Avnet Technical Training Course

Xilinx Libraries
Lab 10
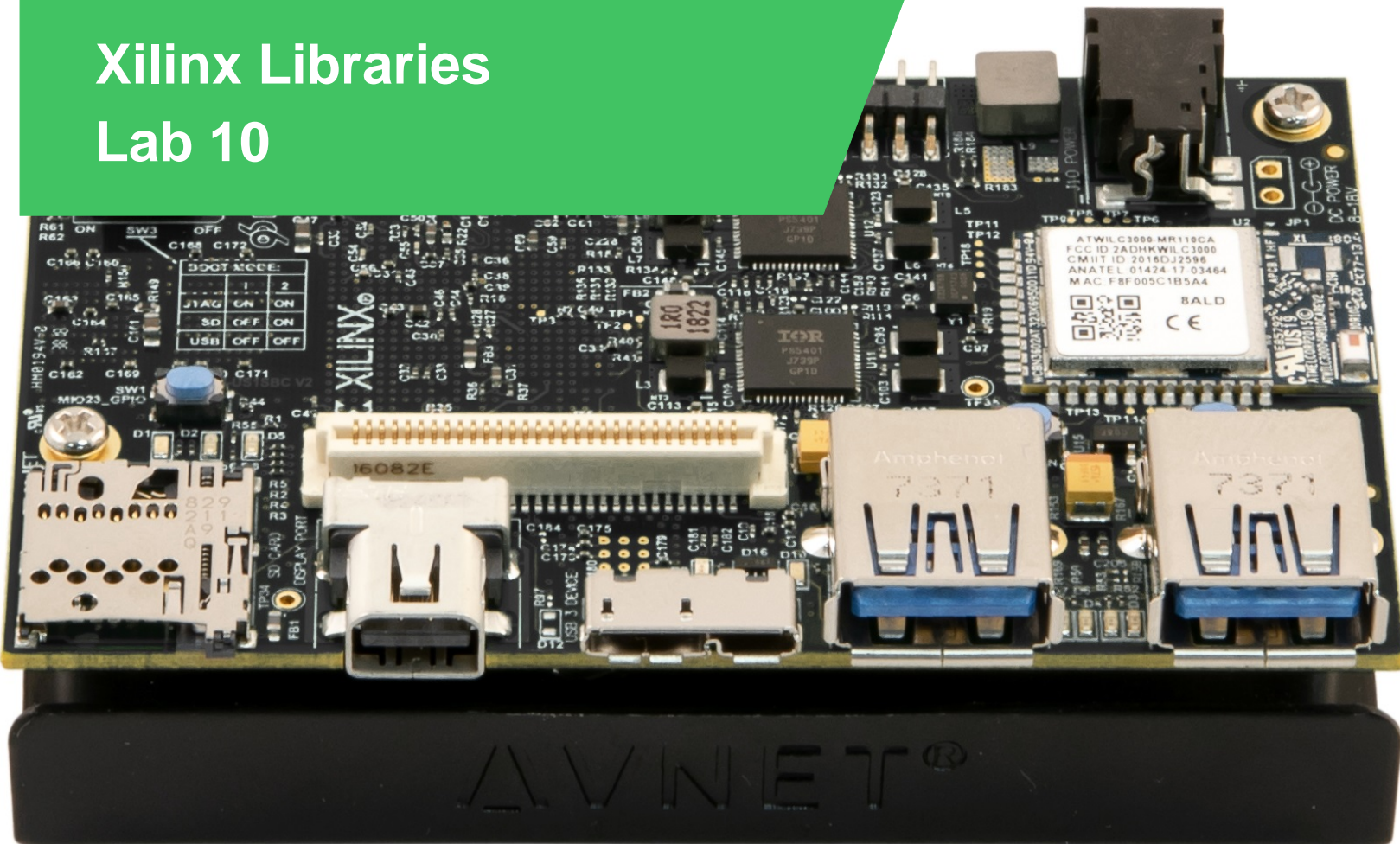
Tools:	2019.1
Training Version:	v11
Date:	July 2019

Lab 10 Overview

Many standalone (bare-metal) applications take advantage of libraries of reusable code in order to reduce application development time and leverage existing proven code.

Xilinx provides several libraries which can be built into a BSP as a selectable option. One of these libraries is the Xilinx In-system and Serial Flash (xilisf) Library which support the Xilinx In-system Flash and external serial Flash memories from Microchip/Atmel, Winbond, Intel/ST Microelectornics/Numonyx, and Micron devices. This library enables higher layer software (like an application) to communicate with the serial Flash device.

The library allows the user to Write, Read, and Erase the serial Flash. The user can also protect the data stored in the serial Flash from unwanted modification by enabling device specific Sector Protection features.

The library supports interrupt driven mode and polled mode based upon the mode in which the lower level SPI driver is configured by the user. The library can also support multiple instances of serial Flash at a time provided that they are of the same device family as the device family selection made at compile time. A serial Flash operates as a slave device on the SPI bus with a Xilinx SPI core operating as the bus master. The library uses lower level SPI drivers to communicate with the serial Flash so using this library is an excellent way to abstract your application away from these lower level implementation details.

In this lab, an application which reads and writes MAC address (EUI-48 only) configuration data to/from the on-board QSPI Flash device is explored. For more information on what a MAC address is and why you might want to use one in an end application, please see the following Wikipedia article:

http://en.wikipedia.org/wiki/MAC_address

Lab 10 Objectives

When you have completed Lab 10, you will know:

- How to enable one of the Xilinx libraries within your standalone BSP

- How to leverage one of the Xilinx libraries from a standalone application

Experiment 1: Create the Application Project

Similar to the flow for importing existing application code in Lab 9, a new blank project is created and existing code is imported. The application code is then executed to verify the application code we were given functions as expected.

Experiment 1 General Instruction:

Create a new blank software application project. Import code from the following folder:

/home/training/AvnetTTC/ZynqSW/2019_1/Support_documents/flash_mac_app/

Run the application on the target hardware and observe the behavior.

Experiment 1 Step-by-Step Instructions:

1. Create a new SDK software application project by selecting the **File**➔**New**➔ **Application Project** menu item.

2. In the **New Project** wizard, change the **Project name** field to the **flash_mac_app** name.

   Change the **Board Support Package** to the **Create New** option and use the default **flash_mac_app_bsp** suggested name.

   Leave the other settings to their default values. Click the **Next** button to continue.
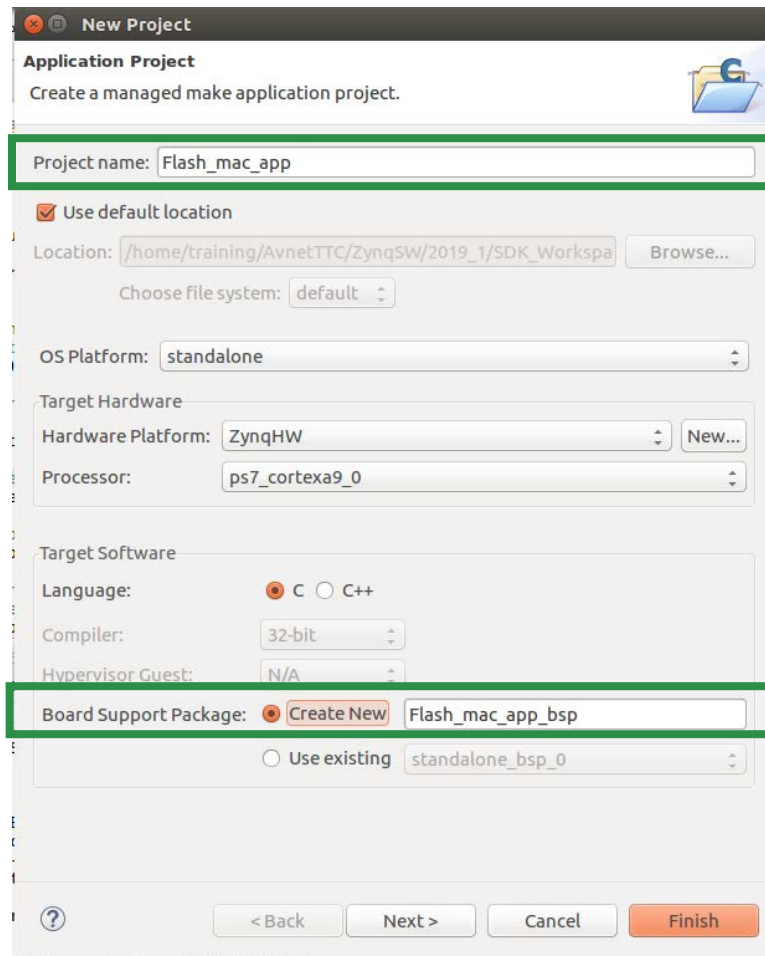
Figure 1 – Creating the flash_mac_app Application

3. Select the **Empty Application** project template and click the **Finish** button to complete the new project creation process using the Empty Application project template.

4. The empty **flash_mac_app** application project and **flash_mac_app_bsp** are created.

   Open the Board Support Package **system.mss** file if it is not already open in the Editor Pane.

5. Click on the **Modify this BSP's Settings** button to configure **flash_mac_app_bsp** settings.
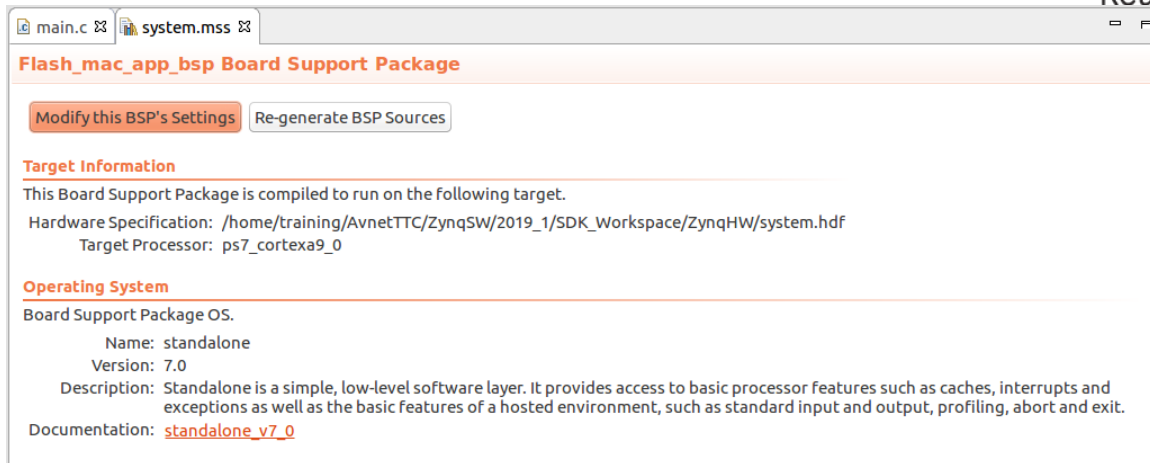
Figure 2 – Modify the BSP Settings for flash_mac_app_bsp Project

6. Enable the **Xilinx In-system and Serial Flash Library** for this BSP by selecting the checkbox next to the **xilisf** item in the **Supported Libraries** list.
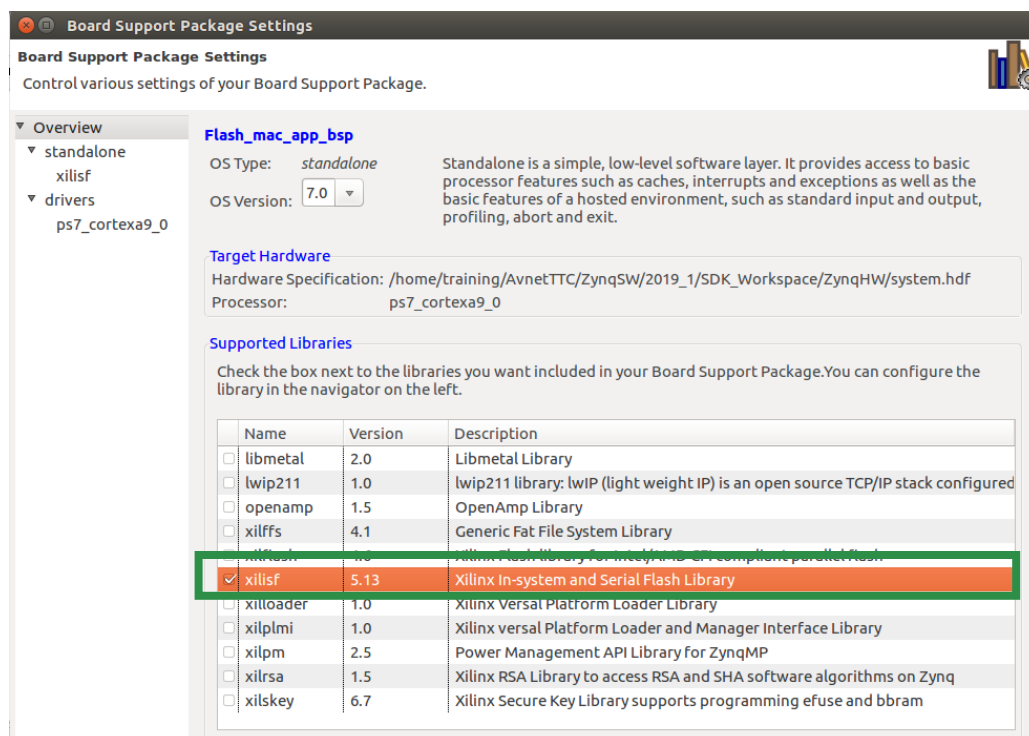

Figure 3 – Enable the Xilinx Library xilisf within the BSP Settings

7. Click on the **xilisf** listing under the tree to the left of the BSP Settings windows under **Overview→standalone**. This will open the **xilisf** library's specific settings.

Since we are using a Micron QSPI flash, according to the description for the **serial_flash_family** field, a value of **5** should be used to specify the Micron Flash family type. Change the **Value** entry for this field to **5**.

Since we are using a Micron QSPI flash, according to the description for the **serial_flash_interface** field, a value of **3** should be used to specify the QSPI Flash interface type. Change the **Value** entry for this field to **3**.

8. Similarly to what we did to previous BSPs, **select standalone** in the Board Support Package Settings window and then **change stdin and stdout to ps7_uart_1**.

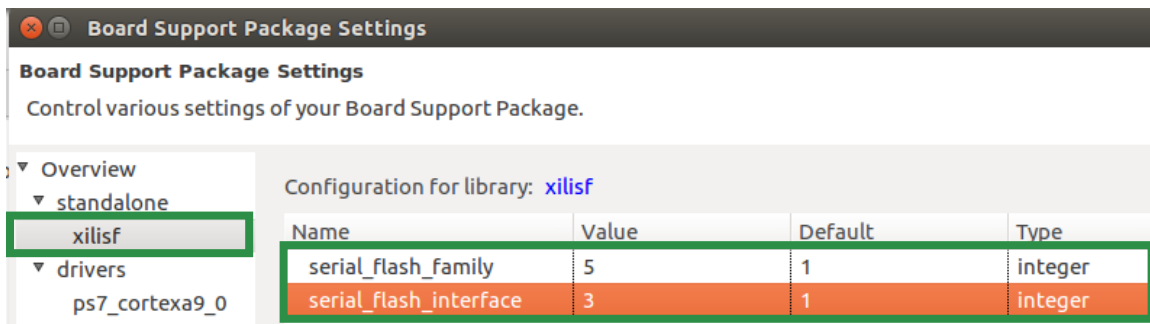   Click on the **OK** button to accept the updated BSP settings.

Figure 4 – Configure the Xilinx Library xilisf for Micron QSPI Flash

9. In the **Project Explorer** tab, expand **flash_mac_app** and right-click on the **src** folder.

   Click on the **Import** option in the pop up menu.

10. In the **Import** window, expand the **General** item, select the **File System** option, and click the **Next** button.

11. Click on the **Browse** button and select the following folder which contains the application code that we wish to start from:

   /home/training/AvnetTTC/ZynqSW/2017_4/Support_documents/flash_mac_app/

   After this folder is selected within the **Browse** dialog, click the **OK** button to search the folder for files to import into the application project.

   Click the **Select All** button to select the 18 source code files and then click the **Finish** button to complete the **Import** operation.
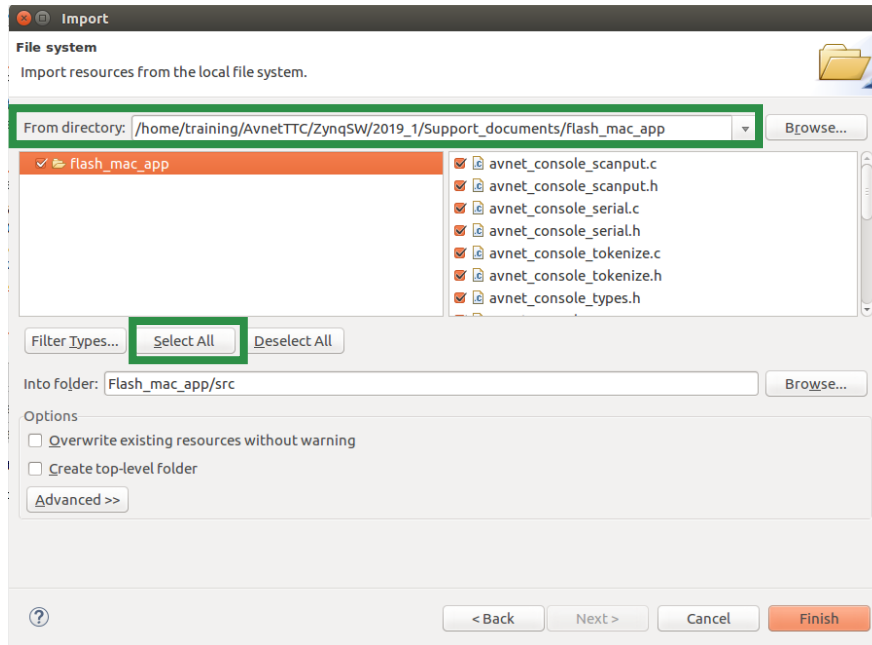
Figure 5 – Selecting flash_mac_app Application Source Files

12. The SDK Console panel shows the results of the build. Make sure that the application is built without errors.
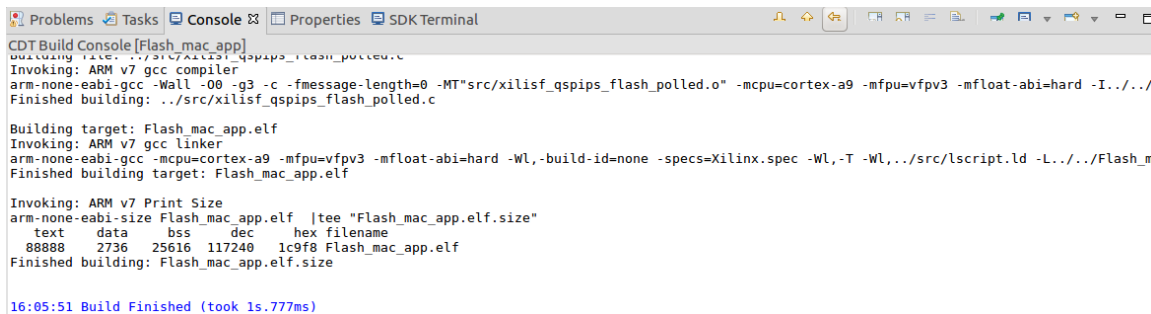

Figure 6 – Application Build Console Window

13. After SDK finishes compiling the new application code, the ELF is available in the following location:
    /home/training/AvnetTTC/ZynqSW/2019_1/SDK_Workspace/flash_mac_app/Debug/flash_mac_app.elf

    At this point the application is ready to be launched on the target hardware.

Answer the following questions:

- *What if you wanted to reuse part of this application on your own Zynq design but you prefer to use Microchip Flash instead of Micron Flash?*

  _____

- *Why is a separate BSP created for this application? Why not reuse the standalone_bsp_0 from earlier lab activities?*

  _____

Experiment 2: Exploring the Application

In this experiment, we will launch the application and experiment with reading and writing to the Micron QSPI Flash device from the application.

This is accomplished by the use of the BSP library **xilisf** API. The following API calls are used to add serial flash support to an application:

```
int    Xisf_Initialize
       (XIsf *InstancePtr,
       XIsf_Iface *SpiInstPtr,
       u8 SlaveSelect,
       u8 WritePtr)
```

- The geometry of the underlying serial Flash device is determined by reading the Joint Electron Device Engineering Council (JEDEC) Device Information and the Status Register of the Serial Flash device.
- A blocking call which reads the JEDEC information of the Flash device and waits until the transfer is complete before checking to see if the information is valid.

```
int    Xisf_Erase
       (XIsf *InstancePtr,
       XIsf_EraseOperation Operation,
       u32 Address)
```

- Erases the contents of the specified memory in the serial Flash.

```
int    Xisf_Read
       (XIsf *InstancePtr,
       XIsf_ReadOperation Operation,
       void *OpParamPtr)
```

- Reads data from the serial Flash.

```
int    Xisf_Write
       (XIsf *InstancePtr,
       XIsf_WriteOperation Operation,
       void *OpParamPtr)
```

- Writes the specified data to the serial Flash.

Experiment 2 General Instruction:

Launch the **flash_mac_app** on the target hardware and experiment with reading the MAC address as well as setting and storing a new MAC address.

Explore the application source code.

Experiment 2 Step-by-Step Instructions:

1. After completing Experiment 1, the application is ready to be launched on the target hardware.

2. **Press Reset** (SW2) on MiniZed.

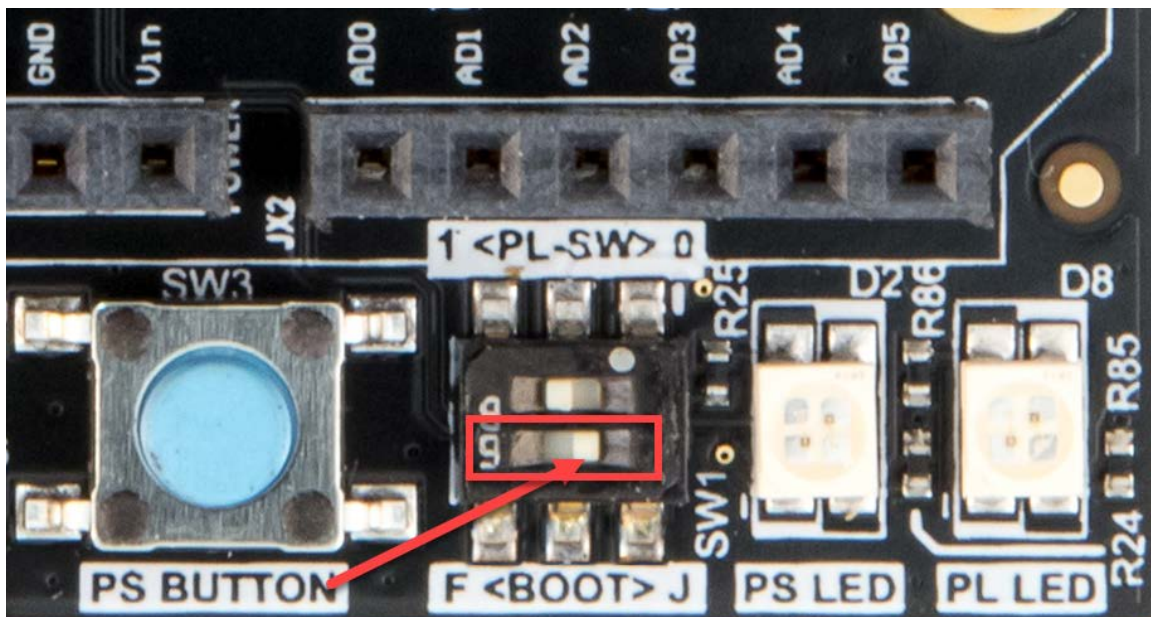3. Set the Boot Mode jumpers to Cascaded JTAG Mode

   a. MiniZed

Figure 7 – MiniZed – JTAG Boot

4. Connect to JTAG as follows:

   a. MiniZed – Connect microUSB to MiniZed USB-UART-JTAG (J2) port and your Host machine.

5. In SDK, select **Xilinx Tools → Program FPGA** or click the 🔧 icon.

6. SDK will already know the correct .bit file since this was imported with the hardware platform. Click the **Program** button.

   When MiniZeds D3 lights blue, the PL has configured successfully. Look for the message "FPGA configured successfully with bitstream" in the **SDK Log** window.

7. Launch a terminal program (GTK Term) with the 115200/8/n/1/n settings.

8. In the **Project Explorer** tab, right-click on the **flash_mac_app** project folder.

   Click on the **Run As→ Launch on Hardware (System Debugger)** option in the pop up menu.

9. Using the Launch on Hardware option will automatically create a run configuration using default settings and launch the application on the target platform. Once the application is launched and is running on the target hardware, a command prompt should appear in the terminal window.

Figure 8 – Initial Terminal Output from flash_mac_app Application

10. Type the command **help** at the prompt and press enter. This will display a listing of the commands supported by this application.

Figure 9 – Initial Terminal Output from flash_mac_app Application

11. One of the commands listed is the **mac** command. Discover how to use this command by typing **mac help** at the prompt and then press enter. This will display the usage for the **mac** command.

```
---------------------------------------------------------------------
AVNET>mac help
Syntax :
        mac                     => Display MAC address value from Flash
        mac {XX-XX-XX-XX-XX-XX} => Write specified MAC address to Flash

AVNET>
```
/dev/ttyUSB1 115200-8-N-1

Figure 10 – Initial Terminal Output from flash_mac_app Application

12. Use the **mac** command to read and display the current MAC address stored in the QSPI Flash.

Since this is likely the first time this command has been run on memory, it is likely that a bunch of random characters are present in the board configuration data space we are using at **0x00090000**. In the example shown here, all zero data is found.

```
AVNET>mac
        Reading MAC Address from Flash... SUCCESS
        MAC Address: 00-00-04-00-00-01

AVNET>
```
/dev/ttyUSB1 115200-8-N-1

Figure 11 – Displaying Initial Flash Data in the MAC Address Space

13. Let's set a new MAC address for this board by again using the mac command. This time, the desired MAC address value will be specified using the following command:

```
AVNET> mac 00-1C-ED-C0-FF-EE
```

```
AVNET>mac 00-1C-ED-C0-FF-EE
        Writing new MAC Address to Flash... SUCCESS
        Reading MAC Address back from Flash... SUCCESS
        MAC Address: 00-1C-ED-C0-FF-EE
```

Figure 12 – Programming a New MAC Address Value into Flash Memory

14. The new MAC address value is now programmed into the QSPI Flash. You can display the new value read back from the Flash with the **mac** command.

If you are suspicious about the data being cached in volatile memory, you can also power cycle your board and re-run the application to demonstrate that the MAC address value is indeed stored in non-volatile Flash memory.

15. Let's explore how this application takes advantage of the XilISF library API to access the QSPI flash.

Expand the **flash_mac_app → src** folder in the **Project Explorer** panel. Double click the application source file **flash_mac.c** to open it in the code editor.

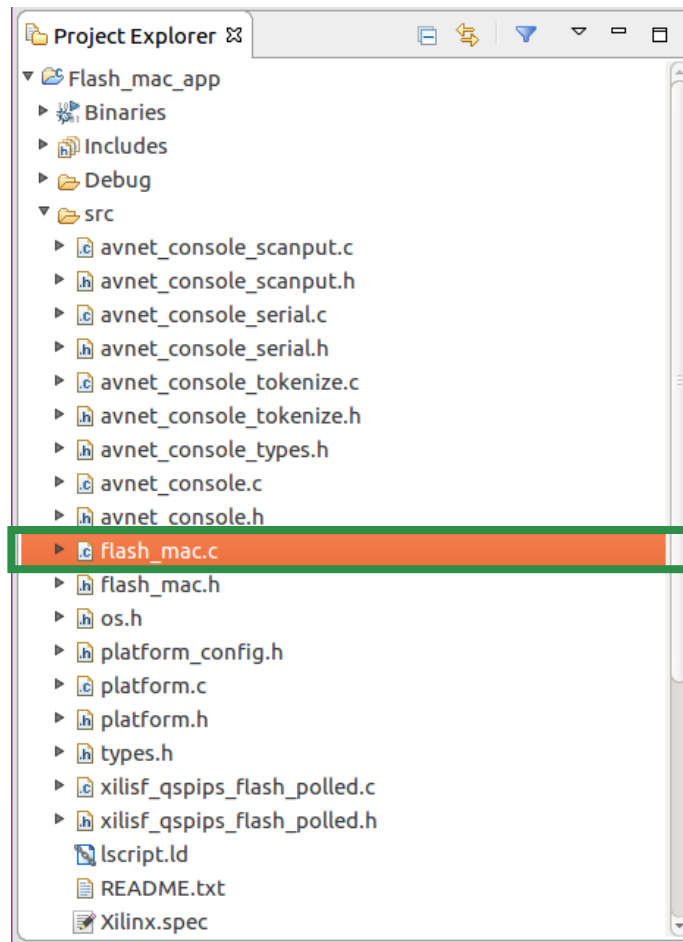**Figure 13 – flash_mac_app Source Code**

16. In the source file, line 123 shows a call to `FlashInit()` which in turn makes a call to `QsipiFlashPolledInit()` which is code leveraged from this Xilinx example file:

   /home/training/SDK/2019.1/data/embeddedsw/lib/sw_services/xilisf_v5_9/examples/xilisf_qspips_stm_polled_example.c

   The code from the example application was adapted for the MAC address application we used in this lab with very little modification to achieve physical Flash data access.

17. There are a few additional routines which were written to support this application.

   The `FlashErase()` function allows data sectors to be erased in preparation for writing new data to the Flash memory. This function takes advantage of the **xilisf** library `Xisf_Erase()` API call.

   The `FlashRead()` function provides support for reading MAC address data from the QSPI Flash. This function takes advantage of the **xilisf** library `Xisf_Read()` API call.

The **FlashWrite()** function provides support for writing MAC address data to the QSPI Flash. This function takes advantage of the **xilisf** library **Xisf_Write()** API call.

> ***Answer the following questions:***
>
> - *How were the functions within the application code which access the flash written? Was it written entirely from scratch?*
>
>   _____

Exploring Further

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

- Add another Xilinx library to your standalone BSP.

- Explore the examples provided for various software libraries within the respective example subfolders under the following directory:

    /home/training/SDK/2019.1/data/embeddedsw/lib/sw_services/

This concludes Lab 10.

Revision History

Date	Version	Revision
12 Nov 13	01	Initial Release
25 Nov 13	02	Revisions after pilot
14 Dec 14	03	Review for update to Vivado 2014.3
07 Jan 15	04	Update to 2014.4
09 Mar 15	05	Merge MicroZed and ZedBoard instructions
18 Mar 15	06	Finalize SDK 2014.4
Oct 15	07	Updated to SDK 2015.2
Aug 16	08	Updated to SDK 2016.2
Jun 17	09	Updated to SDK 2017.1 for MiniZed + Rebranding
Feb 18	10	Updated to Vivado/SDK 2017.4
July 19	11	Updated to Vivado/SDK 2019.1

Resources

www.minized.org

www.microzed.org

www.picozed.org

www.zedboard.org

www.em.avnet.com/drc

www.xilinx.com/zynq

www.xilinx.com/sdk

www.xilinx.com/vivado

www.xilinx.com/products/silicon-devices/soc/zynq-7000/ecosystem/index.htm

Answers

Experiment 1

> - *What if you wanted to reuse part of this application on your own Zynq design but you prefer to use Microchip Flash instead of Micron Flash?*
>
> Avnet also distributes Michrochip (Atmel) devices and the only part of this software application that would need to be changed is the **serial_flash_family** field from Experiment 1, Step 10.
>
> - *Why is a separate BSP created for this application? Why not reuse the standalone_bsp_0 from earlier lab activities?*
>
> A separate BSP was created since the Xilinx xilisf library was to be added specifically to support this MAC programming application. We could have instead modified the standalone_bsp_0 to add the xilisf library but that would cause the library code to built into the BSP each time the project is cleaned and rebuilt. This would unnecessarily increase build time for the full workspace which might not be desirable. However, if xilisf were built into the BSP but not called by our other applications, it would NOT affect their executable code size.

Experiment 2

> - *How were the functions within the application code which access the flash written? Was it written entirely from scratch?*
>
> Not really, the flash access functions found within the **xilisf_qspips_flash_polled.c** source file are adapted from the example code found within the Xilinx **xilisf** library **/home/training/SDK/2017.4/sw/lib/sw_services/xilisf_v5_9/examples/** folder.