, as this does not have access to any RAM region other than its own stack (unless explicitly mentioned).

For more details on how to create restricted and non-restricted tasks, see the HALCoGen examples.

4 Porting FREERTOS on a Different Part Number

HALCoGen supports FreeRTOS driver generation for only a few devices. This section explains how to generate FreeRTOS drivers for any device part number. This example uses the TMS570LS1227PGE device. For the TMS570LS12x family, HALCoGen provides a FreeRTOS port for the TMS570LS1227ZWT part.

1. Create a HALCoGen project with TMS570LS1227ZWT_FREERTOS, as shown in [Figure 4-1](#).

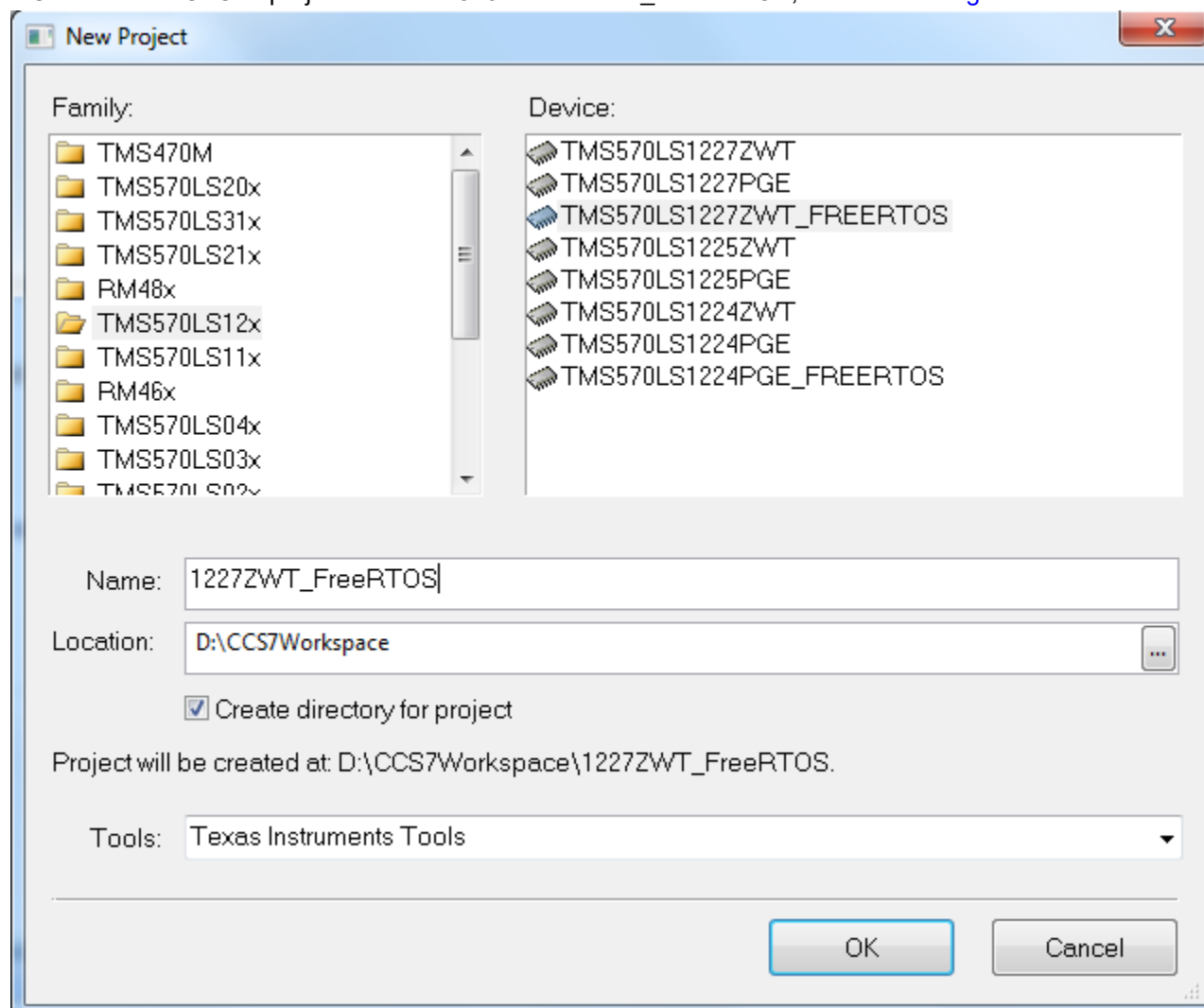


Figure 4-1. Create a HALCoGen Project

- Configure the PLL and other clock settings as per the actual device requirements. In the example shown in Figure 4-2, the maximum CPU clock for the TMS570LS1227PGE is 160 MHz, whereas, with the TMS570LS1227ZWT, it is 180 MHz.

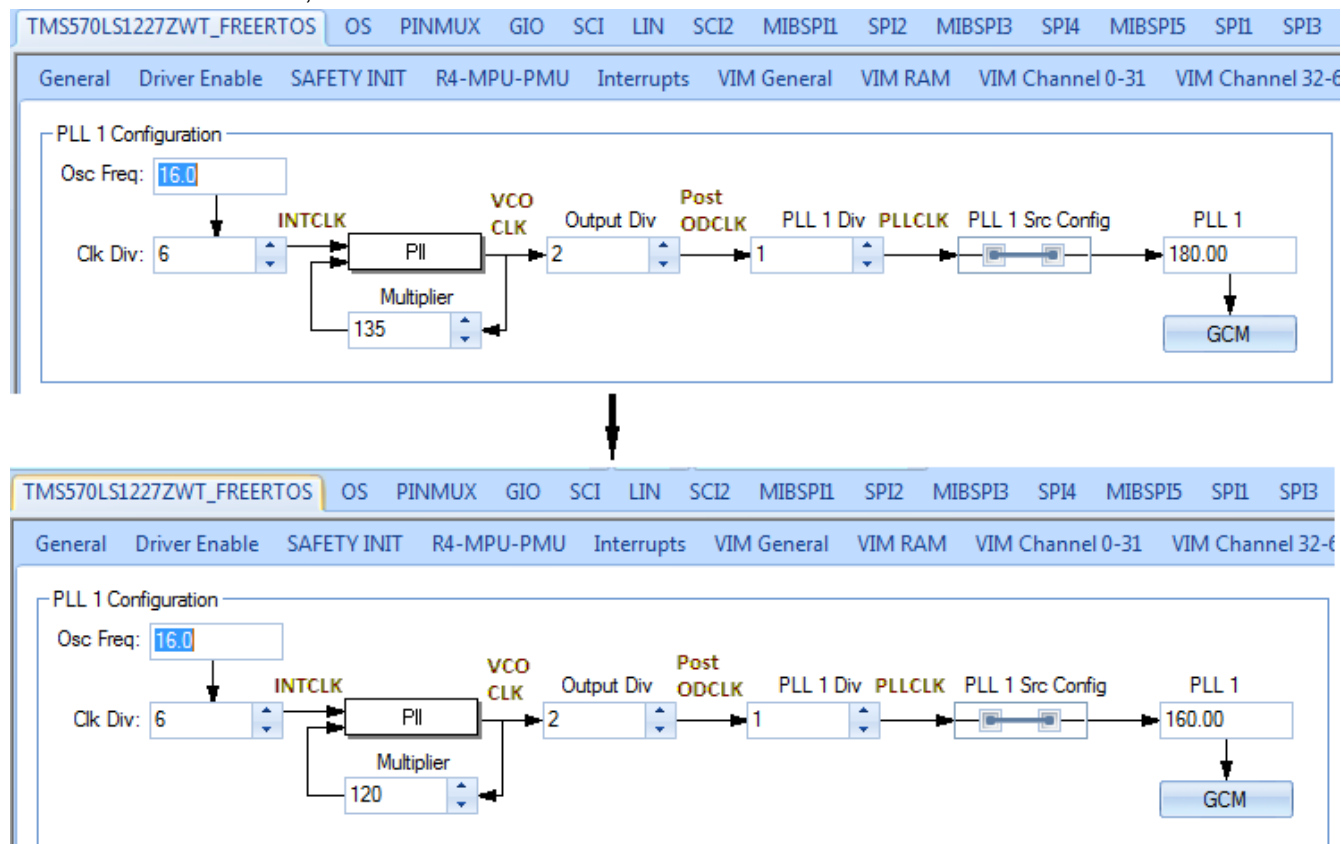


Figure 4-2. Configure PLL

- Configure the FreeRTOS port as required in the OS tab. Save and generate code.

4. Create a HALCoGen project with the TMS570LS1227PGE, as shown in [Figure 4-3](#).

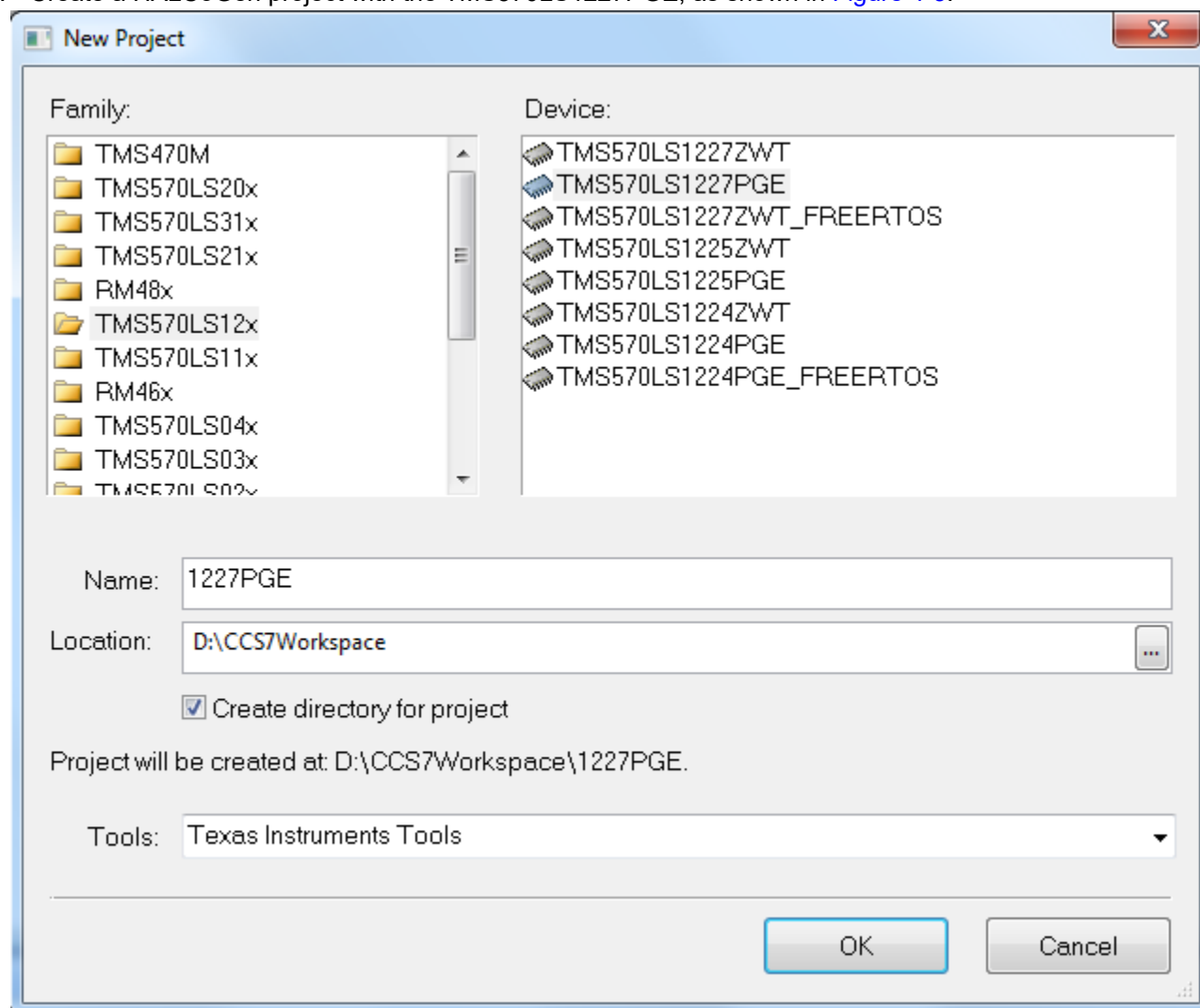


Figure 4-3. New Project

- Set VIM configurations as shown in Figure 4-4. FreeRTOS uses 2 interrupts: RTI Compare 0 and System Software Interrupt (SSI) (see Figure 4-5).

Enable interrupt channel: 2 - RTI Compare 0. Set ISR name as vPortPreemptiveTick

Enable interrupt channel: 21 - SSI. Set ISR name as vPortYieldWithinAPI.

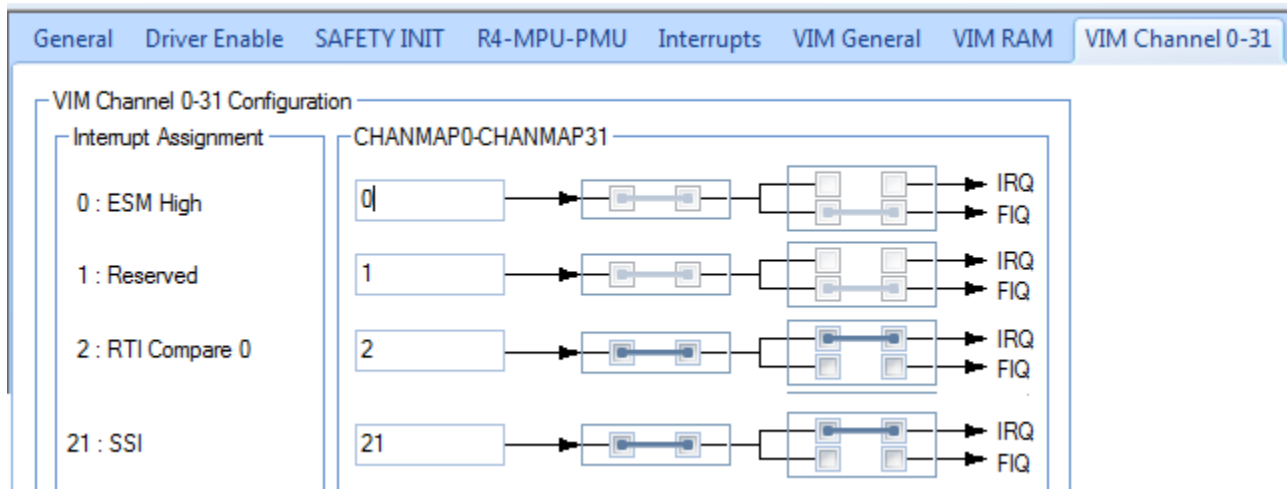


Figure 4-4. VIM Configuration

TMS570LS1227PGE

PINMUX

RTI

GIO

SCI

LIN

SCI2

MIBSPI1

MIBSPI3

SPI4

MI

General

Driver Enable

SAFETY INIT

R4-MPU-PMU

Interrupts

VIM General

VIM RAM

VIM RAM : ISR Assignments

Base Address: 0xFFF82000

☒ VIM RAM Parity enable

0x00000000:PH	phantomInterrupt	0x00000040:15	adc1Group1Interrupt	0x00000080:31
0x00000004:00	esmHighInterrupt	0x00000044:16	can1HighLevelInterrupt	0x00000084:32
0x00000008:01	phantomInterrupt	0x00000048:17	phantomInterrupt	0x00000088:33
0x0000000C:02	vPortPreemptiveTick	0x0000004C:18	phantomInterrupt	0x0000008C:34
0x00000010:03	rtiCompare1Interrupt	0x00000050:19	crcInterrupt	0x00000090:35
0x00000014:04	rtiCompare2Interrupt	0x00000054:20	esmLowInterrupt	0x00000094:36
0x00000018:05	rtiCompare3Interrupt	0x00000058:21	vPortYieldWithinAPI	0x00000098:37
0x0000001C:06	rtiOverflow0Interrupt	0x0000005C:22	phantomInterrupt	0x0000009C:38

Figure 4-5. VIM ISR Assignments

6. FreeRTOS also uses the SVC exception. Enable the exception handler as vPortSWI, as shown in [Figure 4-6](#).

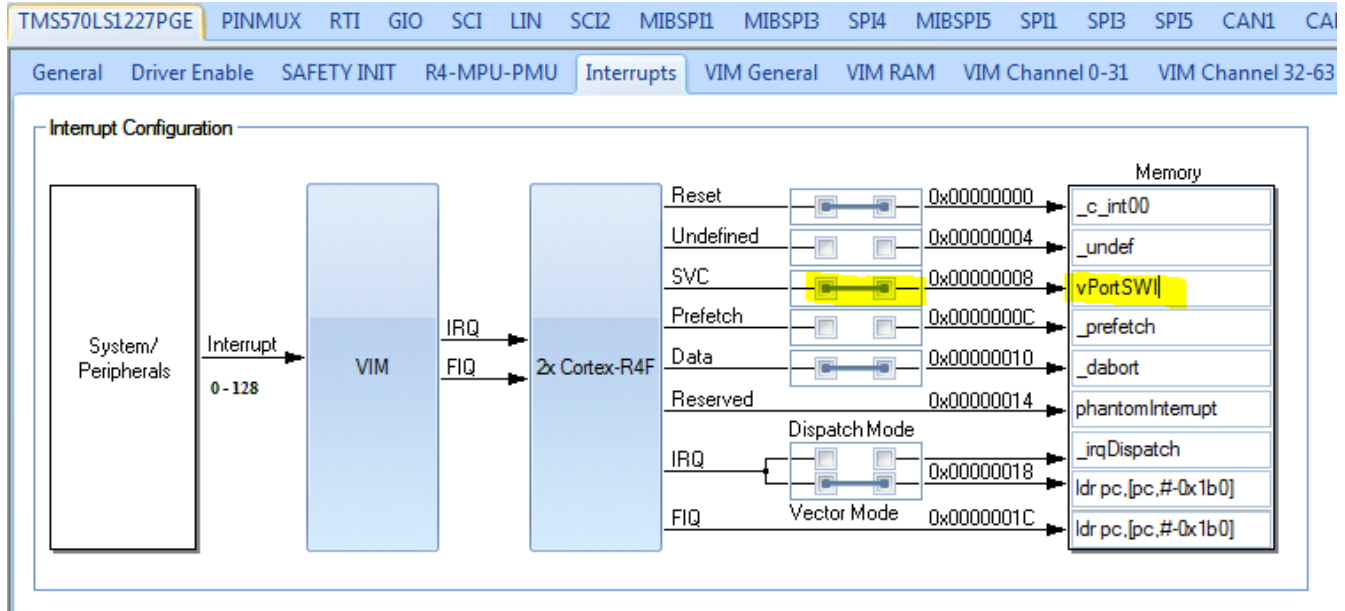


Figure 4-6. Enable PortSWI

- Disable the RTI driver and configure the other peripherals as required, then generate the code.
- Copy the OS files from the 1227ZWT project to the 1227PGE project. This includes the files starting with prefix "os_", FreeRTOS.h and FreeRTOSConfig.h.
- Make the following changes in the linker command file in 1227PGE project:

```
MEMORY
{
    VECTORS (X) : origin=0x00000000 length=0x00000020
    KERNEL (RX) : origin=0x00000020 length=0x00000000
    FLASH0 (RX) : origin=0x00000020 length=0x0013FFE0 - 0x8000
    STACKS (RW) : origin=0x08000000 length=0x00001500
    KRAM (RW) : origin=0x08001500 length=0x00000800
    RAM (RW) : origin=0x08001500 + 0x800 length=0x0002EB00 - 0x800

    /* USER CODE BEGIN (2) */
    /* USER CODE END */
}

/* USER CODE BEGIN (3) */
/* USER CODE END */

/*-----
/* Section Configuration

SECTIONS
{
    .intvecs : {} > VECTORS
    /* FreeRTOS Kernel in protected region of Flash */
    .kernelTEXT : {} > KERNEL
    .cinit : {} > KERNEL
    .pinit : {} > KERNEL
    /* Rest of code to user mode flash region */
    .text : {} > FLASH0
    .const : {} > FLASH0
    /* FreeRTOS Kernel data in protected region of RAM */
    .kernelBSS : {} > KRAM
    .kernelHEAP : {} > RAM
    .bss : {} > RAM
    .data : {} > RAM

    /* USER CODE BEGIN (4) */
    /* USER CODE END */
}
```

- The FreeRTOS port is now ready for the TMS570LS1227PGE device.

5 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (April 2018) to Revision A (October 2026)	Page
• Updated the numbering format for tables, figures, and cross-references throughout the document.....	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025