Implementing Linux on the Zynq™-7000 SoC

Lab 4.2

Using SDK for Linux Application Debug

September 2012
Version 05

Table of Contents

Lab 4.2 Overview

This lab is a short demonstration of the Linux environment on the Zynq™ EPP, illustrating how to debug a Linux software application.

In this lab, you will use the Xilinx Software Development Kit (SDK) to debug a simple Linux software application project. The target ZedBoard will automatically boot Linux from an SD card with the Linux kernel as was configured in the previous lab exercises.

Lab 4.2 Objectives

When you have completed Lab 4.2, you will know how to do the following:

- Set up an SDK software application project for debugging
- Use the many features of the SDK debugger

Experiment 1: Basic Application Debug

This lab comprises three primary steps: You will create an SDK software workspace, add the “LED Dimmer” software application to the workspace, and debug the software application on ZedBoard.

SDK Workspace

The first step in this lab is to create an SDK software workspace.

The working directory of the SDK workspace for this lab is the same as the previous Lab 4.1 **C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\** folder.

You will create the SDK workspace and software platform for the hardware in this directory.

Note that once SDK is launched and a workspace is set up in this directory, you will not be able to move these directories.

Adding a Software Application

By adding a new software project, SDK will automatically build and produce an executable and load format (ELF) file.

Debug a Software Application

The same Linux application created in SDK can be loaded and debugged on the ZedBoard target hardware.

Experiment 1 General Instruction:

Launch Xilinx Software Development Kit (SDK) and continue using the SDK workspace in the Lab 4.1 directory. The activities of this lab build upon the work done in Lab 4.1 exercises. If you have not already completed Lab 4.1 activities, do so now or use the Lab 4.1 solution folder as the starting point for this lab.

Use SDK to create a new debug configuration debug the application code remotely on the target ZedBoard hardware.

Experiment 1 Step-by-Step Instructions:

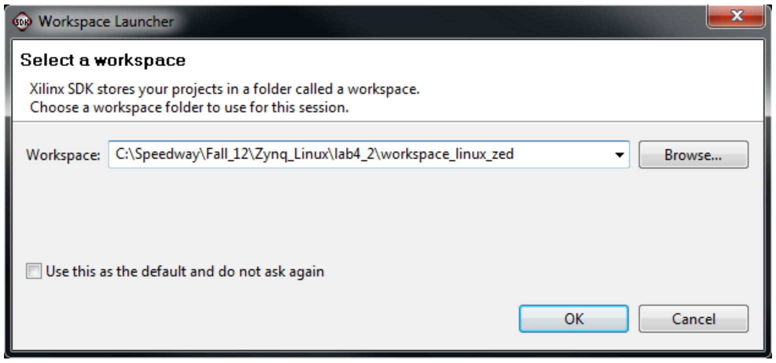
1. Launch Xilinx Software Development Kit (SDK) if not already open. **Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.2 → EDK → Xilinx Software Development Kit**.

Figure 1 – The SDK Application Icon

SDK creates a workspace environment consisting of project files, tool settings, and your software application. Once set, you cannot change the location of this workspace. If it is necessary to move a software application to another location or computer, use the Import and Export facilities built into SDK. A good location for the software workspace is the root directory of your PlanAhead™ or ISE® tool project. In this (and other) software only lab, neither of those other Xilinx tools are used, so workspace location is always at the discretion of the programmer.

2. Set or switch the workspace to the following folder and then click the **OK** button:

C:\Speedway\Fall_12\Zynq_Linux\lab4_1\workspace_linux_zed\



Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Speedway\Fall_12\Zynq_Linux\lab4_2\workspace_linux_zed

Figure 2 – Switching to the Appropriate SDK Workspace

3. If the ZedBoard has **NOT** been powered down or reset since the previous lab, skip ahead to step 13.

   Connect 12 V power supply to ZedBoard barrel jack (J20).

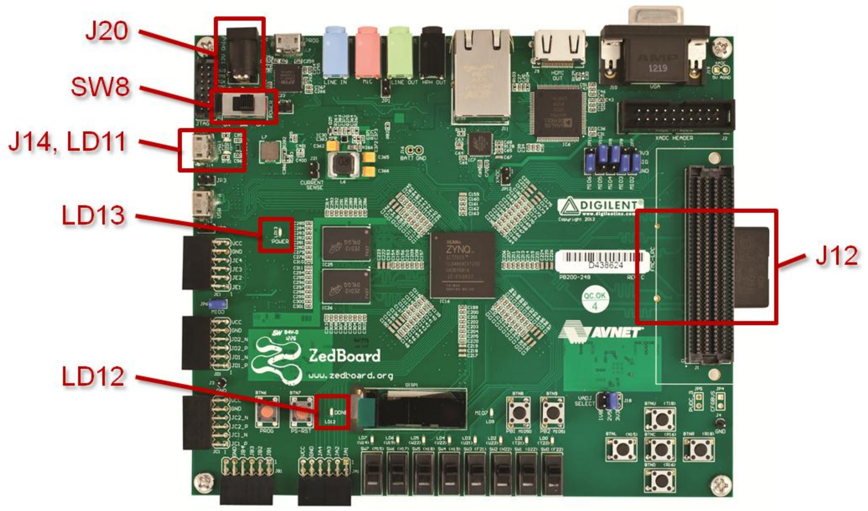


Figure 3 – ZedBoard Hardware Reference

4. Connect the USB-UART port of ZedBoard (J14) which is labeled UART to a PC using the MicroUSB cable.

5. Insert the 4GB SD card included with ZedBoard into the SD card slot (J12) located on the underside of ZedBoard PCB.

6. Connect an Ethernet patch cable between the development PC network adapter and ZedBoard Ethernet jack J11.

   Note that the default static IP address of ZedBoard is 192.168.1.10 so the IP address of the network adapter on the development PC should be set to another non-conflicting IP address in the range 192.168.1.1 to 192.168.1.254.

7. Verify the ZedBoard boot mode (JP7-JP11) and MIO0 (JP6) jumpers are set to SD card mode as described in the Hardware Users Guide:

   http://www.zedboard.org/sites/default/files/ZedBoard_HW_UG_v1_6.pdf

   A copy of the Hardware Users Guide is also located in the SpeedWay **C:\Speedway\Fall_12\Zynq_Linux\support_documents\** folder for your convenience.

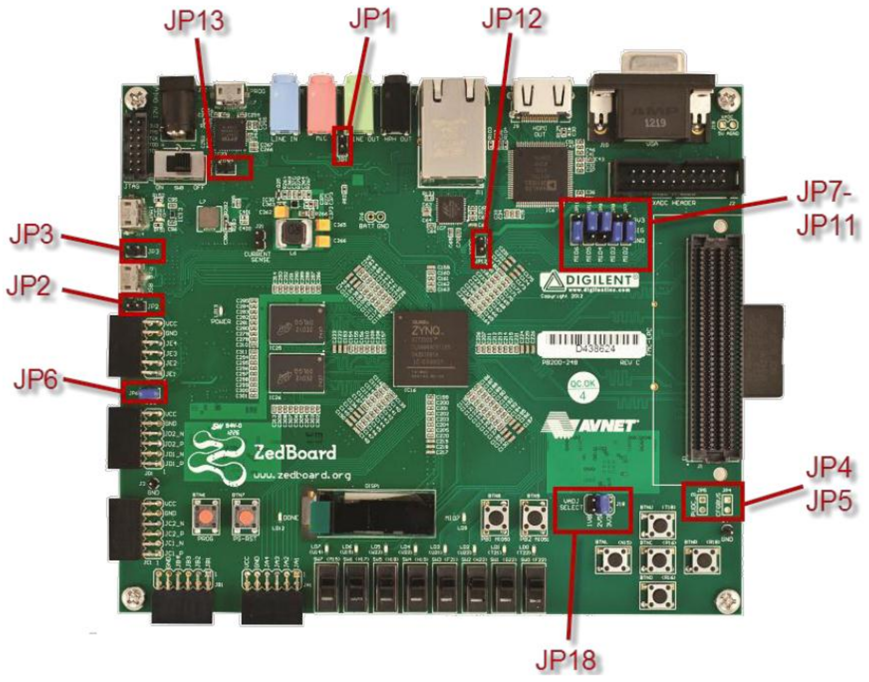


Figure 4 – ZedBoard Jumper Settings

8. Turn power switch (SW8) to the ON position. ZedBoard will power on and the Green Power Good LED (LD13) should illuminate.

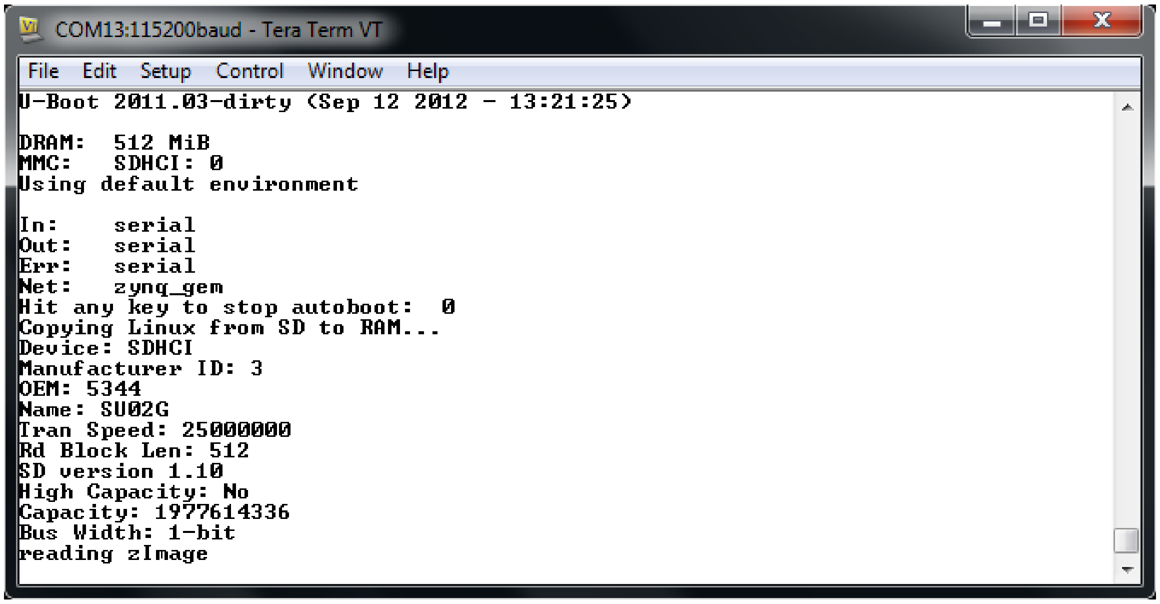
9. Wait approximately 15 seconds. The blue Done LED (LD12) should illuminate.

10. On the PC, if a serial terminal session is not already open, open a serial terminal program. Tera Term was used to show the example output for this lab document.

Figure 5 – Tera Term Icon

11. If the amber USB-Link Status (LD11) does not flicker during boot to indicate activity, check the driver installation to determine if the device driver is recognized and enumerated successfully and that there are no errors reported by Windows.

12. Power cycle the ZedBoard and monitor the Tera Term window. When the terminal output from U-Boot and a countdown is observed, allow the countdown to expire.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
U-Boot 2011.03-dirty (Sep 12 2012 - 13:21:25)

DRAM:  512 MiB
MMC:   SDHCI: 0
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   zynq_gem
Hit any key to stop autoboot:  0
Copying Linux from SD to RAM...
Device: SDHCI
Manufacturer ID: 3
OEM: 5344
Name: SU02G
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.10
High Capacity: No
Capacity: 1977614336
Bus Width: 1-bit
reading zImage
```

Figure 6 – ZedBoard U-Boot Booting Linux

13. In the **Project Explorer** tab, right-click the **linux_led_dimmer** project and select the **C/C++ Build Settings** menu item. .

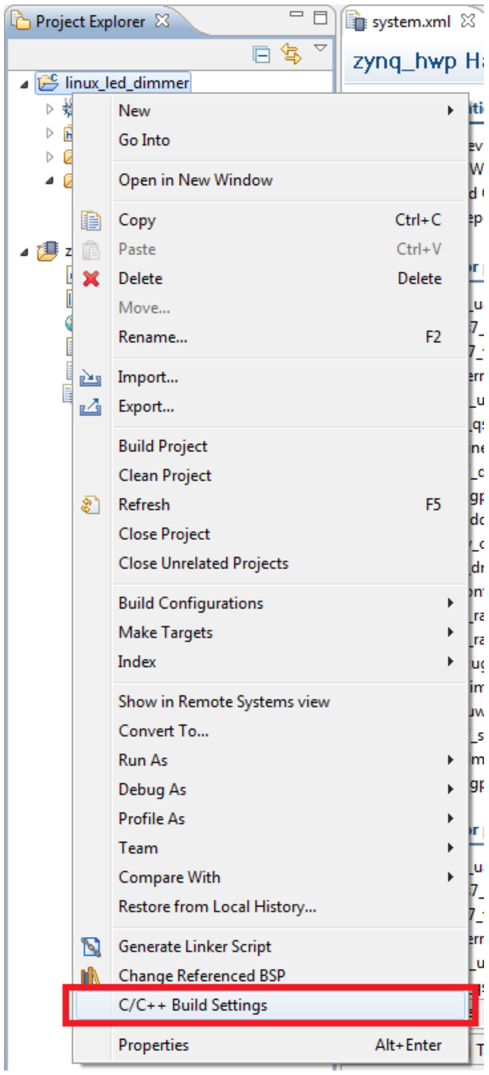
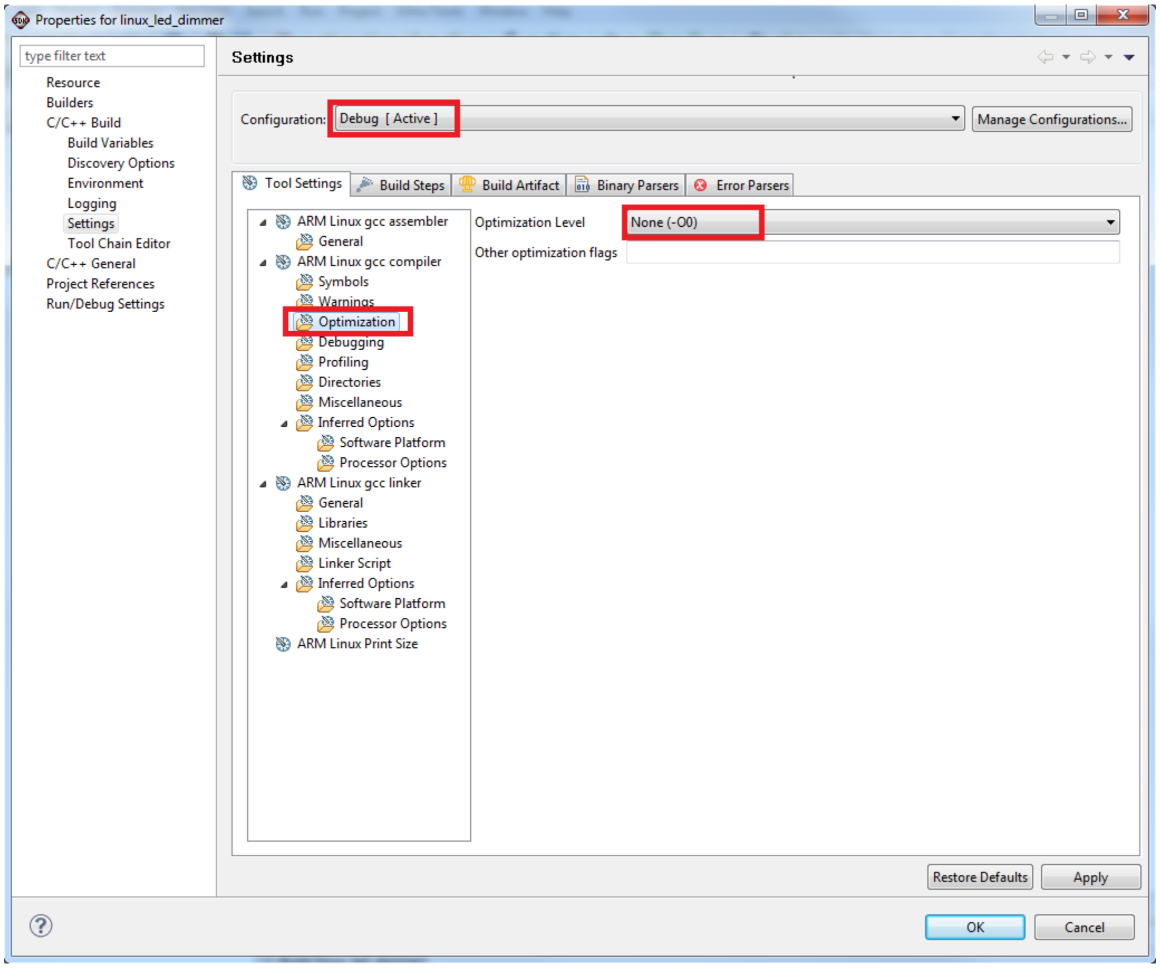


Figure 7 – Setting the Build Properties for the linux_led_dimmer Application

14. Set the **Configuration** setting to the **Debug** option.

    Click on the **Optimization** item in the **Tool Settings** list and set the **Optimization Level** to the **None (-O0)** setting.

    Note that turning optimization off for debugging maintains the 1:1 relationship between the source code and the executing code. When optimization is turned on, the compiler will re-organize the executable code and coherency between the two will be lost.



Configuration: Debug [Active]

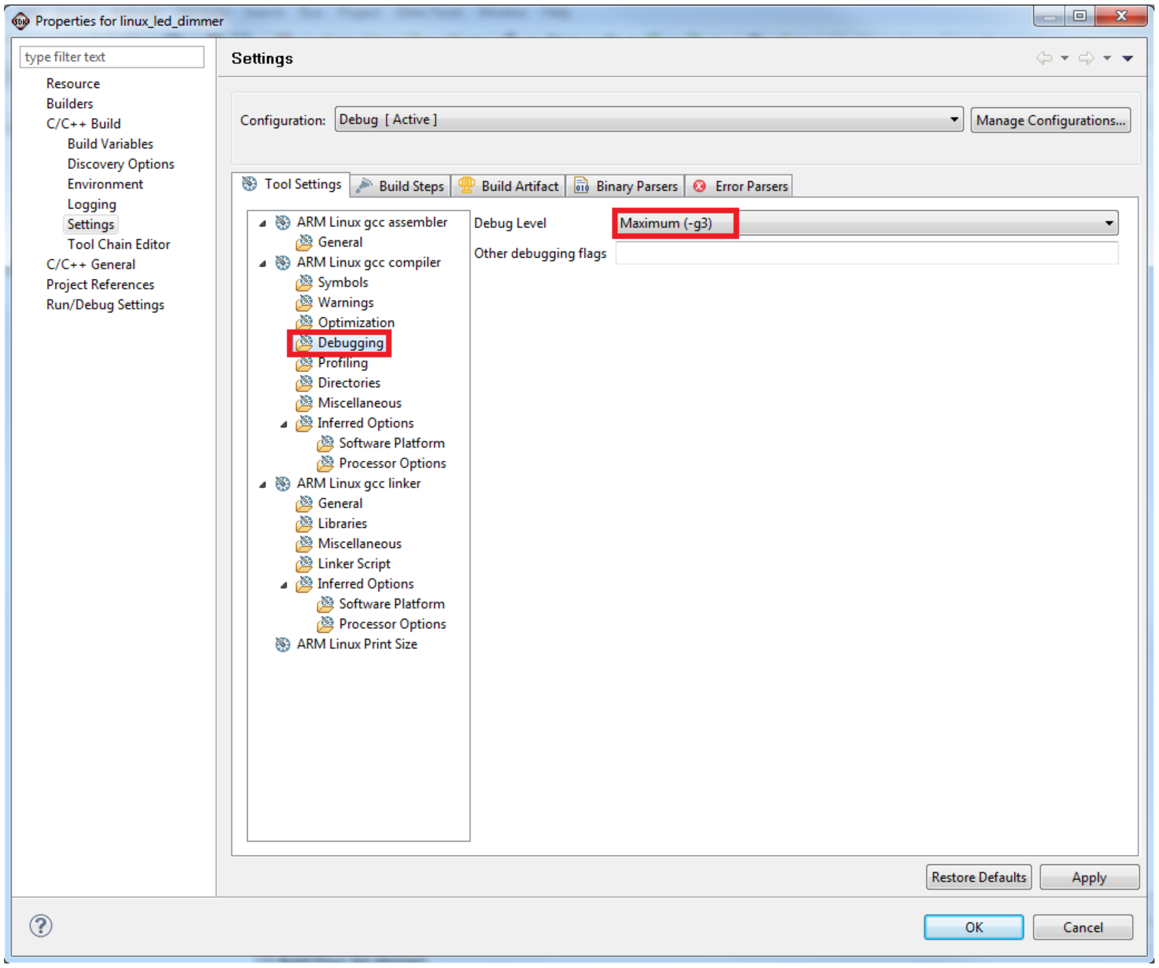
Optimization Level: None (-O0)

Other optimization flags

Figure 8 – Setting the Build Properties for the linux_led_dimmer Application

Click on the **Debugging** item below the Optimization item and set the **Debug Level** to the **Maximum (-g3)** setting. Click the **Apply** button and then the **OK** button to force the new settings into effect.

The application will be rebuilt automatically. A successful build is indicated when the program size is returned in the SDK console panel.



Configuration: Debug [Active]

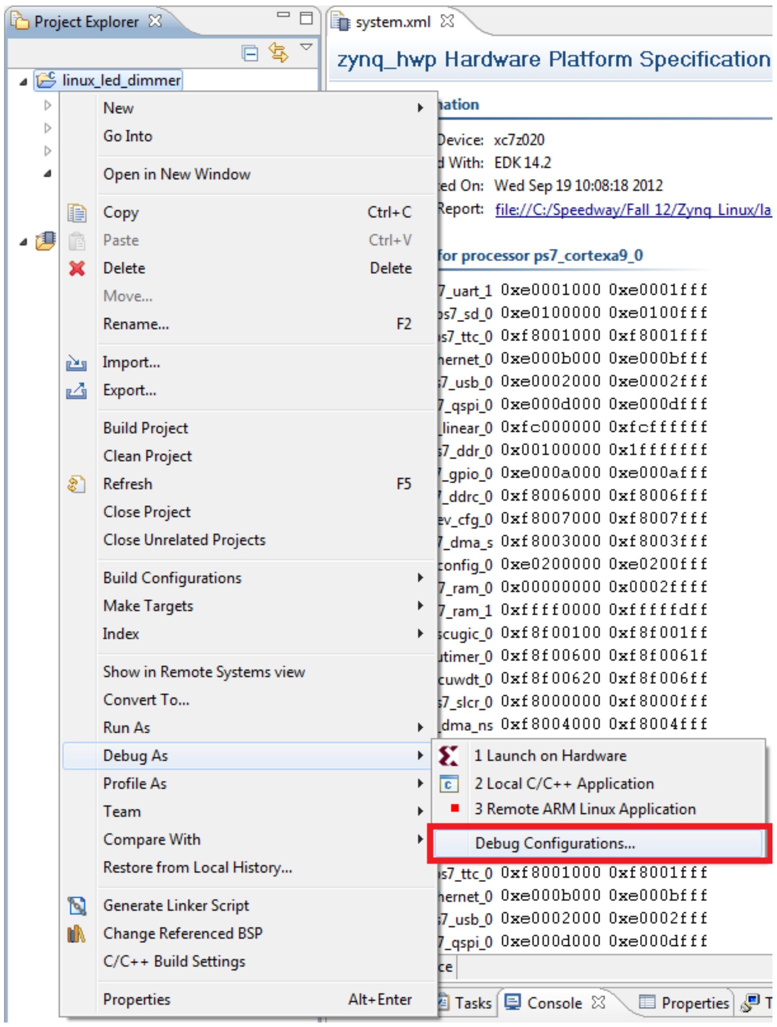
Debug Level: Maximum (-g3)

Other debugging flags:

Figure 9 – Setting the Build Properties for the linux_led_dimmer Application

15. We will reuse the Run configuration from Lab 4.1 as our application Debug configuration. Debug configurations simply associate an ELF object file to a target for execution. In this case, the target is a hardware board accessed over a network TCP/IP connection.

    In the **Project Explorer** tab, right-click the **linux_led_dimmer** application project and select the **Debug As→Debug Configurations...** menu item.

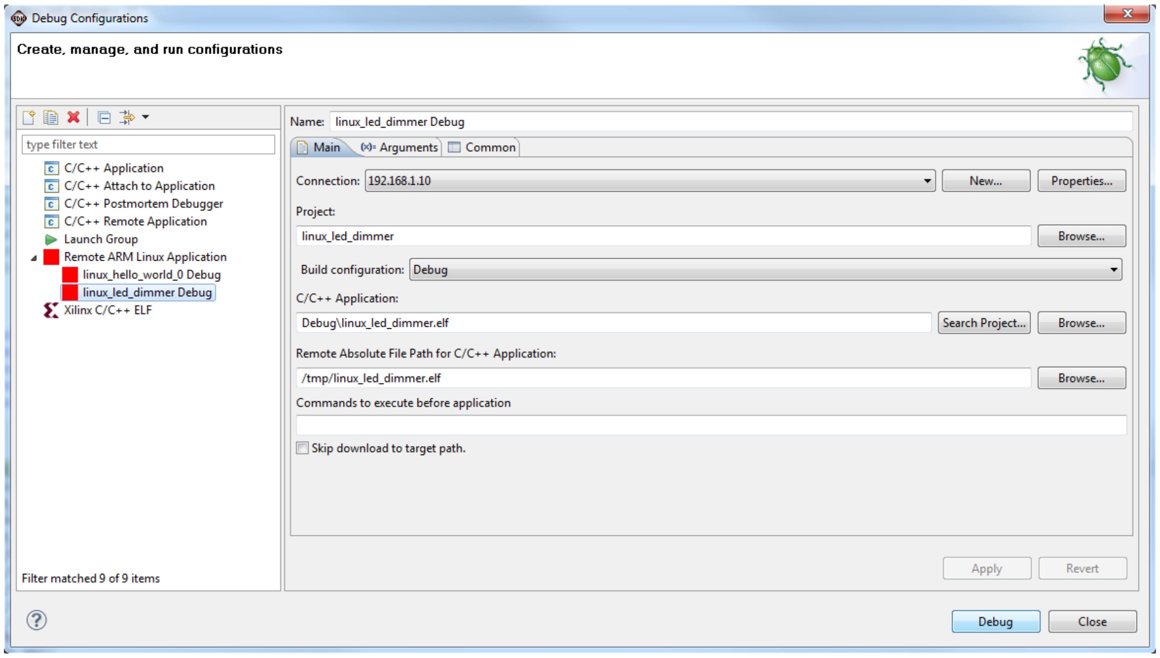


```
ation
Device:  xc7z020
d With:  EDK 14.2
ed On:   Wed Sep 19 10:08:18 2012
Report:  file://C:/Speedway/Fall 12/Zynq Linux/la
for processor ps7_cortexa9_0
7_uart_1 0xe0001000 0xe0001fff
ps7_sd_0 0xe0100000 0xe0100fff
s7_ttc_0 0xf8001000 0xf8001fff
hernet_0 0xe000b000 0xe000bfff
7_usb_0  0xe0002000 0xe0002fff
7_qspi_0 0xe000d000 0xe000dfff
linear_0 0xfc000000 0xfcffffff
7_ddr_0  0x00100000 0x1fffffff
_gpio_0  0xe000a000 0xe000afff
_ddrc_0  0xf8006000 0xf8006fff
ev_cfg_0 0xf8007000 0xf8007fff
7_dma_s  0xf8003000 0xf8003fff
config_0 0xe0200000 0xe0200fff
7_ram_0  0x00000000 0x0002ffff
7_ram_1  0xffff0000 0xfffffdff
scugic_0 0xf8f00100 0xf8f001ff
itimer_0 0xf8f00600 0xf8f0061f
cuwdt_0  0xf8f00620 0xf8f006ff
7_slcr_0 0xf8000000 0xf8000fff
dma_ns   0xf8004000 0xf8004fff
s7_ttc_0 0xf8001000 0xf8001fff
hernet_0 0xe000b000 0xe000bfff
7_usb_0  0xe0002000 0xe0002fff
7_qspi_0 0xe000d000 0xe000dfff
```

Figure 10 – Setting the Linux Application Debug Configuration

16. Keep in mind that that this application is a Linux application rather than a Standalone/bare-metal application. As a result, the Debug Configuration process is different than that used in the Zynq Intro SpeedWay exercises.

    Select the **linux_led_dimmer Debug** configuration from the left and launch the application debug configuration by clicking on the **Debug** button.



Debug Configurations

Create, manage, and run configurations

Filter matched 9 of 9 items

Name: linux_led_dimmer Debug

Connection: 192.168.1.10

Project: linux_led_dimmer

Build configuration: Debug

C/C++ Application: Debug\linux_led_dimmer.elf

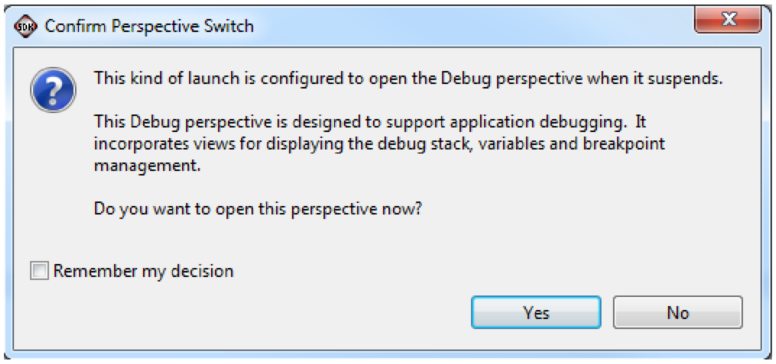
Remote Absolute File Path for C/C++ Application: /tmp/linux_led_dimmer.elf

Commands to execute before application

Skip download to target path.

Figure 11 – Selecting the Linux Application Debug Configuration

17. If prompted to confirm opening the Debug perspective, click the **Yes** button.



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

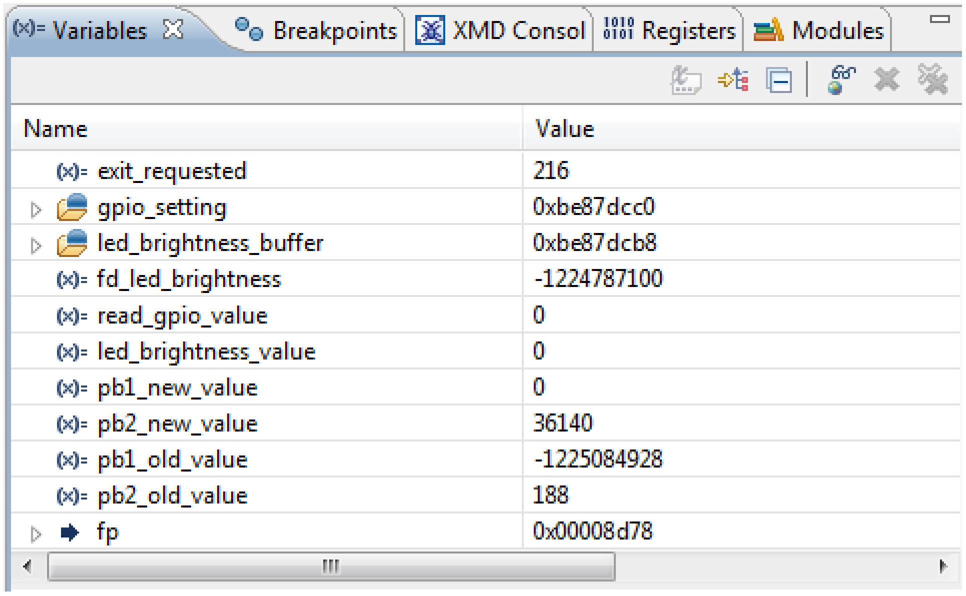
Do you want to open this perspective now?

Remember my decision

Figure 12 – Confirming the Perspective Switch

18. After the application debug configuration launches, observe that program operation is suspended at the first executable statement in **`main()`** (not running).

    Note that local variables for the current function are shown in the **Variables** tab.



Name	Value
exit_requested	216
gpio_setting	0xbe87dcc0
led_brightness_buffer	0xbe87dcb8
fd_led_brightness	-1224787100
read_gpio_value	0
led_brightness_value	0
pb1_new_value	0
pb2_new_value	36140
pb1_old_value	-1225084928
pb2_old_value	188
fp	0x00008d78

Figure 13 – Application Variables Panel

19. If the ZedBoard has been powered down or reset from the previous lab and the led-brightness driver module has not already been inserted into the kernel, it should be done at this point. If the led-brightness driver module is already inserted, skip ahead to step 24.

    To insert the led-brightness driver module, first mount the FAT32 file system on the SD card.

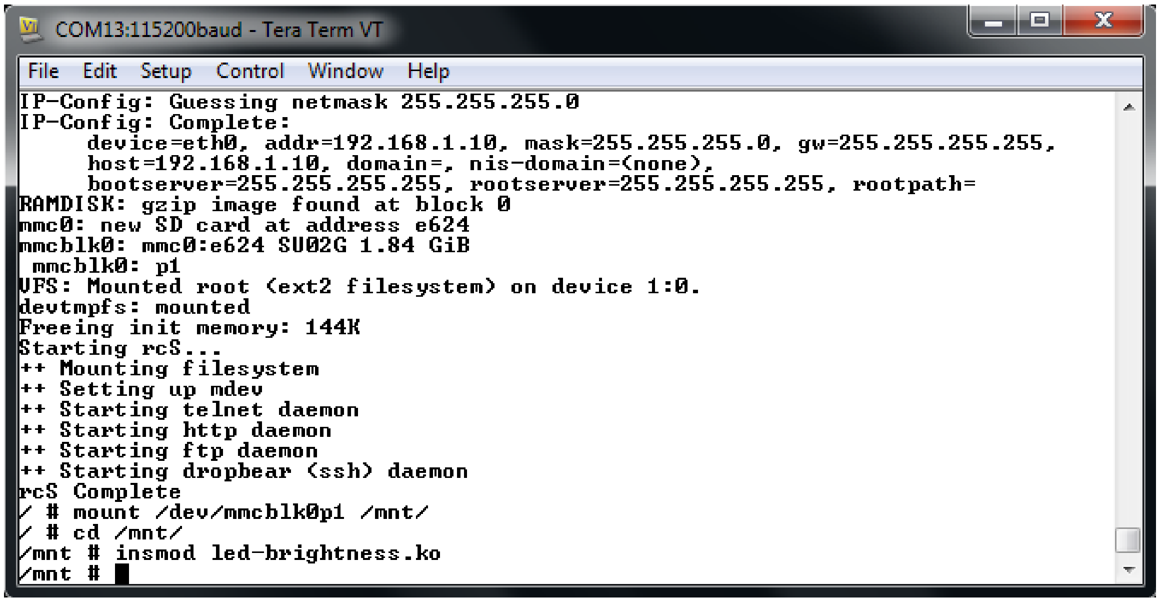
    The first partition on the SD card shows up as **mmcblk0p1** in the device tree. This device represents MMC block device 0, partition 1. We will mount this device to the mount point **/mnt** and change the working directory to that folder so that the FAT32 file system can be accessed.

```
# mount /dev/mmcblk0p1 /mnt/

# cd /mnt/
```

20. Now that the FAT32 partition on the SD card has been mounted, the led-brightness driver module that was placed in Experiment 1 can be loaded into the kernel using the **insmod** command.

```
# insmod led-brightness.ko
```

The good news is that there were no error messages thrown by the kernel when inserting the driver module. However, notice that none of the debug statements our driver sends to the console (via the `printk()` kernel call) are shown on the terminal.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

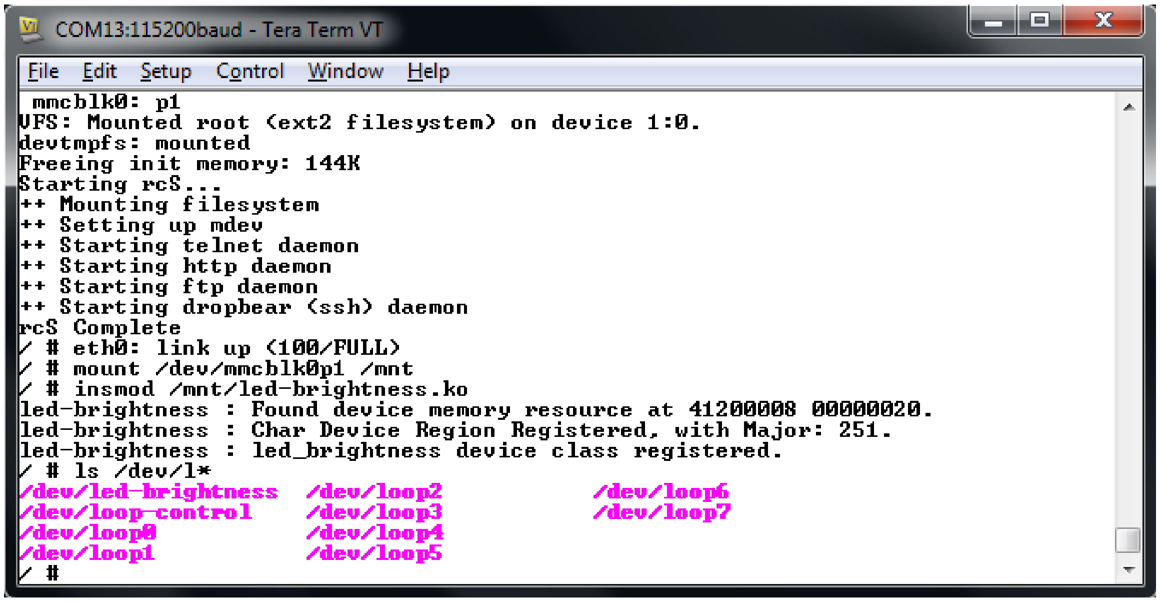
```
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
     device=eth0, addr=192.168.1.10, mask=255.255.255.0, gw=255.255.255.255,
     host=192.168.1.10, domain=, nis-domain=(none),
     bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
mmc0: new SD card at address e624
mmcblk0: mmc0:e624 SU02G 1.84 GiB
 mmcblk0: p1
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 144K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
rcS Complete
/ # mount /dev/mmcblk0p1 /mnt/
/ # cd /mnt/
/mnt # insmod led-brightness.ko
/mnt #
```

Figure 14 – Inserting the Driver Module Into the Kernel

21. Take a look in the **/dev** folder for the led-brightness character device.

```
# ls /dev/l*
```

Notice that this character device is now shown in the **/dev** folder. The driver is working as we expected and the device is ready for use in the rest of the lab.


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
 mmcblk0: p1
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 144K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
rcS Complete
/ # eth0: link up (100/FULL)
/ # mount /dev/mmcblk0p1 /mnt
/ # insmod /mnt/led-brightness.ko
led-brightness : Found device memory resource at 41200008 00000020.
led-brightness : Char Device Region Registered, with Major: 251.
led-brightness : led_brightness device class registered.
/ # ls /dev/l*
/dev/led-brightness  /dev/loop2           /dev/loop6
/dev/loop-control    /dev/loop3           /dev/loop7
/dev/loop0           /dev/loop4
/dev/loop1           /dev/loop5
/ #
```

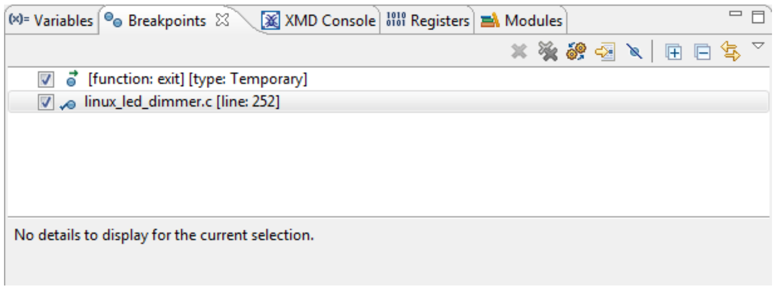
Figure 15 – Device Driver Module Inserted and led-brightness Devices Listed

22. Un-mount the SD card from the **/mnt** mount point.

```
# cd /

# umount /mnt/
```

23. Use the debugger to verify the switch toggling by using breakpoints and identify how to run and halt a program thread.

    Begin by selecting the **Breakpoints** panel. If this tab is not open, you may open the tab by selecting the **Window→Show View→Breakpoints** menu item.

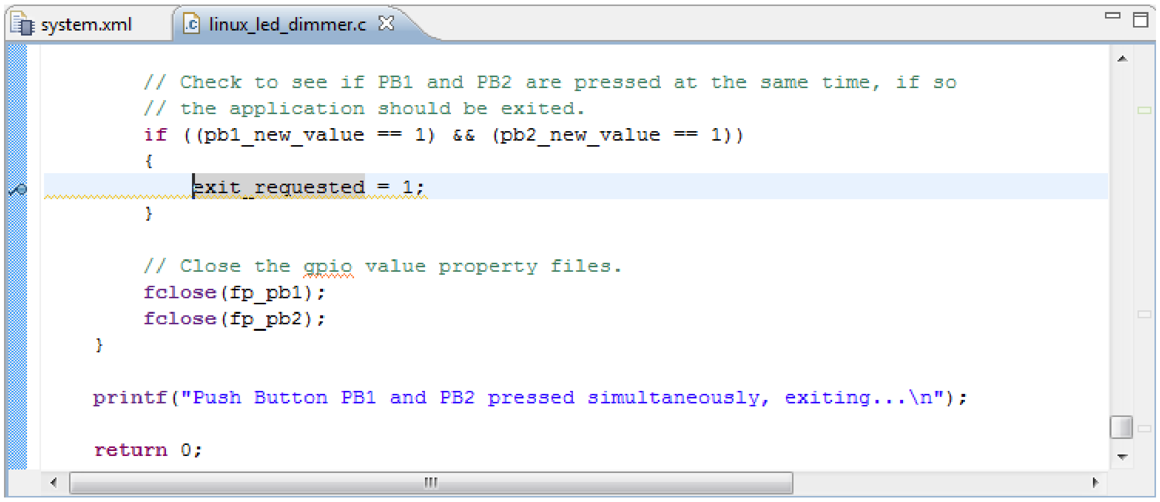


```
[function: exit] [type: Temporary]
linux_led_dimmer.c [line: 252]
```

No details to display for the current selection.

Figure 16 – Application Debug Breakpoints Panel

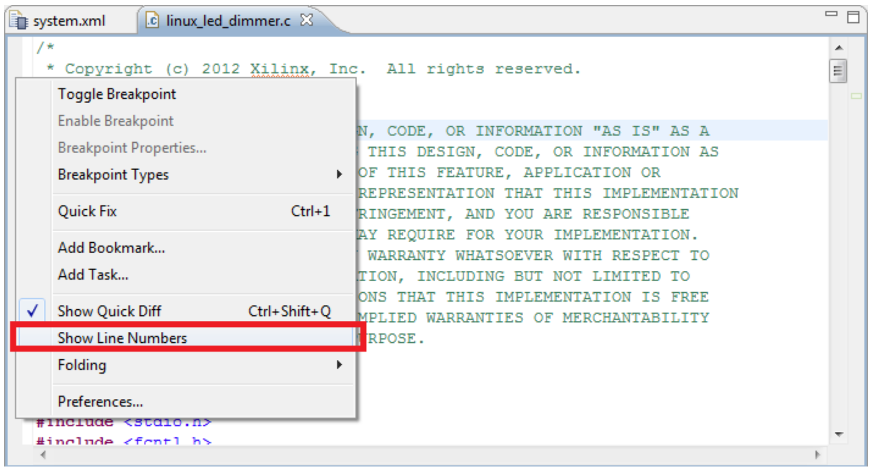
24. The **linux_led_dimmer.c** source file should be visible in the sources panel. Click within the source code panel to make it active.



```
        // Check to see if PB1 and PB2 are pressed at the same time, if so
        // the application should be exited.
        if ((pb1_new_value == 1) && (pb2_new_value == 1))
        {
            exit_requested = 1;
        }

        // Close the gpio value property files.
        fclose(fp_pb1);
        fclose(fp_pb2);
    }

    printf("Push Button PB1 and PB2 pressed simultaneously, exiting...\n");

    return 0;
```

Figure 17 – Application Debug Breakpoints Panel

25. The editor window panel has a line numbers feature that is turned off by default. To make it easier to locate lines of code referenced in this lab turn on line numbers in the editor panel. In the editor panel, right-click in the gray space to the right of the pane and select the **Show Line Numbers** option.



```
/*
 * Copyright (c) 2012 Xilinx, Inc.  All rights reserved.
                                  N, CODE, OR INFORMATION "AS IS" AS A
                                   THIS DESIGN, CODE, OR INFORMATION AS
                                  OF THIS FEATURE, APPLICATION OR
                                  REPRESENTATION THAT THIS IMPLEMENTATION
                                  RINGEMENT, AND YOU ARE RESPONSIBLE
                                  AY REQUIRE FOR YOUR IMPLEMENTATION.
                                   WARRANTY WHATSOEVER WITH RESPECT TO
                                  TION, INCLUDING BUT NOT LIMITED TO
                                  ONS THAT THIS IMPLEMENTATION IS FREE
                                  MPLIED WARRANTIES OF MERCHANTABILITY
                                  RPOSE.
#include <stdio.h>
#include <fcntl.h>
```

Figure 18 – Enabling the Line Numbers Feature

26. Set a breakpoint to check the push button switch toggling. Locate the line of control code that falls within the conditional statement which checks for both PB1 and PB2 being pressed simultaneously on line 252. The line number can also be seen in the lower toolbar of SDK. The first set of digits represents the text line number and the second set of digits represents the text column number.

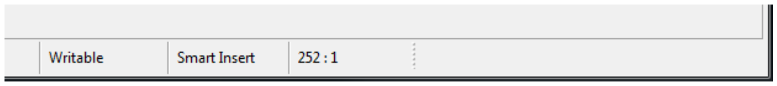
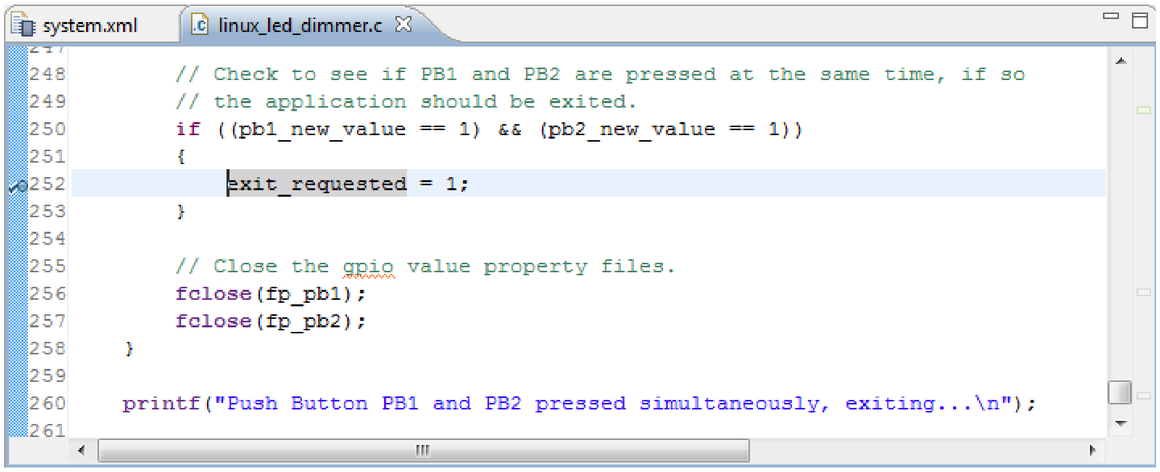


Figure 19 – SDK Lower Toolbar

Double-click on line number 252 within the shaded blue strip on the left of the source editor to set a breakpoint there (a check mark becomes visible). Note that the breakpoint that has just been set now appears in the **Breakpoints** panel.

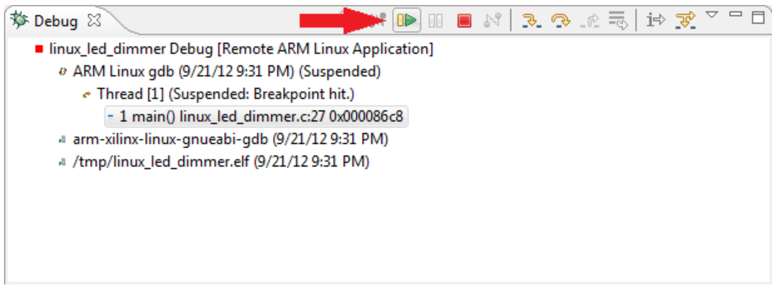


```
248         // Check to see if PB1 and PB2 are pressed at the same time, if so
249         // the application should be exited.
250         if ((pb1_new_value == 1) && (pb2_new_value == 1))
251         {
252             exit_requested = 1;
253         }
254 
255         // Close the gpio value property files.
256         fclose(fp_pb1);
257         fclose(fp_pb2);
258     }
259 
260     printf("Push Button PB1 and PB2 pressed simultaneously, exiting...\n");
261 
```

Figure 20 – Breakpoint Set in Editor Panel

27. Run the program by clicking the **Play/Resume** button (green triangle) to run the program.

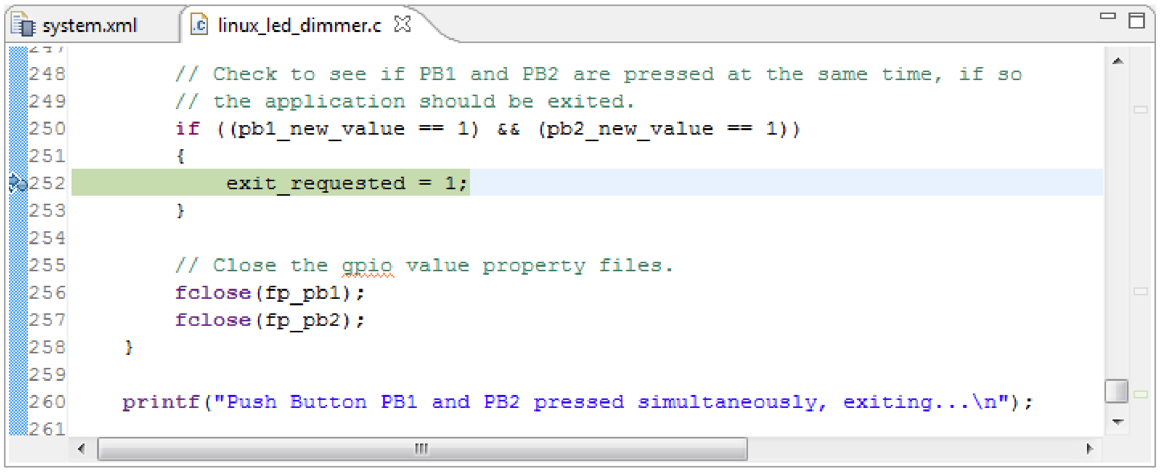
    If you encounter a series of launch errors, simply terminate and relaunch the debug session to recover.



```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() linux_led_dimmer.c:27 0x000086c8
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 21 – Play/Resume Execution Button

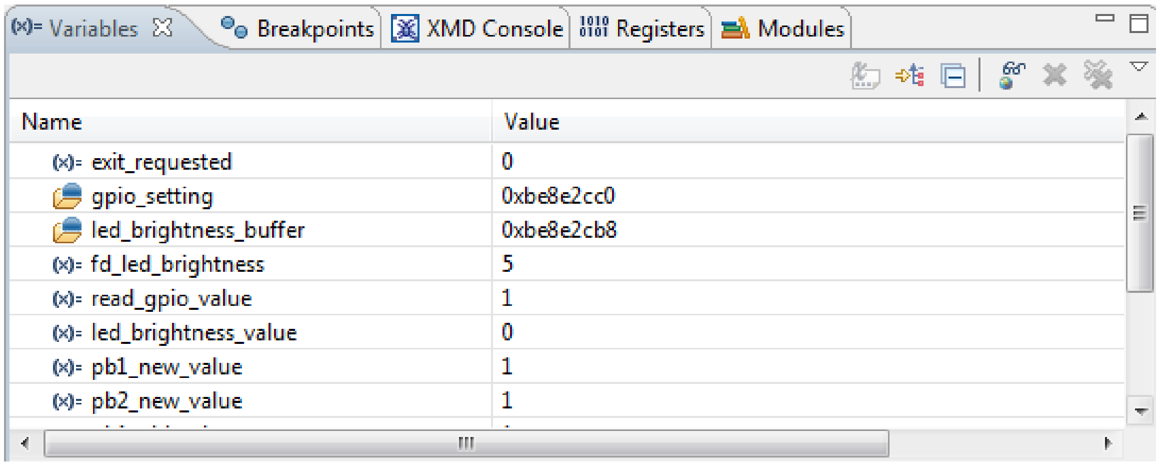
The program will stop at the breakpoint only if there is a change in the switch 2 of SW15 DIP switch. Push the push buttons PB1 and PB2 together simultaneously and watch as the program runs until the breakpoint is reached. The program suspends execution at line number 252.



```
248         // Check to see if PB1 and PB2 are pressed at the same time, if so
249         // the application should be exited.
250         if ((pb1_new_value == 1) && (pb2_new_value == 1))
251         {
252             exit_requested = 1;
253         }
254
255         // Close the gpio value property files.
256         fclose(fp_pb1);
257         fclose(fp_pb2);
258     }
259
260     printf("Push Button PB1 and PB2 pressed simultaneously, exiting...\n");
261
```

Figure 22 – Application Execution Suspended at Line 252 Breakpoint

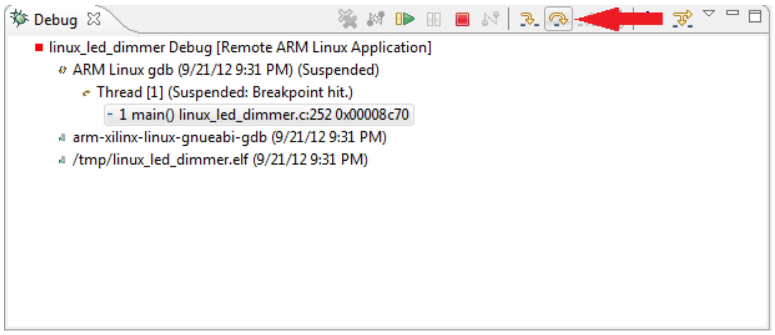
28. Take some time to evaluate each of the variables in the Variables panel. The **pb1_new_value** and **pb2_new_value** variables are getting assigned to a value of 1 as would be expected when both buttons are pressed together simultaneously.



Name	Value
exit_requested	0
gpio_setting	0xbe8e2cc0
led_brightness_buffer	0xbe8e2cb8
fd_led_brightness	5
read_gpio_value	1
led_brightness_value	0
pb1_new_value	1
pb2_new_value	1

Figure 23 – Application Variables

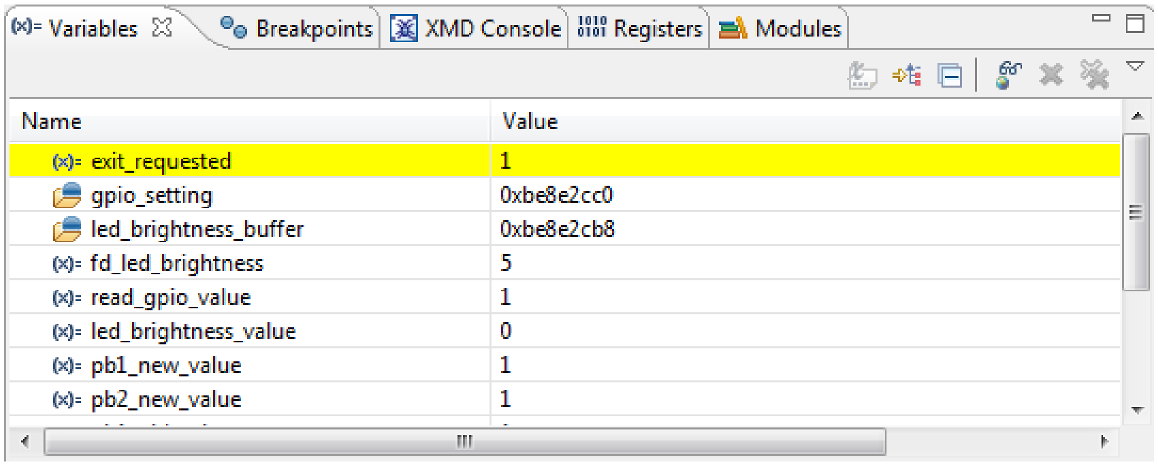
29. The **exit_requested** variable is still set to a value of 0 and if we execute the next line of code, the **exit_requested** variable should then become set to a value of 1. Click on the **Step Over** button to advance code execution ahead by one line so that the updated value of the **exit_requested** variable can be observed.



```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() linux_led_dimmer.c:252 0x00008c70
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 24 – Step Over Execution Control Button

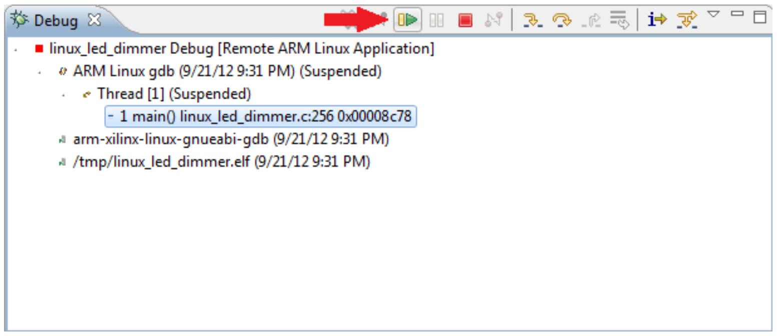
30. Application execution advances to the next line of code and again is suspended. Notice now that the **exit_requested** variable has been updated as expected to a value of 1 and automatically highlighted as a variable that has been modified since the previous suspend on breakpoint.



Name	Value
exit_requested	1
gpio_setting	0xbe8e2cc0
led_brightness_buffer	0xbe8e2cb8
fd_led_brightness	5
read_gpio_value	1
led_brightness_value	0
pb1_new_value	1
pb2_new_value	1

Figure 25 – Application Variables Updated

31. Continue executing the program by clicking the **Play/Resume** button (green triangle) to resume the program execution again. Notice how the application does not exit as it is expected to.

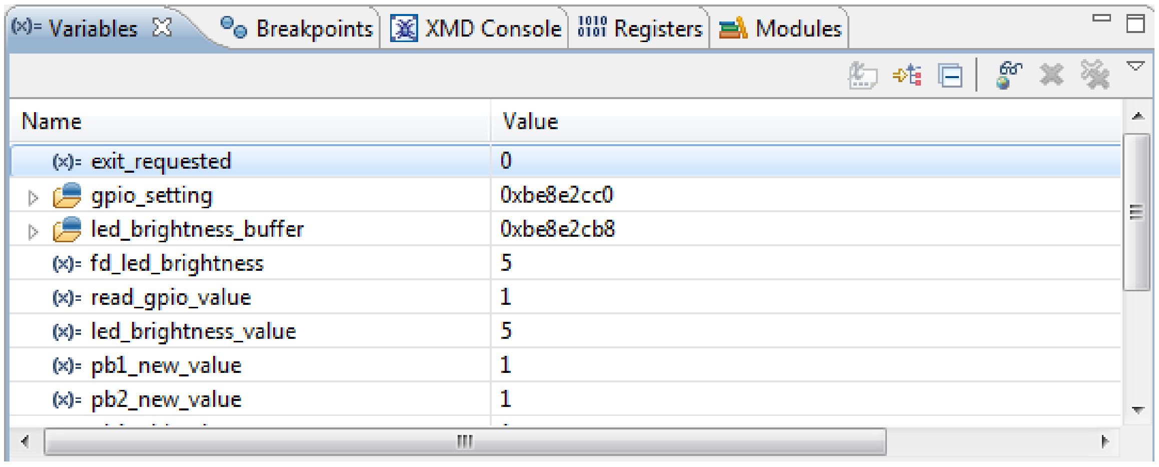


```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended)
      1 main() linux_led_dimmer.c:256 0x00008c78
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 26 – Play/Resume Execution Button

32. Push the push buttons PB1 and PB2 together simultaneously and watch the program run until the breakpoint is reached. The program again suspends execution at line number 252.

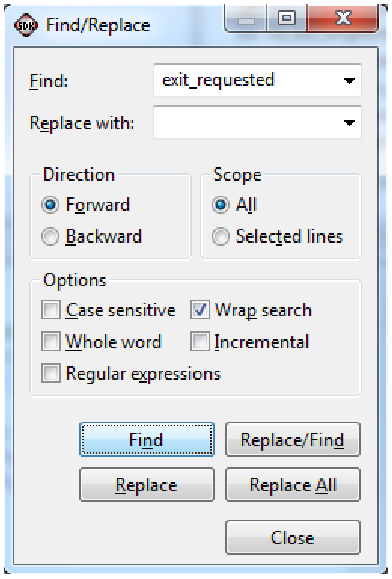
    Notice the **exit_requested** variable has been set back to a value of 0 which is a clue to the nature of the bug that is being tracked down.



Name	Value
exit_requested	0
gpio_setting	0xbe8e2cc0
led_brightness_buffer	0xbe8e2cb8
fd_led_brightness	5
read_gpio_value	1
led_brightness_value	5
pb1_new_value	1
pb2_new_value	1

Figure 27 – Application Variables Updated

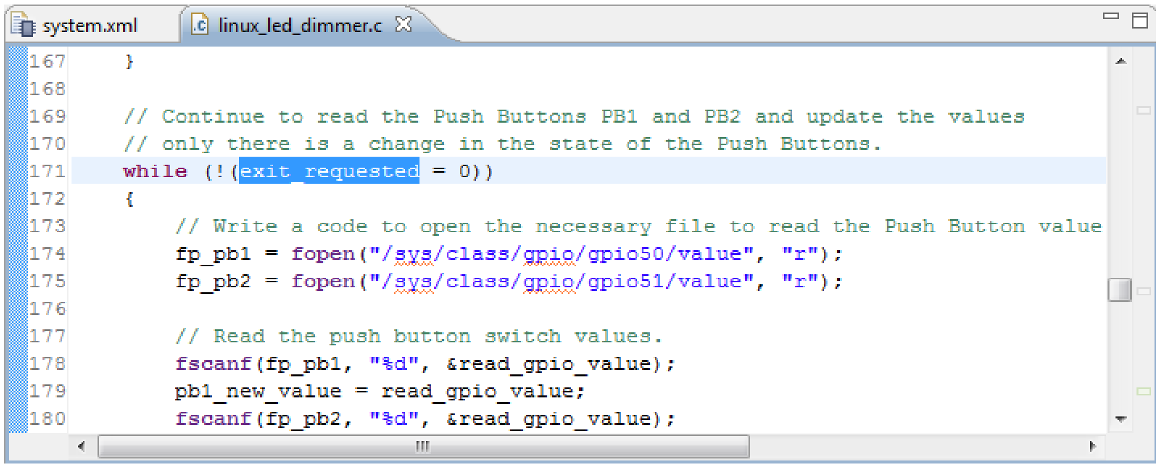
33. Manually looking through the source code main control loop for other instances of the **exit_requested** variable could take up valuable developer time. Click within the source panel and use the **<ctrl>-f** keyboard shortcut to open the **Find/Replace** tool and enter **exit_requested** in the **Find** field. Click the **Find** button to locate other instances where this variable is used.



Find: exit_requested
Replace with:
Direction: [✓ Forward] Backward
Scope: [✓ All] Selected lines
Options: Case sensitive, [✓ Wrap search], Whole word, Incremental, Regular expressions

Figure 28 – The Find/Replace Tool

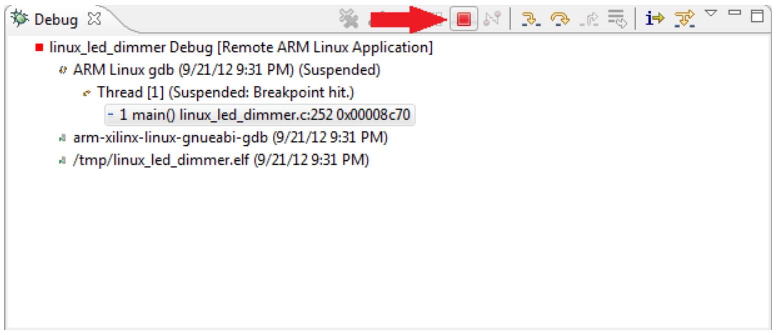
34. We find that the only other code locations where the **exit_requested** variable is accessed is on line 27 where it is initialized to zero before entering the main control loop and again on line 171 where it is evaluated as part of the control loop. Here we see something that is likely the cause of the bug that we have observed. Rather than a comparison statement within the parenthesis, the developer made a mistake by omitting one of the equal signs for a compare equals C operation and inadvertently made this into an assignment statement within a logic evaluation for a control statement. This can be a tricky type of bug to track down but now that it has been located, let us correct the control statement and see if that resolves the errant behavior we have observed.



```
167     }
168
169     // Continue to read the Push Buttons PB1 and PB2 and update the values
170     // only there is a change in the state of the Push Buttons.
171     while (!(exit_requested = 0))
172     {
173         // Write a code to open the necessary file to read the Push Button value
174         fp_pb1 = fopen("/sys/class/gpio/gpio50/value", "r");
175         fp_pb2 = fopen("/sys/class/gpio/gpio51/value", "r");
176
177         // Read the push button switch values.
178         fscanf(fp_pb1, "%d", &read_gpio_value);
179         pb1_new_value = read_gpio_value;
180         fscanf(fp_pb2, "%d", &read_gpio_value);
```

Figure 29 – Control Loop Improper Conditional Evaluation Statement

35. Stop debugger execution by clicking on the **Terminate** button in the Debug control panel.

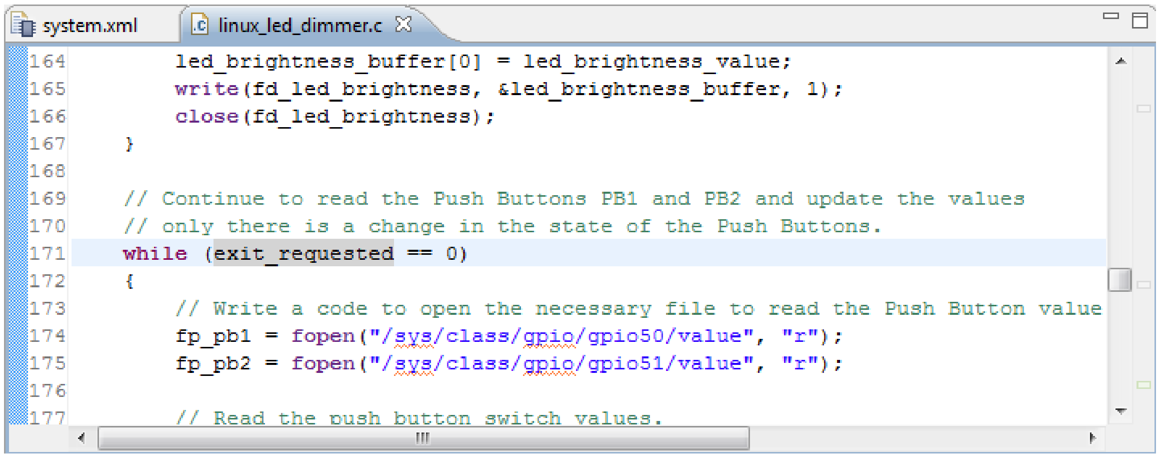


Debug

```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() linux_led_dimmer.c:252 0x00008c70
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 30 – Application Execution Terminate Button

36. Edit the source code to correct the main control loop conditional statement so that it is no longer an assignment statement. After saving the changes to the source file, notice that it is automatically rebuilt in the background.



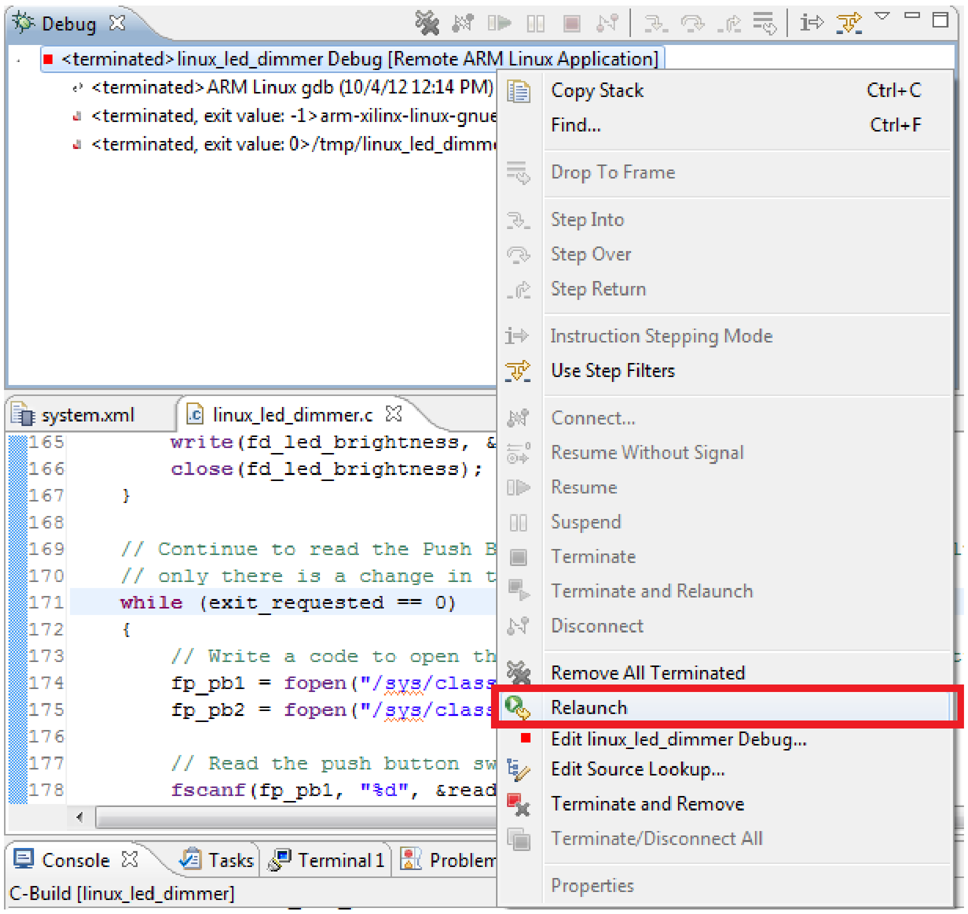
system.xml   linux_led_dimmer.c

```
        led_brightness_buffer[0] = led_brightness_value;
        write(fd_led_brightness, &led_brightness_buffer, 1);
        close(fd_led_brightness);
    }

    // Continue to read the Push Buttons PB1 and PB2 and update the values
    // only there is a change in the state of the Push Buttons.
    while (exit_requested == 0)
    {
        // Write a code to open the necessary file to read the Push Button value
        fp_pb1 = fopen("/sys/class/gpio/gpio50/value", "r");
        fp_pb2 = fopen("/sys/class/gpio/gpio51/value", "r");

        // Read the push button switch values.
```

Figure 31 – Editing the Source Code to Correct the Bug

37. Re-launch the Debug configuration so that we can verify that the bug has been corrected. In the **Debug** panel, right-click the **linux_led_dimmer Debug** configuration and select the **Relaunch** menu item.

    The debug configuration loads the updated target executable to the target platform and halts execution at the main control routine.



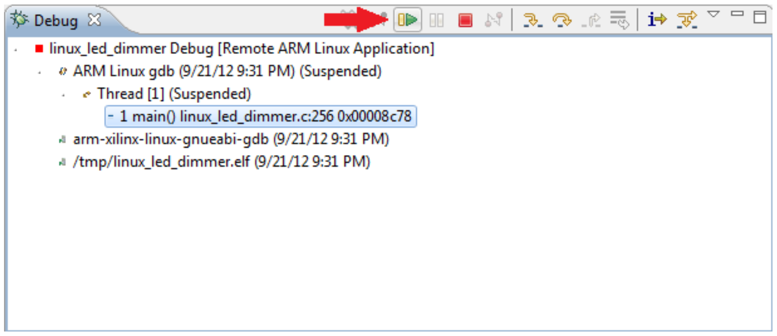
```
<terminated>linux_led_dimmer Debug [Remote ARM Linux Application]
    <terminated>ARM Linux gdb (10/4/12 12:14 PM)
    <terminated, exit value: -1>arm-xilinx-linux-gnue
    <terminated, exit value: 0>/tmp/linux_led_dimm
```

```
        write(fd_led_brightness, &
        close(fd_led_brightness);
    }

    // Continue to read the Push B
    // only there is a change in t
    while (exit_requested == 0)
    {
        // Write a code to open th
        fp_pb1 = fopen("/sys/class
        fp_pb2 = fopen("/sys/class

        // Read the push button sw
        fscanf(fp_pb1, "%d", &read
```

```
C-Build [linux_led_dimmer]
```

Figure 32 – Re-launching the Application Debug Configuration

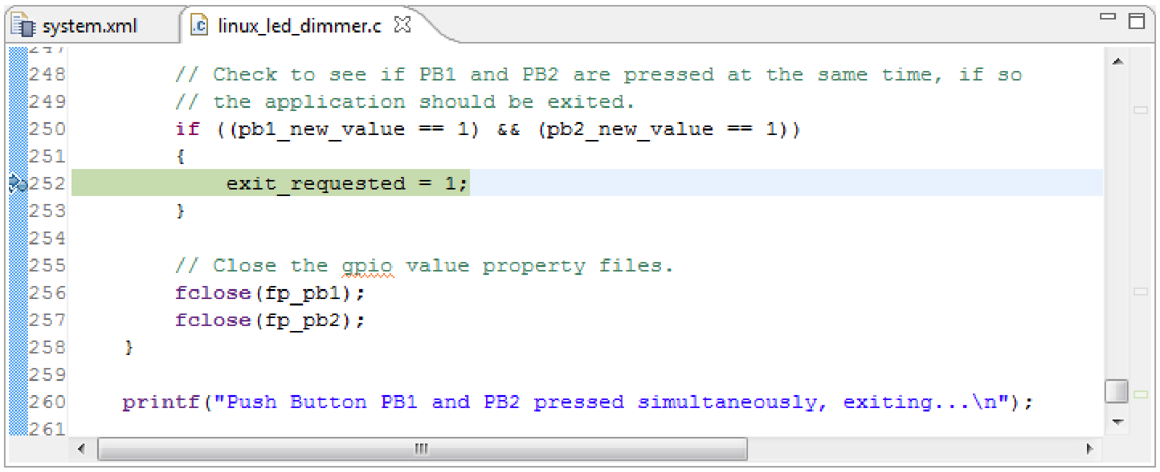
38. Use the debugger to verify the switch toggling by using breakpoints to evaluate whether the code changes were effective in resolving the bug. Run the program by clicking the **Play/Resume** button (green triangle) to run the program.



```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended)
      1 main() linux_led_dimmer.c:256 0x00008c78
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 33 – Play/Resume Execution Button

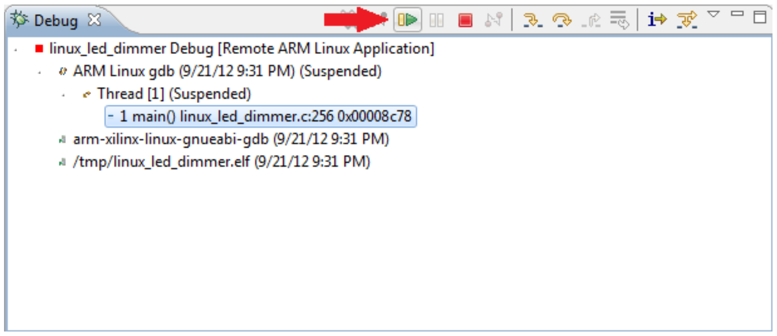
Push the push buttons PB1 and PB2 together simultaneously and watch as the program runs until the breakpoint is reached. The program again suspends execution at line number 252.



```
248        // Check to see if PB1 and PB2 are pressed at the same time, if so
249        // the application should be exited.
250        if ((pb1_new_value == 1) && (pb2_new_value == 1))
251        {
252            exit_requested = 1;
253        }
254
255        // Close the gpio value property files.
256        fclose(fp_pb1);
257        fclose(fp_pb2);
258    }
259
260    printf("Push Button PB1 and PB2 pressed simultaneously, exiting...\n");
261
```

Figure 34 – Application Execution Suspended at Line 252 Breakpoint

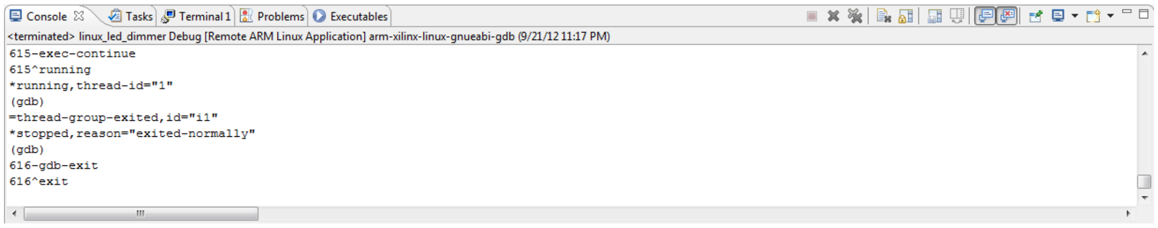
39. Continue executing the program by clicking the **Play/Resume** button (green triangle) to resume the program execution again.



```
linux_led_dimmer Debug [Remote ARM Linux Application]
  ARM Linux gdb (9/21/12 9:31 PM) (Suspended)
    Thread [1] (Suspended)
      1 main() linux_led_dimmer.c:256 0x00008c78
  arm-xilinx-linux-gnueabi-gdb (9/21/12 9:31 PM)
  /tmp/linux_led_dimmer.elf (9/21/12 9:31 PM)
```

Figure 35 – Play/Resume Execution Button

Notice how the application now exits as it is expected to. The bug has been resolved by correcting the control statement as performed in step 36.



```
<terminated> linux_led_dimmer Debug [Remote ARM Linux Application] arm-xilinx-linux-gnueabi-gdb (9/21/12 11:17 PM)
615-exec-continue
615^running
*running,thread-id="1"
(gdb)
=thread-group-exited,id="i1"
*stopped,reason="exited-normally"
(gdb)
616-gdb-exit
616^exit
```

Figure 36 – Application Exited Cleanly As Expected

40. Exit SDK through the **File→Exit** menu item.

Questions:

Answer the following questions:

- *Why did the assignment statement prevent the application from exiting properly?*

____

____

____

Exploring Further

If you have additional time and would like to investigate more…

- Experiment with the SDK debug environment.

This concludes Lab 4.2.

Revision History

Date	Version	Revision
21 Sep 12	00	Initial Draft
01 Sep 12	01	Revised Draft
19 Oct 12	02	Course Release
14 Jan 13	05	ZedBoard.org Training Course Release

Resources

http://www.zedboard.org

http://www.xilinx.com/zynq

http://www.xilinx.com/planahead

http://www.xilinx.com/sdk

Answers

Experiment 1

- *Why did the assignment statement prevent the application from exiting properly?*

The assignment statement prevented the application from exiting properly because when used as a conditional evaluation, it always evaluates to a return the result of the assignment operation rather than a comparison. This is a common programming problem with languages such as C, where the assignment operator also returns the value assigned, and can be validly nested inside expressions (in the same way that a function returns a value). If the intention was to compare two values in a conditional statement, an assignment is quite likely to return a value interpretable as Boolean true, in which case the then clause will be executed, leading the program to behave unexpectedly. Some language processors (such as gcc) can detect such situations, and warn the programmer of the potential error. Unfortunately, in the case of the bug evaluated in this lab, the original developer masked this error by wrapping the assignment within a set of parenthesis which overrides the warning that would otherwise be identified and thrown at compile time.