Xilkernel Configuration and Application Tutorial

Using the Xilinx toolchain to build and deploy a multi-threaded application on the Xilkernel RTOS

Contents

Introduction

This tutorial will guide you through the process of configuring a PowerPC-based system for the Xilinx ML507 development board with the minimalistic, POSIX-compliant Xilkernel real-time operating system in order to run and debug a basic multi-threaded demo application.

After completing this tutorial you will be able to:

1. Create a Xilinx project for the ML507 development board that is configured to use the hard-core PowerPC processor and processor local bus (PLB) hardware interface.
2. Create a Xilkernel RTOS board support package for a specific FPGA system configuration.
3. Write and debug a multi-threaded application that runs on top of the Xilkernel RTOS.

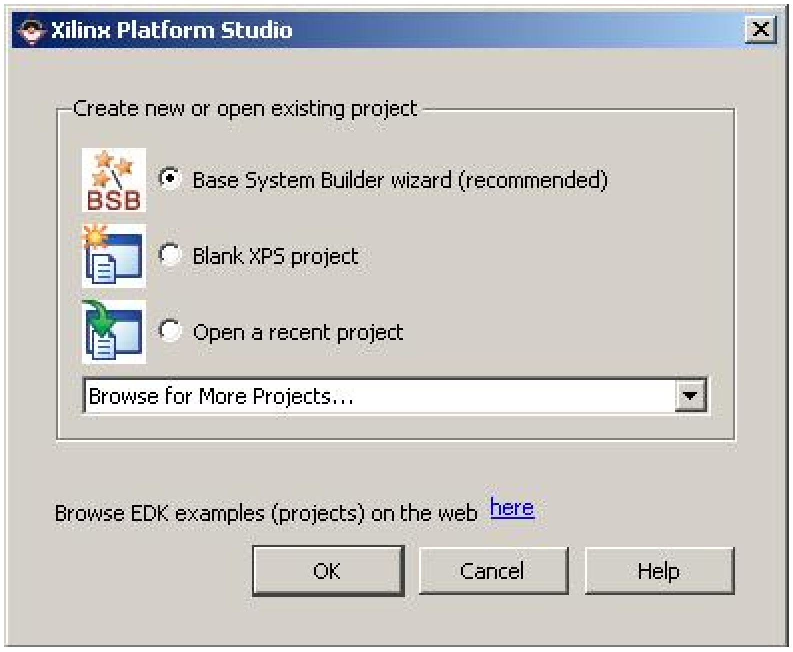
Resources

To complete this tutorial you will need the following resources:

- Xilinx ML507 development board with a Virtex-5 FPGA.
- Platform cable USB II adapter (USB-to-JTAG).
- Licensed copy of version 12.3 of the Xilinx tool suite, including the Xiling Platform Studio (XPS) and Embedded Development Kit (EDK).

Create a PowerPC-Based System Configuration

1. Open the Xilinx Platform Studio tool (version 12.3). When the application is finished loading all of its resources you will be prompted with the following dialog.



[✓ Base System Builder wizard (recommended)]

Blank XPS project

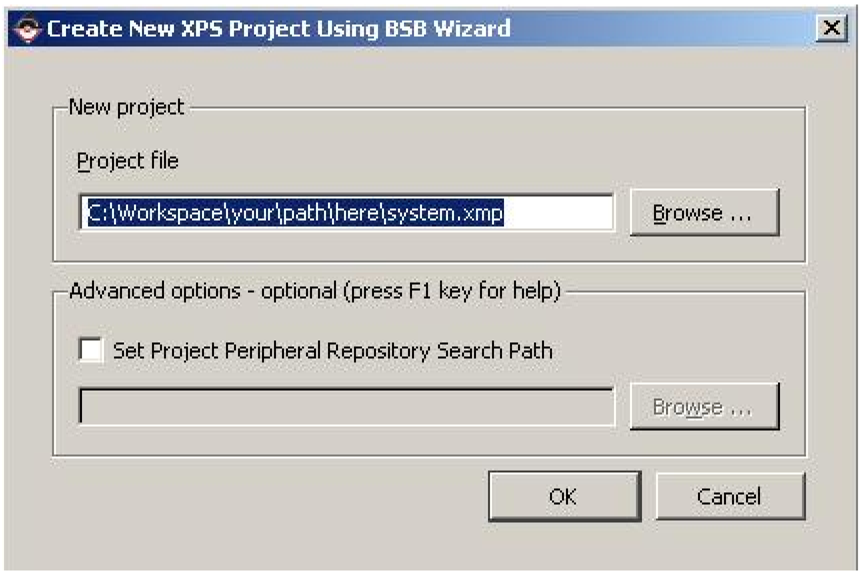
Open a recent project

Browse for More Projects...

Browse EDK examples (projects) on the web here

Make sure the “Base System Builder wizard (recommended)” bubble is filled in and then click on “OK”.

2. When the “Create New XPS Project Using BSB Wizard” window is displayed, browse to the directory where you would like to store the new system configuration and all of the associated project files. The path should be similar to the following:

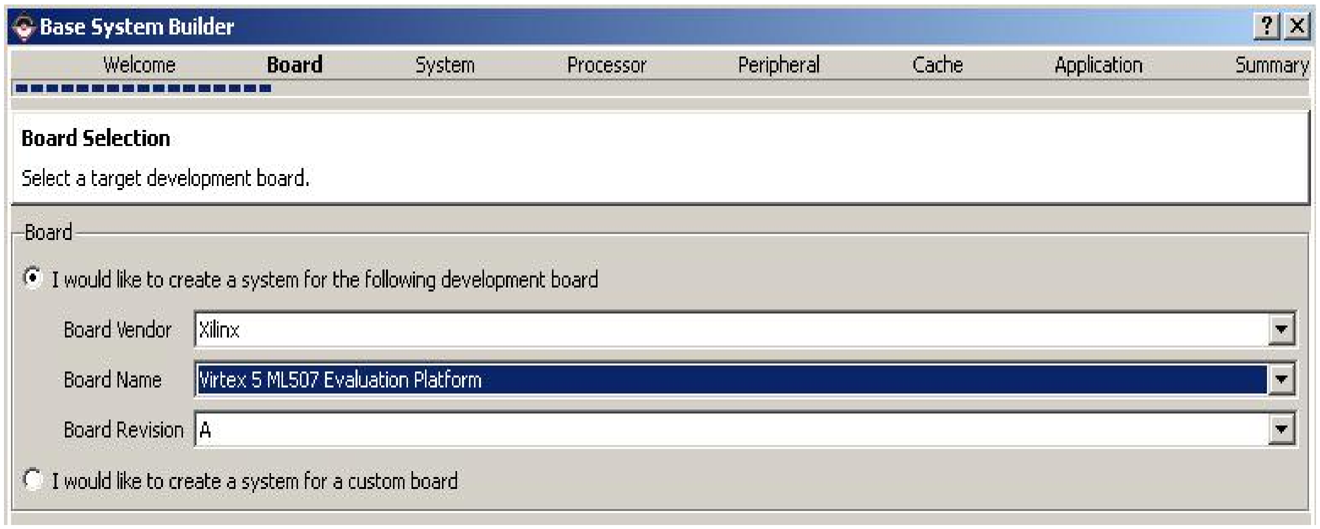

Create New XPS Project Using BSB Wizard
New project
Browse ...
Advanced options - optional (press F1 key for help)
Browse ...
OK
Cancel

Project file: C:\Workspace\your\path\here\system.xmp

Set Project Peripheral Repository Search Path [ ]

When finished, click on “OK”.

1. When the next “Base System Builder” dialog window appears, make sure the “PLB system” bubble is filled in and then click “OK”.
2. You will now be prompted by the initial “Base System Builder” wizard dialog box. Make sure the “I would like to create a new design” bubble is filled in and then click “Next”.
3. In the next window you will need to select “**Xilinx**”, “**Virtex 5 ML507 Evaluation Platform**”, and “**A**” as the board **vendor**, **name**, and **revision**, respectively. The following image depicts this configuration.

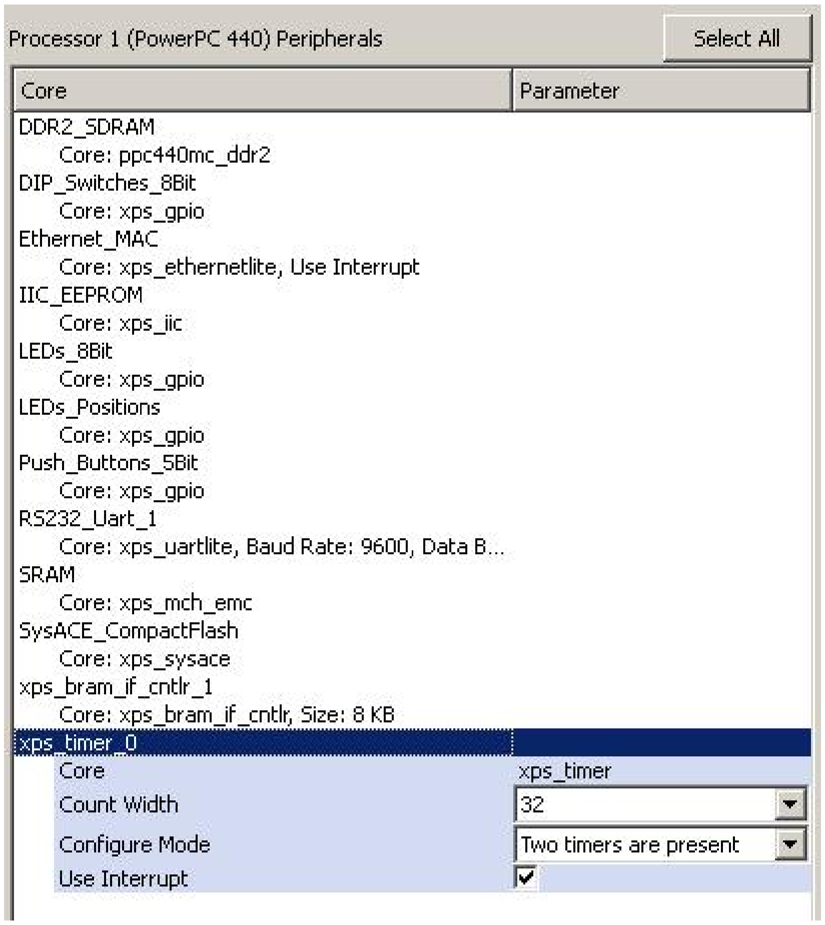


Board Selection
Select a target development board.

Board
(●) I would like to create a system for the following development board
Board Vendor: Xilinx
Board Name: Virtex 5 ML507 Evaluation Platform
Board Revision: A
( ) I would like to create a system for a custom board

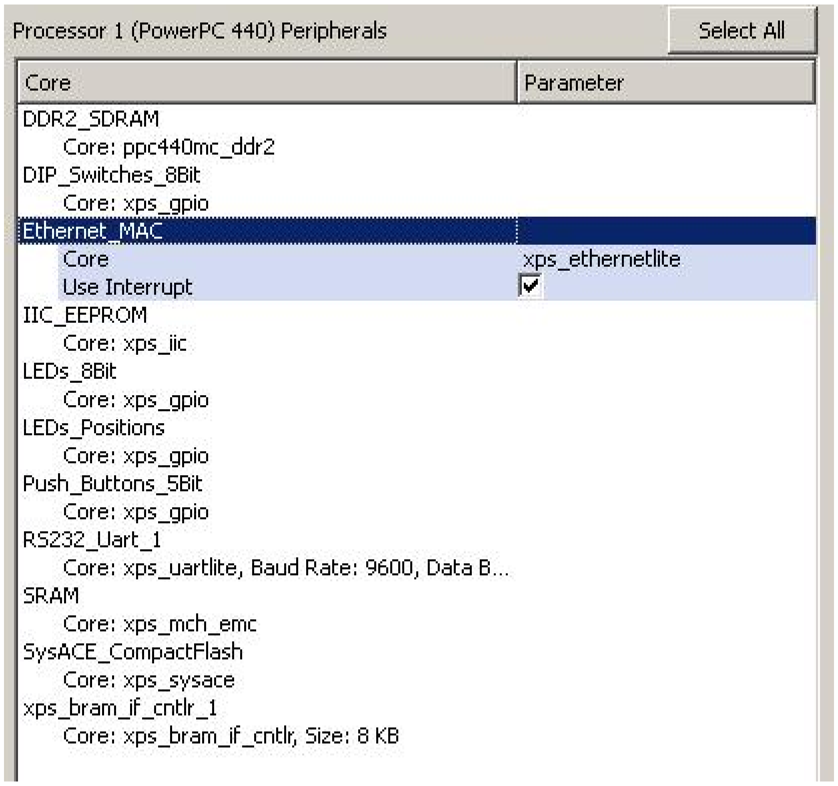
When these parameters are configured correctly, click “Next”.

1. In the next window make sure the “Single-Processor System” bubble is filled in. For the purposes of this exercise and tutorial, we will not need more than one processor to perform computations, so having one is sufficient. When this is complete, click “Next” to proceed.
2. Leave the “Processor Configuration” settings at their defaults and click “Next”. Notice that the processor type we are working with is the PowerPC. The Xilinx toolchain is flexible enough in that it allows us to select either the PowerPC or the MicroBlaze processors to use when running the Xilkernel.
3. From the “Peripheral Configuration” window, remove the **PCIe_Bridge** and **RS232_Uart_2** components from the Processor 1 (PowerPC M440) peripheral list. Add a **xps_timer** instance to the peripheral list and check the “Use Interrupt” box under the newly added **xps_timer_0** peripheral instance, as shown below.



Core	Parameter
DDR2_SDRAM	
Core: ppc440mc_ddr2	
DIP_Switches_8Bit	
Core: xps_gpio	
Ethernet_MAC	
Core: xps_ethernetlite, Use Interrupt	
IIC_EEPROM	
Core: xps_iic	
LEDs_8Bit	
Core: xps_gpio	
LEDs_Positions	
Core: xps_gpio	
Push_Buttons_5Bit	
Core: xps_gpio	
RS232_Uart_1	
Core: xps_uartlite, Baud Rate: 9600, Data B...	
SRAM	
Core: xps_mch_emc	
SysACE_CompactFlash	
Core: xps_sysace	
xps_bram_if_cntlr_1	
Core: xps_bram_if_cntlr, Size: 8 KB	
xps_timer_0	
Core	xps_timer
Count Width	32
Configure Mode	Two timers are present
Use Interrupt	✓

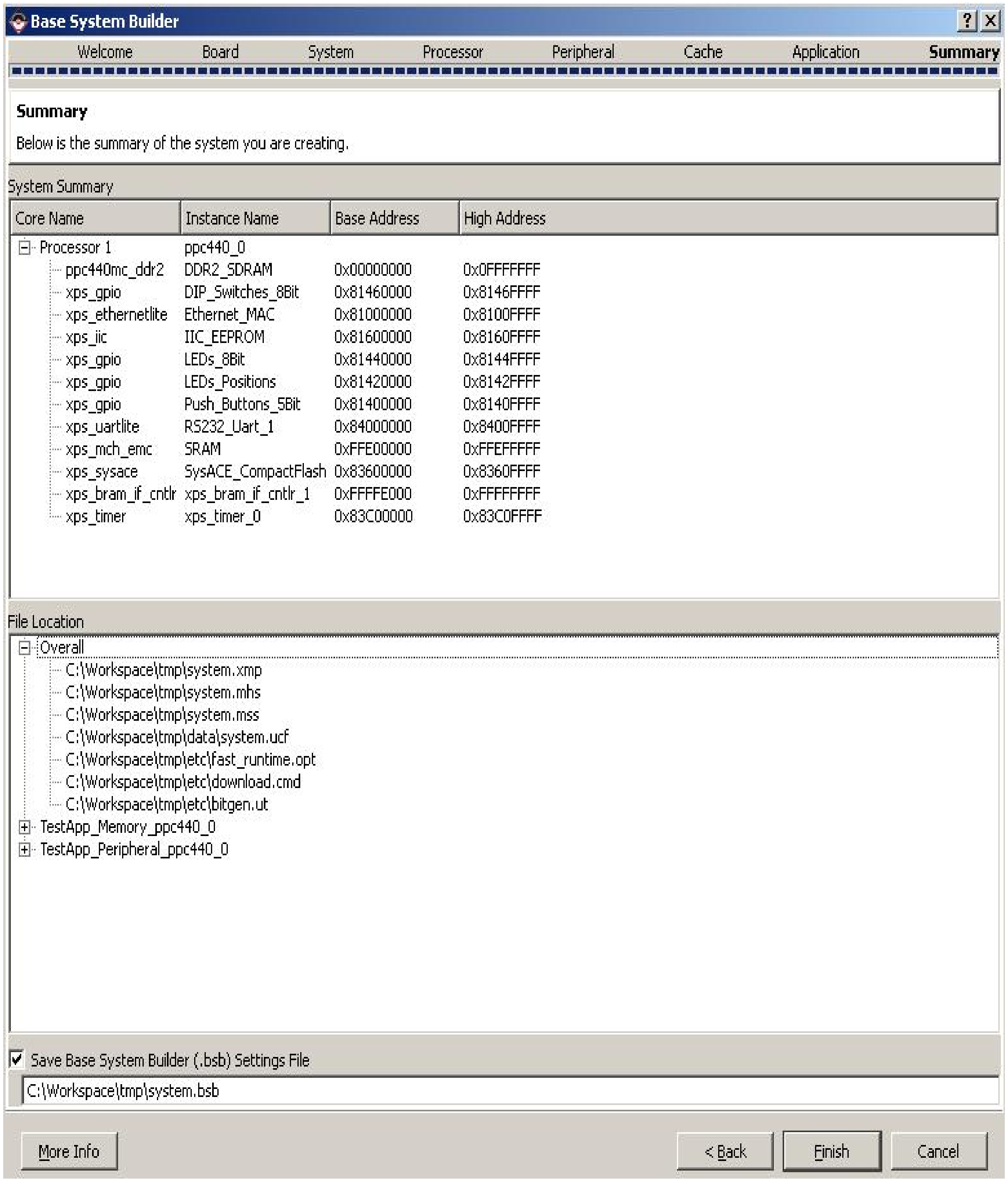
Similarly, make sure the “Use Interrupt” box is checked for the **Ethernet_MAC** peripheral, as shown below.



Processor 1 (PowerPC 440) Peripherals

Core	Parameter
DDR2_SDRAM	
Core: ppc440mc_ddr2	
DIP_Switches_8Bit	
Core: xps_gpio	
Ethernet_MAC	
Core	xps_ethernetlite
Use Interrupt	✓
IIC_EEPROM	
Core: xps_iic	
LEDs_8Bit	
Core: xps_gpio	
LEDs_Positions	
Core: xps_gpio	
Push_Buttons_5Bit	
Core: xps_gpio	
RS232_Uart_1	
Core: xps_uartlite, Baud Rate: 9600, Data B...	
SRAM	
Core: xps_mch_emc	
SysACE_CompactFlash	
Core: xps_sysace	
xps_bram_if_cntlr_1	
Core: xps_bram_if_cntlr, Size: 8 KB	

1. Make sure the “Cache Configuration” settings are set to default and click “Next”. Our sample applications will not need to utilize the instruction or data caches.
2. Click “Next” on the “Application Configuration” dialog window. We will not need to change any of the standard test applications that are built into the project.
3. Inspect the final result of the configuration and make sure it matches the image below (with the exception of directory and file paths, which should vary based on what you entered in step 2). When you are finished, click “Finish” to complete the process.



Summary

Below is the summary of the system you are creating.

System Summary

Core Name	Instance Name	Base Address	High Address
Processor 1	ppc440_0		
ppc440mc_ddr2	DDR2_SDRAM	0x00000000	0x0FFFFFFF
xps_gpio	DIP_Switches_8Bit	0x81460000	0x8146FFFF
xps_ethernetlite	Ethernet_MAC	0x81000000	0x8100FFFF
xps_iic	IIC_EEPROM	0x81600000	0x8160FFFF
xps_gpio	LEDs_8Bit	0x81440000	0x8144FFFF
xps_gpio	LEDs_Positions	0x81420000	0x8142FFFF
xps_gpio	Push_Buttons_5Bit	0x81400000	0x8140FFFF
xps_uartlite	RS232_Uart_1	0x84000000	0x8400FFFF
xps_mch_emc	SRAM	0xFFE00000	0xFFEFFFFF
xps_sysace	SysACE_CompactFlash	0x83600000	0x8360FFFF
xps_bram_if_cntlr	xps_bram_if_cntlr_1	0xFFFFE000	0xFFFFFFFF
xps_timer	xps_timer_0	0x83C00000	0x83C0FFFF

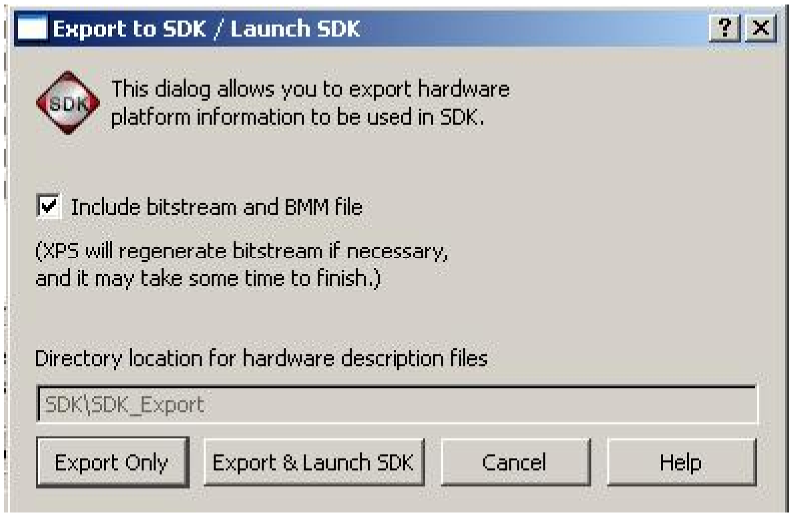
File Location

```
Overall
    C:\Workspace\tmp\system.xmp
    C:\Workspace\tmp\system.mhs
    C:\Workspace\tmp\system.mss
    C:\Workspace\tmp\data\system.ucf
    C:\Workspace\tmp\etc\fast_runtime.opt
    C:\Workspace\tmp\etc\download.cmd
    C:\Workspace\tmp\etc\bitgen.ut
TestApp_Memory_ppc440_0
TestApp_Peripheral_ppc440_0
```

[✓] Save Base System Builder (.bsb) Settings File

C:\Workspace\tmp\system.bsb

1. Inside the XPS application, go to **Project->Export Hardware Design to SDK**. Make sure the “Include bistream and BMM file” box is checked and then click “Export & Launch SDK”, as shown below. The export process will take a relatively long time to complete.


Export to SDK / Launch SDK

This dialog allows you to export hardware platform information to be used in SDK.

[✓] Include bitstream and BMM file

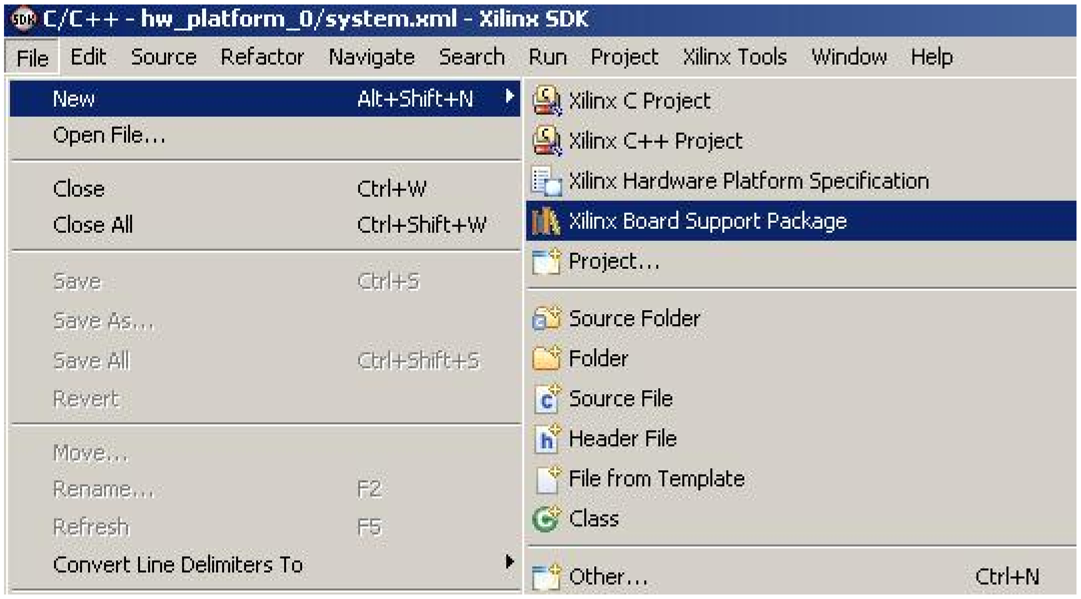
(XPS will regenerate bitstream if necessary, and it may take some time to finish.)

Directory location for hardware description files

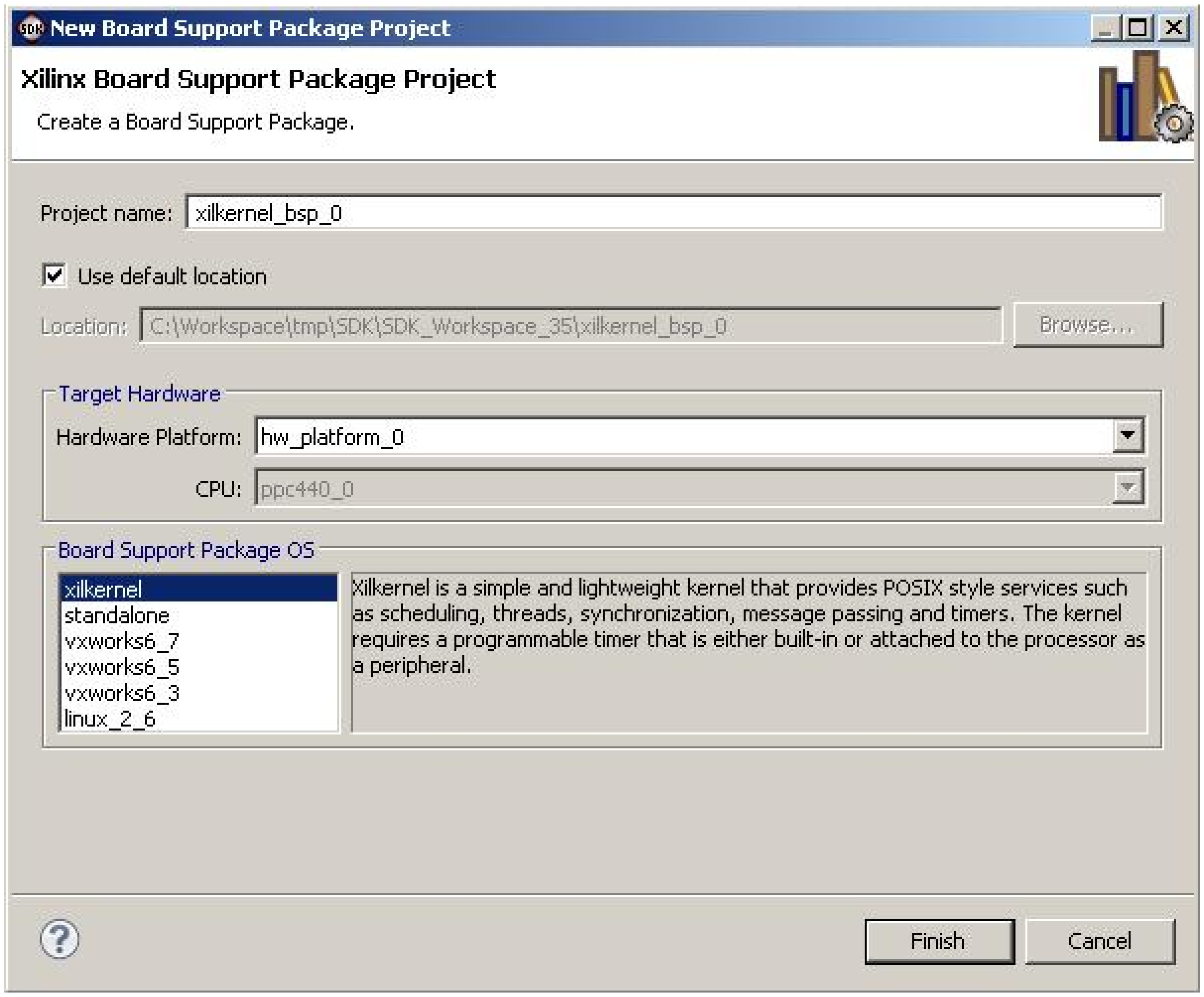
SDK\SDK_Export

Create a Xilkernel BSP

1. When the Xilinx SDK loads the application you will see the **hw_platform_0** configuration under the Project Explorer. This means the configuration was successfully built and exported from the Xilinx Platform Studio.
2. Click on "**File->New->Xilinx Board Support Package**" from the top menu.


C/C++ - hw_platform_0/system.xml - Xilinx SDK

1. In the "New Board Support Package" window that appears, make sure the "xilkernel" Board Support Package OS is selected to target the **hw_platform_0** hardware platform, as shown below. Also, leave the "Use default location" box checked.


New Board Support Package Project
Xilinx Board Support Package Project
Create a Board Support Package.
Browse...
Target Hardware
Board Support Package OS
Finish
Cancel

Project name: xilkernel_bsp_0

[✓] Use default location

Location: C:\Workspace\tmp\SDK\SDK_Workspace_35\xilkernel_bsp_0

Hardware Platform: hw_platform_0

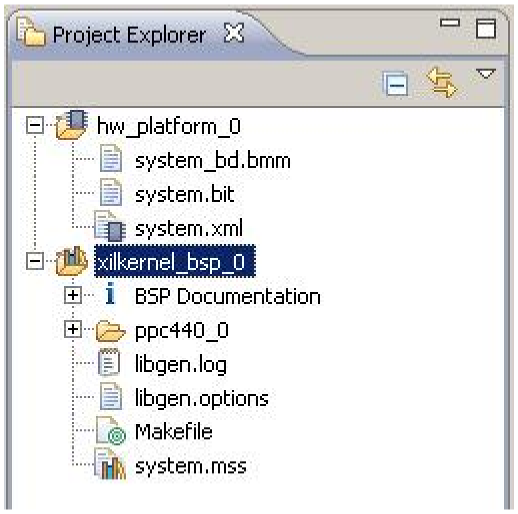
CPU: ppc440_0

[✓ xilkernel]
standalone
vxworks6_7
vxworks6_5
vxworks6_3
linux_2_6

Xilkernel is a simple and lightweight kernel that provides POSIX style services such as scheduling, threads, synchronization, message passing and timers. The kernel requires a programmable timer that is either built-in or attached to the processor as a peripheral.

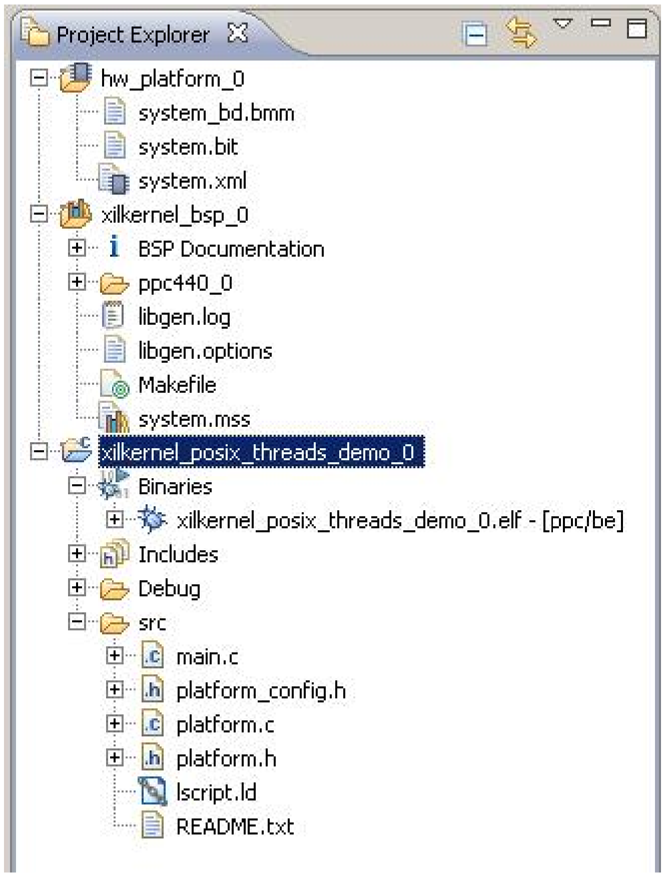
Click “Finish” when these parameters are set up correctly.

1. Leave the default settings on the “Board Support Package Settings” dialog window that appears. Click “OK” to complete the BSP creation process.



Write and Execute a Sample Xilkernel Application

1. Click on “**File->New->Xilinx C Project**” from the top menu of the Xilinx SDK.
2. Select the “**Xilkernel POSIX Threads Demo**” project template from the list of available choices and click “Next”.
3. Make sure the **xilkernel_bsp_0 {OS: xilkernel}** BSP is selected under the available Board Support Packages list and the “Target an existing Board Support Package” bubble is also filled in. This will make sure the Xilkernel BSP we just generated is used as the base OS for this simple application. Click “Finish” when this is complete.
4. The Project Explorer window should now look like the folllowing:



1. You should now see the source code for the multi-threaded application. The functionality of each C source file is described below:

   main.c

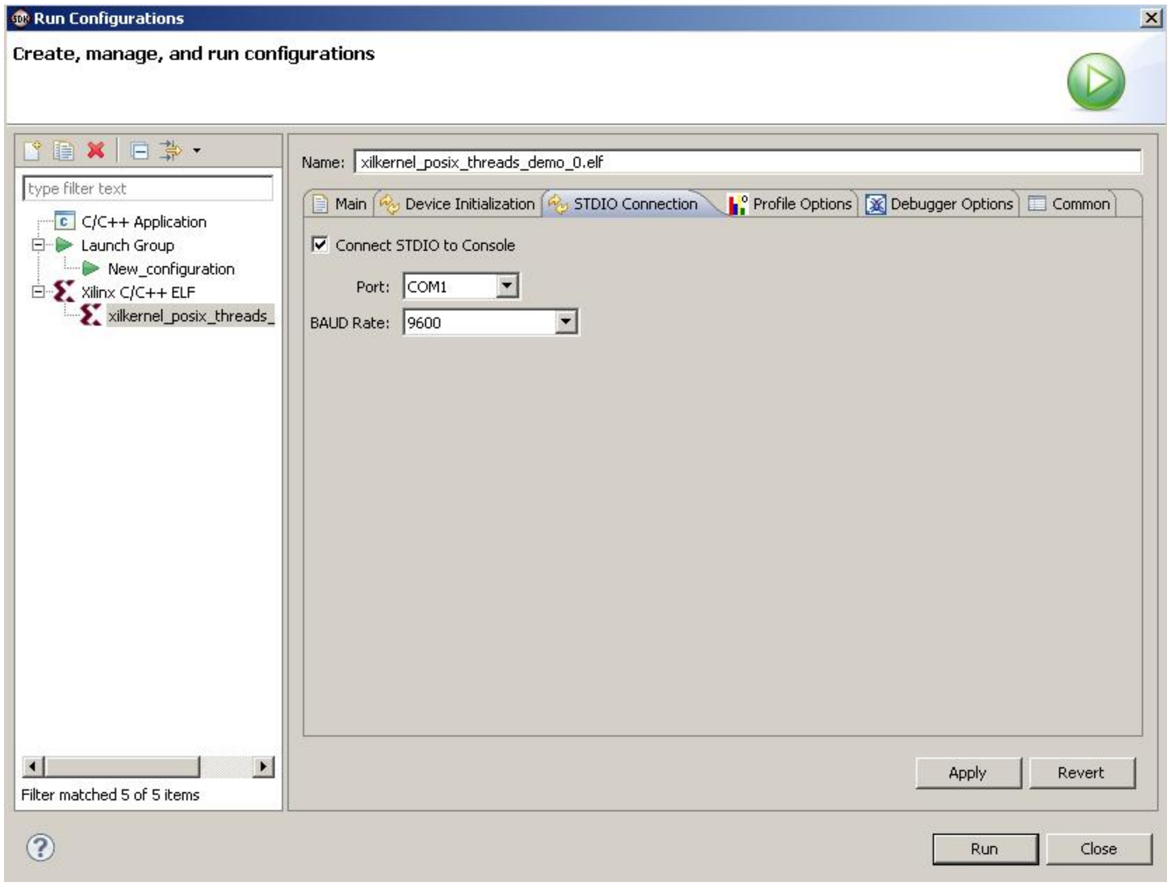
   This file contains the main routine that configures the platform and starts the Xilkernel. It also spawns multiple independent threads that execute concurrently in order to compute the sum of elements in a one-dimensional array.

   platform_config.c

   This contains the platform specific (e.g. the ML507) code necessary to configure the system to run the Xilkernel.

1. The Xilinx SDK will continuously build the application after every code change so you are always ready to download and run it on the FPGA. To run the application, right click on the **xilkernel_posix_threads_demo_0** project under the Project Explorer and then click on "**Run As->Run Configurations**". This will bring up the run configuration settings that we will modify slightly to see standard output displayed in the SDK console (as opposed to opening up a terminal client such as Putty to capture serial data).
2. In the "Run Configurations" window that appears, click on the "STDIO Connection" tab

and then check the “**Connect STDIO to Console**” box, as shown below.



Name: xilkernel_posix_threads_demo_0.elf

[✓] Connect STDIO to Console

Port: COM1

BAUD Rate: 9600

When this is done, click on “Run” to start the application.

1. You should now see the following output on the console:

```
--------------------------------------------------------------------------------
Xilkernel POSIX Threads Demo
--------------------------------------------------------------------------------
This Xilkernel based application provides a simple example of how to create
multiple POSIX threads and synchronize with them when they are complete. This
example creates an initial master thread. The master thread creates 4 worker
threads that go off to compute the sum of subsets of a randomly created array
and return the partial sum as the result. The master thread accumulates the
partial sums and prints the result. This example can serve as your starting
point for your end application thread structure.
--------------------------------------------------------------------------------

Xilkernel Demo: Master Thread Starting.
Xilkernel Demo: Collected result (5050) from worker: 0.
Xilkernel Demo: Collected result (15050) from worker: 1.
Xilkernel Demo: Collected result (25050) from worker: 2.
Xilkernel Demo: Collected result (35050) from worker: 3.
Xilkernel Demo: Result computed by worker threads = 80200.
Xilkernel Demo: Master Thread Completing.
```

Now see if you can modify the program to increase the number of worker threads that compute these results.

Issues and Troubleshooting

All issues, questions, and concerns can be directed to Christopher Wood at caw4567@rit.edu.