Advanced Concepts with Xilinx® SDSoC™ – Streaming I/O

Tools:	2017.4 Vivado & SDSoC
Training Version:	v1
Date:	19 April 2018

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Objectives

- Learn how to create a SDSoC platform that supports streaming input data
- Create a SDSoC application that accesses the input data stream

Overview

This lab demonstrates the steps needed to create a SDSoC platform and application that processes streaming data received from an external source. A programmable counter located in the programmable logic (PL) will be used to model the external source. In reality, the external source may be an analog-to-digital converter (ADC) or some other off-chip device, but for simplicity we will model the external source with a programmable counter. The counter is programmed over an AXI4™-Lite interface by the software portion of our SDSoC application running in the Zynq® processing system (PS). The counter output feeds an AXI4-Stream data FIFO which is accessed by our SDSoC application to move data from PL to PS.

This lab is split into three experiments. In the first experiment we will develop a SDSoC platform capable of supporting streaming data. The second experiment creates and builds the SDSoC application. The third experiment executes the SDSoC application on the MiniZed™. The first experiment can be skipped if desired by using the pre-built SDSoC MiniZed platform found in the supporting documents folder accompanied with this lab.

Lab Setup

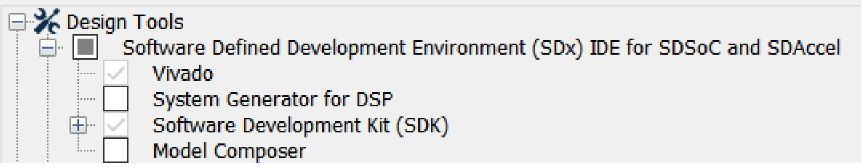
This experiment will build upon work done in “A Practical Guide to Getting Started with Xilinx SDSoC”, which can be downloaded from http://zedboard.org/support/trainings-and-videos. It is not necessary to complete “A Practical Guide to Getting Started with Xilinx SDSoC”, but it is highly recommended.

Setting Up For the Lab

This document is written for use with Windows and there are references throughout to paths starting with C:/. If you wish to complete this lab with a different operating system please convert the file paths appropriately.

1. Install version 2017.4 of the Xilinx SDx tool suite

   https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/sdx-development-environments.html

   ⊟ Design Tools
   ⊟ ■ Software Defined Development Environment (SDx) IDE for SDSoC and SDAccel
   ✓ Vivado
   □ System Generator for DSP
   ⊞ ✓ Software Development Kit (SDK)
   □ Model Composer

2. Become familiar with SDSoC and Vivado.

3. Create a directory on your C-drive named *training*. Extract the contents of the supporting documents zip-file (*support.zip*) to *C:/training*.

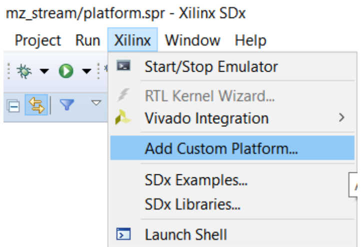
4. If not already done, copy the MiniZed board definition to your *<Xilinx SDx install directory>/Vivado/2017.4/data/boards/board_files* directory. The MiniZed board definition files can be found in the *C:/training/support/board_def* folder or downloaded from http://zedboard.org/sites/default/files/documentations/MiniZed_Board_Definition_File_0.zip.

5. Install a terminal program such as PuTTY or Tera Term.

Experiment 1: Create the Platform

The first experiment in this lab creates the MiniZed hardware platform containing the programmable counter and AXI4-Stream data FIFO. The programmable counter models an external data source and writes data to the AXI4-Stream data FIFO. The AXI4-Stream data FIFO is read by the SDSoC application which will be created in Experiment 2.

Note: A completed SDSoC platform is located in the support folder under a sub-folder named platforms. If you wish to skip SDSoC platform generation then add the completed platform to your repository (SDSoC GUI menu Xilinx → Add Custom Platform) and go to Experiment 2. More detail on adding the custom platform to your repository is given in Experiment 2.

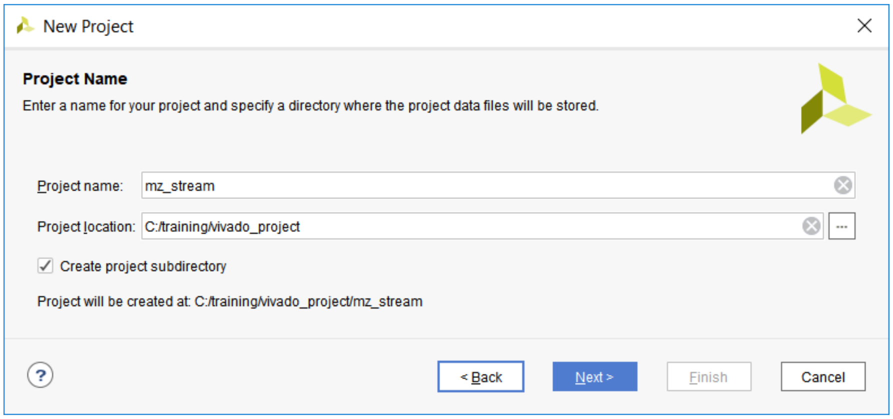


Hardware Platform

The following steps demonstrate creation of the hardware platform definition in Vivado

1. Open Vivado 2017.4

2. Create a new project named **mz_stream** in *C:/training/vivado_project* then click on **Next**



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: mz_stream

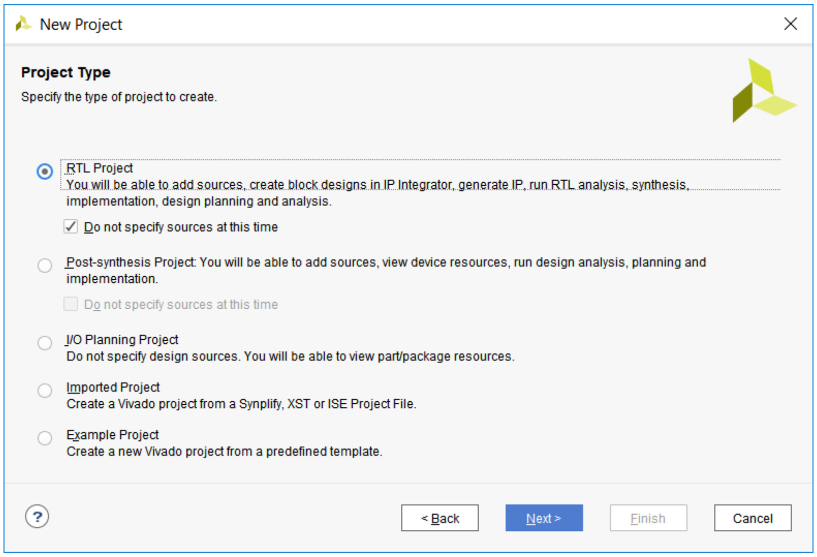
Project location: C:/training/vivado_project

☑ Create project subdirectory

Project will be created at: C:/training/vivado_project/mz_stream

Figure 1 - New Vivado Project

3. Select the **RTL Project** option and make sure the **Do not specify sources at this time** box is checked and then click **Next**



New Project

Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☑ Do not specify sources at this time

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

□ Do not specify sources at this time

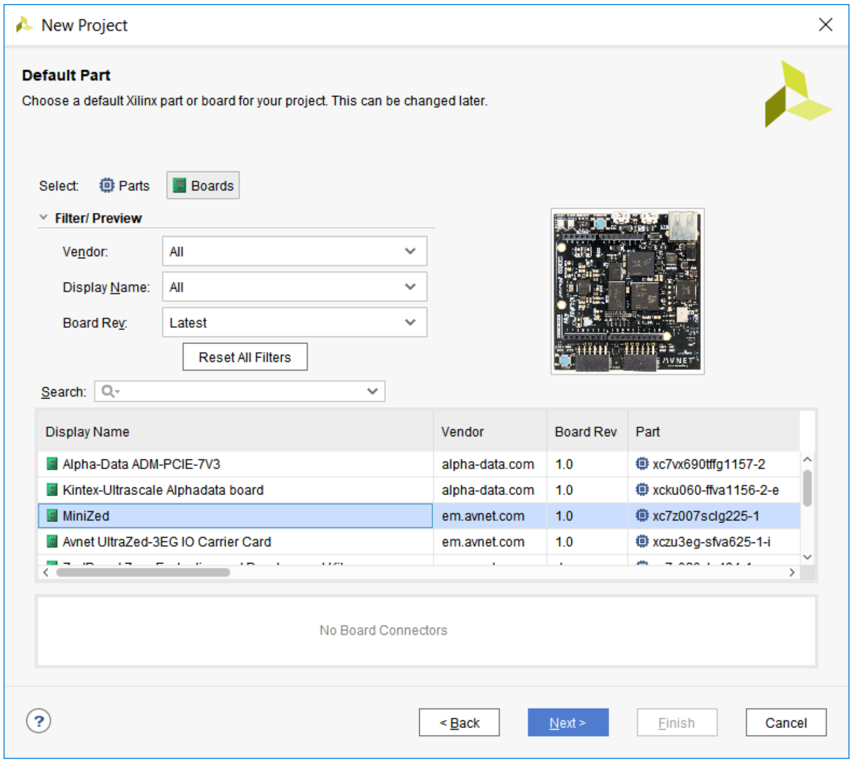
○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

Figure 2 - Vivado Project Type

4. Click on **Boards** then select the **MiniZed** option and click **Next**



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter/ Preview

Vendor: All

Display Name: All

Board Rev: Latest

Search:

Display Name	Vendor	Board Rev	Part
Alpha-Data ADM-PCIE-7V3	alpha-data.com	1.0	xc7vx690tffg1157-2
Kintex-Ultrascale Alphadata board	alpha-data.com	1.0	xcku060-ffva1156-2-e
MiniZed	em.avnet.com	1.0	xc7z007sclg225-1
Avnet UltraZed-3EG IO Carrier Card	em.avnet.com	1.0	xczu3eg-sfva625-1-i

No Board Connectors

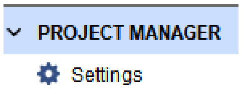
Figure 3 - Part Selection

5. Click **Finish** on the New Project Summary window

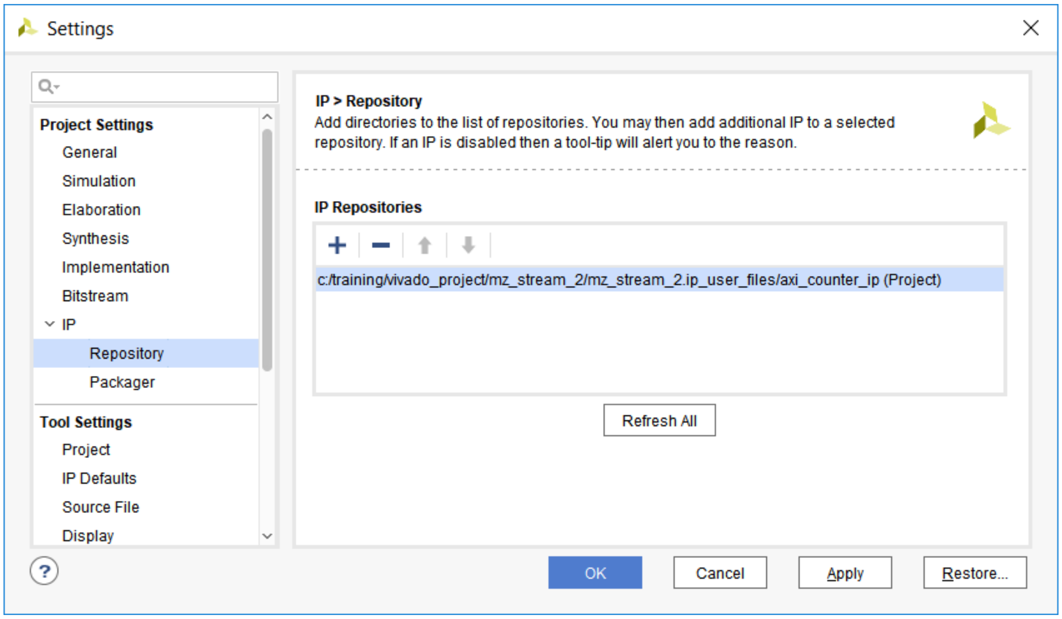
6. Extract the *axi_counter_ip.zip* file located in the support folder to your Vivado project directory

   C:*\training\vivado_project\mz_stream\mz_stream.ip_user_files\*

   In Vivado, click on **Settings** under **PROJECT MANAGER** in the Flow Navigator window



7. Select **IP** (click on the rotated chevron symbol left of IP to expand) → **Repository** and add the **axi_counter_ip** directory from step 6 then click **OK**



IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

c:/training/vivado_project/mz_stream_2/mz_stream_2.ip_user_files/axi_counter_ip (Project)

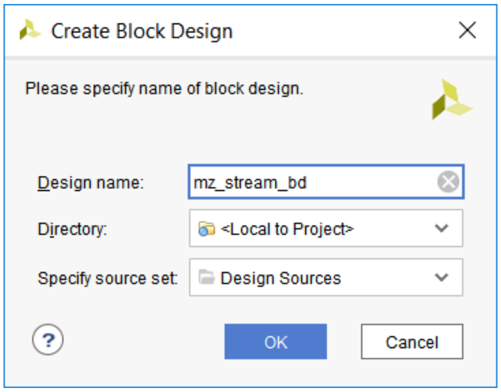
Figure 4 - Adding the IP Repository

Note: SDSoC requires all project files, including IP, to be local to the Vivado project. This is why the AXI Counter IP was copied to the ip_user_files directory in step 6.

Note: Steps 8 through 22 demonstrate how to create the block design for the streaming I/O platform. The process is lengthy and somewhat cumbersome.

You can create the block design automatically by executing the create_mz_stream_bd.tcl script from the supporting documents folder. <u>If you use this script jump to step 23</u>.

8. Create an IP Integrator Block Design named **mz_stream_bd** by clicking on the **Create Block Design** option under **IP Integrator** in **Flow Navigator**



Create Block Design

Please specify name of block design.

Design name: mz_stream_bd

Directory: <Local to Project>

Specify source set: Design Sources

Figure 5 - Create the Block Design

9. Add the following components to the IP Integrator (IPI) canvas

   a. ZYNQ7 Processing System – quantity 1

   b. Processor System Reset – quantity 3

   c. Clocking Wizard – quantity 1

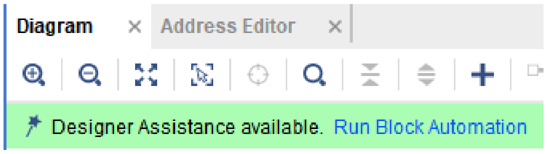
   d. Concat – quantity 1

   e. axi_counter_ip – quantity 1

   f. AXI4-Stream Data FIFO – quantity 1

   g. AXI SmartConnect – quantity 1

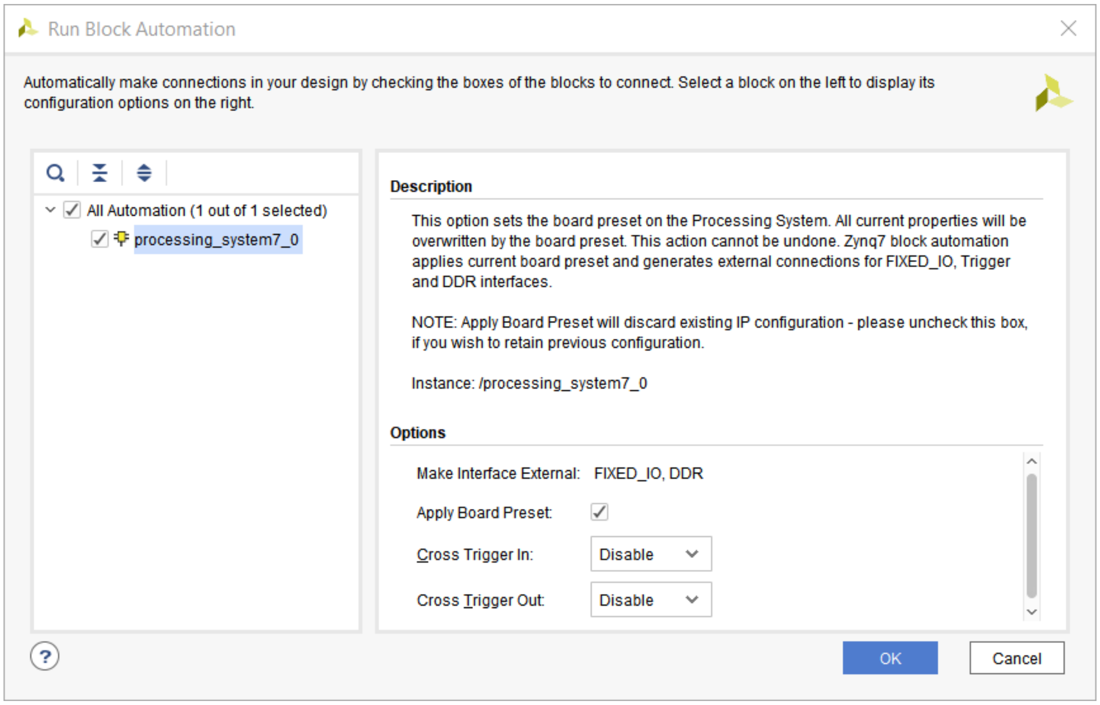
10. Click on **Run Block Automation** in the Designer Assistance Banner to apply MiniZed board presets



Designer Assistance available. Run Block Automation

Figure 6 – Block Automation

11. Click **OK** on the pop-up window



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

☑ All Automation (1 out of 1 selected)
  ☑ processing_system7_0

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ☑

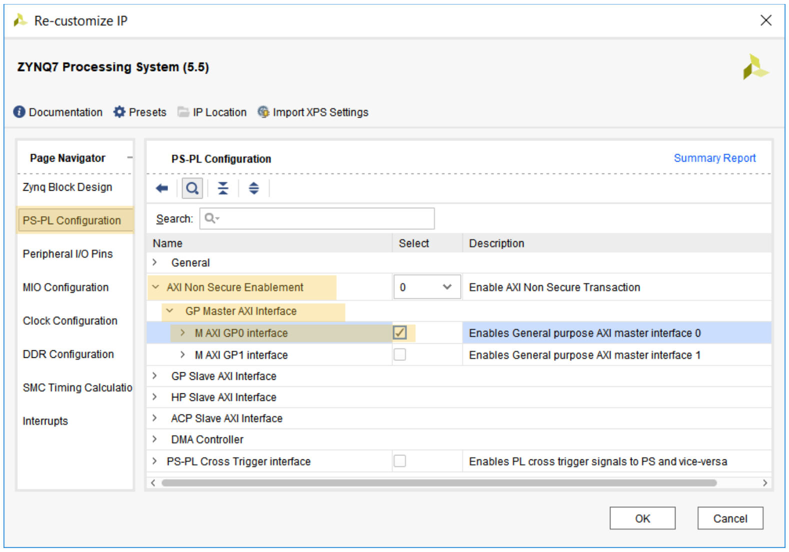
Cross Trigger In: Disable

Cross Trigger Out: Disable

Figure 7 - Block Automation Pop-up

12. **Double-click** on the **ZYNQ7 Processing System** block and customize it as follows

    a. We need to enable the M_AXI_GP0 interface which will be used to write memory-mapped registers within the AXI Counter IP. The M_AXI_GP0 interface is enabled by navigating to **PS-PL Configuration → AXI Non Secure Enablement → GP Master AXI Interface → M AXI GP0 interface** (shown in the figure below).



Re-customize IP

ZYNQ7 Processing System (5.5)

Name	Select	Description
› General		
⌄ AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
⌄ GP Master AXI Interface		
› M AXI GP0 interface	☑	Enables General purpose AXI master interface 0
› M AXI GP1 interface	☐	Enables General purpose AXI master interface 1
› GP Slave AXI Interface		
› HP Slave AXI Interface		
› ACP Slave AXI Interface		
› DMA Controller		
› PS-PL Cross Trigger interface	☐	Enables PL cross trigger signals to PS and vice-versa

Figure 8 – Enabling M_AXI_GP0 PL Interface

b. There is an issue with using UART0 on the MiniZed so we must disable it in order to force UART1 to be used. To disable UART0 select **Peripheral I/O Pins** and deselect **UART 0**

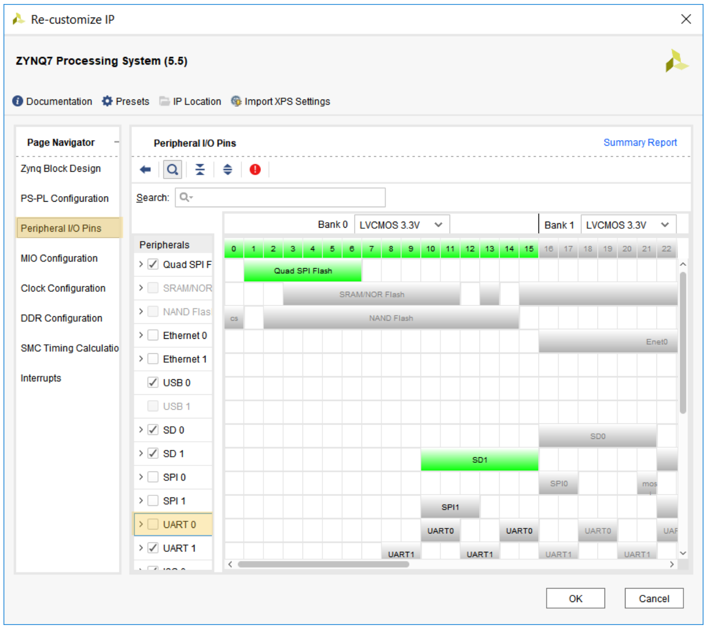
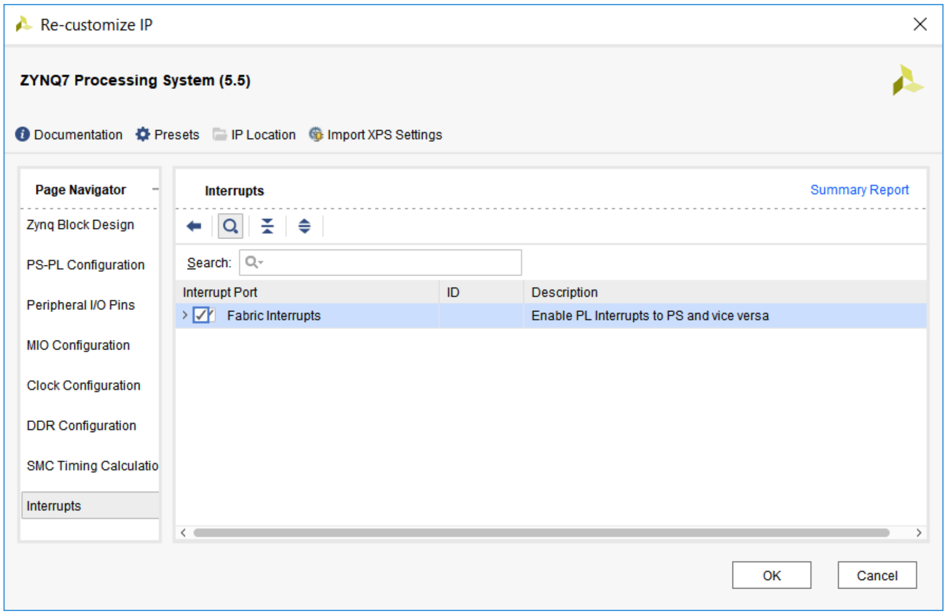


Figure 9 – Disabling UART 0

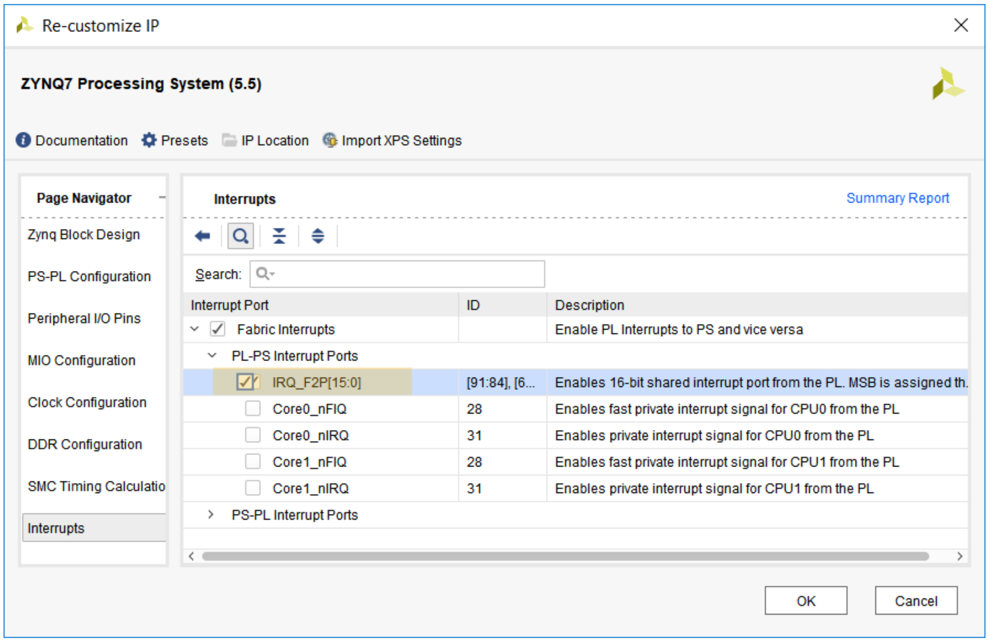
c. Per UG1146 we must enable fabric interrupts and connect a Concat block to the PS interrupt port. To enable interrupts select **Interrupts** and enable **Fabric Interrupts**.



Interrupt Port	ID	Description
☑ Fabric Interrupts		Enable PL Interrupts to PS and vice versa

Figure 10 - Fabric Interrupts

d. Next enable **PL-PS Interrupt Ports** by expanding Fabric Interrupts and selecting **IRQ_F2P[15:0]**



Re-customize IP

ZYNQ7 Processing System (5.5)

Interrupts

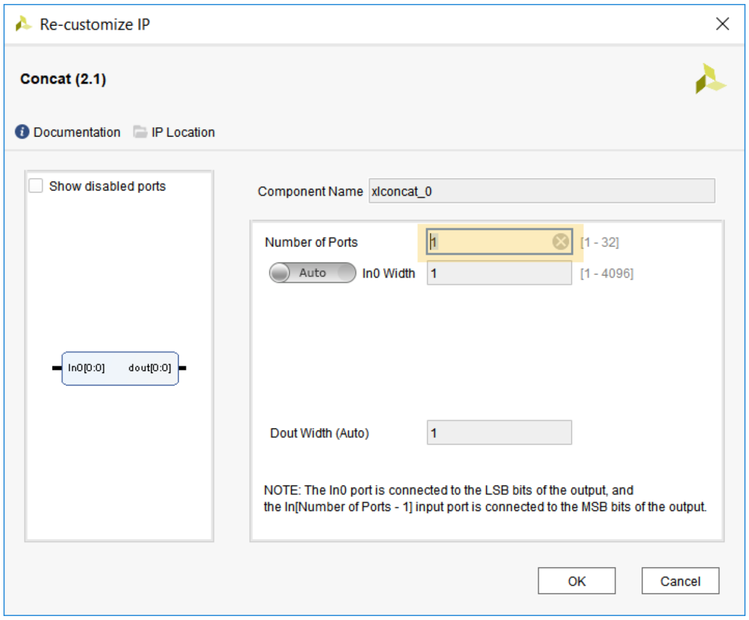
Search:

Interrupt Port	ID	Description
☑ Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
☑ IRQ_F2P[15:0]	[91:84], [6...	Enables 16-bit shared interrupt port from the PL. MSB is assigned th.
□ Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
□ Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
□ Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
□ Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
PS-PL Interrupt Ports		

Figure 11 - IRQ_F2P[15:0]

e. Click **OK**

13. Double-click on the **Concat** block and customize it with **1 input port** then click **OK**



Re-customize IP

Concat (2.1)

Component Name: xlconcat_0

Number of Ports: 1 [1 - 32]

In0 Width: 1 [1 - 4096]

Dout Width (Auto): 1

NOTE: The In0 port is connected to the LSB bits of the output, and the In[Number of Ports - 1] input port is connected to the MSB bits of the output.

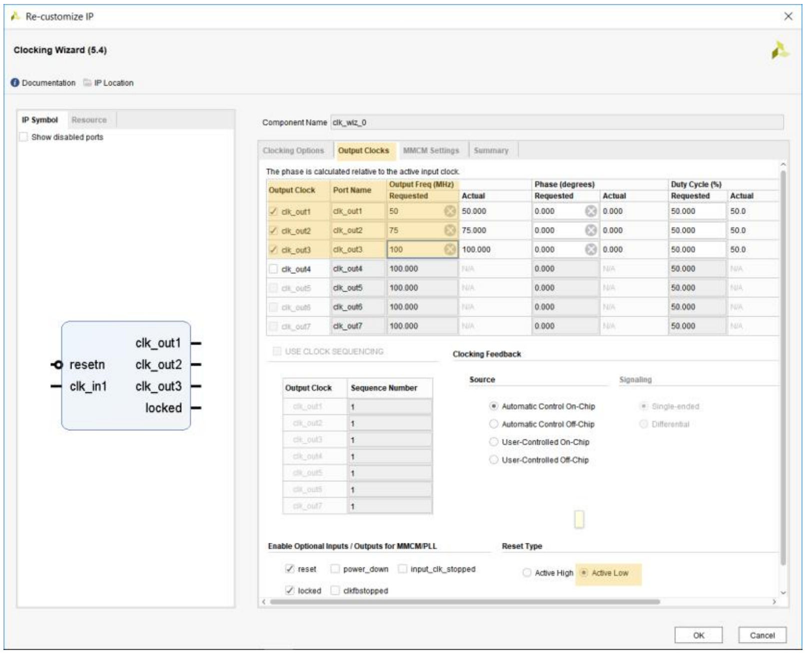
Figure 12 - Concat Block with 1 Input Port

14. For our platform we are adding 3 PL clocks using the Clocking Wizard. We could use clocks generated by the PS, but using the Clocking Wizard allows for more flexibility during routing by Vivado. **Double-click** on the **Clocking Wizard** block and customize it with 3 output clocks

    a. Click on the **Output Clocks tab** and set the following output clock frequencies and reset option

        i. **clk_out1** = **50 MHz, clk_out2** = **75 MHz, clk_out3** = **100 MHz**

        ii. **Reset Type → Active Low**



Component Name clk_wiz_0

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz) Requested	Actual	Phase (degrees) Requested	Actual	Duty Cycle (%) Requested	Actual
☑ clk_out1	clk_out1	50	50.000	0.000	0.000	50.000	50.0
☑ clk_out2	clk_out2	75	75.000	0.000	0.000	50.000	50.0
☑ clk_out3	clk_out3	100	100.000	0.000	0.000	50.000	50.0
□ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A

□ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source

● Automatic Control On-Chip
○ Automatic Control Off-Chip
○ User-Controlled On-Chip
○ User-Controlled Off-Chip

Signaling

● Single-ended
○ Differential

Enable Optional Inputs / Outputs for MMCM/PLL

☑ reset □ power_down □ input_clk_stopped
☑ locked □ clkfbstopped

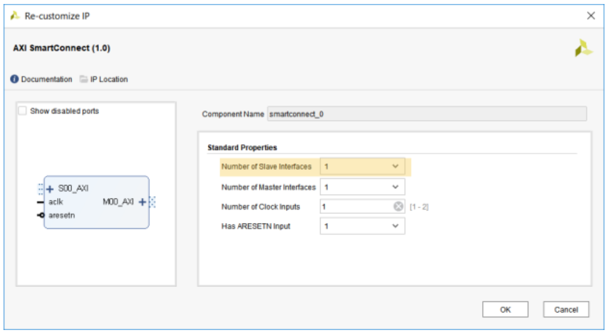
Reset Type

○ Active High ● Active Low

Figure 13 - Clocking Wizard

    b. Click **OK**

15. **Double-click** on the **AXI SmartConnect** block and customize it with **1 AXI Slave interface** then click **OK**. The AXI SmartConnect block is used to handle port differences between the AXI interface on the PS and the AXI Counter IP.



Component Name smartconnect_0

Standard Properties

Number of Slave Interfaces 1
Number of Master Interfaces 1
Number of Clock Inputs 1 [1 - 2]
Has ARESETN Input 1

Figure 14 - AXI SmartConnect with 1 AXI Slave Interface

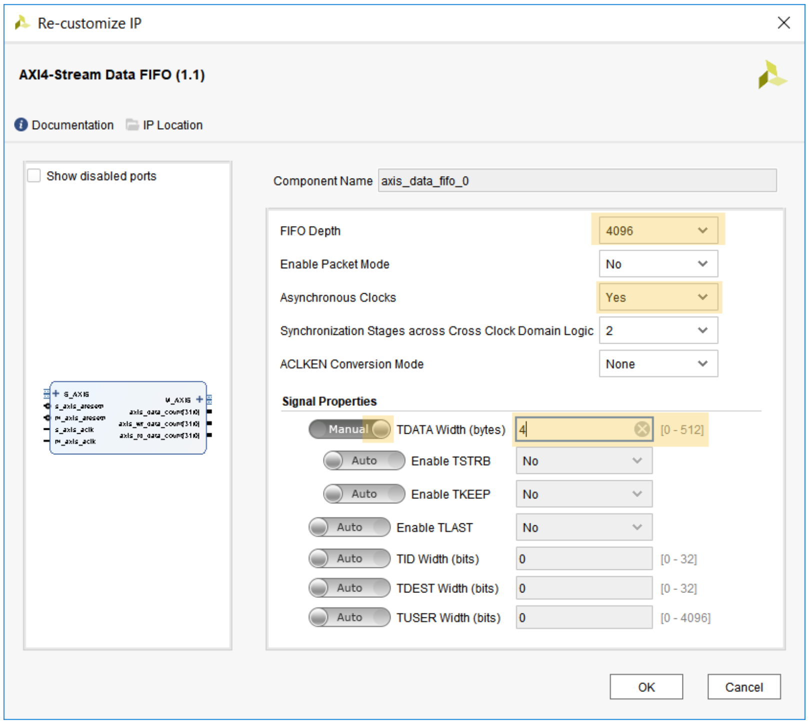
16. Double-click on the **AXI4-Stream Data FIFO** and set the following parameters

    a. **FIFO Depth** to **4096**

    b. **Asynchronous Clocks** to **Yes**

    c. **TDATA Width** (bytes) to **4** – Note: You need to change the toggle from Auto to Manual

    d. Click **OK**



□ Show disabled ports

Component Name axis_data_fifo_0

FIFO Depth 4096

Enable Packet Mode No

Asynchronous Clocks Yes

Synchronization Stages across Cross Clock Domain Logic 2

ACLKEN Conversion Mode None

Signal Properties

Manual	TDATA Width (bytes)	4	[0 - 512]
Auto	Enable TSTRB	No	
Auto	Enable TKEEP	No	
Auto	Enable TLAST	No	
Auto	TID Width (bits)	0	[0 - 32]
Auto	TDEST Width (bits)	0	[0 - 32]
Auto	TUSER Width (bits)	0	[0 - 4096]

Figure 15 - AXI4-Stream Data FIFO Customization

17. We need to connect all of our components now that they are customized. Make the following connections (components shown in **bold** font, ports shown in *green italicized* font)

    a. **xlconcat_0** *dout* port to **processing_system7_0** *IRQ_F2P* port

    b. **processing_system7_0** *FCLK_RESET0_N* port to

        i. **proc_sys_reset_0** *ext_reset_in* port

        ii. **proc_sys_reset_1** *ext_reset_in* port

        iii. **proc_sys_reset_2** *ext_reset_in* port

        iv. **clk_wiz_0** *resetn* port

    c. **processing_system7_0** *FCLK0* port to **clk_wiz_0** *clk_in_1* port

    d. **clk_wiz_0** *clk_out_1* port to **proc_sys_reset_0** *slowest_sync_clk* port

e. **clk_wiz_0** *clk_out_2* port to

   i. **proc_sys_reset_1** *slowest_sync_clk* port

   ii. **processing_system7_0** *M_AXI_GP0_ACLK* port

   iii. **smartconnect_0** *aclk* port

   iv. **axis_data_fifo_0** *s_axis_aclk* port

   v. **axi_counter_ip_0** *clk* and *s_axi_aclk* ports

f. **clk_wiz_0** *clk_out_3* port to

   i. **proc_sys_reset_2** *slowest_sync_clk* port

   ii. **axis_data_fifo_0** *m_axis_aclk* port

g. **clk_wiz_0** *locked* port to

   i. **proc_sys_reset_0** *dcm_locked* port

   ii. **proc_sys_reset_1** *dcm_locked* port

   iii. **proc_sys_reset_2** *dcm_locked* port

h. **proc_sys_reset_1** *peripheral_aresetn* port to

   i. **axi_counter_ip** *rst_n* and *s_axi_aresetn* ports

   ii. **axis_data_fifo_0** *s_axis_aresetn* port

   iii. **smartconnect_0** *aresetn* port

i. **proc_sys_reset_2** *peripheral_aresetn* port to **axis_data_fifo_0** *m_axis_aresetn*

j. **Expand** the **S_AXIS** interface on the **axis_data_fifo_0** component to expose the *s_axis_tdata* and *s_axis_tvalid* ports. Make the following connections

   i. **axi_counter_ip** *counter[31:0]* port to **axis_data_fifo_0** *s_axis_tdata[31:0]*

   ii. **axi_counter_ip** *counter_valid* port to **axis_data_fifo_0** *s_axis_tvalid* port

k. **axi_counter_ip_0** *s_axi* interface to **smartconnect_0** *M00_AXI* interface

l. **smartconnect_0** *S00_AXI* interface **to processing_system7_0** *M_AXI_GP0* interface

18. Regenerate the block design layout by clicking on ↻ at the top of the IPI canvas. The block diagram should look like Figure 16.

    Note: the M_AXIS interface of the axis_data_fifo_0 component is left unconnected in the block design. We will define this interface for use by the SDSoC application.

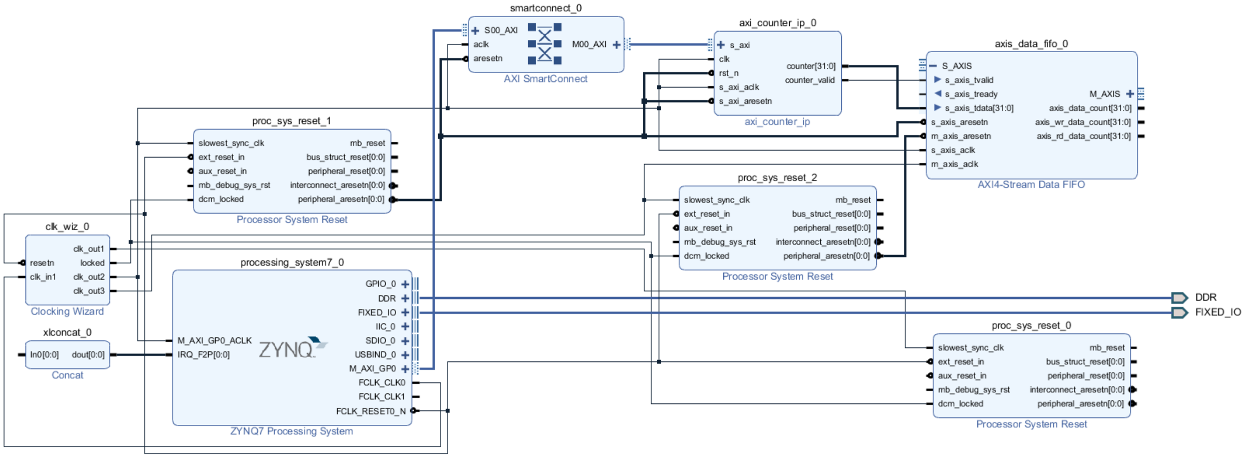
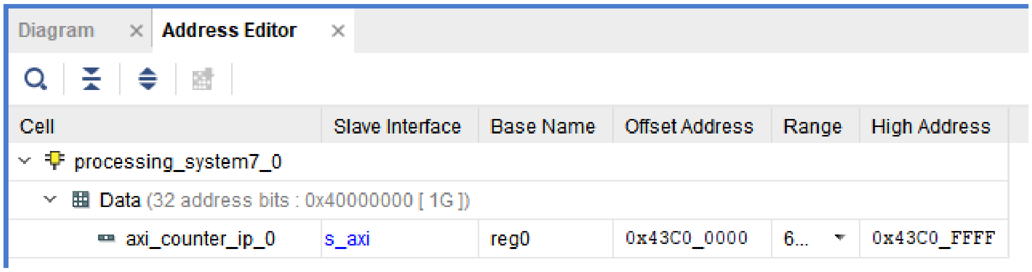


Figure 16 – Final Hardware Platform IPI Block Diagram

19. Click on **Address Editor** tab, if the **Address Editor** tab does not appear in your **BLOCK DESIGN** window then Navigate to **Window → Address Editor** using the Vivado menu

    a. Click on the **Auto Assign Address** button ▦ then **OK** at the Auto Assign Address pop-up

    b. Notice that an address offset of **0x43C0_0000** was set for the **axi_counter_ip_0** component. If it was not set to that value you will need to change it or remember to change the SDSoC application source code (*counter_control.h*) to use the address assigned by Vivado.

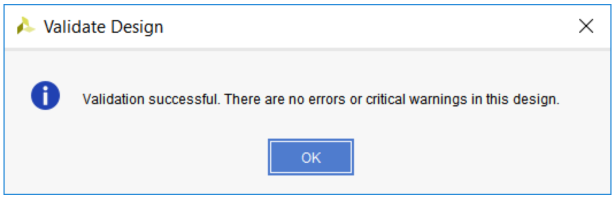


Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_counter_ip_0	s_axi	reg0	0x43C0_0000	6...	0x43C0_FFFF

Figure 17 - Address Editor

20. Click on the **Diagram** tab and **validate** the block design by clicking on ☑

21. You should get a pop-up that says **"Validation Successful. There are no errors or critical warnings in this design."** If you don't get this message then fix the issues and revalidate. Click **OK** to dismiss the pop-up.



Validation successful. There are no errors or critical warnings in this design.

Figure 18 - Validation Successful

22. **Save** the block diagram

23. Our block design is now complete and we are ready to generate output products. In order to proceed we need to create a HDL wrapper for the block design and set it as the top level design file. Vivado can automatically create a wrapper and set it as the top file. In the **Sources** window right-click on the **mz_stream_bd** design source and select **Create HDL Wrapper**

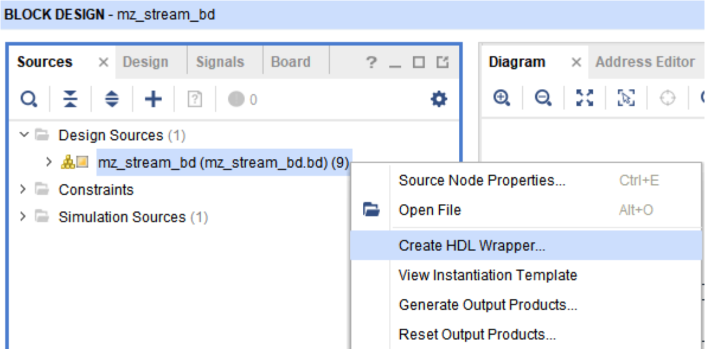
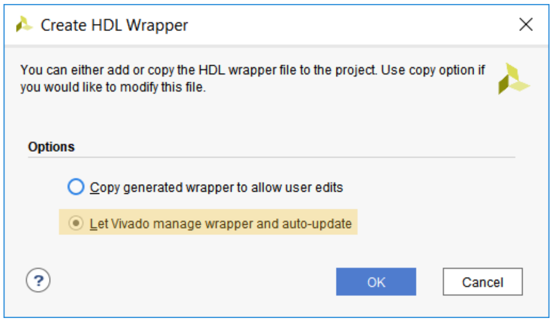


Figure 19 - Create HDL Wrapper

a. At the pop-up leave the **Let Vivado manage wrapper and auto-update** radio button selected and click **OK**



You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

○ Copy generated wrapper to allow user edits

● Let Vivado manage wrapper and auto-update

Figure 20 - Create HDL Wrapper Dialog

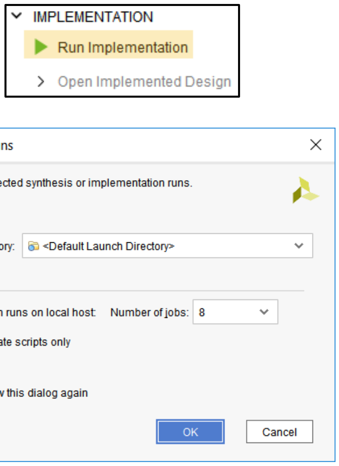
24. Next we need to define the interfaces that our SDSoC application can access. The Tcl commands used to define these interfaces are described in detail in UG1146 if you are interested in exploring. To save time, execute the **mz_stream_pfm.tcl** script from the **support** directory which already defines the SDSoC interfaces.

Note: It is assumed that you have followed the naming conventions specified in this document. If that is not the case then you will need to update the Tcl script appropriately.

In the Tcl Console execute

source C:/training/support/mz_stream_pfm.tcl

25. **Save** the block diagram and **Run Implementation**. At the pop-up click **OK** to launch the runs.



Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

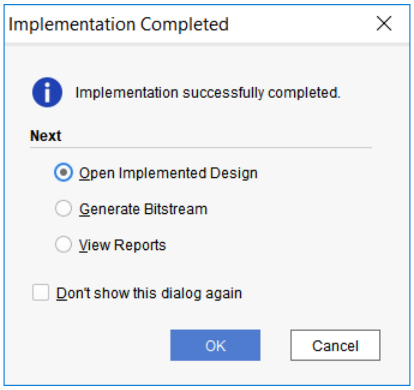
● Launch runs on local host: Number of jobs: 8

○ Generate scripts only

□ Don't show this dialog again

Figure 21 – Run Implementation

26. When implementation is complete **open the implemented** design



Implementation successfully completed.

Next

● Open Implemented Design

○ Generate Bitstream

○ View Reports

□ Don't show this dialog again

Figure 22 – Open Implemented Design

27. Next we will create the Device Support Archive (DSA) which is used by the SDSoC tool to understand the hardware platform

    a. In the **Tcl Console** change directories to the Vivado project directory
    cd C:/training/vivado_project/mz_stream

    b. Execute
    write_dsa –force mz_stream.dsa

    c. Execute
    validate_dsa mz_stream.dsa

28. **Close** the **Vivado** project and exit Vivado

Software Platform

The software platform definition is created automatically by SDSoC in this lab using the SDSoC "Build software platform components" option. The SDSoC Platform section below describes how to use this option.

SDSoC Platform

The SDSoC platform generation procedure in this experiment is similar to "A Practical Guide to Getting Started with Xilinx SDSoC". The only exception is that we will be using a new Device Support Archive file.

For this experiment we will create a new workspace that will contain the SDSoC platform and the SDSoC application.

1. Launch the SDx GUI

2. In the "Select a directory as workspace" dialog set the workspace to **C:\training\SDx_wksp_mz_strm** and click **OK**

3. Close the Welcome window if it appears

4. In the Project Explorer pane, **right-click** and select **New → SDx Project**

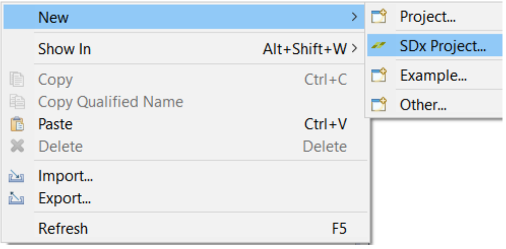
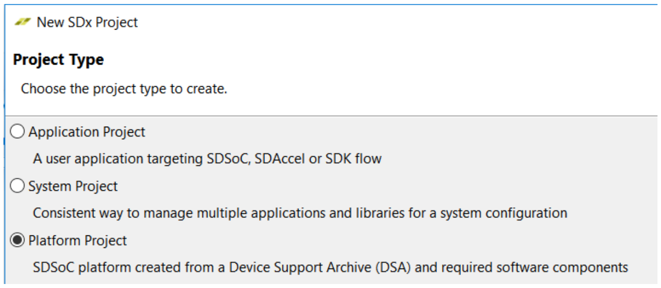


Figure 23 – New SDx Project

5. Select **Platform Project** and click **Next**



New SDx Project

Project Type

Choose the project type to create.

○ Application Project
A user application targeting SDSoC, SDAccel or SDK flow

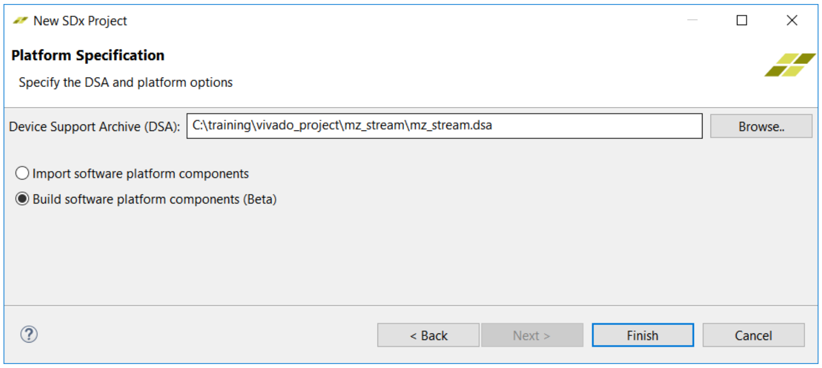
○ System Project
Consistent way to manage multiple applications and libraries for a system configuration

● Platform Project
SDSoC platform created from a Device Support Archive (DSA) and required software components

Figure 24 – Platform Project

6. Select the DSA file that was generated in step 27 of the Hardware Platform section.

7. Select the **Build software platform components (Beta)** option and click **Finish**.



New SDx Project

Platform Specification

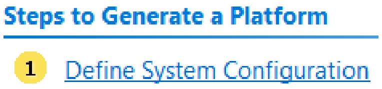
Specify the DSA and platform options

Device Support Archive (DSA): C:\training\vivado_project\mz_stream\mz_stream.dsa

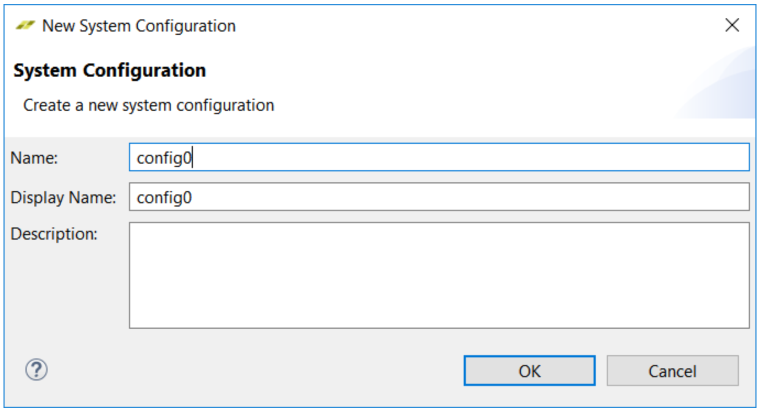
○ Import software platform components

● Build software platform components (Beta)

Figure 25 – Platform Specification

8. In the “Platform: mz_stream” window **click** on 

9. Fill out the dialog box with the **information shown in the figure below** and click **OK**.



New System Configuration

System Configuration

Create a new system configuration

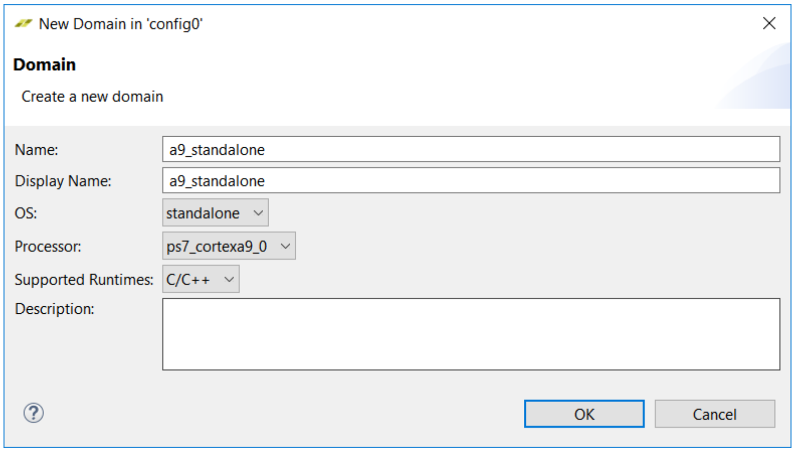
Name: config0

Display Name: config0

Description:

Figure 26 - SDSoC Platform System Configuration

10. **Click** on 2 Add Processor Group/Domain and enter the **information shown in the figure below** then click **OK**



New Domain in 'config0'

Domain

Create a new domain

Name: a9_standalone

Display Name: a9_standalone

OS: standalone

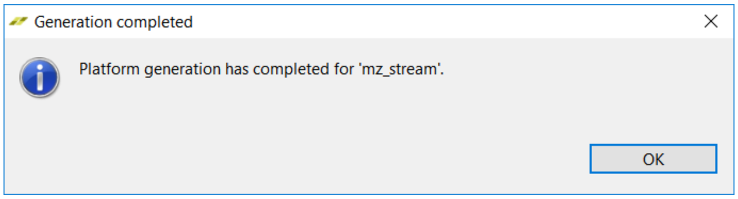
Processor: ps7_cortexa9_0

Supported Runtimes: C/C++

Description:

Figure 27 - SDSoC Platform Domain

11. **Click** 3 Generate Platform and wait for platform generation to complete then click **OK** to dismiss the pop-up

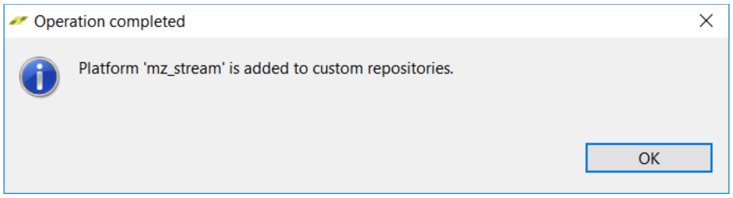


Generation completed

Platform generation has completed for 'mz_stream'.

Figure 28 – Platform Generation Completed

12. **Click** 4 Add to Custom Repositories to add the SDSoC platform to the platform repository for the current workspace then click **OK** to dismiss the pop-up



Operation completed

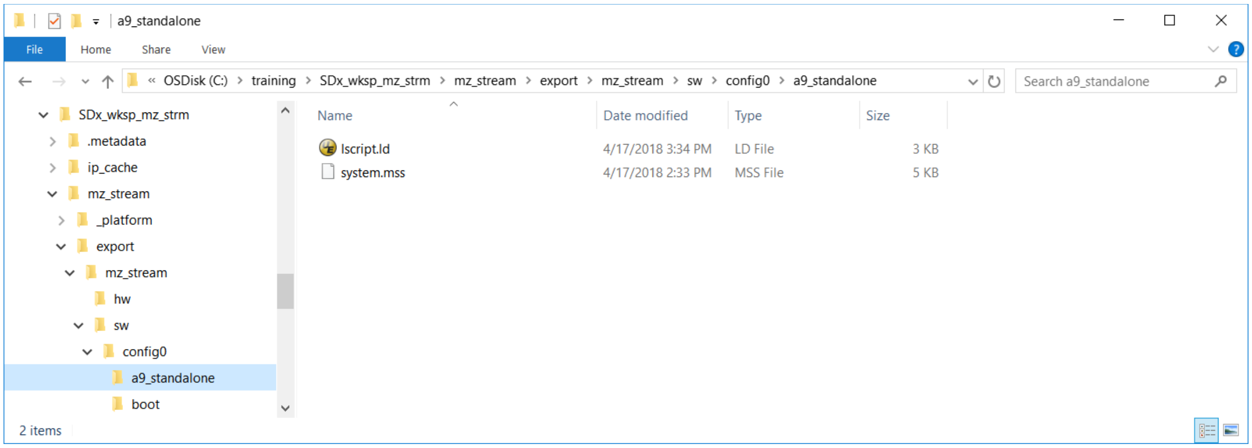
Platform 'mz_stream' is added to custom repositories.

Figure 29 – Platform Added to Custom Repositories

13. There is an issue with the linker script generated by SDSoC which defines a heap size that is too large for the DDR memory on the MiniZed, so we need to update the linker script with an acceptable heap size

    a. In **Windows Explorer** navigate to C:\training\SDx_wksp_mz_strm\mz_stream\export\mz_stream\sw\config0\a9_standalone

    b. Open **lscript.ld** with a text editor



Name	Date modified	Type	Size
lscript.ld	4/17/2018 3:34 PM	LD File	3 KB
system.mss	4/17/2018 2:33 PM	MSS File	5 KB

Figure 30 – Linker Script in Windows Explorer

    c. Search for the line containing **_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 0x30000000;**

    d. Modify the line with a heap size of **0x10000000** **_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 0x10000000;**

14. **Save** and **close** the linker script

15. SDSoC platform generation is complete

Discussion: Create the Platform

In Experiment 1 we created the SDSoC platform for the MiniZed that will allow the SDSoC application to access streaming input data. Figure 31 below shows the AXI4-Stream Data FIFO which temporarily stores the streaming input data. In Experiment 2 we will create the SDSoC application which will connect to the FIFO master AXI4-Stream interface to read the data from the input stream.

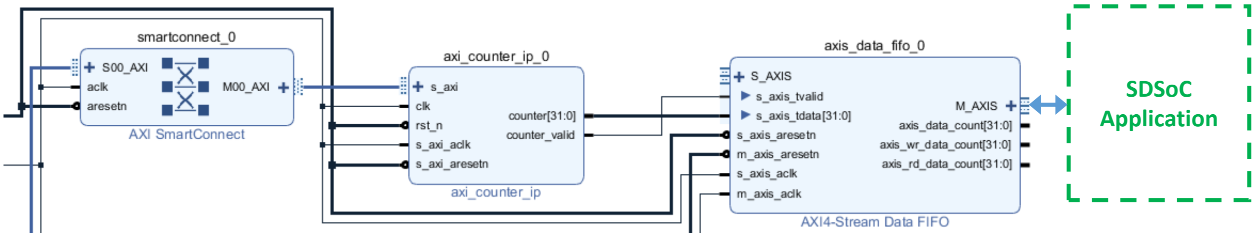


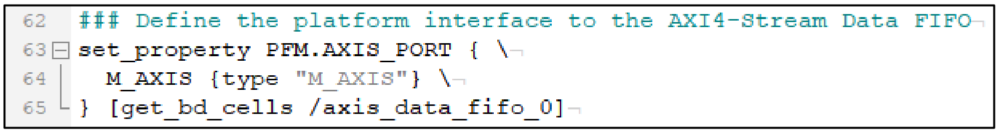
Figure 31 – SDSoC Platform and Application Connection

During platform creation we added a Clocking Wizard IP block instead of enabling more output clocks from the PS. There can be issues with closing timing when all clocks are sourced from the PS. In order to improve our timing results the Clocking Wizard was added which puts PL clock generation in the PL rather than the PS, and gives the tool more flexibility when routing the design.

The programmable counter operates at a maximum sample rate of 75 MHz. The highest clock available in the platform is 100 MHz which we will use in our application for the data movement clock. We want our data movement clock to be higher than the input data rate in order to be able to read the FIFO fast enough to prevent overflows.

The programming interface for the counter is an AXI4-Lite memory-mapped interface that is defined as part of the hardware platform. This method of platform definition/connection follows the traditional Vivado/SDK development flow. SDSoC supports writing data to memory-mapped addresses in the same way that is supported by SDK. This is very handy if we have a previous application that used the traditional flow, but want to update the application to use SDSoC. In that case we may not need to update the hardware or software platforms, we just need to create the SDSoC platform by following the steps outlined in the SDSoC Platform section of Experiment 1.

In step 24 of Experiment 1 we executed the mz_stream_pfm.tcl script which defines the platform interfaces available to the SDSoC application. The Tcl script defines PL clocks as well as AXI and interrupt ports that SDSoC uses to move data between PS and PL. A portion of the Tcl script is shown in Figure 32 below. The script excerpt shows the AXI4-Stream master port exported for use with SDSoC. It can be seen that the "M_AXIS" port of type "M_AXIS" from the axis_data_fifo_0 component is being exported for use by the SDSoC application. We will use the sys_port pragma in our SDSoC application to directly connect to the M_AXIS port of the axis_data_fifo_0 component.



```
### Define the platform interface to the AXI4-Stream Data FIFO
set_property PFM.AXIS_PORT { \
  M_AXIS {type "M_AXIS"} \
} [get_bd_cells /axis_data_fifo_0]
```

Figure 32 – AXI4-Stream Data FIFO Port Definition

Experiment 2: Create the Application

In this experiment we will create the SDSoC application which programs a PL counter by writing memory-mapped registers over an AXI4-Lite interface. The output of the counter feeds an AXI4-Stream Data FIFO which the SDSoC application accesses to get data from the PL.

1. If you skipped Experiment 1 perform a-c below. Proceed to step 2 below if you completed Experiment 1.

    a. Launch the SDx GUI

    b. In the “Select a directory as workspace” dialog set the workspace to **C:\training\SDx_wksp_mz_strm** and click **OK**

    c. Close the Welcome window if it appears

    d. In the SDx GUI menu navigate to **Xilinx → Add Custom Platform**

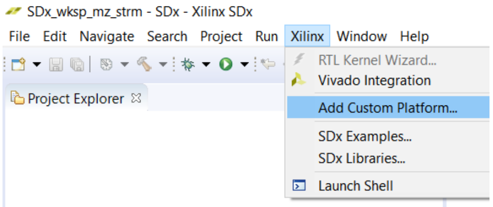


Figure 33 – Add Custom Platform

    e. **Click** on the **Add Custom Platform** button and then browse to C:\training\support\platforms\mz_stream and click **OK**

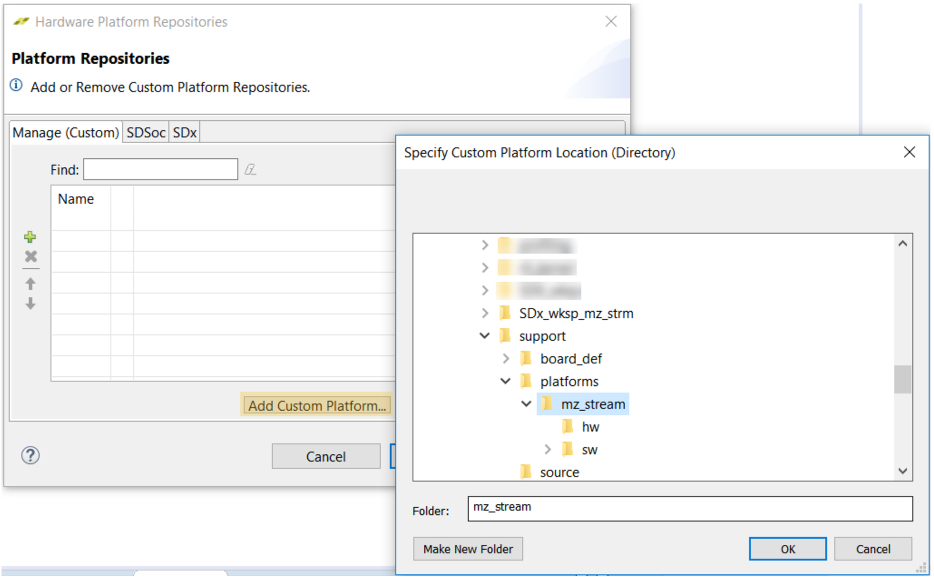
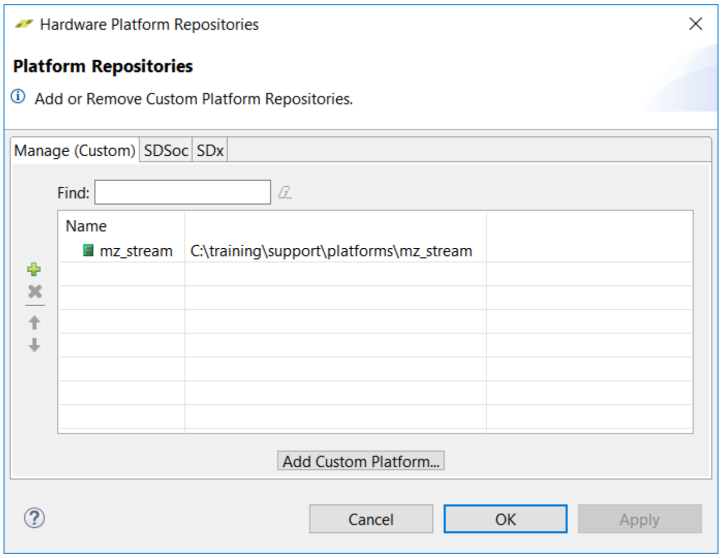


Figure 34 – Add Custom Platform

f. Finally, click **OK** on the Platform Repositories pop-up. You should see the mz_stream platform listed as shown in the figure below.



Hardware Platform Repositories

Platform Repositories

Add or Remove Custom Platform Repositories.

Find:

Name		
mz_stream	C:\training\support\platforms\mz_stream	

Figure 35 – mz_stream Platform Added to Repository

2. Create the application project

   a. In the **Project Explorer** pane, **right-click** and select **New → SDx Project**

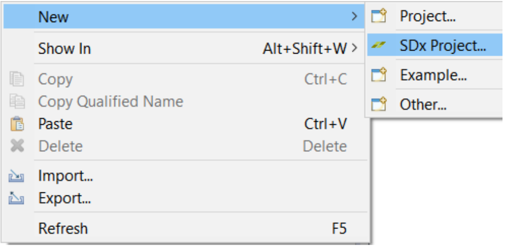
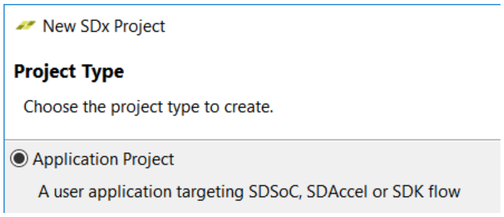


Figure 36 - New SDx Project

   b. Select **Application Project** and click **Next**



New SDx Project

Project Type

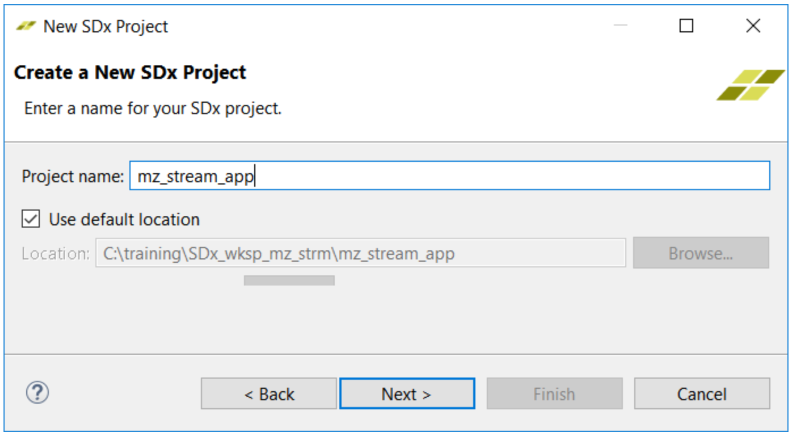
Choose the project type to create.

● Application Project

A user application targeting SDSoC, SDAccel or SDK flow

Figure 37 – Application Project

c. For the project name use **mz_stream_app** and click **Next**



New SDx Project

Create a New SDx Project

Enter a name for your SDx project.

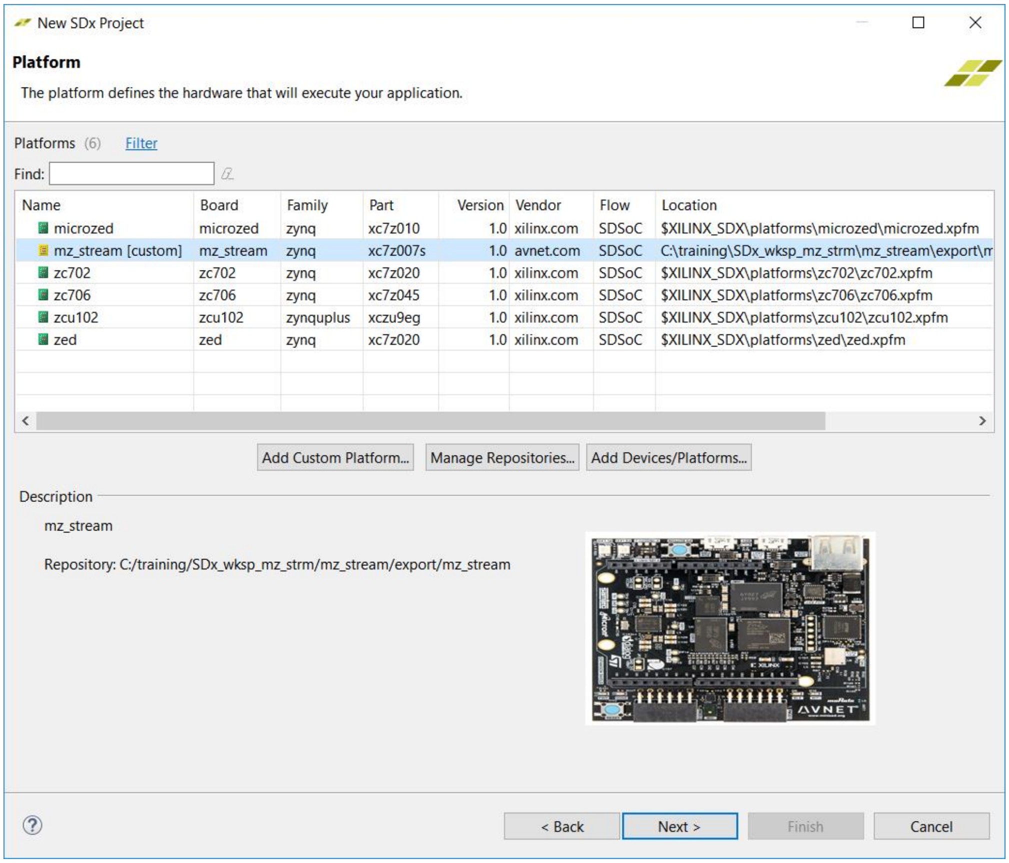
Project name: mz_stream_app

☑ Use default location

Location: C:\training\SDx_wksp_mz_strm\mz_stream_app

Figure 38 – MiniZed Streaming I/O Application

d. Select the **mz_stream [custom]** platform and click **Next**



New SDx Project

Platform

The platform defines the hardware that will execute your application.

Platforms (6) Filter

Find:

Name	Board	Family	Part	Version	Vendor	Flow	Location
microzed	microzed	zynq	xc7z010	1.0	xilinx.com	SDSoC	$XILINX_SDX\platforms\microzed\microzed.xpfm
mz_stream [custom]	mz_stream	zynq	xc7z007s	1.0	avnet.com	SDSoC	C:\training\SDx_wksp_mz_strm\mz_stream\export\m
zc702	zc702	zynq	xc7z020	1.0	xilinx.com	SDSoC	$XILINX_SDX\platforms\zc702\zc702.xpfm
zc706	zc706	zynq	xc7z045	1.0	xilinx.com	SDSoC	$XILINX_SDX\platforms\zc706\zc706.xpfm
zcu102	zcu102	zynquplus	xczu9eg	1.0	xilinx.com	SDSoC	$XILINX_SDX\platforms\zcu102\zcu102.xpfm
zed	zed	zynq	xc7z020	1.0	xilinx.com	SDSoC	$XILINX_SDX\platforms\zed\zed.xpfm

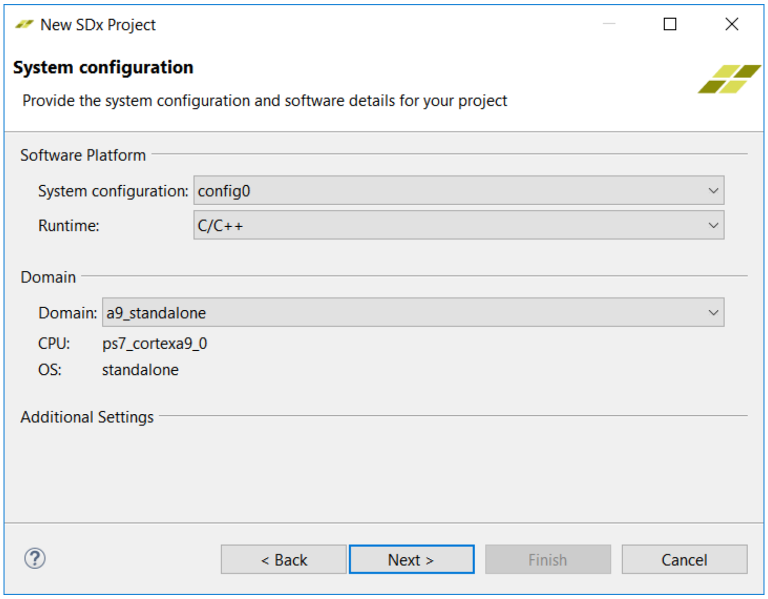
Description

mz_stream

Repository: C:/training/SDx_wksp_mz_strm/mz_stream/export/mz_stream

Figure 39 – Platform Selection

e. On the System configuration window click **Next**



New SDx Project

System configuration

Provide the system configuration and software details for your project

Software Platform

System configuration: config0

Runtime: C/C++

Domain

Domain: a9_standalone

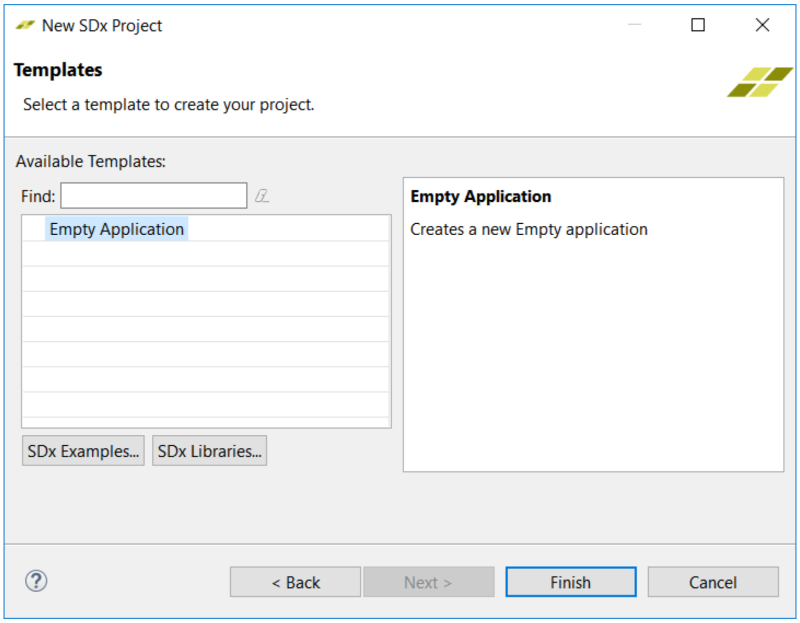
CPU: ps7_cortexa9_0

OS: standalone

Additional Settings

Figure 40 – System Configuration

f. In the Templates window select **Empty Application** and click **Finish**



New SDx Project

Templates

Select a template to create your project.

Available Templates:

Find:

Empty Application

Empty Application

Creates a new Empty application

Figure 41 – Empty Application

3. Import the project source code
    a. Expand the project in the Project Explorer pane

    b. **Right-click** on the **src** directory and select **Import**

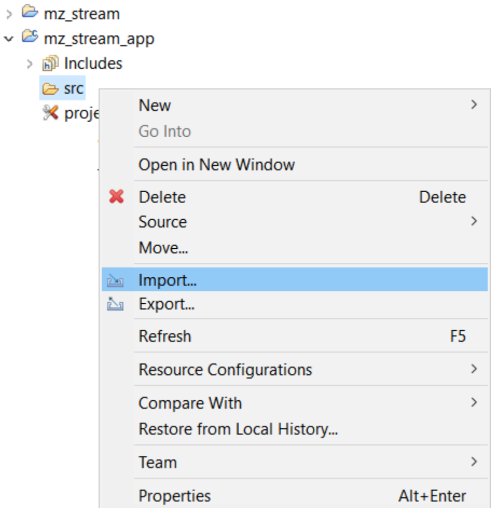
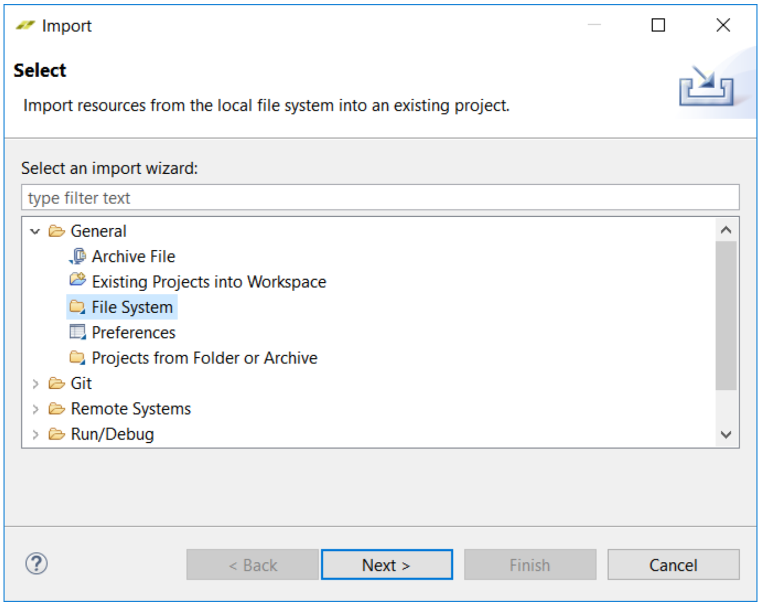


Figure 42 – Import Source Code

    c. In the import window expand the **General** category and select **File System** then click **Next**


Import

Select

Import resources from the local file system into an existing project.

Select an import wizard:

type filter text

Figure 43 – Import from File System

d. **Browse** to the **C:\training\support\source** directory and **Select All** source files for import then click **Finish**

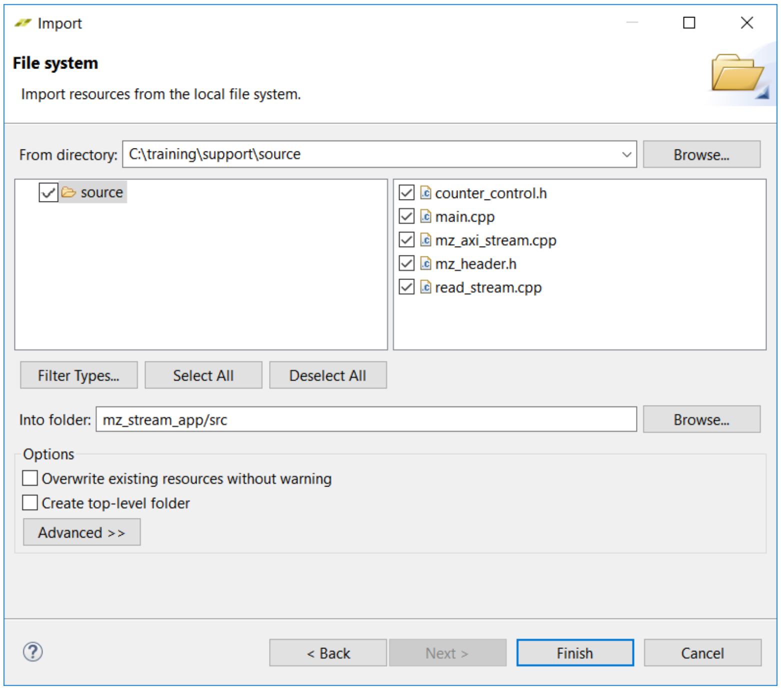
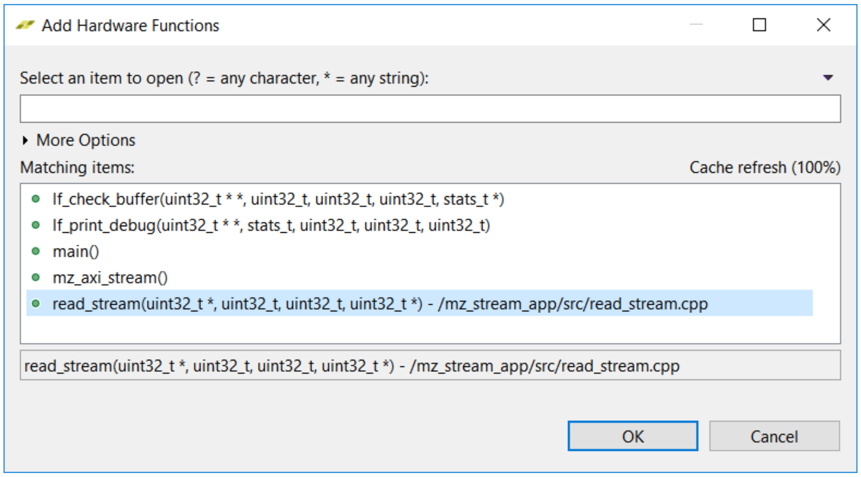


Figure 44 - File System Browser

4. Select the function for Hardware Acceleration that will move data between the PL FIFO buffer and the PS (Note: we aren't really hardware accelerating this function, we are just moving it to programmable logic to act as a shim between the PL FIFO and the Zynq PS)
   a. In the **Hardware Functions** pane click on the ⚡ icon

   b. After the source code is indexed a list of functions will be available for acceleration – Choose the ***read_stream*** function and click **OK**

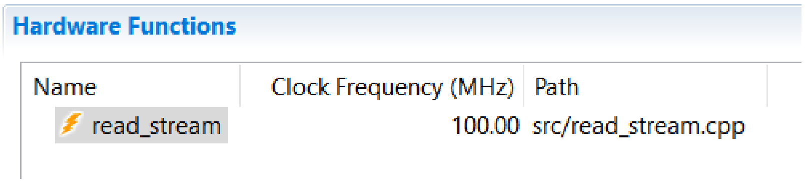


- lf_check_buffer(uint32_t * *, uint32_t, uint32_t, uint32_t, stats_t *)
- lf_print_debug(uint32_t * *, stats_t, uint32_t, uint32_t, uint32_t)
- main()
- mz_axi_stream()
- read_stream(uint32_t *, uint32_t, uint32_t, uint32_t *) - /mz_stream_app/src/read_stream.cpp

read_stream(uint32_t *, uint32_t, uint32_t, uint32_t *) - /mz_stream_app/src/read_stream.cpp

Figure 45 – Application Function List

c. Notice that the *read_stream* function now shows up in the Hardware Functions pane



Hardware Functions

Name	Clock Frequency (MHz)	Path
read_stream	100.00	src/read_stream.cpp

Figure 46 - Hardware Functions

d. The maximum counter rate for the programmable PL counter is 75 MHz. We want to select **accelerator** and **data motion network clock** frequencies of **100 MHz** to ensure that we are emptying the PL FIFO as fast as possible. These values should be default (defined in the mz_stream_pfm.tcl script).

Note: The master side of the AXI-Stream FIFO in the HW platform was connected to the 100 MHz clock. Choosing a data mover clock other than 100 MHz will result in an error.

5. In order to speed up the build process we will enable parallel build capabilities
   a. **Right-click** on the project folder and select **Properties**

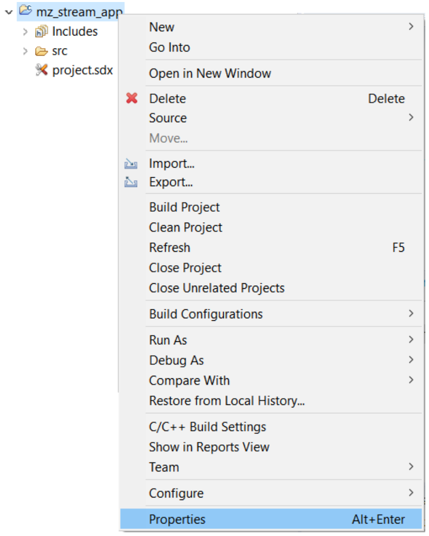
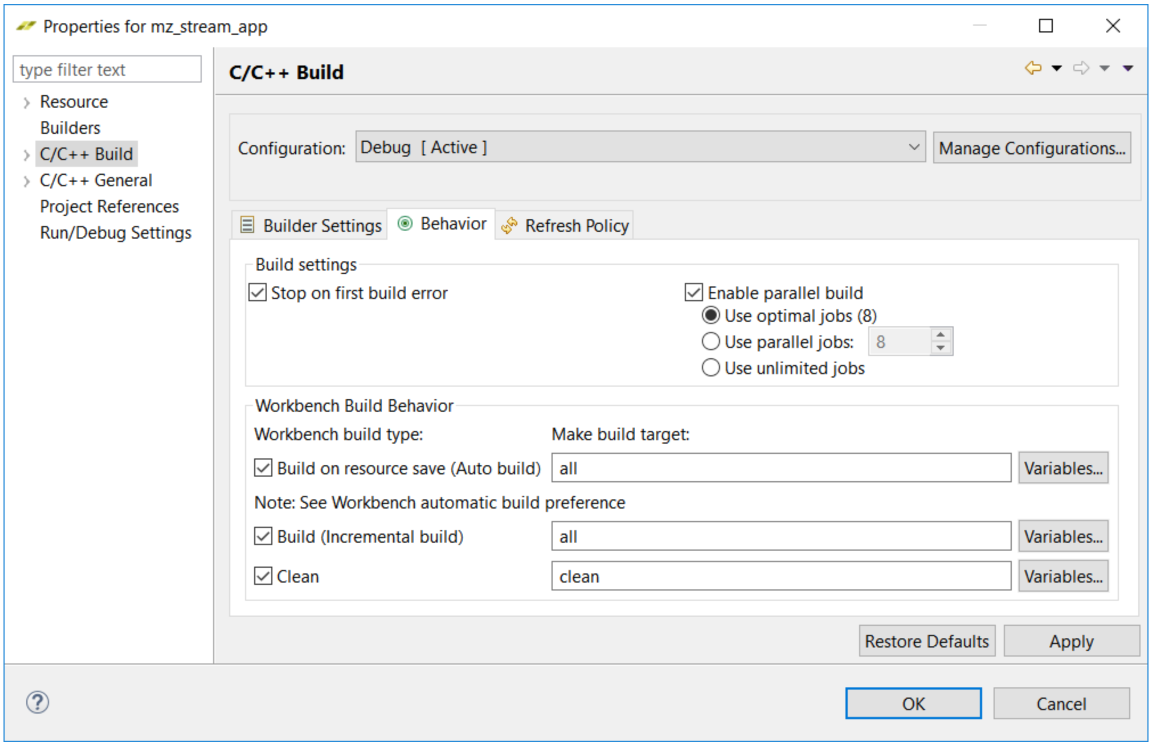


Figure 47 – Project Properties

   b. Click on the **C/C++ Build** category

c. Select the **Behavior tab**, select **Enable parallel build** and click **OK**



Properties for mz_stream_app

C/C++ Build

Configuration: Debug [Active]

Build settings

☑ Stop on first build error

☑ Enable parallel build

● Use optimal jobs (8)

○ Use parallel jobs: 8

○ Use unlimited jobs

Workbench Build Behavior

Workbench build type: Make build target:

☑ Build on resource save (Auto build) all

Note: See Workbench automatic build preference

☑ Build (Incremental build) all

☑ Clean clean

Figure 48 - C/C++ Build Settings

6. Build the application
   a. **Right-click** on the project folder and select **Build Project**

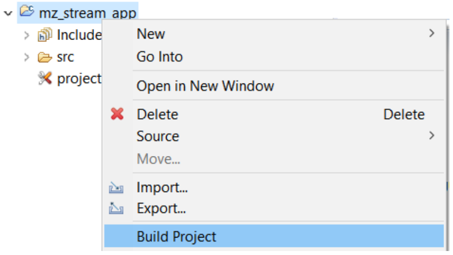
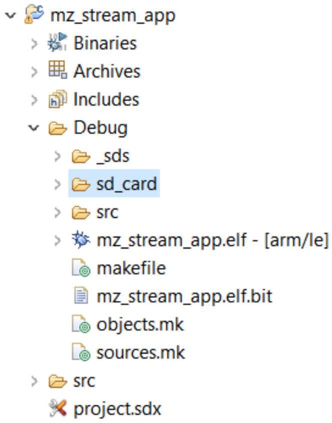


Figure 49 – Build Project

b. The build process will start and you may see a critical warning stating “failed DSA integrity check: digest mismatch.” This warning indicates that the DSA has been modified outside of the Vivado tool suite. We can ignore this warning because the DSA was not modified outside of Vivado and does not impact this lab.

7. When the build completes a directory named sd_card will be created under the Debug folder in the project (assuming Debug build is selected instead of Release). The sd_card directory contains the necessary files for running the application on the MiniZed.



```
mz_stream_app
  Binaries
  Archives
  Includes
  Debug
    _sds
    sd_card
    src
    mz_stream_app.elf - [arm/le]
    makefile
    mz_stream_app.elf.bit
    objects.mk
    sources.mk
  src
  project.sdx
```

Figure 50 – SD Card Directory Containing MiniZed Boot Files

8. Creation of the SDSoC application is complete

9. Experiment 3 will cover programming the MiniZed and running the application.

Discussion: Create the Application

In Experiment 2 we created and built the SDSoC application which configures the programmable counter and reads data from an AXI4-Stream FIFO in the hardware platform. Figure 51 below shows a block diagram representation of the application mapped to PS and PL.

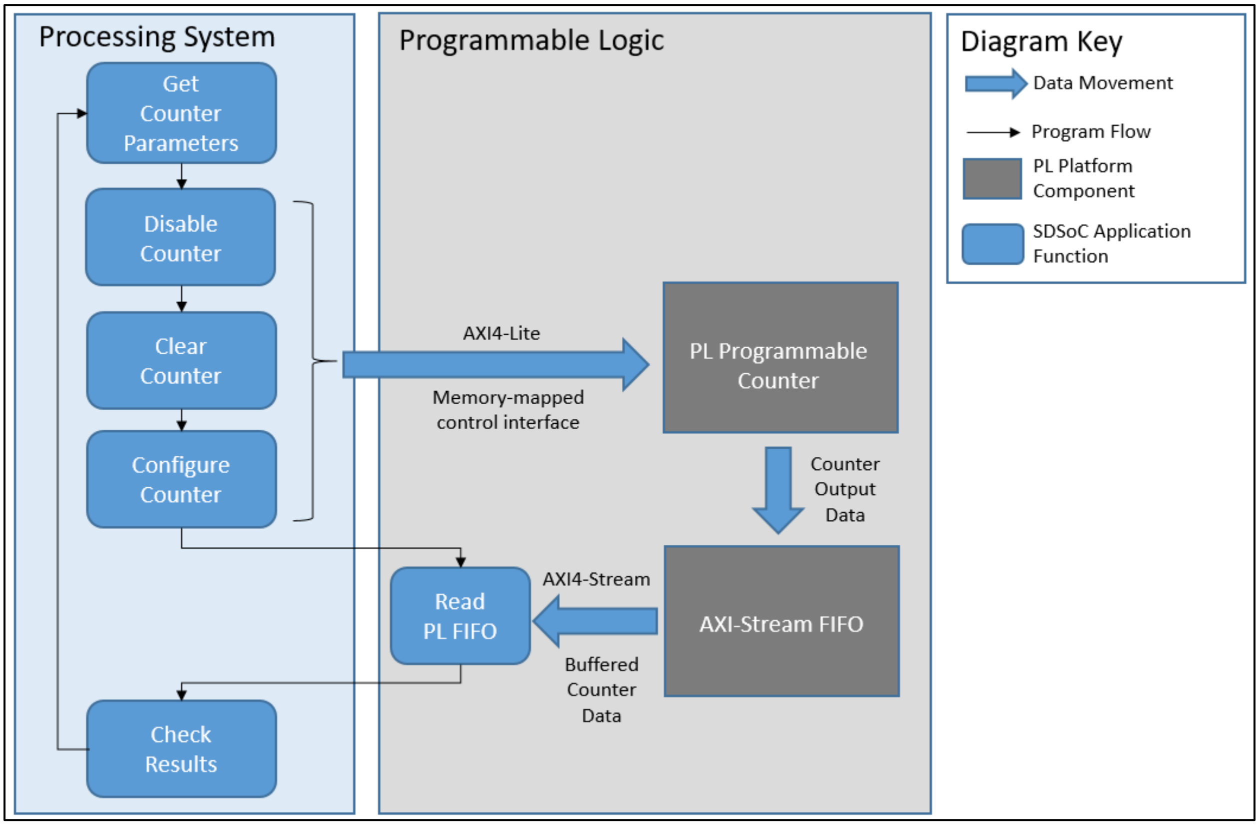


Figure 51 – SDSoC Application

The SDSoC application starts by querying the user for counter parameters (initial value, increment, and update rate). Next the application programs and enables the counter using an AXI4-Lite memory-mapped control interface. Counter control functions are located in the counter_control.h file as a class of functions which write to memory-mapped registers using the Xil_Out32 driver. Another method for writing to the memory-mapped registers would be to use a pointer to the counter's memory-mapped address space. These methods are valid for bare-metal applications only. For Linux applications the UIO device driver must be used with an appropriate device tree entry. Once the counter is configured and enabled the SDSoC application calls the read_stream() function which corresponds to the "Read PL FIFO" operation shown in Figure 51 above. The read_stream() function reads data from the AXI4-Stream data FIFO in the PL and moves it to memory allocated within DDR for the PS part of the SDSoC application to use. Finally, the counter data located in DDR is checked for correctness and the test completes.

Figure 51 above shows a simplified view of the system. For a detailed view open the Vivado project located in C:\training\SDx_wksp_mz_strm\mz_stream_app\Debug_sds\p0_vpl\ipi\syn. Figure 52 below shows the full system after the SDSoC application has been mapped to PS and PL. The details of the IPI block diagram are difficult to see in Figure 52, but we can see a Vivado HLS component located in the top right of the diagram which corresponds to the read_stream function. The remainder of the additional blocks, when compared to Figure 16, are to handle the data movement between the PS and PL.

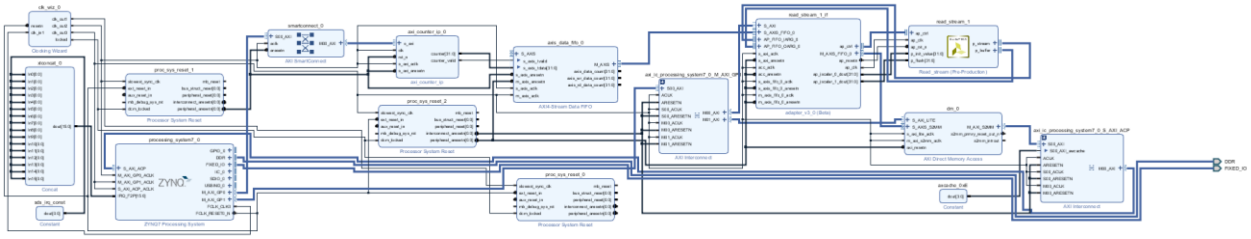
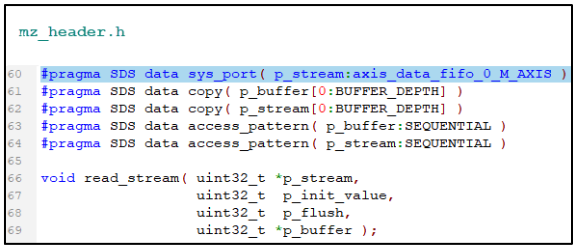


Figure 52 - Vivado IPI Block Diagram after SDSoC Application Partitioning

Our SDSoC application connects to the AXI4-Stream Data FIFO through use of the sys_port pragma. The figure below shows use of the pragma which is applied to the read_stream() function. This pragma informs the SDSoC compiler that function argument p_stream (p_ is handy nomenclature to indicate a variable is a port) should directly connect to the M_AXIS port of the PL FIFO. The tag “axis_data_fifo_0_M_AXIS” defines the system port to bind argument p_stream to and is comprised of the AXI4-Stream Data FIFO component name (axi_data_fifo_0) and the component port (M_AXIS). This information was defined in our mz_stream_pfm.tcl script (Figure 32). The additional pragmas shown in Figure 53 inform the SDSoC compiler how much data to move between the PL and PS as well as the data access pattern. See UG1253 for a detailed explanation of SDSoC pragmas.



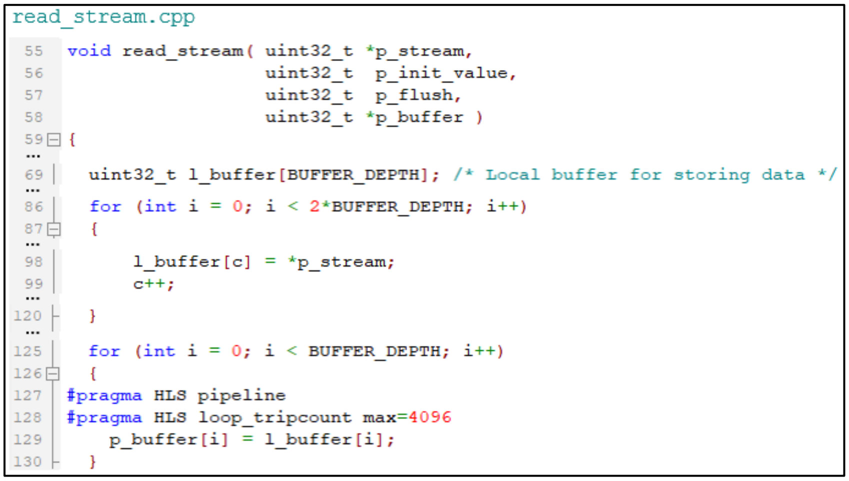
mz_header.h

```
#pragma SDS data sys_port( p_stream:axis_data_fifo_0_M_AXIS )
#pragma SDS data copy( p_buffer[0:BUFFER_DEPTH] )
#pragma SDS data copy( p_stream[0:BUFFER_DEPTH] )
#pragma SDS data access_pattern( p_buffer:SEQUENTIAL )
#pragma SDS data access_pattern( p_stream:SEQUENTIAL )

void read_stream( uint32_t *p_stream,
                  uint32_t  p_init_value,
                  uint32_t  p_flush,
                  uint32_t *p_buffer );
```

Figure 53 – SDSoC Pragmas for Function read_stream()

Figure 54 shows code fragments of read_stream.cpp which demonstrate reading data from the PL FIFO (line 98), copying the stream data to a local PL memory (line 98), and writing the contents of local PL memory to the PS (lines 125 – 130).

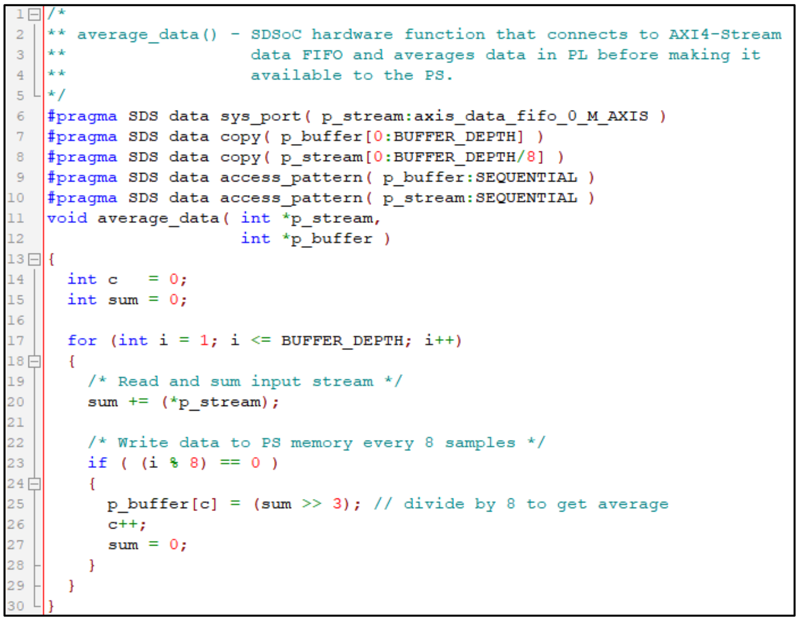


read_stream.cpp

```
 55  void read_stream( uint32_t *p_stream,
 56                    uint32_t  p_init_value,
 57                    uint32_t  p_flush,
 58                    uint32_t *p_buffer )
 59  {
...
 69    uint32_t l_buffer[BUFFER_DEPTH]; /* Local buffer for storing data */
...
 86    for (int i = 0; i < 2*BUFFER_DEPTH; i++)
 87    {
...
 98        l_buffer[c] = *p_stream;
 99        c++;
...
120    }
...
125    for (int i = 0; i < BUFFER_DEPTH; i++)
126    {
127  #pragma HLS pipeline
128  #pragma HLS loop_tripcount max=4096
129      p_buffer[i] = l_buffer[i];
130    }
```

Figure 54 – Function read_stream Code Fragments

Our application copies streaming data to a local PL memory (l_buffer) before transferring to the PS. We could modify the application to remove the local PL buffer as shown in the example code of Figure 55. Figure 55 also shows an averaging operation which is computed over 8 input samples and demonstrates how a function can operate on streaming data in the PL before being transferred to the PS.



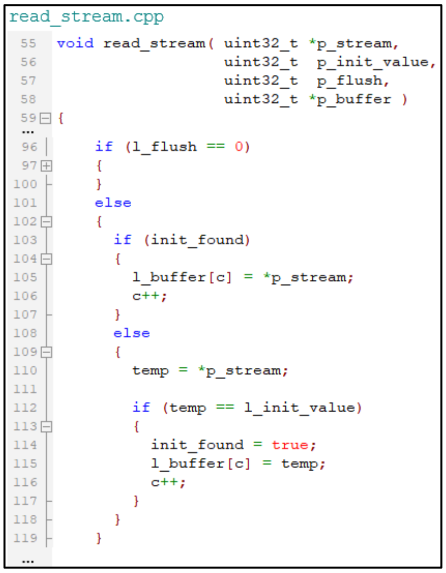
```
/*
** average_data() - SDSoC hardware function that connects to AXI4-Stream
**                  data FIFO and averages data in PL before making it
**                  available to the PS.
*/
#pragma SDS data sys_port( p_stream:axis_data_fifo_0_M_AXIS )
#pragma SDS data copy( p_buffer[0:BUFFER_DEPTH] )
#pragma SDS data copy( p_stream[0:BUFFER_DEPTH/8] )
#pragma SDS data access_pattern( p_buffer:SEQUENTIAL )
#pragma SDS data access_pattern( p_stream:SEQUENTIAL )
void average_data( int *p_stream,
                   int *p_buffer )
{
  int c   = 0;
  int sum = 0;

  for (int i = 1; i <= BUFFER_DEPTH; i++)
  {
    /* Read and sum input stream */
    sum += (*p_stream);

    /* Write data to PS memory every 8 samples */
    if ( (i % 8) == 0 )
    {
      p_buffer[c] = (sum >> 3); // divide by 8 to get average
      c++;
      sum = 0;
    }
  }
}
```

Figure 55 – Example Code

The read_stream.cpp file also contains logic to flush the PL FIFO (shown in Figure 56 below). The flushing logic works by searching for the first expected sample value in the stream. All samples found prior to this value are discarded. Flushing is performed between iterations of the while-loop in mz_axi_stream() so that we are not analyzing stale data.

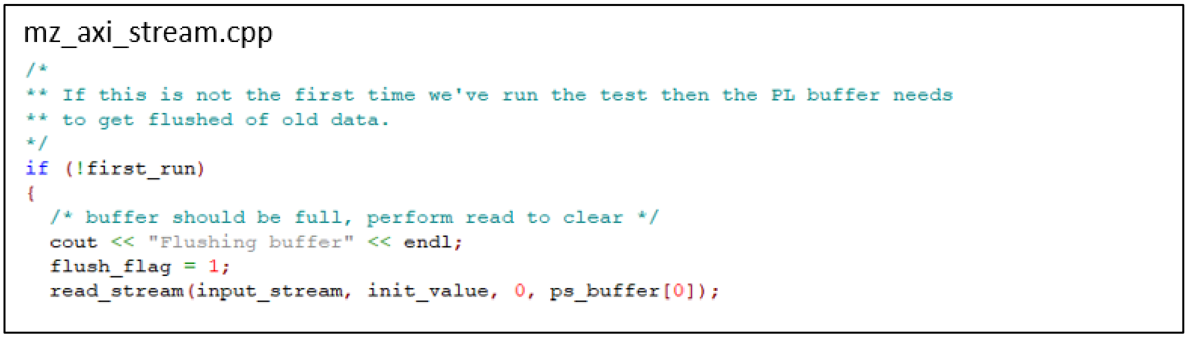


```
read_stream.cpp
void read_stream( uint32_t *p_stream,
                  uint32_t  p_init_value,
                  uint32_t  p_flush,
                  uint32_t *p_buffer )
{
...
    if (l_flush == 0)
    {
    }
    else
    {
      if (init_found)
      {
        l_buffer[c] = *p_stream;
        c++;
      }
      else
      {
        temp = *p_stream;

        if (temp == l_init_value)
        {
          init_found = true;
          l_buffer[c] = temp;
          c++;
        }
      }
    }
...
```

Figure 56 – PL FIFO Flush Logic

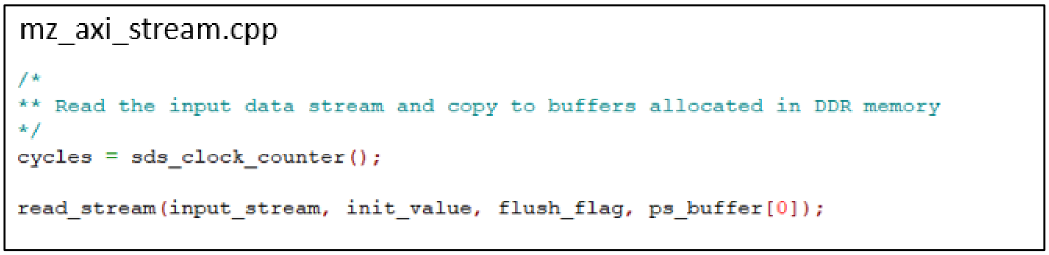
The AXI4-Stream data FIFO that we inserted in the PL during hardware platform creation had a depth of 4,096. The data movement framework provided by the SDSoC compiler adds an additional 1K buffer which must also be cleared before starting a new test. Thus, to clear the data path of old data we first read 4,096 samples from the PL FIFO (normal read shown in Figure 57 below) followed by a read to flush the remaining 1K samples in the data path (flushing read shown in Figure 58 below).



mz_axi_stream.cpp

```
/*
** If this is not the first time we've run the test then the PL buffer needs
** to get flushed of old data.
*/
if (!first_run)
{
  /* buffer should be full, perform read to clear */
  cout << "Flushing buffer" << endl;
  flush_flag = 1;
  read_stream(input_stream, init_value, 0, ps_buffer[0]);
```

Figure 57 - Normal Read to Clear 4,096 Samples



mz_axi_stream.cpp

```
/*
** Read the input data stream and copy to buffers allocated in DDR memory
*/
cycles = sds_clock_counter();

read_stream(input_stream, init_value, flush_flag, ps_buffer[0]);
```

Figure 58 - Flushing Read to Clear 1K Samples from Data Movement Framework (flush_flag = 1)

Experiment 3: Test the Application

In this experiment we will program the MiniZed with the application built during Experiment 2. The following steps cover programming the MiniZed and executing the application.

Connect the Hardware

1. First configure the boot jumper. You can select between **F**LASH and **J**TAG booting. We want to ensure switch 1 is set towards the F or PS_Button.

    a. Note: From the Factory the switch's protective film should be removed and already set to F. If it is not, the switch will look similar to Figure 59.

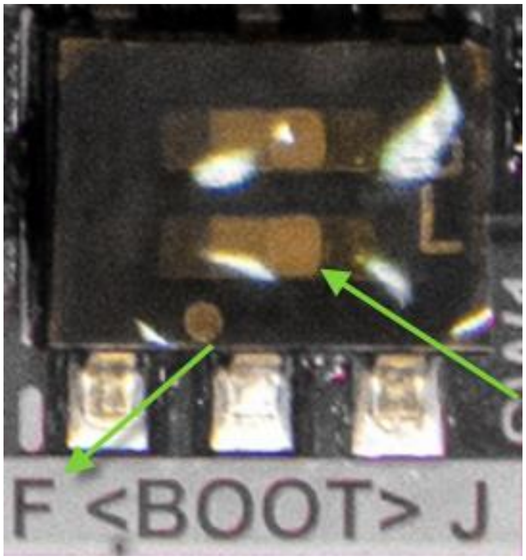


Figure 59 - Untouched Boot Switch

    b. If your MiniZed boot configuration switch is similar to the above, remove the protective film and slide switch 1 (indicated by the silkscreen DASH above the F) to be toggled to FLASH Booting (F).

2. Next plug the MiniZed into your PC in order to register the board with a COM port

    a. Note: Windows 10 has been known to create two COM ports when plugging the MiniZed into the PC.

    b. With a factory fresh board, open two instances of your terminal program (PuTTY or Tera Term), one for each COM port; 8,N,1,115200

    c. Reboot your MiniZed using the Reset button

    d. The terminal window that shows text output from the MiniZed is connected to the COM port of interest. Close the other terminal window.

Program the MiniZed

1. From the SDx GUI launch a shell window
    a. Under the **Xilinx** menu in SDx click on **Launch Shell**

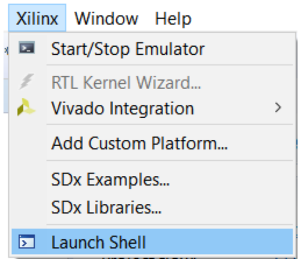


Figure 60 – Launch SDx Shell

    b. Change directories to your workspace (C:\training\SDx_wksp_mz_strm) if not already there
        i. cd C:\training\SDx_wksp_mz_strm

2. From the command prompt execute the following to program the MiniZed

    a. If you completed Experiment 1 use:

    program_flash –f mz_stream_app\Debug\sd_card\BOOT.BIN –fsbl mz_stream\export\mz_stream\sw\config0\boot\fsbl.elf –flash_type qspi_single

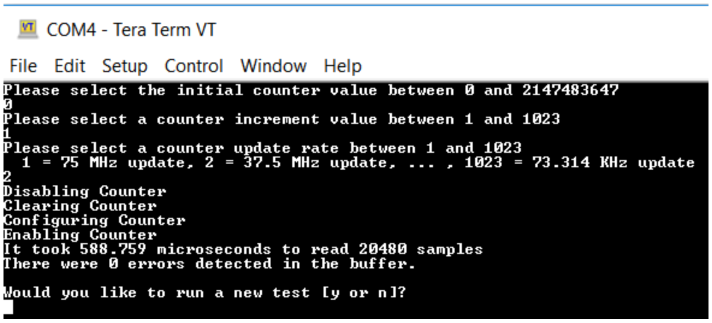
    b. If you skipped Experiment 1 use:

    program_flash –f mz_stream_app\Debug\sd_card\BOOT.BIN –fsbl C:\training\support\platforms\mz_stream\sw\config0\boot\fsbl.elf –flash_type qspi_single

3. After programming completes **reset** the MiniZed **by pressing** and releasing the **reset button** near the MiniZed PMOD connectors

4. You should see a prompt from the MiniZed in your terminal window asking for input. Try the following values
    a. Initial counter value = **0**
    b. Counter increment value = **1**
    c. Counter update rate = **2**

5. You should see the following output in your terminal window



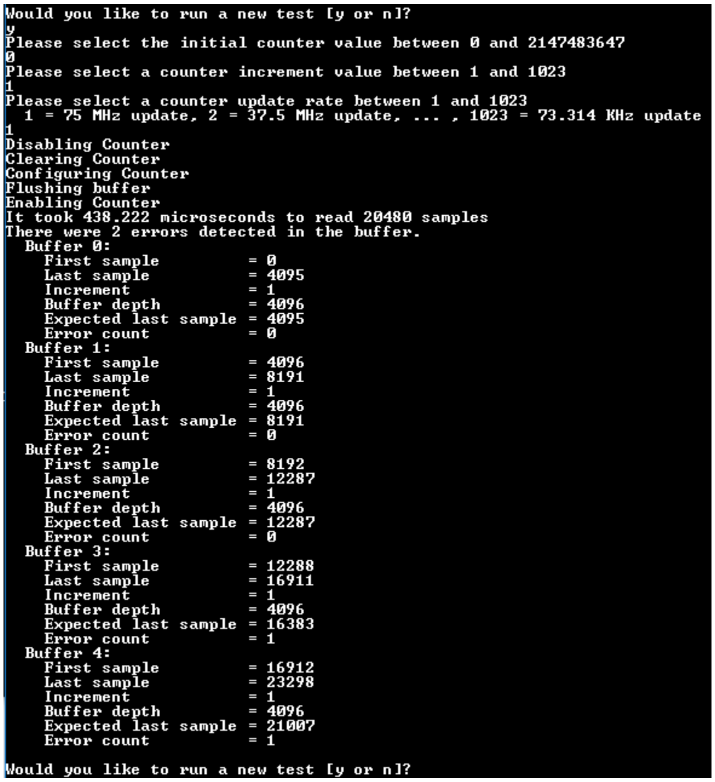
```
Please select the initial counter value between 0 and 2147483647
0
Please select a counter increment value between 1 and 1023
1
Please select a counter update rate between 1 and 1023
  1 = 75 MHz update, 2 = 37.5 MHz update, ... , 1023 = 73.314 KHz update
2
Disabling Counter
Clearing Counter
Configuring Counter
Enabling Counter
It took 588.759 microseconds to read 20480 samples
There were 0 errors detected in the buffer.

Would you like to run a new test [y or n]?
```

Figure 61 – First Program Execution Output

6. The program halts and asks if you would like to run again. Type **y** and press **Enter** on your keyboard.

7. Repeat the test with a **counter update rate** of **1** (75 MHz)
   a. Notice how errors are detected in the last buffer(s). This indicates that we are not able to empty the PL buffer fast enough to support the 75 MHz sample rate
   b. This issue will be addressed in an additional lab



```
Would you like to run a new test [y or n]?
y
Please select the initial counter value between 0 and 2147483647
0
Please select a counter increment value between 1 and 1023
1
Please select a counter update rate between 1 and 1023
  1 = 75 MHz update, 2 = 37.5 MHz update, ... , 1023 = 73.314 KHz update
1
Disabling Counter
Clearing Counter
Configuring Counter
Flushing buffer
Enabling Counter
It took 438.222 microseconds to read 20480 samples
There were 2 errors detected in the buffer.
  Buffer 0:
    First sample          = 0
    Last sample           = 4095
    Increment             = 1
    Buffer depth          = 4096
    Expected last sample  = 4095
    Error count           = 0
  Buffer 1:
    First sample          = 4096
    Last sample           = 8191
    Increment             = 1
    Buffer depth          = 4096
    Expected last sample  = 8191
    Error count           = 0
  Buffer 2:
    First sample          = 8192
    Last sample           = 12287
    Increment             = 1
    Buffer depth          = 4096
    Expected last sample  = 12287
    Error count           = 0
  Buffer 3:
    First sample          = 12288
    Last sample           = 16911
    Increment             = 1
    Buffer depth          = 4096
    Expected last sample  = 16383
    Error count           = 1
  Buffer 4:
    First sample          = 16912
    Last sample           = 23298
    Increment             = 1
    Buffer depth          = 4096
    Expected last sample  = 21007
    Error count           = 1

Would you like to run a new test [y or n]?
```

Figure 62 – Program Output with 75 MHz Counter Update Rate

8. To exit the program type **n** at the prompt and then press **Enter** on your keyboard

Discussion: Test the Application

In Experiment 3 we tested the SDSoC application on the MiniZed platform. We noticed that errors were detected in the DDR buffers when a sample rate of 75 MHz was selected. This indicates that we are unable to read from the PL FIFO fast enough to prevent an overflow. This was by design to set up for an additional lab titled “Advanced Concepts with Xilinx SDSoC – Asynchronous Accelerators” where we will look at running our PL accelerators asynchronously to reduce the function call overhead.

Conclusion

This lab covered creating a SDSoC platform to support streaming input data as well as accessing the streaming data from a SDSoC application running a standalone operating system. The streaming input data was modeled with a programmable counter residing in the PL that is configured using an AXI4-Lite memory-mapped interface. The output of the PL counter filled an AXI4-Stream data FIFO which our SDSoC application was directly connected to move streaming data from PL to DDR. Our SDSoC application running on the PS then read the DDR data and checked it for correctness.

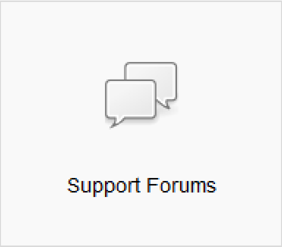
After completing this lab you should be able to develop your own streaming platform for use with SDSoC!

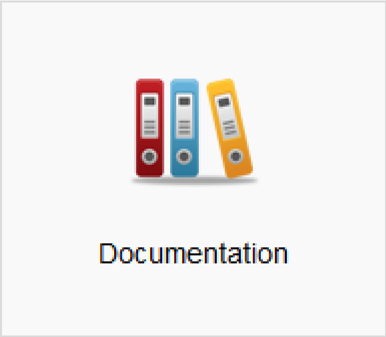
Appendix A: Getting Support

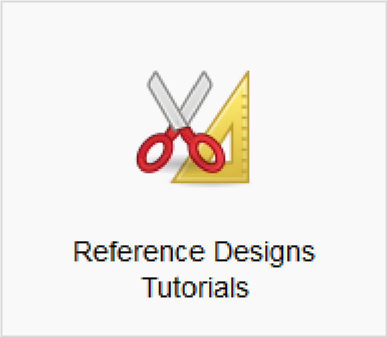
Avnet Support

- Technical support is offered online through the minized.org website support forums. MiniZed users are encouraged to participate in the forums and offer help to others when possible.

- To access the most current collateral for the MiniZed, visit the community support page (www.minized.org/content/support) and click one of the icons shown below:







- MiniZed Documentation
  http://minized.org/support/documentation/18891

- MiniZed Reference Designs
  http://minized.org/support/design/18891/146

Xilinx Support

The following technical support options are available to Xilinx customers:

- Technical information is available online 24 hours a day from the Support website
- Technical Support staff are available to respond to your questions in the Community Forums
- Individual assistance from Xilinx Technical Support ***may*** be available through Service Portal
- Phone support is ***only*** available with an active open case number

Global Phone Number

Region	**Language**	**Phone****	**Support Hours***
North America	EN	1 800-255-7778 or +1 408-879-5199	M-F 7:00 -17:00 PST
Europe, Middle East and Africa	EN, DE, FR	00 800-5152-5152 or +353 1-461-5700	M-F 8:00 -17:00 GMT
China	CH (Mandarin), EN	+86 800 988 0218 +86 400 880 0218 (Mobile Phone)	M-F* 9:00 -18:00 CST
Taiwan	CH (Mandarin), EN	+886 2-8176-1060	M-F 9:00 -18:00 CST
Hong Kong	CH (Mandarin), EN	+852 3187-3855	M-F 9:00 -18:00 CST

* Support hours listed apply for both standard and daylight savings (summer) time. Please check the Technical Support Holiday Calendar for support availability during holidays in your region.

** 00 800-5152-5152 is a international free phone (toll free) number available in the following countries: Austria, Belgium, Denmark, Finland, France, Germany, Ireland Israel, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, and United Kingdom. All other countries must use +353 1-461-5700.

** For the numbers listed, '**+**' represents the International Direct Dialing (IDD) prefix of the country from which you are calling. Please consult your local telephone service provider for more information on specific IDD instructions.

Revision History

Date	Version	Revision
4/19/2018	1	Initial release