Introduction to Systems-on-Chip Interconnects and System Integration

Abigail Butka, Christophe Bobda

March 29, 2024

1 Objective

The learning objective of this module is to gain hands-on experience in System-on-Chip design, peripheral IP integration, and both the AXI and APB bus protocols. In this lab you will use the Xilinx Zynq ARM Cortex-A9 hard-core processor and pre-provided Verilog/VHDL code to build your own SoC in Vivado. Then, you will write the C applications that will run on the Zynq processor.

For more conceptual details of this module, please refer to the accompanying PowerPoint presentation and video.

In the unfortunate case that the following tutorial no longer produces a bitstream due to software updates to Vivado, the bitstreams for each task are provided in the accompanying zip file for both the ZYBO Z7-10 and the PYNQ-Z2.

2 Equipment and Software Needed for this Module

1. Xilinx Vivado 2021.1.
2. Xilinx FPGA with a ZYNQ ARM Cortex-A9 processor.
   (a) This lab was tested using the PYNQ-Z2 and ZYBO Z7-10
3. A USB-UART cable to program the board and display the output in the terminal

3 Module Description

3.1 Task 1: Creating a Custom AXI Peripheral

In this task, you will create a custom AXI peripheral IP in Vivado, then use Vitis to write a bare-metal C application that will write to and read from this peripheral IP using the ZYNQ processor. We have provided a VHDL implementation of a median filter, a smoothing filter that removes noise from data, and its testbench. This code has one input and two outputs that user data can be sent to, input_data, output_data, and mid respectively.

Follow the below instructions to create an AXI peripheral IP of the median filter and package it in Vivado.

NOTE: Steps 1-8 of Task 1 are very brief and lack images because this information is covered in MEST Module 1. Please refer to this module if there are any questions.

1. Create a Vivado project, make sure you choose the board you are going to use.
2. Create Block Design.
3. Add IP ZYNQ7 Processing System to the Block Diagram.
4. Run Block Automation.

5. Connect M_AXI_GP0_ACLK to FCLK_CLK0 of the Zynq IP.

6. Validate Design

7. Under Sources/Design Sources, right click design_1.bd, Create HDL Wrapper, and Let Vivado Manage Wrapper and Auto-Update.

8. This will create a design_1_wrapper.v file, that will now be the top file of the design

9. Click Tools/Create and Package New IP, once the window loads, click next

10. Select Create a new AXI4 peripheral and click next

11. Re-name your IP to “AXI_MedianFilter”and save it to a folder titled “ip_repo.” Vivado will automatically create a folder titled ip_repo in the folder above your project. Click next

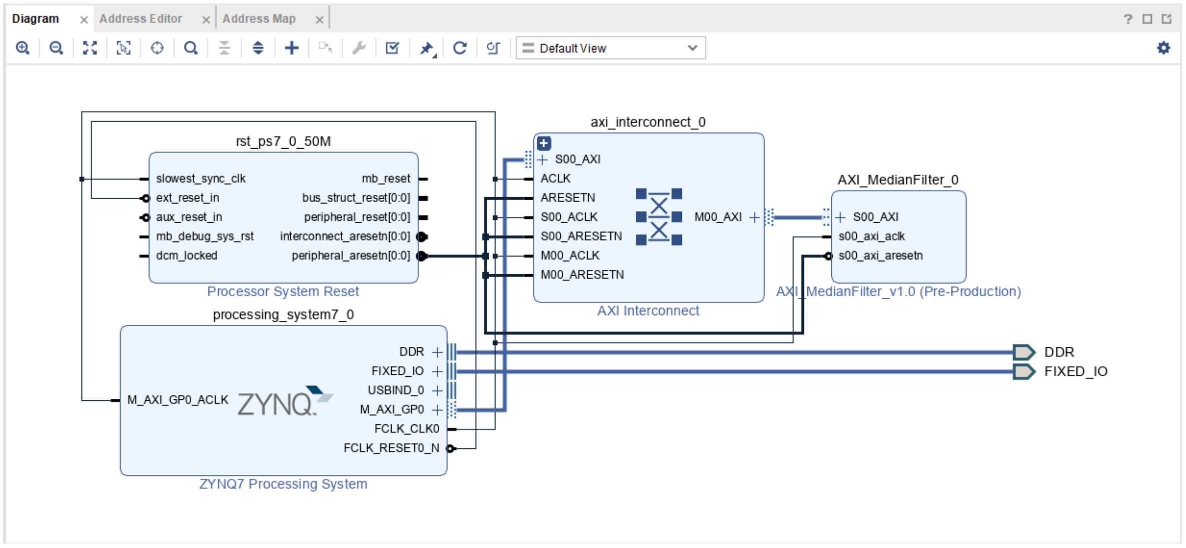
12. On the Add Interfaces page, ensure the number of registers is set to 4, click next, then finish.

13. Now that you have created and automatically added this IP to your project, right-click on the block diagram, click Add IP, and type in AXI_MedianFilter. Select AXI_MedianFilter_v1.0.

14. Click Run Connection Automation, ensure all boxes are checkmarked, then click OK.

15. To make the block diagram look more readable, click on the circular arrow across the top of the diagram, titled Regenerate Layout.

16. At this point, your block design should look like the following image.



17. This custom Median Filter IP only contains the AXI interface, we now need to instantiate the provided Median Filter VHDL code.

18. To open the AXI_MedianFilter IP, select the block in the diagram, right-click, and then select Edit in IP Packager. Click OK.

    (a) In the future, when you open any IP modules this way, it will pop up with a notification saying this project already exists. Click OK.

19. Under design sources, double-click on AXI_MedianFilter_v1_0_S00_AXI_inst to open the Verilog file, scroll to the bottom of the module to where it says “// Add user logic here”, and insert the following code:

```
    // Add user logic here
        wire [31:0] output_data;
        wire [31:0] mid;
        MedianFilter # (
        .NUM_VALS(8),
        .SIZE(4)
        ) MedianFilterInst (
        .clk(S_AXI_ACLK),
        .input_data(slv_reg0),
        .output_data(output_data),
        .mid(mid)
        );
    // User logic ends
```

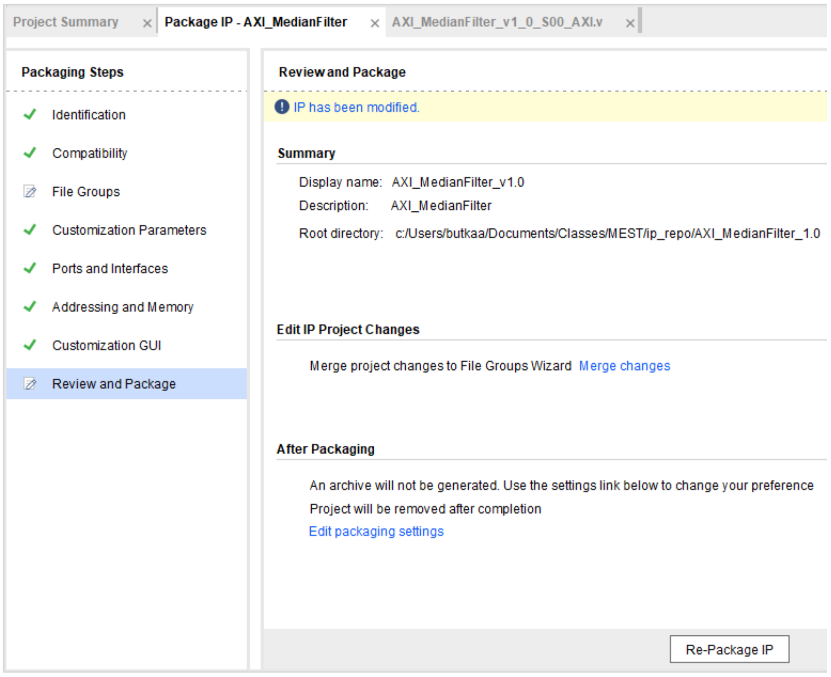
20. Then, on lines 374 and 375, replace slv_reg1 and slv_reg2's inputs with the following code:

```
    2'h1   : reg_data_out <= output_data;
    2'h2   : reg_data_out <= mid;
```

Ensure the final Case Statement looks as follows:

```
case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
    2'h0   : reg_data_out <= slv_reg0;
    2'h1   : reg_data_out <= output_data;
    2'h2   : reg_data_out <= mid;
    2'h3   : reg_data_out <= slv_reg3;
    default : reg_data_out <= 0;
endcase
```

21. Next, under Sources, click on the plus sign to Add Sources. Select Add or create design sources and click Next.
22. Click Add Files and navigate to this module's Task 1 folder. Select MedianFilter.vhd. Click OK.
23. Ensure that "Copy sources into IP Directory" is checked, and then click Finish.
24. Now that you have completed modifying the AXI code, navigate to the "Package IP – AXI_MedianFilter" tab. There will be multiple unchecked packaging steps, such as "File Groups" and "Review and Package", navigate to each of these steps and click "merge" in the top left corner as seen in the image below.


Project Summary | Package IP - AXI_MedianFilter | AXI_MedianFilter_v1_0_S00_AXI.v

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- Review and Package

Review and Package

IP has been modified.

Summary

Display name:	AXI_MedianFilter_v1.0
Description:	AXI_MedianFilter
Root directory:	c:/Users/butkaa/Documents/Classes/MEST/ip_repo/AXI_MedianFilter_1.0

Edit IP Project Changes

Merge project changes to File Groups Wizard Merge changes

After Packaging

An archive will not be generated. Use the settings link below to change your preference

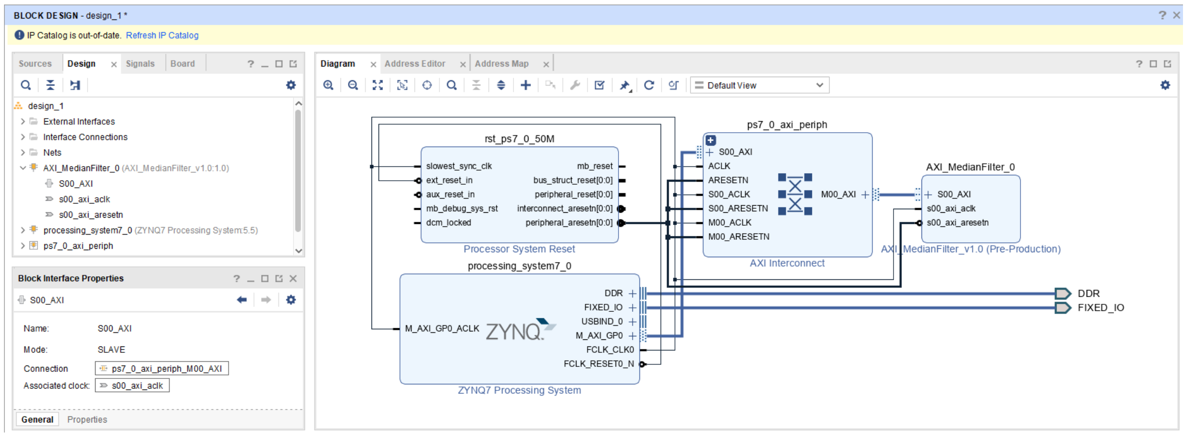
Project will be removed after completion

Edit packaging settings

Re-Package IP

25. When all steps are checked, in "Review and Package" click on Re-Package IP as seen in the above image, this may pop up with some notification windows, click Yes. Finally, there will be a window asking if you want to close the project, select yes.

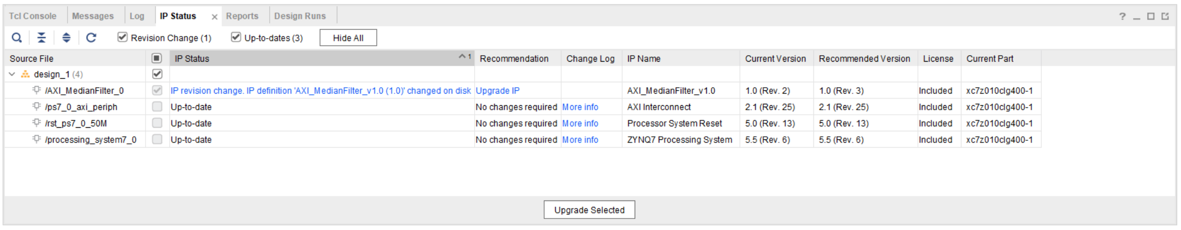
26. Back on the block diagram screen you will see a yellow flag across the top of the screen stating Refresh IP Catalog and/or Report IP Status like in the image below. Click on the blue text until it shows Show IP Status.


BLOCK DESIGN - design_1 *

IP Catalog is out-of-date. Refresh IP Catalog

Name:	S00_AXI
Mode:	SLAVE
Connection	ps7_0_axi_periph_M00_AXI
Associated clock:	s00_axi_aclk

27. Then, under IP Status, if there is an option for Refresh, click on it. Then click on "Upgrade Selected."


Tcl Console | Messages | Log | IP Status | Reports | Design Runs

☑ Revision Change (1) ☑ Up-to-dates (3) Hide All

Source File	▣	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
design_1 (4)	☑								
/AXI_MedianFilter_0	☑	IP revision change. IP definition 'AXI_MedianFilter_v1.0 (1.0)' changed on disk	Upgrade IP		AXI_MedianFilter_v1.0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7z010clg400-1
/ps7_0_axi_periph	☐	Up-to-date	No changes required	More info	AXI Interconnect	2.1 (Rev. 25)	2.1 (Rev. 25)	Included	xc7z010clg400-1
/rst_ps7_0_50M	☐	Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7z010clg400-1
/processing_system7_0	☐	Up-to-date	No changes required	More info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z010clg400-1

Upgrade Selected

28. When finished, Vivado will pop up a window stating IP Upgrade Completed, click OK.

29. A new window will pop up stating Generate Output Products. Click Generate. When the final window pops up, click OK.

30. To remove the yellow flag across IP Status, click on Rerun.

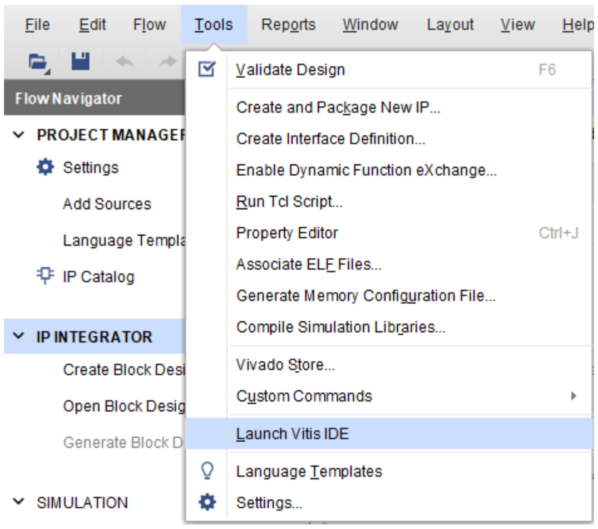
31. Next, click on Generate Bitstream

32. When the window "Bitstream Generation successfully completed" pops up, click cancel.

33. Click on File/Export/Export Hardware

34. Click Next, select Include bitstream, then click Next. Keep the file named design_1_wrapper, click OK.

35. Click on Tools/Launch Vitis ID as seen in the image below.



36. When the new window loads titled "Select a directory as workspace", click on browse.

37. Browse to the location of your Vivado project, you should see design_1_wrapper.xsa.

38. Right-click and create a folder titled "Vitis_WS." This is where all your Vitis projects will be stored. Click Select Folder.

39. Click Launch

40. When Vitis loads, click Create Application Project

41. You will open the home page showing the Welcome Page, click next

42. You will now be on the Platform Page. Click the "Create a new platform from hardware (XSA)" tab.

43. Click browse, navigate to your Vivado project again, and select the design_1_wrapper.xsa file, click next.

44. Name the project HelloWorld.

45. On the Domain Page, click Next.

46. On the Template page, select Hello World, and then click Finish.

47. This will create both the platform project and application project in your Vitis Workspace.

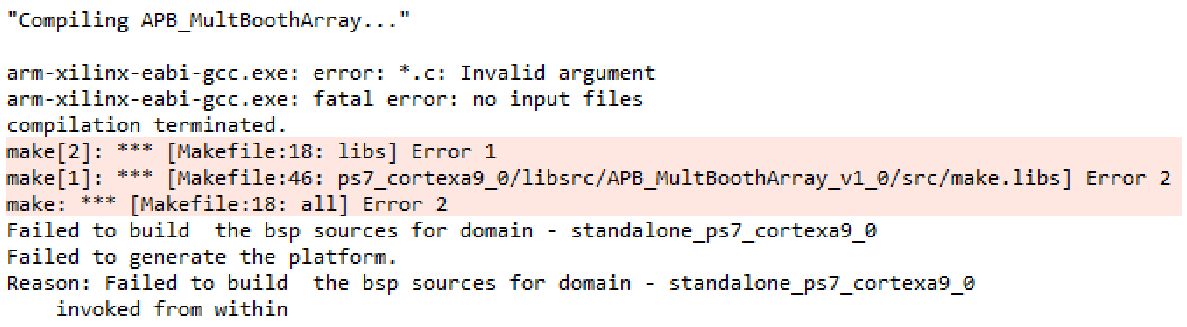
48. To find the helloworld.c file, click on the arrows beside each of the following folders in the Explorer to expand them: HelloWorld_System/HelloWorld/src/helloworld.c.

49. Double-click on helloworld.c to open it.

50. Now that you have opened helloworld.c, replace all of the code within this file with the code from the provided file in the module's Task1 folder, "HelloWorld_Task1.c".

51. The following step will create an error in Vivado 2021.1. This is for demonstration purposes.

52. Now that you have added the code to communicate with the AXI peripheral, right click on the green design_1_wrapper platform project and select "Build Project." The error this step produces can be seen in the image below.



```
"Compiling APB_MultBoothArray..."

arm-xilinx-eabi-gcc.exe: error: *.c: Invalid argument
arm-xilinx-eabi-gcc.exe: fatal error: no input files
compilation terminated.
make[2]: *** [Makefile:18: libs] Error 1
make[1]: *** [Makefile:46: ps7_cortexa9_0/libsrc/APB_MultBoothArray_v1_0/src/make.libs] Error 2
make: *** [Makefile:18: all] Error 2
Failed to build  the bsp sources for domain - standalone_ps7_cortexa9_0
Failed to generate the platform.
Reason: Failed to build  the bsp sources for domain - standalone_ps7_cortexa9_0
    invoked from within
```

53. The above step created an error because of an issue in Vivado 2021.1 that causes the auto-generated Vitis projects to not be able to locate the .c files of custom IP components.

54. To fix this error, navigate to the following directories and paste the following code as seen in the images below.

    (a) design_1_wrapper/HW/Drivers/AXI_MedianFilter_v1_0/src/Makefile

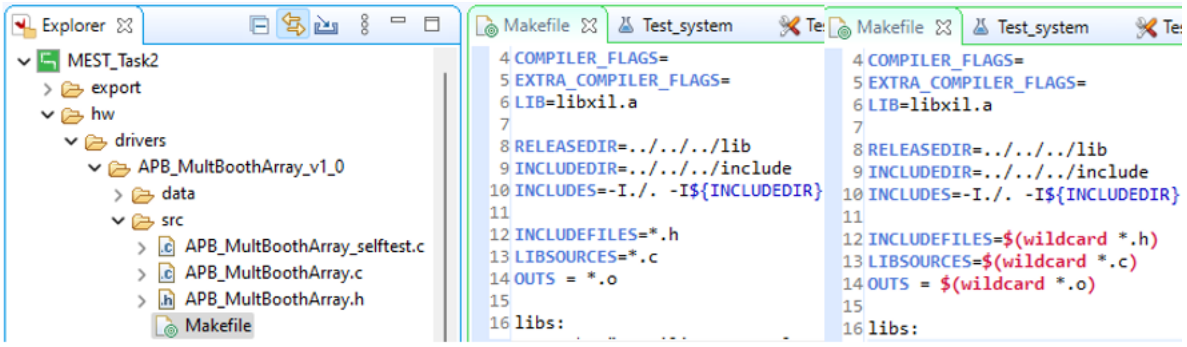
    (b) design_1_wrapper/ps7_cortexa9_0/standalone_ps7_cortexa9_0/libsrc/AXI_MedianFilter_v1_0/src/Makefile

    (c) design_1_wrapper/zynq_fsbl/zynq_fsbl_bsp/ps7_cortexa9_0/libsrc/AXI_MedianFilter_v1_0/src/Makefile

    In each Makefile replace lines 12-14 with the following code:

```
INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS = $(wildcard *.o)
```

    An image of the Makefile before and after the fix can be seen below



```
 4 COMPILER_FLAGS=
 5 EXTRA_COMPILER_FLAGS=
 6 LIB=libxil.a
 7
 8 RELEASEDIR=../../../lib
 9 INCLUDEDIR=../../../include
10 INCLUDES=-I./. -I${INCLUDEDIR}
11
12 INCLUDEFILES=*.h
13 LIBSOURCES=*.c
14 OUTS = *.o
15
16 libs:
```

```
 4 COMPILER_FLAGS=
 5 EXTRA_COMPILER_FLAGS=
 6 LIB=libxil.a
 7
 8 RELEASEDIR=../../../lib
 9 INCLUDEDIR=../../../include
10 INCLUDES=-I./. -I${INCLUDEDIR}
11
12 INCLUDEFILES=$(wildcard *.h)
13 LIBSOURCES=$(wildcard *.c)
14 OUTS = $(wildcard *.o)
15
16 libs:
```

File Location   Makefile Before Fix   Makefile After Fix

55. After making the above fix, right-click on the green design_1_wrapper platform project and select "Build Project" again. This time, the build will complete with no errors.

56. Next, right-click on the blue HelloWorld_system application project and select "Build Project"

(a) If either step 55 or 56 fails with an unexpected SD Card error, clean the project, then build the project again, it should work the second time.

57. After building both projects, open the Vitis Serial Terminal by clicking on the magnifying glass in the top right corner and searching for Vitis Serial Terminal.

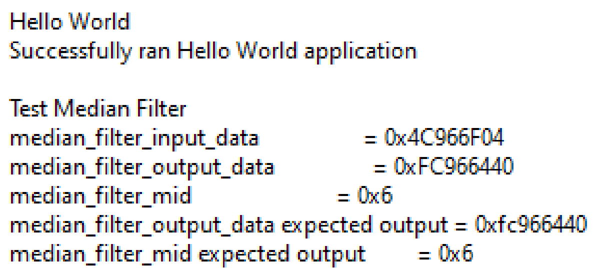
58. Ensure your FPGA is plugged into your computer and turned on. Locate the COM port of your device through the Device Manager.

59. Click on the green plus sign within the Vitis Serial Terminal to connect to the serial port.

60. Select the FPGA's COM port, and ensure the baud rate is set to 115200. Click OK.

61. Right click on the blue HelloWorld_system Application project and select "Run as/ 1. Launch Hardware."

62. Wait for the board to be programmed, then watch the serial terminal output the results as shown in the image below.



```
Hello World
Successfully ran Hello World application

Test Median Filter
median_filter_input_data              = 0x4C966F04
median_filter_output_data              = 0xFC966440
median_filter_mid                     = 0x6
median_filter_output_data expected output = 0xfc966440
median_filter_mid expected output        = 0x6
```

3.2 Task 2: Creating a Custom APB Peripheral

1. To begin, open the project created in the previous task.

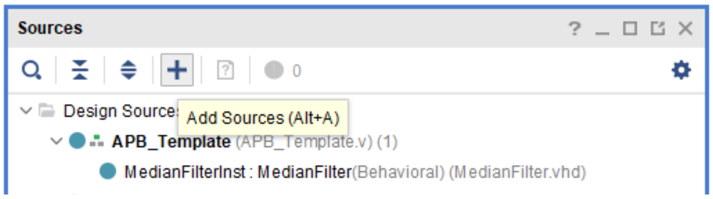
2. Delete the Median Filter AXI4 peripheral from the block diagram.

3. Begin by following the steps in Task 1 to create a new AXI peripheral, call this IP "APB_MultBoothArray."

4. Leave all settings as their defaults.

5. Add APB_MultBoothArray to the Block Diagram, then edit the peripheral in the IP Packager. Do not Run Connection Automation or connect the IP.

6. Click on Add Design Sources as seen in the image below, and click on Add Files.



Design Sources
APB_Template (APB_Template.v) (1)
MedianFilterInst : MedianFilter(Behavioral) (MedianFilter.vhd)

7. Navigate to this module's folder, Task2. Select all .v and .vhd files, click OK

   (a) APB_Wrapper.v

   (b) bd_mult_slice.vhd

   (c) mult_booth_array.vhd

   (d) register_chain.vhd

8. Ensure that "Copy sources into IP Directory" is checked, then click Finish.

9. Next, delete the two top-level AXI files, APB_MultBoothArray and APB_MultBoothArray.v

10. After deleting the top-level AXI files, a window will pop up stating the top module is invalid. Select Automatically pick new top module, and then click OK.

11. Next, double-click APB_Wrapper.v to open it and view the differences between the AXI top-files and the APB top-file.

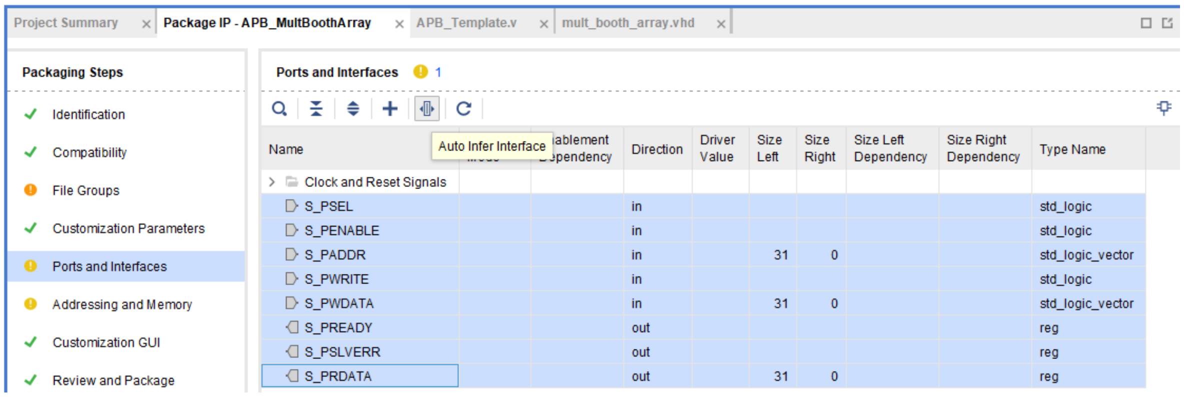
12. NOTE: If you rename APB_Wrapper.v, ensure you change the module name to match the file name.

13. Finally, ensure that APB_Wrapper.v is the top file as denoted by the three dots in a triangle seen in the image above, under step 6. If your file does not look like this, right-click on APB_Template and select "Set as Top."

14. Navigate to the Package IP tab. For all Packaging Steps, select "Merge Changes."

15. Navigate to the Ports and Interfaces step.

16. Now you will define the APB interface. Select all of the listed signals and click the Auto Infer Interface icon highlighted in the image below.



Project Summary | Package IP - APB_MultBoothArray | APB_Template.v | mult_booth_array.vhd

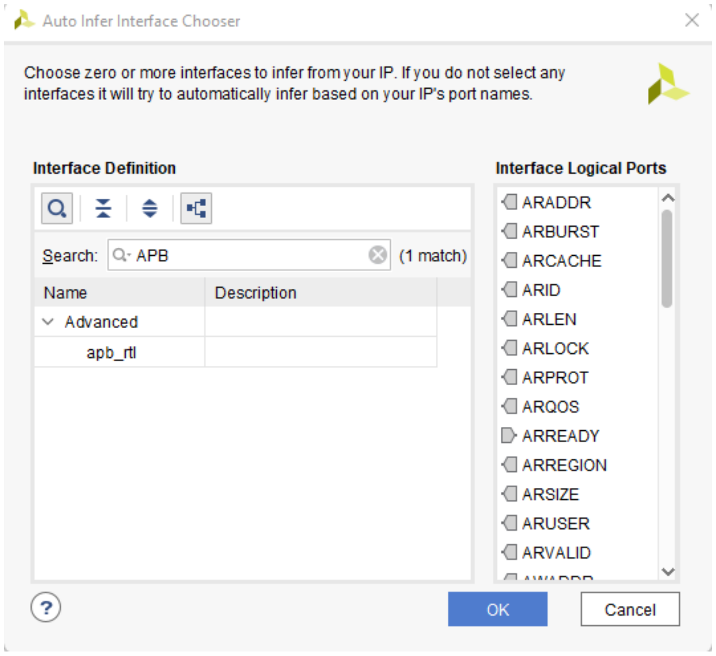
Packaging Steps
- ✓ Identification
- ✓ Compatibility
- ● File Groups
- ✓ Customization Parameters
- ● Ports and Interfaces
- ● Addressing and Memory
- ✓ Customization GUI
- ✓ Review and Package

Ports and Interfaces ● 1

Auto Infer Interface

Name		Enablement Dependency	Direction	Driver Value	Size Left	Size Right	Size Left Dependency	Size Right Dependency	Type Name
Clock and Reset Signals									
S_PSEL			in						std_logic
S_PENABLE			in						std_logic
S_PADDR			in		31	0			std_logic_vector
S_PWRITE			in						std_logic
S_PWDATA			in		31	0			std_logic_vector
S_PREADY			out						reg
S_PSLVERR			out						reg
S_PRDATA			out		31	0			reg

17. This will open a new window. Under the search bar, enter APB, expand the Advanced tab, and select apb_rtl as seen in the image below



Auto Infer Interface Chooser

Choose zero or more interfaces to infer from your IP. If you do not select any interfaces it will try to automatically infer based on your IP's port names.

Interface Definition

Search: APB (1 match)

Name	Description
Advanced	
apb_rtl	

Interface Logical Ports
- ARADDR
- ARBURST
- ARCACHE
- ARID
- ARLEN
- ARLOCK
- ARPROT
- ARQOS
- ARREADY
- ARREGION
- ARSIZE
- ARUSER
- ARVALID

OK | Cancel

18. A new tab will open, click OK. This creates the definition of S, an APB interface, under Ports and Interfaces, that contains all APB signals.

19. Follow the instructions from Task 1 to re-package the IP. Ensure that all packaging steps either have a check or either an orange or yellow circle.

20. Back on the block diagram, add APB_MultBoothArray to the design.

    (a) If you already added APB_MultBoothArray to your design, follow the instructions from Task 1 to upgrade your IP in steps 26-30.

21. Add a new IP called the AXI APB Bridge to your design.

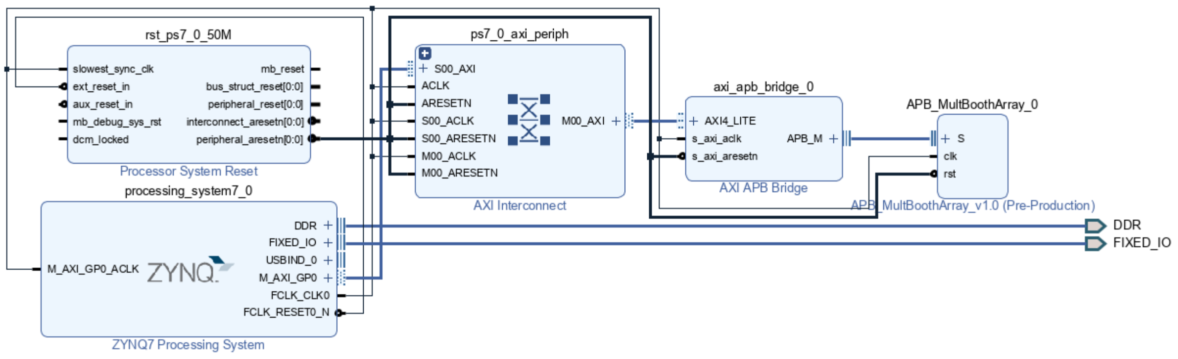
22. Double-click on this IP and set "Number of Slaves" to 1.

23. Manually connect the AXI APB Bridge's AXI4_LITE interface to the AXI Interconnect's M00_AXI interface.

24. Next, connect APB_MultBoothArray's APB interface, "S", to AXI APB Bridge's APB_M.

25. Run connection automation, then Regenerate the Layout.

26. At this point, your block design should look like the following image.



27. Finally, navigate to the Address Editor.

    (a) Expand the tab labelled "Unassigned", select and right-click on APB_MultBoothArray_0/S, and select "Assign."

    (b) This will automatically assign the APB to address 0x43c0_0000 for Memory-Mapped I/O.

    (c) Because the APB is no longer the native interface, the Memory-Mapped I/O interface is not automatically assigned.

28. Validate the design to ensure there are no errors.

29. Generate the bitstream.

30. Follow the instructions from Task 2 to export the hardware design, open Vitis, and create a new Hello World program called HelloWorld_Task2.

31. Add the code from this Module's Task 2 folder, helloworld.c to HelloWorld_Task2_system/HelloWorld_Task2/src/helloworld.c.

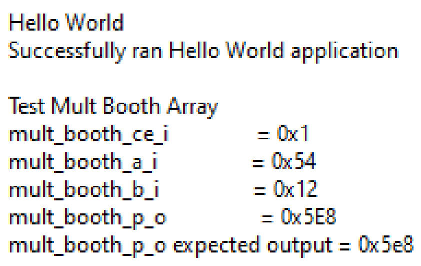
32. Finally, run the Vitis project by right-clicking on the blue "HelloWorld_Task2_system, and selecting Run As/1. Launch Hardware.

    (a) If you do not close, then re-open Vitis you will receive two new pop-up windows.

    (b) The first window will say "Conflict, Existing launch configuration..." Click Yes.

    (c) The second window that will pop up will say "Error Launching Program." Click OK, then Run As/1. Launch Hardware again.

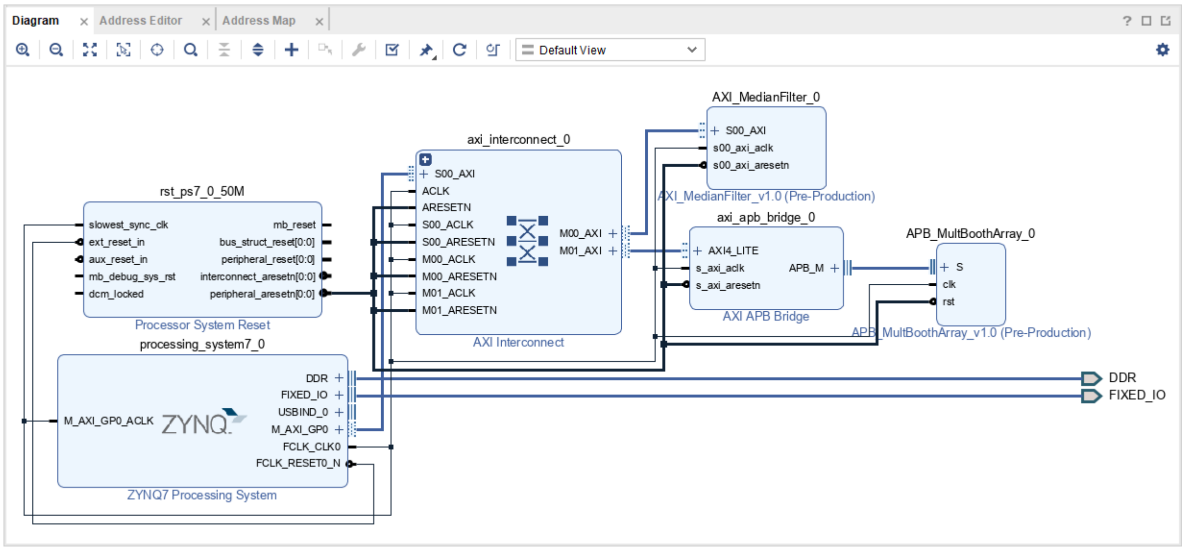
33. The output shown in the image below will display on your terminal.



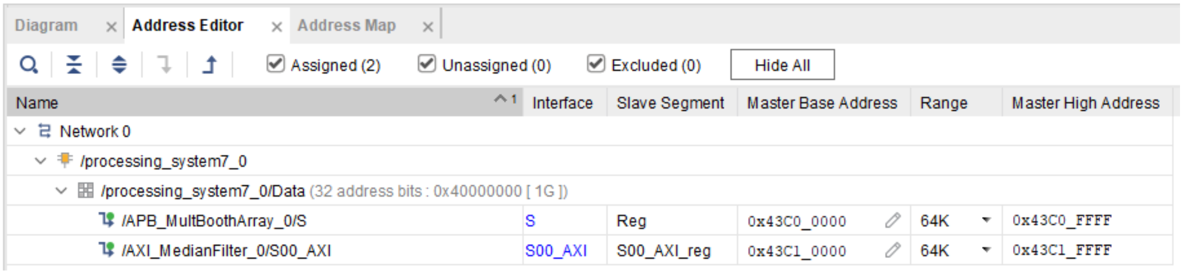
```
Hello World
Successfully ran Hello World application

Test Mult Booth Array
mult_booth_ce_i            = 0x1
mult_booth_a_i            = 0x54
mult_booth_b_i            = 0x12
mult_booth_p_o            = 0x5E8
mult_booth_p_o expected output = 0x5e8
```

3.3 Task 3: Communicating with Multiple Peripherals

1. To begin, open the project created in the previous task.
2. Double-click on the AXI Interconnect and modify ”Number of Master Interfaces” from one to two.
3. In addition to the APB peripheral already hooked up, add the AXI_MedianFilter IP.
4. Run block automation to automatically connect the AXI IP to the interconnect.
5. Regenerate the layout.
6. At this point, your block design should look like the following image.



7. Before generating the bitstream, open the tab titled Address Editor.
8. Inside the address editor, ensure that APB_MultBoothArray is set to 0x43c0_0000 and AXI_MedianFilter is set to 0x43c1_0000, as seen in the image below.



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/APB_MultBoothArray_0/S	S	Reg	0x43C0_0000	64K	0x43C0_FFFF
/AXI_MedianFilter_0/S00_AXI	S00_AXI	S00_AXI_reg	0x43C1_0000	64K	0x43C1_FFFF

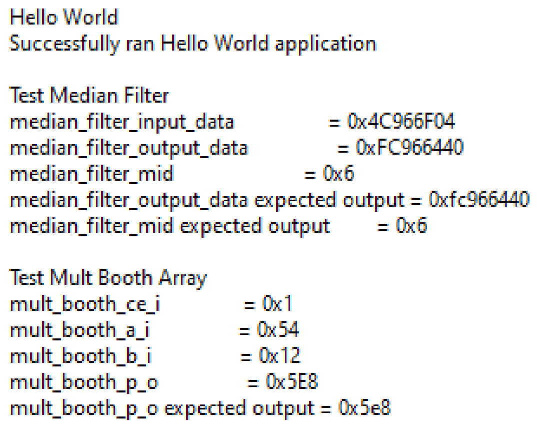
9. Generate the bitstream

10. Follow the steps from Task 1 and 2 to create a Hello World application.

11. As done in Task 1 and 2, copy the helloworld.c code from this Module's Task 3 folder to HelloWorld_Task2_system/HelloWorld_Task2/src/helloworld.c.

12. Follow the steps from Task 1 and 2 to build the project and run.

13. You should see the outputs from both Task 1 and 2, like in the image below.



```
Hello World
Successfully ran Hello World application

Test Median Filter
median_filter_input_data              = 0x4C966F04
median_filter_output_data              = 0xFC966440
median_filter_mid                     = 0x6
median_filter_output_data expected output = 0xfc966440
median_filter_mid expected output        = 0x6

Test Mult Booth Array
mult_booth_ce_i            = 0x1
mult_booth_a_i             = 0x54
mult_booth_b_i             = 0x12
mult_booth_p_o             = 0x5E8
mult_booth_p_o expected output = 0x5e8
```