XAPP1198 (v1.1) November 19, 2014

In-System Eye Scan of a PCI Express Link with Vivado IP Integrator and AXI4

Author: Luis Bielich

Summary

This application note describes a flow for integrating in-system Eye Scan into an AXI4-based system with Vivado® IP integrator. When an AXI4-based processor is planned to be used in a design, it is simple to include Eye Scan functionality to a design by memory mapping the DRP interface of the transceiver to the AXI4 system. A reference design is provided to show how this is implemented with IP integrator. The Eye Scan algorithm from *Eye Scan with MicroBlaze MCS* [Ref 1] is leveraged in this application note to implement the Eye Scan processing. *Eye Scan with MicroBlaze MCS* [Ref 1] is a great application note for standalone Eye Scan, whereas this application note focuses on integration into an AXI4-based system.

The reference design implements the Eye Scan functionality on a PCI Express® link on the following reference boards: AC701, KC705, VC709, ZC706, and KCU105. Starting in 2013.3, Xilinx IP containing transceivers output the DRP interface to the top level of the IP. The integrated block for PCI Express IP is one of the IP cores including this major ease-of-use enhancement. This option enables the AXI4 system to memory map the DRP space of the transceivers without having to dive deep into the PCIe® IP RTL. This ultimately maintains a clean IP flow enabling all of the plug-and-play benefits of the Vivado Integrated Design Environment (IDE).

The Eye Scan results provide important information for validating the design of each channel across voltage, temperature and process. Along with the ability to validate the design of a channel, Eye Scan also provides a very high level of debug information not previously available in FPGA families.

Introduction

All 7 series transceivers include the functionality to run an in-system 2-D statistical Eye Scan on the receiver without requiring a predetermined pattern. This is implemented with a second sampler parallel to the primary data sampler shown in Figure 1. Running an Eye Scan does not affect the link integrity, while maintaining a representation of margin in a running system. The

Eye Scan results also incorporate the function by the transceiver equalizer showing the most accurate analysis of the margin at the receiver.

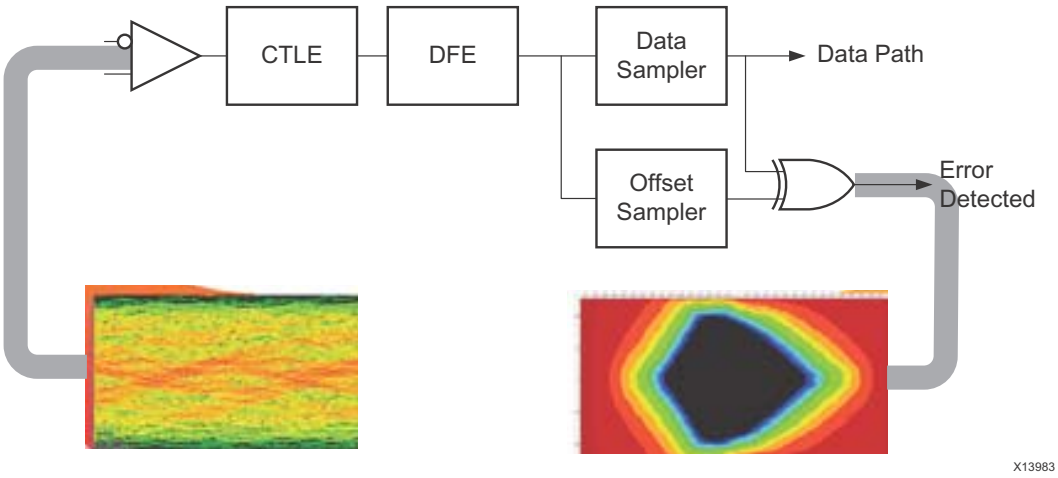

Figure 1: **Offset Sample Image**

This application note describes how to perform an Eye Scan on a PCI Express link. Although this focuses on a PCI Express link, the reference design files can be leveraged for any link at any rate.

PCI Express provides an ideal protocol to use in-system Eye Scan because it is uncommon to place a PCI Express link in a loopback state for debug purposes. Because placing a link in a loopback is not common in a PCI Express link, this means a predetermined pattern is generally not available. Another reason why in-system Eye Scan is valuable for PCI Express is because a PCI Express link can stay in the L0 state even when there are bit errors on the link. This PCI Express protocol handles these bit errors with a replay mechanism including CRC checks. Although the protocol maintains data integrity, the replay mechanism affects throughput; making it important to address a marginal link. In a multi-lane implementation of PCI Express, it can be difficult to identify which lane is marginal. Determining the marginal lane can be quickly detected with an in-system Eye Scan.

In-system Eye Scan also takes into account the real system dependencies when all portions of the design are running. Because the Eye Scan logic does not require a predetermined pattern, it does not require a test mode which can change other conditions. In other words, the Eye Scan data can be extracted during normal operation which could cover corner-case conditions only enabled in a real system. For PCI Express, the power consumption and switching noise differs greatly between an idle mode and a DMA mode. The results from in-system Eye Scan include these dependencies and allow for margin analysis for all modes.

When to Use Eye Scan with PCI Express

PCI Express has several use cases for the Eye Scan feature, and are described in this section.

Debug Link Training Issues

Link training issues can occur due to signal integrity. Eye Scan can be used to debug link training issues. If the link does not train to any speed, including gen 1 speeds (2.5 Gb/s), then using Eye Scan is not recommended, and using the `ltssm` signal from the core is a better option. If the link is training to a slower speed but not training to a maximum speed, then using Eye Scan is recommended. An Eye Scan cannot be extracted when the link is changing rates, so extracting an Eye Scan with a slower rate is the easiest solution. After the link is trained to a slower speed, the margin can be analyzed to see if there is enough margin for a faster rate. Each lane of the link should be analyzed to see if one specific lane is causing the issue. If the eye appears acceptable, then it is possible that the link partner is initiating the slower speed.

Data Transfer Recovery

Going in and out of the Recovery state of `ltssm` might be an indication that Transaction Layer Packets (TLP) are failing the Link CRC (LCRC) check in the data link layer. Failing LCRCs might be an indication of a marginal link. Extracting an Eye Scan from the link while data is not being transferred would validate that the link is marginal. It can also pinpoint if there is one particular lane contributing to the LCRC errors.

Detection of Correctable Endpoint Errors

Reading the Device Status Register indicates whether there have been any correctable errors detected. See the PCI Express Specification [Ref 2] for details on correctable errors. If correctable errors are detected, it could indicate that there is a marginal link. An Eye Scan helps determine which lane is causing the marginal link and helps determine if the link is marginal under certain conditions. For example, if correctable errors are detected under a certain test patterns, an Eye Scan can be extracted during the specific test pattern to see how the link is affected.

Additional Diagnostics

When Eye Scan is integrated into a design, the data can help remotely debug a link. Along with the XADC, the die temperature can be swept to look at the eye under different die temperatures. Where there is control of the link partner's transmitter, the link can also be tuned for the optimal transmitter settings. This diagnostic capability allows a design to optimize the margin in a system.

Improving Product Yields

Learning that one out of 100 boards is not functional can make it difficult to debug. With Eye Scan, it is possible to determine whether margin issue on the link exists. If there is a margin issue, then the channel should be reanalyzed with IBIS-AMI simulations. IBIS-AMI simulations can help determine the root cause from where the issue is coming. Making improvements to the design of the serial channel improves product yields.

Hardware Description

The reference design provided with this application note utilizes the PCI Express example design provided from the generation of the 7 series FPGAs Integrated Block for PCI Express IP or the Virtex®-7 FPGA Gen3 Integrated Block for PCI Express IP. The example design from the IP implements a Programmed Input/Output (PIO) example design intended to function with the drivers described in *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 3]. Although the drivers described in *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 3] can be used with the PIO example design, the drivers are not required to extract an Eye Scan.

The reference design adds a subsystem created by IP integrator intended to extract the Eye Scan data from the transceiver. Implementing an Eye Scan requires access to the transceiver dynamic reconfiguration port (DRP) interface to access the Eye Scan registers. All Xilinx IP cores containing transceivers provide an option that allows access to the transceiver DRP interface. This reference design does not require any other connections to the Eye Scan subsystem, making the application note simple to integrate with any Xilinx-based IP core that contain transceivers.

The PIO example design handles the TLPs coming from the PCI Express core, while the Eye Scan subsystem handles the DRP accesses to the transceiver. Figure 2 shows a block diagram of how the system is constructed.

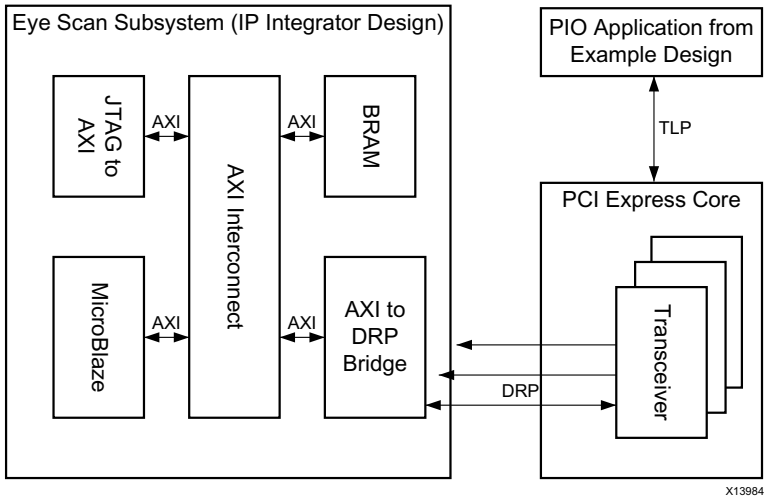

Figure 2: **High-Level Block Diagram of the Reference Design**

AXI Bridge to Dynamic Reconfiguration Port

The DRP interface from the PCI Express core is directly connected to the AXI to DRP Bridge IP core. The AXI to DRP Bridge core is part of the reference design and is packaged as an IP core in IP integrator. This IP makes it simple to access DRP registers of the transceiver through any AXI master such as MicroBlaze™, Xilinx Zynq®-7000 All Programmable Processing System (PS) or JTAG to AXI Master IP. Figure 3 shows how the AXI to DRP Bridge IP appears in IP integrator. On the left-hand side, the AXI interface is connected to the AXI interconnect so that it can be accessed by any AXI Master. The right-hand side drives the transceiver DRP interface provided from the PCI Express IP core.

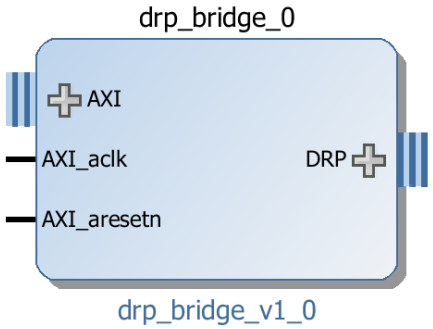

Figure 3: **AXI to DRP Bridge IP within IP Integrator**

The address mapping from the AXI interface is offset by two bits to the DRP address. The bridge was implemented this way to keep it light weight by not supporting unaligned transfers. This means the lower two bits of the AXI address are not used. The DRP address width of the transceiver is nine bits wide, so only the lower 11 bits of the AXI address are used to map to the DRP address. For example, if the base address for the AXI to DRP Bridge is 0xC0000000, then an access to address 0xC0000010 results in an access to address 0x004 to the transceiver DRP. The data returned to the AXI interface comes directly from DRP interface. The transceiver DRP data width is 16 bits, therefore only the lower 16 bits of the AXI Read and Write Data

Channels are used. Software must account for both the data width and the address mapping. Table 1 shows examples of the mapping.

Table 1: **Example of Address Correlation between AXI and DRP**

AXI Address	AXI Data	DRP Address	DRP Data
x0	x00001234	x0	x1234
x4	x00005678	x1	x5678
x8	x00009ABC	x2	x9ABC
xC	x0000DEF0	x3	xDEF0
…	…	…	…

One bridge is required for every transceiver that performs an Eye Scan. For example, if you want to scan all eight lanes of a x8 link, you must have eight AXI to DRP Bridge cores in the design. If only four lanes are desired to scan, then only four AXI to DRP Bridges are needed. Each AXI to DRP Bridge occupies only 34 LUTs and 60 FFs; making it a fairly light-weight IP.

When multiple lanes are used, each lane needs an offset of at least $2^{(\#\text{ of DRP address bits}+2)} = 2^{11} = 2$ KB. As of 2013.3, IP integrator has a limitation of requiring 4k of addressable space for each IP, so a 4k offset is used in the example design.

Processing System

MicroBlaze or the ARM® A9 core communicates to the transceiver DRP interface through the AXI to DRP Bridge. The software running through MicroBlaze is derived from *Eye Scan with MicroBlaze MCS* [Ref 1], however, it has been expanded to support GTP, GTX, and GTH transceivers. The software included with the reference design also makes it simple to scale for the amount of lanes in the design. To switch between GTP, GTX, and GTH transceivers, modify the `drp.h` header file and include a `#define GTP`, `#define GTX`, or `#define GTH`. The appropriate `#define` is already used depending on whether the files from the AC701, KC705, VC709, ZC706, or KCU105 directory are used. To scale for a different number of lanes, the header file `es_controller.h` needs to be modified for the appropriate lane width. The definition of `NUM_LANES` represents the number of lanes in the design.

The software running on the processor implements the Eye Scan algorithm on the transceiver and stores the data in the AXI block RAM. When the block RAM is filled with Eye Scan data, the JTAG to AXI IP core reads the data out of the block RAM. Then it stores the data on the local machine in a text file through the JTAG connection to the PC. The block RAM is AXI based and can be scaled for a change in lane width. Each lane requires a minimum of 2 KB of block RAM depth for this reference design. This can be modified with IP integrator in the **Address Editor**.

Local Storage

As mentioned in Processing System, the JTAG to AXI IP reads the Eye Scan data out of the AXI block RAM. The JTAG to AXI IP places the Eye Scan data on the local machine through the JTAG connection to the board. This continues until a full Horizontal and Vertical scan is completed. After the Horizontal and Vertical scan completes, the processor is notified and stops implementing the Eye Scan. When a block RAM is sized to be 2 KB, this allows for 2 KB of data. A full Eye Scan typically requires more memory space than 2 KB. There is a maximum of 4,096 horizontal taps and 256 vertical taps. Each error count measurement is two bytes wide, meaning that a maximum-sized scan involves 256 × 4,096 × 2 = 2 MB of error scan data. (Several more bytes are required to store the prescale, horizontal and vertical offsets, and the UT sign). Because the AXI block RAM depth is only 2 KB per lane, a handshaking between the JTAG to AXI IP and the processor is necessary. The handshaking allows for JTAG to empty the block RAM content while the processor is filling up the block RAM with scan data. This handshaking implementation is a drop-in solution but is not required. If the a design contains 2

MB of memory space available for each lane, then the Eye Scan data can be fully stored and not require the handshake between the JTAG to AXI IP and the processor.

Tcl Post Processing

After the JTAG to AXI IP reads the Eye Scan data out of the block RAM, the data is stored locally in a file labeled `CH#.dump` where the # represents the lane number. This file contains all of the error count data read from the `error_count` register of each transceiver's DRP space at each horizontal and vertical location. The `CH#.dump` file must be converted to a format that Vivado tools can read so Vivado can display the Eye Scan data in the Vivado IDE. The Vivado Eye Scan viewer requires bit error rates as inputs instead of the error count. The error count values must be converted to bit error rates. This post-processing is done in the Tcl script provided in the reference design. This Tcl script is derived from XAPP743 [Ref 1] and modified to allow simple scaling for multiple lanes and also can be used with the JTAG to AXI IP core instead of XMD. The output from the post processing is a CSV file that can be read in by Vivado as an Eye Scan. Figure 4 shows how a scan appears in the Vivado IDE.

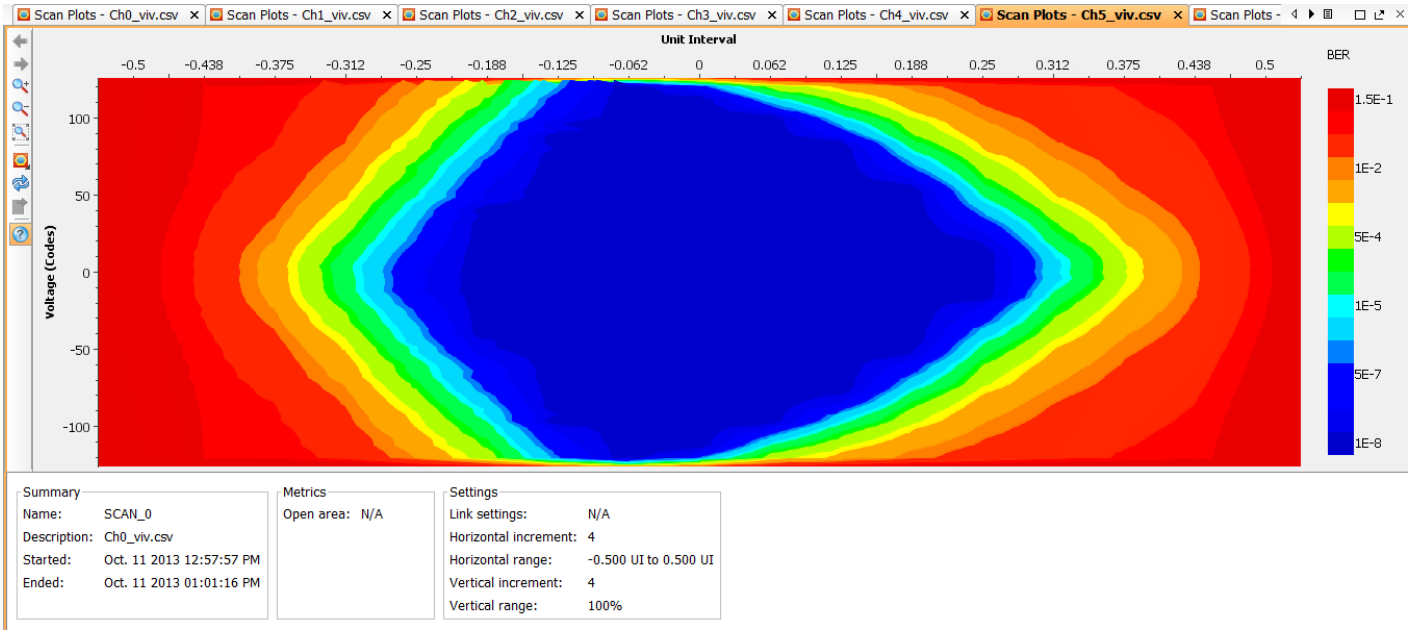

Figure 4: **Eye Scan Result in Vivado IDE**

IP Integrator System

The Eye Scan subsystem is built with the `ipi_<board name>.tcl` file located in the `<board name>/source/ipi_tcl` directory. When this file is sourced, all of the previously-described components are connected together with IP integrator as shown in Figure 5. This subsystem can be added into any design with or without an already existing processor in the design. The

Zynq-7000 example is not a drop-in solution because there is only one Processing System per Zynq-7000 device. When using Zynq-7000, the software from the example must be integrated.

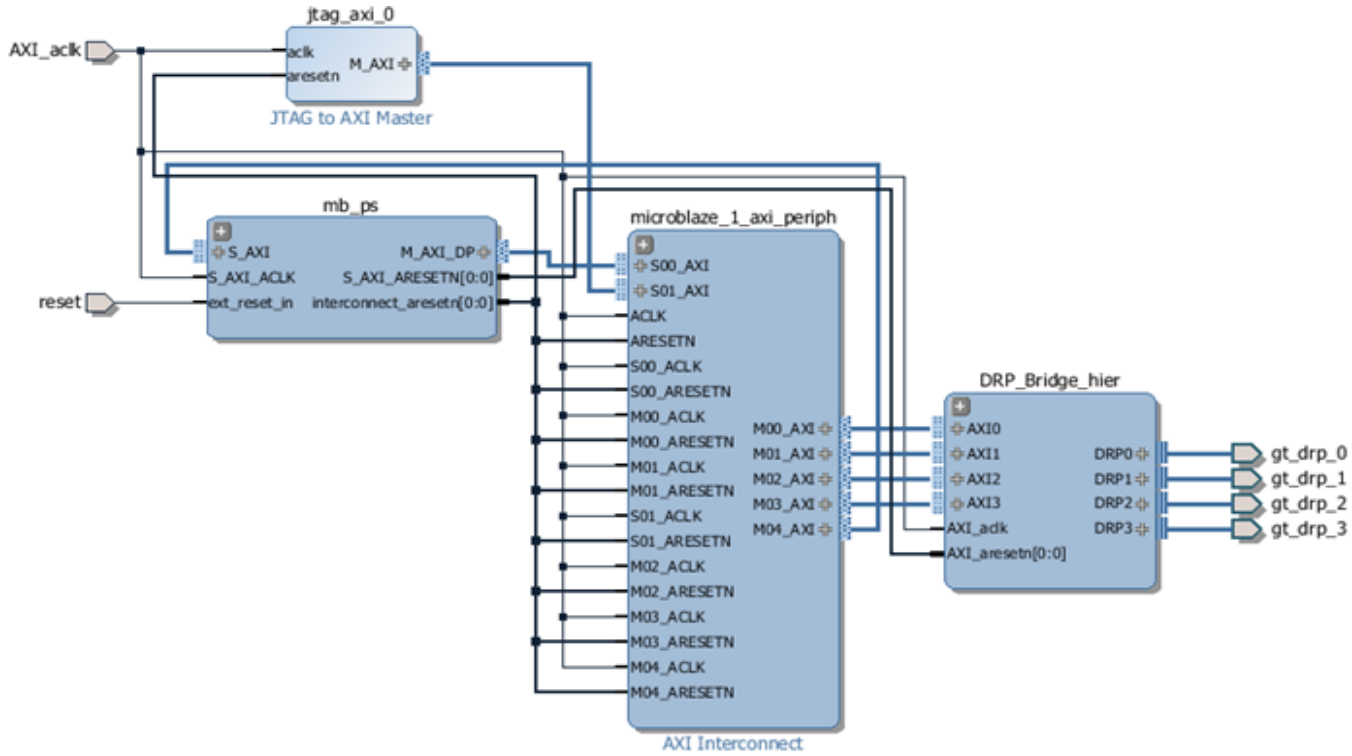

Figure 5: **IP Integrator Eye Scan Subsystem**

Clocks and Resets

The `AXI_aclk` and reset input to the subsystem are provided by existing signals in the PIO example design. The reset line is tied to the `user_lnk_up` output of the PCI Express core, holding the subsystem in reset until the link is at L0. The `AXI_aclk` is a clock coming from the `ext_ch_gt_drpclk` output from the PCI Express core, which is the same clock driving the transceiver DRP interface.

For a custom implementation, the `AXI_aclk` must be the same as the DRP clock to the transceiver. The reset can come from anywhere provided that the signal is deasserted when implementing an Eye Scan. The reset must also be deasserted while the `AXI_aclk` is stable.

Implementing the Reference Design

Click here to download the design files associated with this application note.

The reference design source files are provided in the following five directories:

- AC701
- KC705
- VC709
- ZC706
- KCU105

The reference designs target a particular evaluation board indicated by the name of the directory. To leverage the design files for a custom design, choose the appropriate design according to the transceiver type. If using Virtex-7, note that some Virtex-7 devices contain GTX transceivers. If a Virtex-7 device contains GTX transceivers, then it is best to use the KC705 files as a reference.

To create a bit file, open the Vivado IDE and go to the **Tcl Console** at the bottom of the Vivado IDE or go to **Window > Tcl Console**.

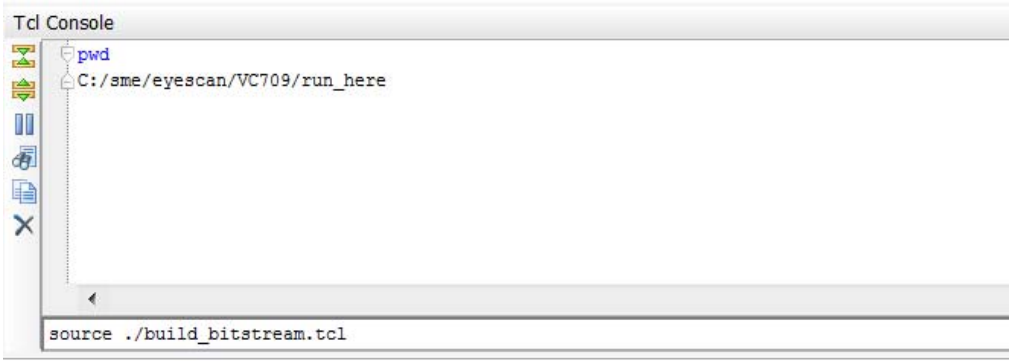

Figure 6: **Tcl Console Showing How to Source Build Files**

In the Tcl console, change the working directory to the `run_here` directory within the appropriate directory according to the board. In the `run_here` directory, type the following command:

```
>> source ./build_bitstream.tcl
```

Running this Tcl script performs the following:

- Creates the Vivado Project with the IP integrator design.
- Adds all the RTL and IP cores into the project.
- Runs Synthesis and Implementation.
- Creates a bitstream with the software embedded in the LMB block RAM.
- Sources the Scanning Tcl files to prepare Vivado for a scan.
  - This allows the you to type “run_eyescan” in the Tcl console to implement a scan.

After the Tcl script completes, it is now possible to program the FPGA. It is necessary to plug the evaluation board into a PCI Express slot. Apply power to the FPGA and program the FPGA within the Vivado IDE. Do not remove the JTAG connection as this connection is required for the JTAG to AXI IP. Turn on the host containing the evaluation board plugged in and programmed. This establishes a link and the system is now ready to perform an Eye Scan on the transceiver. Type the following in the Tcl console:

```
>>run_eyescan
```

The **`run_eyescan`** command comes from the `run_eyescan.tcl` file in the `scan_time` directory. If the project is closed, then the `run_eyscan.tcl` file needs to be sourced again. If the `run_eyescan.tcl` file is modified, it also must be sourced again within Vivado. The Tcl file might need to be modified for several variables shown in the following example. The explanation of when to modify them is listed in Table 2.

```
# horz_step: Horizontal sweep step size. Minimum value is 1. Maximum value
depends on data rate mode (see user guide and rate variable below).
set horz_step($i) 4

# vert_step: Vertical sweep step size. Minimum value is 1. Maximum value is
127.
set vert_step($i) 4

# max_prescale: Maximum prescale value for sample count. Minimum value is
0. Maximum value is 31.
set max_prescale($i) 5
```

```
# data_width: Parallel data width interface. Valid values are 16, 20, 32,
and 40.
set data_width($i) 40

# lpm_mode: Set to 1 for LPM mode. Set to 0 for DFE mode.
set lpm_mode($i) 0

# rate: Set depending on full, half, quarter, octal, or hex modes (see user
guide).
set rate($i) 64
```

Table 2: **Description on Tcl Variables**

Variables	Description	When to modify
horz_step	Describes the resolution of the horizontal scan.	This value can be increased to reduce the amount of time the scan takes. A larger value results in less horizontal resolution. For optimal resolution, set to 1.
vert_step	Describes the resolution of the vertical scan.	This value can be increased to reduce the amount of time the scan takes. A larger value results in less vertical resolution. For the optimal resolution, set to 1.
max_prescale	Changes the amount of time spent at each vertical/horizontal sample. See the *Gear Shifting* section of *Eye Scan with MicroBlaze MCS* [Ref 1] for more information.	If a need for a finer BER, make this value larger; this results in a longer scan time. For a faster scan, make this value smaller.
data_width	The internal data width. See *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 5] and *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 6] for more information.	This is design dependent. The BER calculations is incorrect if this is not set correctly.
lpm_mode	This specifies whether the transceiver is in DFE or LPM mode	If the LPMEN port of the transceiver is High set this to 1. Set to 0 if LPMEN is set Low.
rate	Specifies the horizontal resolution	Use the following to determine the correct value. RXOUT_DIV = 1 rate = 32 RXOUT_DIV = 2 rate = 64 RXOUT_DIV = 4 rate = 128 RXOUT_DIV = 8 rate = 256 RXOUT_DIV = 16 rate = 512

Scaling for Different Lane Widths

Follow the steps to scale for different lane widths.

1. The IP integrator AXI block RAM must be scaled.

   Each lane needs 2 KB of block RAM. This can be scaled with the **Address Editor** in IP integrator. Be sure to modify the width for **jtag_axi_0** and **mb_ps/microblaze_1**. Figure 7 displays the **Address Editor**.

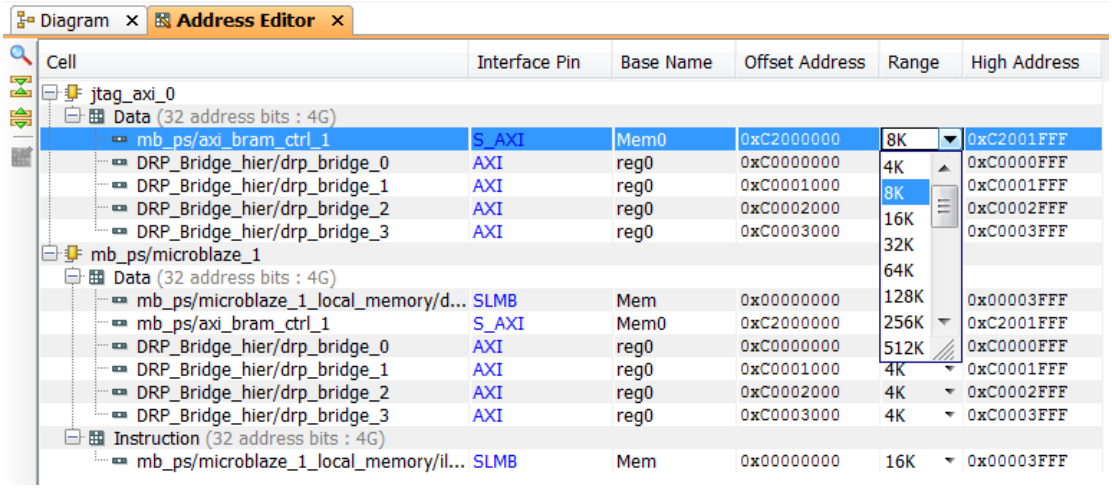

Figure 7: **Adding More Block RAM to Scale for More Lanes**

If the base address of the AXI block RAM is moved from 0xC2000000, then the `run_eyescan.tcl` file also must be modified. The **axi_bram_addr** variable is shown in Figure 8 in the in the `run_eyescan.tcl` file and it needs to match what is listed in the **Address Editor**.

```
1
2 # Base Address of AXI BRAM
3 set axi_bram_addr 0xC2000000
4
```

Figure 8: **axi_bram_addr Variable in the run_eyescan.tcl File**

2. Match the AXI to DRP Bridge count with the lane width.

   This can be done by removing or adding AXI to DRP Bridges into/from the design. When adding or removing the AXI to DRP Bridges, make sure to keep the addresses contiguous in the **Address Editor**. If they are not contiguous, then the software also needs to be modified. Notice the difference between a 4-lane and 8-lane implementation in the IP integrator canvas in Figure 9.

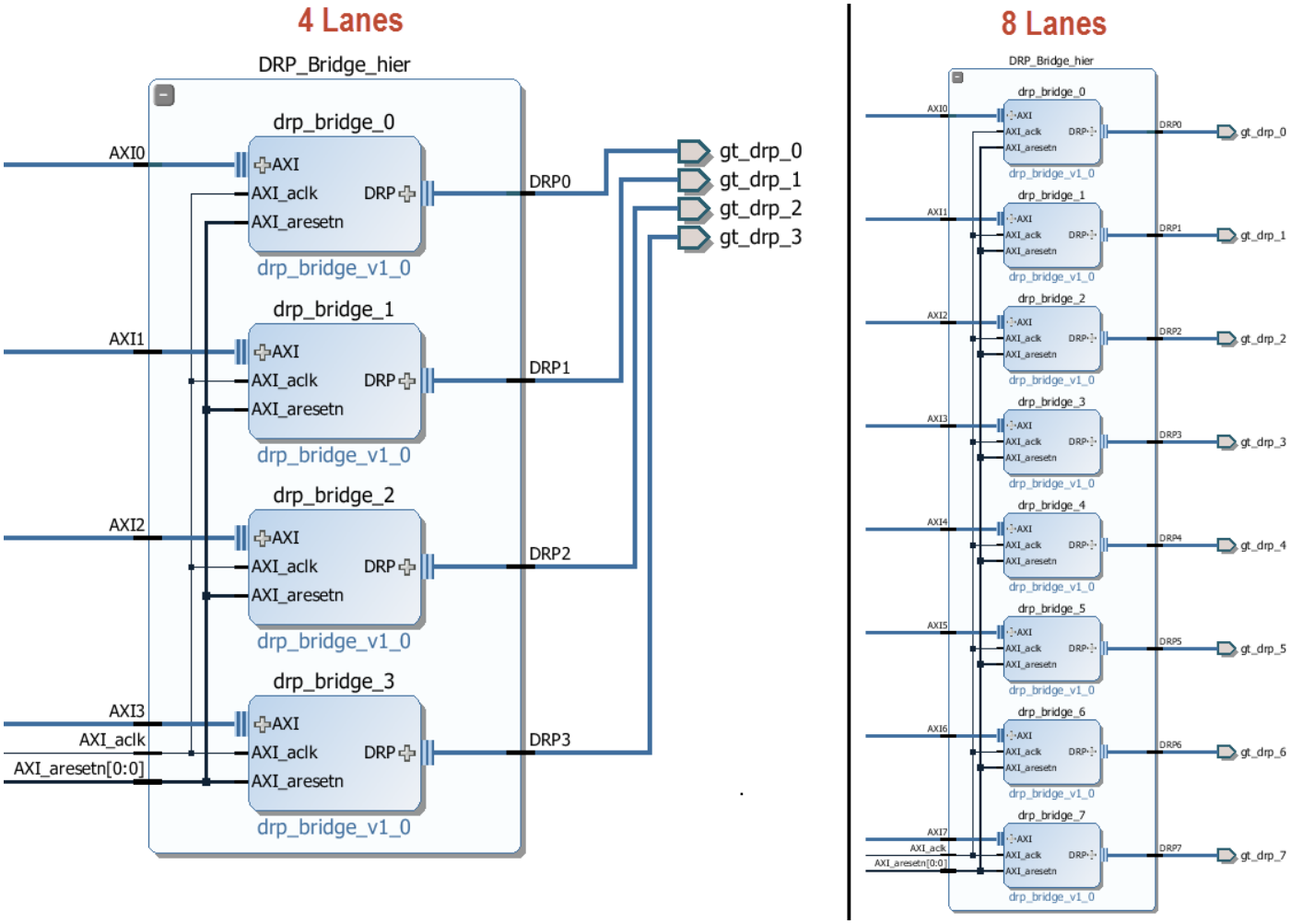

Figure 9: **IP Integrator Canvas When Scaling for Different Lane Widths**

Because more AXI to DRP Bridges are used, the AXI Interconnect must be adjusted to accommodate for the number of AXI to DRP Bridges.

3. A new ELF must be generated.

   This requires modifying the original C code. To modify the C code and create a new ELF, the hardware specification must be exported. This can be done at several different stages, but it is easiest to do immediately after synthesis. In IP integrator, right-click the **`eyescan_subsystem`** and select **Export Hardware for SDK** shown Figure 10.

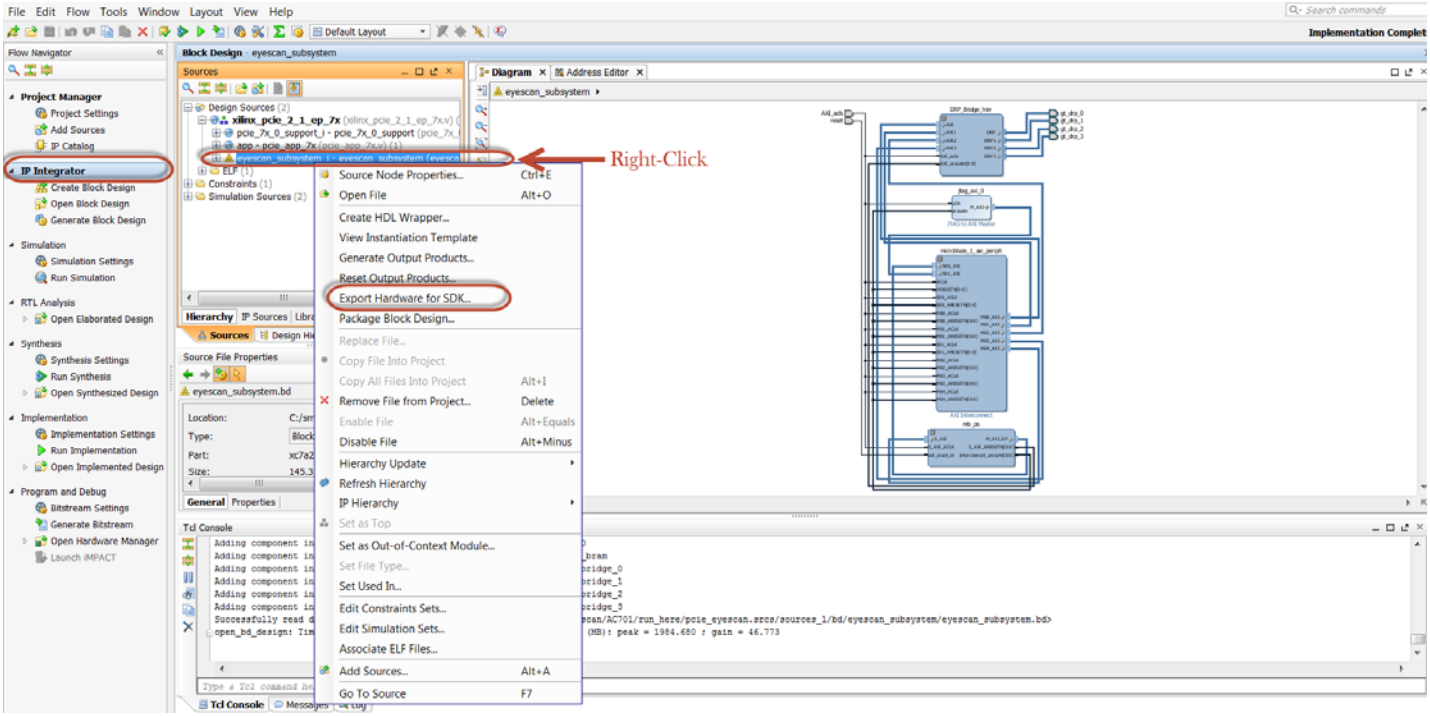

Figure 10: **Exporting IP Integrator Subsystem to SDK**

4. After selecting **Export Hardware for SDK**, select the options shown in Figure 11. This launches the SDK with the hardware specification.

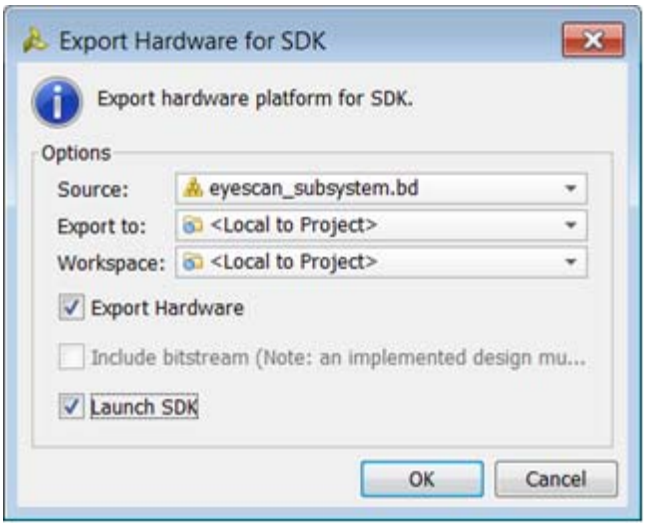

Figure 11: **Launching SDK from Vivado**

5. Next, create a new Board Support Package (BSP) by clicking **File > New > Board Support Package** as shown in Figure 12.

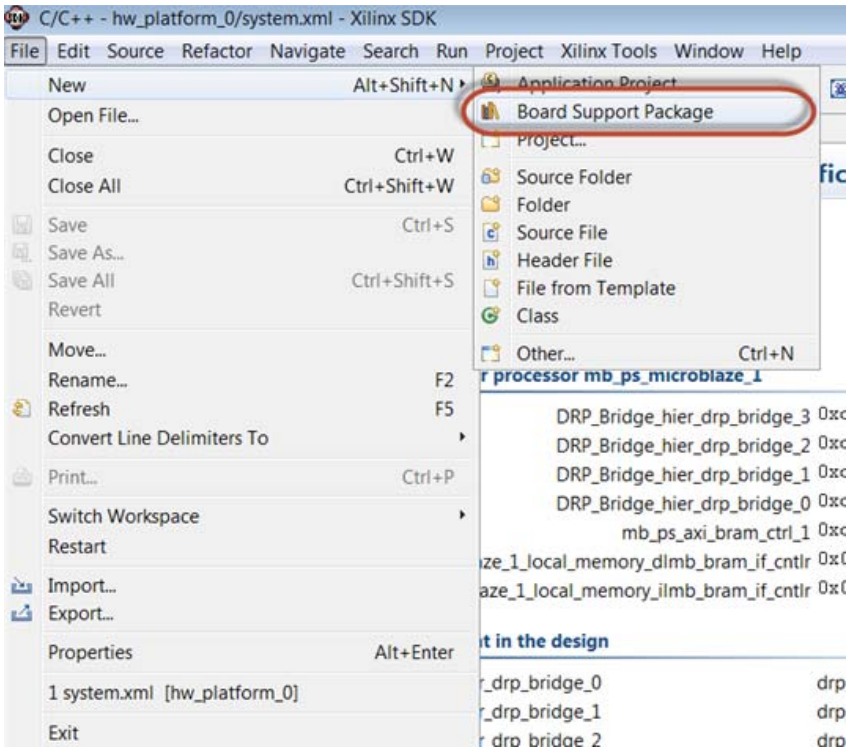

Figure 12: **Creating a Board Support Package**

6. Change the name of the BSP to **`eyescan_software_bsp`** as shown in Figure 13 and select **Finish**. The name of the BSP must be changed so the application can be imported correctly.

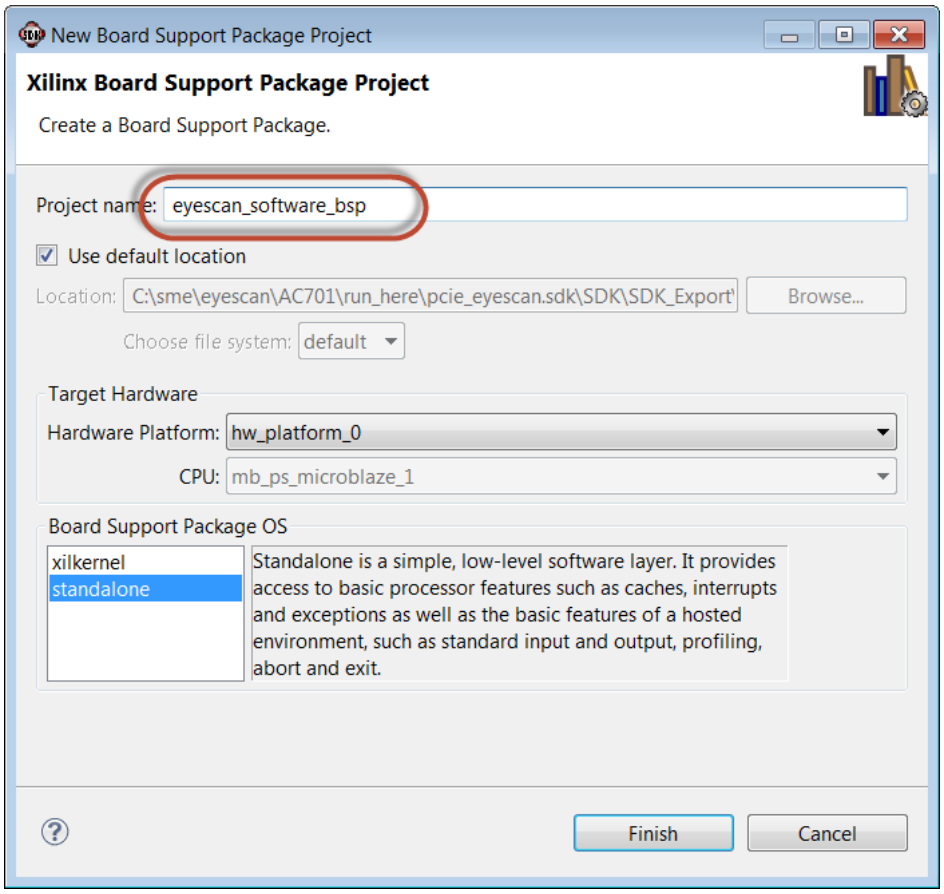

Figure 13: **Changing the BSP Project Name**

7. Select the default options for the BSP shown in Figure 14.

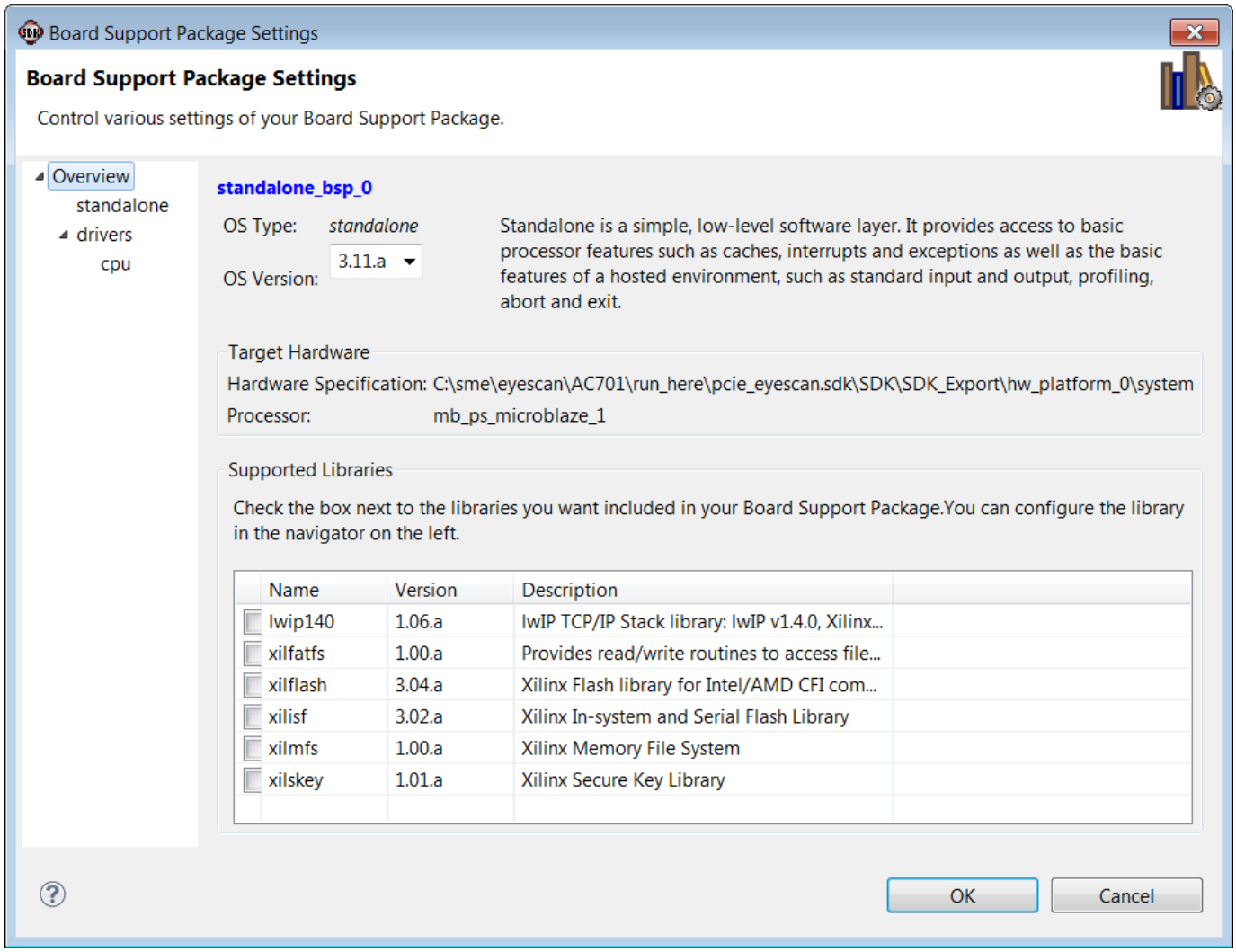

Figure 14: **BSP Settings**

8. Right-click in the **Project Explorer** panel and select **Import** as shown in Figure 15.

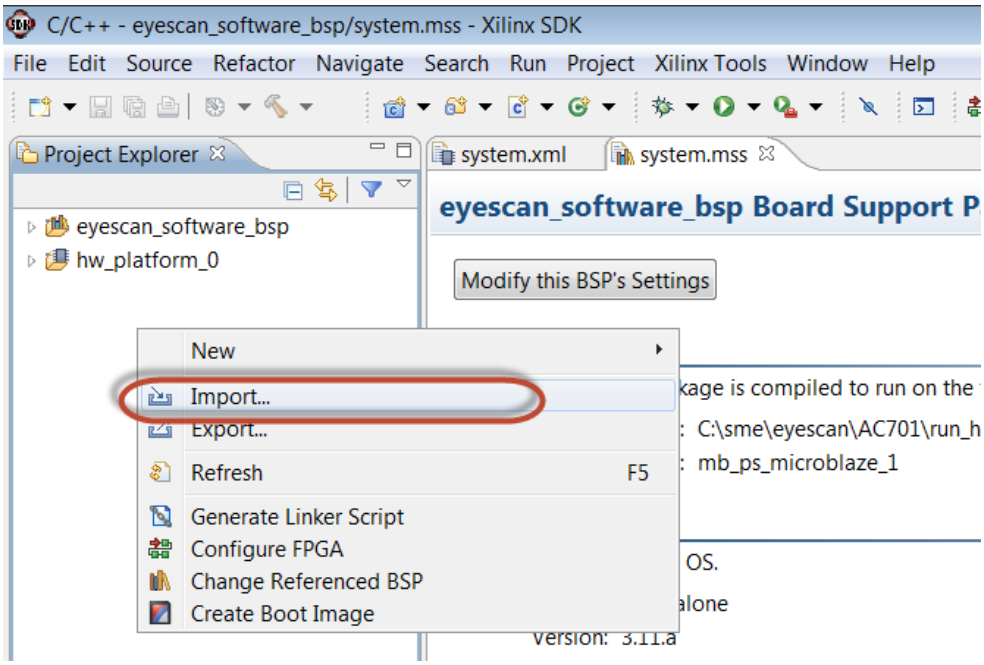

Figure 15: **Importing Eye Scan Software Step 1**

9. An Import window launches. Select **General > Existing Projects** into Workspace and then click **Next** as shown in Figure 16.

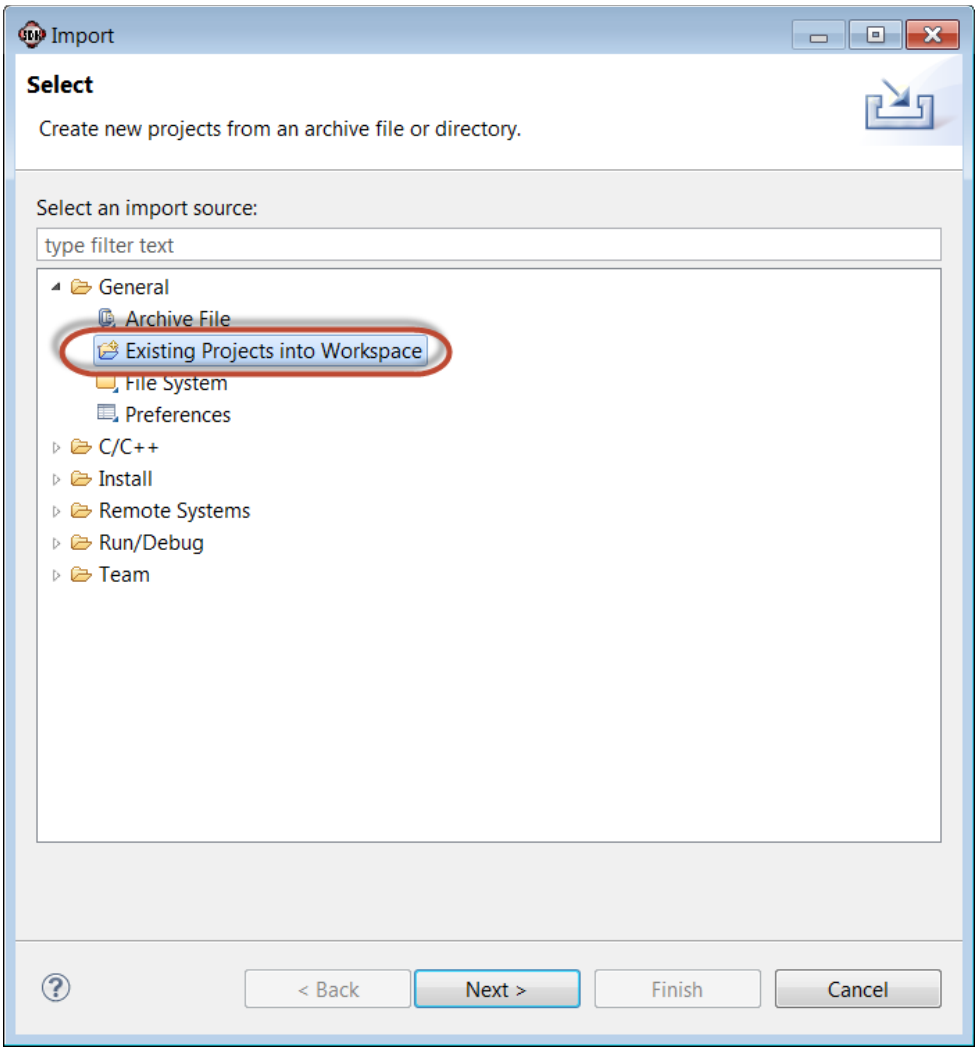

Figure 16: **Importing Eye Scan Software Step 2**

10. In the next window, choose the **Select** archive file: option and browse for the appropriate zip file in the “source/software” directory. The appropriate file is referring to either the GTP, GTX, or GTH software file. In the **Projects** panel, make sure the software directory is selected as shown in Figure 17. Click **Finish**.

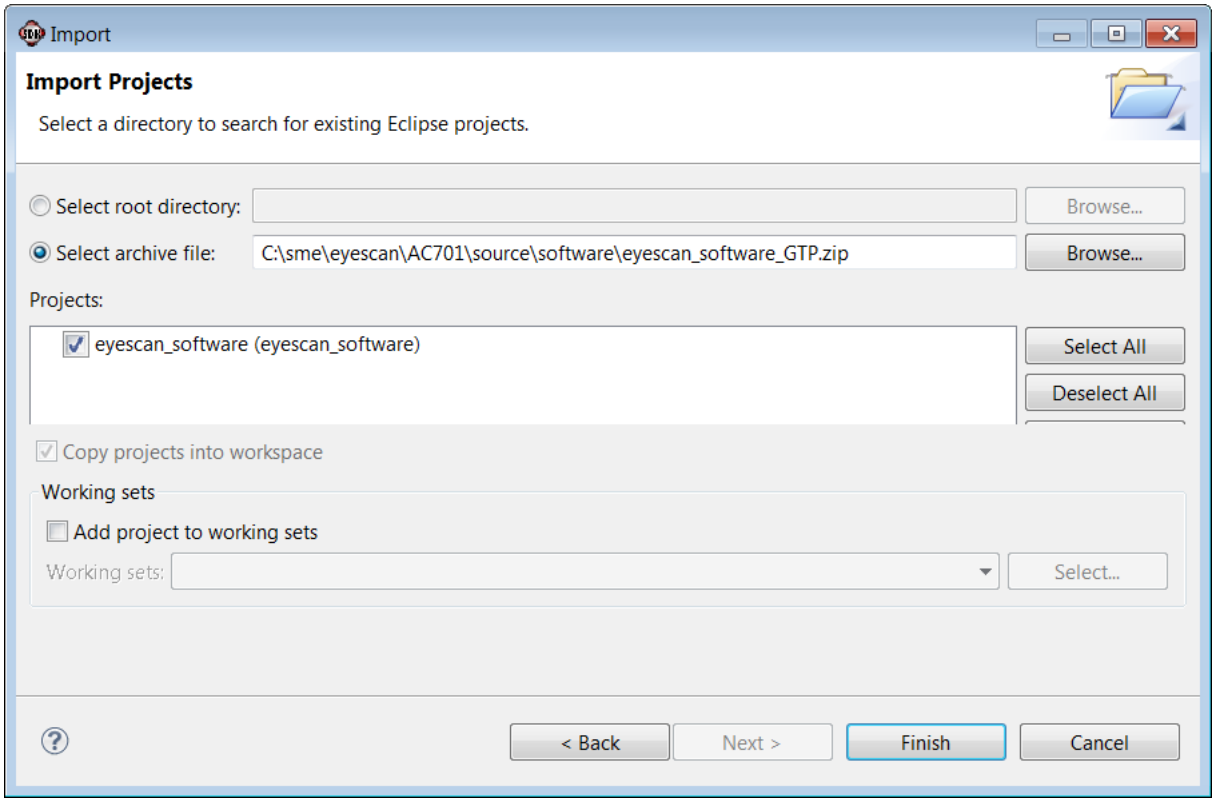

Figure 17: **Importing an Archive File**

11. Locate the `es_controller.h` file and change `NUM_LANES` to the appropriate number of lanes. Save the changes and recompile to create a new ELF file in the `Debug` directory shown in Figure 18.

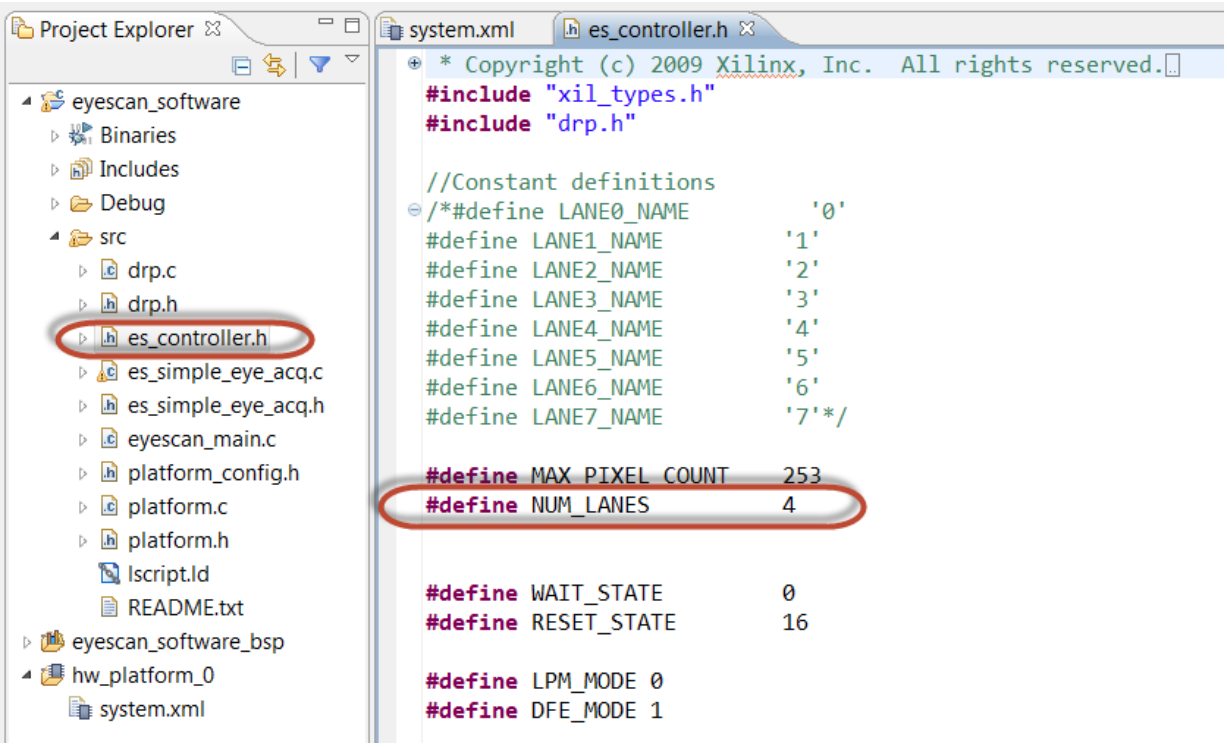

Figure 18: **Modifying es_controller.h for Different Lane Widths**

12. After modifying the ELF, include the ELF in Vivado project (**MicroBlaze only**). To perform this, make sure to be in the Project Manager view and go to **Tools > Associate ELF Files** (Figure 19).

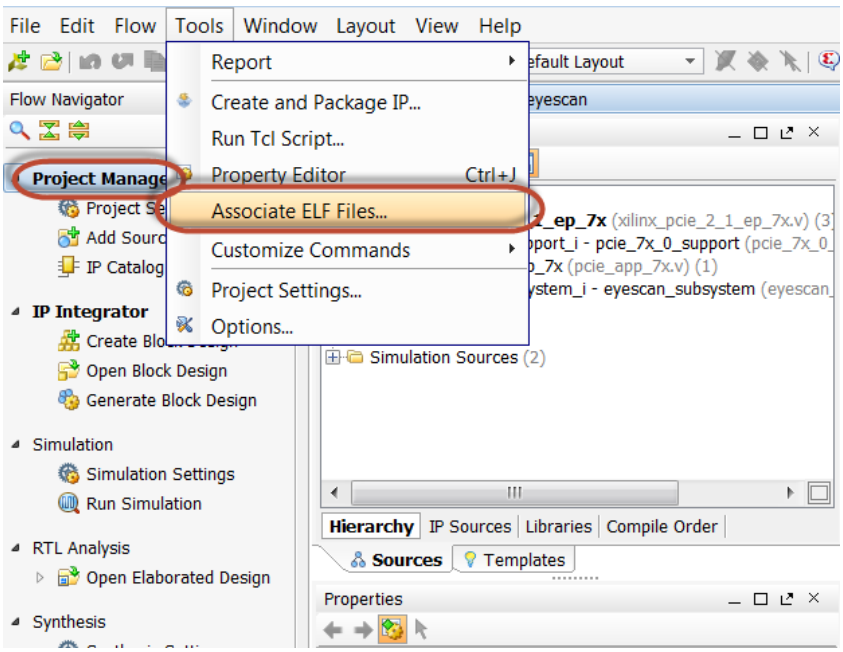

Figure 19: **Associating ELF Files to MicroBlaze Part 1**

13. Select the new ELF file in the subsequent window and click **OK**.

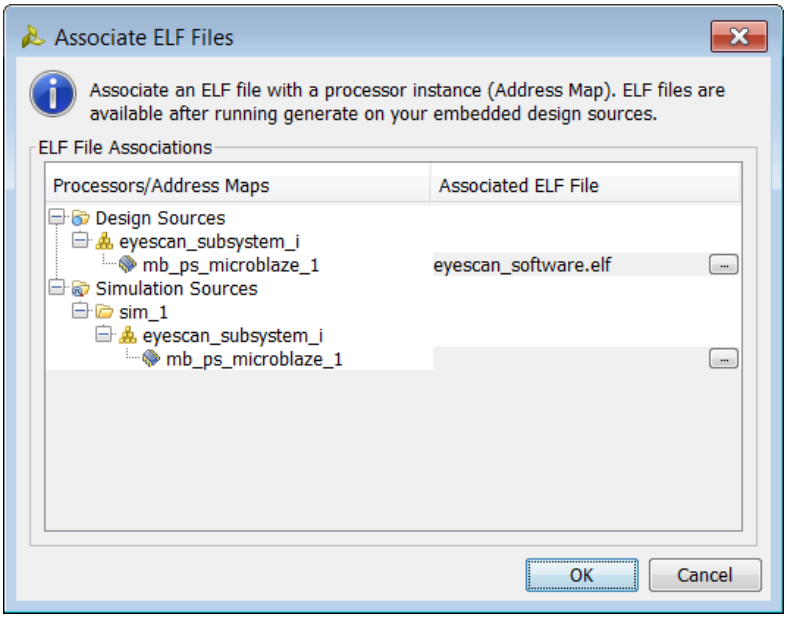

Figure 20: **Associating ELF Files to MicroBlaze Part 2**

14. After the ELF is mapped, generate the bitstream. The `run_eyescan.tcl` file must be modified. The `NUM_CH` must be modified the actual number of lanes.

```
12  ####################
13  # Set test parameters
14  ####################
15  set NUM_CH 8
16  set ch_list 0
17
```

Figure 21: **Modifying run_eyescan.tcl for Different Lane Widths**

15. Create a boot image (**Zynq-7000 only**)

    A boot image generally requires three files. A bit file, a software application, and an FSBL. The bit file and software application are already created. To create an FSBL, go to **File > New > Application Project**, shown in Figure 22.

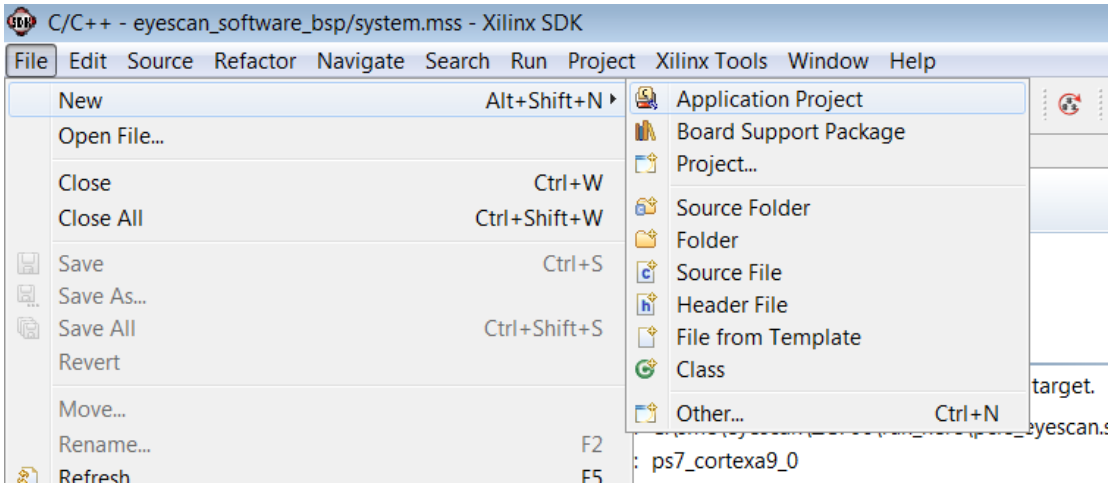

Figure 22: **Creating an FSBL**

16. The name of the FSBL is not important, but this application note names it `myfsbl`. It is important to use the existing BSP linked to the software application as shown in Figure 23.

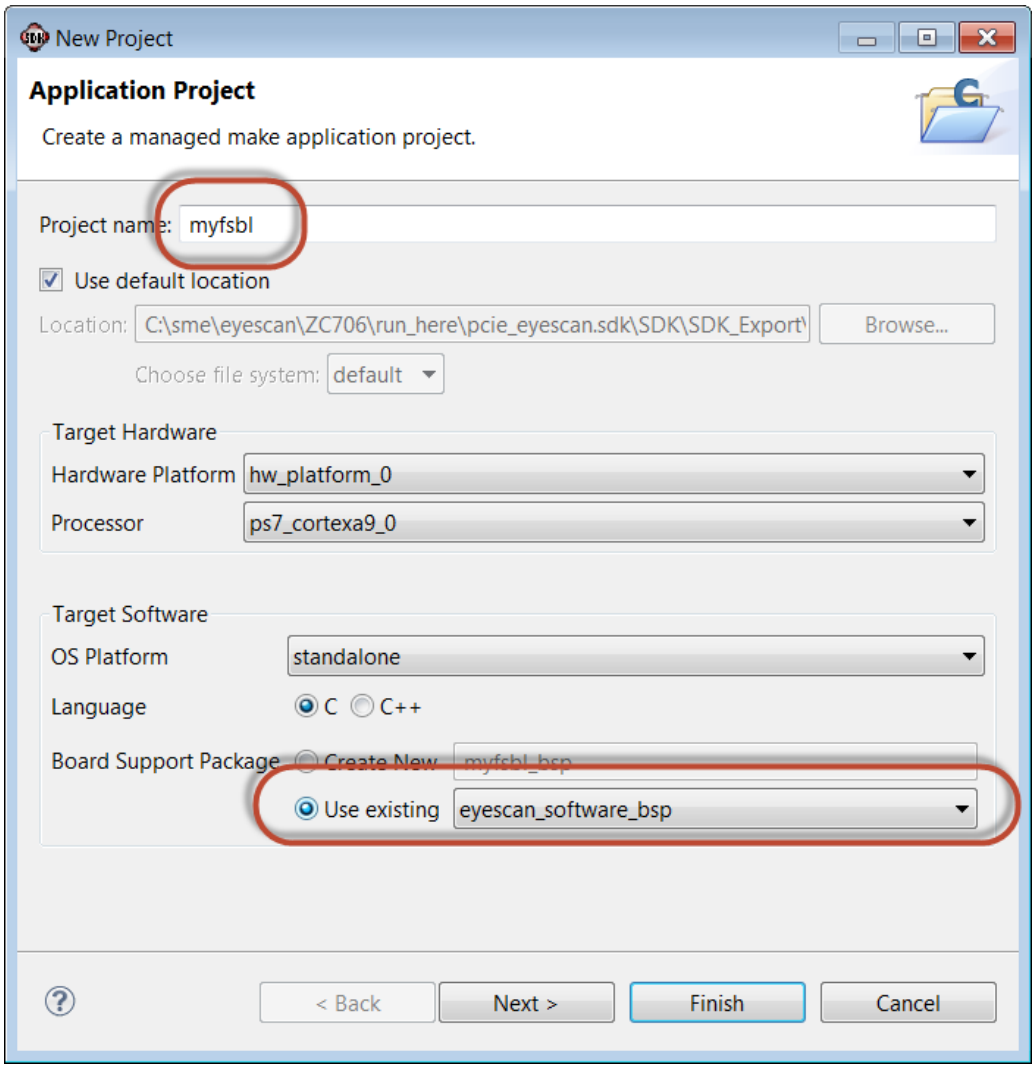

Figure 23: **Labeling FSBL Project**

17. Select **Zynq FSBL** and click **Finish**.

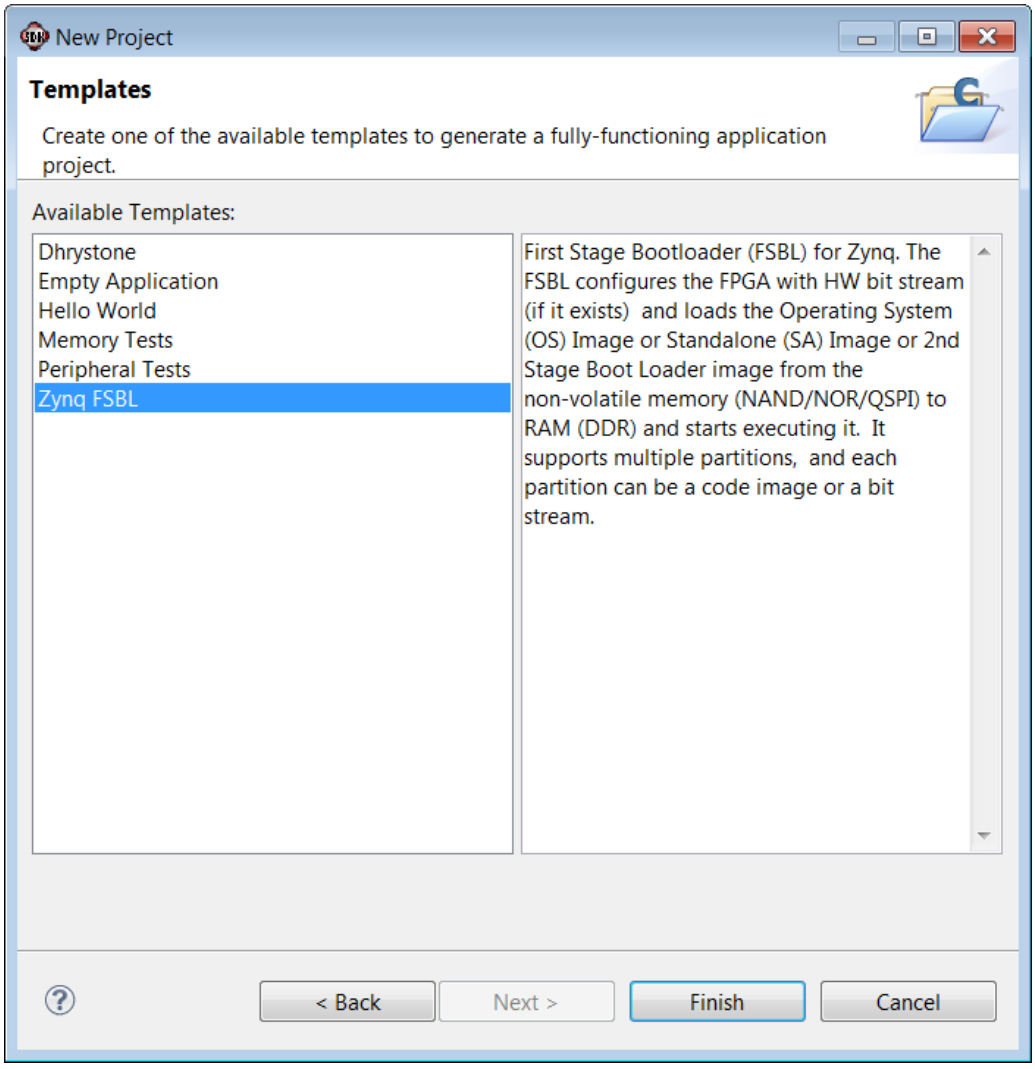

Figure 24: **Specifying a Zynq FSBL Template**

18. Next, create the image by right-clicking the `eye_scan_software` project and select **Create Boot Image** as shown in Figure 25.

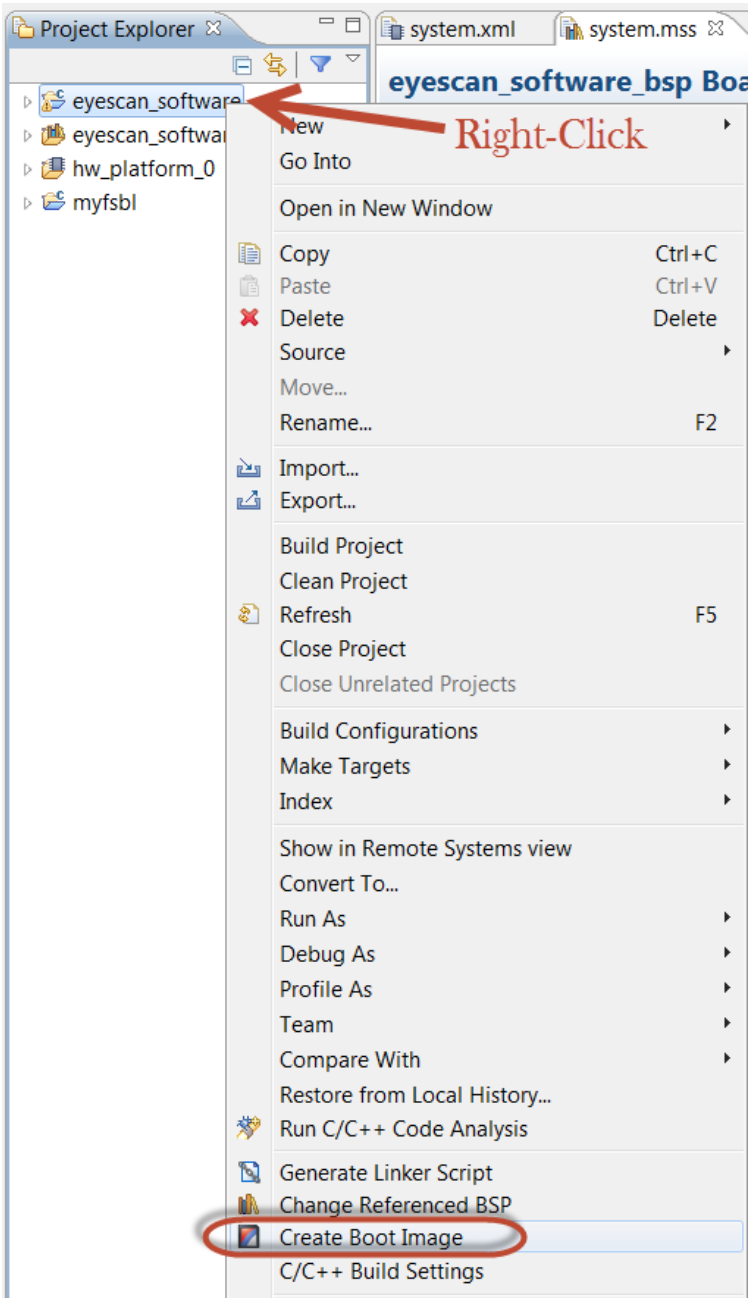

Figure 25: **Creating a Zynq-7000 Boot Image**

19. Change the output extension to `.mcs` as shown in Figure 26 and select **Create Image**.

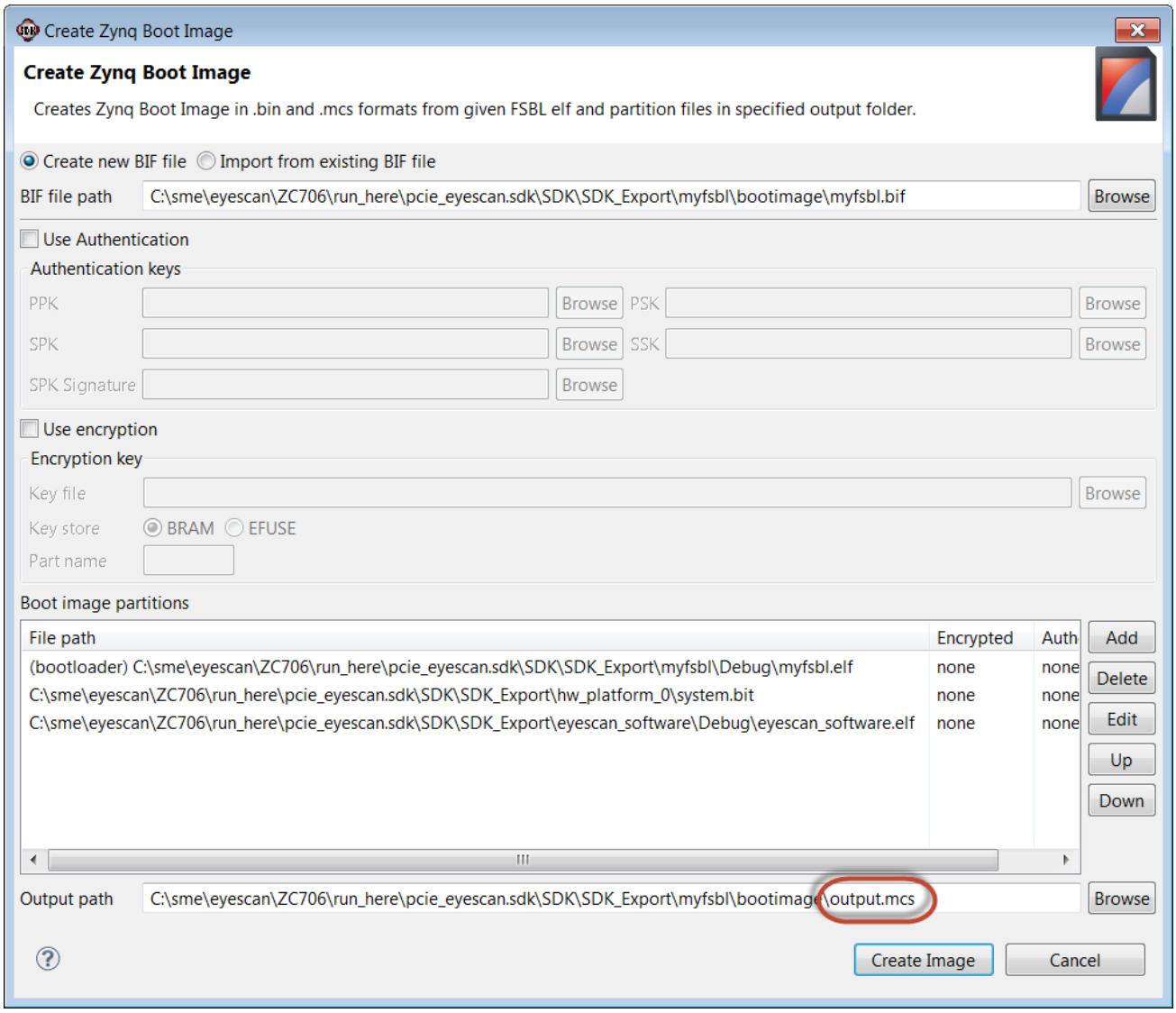

Figure 26: **Specify Boot Image File Name**

Incorporate a Custom Design

There are two primary ways to incorporate the reference design files into a custom design.

1. Add the entire IP integrator subsystem and connect to DRP interfaces
2. Add the IP integrator design and consider modifications:
   - Remove MicroBlaze and replace with another AXI master. Potential options:
     - PCI Express
     - Ethernet
   - Remove the AXI block RAM and replace with a deeper memory. Potential options:
     - DDR3
     - Flash
     - Local memory of system
   - Replace JTAG to AXI IP with another AXI master. Potential options:
     - PCI Express
     - Ethernet

- Use an existing MicroBlaze and incorporate software

The option 1 is fairly straight-forward and is not discussed further in this section due to the simplicity. Option 2 allows for flexibility, yet is very simple from a hardware standpoint due to the AXI4 plug-and-play functionality with IP integrator.

Figure 27 shows an example of how to integrate the example into a design.

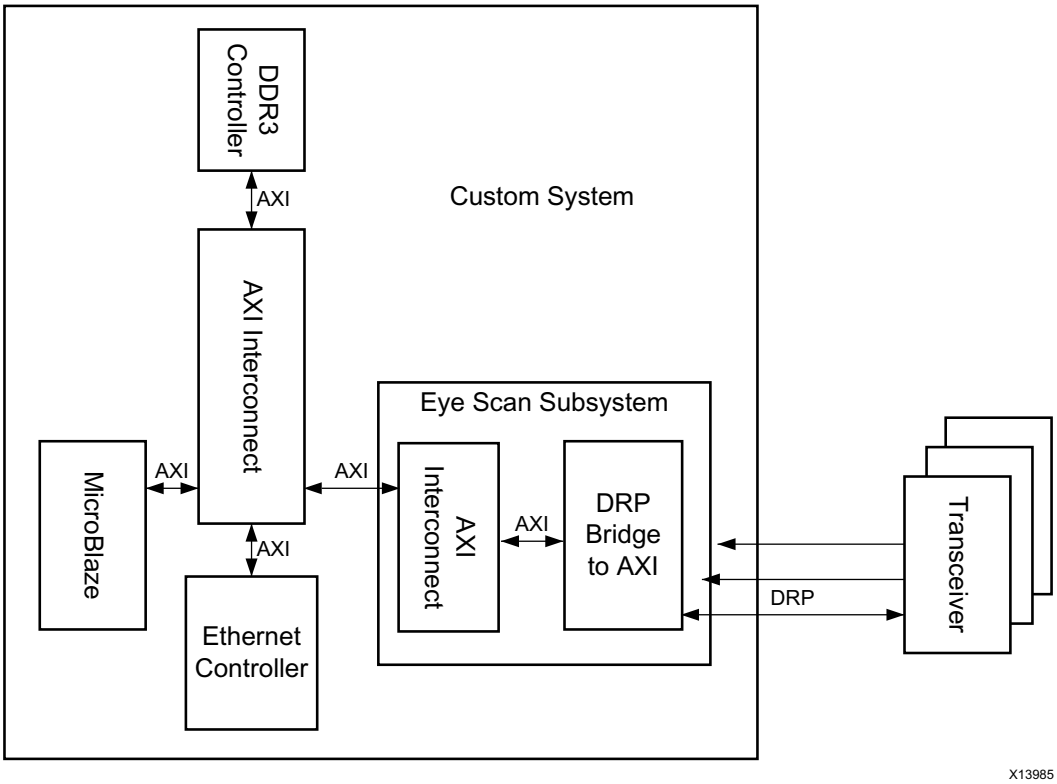

Figure 27: **Example System Showing Integration Options**

Figure 27 shows one implementation option where MicroBlaze can store all of the scan data in the DDR3 SDRAM. After it completes, then it can upload the data using the Ethernet Controller so the scan can be displayed later. In this example, the MicroBlaze performs the functions of the original MicroBlaze from the reference design as well as the function of the JTAG to AXI IP where it pulls the data from storage. Instead, the communication off of the FPGA would be through Ethernet.

Upgrading

To upgrade to a newer version of Vivado, the reference design needs to generate all of the output products in Vivado 2013.3. After the output products are generated, the project can be opened with a newer version of Vivado to upgrade the IP core. For details, follow the upgrade guidelines from *Vivado Design Suite User Guide: Designing with IP* [Ref 4].

Using the Prebuilt Images

Prebuilt bitstreams for the AC701, KC705, VC709, and KCU105 are provided to run the examples quickly without building all of the outputs files. An MCS image is provided for the ZC706 to load the PS software and PL bitstream. The bitstreams can be programmed with Vivado, while the MCS image needs to be programmed with SDK.

To use the prebuilt images open the Vivado IDE and change the working directory to the `prebuilt_images` directory within the Tcl console. After the boards are programed, run the following command in the Vivado Tcl console:

```
>> open_hw
```

Source the source_eyescan.tcl file with the following command in the Tcl console:

```
>> source -quiet ./source_eyescan.tcl
```

Depending on the targeted board, the next command varies. For example, if using the VC709 evaluation board, run the following command in the Tcl Console:

```
>> source_eyescan -board VC709
```

A scan should begin and finish by uploading the Eye Scan results to the Vivado IDE.

Resource Utilization

Table 3 describes the resource utilization for x4 lane subsystem.

Table 3: **Resource Utilization for x4 Lane Subsystem**

Eye Scan Subsystem (x4 Lanes)	LUTs	FFs	RAMs
MicroBlaze	788	655	0
Interconnect	1,864	1,677	0
4 DRP Bridges	136	240	0
AXI block RAM	293	270	2
Local block RAM	12	6	4
JTAG to AXI	1,488	1,801	0

Table 4 describes the resource utilization for x8 lane subsystem.

Table 4: **Resource Utilization for x8 Lane Subsystem**

Eye Scan Subsystem (x8 Lanes)	LUTs	FFs	RAMs
MicroBlaze	795	655	0
Interconnect	3,522	3,270	0
8 DRP Bridges	272	480	0
AXI block RAM	295	276	4
Local block RAM	12	6	4
JTAG to AXI	1,487	1,801	0

File Description

Table 5 describes the directory structure of the reference design.

Table 5: **File and Directory Structure Description**

Directory and Files <> Refers to a Directory	Description
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <run_here> build_bitstream.tcl	This Tcl script creates the Vivado project and run through the tools to generate a bit file. This also sources the Eye Scan scripts so that it is ready to be scanned.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <scan_time> run_eyescan.tcl	This Tcl script contains the procedures for running an Eye Scan. After this file is sourced, it is required to execute the run_scan command to extract an Eye Scan. This file contains variables that need to be correct for the Eye Scan to work properly.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <scan_time> Es_pc_host.tcl	This Tcl script contains the primary procedures which poll control registers and also reads the Eye Scan data from the block RAM.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <scan_time> Load_vivado_scans.tcl	This Tcl script takes the CSV files created from the Eye Scan and display them within the Vivado IDE.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/rtl>	This folder contains all the RTL for the PCI Express example design
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/ipi_design> [board name]_ipi.tcl	This Tcl script creates the IP integrator design. This was originally created with the write_bd_tcl command when the subsystem was originally created.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/custom_ip> <drp_bridge_ip> <drp_mux_ip>	These two directories contain custom IP for the IP integrator design. The <custom_ip> directory needs to be added to the project's IP repository. These IPs were packaged with the Vivado IP packager. This example does not use the drp MUX IP, however, the MUX was provided two separate modules want to access the DRP.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/core>	This directory contains the source file for the PCI Express core. The extension is .XCI.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/constraints>	This directory contains the Vivado constraints file for the PCI Express example design. The extension is .XDC.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/software> Prebuilt.er	The prebuilt_elf.elf is the software compiled for MicroBlaze that runs the Eye Scan algorithm. The ELF is ultimately packed into the MicroBlaze local block RAM.
<Board Name: AC701, KC705, VC709, KCU105, or ZC706> <source/software> eyescan_software_[GT type: GTP, GTX, or GTH].zip	This zip file is an archived software project for SDK. This file can be imported into SDK to compile your own ELF.

Table 5: **File and Directory Structure Description *(Cont'd)***

Directory and Files <> Refers to a Directory	Description
<prebuilt_image>	This directory contains bit files for the AC701, KC705, KCU105, and VC709. For the ZC706, an MCS image is provided. All images are ready to execute an Eye Scan.
<prebuilt_image>/source_eyescan.tcl	This Tcl Script executes an Eye Scan for the targeted board.

Conclusion

Extracting an Eye Scan from a link allows you to measure an accurate amount of margin in a real system. A predetermined pattern is not necessary and running an Eye Scan is completely non-destructive, allowing Eye Scan to be added into any transceiver link. By utilizing the files provided in the reference design, including the logic to extract the Eye Scan is very simple to implement. The reference design files leverage the ease of use of the IP integrator tool as well as the plug and play functionality of AXI. Such ease of use allows you to quickly add Eye Scan as a new feature for your product.

References

This document uses the following references

1. *Eye Scan with MicroBlaze MCS* (XAPP743)
2. PCI-SIG® Specifications
3. *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* (XAPP1022)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
6. *7 Series FPGAs GTP Transceivers User Guide* (UG482)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
11/19/2014	1.1	• Added UltraScale support.
01/06/2014	1.0	• Updated DRP Address x3 in Table 1: Example of Address Correlation between AXI and DRP. • Updated eye_scan_software in step 18. • Updated description in Table 5: File and Directory Structure Description.
12/18/2013	1.0	Initial Xilinx release.

Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.