

# Lab Guide: VxWorks Introduction

## Introduction

You are now familiar with FreeRTOS, you can move to a more complete toolkit, the VxWorks RTOS. It is very powerful, but also much more expensive. For the price you get a powerful scheduler, set of inter-process communication protocols, semaphore system, and drivers. But most importantly you get a fully documented and supported system with traceability. This means that you can guarantee the real-time performance of your system.

## Constructing the Hardware

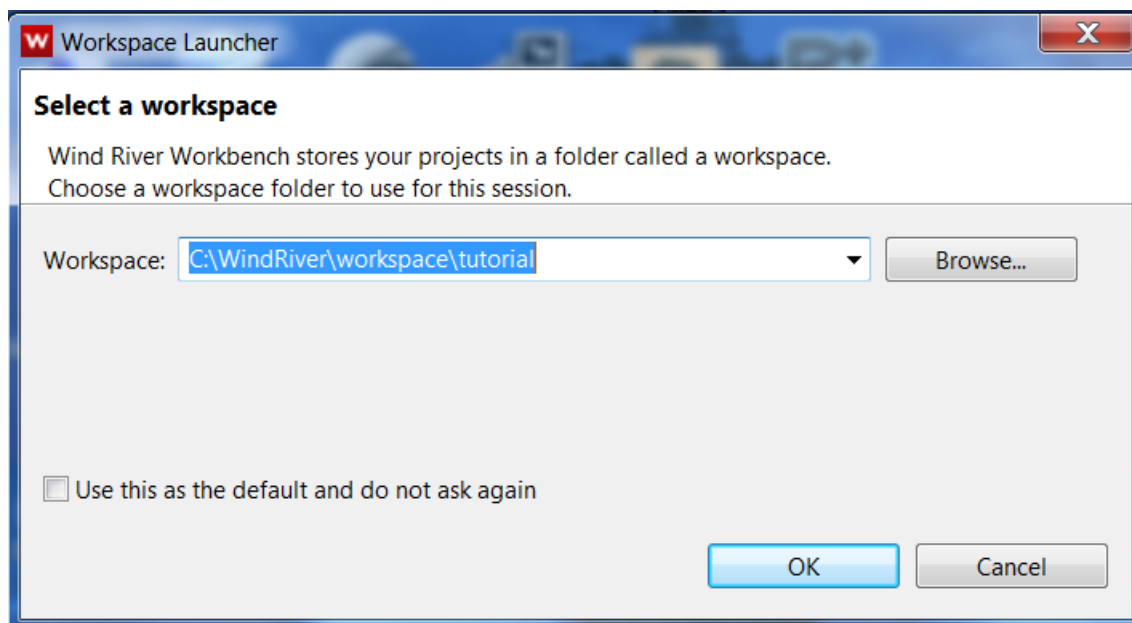
For this first VxWorks Lab there is no hardware.

## Initiating a VxWorks Session

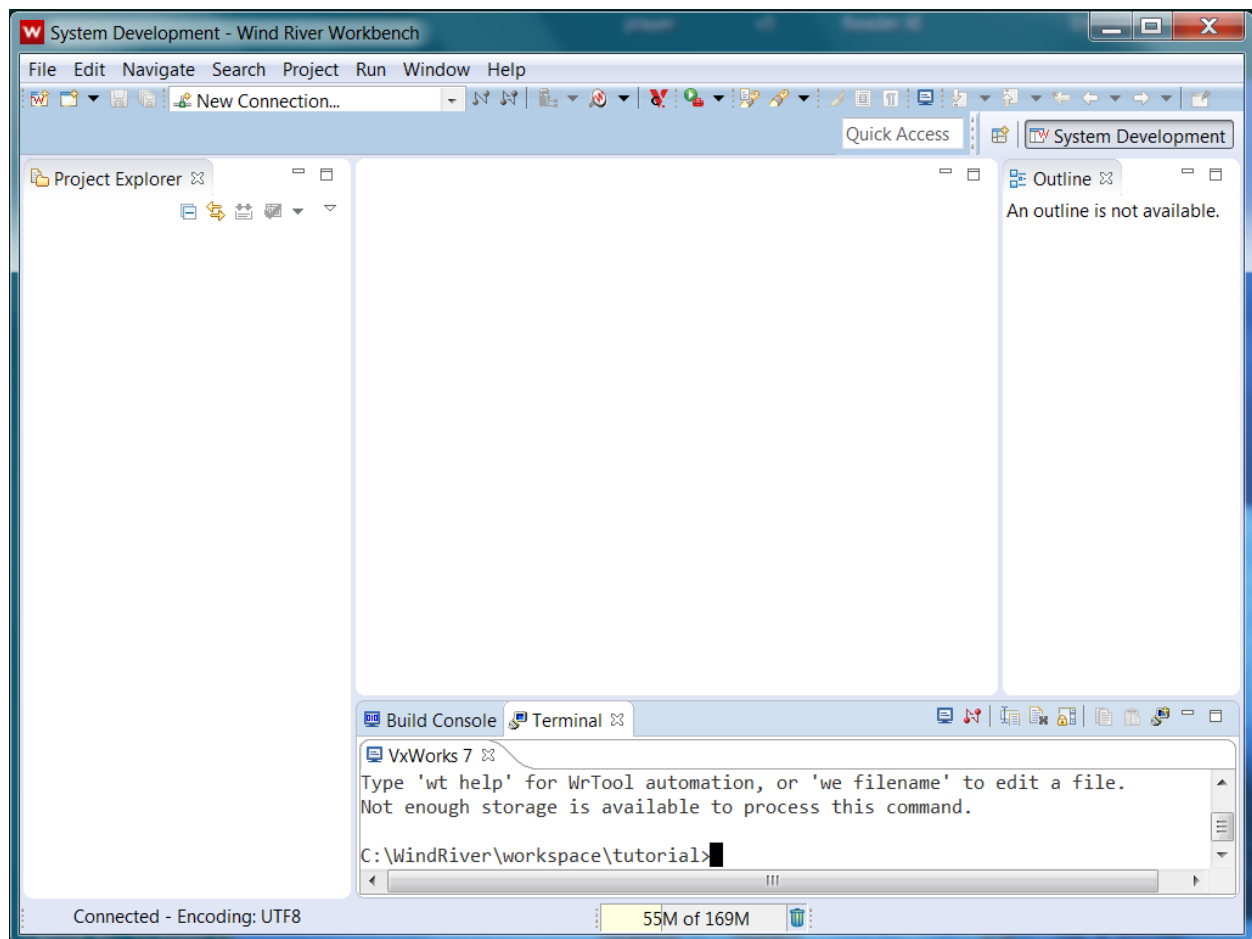
VxWorks is installed on the set of computers in the RF lab. You should look for the VxWorks Workbench icon on the desktop, it looks like this:



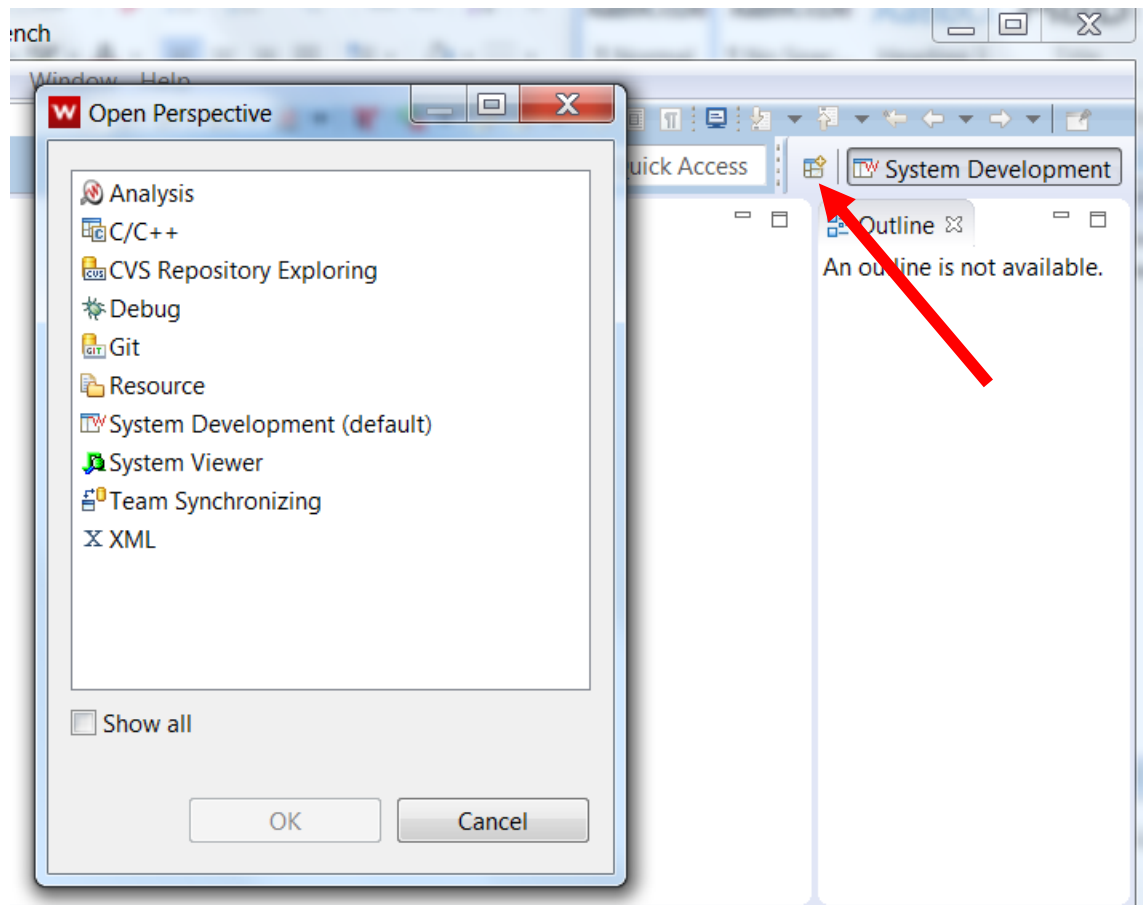
Double click on the icon and you should see this selection screen:



You should create a workspace name that is unique for you. That way you won't see others files. Click OK. Then you should see this screen (Although if this is the first time you are running VxWorks you might get an introduction screen asking if you want to see tutorials. Simply select to go to the Workbench):

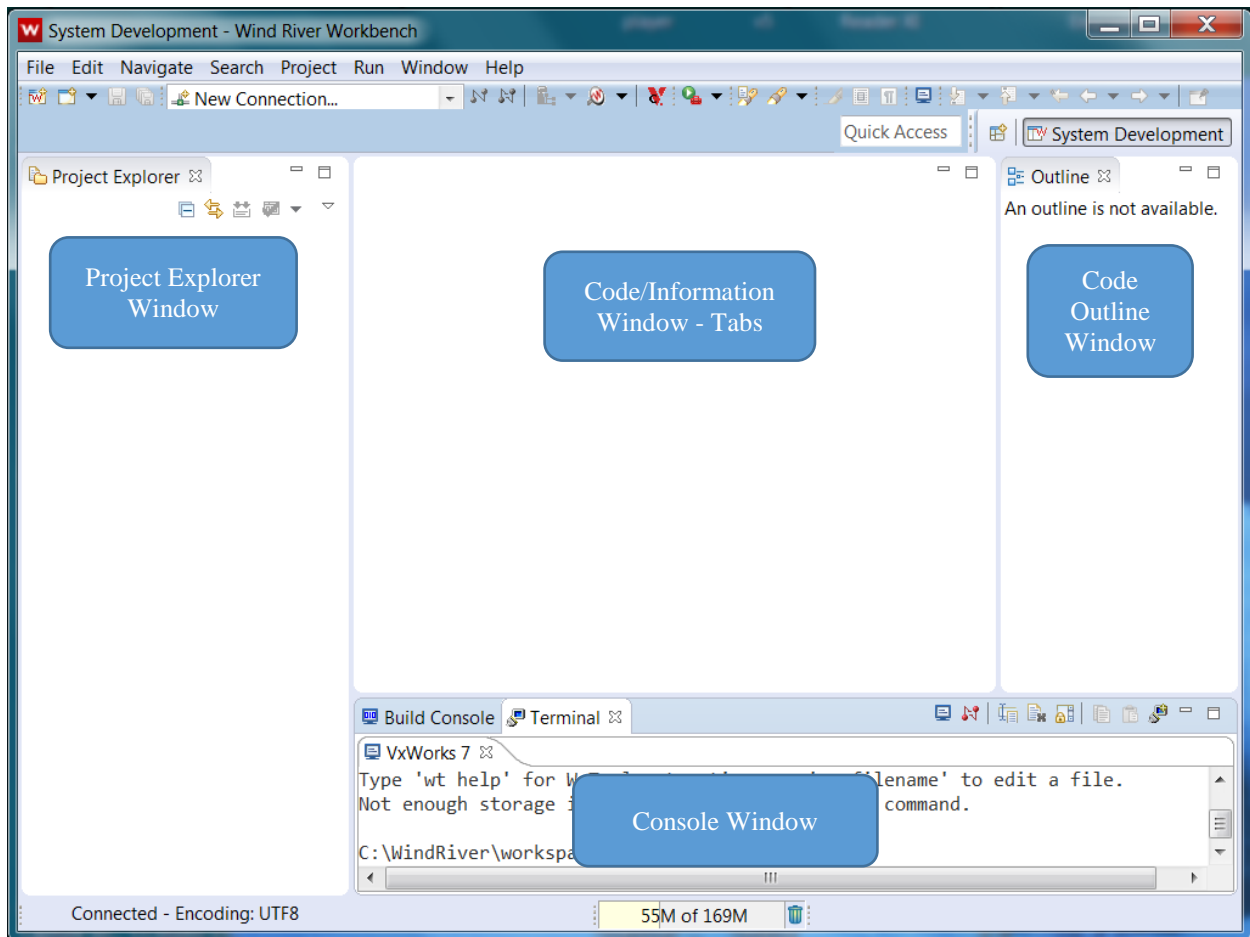


VxWorks is based on the Eclipse user interface, which should look familiar to students who have taken ECEN 260, it is the same interface as the TI Code Composer tool uses. The interface provides lots of flexibility of what is displayed, and even includes organized perspectives. These can be selected by clicking on the icon in the upper right hand corner:



Each of these perspectives provides a set of windows that are designed to optimize that stage of the code development and debug process. We'll cover most of them as we go along. But let's leave it on the System Development perspective for now.

The System Development Perspective provides a number of individual sub-windows. Here are the details for the default System Development View windows:



**Project Explorer Window** – This will show all of the projects that you have created and you can select between them to compile, download code, or debug.

**Code/Information Window** – As you open files to edit, they will be displayed here. This area is also used to display information like details on the connection.

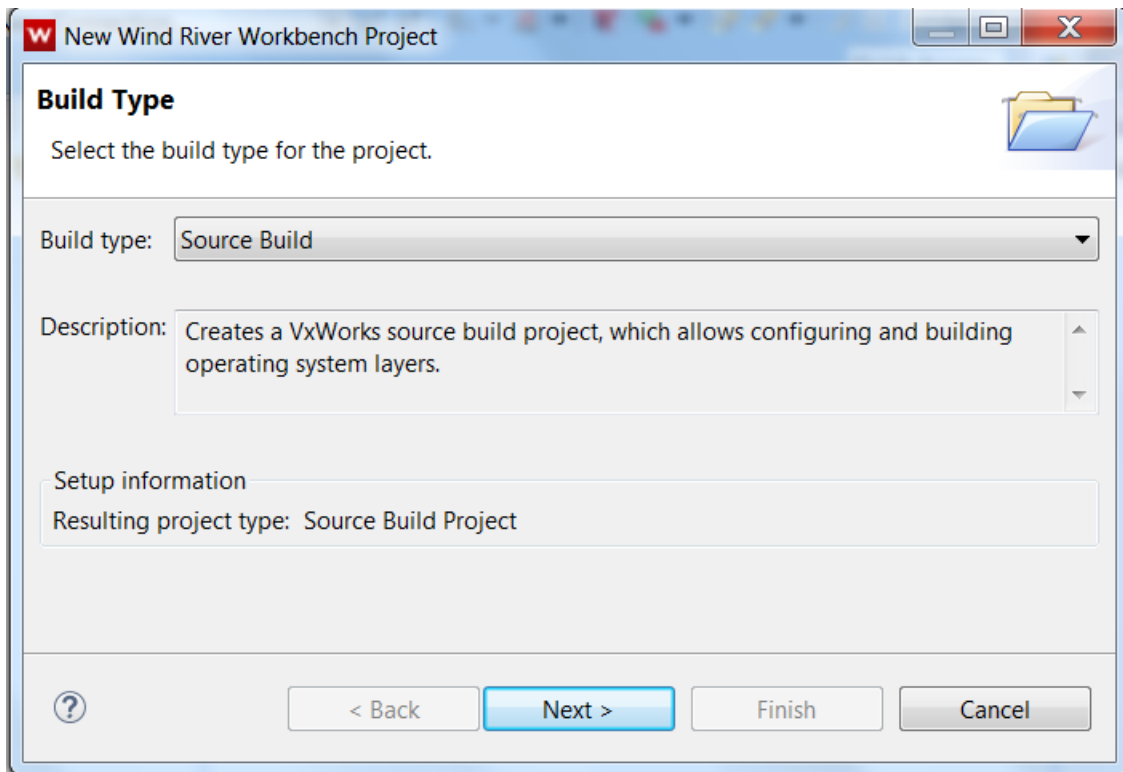
**Code Outline Window** – This will show and outline of the functions associated with the code that you are creating.

**Console Windows** – This windows holds details on the Build Console, or a look on the details of the compile step. Also any terminals you may have connected to your Target System.

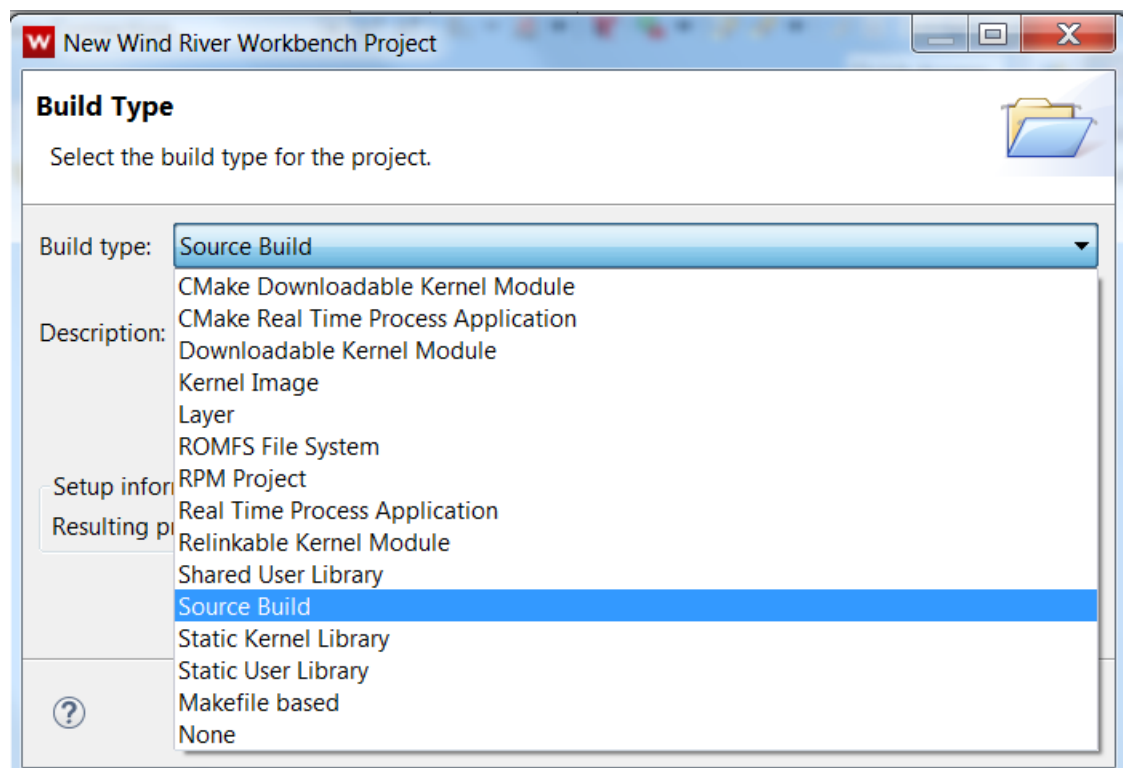
So now that you are at least familiar with the interface, let's open up a project. We're going to create and build some code that will run on the simulator, so for this lab the connections will all be internal.

So to start you are going to create a VxWorks Source build (VSB) project based on a board support package for the VxWorks Simulator.

To create a new project choose from menu File>New->Wind River WorkBench Project. You should see this menu:



Select the Build type: field and you should see this:



Let's look at some of these types.

Downloadable Kernel Module - Creates a munched, downloadable VxWorks kernel module, which can be dynamically linked to the operating system at run time and executes in kernel memory space and in kernel mode. This type of project is integrated right into the RTOS, but you are only creating a module for an installed VxWorks system. You might chose this if you are developing a module for a larger system with modules created by other developers but you want your module to appear as if it is part of the operating system itself.

Kernel Image - Creates a bootable VxWorks kernel image, which can be linked with kernel modules, and can be configured to include a ROMFS file system. This type of project is designed to be held on a file system. This is complete VxWorks kernel image.

Real Time Process Application - Creates a fully linked, relocatable VxWorks executable, which executes in user memory space and in user mode. It can be stored on a host file system, a network file system, or a local file system on the target (including ROMFS). This creates an entire executable system. This is similar to the Downloadable Kernel Module, however this code will execute not in kernel memory space and mode, but in application mode. This Application will not look like it is part of the operating system, but an application running on top of the operating system.

Source Build - Creates a VxWorks source build project, which allows configuring and building operating system layers. This type of project builds an entire executable, normally based on a BSP.

Normally you'll chose one of these four types, based on your project needs. For this lab you'll choose the Source Build. This will build the entire VxWorks system on top of your simulator BSP. This will serve as a base for your code specific projects.

So select Source Build. Then hit Next. Give your project a name. Then hit Next. You should see this selection:

New VxWorks Source Build Project

### Project Setup

Base the new project either on an existing configuration, a board support package, or a CPU type.

Setup the project

Based on: a VxWorks 7 board support package

BSP: vxsim\_windows\_1\_0\_2\_5 ☐ Show latest versions only

Address mode: ILP32 32-bit libraries


Options:

Compiler support:	Primary and secondary compiler
Board CPU:	SIMNT
Floating point setting:	hard
Endianness setting:	little
VxWorks 6.9 compatibility:	Disabled for this BSP
Processor mode:	UP support in Libraries
Debug mode:	Off, and normal compiler optimizations enabled

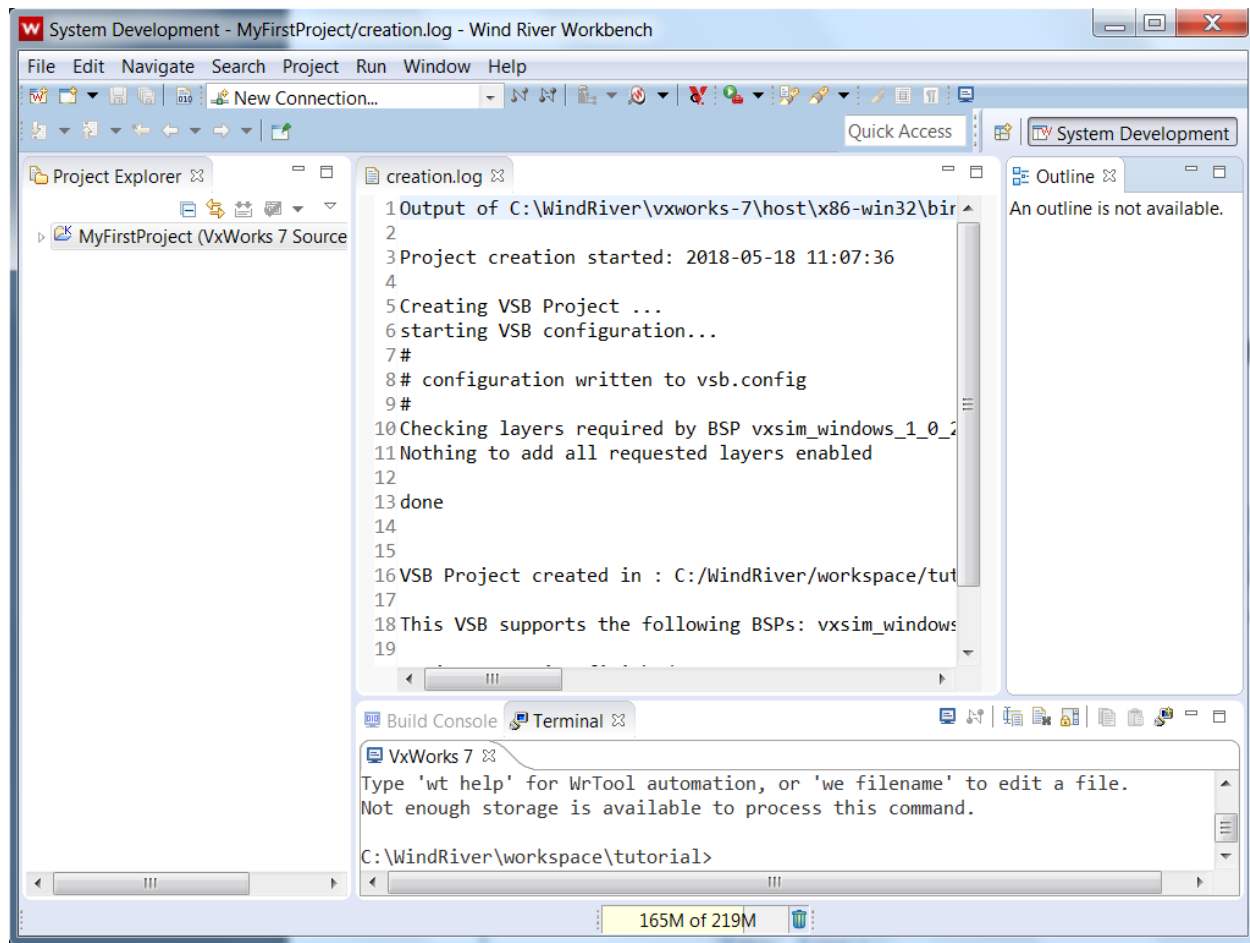
☐ Link in sources to project

Setup information

Base location: vxsim\_windows\_1\_0\_2\_5

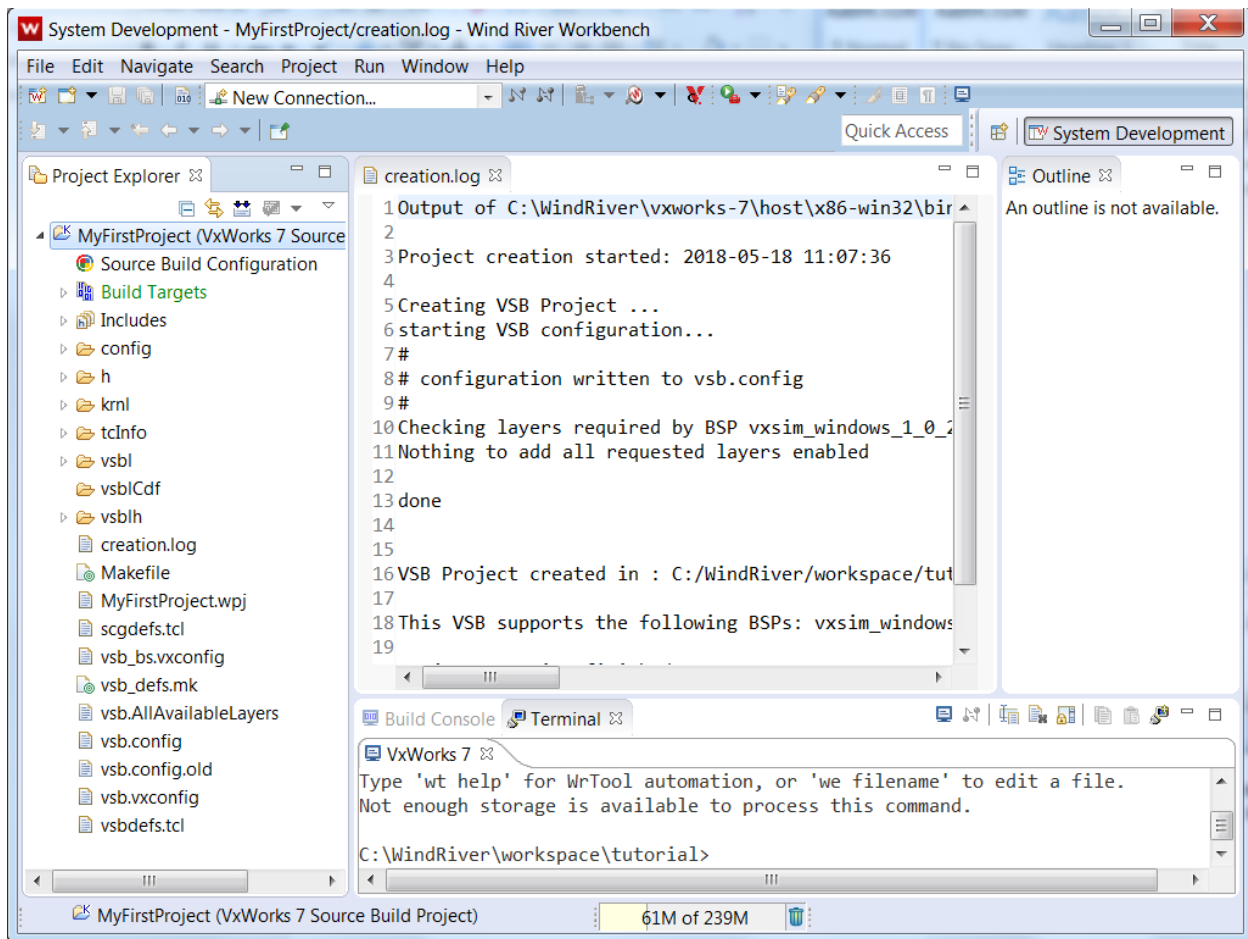


In this case you'll want to base your VxWorks system on a BSP package. Since you are going to use the simulator running on the Windows machine, you'll select the vxsim\_windows\_1\_0\_2\_5 BSP. Leave the rest with the defaults. Click Finish and you should see this:

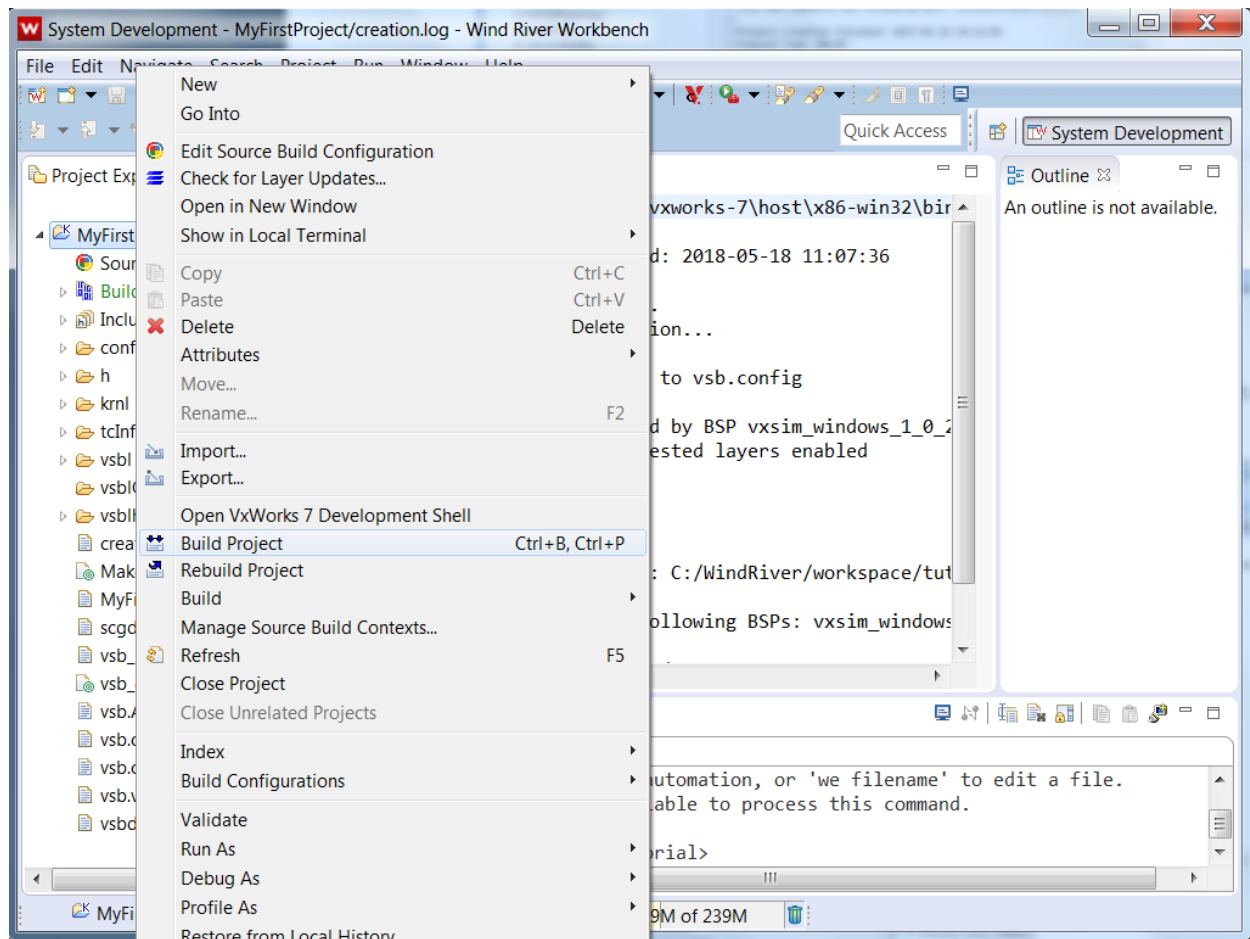


The Project Explorer has update with your project. The creation.log file is shown in the Code/Information window. You can expand the view of your project by clicking on the arrow indicator right next to the project name:

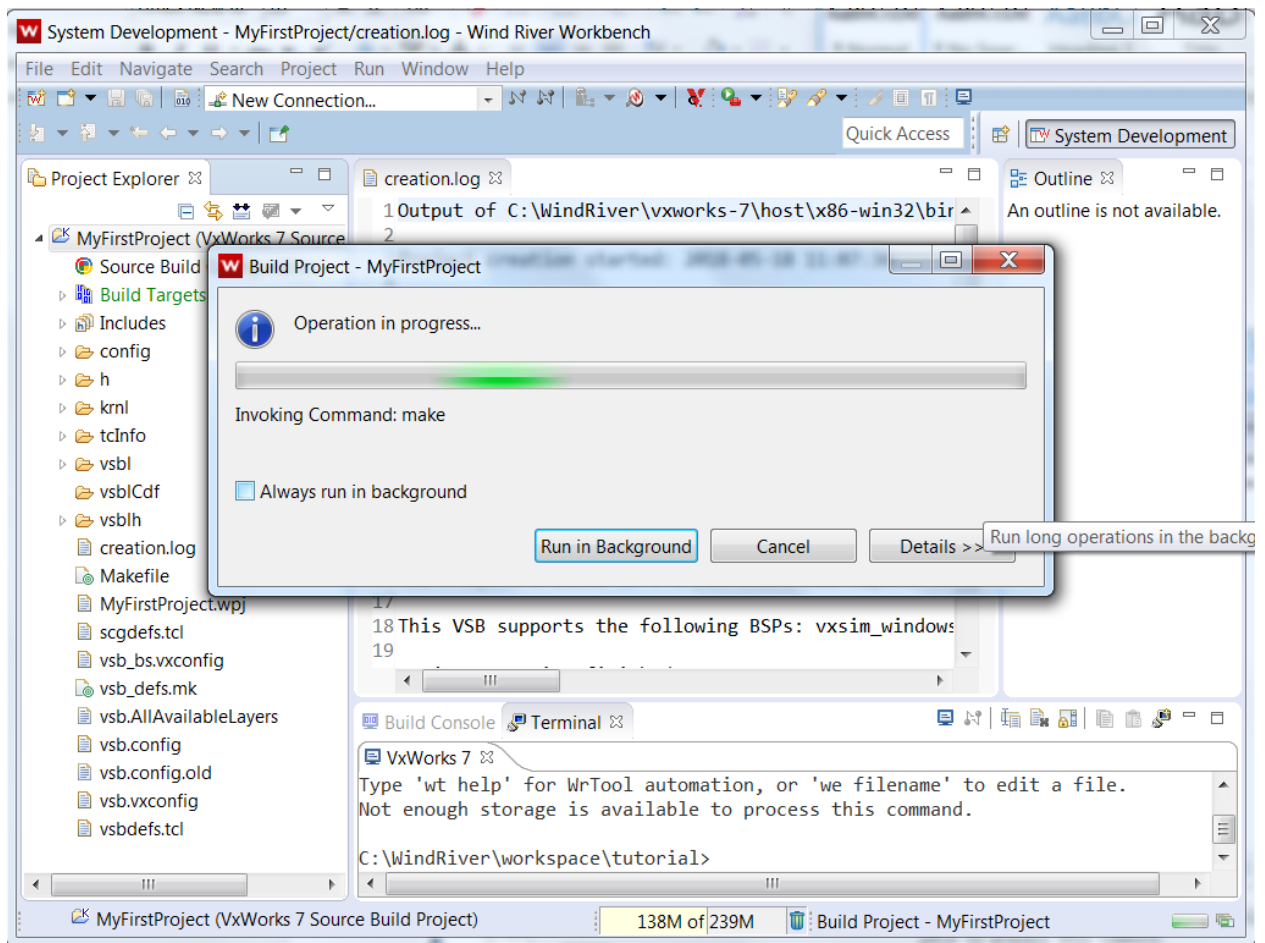




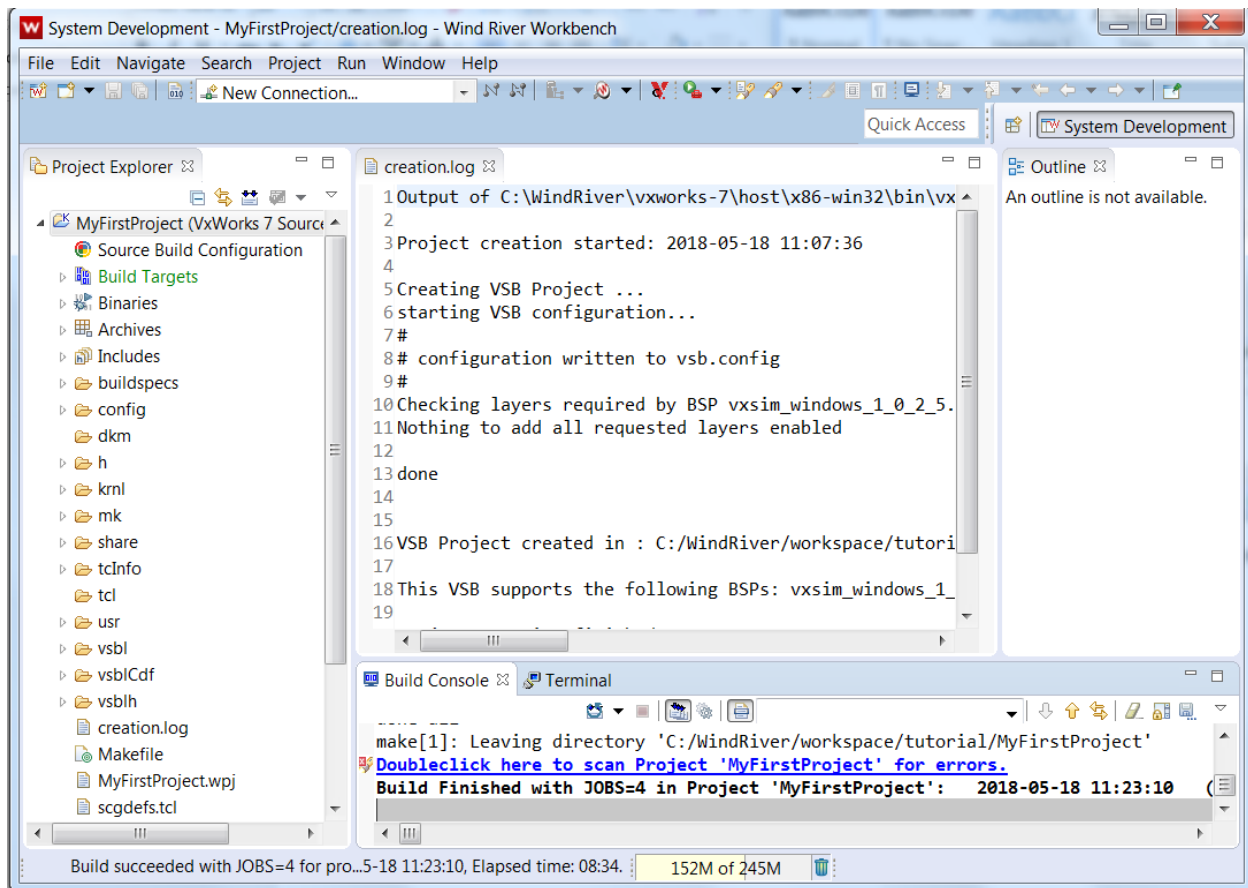
You can see for your project a significant number of files have been created. We won't go through them now, we'll look at the ones that make sense as we go. Now that you've created all the files, let's go ahead and build the code set. To do this right click on the Project, and then select Build Project from the drop down menu:



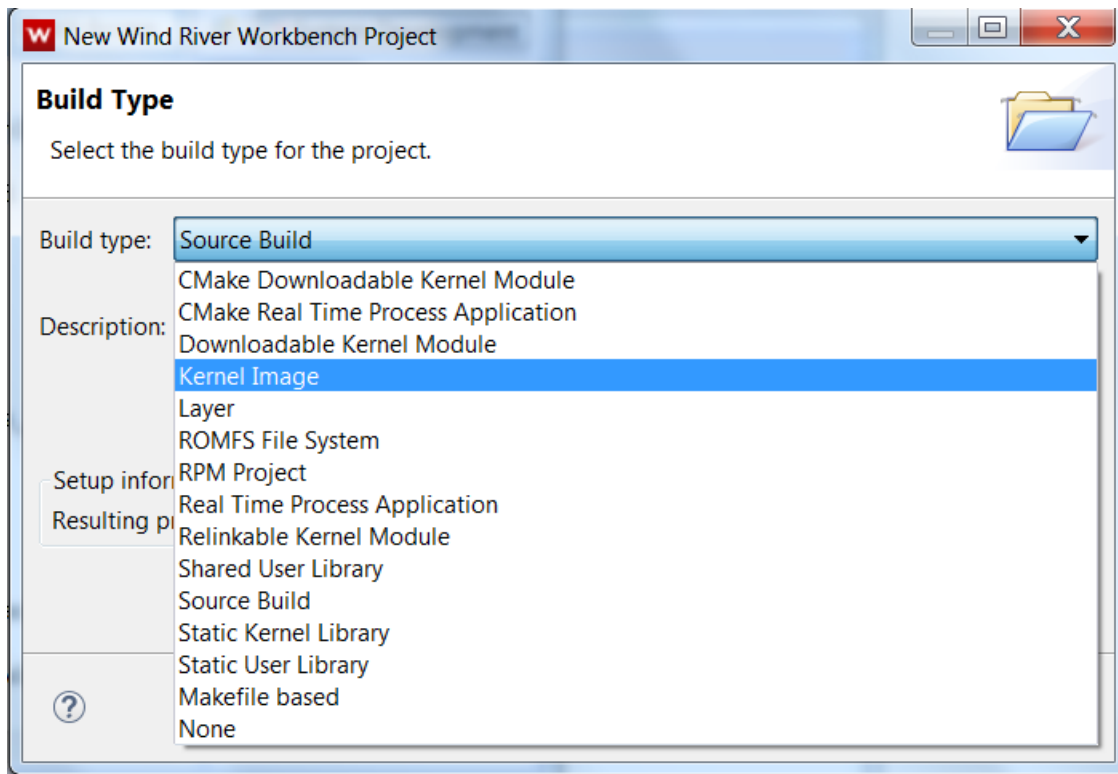
The system may ask about parallel builds, but just click OK. Then you should see this:



As you are building the entire VxWorks system from source this may take a very long time. When finished you can click on the Build Console Tab in the Console window and you should see this:



The Build finished. In my case it took around 20 minutes. You'll want to add code to run on top of the base system that you've built. To do this you'll build a VxWorks Image Project. This will create a complete set of code with your code built on top of the base code you've already built. To do this select File->New->Wind River Workbench Project. This time select a Kernel Image file.



Chose a name, then hit next, and the dialog should auto fill with the information from the last file you created:

**New VxWorks Image Project**

### Project Setup

Base the new project either on an existing project, or on a source build project providing board support package and tool chain selections.

Setup the project

Based on: a source build project

Project: MyFirstProject

BSP: vxsim\_windows\_1\_0\_2\_5

Tool chain: gnu

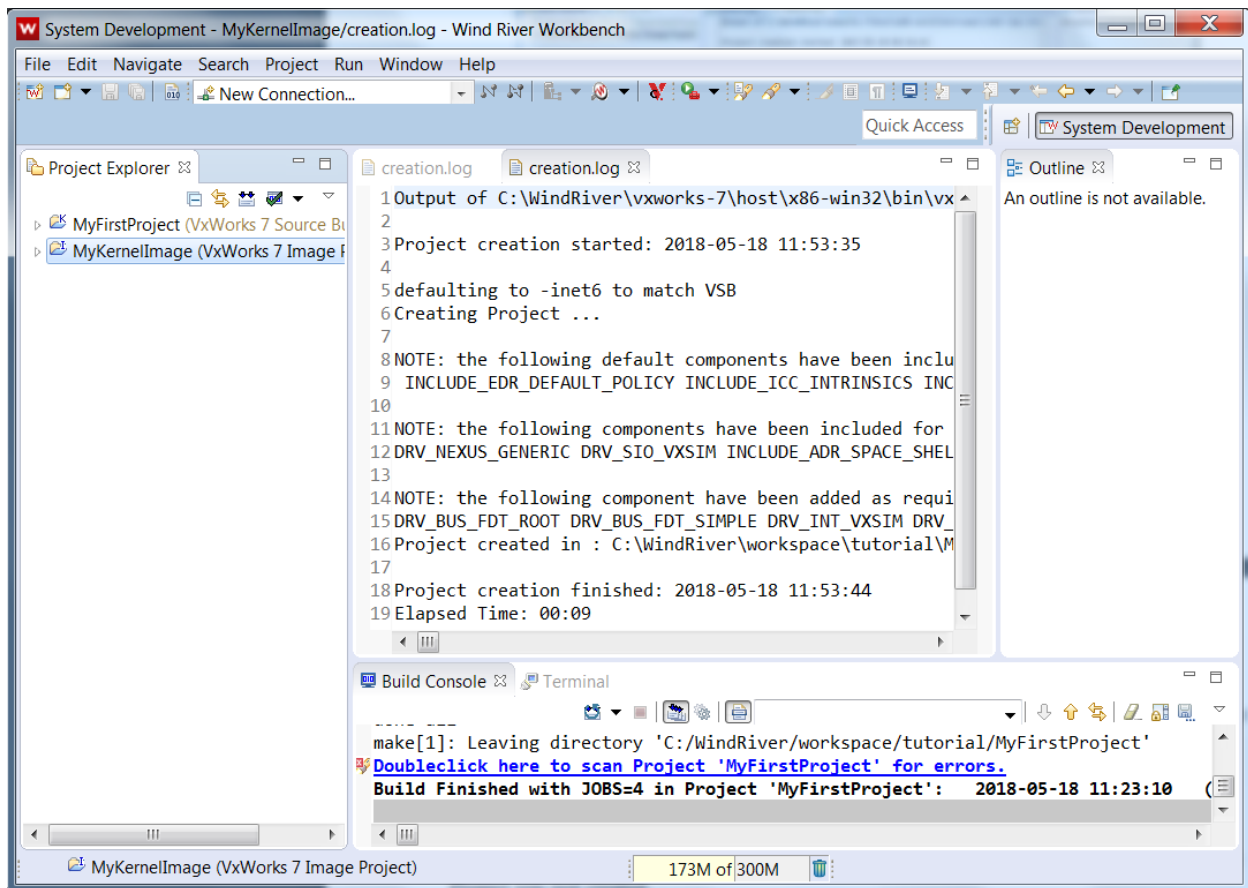
Options:

- Address mode: ILP32 32-bit libraries
- Processor mode: UP support in Libraries
- IP version setting: IPv6 and IPv4 enabled libraries
- Debug mode: Off, and normal compiler optimizations enabled

Setup information

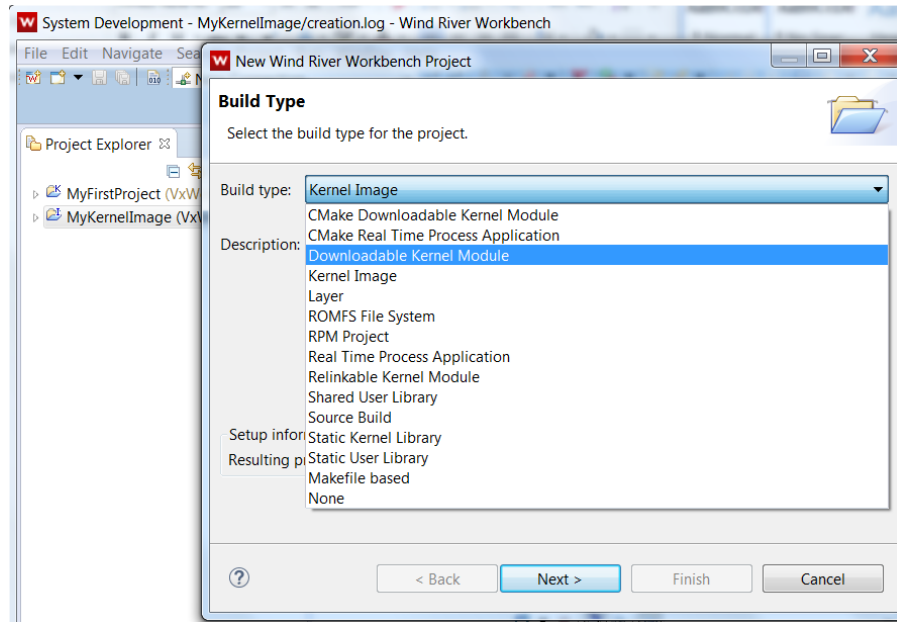
Base location: C:/WindRiver/workspace/tutorial/MyFirstProject

The new full implementation will be built on the kernel image that I just built. Now click finish. Notice in your Project Explorer window that you have a new project.

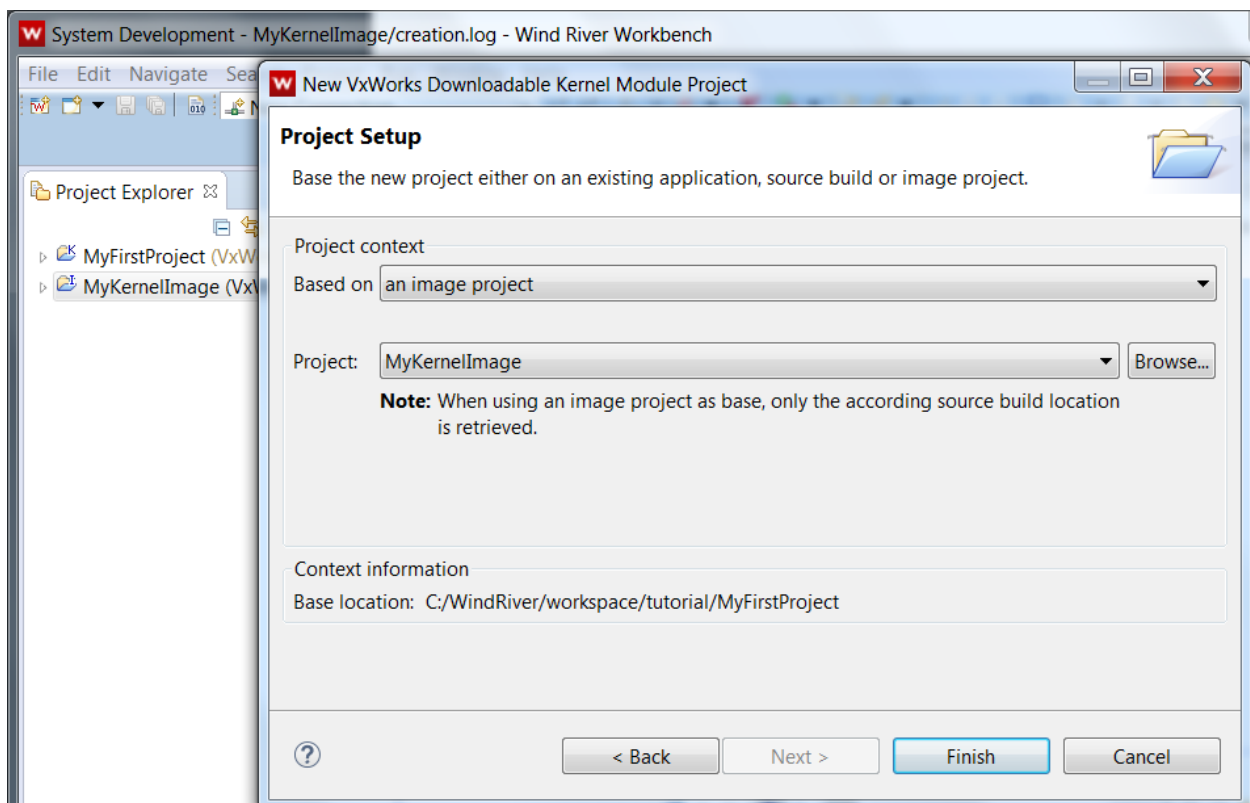


Now build this Project by right clicking on the Project and selecting Build Project. This project should take much less time to build as it is based on the Kernel that you built in the last project. This Kernel Image project has all the code you need to actually run. However, you haven't added any code that needs to be run. So you will create one more project, this time adding just your code. This will be a downloadable Kernel Project that has just your code.

Select File->New->Wind River Workbench Project. Select Downloadable Kernel Module for the Build Type.

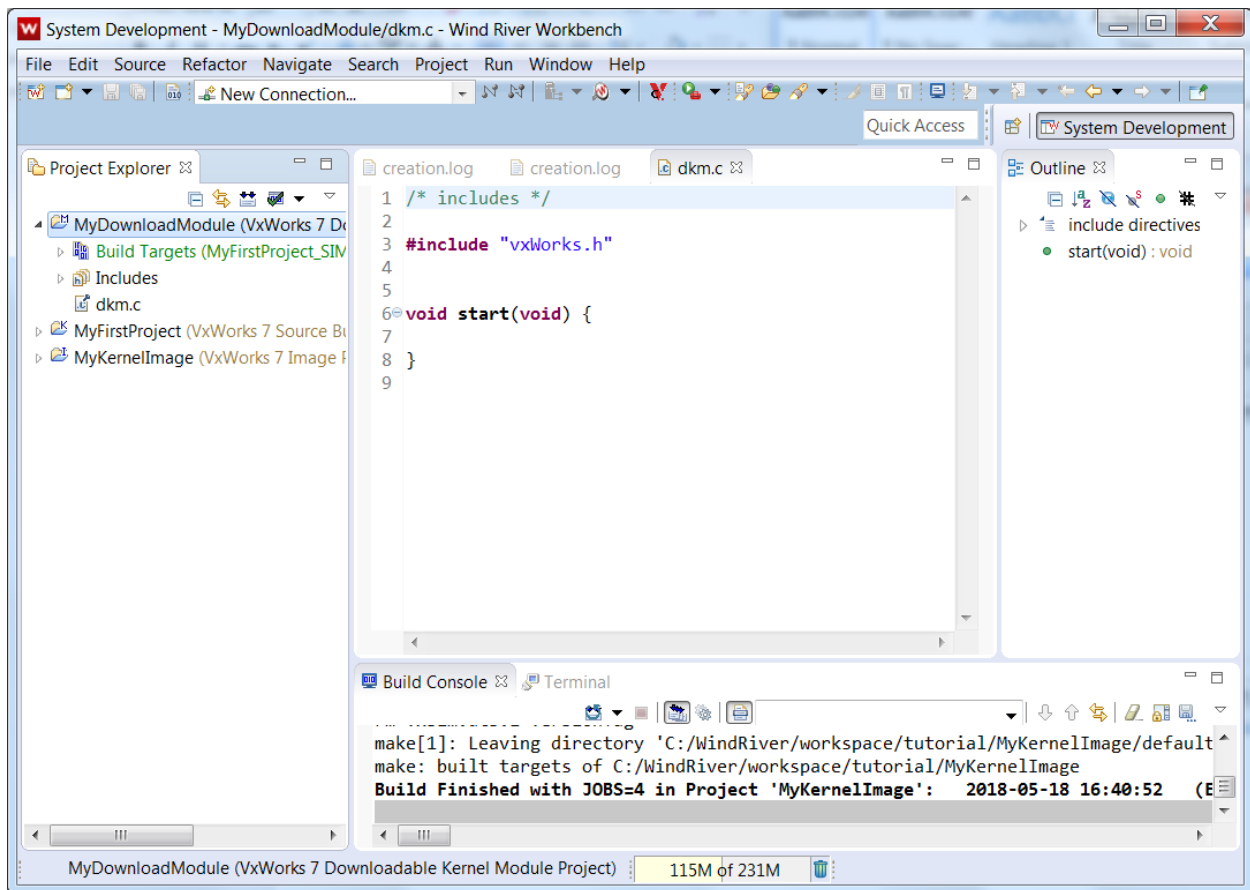


Enter a Project name, then hit Next. Now select your Project to be based on an image project, and chose the image project you just created.



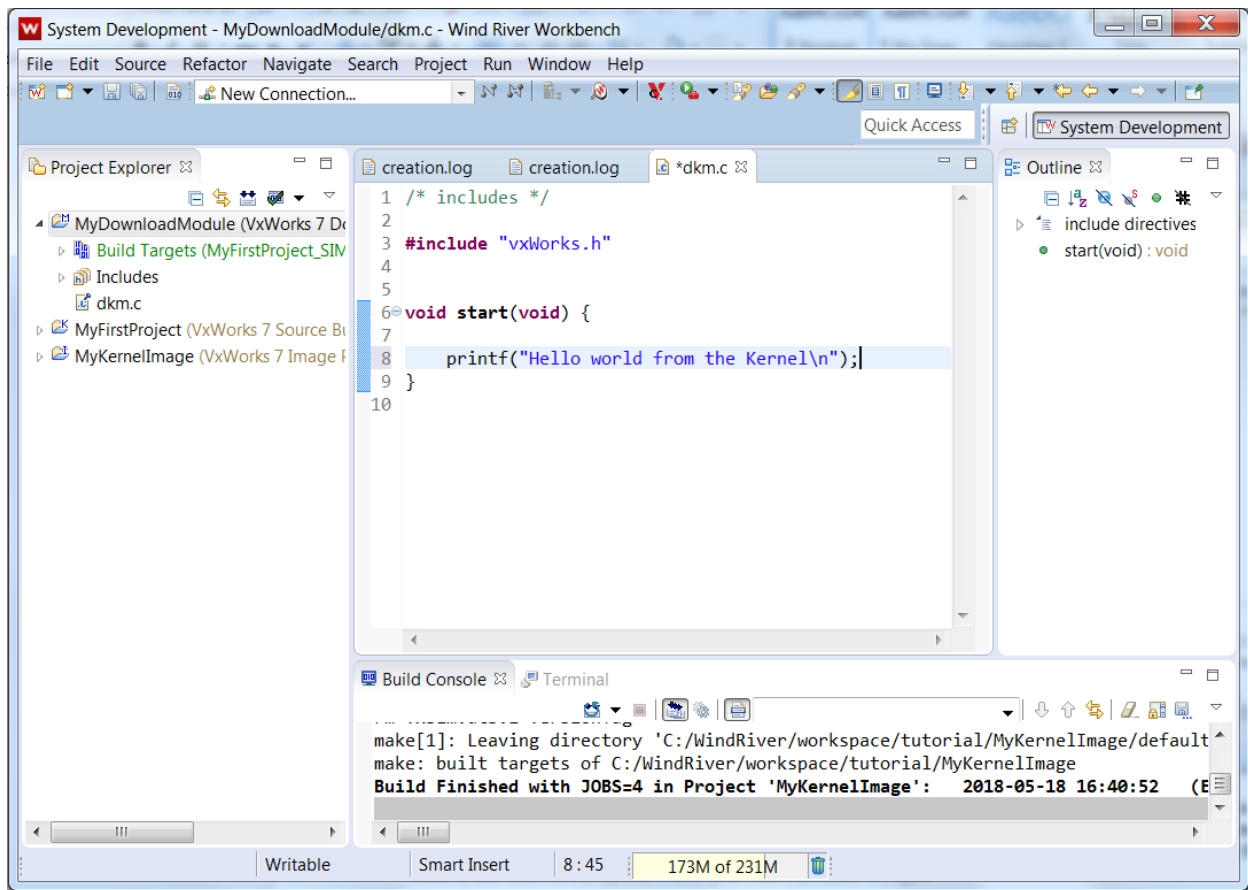
Now click Finish. The project will be created in the Project Explorer window.





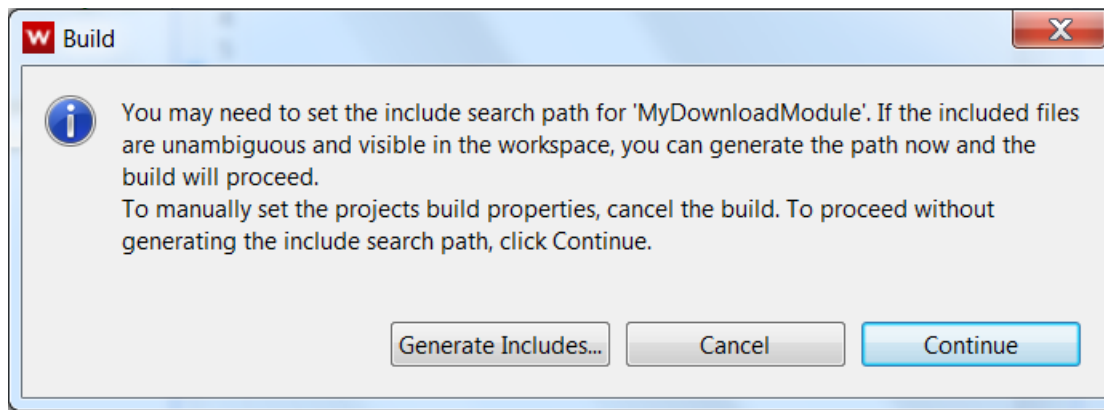
You'll also notice something new and interesting. The Code Window now shows access to a code file. This is where you will place the code you want to create for your project. It is in the file `dkm.c` (short for downloadable kernel module). You'll also note that there is no `main()` function. Rather you are going to place a set of functions in the file, build it, and then tell the system where you want execution to begin.

So let's put some simple code there:

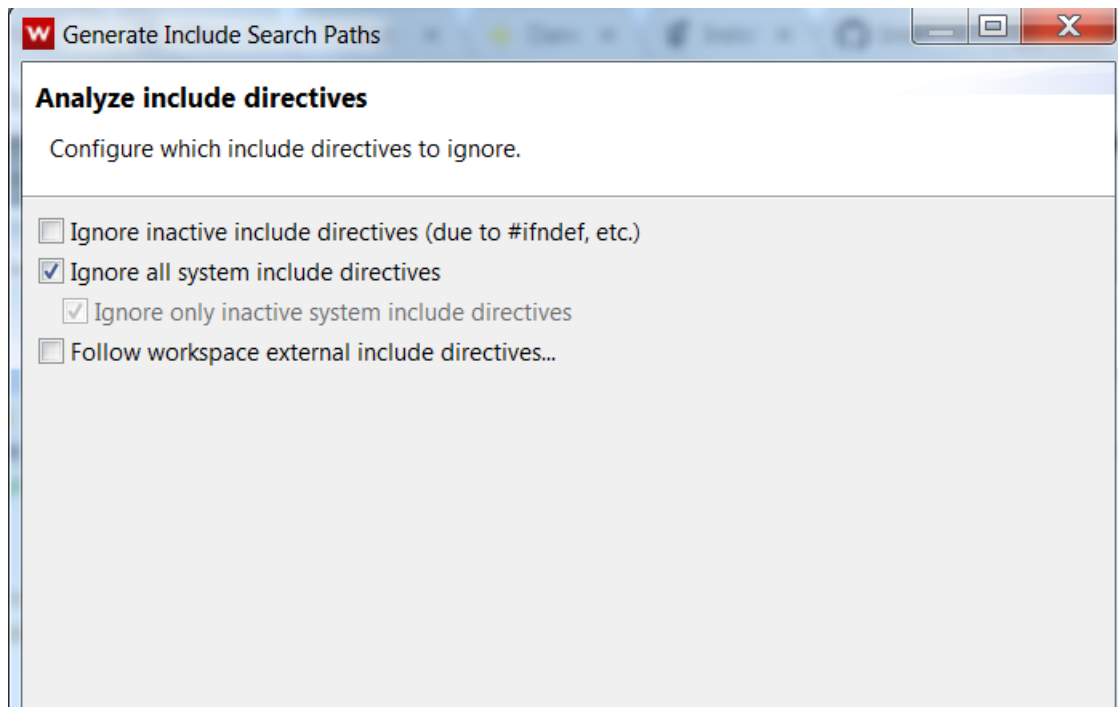


You'll also note, by the way, that the Outline window now shows the outline for your code, which functions are in the code.

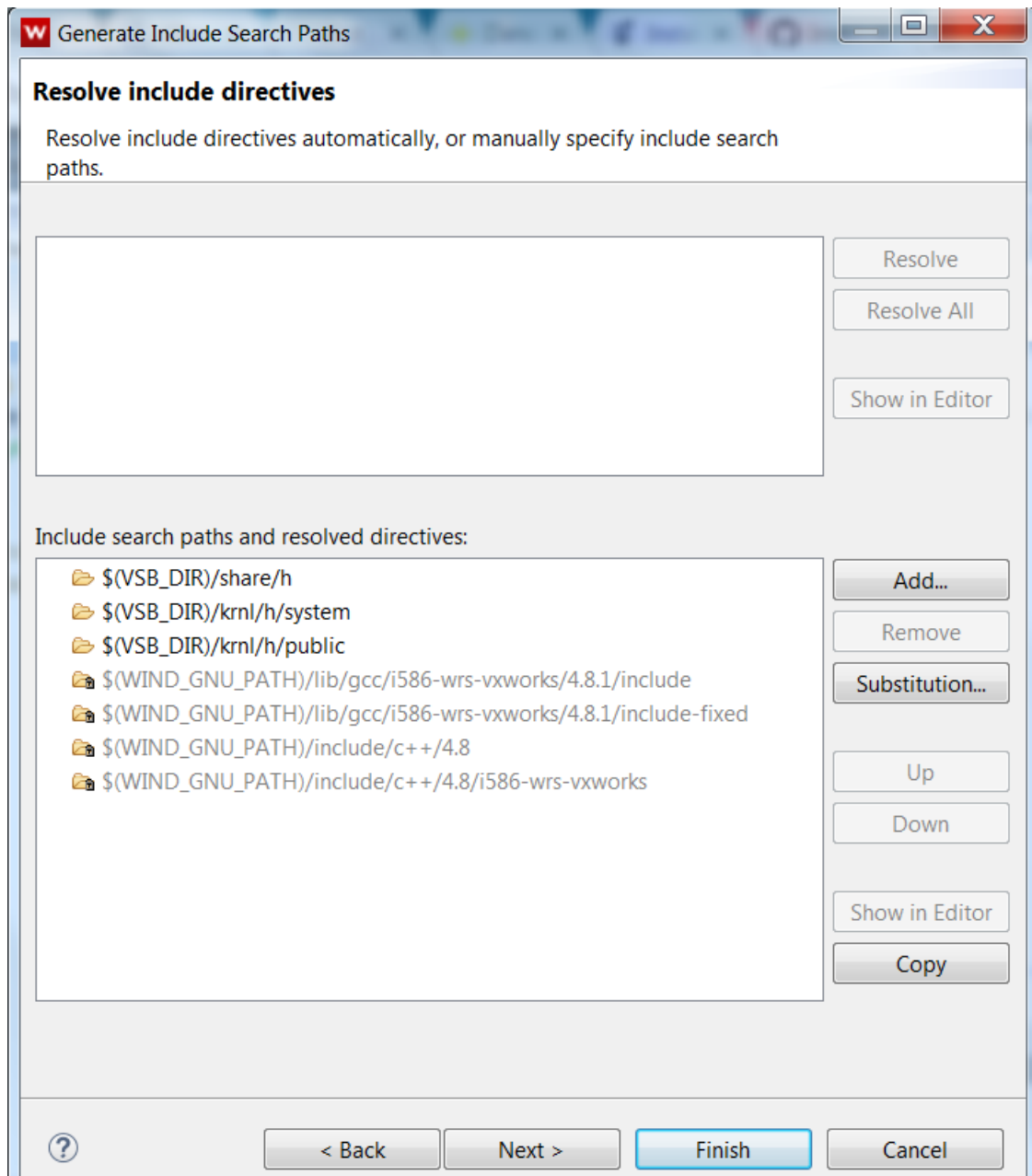
Now right click on the Project and select Build Project. You should first see this prompt:



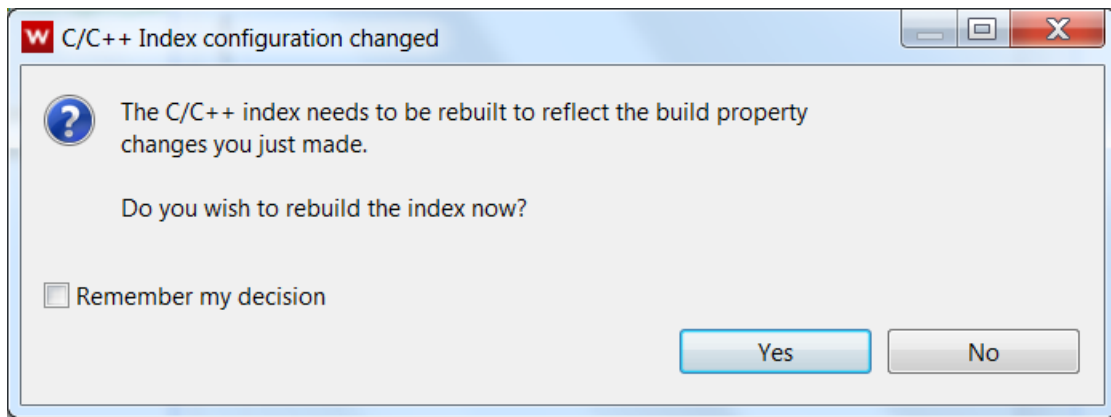
Go ahead and select Generate Includes. Now you will see this prompt:



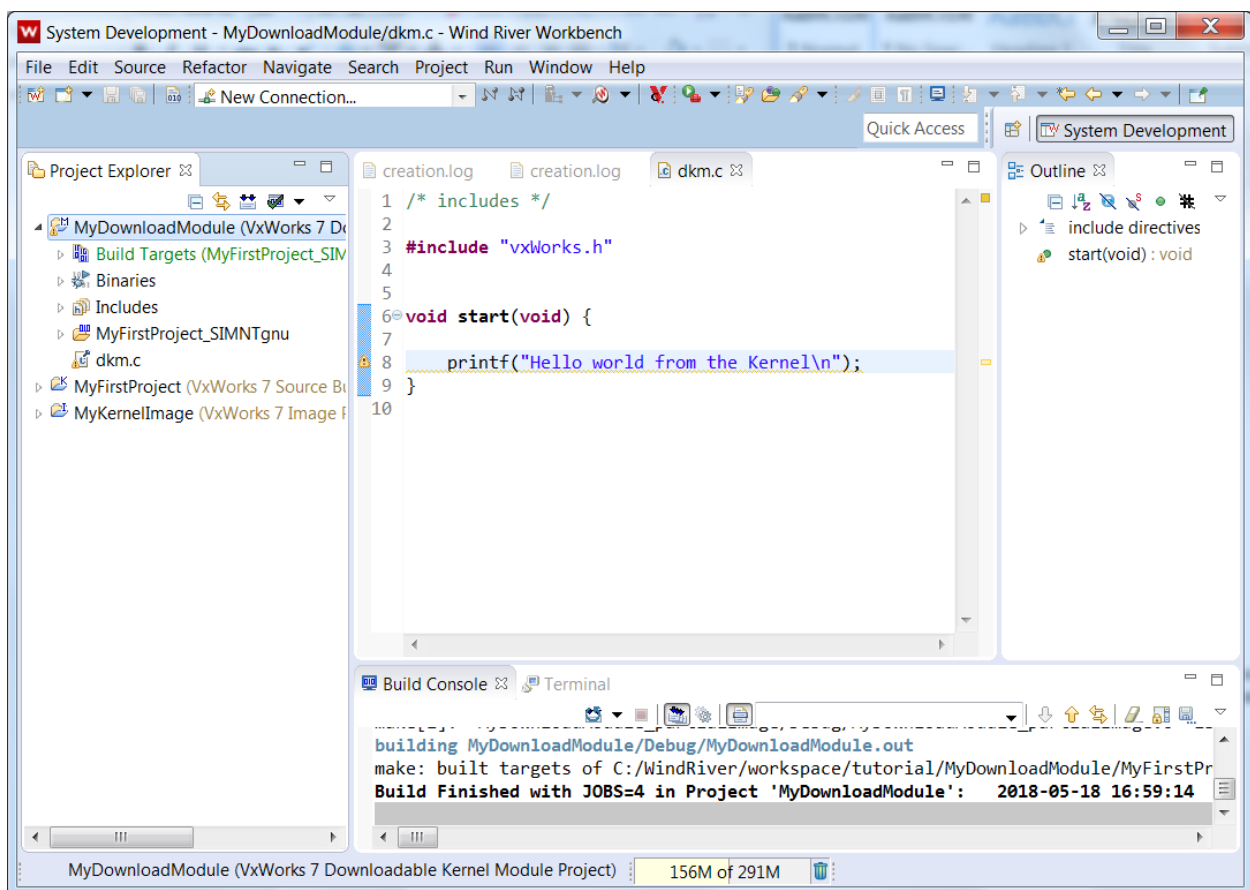
Select just the Ignore all system directives. At the next prompt just select the Finish button:



The last step will be this dialogue box:

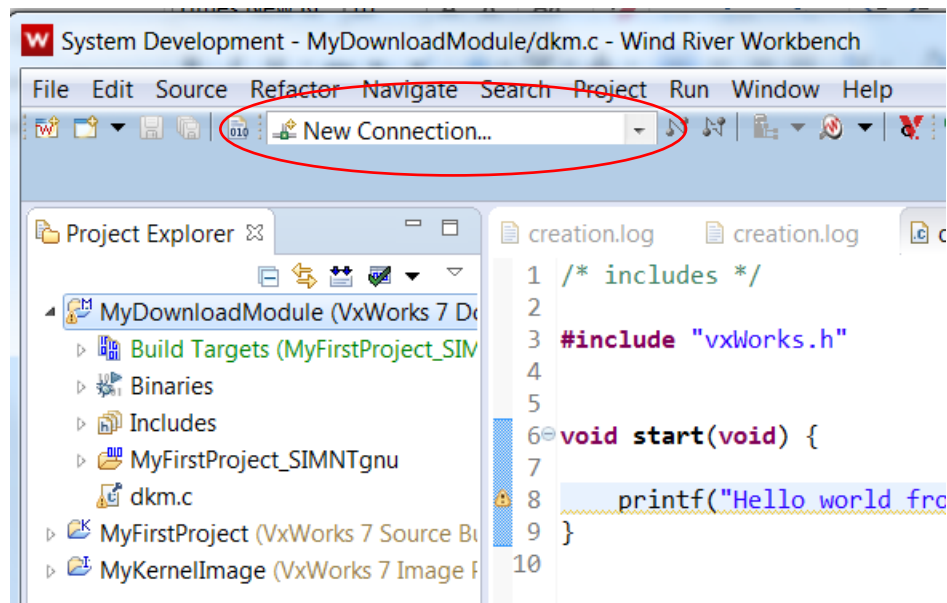


Go ahead and select Yes. The code will build, as shown in the Build Console window.

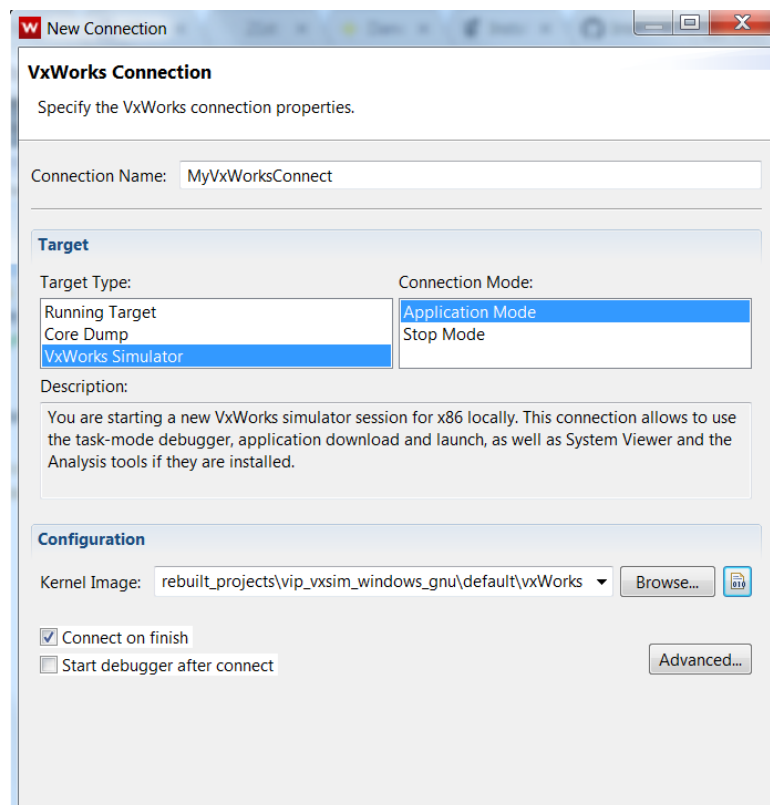


Now you have code that you can actually run. Normally you would run this on the target hardware, but for this example you are going to use the simulator running on the host machine.

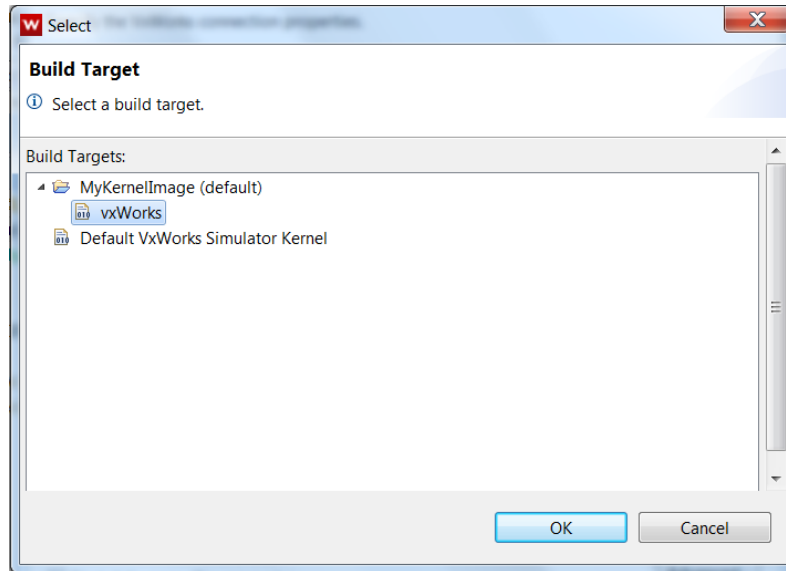
Now you have the code, let's connect to the simulator to run the code. At the top of the Wind River IDE is a tool bar. Select the New Connection... selection from the bar.



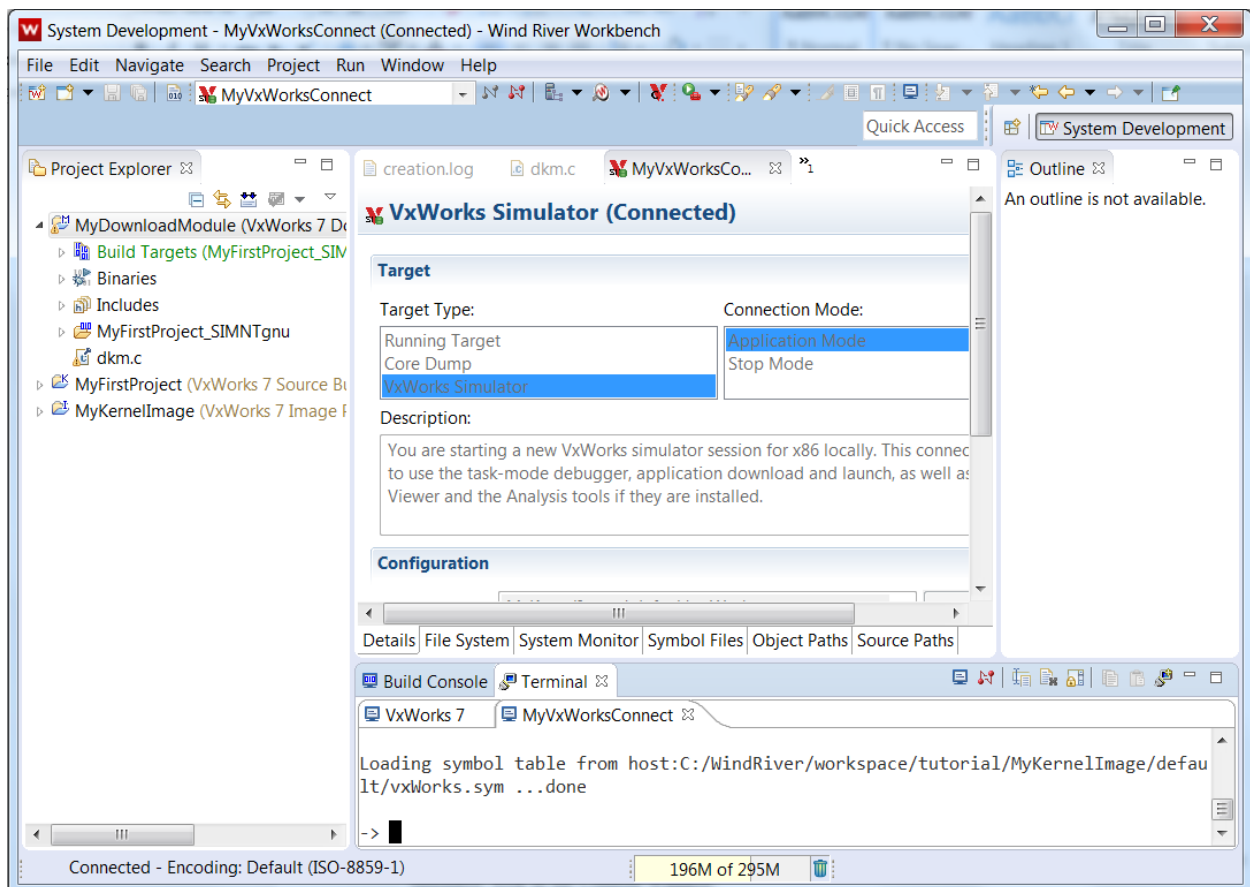
Fill in the information by selecting a name for your simulator connection, selecting the VxWorks Simulator, and the Application Mode:



Now you need to select the kernel image you created earlier. To do this select the button with the 1010 on it, then you should see this prompt:



Select your kernel image and click OK. Then click Finish. Now you should see that you are connected to the simulator, look in the Console Window:



If you expand the Console Window you'll see more information on VxWorks:



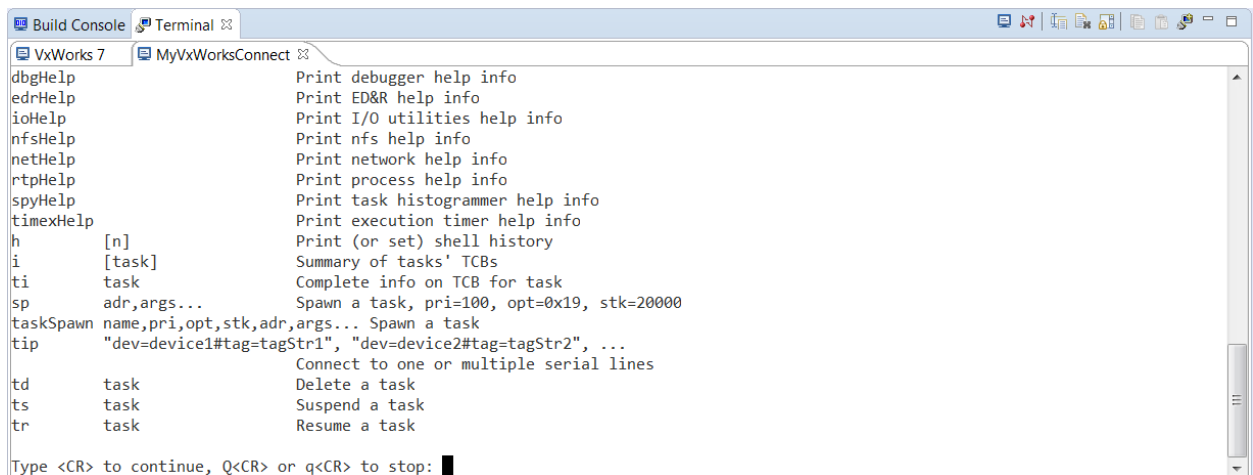
```
Build Console Terminal
VxWorks 7 MyVxWorksConnect
\7777777\ \7777777/
\7777777\ \7777777/
\7777777\ \77777/
\7777777\ \777/
\7777777\ \7/
\7777777\ -
\7777777\ -
\7777777\ Copyright Wind River Systems, Inc.
\777777/ 1984-2018
\7777/ -
\777/ \7\
\7/ \777\
- -----

Board: SIMNT board
OS Memory Size: 508MB
ED&R Policy Mode: Deployed
Application Mode Agent: Started (always)
Stop Mode Agent: Started (always)

Loading symbol table from host:C:/WindRiver/workspace/tutorial/MyKernelImage/default/vxWorks.sym ...done

->
```

Just like an actual target, we can control the target through this interface. You can type help, and see this:

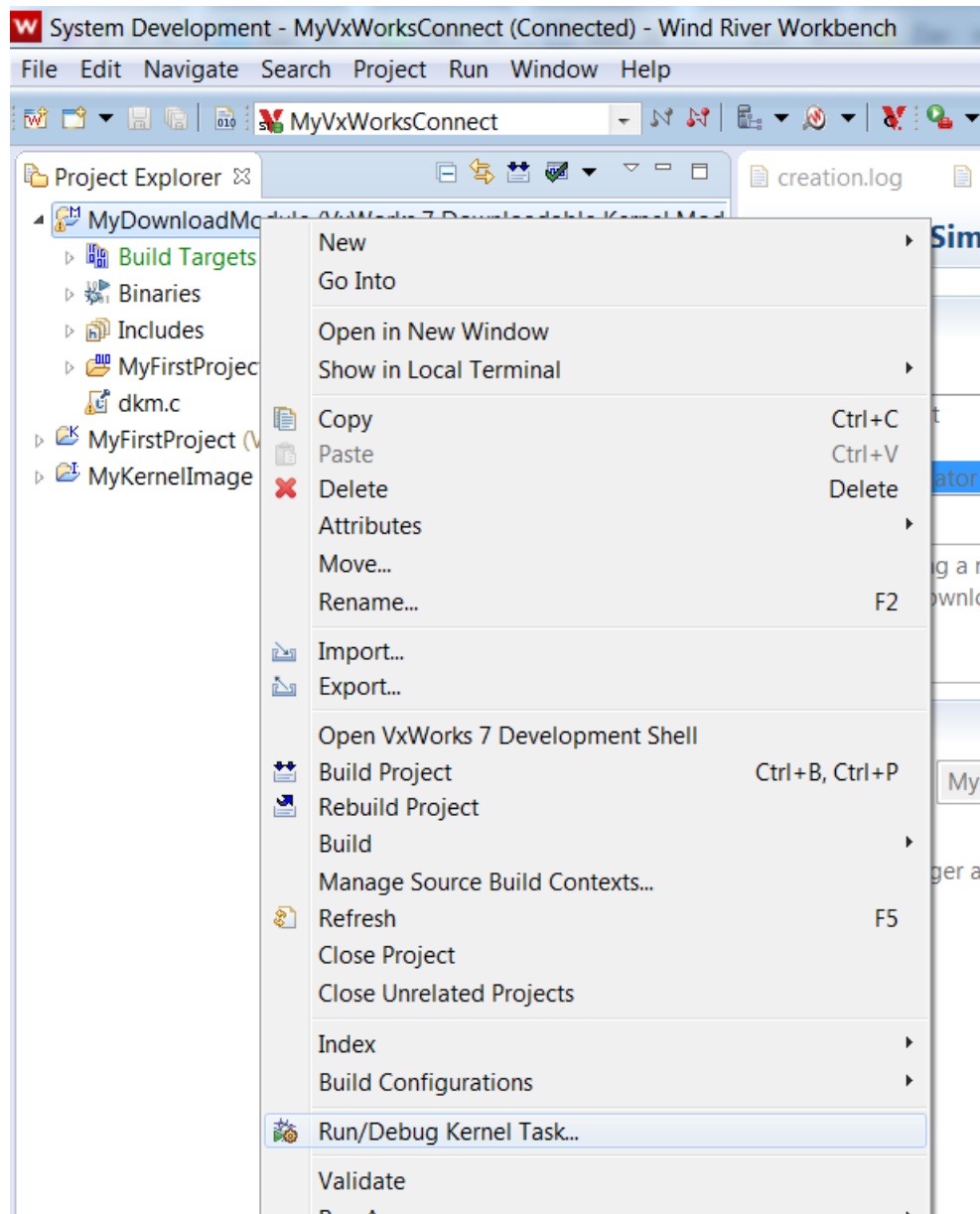


```
Build Console Terminal
VxWorks 7 MyVxWorksConnect
dbgHelp      Print debugger help info
edrHelp      Print ED&R help info
ioHelp       Print I/O utilities help info
nfsHelp      Print nfs help info
netHelp      Print network help info
rtpHelp      Print process help info
spyHelp      Print task histogrammer help info
timexHelp    Print execution timer help info
h            [n]      Print (or set) shell history
i            [task]   Summary of tasks' TCBs
ti           task    Complete info on TCB for task
sp           adr,args... Spawn a task, pri=100, opt=0x19, stk=20000
taskSpawn    name,pri,opt,stk,adr,args... Spawn a task
tip          "dev=device1#tag=tagStr1", "dev=device2#tag=tagStr2", ...
            Connect to one or multiple serial lines
td           task    Delete a task
ts           task    Suspend a task
tr           task    Resume a task

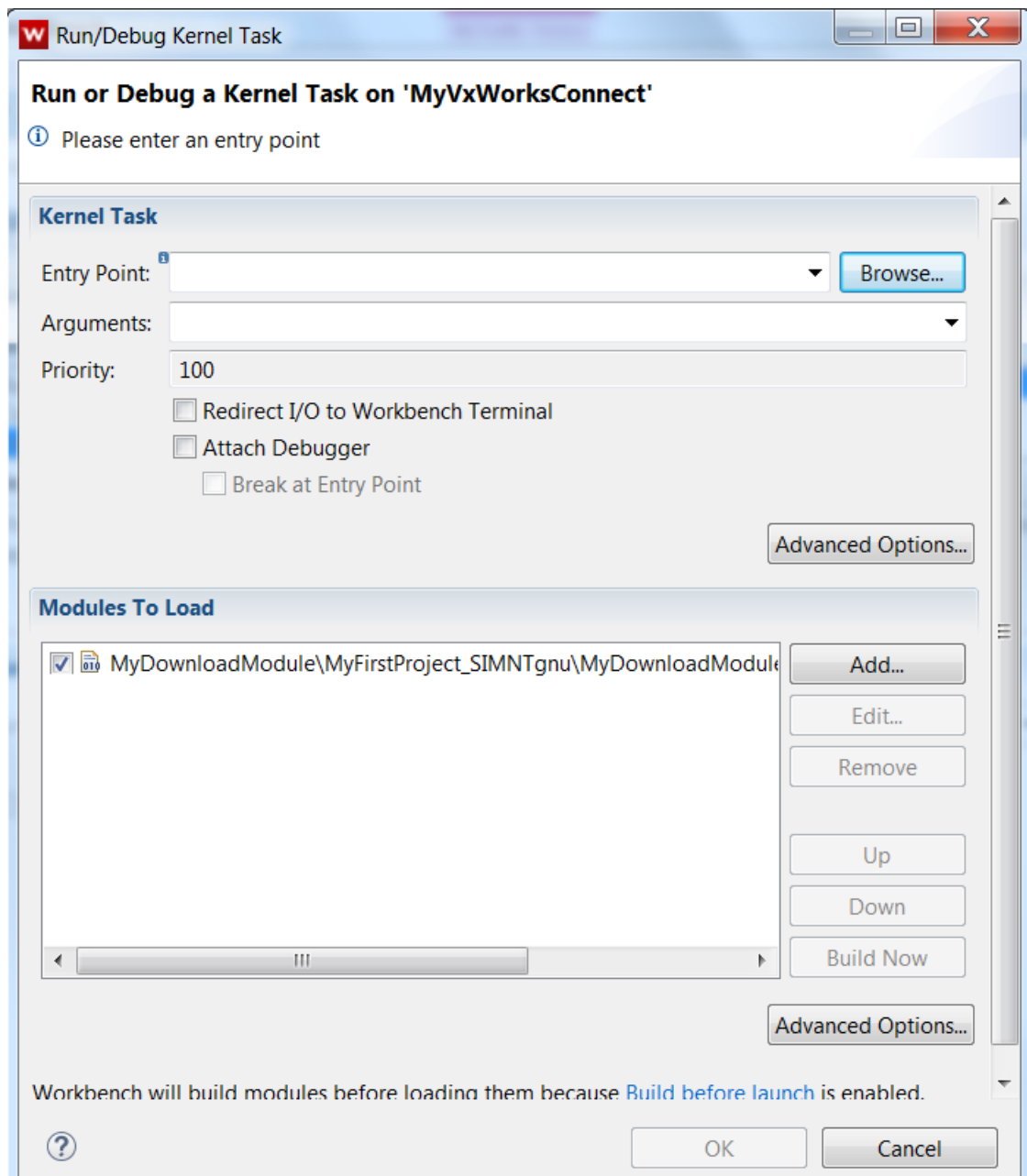
Type <CR> to continue, Q<CR> or q<CR> to stop: █
```

You can also reboot the target. You can also connect and disconnect the simulator through the Connection button at the top of the IDE, the one you used to create the connection. Now that the simulator is connected you can load and run the Kernel Task on the target. To do that Right-click the downloadable kernel module (DKM) project and select Run/Debug Kernel Task.

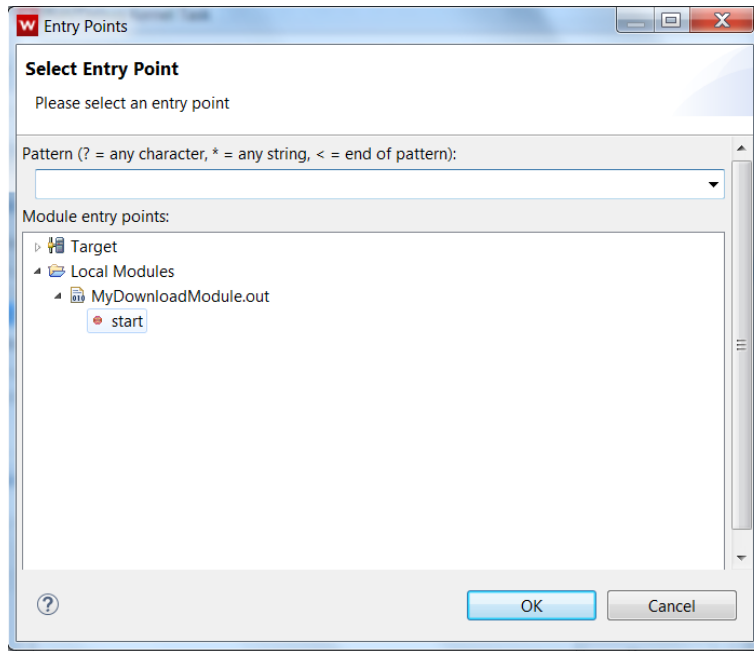




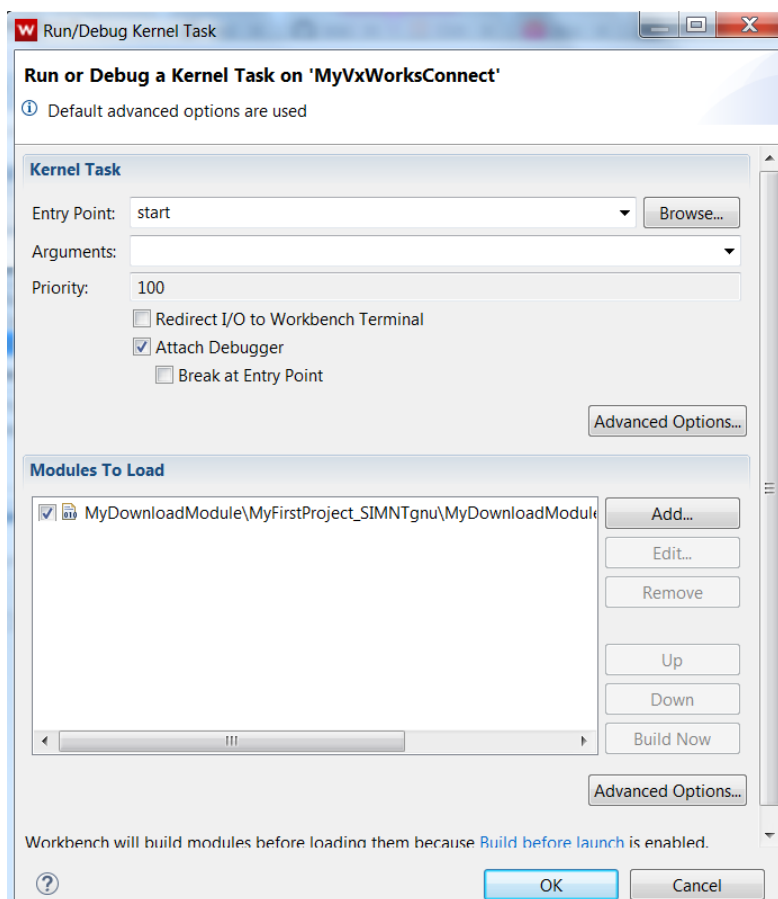
The following dialogue box will pop up:



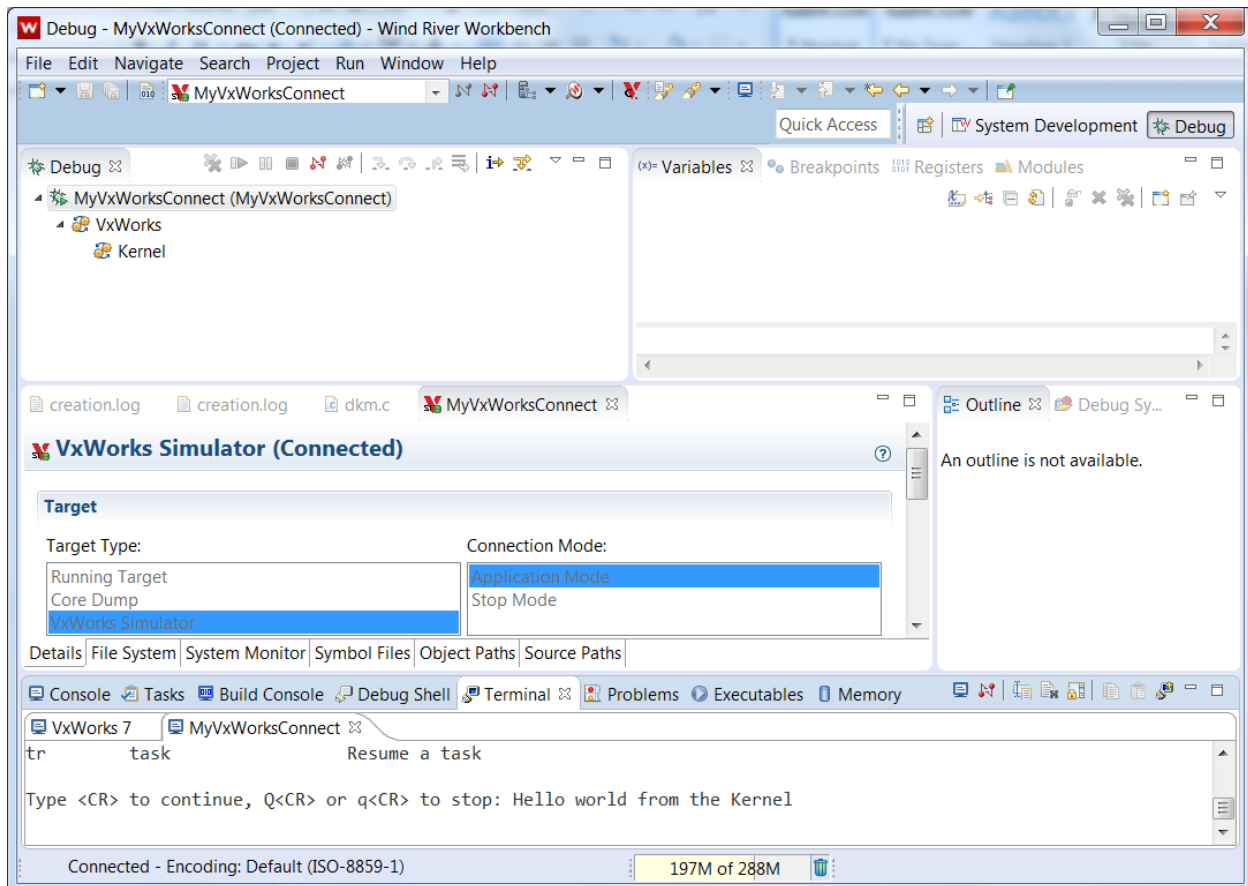
Since there is no main you'll need to tell the code where to start executing. To do this select the Browse.. Button and you will see this dialogue box:



Go ahead and select the start function and hit OK.



Now also click the Attach Debugger selection, and hit OK. The system will load the module, attach the debugger, and take you to the debug perspective, like this:



If you click on the VxWorks arrow in the debug window you'll see the Kernel image running. You'll also want to click on the Terminal window at the bottom of the screen and you'll see your printf!

Now you'll change the code for your project: Now cut and paste the following code over the dkm.c file.

```
#include <taskLib.h>
#include <stdio.h>
#include <kernelLib.h>
long int task_run[]={20000000, 450000000, 900000000};
int task_stop[]={18, 25, 30};
void task(int n)
{
    long int x;
    printf("Task %i has been started\n", n);
    while(1)
    {
        printf("%i: running\n", n);
        x=task_run[n];
        while(x>0) x--;
        printf("%i: stopped\n", n);
        taskDelay(task_stop[n]);
    }
}
void CreateTasks(void)
{

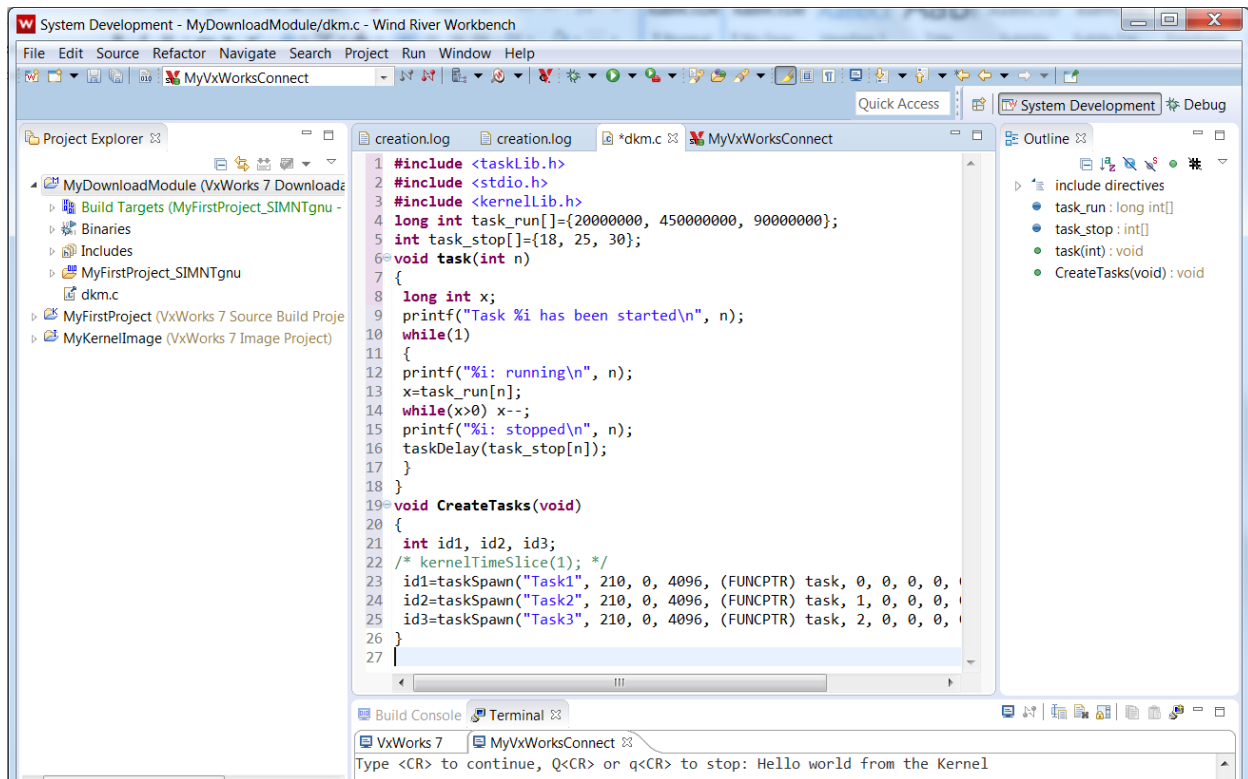
```

```

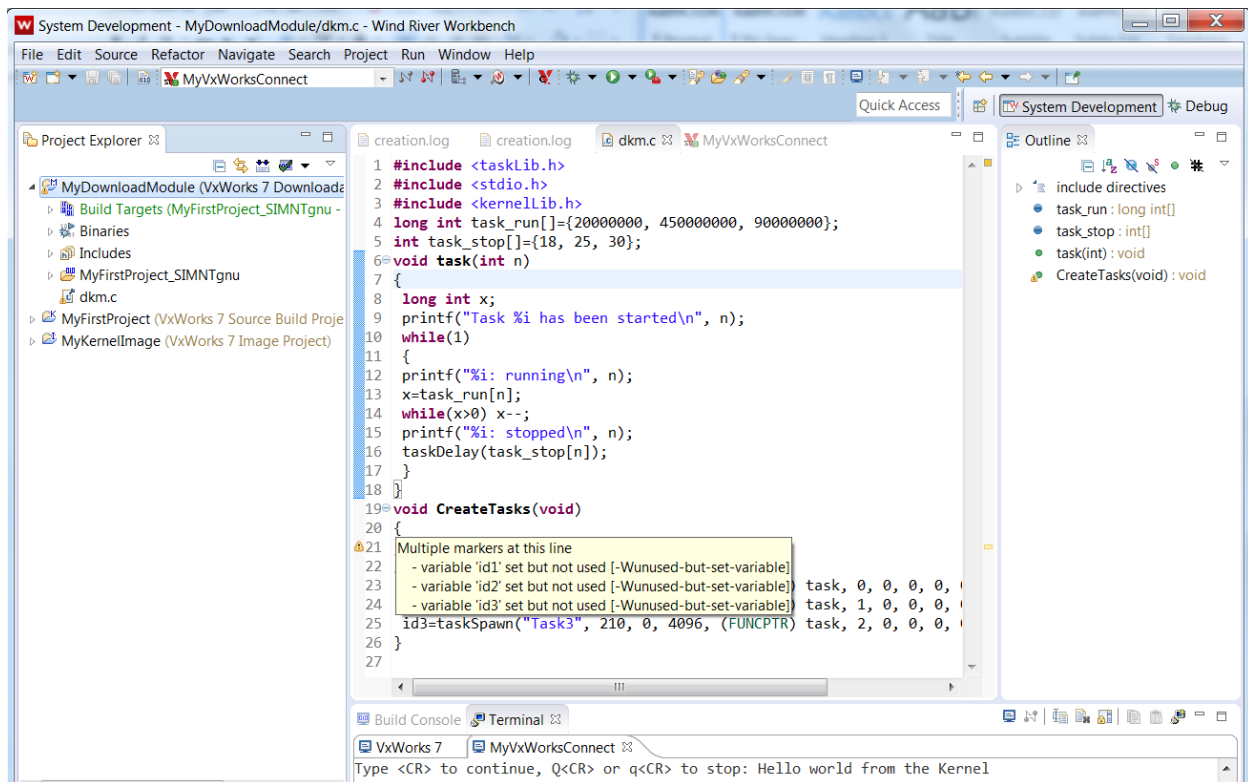
int id1, id2, id3;
/* kernelTimeSlice(1); */
id1=taskSpawn("Task1", 210, 0, 4096, (FUNCPTR) task, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
id2=taskSpawn("Task2", 210, 0, 4096, (FUNCPTR) task, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0);
id3=taskSpawn("Task3", 210, 0, 4096, (FUNCPTR) task, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}

```

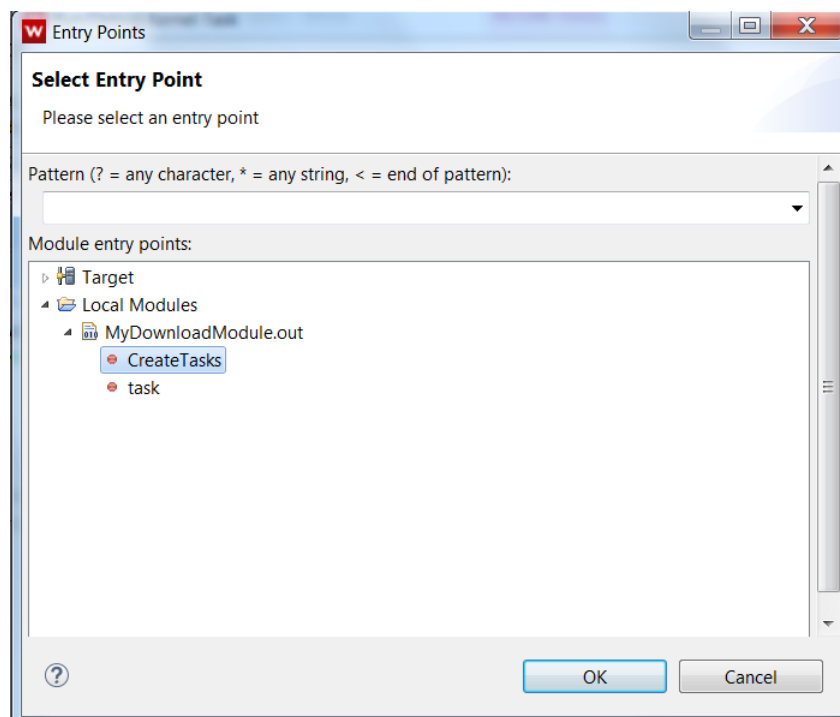
To access the file click the System Development selection in the upper right corner. When you get back to the System Development Perspective select the dkm.c tab in the Code Window. Then paste the code, like this:



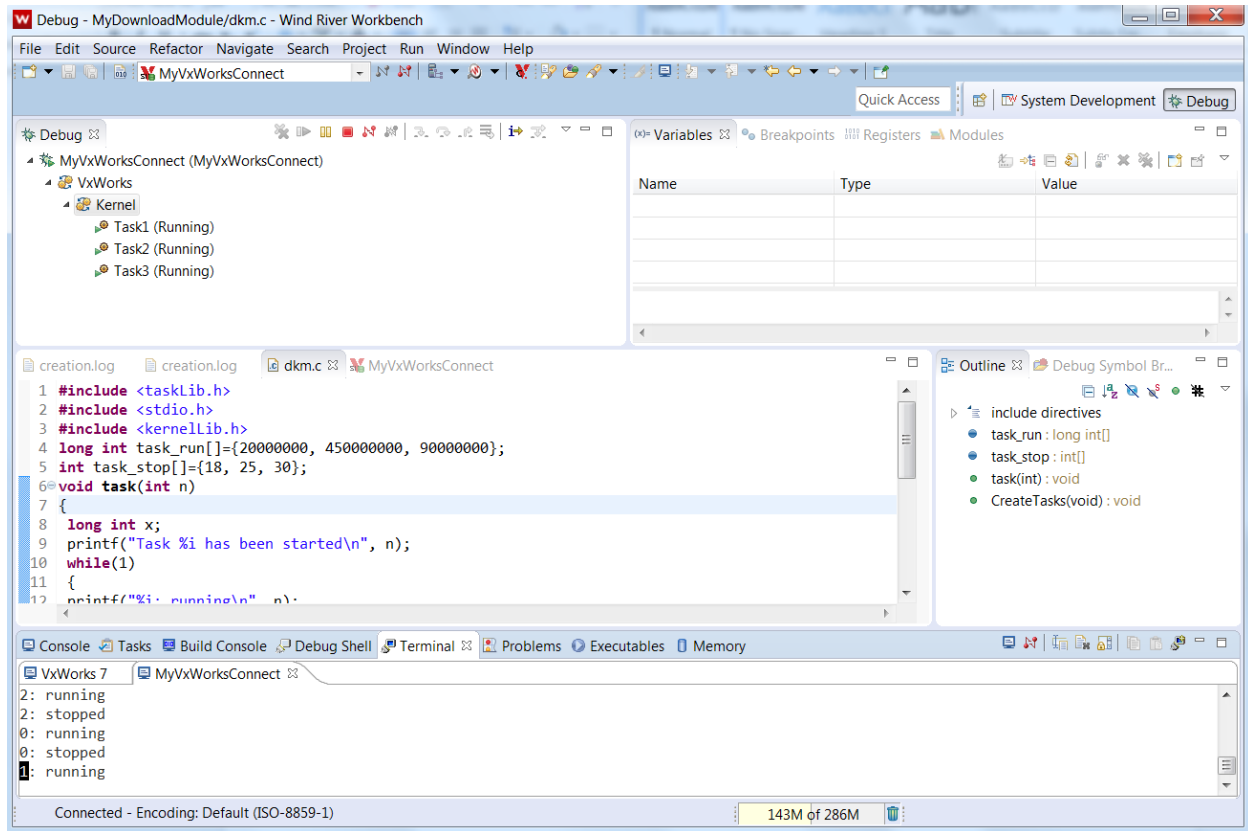
Notice the outline window has changed. Save your file using the File menu. Then build your code again.



Some interesting notes. Notice the yellow warning flag on line 27. These are variables that are declared but not used. To see this, put your mouse button over the warning flag. Now select Run/Debug Kernel Task... again by Right Clicking on your project. But you'll need to select a different function to run, so to do this select the Browse to the right of the entry point and it will show you the different functions that are choices. Select the CreateTasks function:



Then hit OK, then hit OK again.



Notice you are now running three tasks, Task1, Task2, and Task3, and all three are printing out to the console. That's the introduction to VxWorks.

## Lab Submission:

- 1) Simply Submit a screen shot of this final screen.