Developing Zynq Software with Xilinx SDK Lab 3

Bare Metal Board Support Package

Jan 2018
Version 11

AVNET
Reach Further™

Lab 3 Overview

One of the many time-saving features that SDK offers is the automatic generation of a board support package (BSP) for application development. This topic describes the bare metal BSP used within SDK.

A board support package (BSP) is a collection of libraries and drivers that will form the lowest layer of your application software stack. Your software applications must link against or run on top of a given software platform using the application programming interfaces that it provides. Therefore, before you can create and use software applications in SDK, you must create a board support package.

SDK includes the following board support package for application development:

- **Standalone**

  A simple, semi-hosted, and single-threaded environment that provides basic features such as standard input/output and access to processor hardware features.
  Semi-hosting is a mechanism for ARM targets to communicate I/O requests from application code to a host computer running a debugger. This mechanism could be used to enable functions in the C library, such as printf, or to use the screen and keyboard from the host to communicate with the target.
  "Standalone" is synonymous with "Bare-Metal." When a board support package is built, the following software components are automatically included into the library:

- Device Drivers for all the PS7 and Xilinx PL peripherals in your design (libxil.a)

Other libraries

The Standalone BSP can be built with libraries from the Xilinx software library collection, including:

- Libmetal – Libmetal Library
- Lwip141 -- An open source light-weight TCP/IP networking library
- openamp – OpenAmp Library
- xilffs – Generic Fat File System library
- xilflash – A library that provides read/write/erase/lock/unlock and device specific functionalities for parallel flash devices
- xilisf – In-System-Flash library that supports the Xilinx In System Flash hardware
- xilmfs – A Memory File system
- xilpm – Power Management API Library for Zynq
- xilrsa – Xilinx RSA library
- xilskey – Secure Key library

You can configure these libraries to be in or out of your software application compilation.

Lab 3 Objectives

When you have completed Lab 3, you will know:

- How to generate the Standalone BSP
- Where to find device driver datasheets
- Where to find example code

Experiment 1: Generate the Standalone Board Support Package

In Lab 2, a Zynq hardware platform was imported. In this experiment, a bare-metal BSP will be generated. This BSP will assemble drivers for all the IP associated with the previously imported hardware platform.

Experiment 1 General Instruction:

Generate the Standalone BSP. Do not include any extra libraries or special settings.

Experiment 1 Step-by-Step Instructions:

1. If you had previously closed SDK, for instructions onto how to reopen SDK please refer to lab 2 experiement 1.
2. Select **File → New → Board Support Package**.
3. Accept the default settings for the standalone BSP OS. Click **Finish**.The BSP OS indicated what operating system is going to be ran, this can range from freertos, standalone, or a petalinux environment.

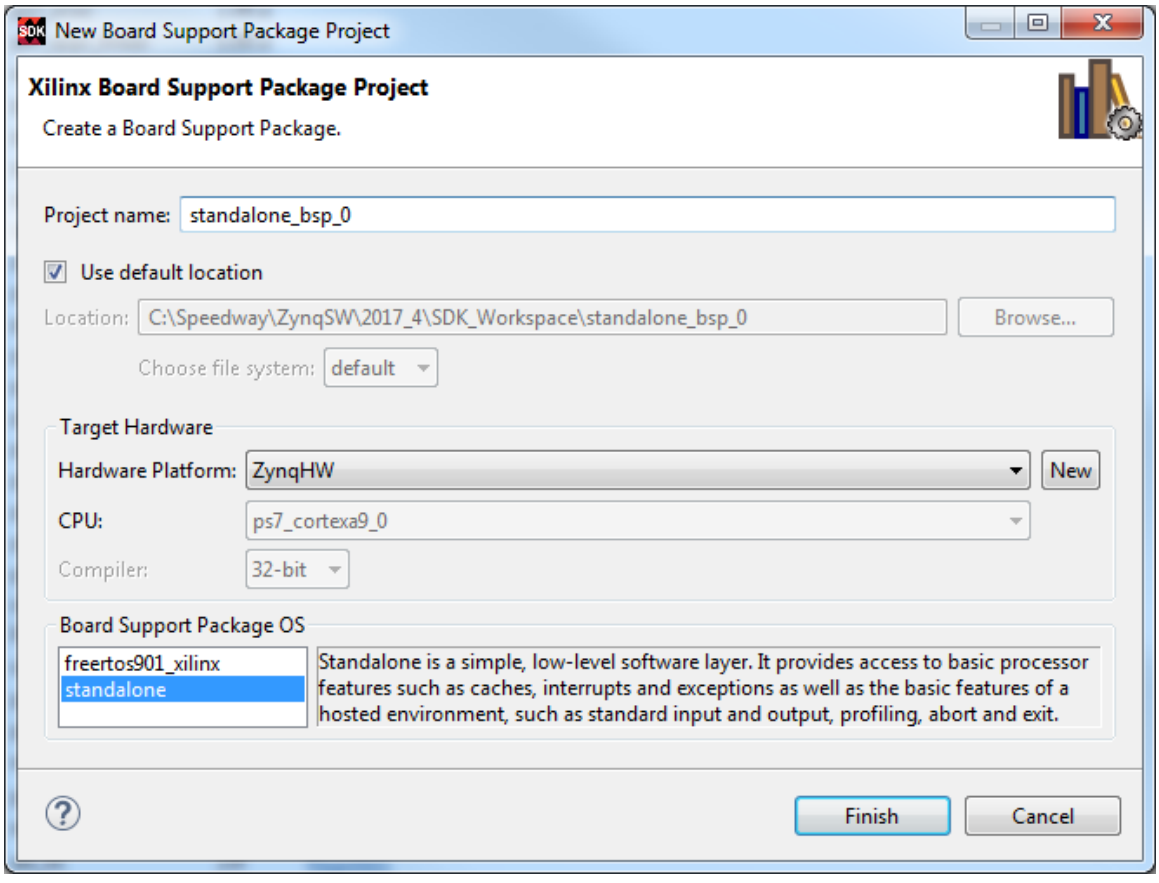

Figure 1 – Standalone BSP

4. None of the Supported Libraries are needed for this experiment. We will examine a library in a later lab.

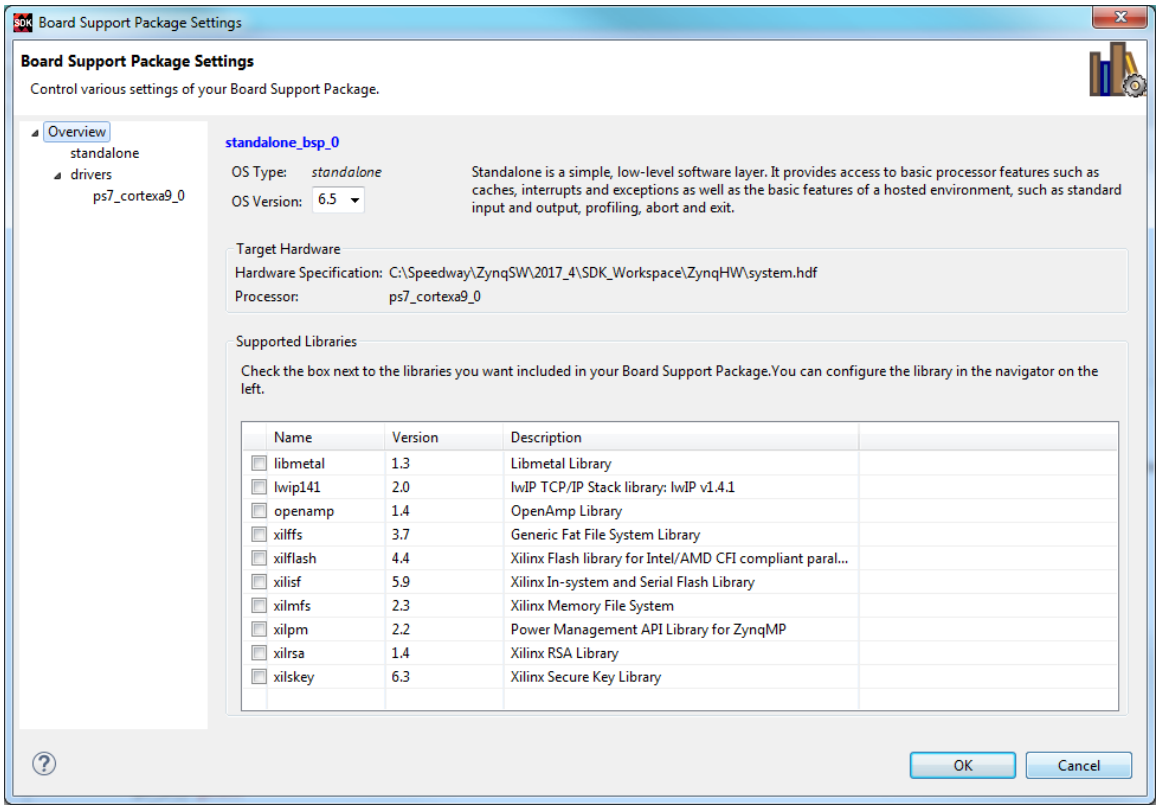

Figure 2 – BSP Settings

5. Click on **standalone**. Notice that the **stdin** and **stdout** are set to the ps7_uart_0 peripheral. This is not the correct pinout for the UART setup on the MiniZed. Please refer to the schematic.

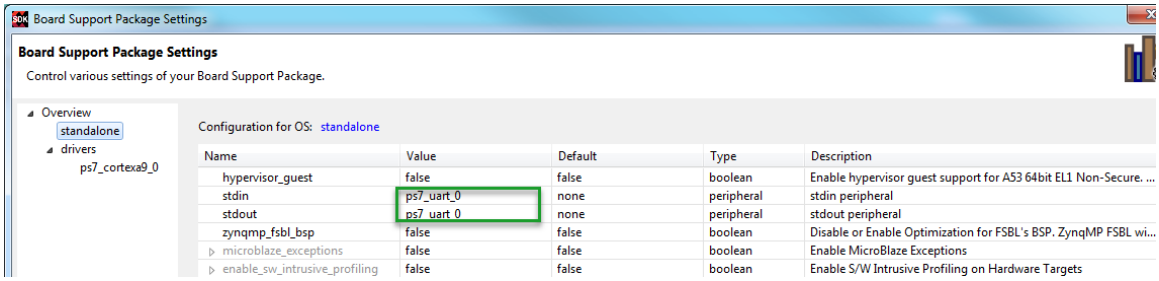

Figure 3 – Incorrect BSP Settings for stdin and stdout

6. Change the Value to ps7_uart_1 to align with the MiniZeds UART peripheral.

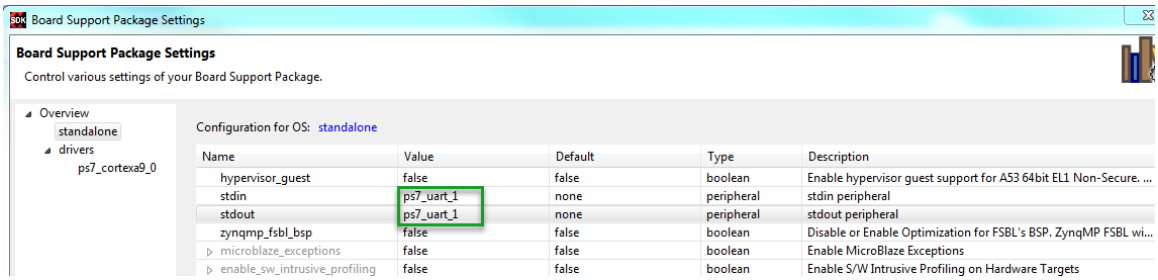

Figure 4 -- Correct BSP Settings for stdin and stdout

7. Click on **drivers**. This shows all of the drivers that the BSP Generator will pull into the project.

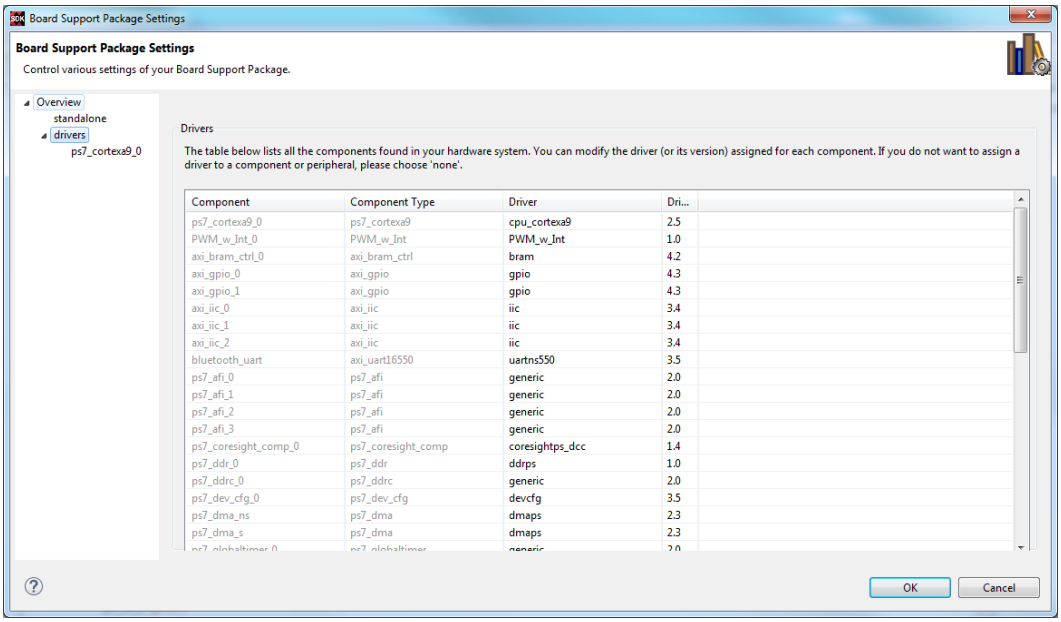

Figure 5 – BSP drivers

8. Click on **ps7_cortexa9_0**. This shows the default compiler, archiver, and flags. If you wanted to debug the BSP, this is where you would turn off optimization and add debug symbols. Click **OK** to accept the defaults and close this dialog.

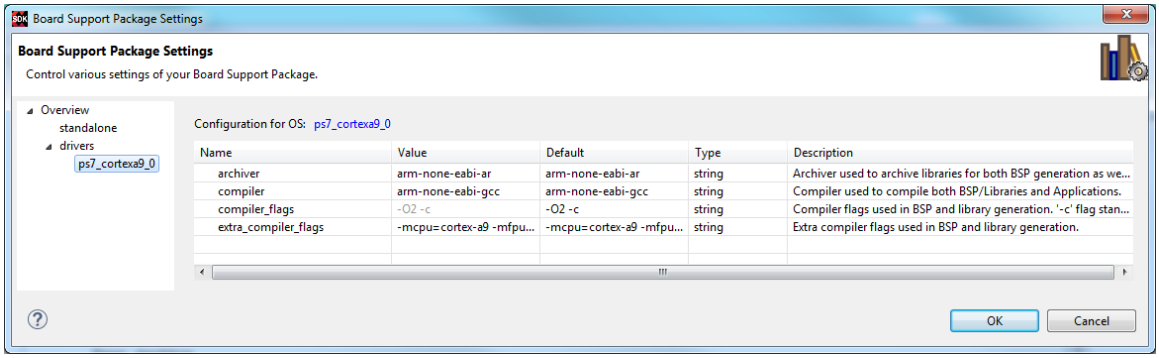

Figure 6 – Compiler and Flags

The standalone_bsp_0 is now visible in the Project Explorer. Based on the default settings in SDK, the BSP will automatically be built once added to the project.

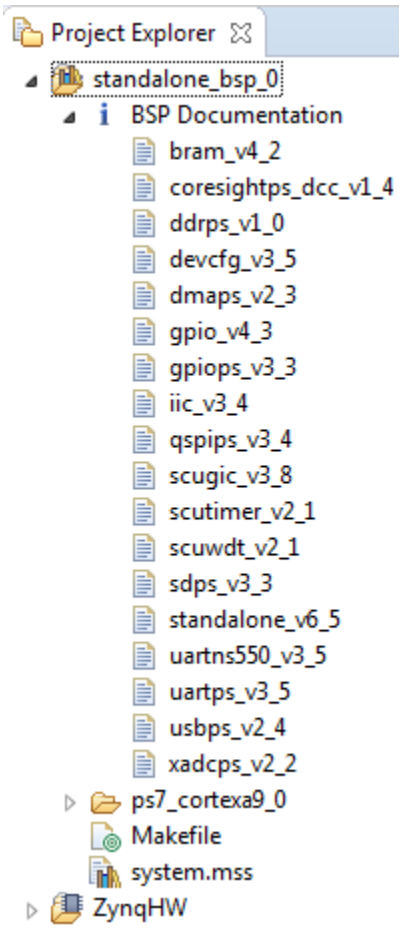

Figure 7 – BSP Added to the Project

Experiment 2: Explore the BSP

Now that the BSP is generated, we will explore what has been created.

Experiment 2 General Instruction:

Explore the BSP. Examine the Cortex A9 API specific to the Standalone OS. Review documentation for a device driver. Open the example code provided for one of the device drivers.

Experiment 2 Step-by-Step Instructions:

Notice that a BSP report is opened in the primary viewing window. This BSP report can also be opened by clicking on **system.mss** under the BSP folder in *Project Explorer*.

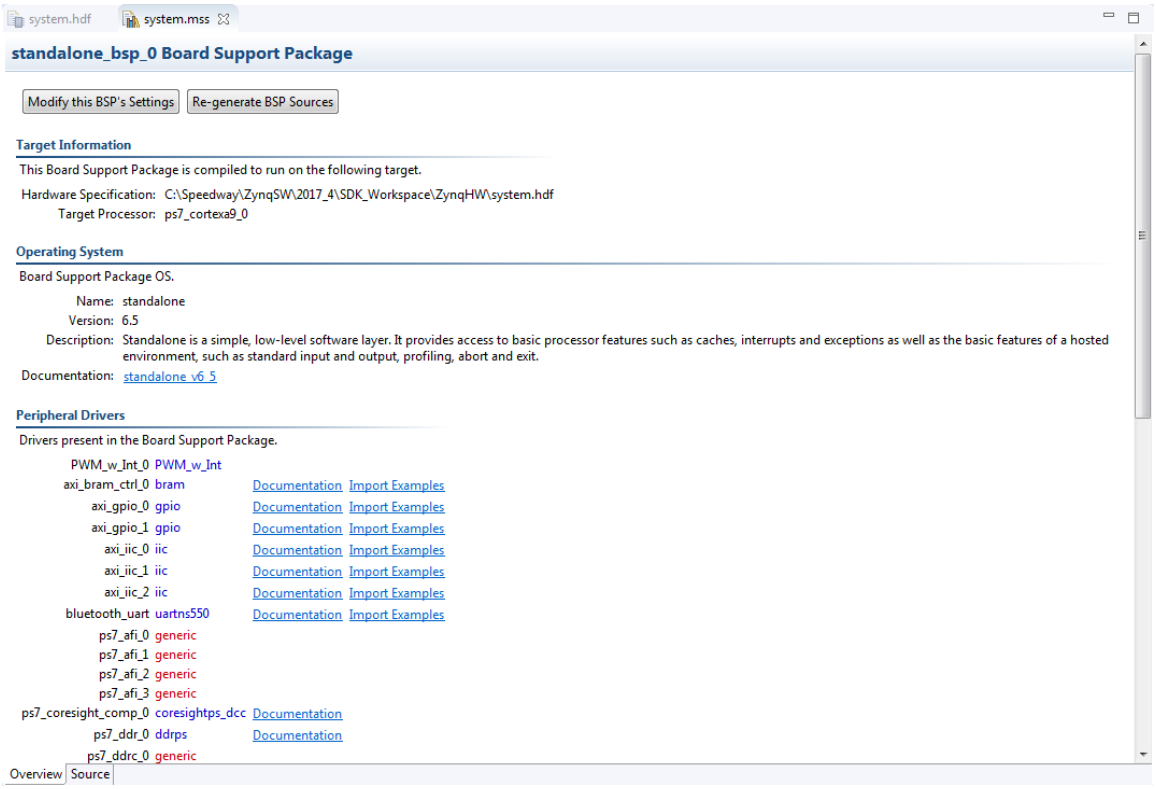

Figure 8 – BSP Report

You will see generic(red) and specific driver references(blue) under the Peripheral Drivers section. Generic means that in the standalone environment there is no defined driver present. While the blue more specific annontation is an actual reference to a driver in the standalone enviroment.

1. Under the *Operating System* section, click on the hyperlink for **standalone_v6_5** to open the PDF documentation for Standalone. Browse to the Cortex A9 Processor API chapter. This document is UG647 and simply click on the "Cortex A9 Processor API" as shown in Figure 8 below. The Zynq ARM Cortex A9 specific APIs are shown here. Close the PDF when finished.

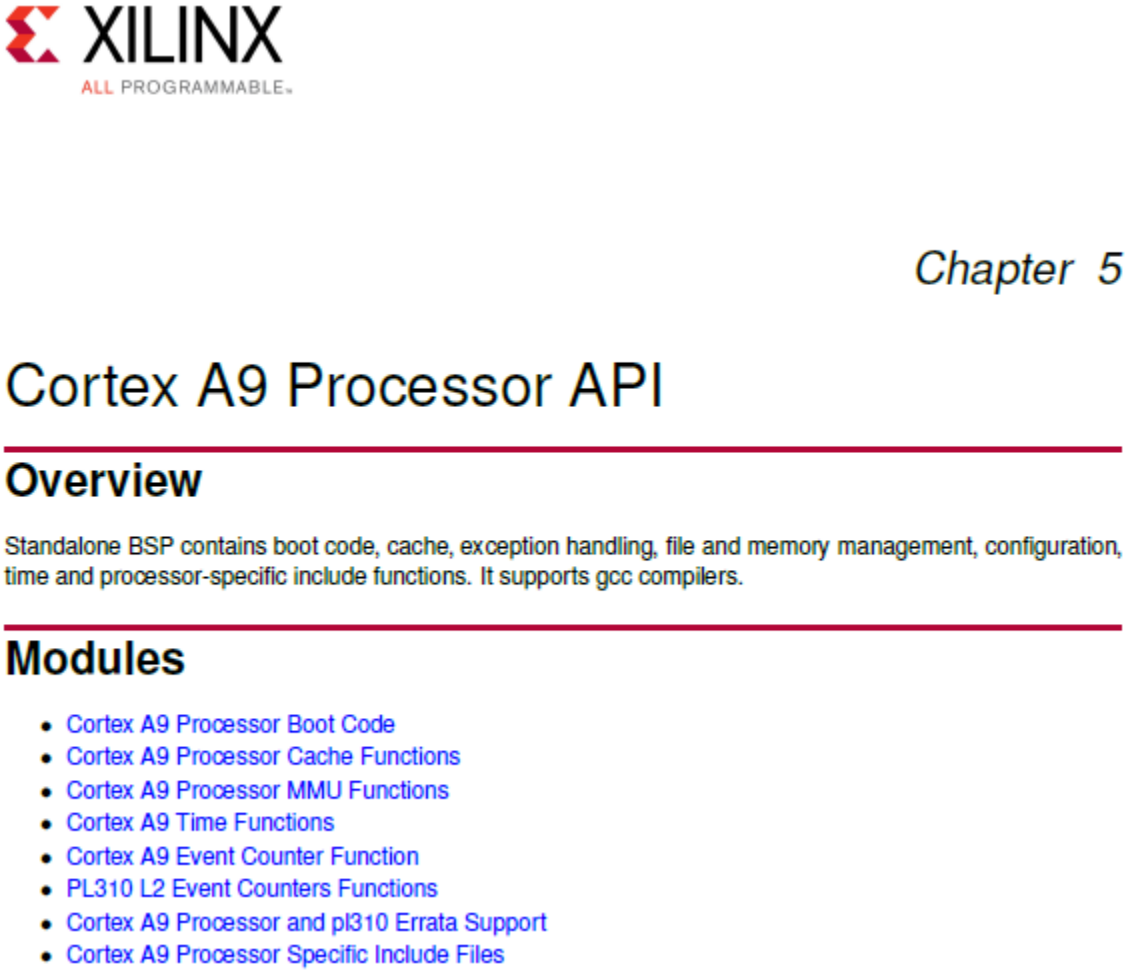

Figure 9 – UG647 doc

2. Under the *Peripheral Drivers* section shown in the system.mss report, each hardware peripheral is listed along with the driver associated with that peripheral. Click on one of the **Documentation** links to view the datasheet for the driver. For example, open the gpiops documentation found next to the **ps7_gpio_0** peripheral.

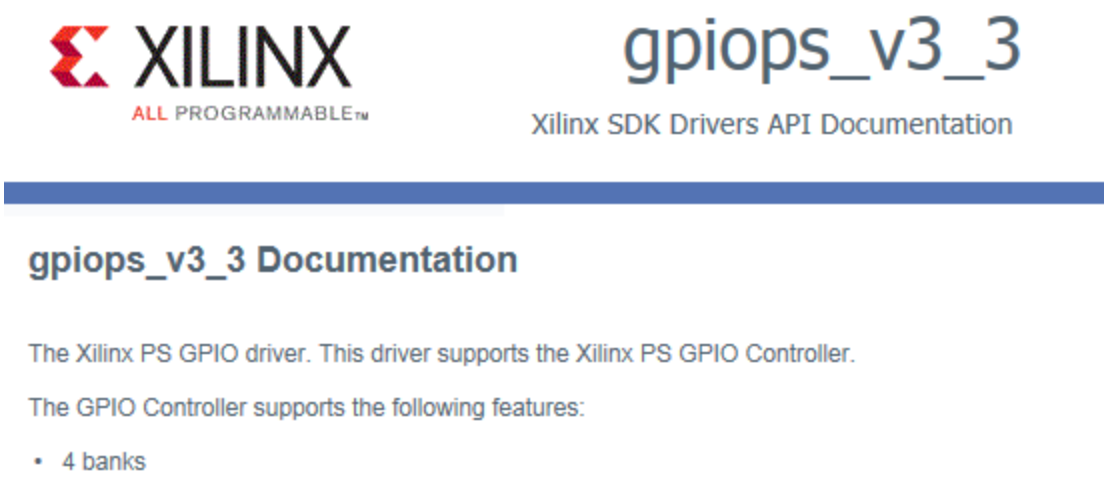

Figure 10 – gpiops Device Driver Documentation

3. Click on the links for **Data Structures (Classes)** and **Files** to explore the device driver. Close the documentation.

4. Click on the **Import Examples** hyperlink next to gpiops. Notice that this opens an SDK window showing examples, in this case a polled and an interrupt based example.

5. Click the checkbox for the **xgpiops_polled_example**. Then click OK to import the polled example into the workspace.

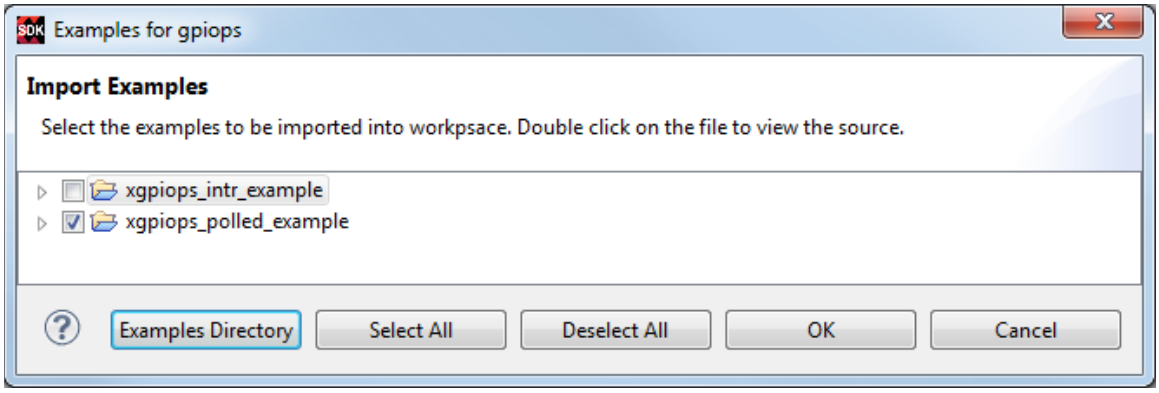

Figure 11 – Import the GPIO Polled Example

6. Expand the polled example **src** folder and double-click the xgpiops_polled_example.c to review the code.

7. Briefly examine the code and correlate to the documentation previously reviewed. (review the xgpiops v3_3 document in step 3, specifically the files link). Close the code.

8. Right-click on the imported application standalone_bsp_0_xgpiops_polled_example_1. Select **Delete**, and the check box for **Delete project contents on disk**. Click **OK**.

Experiment 3: Library Generator (LibGen)

The underlying Xilinx tool that assembled the BSP was Library Generator, or simply LibGen. You've already seen some of what this tool accomplished by browsing the system.mss report. In this section, we will explore more of the Board Support Package created by Library Generator.

Detailed information for LibGen is available in UG1043 Embedded System Tools Reference Manual

Experiment 3 General Instruction:

View what LibGen created. Examine the contents of each folder created in the BSP. Investigate the contents of xparameters.h.

Experiment 3 Step-by-Step Instructions:

Notice that a BSP report is opened in the primary viewing window. This BSP report can also be opened by clicking on **system.mss** under the BSP.

Folder Structure

1. In SDK's Project Explorer, expand **standalone_bsp_0** to examine the contents.

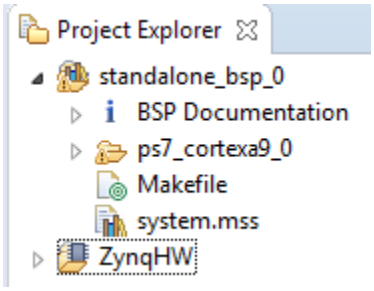

Figure 12 – BSP Contents

2. You've already examined system.mss. You can open the Makefile for viewing if you would like.

3. Expand **BSP Documentation**. This provides individual links for each peripheral driver, which we have already seen in the system.mss report. This is just another way to find the same documentation.

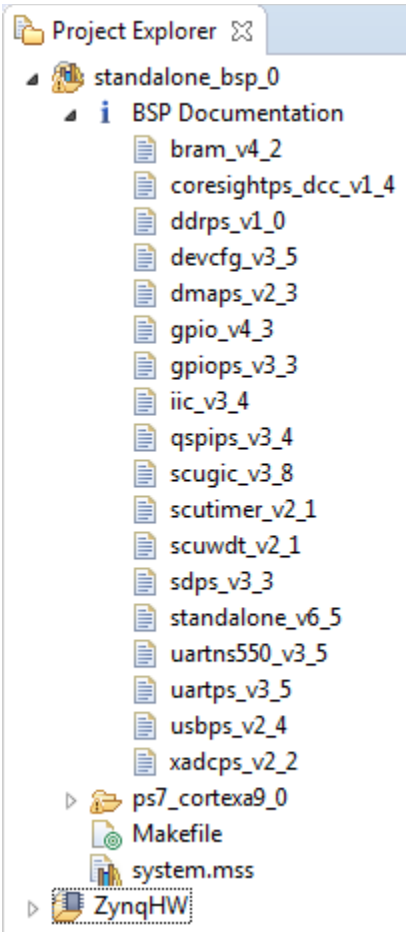

Figure 13 – BSP Documentation

4. Expand **ps7_cortexa9_0**. You'll notice that there are four sub-folders, each of which will be examined in the next few sections.

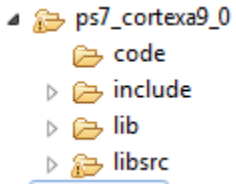

Figure 14 – Four ps7_cortexa9_0 Sub-folders

5. Open Windows Explorer and browse to the SDK_Workspace directory at `C:\Speedway\ZynqSW\2017_4\SDK_Workspace\`. Notice that SDK's *Project Explorer* is a copy of what is actually on disk.

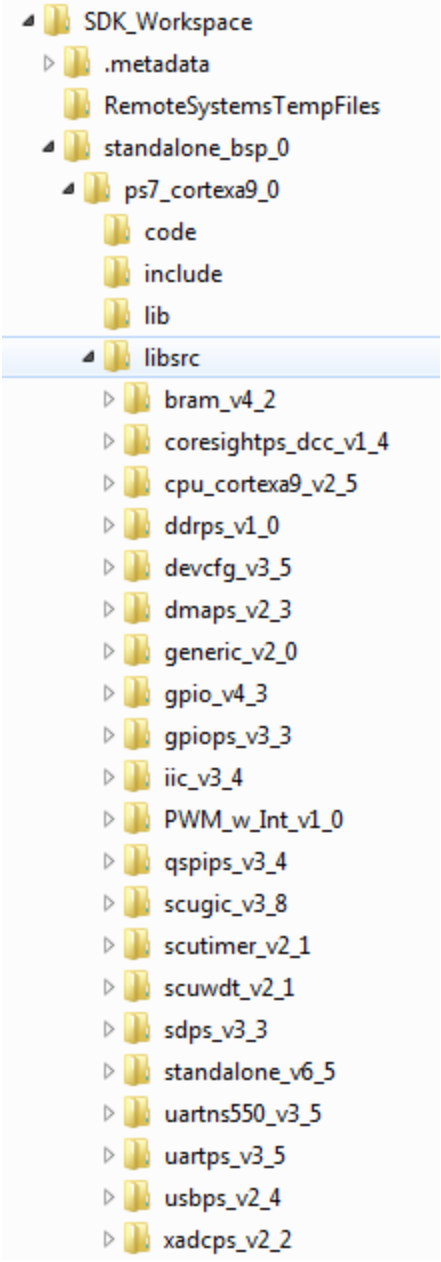

Figure 15 – Folder Structure on Disk

6. Take note of the `.metadata` folder. The presence of this folder indicates that its parent directory was used as an SDK_Workspace.

7. One thing worth taking a look at is the MDD (Microprocessor Driver Definition) files associated with the various drivers you have. In a windows explorer window go to

C:\Xilinx\SDK\2017.4\data\embeddedsw\XilinxProcessorIPLib\drivers\gpiops_v3_3\data

Open gpiops.mdd in a text editor.

8. Near the bottom of this file are various options that define the driver within your SDK project. Below is what each option I refers too

OPTION psf_version = 2.1; - This is where the psf_version of the MDD file is defined
OPTION supported_peripherals = (ps7_gpio psu_gpio); -
OPTION driver_state = ACTIVE; - There a total of 3 possible states, Active(this is the active driver), Depreciated(This driver is scheduled to be removed), and Obsolete(This driver is obsolete and is not recognized by any tool)
OPTION copyfiles = all; - Indicates the files to be copied for the library
OPTION VERSION = 3.3; - Option Version indicates the version of the driver
OPTION NAME = gpiops; - Option Name indicates the name of the driver

9. Close the MDD file and return to SDK.

code Folder

10. The code directory is a repository for SDK executables. For Zynq designs, there are no executables, which is why this folder is empty.

include Folder

11. The `include` directory contains C header files needed by drivers. The include file **`xparameters.h`** is also created through Libgen in this directory. This file defines base addresses of the peripherals in the system, #defines needed by drivers, OSs, libraries and user programs, as well as function prototypes. You will utilize this file in every application you develop.

12. Expand the **include** folder. Open the xparameters.h file. Examine this file for a moment then use it to answer the following questions.

Questions:

Answer the following questions:

- *What are the base address for the on-chip RAMs, XPAR_PS7_RAM_0_S_AXI_BASEADDR and XPAR_PS7_RAM_1_S_AXI_BASEADDR?*

  0x00000000 0xFFFC0000

- *What size is PS7_RAM_0 ?*

  256KBytes

- *What parameter name would you access for the PWM peripheral interrupt?*

  XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR

lib Folder

13. Expand the `lib` folder. The `lib` directory contains the libxil.a library. The `libxil` library contains driver functions that the particular processor can access. For more information about the libraries, refer to the introductory section of the *OS and Libraries Document Collection (UG643)*.

libxil.a – Contains Xilinx-developed drivers, software services (such as XilMFS) and initialization files

libsrc Folder

14. Expand the libsrc folder. The libsrc directory contains intermediate files and make files needed to compile the OSs, libraries, and drivers. The directory contains peripheral-specific driver files, BSP files for the OS, and library files that are copied from the SDK and your driver, OS, and library directories. Typically, you will make most use of the .h files for the peripherals in the include folder. However, if the .h does not provide enough information, it is useful to examine the peripheral driver's source code provided in this folder.

One very important note is that all of these directories are removed and recreated every time a change is made to the BSP and LibGen runs. *DO NOT* edit these files, as your changes will be eliminated during the next clean or re-build operation.

Exploring Further

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

- Use what you have learned in these experiments to determine what the default UART baud rate is and how it gets set.

This concludes Lab 3.

Revision History

Date	Version	Revision
12 Nov 13	01	Initial release
23 Nov 13	02	Revisions after pilot
01 May 14	03	MicroZed.org Training Course Release
29 Oct 14	04	Revision for 2014.3
11 Dec 14	05	Revision for 2014.4
05 Jan 15	06	Minor typo correction
18 Mar 15	07	Finalize SDK 2014.4
Oct 15	08	Updated to 2015.2
Aug 16	09	Updated to 2016.2
Jun 17	10	Updated to 2017.1 for MiniZed + Rebranding
Jan 18	11	Updated to Vivado/SDK 2017.4

Resources

www.minized.org

www.microzed.org

www.minized.org

www.picozed.org

www.zedboard.org

www.xilinx.com/zynq

www.xilinx.com/sdk

www.xilinx.com/vivado

www.xilinx.com/support/documentation/sw_manuals/ug949-vivado-design-methodology.pdf

www.xilinx.com/support/documentation/sw_manuals/ug1046-ultrafast-design-methodology-guide.pdf

Answers

Experiment 3

- *What are the base address for the on-chip RAMs, XPAR_PS7_RAM_0_S_AXI_BASEADDR and XPAR_PS7_RAM_1_S_AXI_BASEADDR?*

0x00000000 and 0xFFFC0000

- *What size is PS7_RAM_0 ?*

0x0003FFFF – 0x00000000 + 1 = 0x40000 = 262144 bytes = 256KB

- *What parameter name would you access for the PWM peripheral interrupt?*

XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR, which happens to be set to interrupt #61