µC/OS-II™

The Real-Time Kernel

For the Xilinx® Zynq™-7000 EPP

µC/OS-II™ Demo on the Xilinx® Zynq™-7000

ZC702 Evaluation Board

Walkthrough Guide

V1.01

Micriμm

Introduction

This walkthrough guide provides an introduction to µC/OS-II running on the Xilinx ZC702 Evaluation Board.

The document describes the steps necessary to run a µC/OS-II demo application on the Xilinx ZC702 Evaluation Board using the Xilinx ISE Design Suite. Therefore, it is assumed that the ISE Design Suite v14.2 is already installed in your PC.

The document is divided in 5 sections:

- Requirements
- Walkthrough Guide Map
- Walkthrough Guide
- Files Manifest
- How the µC/OS-II Demo Works

1. Requirements

- Windows PC
- Xilinx ISE Design Suite 14.2
- Micriμm Software (μC/OS-II and Demo Application)
- Serial Terminal Program (HyperTerminal, PuTTY, TeraTerm, etc.)
- Xilinx ZC702 Evaluation Board Rev C
- Xilinx USB Platform Cable

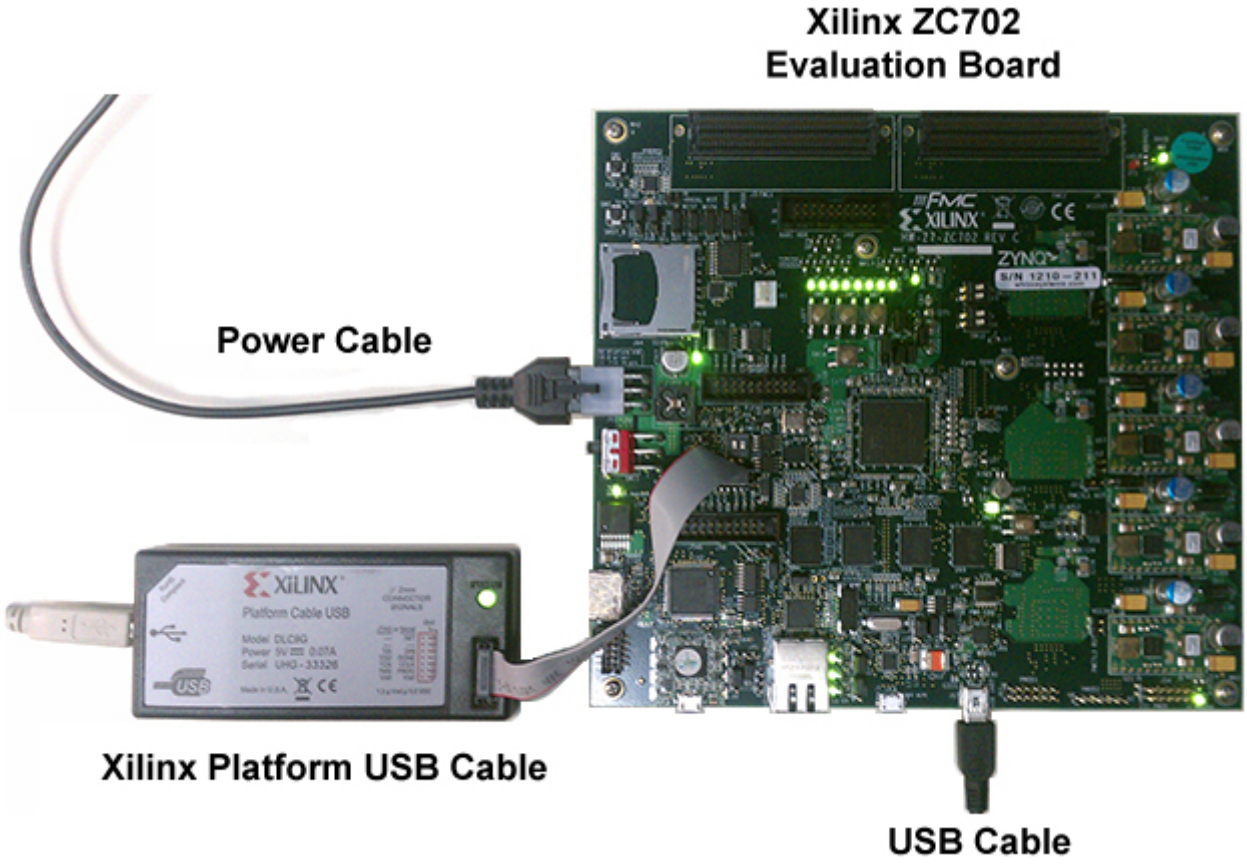

Figure 1-1 **Hardware Requirements**

2. Walkthrough Guide Map

Figure 2-1 illustrates the components required to run the μC/OS-II demo and serves as the map for this walkthrough guide. Use this map to quickly navigate through the guide.

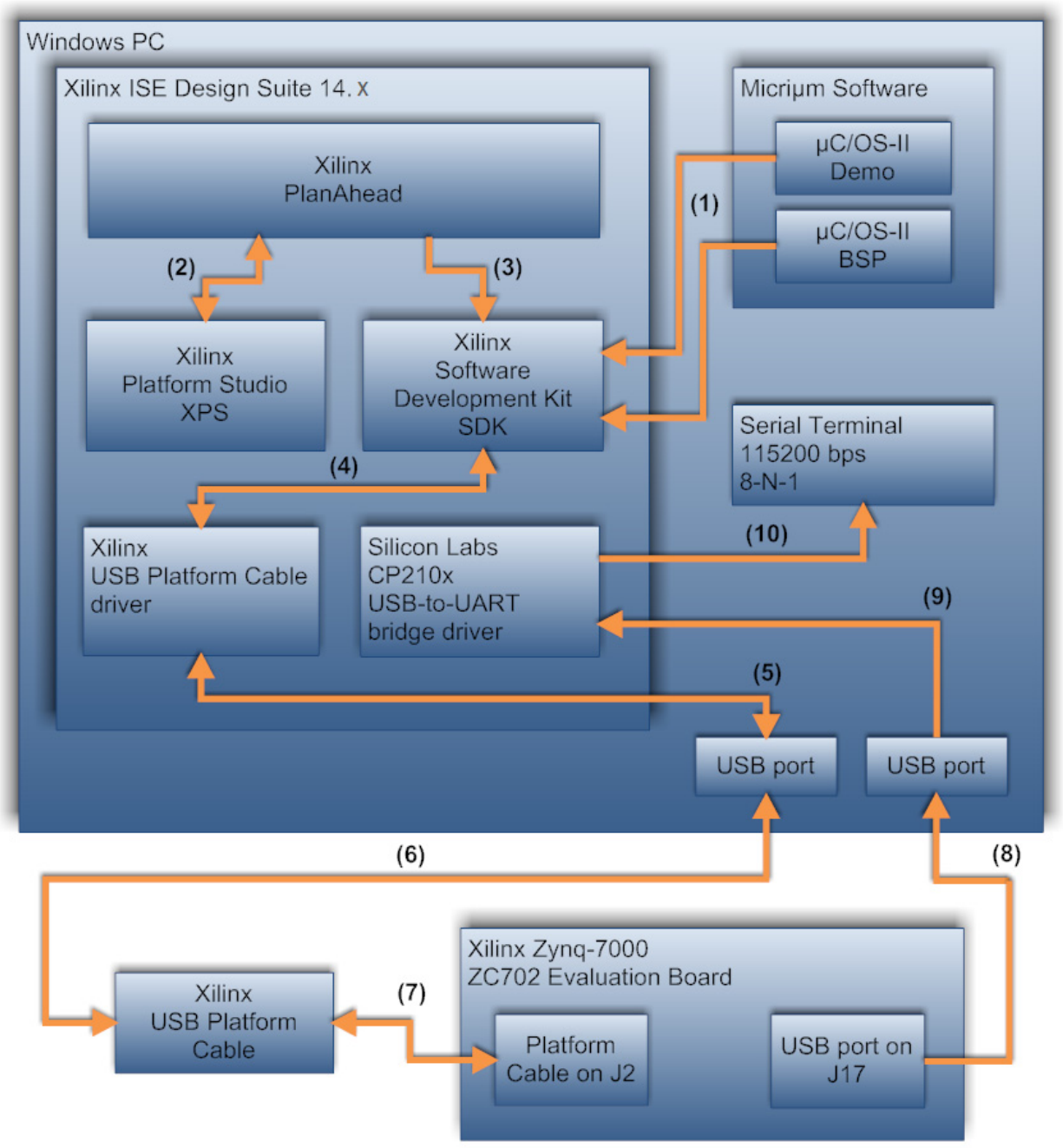

Figure 2-1 **μC/OS-II Demo Walkthrough Guide Map**

F2-1(1) The software available from Micriμm includes the source code of μC/OS-II and a demo application. Follow step #1 in the next section, in order to install these files in your PC.

F2-1(2) The ISE Design Suite from Xilinx includes the PlanAhead software (PA), the Xilinx Platform Studio (XPS) and the Software Development Kit (SDK). The suite also includes all the necessary USB drivers. This document assumes you already have successfully installed the ISE Design Suite and USB drivers in your PC. For more information, please refer to the document ISE Design Suite Installation and Licensing Guide at http://www.xilinx.com/support/documentation/zynq-7000.htm.

The PlanAhead software is the Zynq tools main entry point. From the PA software you can create a new PA project that includes both, the hardware and software data files. Follow steps 2 through 25 in the next section to learn how to use the PlanAhead software and the Xilinx Platform Studio (XPS) tool to create the hardware platform design specification necessary to run the μC/OS-II demo on the ZC702 evaluation board.

F2-1(3) The Software Development Kit (SDK) is the tool that includes the Eclipse-based IDE and GNU toolchain. The PlanAhead software exports the hardware platform design specification files to the SDK. Follow step 26 to export the ZC702 hardware platform design specification files to the SDK before running the μC/OS-II demo.

F2-1(4) The SDK tool allows you to compile and debug the μC/OS-II demo application. Follow steps 29 through 49 in order to compile and debug the μC/OS-II demo application.

F2-1(5) The Xilinx Platform Cable USB driver is included with the ISE Design Suite. This driver is necessary for Windows to recognize the Platform Cable as a valid USB programming device.

F2-1(6) The Xilinx Platform Cable itself is a device that allows you not only to download and program the μC/OS-II demo application into the ZC702 evaluation board, but also to debug the application through JTAG. Follow steps 29 through 36 to learn how to create and compile a μC/OS-II demo application.

F2-1(7) The Xilinx Platform Cable comes with a ribbon cable that needs to be connected to the J2 connector on the ZC702 evaluation board.

F2-1(8) A USB cable connects the ZC702 evaluation board to the PC through a serial interface configured at 115200 bps 8-N-1. This serial interface will be used to output messages from the μC/OS-II demo application.

F2-1(9) The ISE Design Suite from Xilinx includes a USB-to-UART bridge driver from Silicon Labs. The driver is necessary to recognize the ZC702 evaluation board. The Windows PC will configure the connection as a virtual COM port.

F2-1(10) The PC needs to have a serial terminal available to display the messages from the μC/OS-II demo application. Some examples of serial terminal programs include HyperTerminal, PuTTY or TeraTerm Pro. Follow steps 37 through 49 in order to run the μC/OS-II demo application and display messages in the serial terminal.

3. Walkthrough Guide

This section provides step-by-step instructions for installing and running the μC/OS-II demo application on the Xilinx ZC702 evaluation board.

The Xilinx Zynq-7000 is probably the most flexible and extensible system in the market today. With flexibility comes complexity, therefore, we strongly recommend to follow each step of this walkthrough guide carefully. At the end of each step you will be presented with a screen capture that must match your screen. Do not take the next step if your screen does not match the current step's screen capture and contact technical support at http://www.xilinx.com/support before proceeding.

Please take the following steps in order to get μC/OS-II running on your ZC702 evaluation board:

3-1 Step #1

In order to install the μC/OS-II source code and demo application in your PC, locate the zip files `zynq-7000-ucosii-bsp.zip` and `zynq-7000-ucosii-demo.zip`, and extract them into a **`bsp`** and **`sw_apps`** folders respectively. For example, you can create folders `bsp` and `sw_apps` in your own `My Documents` folder or, you can use the existing `bsp` and `sw_apps` folders of your Xilinx ISE Design Suite installation directory as illustrated in Figure 3-1.

- The zip file `zynq-7000-ucosii-bsp.zip` needs to be extracted into the **`bsp`** folder of your Xilinx ISE Design Suite installation directory at:
- `C:\Xilinx\14.2\ISE_DS\EDK\sw\lib\bsp`
- The zip file `zynq-7000-ucosii-demo.zip` needs to be extracted into the **`sw_apps`** folder of your Xilinx ISE Design Suite installation directory at:
- `C:\Xilinx\14.2\ISE_DS\EDK\sw\lib\sw_apps`

Verify that your final folder tree looks like the one shown in Figure 3-1 before proceeding to the next step.

If you chose to install the files into your own directory, then the SDK needs to be aware of this new repository path. The note at the bottom of Step #33 describes how to do that.

Please note that μC/OS-II is provided in source form for free evaluation, for educational use or for peaceful research. If you plan on using μC/OS-II in a commercial product you need to contact Micriμm to properly license its use in your product. We provide all the source code for your convenience and to help you experience μC/OS-II. The fact that the source is provided does not mean that you can use it without paying a licensing fee.

If you are unsure about whether you need to obtain a license for your application, please contact Micriμm and discuss the intended use with a sales representative.

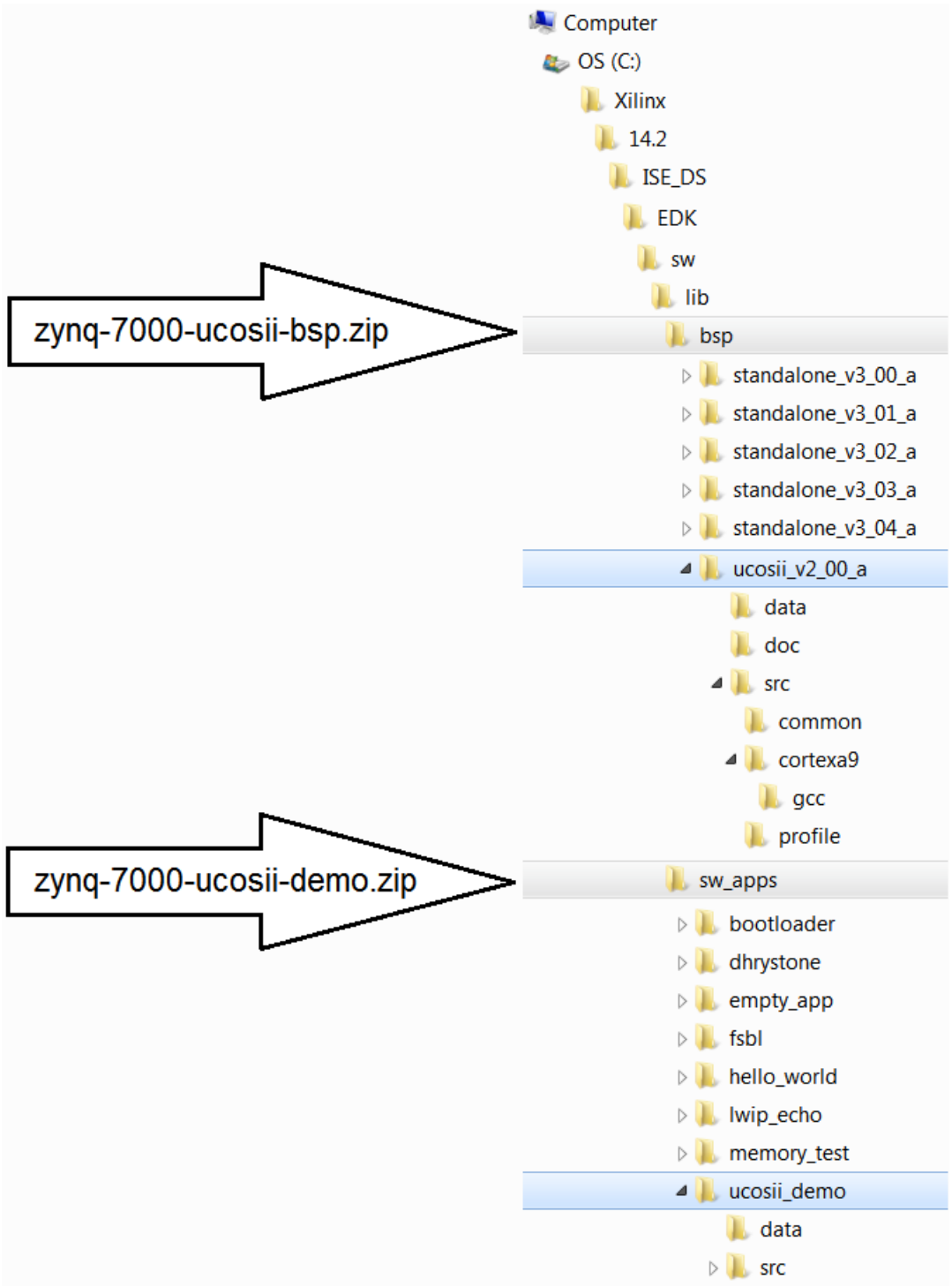

Figure 3-1 **Micriμm Software destination folders**

3-2 Step #2

Start the Xilinx PlanAhead software by using the shortcut on your desktop that looks like the one shown in Figure 3-2.

Figure 3-2 **Xilinx PlanAhead Shortcut**

3-3 Step #3

Figure 3-3 shows the PlanAhead software main screen. The PA software will help you create a PA project that includes both hardware and software data files necessary to support the ZC702 evaluation board. Select Create New Project to open the New Project wizard.

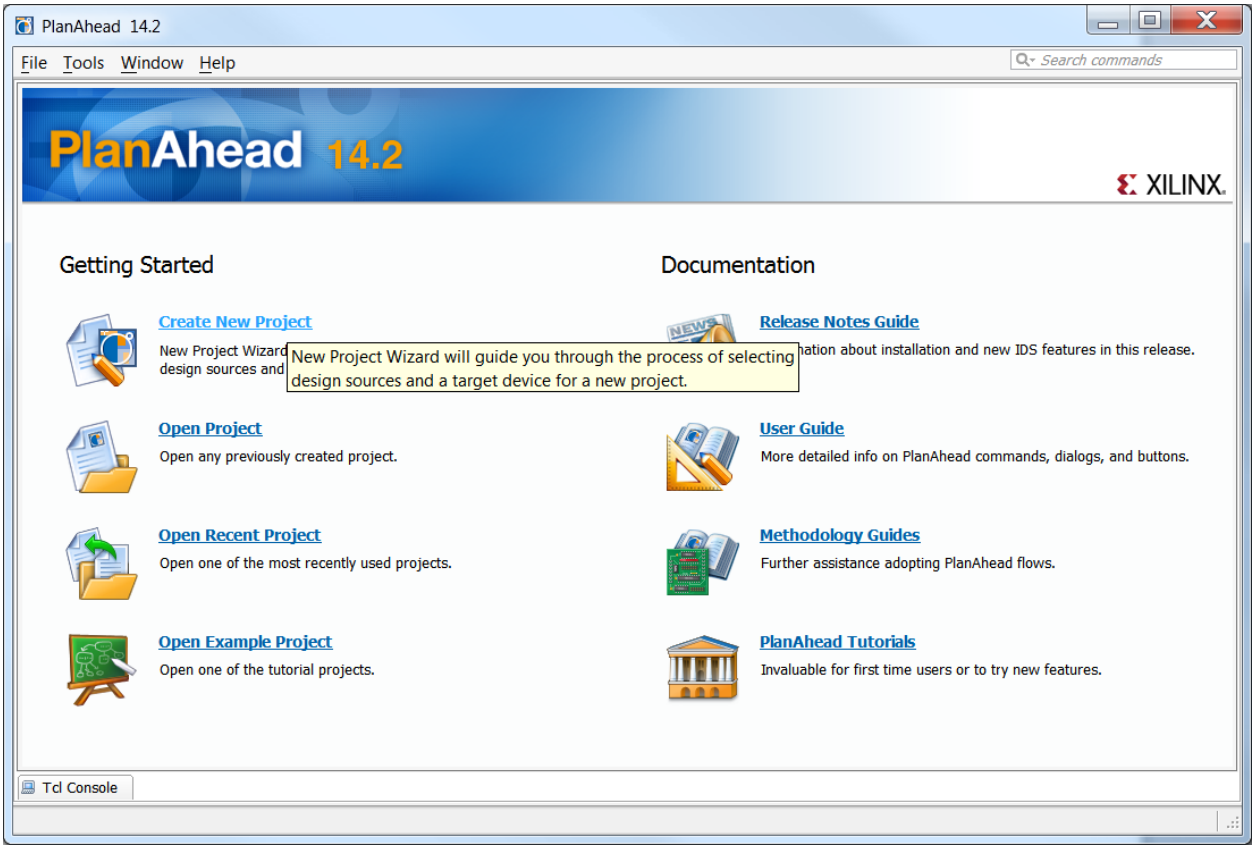

Figure 3-3 **PlanAhead Software: Main Screen**

3-4 Step #4

Figure 3-4 shows the first screen of the new PlanAhead project wizard. Click next to continue.

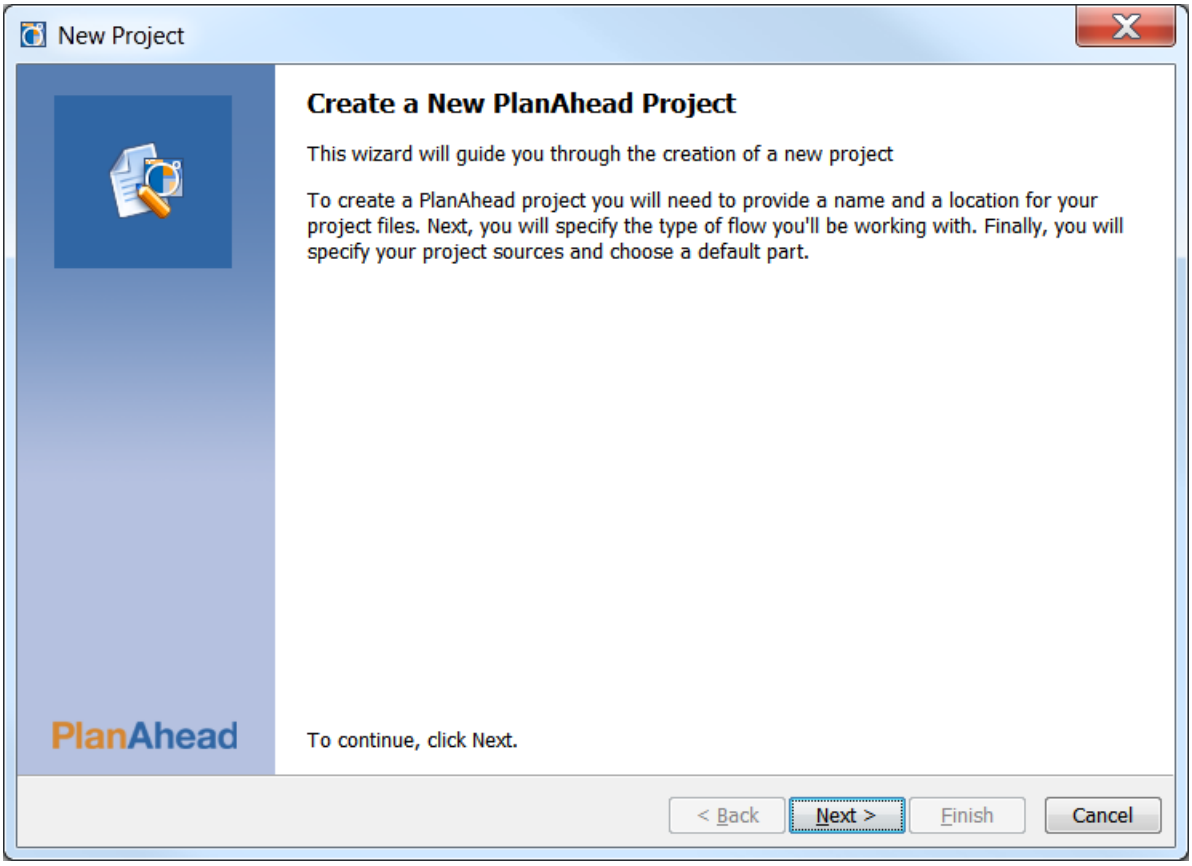

Figure 3-4 **PlanAhead Software: New PlanAhead Project wizard**

3-5 Step #5

Enter a name for your project and specify a directory where the project data files will be stored. Keep in mind that this project will contain both hardware and software data files. Click **Next** to continue as shown in Figure 3-5:

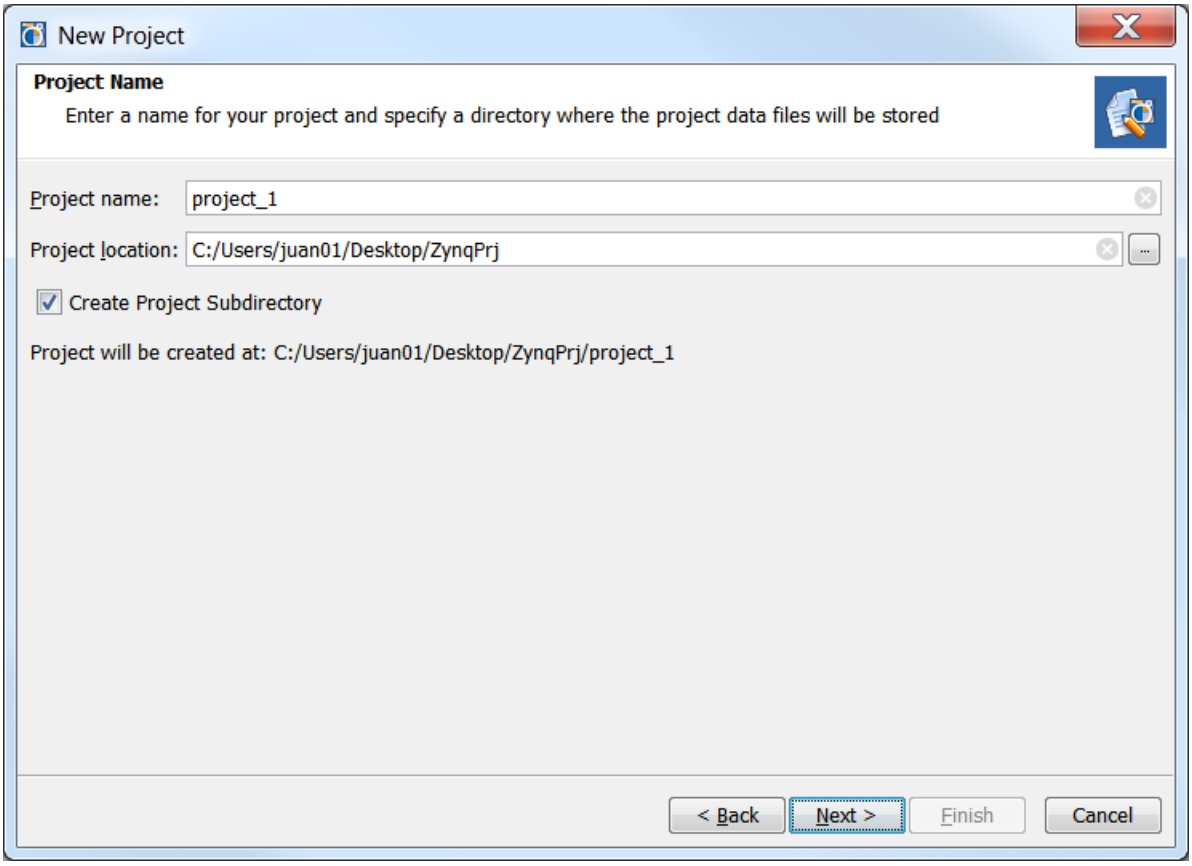

Figure 3-5 **PlanAhead Software: New PlanAhead Project wizard: Project name**

3-6 Step #6

Select the **Specify RTL Sources** checkbox as shown in Figure 3-6 and click **Next** to continue.

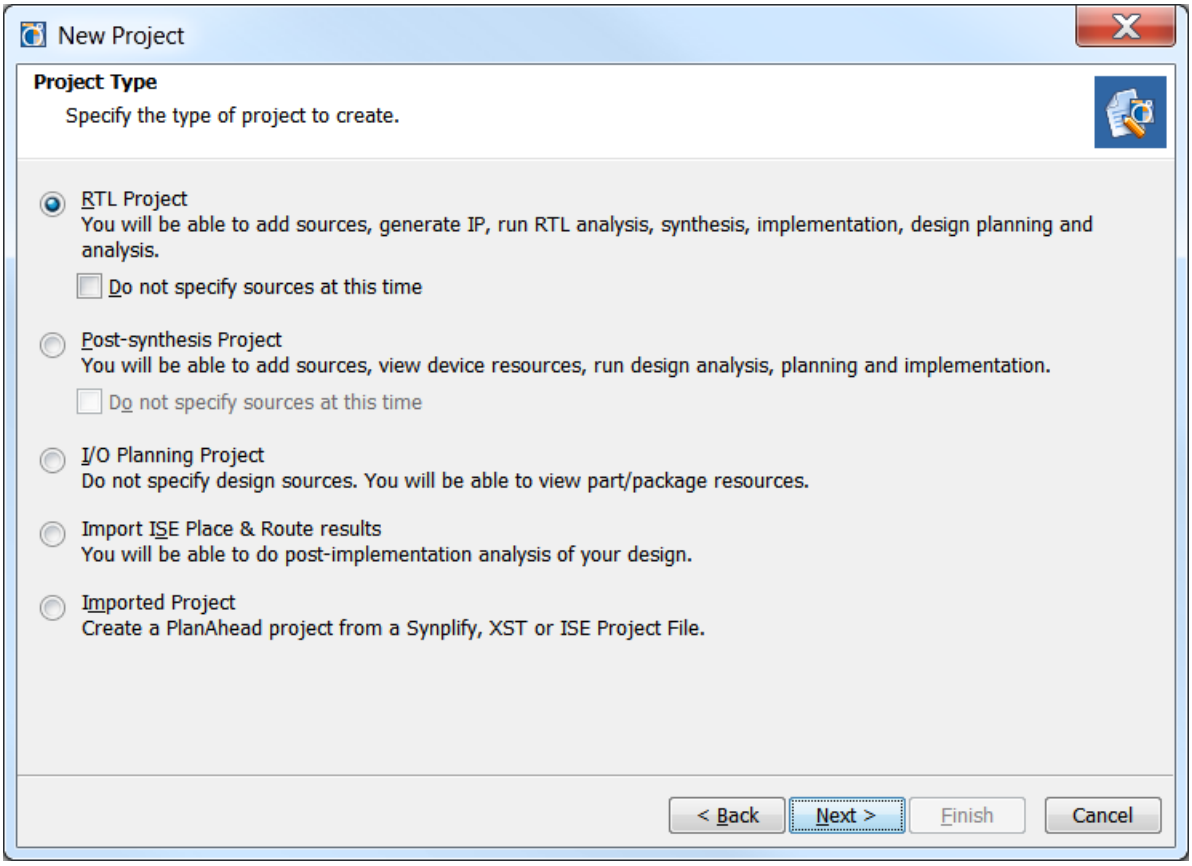

Figure 3-6 **PlanAhead Software: New PlanAhead Project wizard: Design Source**

3-7 Step #7

Leave the **Add Sources** screen unchanged as shown in Figure 3-7 and click **Next** to continue:

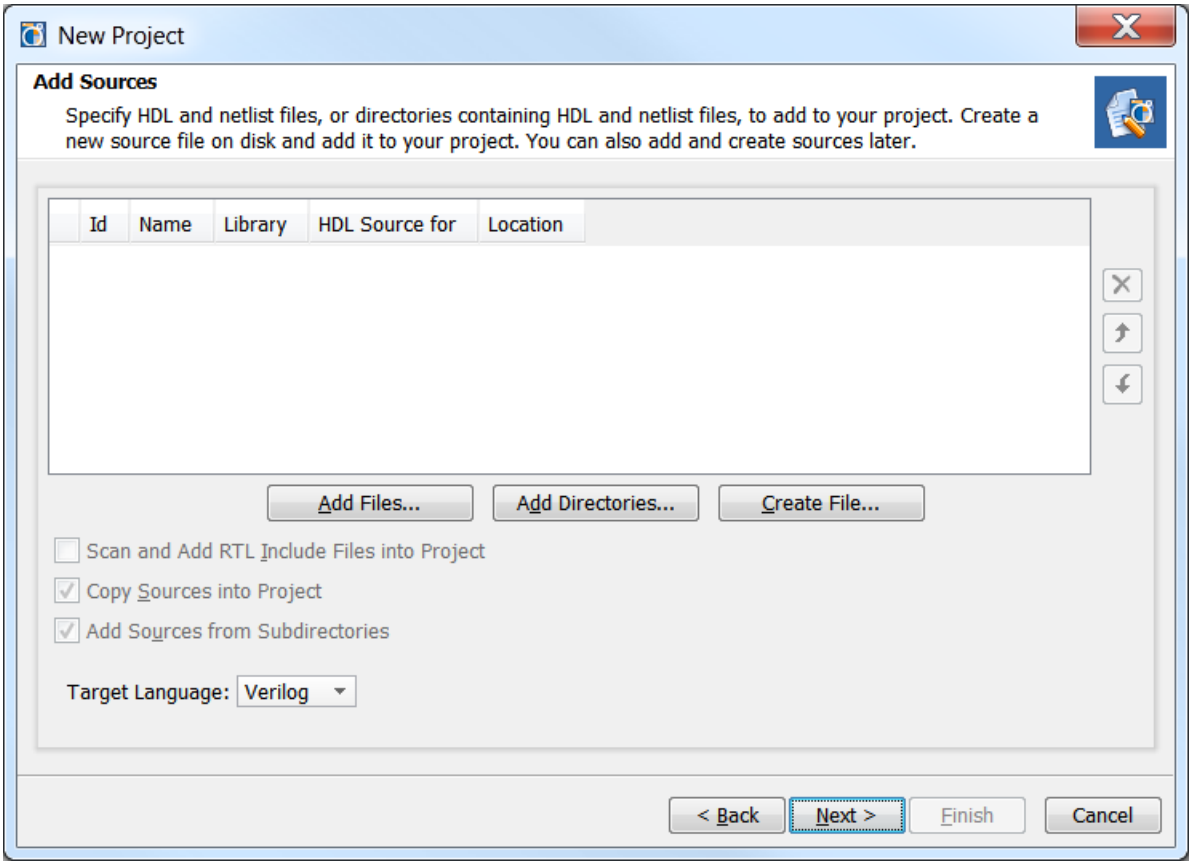

Figure 3-7 **PlanAhead Software: New PlanAhead Project wizard: Add Sources**

3-8 Step #8

Leave the **Add Existing IP** screen unchanged as shown in Figure 3-8 and click **Next** to continue:

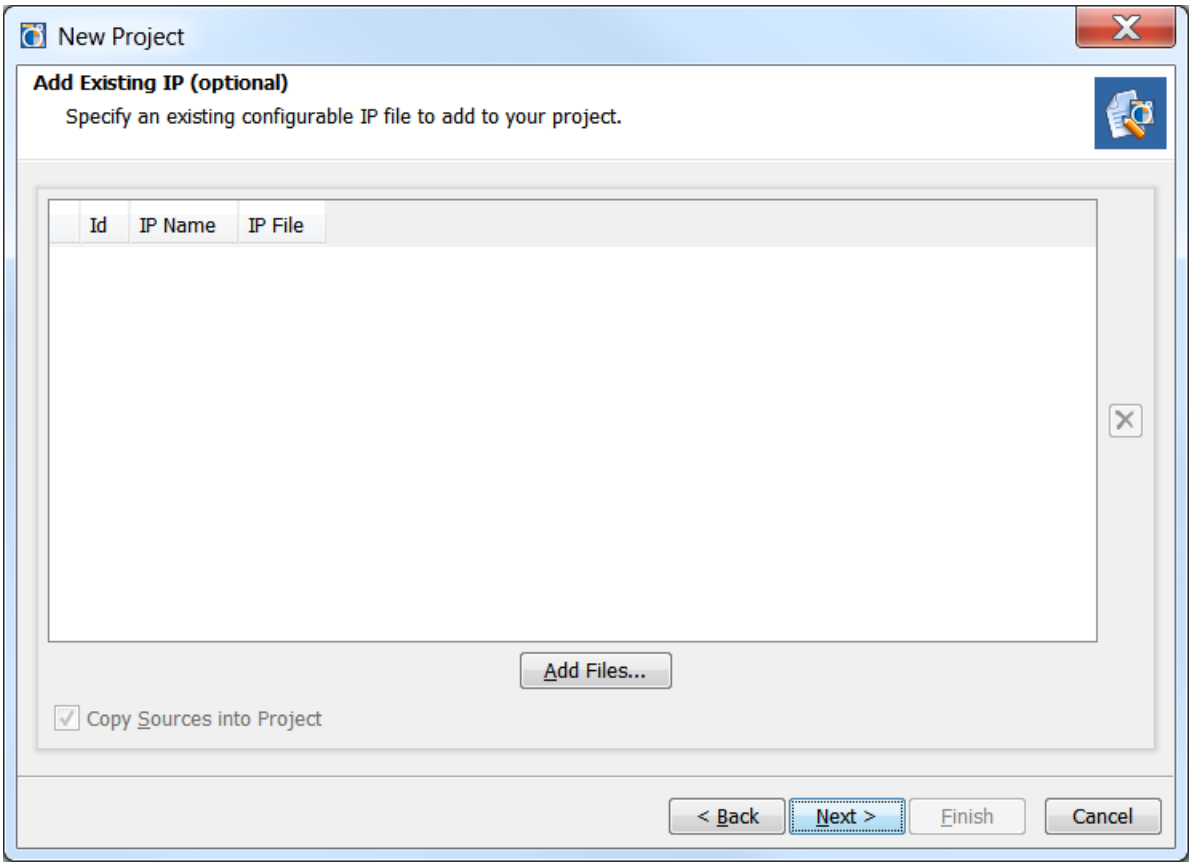

Figure 3-8 **PlanAhead Software: New PlanAhead Project wizard: Add Existing IP**

3-9 Step #9

Leave the **Add Constraints** screen unchanged as shown in Figure 3-9 and click **Next** to continue:

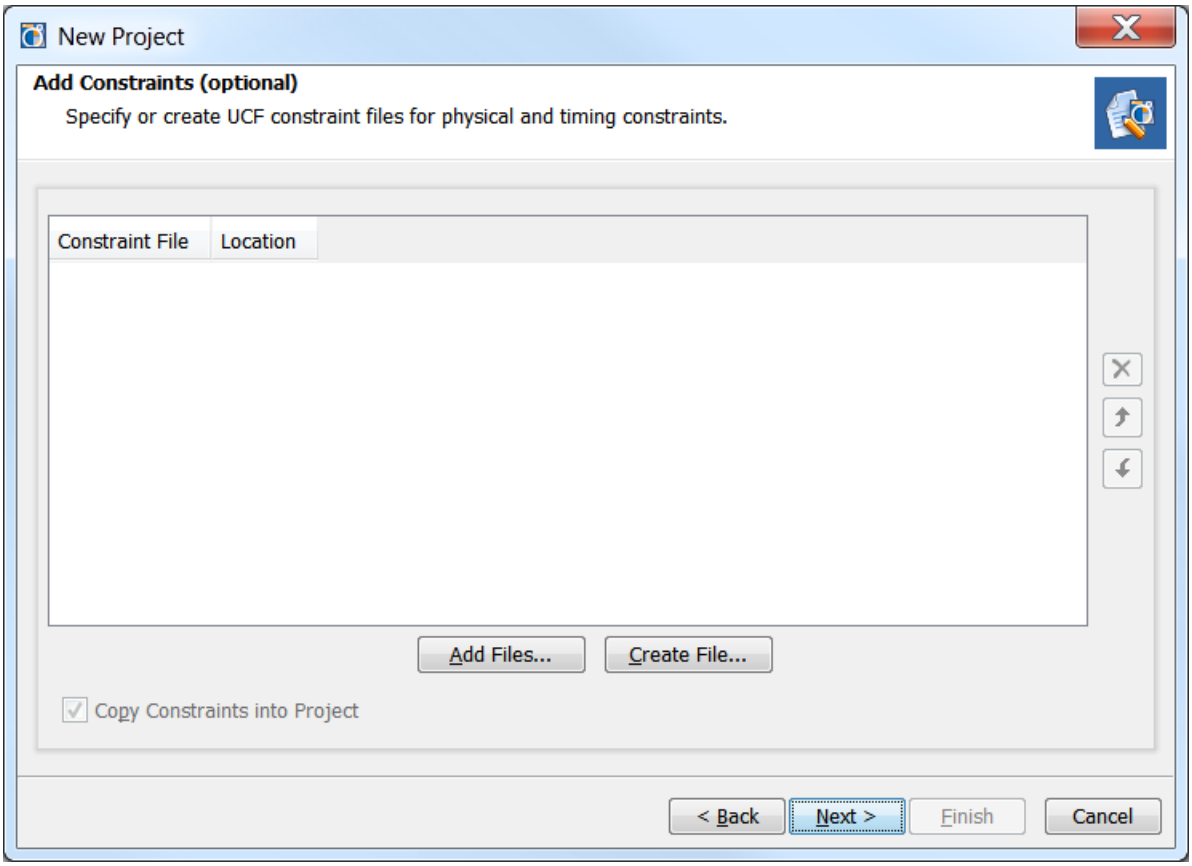

Figure 3-9 **PlanAhead Software: New PlanAhead Project wizard: Add Constraints**

3-10 Step #10

Select the item **Boards** under the **Specify** panel, filter by **Zynq-7000** family and select the **ZC702 Evaluation Board** as shown in Figure 3-10. Click **Next** to continue:

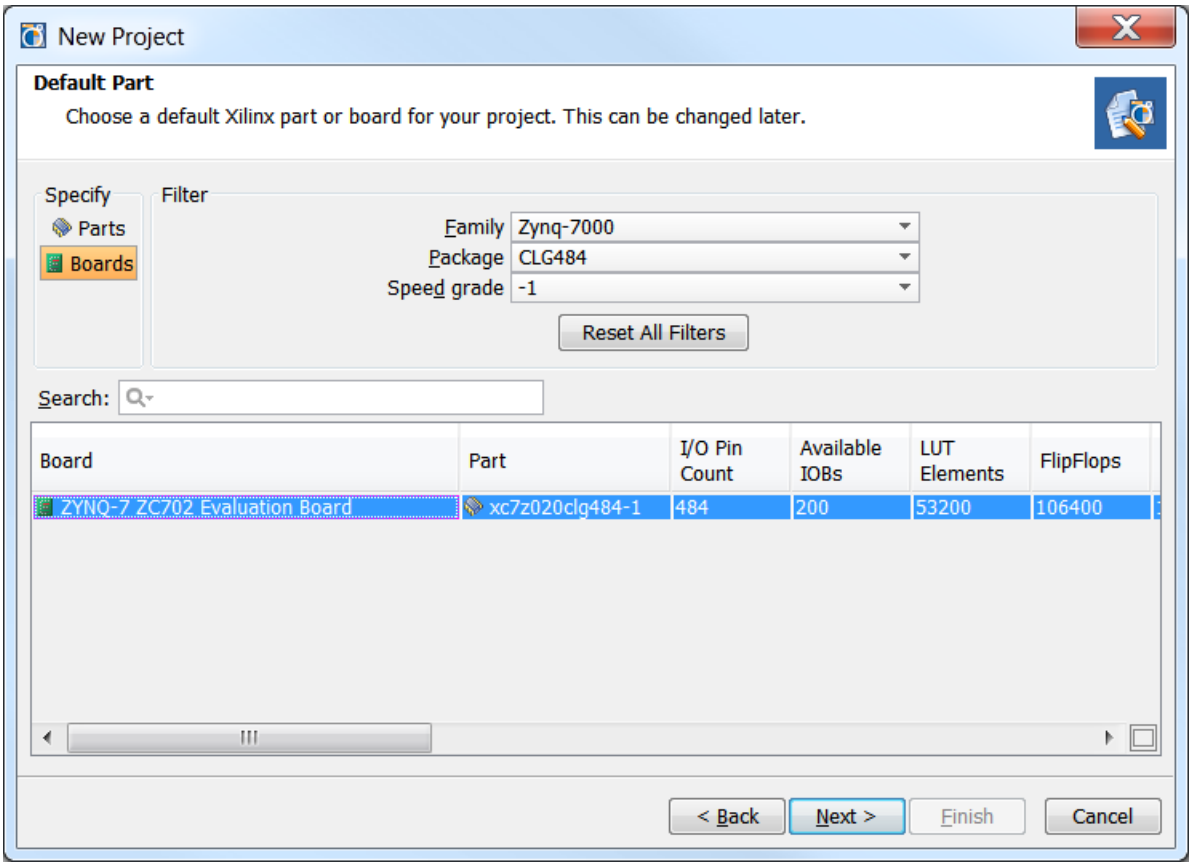

Figure 3-10 **PlanAhead Software: New PlanAhead Project wizard: Default Part**

3-11 Step #11

The final screen in the New PlanAhead Project Wizard is shown in Figure 3-11. Make sure it looks similar and click **Finish** to create your new PlanAhead project:

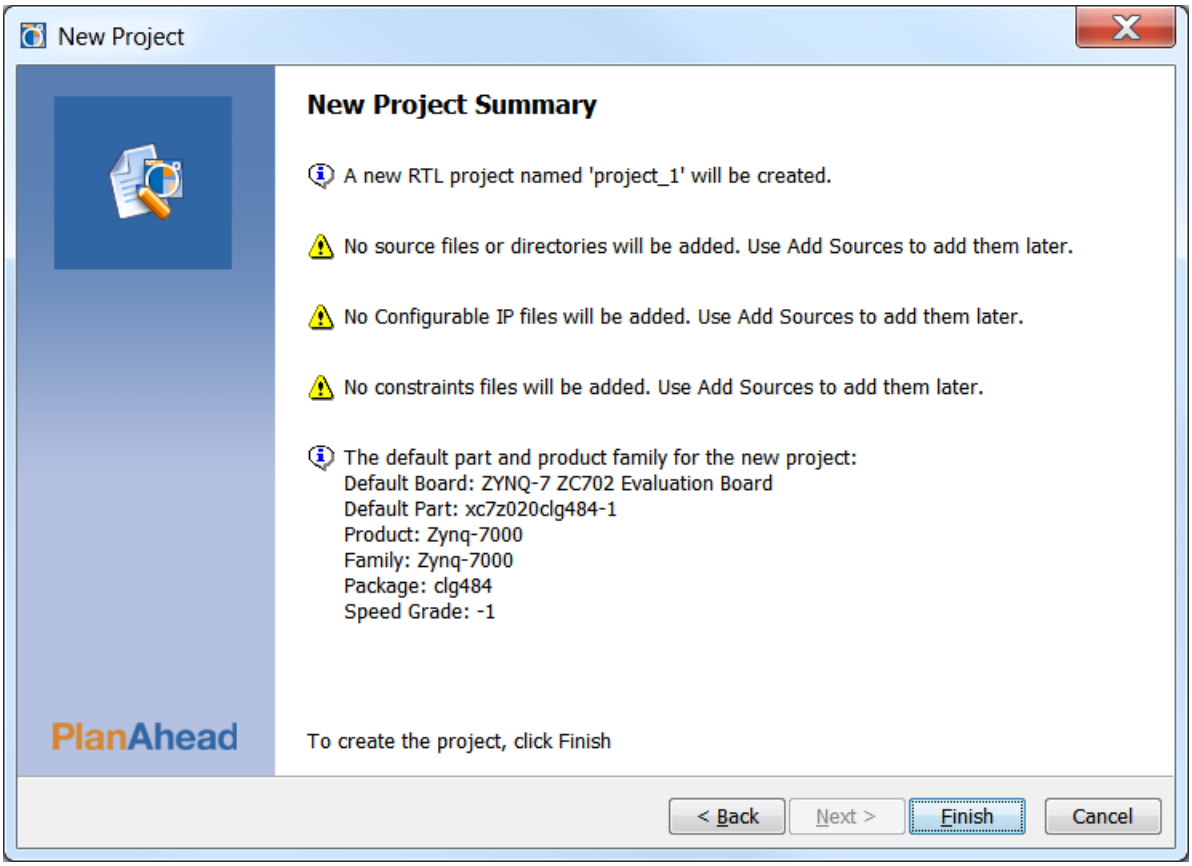

Figure 3-11 **PlanAhead Software: New PlanAhead Project wizard: Summary**

3-12 Step #12

After you click **Finish**, the New Project wizard closes and the project you just created opens in PlanAhead just as shown in Figure 3-12.

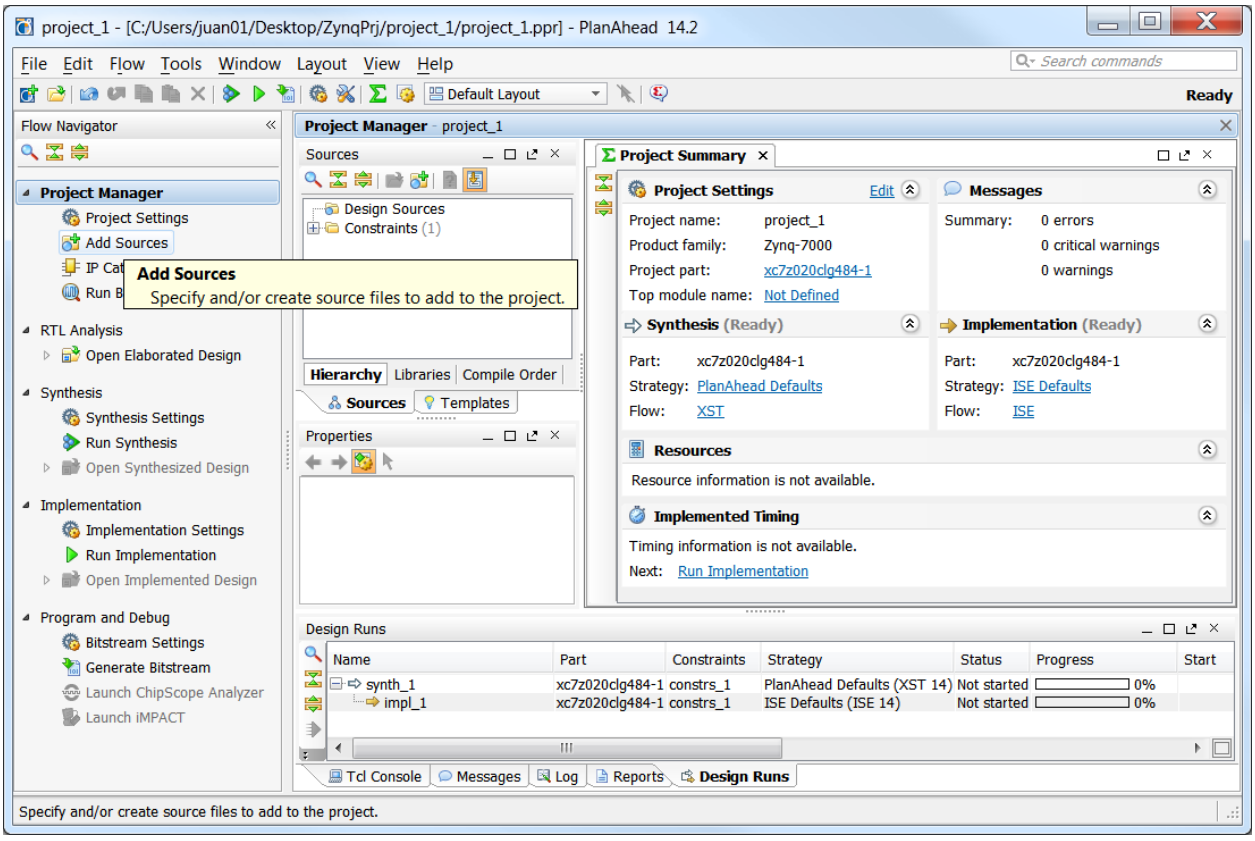

Figure 3-12 **PlanAhead Software: Project Manager**

3-13 Step #13

The next step is to create a new design source by making click on **Add Sources** under the PlanAhead Project Manager as shown in Figure 3-13:

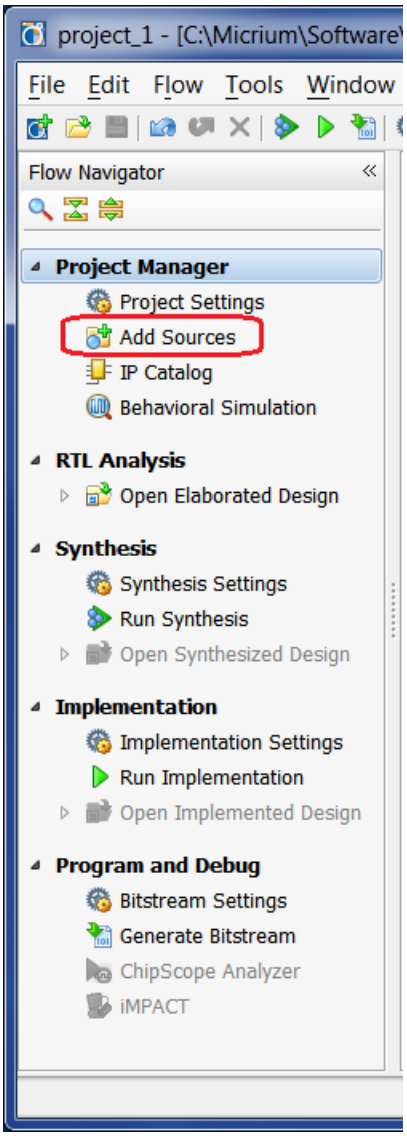

Figure 3-13 **PlanAhead Software: Add Sources**

3-14 Step #14

The **Add Sources** wizard opens. Select the **Add or Create Embedded Sources** option as shown in Figure 3-14:

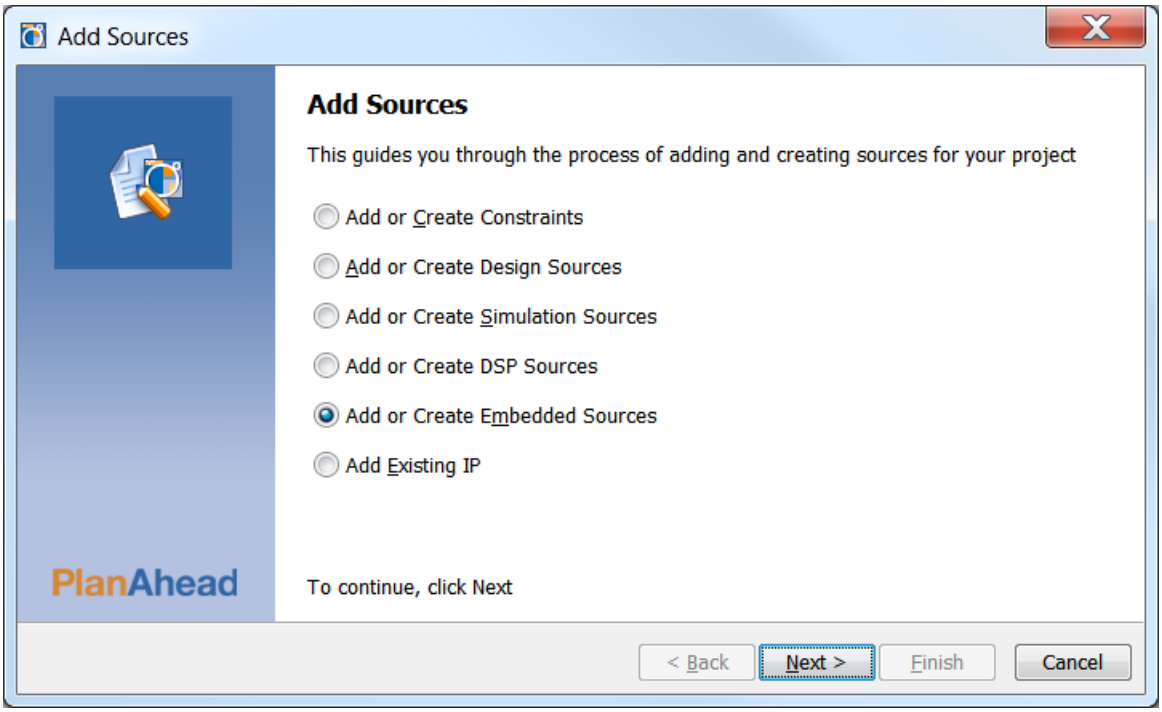

Figure 3-14 **PlanAhead Software: Add Sources wizard**

3-15 Step #15

In the **Add or Create Embedded Sources** window, click the **Create Sub-Design** button as shown in Figure 3-15:

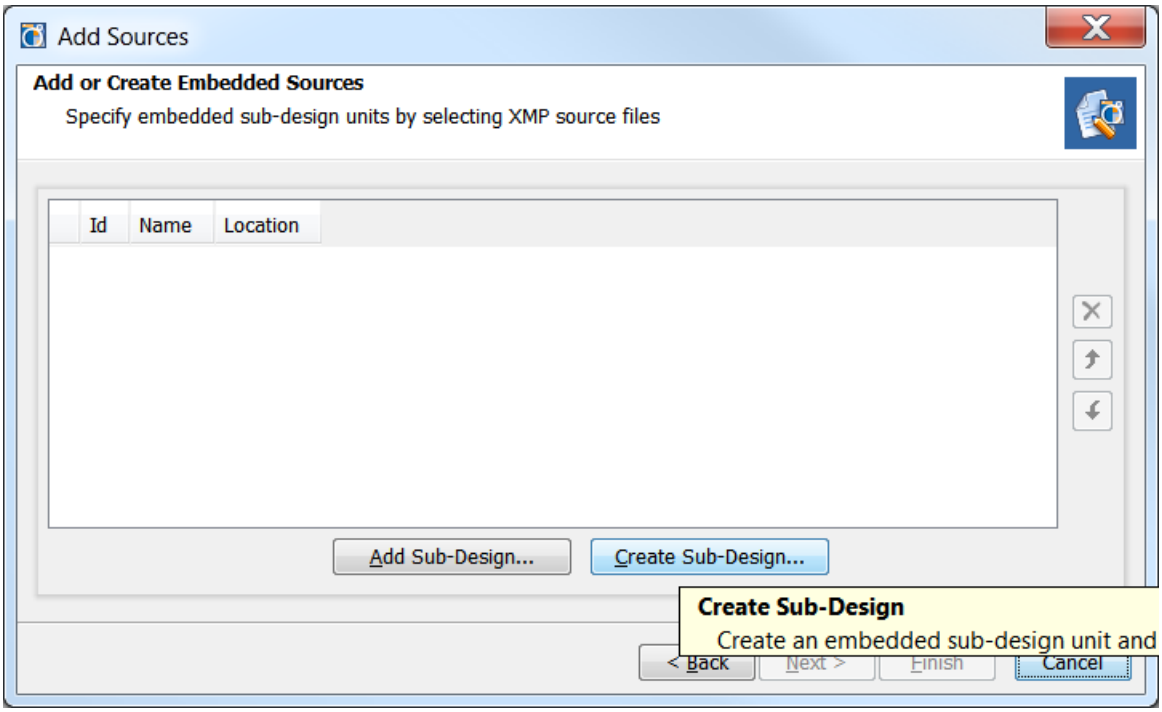

Figure 3-15 **PlanAhead Software: Add Sources Add or Create Embedded Sources**

3-16 Step #16

Type a name for the module and click **OK** as illustrated in Figure 3-16:

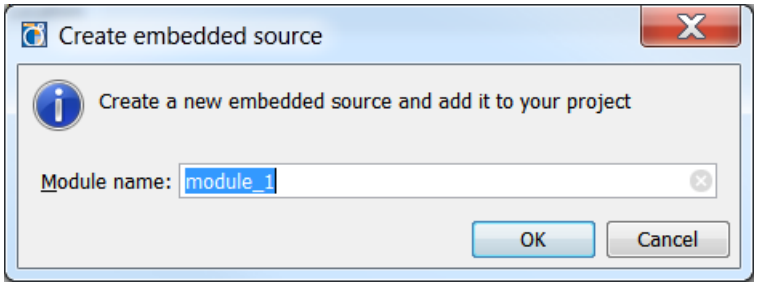

Figure 3-16 **PlanAhead Software: Add Sources: Create embedded source**

3-17 Step #17

Verify that the module you created displays in the sources list as shown in Figure 3-17 and click **Finish** to add the embedded source.

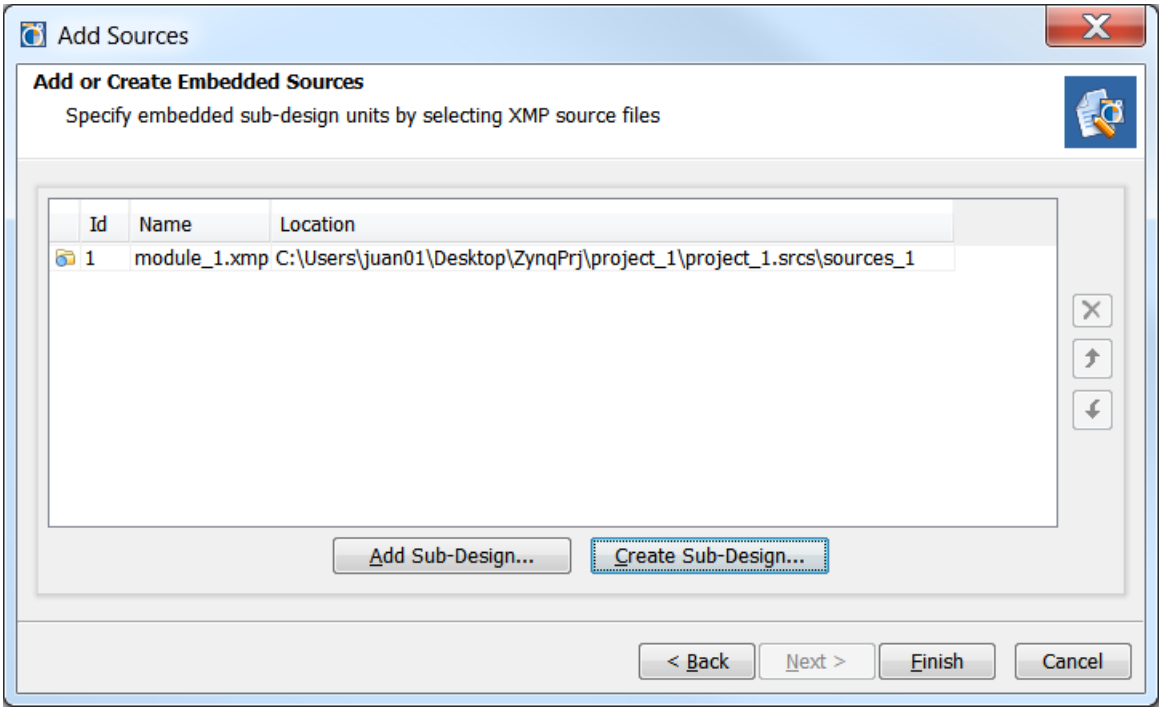

Figure 3-17 **PlanAhead Software: Add Sources: Add or Create Embedded Sources**

3-18 Step #18

A progress bar as shown in Figure 3-18 indicates the progress in the process. Just wait until the Xilinx Platform Studio (XPS) tool opens.

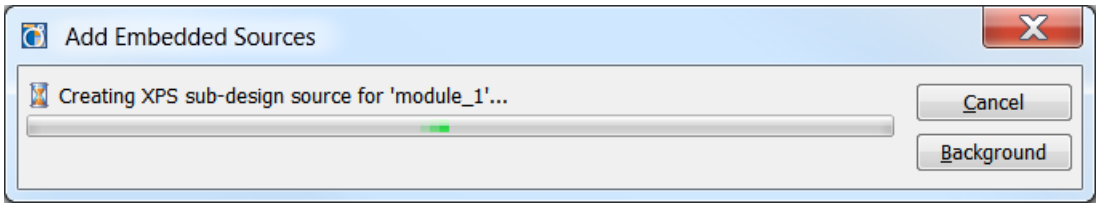

Figure 3-18 **PlanAhead Software: Add Sources: Progress Bar**

3-19 Step #19

The PlanAhead design tool creates your embedded design source project. It recognizes that you have an embedded processor system and starts the XPS tool.

When the XPS tool opens, the PlanAhead tool will go to the background. Leave the PlanAhead tool running in the background.

When the XPS tool opens it also asks you if you want to use the Base System Builder (BSB) wizard to design your new embedded system. Click **Yes** as shown in Figure 3-19.

In the BSB wizard, you can select and configure the processing system I/O interface and add default peripherals to the fabric. Xilinx recommends using the BSB wizard to create the foundation for any new embedded design project.

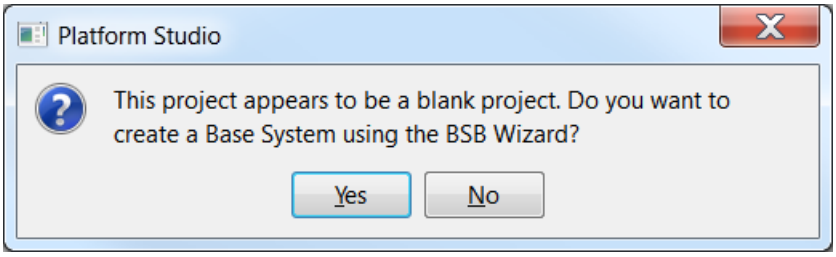

Figure 3-19 **Xilinx Platform Studio: Adding embedded source**

3-20 Step #20

When designing a new embedded system using the BSB Wizard, the first window of the BSB asks you to select whether to create an AXI-based or PLB-based system. Select AXI system and click **OK** as shown in Figure 3-20:

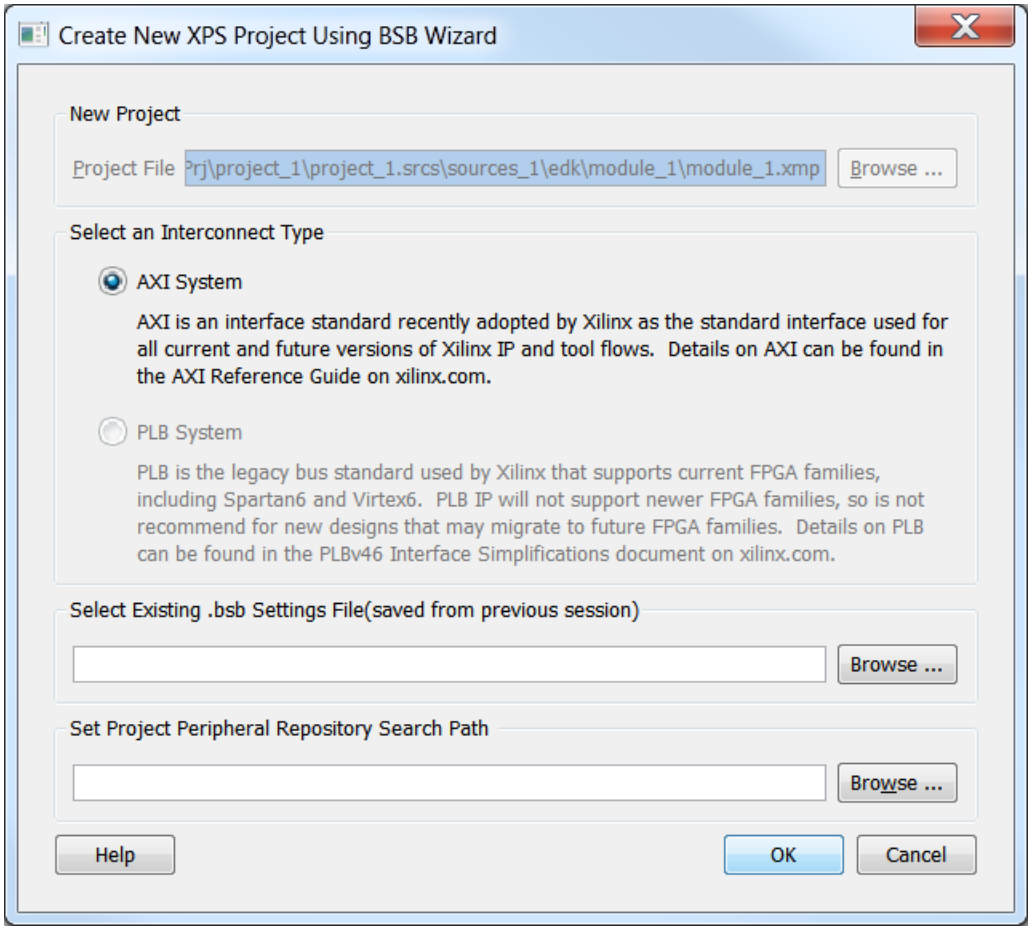

Figure 3-20 **Xilinx Platform Studio: BSB Wizard**

3-21 Step #21

The next screen in the BSB wizard asks you to select the evaluation board and processing system. Make the selections shown in Figure 3-21 and click **Next**:

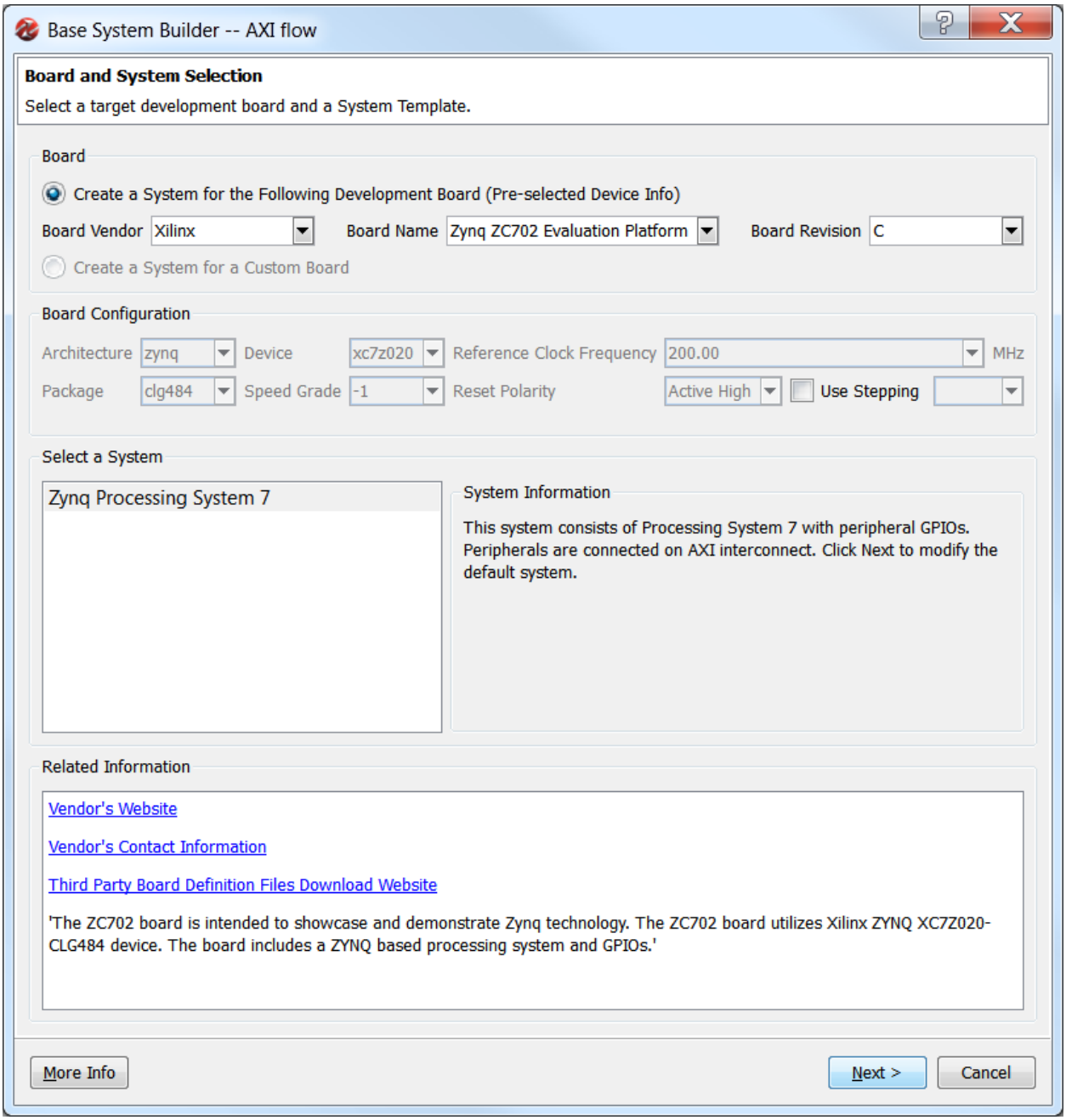

Figure 3-21 **Xilinx Platform Studio: BSB Wizard: Board and System Selection**

3-22 Step #22

From the list of included peripherals for the processing system, remove the **GPIO_SW** and **LEDs_4Bits** as shown in Figure 3-22 and click **Finish** to generate your design.

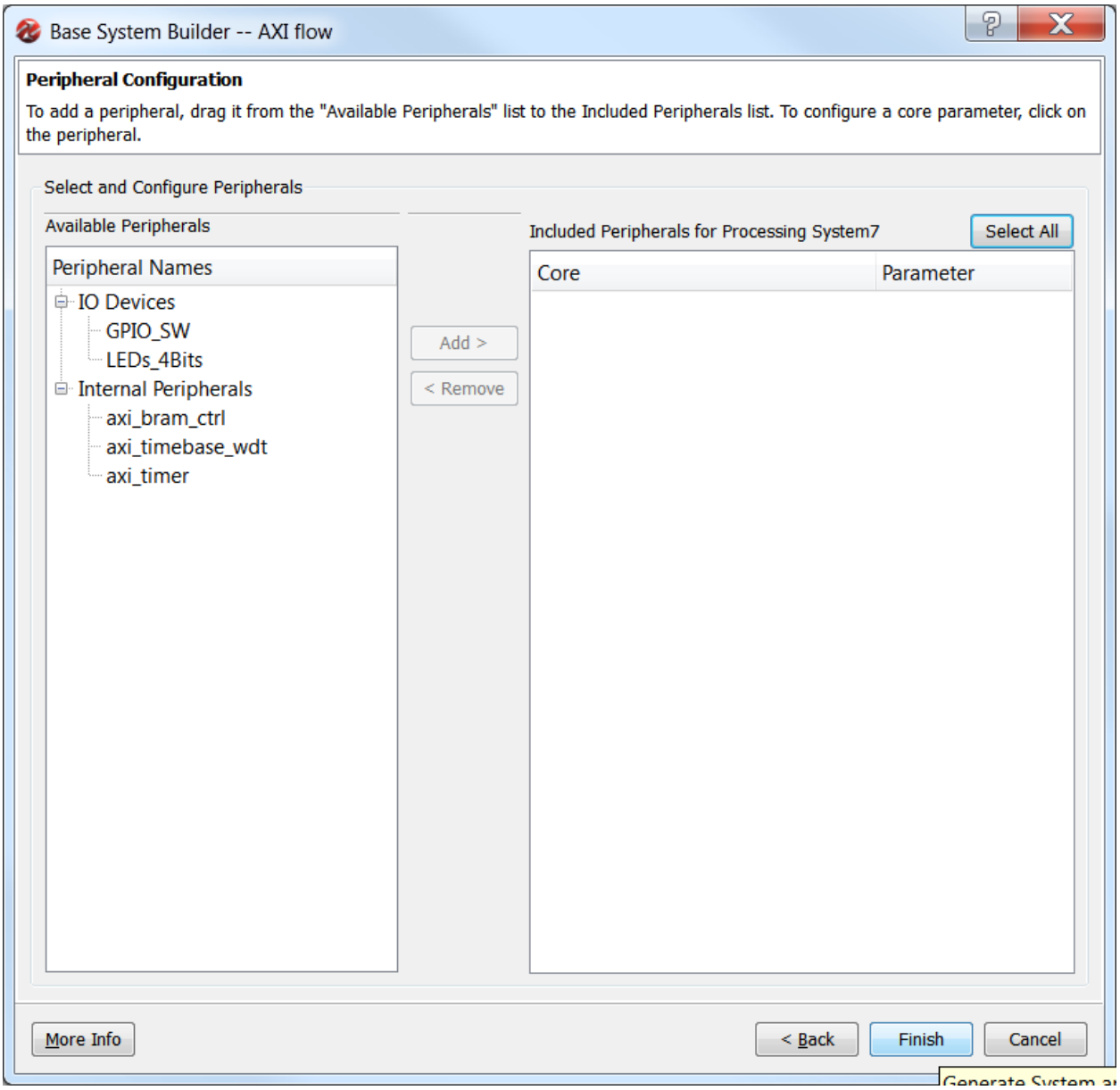

Figure 3-22 **Xilinx Platform Studio: BSB Wizard: Peripheral Configuration**

3-23 Step #23

Close the Xilinx Platform Studio tool window to go back to the PlanAhead Software and click **Yes** to confirm.

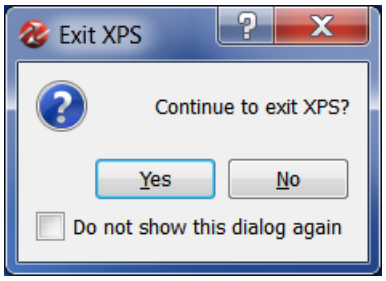

Figure 3-23 **Xilinx Platform Studio: Closing the XPS tool**

3-24 Step #24

The active PlanAhead session that has been running in the background this whole time updates with the new project settings as shown in Figure 3-24:

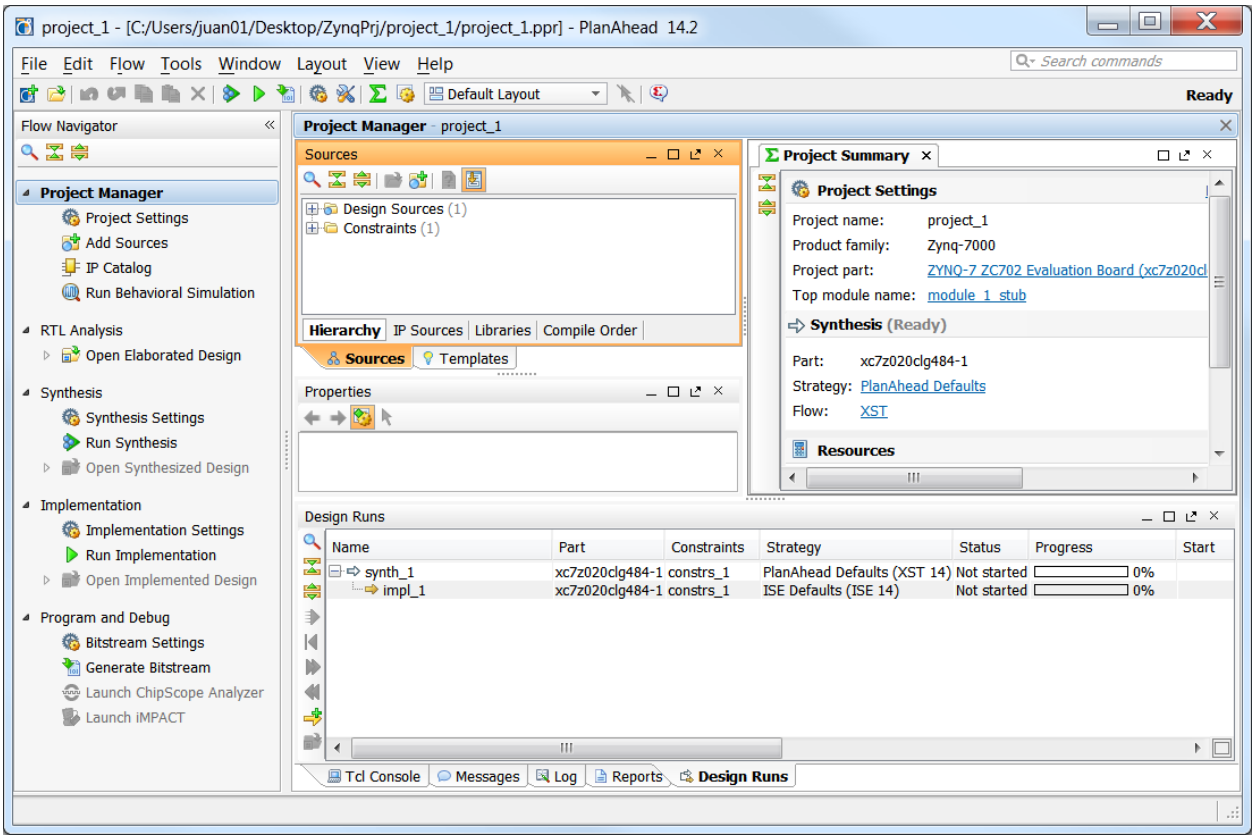

Figure 3-24 **PlanAhead Software: Project Manager**

3-25 Step #25

In order to generate the top-level module for the hardware design, expand Design Sources in the Sources pane, right-click the file module_1.xmp (or your file with extension .xmp) and select **Create Top HDL** as shown in Figure 3-25:

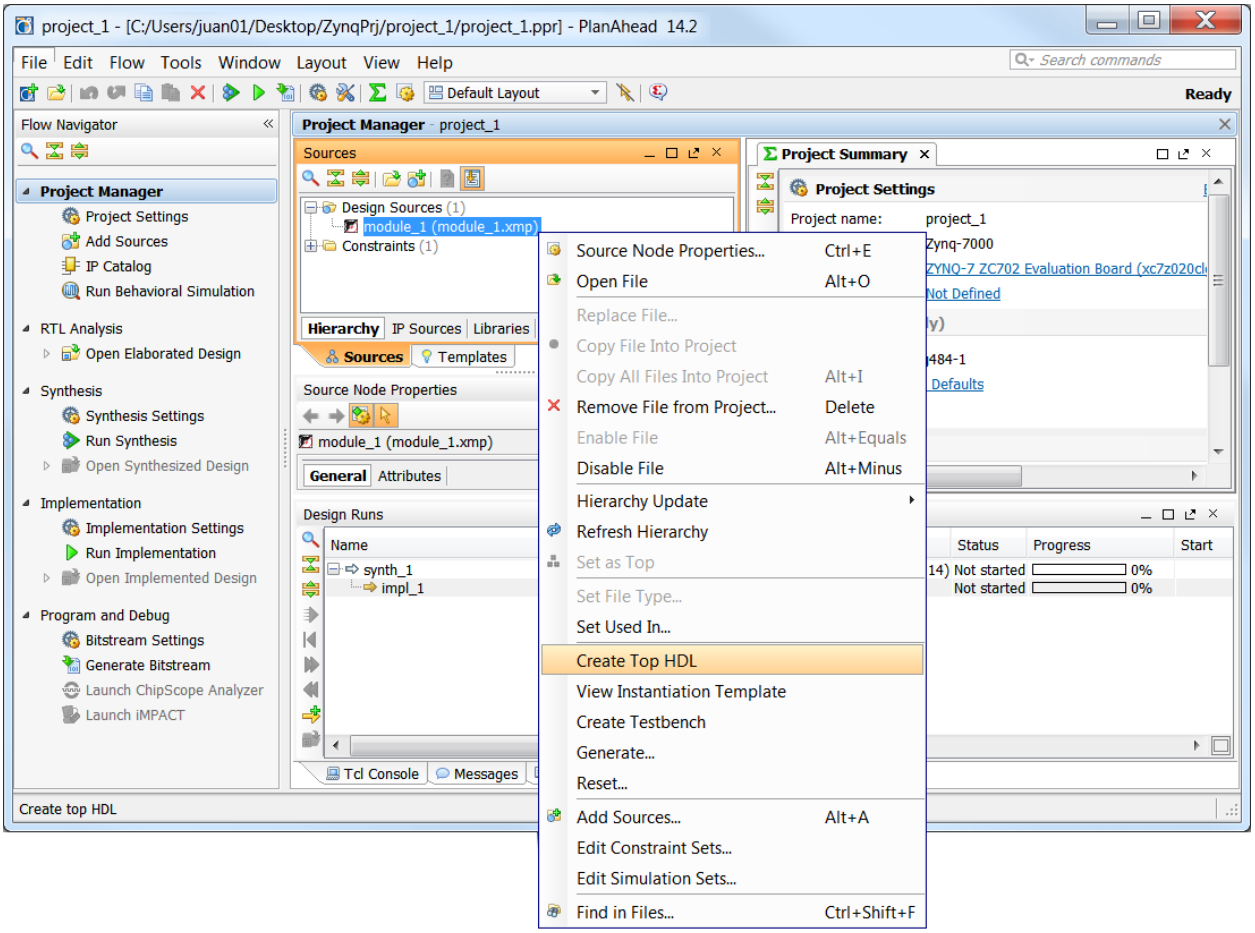

Figure 3-25: **PlanAhead Software: Generating the Top Level Hardware Design**

3-26 Step #26

Now that the hardware platform design specification for the ZC702 evaluation board has been setup, it is time to export it to the Software Development Kit (SDK).

From the PlanAhead software, select **File → Export → Export Hardware** as illustrated in Figure 3-26:

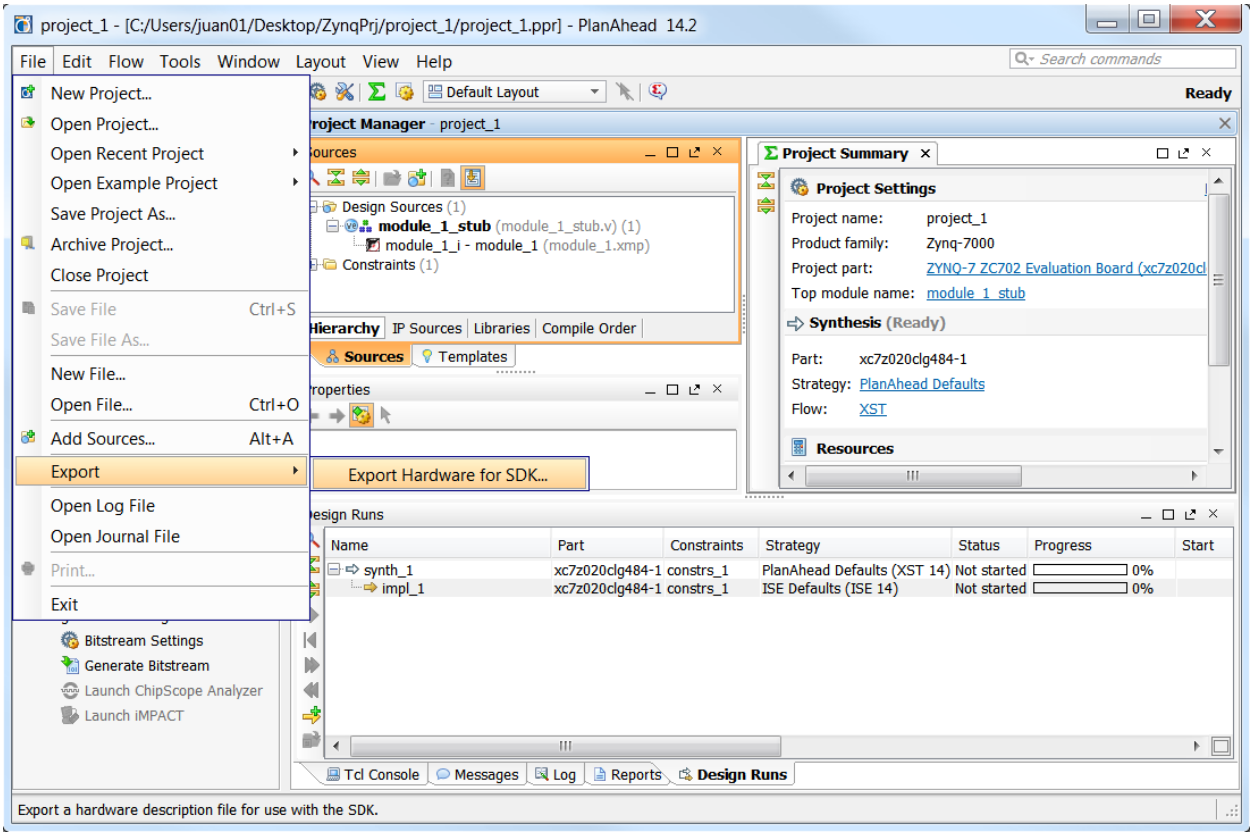

Figure 3-26: **PlanAhead Software: Exporting the Hardware Platform Design Specification**

3-27 Step #27

The Export Hardware dialog box opens. Select both, the **Export Hardware** and **Launch SDK** check boxes as shown in Figure 3-27 and click **OK** to continue.

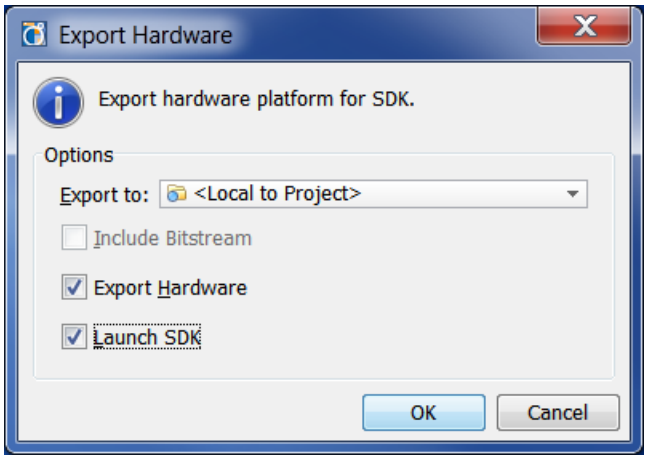

Figure 3-27 **PlanAhead Software: Exporting Hardware**

3-28 Step #28

PlanAhead exports the Hardware Platform Specification for your design (`system.xml` in this example) to the SDK. Notice that when the SDK launches, the hardware description file is automatically read in. The **system.xml** tab shows the address map for the entire Processing System as shown in Figure 3-28.

The `ps7_init.c` and `ps7_init.h` files contain the initialization code for the Zynq Processing System. These files contain initialization settings for DDR, clocks, PLLs, and MIOs. The SDK uses these settings to initialize the processing system so that applications can be run on top of the processing system.

There are some settings in the processing system that are fixed, such as MIO as per the ZC702 evaluation board.

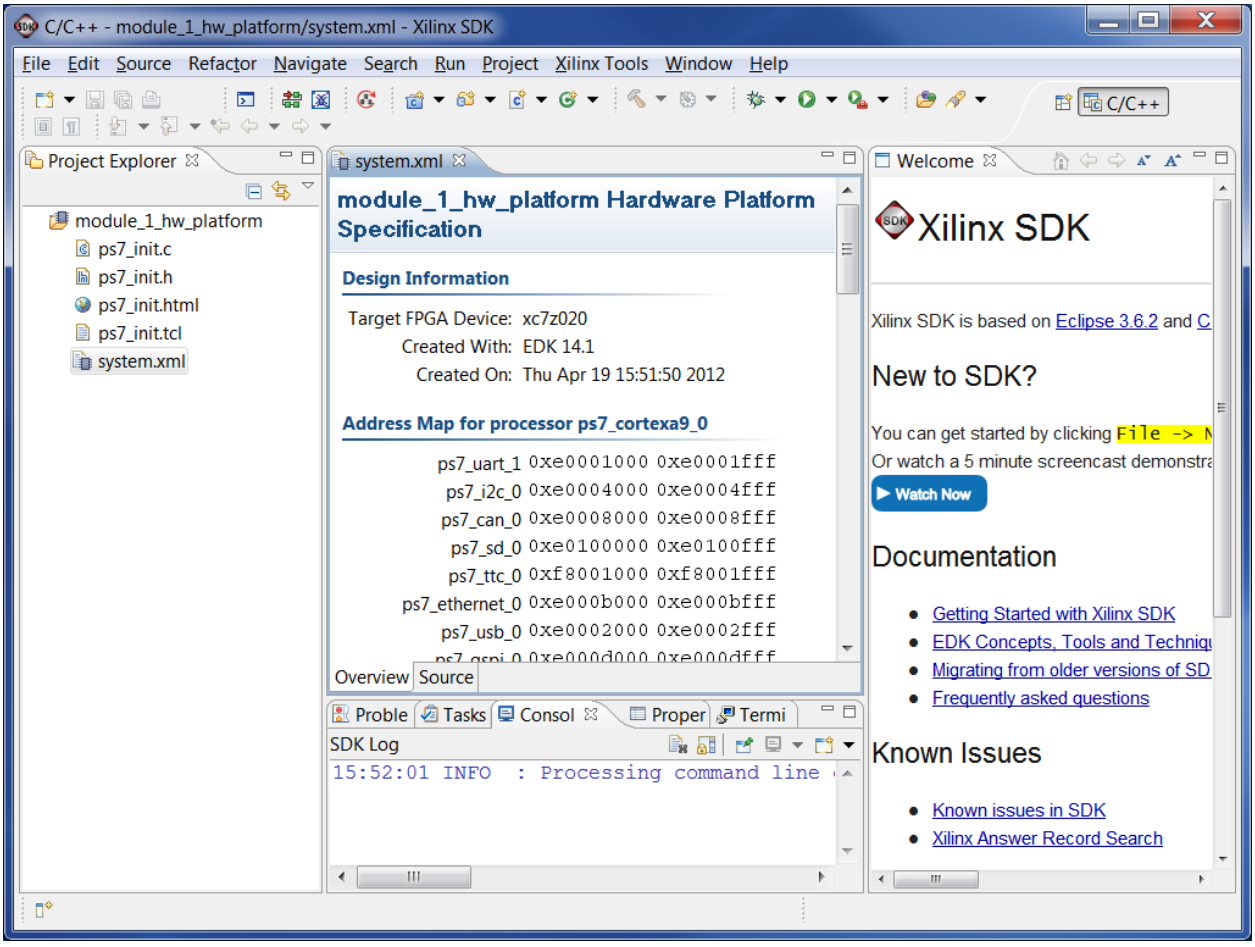

Figure 3-28 **Software Development Kit: Hardware Platform Design Specification**

3-29 Step #29

Now that the Hardware Platform Specification for the ZC702 evaluation board has been exported to the Software Development Kit, you can start developing the software for your project using the SDK. Select **File → New → Xilinx C Project** as shown in Figure 3-29:

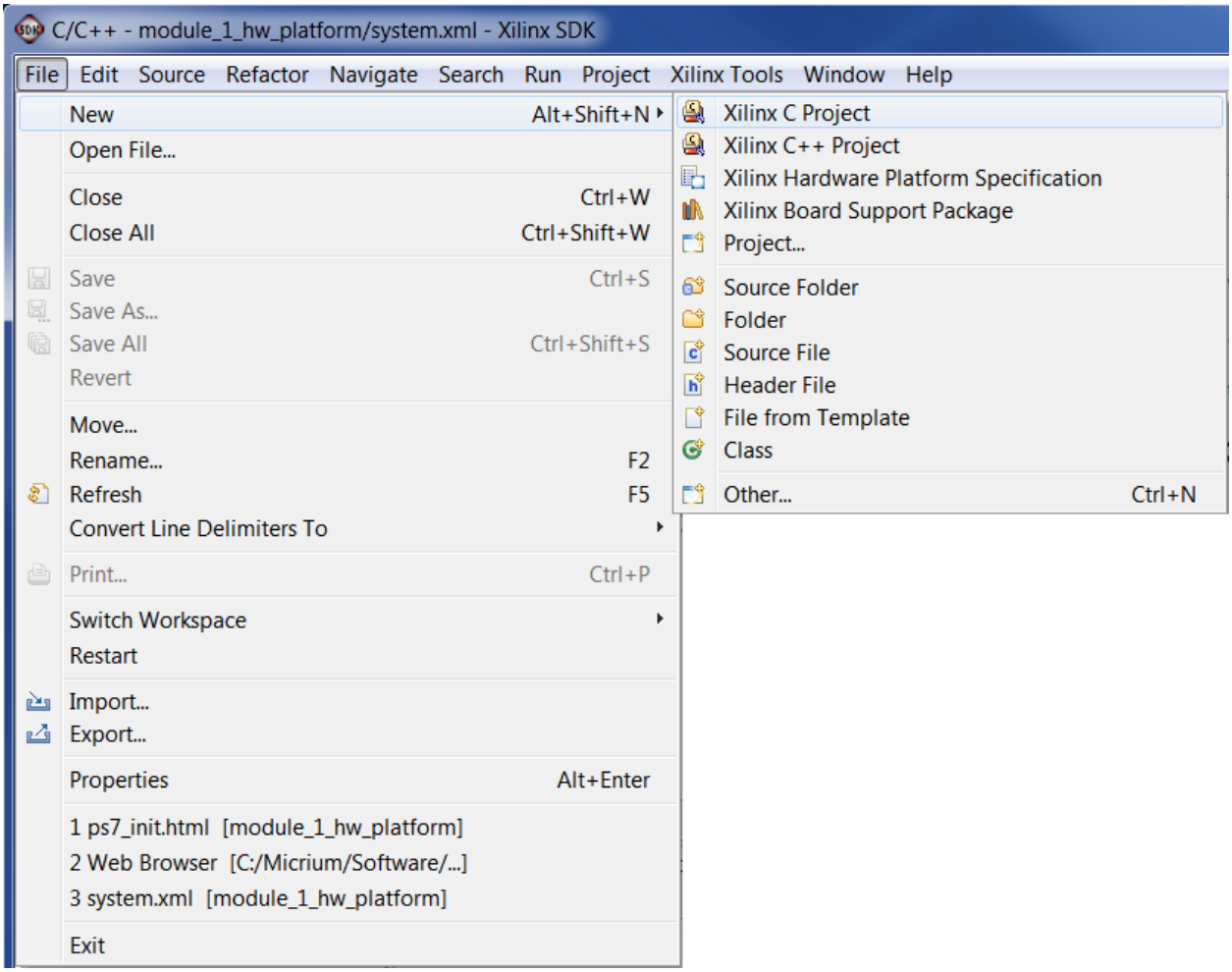

Figure 3-29 **Software Development Kit: New Xilinx C Project**

3-30 Step #30

The **New Xilinx C Project** wizard opens and displays a list of demonstration applications. Select the **μC/OS-II Demo** from Micriμm as shown in Figure 3-30 and click **Next** to continue.

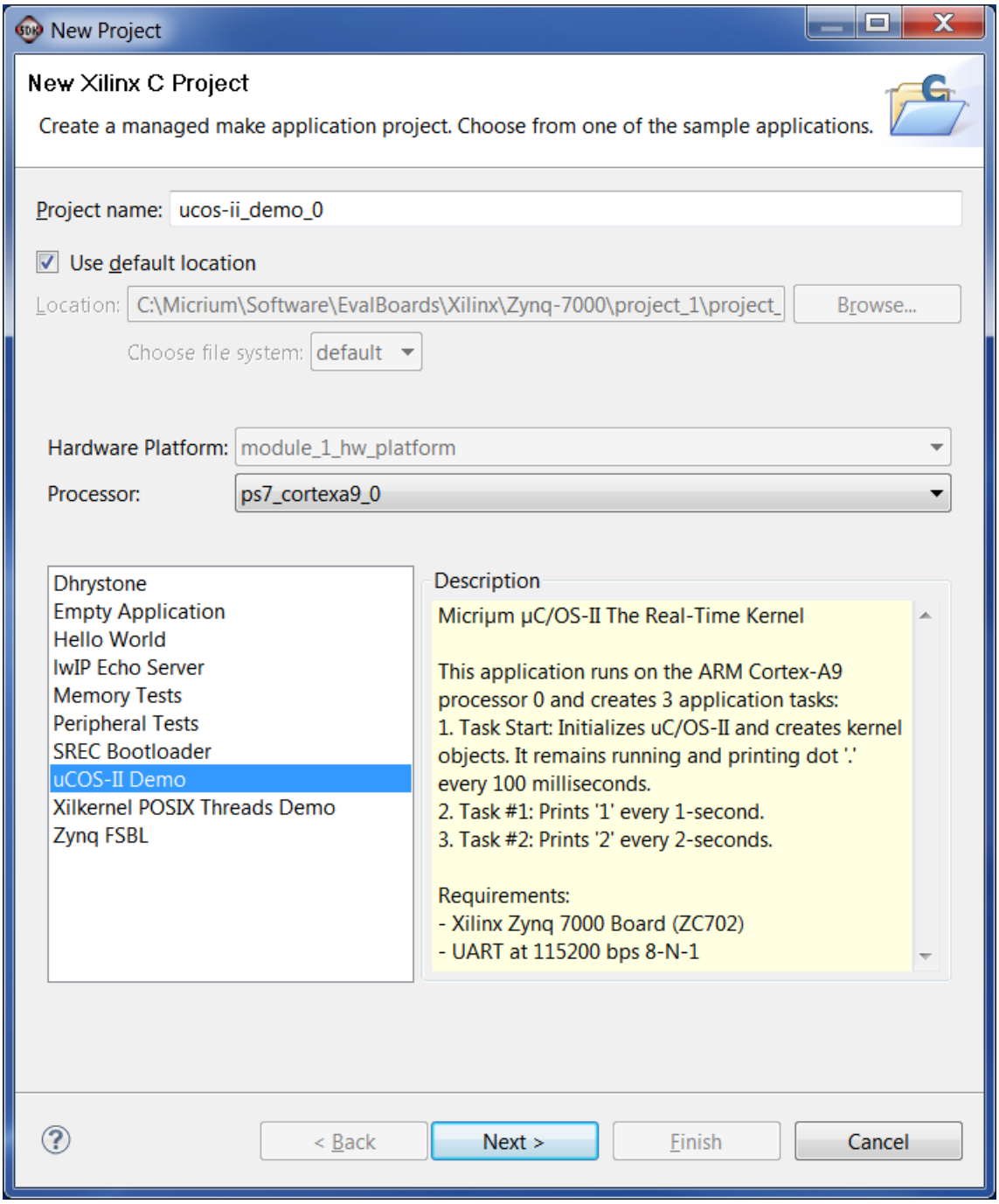

Figure 3-30 **Software Development Kit: New Xilinx C Project: μC/OS-II Demo**

Note: If your **New Xilinx C Project** window doesn't look like the image above, please verify two things before trying again from step #29:

- Verify that you have installed the Micriμm source code in the right repository path by looking at step #1.

- Make sure that the repository search path points to the folder containing the BSP and SW_APPS folders, as the SDK expects to have a folder that contains two sub folders named **bsp** and **sw_apps** as shown in Figure 3-1.

.

- Verify that the SDK has refreshed the repositories path. In order to do so, click **Xilinx → Tools → Repositories → Rescan Repository Paths** and click the **Apply** button.

3-31 Step #31

In this step the BSP for the project is selected. By default the μC/OS-II demo application is configured to use the BSP provided by Micriμm. Keep the default options as shown in Figure 3-31 and click **Finish** to create your new μC/OS-II demo application.

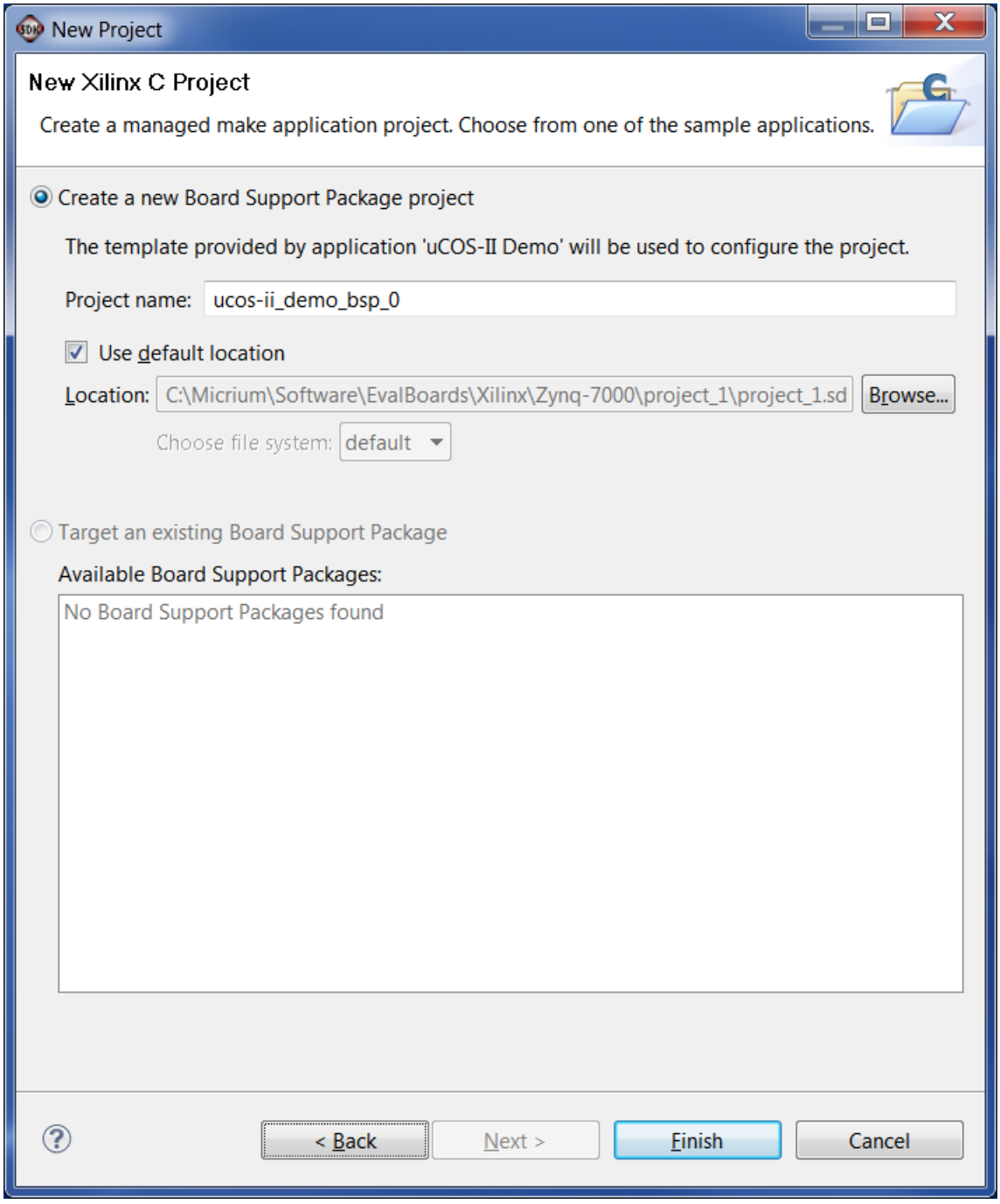

Figure 3-31 **Software Development kit: New Xilinx C Project: μC/OS-II Demo**

3-32 Step #32

As soon as you click **Finish**, the μC/OS-II demo application and its BSP are both automatically compiled but the console reports errors as shown in Figure 3-32.

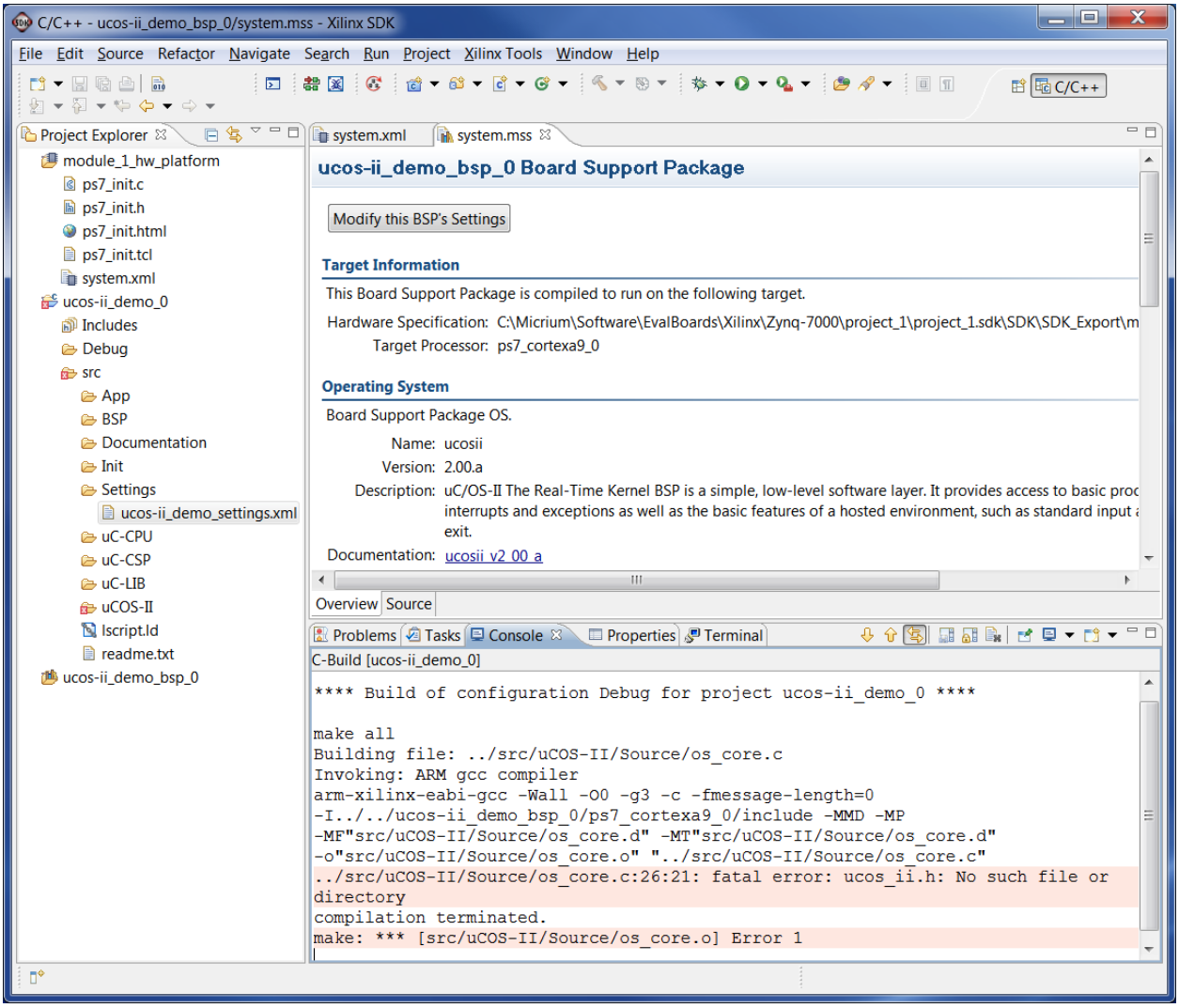

Figure 3-32 **Software Development Kit: Compiling the μC/OS-II Demo**

3-33 Step #33

In order to compile the μC/OS-II demo application you need to import the project settings file that contains the include paths for the μC/OS-II source code. Figure 3-32 in the previous page shows the SDK Project Explorer. The file `ucos-ii_demo_settings.xml` in your SDK Project Explorer contains such settings. Right-click over the file `ucos-ii_demo_settings.xml` and copy the file path to your clipboard as shown in Figure 3-33.

Click **Cancel** to close the window.

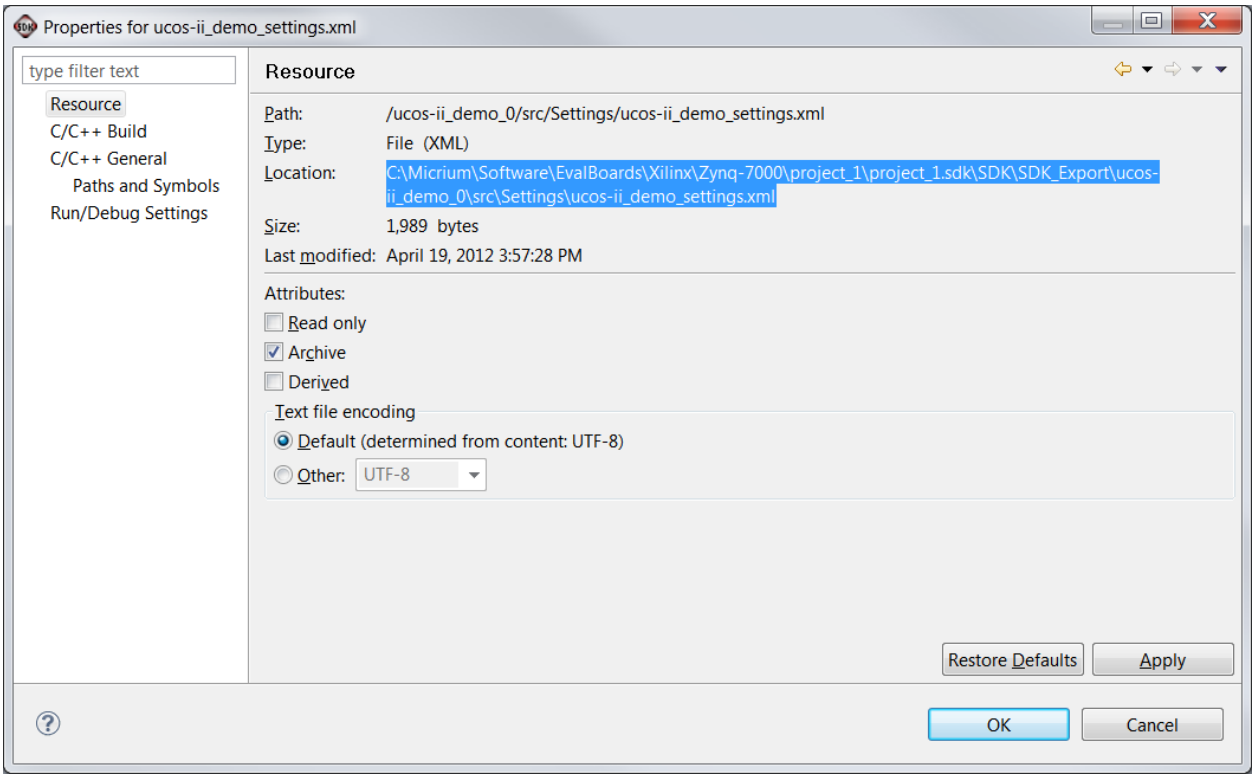

Figure 3-33 **Software Development Kit: Project Settings**

3-34 Step #34

Import the settings from the XML file by opening the project properties. From your project explorer shown in Figure 3-32, make right-click over the name of your project **ucos-ii_demo_0** and select **Properties**. The **Project Properties** window will open as shown in Figure 3-34 below.

Expand the node **Paths and Symbols** under the **C/C++ General** tree and click the **Import Settings** button.

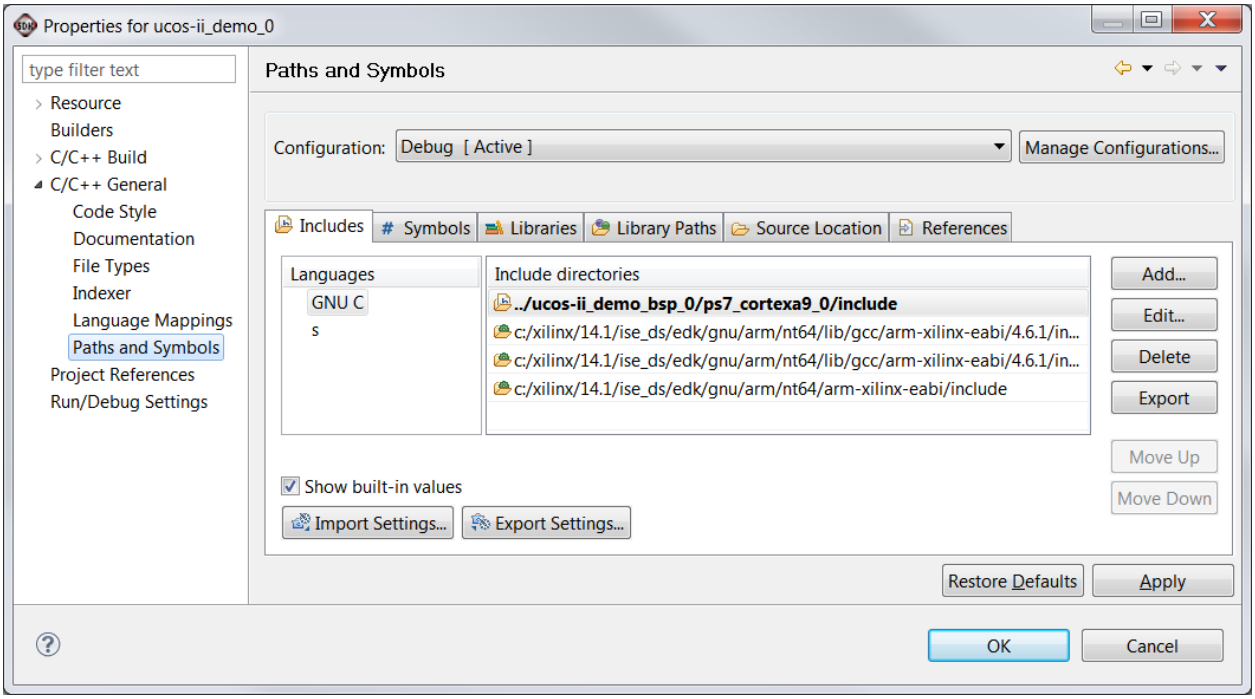

Figure 3-34 **Software Development Kit: Project Settings**

3-35 Step #35

Paste from your clipboard the file path to `ucos-ii_demo_settings.xml` inside the **Settings File** text box as shown in Figure 3-35 or browse to the file. Make sure that the **ucos-ii_demo_0** project and the **Debug (Active)** configuration options are selected.

Click **Finish** to import the settings.

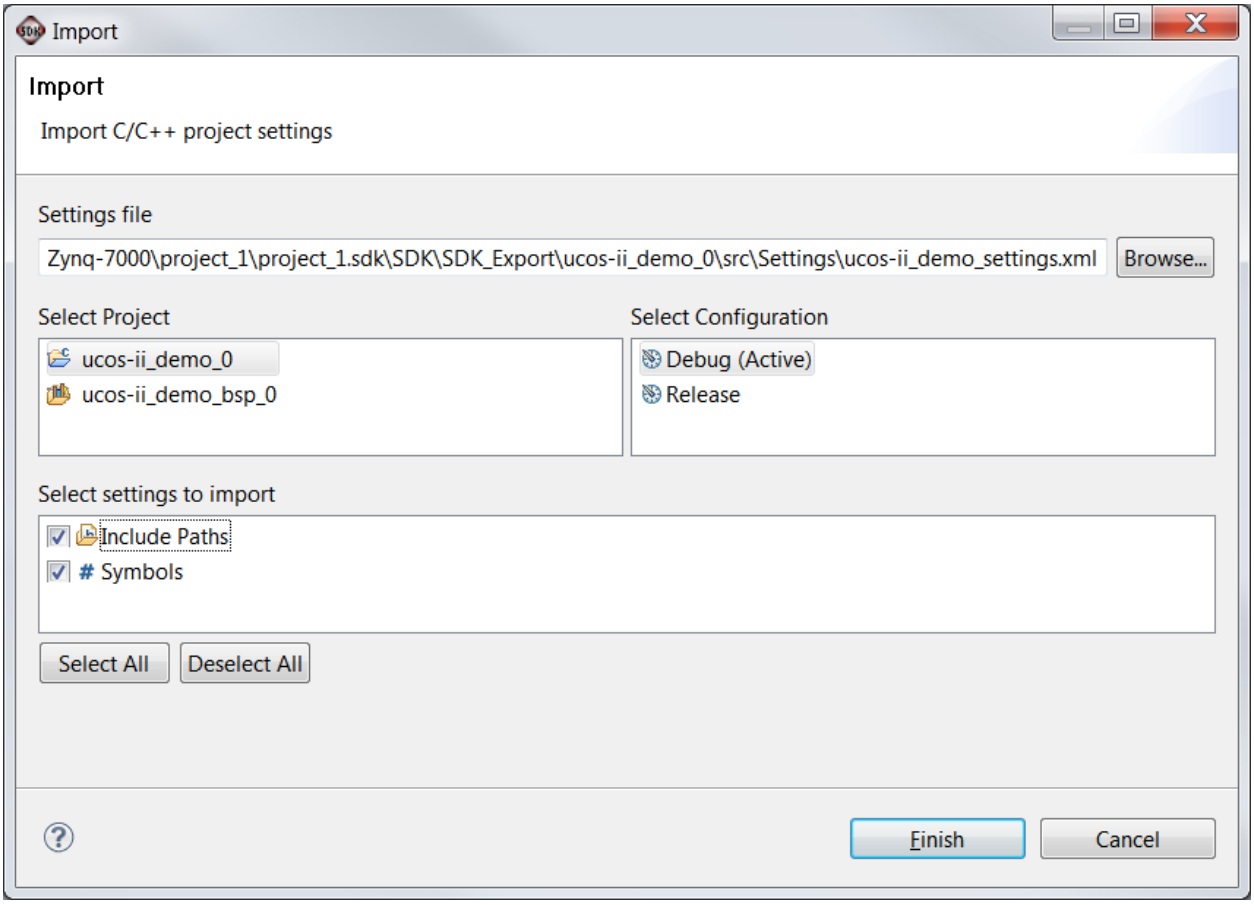

Figure 3-35 **Software Development Kit: Importing Project Settings**

3-36 Step #36

From the SDK Project Explorer right-click over the project name **ucos-ii_demo_0** and select **Build** to compile the project. This time, the console should report a successful build as shown in Figure 3-36:

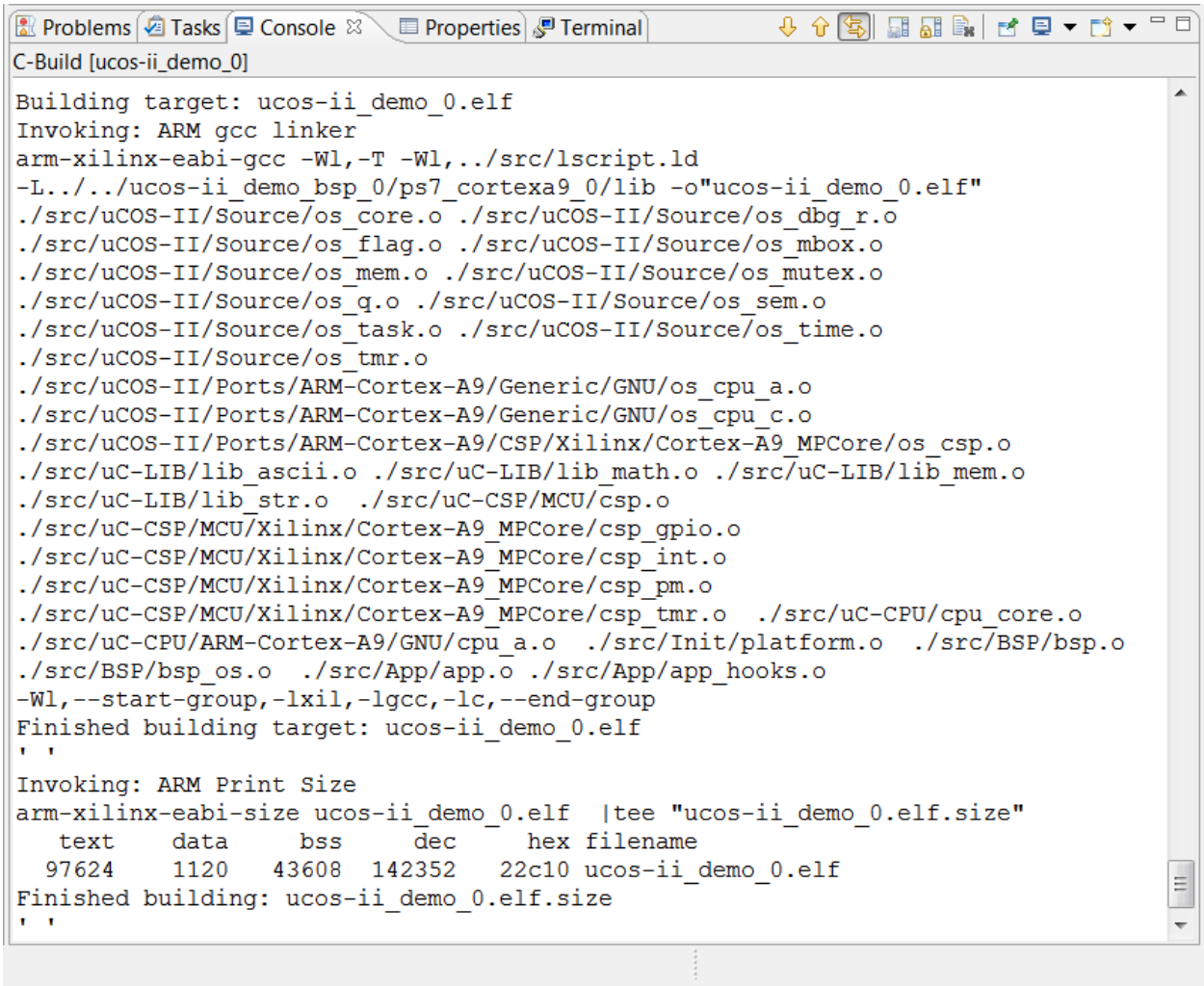

Figure 3-36 **Software Development Kit: Console: Successful Build**

3-37 Step #37

Before powering up the Xilinx ZC702 evaluation board, verify that the jumpers are configured as shown in Figure 3-37:

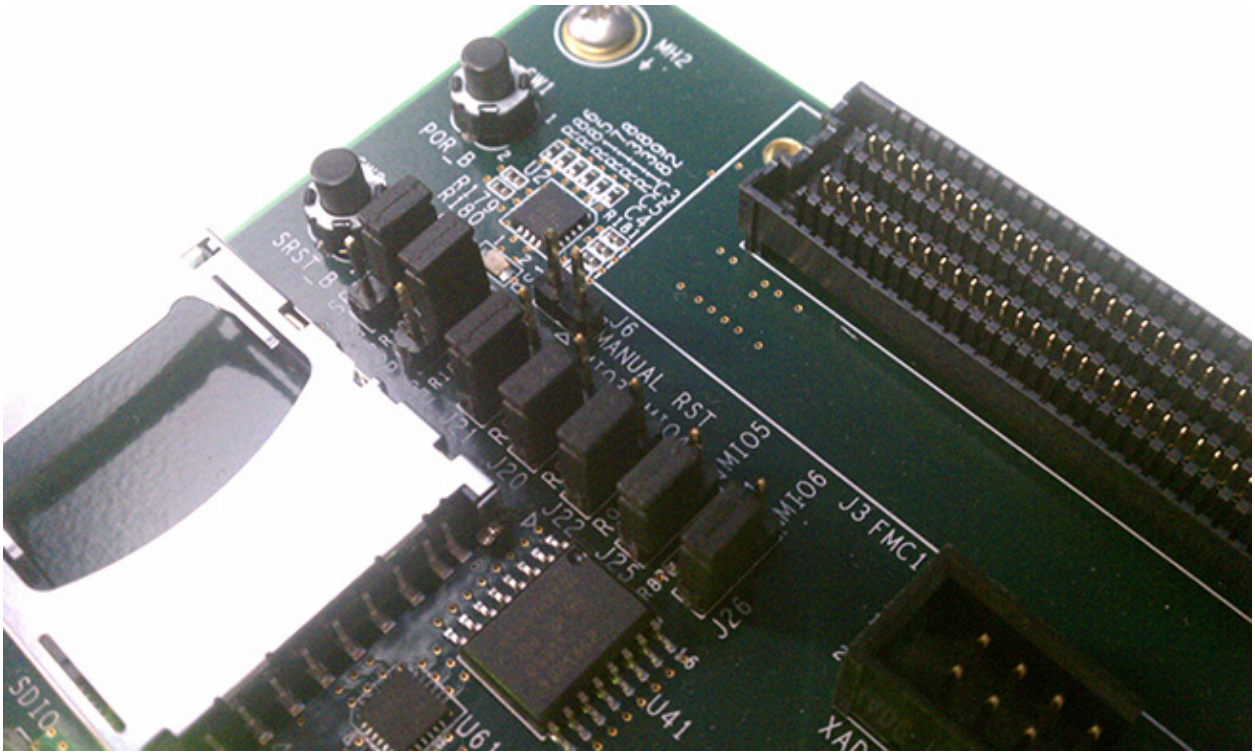

Figure 3-37 **ZC702 Evaluation Board: Jumper settings**

3-38 Step #38

Make sure that the other set of jumpers are configured as shown in Figure 3-38:

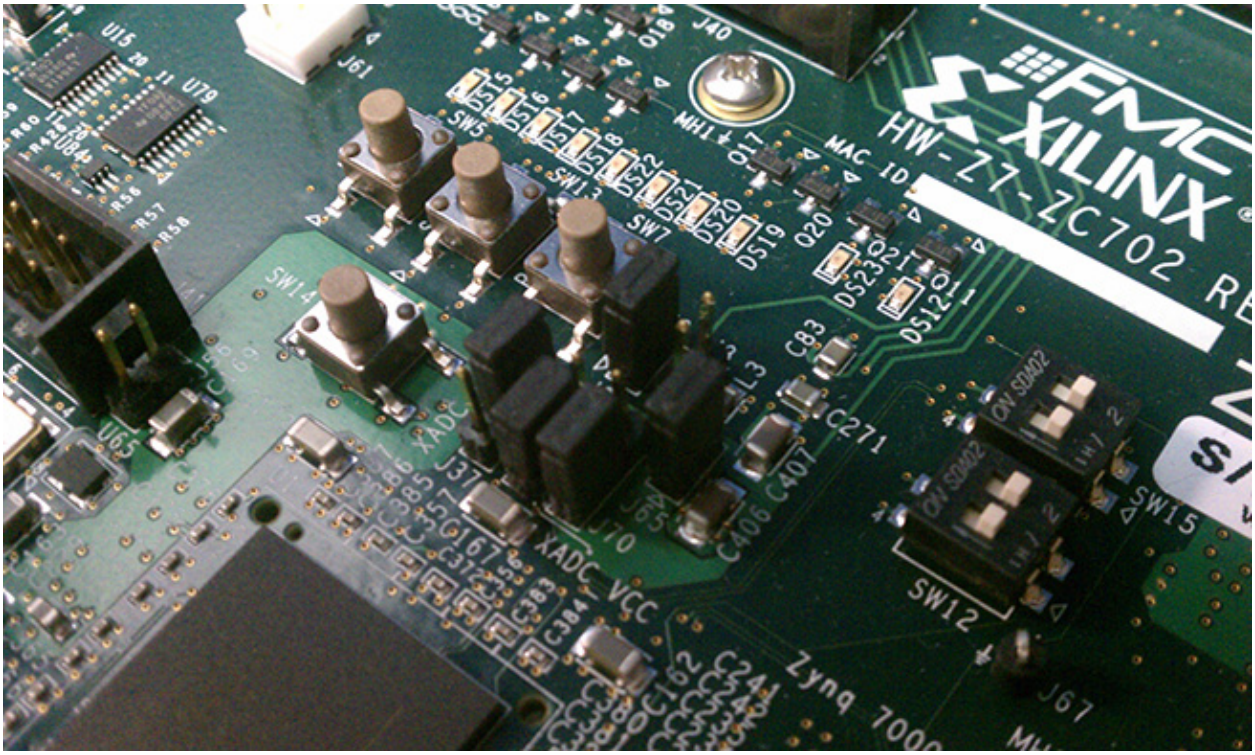

Figure 3-38 **ZC702 Evaluation Board: Jumper settings**

3-39 Step #39

Connect a USB cable on **J17** as shown in Figure 3-39. This USB cable will be used to create a virtual COM port in your Windows PC.

Connect the other end of the cable to a USB port in your **PC** and search for the number of the new COM port by looking at your Windows **Control Panel → System → Device Manager**.

Open a serial terminal program such as HyperTerminal, TeraTerm Pro or PuTTY and create a new serial connection to this virtual COM port. Configure the serial connection to a baudrate of 115200 bps and 8-N-1.

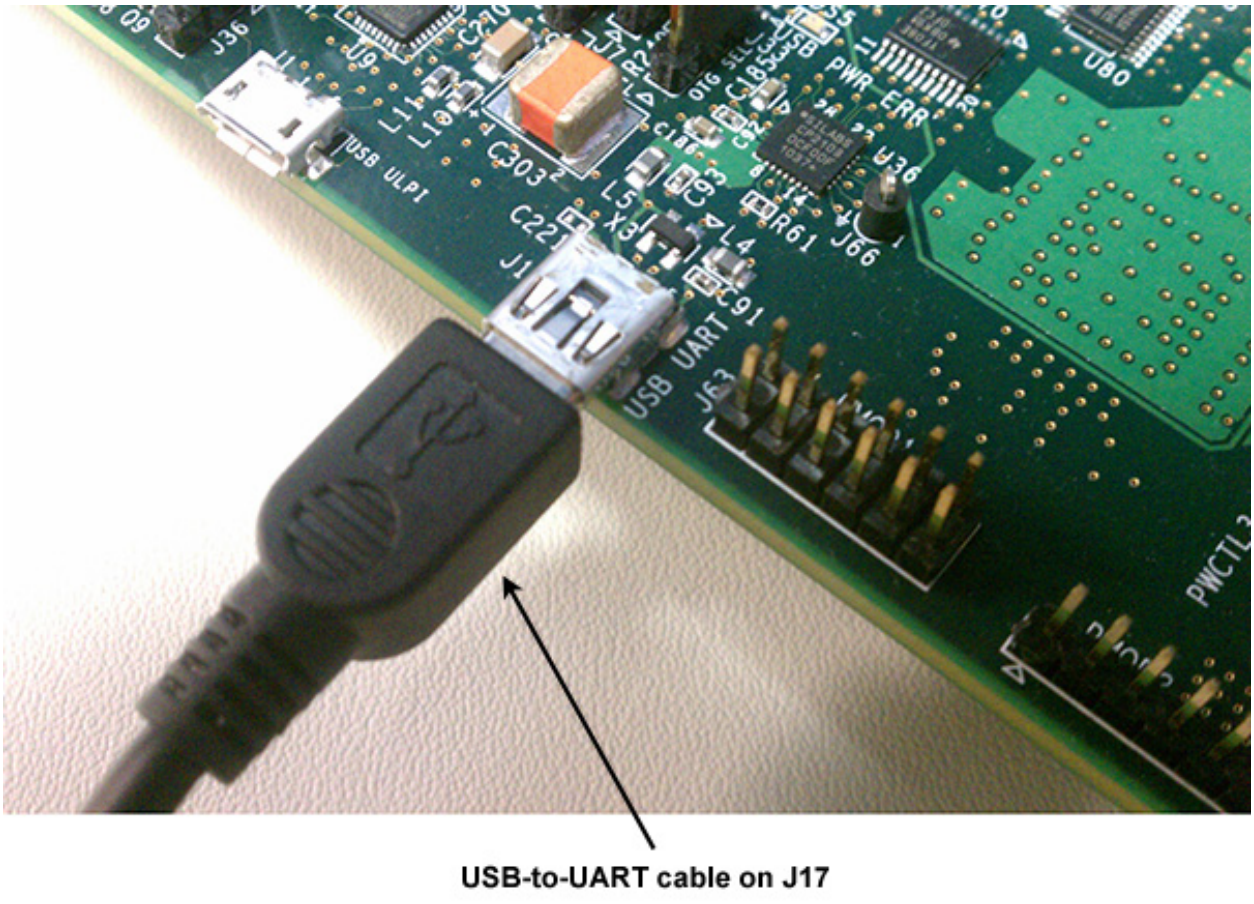

Figure 3-39 **ZC702 Evaluation Board: Connections**

3-40 Step #40

The Xilinx Platform Cable not only allows you to download the firmware and program the FPGA, but also allows you to debug the code through JTAG. Connect the **Xilinx Platform Cable** to **J2** on the board and the other end of the cable to one USB port in your **PC**. Windows should recognize the device automatically assuming you already installed the Zynq Tools successfully.

Turn the power switch ON as shown in Figure 3-40:

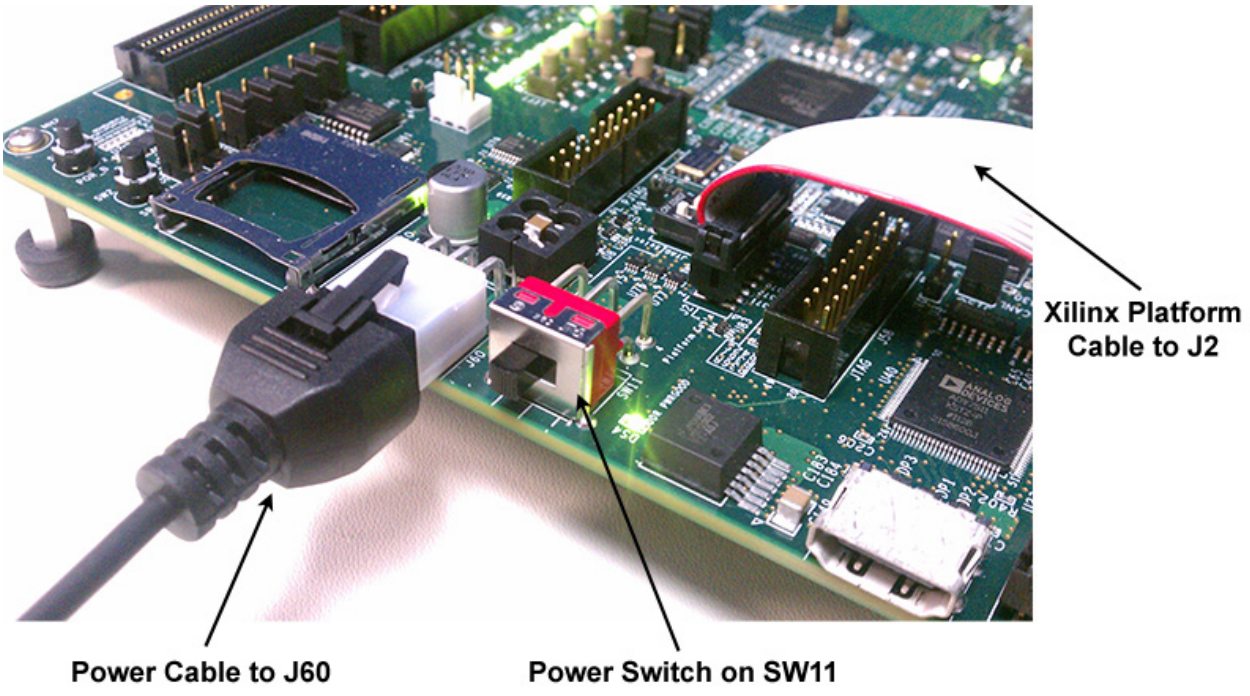

Figure 3-40 **ZC702 Evaluation Board: Connections**

3-41 Step #41

Once your hardware is all setup, you are ready to start running the μC/OS-II demo application.

From the SDK Project Explorer, right-click over the new ELF file under the Binaries group as shown in Figure 3-41:

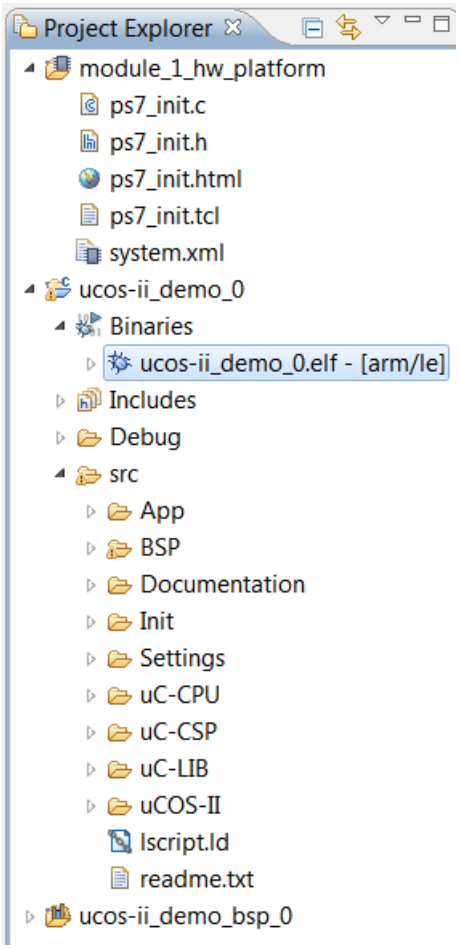

Figure 3-41 **Software Development Kit: ELF file**

3-42 Step #42

From the ELF file context menu, select **Debug as → Debug Configurations** as shown in Figure 3-42:

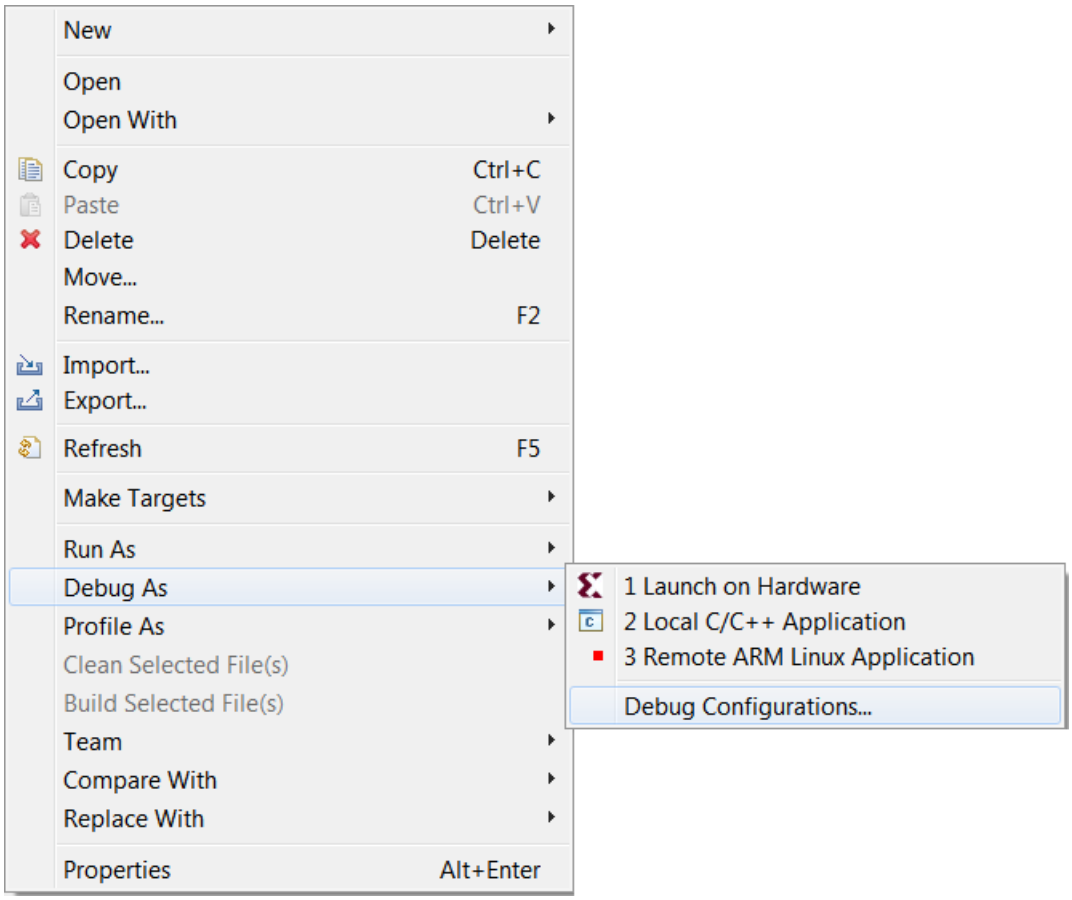

Figure 3-42 **Software Development Kit: Debugging**

3-43 Step #43

From the Debug Configurations window, select **Xilinx C/C++ ELF** and click the **New launch configurations** button as shown in Figure 3-43:

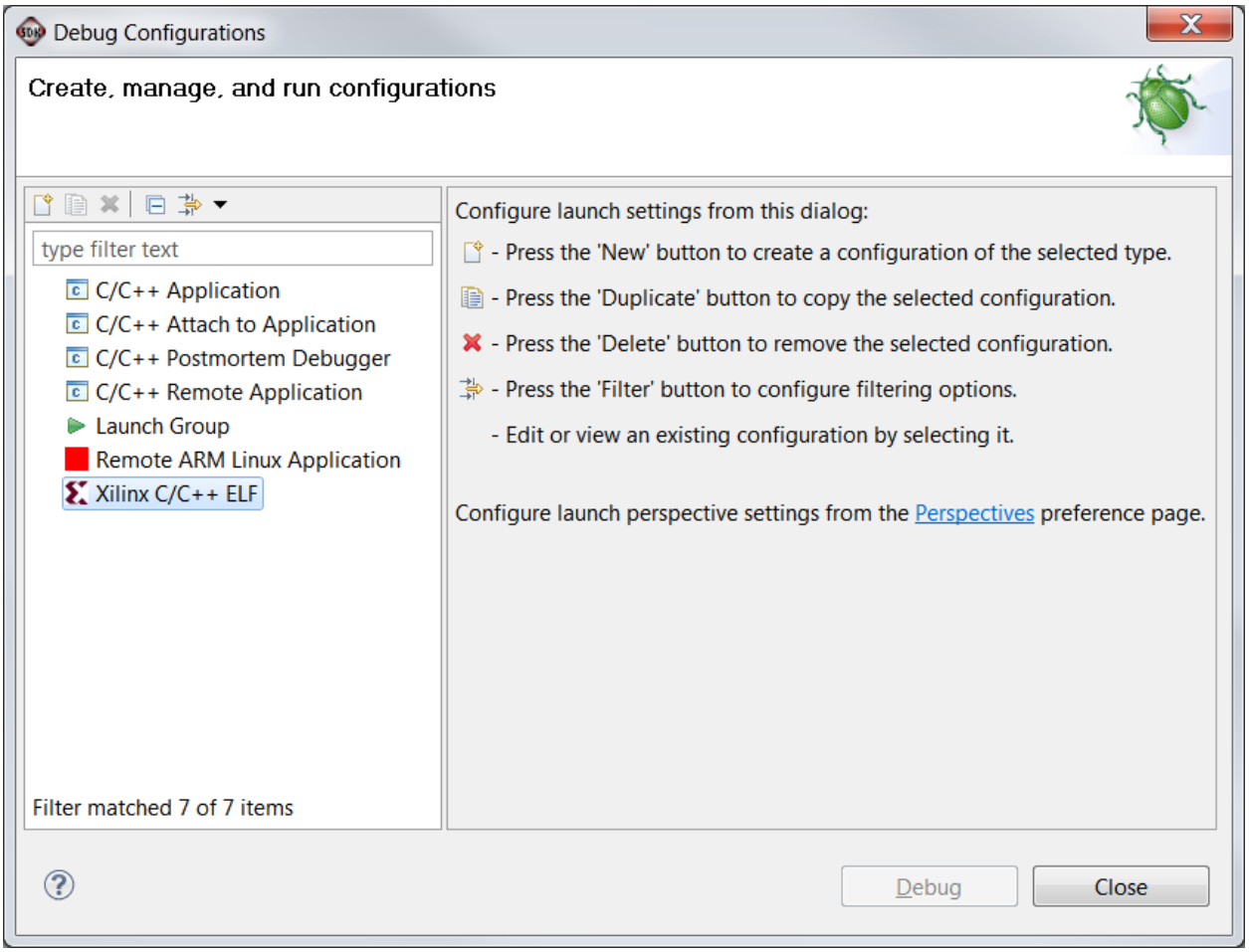

Figure 3-43 **Software Development Kit: New Launch Configuration**

3-44 Step #44

The new debug configuration is created and named **ucosii_demo_0 Debug**. The configuration settings associated with the application are pre-populated in the main tab of this launch configuration as shown in Figure 3-44. Make sure the evaluation board is powered ON and click **Debug** to start debugging.

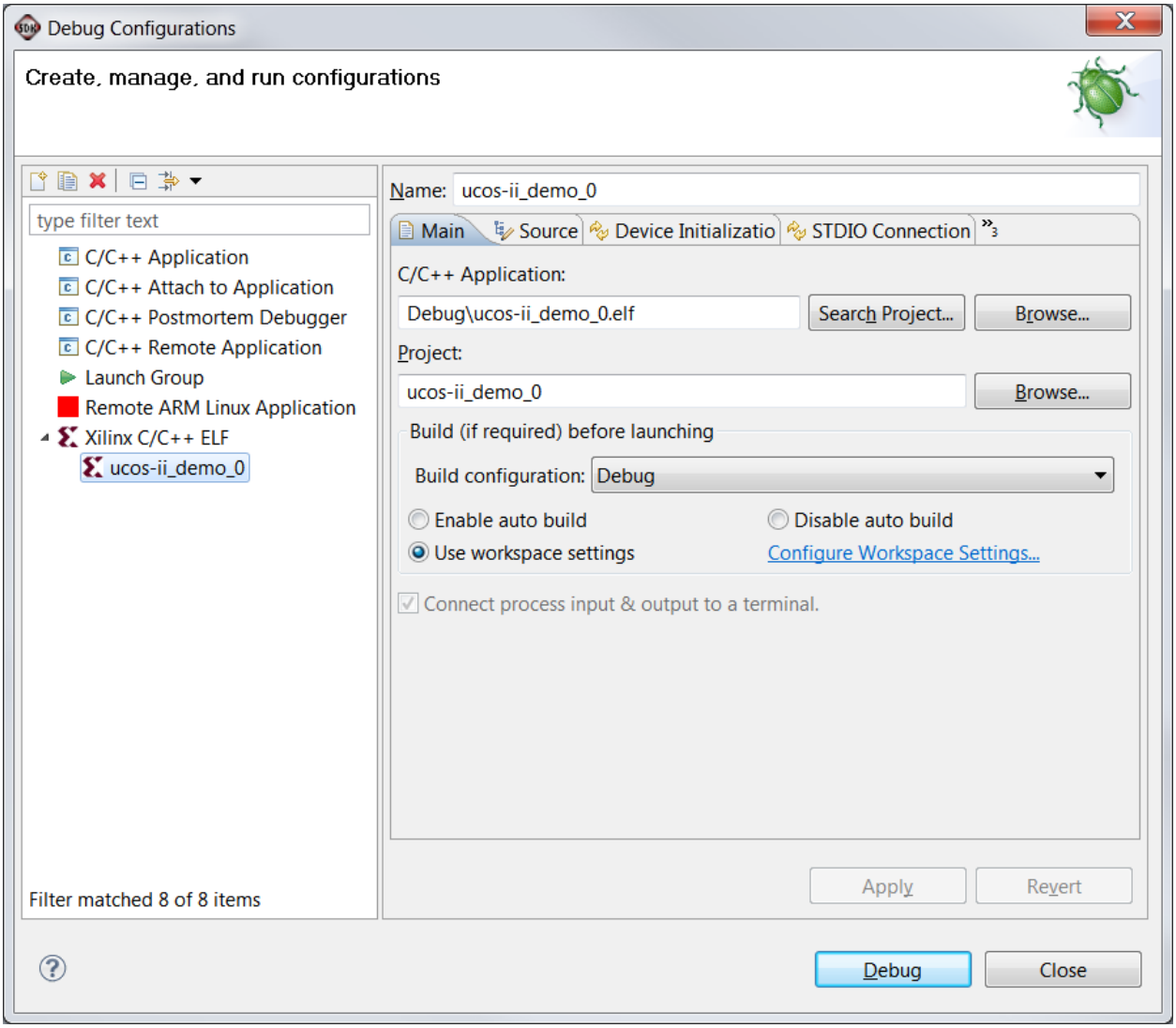

Figure 3-44 **Software Development Kit: Launching a Debugging Session**

3-45 Step #45

Because in this application μC/OS-II only runs in one of the processors, you can click **OK** to continue in the message box shown in Figure 3-45

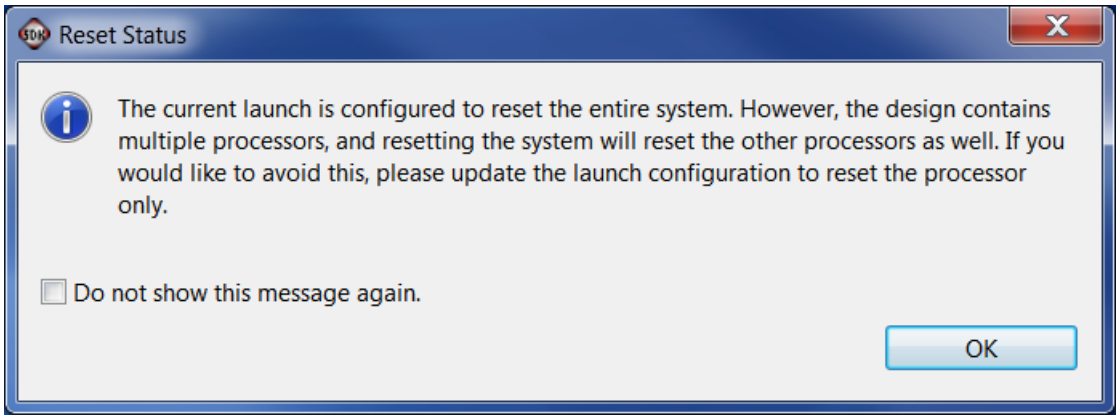

Figure 3-45 **Software Development Kit: Message Box**

3-46 Step #46

Launching the debug session takes some time. You can follow the progress of the launching process by looking at the progress bar at the bottom right corner of the SDK as shown in Figure 3-46:

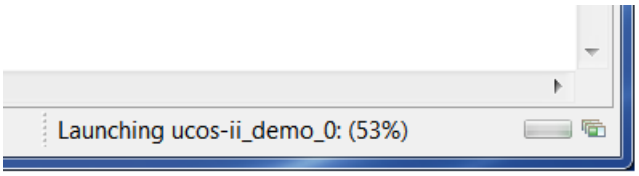

Figure 3-46 **Software Development Kit: Debug Session Launching Progress**

Note: The debug session launching process should not take more than one minute. If you find that it is taking more than one minute, something is wrong and you need to do the following things:

- Click the **Stop** debugging session red button
- Power **OFF** the ZC702 Evaluation Board
- **Unplug** the Xilinx Platform Cable from the PC
- Close the Software Development Kit (SDK)

- Open Windows Task Manager and **kill** any process named **xmd**
- **Re-launch** the SDK from the PlanAhead software (no need to include the hardware configuration this time).
- **Reconnect** the Xilinx Platform Cable to the PC
- Turn the ZC702 Evaluation Board power **ON**
- **Re-start** a debugging session by making rich-click over the ELF file and selecting the debugging launch configuration you created in steps 43 and 44.

3-47 Step #47

The debugging session takes some time to launch and eventually a message box like the one shown in Figure 3-47 shows. Click **Yes** to continue:

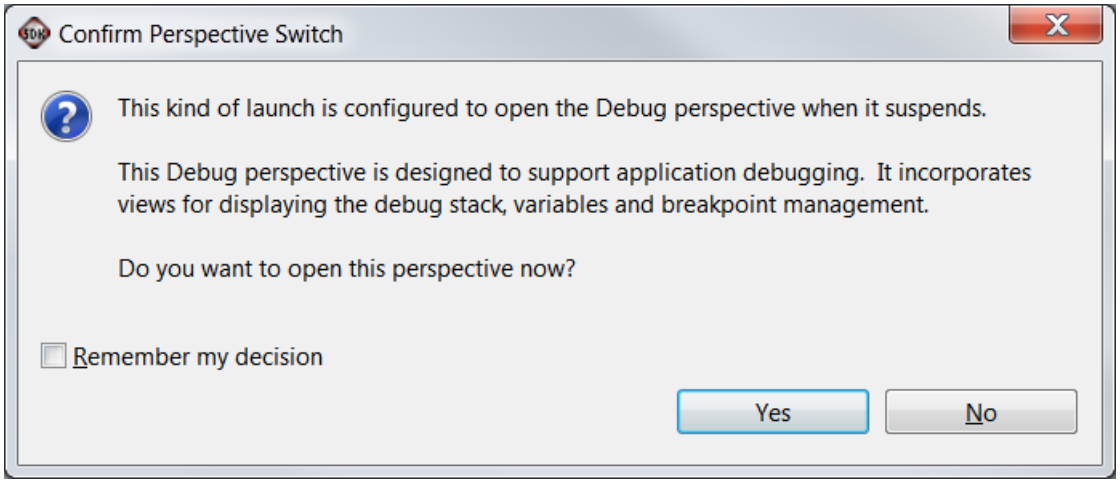

Figure 3-47 **Software Development Kit: Debug Perspective**

3-48 Step #48

The SDK switches to the debugging perspective and the cursor points at the `main()` function as shown in Figure 3-48. You can use the **Run** menu to step through the code or run freely the application.

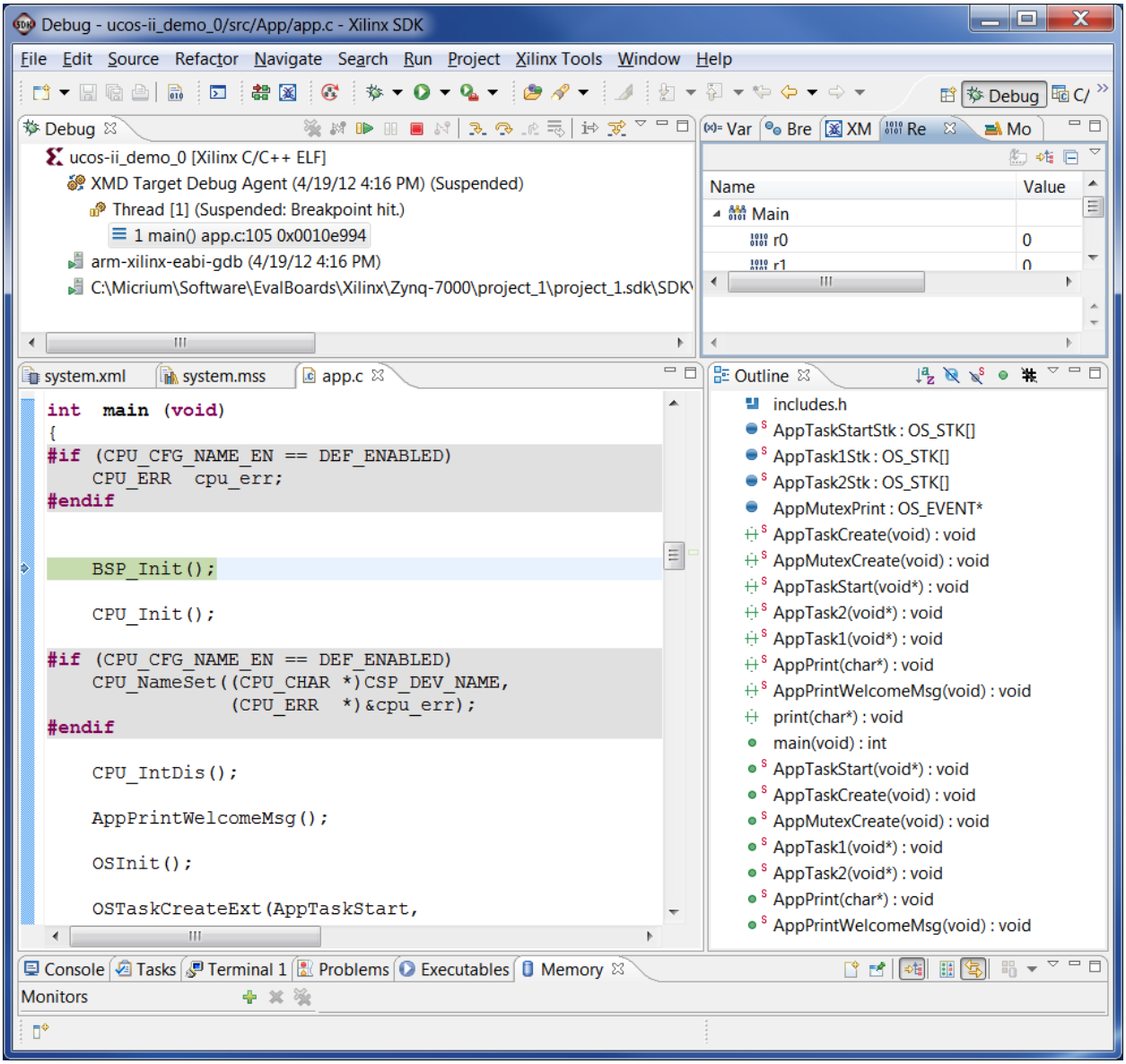

Figure 3-48 **Software Development Kit: Stepping through the µC/OS-II Demo**

3-49 Step #49

As soon as you run the application you should see the following output through the serial interface in your PC:

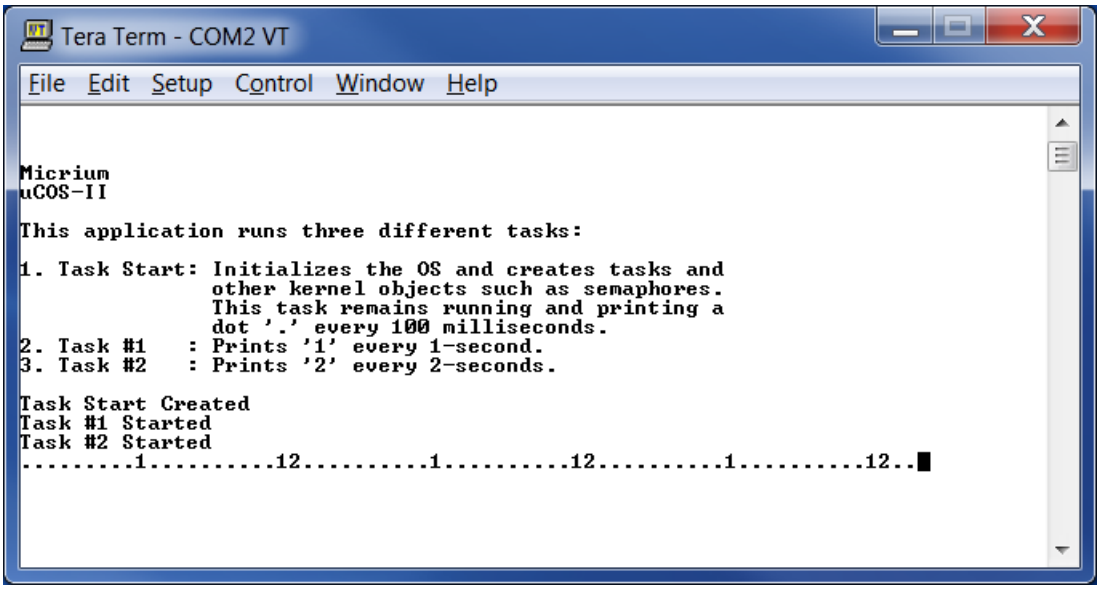

Figure 3-49 **Serial Terminal: μC/OS-Demo output**

Note: If you do not see the output in your serial terminal as shown in Figure 3-49, please verify three things:

- Make sure the serial terminal is configured to 115200 bps 8-N-1
- Verify that the USB-to-UART bridge driver from Silicon Labs is installed in your PC and working properly.
- Confirm that the serial terminal is opening a connection to the correct COM port number.

4. Files Manifest

4.1 Folders App, BSP and Documentation

The **App** folder illustrated in Figure 4-1 contains all the files that are specific for the application. In this example, `app.c` is where all the application code is while the rest of `*_cfg.h` files contain a set of `#defines` to configure not only μC/OS-II but the rest of Micriμm modules such as μC/CPU, μC/CSP and μC/LIB.

The **BSP** folder contains the Board Support Package files. Except for the `bsp_os.*` files, most of these files contain function wrappers for the Xilinx drivers.

The **Documentation** folder contains not only this walkthrough guide, but also the μC/OS-II Configuration Manual and μC/OS-II Reference Manual.

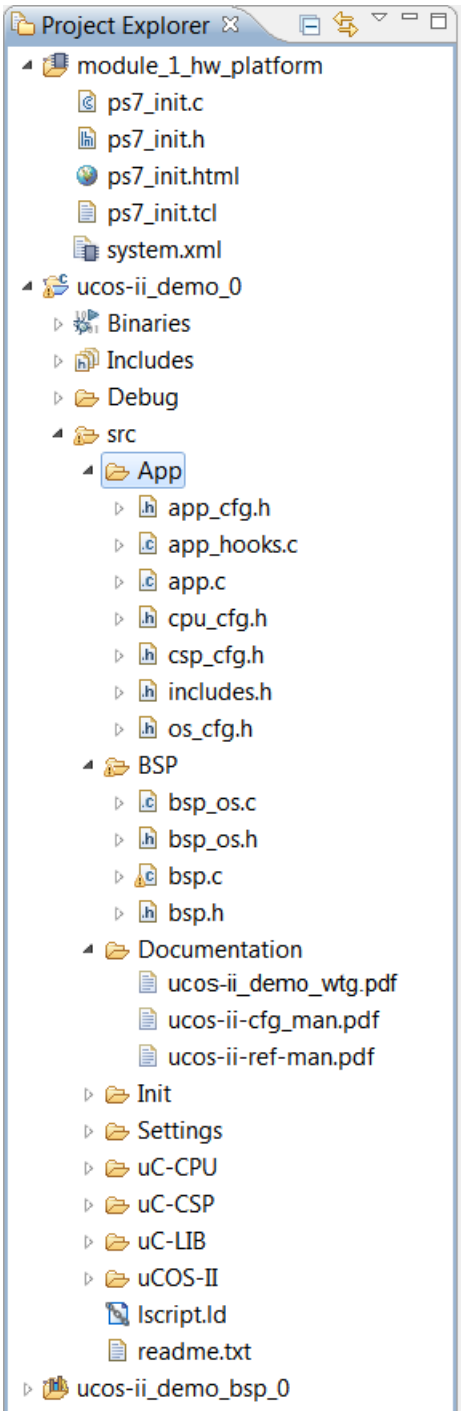

Figure 4-1 **Folders App, BSP and Documentation**

4.2 μC/CPU Folder

The **μC/CPU** module is Micriμm's way to abstract all the specifics of a CPU architecture. The module implements data type definitions and functions such as counting leading and trailing zeros.

The **uC-CPU** folder illustrated in Figure 4-2 contains all the files for the μC/CPU module including the port for the ARM Cortex-A9.

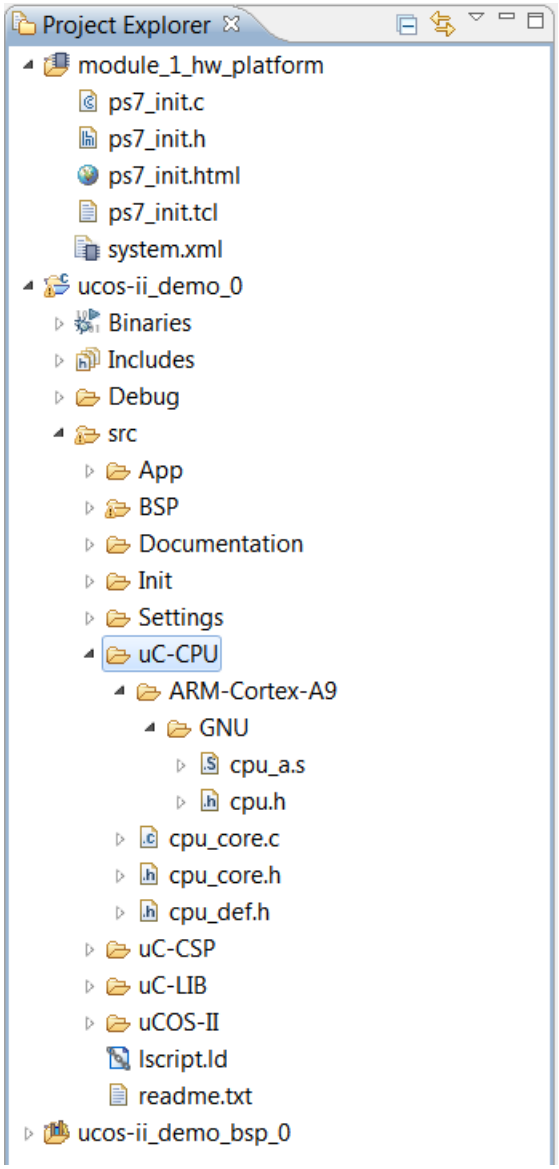

Figure 4-2 **μC/CPU Folder**

4.3 μC/CSP Folder

CSP stands for Chip Support Package. The **μC/CSP** module, similar to any Board Support Package, is used to abstract all the specifics of a certain chip. Because not all ARM Cortex-A9 are implemented equal, this module contains all those little tweaks such as register base addresses and interrupt controllers.

The **uC-CSP** folder illustrated in Figure 4-3 contains all the μC/CSP files including those specific for the chip onboard the ZC702 evaluation board.

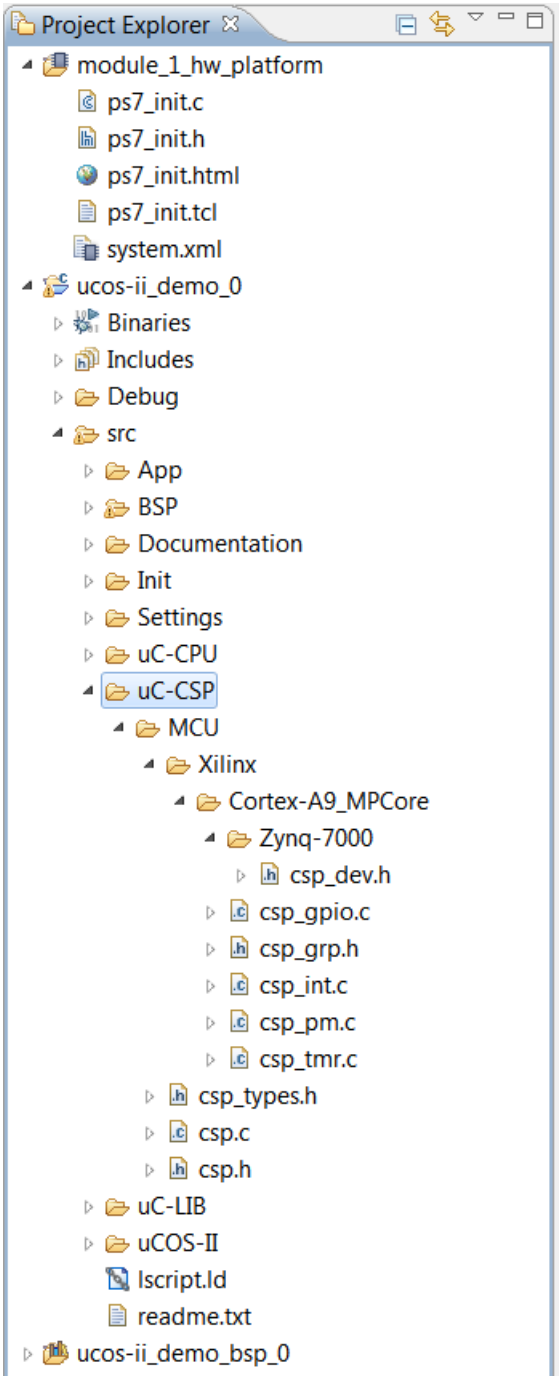

Figure 4-3 **μC/CSP Folder**

4.4 μC/LIB Folder

The **μC/LIB** module is Micriμm's way to provide a clean and organized ANSI C implementation of the most common standard library functions, macros and constants.

These files are independent of and used with any processor and compiler.

The μC/LIB module was created to obey MISRA C and other safety critical certifications.

The **uC-LIB** folder illustrated in Figure 4-4 contains all the μC/LIB module files.

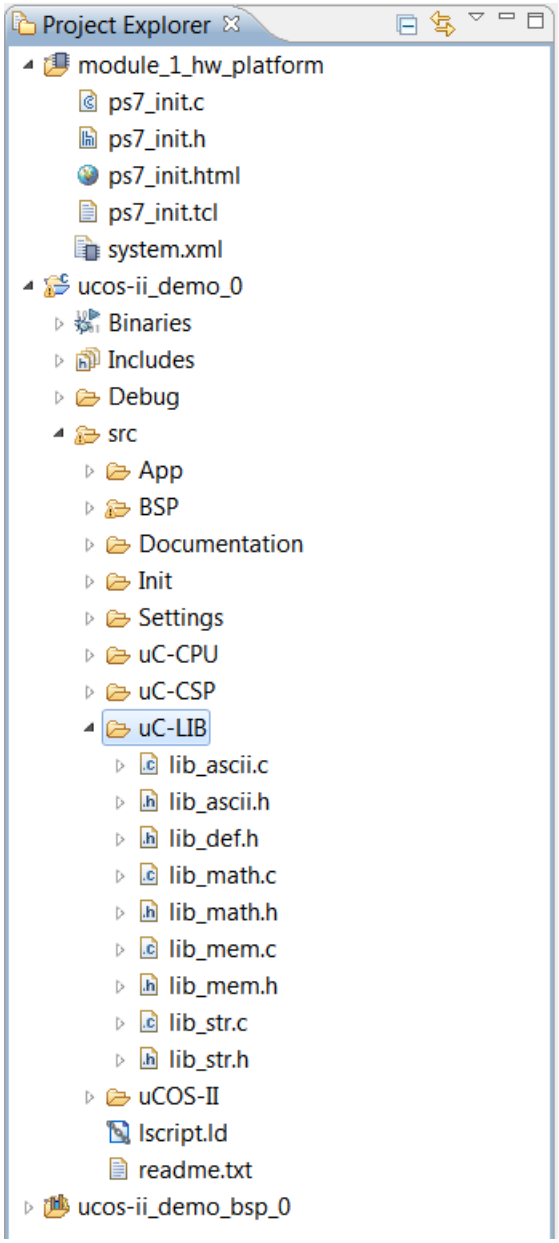

Figure 4-4 **μC/LIB Folder**

4.5 μC/OS-II Folder

The **μCOS-II** folder illustrated in Figure 4-5 contains all the source code of μC/OS-II including the port for the ARM Cortex-A9.

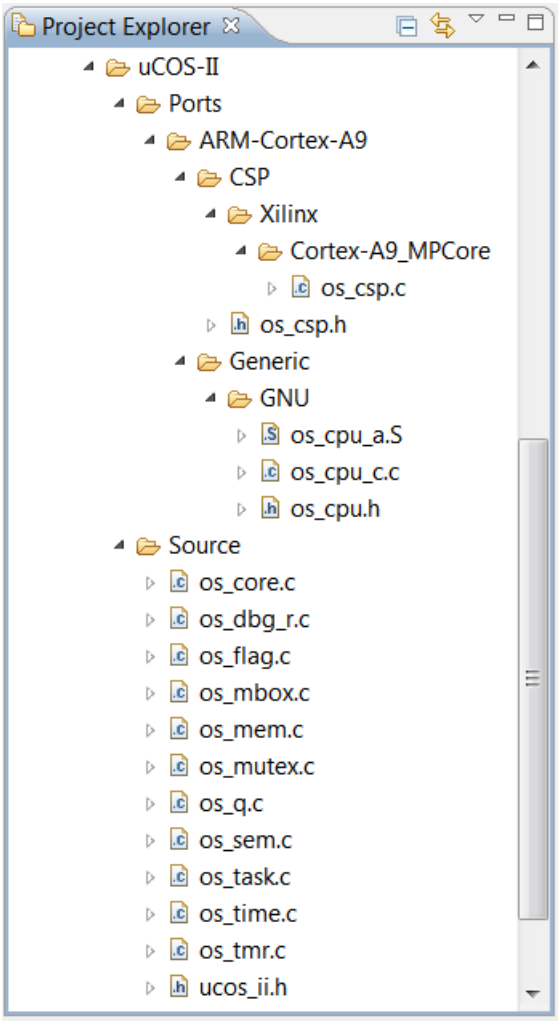

Figure 4-5 **μC/OS-II Folder**

4.6 Xilinx BSP for μC/OS-II Folder

The folder illustrated in Figure 4-6 contains all the Xilinx BSP files that are specific for the ZC702 evaluation board.

The BSP includes Xilinx drivers for the CAN bus, IIC, UART and USB among others.

One subfolder named ucosii_v2_00_a contains some startup code responsible to set the interrupt vectors to those exception handlers implemented by Micriμm.

Any Zynq-7000 application based on μC/OS-II requires this BSP and this custom interrupt vector table.

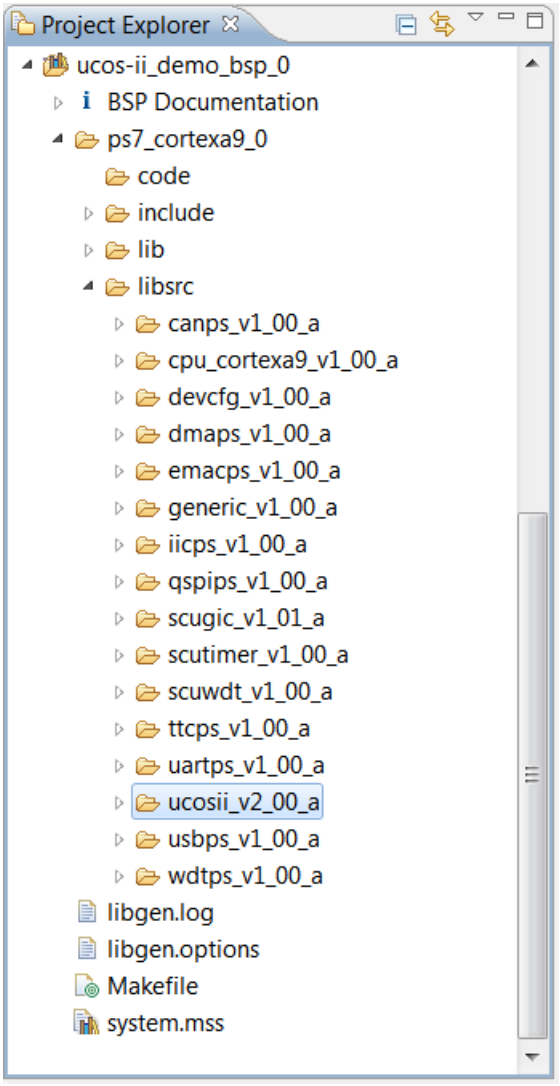

Figure 4-6 **Xilinx BSP for μC/OS-II folder**

5. How the μC/OS-II Demo Works

5.1 Main()

Code Listing 5-1 shows the main entry point for the μC/OS-II demo application.

```
int  main (void)
{
#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_ERR  cpu_err;
#endif


    BSP_Init();                     /* Initialize the onboard peripherals            */  (1)

    CPU_Init();                     /* Initialize the uC/CPU services                */  (2)

#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_NameSet((CPU_CHAR *)CSP_DEV_NAME,
                (CPU_ERR  *)&cpu_err);                                                  (3)
#endif

    CPU_IntDis();                   /* Disable Interrupts                            */  (4)

    AppPrintWelcomeMsg();                                                               (5)

    OSInit();                       /* Initialize uC/OS-II.                          */  (6)

    OSTaskCreateExt(AppTaskStart,  /* Create the start task                          */  (7)
                    (void       *)0,
                    (OS_STK     *)&AppTaskStartStk[APP_CFG_TASK_START_STK_SIZE - 1],
                    (OS_PRIO     )APP_CFG_TASK_START_PRIO,
                    (OS_PRIO     )APP_CFG_TASK_START_PRIO,
                    (OS_STK     *)&AppTaskStartStk[0],
                    (CPU_INT32U  )APP_CFG_TASK_START_STK_SIZE,
                    (void       *)0,
                    (CPU_INT16U  )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));

    OSStart();                      /* Start multitasking (give control to uC/OS-II). */  (8)

    return 0;
}
```

Listing 5-1 **app.c: main()**

L5-1(1) The entry point `main()` is defined in `app.c`. The `main()` starts by calling `BSP_Init()`. The code for `BSP_Init()` is found in `bsp.c`. `BSP_Init()` initializes all the necessary peripherals onboard the ZC702 including the UART. The reason a BSP function is used instead of simply calling the Xilinx drivers functions directly is that the application code can easily be ported to another processor.

L5-1(2) The function `CPU_Init()` is part of the μC/CPU module and it is defined in `cpu_core.c`. The function initializes the μC/CPU module by initializing the CPU timestamps, interrupt-disabled time measurements and the CPU name. This function must be called only once and before making any function calls to the μC/CPU module.

L5-1(3) The function `CPU_NameSet()` is also part of the μC/CPU module and it's used to set the name of the CPU by passing the ASCII string defined by `CSP_DEV_NAME`. `CSP_DEV_NAME` is part of the μC/CSP module (Chip Support Package).

L5-1(4) The function `CPU_IntDis()` is part of the μC/CPU module and it is used to disable interrupts.

L5-1(5) The function `APP_PrintWelcomeMsg()` is defined in `app.c` and it is called to display a welcome message on the serial terminal.

L5-1(6) The function `OSInit()` is defined in `os_core.c`. This function is used to initialize the internals of μC/OS-II and must be called prior to creating any μC/OS-II kernel object and, prior to calling `OSStart()`.

L5-1(7) The function `OSTaskCreateExt()` is defined in `os_task.c`. This function is the extended version of `OSTaskCreate()` they are both used to have μC/OS-II manage the execution of a task. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR. In this example, `OSTaskCreateExt()` creates the Startup Task which will be the subject of discussion in the next section.

L5-1(8) The function `OSStart()` is defined in `os_core.c` and it is used to start the multitasking process which lets μC/OS-II manage the task you have previously created. Before you can call `OSStart()`, you must have called `OSInit()` and you must have created at least one task.

5.2 AppTaskStart()

Code Listing 5-2 shows the body of the Startup Task.

```
static  void  AppTaskStart (void *p_arg)
{
    (void)p_arg;


    print("Task Start Created\r\n");                                          (1)

    OS_CSP_TickInit();                /* Initialize the Tick interrupt       */ (2)

    Mem_Init();                       /* Initialize memory management module */ (3)
    Math_Init();                      /* Initialize mathematical module      */ (4)

#if (OS_TASK_STAT_EN > 0u)
    OSStatInit();                     /* Determine CPU capacity              */ (5)
#endif

#ifdef CPU_CFG_INT_DIS_MEAS_EN
    CPU_IntDisMeasMaxCurReset();                                              (6)
#endif

    AppTaskCreate();                  /* Create Application tasks            */ (7)

    AppMutexCreate();                 /* Create Mutual Exclusion Semaphores  */ (8)

    while (DEF_ON) {                  /* Task body, written as an infinite loop */ (9)

        OSTimeDlyHMSM(0, 0, 0, 100);  /* Waits 100 milliseconds.             */ (10)

        AppPrint(".");                /* Prints a dot every 100 milliseconds.  */
    }
}
```

Listing 5-2 **app.c: AppTaskStart()**

L5-2(1) The Startup Task is implemented by `AppTaskStart()` in `app.c`. The function starts by sending a message through the UART to the serial terminal.

L5-2(2) The function `OS_CSP_TickInit()` is defined in `os_csp.c`. This is one of the files specific to the Xilinx Zynq-7000. The `OS_CSP_TickInit()` function is used to initialize the tick interrupt. The tick interrupt in the Cortex-A9 is handled by the processor's private timer.

L5-2(3) The function `Mem_Init()` is part of the μC/LIB Memory module. The function `Mem_Init()` is defined in `lib_mem.c` and it is used to initialize the memory management module by initializing the heap memory pool and its tables.

L5-2(4) The function `Math_Init()` is also part of the μC/LIB Math module. The function `Math_Init()` is defined in `lib_math.c` and it is used to initialize the math module by initializing the random number seed value.

L5-2(5) The function `OSStatInit()` is optional to include some statistics into your application. The function is defined in `os_core.c` and it is called by the application to establish CPU usage by first determining how high a 32-bit counter would count to 1 in 1 second if no other tasks were to execute during that time. CPU usage is then determined by a low priority task that keeps track of this 32-bit counter every second but this time, with other tasks running.

L5-2(6) The function `CPU_IntDisMeasMaxCurReset()` is also optional and as part of the `μC/CPU` module, is defined in `cpu_core.c`. The function is used to reset the current maximum interrupts-disabled time.

L5-2(7) The function `AppTaskCreate()` is defined in `app.c` and it is used to group all the `OSTaskCreate()` functions necessary to create all the tasks in your application.

L5-2(8) Similar to `AppTaskCreate()` this function is defined in `app.c` and it is used to group all the `OSMutexCreate` functions necessary to create all the mutual exclusion semaphores in your application.

L5-2(9) The task body is enclosed inside a while loop. The loop is always infinite because in μC/OS-II a task can never return.

L5-2(10) The function `OSTimeDly()` is defined in `os_time.c`. This function is called to delay execution of the currently running task until the specified number of system ticks expires. This, of course, directly equates to delaying the current task for some time to expire. No delay will result if the specified delay is 0. If the specified delay is greater than 0 then, a context switch will result. In this example, the idea is for this task to send a dot “.” through the UART every 100 milliseconds.

5.3 AppTaskCreate()

The function that groups all the calls to `OSTaskCreateExt()` to create the two tasks for this demo application is listed in Code Listing 5-3. The arguments for `OSTaskCreateExt()` such as `APP_CFG_TASK_1_STK_SIZE` are defined in `app_cfg.h`.

```
static  void  AppTaskCreate (void)
{
    OSTaskCreateExt(AppTask1,              /* Create the Task #1.                      */
                    (void       *)0,
                    (OS_STK     *)&AppTask1Stk[APP_CFG_TASK_1_STK_SIZE - 1],
                    (OS_PRIO     )APP_CFG_TASK_1_PRIO,
                    (OS_PRIO     )APP_CFG_TASK_1_PRIO,
                    (OS_STK     *)&AppTask1Stk[0],
                    (CPU_STK_SIZE)APP_CFG_TASK_1_STK_SIZE,
                    (void       *)0,
                    (CPU_INT16U  )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));

    OSTaskCreateExt(AppTask2,              /* Create the Task #2.                      */
                    (void       *)0,
                    (OS_STK     *)&AppTask2Stk[APP_CFG_TASK_2_STK_SIZE - 1],
                    (OS_PRIO     )APP_CFG_TASK_2_PRIO,
                    (OS_PRIO     )APP_CFG_TASK_2_PRIO,
                    (OS_STK     *)&AppTask2Stk[0],
                    (CPU_STK_SIZE)APP_CFG_TASK_2_STK_SIZE,
                    (void       *)0,
                    (CPU_INT16U  )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
}
```

Listing 5-3 **app.c: AppTaskCreate()**

5.4 AppMutexCreate()

Code Listing 5-4 shows the `AppMutexCreate()` function that groups all the mutual exclusion semaphores necessary for this application. In this case only one mutex is necessary to share the UART resource. The UART is used by the `AppPrint()` function which in turn is called by more than one task in the application. For that reason and for illustration purposes a mutex named `AppMutexPrint` will be created.

```
static  void  AppMutexCreate (void)
{
    CPU_INT08U  err;


    AppMutexPrint = OSMutexCreate(20, &err);  /* Creates the UART mutex.              */
}
```

Listing 5-4 **app.c: AppMutexCreate()**

5.5 AppTask1()

Code Listing 5-5 shows the body of task #1. The task #1 is responsible for printing “1” through the UART every second. The second task #2 will not be listed because it is identical to this one except that it prints “2” every 2-seconds.

```
static  void  AppTask1 (void *p_arg)
{
    (void)p_arg;

    AppPrint("Task #1 Started\r\n");

    while (DEF_ON) {                    /* Task body, written as an infinite loop.    */

        OSTimeDlyHMSM(0, 0, 1, 0);      /* Waits for 1-second.                        */

        AppPrint("1");                  /* Prints 1 to the UART.                      */

    }
```

Listing 5-5 **app.c: AppTask1()**

5.6 AppPrint()

Code Listing 5-6 shows an example of how to use a mutual exclusion semaphore in μC/OS-II to gain access to a shared resource. In this case the shared resource is the UART.

```
static  void  AppPrint (char *str)
{
    CPU_INT08U  err;


    OSMutexPend(AppMutexPrint, 0, &err); /* Wait for the shared resource to be released */

    print(str);                         /* Access the shared resource.                */

    OSMutexPost(AppMutexPrint);         /* Releases the shared resource.              */
}
```

Listing 5-6 **app.c: AppPrint()**

5.7 app_cfg.h

This μC/OS-II demo can serve as a template to create more sophisticated μC/OS-II based applications with the Zynq-7000. If you want to change the task's stack sizes and priorities you can do so from the `app_cfg.h` file. Code Listing 5-7 shows some of the definitions in this file.

```
/*
*********************************************************************************************
*                                       TASK PRIORITIES
*********************************************************************************************
*/

#define  APP_CFG_TASK_START_PRIO                           8u
#define  APP_CFG_TASK_1_PRIO                               2u
#define  APP_CFG_TASK_2_PRIO                               3u

#define  OS_TASK_TMR_PRIO                      (OS_LOWEST_PRIO - 2)


/*
*********************************************************************************************
*                                       TASK STACK SIZES
*                         Size of the task stacks (# of OS_STK entries)
*********************************************************************************************
*/

#define  APP_CFG_TASK_START_STK_SIZE                     512u
#define  APP_CFG_TASK_1_STK_SIZE                         512u
#define  APP_CFG_TASK_2_STK_SIZE                         512u
```

Listing 5-7 **app_cfg.h**

Revision History

The following table shows the revision history for this document:

Date	Version	Description	By
April 23rd. 2012	1.00	Initial release based on ISE Design Suite 14.1	JPB
October 22nd. 2012	1.01	Support for ISE Design Suite 14.2	JPB

Xilinx Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials, or to advise you of any corrections or update.

You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapp.

Micriμm Notice of Disclaimer

The μC/OS-II demo application for the Xilinx ZC702 Evaluation Board described in this document and the information in the document itself are provided solely as a reference to help engineers use μC/OS-II on the Xilinx ZC702 evaluation board.

Micriμm makes no warranty, representation or guarantee regarding the suitability of this μC/OS-II demo for any particular purpose, nor does Micriμm assume any liability arising out of the application or use of any example design, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.