ID 9L02I: Software Development with an Open Source Real-Time Operating System
RX600 Series

Description: As embedded applications are getting complex, the need for a scheduler to help coordinate the tasks that make up the application is increasing. With an RTOS, an embedded designer can better integrate independent modules with different processing requirements. RTOS helps multiple designers to work simultaneously on different tasks. Code re-use and improved efficiency are other benefits of using a kernel. In this lab, we will cover RTOS concepts including task priorities, scheduling techniques, inter-task communications, and synchronization of tasks and interrupts. We will use FreeRTOS to examine queues, semaphores, and mutexes. We will experiment with priority inversion and inheritance. Lastly, we will design a gatekeeper task that manages how a resource can be shared between other tasks and interrupts.

Lab Objectives
1. Experience hands-on RTOS features that you all heard about.
2. Find out freely available RTOS tools.
3. Learn more advanced RTOS concepts such as priority inversion and inheritance.

Lab Materials
Please verify you have the following materials at your lab station:
- RDKRX63N
- USB debugger cable
- Jumper wire
- e2Studio Version 1.1.0.25
- RX Toolchain Version 1.02.00
- Tera Term Version 4.71
- RX_FreeRTOS lab workspace

Skill Level
1. Real-time embedded programming.
2. C language.

Lab Sections
0 Lab Setup .................................................................................................................................................. 2
1 Lab Exercise 1: Task Creation, Priorities, Task States, Idle Task .................................................. 10
2 Lab Exercise 2: Queue Management .................................................................................................. 12
3 Lab Exercise 3: Interrupt Management ................................................................................................. 15
4 Lab Exercise 4: Resource Management .............................................................................................. 18
5 Lab Exercise 5: Priority Inversion & Inheritance ............................................................................... 20
6 Lab Exercise 6: Gatekeeper Tasks ........................................................................................................ 22

Time to Complete Lab
90 Minutes
0 Lab Setup

Overview
In this section you will configure your RDKRX63N board and learn the directory structure of the lab.

Configuring the Lab
The lab setup and the major components of the RDKRX63N are shown in the Figure 1.

![Figure 1 RDKRX63N and Lab Setup](image)

Step 0.1 Make sure SW5 on RDKRX63N is set as follows.

<table>
<thead>
<tr>
<th>SW5/MCU MODE</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 1 = OFF</td>
<td></td>
</tr>
<tr>
<td>Pin 2 = OFF</td>
<td></td>
</tr>
<tr>
<td>Pin 3 = OFF</td>
<td></td>
</tr>
<tr>
<td>Pin 4 = ON</td>
<td></td>
</tr>
</tbody>
</table>

Step 0.2 Connect the pin jumper wire to J8 pins 23 and 25 as shown in Figure 1.

Step 0.3 Connect USB debugger cable to debug port and the other end to PC. RDKRX63N will be powered by this cable.
Step 0.4 Connect USB/Serial adaptor to serial port and the other end to PC.

Step 0.5 Run Tera Term. There should be a shortcut on the Desktop. If not, go to All Programs>>Tera Term>>Tera Term to launch it.

Step 0.6 Select the “Serial” button and the correct serial port in New Connection window as shown in Figure 2. Your USB/Serial adaptor can be on a different port. To make sure go to Device Manager>>Ports and look for a USB Serial Port and its COM number.

![Figure 2 Tera Term New Connection Settings](image)

Step 0.7 On Tera Term, go to Setup>>Serial port and select 115200 for the baud rate. Use defaults for other settings.
Step 0.8 On Tera Term, go to Setup>>Window and select “Scroll buffer” box and enter 5000 for lines as shown in Figure 3.

Figure 3 Tera Term Window Setup
Directory Structure of the Lab

Figure 4 shows the lab directory structure. This is for your information only. The \src\folder contains the source code and has 4 major subfolders: freertos, r_bsp, r_glyph, and r_rspi_rx600. The lab exercise code is in the \src\folder. We will compile in and out several main.c files for each lab exercise. The src\freertos folder contains the FreeRTOS port to RX600 family of MCUs. The \src\r_bsp folder has Renesas board specific source code for the RDKRX63N. The src\r_glyph and src\r_rspi_rx600 folders are used for the graphical display.

Build and Go

Now you will launch e2Studio, build the lab code, connect to the RDKRX63N via the debugger, download the code and run it. During this lab, we will use different main.c files. They are named main1.c to main6.c. You will use e2Studio’s file include/exclude feature to add/remove main.c files. Initially, the project should open with main1.c file included.

Step 0.9 Run e2Studio. There should be a shortcut on the Desktop. If not, go to All Programs>>Renesas Electronics e2Studio>>Renesas e2Studio.

Step 0.10 Click on Browse and go to C:\Workspace\RX_FreeRTOS. Click OK to open the workspace.

Step 0.11 On the Project Explorer view (far left), expand the RX_FreeRTOS project. In src folder, you should see main1.c is included in the project. The other main.c files should be grayed out and not included.

Step 0.12 Go to Project>>Build All. When complete you should see ‘Build complete.’ in the console view.
Step 0.13 Right click on the RX_FreeRTOS project and select Debug As>>Renesas GDB Hardware Launch.

Step 0.14 If you see Renesas Hardware Debugging window, you need to setup the debug settings. Select Segger JLink from the list as shown below. Click OK.

Step 0.15 Now select R5F563NB from the MCU list as shown below. Click OK.
Step 0.16 If you see Confirm Perspective Switch window, click in “Remember my decision” box first and then click Yes as shown below.

![Confirm Perspective Switch Image]

Step 0.17 Click pull down arrow next to Debug icon to run debug configurations as shown below.

![Debug Configurations Image]
Step 0.18 In Debug Configurations window, go to Startup tab and uncheck the “Set breakpoint at” box as shown below. Click Apply and Debug.

Step 0.19 At this point, you should be connected to the debugger.

Step 0.20 Click Restart icon to run the code.
Step 0.21 Observe that the LCD display on RDKRX63N. It should show a Renesas logo and display “FreeRTOS lab, Exercise 1”. The Tera Term window should print out the following messages.

![Tera Term output](image)

Step 0.22 Click Suspend icon to stop the code.

![Warning message](image)

We will use Tera Term throughout this lab. You can clear the output with Edit>>Clear buffer command. This will help you with locating the start of debug output. Try it now.

Also, LEDs on the RDKRX63N are controlled by corresponding tasks to provide a visual perspective of running tasks. The table below shows which LED turns on when a task is in running state.

<table>
<thead>
<tr>
<th>LED</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED5</td>
<td>Task1</td>
</tr>
<tr>
<td>LED6</td>
<td>Task2</td>
</tr>
<tr>
<td>LED7</td>
<td>Task3</td>
</tr>
<tr>
<td>LED8</td>
<td>Task4</td>
</tr>
<tr>
<td>LED9</td>
<td>Task5 (Run Time Statistics)</td>
</tr>
<tr>
<td>LED10</td>
<td>Idle Task</td>
</tr>
</tbody>
</table>
1 Lab Exercise 1: Task Creation, Priorities, Task States, Idle Task

Overview
In this section we will cover task creation, task priorities, states that task can be in FreeRTOS. We will also cover task delay and timing API calls and how they can put a task in blocked state allowing idle task to run.

Step 1.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 1.2 On the Project Explorer view (far right in Renesas Debug perspective), double click on main1.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 1.3 As you can see initially there are two tasks created: Task1 and Task2. FreeRTOS uses xTaskCreate() API to create tasks. Read the comments next to this API to understand the parameters and what they do.

Step 1.4 Both Tasks1 and Task2 runs at priority 1 and they do not block. What is the expected debug output? How do you think tasks are scheduled to run?

Step 1.5 Click Restart icon to run the code.

Step 1.6 Observe LEDs. Which ones do turn on? What is the run time statistics information?

Step 1.7 Click Suspend icon to stop the code.

Step 1.8 Change Task2 priority from 1 to 2.

Step 1.9 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 1.10 Observe LEDs, run time statistics, and debug output. Why is only Task2 running?

Step 1.11 Click Suspend icon to stop the code.
Step 1.12 Change Task2 priority back to 1. Set mainUSE_LOOP_DELAY to 0 in macro definitions.

Step 1.13 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 1.14 Observe LEDs. Are Task1 and Task2 still running? Do LED5 and LED6 turn on? Which LED is on most of the time? Why?

Step 1.15 Click Suspend icon to stop the code.
Step 1.16 Set mainUSE_PERIODIC_TASK to 1 in macro definitions. This will include Periodic task in our exercise. What is the expected debug printout? Why?

Step 1.17 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 1.18 Observe LEDs and explain the debug output. Which task does start running first? Why Periodic task does run twice between Task1 and Task2?

You may want to stop the code by Suspend icon, clear the debug output by Edit>>Clear buffer on Tera Term, and re-start the code by Restart icon to see the first running task.

Step 1.19 Click Suspend icon to stop the code.

2 Lab Exercise 2: Queue Management

Overview
Applications that use FreeRTOS consist of independent tasks each is effectively a small program in its own right. For a whole application to work, these individual tasks must communicate with each other. Queues form the main mechanism used by all FreeRTOS communication and synchronization tools. In this lab exercise, we will look at queue creation, and sending to and receiving from a queue. We will experiment with queue blocking and effects of task priorities when writing to and reading from a queue.

Step 2.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 2.2 On the Project Explorer view (far right in Renesas Debug perspective), right click on main1.c and select Resource Configurations>>Exclude from build. Select HardwareDebug box and click on OK. This will exclude main1.c from the project.

Step 2.3 On the Project Explorer view (far right in Renesas Debug perspective), right click on main2.c and select Resource Configurations>>Exclude from build. Unselect HardwareDebug box and click on OK. This will include main2.c into the project.
Step 2.4  On the Project Explorer view (far right in Renesas Debug perspective), double click on main2.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 2.5  There are two sender tasks: Sender1 and Sender2. These tasks run at priority 1. Sender1 turns on LED5 and Sender2 turns on LED6. There is one Receiver task. It runs at priority 2 and turns on LED7 when it runs. A queue is created with length of 5 and size of long. Sender1 sends value 100 and Sender2 sends value 200 to the queue. These values are passed as parameters to the sender tasks when they are created. When running, receiver task first prints out the number of items in the queue. Then it receives the data from the queue and prints its numeric value. What is expected debug output?

Step 2.6  Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 2.7  Why is the number of items in queue always zero? Observe the LEDs. Which one is turned on most of the time or brighter? Why?

Step 2.8  Capture a run time statistics. If you can not see them printed on debug console, put a breakpoint in Task5 on line vTaskDelay(). When you hit the breakpoint, put another breakpoint in vSenderTask on line taskYIELD(). Click on Resume icon couple times to allow the statistics to print out completely.

Step 2.9  Does the statistics information match the LEDs when the code was running freely? Why is Receiver task running the most of the time?

Step 2.10  On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 2.11  Remove all breakpoints.

Step 2.12  Set the priority of Receiver task to 0. What do think the debug output will be?

Step 2.13  Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code. Quickly, click Suspend icon to stop the code and scroll to top of the debug output.
Step 2.14 Do you see below output? Why sender tasks can not send messages after the 5th one. Why is there no reception of messages? Did you see Receive task LED (LED7) turned on? You can also look at the run time statistic to verify if Receiver task is running.

![Tera Term - COM4 VT](image)

Step 2.14
- Sending = 200
- Sending = 100
- Sending = 200
- Sending = 100
- Sending = 200
- Sending = 100

Could not send to the queue.
- Sending = 200
- Could not send to the queue.
- Sending = 100
- Could not send to the queue.
- Sending = 200
- Could not send to the queue.

Step 2.15 Set the priority of Receiver task to 1. What is the expected debug output?

Step 2.16 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code. Let the lab exercise run several seconds.

Step 2.17 Click Suspend icon to stop the code.

Step 2.18 What is the maximum number of messages in the queue? Why is there more than one message in the queue?

Step 2.19 Comment out below two lines of code in main2.c file. By doing this you are not creating sender tasks. Ignore the compiler warning. What is the expected debug output?

```c
xTaskCreate( vSenderTask, "Sender1", 1000, ( void * ) 100, 1, NULL );
xTaskCreate( vSenderTask, "Sender2", 1000, ( void * ) 200, 1, NULL );
```

Step 2.19
- xTaskCreate( vSenderTask, "Sender1", 1000, ( void * ) 100, 1, NULL );
- xTaskCreate( vSenderTask, "Sender2", 1000, ( void * ) 200, 1, NULL );

Step 2.20 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.
Step 2.21 Do you see below output? Why is Receiver task running even though there are no sender tasks?

![Tera Term - COM4 VT](image)

Step 2.22 Click Suspend icon to stop the code.

3 Lab Exercise 3: Interrupt Management

Overview
Tasks are a good way to process periodic and continuous functions. In a real world environment, embedded systems must also process events such as reception of an Ethernet frame, a button push, or exceeding a threshold. Embedded systems have to service events that originate from multiple sources all of which have different processing and response time requirements. In each case a judgment has to be made for best detection methods and how to process them. Events are detected either by polling or Interrupt Service Routines (ISR). In this lab exercise, we will use semaphores to synchronize interrupts with tasks. We will first cover binary semaphores and then counting semaphores.
Step 3.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 3.2 On the Project Explorer view (far right in Renesas Debug perspective), right click on main2.c and select Resource Configurations>>Exclude from build. Select HardwareDebug box and click on OK. This will exclude main2.c from the project.

Step 3.3 On the Project Explorer view (far right in Renesas Debug perspective), right click on main3.c and select Resource Configurations>>Exclude from build. Unselect HardwareDebug box and click on OK. This will include main3.c into the project.

Step 3.4 On the Project Explorer view (far right in Renesas Debug perspective), double click on main3.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 3.5 There are 2 tasks created: Handler task and Periodic task. Handler task is synchronized with an external event via a binary semaphore. Periodic task generates an interrupt every 500ms emulating an event. Semaphore give operation is implemented within the interrupt routine. This causes the Handler task to unblock and process the event. What is expected debug output?

Step 3.6 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code. Quickly, click Suspend icon to stop the code and scroll to top of the debug output.
Step 3.7 You should see below output. Observe the very first line. Why is Handler task the first task to run? In rest of the debug output, why does Handler task run in between Periodic task? Can you tell the sequence of events?

![Tera Term - COM4 VT](image)

**Handler task** - Processing event.
**Periodic task** - About to generate an interrupt.
**Handler task** - Processing event.
**Periodic task** - Interrupt generated.

**Periodic task** - About to generate an interrupt.
**Handler task** - Processing event.
**Periodic task** - Interrupt generated.

**Periodic task** - About to generate an interrupt.
**Handler task** - Processing event.
**Periodic task** - Interrupt generated.

**Periodic task** - About to generate an interrupt.
**Handler task** - Processing event.
**Periodic task** - Interrupt generated.

**Periodic task** - About to generate an interrupt.

Step 3.8 Click on Resume icon to continue with the code.

Step 3.9 Observer the LEDs. Which task is in the running state most of the time? Verify your answer with run time statistics. You may have to run the lab until you see statistics output.

Step 3.10 Click Suspend icon to stop the code.

Step 3.11 We will now look at the timing of handler task. At the beginning of the lab, we covered real-time requirements and defined hard and soft real-time response of a system. Comment out the below line in ISR code.

```c
portYIELD_FROM_ISR(pdTRUE);
```

Step 3.12 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 3.13 Click Suspend icon to stop the code.

Step 3.14 Did the debug output change? Why? Can you tell the sequence of events?

Step 3.15 We will now experiment with counting semaphores. Include the line we commented out at Step 3.11 and set mainBINARY_SEMAPHORE to 0 in macro definitions. What is the expected debug output?
Step 3.16 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 3.17 Explain the debug output. Why does Handler task run more than once? Can you summarize the sequence of events?

Step 3.18 Click Suspend icon to stop the code.

4 Lab Exercise 4: Resource Management

Overview
In a multitasking system there is a potential for a conflict to arise if one task starts to access a resource but does not complete its access before being switched out of running state. If the task left the resource in an inconsistent state then access to the same resource by another task or interrupt results in a data corruption or a similar type error. There are also critical sections of the code that must be protected from context switch. In this exercise, we will look at techniques within FreeRTOS to share resources and create critical sections.

Step 4.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 4.2 On the Project Explorer view (far right in Renesas Debug perspective), right click on main3.c and select Resource Configurations>>Exclude from build. Select HardwareDebug box and click on OK. This will exclude main3.c from the project.

Step 4.3 On the Project Explorer view (far right in Renesas Debug perspective), right click on main4.c and select Resource Configurations>>Exclude from build. Unselect HardwareDebug box and click on OK. This will include main4.c into the project.

Step 4.4 On the Project Explorer view (far right in Renesas Debug perspective), double click on main4.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 4.5 There are two print tasks that write to debug output. Print1 task runs at priority 1 and Print2 task runs at priority 2. Each task selects a random number of ticks from 0 to 2 to block itself. What is the expected debug output?

Step 4.6 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 4.7 Click Suspend icon to stop the code.

Step 4.8 Explain the debug output. Do you see corrupted print out. Why? We have been using debug output before and have not seen any corruption. Why? [Hint: Look for where debug print is implemented now and before (basic_io.c file).]
Step 4.9  How can you fix the corruption?  [Hint:  Modify the prvNewPrintString() function to use the mutex.]

Step 4.10  Click Build icon to re-build the project.  On Confirm Image Reload window, click Yes. Project will download automatically.  Click Restart icon to run the code.

Step 4.11  Click Suspend icon to stop the code.
5 Lab Exercise 5: Priority Inversion & Inheritance

Overview
Best way to explain priority inversion and inheritance is to draw a sequence of events that leads up to it. Let's look at an example shown in Figure 6. Low priority (LP) task obtains the resource. At time t1, high priority (HP) task preempts LP task and attempts to use same resource but it can not since it is taken. So, it transitions into blocked state. LP task resumes and at time t3 medium priority (MP) task preempts LP task. MP task does not need the resource and it is a higher priority task so it runs to completion delaying HP task further. MP task completes at t5 and LP task at t6 eventually giving the resource back. Only then, the HP task takes the resource and able to run. This is called priority inversion where a lower priority task delays a higher priority task. Making things worse, another task gets in between low priority task and high priority task delaying the high priority task more.

![Priority Inversion and Inheritance](image)

Priority inversion is a scheme that minimizes the negative effects of priority inversion. Again looking at, Figure 6 when HP task attempts to take the resource at time t1, it blocks because the resource is not available. However, what happens now is that the LP task’s priority is raised to the same priority level of HP task preventing MP task to run. In this case, LP task inherits the priority of HP task. Although MP task is ready to run at time t3, it can not because, LP task is now running at a higher priority. This allows LP task to release the resource sooner and HP tasks to take the same resource and run. FreeRTOS mutexes have built-in priority inheritance mechanism and we will examine it in this lab exercise.
Step 5.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 5.2 On the Project Explorer view (far right in Renesas Debug perspective), right click on main4.c and select Resource Configurations>>Exclude from build. Select HardwareDebug box and click on OK. This will exclude main4.c from the project.

Step 5.3 On the Project Explorer view (far right in Renesas Debug perspective), right click on main5.c and select Resource Configurations>>Exclude from build. Unselect HardwareDebug box and click on OK. This will include main5.c into the project.

Step 5.4 On the Project Explorer view (far right in Renesas Debug perspective), double click on main5.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 5.5 Just as in the overview section, low and high priority tasks share a common resource: debug output. Middle priority task simply blinks LED5. Initially, resource sharing is managed by a semaphore. Low priority task takes the resource and resumes the middle priority task that lasts about 7 seconds. Since now middle priority task is running, low priority task cannot complete and the resource stays with low priority task. High priority task wakes up every 1 second and attempts to take the resource and run.

Step 5.6 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 5.7 Explain the debug output. Does high priority task run every 1 second as intended? How often does it run? Why? Is this a priority inversion? Explain.

Step 5.8 Click Suspend icon to stop the code.

Step 5.9 How can you fix the code so that high priority task runs every 1 second? [Hint: Examine the code. Use mutex instead of a semaphore. Look at the macro definitions.]

Step 5.10 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 5.11 Does debug printout change? Why? Does middle priority task still perform its duties? Does the high priority task run every 1 second?

Step 5.12 Click Suspend icon to stop the code.
6 Lab Exercise 6: Gatekeeper Tasks

Overview
This is our last exercise. One of the best practices when managing shared resources is to create a gatekeeper task. Gatekeeper task has exclusive ownership of the resource and manages it. Other tasks have to go through an established interface and services provided by the gatekeeper to use the resource. Only the gatekeeper task is allowed to access the resource directly.

Step 6.1 On Tera Term window, go to Edit>>Clear buffer to clear the debug console.

Step 6.2 On the Project Explorer view (far right in Renesas Debug perspective), right click on main5.c and select Resource Configurations>>Exclude from build. Select HardwareDebug box and click on OK. This will exclude main5.c from the project.

Step 6.3 On the Project Explorer view (far right in Renesas Debug perspective), right click on main6.c and select Resource Configurations>>Exclude from build. Unselect HardwareDebug box and click on OK. This will include main6.c into the project.

Step 6.4 On the Project Explorer view (far right in Renesas Debug perspective), double click on main6.c file to open it. Examine the code. You may have to unfold some of the lines to view all the code.

Step 6.5 There are four tasks. Two print tasks Print1 and Print2 run at priority level 1 and 2. They use the services of Gatekeeper task to print out their strings. Periodic task runs at priority level 3 and creates an interrupt every 200ms. The ISR uses the services of Gatekeeper task to print out a message from the interrupt routine. The run-time statistics task also uses the gatekeeper task.

Step 6.6 Click Build icon to re-build the project. On Confirm Image Reload window, click Yes. Project will download automatically. Click Restart icon to run the code.

Step 6.7 Explain the debug output. Do you see corrupted print out? Why? What does the interface/service Gatekeeper task provide? How do the other tasks and interrupt routine use the debug output?

THE END

THANK YOU FOR ATTENDING THIS LAB