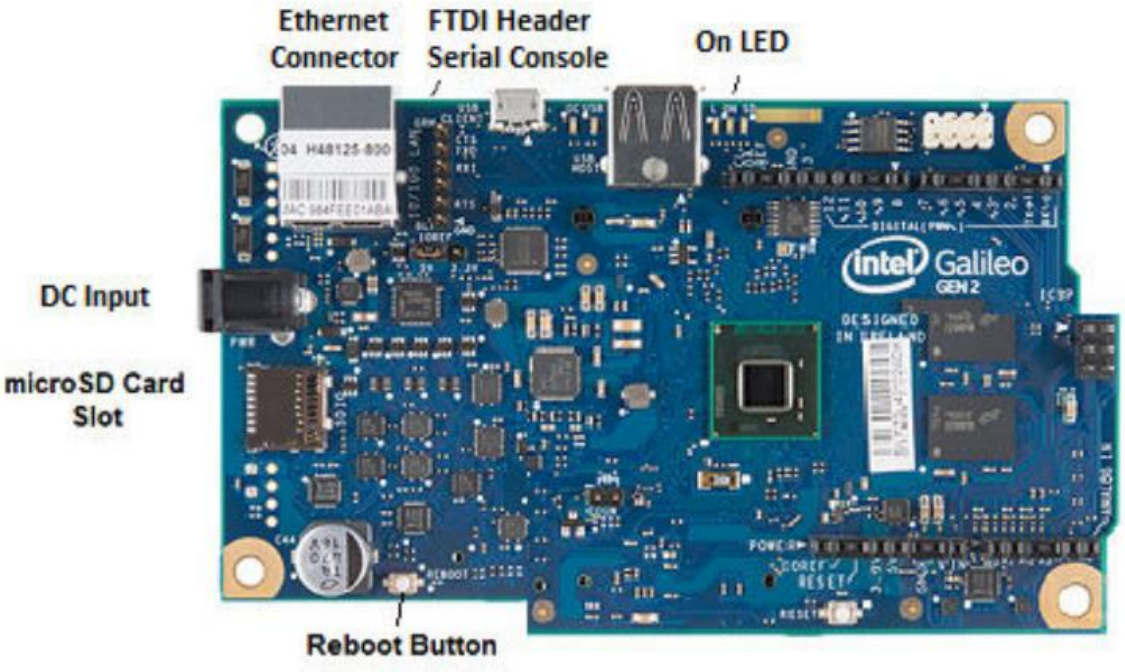
Lab Guide: VxWorks Introduction to the Galileo Target System

Introduction

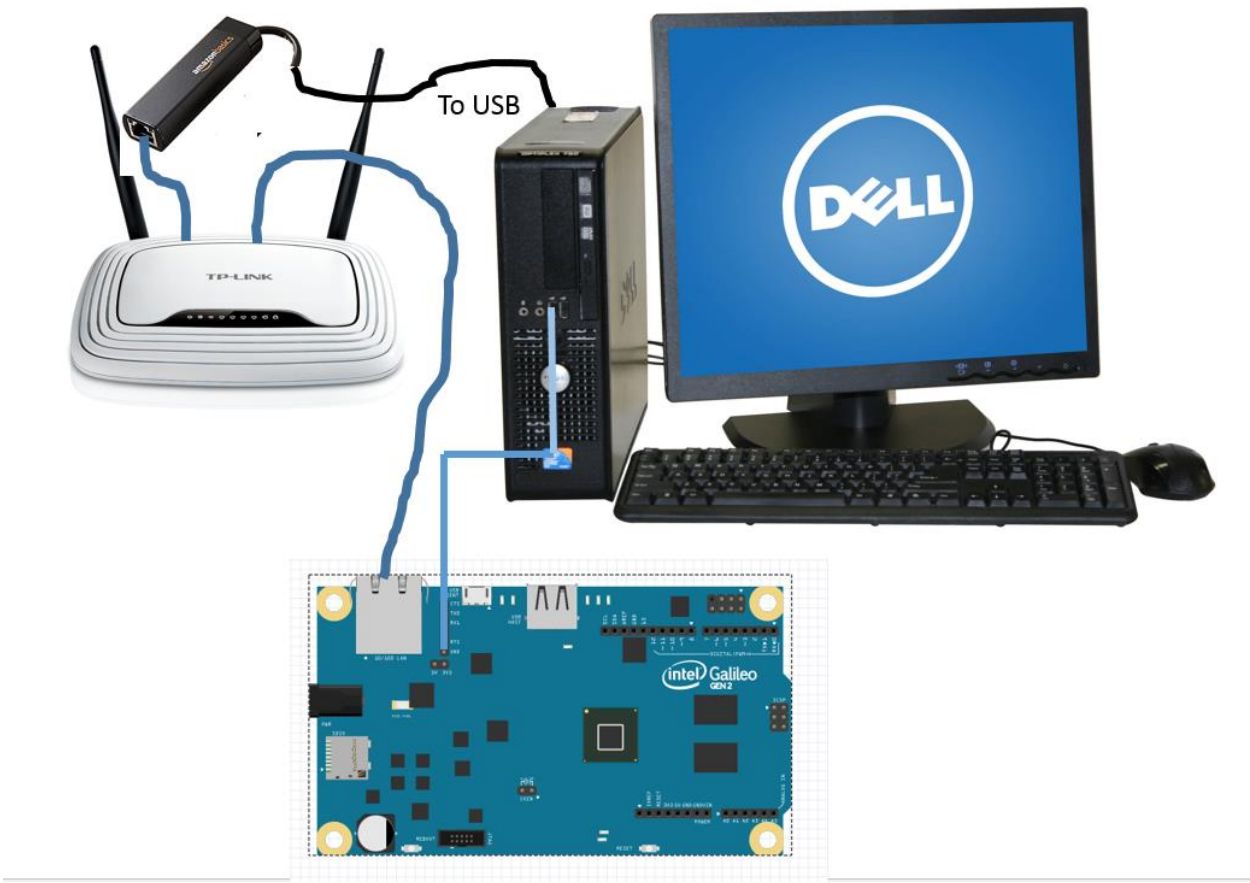
Now that you've complete a couple of labs with the VxWorks system, now you'll connect to the Galileo board to do some real target development.

Constructing the Hardware

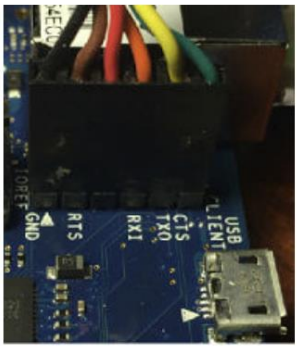
You'll use the Galileo Gen2 board for this lab. Here is the layout of the board:



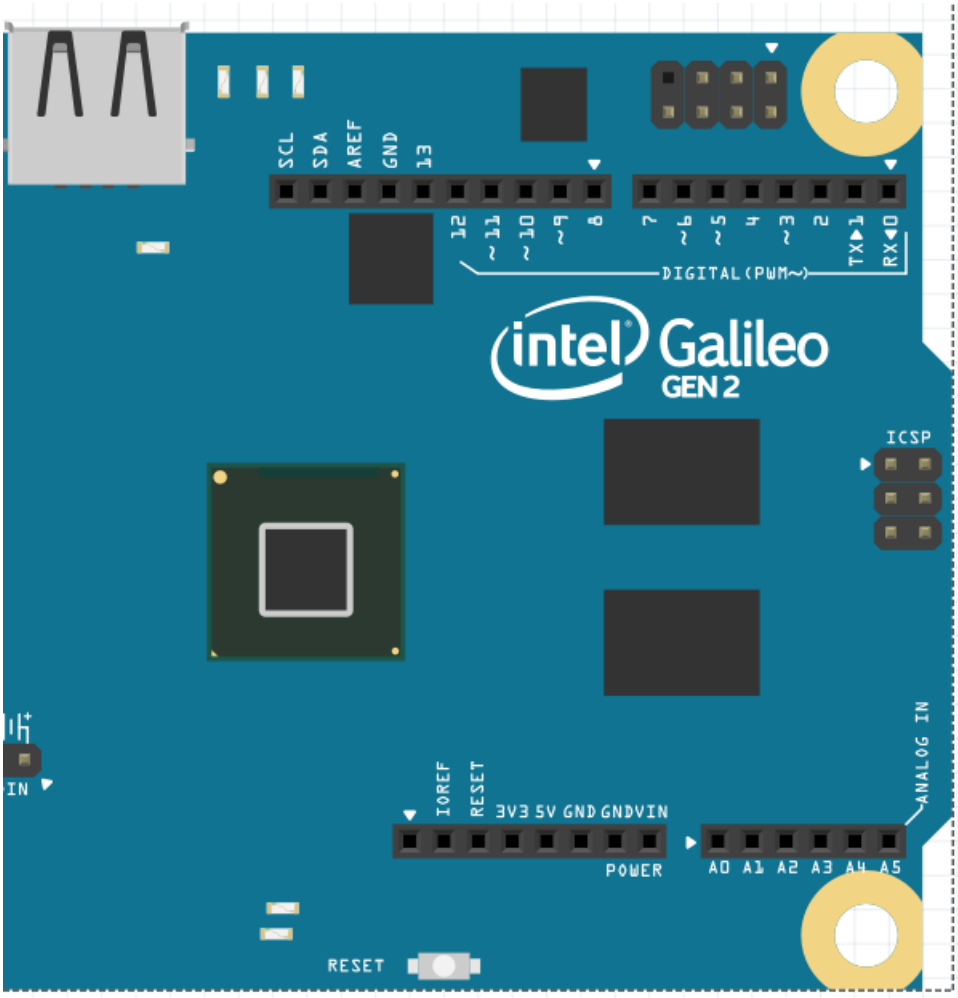
To work with the Galileo board you'll need two connections. The first is the LAN connection. To make this work you'll need to connect your host computer and the Galileo to a router so that both will be served a compatible DHCP address. You'll also need a Serial Console Connection. Here are the complete hardware connections for this lab:



You'll need to connect the FTDI cable to the Galileo, here is a zoom on that connection:

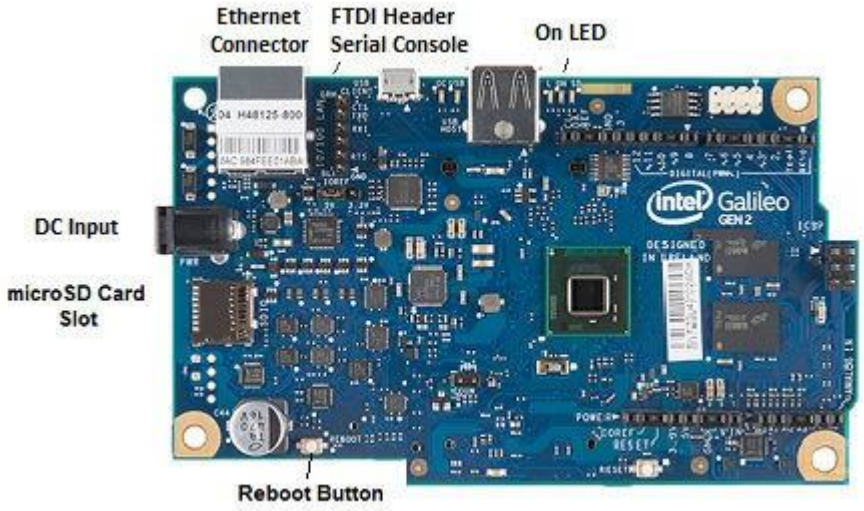


When you run the lab you'll need to drive an LED on D2, Input a digital value on D3, and input an analog value on A1. Here is a close up of the pins on the Galileo (They are the same as for an Arduino):



Connecting and Booting the Board

The onboard interfaces and controls associated with the installation are as follows:



To boot the board and connect to your host, proceed as follows:

Step 1 Connect an Ethernet cable between the board and your network.

Step 2 Connect the FTDI cable between the board and a USB port on your host.

a) Connect the header part of the FTDI cable to the board as shown (black wire aligned with theGND pin):

b) Connect the USB end to your host.

Step 3 Verify that the media card is installed in the card slot of the board.

Step 4 Apply power to the board and verify that the **On** LED is on and that both Ethernet connector LEDs are on or flashing.

Establishing an Ethernet Connection to the Board

Before you can establish a connection to the board over Ethernet, you need the IP address of the board.

You can use the DHCP-assigned IP address acquired by VxWorks or specify a static IP address.

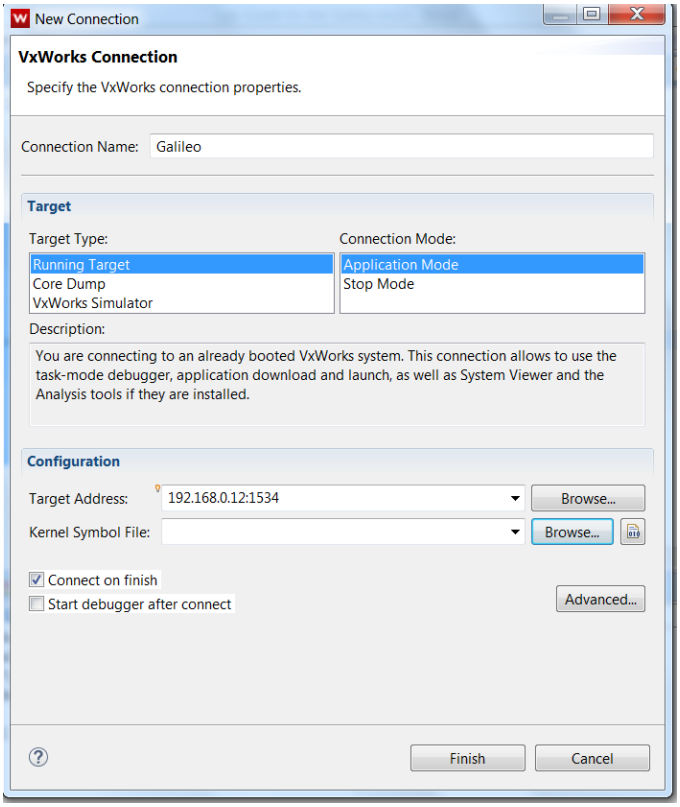
NOTE: For information on assigning a static address to the board, see *Specifying the Boot Parameters for a Static IP Address* on page 31.

To obtain the DHCP-assigned IP address of the board from Workbench, proceed as follows:

Step 1 Launch Workbench.

Step 2 From the Workbench tool bar, select **New Connection**.

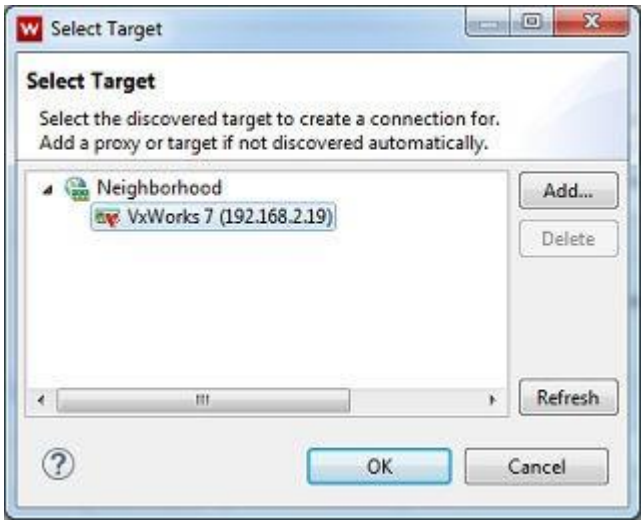
Step 3 Here is the configuration dialog box:



Step 4 Give your Connection a name.

Step 4 On the **Target Address** selection, click **Browse**.

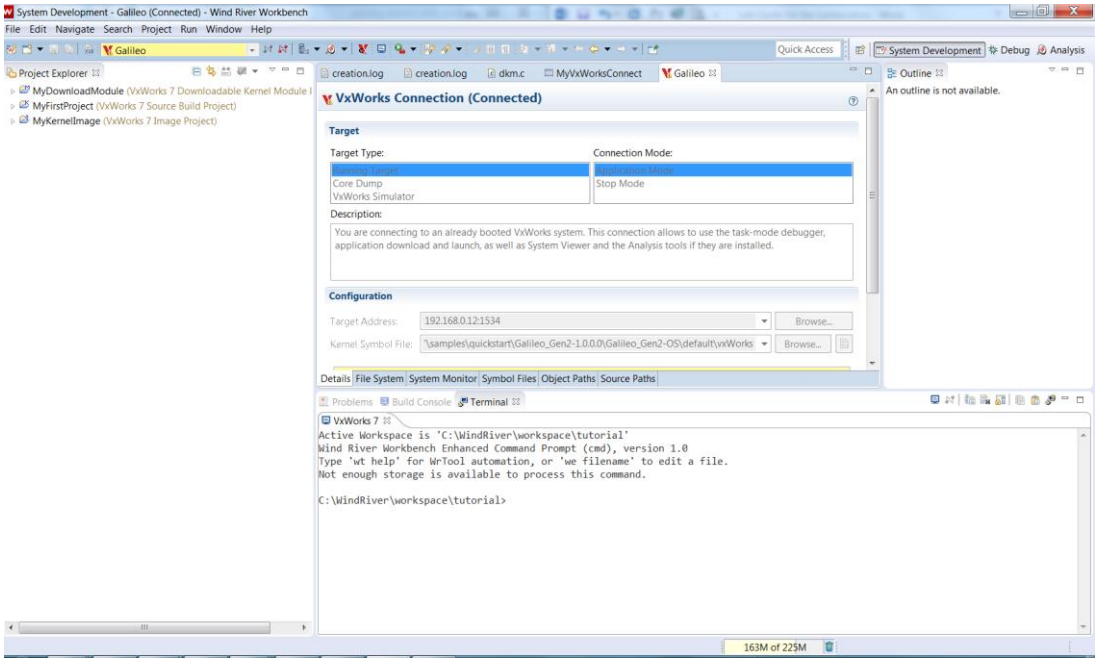
Step 4 From the Select Target dialog, click **Refresh** to discover the target IP address; for example:



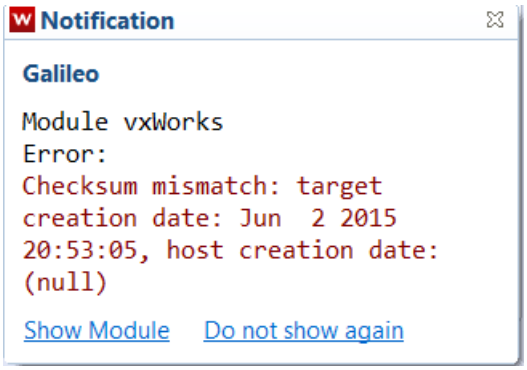
Step 5 If you have obtained the target IP address, you can close the New Connection dialog. If not, proceed to *Establishing a Serial Connection to the Board* on page 11 to obtain the IP address from the serial console.

Step 6 You'll also need to define the Kernel Symbol File. Select the Browse Button and find the file at C:\\WindRiver\vxworks-7\samples\quickstart\Galileo_Gen2-OS\default\vxWorks

Step 7 Click **Finish.** You should see this:



Note: You may get this warning:



This warning will not affect your ability to develop code.

Establishing a Serial Connection to the Board

If the IP address of the board is not automatically discovered by Workbench, you can use a terminal emulator to connect to the board and query the IP address.

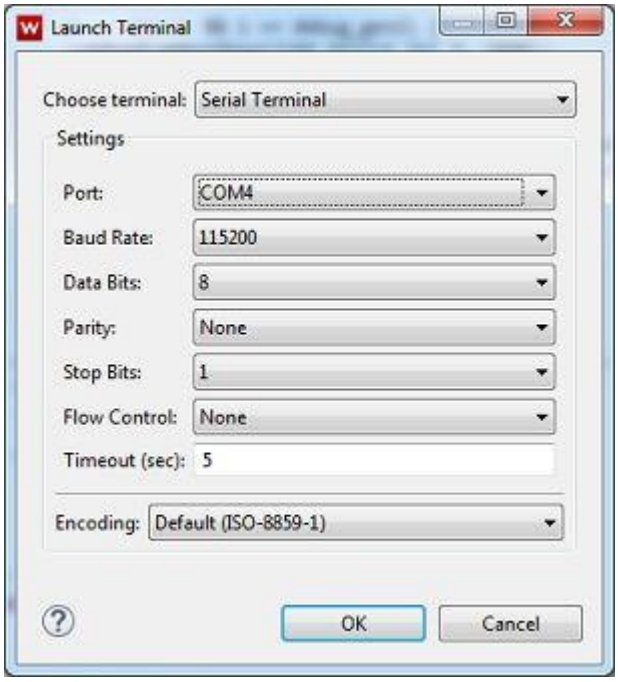
To launch a terminal session from Workbench and obtain the IP address of the board, proceed as follows:

Step 1 If not already done, connect the USB to serial console header cable between the board and your host.

Step 2 From the Workbench tool bar, click the terminal icon.

Step 3 Configure the Launch Terminal dialog settings.

Use the following settings for the interface and set the **Port** identifier to the one you obtained from the Device Manager.



Step 4 Click **OK** and observe the presence of a Terminals window.

You should see the VxWorks 7 kernel prompt from the terminal window; for example:

```
... qrkgmac0        Link type:Ethernet  HWaddr
98:4f:ee:05:4e:c0
        capabilities: TXCSUM VLAN_MTU
        inet6 unicast fe80::9a4f:eeff:fe05:4ec0%qrkgmac0  prefixlen 64  tentative
automatic
        UP SIMPLEX BROADCAST MULTICAST DHCP
        MTU:1500  metric:1  VR:0  ifindex:2
        RX packets:0 mcast:0 errors:0 dropped:0
        TX packets:0 mcast:0 errors:0
collisions:0 unsupported proto:0
RX bytes:0  TX bytes:0

Done executing startup script '/sd0:1/startup.cmd'.
->
```

If the IP address of the **qrkgmac0** link does not appear in the output as shown in the above example, proceed to the next step to obtain the IP address of the target.

Step 5 To display information about the network interfaces of the target, access the VxWorks 7 command shell and use the **ifconfig** command; for example:

```
-> cmd
[vxWorks *]# ifconfig -a lo0
Link type:Local loopback
... qrkgmac0        Link type:Ethernet  HWaddr
98:4f:ee:05:4e:c0
        capabilities: TXCSUM VLAN_MTU
        inet 192.168.2.19  mask 255.255.255.0  broadcast 192.168.2.255
        inet6 unicast fe80::9a4f:eeff:fe05:4ec0%qrkgmac0  prefixlen 64  automatic
        UP RUNNING SIMPLEX BROADCAST MULTICAST DHCP
        MTU:1500  metric:1  VR:0  ifindex:2
        RX packets:8983 mcast:0 errors:0 dropped:13
        TX packets:805 mcast:24 errors:0
collisions:0 unsupported proto:0
        RX bytes:553k  TX bytes:127k
```

Step 6 Note the IP address of the **qrkgmac0** interface. This is the IP address of the Galileo Gen 2 board.

To return to the C interpreter, enter **C** at the prompt.

Step 7 Re-access the Workbench Select Target dialog and click **Add**.

Step 8 Add the IP address in the **Address** field and click **OK** to close the dialog.

Note that the port number is preset to 1534.

Step 9 Click **Refresh** to initiate the discovery process. The target should be discovered.

Developing Code for Your Target System

Creating a Real-Time Process Project

Real-time process (RTP) application projects allow you to manage and build executables that run as real-time processes in user mode.

For this example, you will base the real-time process (RTP) project on the prebuilt VxWorks source build (VSB) project.

NOTE: Sample RTP application projects are available from **File** > **New** > **Example**.

Step 1 From Workbench, select **File** > **New** > **Wind River Workbench Project**.

Step 2 From the **Build type** drop-down menu, select **Real Time Process Application** and click **Next**.

Step 3 Enter a project name, **myRTPproject** for example, and click **Next**.

Step 4 Configure the RTP project options.

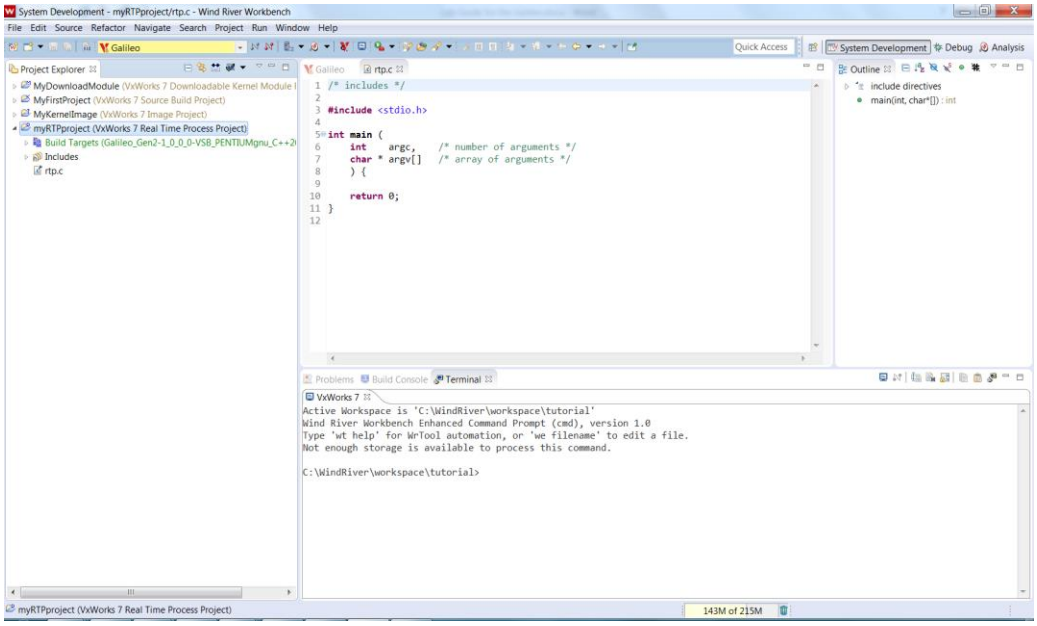
a) From the **Based on** drop-down menu, select **a source build project**.

b) From the **Project** drop-down menu, select **Galileo_Gen2-*x.x.x.x*-VSB (prebuilt)**.

The prebuilt VSB project is in: ***installDir*\vxworks-7\samples\prebuilt_projects\Galileo_Gen2-*x.x.x.x*-VSB**.

Step 5 Click **Finish** to create the RTP project.

The new RTP project appears in the Project Explorer. Workbench automatically creates and opens a stub C source file named **rtp.c** for the project.



The stub C source file resides in the project folder on your host; for example:

***installDir*/workbench-4/workspace/myRTPproject/**

Adding Code to the Real-Time Process Project and Building the

Project

For a simple application, you can add source code to the stub C source file and build the RTP project to create a real-time process application.

Add code to **rtp.c** to create an RTP application that displays:

```
Hello World!

From your RTP application....
```

Step 1 Add a call to **printf( )** to the **rtp.c** file.

Your source code should be as follows:

rtp.c

```
1  /* includes */
2
3  #include <stdio.h>
4
5  int main (
6      int    argc,    /* number of arguments */
7      char * argv[]   /* array of arguments */
8      ) {
9      printf("Hello World!\nFrom your RTP application...\n");
10     return 0;
11 }
```

Step 2 Save the file.

Step 3 To start the build process, right-click the RTP project in the Project Explorer and select **Build Project**.

If this is the first time the RTP project is built, Workbench prompts you to edit the header file search path.

a) Click **Generate Includes**.

b) In the Generate Include Search Paths dialog, select **Ignore all system include directives**. c) Click **Next**, then **Finish**.

d) From the C/C++ Index configuration changed dialog, click **Yes** to builld the RTP project.

e) Observe the build process from the Build Console tab.

If you are not already connected, connect to the Board

You connect to a board to run and debug your application.

Step 1 From the Workbench tool bar, select **New Connection**.

Step 2 Enter **Galileo Gen 2 Connection** as the **Connection Name**.

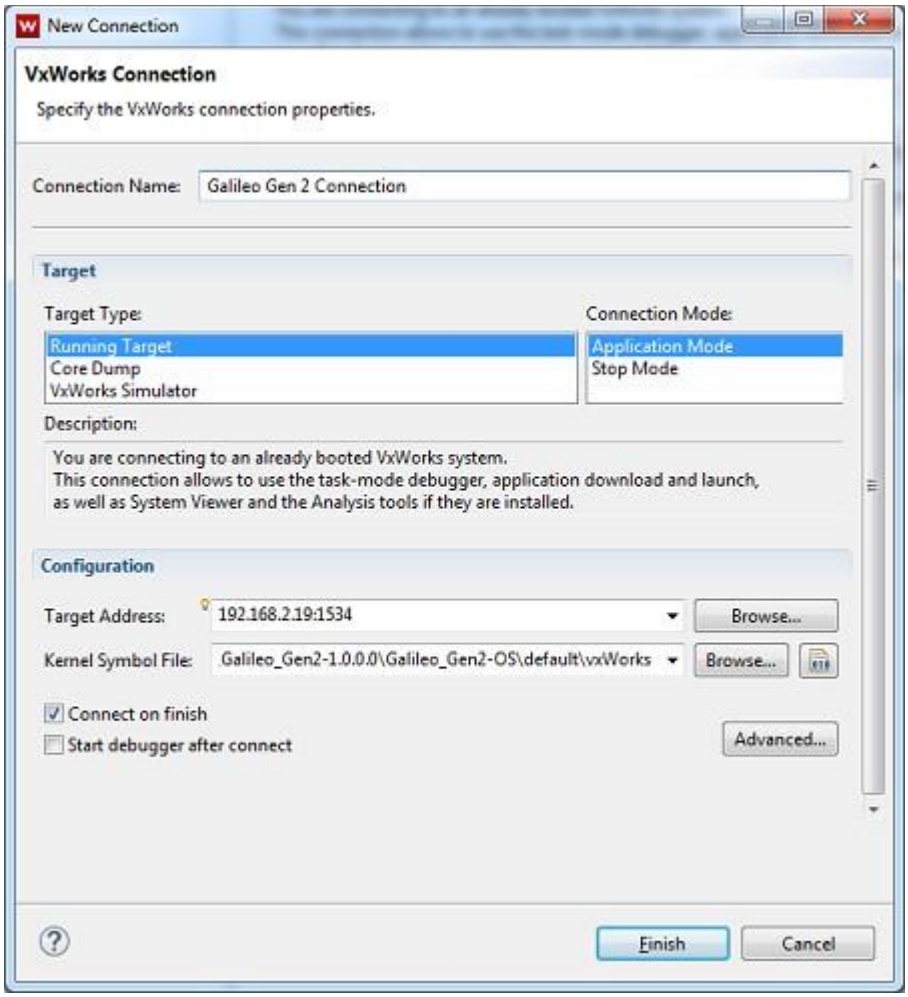
Step 3 From the Target panel, set the **Target Type** to **Running Target** and set the **Connection Mode** to **Application Mode**.

Step 4 From the Configuration panel, set the **Kernel Symbol File** field to the path of the prebuilt kernel image.

The kernel image is at:

***installDir*\vxworks-7\samples\quickstart\Galileo_Gen2-*x.x.x.x*\Galileo_Gen2-OS\default\vxWorks**.

The New Connection dialog should be as follows:



Step 5 Click **Finish** to save the configuration as **Galileo Gen 2 Connection** and connect to the target.

Step 6 Observe that a green indicator is added to the connection name in the Workbench tool bar. This indicates a successful connection to the target. For example:

The Galileo Gen 2 Connection tab appears.

The tabs at the bottom of the VxWorks Connection window display information relevant to the active session.

Details

Shows the Target and Configuration panels.

System Monitor

Allows browsing the target system contexts (such as kernel tasks and RTPs).

Symbol Files

Allows browsing the symbol files currently associated with each memory context. If a symbol file was not resolved automatically, you can locate the right symbol file by clicking **Locate Symbol File**.

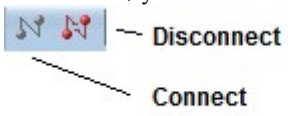
Object Paths

Contains the path mapping rules applied to the target module. These mappings are used by the tools to automatically locate the symbol files for a given target module (DKM or RTP). Some entries will be generated automatically as a result of the kernel symbol file location work-flow, or any symbol file *locate* actions.

Source Paths

Contains the path mapping rules applied to locate source code. These mappings are used by the tools to locate the source files on the host from the path information embedded in the debug information.

Step 7 If the **Connect on finish** checkbox is unchecked when you clicked **Finish** from the Galileo Gen 2 Connection dialog, you can manually connect to the target by clicking the **Connect** button. In either case, you can disconnect from the target by clicking the **Disconnect** button.

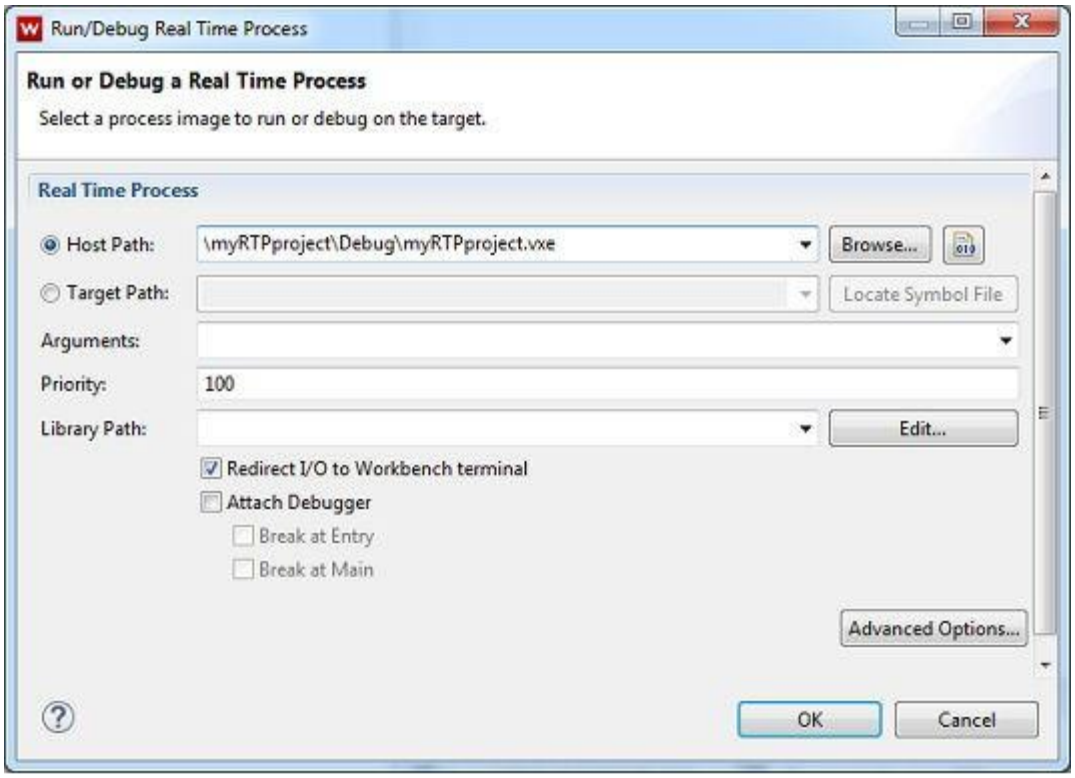


Loading and Running the Real-Time Process Application

Once you have connected to the target, you can load and run your real-time process (RTP) application using Workbench.

Step 1 In the Project Explorer, right-click your real-time process (RTP) project and select **Run/Debug Real Time Process** to load and run the application.

The Run/Debug Real Time Process dialog appears.



Step 2 Click **OK**.

The process loads and executes the application in user space. The terminal output is as follows:

```
Hello World!
From your RTP application...
```

When the application exits, the process terminates.

Creating and Running Your First Downloadable Kernel Module Application

Creating a Downloadable Kernel Module Project

Downloadable kernel module (DKM) projects allow you to manage and build kernel application modules that execute in kernel space.

For this example, you will base the downloadable kernel module (DKM) project on the prebuilt VxWorks source build (VSB) project.

NOTE: Sample kernel application projects are available from **File** > **New** > **Example**.

Step 1 From Workbench, select **File** > **New** > **Wind River Workbench Project**.

Step 2 From the **Build type** drop-down menu, select **Downloadable Kernel Module** and click **Next**.

Step 3 Enter a project name, **myDKMproject** for example, and click **Next**.

Step 4 Configure the DKM project options.

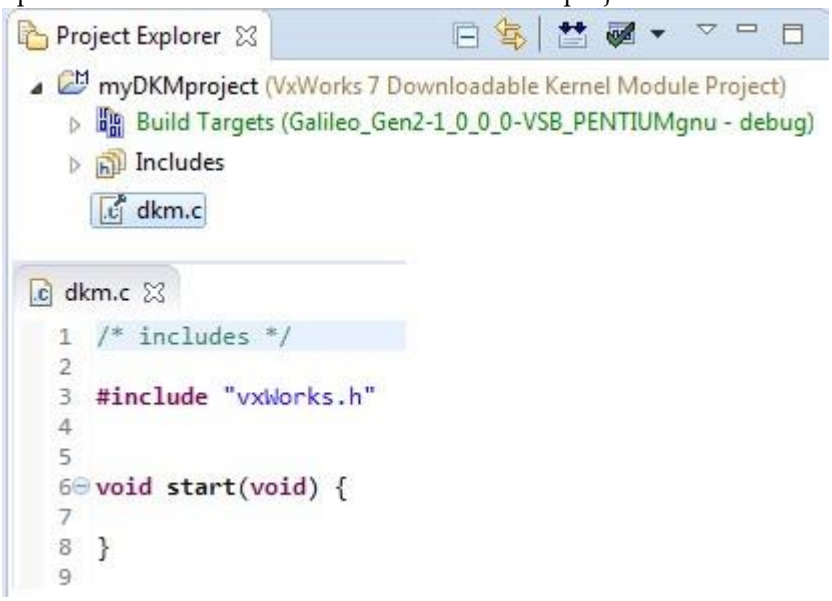
a) From the **Based on** drop-down menu, select **a source build project**.

b) From the **Project** drop-down menu, select **Galileo_Gen2-*x.x.x.x*-VSB (prebuilt)**.

The prebuilt VSB project in: ***installDir*\vxworks-7\samples\prebuilt_projects\Galileo_Gen2-*x.x.x.x*-VSB**.

Step 5 Click **Finish** to create the DKM project.

The new DKM project appears in the Project Explorer. Workbench automatically creates and opens a stub C source file named **dkm.c** for the project.



The stub C source file resides in the project folder on your host; for example:

***installDir*/workbench-4/workspace/myDKMproject/**

Adding Code to the Downloadable Kernel Module Project

For a simple application, you can add source code to the stub C source file and build the DKM project to create a real-time process application.

To add existing code to the **dkm.c** source file from the example code, proceed as follows:

Step 1 Delete the existing code from the stub C source file. You should have an empty file.

Step 2 Add the code below to the **dkm.c** file. The code in the **dkm.c** file should start as follows (after the initial comment block):

```
/* gg2GpioTest.c
...
*/
...
#include "vxWorks.h"
#include <vxBusLib.h>
#include <hwif/util/vxbAdcLib.h>
#include <hwif/i2c/vxbQrkI2c.h>
#include <hwif/gpio/vxbNxpPCA95xx.h>
#include <hwif/gpio/vxbQrkGpio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dosFsLib.h>
#include <hwif/vxbus/vxBus.h>
#include <isrLib.h>
#include <private/isrLibP.h>
#include <vxBusLib.h>
#include <taskLib.h>
#include
<hwif/util/vxbParamSys.h>
#include <hwif/pwm/vxbPca9685.h>
...
```

Step 3 Save the file.

Step 4 To start the build process, right-click the DKM project in the Project Explorer and select **Build Project**.

If this is the first time the DKM project is built, Workbench prompts you to edit the header file search path.

a) Click **Generate Includes**.

b) In the Generate Include Search Paths dialog, uncheck **Ignore all system include directives**. c) Click **Next**, then **Finish**.

d) From the C/C++ Index configuration changed dialog, click **Yes** to builld the DKM project.

e) Observe the build process from the Build Console tab.

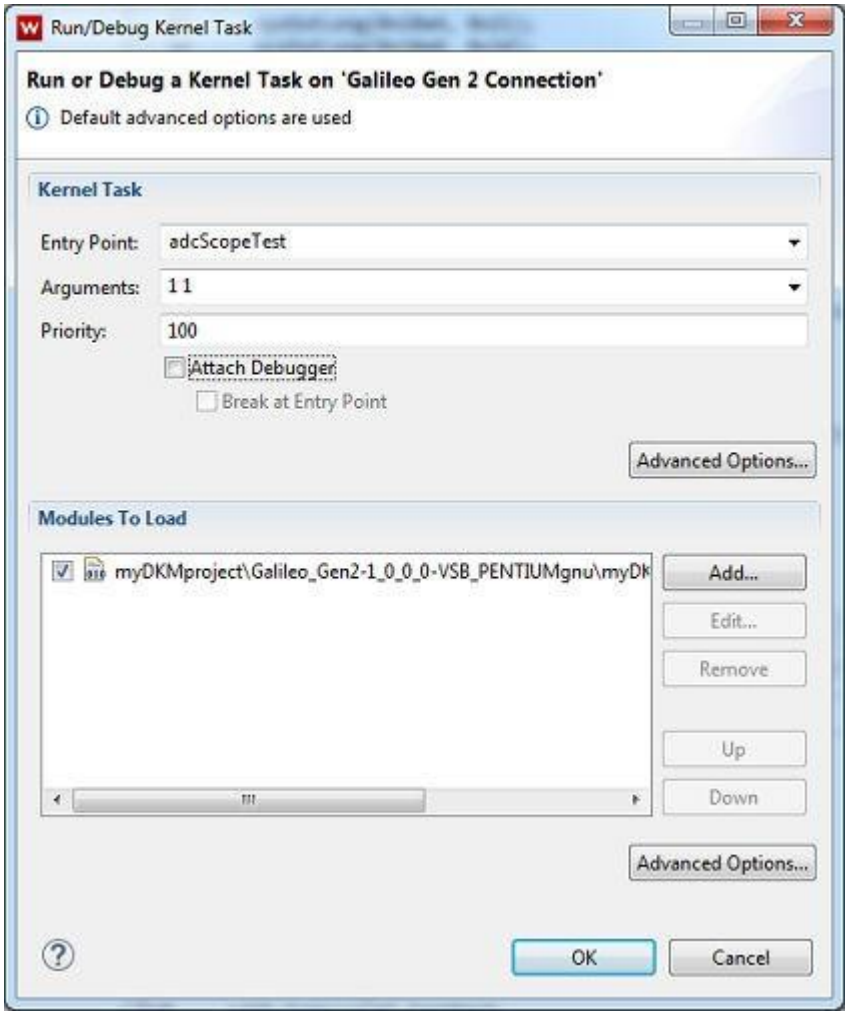
Loading and Running Tasks from a Downloadable Kernel Module Application

Once the code is added to a downloadable kernel module (DKM) application, you can simply connect to the target and spawn a task from Workbench.

Step 1 If not connected to the target, follow the procedure in *Connecting to the Board* on page 17.

Step 2 In the **Project Explorer**, right-click your DKM project and select **Run/Debug Kernel Task**.

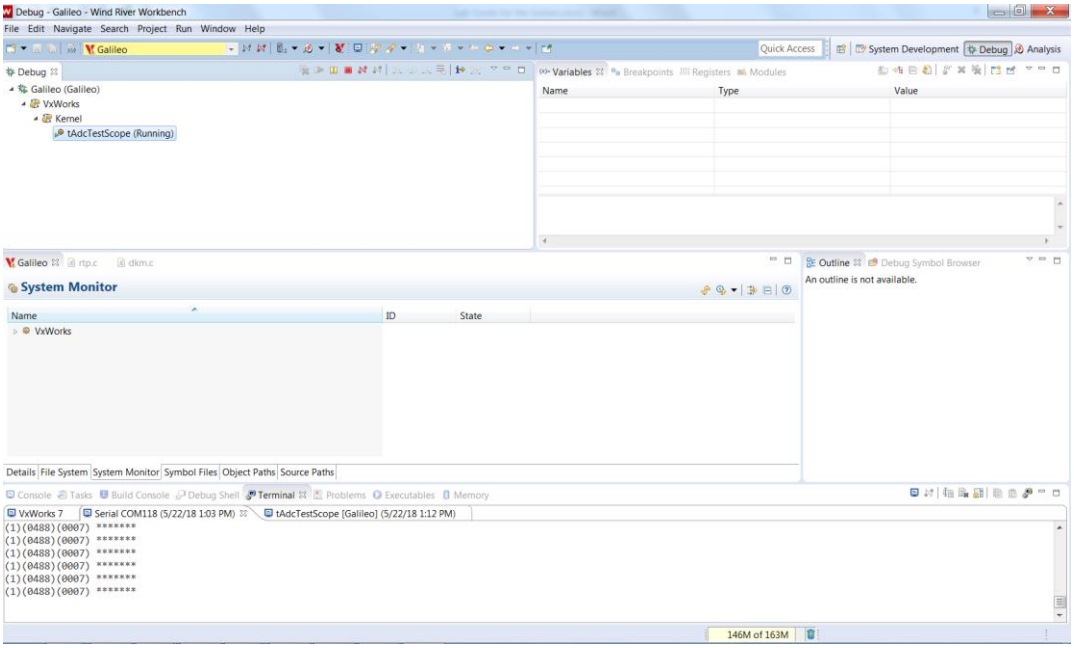
Step 3 From the Run/Debug Kernel Task dialog, enter the name of the function in the **Entry Point** field and enter the arguments in the **Arguments** field; for example:



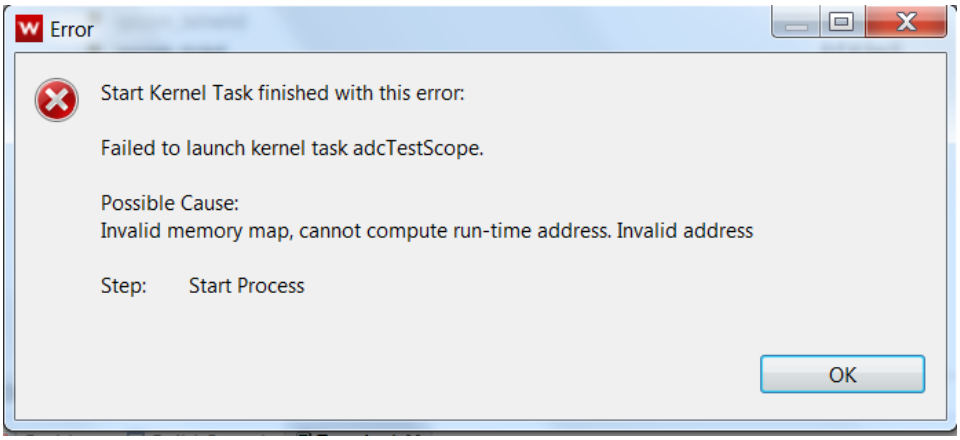
Step 4 Click **OK** to load and run the function as a task.

The task runs in kernel space.

Running this code should result in you seeing this:



Note: If you see this error:



You will need to reset the board before debugging.

Example Code for Working with Inputs and Outputs

Input and Output Functions

The example code contains functions that allow you to test a number of input and output ports on the board.

The function prototypes and their descriptions are as follows:

- **void adcScopeTest(int channel, int show)**:

Enables an analog input to be read from the designated channel.

- **void pcaOutputTest(void)**:

Changes the state of digital output pin **D2** every 500 milliseconds over a 10 second period.

- **void pcaInputTest(void)**:

Reads the state of digital input pin **D3** every 500 milliseconds and prints the state of the input over a 10 second period.

The example code in *Example Code for the Input and Output Functions* includes comments that describe the functions and their operation.

Example Code for the Input and Output Functions

The example code to test input and output ports is as follows:

```
/* gg2GpioTest.c */
/*  Copyright (c) 2015 Wind River Systems, Inc.
*
*       The right to copy, distribute, modify or otherwise make use
*       of this software may be licensed only pursuant to the terms
*       of an applicable Wind River license agreement. */
/* includes */
#include "vxWorks.h"
#include <vxBusLib.h>
#include <hwif/util/vxbAdcLib.h>
#include <hwif/i2c/vxbQrkI2c.h>
#include <hwif/gpio/vxbNxpPCA95xx.h>
#include <hwif/gpio/vxbQrkGpio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dosFsLib.h>
#include <hwif/vxbus/vxBus.h>
#include <isrLib.h>
#include <private/isrLibP.h>
#include <vxBusLib.h>
#include <taskLib.h>
#include <hwif/util/vxbParamSys.h>
#include <hwif/pwm/vxbPca9685.h>


IMPORT VOID   vxbNxpPca95xxShow(VXB_DEVICE_ID pDev, int verbose);
IMPORT STATUS vxbAdcCtrlSet (UINT16 chanMask, BOOL readTemp, BOOL repeatRead);
IMPORT STATUS gpioOutputWrite (VXB_DEVICE_ID pDev, UINT32 pin, int val);
IMPORT UINT32 sysInLong (UINT32 address);
IMPORT VOID   sysOutLong (UINT32 address, UINT16 data);
IMPORT STATUS vxbAdcDataRead (UINT16 chanMask, BOOL getTemp, UINT16 * buf);

/* Set to 1 if you require debug messages */
BOOL debug_gen2 = 1;
BOOL d2Init = 0;
BOOL d3Init = 0;
/*
          *    Invoke by typing
          *    ->sp adcTestScope(N, 1)
          *    where N = analog port number to monitor
 *
          *    For example, a temperature sensor attached to analog port 1 can be scoped by
          typing
          *    ->sp adcScopeTest(1,1)
 */
void adcTestScope(int channel, int show) {
    UINT16 adcTestBuf[9];
    int pin = 0; /* used as entry into buffer */
    int i;
    int len1, len2;
    pin = 0;
    sysOutLong(0x10a4, 0x21);
```

```
     sysOutLong(0x10a8, 0x3d);
     gpioOutputSelect(0, 0, 0);
     gpioOutputWrite(0, 0, 1); /*set as output*/
     vxbAdcCtrlSet(1 << channel, 0, 0);
     for (;;)
        {
        taskDelay(10);
         vxbAdcCtrlSet(1 << channel, 0, FALSE);
        vxbAdcDataRead(1 << channel, 0, adcTestBuf);
         /* shift by 6 to fit display into 80 columns wide */
         len1 = (adcTestBuf[pin]);
         len2 = len1 >> 6;
         printf("(%d)(%04d)(%04d) ", channel, len1, len2);
         if (show)
             {
             for (i = 0; i < len2; i++)
                 {
                 printf("*");
                 }
             }
         printf("\n");
     }
}
/*
 * Helper function that writes value to D2 pin
 */
LOCAL STATUS digitalWrite2(UINT32 value)
    {
    VXB_DEVICE_ID pDev2;
    pDev2 = vxbInstByNameFind("nxpPcal9535", 2);
    if (pDev2 == NULL)
        {
        if (debug_gen2) printf("ERROR: could not find EXP2 \n");
        return ERROR;     }
    /* EXP2 P1_5 is the MUX pin for Arduino Digital 2 */
       if (0 == d2Init && OK != vxbNxpPca95xxPadConf(pDev2, NXP_PCA95XX_P1_5,
                                    NXP_PCA95XX_CONFIG_OUTPUT,
                                    NXP_PCA95XX_FLAG_DRIVE_STRENGTH_100 |
                                    NXP_PCA95XX_FLAG_PULLUP_PULLDOWN_EN_NO,
                                    1))
          {
          if (debug_gen2) printf("ERROR: could not set output on D2\n");
          return ERROR;
          }
         if (0 == d2Init && 1 == debug_gen2)
          {
          vxbNxpPca95xxShow((VXB_DEVICE_ID) 2, 1000);
          }

    d2Init = 1;

    if (OK != vxbNxpPca95xxOutputPinSet(pDev2, NXP_PCA95XX_P1_5, value))
        {
        if (debug_gen2) printf("ERROR: could not write on D2\n");
        return ERROR;
        }
    return OK;
}
/*
      * Connect an LED to D2 on your Galileo with the appropriate resistor
      * and spawn this task to see an LED connected to Arduino D2 flash 10 times
      * -> pcaOutputTest
 */
void pcaOutputTest(void)
        {
        UINT32 i;
        /* flash the LED on IO2, 10 times */
        printf("Flashing LED on Digital 2 now...\n");
        for (i = 0; i < 10; i++)
               {
```

```
                digitalWrite2(1);
                taskDelay(30);
                digitalWrite2(0);
                taskDelay(30);
                }
        }
/*
 * Helper function to read D3
 */
LOCAL STATUS digital3Read(UINT32 *value)
        {
        VXB_DEVICE_ID pDev2;
        pDev2 = vxbInstByNameFind("nxpPcal9535", 2);
        if (pDev2 == NULL)
                {
                if (debug_gen2) printf("ERROR: could not find EXP2 \n");
                return ERROR;
                 }
    /* EXP2 P1_6 is the MUX pin for Arduino Digital 3 */
        if (0 == d3Init && OK != vxbNxpPca95xxPadConfMasked(pDev2,
                                NXP_PCA95XX_P1_6,

                                NXP_PCA95XX_CONFIG_INPUT,

                                NXP_PCA95XX_FLAG_POLARITY_INVERT_NO |
                                NXP_PCA95XX_FLAG_PULLUP_PULLDOWN_EN_YES |
                                NXP_PCA95XX_FLAG_PULLUP_PULLDOWN_SEL_DN |
                                NXP_PCA95XX_FLAG_LATCH_YES,

                                NXP_PCA95XX_FLAG_POLARITY_INVERT_MASK |
                                NXP_PCA95XX_FLAG_PULLUP_PULLDOWN_EN_MASK |
                                NXP_PCA95XX_FLAG_PULLUP_PULLDOWN_SEL_MASK |
                                NXP_PCA95XX_FLAG_LATCH_MASK,

                                0))
        {
         if (debug_gen2) printf("ERROR: could not set input on D3\n");
        return ERROR;
        }
    if (0 == d3Init && 1 == debug_gen2)
        {
         vxbNxpPca95xxShow((VXB_DEVICE_ID) 2, 1000);
        }
    d3Init = 1;
    if (OK != vxbNxpPca95xxInputPinGet(pDev2, NXP_PCA95XX_P1_6, value))
        {
        if (debug_gen2) printf("ERROR: could not read input on D3\n");
            return ERROR;
       }
    return OK;
 }
/*
    *   Connect a switch to D3 using the appropriate resistor and spawn to
    *   read value 10 times
    *   -> pcaInputTest
    *   The output at each sample interval should be
    *   EXP2 inputs = 0x00000000
    *   or
    *   EXP2 inputs = 0x00000001
    *
 */
void pcaInputTest(void)
    {
    int i;
    UINT32 value = 0;
        /* read values on EXP2, 20 times sampling every half second */
    for (i = 0; i < 20; i++)
        {
         if (OK == digital3Read(&value))
```

```
        {
        printf("EXP2 inputs = 0x%08x\n", value);
        }
      taskDelay(30);
     }
}
```

Lab Submission

1) Using the code provided, run the pcaOutputTest() function and flash an LED on Pin 2 using the appropriate resistor in the circuit.

2) Using the code provided, run the pcaInputTest() function and input a voltage from 0 to 5 volts on EXP2, to read a digital input.

3) Using the code provided, run the adcScopeTest() function to measure an input sine wave.