FreeRTOS User Guide

User Guide

FPGA-IPUG-02225-1.0

March 2023

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
RTOS	Real time operating system
SOC	System on chip
CPU	Central processing unit
BSP	Board support package
API	Application programming interface
CLINT	Core Local Interrupt controller

1. Introduction

Real-time operating system (RTOS) is an operating system that executes tasks with predictability and determinism. It is designed to execute tasks in a quick and effective manner. RTOS has a small footprint which makes it suitable for embedded systems. It is particularly important for mission or safety critical systems to behave predictably, hence RTOS is useful in such systems.

The FreeRTOS open-source project is one of the market leading RTOS available today. Lattice Semiconductor offers FreeRTOS for its RISC-V embedded processors. The FreeRTOS software is included in Lattice Propel™ 2022.1 and beyond. It includes the FreeRTOS kernel and set of libraries. The FreeRTOS software is distributed under MIT license.

2. Functional Descriptions

2.1. Features

The Lattice port of FreeRTOS software has the following features by default. These features are configurable to suit the application's needs. For more information, refer to the FreeRTOS Demo Customization section.

Table 2.1. FreeRTOS Features

Features	Default Value	Description
Pre-emptive scheduler	Enabled	The kernel scheduling algorithm is set as **Fixed Priority Pre-emptive Scheduling with Time Slicing**. The scheduler does not change the task's priority hence fixed priority (but does not prevent user from changing that). It supports task pre-emption where higher priority task (that is ready) can pre-empt the lower priority task. When scheduling ready tasks of equal priority, time slicing algorithm ensures switching between those tasks on every OS tick.
Use time slicing	Enabled	
Idle task should yield	Enabled	FreeRTOS automatically creates the idle task (with lowest priority) to ensure there is at least one task ready to execute. The Idle should yield feature ensures minimal amount of time is spent on the idle task. When there are other user tasks (same priority as idle task) that are ready to execute, the idle task will yield.
Task stack size	2 KBytes	The stack size is defined by configMINIMAL_STACK_SIZE. It is used to create the idle task and application tasks in Lattice FreeRTOS demo example.
Total heap size	10 KBytes	FreeRTOS heap size is set at 10 Kbytes, by default.
Memory allocation scheme	heap_2	Heap memory allocation scheme is set to heap_2. For more information, refer to the Memory Management section.

2.2. Memory Management

Each FreeRTOS object that is created requires allocation in the system memory. These objects include tasks, queues, and others. This memory allocation can be done either statically at compile time or dynamically during runtime.

For FreeRTOS objects that are created dynamically, they are allocated from the heap region using memory allocation or free related API namely pvPortMalloc() and vPortFree(). These API implementations are defined in the FreeRTOS portable layer (instead of core source code) as they can be application and system specific. Lattice provided FreeRTOS demo does not implement system specific memory allocation API. Instead, it uses the API provided by FreeRTOS heap_2 implementation.

The heap_2 implementation permits memory to be allocated and freed when unused. However, it does not coalesce adjacent free memory blocks. The heap_2 algorithm is demonstrated in Figure 2.1.

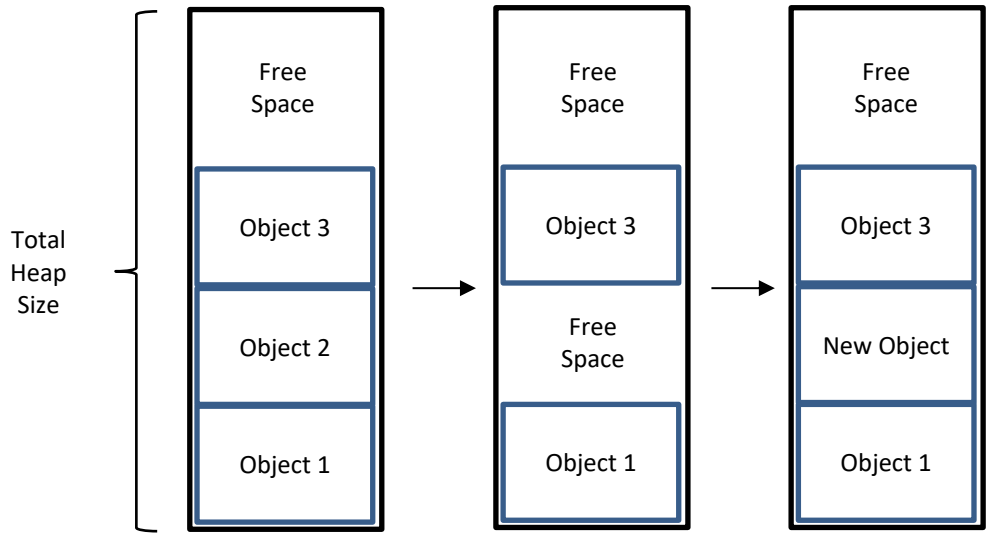


Figure 2.1. heap_2 Memory Allocation

From Figure 2.1:

- Three objects are created and allocated with memory space in the heap region. Free spaces are available at the top of the heap.
- Object 2 is deleted, and its corresponding space is freed up in the heap.
- A new object is created (which has an identical size as object 2) and the memory allocation algorithm ensures that the previously freed-up space is reused in this case.

Heap 2 is suitable for applications that create and delete objects repeatedly, provided the object size does not change. FreeRTOS provides other heap memory allocation algorithms. Further information can be found in the documentation from FreeRTOS.org.

2.3. Supported System and Device

Table 2.2 shows the FreeRTOS system requirements and supported devices.

Table 2.2. FreeRTOS System Requirements and Support Devices

Attribute	Description
RISC-V Processor Variant	RISC-V RX[1]
FPGA Families	Avant, MachXO5™-NX, Certus™-NX, CertusPro™-NX, CrossLink™-NX
System Memory	64 Kbytes[2]
Propel Version	2022.1 and above

Notes:

1. FreeRTOS is ported to support only RISC-V RX processor variant.
2. System memory size depends on the FreeRTOS applications e.g., number of tasks created, total heap size, and other factors. In Propel 2022.1, the FreeRTOS demo example size is 34 Kbytes.

3. Lattice Propel FreeRTOS Demo Example

This section provides information on how to generate the FreeRTOS demo example using Lattice Propel software, the demo source code walk-through, and the user configurable options. This tutorial uses the CertusPro-NX Versa board as an example. Boards with other FPGA families as highlighted in Table 2.2 can be used as well.

3.1. Generating the FreeRTOS Demo

To create the FreeRTOS demo example, users need to first create the template SOC project, followed by the software project. The following sections provide the information:

3.1.1. Creating FreeRTOS Template SOC Project

1. Launch Proper Builder.
2. Choose **File > New Design** from the Lattice Propel Builder Menu bar.
3. Select SoC Project type, provide a project name and the project workspace location. Figure 3.1 shows an example of creating a new system design.

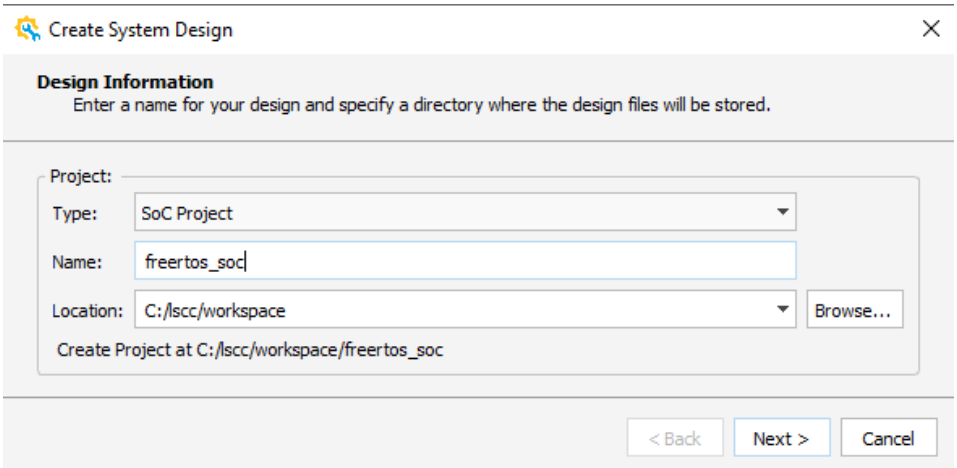


Figure 3.1. Create System Design

4. Click **Next** and the Configure Propel Project wizard opens.

5. In the Configure Propel Project wizard, make selections as shown in Figure 3.2:
   a. Select the **Family**, **Device**, **Package**, and **Speed** according to the board.
   b. Select **RISC-V RX** in the Processor category.
   c. Select **FreeRTOS** Project in the Templates category.

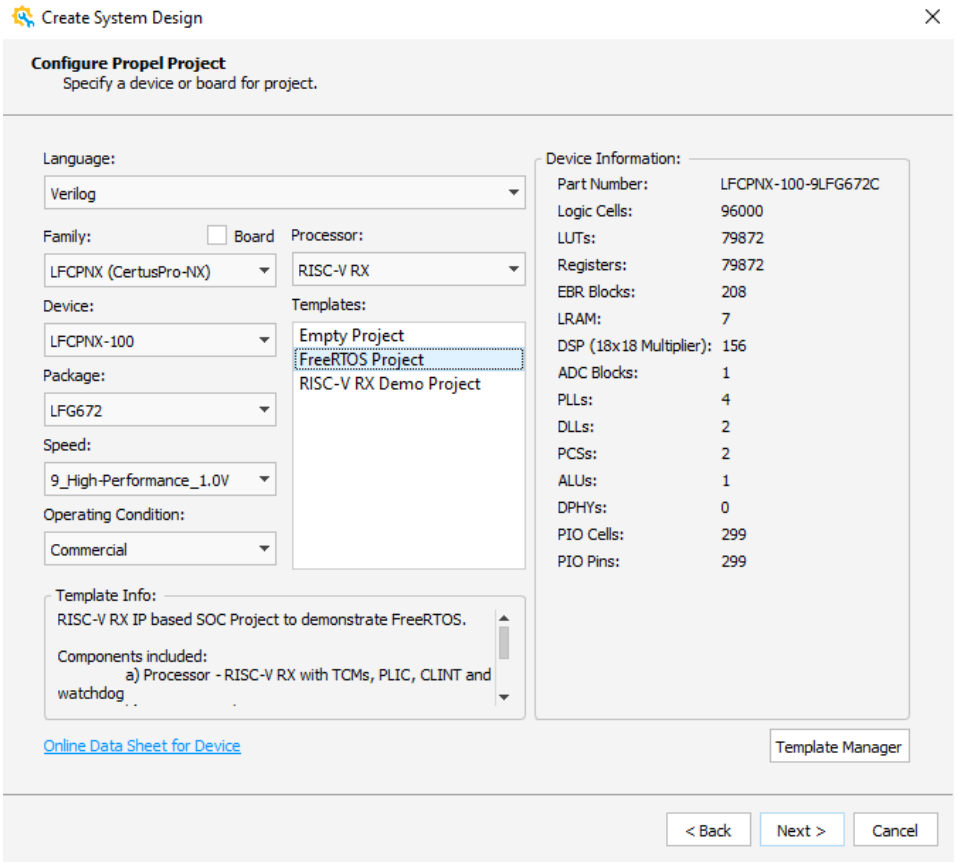


Figure 3.2. Configure Propel Project for FreeRTOS

6. Click **Next** to review the Project Information.
7. Click **Finish** to create the template project.
8. Click **Design > Generate** from the Propel Builder menu.
9. Click **Design > Run Radiant** to launch the Lattice Radiant software. Do not exit the Propel Builder.

3.1.2. Compiling Project in Radiant

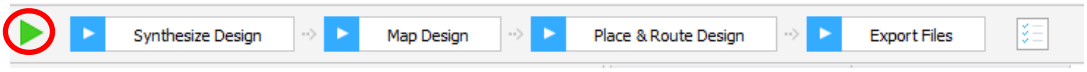
In Radiant software, continue the following steps to compile the project into bitstream file:

1. Click **File > New > File** from Radiant menu.
2. Select **Post-Synthesis Constraint Files**
3. Provide name to the file: cpnx_versa.pdc
4. Click **New**.
5. Copy and paste the following constraints into the cpnx_versa.pdc file. Save when it is done.

```
ldc_set_location -site {R5} [get_ports {s0_gpio[0]}]
ldc_set_location -site {R4} [get_ports {s0_gpio[1]}]
ldc_set_location -site {R8} [get_ports {s0_gpio[2]}]
ldc_set_location -site {R9} [get_ports {s0_gpio[3]}]
```

```
ldc_set_location -site {U8} [get_ports {s0_gpio[4]}]
ldc_set_location -site {R7} [get_ports {s0_gpio[5]}]
ldc_set_location -site {R6} [get_ports {s0_gpio[6]}]
ldc_set_location -site {P8} [get_ports {s0_gpio[7]}]
ldc_set_location -site {M9} [get_ports s1_uart_txd_o]
ldc_set_location -site {L8} [get_ports s1_uart_rxd_i]
ldc_set_location -site {N9} [get_ports rstn_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports rstn_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports s1_uart_rxd_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports s1_uart_txd_o]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports {s0_gpio[*]}]
```

6. Click Run All to start the compilation.



3.1.3. Creating FreeRTOS Software Project

Go back to Propel Builder software.

1. Click **Design > Run Propel** from the Propel Builder menu.
2. In Lattice Propel Launcher, select the workspace as shown in Figure 3.3, and then click **Launch**. The Load System and BSP dialog box appears.

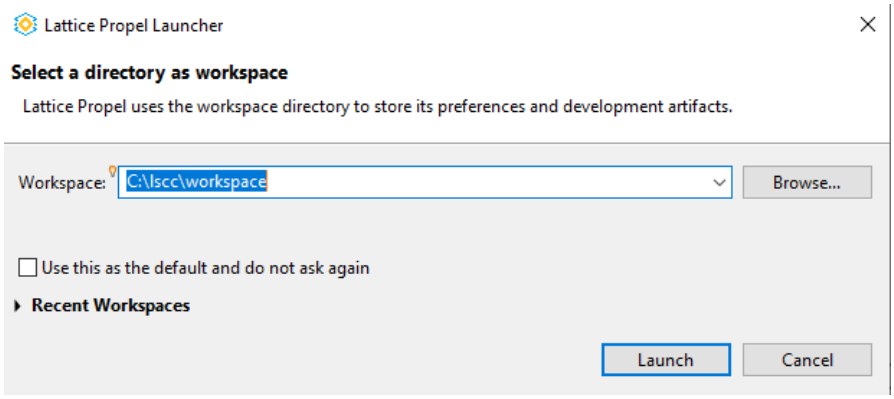


Figure 3.3. Lattice Propel Launcher

3. Type in freertos_sw in the Project Name field.
4. Click **Next**.
5. Maintain default settings in the Lattice Toolchain Setting page.
6. Click **Finish**.
7. Click **Project > Build Project**.

8. Build the project without errors as shown in Figure 3.4.

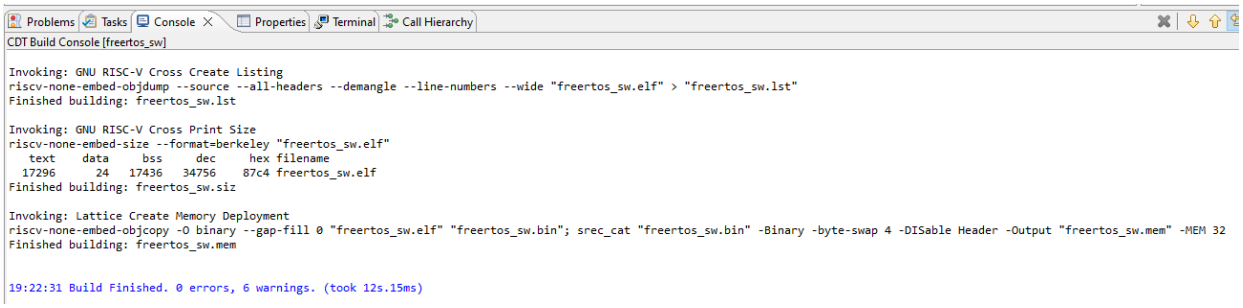


Figure 3.4. Successful Build of FreeRTOS Software Project

3.1.4. Running FreeRTOS on the Board

1. Setup the CertusPro-NX Versa board using the following steps:
   a. Connect the power cable to J44.
   b. Connect the USB cable from computer to J34.
   c. Using jumpers, short pins 1 and 2 on J32. Repeat the same for J33. This step is for enabling the UART interface.
   d. Turn on switch SW6.
2. Programming the Bitstream into FPGA using the following steps:
   a. In Radiant software, click **Tools > Programmer** from the menu. The Programmer is set up automatically as identical to Figure 3.5.

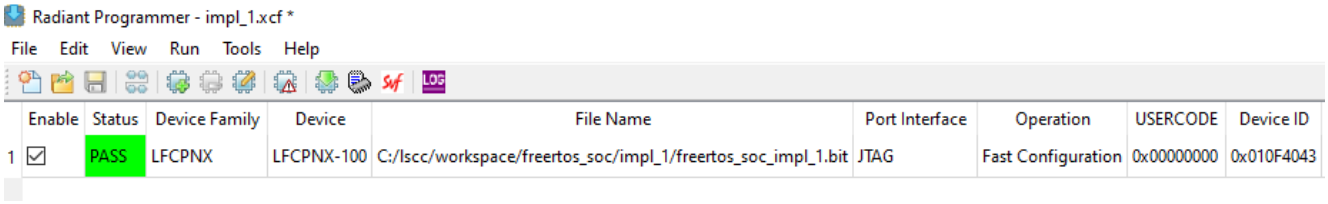


Figure 3.5. Radiant Programmer Settings

   b. In Programmer tool, click **Run > Program Device**.
3. Run FreeRTOS Software via Propel Debugger:
   a. In Propel, click **Run > Debug Configurations**.
   b. Select **GDB OpenOCD Debugging** from list on the left, and double-click on it as shown in Figure 3.6.

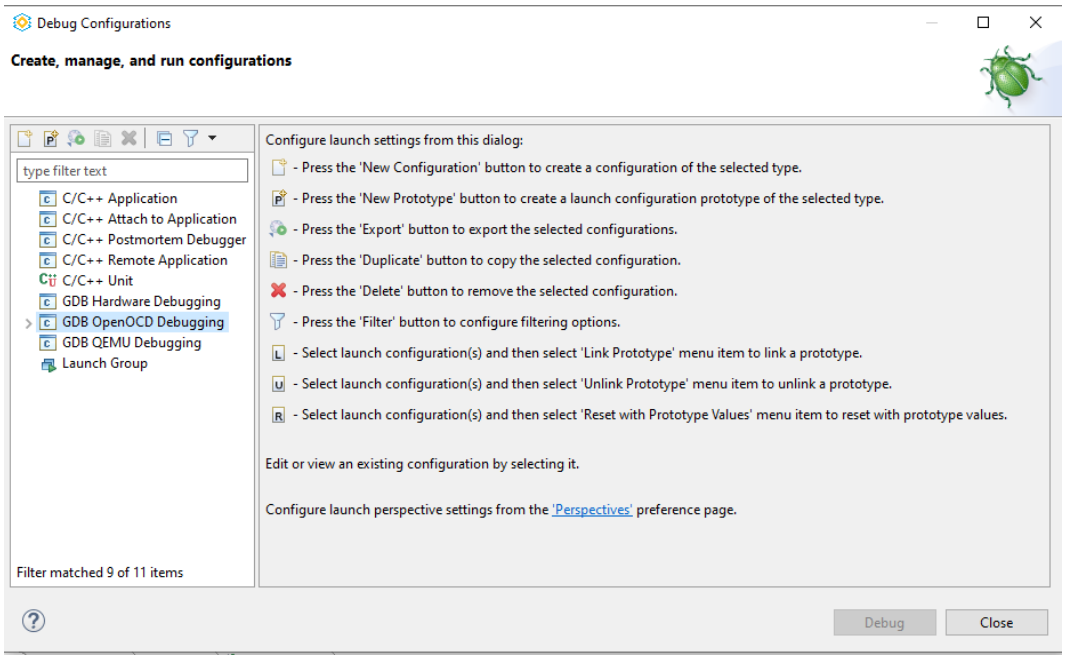


Figure 3.6. Create OpenOCD Debug Configuration

c. Maintain the default settings in the Main tab as shown in Figure 3.7.

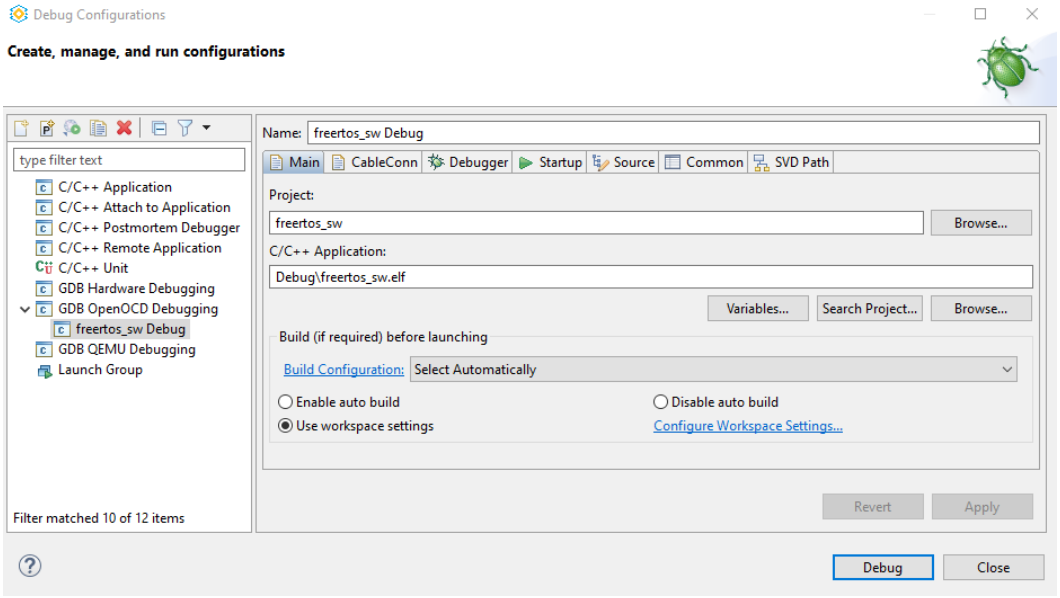


Figure 3.7. Default Settings for Main Tab

d. In the CableConn tab, perform the following:

i. Click **Detect Cable** and make sure that the proper cable is displayed.

ii. Click **Scan Device** and make sure LFCPNX-100 is displayed.

iii. Click **Apply**.

iv. Click **Debug**.

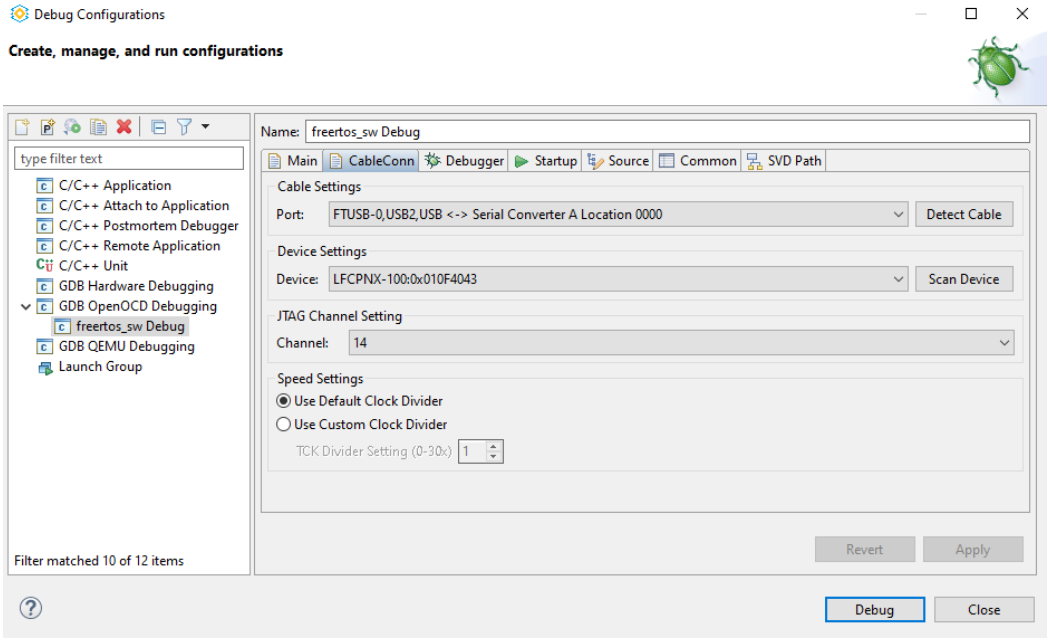


Figure 3.8. Setup Cable Connection for Debugging on LFCPNX-100

e. Click Switch when asked to confirm perspective switch.

f. In Lattice Propel, click the Terminal tab located at bottom of the GUI.

g. Click the Open a Terminal icon as shown in Figure 3.9.

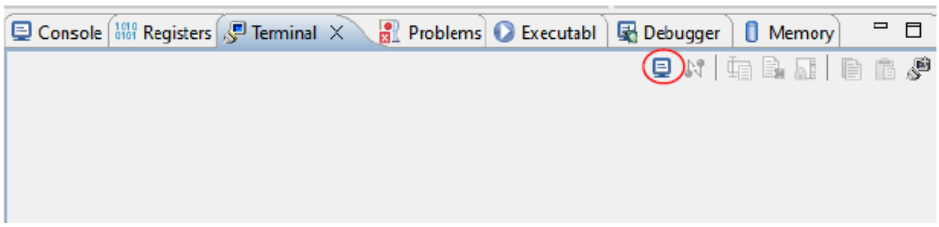


Figure 3.9. Open a Terminal

h. Setup the Serial Terminal with the following:

   i. **Choose terminal: Serial Terminal**

   ii. **Serial Port**: COMXX (depends on the COM port number on your computer)

   iii. **Baud rate**: **115200**

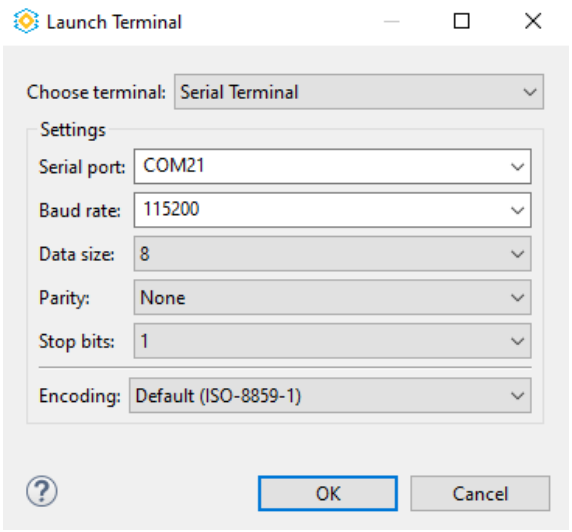


Figure 3.10. Serial Terminal Setup

i. Click **OK**.

j. In Lattice Propel, click **Run > Resume** from the menu.

k. Observe that FreeRTOS starts running and messages are printed in the serial terminal as shown in Figure 3.11.

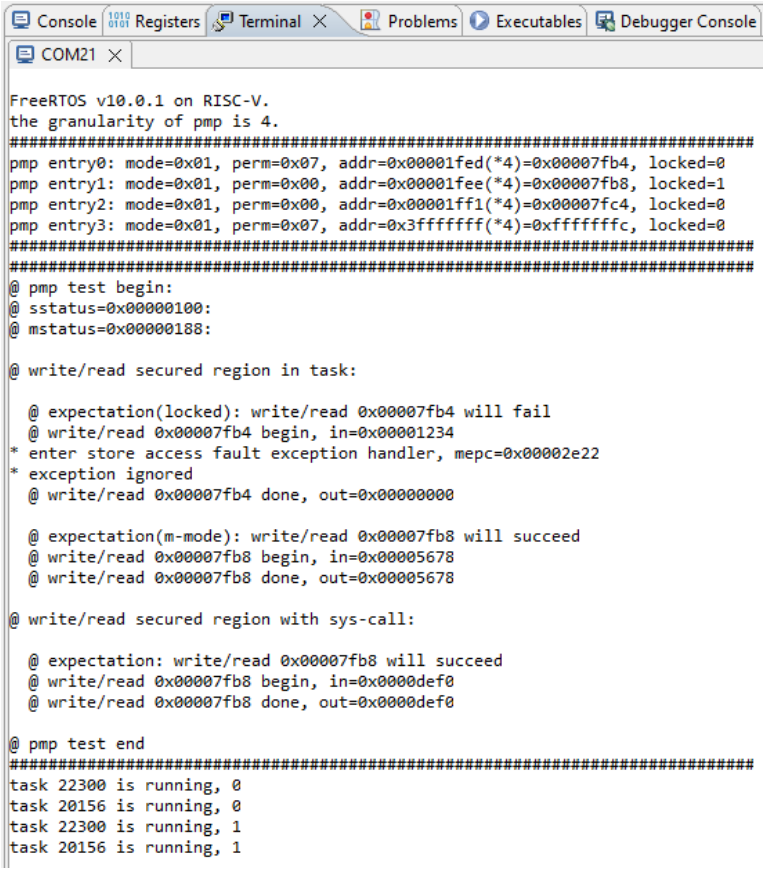


Figure 3.11. FreeRTOS Running with Serial Terminal Messages

l. From Figure 3.11, it can be observed that the FreeRTOS demo creates two tasks that perform the following:

UART Task

i. Print its task ID to serial terminal.

ii. Suspend and resume the LED task on intervals.

LED Task

i. Perform the PMP test (physical memory protection) on initial execution.

ii. Print its task ID to serial terminal.

iii. Light up the LED in a sequential pattern.

3.2. FreeRTOS Demo Source Code Structure

This section describes the source code structure in the FreeRTOS demo. Table 3.1 shows the details of each source file in the demo project. The source code can be viewed in Propel SDK.

Table 3.1. FreeRTOS Demo Source Code Structure

Source File	Description
src	Parent directory of all source files
|_bsp |_driver |_sys_platform.h	Board support package (BSP) directory contains the SOC IP drivers and the system platform header. BSP is automatically generated by Propel.
|_config |_config.h	Header file that defines various RISC-V RX CPU privilege modes. The default mode is set to Machine mode.
|_FreeRTOS |_include |_croutine.c |_event_groups.c |_list.c |_queue.c |_stream_buffer.c |_tasks.c |_timers.c |_portable |_MemMang |_port_asm.S |_port.c |_portmacro.h	Include directory containing all the FreeRTOS header files. FreeRTOS kernel source files from open-source project. These source files are not modified for Lattice demo. The portable layer of Lattice demo which contains platform specific implementation. MemMang contains the code that implements the heap_2 algorithm. FreeRTOS Demo Portable Layer describes the Lattice portable layer in detail.
|_FreeRTOSConfig.h	Lattice FreeRTOS demo customization is defined in this configuration file. FreeRTOS Demo Customization describes the customization in detail.
|_main.c	The main application code that creates FreeRTOS tasks and starts the scheduler.
|_os.c	Defines the exception service routines and FreeRTOS task API wrappers. FreeRTOS Demo API Wrappers describes the Lattice API wrappers in detail.
|_syscall.h |_syscall.S	Defines the system call implementation to support non machine mode operations.

3.3. FreeRTOS Demo Portable Layer

Platform specific code is implemented in the portable layer of FreeRTOS source tree. This section describes these implementations.

3.3.1. Task Switching

Task switching is implemented in switch_task(). The FreeRTOS API vTaskSwitchContext () is called to obtain the next task context. Once the next task is decided, the following operations happen before switching to the new task:

1. Load the new task context from its stack into the CPU registers.
2. Set the mstatus register bits depends on which RISC-V privilege mode as shown in Table 3.2.

Table 3.2. Privilege Mode Settings

mstatus Register Bit	User Mode[1]	Supervisor Mode[1]	Machine Mode[1]
MPIE	1	1	1
MPP	0	1	3

Note:

1. The privilege mode is statically defined in config.h.

3.3.2. Tick Interrupt

The FreeRTOS tick interrupt is driven by the CLINT Timer. In this demo, the timer is configured to generate the tick interrupt at 1 second intervals. The timer period is defined using portTICK_PERIOD_MS in the portmacro.h file.

3.4. FreeRTOS Demo Customization

FreeRTOS permits configuration or customization by changing definitions within the FreeRTOSConfig.h file. Table 3.3 lists the configuration that the Lattice FreeRTOS demo applies. The user can modify the configuration to match the application needs.

Table 3.3. FreeRTOS Demo Configuration

Configuration Name	Default Value	Description
configUSE_PREEMPTION	1	Use pre-emptive scheduler: Kernel will switch context during each tick interrupt
configUSE_TIME_SLICING	1	Scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick interrupt
configUSE_TICKLESS_IDLE	0	Keep the tick interrupt running at all times (not supporting low power tick-less mode)
configUSE_IDLE_HOOK	0	Not using callback function that will be called by Idle task
configUSE_TICK_HOOK	0	Not using callback function that will be called during each tick interrupt
configCPU_CLOCK_HZ	100000000	CPU clock is 100 MHz (per system in Propel Builder)
configTICK_RATE_HZ	32768	Tick interrupt frequency. Equal to CLINT timer clock which is 32 KHz
configMAX_PRIORITIES	5	Tasks can be assigned a priority from 0 (lowest) to 4 (highest)
configMINIMAL_STACK_SIZE	512	Minimum recommended stack size (in words) 512*4 = 2Kbytes
configTOTAL_HEAP_SIZE	10*1024	10Kbytes of total heap can be used for tasks creation
configMAX_TASK_NAME_LEN	16	Maximum number of characters that can be used for the name of a task
configUSE_TRACE_FACILITY	0	Execution visualization and tracing feature (disabled)
configUSE_16_BIT_TICKS	0	Use to set tick count variable to unsigned 16-bit type. Usually enable this for 8/16-bit microcontrollers (disabled)
configUSE_CO_ROUTINES	0	Not using co-routines (Co-routines are light weight tasks that save RAM by sharing a stack, but have limited functionality)
configMAX_CO_ROUTINE_PRIORITIES	2	Maximum priority of co-routine tasks that can be assigned (disabled)
configIDLE_SHOULD_YIELD	1	This ensures a minimum amount of time is spent in the Idle task when application tasks (with same priority as Idle task) are available to run.
INCLUDE_vTaskPrioritySet	1	Include Task Priority Set API
INCLUDE_uxTaskPriorityGet	1	Include Task Priority Get API
INCLUDE_vTaskDelete	0	Exclude Task Delete API
INCLUDE_vTaskCleanUpResources	0	Exclude Task Clean Up Resources API
INCLUDE_vTaskSuspend	1	Include Task Suspend API
INCLUDE_vTaskDelayUntil	1	Include Task Delay Until API
INCLUDE_vTaskDelay	1	Include Task Delay API
INCLUDE_xTaskGetCurrentTaskHandle	1	Include Task Get Current Handle API

3.5. FreeRTOS Demo API Wrappers

This section describes the API wrappers that are included in Lattice FreeRTOS demo. These API wrappers are defined in os.c file as shown in Table 3.1. The standard FreeRTOS API calls are not discussed here but can be found in FreeRTOS references.

3.5.1. task_create()

Creates a new FreeRTOS task. This API is based on FreeRTOS xTaskCreate() with the following modifications:

- Task_create() always creates task that point to an entry (task_entry()) which handles the privilege mode setup before executing the user task.
- Actual user task is passed to xTaskCreate via the pvParameter argument. This restricts the task_create API to accept any user task parameter.
- Task_created() assigns fixed priority defined by tskIDLE_PRIORITY + 1.
- Task_created() assigns fixed stack size defined by configMINIMAL_STACK_SIZE.

Table 3.4 shows the API arguments and return value.

Table 3.4. task_create API

Arguments/Return Value	Description
pvTaskCode	C function that user defined (the name of the function)
name	Descriptive name for the task (in string) for debugging aid
handle	Task handle that can be used to reference the created task (example changing priority or deleting task)
Returned Value	1. pdPASS This indicates that the task has been created successfully. 2. pdFAIL This indicates that the task has not been created.

3.5.2. task_delay()

Set the calling task of this API into Blocked state for a fixed number of tick interrupts. This API is based on FreeRTOS vTaskDelay(). This wrapper is used to handle machine and non-machine modes. The latter case uses system call to the vTaskDelay().

Table 3.5 shows the API arguments and return value.

Table 3.5. task_delay API

Arguments/Return Value	Description
tick	The number of tick interrupts that the calling task will remain in the Blocked state before being transitioned back into the Ready state.
Returned Value	None

3.5.3. task_suspend()

Suspend any task where the task will never get any processing time regardless of its priority. This API is based on FreeRTOS vTaskSuspend(). This wrapper is used to handle machine and non-machine modes. The latter case uses system call to the vTaskSuspend().

Table 3.6 shows the API arguments and return value.

Table 3.6. task_suspend API

Arguments/Return Value	Description
task	Handle the task being suspended. Passing a NULL handle will cause the calling task to be suspended.
Returned Value	None

3.5.4. task_resume()

Resume a suspended task. This API is based on FreeRTOS vTaskResume(). This wrapper is used to handle machine and non-machine modes. The latter case uses system call to the vTaskResume().

Table 3.7 shows the API arguments and return value.

Table 3.7. task_resume API

Arguments/Return Value	Description
task	Handle to the suspended task to be resumed.
Returned Value	None

3.6. Known Issue

This section describes the known issue that exists in the Propel 2022.1 FreeRTOS demo.

Problem

If multiple tasks are assigned with different priorities, the FreeRTOS scheduler is unable to start. This renders none of the tasks get executed.

Workaround

Add the call to task_entry in the definition of xPortStartScheduler() in port.c file as shown in Figure 3.12.

```
BaseType_t xPortStartScheduler( void )
{
    global_context()->sp = (reg_t)&s_stack[INIT_STACK_SIZE - 1];
    w_mscratch((reg_t)global_context());
    clint_start_timer(portTICK_PERIOD_MS * 1000,
                ePRIV_M,
                vPortMtipHandler,
                NULL);
    clint_install_soft_isr(soft_irq_handler, NULL);

    /* enable interrupt */
    __asm volatile( "csrs mstatus,8" );

    /* switch to tasks created */
    task_entry(NULL);

    for(;;);

    return 0;
}
```

Figure 3.12. Workaround for Task Priority Issue

Appendix A. FreeRTOS Reference Resources

This section lists resources available online for FreeRTOS reference.

- FreeRTOS Quick Start Guide
- FreeRTOS API Reference
- FreeRTOS Reference Manual
- FreeRTOS Hands On Tutorial Guide

Appendix B. Resource Utilization

Table B.1 shows the FreeRTOS Demo Template design configuration and resource utilization. This design is generated using Propel Builder 2022.1 and Radiant 2022.1.

Table B.1. Resource Utilization for FreeRTOS Demo Template Design

Configuration	LUTs	Registers	EBRs
FreeRTOS Demo Template Design consists of the following IP: • RISC-V RX CPU • UART • GPIO • System Memory • Tightly Coupled Memory • AXI Interconnect • AXI to AHB-lite Bridge • AXI to APB Bridge • PLL • OSC	7113	5645	66

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/en/Support/AnswerDatabase.

Revision History

Revision 1.0, March 2023

Section	Change Summary
All	Initial release.

www.latticesemi.com