WIND RIVER

VxWorks® SMP

EVALUATION TUTORIAL

6.6

Edition 2

Corporate Headquarters
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

http://www.windriver.com

For information on how to contact Customer Support, please visit the following URL:

http://www.windriver.com/support

VxWorks SMP
Evaluation Tutorial
6.6

Edition 2
21 Apr 08
Part #: DOC-16255-ND-01

Contents

1 Introduction

Thank you for taking the time to evaluate Wind River VxWorks 6.6 SMP. This evaluation tutorial will introduce you to the process of migrating an application running in UP mode to run in SMP mode, taking advantage of the multiple cores to gain an increase in performance. You will also learn during this evaluation how Wind River's Workbench can shorten the process of migrating an application from UP to SMP by:

- Helping you analyze your UP application execution and identify the parts that should be parallelized.
- Allowing you to see how your application executes on the target, and how to identify run-time problems that arise when migrating an application from UP to SMP mode.
- Simplifying the process of debugging multiple cores / CPU's system via the multi context debugging capability.

If you have received an actual DVD containing the evaluation, you may skip chapter 2 and proceed straight to chapter 3. Otherwise, if you have downloaded a DVD image file, you will need to review chapter 2 for instructions on how to use the ISO image file.

Note that this document was written for a U.S.-English keyboard. Please adjust key references to the keyboard for your region.

2

Booting the VxWorks 6.6 SMP Evaluation

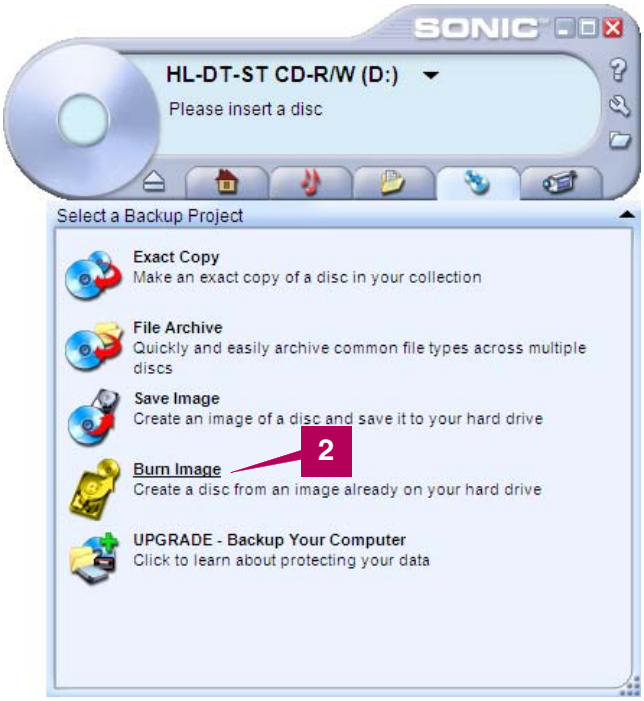
2.1 Preparations Before You Can Use the Image

The Wind River VxWorks SMP Evaluation comes in the form of an ISO image file. The ISO image contains a Fedora 7 Live CD Linux host distribution, with an installation of Workbench 3.0 and Wind River VxWorks 6.6 SMP for IA32. To use the evaluation, burn the ISO to a single layer DVD, and boot it on any PC.

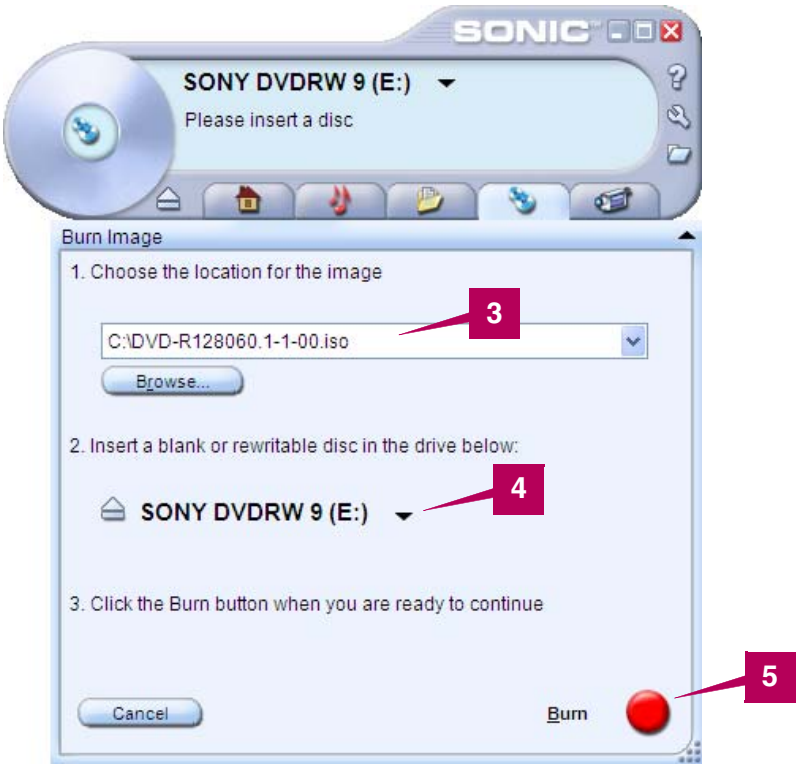
2.2 Burning the ISO Image to a DVD

To burn the ISO to a single layer DVD you can use any DVD burning program that supports burning ISO images. There are many good software packages on the market, some free and some commercial. It would be impossible to cover them all. The instructions in this part will describe how to use the *Sonic RecordNow! Plus* software, but the ideas are the same when using other DVD burning software.

1. Start the DVD burning software.
2. Select the option to **Burn Image**.



3. Choose the location for the image, the **DVD-R138215.1-1-00.iso** image that you downloaded.
4. Insert a blank DVD into your DVD burner, and select the appropriate drive.
5. Click the **Burn** button when you are ready to continue.



6. Once you have successfully burned the DVD, continue to the next chapter.

2.3 Using the DVD

The evaluation runs in a "Live-CD/DVD" environment. Your laptop/PC boots Linux, Fedora Core 7 from a DVD. Minimum machine requirement is a Pentium4 machine + 1GB RAM and BIOS with the ability to boot from DVD drive.

1. Insert the Wind River VxWorks SMP Evaluation DVD.
2. Power up your laptop/PC and set it to boot from CD/DVD/CD-RW: if your BIOS provides a boot menu option, select that key (**F12** for Dell laptops) and choose to boot up from CD/DVD/CD-RW. Or press the BIOS setup key to enter into the BIOS setup mode and select the option to boot from CD/DVD/CD-RW device first.

3. Boot the machine. Wait for the boot menu to appear, and then select the option **Boot Wind River VxWorks 6.6 SMP** to continue, or wait until the timeout expires.

4. Wait till Fedora Core 7 finishes booting and the GNOME desktop appears on the screen.

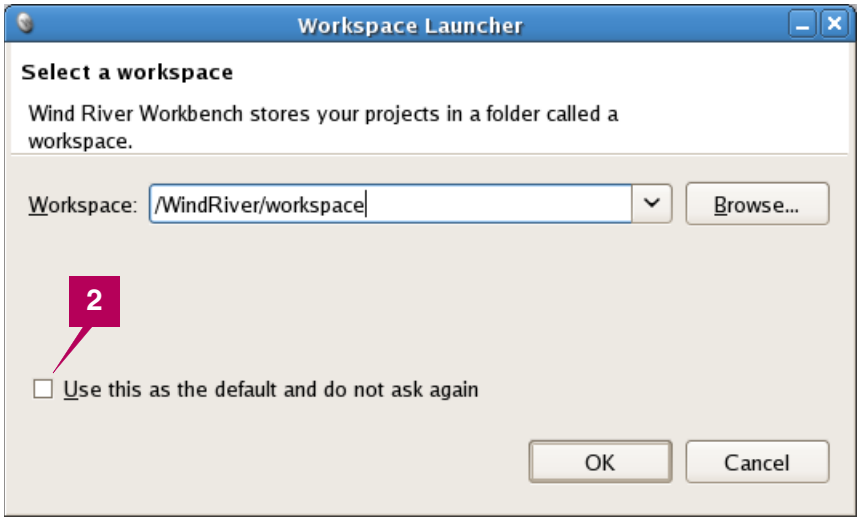
3 Workbench Familiarization

This chapter will familiarize you with the basic capabilities of the Workbench Integrated Development Environment (IDE).

3.1 Starting Wind River Workbench

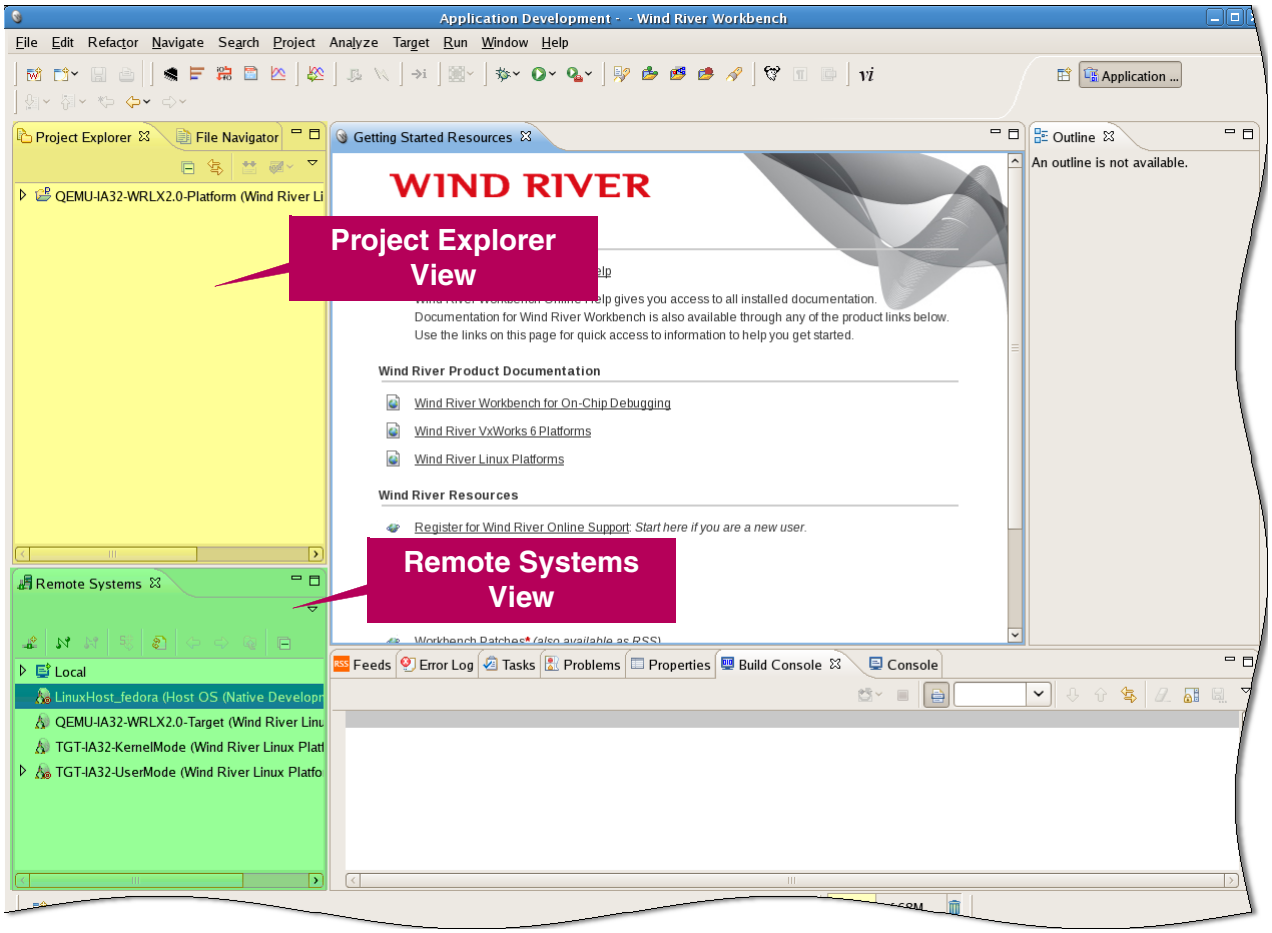
1. On the desktop, click on the Wind River Workbench icon 

. This starts the Wind River Workbench tool.
2. When the Workspace Launcher dialog box appears, you can select **Use this as the default** and do not ask again to avoid this dialog in the future.
3. Click **OK** to continue.



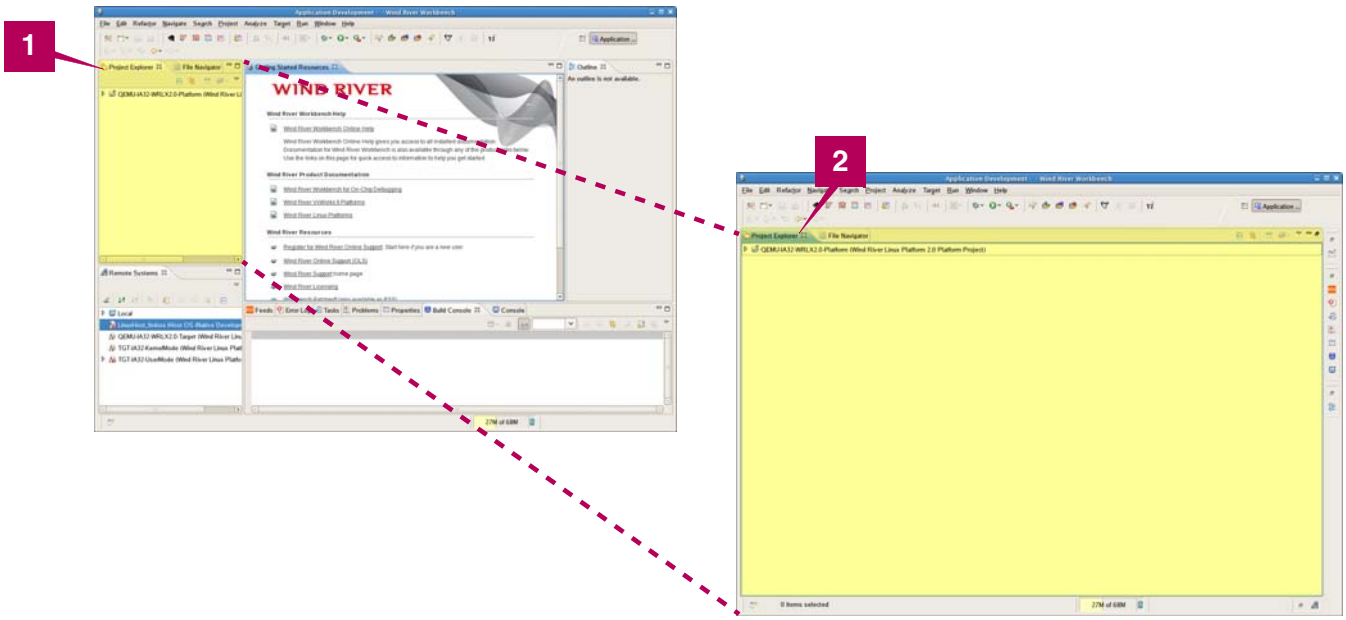
3.2 Definition of a View in Eclipse

- Each of the sub-windows in Workbench is called a *view* in Eclipse terminology.
- Make sure you can find the Project Explorer and Remote Systems views as you'll need to revisit them periodically in this evaluation.

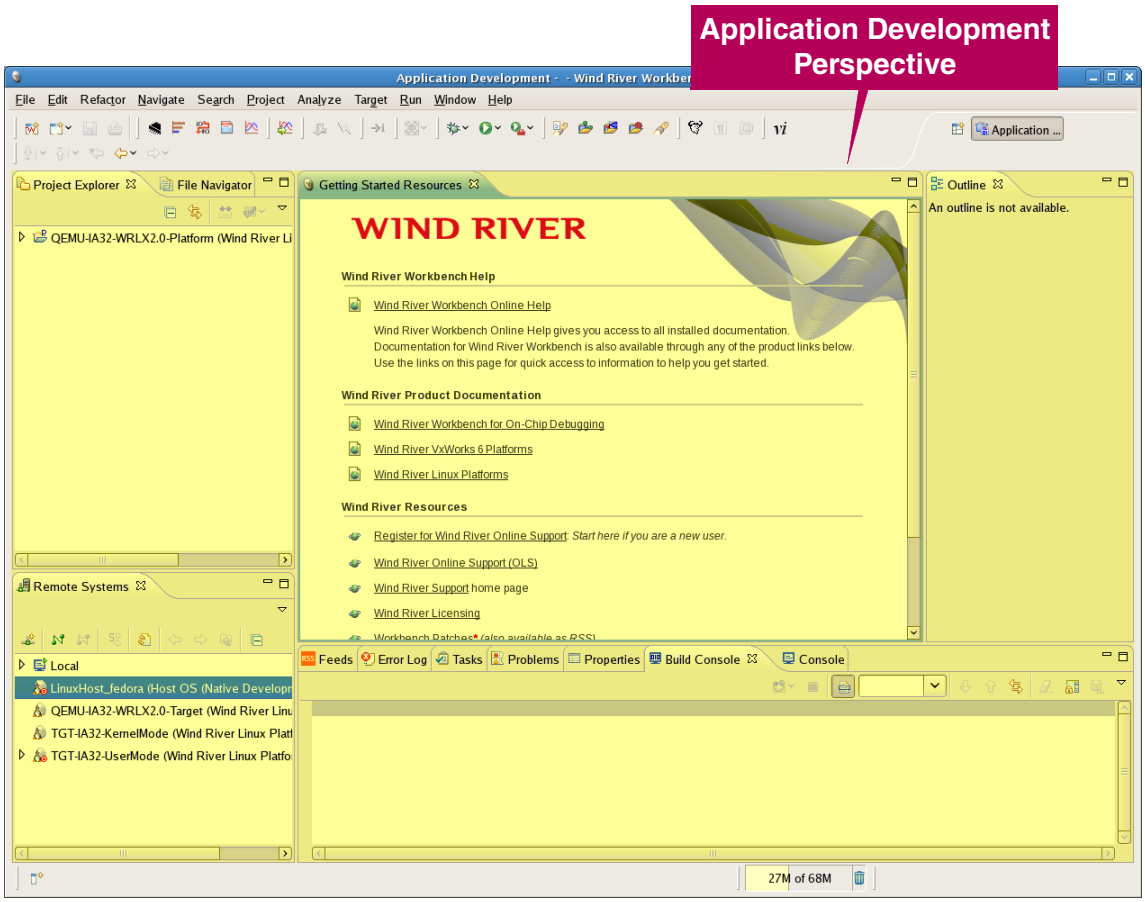


3.2.1 **Resizing a View**

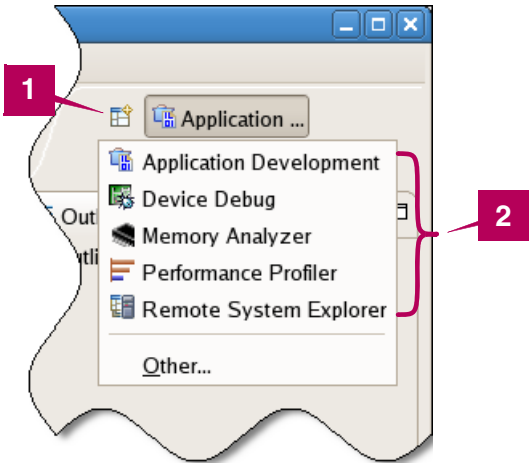
1. Double-click on the view tab. This gives a full screen view of the window.
2. Double-click on it again to switch the view back to its previous size.
3. Right-click on the tab to look at other options for windowing.



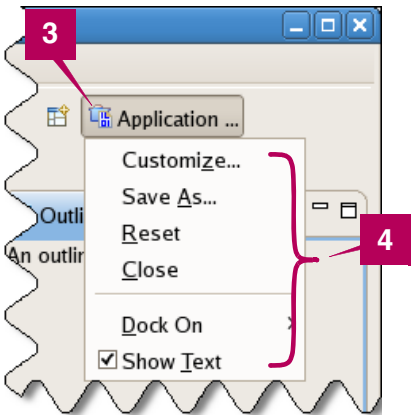
3.2.2 Working with Perspectives



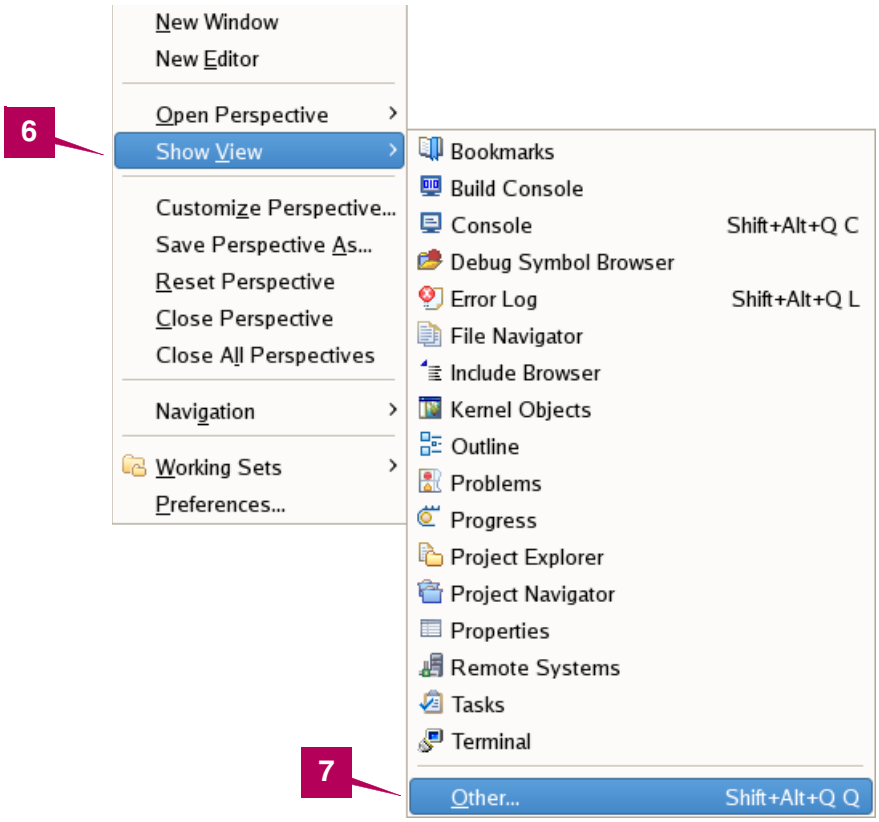
1. Click on the **Open Perspective** icon, and look at the options.
2. Click through different perspectives to see how windows are arranged.



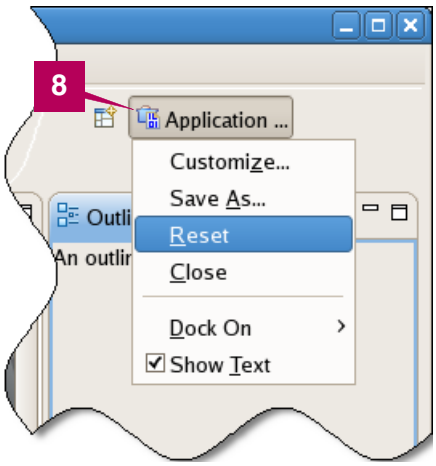
3. Right-click on the perspective tab.
4. Look at the context menu options.



5. Try resizing views (drag dividers). Move, drag, and resize windows in the Device Debug perspective.
6. Add/Delete Windows. Look under the **Window** menu, find **Show View**.
7. Select **Other** – try expanding some of the folders to see what views are available in Workbench. Select some windows. If you click on '**x**' the window is removed and you can get it back from this menu.

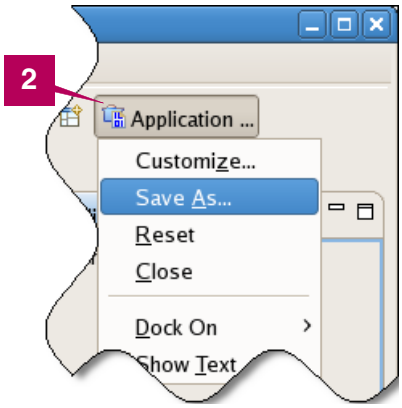


8. To restore and clean up the changes you made, right-click on the perspective tab.
9. Select **Reset** from the context menu to get back the default window arrangement.

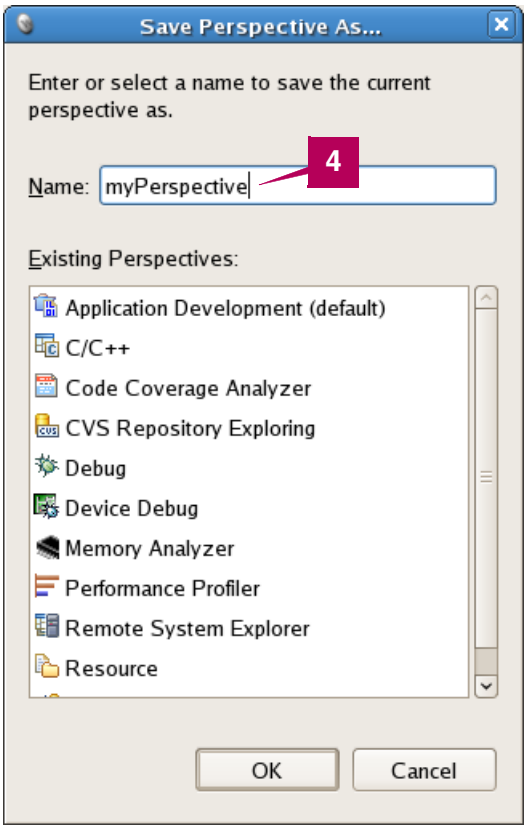


3.2.3 Creating a New Perspective

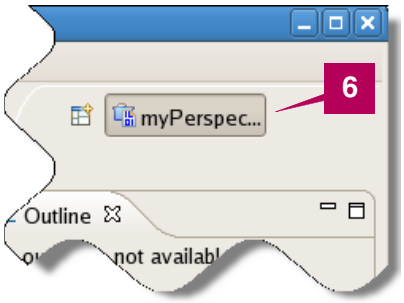
1. Add some views of your choice using **Window > Show View**.
2. Right-click on the **Application** perspective icon.
3. Select the **Save As** option from the context menu.



4. Call this perspective **myPerspective**.
5. Click **OK**.

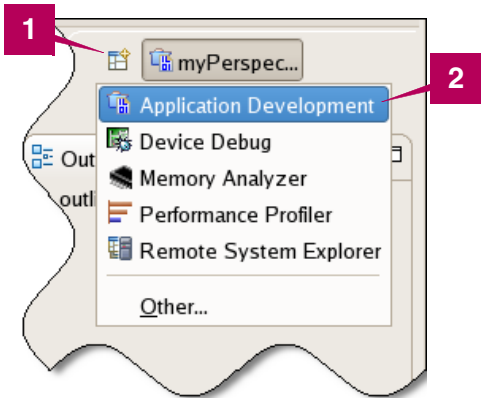


6. Observe the new perspective tab.



3.2.4 Switch Back to the Application Perspective

1. Click on the **Open Perspective** icon.
2. Select the **Application Development** option from the context menu.



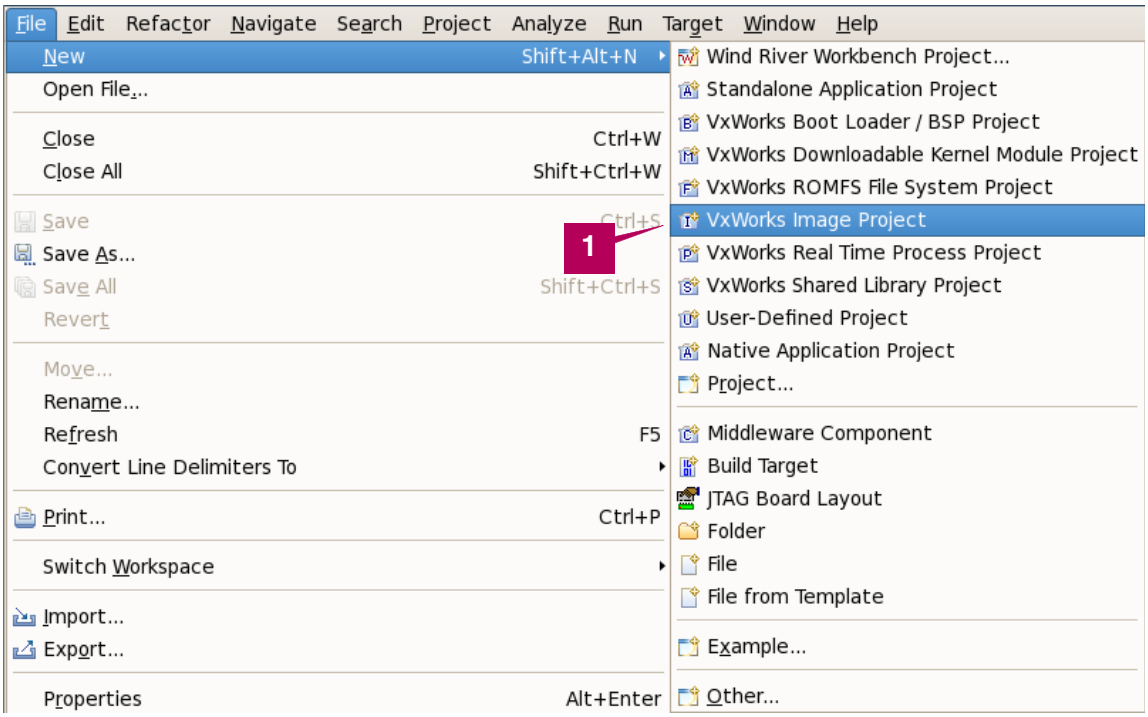
4 Working with the VxWorks Simulator

This chapter will familiarize you with the use of the Wind River VxWorks Simulator to develop an SMP application. It will help you understand how to create and use a VxWorks image with SMP support. The image will be used on a simulator to practice the steps of migration of UP to SMP application.

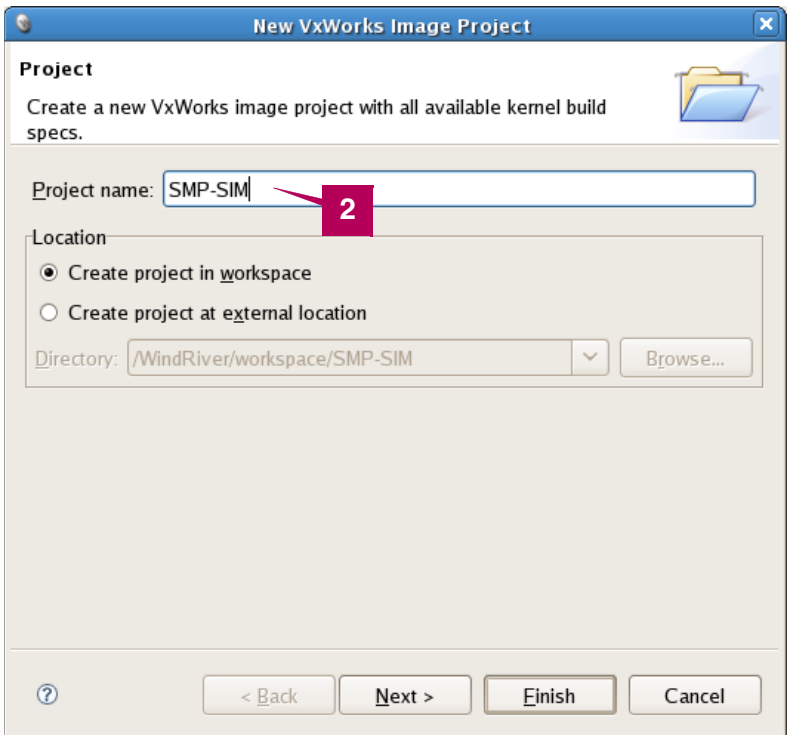
4.1 Creating an Image with SMP Support for the VxWorks Simulator

These steps will guide you through the processes required to build a functional VxWorks simulator that can simulate up to eight cores. The simulator can be used during the development phase or when hardware is too costly or not available. All Wind River tools are fully functional on the VxWorks simulator as if you were connected to a real target board.

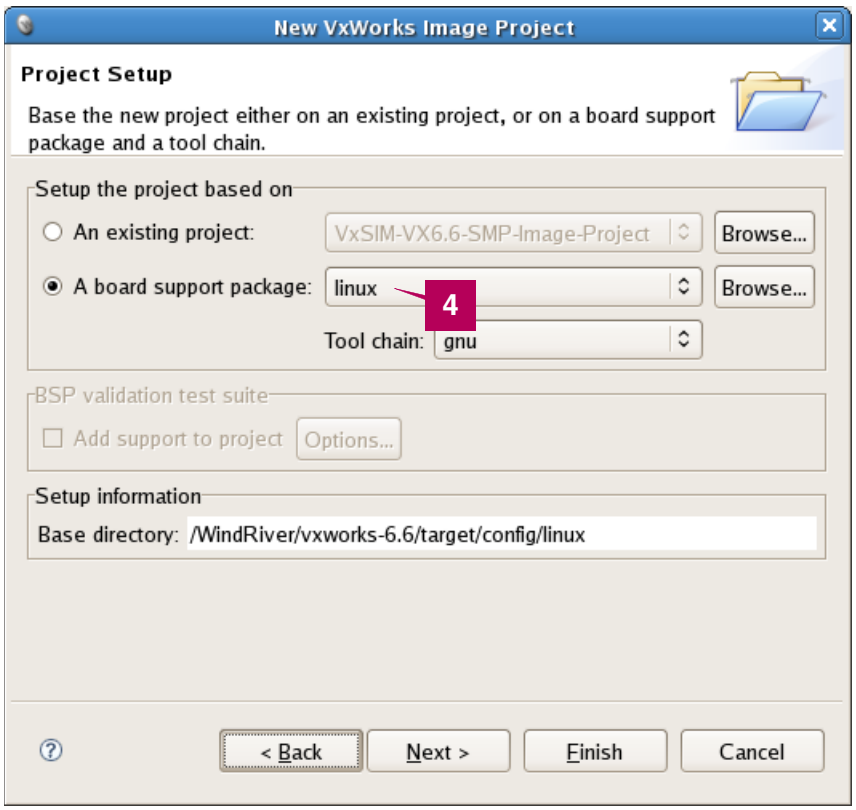
1. From Workbench menu bar, click on **File > New > VxWorks Image Project**.



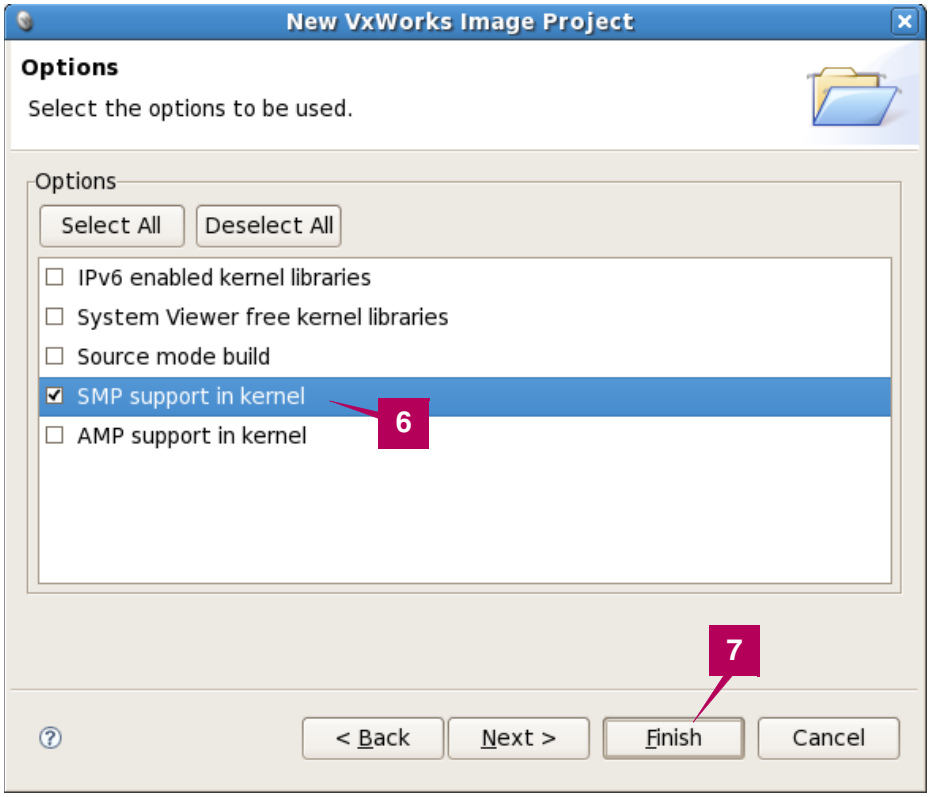
2. In the New VxWorks Image Project dialog box, enter **SMP-SIM** as the name of the project.
3. Press **Next** to continue.



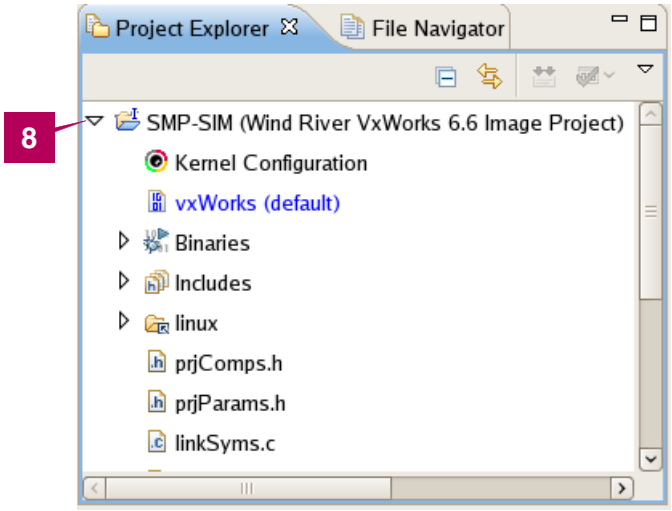
4. In the New VxWorks Image Project – Project Setup dialog, leave all parameters in their default state. The simulator runs on Linux, so we use **linux** as a base for the project.
5. Click **Next** to continue.



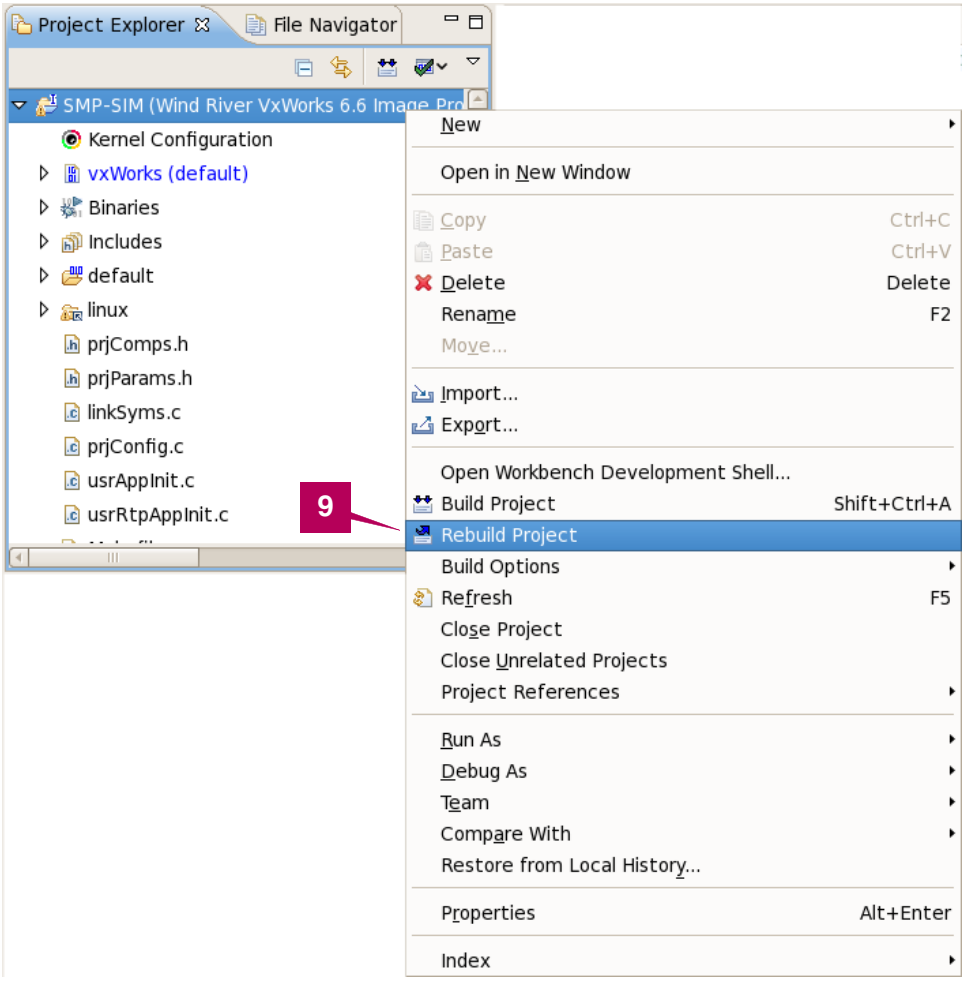
6. In the New VxWorks Image Project - Options dialog, select the **SMP support in kernel** option. This will include all the necessary modules required for SMP VxWorks kernel.
7. These are all the options that need to be configured, so we can now click on **Finish** to proceed.



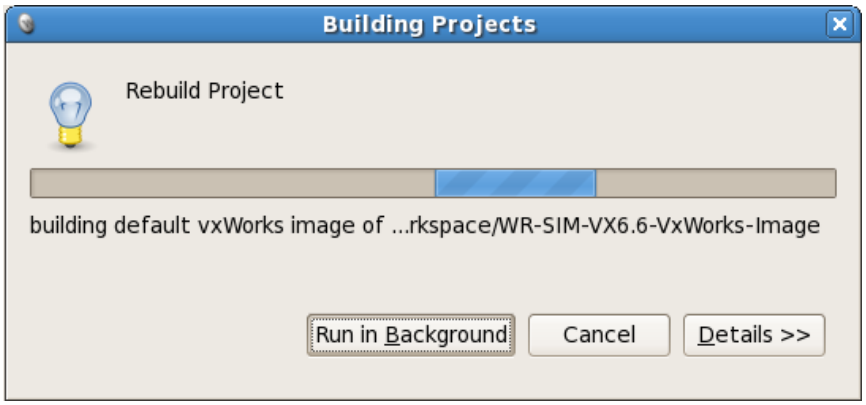
8. The new project is displayed in the Project Explorer view. You can click on the triangle ▷ symbol on the left to expand and collapse the project's build and configuration files.



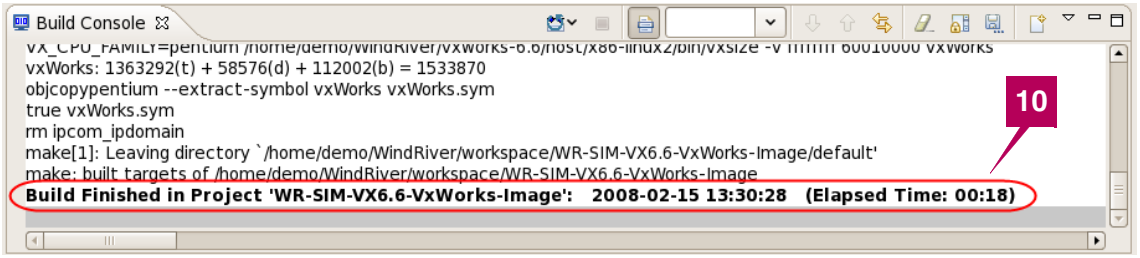
9. To build the new VxWorks image, right-click on the **SMP-SIM** project in the Project Explorer view, and select **Rebuild Project** from the context menu.



10. A project build progress window will appear.



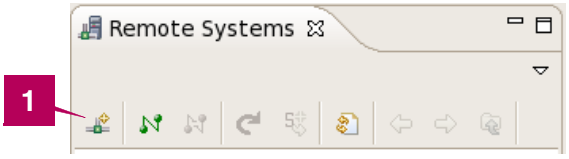
11. Finally, a notification appears in the build console window indicating that the build process has completed.



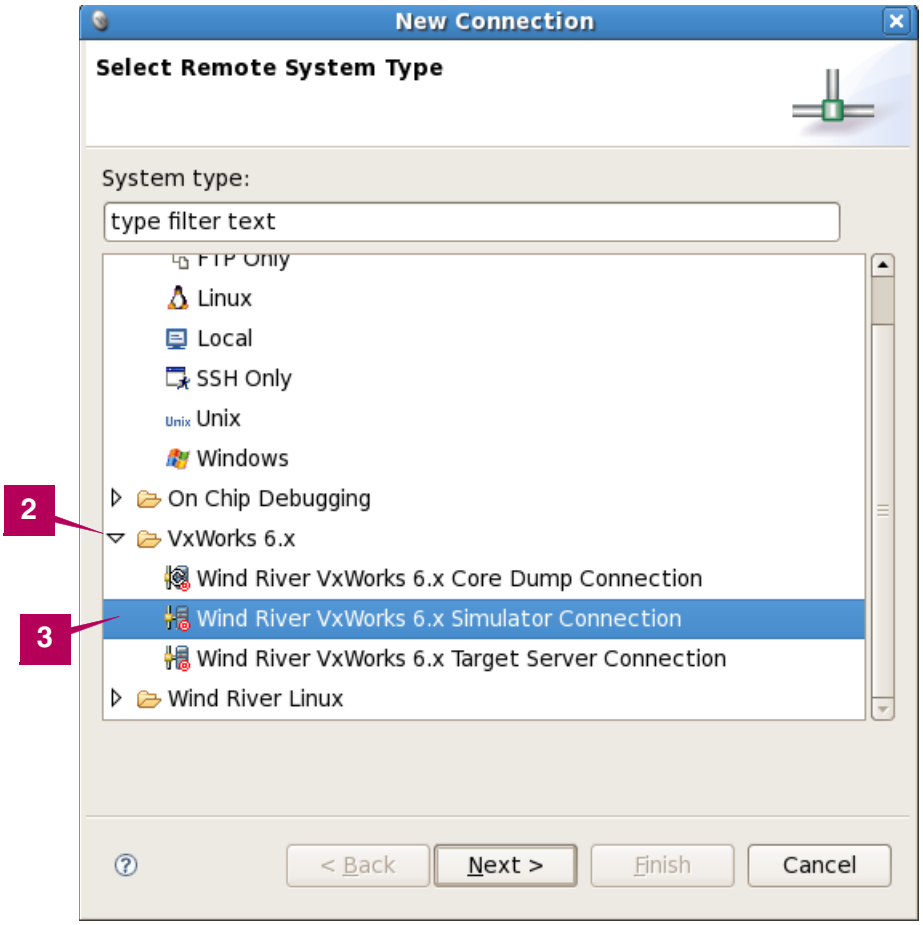
4.2 Using the VxWorks Simulator

In the following steps you will create a new VxWorks simulator and connection, and link the simulator's connection to the VxWorks image you have created in the previous section.

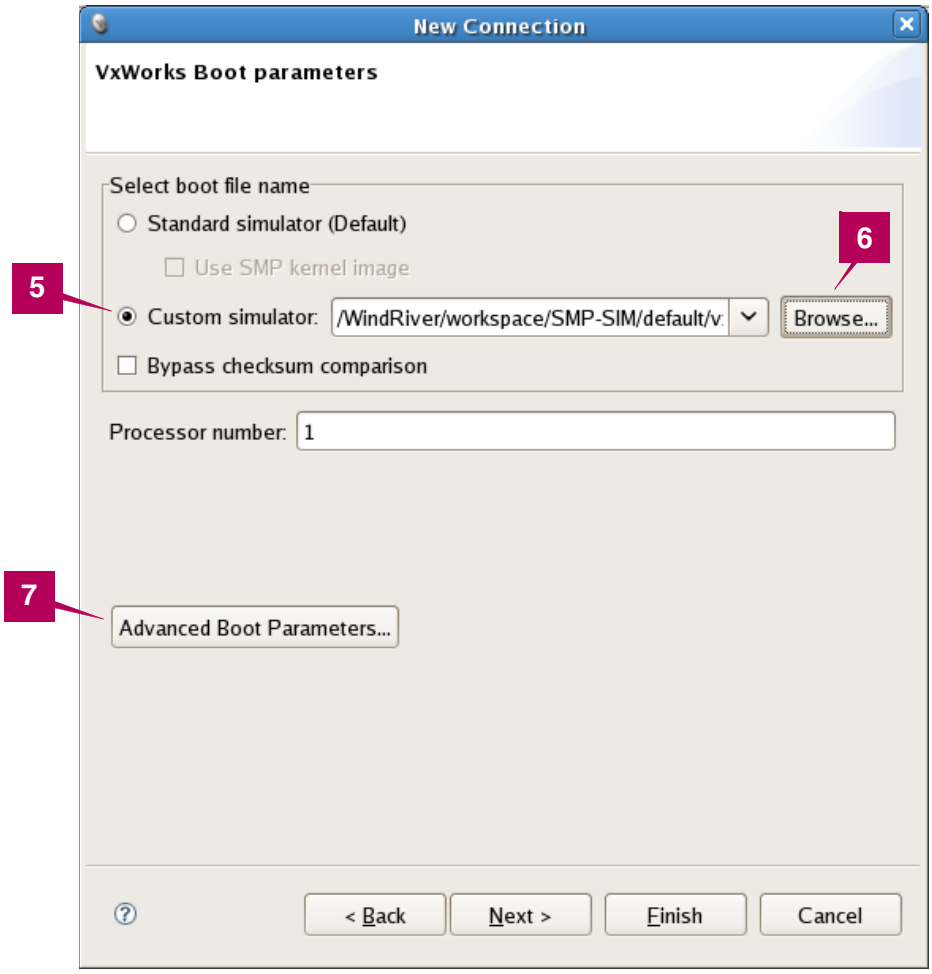
1. In the Remote Systems view, select **Define a connection to remote system**



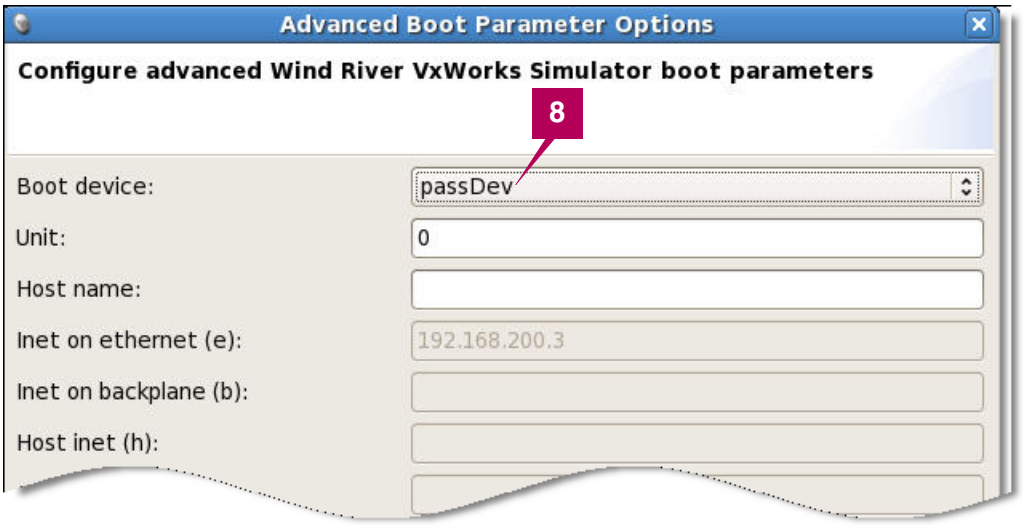
2. In the New Connection dialog, expand **VxWorks 6.x**.
3. Select **Wind River VxWorks 6.x Simulator Connection**
4. Select **Next**.



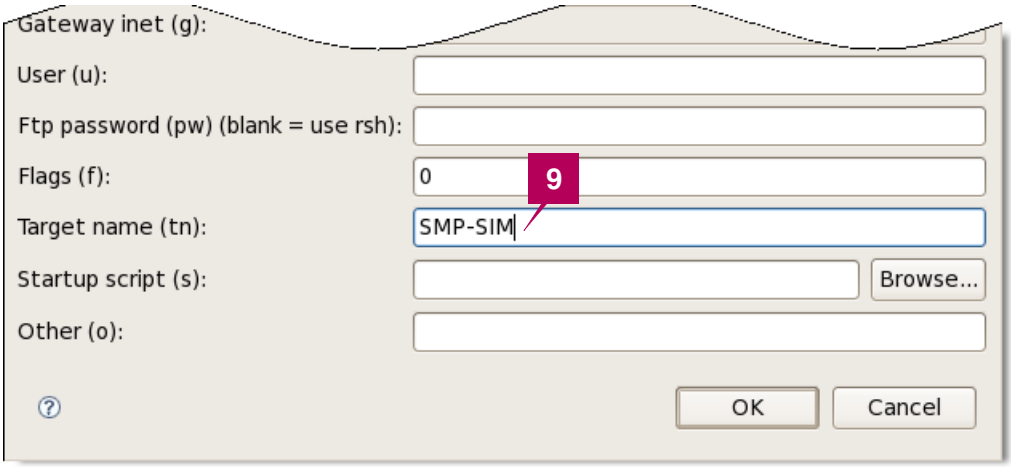
5. In the New Connection - VxWorks Boot Parameters dialog, select **Custom simulator**.
6. Click on **Browse** to select the following file: **/WindRiver/workspace/SMP-SIM/default/vxWorks**
7. After a VxWorks image file has been selected, click on **Advanced Boot Parameters**.



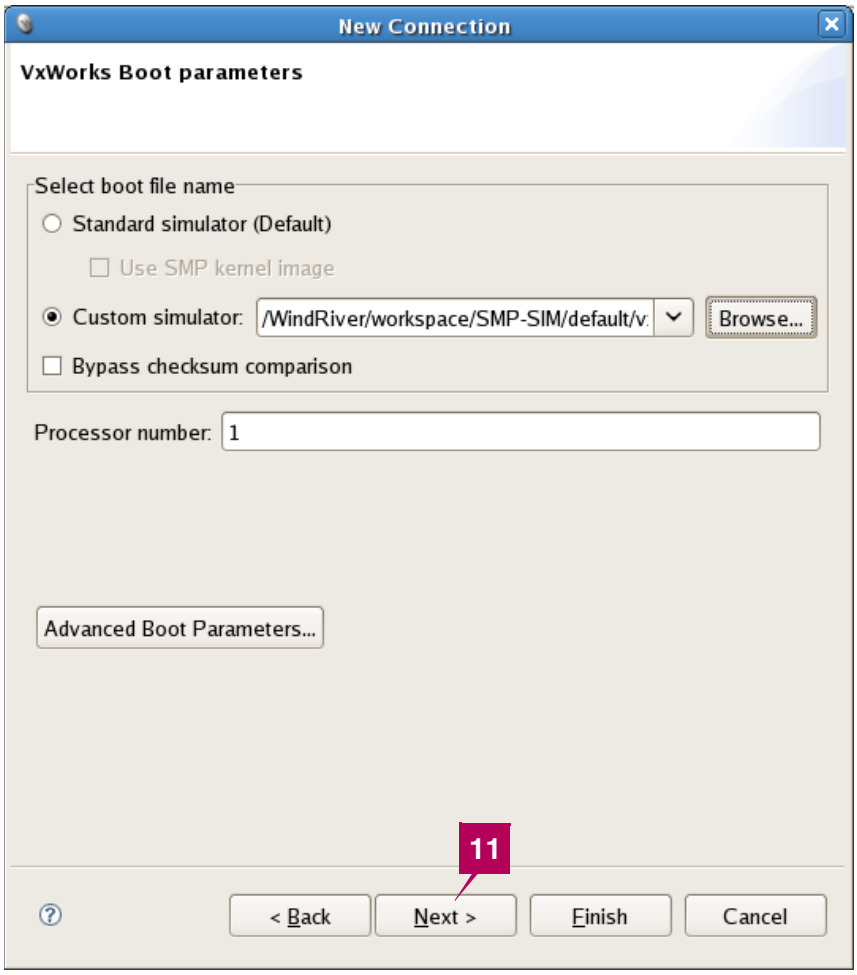
8. In The Advanced Boot Parameter Options window, click on the **Boot device** parameter and change the value from **passDev** to **simnet**. This is the method that Workbench uses to attach to the target (simulator).



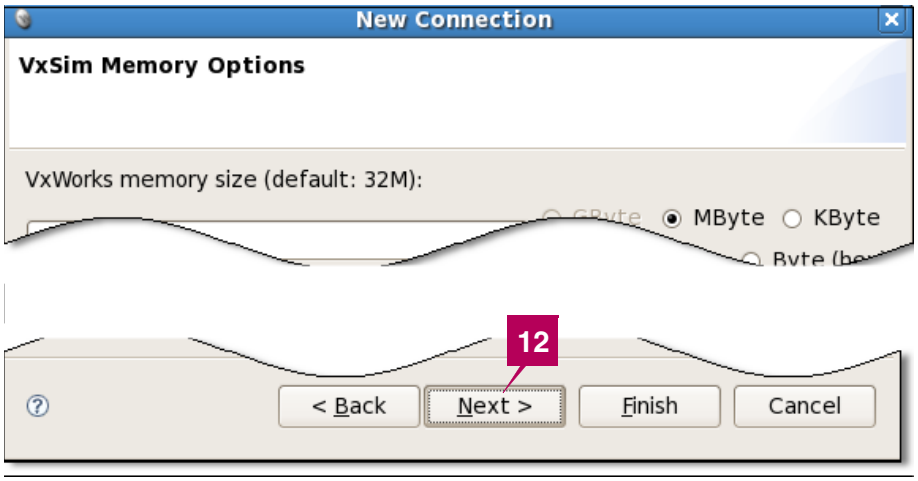
9. Set the **Target Name** setting to **SMP-SIM**.
10. Select **OK**.



11. Back in the New Connection dialog, click **Next**.

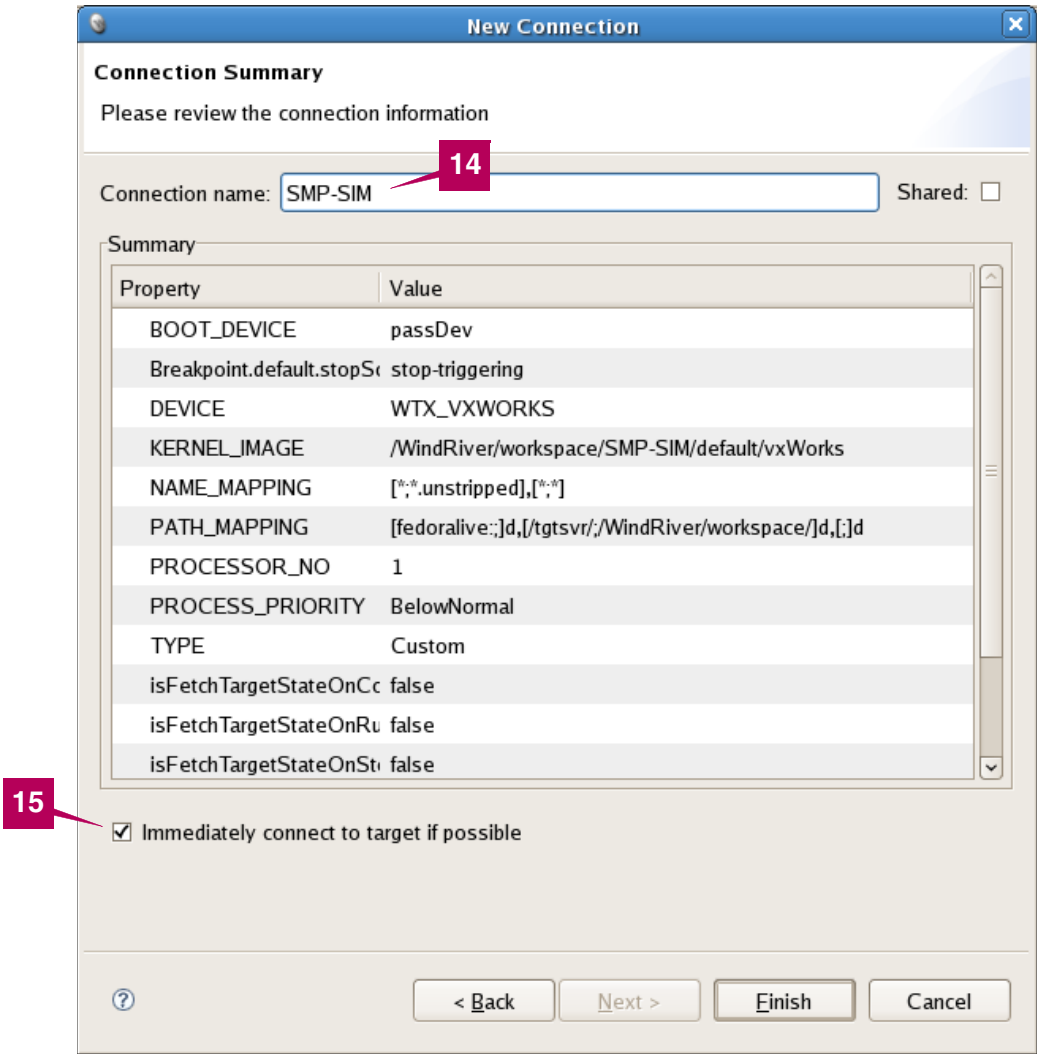


12. In the New Connection - VxSim Memory Options dialog, select **Next**.



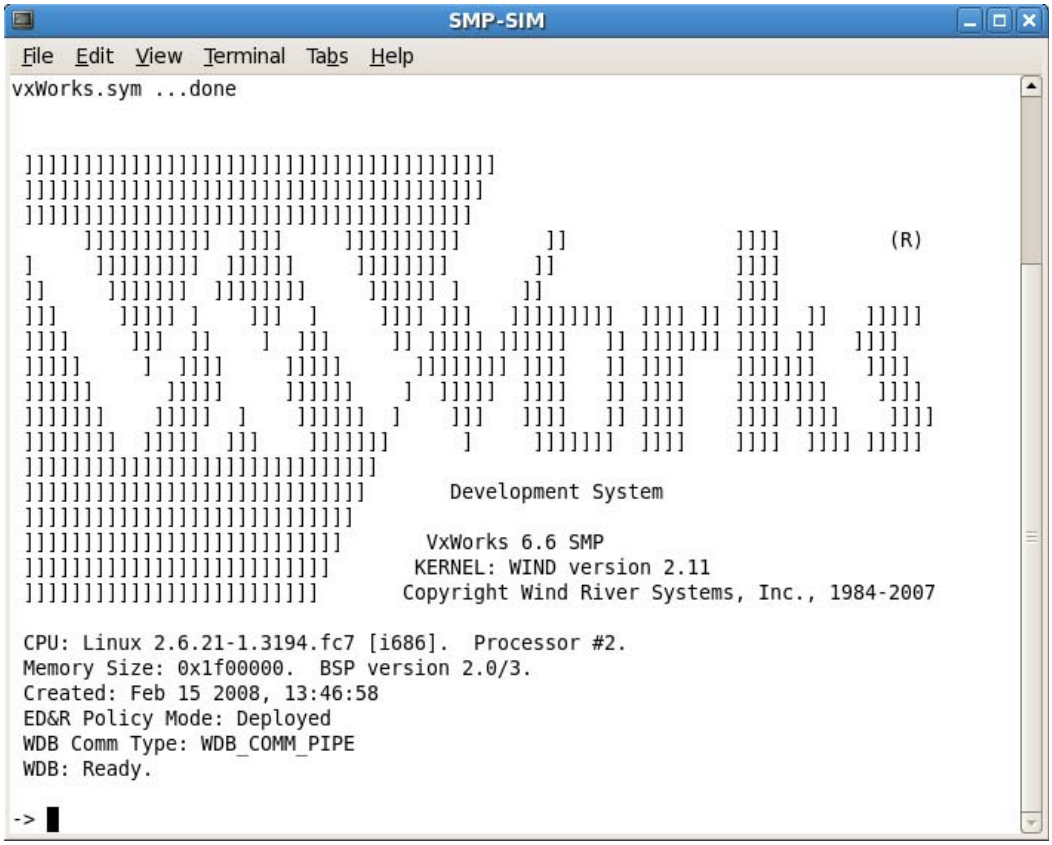
13. Continue clicking **Next** to advance through the wizard without modifying options until you reach the **New Connection – Connection Summary** dialog.

14. In the New Connection – Connection Summary dialog, enter **SMP-SIM** for **Connection name**. This will help you to identify your simulator session amongst other available targets.
15. Leave **Immediately connect to target if possible** checked. This will cause the target to automatically launch after we finish.
16. Click **Finish** to complete the simulator configuration.



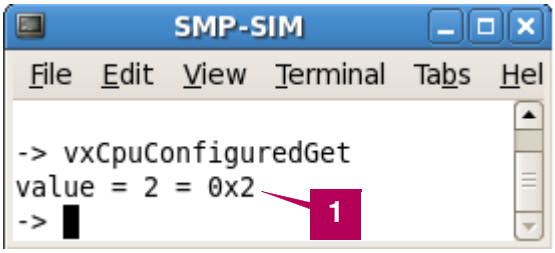
17. Your new connection should appear in the Remote Systems view.

18. A new window will pop up with a VxWorks welcome message. This is your simulator session that runs the VxWorks SMP Image. You can download kernel downloadable modules (DKM) and Real-Time Applications (RTP) as if you were working on a target board.

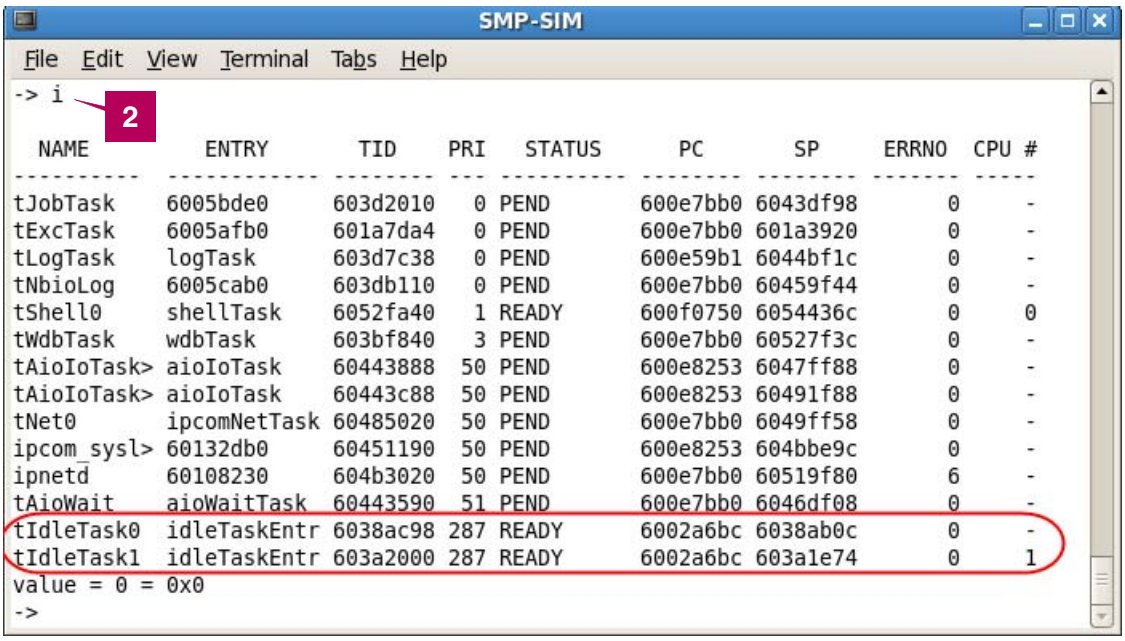


4.3 Verifying Simulated Code

1. The simulator by default is configured to simulate two cores. In the simulator, enter the command **vxCpuConfiguredGet** to see how many cores are configured.



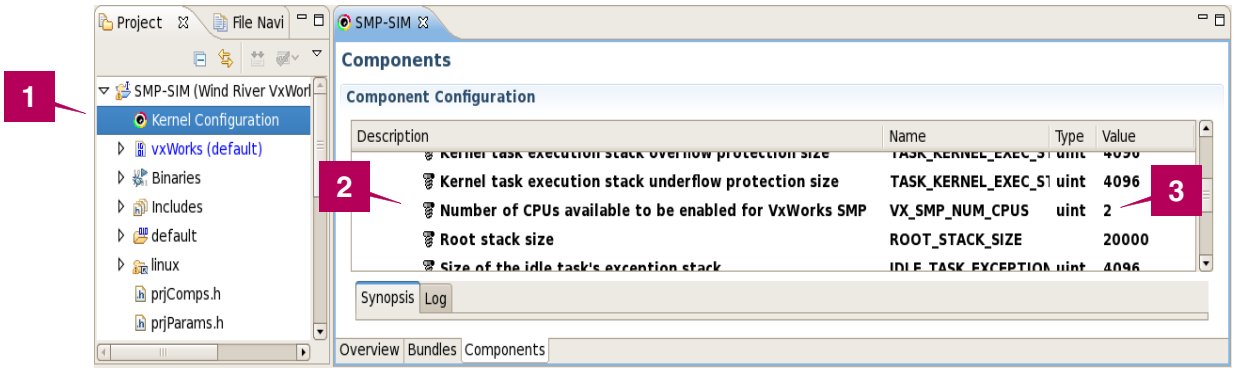
2. The **i** command lists all tasks in the kernel. Enter **i** at the prompt and observe that there are two idle tasks, since each core in an SMP system has its own idle task



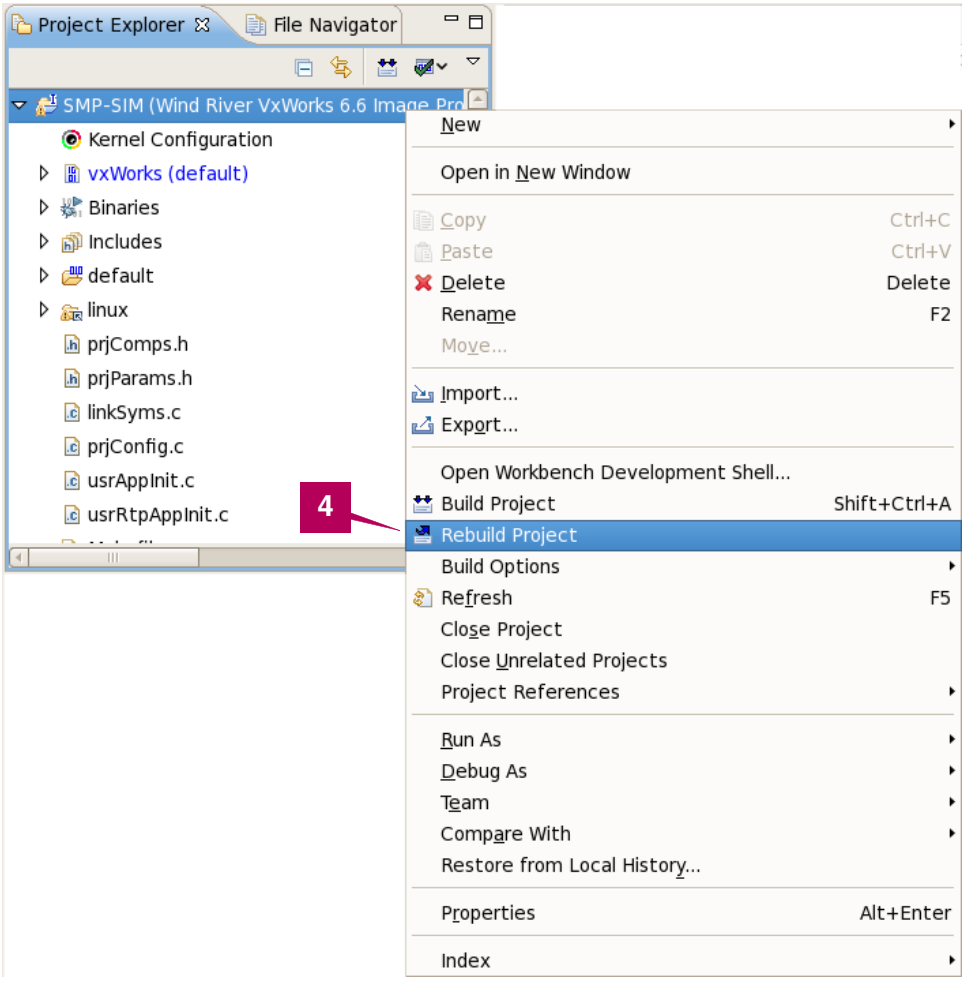
4.4 Adding More Cores to the Simulator

The VxWorks simulator enables you to simulate up to 32 cores, following these steps to change the number of simulated cores:

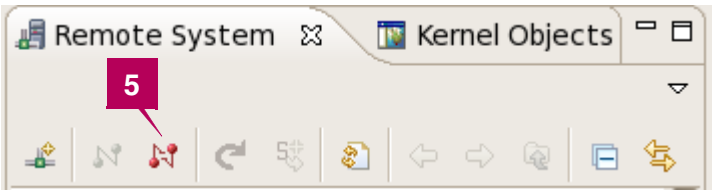
1. Double-click the **Kernel Configuration** item under the **SMP-SIM** project in the Project Explorer view.
2. In the Components view, browse to **operating system components => kernel components => kernel**. Expand this item by clicking the triangle next to it.
3. Change the value of **Number of CPUs to be enabled for VxWorks SMP** from **2** to **4**.



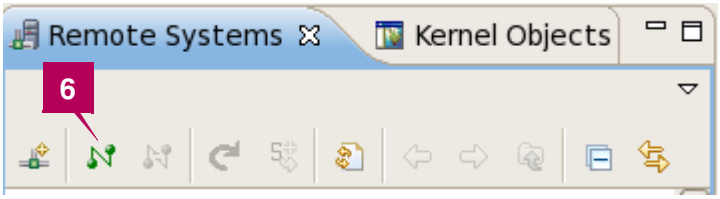
4. To apply the kernel configuration, the VxWorks image must be rebuilt. Right-click on the **SMP-SIM** project in the Project Explorer view and select **Rebuild Project** from the context menu.



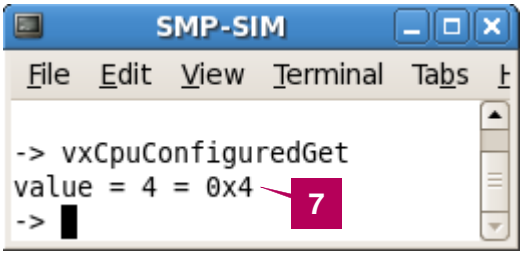
5. Disconnect from the simulator if the simulator is running by left-clicking on the **SMP-SIM** entry in the Remote System view to select it, followed by clicking on the **Disconnect** button from the Remote Systems view.



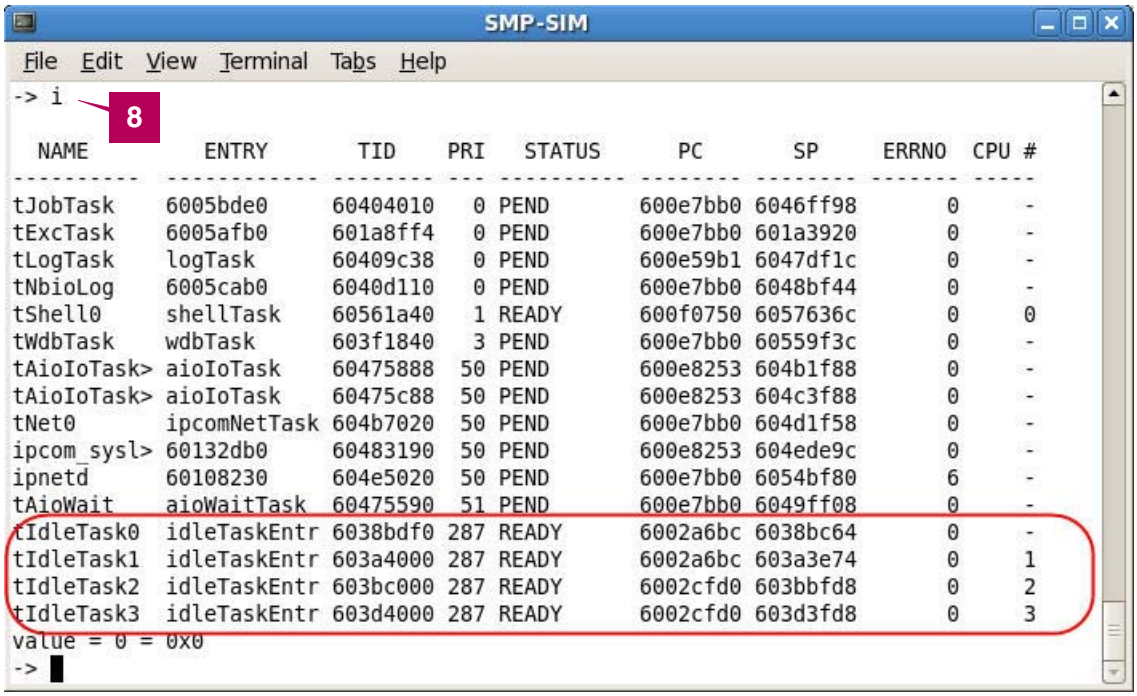
6. Connect to the simulator again by clicking on the connect button. This will launch a VxWorks simulator with the newly updated configuration.



7. To verify your changes, once again enter the **vxCpuConfiguredGet** command in the simulator session:



8. Again enter the **i** command, and you can see that four idle tasks are running, one for each configured core.



9. Finally disconnect from the simulator, if the simulator is running, by left-clicking on the **SMP-SIM** entry in the Remote System view to select it, followed by clicking on the **Disconnect** button from the Remote Systems view.

4.5 VxWorks Simulator Summary

In this section, you built a VxWorks SMP image for simulator. The default number of cores is two, but this can easily be changed using the kernel configuration tool. VxWorks simulates the specified number of cores, and as such can be used as a sandbox for your research phase and project development. Wind River tools such as the System Viewer (task and event monitoring), the Performance Profiler (performance monitoring), and the SMP debugger are all supported and can be used to analyze your application on the simulator.

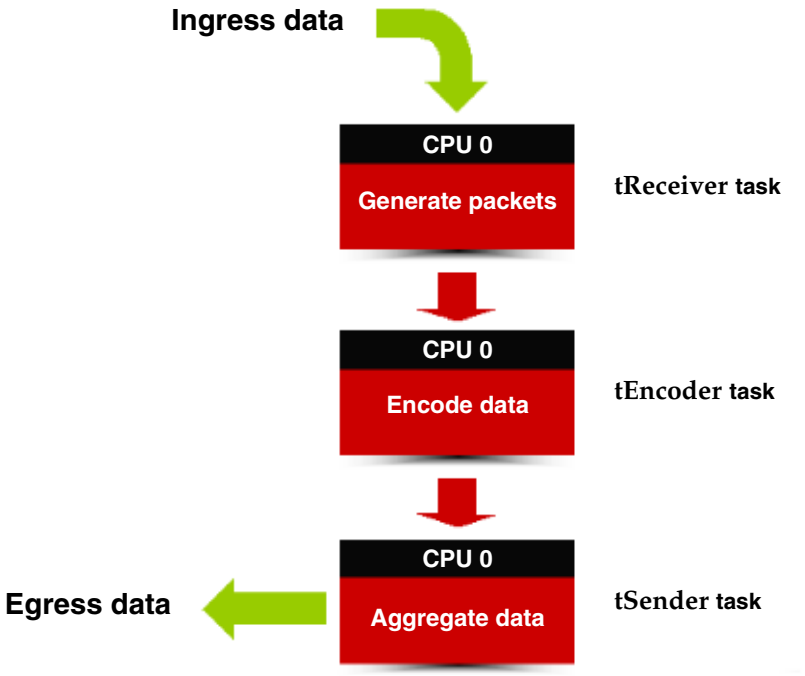
NOTE: Please note that the simulator should not be used for performance benchmarking, because simulator performance is dependent on the host hardware profile (CPU type and speed, available RAM, and so on) and available CPU cycles. The simulator is a great tool to plan and design your migration phase from UP to SMP, but do not pay much attention to performance figures obtained from this simulated environment.

5 VxWorks UP to SMP Application Migration

This chapter will help you understand the process of migrating a UP application to an SMP application. The exercises will show you how to migrate a simple network device application from UP to SMP.

5.1 Application Overview

In this evaluation, the UP application you will migrate to SMP is a small version of a network decoder device. It receives and buffers incoming data, encodes the data, and finally sends the encoded data out. The following figure depicts the application building blocks.



The receiver task buffers the incoming data, divides it into equal packets or data elements, tags each packet, and finally builds a tree from the data structure.

The encoder task retrieves packets from the tree and encodes the data. Finally the sender task removes all internal tags.

5.2 UP-to-SMP Migration Overview

The following steps are recommended to migrate your application from UP to SMP:

Step 1: Analyze Your UP Application

Analyze and identify the area that will benefit from SMP; this means identify the area in your application that should run in parallel when running on an SMP-enabled system.

Step 2: Redesign Your Application for SMP

Redesign your application to benefit from SMP by multi-threading the area that should be executing in parallel when running on an SMP-enabled system.

Step 3: Test Redesigned (Multi-Threaded) Application in UP Mode

Ensure that the area that was redesigned to run in parallel runs properly in a multi-threaded environment.

Step 4: Test Your Redesigned (Multi-Threaded) Application in SMP Mode

Confirm that your application runs on an SMP-enabled system.

The first three migration steps will be done while using only one core (on a single-core image). This will ensure that the SMP design works on UP (that is, on one core).

5.3 Analyzing Your UP Application

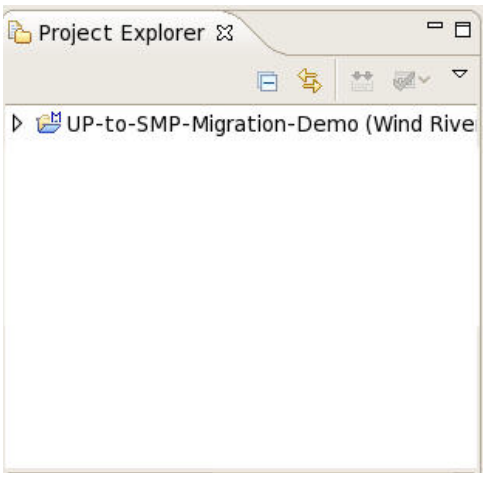
The first step to redesigning a UP application for SMP is to determine how SMP can benefit the application. In order to do this, you need to identify which tasks consume a lot of CPU cycles, and how those tasks can be broken into smaller pieces that can be run simultaneously.

To perform this step you will use the Wind River Performance Profiler. It is an excellent tool for assisting in this investigation.

5.3.1 Loading and Starting the Application

Before you can use the Performance Profiler you will need to load the application to the target (simulator in this case), and start it. The following steps describe how to perform this task.

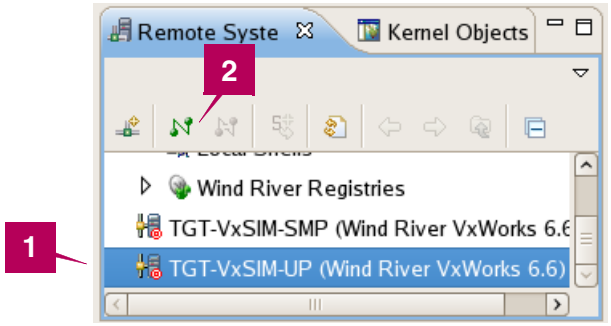
The UP application you will analyze is included as a downloadable kernel module (DKM) named **UP-To-SMP-Migration**. You can find it in the Project Explorer view.



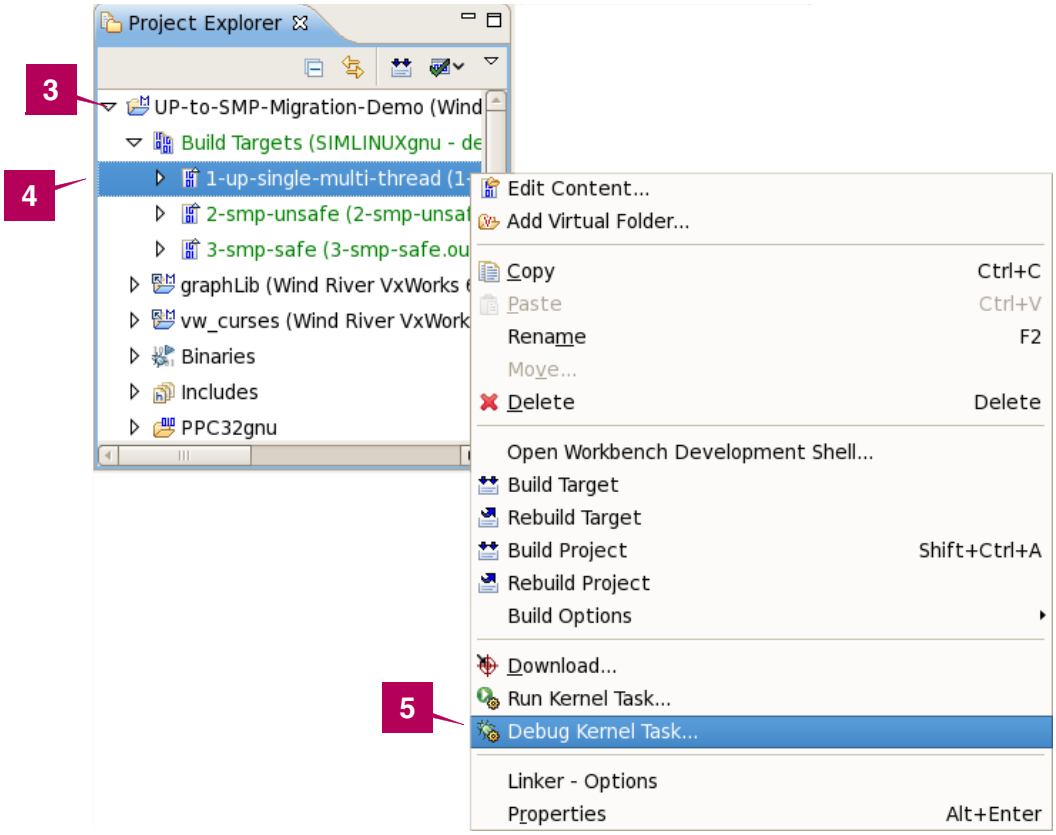
NOTE: Make sure the previous instance of the UP VxWorks simulator is closed!

To close it simply left-click on the **TGT-VxSim-UP** entry in the Remote Systems view to select it, and then click the **Disconnect** button to close it.

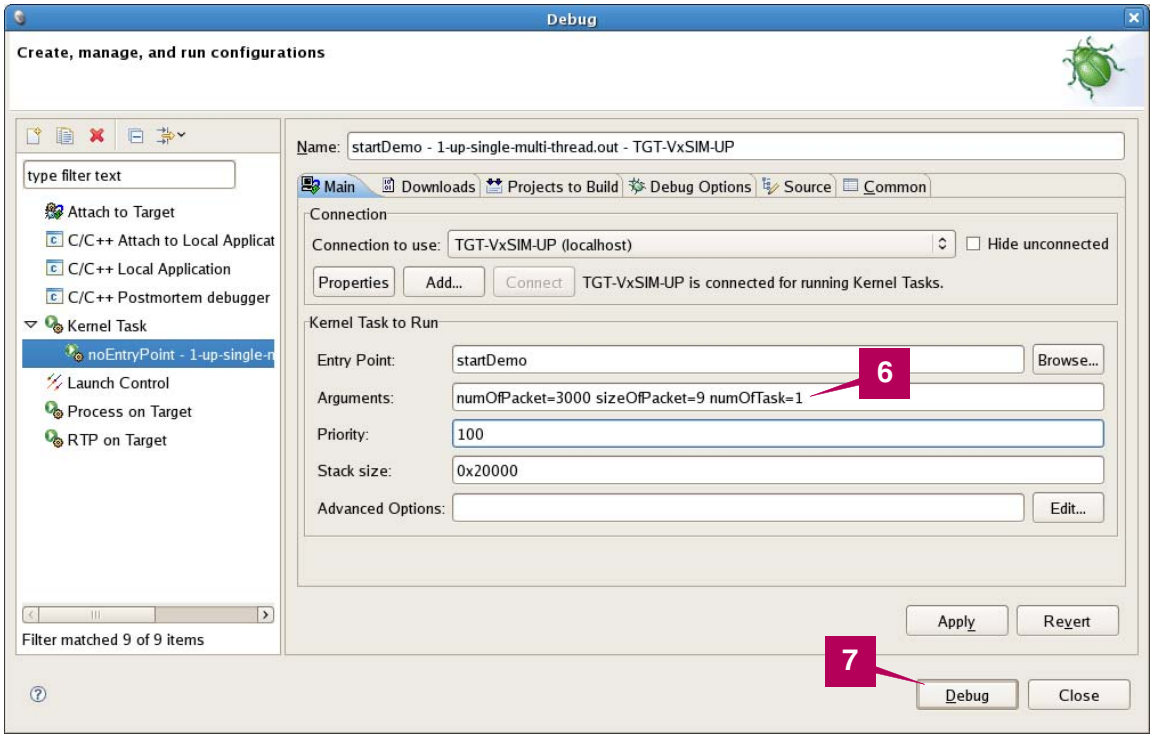
1. Start the UP simulator. Select **TGT-VxSim-UP** in the Remote Systems view.
2. Click on **Connect** to launch the simulator

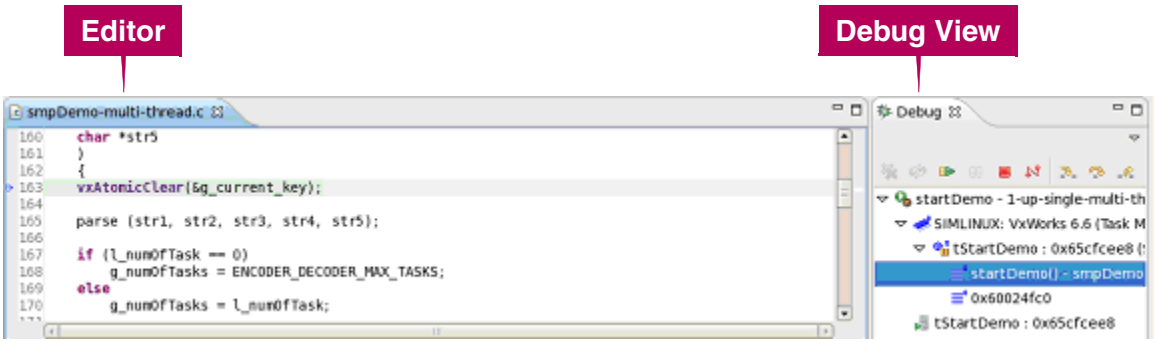


3. In the Project Explorer view, expand the **UP-to-SMP-Migration-Demo** project and the **Build Targets** item within.
4. Right-click on the first project, **1-up-single-multi-thread**.
5. From the context menu, select **Debug Kernel Task**.



6. In the Debug dialog, verify that **Arguments** is set to **numOfPacket=3000 sizeOfPacket=9 numOfTask=1**.
7. Select **Debug**. This will launch the application, and you should see the source file in Workbench editor, and that the application is stopped in the Debug view.





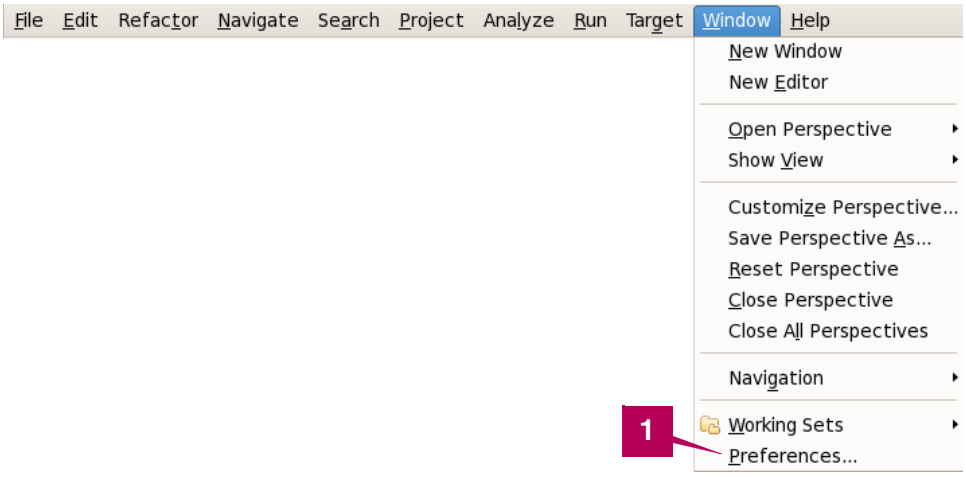
5.3.2 Using the Wind River Performance Profiler to Analyze Your Application

The Wind River Performance Profiler is a dynamic execution profiler that provides detailed function-by-function performance analysis, specifying individual routines within the program that are consuming the CPU cycles. It creates a direct map of what the CPU is doing, what routines are being called, and what routines they call. It points out where the inefficiencies are. This information is exactly what is needed to tune a time-critical system for optimum performance.

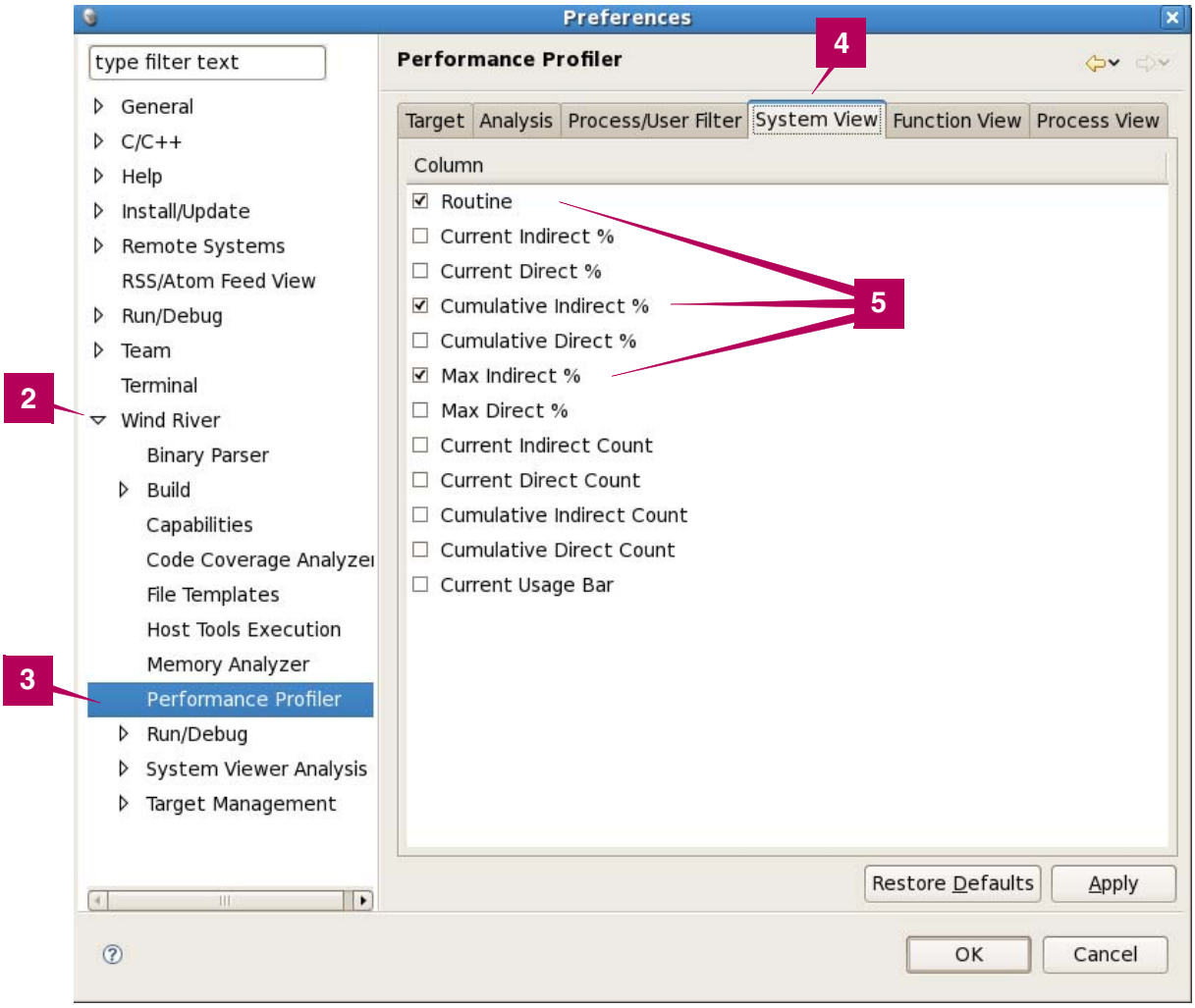
Performance Profiler View Configuration

The Performance Profiler can show many CPU statistics; in this application we are mostly interested in the tasks and functions that consume most of the CPU time. To configure the Performance Profiler to display these values, follow these steps:

1. From the main menu select **Window > Preferences**.

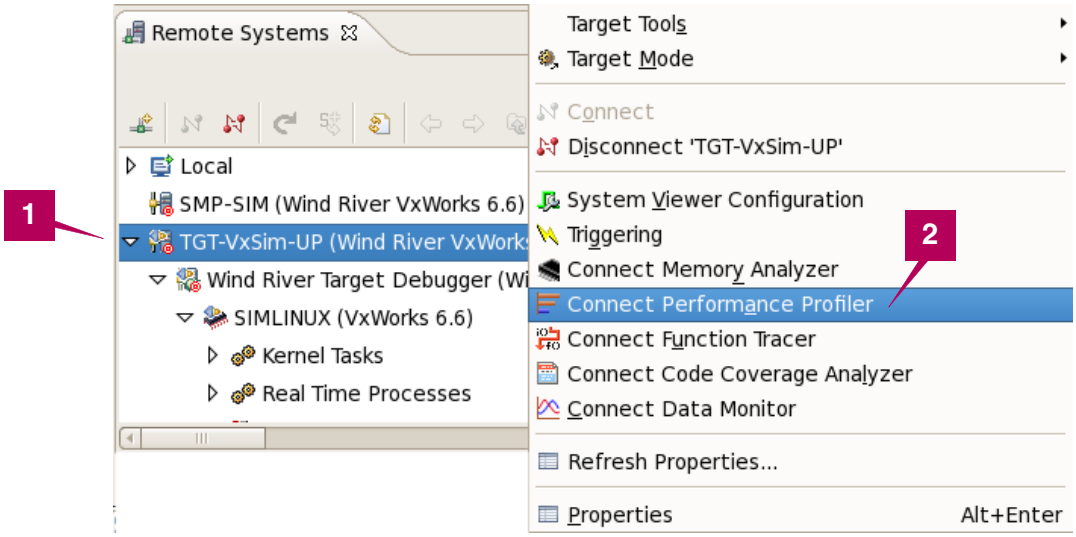


2. Expand the **Wind River** entry.
3. Select **Performance Profiler** from the drop-down options.
4. Select the **System View** tab.
5. Select the **Routine**, **Cumulative Indirect**, and **Max Indirect** check boxes.
6. Click **OK** to complete Performance Profiler view configuration.

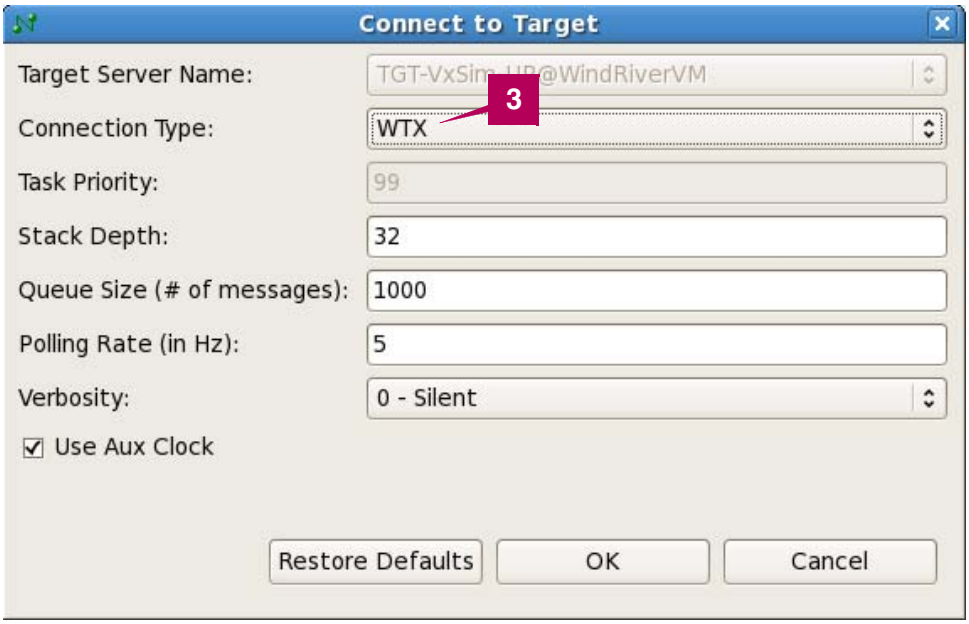


Starting Performance Profiler

1. Right-click **TGT-VxSim-UP** in the Remote Systems view.
2. From the context menu, select Connect Performance Profiler.

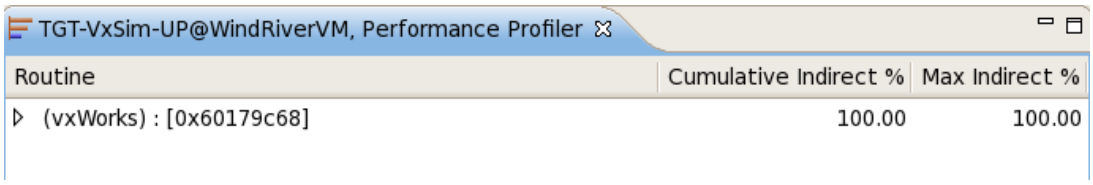


3. In the Connect to Target dialog, change **Connection Type** from **TCP/IP** to **WTX**.
4. Select **OK**.

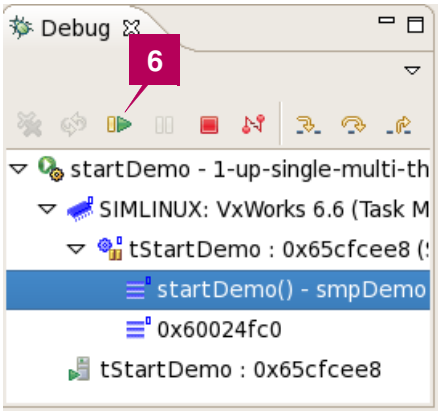


NOTE: TCP/IP is used to connect the Performance Profiler to a target over Ethernet when using a real target, whereas WTX is used when using the VxWorks simulator.

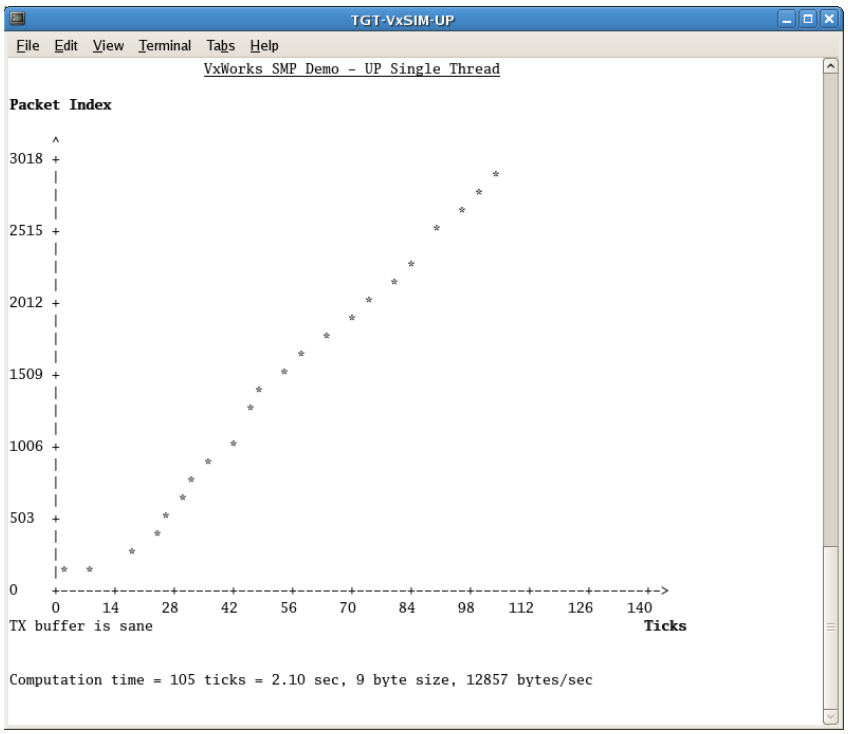
5. You should see the following window in Workbench:



6. The application is now ready for execution. Start the application by clicking on the **Resume** button in the Debug view.

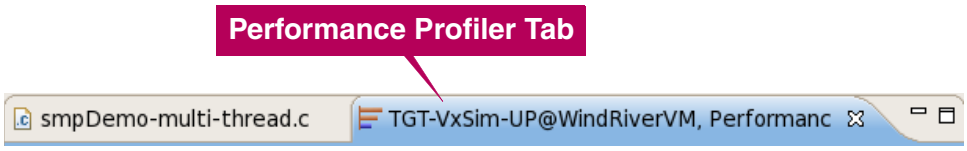


7. When the application has completed, you should see the following output on the simulator:

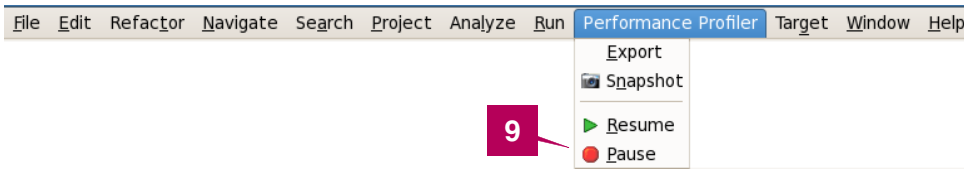


NOTE: Results may vary depending on your host CPU type and speed, number of cores, memory size, and system load.

8. Stop the performance data collection by selecting the Performance window in the editor.

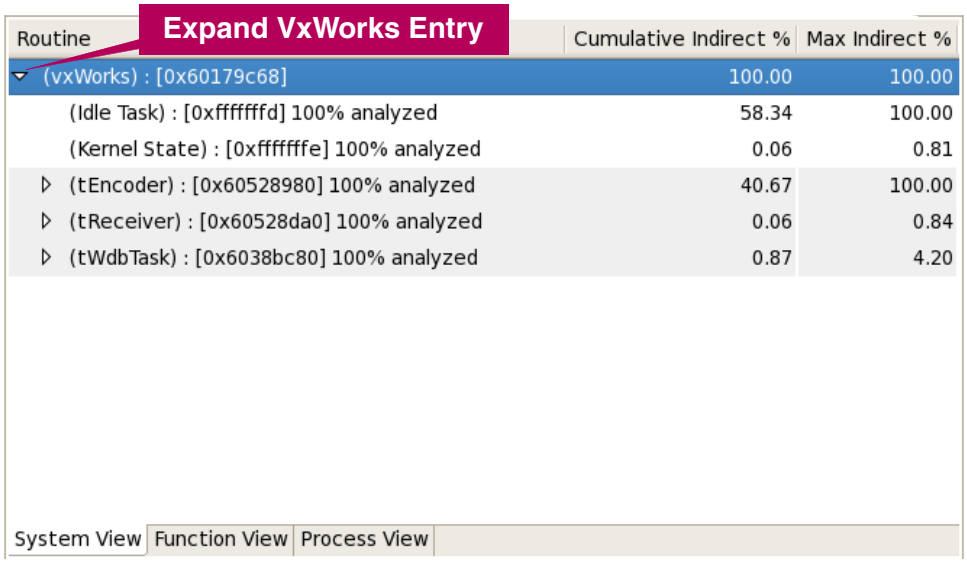


9. From the main menu, select **Performance Profiler > Pause**.



Analyzing the Data

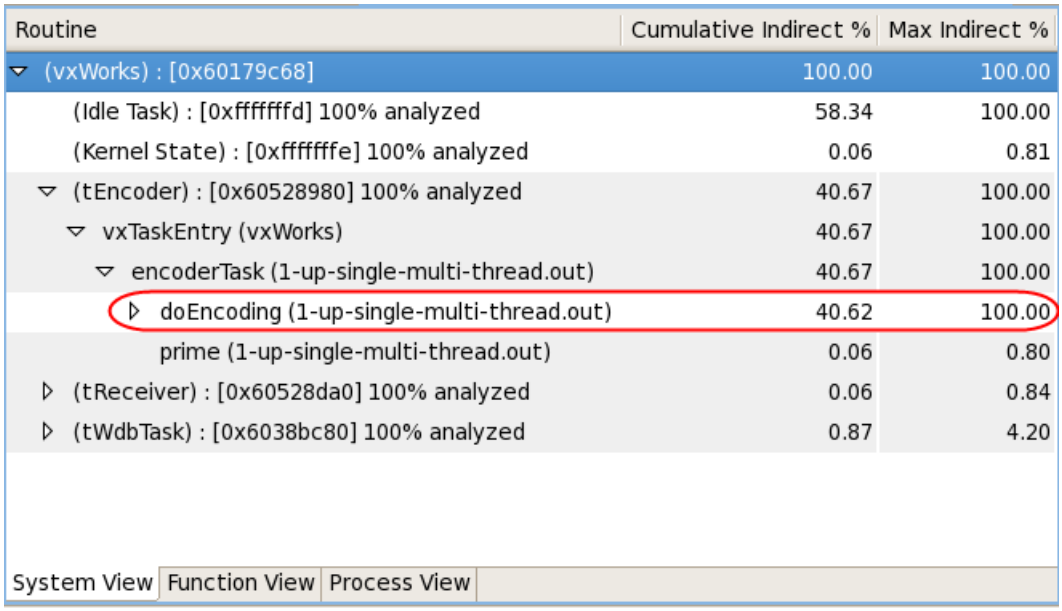
The Performance Profiler should have the following status after you have expanded the **VxWorks** entry:



This view tells us how much of the CPU time (in terms of percentages) was allocated for each task in the system. In particular, the **Max Indirect %** column tells us the **maximum CPU consumption value** over the sampling duration, so it shouldn't then surprise us that the sum of the totals in this column will exceed 100%. We use this figure in the investigation since it gives us a clearer postmortem

view of where CPU resources were allocated during the sampling duration. You can clearly identify that the most CPU-intensive task is **tEncoder**.

In fact, you can expand each task by clicking on the triangle on the left of the task to see which functions in the task are doing most of the application work:



Notice that the **doEncoding( )** routine is the most expensive in the **tEncoder** task. This makes sense, because the CPU has to run an encoding algorithm, while the receiver and sender tasks are just moving buffer pointers.

5.4 Redesigning Your Application for SMP

After you have identified what portion of your UP application will benefit from SMP, you need to redesign this part, and change it to be able to take advantage of SMP. In this section you will:

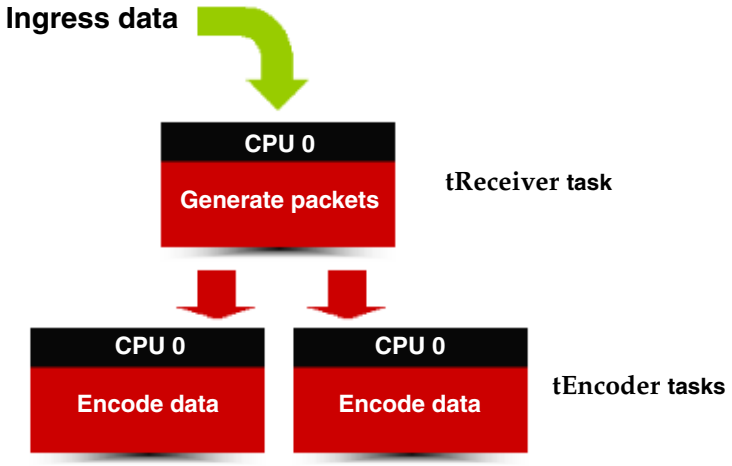
- Get an overview of design considerations for SMP applications.
- Learn about the API changes that you should be aware of when migrating from UP to SMP.

5.4.1 SMP Design Considerations

Now that you have identified the most CPU-intensive task, you are ready to design the SMP application. Since we found that **tEncoder** is the greediest task in terms of CPU usage, it makes sense to replicate this task. This way a performance benefit will be realized if these tasks can be scheduled concurrently on multiple cores.

Identifying the expensive/busy tasks in the code is essential for SMP design; these are the areas where parallelism should be used to improve performance.

The following figure depicts the application's SMP design:



However, this redesign introduces a new problem: we now have two **tEncoder** tasks competing for the same data.

In a single-core world, a typical solution is to disable preemption when a task accesses a shared resource. Thus, when an encoder task accesses data from the receiver task, the task disables the scheduler from preempting itself and starts a critical section. When the task has finished retrieving data from the receiver task, it re-enables preemption and this is the end of the critical section. This type of mutual exclusion is referred to as a preemption lock, and is supported in VxWorks UP by **taskLock( )** and **taskUnlock( )**.

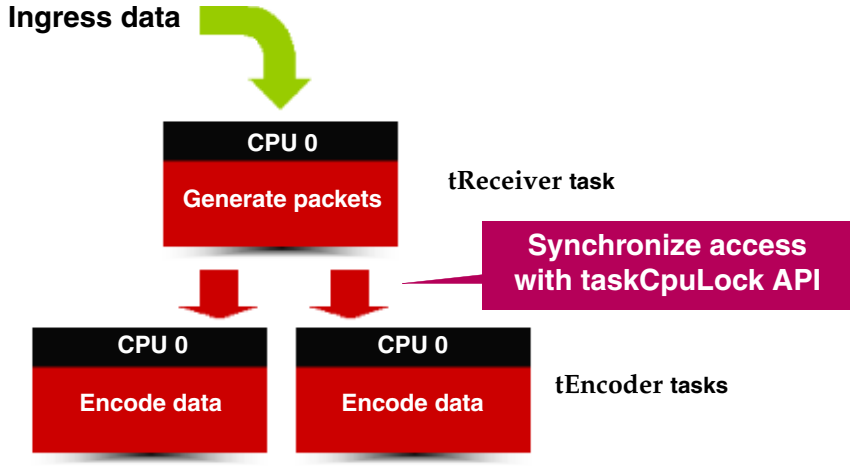
5.4.2 UP-to-SMP API Changes

In VxWorks UP, task-locking routines are used by a task to prevent the scheduling of any other task in the system, until it calls the corresponding unlock routine. The typical use of these routines is to guarantee mutually exclusive access to a critical section of code.

This mechanism would be inappropriate for a multiprocessor system, which is why routines like **taskLock( )** and **taskUnlock( )** are not supported on VxWorks SMP. When such an unsupported API is used on an SMP-enabled kernel, the VxWorks dynamic linker will complain about unresolved functions.

For task locking, VxWorks SMP provides the alternative **taskCpuLock( )** routine for situations where all tasks taking part in the task-locking scenario have the same CPU affinity (bounded to the same core).

However, note that this approach should not be used in custom extensions to the operating system other than as a temporary measure when migrating the code from UP to SMP, since it limits the ability for the OS to effectively balance the workload across cores.



5.5 Testing the Redesigned Application in UP Mode

In the previous step you identified what portions of your UP application will benefit from SMP. You now need to redesign these portions, and change them to be able to take advantage of SMP. Now you will test your redesigned application to see if it runs properly in UP mode. This is an important step in UP-to-SMP migration, since the first step to achieve parallelism is by multi-threading the part that should run in parallel, and the easiest way to verify that the multi-threading works properly is by testing it in UP mode.

In this section, you will see how to do this part by:

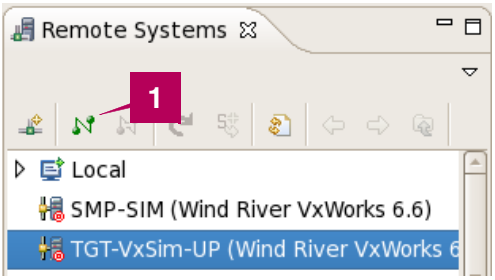
- Running a precreated redesigned version of the demo application that implements the SMP design consideration (multi-threading) in UP mode.
- Learning how to use the Wind River System Viewer to determine if the redesign of the application works as expected.

5.5.1 Running the Redesigned Application in UP Mode

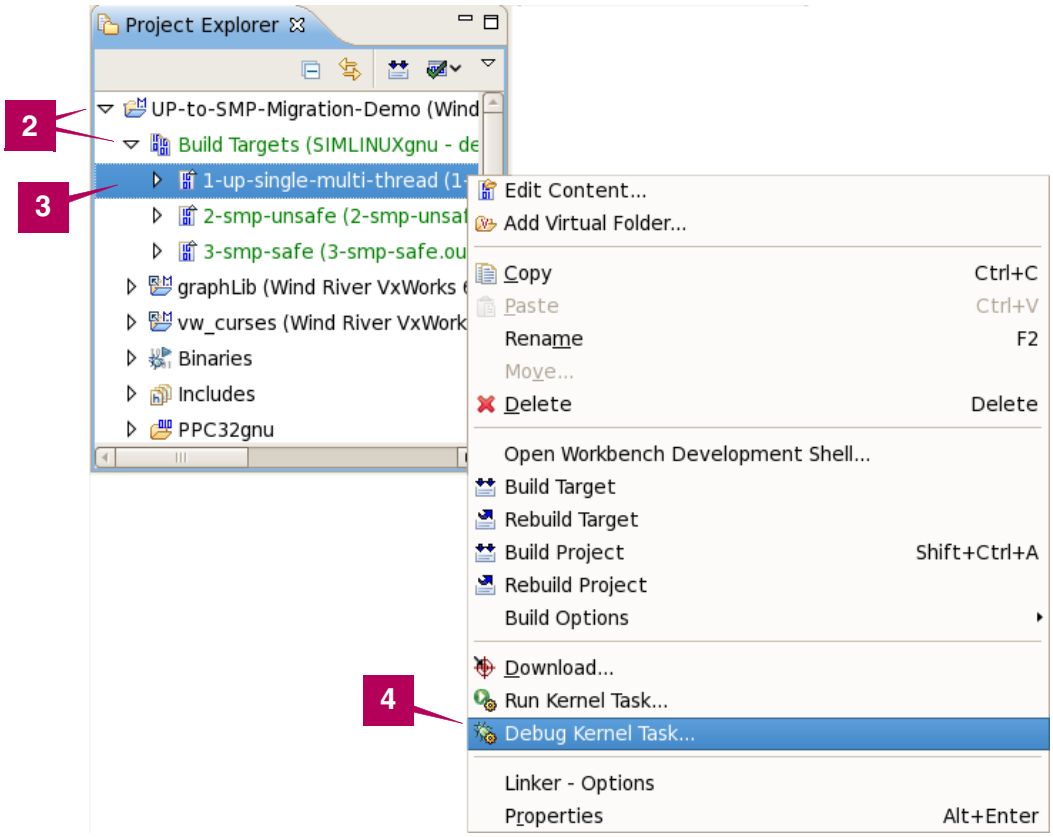
To save time we have supplied a premodified version of the demo application which implements the SMP design considerations using two encoder tasks. Follow the steps below to run the new SMP application design (with two Encoder tasks) on UP.

NOTE: Make sure the previous instance of the UP VxWorks simulator is closed! To close it, simply left-click on the **TGT-VxSim-UP** entry in the Remote Systems view to select it, then click the **Disconnect** button to close it.

1. Launch the simulator by selecting the **TGT-VxSim-UP** simulator name in the Remote Systems view, and clicking on the **Connect** button.



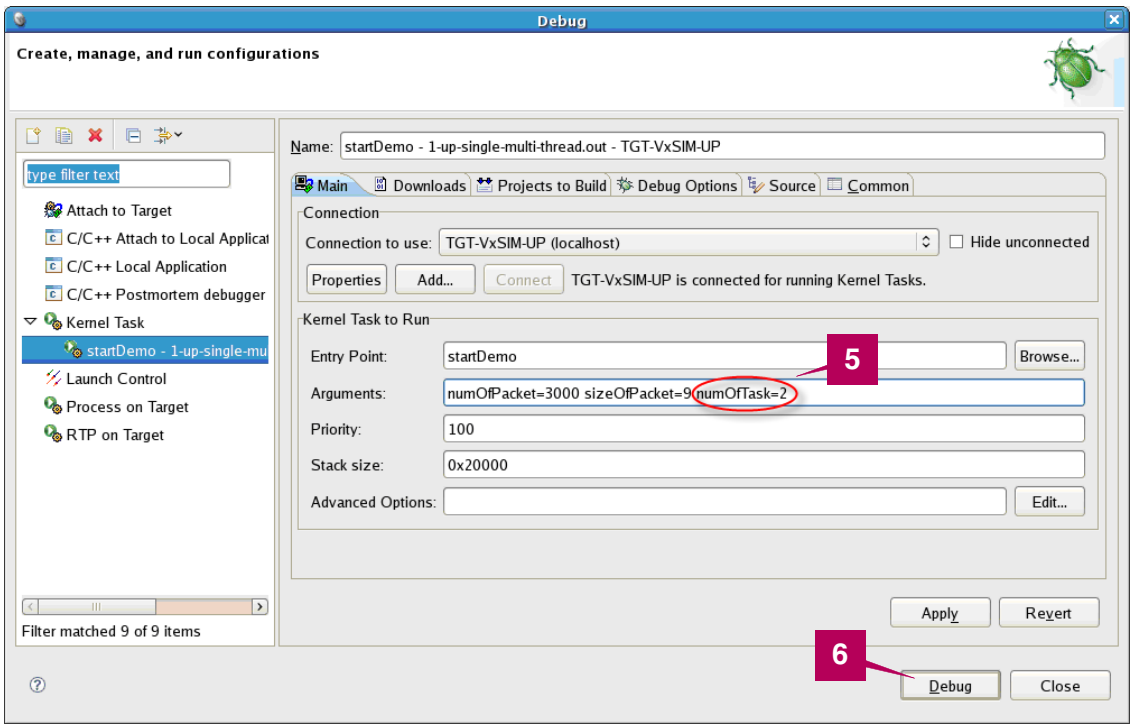
2. From the Project Explorer view, expand the **UP-To-SMP-Migration-Demo** project and the **Build Targets** item within it.
3. Right click on the first project, **1-up-single-multi-thread**.
4. Select **Debug Kernel Task** from the context menu.



5. In the Debug dialog, verify that **Arguments** is set to **numOfPacket=3000 sizeOfPacket=9 numOfTask=2**.

NOTE: The **numOfTask** argument is now set to 2 rather than 1.

6. Select **Debug**. This will launch the application, and you should see the source file in Workbench editor, and that the application is stopped in the Debug view.



5.5.2 Using the Wind River System Viewer

The Wind River System Viewer is the run-time analysis tool for device software developers who need to inspect the *dynamic* behavior of a system in order to detect run-time problems and improve system performance.

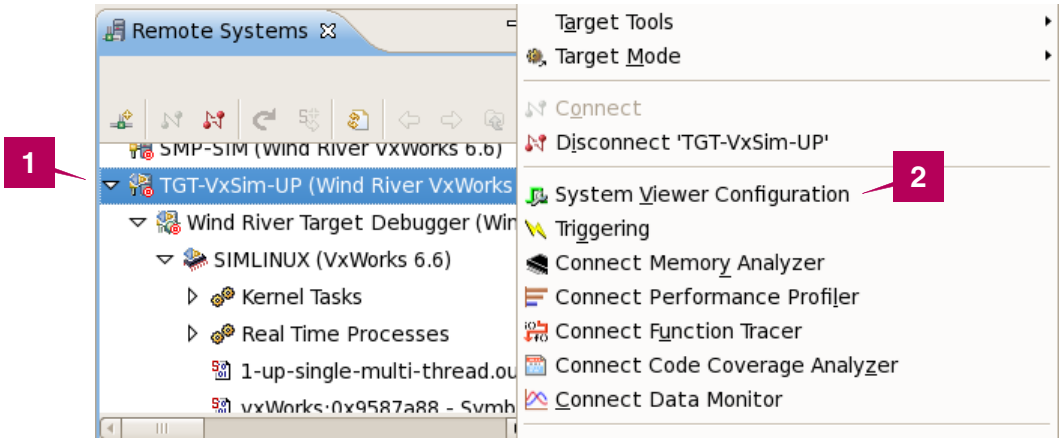
The System Viewer logs system events during execution in real time and then displays system events and task states in a graphical display that is intuitive and informative. There are two types of events that can be captured by the System Viewer:

- *System events* – such as a semaphore acquisitions, or task states changing from *running* to *pending*.
- *User events* – these events are configured by a user and can also include customized strings.

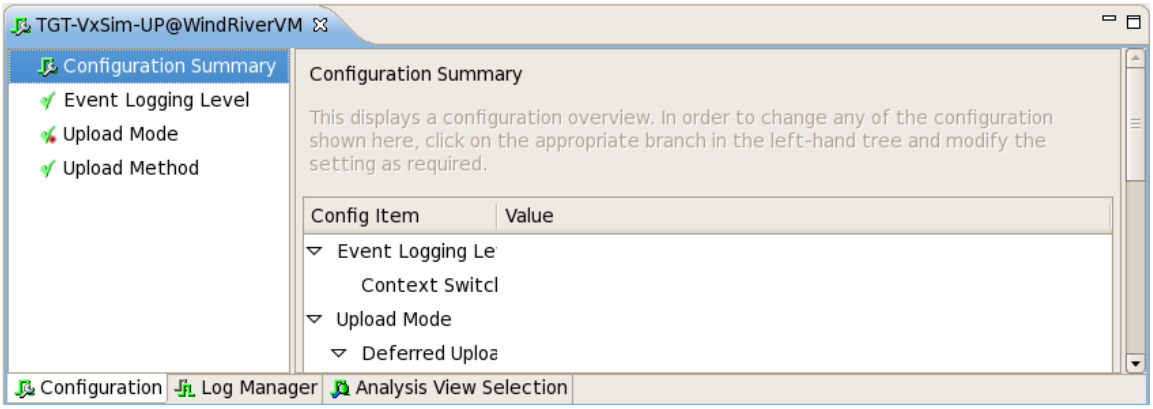
See the System Viewer documentation for more information about the tool's capabilities.

Starting the System Viewer

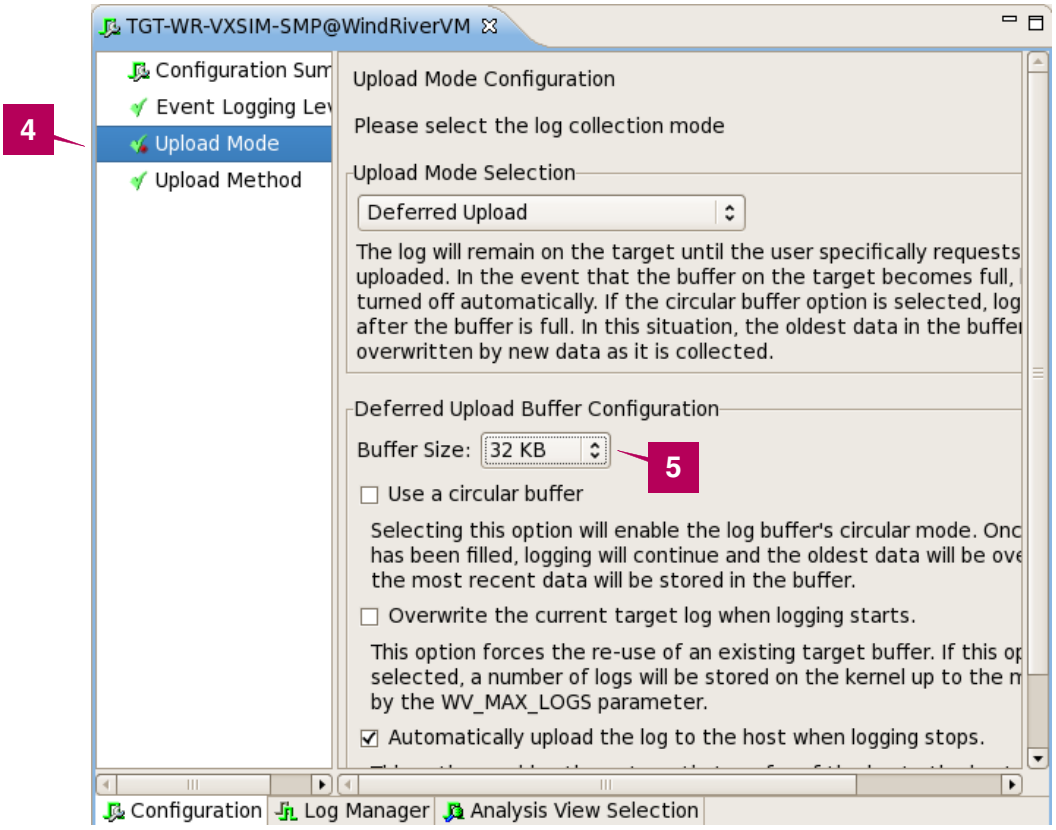
1. To launch the System Viewer, right-click on **TGT-VxSim-UP** in the Remote Systems view.
2. From the context menu, select **System Viewer Configuration**.



3. The following view will appear in the **Editor** area:



4. The default buffer size to store the System Viewer events is 32KB. To ensure that all data is captured, we need to enlarge the buffer. Select **Upload Mode**.
5. Use the **Buffer Size** control to adjust the size to **1024 KB**.

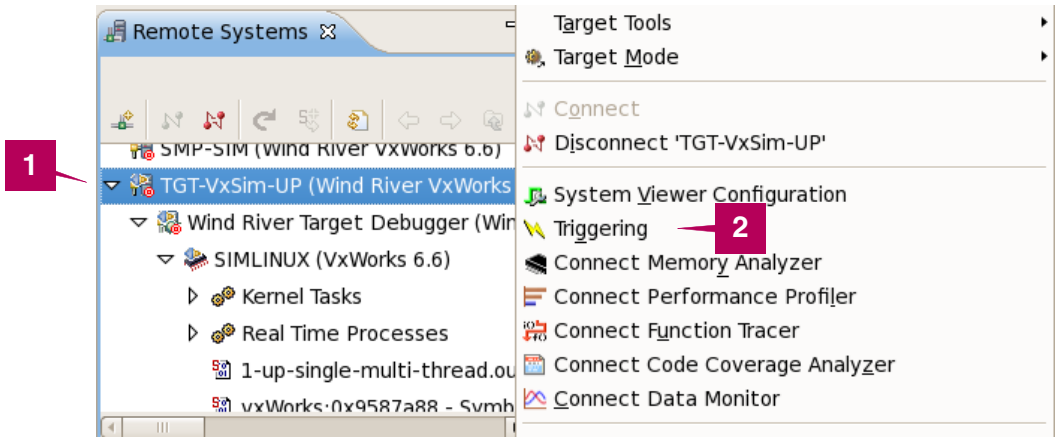


Using System Viewer Triggering

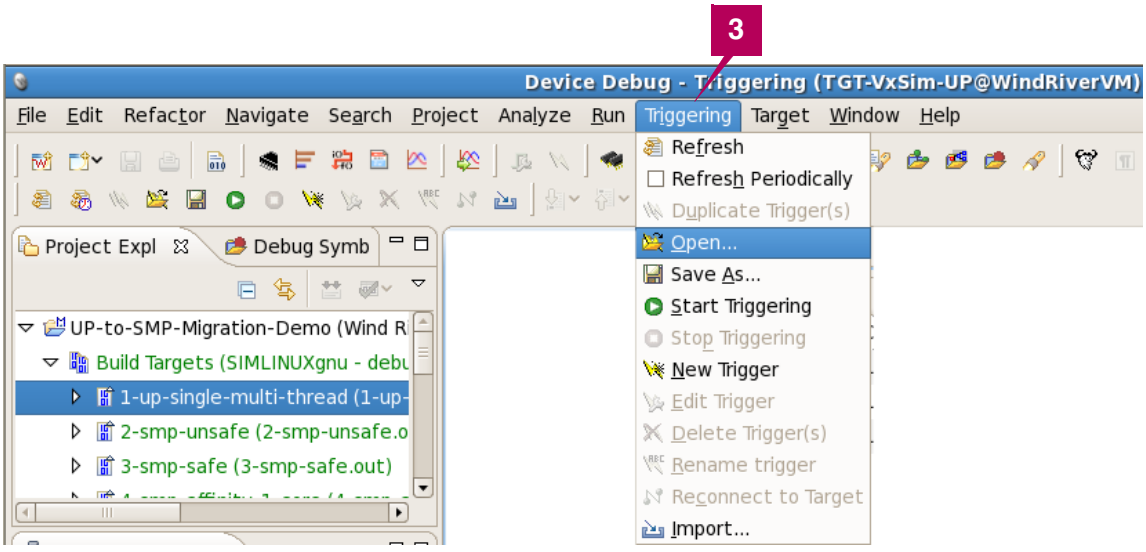
Triggering starts and stops diagnostic data collection upon invocation. For example, assume your application is very long with thousands of events; however, you are interested only in one small section of the application execution. With the help of triggering, you can configure the tool to start the data collection on only a small region of the application. To get the most out of the System Viewer, you will use a predefined triggering file. This will focus the System Viewer data collection on the interesting area of the execution. The interesting portion of the application is the encoder task's lifetime.

Follow these steps to start a triggering file:

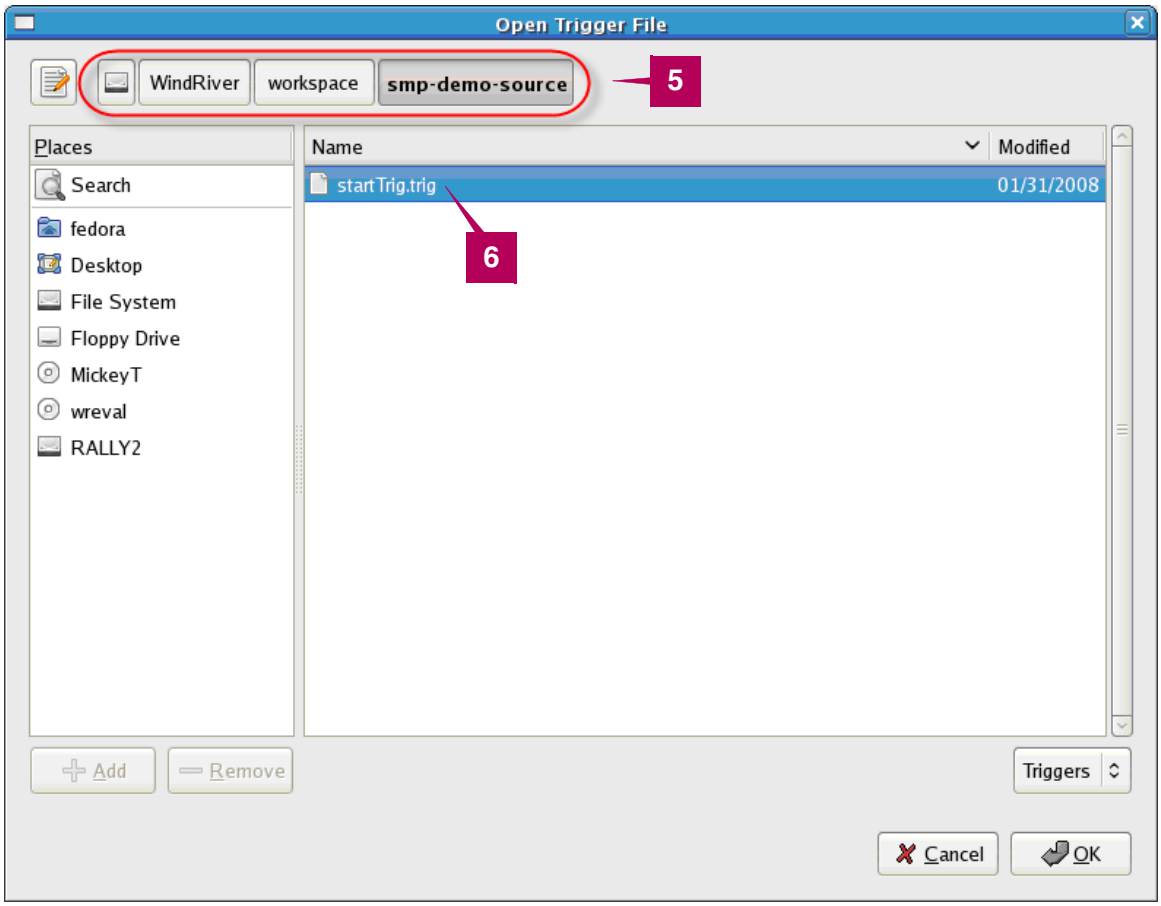
1. Right-click on **TGT-VxSim-UP** from the Remote Systems view.
2. From the context menu select **Triggering**.



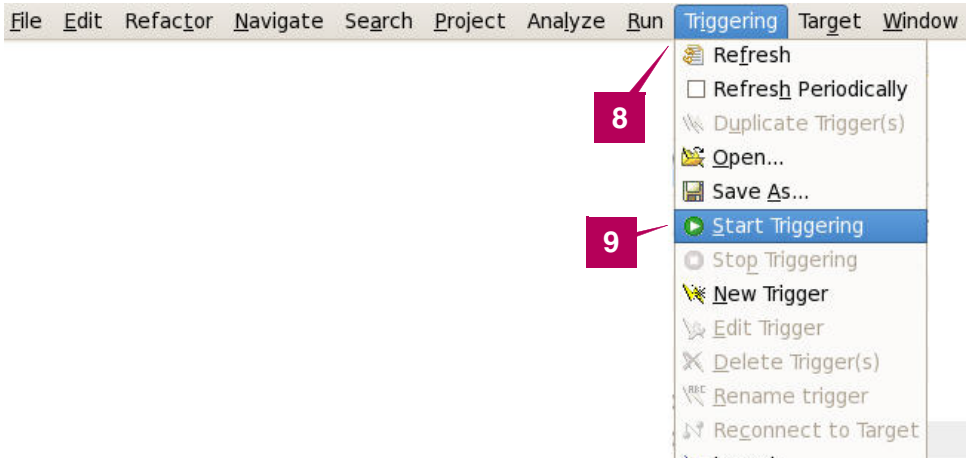
3. A new **Triggering** entry is now added to Workbench.
4. Click on the new **Triggering** entry and select **Open**.



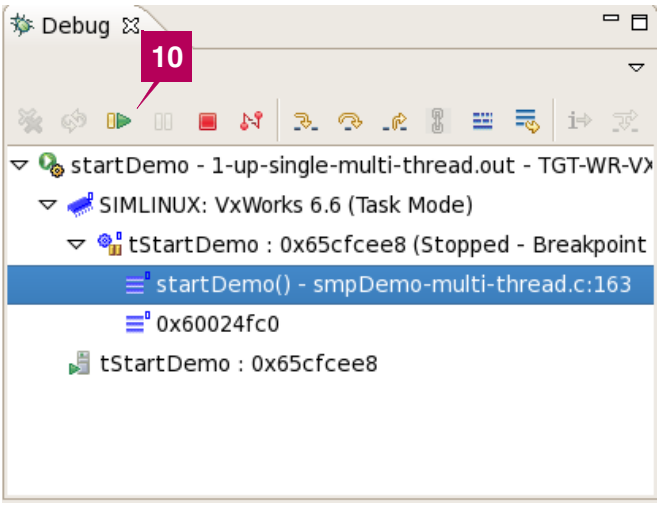
5. In the Open Trigger File dialog, browse to **/WindRiver/workspace/smp-demo-source**.
6. Select the **startTrig.trig** file.
7. Click **OK**.



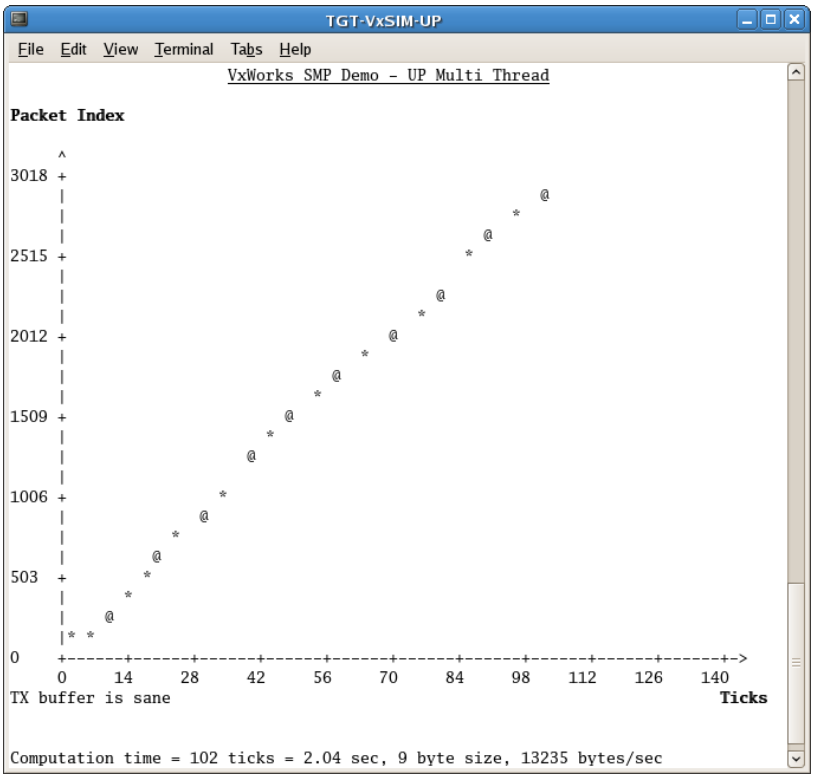
8. Click on the **Triggering** menu item
9. Select **Start Triggering**. This will arm the trigger.



10. Start the application by clicking the **Resume** button in the Debug window.



11. When the application has completed, you should see the following output on the simulator:



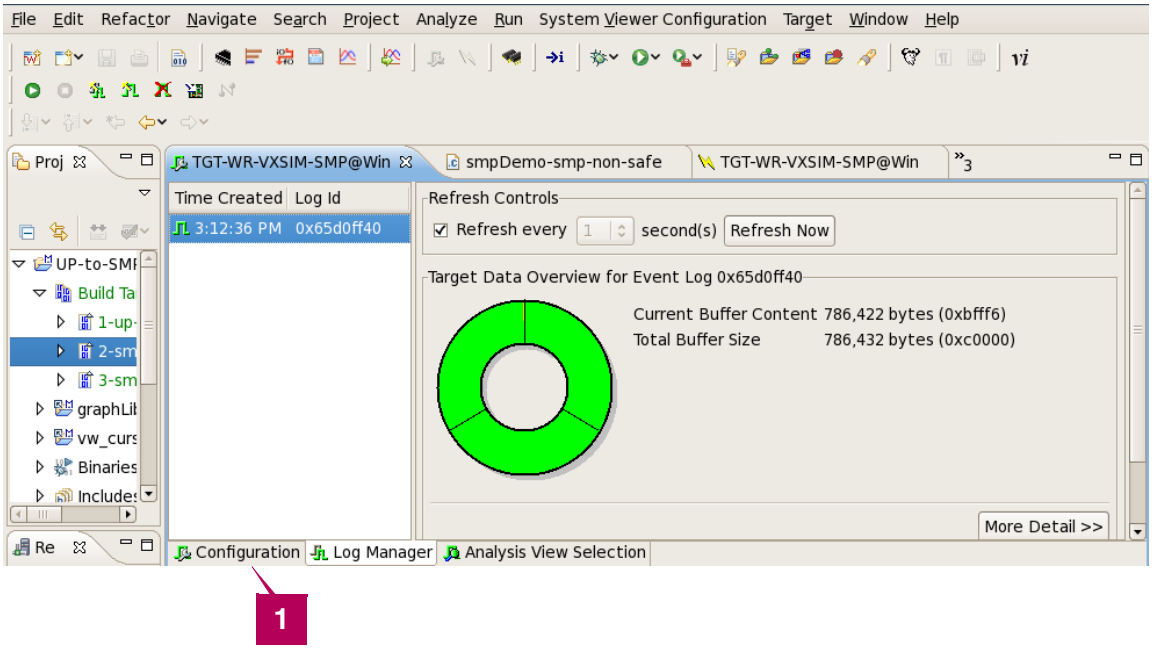
The asterisk (*) indicates data processed from the first encoder task, while the at sign (@) indicates data processed from the second encoder.

NOTE: Results may vary depending on your host CPU type and speed, number of cores, memory size, and system load.

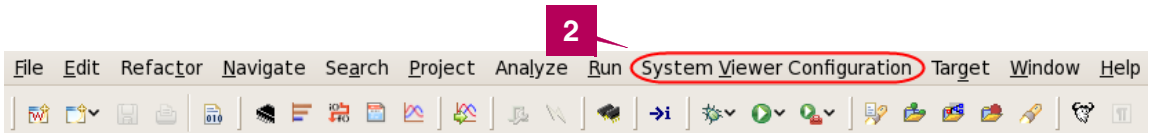
Displaying System Viewer Results

You can learn more about the program execution from the System Viewer. To view the execution events in the System Viewer, follow these steps:

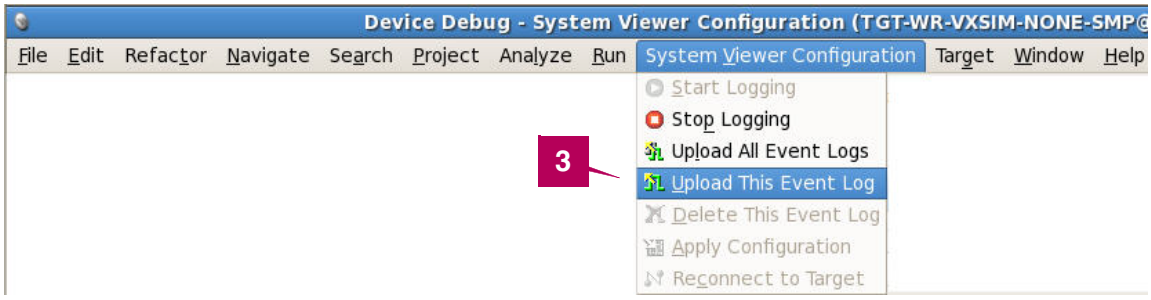
1. Click on **System Viewer Configuration** tab.



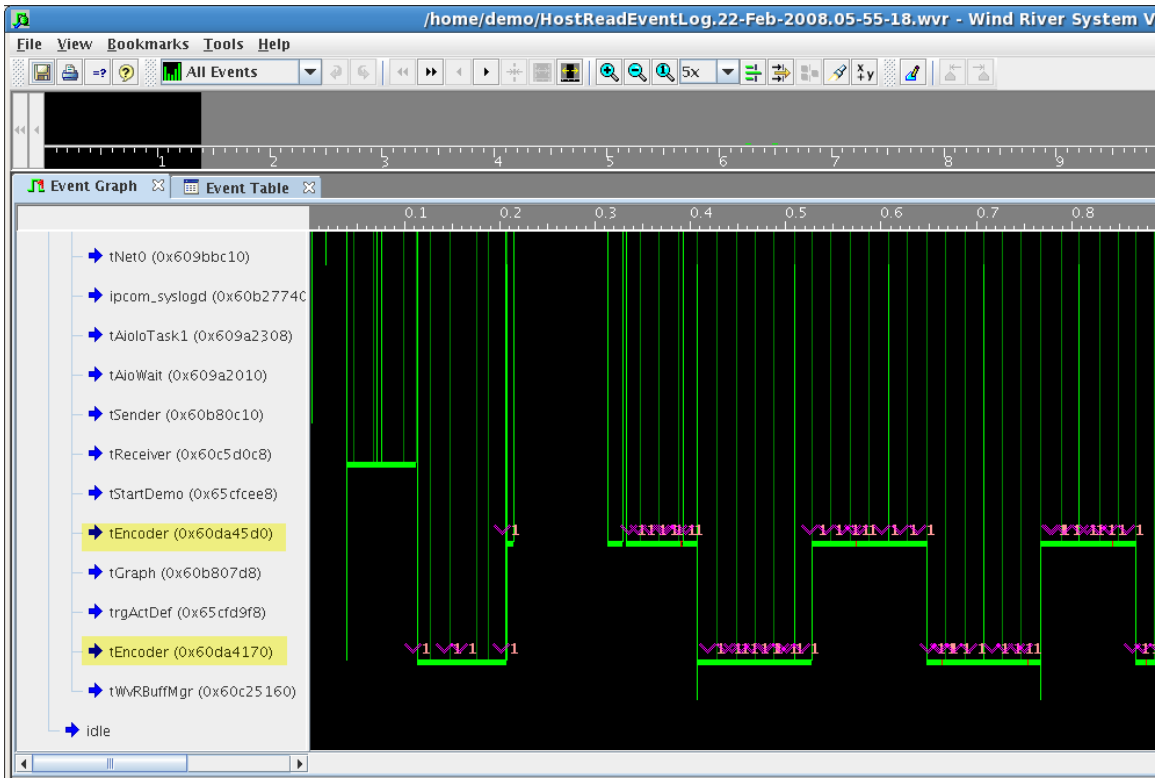
2. When the System Viewer configuration window is displayed, a new **System Viewer Configuration** menu entry is added to Workbench.



3. From the menu select **System Viewer Configuration Upload This Event Log**.

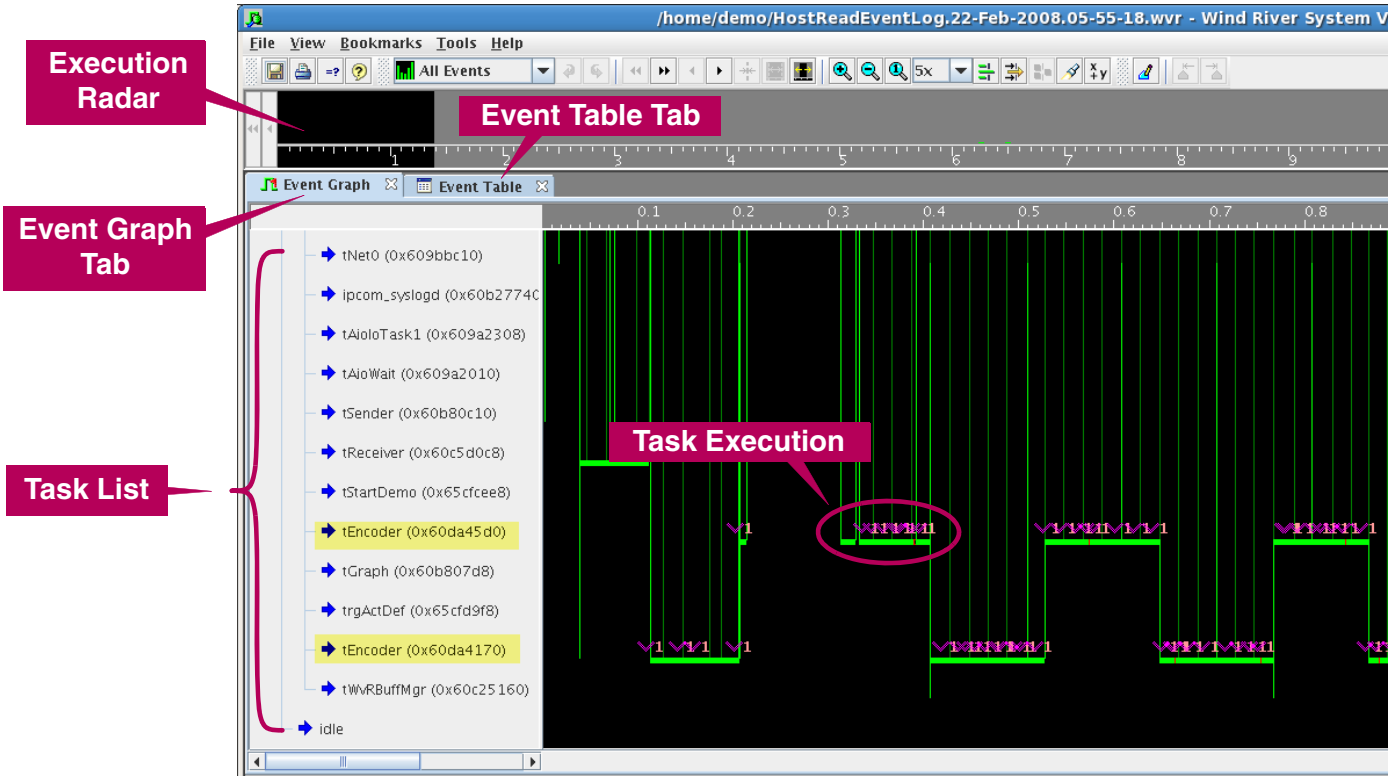


4. The following window appears:



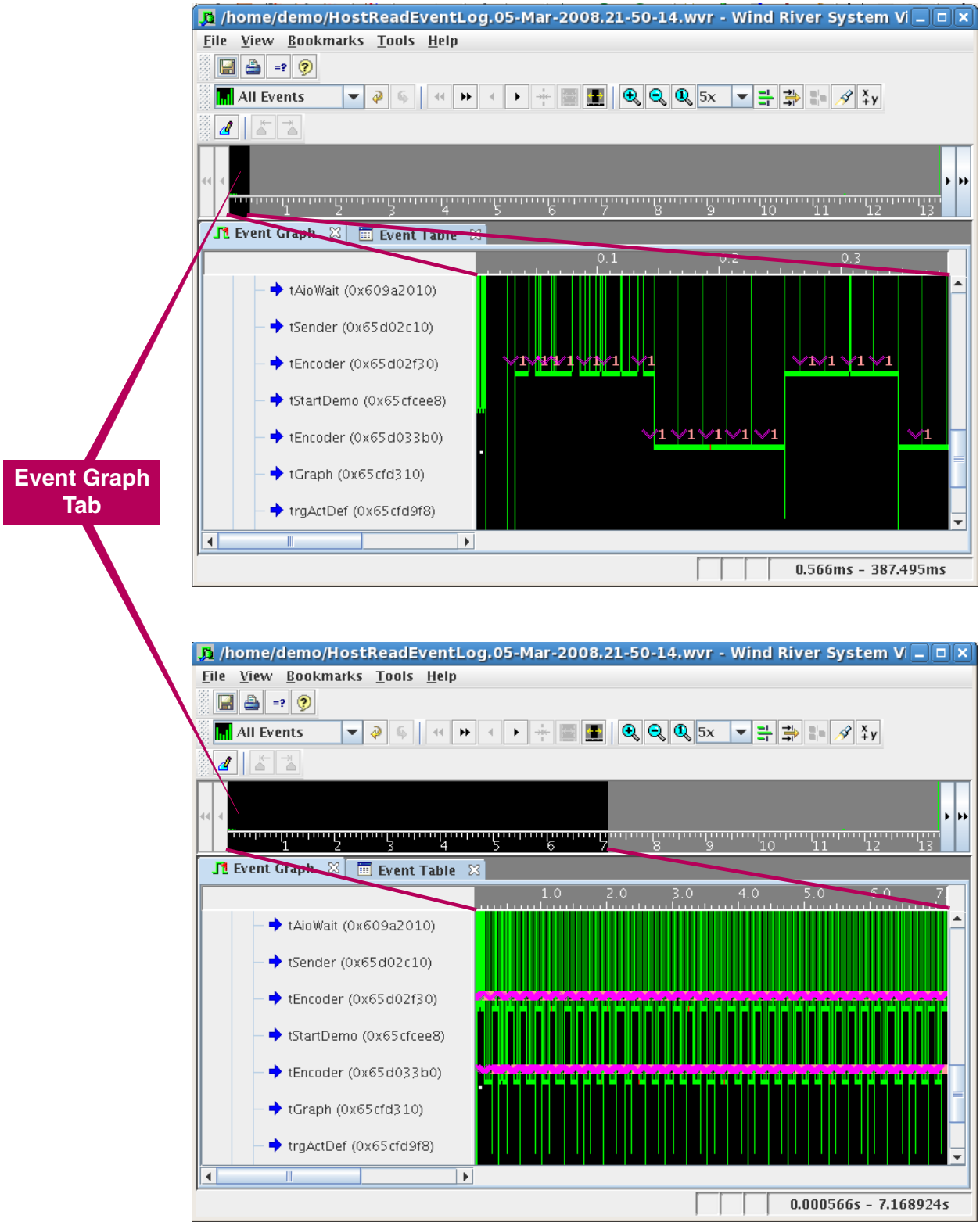
Using the System Viewer to Understand the Execution of the Redesigned Application

- Tasks are listed on the left side; a solid line represents an execution. You can see and trace the execution at each millisecond of execution.
- The execution radar is located at the top of the graph and highlights the current area shown in the graph.



Extending the System Viewer Display Area

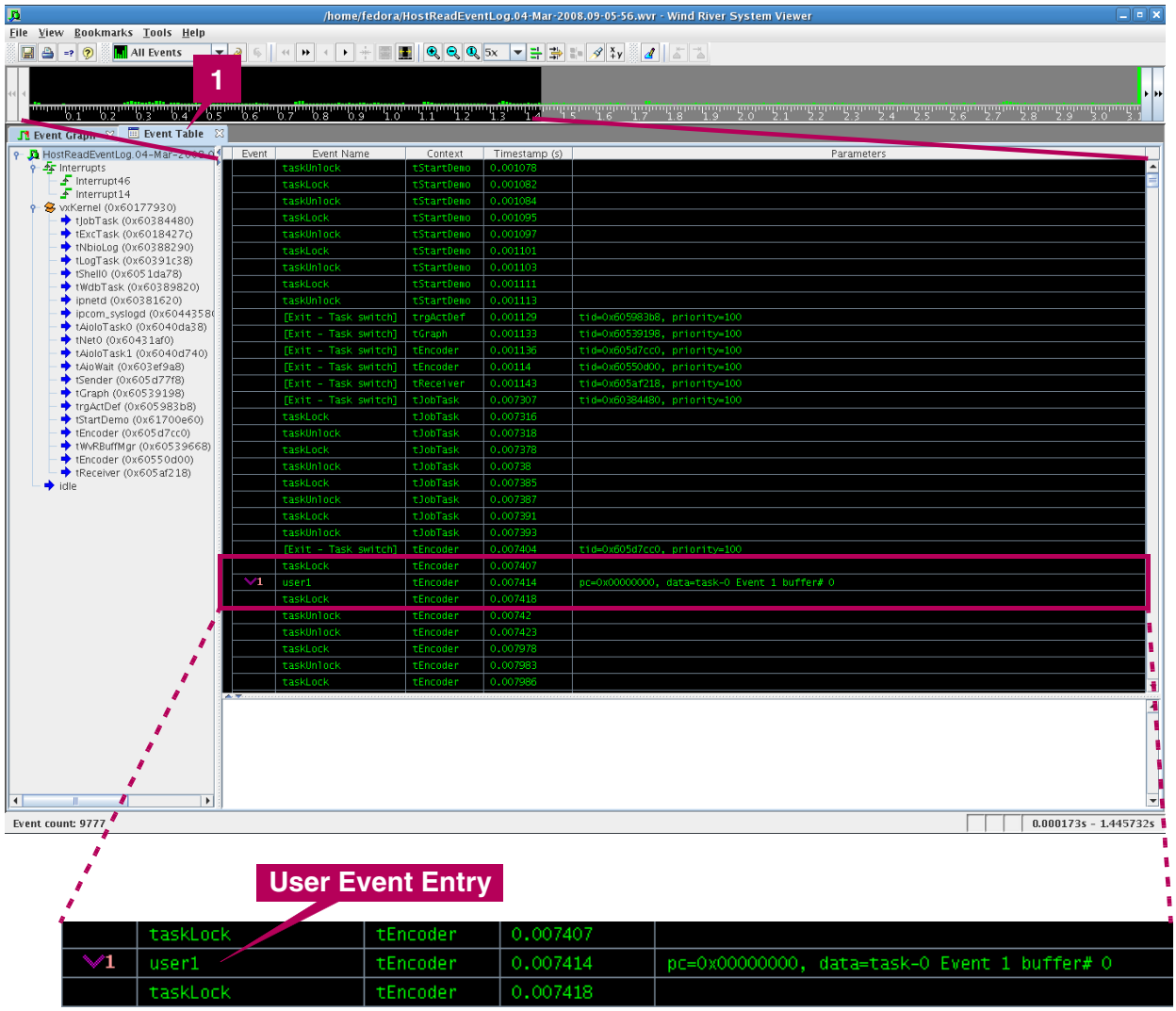
- To enlarge the execution display area click and drag the radar to the right:



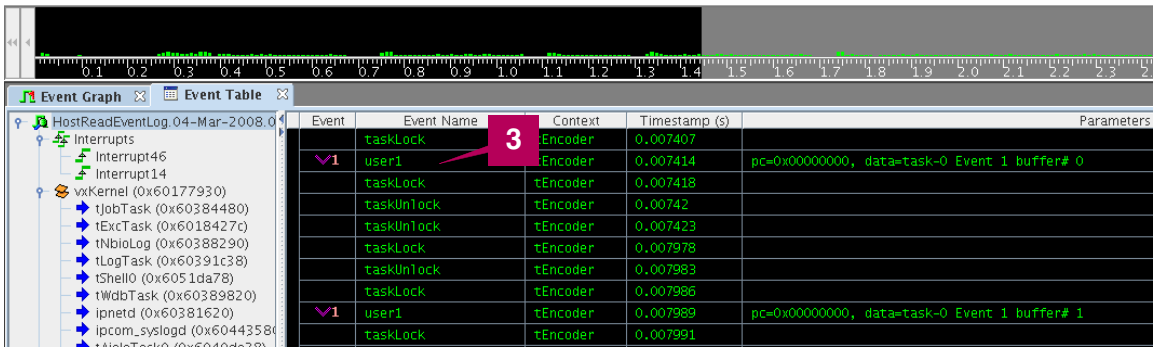
Identifying System Viewer Events

Notice that there are two encoder tasks. The numbers in the parentheses are the task IDs. The encoder tasks were busy most of the time, and the CPU bounces between executions of these tasks. The critical section has increased in the new SMP design, because the same data structure is shared between two encoder tasks. The application code generates a user event at the entry point of the critical section. This event is captured by the System Viewer and marked in the graph as **V1**. Each event is represented as an icon in the System Viewer graph; in addition, all events are listed in an event table.

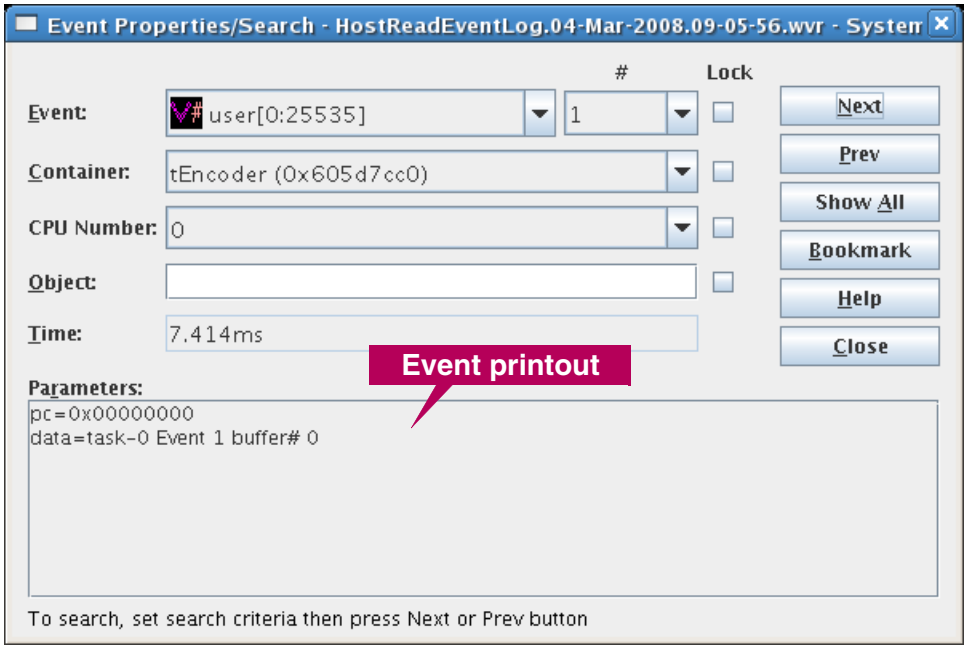
1. Click on the **Event Table** tab to switch to event table list.
2. Browse through the events in the table and identify a user event. User events can also include strings that are displayed in the table. Each time a user event is generated, the index of the buffer is saved in the event object. These strings can be seen in the event table.



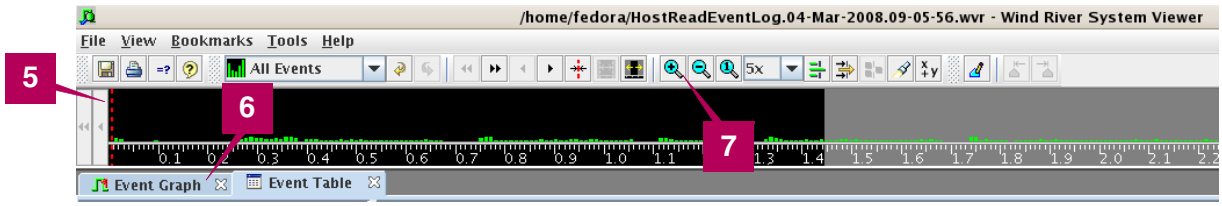
3. In order to establish a link between a specific event to an icon of that event in the execution graph, double-click on the event to highlight it.



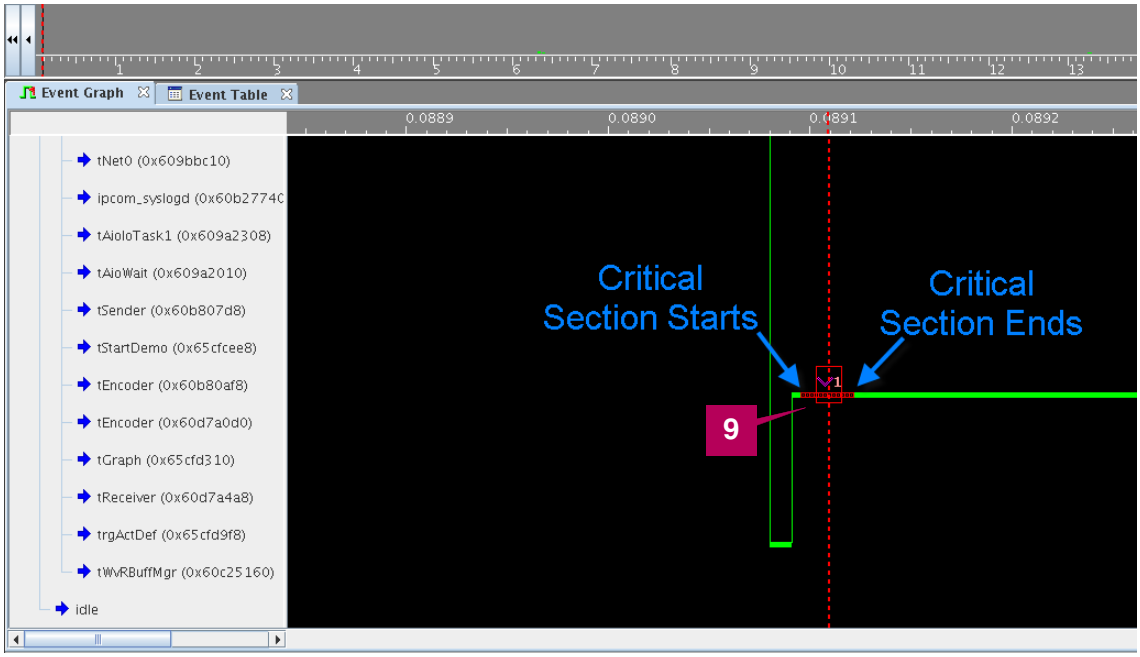
4. In the **Event Properties/Search** dialog, click **Close**.



5. Notice that a red arrow dot was added to the System Viewer radar.
6. Click on the **Event Graph** tab.
7. Zoom in with the **Zoom** button. The graph now shoes a more detailed area of the execution.



8. Move the cursor over the user event icon and notice that the event's string is displayed on the bottom-left side of the graph.
9. Notice that the color of the task changes from green to red as the task enters the critical section.



In the figure above, the arrows point to the critical section, the critical section in this application begins when one of the Encoder tasks retrieves data from the Receiver task. Since the status of the task is changed from **running** to **running+locked**, the color of the task's execution line changes from green to dotted red.

5.6 Testing the Redesigned Application in SMP Mode

In this step, you will enable multicore and run the application. This means that two or more tasks can run concurrently. Thus, if the application depends on implicit synchronization—for example if synchronization is based on task priority—then a race condition will exist, leading to unpredictable behavior.

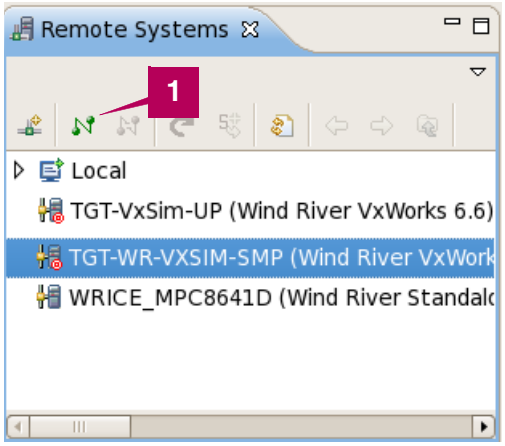
5.6.1 Running the Redesigned Application in SMP Mode

Follow these steps to run the new SMP application design (with two Encoder tasks) on SMP. This time you will use an SMP-enabled simulator.

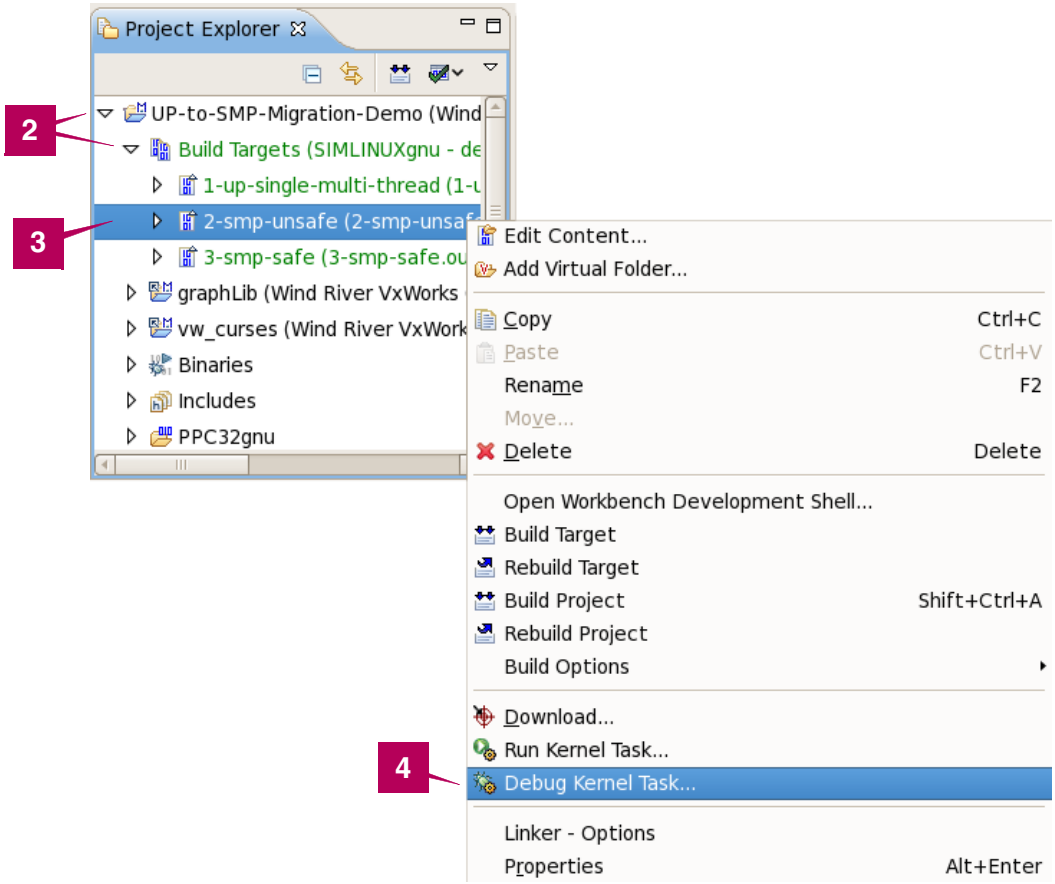
As in the previous part, here we have also supplied a premodified version of the demo application in order to save time. Follow these steps to run the new SMP application design (with two Encoder tasks) on SMP.

NOTE: Make sure the previous instance of the UP VxWorks simulator is closed! To close it, simply left-click on the **TGT-VxSim-UP** entry in the Remote Systems view to select it, followed by pressing on the **Disconnect** button to close it.

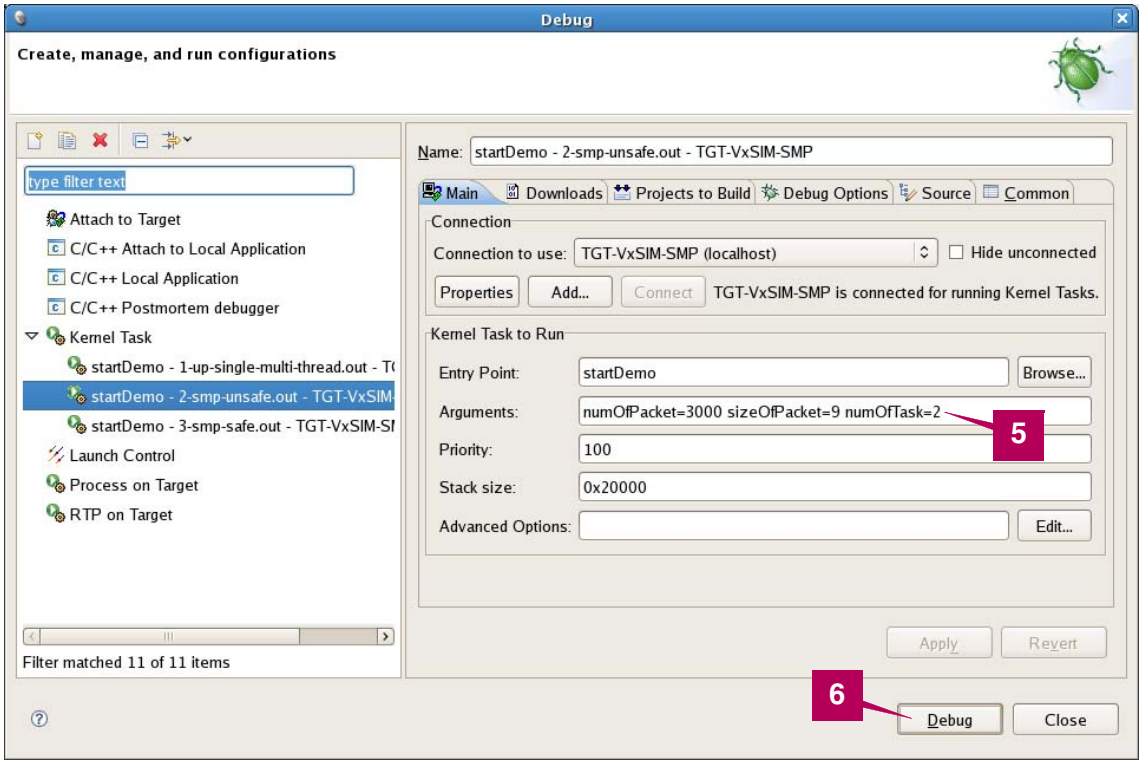
1. Launch the simulator by selecting the **TGT-WR-VXSIM-SMP** simulator in the Remote Systems view and clicking on the **Connect** button.



2. In the Project Explorer view, expand the **UP-To-SMP-Migration-Demo** and the **Build Targets** item within.
3. Right click on the second project, **2-SMP-unsafe**.
4. In the context menu, select **Debug Kernel Task**.



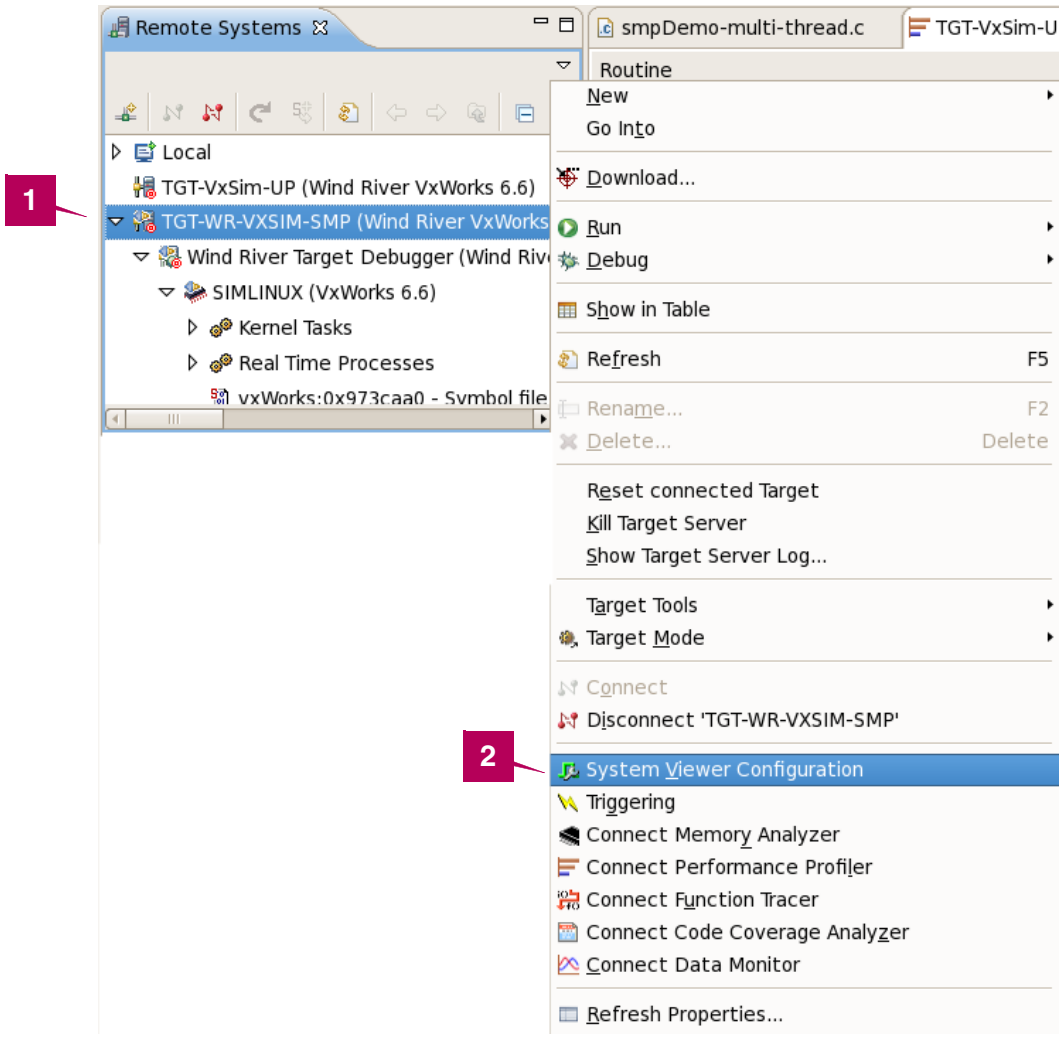
5. In the Debug dialog, verify that **Arguments** is set to **numOfPacket=3000 sizeOfPacket=9 numOfTask=2**.
6. Select **Debug**. This will launch the application, and you should see the source file in Workbench editor, and that the application is stopped in the Debug view.



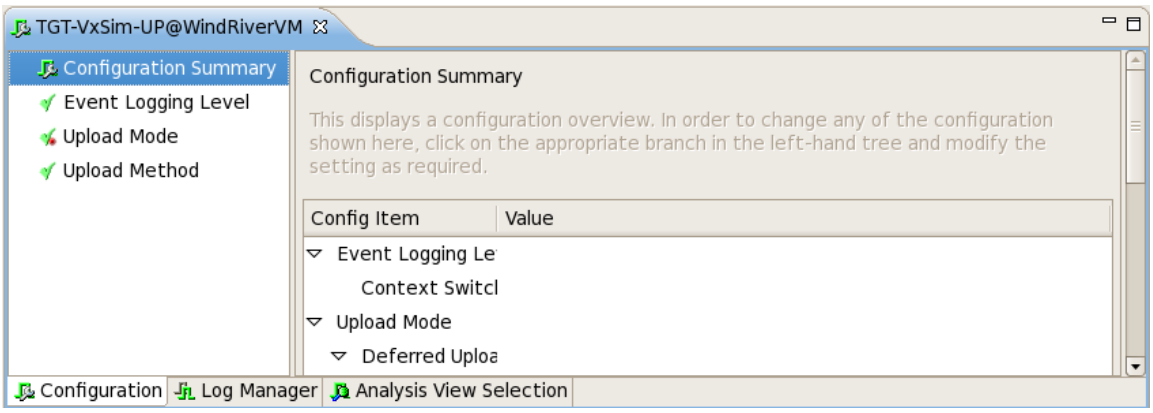
5.6.2 Using the Wind River System Viewer

Starting the System Viewer

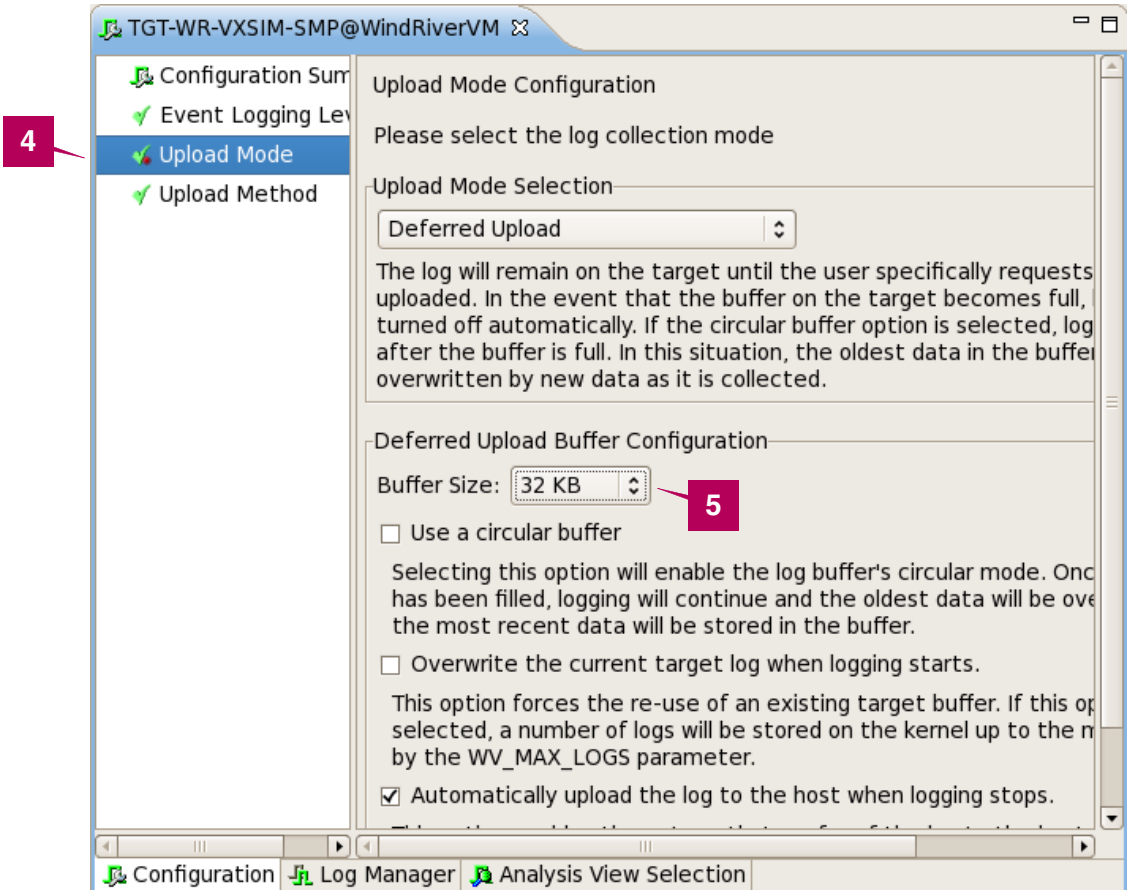
1. To launch the System Viewer, right-click on the **TGT-VxSim-SMP** from the Remote Systems view.
2. From the menu select **System Viewer Configuration**.



3. The following view will appear in the Editor area:



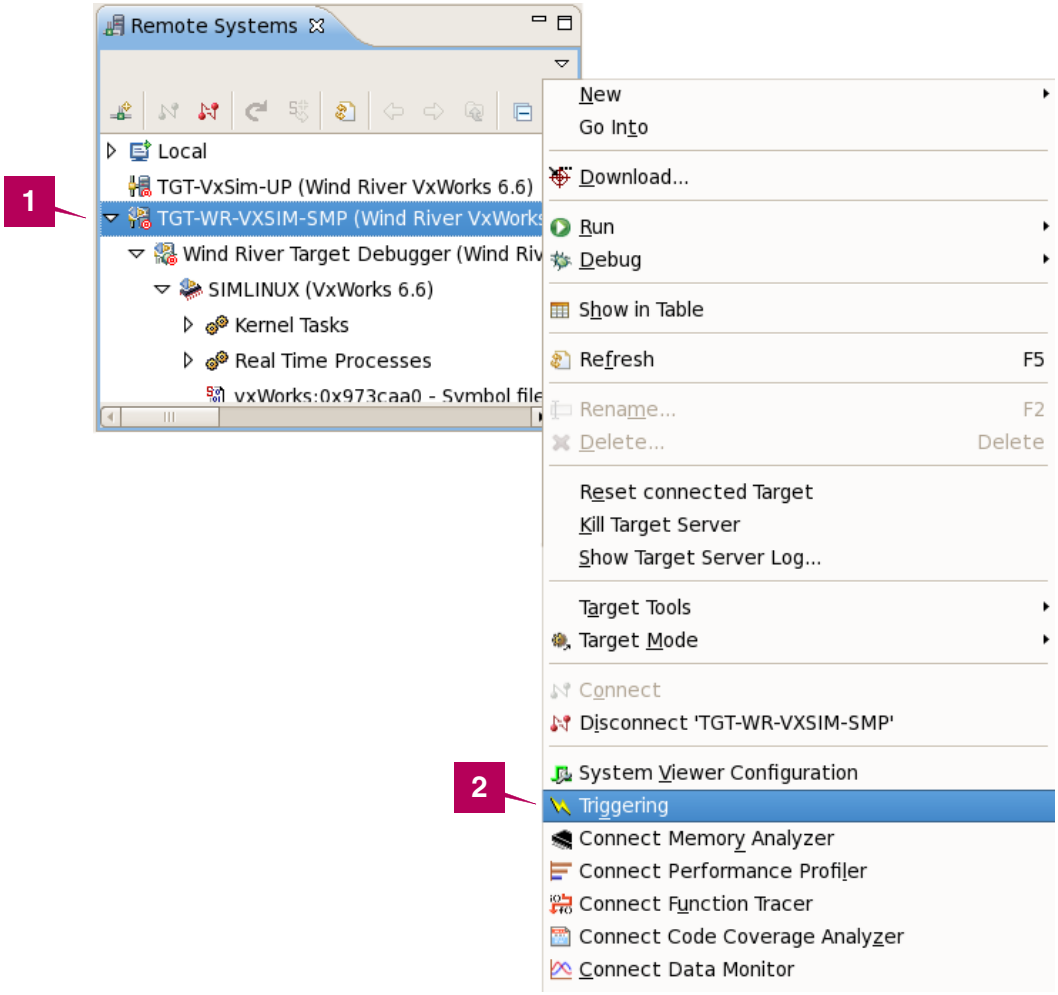
4. Again, we need to enlarge the buffer; otherwise there will be insufficient space to log all the events. Select **Upload Mode**.
5. Use the **Buffer Size** control to adjust the size to **1024 KB**.



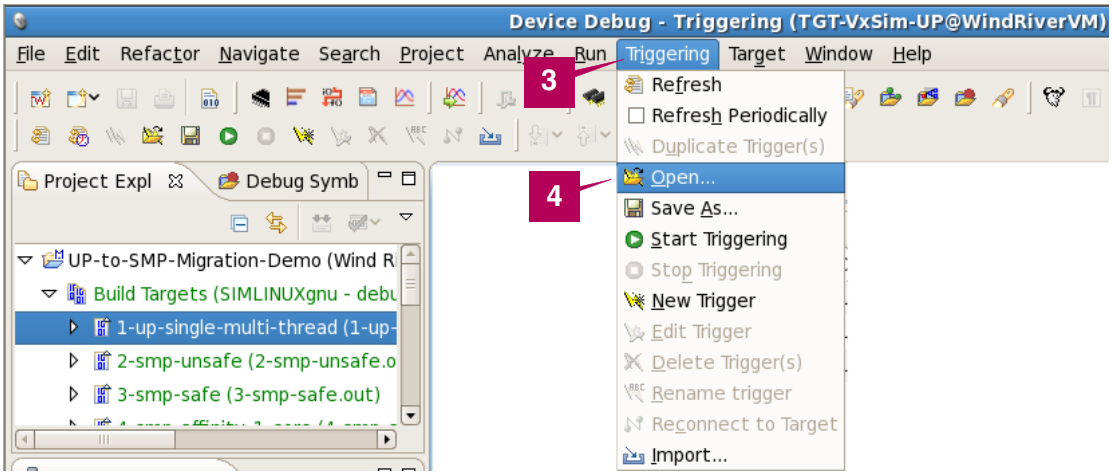
Using System Viewer Triggering

Start a triggering file as follows:

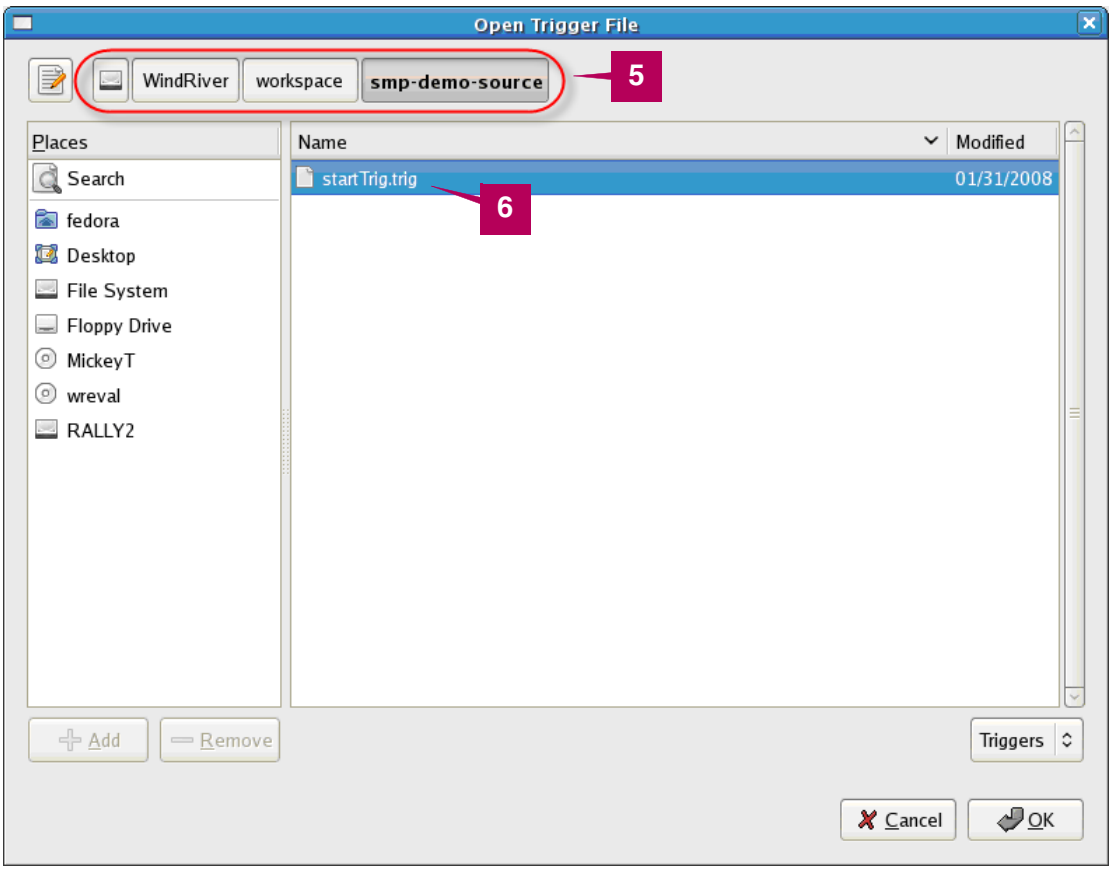
1. Right-click **TGT-VxSim-SMP** in the Remote Systems view.
2. In the context menu, select **Triggering**.



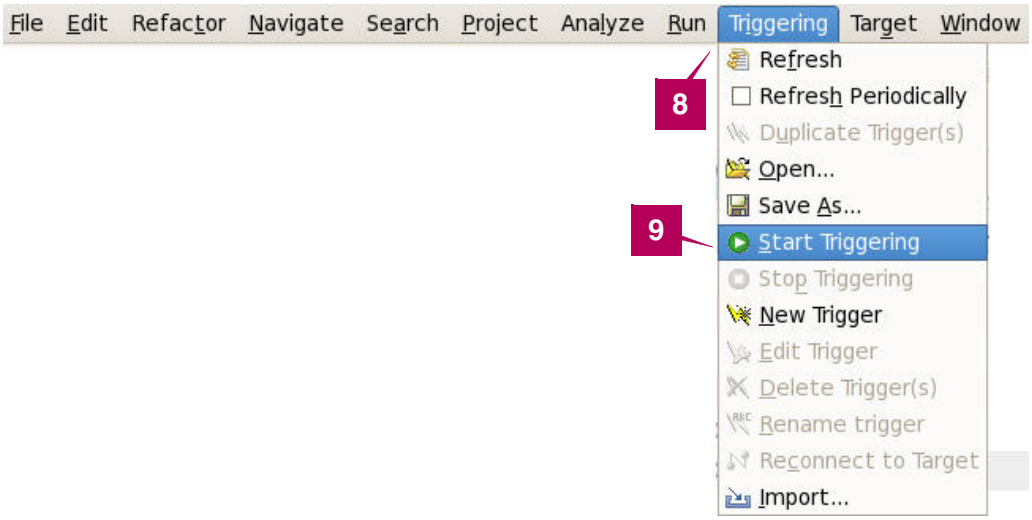
3. A new **Triggering** menu entry is added to Workbench. Select this new entry.
4. Select **Open**.



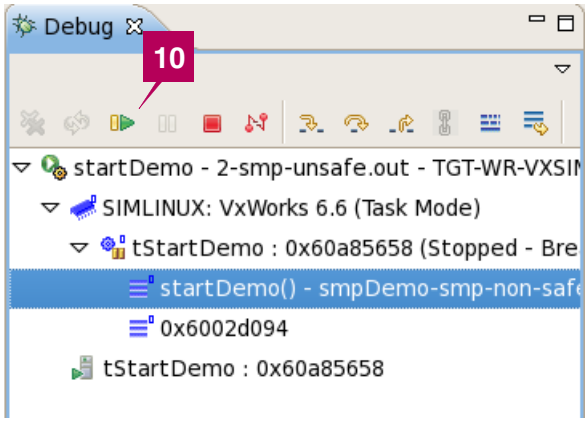
5. In the Open Trigger File dialog, browse to **WindRiver/workspace/smp-demo-source**.
6. Select the **startTrig.trig** file.
7. Select **OK**.



8. Select the **Triggering** menu item again.
9. Select **Start Triggering**, which will arm the trigger.

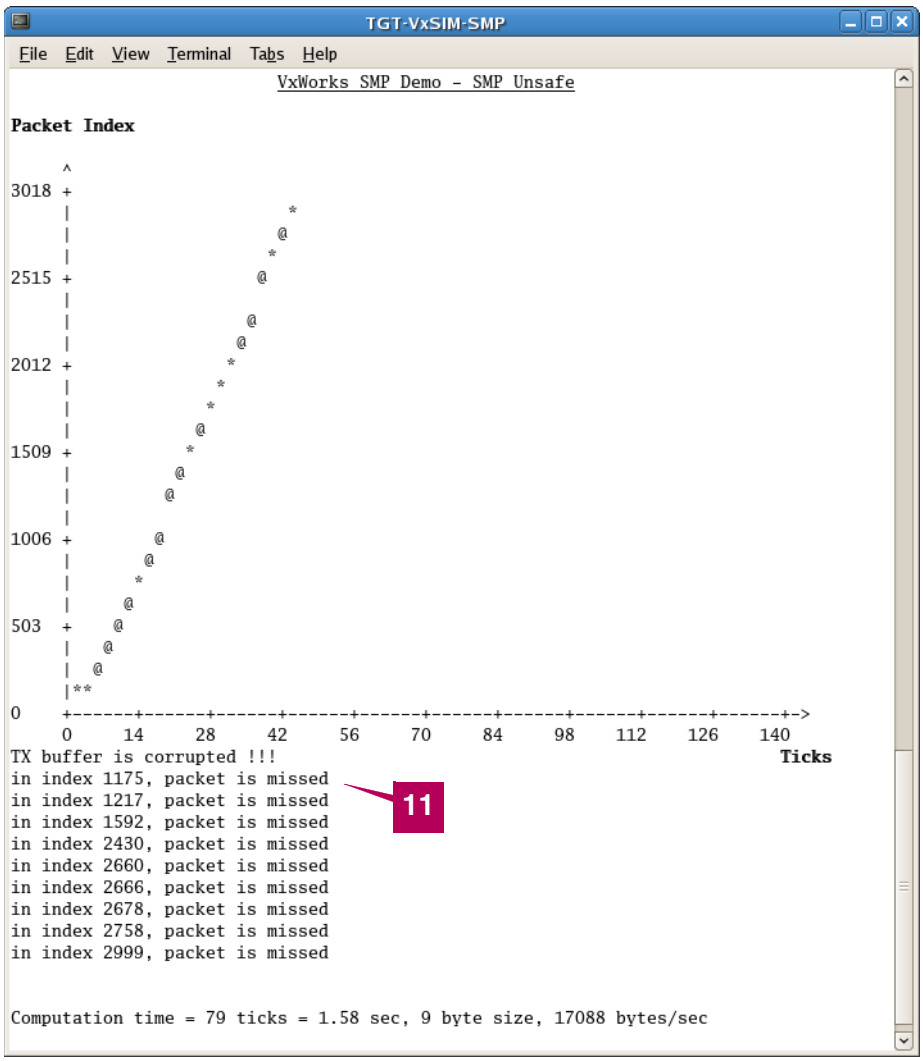


10. Start the application by clicking on the **Resume** button in the Debug view.



NOTE: This demo shows an unsynchronized scenario, in which two tasks access the same resource at the same time. As a result, the execution may occasionally fail due to a memory segmentation error. If the simulator is not responsive for more than 30 seconds, disconnect from the simulator and start again from step 1.

11. Note that the application now generates some errors in the form "**index X, packet is missed**."
12. When the application has completed, you should see the following output on the simulator.

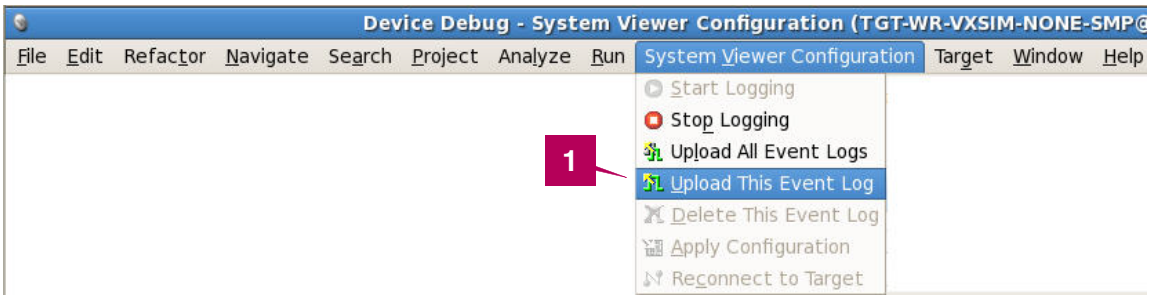


NOTE: Results may vary depending on your host CPU type and speed, number of cores, memory size, and system load.

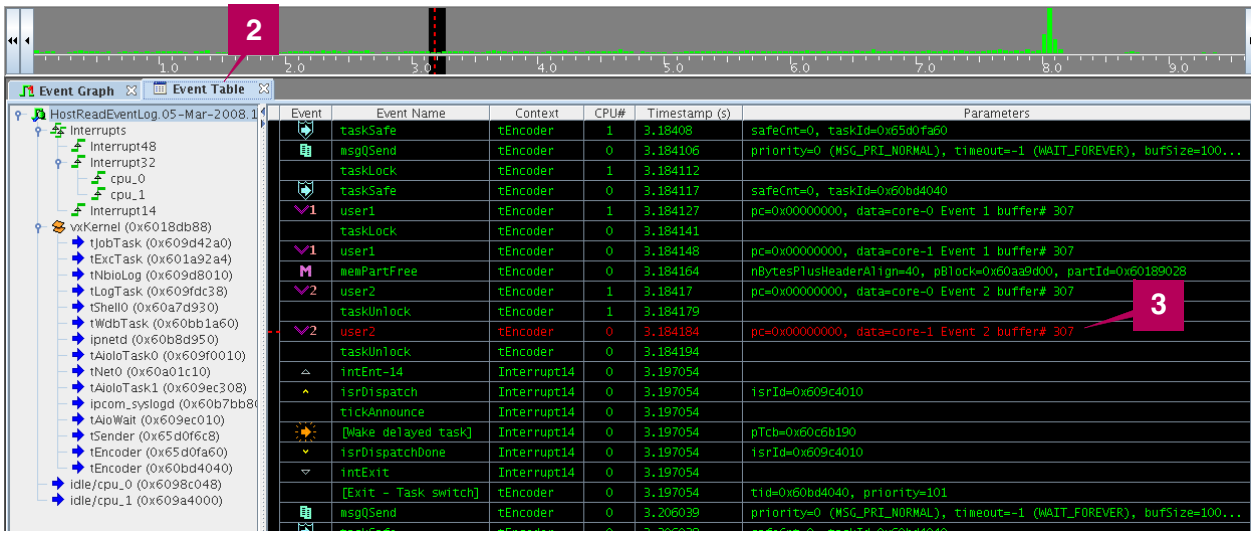
Analyzing New SMP Errors with the System Viewer

Notice that the application complains that one or some buffers are missing. In the example above, packet number 324 is missing. You may have other missing buffers in your execution.

1. Launch the the System Viewer by selecting **System Viewer Configuration** from the main menu and select **Upload This Event Log** from the sub-menu.



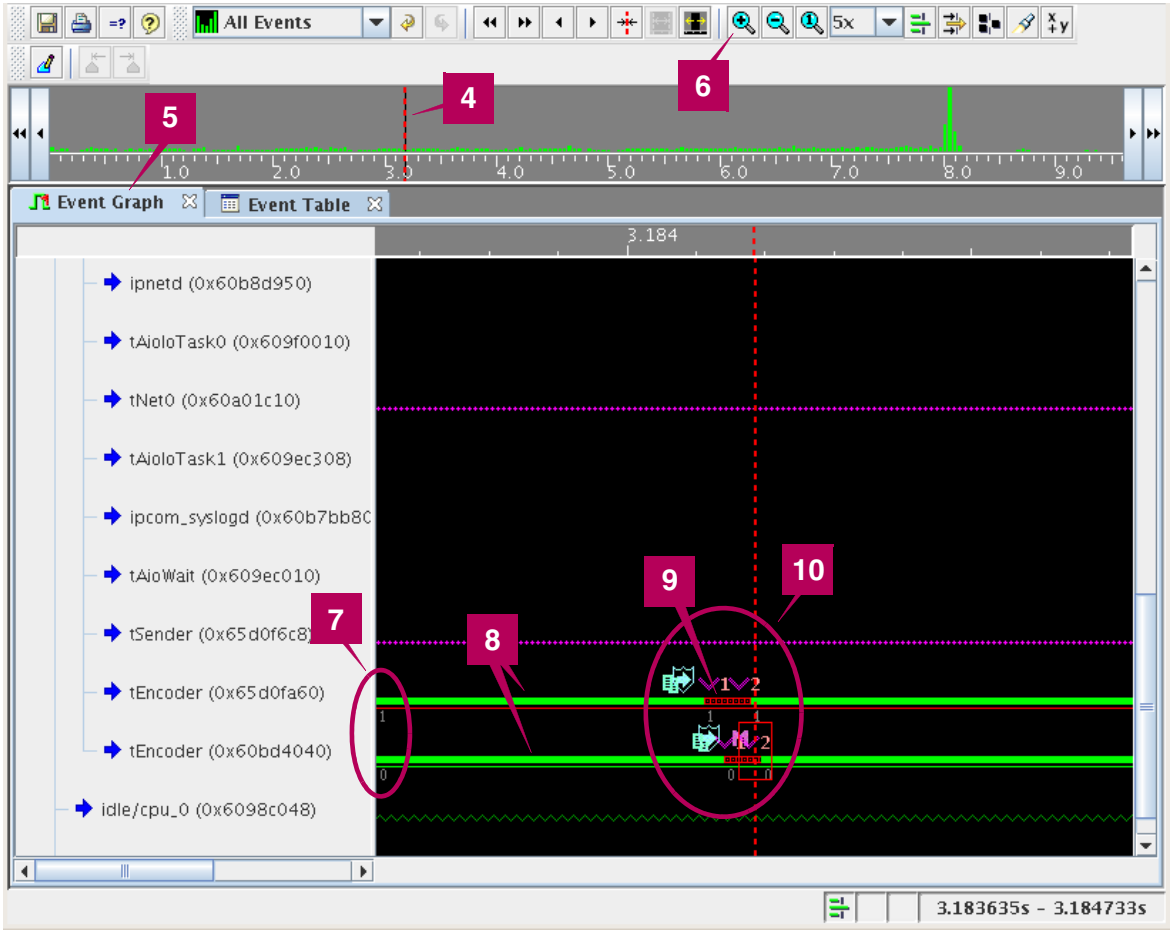
2. Click on the **Event Table** tab. This will help you to focus the view on a particular event, as there are thousands of events during the execution.
3. Browse to an event where an error had occurred. For example if an error message indicates that buffer 308 is missing, look for an event with 307 in its parameters. Double-click the event.



4. Notice that a red-dotted line is shown on the System Viewer radar.
5. Click on **Event Graph**.
6. Zoom into the selected event.

Let's examine some the result and learn more about the System Viewer's SMP capabilities:

7. Zoom into the selected event. The numbers below the task execution represent the core on which the task was scheduled. In the figure below, the **tEncoder** task with the ID 0x65d0fa60 was executed on core 1, while the **tEncoder** task with the ID 0x60bd4040 was executed in core 0.
8. Every core is annotated with an under bar color drawn under the main execution line. The System Viewer can show events from 32 cores running concurrently and each core can be colored, which helps to follow the task execution in each core. Thus, you can tell at any point in time which core executed which task.
9. The critical section in this application is colored in red, because of the task state change from **running** to **running+locked** due to the **taskCpuLock( )** call.
10. If you look carefully, you can notice that two cores execute the critical section at the same time. *Only one task can by running its critical section at any time* (unless a count semaphore is used). Thus, the data is corrupted because two tasks access the same data at the same time.



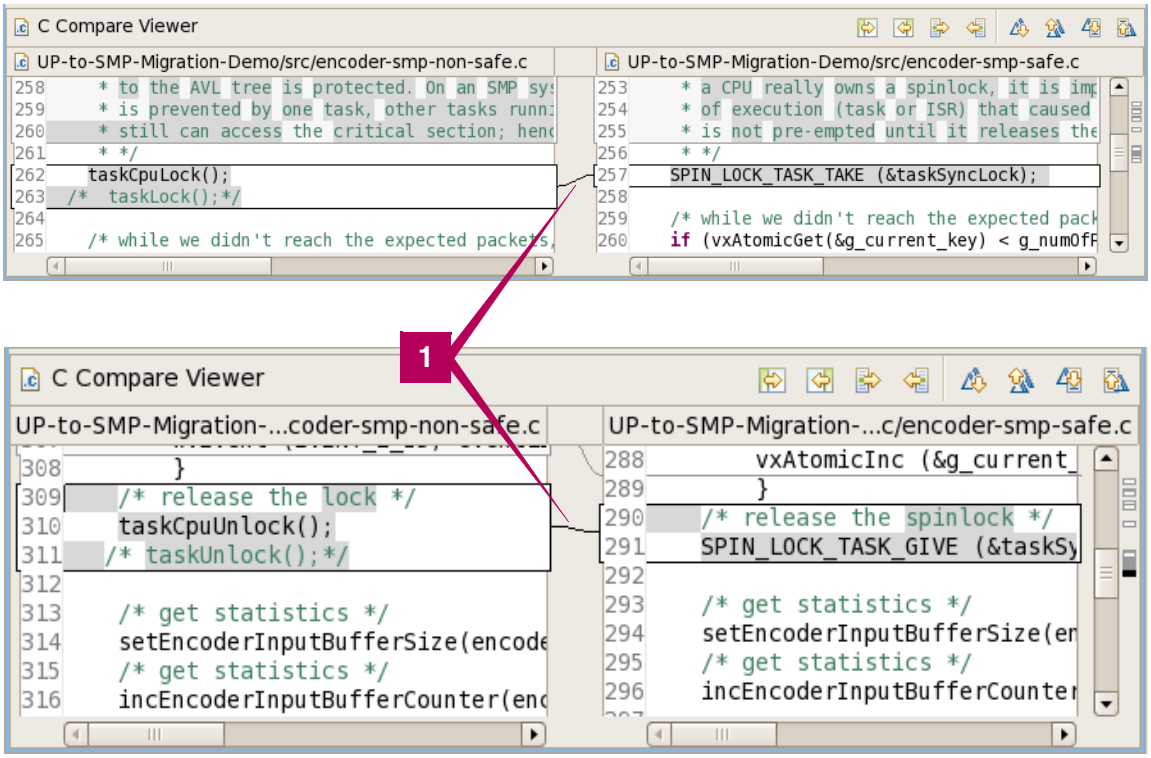
Understanding the Problem and its Solution

The problem is that **taskCpuLock( )** only disables preemption for the core on which it is executed, and not on all cores. Thus, when one **tEncoder** task accesses the critical section, the other encoder task can be scheduled on a different core even though it called **taskCpuLock( )**, because **taskCpuLock( )** only applies to the core on which it is running.

If both tasks access the same data from the receiver task, the data will in all likelihood be corrupted, which is what we're seeing here. This is a common problem in developing systems with concurrency.

To fix the problem, we'll use a spinlock instead of **taskCpuLock( )**. Spinlocks provide a facility for short-term mutual exclusion and synchronization in SMP systems. A task running on any one of the cores that acquires a spinlock gains exclusive access to the critical section it protects. When the task is done with the shared resource, it releases the spinlock. In the meantime, any other running tasks (on other cores) trying to access their critical section will wait to acquire the spinlock, and spin idly while checking for spinlock to become available.

1. You can see the code changes in the following figure; **taskCpuLock( )** is replaced with **SPIN_LOCK_TASK_TAKE( )**, and **taskCpuUnlock( )** is replaced with **SPIN_LOCK_TASK_GIVE( )**.

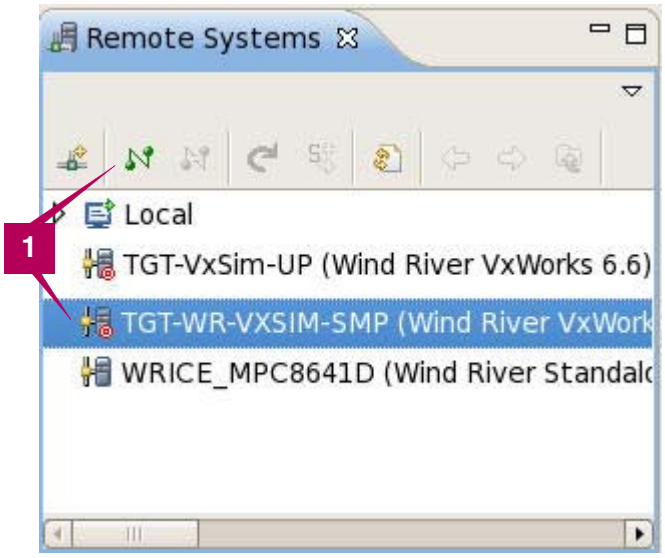


5.6.3 Running the Application in SMP Mode Using Spinlocks

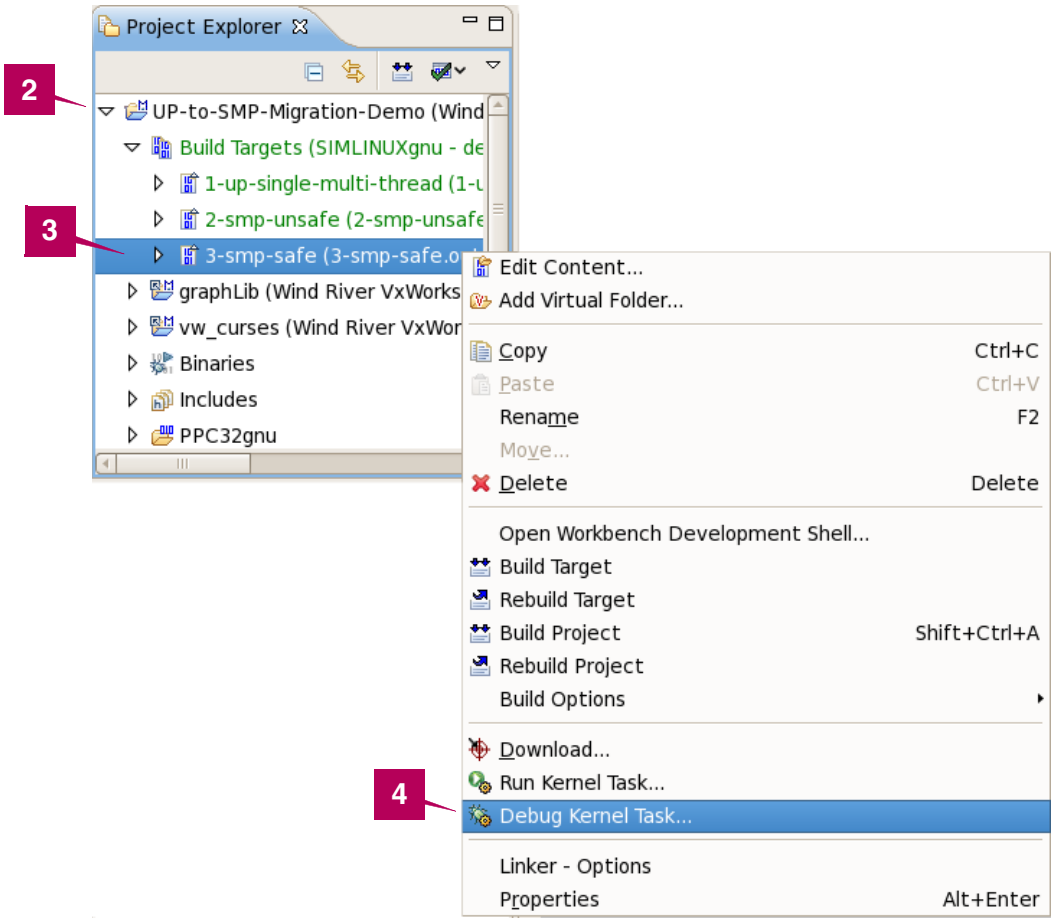
To save time, we have supplied a premodified version of the demo application to implement the use of spinlocks to solve the problem of the data corruption you saw in the previous step due to the use of **taskCpuLock( )** and **taskCpuUnlock( )**. To run the new code that uses spinlocks, follow the steps below:

NOTE: Make sure the previous instance of the SMP VxWorks simulator is closed! To close it, simply left-click on the **TGT-WR-VXSIM-SMP** entry in the Remote Systems view to select it, then click the Disconnect button to close it.

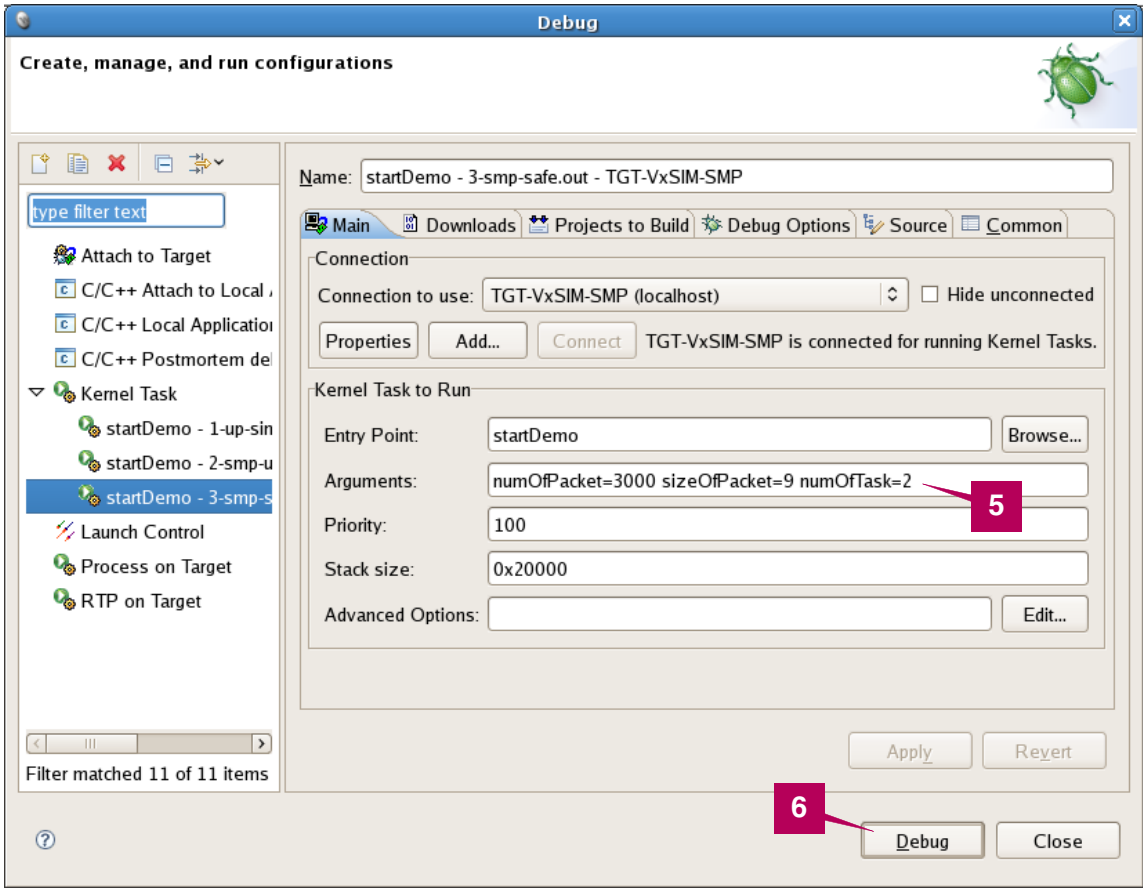
1. If the SMP simulator is not already running, launch it by selecting **TGT-WR-VXSIM-SMP** in the Remote Systems view and clicking on the **Connect** button.



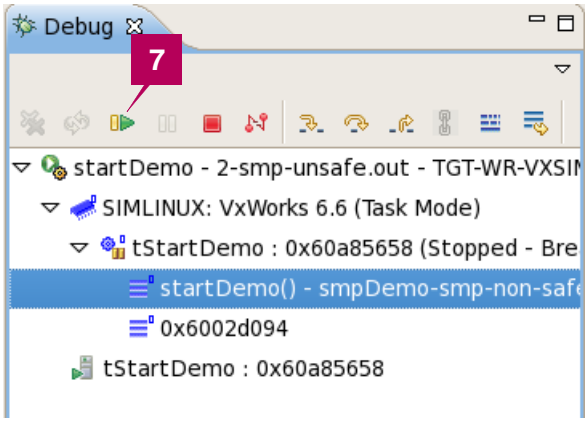
2. In the Project Explorer view, expand the **UP-To-SMP-Migration-Demo** project and the **Build Targets** item within.
3. Right-click on the third project, **3-smp-safe**.
4. From the context menu, select **Debug Kernel Task**.



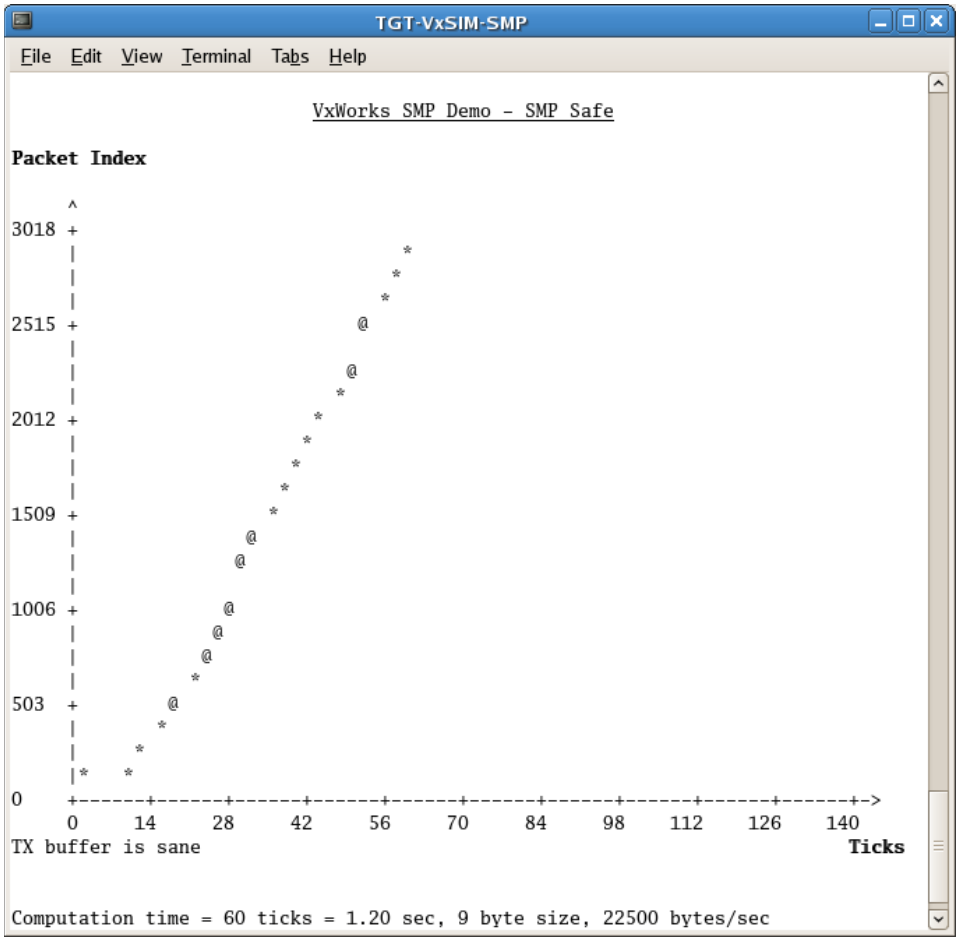
5. In the Run dialog, verify that **Arguments** is set to **numOfPacket=3000 sizeOfPacket=9 numOfTask=2**.
6. Select **Debug**. This will launch the application (we don't need any debugging facilities this time).



7. Start the application by clicking the **Resume** button in the Debug view.



8. When the application has completed, you should see the following output on the simulator.
9. Observe that there is no longer an error message.



NOTE: Results may vary depending on your host CPU type and speed, number of cores, memory size, and system load.

6 SMP Debugger

6.1 Debugger Overview

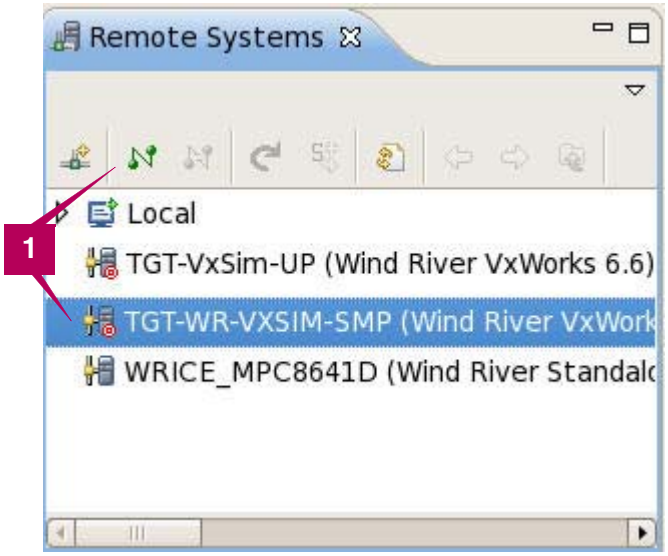
The Wind River debugger supports SMP. The challenge with SMP debugging is that a task can migrate from one core to another at any point in time. Additionally, to exploit the real power of SMP, multiple tasks (or threads) can be launched simultaneously. Each task holds it own stack, heap, and internal variables. In this section you will experience the SMP debugger and see some of it capabilities.

6.2 Starting the Debugger

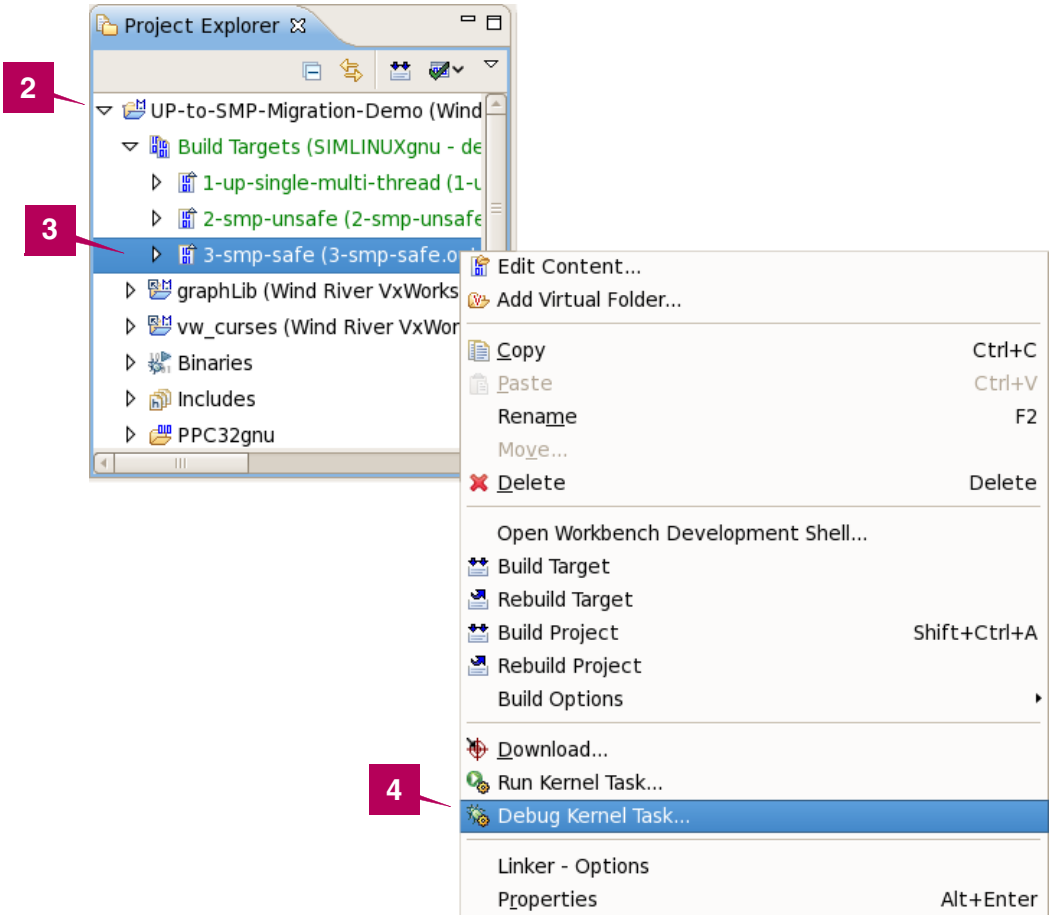
To start the debugger, download the **smp-safe** application to the SMP-enabled simulator.

NOTE: Make sure the previous instance of the SMP VxWorks simulator is closed! To close it, simply left-click on the **TGT-WR-VXSIM-SMP** entry in the Remote Systems view to select it, then click the Disconnect button to close it.

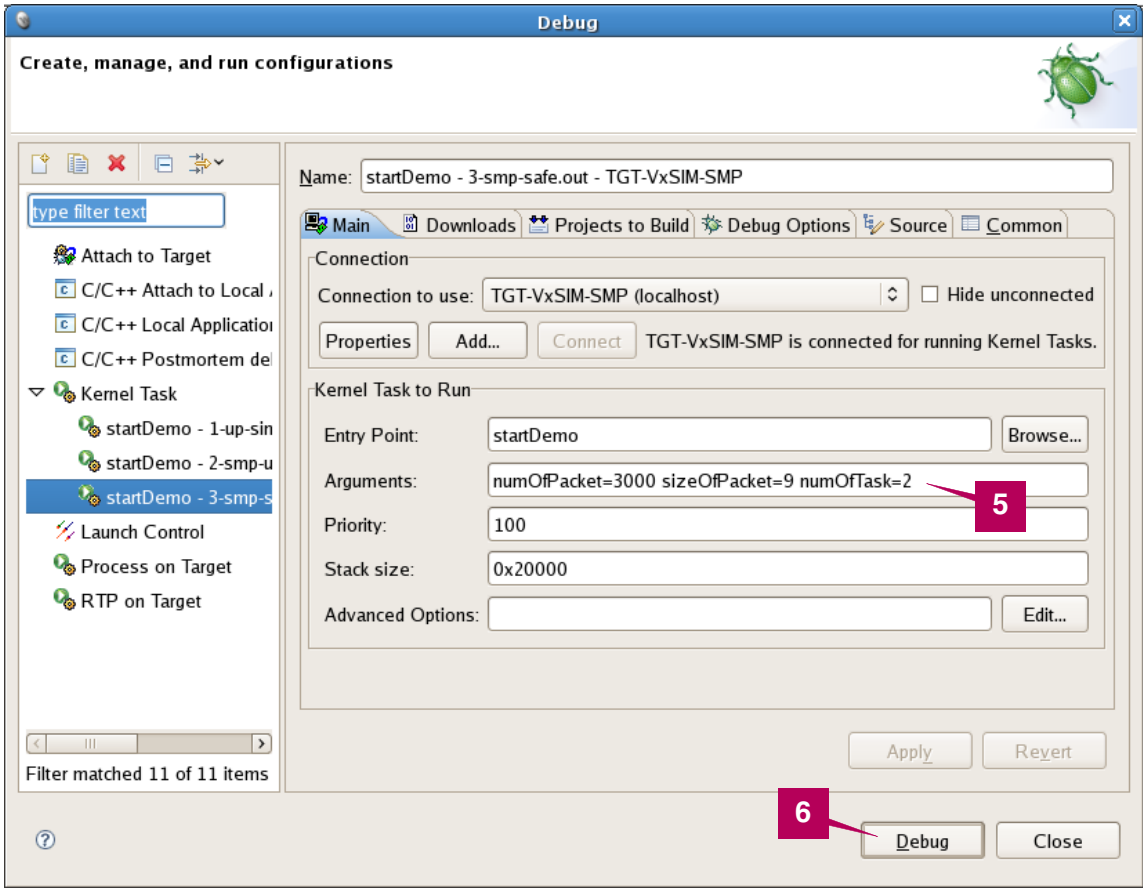
1. If the SMP simulator is not already running, launch it by selecting **TGT-WR-VXSIM-SMP** in the Remote Systems view and clicking on the **Connect** button.



2. From the Project Explorer view, expand the **UP-To-SMP-Migration-Demo** project and the Build Targets item within.
3. Right-click on the third project, **3-smp-safe**.
4. From the context menu, select **Debug kernel Task**.



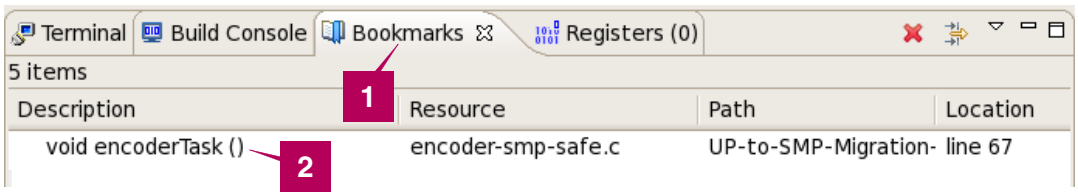
5. In the Debug dialog, verify that **Arguments** is set to **numOfPacket=3000 sizeOfPacket=9 numOfTask=2**.
6. Select **Debug**. This will launch the application, and you should see the source file in Workbench editor, and that the application is stopped in the Debug view.



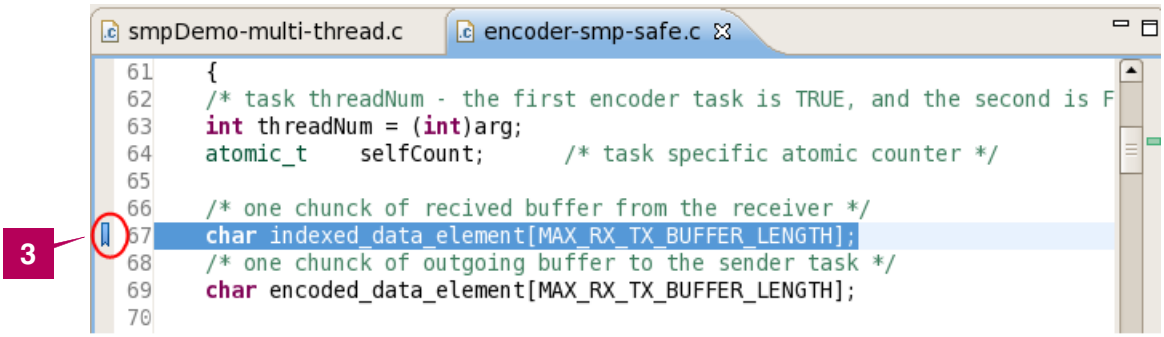
6.3 Installing Breakpoints

The task of interest in this application is the encoder task, so we'll set a breakpoint at the beginning of that task.

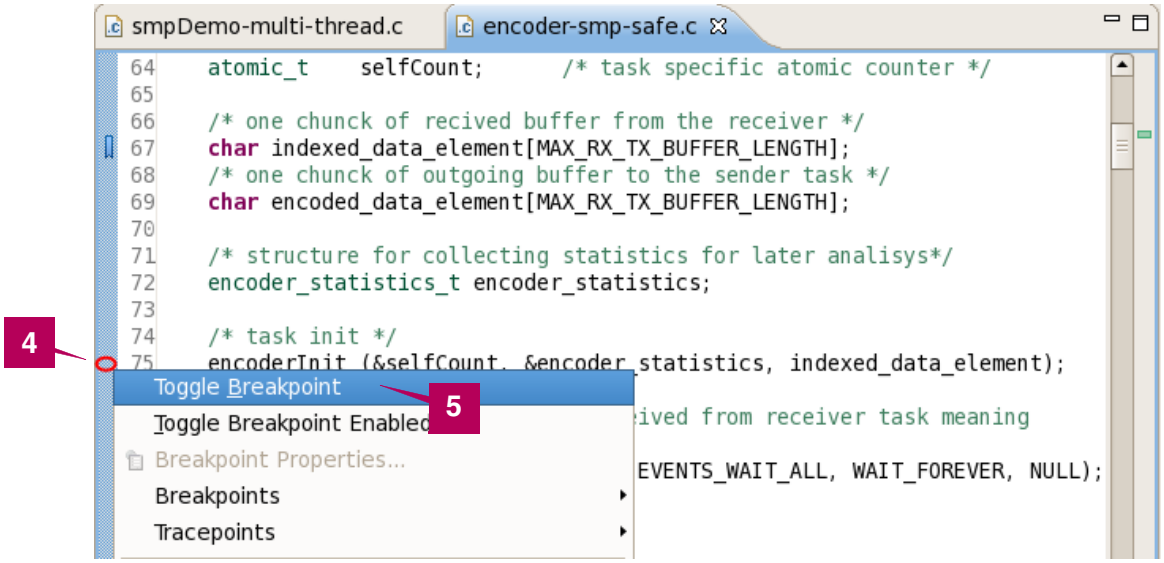
1. Locate and click on the **Bookmarks** tab; by default it's located in the view near the bottom of the workspace.
2. Double-click on **encoderTask( )**, which is a predefined bookmark



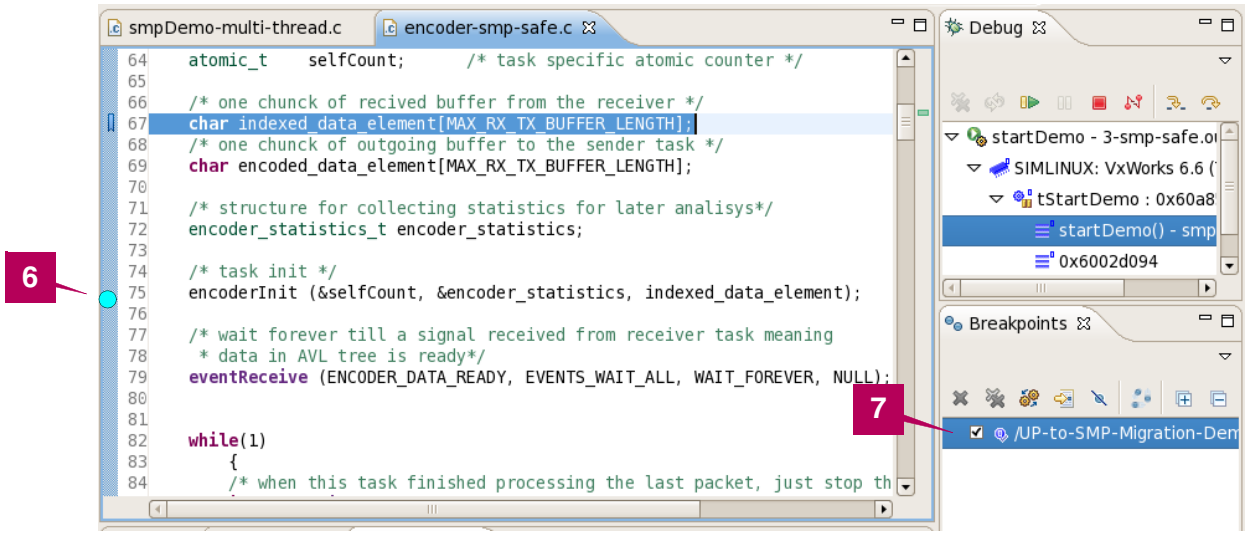
3. The **encoder-smp-safe.c** source file will open in the Editor; you can see the bookmark sign beside line 67.



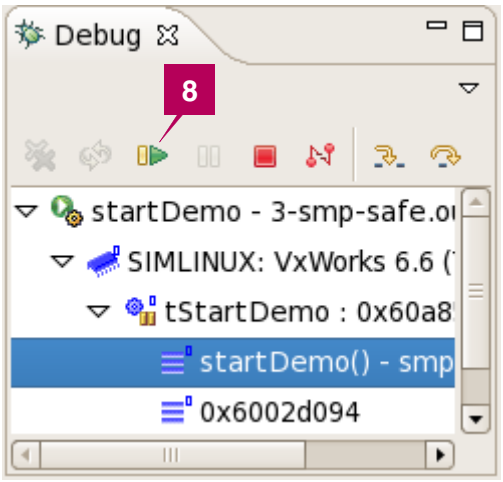
4. To set a breakpoint, right-click in the left gutter beside line 75.
5. From the context menu, select **Toggle Breakpoint**.



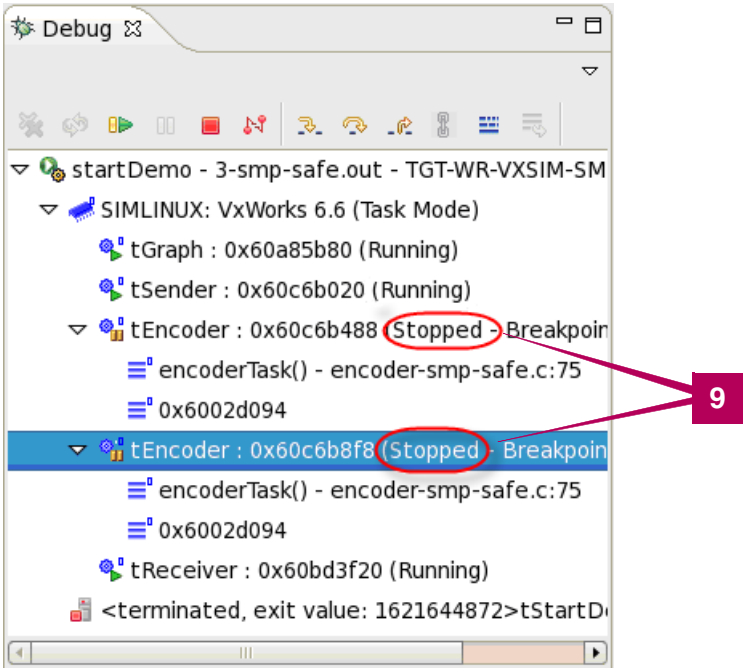
6. A new breakpoint icon is drawn on line 75.
7. A breakpoint entry is also added to the Breakpoints view.



8. Run the application by clicking the **Resume** button in the Debug view.



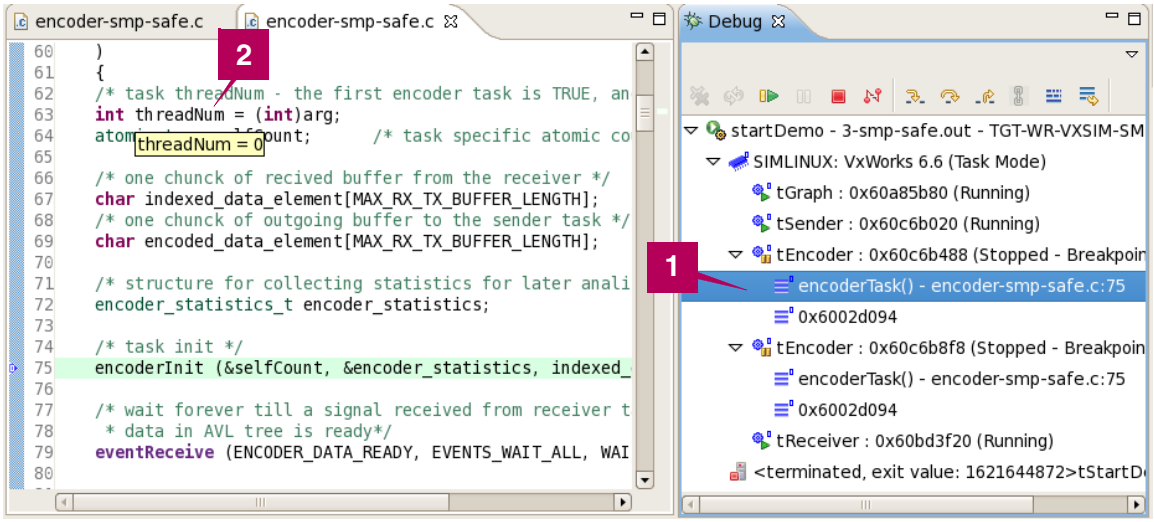
9. The application begins to execute and stops when the breakpoint is hit. You can tell that the encoder tasks hit the breakpoint from the task status in the Debug view. Since two identical **tEncoder** tasks execute the same code, both tasks hit the breakpoint.



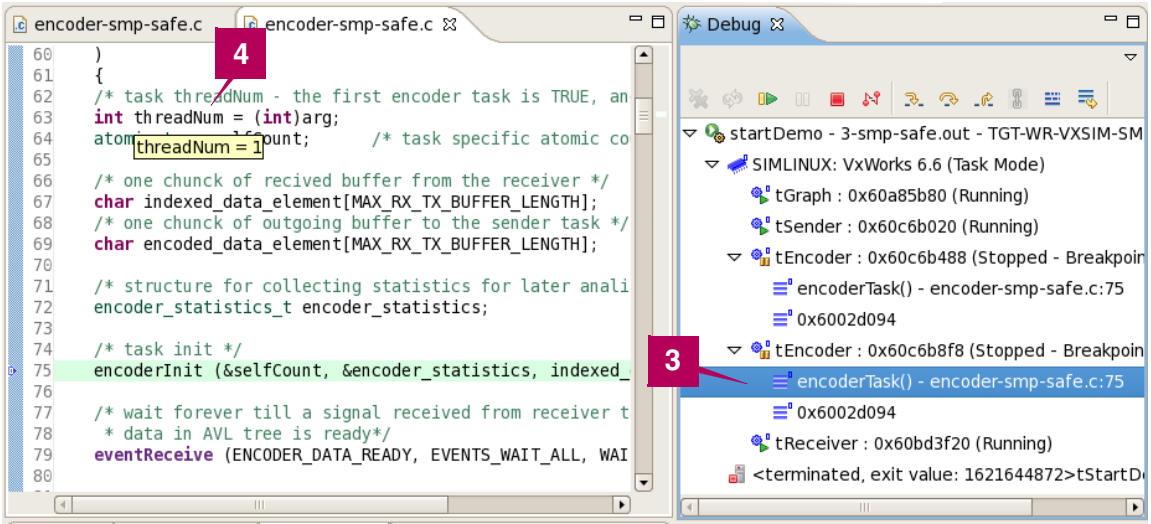
6.4 SMP Context Debugging

The VxWorks SMP debugger uses a context debugger methodology in which the context of the debugged task is related to the current highlighted task. With context debugging you can install breakpoints in any line of code, and if the line of code is executed on multiple cores, you can debug each instance of the execution on each core. This section shows how to debug two instances of the Encoder task that are executed on two cores simultaneously.

1. Select the first encoder task in the Debug view.
2. Drag the cursor over line **63: int threadNum = (int) arg**; Note the value of **threadNum** is **0**.



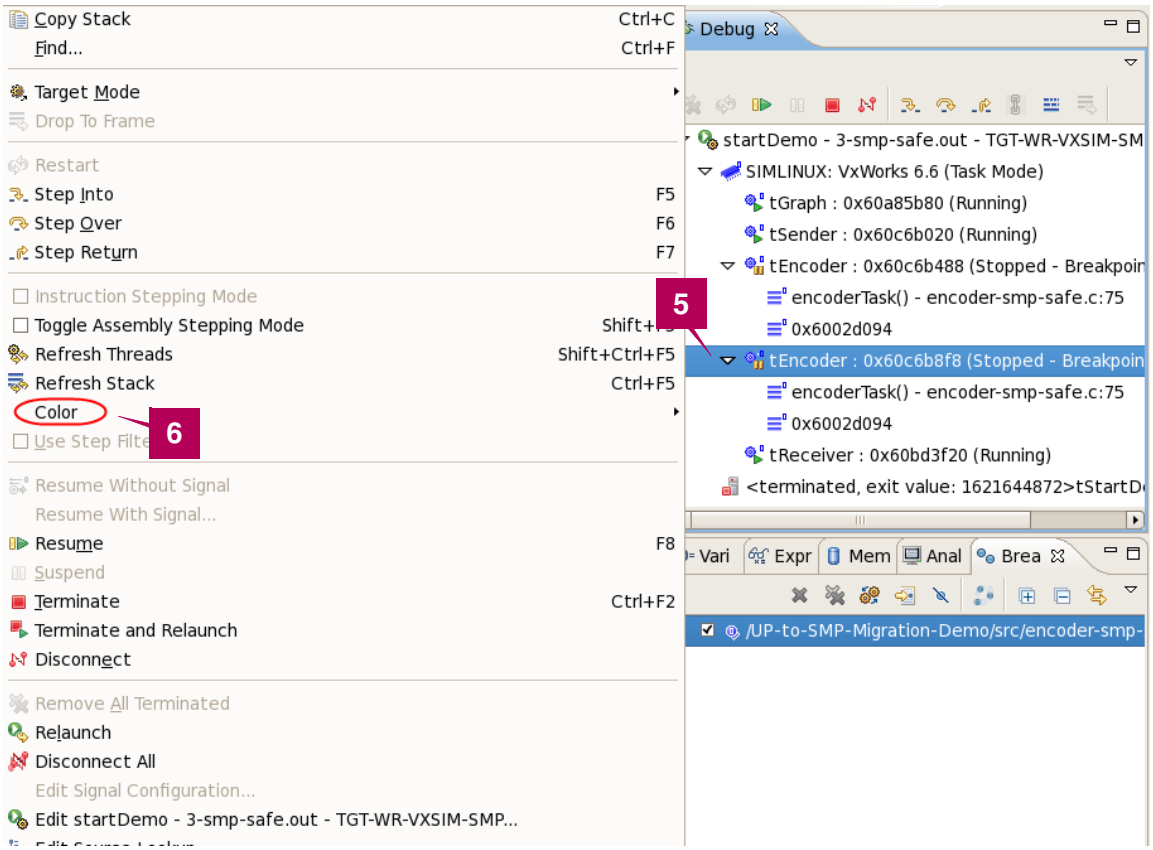
3. Select the second **tEncoder** task in the Debug view.
4. Drag the cursor over line 63 again. Note now that the value of **threadNum** is **1**, reflecting a different state for a different task.



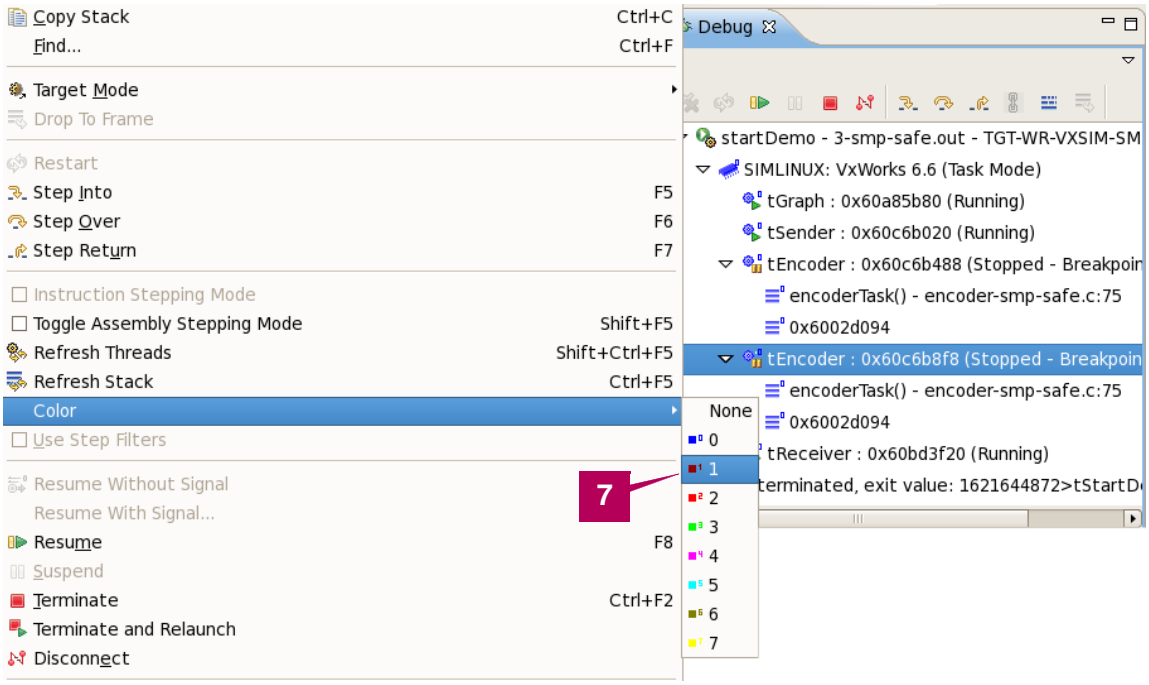
Since there are multiple tasks running on different cores, to identify each line of execution, each task can be colored differently. For example, both of the **tEncoder** tasks are currently colored blue.

To color the second encoder task red, follow these steps:

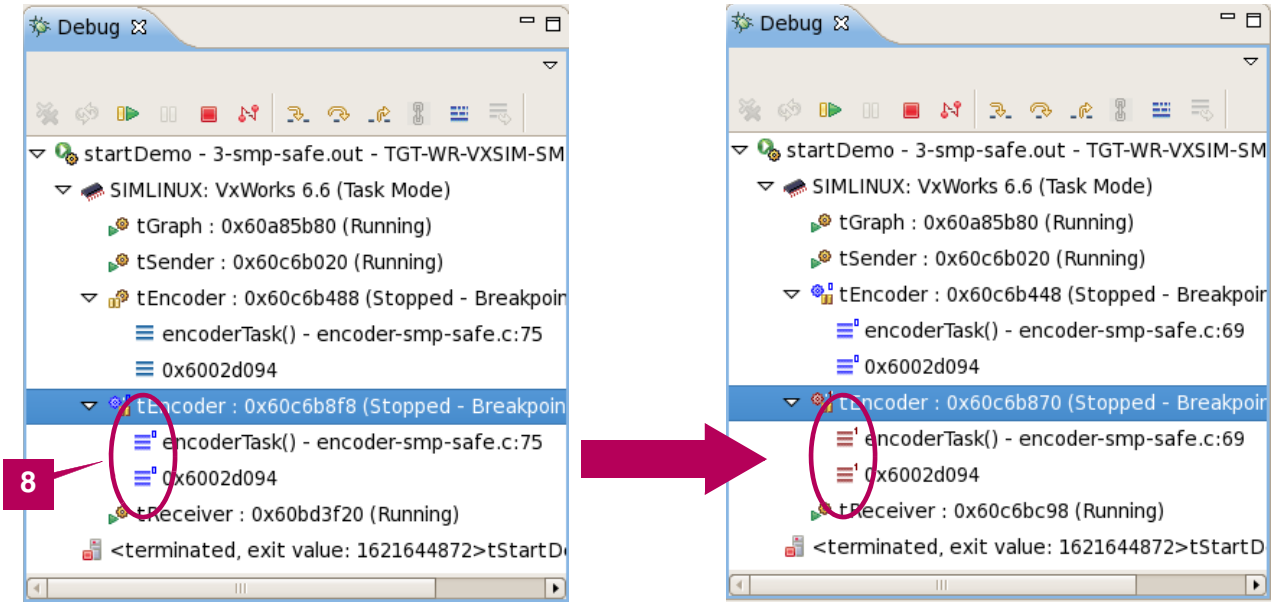
5. Right-click on the second **tEncoder** task in the Debug view.
6. From the context menu, select **Color**.



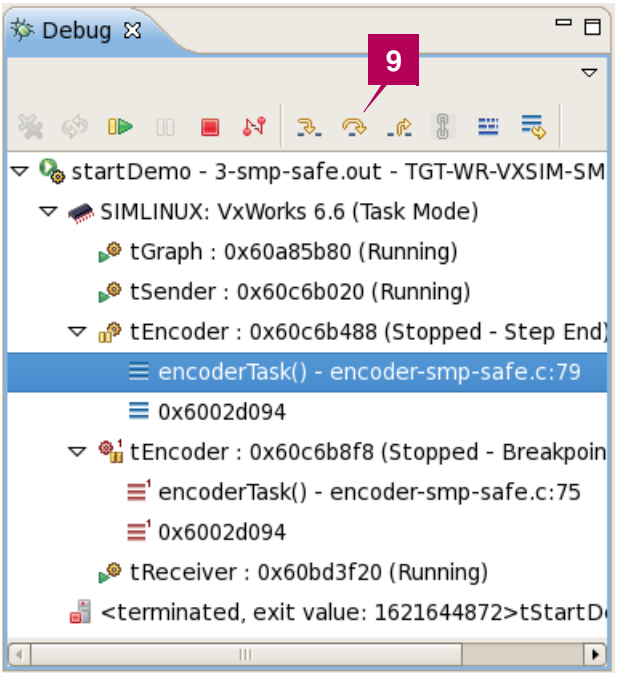
7. The following color submenu will appear. Select the red color, marked number **1**.



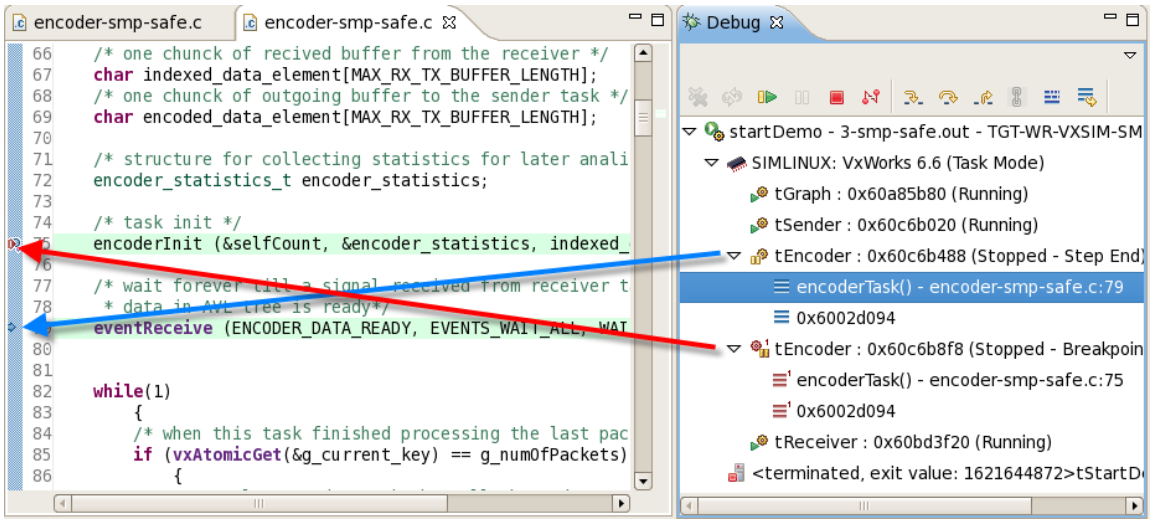
8. The color of the second **tEncoder** task in the Debug view will change from blue to red.



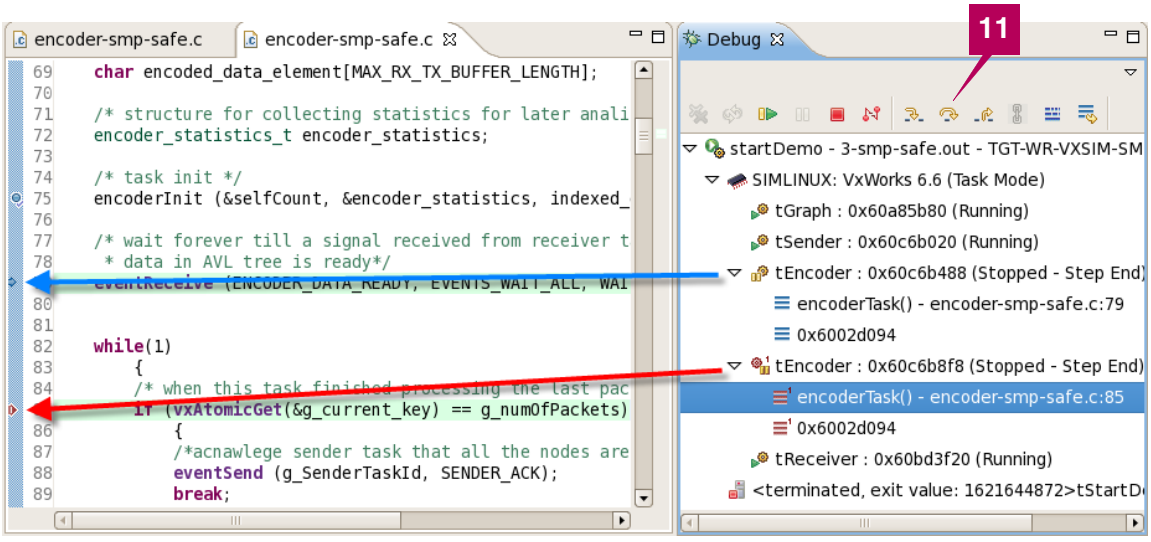
9. Highlight the first **tEncoder** task (the blue one) in the Debug view (colored in blue), and click on the **Step Over** button.



10. Notice that two lines are highlighted in the Editor, accompanied by colored arrows in the left gutter. These arrows correspond to the program counter for each task. The blue arrow tells you where the first **tEncoder** task is in its execution, while the red arrow tells you where the second **tEncoder** task is in its execution.



11. Now select the red task in the Debug view and click on the **Step Over** button twice.

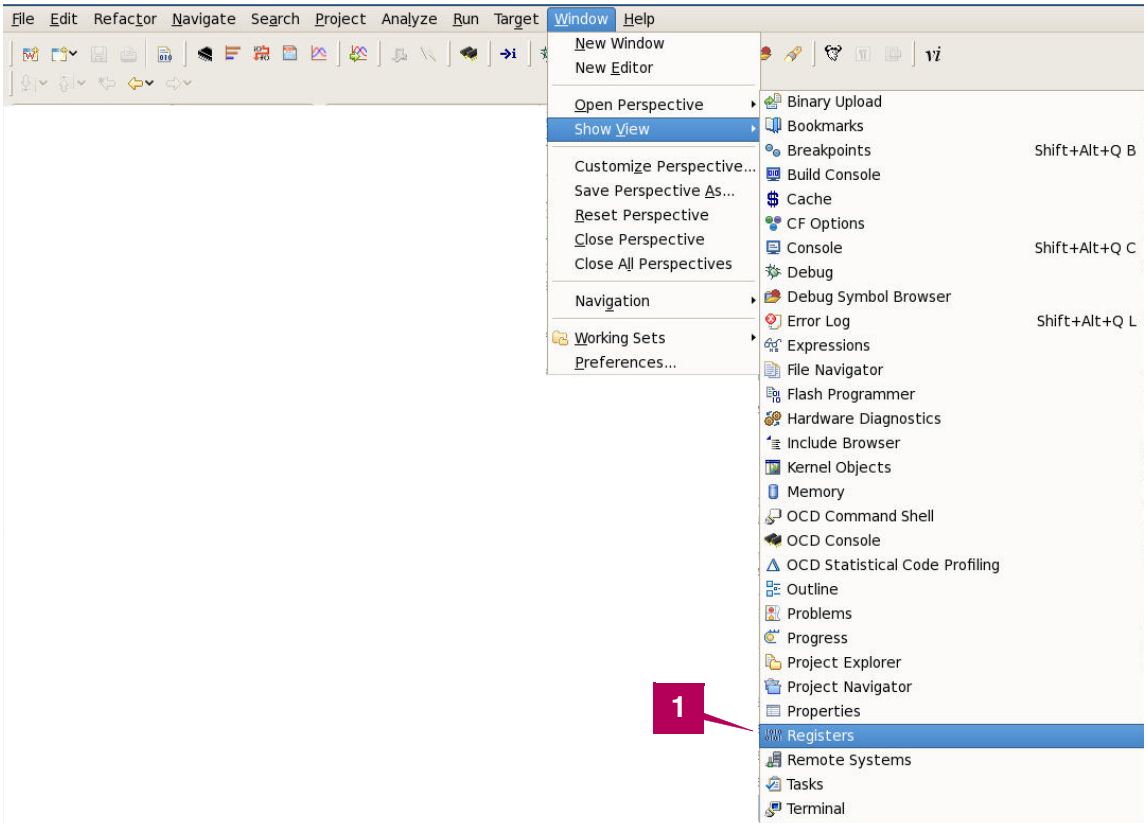


6.5 Per-Task Registers Window

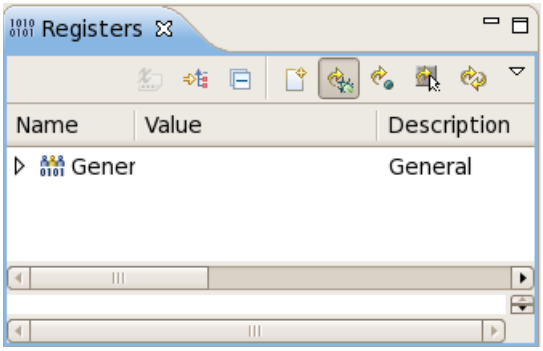
In SMP architecture, tasks can be executed on any core at any given time. Thus tasks can migrate from one core to another. This feature enables true parallelism but also introduces debug challenges. In some cases, you may need to know the values of internal hardware registers for the task you are currently monitoring in the debugger. Workbench can display the internal registers of the core running each task. If the task has moved to another core, the display will show you the internal register from the new core.

Continuing from the previous section, where the two encoder tasks are stopped on a breakpoint, add two more register views:

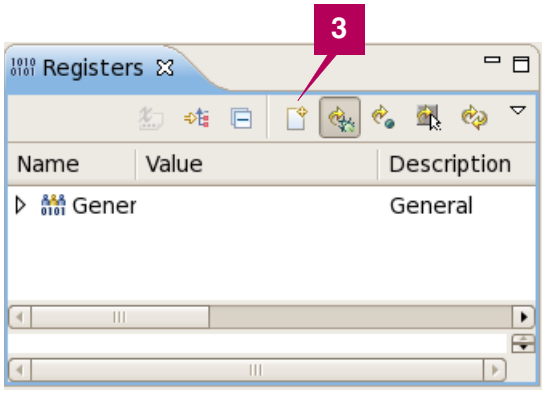
1. From the main menu, click on **Window > Show Window > Registers**.



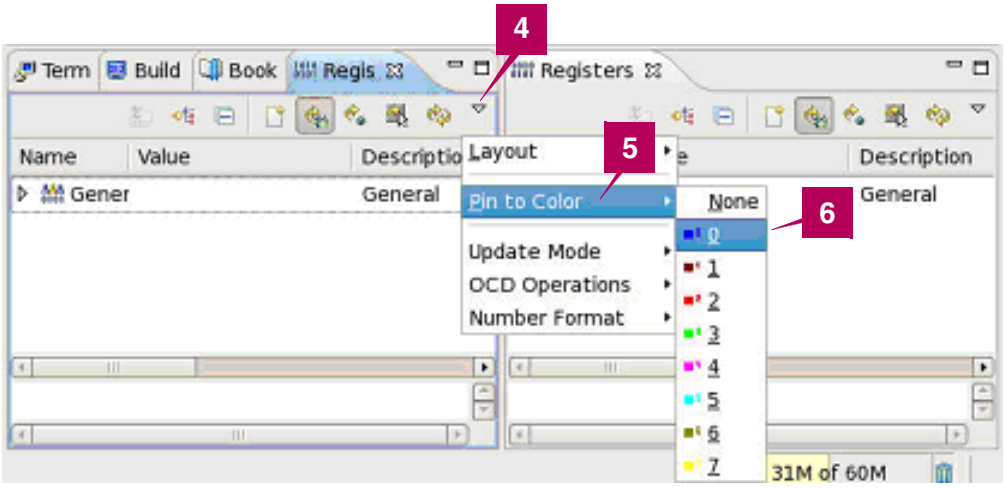
2. A Registers view opens:



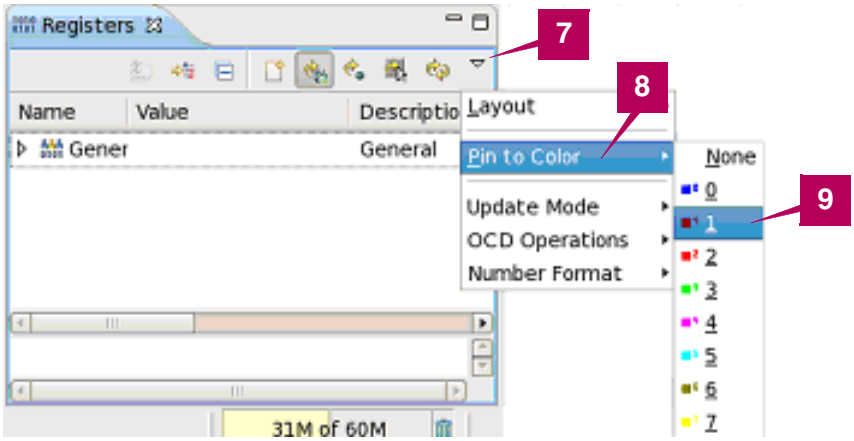
3. If you had two Registers views, you would be able to see the values for each core concurrently. To open another Registers view, click the **New Register View** button.



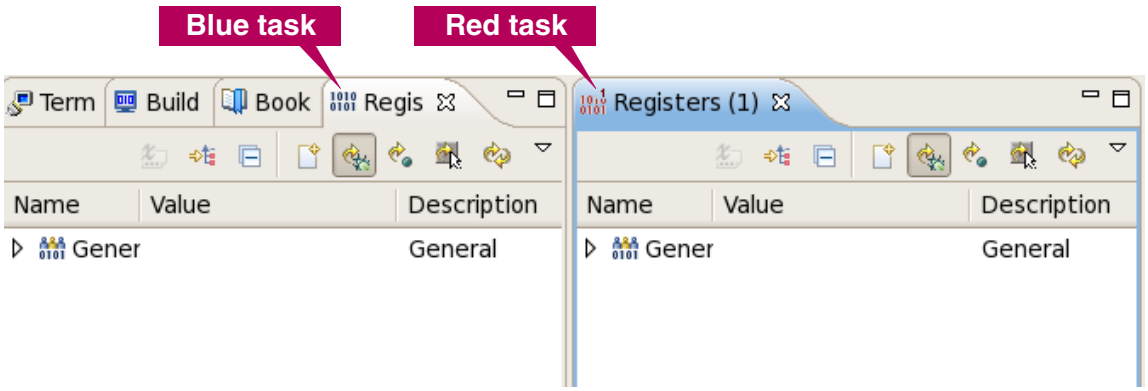
4. Now you can link each Registers view to a task. Click on the triangle on the right side of the Registers view.
5. Select **Pin To Color** from the drop-down menu.
6. From the sub-menu, select **0** (blue). This will link this Registers view to the blue **tEncoder** task.



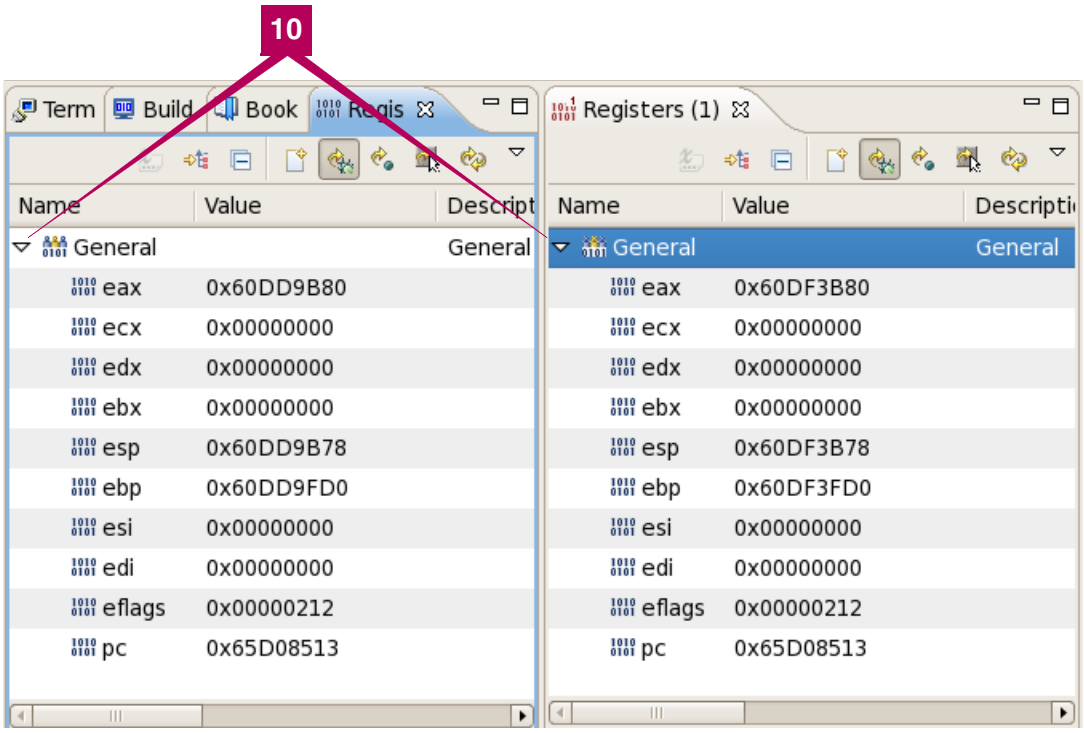
7. To link the other Registers view to the red **tEncoder** task, again click on the triangle ▽ on the right side of the Registers view.
8. Select **Pin To Color** from the drop-down menu.
9. From the sub-menu, select **1** (red). This will link this Registers view to the red **tEncoder** task.



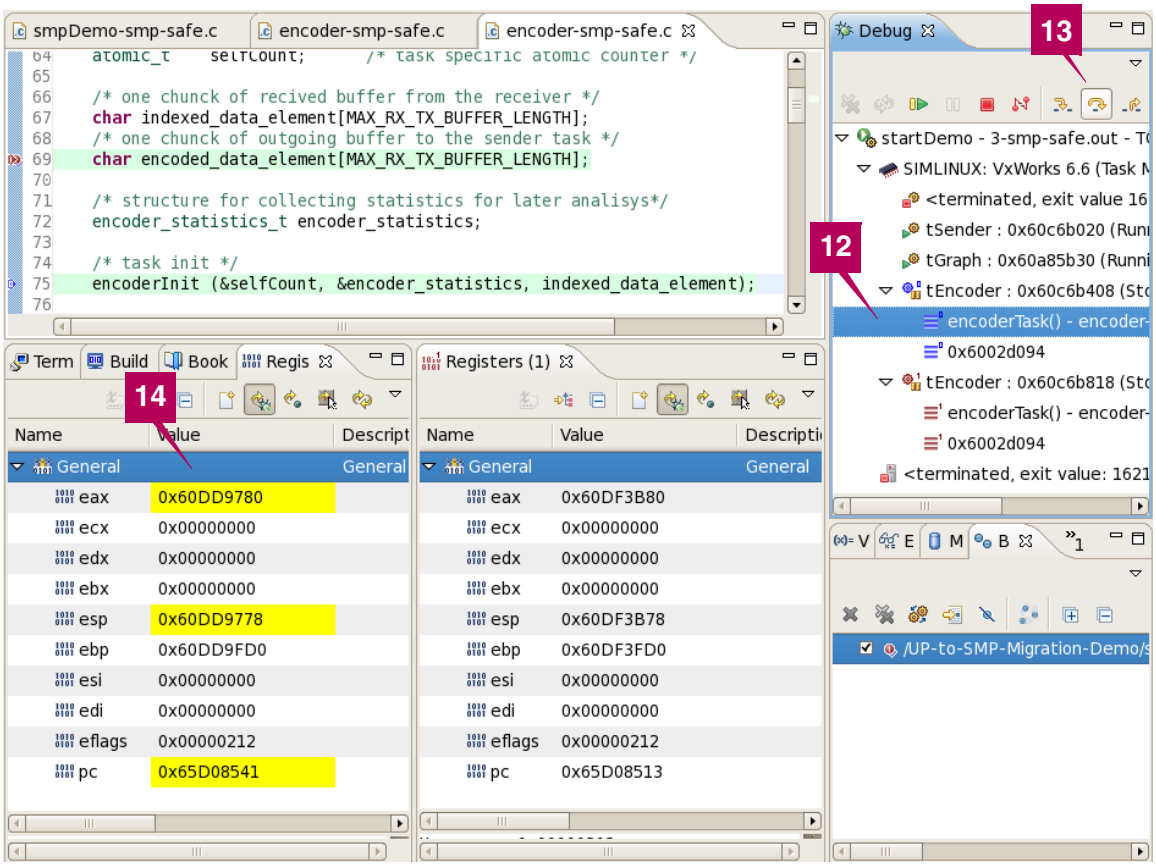
10. Notice that the Registers window header color has changes accordingly.



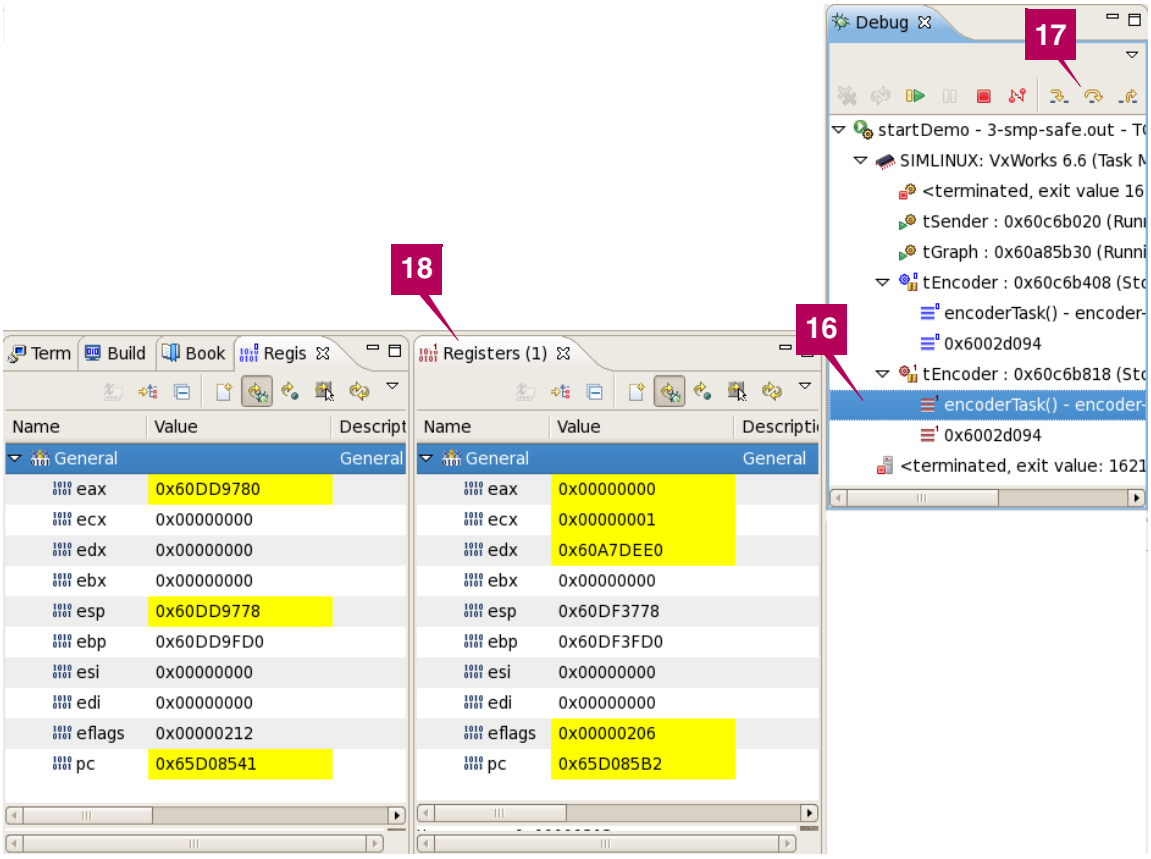
11. Collapse the registers by clicking on the triangle beside the **General** string



12. Highlight the blue **tEncoder** task from the Debug view.
13. Click on **Step Over** button.
14. The internal registers of the core running the blue task are shown in the blue task's Registers view.



15. If the task moves to a different core and hits a breakpoint, the register values for the core running the task will display in the proper Registers view. This is a useful feature that was designed to help you debug in a multicore environment.
16. Repeat the same steps for the red **tEncoder** task: select the red **tEncoder** task in the Debug view.
17. Click the **Step Over** button.
18. The internal registers of the core running the red task are shown in the red task's Registers view.



19. Finally, if the simulator is running, disconnect from it by left-clicking on the **TGT-WR-VXSIM** entry in the Remote System view to select it, then clicking on the **Disconnect** button from the Remote Systems view.

7 VxWorks SMP Evaluation Summary

Migrating an application from UP to SMP involves several steps:

1. Identifying the areas in which the application spends most of the time; this is the area that should be parallelized. The Wind River Performance Profiler tool enables detailed monitoring of each task and function call of the application.
2. Identifying implicit synchronization issues. UP applications may bring implicit synchronization issues; for example, priority-based synchronization where high-priority tasks always preempt lower-priority tasks. Remember that in SMP system, X tasks can run concurrently, where X represents the number of cores.
3. Executing the new designed SMP application on UP, where only one core is enabled. Notice that in most cases parallelism means more shared data between tasks and thus more data synchronization and task communication is required.
4. Execute the SMP application on an SMP system where some or all of the available cores are enabled. The Wind River System Viewer helps you visualize the execution. The System Viewer supports multicore and provides detailed event information per core.
5. When needed, Wind River's debugger supports SMP. Context debugging shows task variables, internal registers, and more, on per task basis.

This concludes the Wind River VxWorks SMP evaluation. During this step-by-step evaluation, we hope you had the chance to have a first-hand experience with the migration steps required to migrate an existing UP application to multicore. During this step-by-step evaluation, we showed you in short how Wind River VxWorks SMP and Wind River tools can help you to:

- Isolate areas where parallelism is required.
- Enable SMP debugging and monitoring with the VxWorks SMP tools. All the tools are integrated in Workbench and function the same for both UP and SMP.
- Troubleshoot problems. SMP development may be challenging and exciting; however, parallelism introduces intertask synchronization and communication issues. These issues can lead to deadlocks and livelocks. The System Viewer is an execution monitor that collects data from up to 32 cores and displays the data in an event list and graph. This tool can help you understand and visualize the execution of your code.

- Quickly debug your application code and kernel modules with the same development environment and processes for both UP and SMP. This will help you focus on SMP development without learning new tools or working in a foreign environment.
- Monitor your system execution and identify run-time problems like starvations, bottlenecks, priority inversions, and so on.

The team at Wind River hopes you are impressed with the capabilities we have highlighted in this evaluation. To get more information about other technical capabilities of VxWorks SMP, and how Wind River can help you to develop faster and better multicore device software, please contact a Wind River sales representative in your area.

For a list of Wind River contacts please use the link below:

http://www.windriver.com/company/contact/index.html