

Developing Zynq Software with Xilinx SDK

Lab 11

Xilinx Libraries



November 2013
Version 03

Lab 11 Overview

Many standalone or baremetal 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 Atmel, Winbond, Intel/ST Microelectronics/Micron/Numonyx, and Spansion 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 unwarranted 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 instance 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 on in an end application, please see the following Wikipedia article:

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

IMPORTANT NOTE: *This lab makes use of the ability to configure SDK for using a local repository. If at any point after this lab, you decide to use a lab solution, be sure to also configure SDK for using a local repository by following the Experiment 1, Steps 4 and 5 as well.*

Lab 11 Objectives

When you have completed Lab 11, 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 8, 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:

C:\Speedway\ZynqSW\2013_3\Support_documents\flash_mac_app

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

Experiment 1 Step-by-Step Instructions:

1. Launch Xilinx Software Development Kit (SDK) if not already open. **Start → All Programs → Xilinx Design Tools → Vivado 2013.3 → SDK → Xilinx SDK 2013.3.**



Figure 1 – The SDK Application Icon

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

C:\Speedway\ZynqSW\2013_3\SDK_Workspace

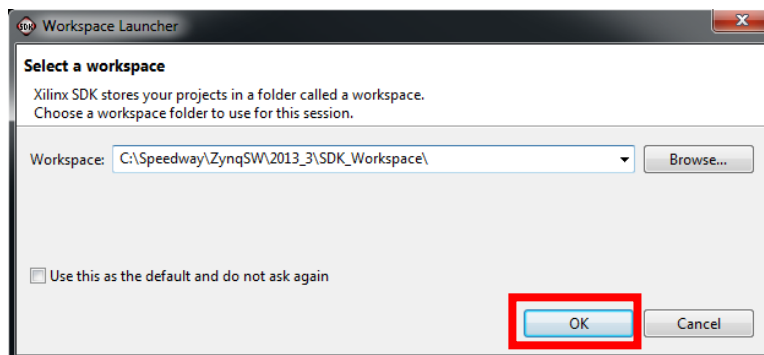


Figure 2 – Switching to the Appropriate SDK Workspace

3. Close the Welcome screen if it appears in the SDK window by clicking on the **X** control in the tab.

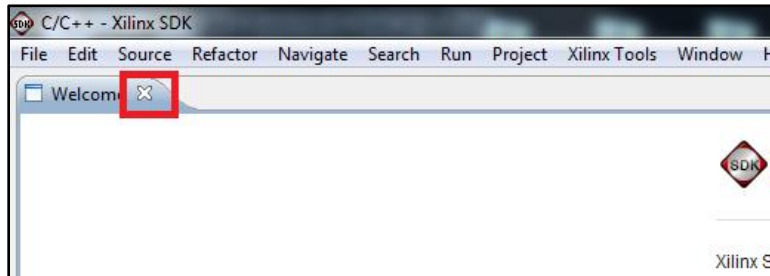


Figure 3 – Closing the SDK Welcome screen

4. Make sure that SDK is configured with the local repository which contains the ZedBoard supported version of the **xililf** library.

Open the SDK repository preferences configuration by selecting the **Xilinx Tools→Repositories** menu item.

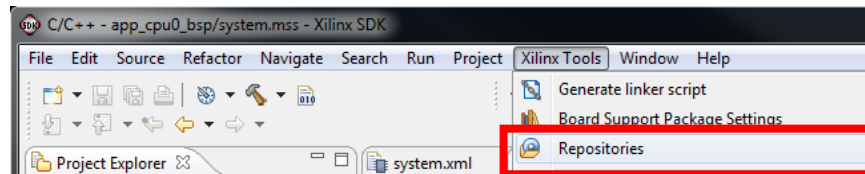


Figure 4 – Configuring SDK Repositories

5. Confirm that a Local Repository item listing is present for the following folder:

C:\Speedway\ZynqSW\2013_3\Support_documents\sdk_repo

If there is not a listing present for this folder, add a new Local Repository entry by clicking the **New** button beside the **Local Repositories** listing.

Browse to the following folder which contains the local repository containing the ZedBoard supported version of **xilisf** library:

C:\Speedway\ZynqSW\2013_3\Support_documents\sdk_repo

After this folder is selected within the **Browse For Folder** dialog, click the **OK** button to add the folder to the Local Repositories listing.

Click the **Apply** and then the **OK** buttons to close the SDK preferences window.

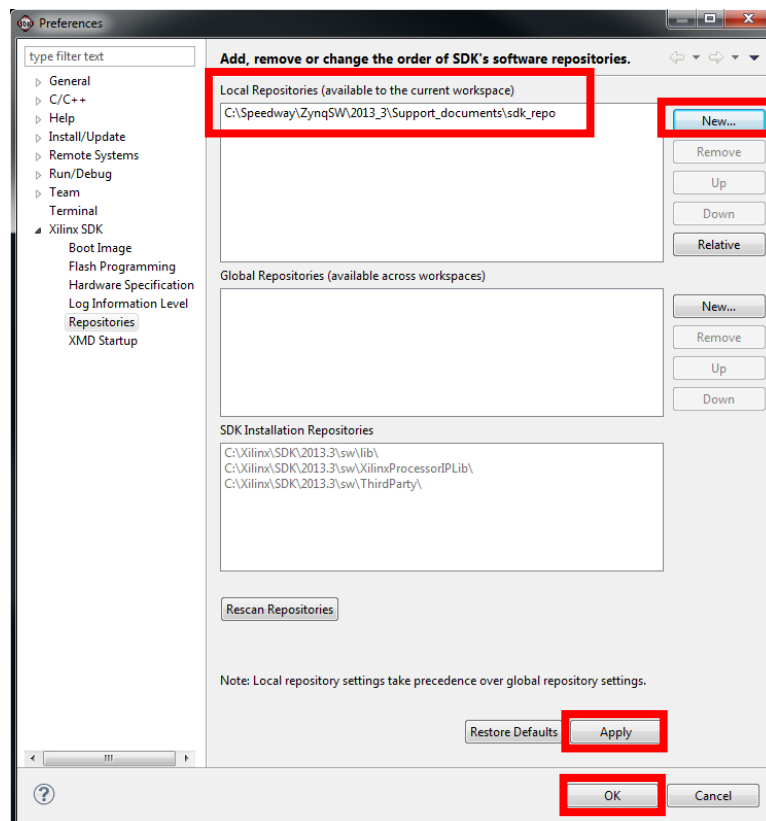


Figure 5 – Adding a Local Repository

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

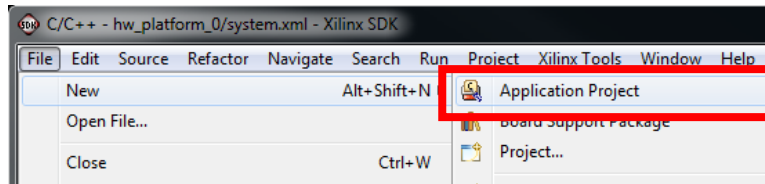


Figure 6 – Creating a New C Application Project

7. 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.

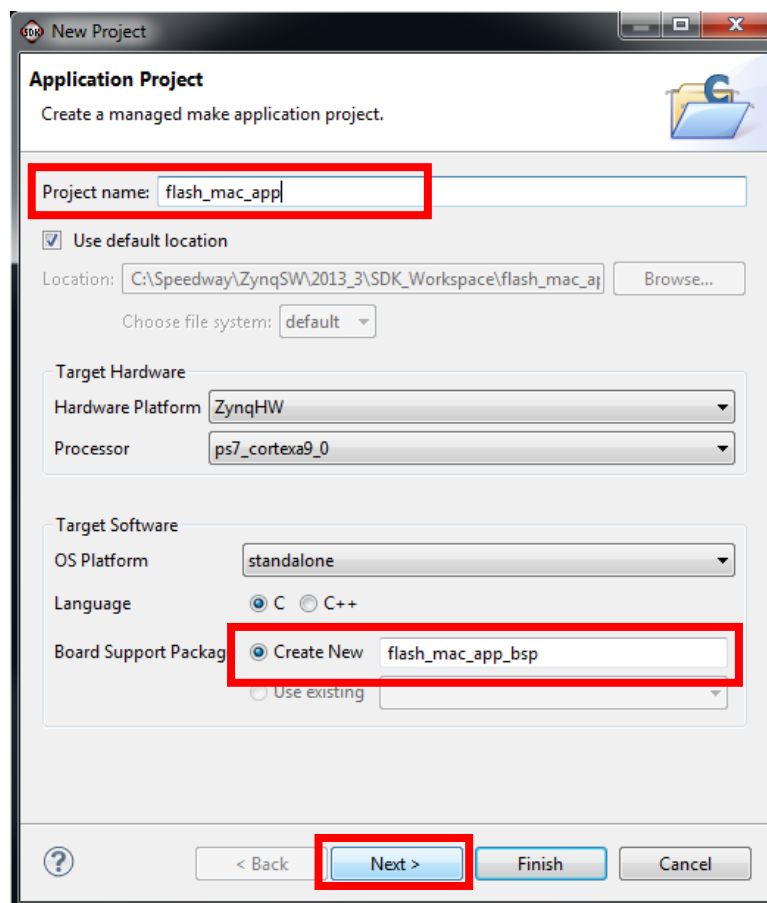


Figure 7 – Creating the flash_mac_app Application

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

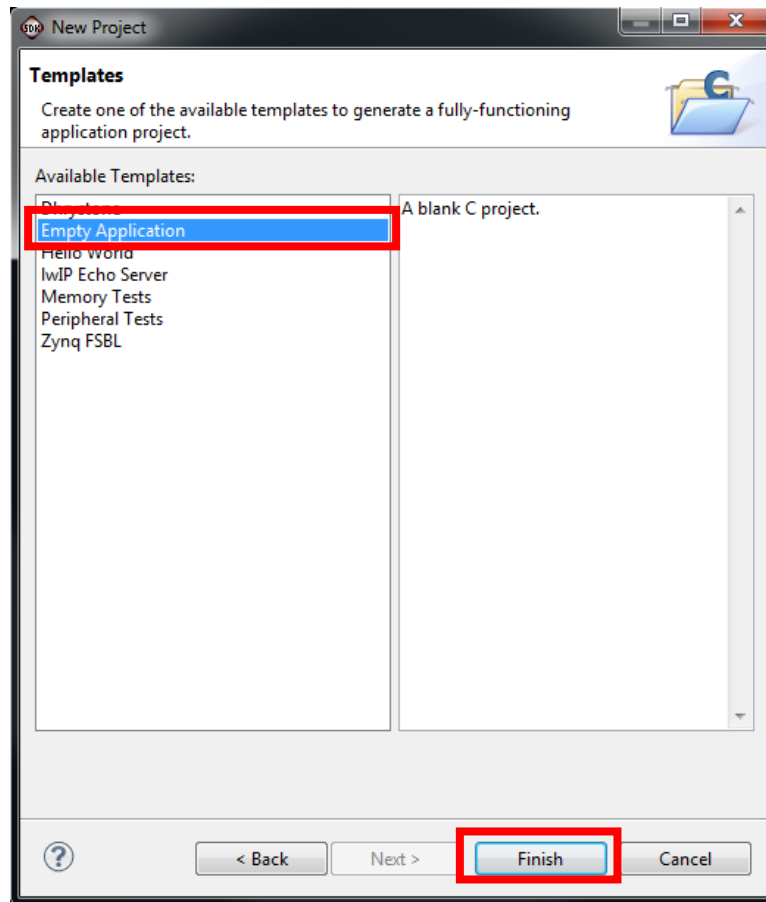


Figure 8 – Using the Empty Application Project Template

9. 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.

10. Click on the **Modify this BSP's Settings** button to configure **flash_mac_app_bsp** settings.

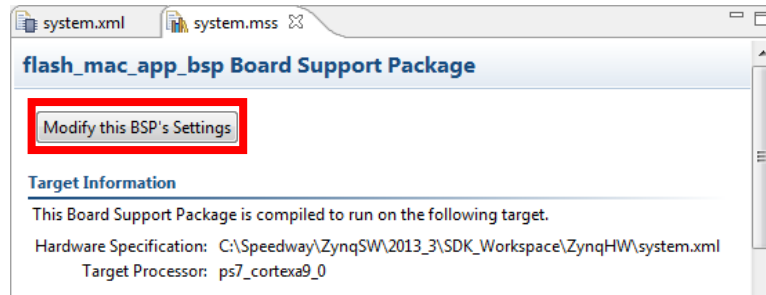


Figure 9 – Modify the BSP Settings for flash_mac_app_bsp Project

11. 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.

ZedBoard contains a Spansion S25FL256 QSPI device for which the latest built in **xilisf V3.02.a** library does not currently have support. A customized version of this library **xilisf V3.02.z** is provided in the SDK repository added during Lab 9 activities. This customized version of the library enables unofficial access to this currently unsupported serial Flash device. However, this access to the ZedBoard S25FL256 device is accomplished by treating it as a S25FL128 device so there is currently a limitation where only the first 128 Mbits of memory space can be accessed when using this custom library.

Click on the version drop down menu for the **xilisf** library and select the **3.02.z** option.

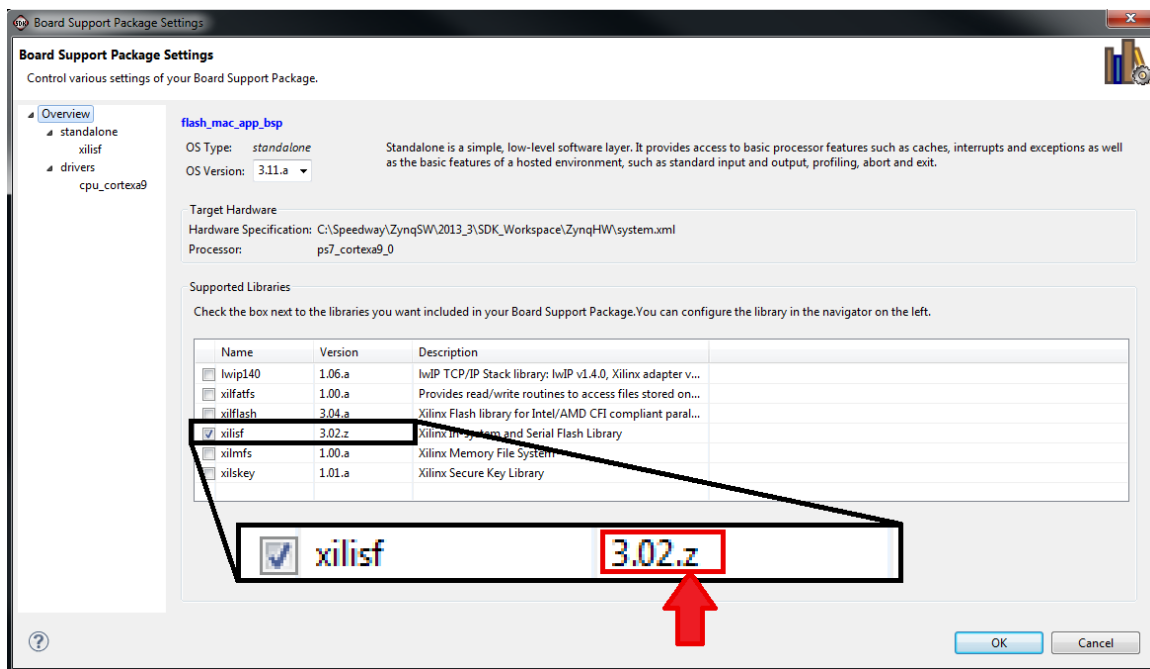


Figure 10 – Enable the Custom Library xilisf within the BSP Settings

IMPORTANT NOTE: Make sure that the 3.02.z version of xilisf is selected. This is the only version which contains support for ZedBoard Quad SPI Flash.

- 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 specify settings.

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

Since we are using a Spansion 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**.

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

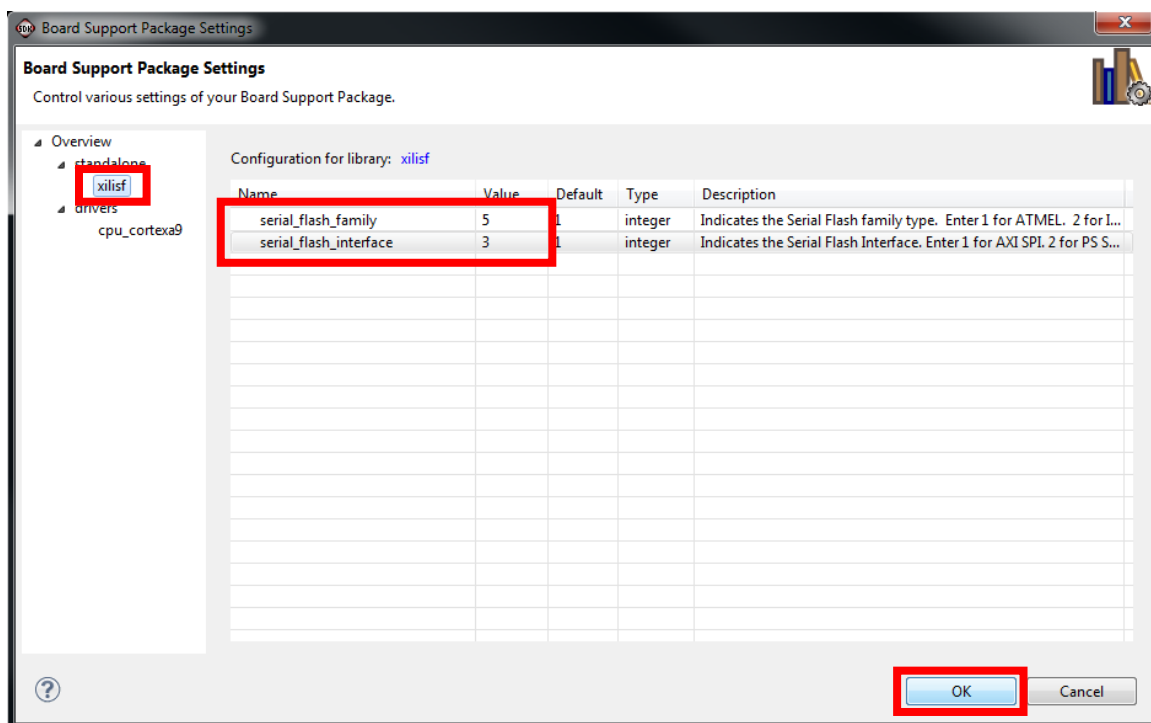


Figure 11 – Configure the Xilinx Library xilisf for Spansion QSPI Flash

13. 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.

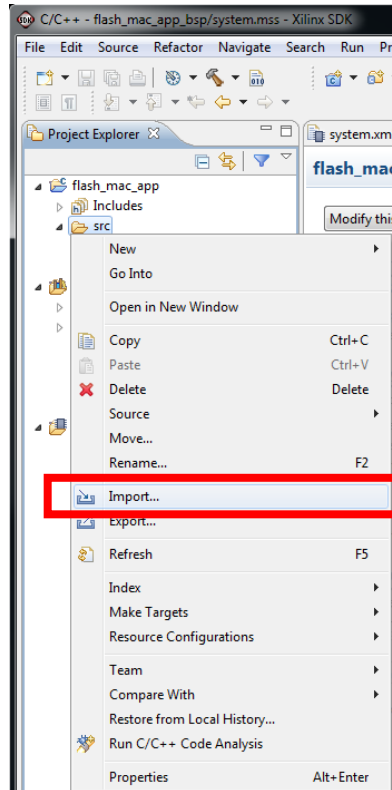


Figure 12 – Import flash_mac_app Application Source Code

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

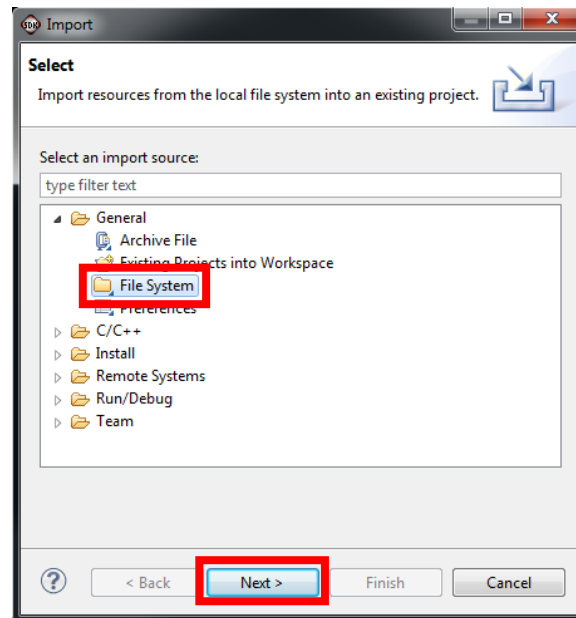


Figure 13 – Importing from a File System

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

C:\Speedway\ZynqSW\2013_3\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.

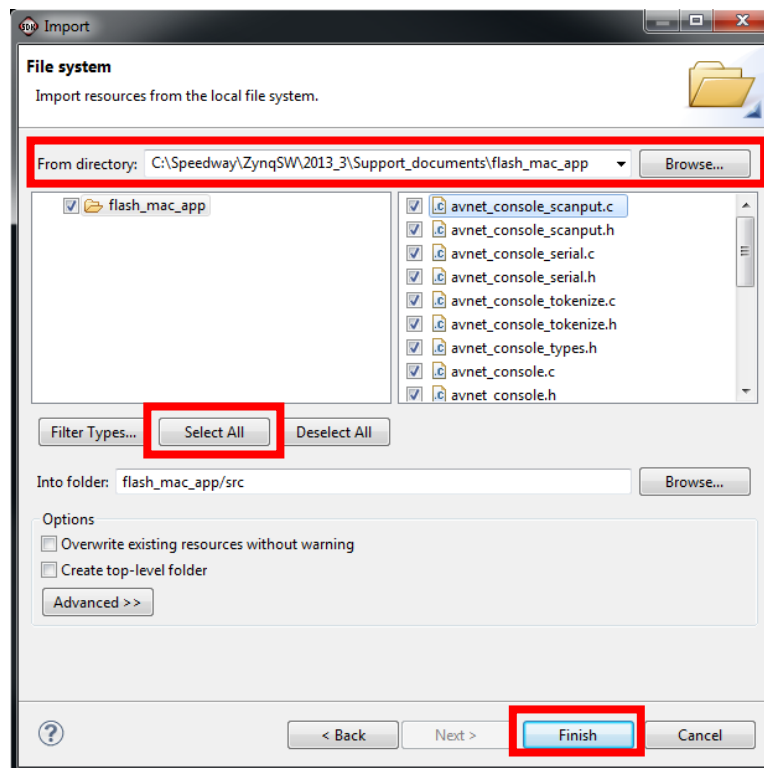


Figure 14 – Selecting flash_mac_app Application Source Files

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



```
CDT Build Console [flash_mac_app]

'Building target: flash_mac_app.elf'
'Invoking: ARM gcc linker'
arm-xilinx-eabi-gcc -Wl,-T -Wl,../src/ldscript.ld -L../flash_mac_app_bsp/ps7_cortexa9_0/lib -o
"flash_mac_app.elf" ./src/avnet_console.o ./src/avnet_console_scanput.o ./src/avnet_console_serial.o
./src/avnet_console_tokenize.o ./src/flash_mac.o ./src/platform.o ./src/xilisf_qspips_flash_polled.o
-Wl,--start-group,-lxil,-lgcc,-lc,--end-group -Wl,--start-group,-lxilisf,-lxil,-lgcc,-lc,--end-group
'Finished building target: flash_mac_app.elf'
'Invoking: ARM Print Size'
arm-xilinx-eabi-size flash_mac_app.elf |tee "flash_mac_app.elf.size"
      text    data    bss     dec     hex filename
 83412    2404    32108   117924   1cca4 flash_mac_app.elf
'Finished building: flash_mac_app.elf.size'

11:00:53 Build Finished (took 1s.472ms)
```

Figure 15 – Application Build Console Window

17. After SDK finishes compiling the new application code, the ELF is available in the following location:

C:\Speedway\ZynqSW\2013_3\SDK_Workspace\flash_mac_app\Debug\flash_mac_app.elf

Questions:

Answer the following questions:

- What if you wanted to reuse part of this application on your own Zynq design but you prefer to use Atmel Flash instead of Spansion 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 Spansion 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 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. Set the Boot Mode jumpers to Cascaded JTAG Mode, with JP7 through JP11 in the GND position.




Figure 16 – Cascaded JTAG Boot Mode

2. Attach another Micro-USB cable to the USB-UART port (J17) and make sure that the other end of this cable is connected to the development PC. For this example, the on-board USB-JTAG port will be used however the Digilent HS2 USB-JTAG cable or Xilinx Platform Cable USB II should work in a similar fashion through a connection to the PC4 header J15.



Figure 17 – Connecting On-board USB-JTAG Port

3. Turn power switch (SW8) to the ON position. ZedBoard will power on and the Green Power Good LED (LD13) should illuminate.
4. If necessary, launch or re-connect Tera Term.
5. In SDK, select **Xilinx Tools** → **Program FPGA** or click the  icon.

6. SDK will already know the correct .bit file (and .bmm if your future hardware platform includes that) since this was imported with the hardware platform. Click the **Program** button.

When D2 lights blue, the PL has configured successfully. Look for the message “FPGA configured successfully with bitstream” in the Console window.

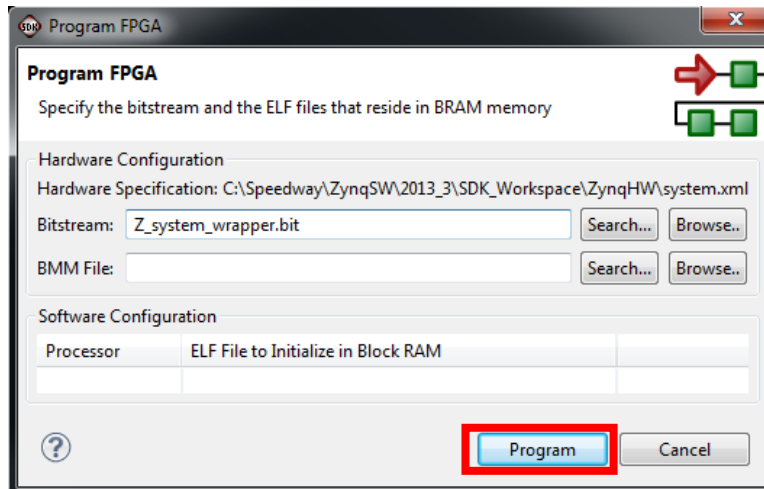


Figure 18 – Configure the Zynq Programmable Logic

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

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

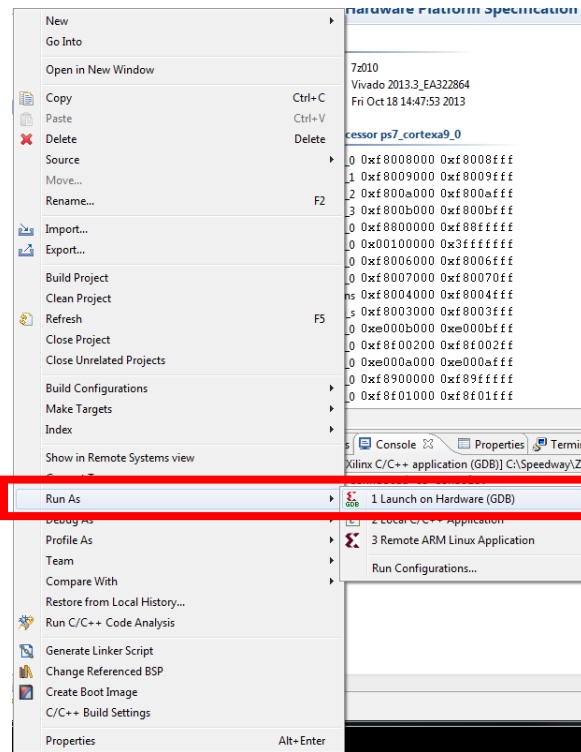


Figure 19 – Running flash_mac_app Application Code

- 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.

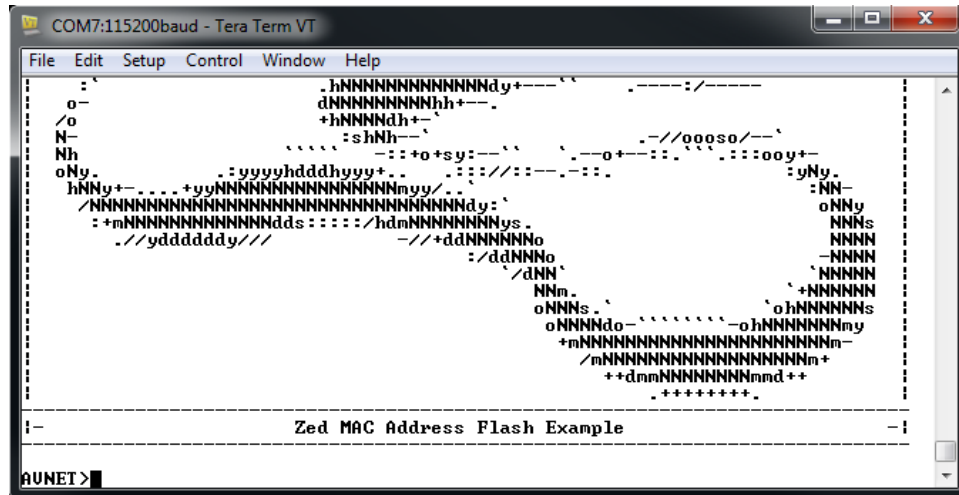


Figure 20 – Initial Terminal Output from flash_mac_app Application

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

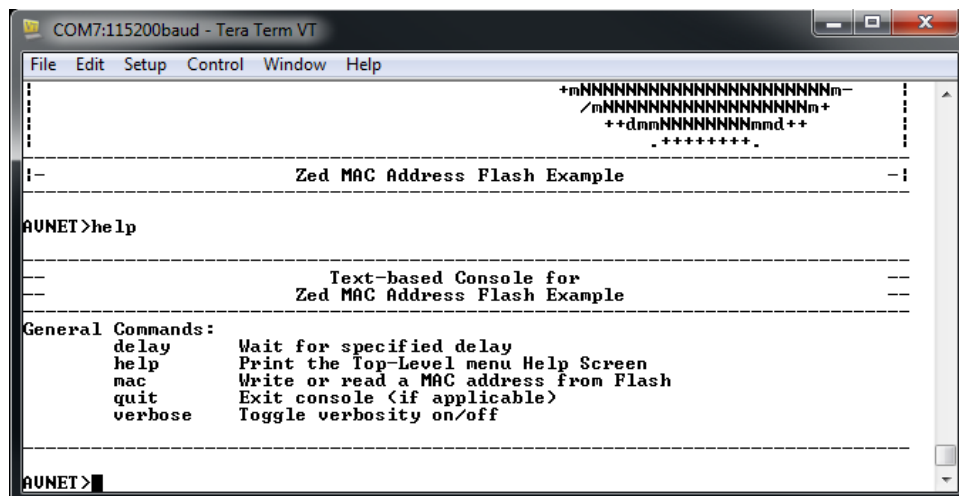
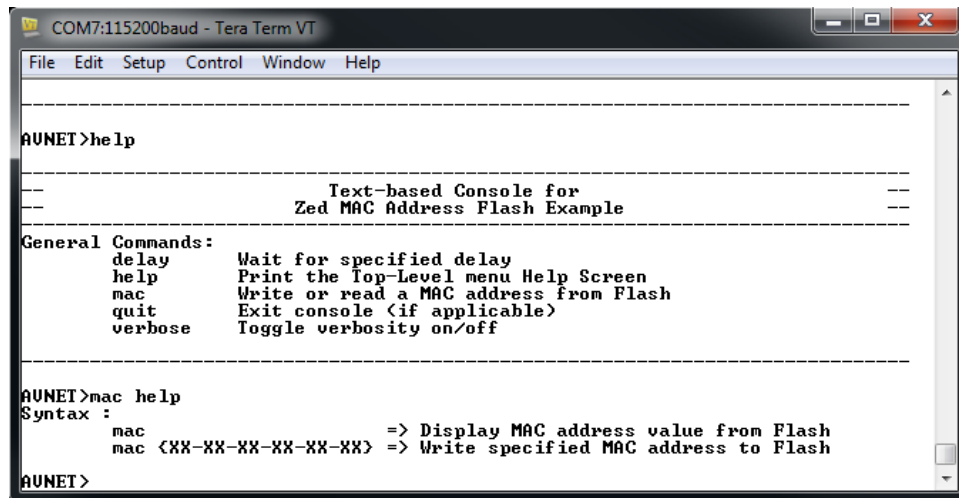


Figure 21 – Initial Terminal Output from flash_mac_app Application

10. 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.



```
COM7:115200baud - Tera Term VT
File Edit Setup Control Window Help

-----
Text-based Console for
Zed MAC Address Flash Example
-----

General Commands:
delay      Wait for specified delay
help       Print the Top-Level menu Help Screen
mac        Write or read a MAC address from Flash
quit       Exit console <if applicable>
verbose    Toggle verbosity on/off

AUNET>help

-----
Text-based Console for
Zed MAC Address Flash Example
-----

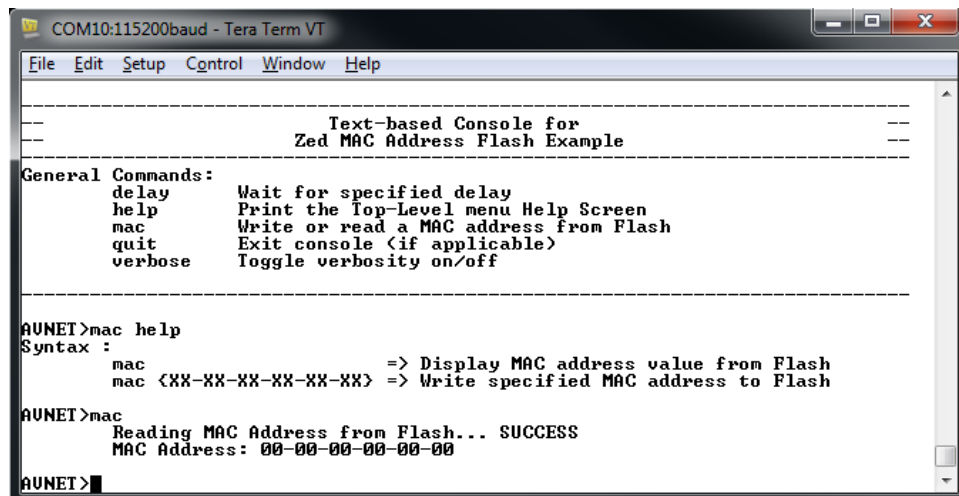
AUNET>mac help
Syntax :
mac              => Display MAC address value from Flash
mac <XX-XX-XX-XX-XX-XX> => Write specified MAC address to Flash

AUNET>
```

Figure 22 – Initial Terminal Output from flash_mac_app Application

11. 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.



```
COM10:115200baud - Tera Term VT
File Edit Setup Control Window Help

-----
Text-based Console for
Zed MAC Address Flash Example
-----

General Commands:
delay      Wait for specified delay
help       Print the Top-Level menu Help Screen
mac        Write or read a MAC address from Flash
quit       Exit console <if applicable>
verbose    Toggle verbosity on/off

AUNET>mac help
Syntax :
mac              => Display MAC address value from Flash
mac <XX-XX-XX-XX-XX-XX> => Write specified MAC address to Flash

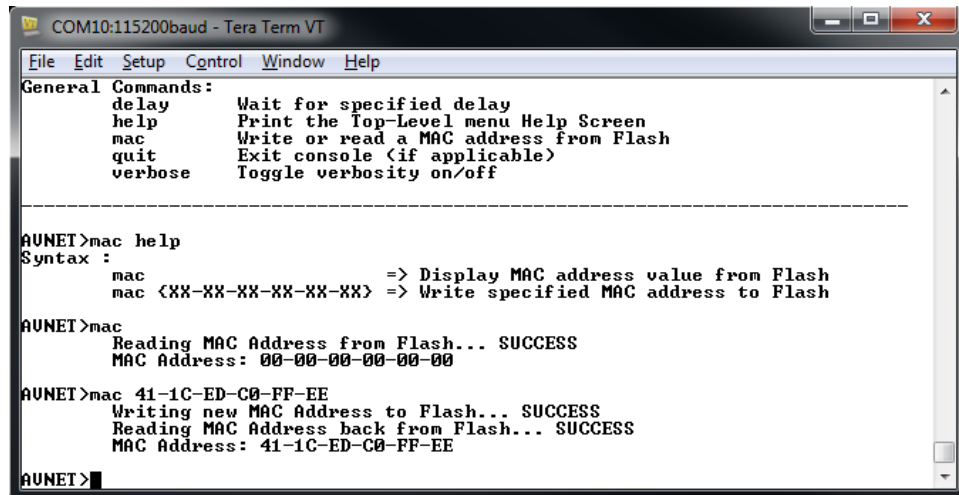
AUNET>mac
Reading MAC Address from Flash... SUCCESS
MAC Address: 00-00-00-00-00-00

AUNET>
```

Figure 23 – Displaying Initial Flash Data in the MAC Address Space

12. 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 41-1C-ED-C0-FF-EE
```



```
COM10:115200baud - Tera Term VT
File Edit Setup Control Window Help
General Commands:
delay      Wait for specified delay
help       Print the Top-Level menu Help Screen
mac        Write or read a MAC address from Flash
quit       Exit console <if applicable>
verbose    Toggle verbosity on/off

-----

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>mac
Reading MAC Address from Flash... SUCCESS
MAC Address: 00-00-00-00-00-00

AVNET>mac 41-1C-ED-C0-FF-EE
Writing new MAC Address to Flash... SUCCESS
Reading MAC Address back from Flash... SUCCESS
MAC Address: 41-1C-ED-C0-FF-EE

AVNET>
```

Figure 24 – Programming a New MAC Address Value into Flash Memory

13. 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.

14. 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.

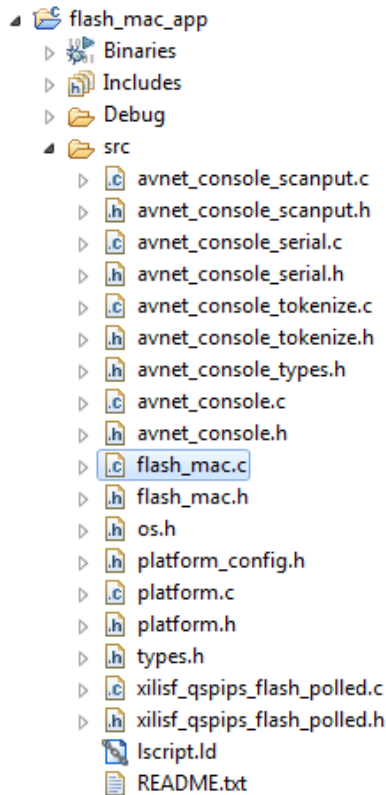


Figure 25 – flash_mac_app Source Code

15. In the source file, line 122 shows a call to **FlashInit()** which in turn makes a call to **QsippiFlashPolledInit()** which is code leveraged from this Xilinx example file:

C:\Xilinx\SDK\2013.3\lib\sw_services\xilisf_v3_02_a\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.

16. 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 **xilidf** 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 **xilidf** 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 **xilidf** library **Xisf_Write()** API call.

Questions:

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:

C:\Xilinx\SDK\2013.3\lib\sw_services

This concludes Lab 11.

Revision History

| Date | Version | Revision |
|-----------|---------|--------------------------------------|
| 12 Nov 13 | 01 | Initial Release |
| 25 Nov 13 | 02 | Revisions after pilot |
| 01 May 14 | 03 | ZedBoard.org Training Course Release |
| | | |

Answers

Experiment 1

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

Avnet also distributes 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 application. We could have instead modified the **standalone_bsp_0** to add the **xilisf** library but that would cause the library code to become linked into our other standalone applications which rely upon **standalone_bsp_0** project as the BSP. This would unnecessarily increase the code size of those other standalone applications which might not be desirable.

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 **C:\Xilinx\SDK\2013.3\lib\sw_services\xilisf_v3_02_a\examples** folder.