Spartan-6 LX9 MicroBoard Embedded Tutorial

Tutorial 6

Software Debugging with SDK

Version 12.4.01

Revision History

Version	Description	Date
12.4.01	Initial release for EDK 12.4	3/18/2011

Table of Contents

Table of Figures

Overview

This is the sixth tutorial in a series of training material dedicated to introducing engineers to creating their first embedded designs. These tutorials will cover all the required steps for creating a complete MicroBlaze design in the Spartan-6 LX9 MicroBoard. While dedicated to this platform, the information learned here can be used with any Xilinx FPGA.

The tutorial is divided into two main steps: looking at the executable and using Platform Studio SDK to debug the code. The test application will reside in Block memory inside the FPGA. We will be using the design from the previous tutorials.

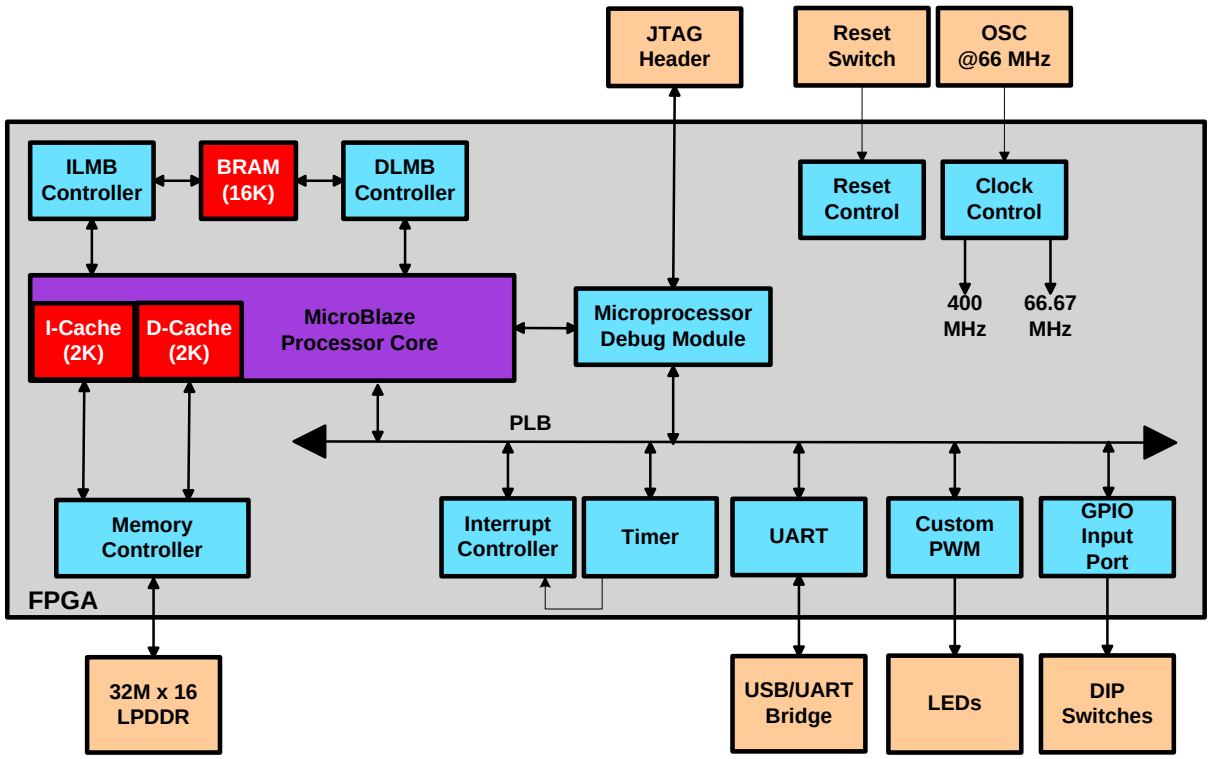

Figure 1 - Hardware Platform

Objectives

This tutorial demonstrates how to use Platform Studio SDK to debug a C application. The tutorial will show:

- How to look at the code sections and disassembly
- How to start the debugger
- How to look at registers and memory
- How to set breakpoints, watchpoints and step through the code

Requirements

The following items are required for proper completion of this tutorial.

- Completion of the Adding Custom IP to an Embedded System Tutorial (Tutorial #3)

Software

The following software setup was used to test this reference design:

- WindowsXP 32-bit Service Pack 2
- Xilinx ISE WebPack with the SDK add-on or ISE Embedded Edition version 12.4
- Installed Digilent Adept and Xilinx 3rd-party USB Cable driver
- Installed Silicon Labs CP210x USB-to-UART Bridge Driver
- Installation of the Spartan-6 LX9 MicroBoard XBD files

Hardware

The hardware setup used by this reference design includes:

- Computer with a minimum of 300-900 MB (depending on O/S) to complete an XC6SLX9 design[1]
- Avnet Spartan-6 LX9 MicroBoard Kit
  - Avnet Spartan-6 LX9 MicroBoard
  - USB Extension cable (if necessary)
  - USB A-to-MicroB cable

Setup

- Install ISE Embedded Edition or ISE WebPack with the EDK add-on.
- Install Digilent Adept and Xilinx 3rd-party USB Cable driver (see Installation Guide on the DRC)

Recommended Reading

Available from Avnet: *http://em.avnet.com/s6microboard*

- The hardware used on the Spartan-6 LX9 MicroBoard is described in detail in Avnet document, *Spartan-6 LX9 MicroBoard User Guide*.
- An overview of the configuration options available on the Spartan-6 LX9 MicroBoard, as well as Digilent driver installation instructions can be found in the Avnet document, *Spartan-6 LX9 MicroBoard Configuration Guide.*
- Instructions on installing the Silicon Labs CP210x USB-to-UART drivers can be found in the Avnet document, *Silicon Labs CP210x USB-to-UART Setup Guide*.

Available from Xilinx: *http://www.xilinx.com/support/documentation/spartan-6.htm*

- Details on the Spartan-6 FPGA family are included in the following Xilinx documents:
  - *Spartan-6 Family Overview (DS160)*
  - *Spartan-6 FPGA Data Sheet (DS162)*
  - *Spartan-6 FPGA Configuration User Guide (UG380)*
  - *Platform Studio Help (available in tool menu)*
  - *Platform Studio SDK Help (available in tool menu)*
  - *MicroBlaze Reference Guide v.12.4 (UG081)*
  - *Embedded System Tools Reference Manual v.12.4 (UG111)*

[1] Refer to www.xilinx.com/ise/products/memory.htm

I. Looking at the Executable

When debugging programs it is often useful to look at the code generated by the compiler and check where the code is located in memory.

1) Start Xilinx SDK and select the Workspace from Tutorial_01. (Start from Tutorial #3, a zip file of the Tutorial #3 solution is available on the Avnet DRC.)

2) Expand the **Tutorial_Test** project.

3) Expand the **Debug** folder. The Debug folder contains the compiler output for the Debug build configuration. To view the build configuration settings right-click on the **Tutorial_Test** project and select **C/C++ Build Settings**. The optimization level is set to none and the debug level is set to maximum (MicroBlaze gcc compiler Optimization and Debugging settings).

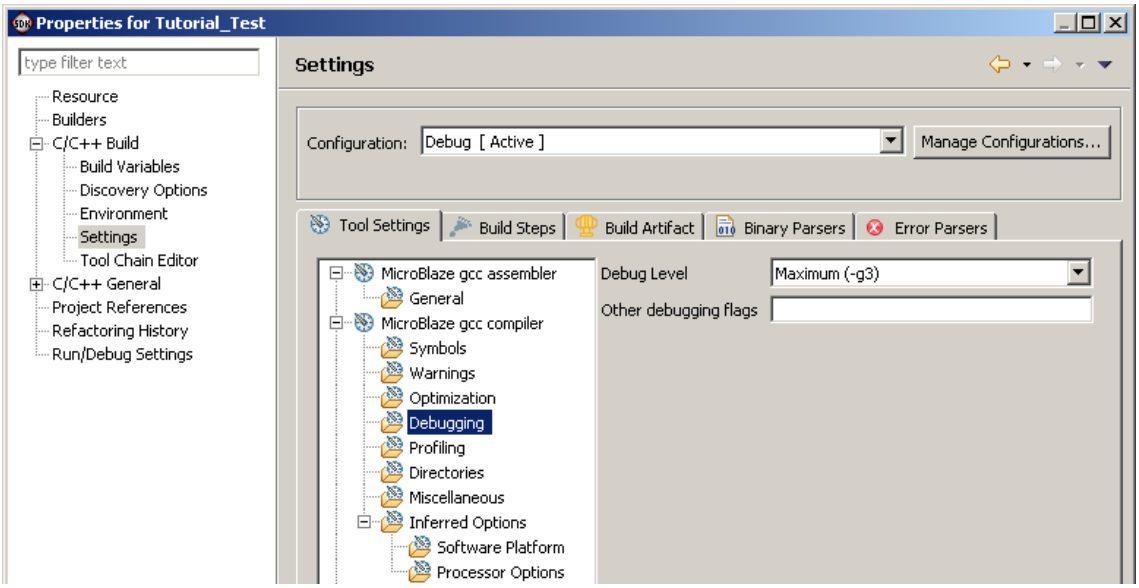

Figure 2 - C/C++ Build Settings

4) The executable can be opened in the editor. Double-click on **Tutorial_Test.elf**.

5) We will first look at the program headers. The beginning of the file shows information on the sections and symbols used in the code. It is useful to verify the stack size, the memory map, or the location of specific functions.

6) Scroll down to view the code disassembly

7) The first instruction at the reset vector is a branch immediate to address 0x50, which is _start1. All the instructions are documented in the MicroBlaze Reference Guide.

```
0:      b8080050      brai    80      // 50 <_start1>
```

8) Scroll down to <main>. The C code is interleaved with the disassembly making it easier to view the instructions associated with the code.

9) Close the file. (Leave SDK open.)

II. Using the Debugger

Platform Studio SDK includes the GNU debugger (GDB). It uses the Xilinx Microprocessor Debugger (XMD) as the underlying engine to communicate with the processor target through JTAG. We will add hardware Watchpoints to MicroBlaze for debugging. While not necessary, watchpoints provide another debugging tool. Watchpoints will stop execution whenever a read or write of an expression is detected.

1) Start Project Navigator and open the **Tutorial_01** project.

2) Double-click on the **mb_system** module to open the system in XPS.

3) In the **System Assembly View** window in XPS double-click on the **microblaze_0** instance.

4) Click on the **Advanced** button (under the Select Configuration Window).

5) Select the **Debug** tab and add one write address Watchpoint and one read address Watchpoint. Click **OK**.

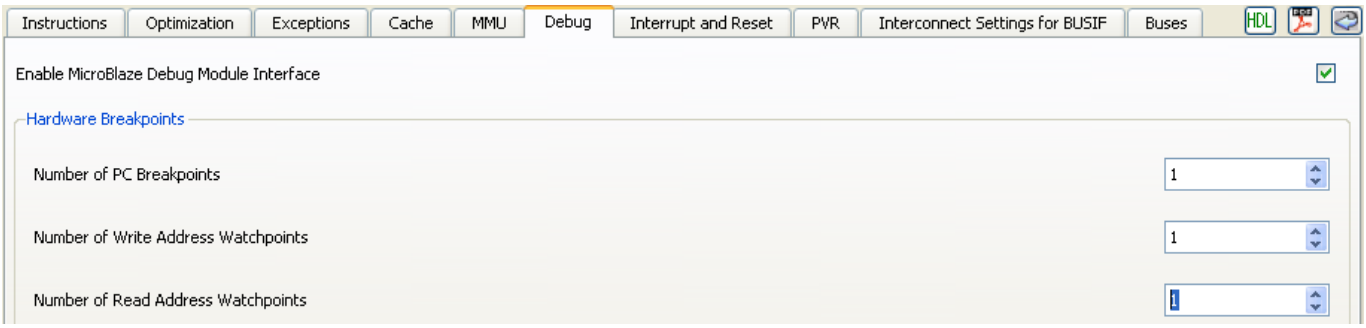

Figure 3 - Debug Settings

6) We need to update the design information for SDK. Go to **Project > Export Hardware Design to SDK…** Select **Export Only**.

7) Close XPS.

8) SDK should detect that the hardware system has changed. Click **Yes** to update the hardware platform and BSP.

9) In Project Navigator, Select the top level module in the **Hierarchy** window.

10) Double-click on **Generate Programming File** in the **Processes** window to update the design.

11) Plug the MicroBoard board into the PC or connect a JTAG cable to J6. It is recommended to use a JTAG cable for faster debugging response.

12) In SDK, You will need to change the JTAG settings when using a JTAG cable.

13) Go to **Xilinx Tools > Configure JTAG Settings**. Change the JTAG Cable type to the correct JTAG cable type. Plug the USB UART cable between the MicroBoard and the PC.

14) Click on the Program FPGA icon 

    a. For the Bitstream, browse to the **Tutorial_01** directory and select **mb_system.bit**

    b. For the BMM File, browse to the **Tutorial_01** directory and select **edkBmmFile_bd.bmm**

15) Click on **Program**

16) We will now start a new debug session. Make sure the **Tutorial_Test** project is selected in the Project Explorer window. Right-click and select **Debug As > Debug Configurations…**

17) Click on the **New launch configuration** icon to add a new configuration for **Tutorial_Test**.

18) Click on the **STDIO Connection** tab and verify that the correct COM port is being used for the UART.

19) Click on **Debug** to start the session. Platform Studio SDK will switch to a Debug perspective with an arrow pointing to the program current location.

20) The Debug perspective provides all the necessary tools and information to debug a software application. We will first go through the different windows. The editor window shows the file being debugged. Placing the cursor over variables would show the current value for the variable.

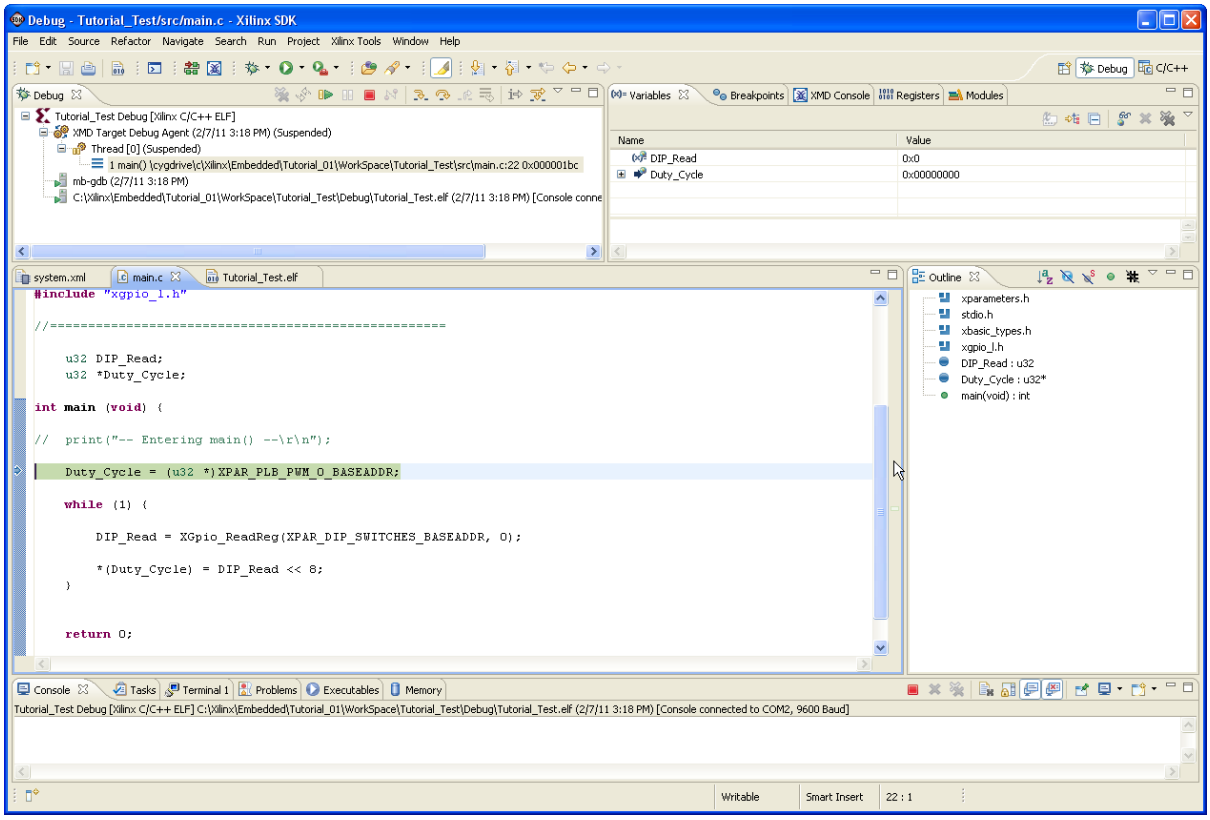

Figure 4 - Debug Perspective

21) The **Debug** window provides information on the status of the session. The application is currently suspended, waiting for input. The green arrow would resume the application while the red square would terminate the session. Multiple stepping options are available, including stepping in assembly mode.

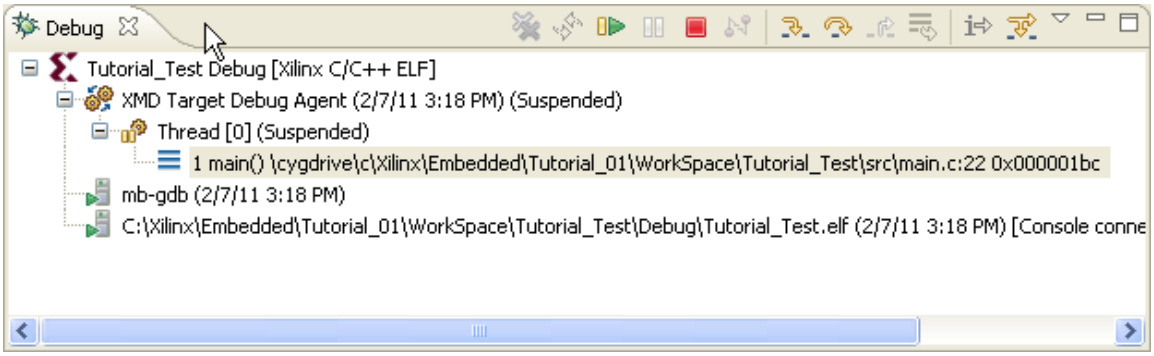

Figure 5 - Debug Window

22) Click on the **Registers** tab in the top right window to view the MicroBlaze registers. Any changes from the last context would show in red. The **rpc** register is the program counter. Notice what address of the program counter. It should read 0x1BC, which is right before the while loop in main.c.

23) To view the disassembly of the source, go to **Window > Show View > Disassembly**.

24) We will set a write watchpoint for **DIP_Read** to check when it is being written to. In the **Outline** window on the right side, right-click on **DIP_Read** and select **Toggle Watchpoint**. Select **Write Access** then click **OK**.

25) Click on the **Variables** tab in the top right window. Right-click in the window and select **Add Global Variables...** Select **DIP_Read** and **Duty_Cycle** from the list then click **OK**.

26) Carefully change the DIP switches setting to a non-zero (default) value.

27) Click on the **Resume** icon on the toolbar. The debugger will stop after the write access to **DIP_Read**. The value shown in the **Variables** window will show the address for Duty_Cycle (0xc9c00000) as well as the **DIP_Read** value.

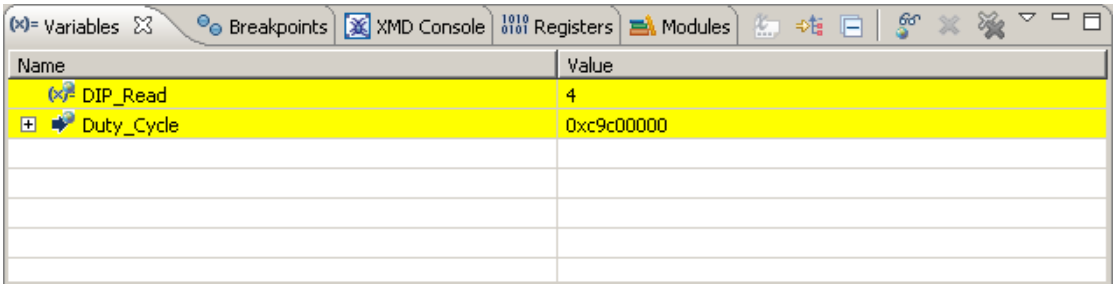

Figure 6 - Variables Window

28) Right-click on **DIP_Read** in the Outline window and select **Toggle Watchpoint** to remove the Watchpoint.

29) Double-click in the column on the left of the DIP_Read = XGpio_ReadReg line to place a breakpoint.

30) Click on the **Resume** icon on the toolbar to go to the breakpoint. The line will be highlighted in green.

31) We will modify the Duty_cycle value. **Step Over** the current line.

32) The value of **Duty_Cycle** is the **DIP_Read** value shifted left by 8. The **Variables** view will show the value of Duty_Cycle. We can modify that value. Expand the Duty_Cycle pointer and click on the value column. Change the value and press enter. 0x0 would turn off the LEDs. 0xFFF would have the highest intensity.

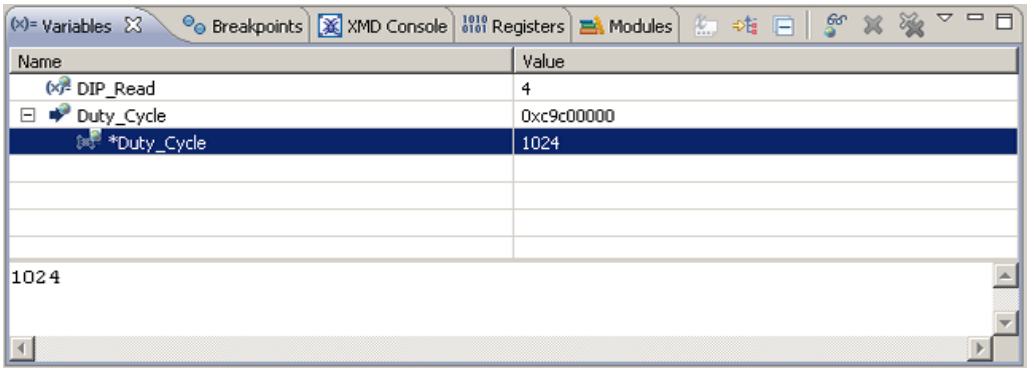

Figure 7- Changing LED Brightness in Variables View

33) The same variables can be seen in the **Memory** View. **Window → Show View → Memory**

34) Click on the **Memory** View tab by the console window.

35) Click in the **Add** icon and type **&DIP_Read**. Click **OK**. The Memory View will show the location and value of the variable in memory.

36) Modify the DIP Switches on the board.

37) Click on the **Resume** icon to get the existing breakpoint.

38) Step Over the line and observe the **Memory** View. A small triangle will indicate that the memory value was modified.

39) Memory can be modified in the Memory View. Change the value of **DIP_Read** in the Memory View and press enter. Step Over the line.

40) Click on the **Terminate** icon to finish the session.

41) Changes can be made to the code in the editor window. Clicking on the **Debug** icon in the toolbar would bring back the same debug session with all the previous breakpoints and watchpoints.

42) Click on the C/C++ perspective C/C++ on the top right corner to go back to the previous perspective.

43) The peripheral_tests_0 application could also be debugged. The testperiph.c file would need to be fixed to compile the application.

44) To view the line numbers in the editor window, go to **Window > Preferences**. Expand **General > Editors** and select **Text Editors**. Check the **Show Line Numbers** box.

45) Open **testperiph.c** and comment line **111**. Save the file.

46) Right-click on **peripheral_tests_0** and select **Debug As > Launch on Hardware**.

47) Place a breakpoint on line **127** for the Timer test example.

48) Click on the **Resume** button.

49) Click on the **Step Into** icon to enter the function. The **Debug** View will show the stack trace and all the views are updated to the current context.

50) Continue stepping and observe the variables. The Xilinx device drivers can also be stepped into to follow the execution of the code.

51) The register R1 provides the stack pointer address.

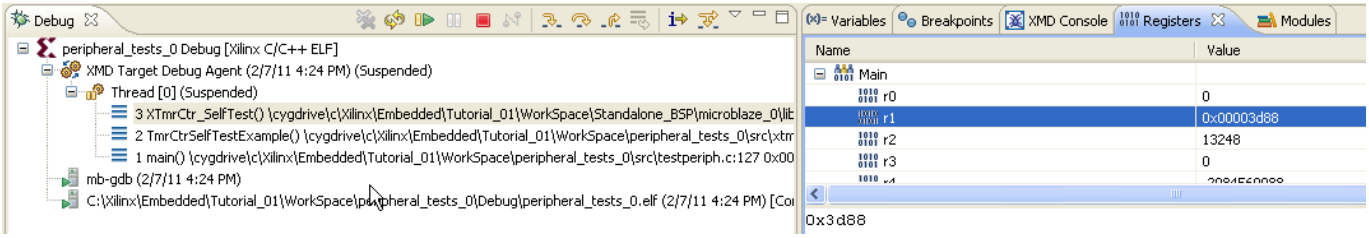

Figure 8 - Registers Window

52) Click on the **Terminate** icon when finished.

53) Close SDK and Project Navigator.

That concludes this tutorial. We have now learned how to debug a C application using SDK.

Getting Help and Support

Evaluation Kit home page with Documentation and Reference Designs

http://em.avnet.com/s6microboard

Avnet Spartan-6 LX9 MicroBoard forum:

http://community.em.avnet.com/t5/Spartan-6-LX9-MicroBoard/bd-p/Spartan-6LX9MicroBoard

For Xilinx technical support, you may contact your local Avnet/Silica FAE or Xilinx Online Technical Support at www.support.xilinx.com. On this site you will also find the following resources for assistance:

- Software, IP, and Documentation Updates
- Access to Technical Support Web Tools
- Searchable Answer Database with Over 4,000 Solutions
- User Forums
- Training - Select instructor-led classes and recorded e-learning options

Contact Avnet Support for any questions regarding the Spartan-6 LX9 MicroBoard reference designs, kit hardware, or if you are interested in designing any of the kit devices into your next design.

- http://www.em.avnet.com/techsupport

You can also contact your local Avnet/Silica FAE.