Lab 2 – Designing an AXI Accelerator for Matrix Multiplication

In this lab you will be designing an AXI accelerator for matrix multiplication for the Xilinx Zynq device.

Requirements:

- The know-how gained during the previous lab and exercise.
- Knowing AXI Stream communication protocol.

Tools:

- Xilinx Vivado HLx Edition 2018.3[1]

Intended Learning outcomes:

- Creating a custom IP
- IP-based design
- Developing software for FPGA designs using the Xilinx Software Development Kit (SDK)

Assessment:

- Verify your design using the provided test bench.
- You have to demonstrate your solution, and explain your design to the lab assistants in one of the following lab sessions:
  - Thursday May 9th, 13:15 – 17:00, room 2315
  - Friday, May 24th, 13:15 – 17:00, room 2315
- Submit the Verilog code for your solution afterwards on the student portal.

[1] The description contains some screenshots taken using Vivado v2015.4, but the description applies also to the recent version.

Matrix Multiplication

In this lab we consider matrix multiplication, a basic mathematical operation that is important in a wide variety of applications and algorithms (e.g., in scientific computing or machine learning). Matrix multiplication is a good candidate for acceleration, since the matrices needed in practice are often huge, and processing them can be a bottleneck.

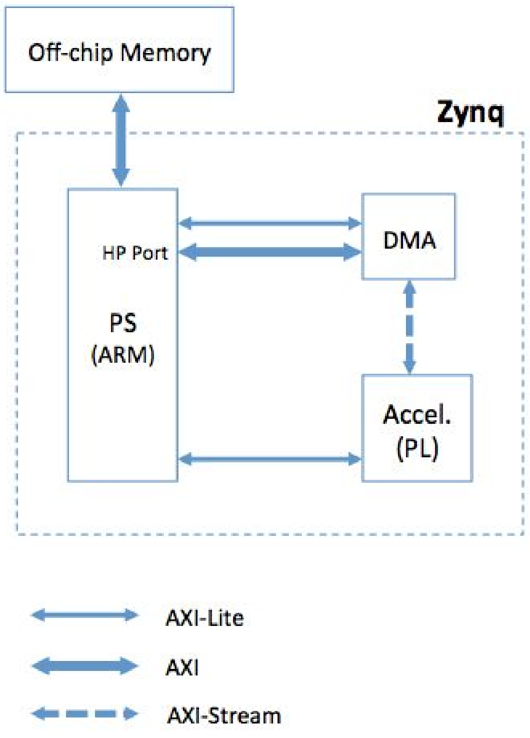
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix}$$

An unoptimised sequential implementation of matrix multiplication is as follows:

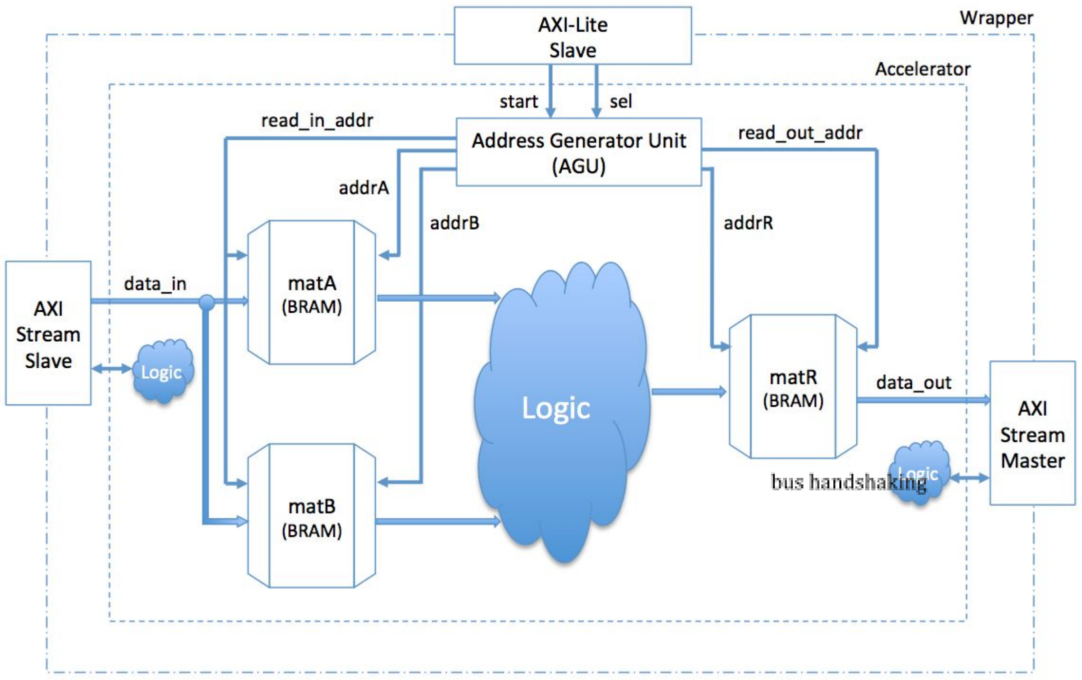
```
for (row = 1; row <= 4; ++row)
        for (col = 1; col <= 4; ++col) {
                r[row][col] = 0;
                for (tmp = 1; tmp <= 4; ++tmp)
                        r[row][col] = a[row][tmp] * b[tmp][col] + r[row][col]
        }
```

System Overview

- PS: Processing System (ARM Cores)
- PL: Programmable Logic, i.e. the FPGA fabrique (IPs and buses)
- AXI-Lite: simple AXI bus used by the processor to configure the IPs (memory-mapped)
- AXI: high performance AXI bus (memory-mapped)
- AXI-Stream: streaming bus (address-less)
- HP: High Performance port, used by the Direct Memory Access controller to access the off-chip memory



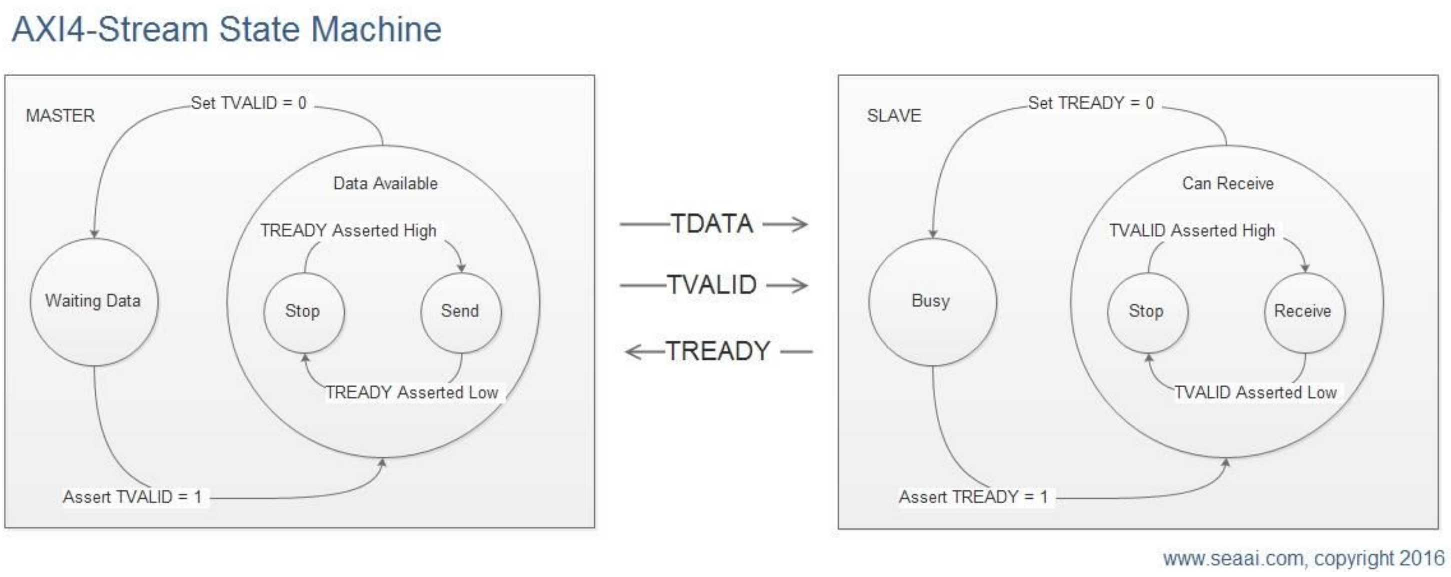
Proposed Architecture for the MatMul Accelerator



- sel: used by the PS to select matA or matB before sending the matrix to the accelerator (memory-mapped via the axi-lite config bus)
- start: used by the PS to start the accelerator after transferring the matrices (memory-mapperd via the axi-lite config bus)
- AGU: Address Generator Unit; used to generate addresses to:
    - Fill-in the input matrices
    - Read from the matrices during multiplication
    - Write the result to the output matrix
    - Sending the result to the processor
- matX: input and output matrices implemented as dual-port block RAMs
- Logic: used to implement the data path for the multiplication and also handshaking signals for the AXI Stream buses

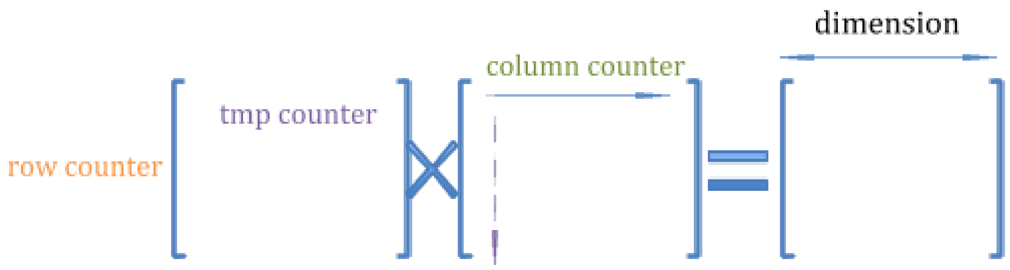
AXI-Stream State Machine

We will be using the following handshaking protocol to receive and send data from and through the AXI-Stream bus (can be implemented as a state machine)

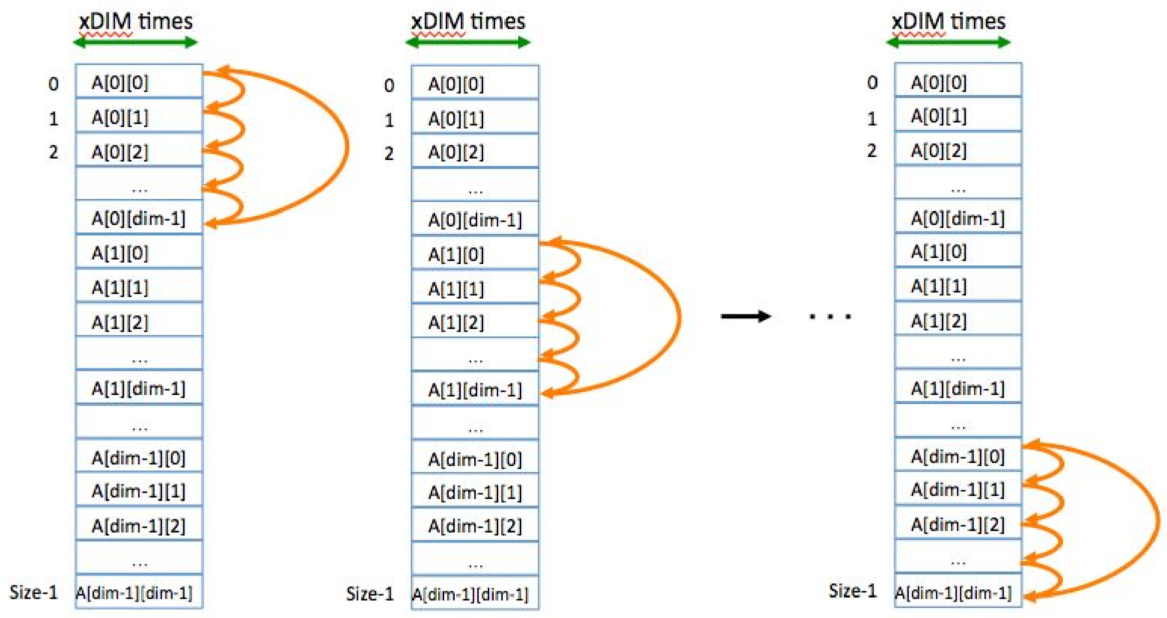


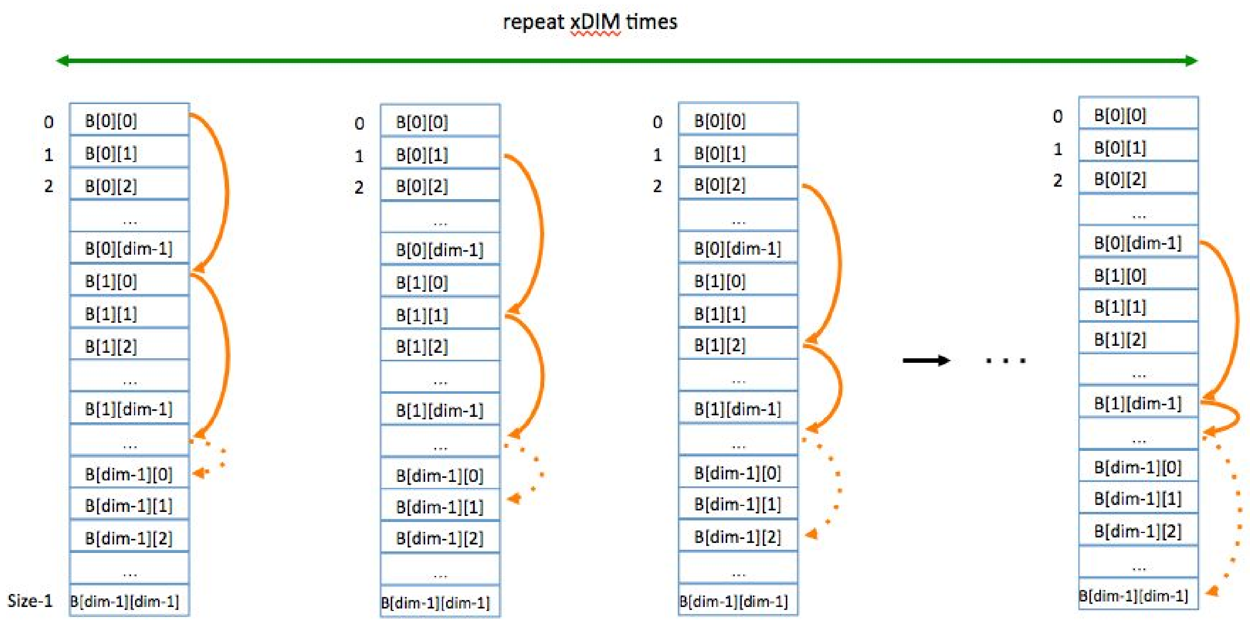
More on AGU

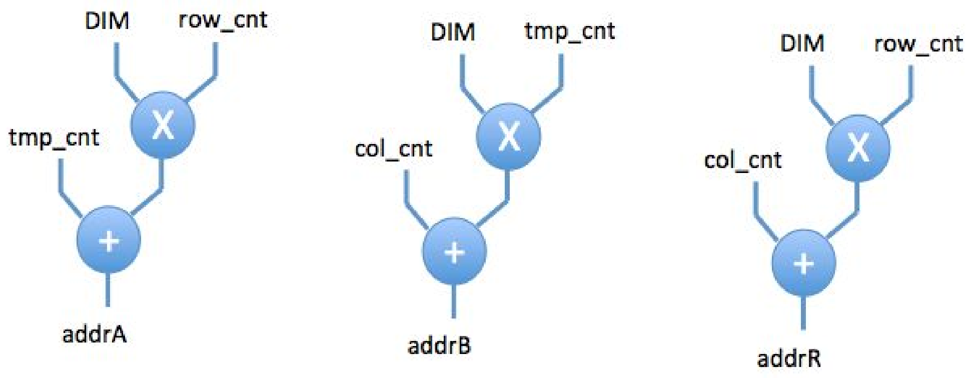
```
for (row = 1; row <= 4; ++row)
        for (col = 1; col <= 4; ++col) {
                r[row][col] = 0;
                for (tmp = 1; tmp <= 4; ++tmp)
                        r[row][col] = a[row][tmp] * b[tmp][col] + r[row][col]
        }
```



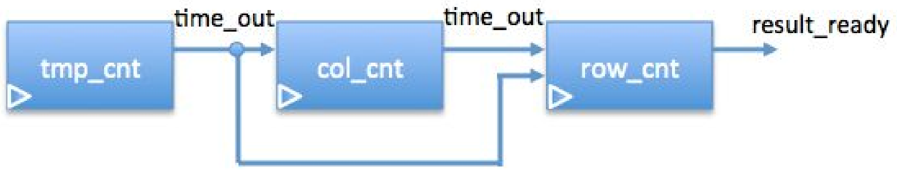
- size: dim*dim
- row counter: 0 … dim-1
- column counter: 0 … dim-1
- temporary counter: 0 … dim-1
- address: 0 … size-1


xDIM times
0
1
2
Size-1
A[0][0]
A[0][1]
A[0][2]
...
A[0][dim-1]
A[1][0]
A[1][1]
A[1][2]
...
A[1][dim-1]
...
A[dim-1][0]
A[dim-1][1]
A[dim-1][2]
...
A[dim-1][dim-1]

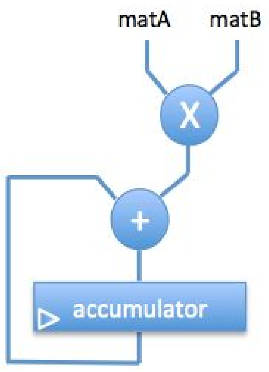

repeat xDIM times
0
1
2
Size-1
B[0][0]
B[0][1]
B[0][2]
...
B[0][dim-1]
B[1][0]
B[1][1]
B[1][2]
...
B[1][dim-1]
...
B[dim-1][0]
B[dim-1][1]
B[dim-1][2]
...
B[dim-1][dim-1]



read_addr_in and read_addr_out that are used to fill in the matrices and to read the result out are regular sequence counters in the range of 0 … size-1.



Multiplier and Accumulator (MAC) Unit



Inferring Dual-Port Block RAMs from HDL Code

```
always @(posedge clka) begin
        if (wea)
                BRAM[addra] <= dina;
        if (enb)
                ram_data <= BRAM[addrb];
end

assign doutb = ram_data;
```

Your Task

You are required to design the accelerator based on the aforementioned components. Moreover, you should make sure that all the components are correctly synchronized. This is achieved by designing the correct control logic that ensures correct timing between all the components.

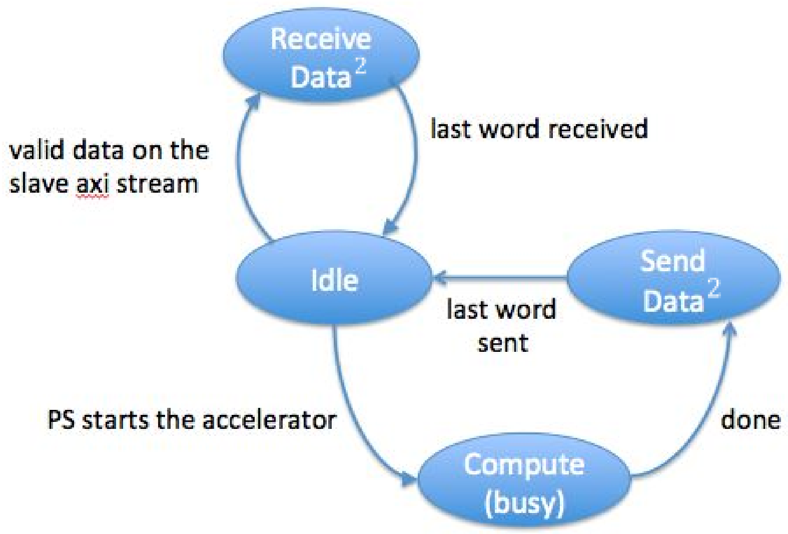
You can open the design files in Vivado HLx and complete the design according to the instructions in this manual (you will need to complete the TODO parts in the skeleton files).

Finally, you need to test your design. It might be easier to simulate your stand-alone accelerator in a separate file since we are using IP-based design. You can prepare a test bench yourself or use the provided one "`lab2_matmul/src/mat_mul_tb.v`". The provided test bench should produce the output "`HW/SW result match!`" when simulated to completion.

If you are interested in starting the design from the scratch, you can refer to Appendix I.
You can learn how to programme the FPGA and run your software via the SDK in Appendix II.

Some Hints

Think of the following state machine (algorithm) when designing the accelerator.[2]

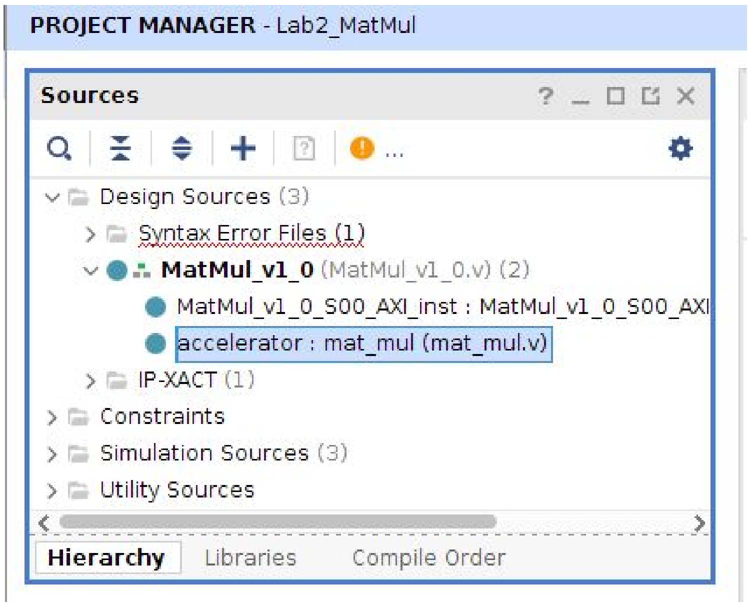


[2] Also notice the axi stream protocol (page 4) when sending and receiving data.

Opening the Skeleton Files in Vivado

After downloading the zip file and extracting it in a folder, open the project for the accelerator by opening `lab2_matmul/Lab2_MatMul/Lab2_MatMul.xpr` in Vivado.

When the project is opened, double-click on **`mat_mul.v`** file in the Sources pane, and complete the TODO parts (you can find and edit the same file under `lab2_matmul/src` as well).


PROJECT MANAGER - Lab2_MatMul
Sources
Design Sources (3)
Syntax Error Files (1)
MatMul_v1_0 (MatMul_v1_0.v) (2)
MatMul_v1_0_S00_AXI_inst : MatMul_v1_0_S00_AXI
accelerator : mat_mul (mat_mul.v)
IP-XACT (1)
Constraints
Simulation Sources (3)
Utility Sources
Hierarchy
Libraries
Compile Order

Test and Debug your Design using the included Test Bench

After completing your modifications, you should test your design by simulating it. A possible test bench is already provided in the file `mat_mul_tb.v`, which will also be used to verify that your design works. You should also study the provided test bench carefully.

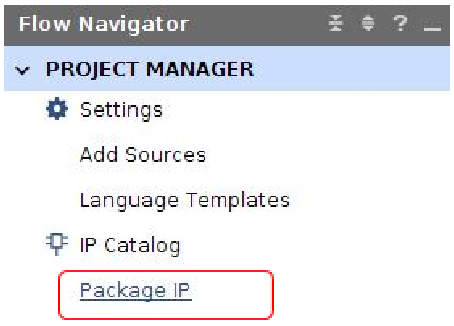
You should be able to explain to the TAs:

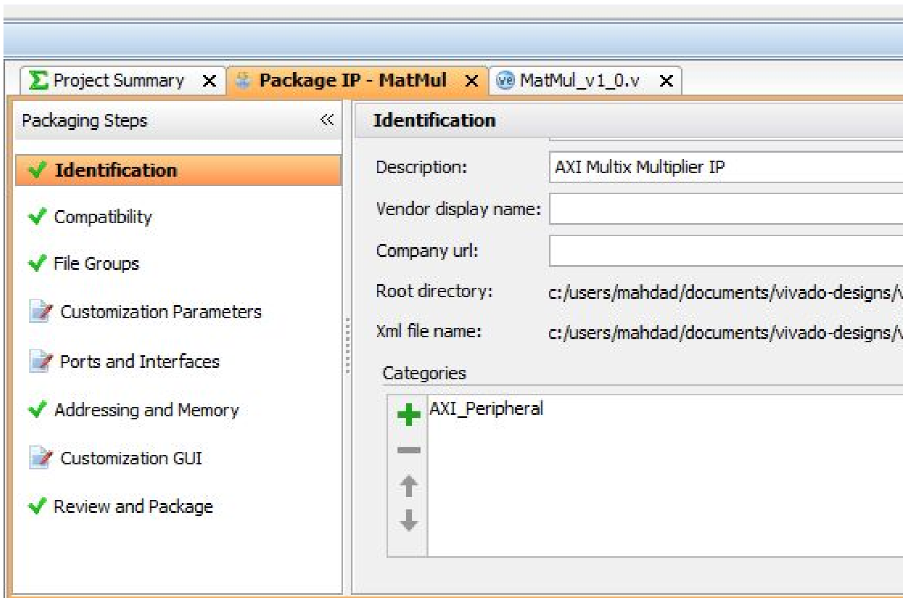
- the stimulus file that you are using (either your own or the provided one),
- how to interpret the results.

If launching the behavioural simulation fails, make sure that the library “xil_defaultlib” is selected in the “Source File Properties” pane for all source files!

Optional: Re-Package IP

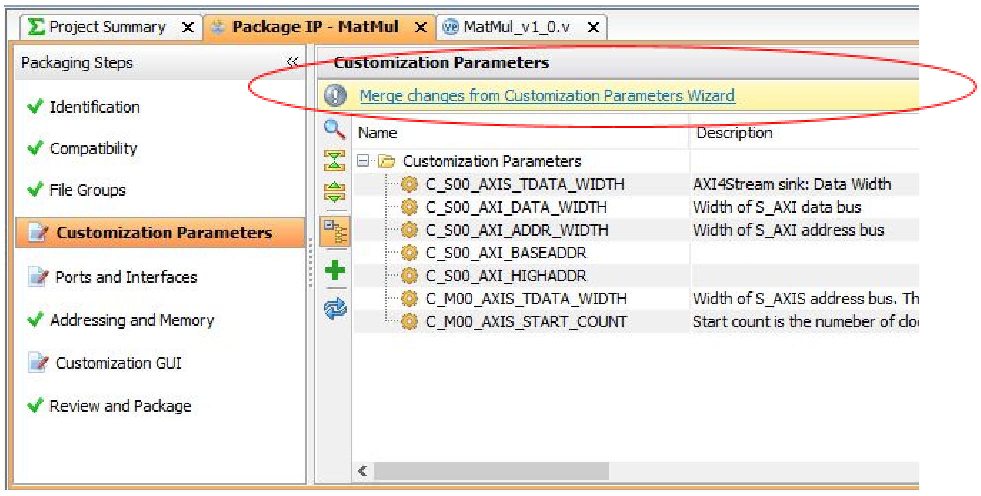
Once your modifications are complete, save your work by selecting the “Package IP” tab.





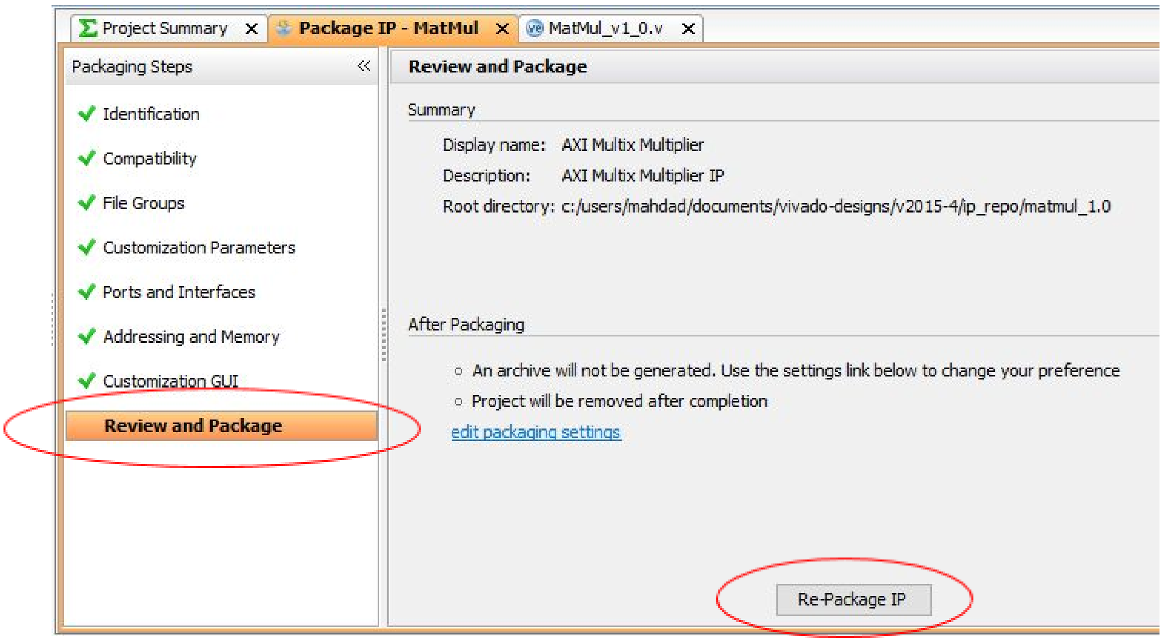
Description: AXI Multix Multiplier IP
Vendor display name:
Company url:
Root directory: c:/users/mahdad/documents/vivado-designs/
Xml file name: c:/users/mahdad/documents/vivado-designs/

Select the steps without a green check mark. Select the wizard to resolve the issues and having green check marks for all the steps (having some warnings might be OK, as long as you understand the reason)..



Name	Description
Customization Parameters	
C_S00_AXIS_TDATA_WIDTH	AXI4Stream sink: Data Width
C_S00_AXI_DATA_WIDTH	Width of S_AXI data bus
C_S00_AXI_ADDR_WIDTH	Width of S_AXI address bus
C_S00_AXI_BASEADDR	
C_S00_AXI_HIGHADDR	
C_M00_AXIS_TDATA_WIDTH	Width of S_AXIS address bus. Th
C_M00_AXIS_START_COUNT	Start count is the numeber of clo

Select "Review and Package" and click on the "Re-Package IP", you should receive a notification that the packaging is finished successfully.



Review and Package

Summary

Display name: AXI Multix Multiplier

Description: AXI Multix Multiplier IP

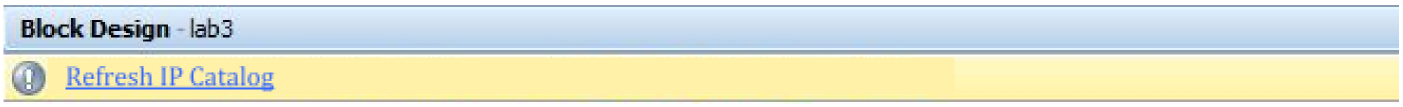
Root directory: c:/users/mahdad/documents/vivado-designs/v2015-4/ip_repo/matmul_1.0

After Packaging

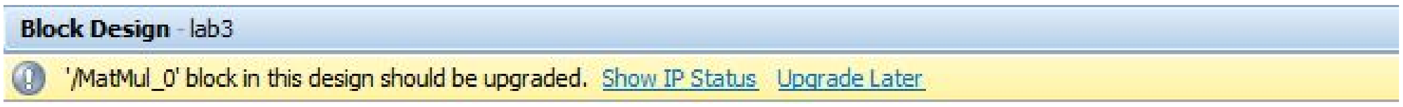
- An archive will not be generated. Use the settings link below to change your preference
- Project will be removed after completion

You can now open the main project which uses the accelerator, located under `lab2/lab2/lab2.xpr`.

Click on the "Refresh IP Catalog" if the yellow warning bar appears at the top of the window.



If the following message appears, the easiest way to solve the problem would be to right-click on the MatMul IP block and delete it, then instantiating it again.



'/MatMul_0' block in this design should be upgraded. Show IP Status Upgrade Later

To instantiate the IP, click on "Add IP" and type the IP name (MatMul) , then click on your IP.

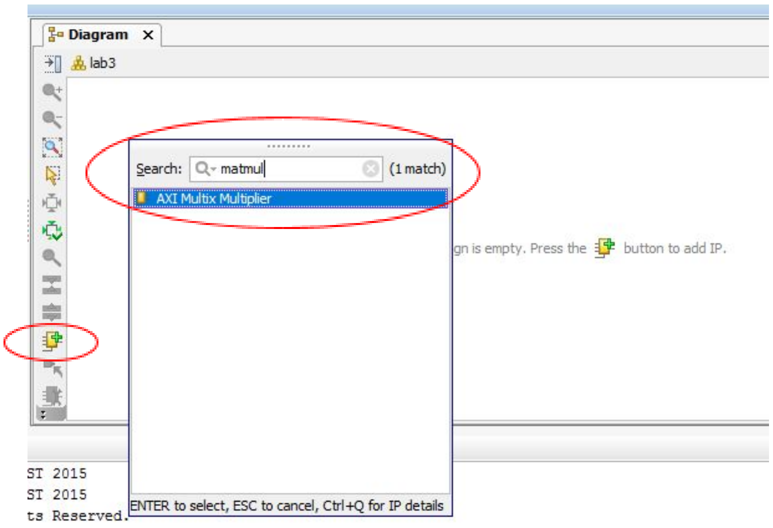
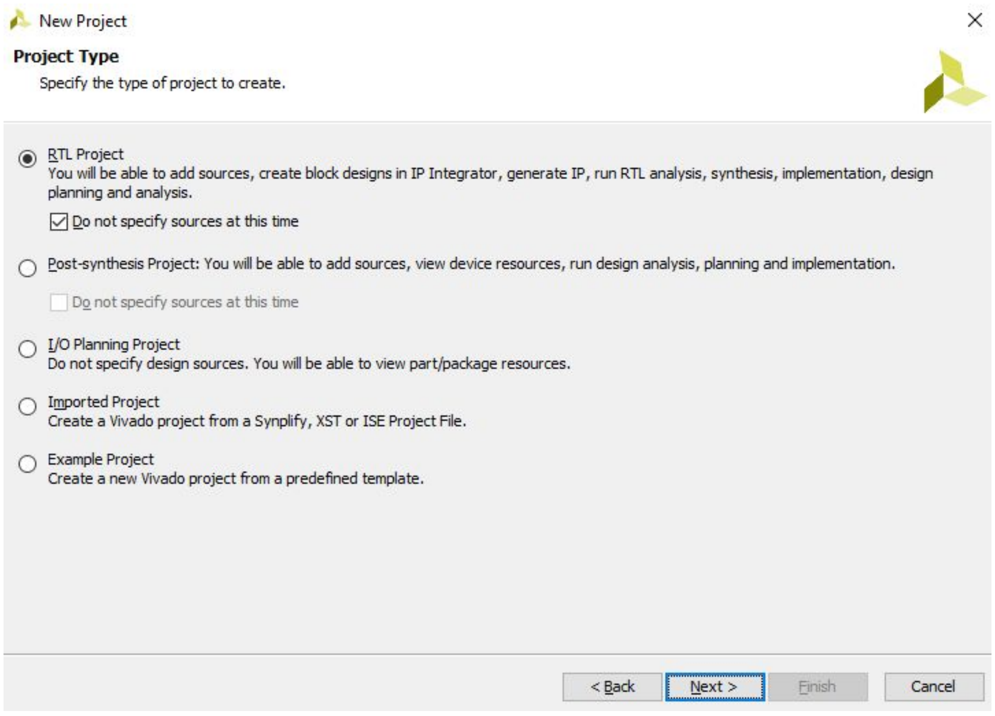


Diagram
lab3
Search: matmul
(1 match)
AXI Multix Multiplier
gn is empty. Press the button to add IP.
ENTER to select, ESC to cancel, Ctrl+Q for IP details

You can then continue from page 17 of this manual (Run Connection Automation).

Appendix I – Create a Custom AXI IP

Create a new project in Vivado. Choose the board ZC702 or ZC706.



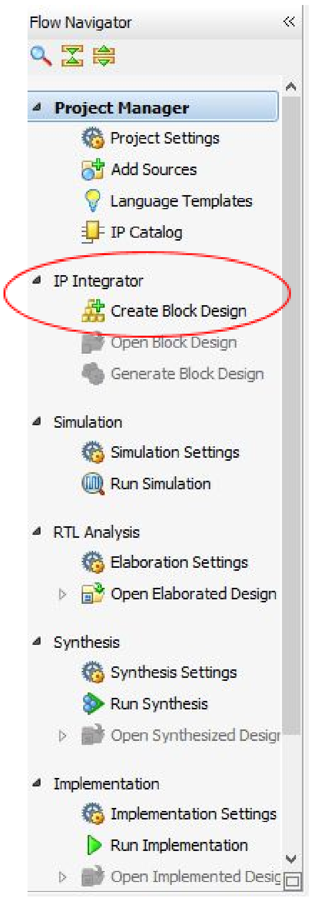
New Project

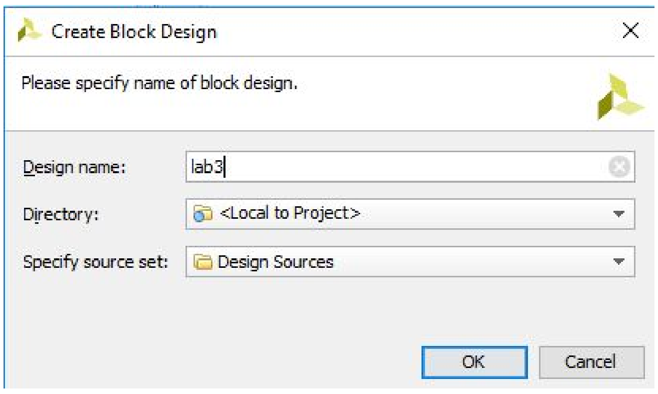
Project Type

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.

Create a block design using the IP integrator.



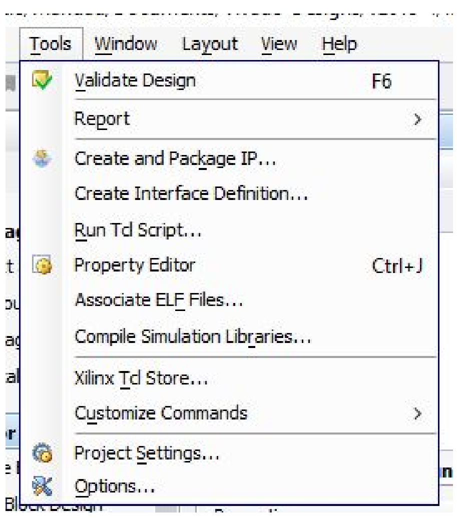


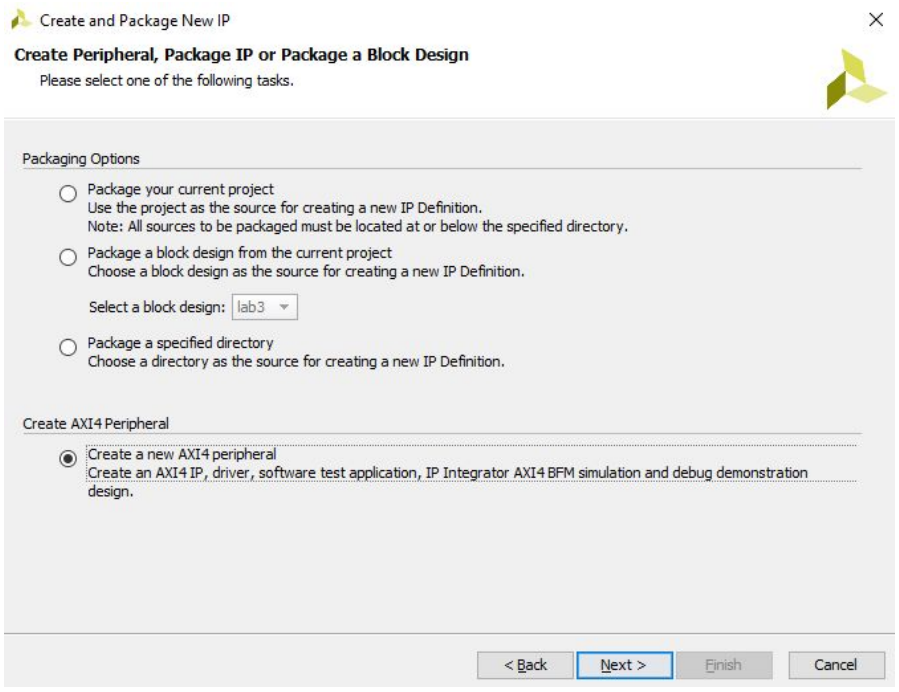
Create Block Design

Please specify name of block design.

Design name:	lab3
Directory:	<Local to Project>
Specify source set:	Design Sources

Select "Tools → Create and Package IP …", then select "Create a new AXI4 peripheral". Give your IP a name and description.




Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

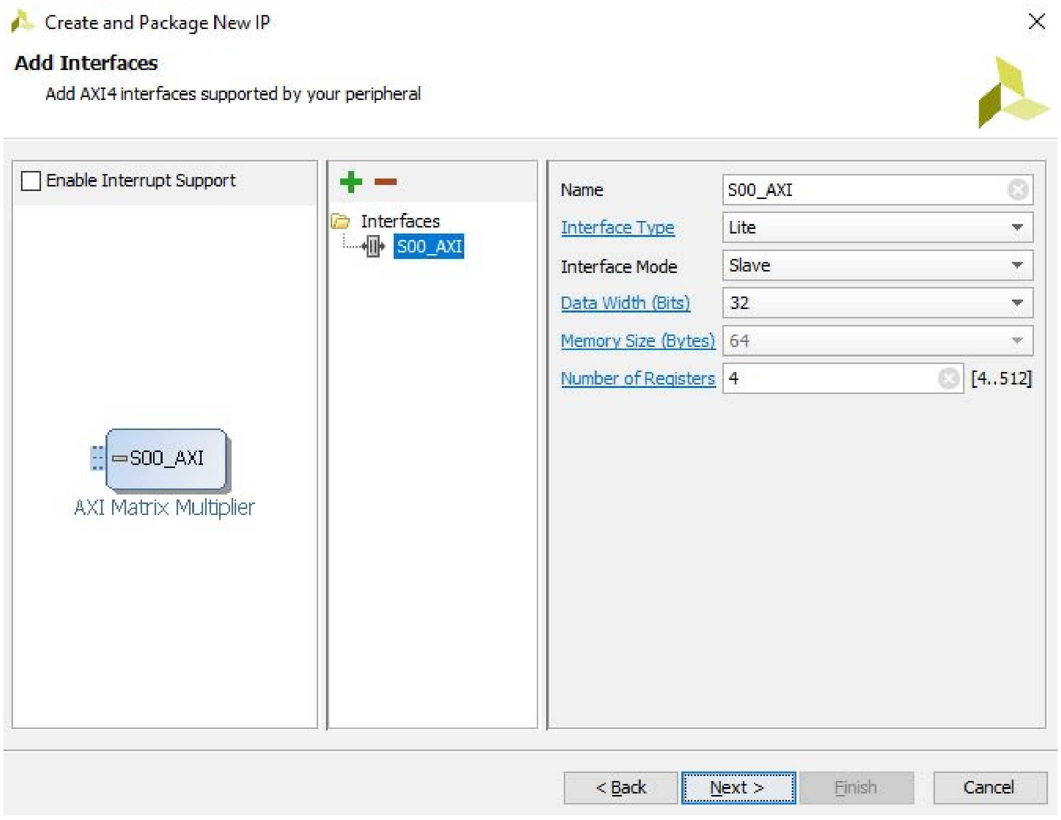
Packaging Options

- ○ Package your current project
  Use the project as the source for creating a new IP Definition.
  Note: All sources to be packaged must be located at or below the specified directory.
- ○ Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
  Select a block design: lab3
- ○ Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

- ● Create a new AXI4 peripheral
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 BFM simulation and debug demonstration design.

We will need three interfaces for the MatMul IP: one AXI-Lite for configuration (slave), and two AXI-Stream (one master and one slave). The AXI-Lite is created by default. Add the AXI-Stream interfaces by clicking on the green + sign. Change the type to "Stream". Select the proper mode (slave or master).


Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

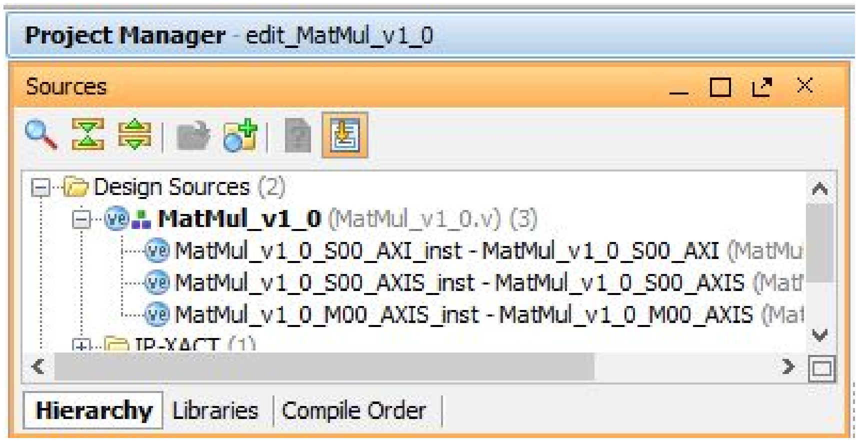
☐ Enable Interrupt Support

Interfaces
S00_AXI

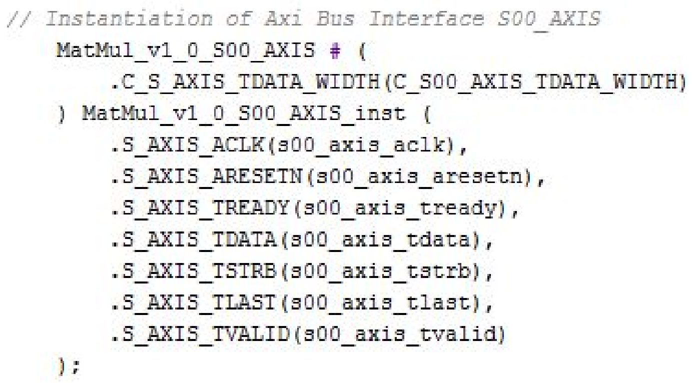
Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

Select "Edit IP" and click "Finish". A new project window opens.

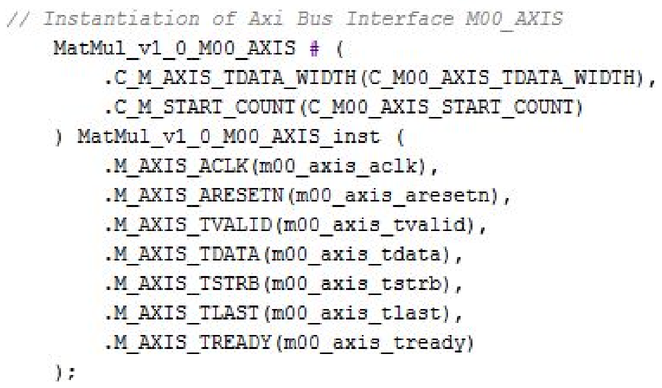
The top module for the IP is created. It contains three Verilog files that correspond to the three port instances.


Project Manager - edit_MatMul_v1_0
Sources
Design Sources (2)
MatMul_v1_0 (MatMul_v1_0.v) (3)
MatMul_v1_0_S00_AXI_inst - MatMul_v1_0_S00_AXI (MatMu
MatMul_v1_0_S00_AXIS_inst - MatMul_v1_0_S00_AXIS (Mat
MatMul_v1_0_M00_AXIS_inst - MatMul_v1_0_M00_AXIS (Mat
IP-XACT (1)
Hierarchy
Libraries
Compile Order

Double-click on the top module (MatMul_V1_0). Inside the file you can observe the instantiation of the interfaces. Vivado implements the interfaces as FIFO streams. However, we intend to use Block RAMs. As a result, we keep the interface signals in the top module, but delete the instantiations of Axi Bus Interfaces S00_AXIS and M00_AXIS (stream slave and master). We will later implement correct AXI stream bus handshaking logic and connect them to the corresponding interface signals in the top module. Note that the design hierarchy updates after you delete the instances and save your design.



```
// Instantiation of Axi Bus Interface S00_AXIS
    MatMul_v1_0_S00_AXIS # (
        .C_S_AXIS_TDATA_WIDTH(C_S00_AXIS_TDATA_WIDTH)
    ) MatMul_v1_0_S00_AXIS_inst (
        .S_AXIS_ACLK(s00_axis_aclk),
        .S_AXIS_ARESETN(s00_axis_aresetn),
        .S_AXIS_TREADY(s00_axis_tready),
        .S_AXIS_TDATA(s00_axis_tdata),
        .S_AXIS_TSTRB(s00_axis_tstrb),
        .S_AXIS_TLAST(s00_axis_tlast),
        .S_AXIS_TVALID(s00_axis_tvalid)
    );
```



```
// Instantiation of Axi Bus Interface M00_AXIS
    MatMul_v1_0_M00_AXIS # (
        .C_M_AXIS_TDATA_WIDTH(C_M00_AXIS_TDATA_WIDTH),
        .C_M_START_COUNT(C_M00_AXIS_START_COUNT)
    ) MatMul_v1_0_M00_AXIS_inst (
        .M_AXIS_ACLK(m00_axis_aclk),
        .M_AXIS_ARESETN(m00_axis_aresetn),
        .M_AXIS_TVALID(m00_axis_tvalid),
        .M_AXIS_TDATA(m00_axis_tdata),
        .M_AXIS_TSTRB(m00_axis_tstrb),
        .M_AXIS_TLAST(m00_axis_tlast),
        .M_AXIS_TREADY(m00_axis_tready)
    );
```

PS configures the accelerator by writing to the internal control registers of the accelerator via the AXI-Lite slave interface. For instance, before writing a matrix into the accelerator, PS first sets the appropriate bit in the control register of the accelerator to indicate which matrix (A or B) is being transferred to the accelerator. Moreover, when both matrices are transferred, PS starts the accelerator by setting the start bit in the accelerator's control register. We can use the register at the base address of the accelerator as the control register. Each IP has a base address, which is used by the PS to communicate with the IPs. IPs are memory-mapped, which means that they are connected to the bus and the PS can simply read from or write to them by issuing requests for those addresses on the bus.

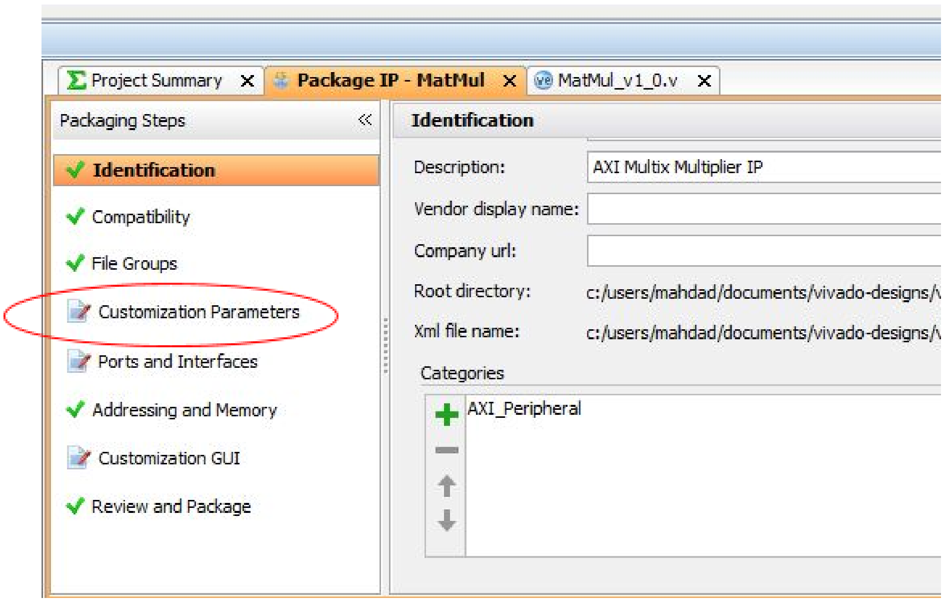
By default there are four slave registers in the AXI-Lite interface (slv_reg0 to slv_reg3). We will use the slv_reg0 as the control register. The address of this register is the same as the base address of the IP. Vivado manages all the base addresses and makes them available to the SDK in a header file named "xparameters.h". We will use this header file to communicate to the IPs from the PS by reading from and writing to those base addresses.

As a design choice, we consider slv_reg0[0] as the matrix select bit, and the slv_reg0[1] as the start bit. We need to create user-defined ports and make this bits available in the ports of the AXI-Lite module. We use these bits in the top module to implement the accelerator's logic.

N.B. as a design choice, we choose that the "start" bit should be asserted only for one clock cycle, i.e. it should be cleared after it is set.

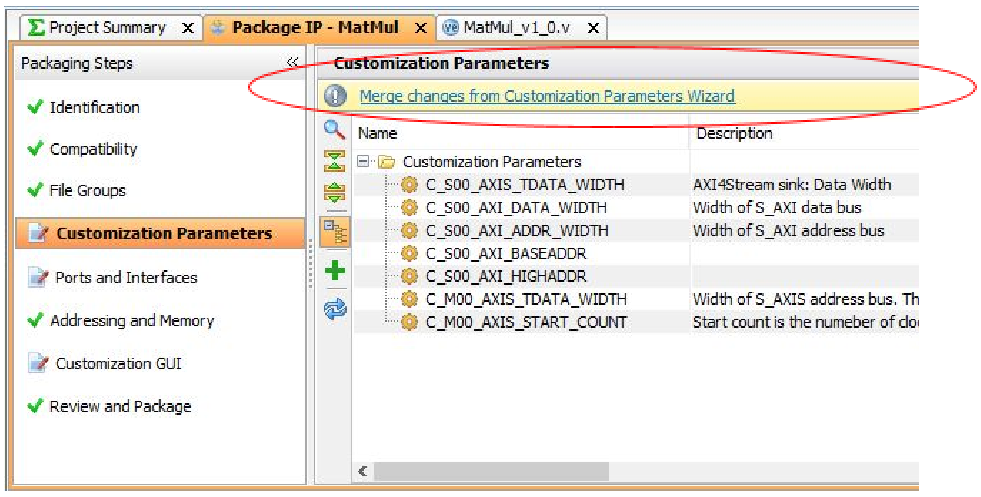
You can browse through the lab skeleton files and observe how these modifications have been made (look for comments including the word "user").

Once your design is complete, save the project and select the "Package IP" tab.



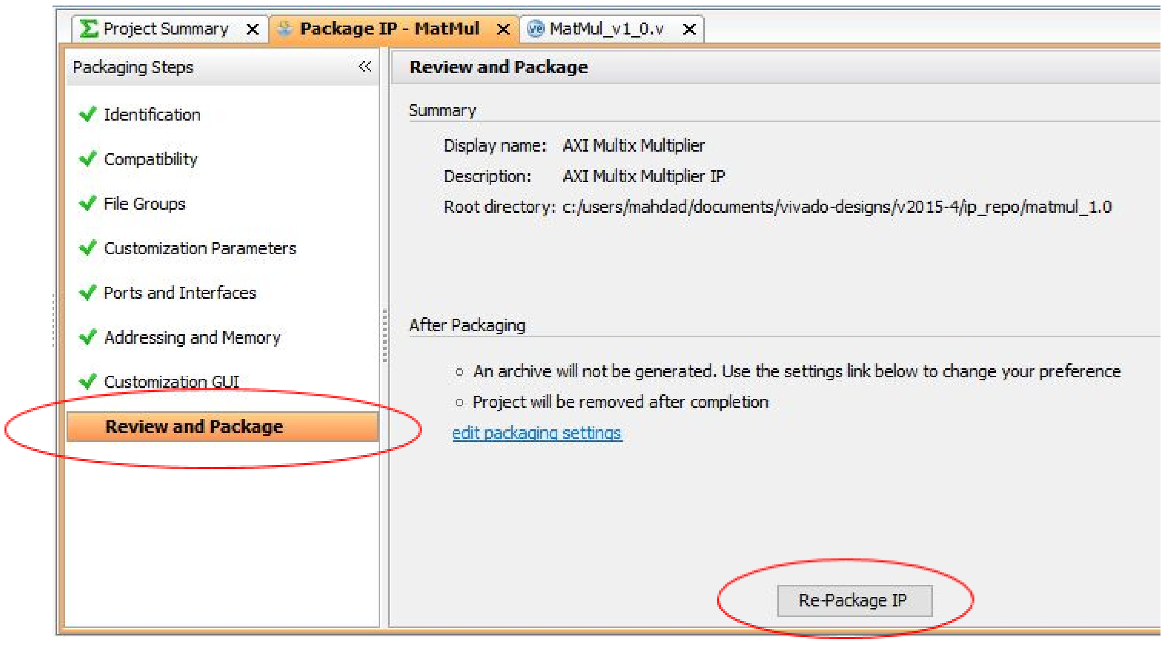
Description: AXI Multix Multiplier IP
Vendor display name:
Company url:
Root directory: c:/users/mahdad/documents/vivado-designs/
Xml file name: c:/users/mahdad/documents/vivado-designs/
AXI_Peripheral

Select the steps without a green check mark. Select the wizard to resolve the issues and having green check marks for all the steps.



Name	Description
Customization Parameters	
C_S00_AXIS_TDATA_WIDTH	AXI4Stream sink: Data Width
C_S00_AXI_DATA_WIDTH	Width of S_AXI data bus
C_S00_AXI_ADDR_WIDTH	Width of S_AXI address bus
C_S00_AXI_BASEADDR	
C_S00_AXI_HIGHADDR	
C_M00_AXIS_TDATA_WIDTH	Width of S_AXIS address bus. Th
C_M00_AXIS_START_COUNT	Start count is the numeber of clo

Select “Review and Package” and click on the “Re-Package IP”, then select “yes” to close the project. The initial project window appears.



Review and Package

Summary

Display name: AXI Multix Multiplier
Description: AXI Multix Multiplier IP
Root directory: c:/users/mahdad/documents/vivado-designs/v2015-4/ip_repo/matmul_1.0

After Packaging

- An archive will not be generated. Use the settings link below to change your preference
- Project will be removed after completion

Now you can add the IP that you just created. Click on “add IP” and type your IP name. Click on your IP name and it will be instantiated in the block design.

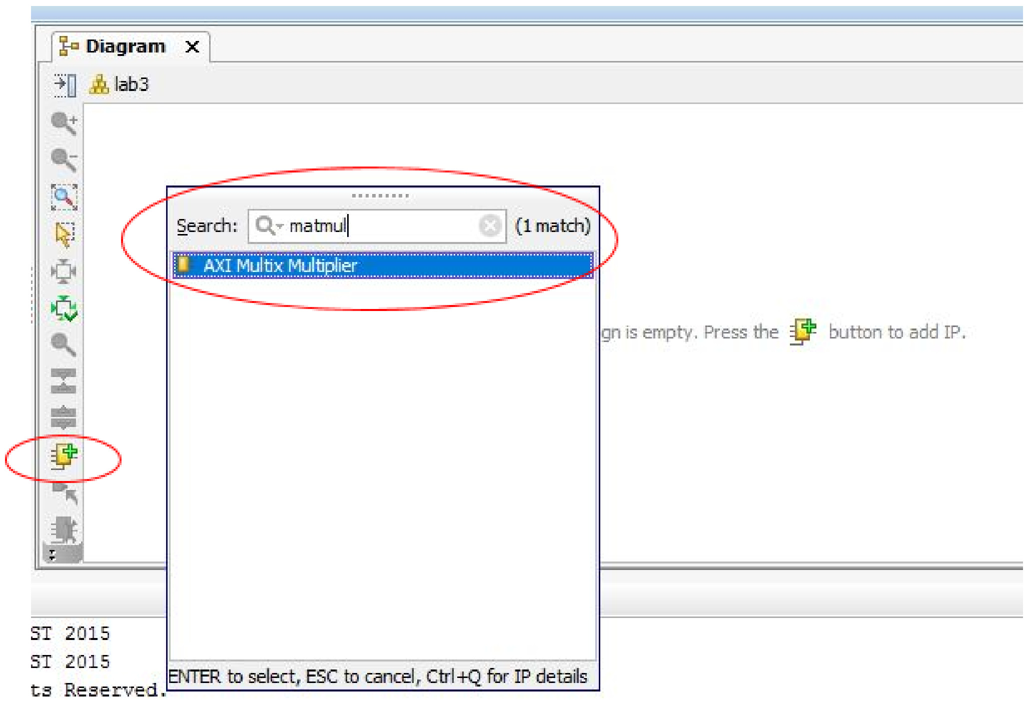
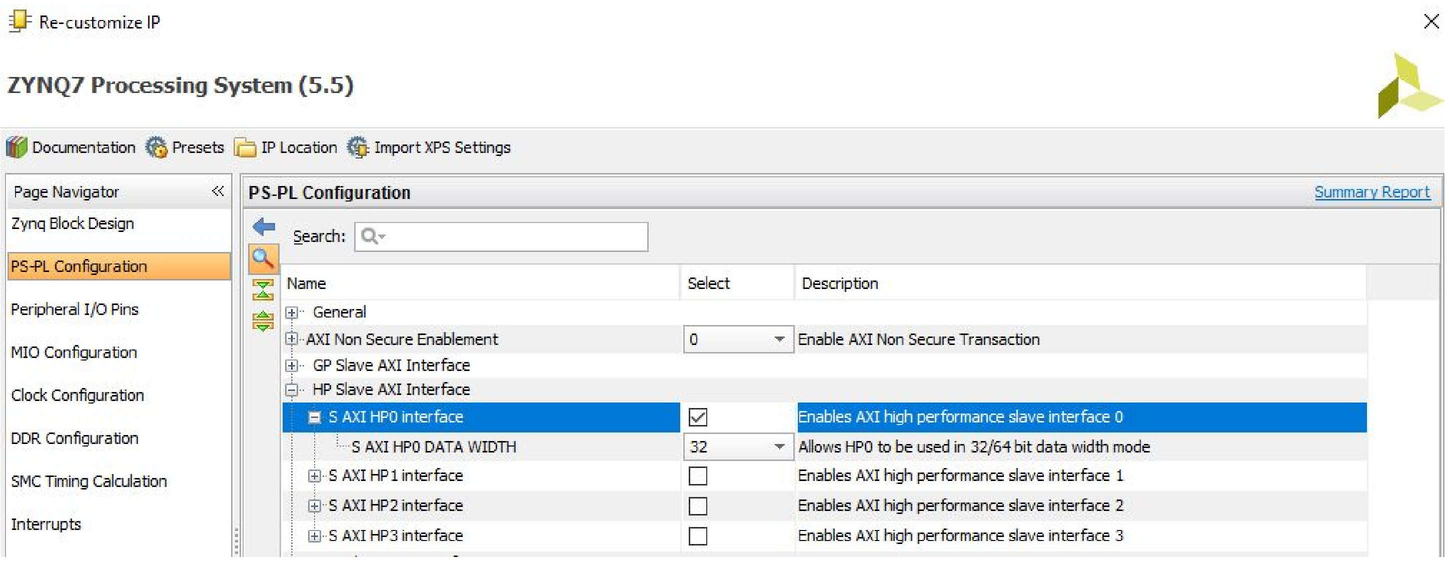


Diagram
lab3
Search: matmul
(1 match)
AXI Multix Multiplier
gn is empty. Press the button to add IP.
ENTER to select, ESC to cancel, Ctrl+Q for IP details

```
ST 2015
ST 2015
ts Reserved.
```

Similarly add the Zynq Processing System, AXI Direct Memory Access, and an AXI Timer.

Double-click on the Zynq IP (or right-click and choose customize block). Go to "PS-PL" configuration and enable the HP0 AXI interface (be aware that you need to invalidate the cache manually if you are using this port. Alternatively, you can use the Accelerator Coherency Port (ACP), which handles the data coherency). It will be used by the DMA to access the external memory. Choose the data width to be 32.


Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation · Presets · IP Location · Import XPS Settings

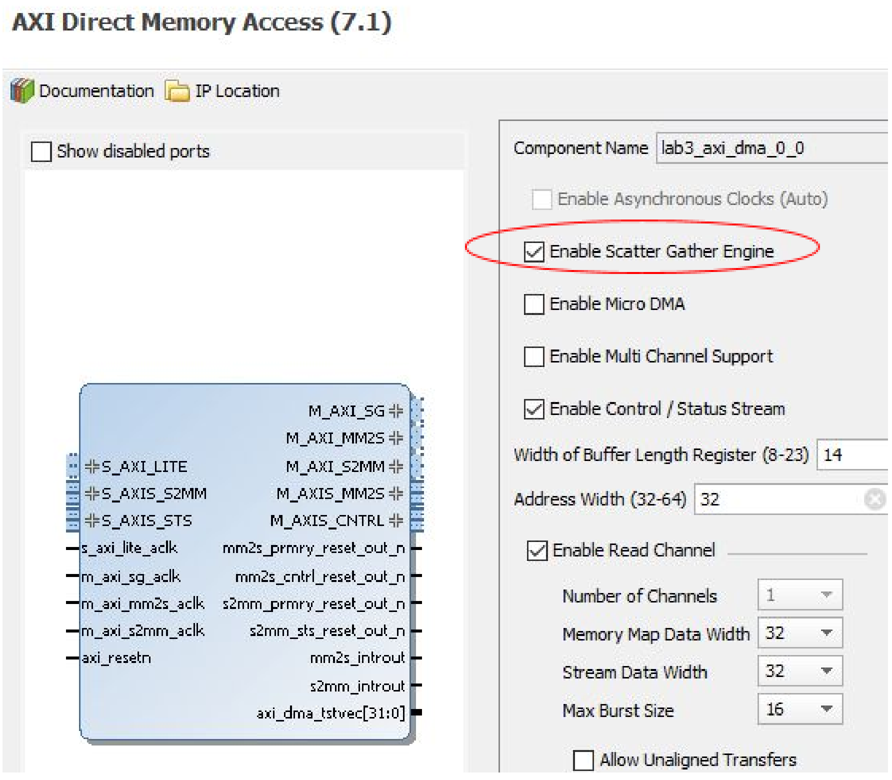
Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

PS-PL Configuration — Summary Report

Search:

Name	Select	Description
General		
AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
GP Slave AXI Interface		
HP Slave AXI Interface		
S AXI HP0 interface	☑	Enables AXI high performance slave interface 0
S AXI HP0 DATA WIDTH	32	Allows HP0 to be used in 32/64 bit data width mode
S AXI HP1 interface	☐	Enables AXI high performance slave interface 1
S AXI HP2 interface	☐	Enables AXI high performance slave interface 2
S AXI HP3 interface	☐	Enables AXI high performance slave interface 3

Double-click on the DMA and de-select the "scatter-gather" mode.


AXI Direct Memory Access (7.1)

Documentation · IP Location

☐ Show disabled ports

Component Name: lab3_axi_dma_0_0

☐ Enable Asynchronous Clocks (Auto)

☑ Enable Scatter Gather Engine

☐ Enable Micro DMA

☐ Enable Multi Channel Support

☑ Enable Control / Status Stream

Width of Buffer Length Register (8-23): 14

Address Width (32-64): 32

☑ Enable Read Channel

Number of Channels: 1

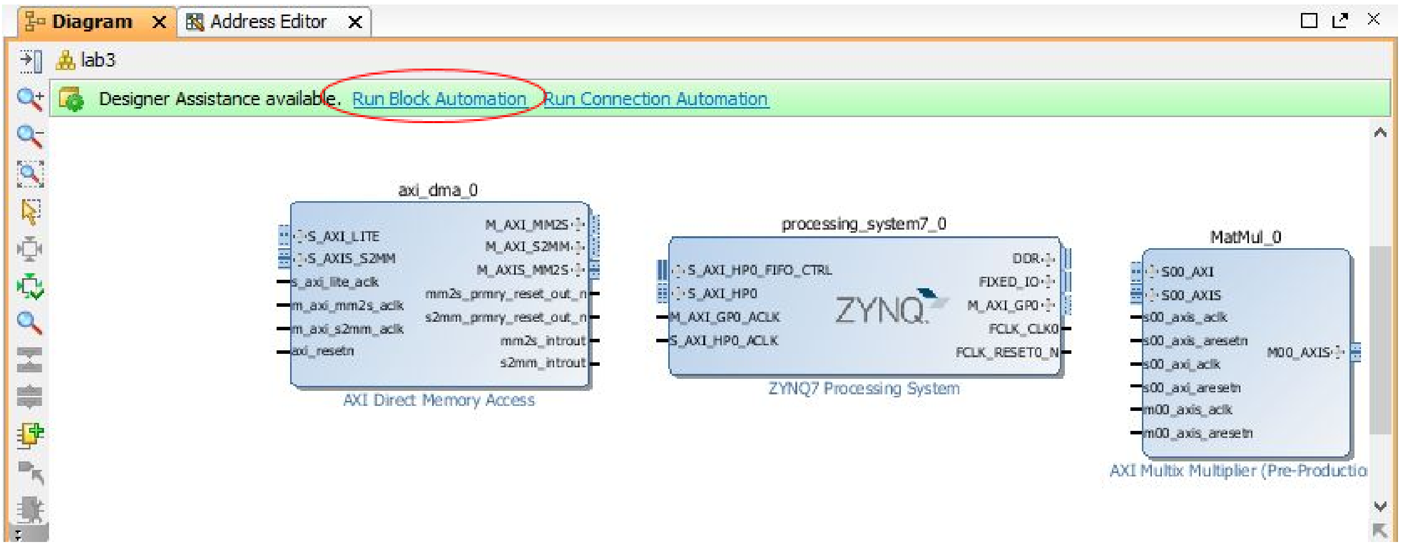
Memory Map Data Width: 32

Stream Data Width: 32

Max Burst Size: 16

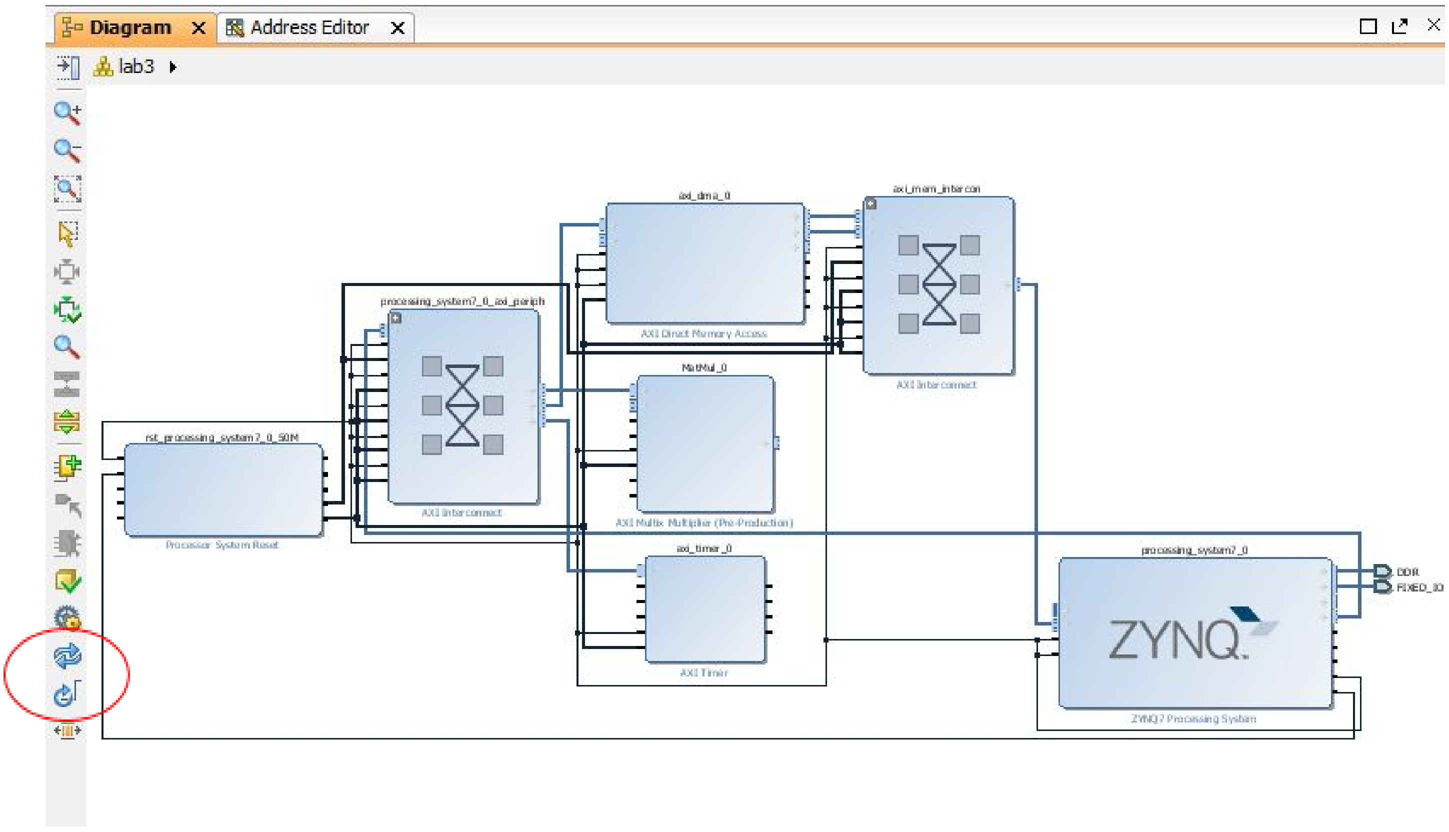
☐ Allow Unaligned Transfers

Click on "Run Block Automation". This will add interfaces to connect the PS to the external memory.



Then click on "Run Connection Automation" to add the necessary interconnections (from the window that appears choose "All Automation"). If needed, repeat this step until "Run Connection Automation" disappears.

Click on the "Regenerate Layout" and "Optimize Routing".



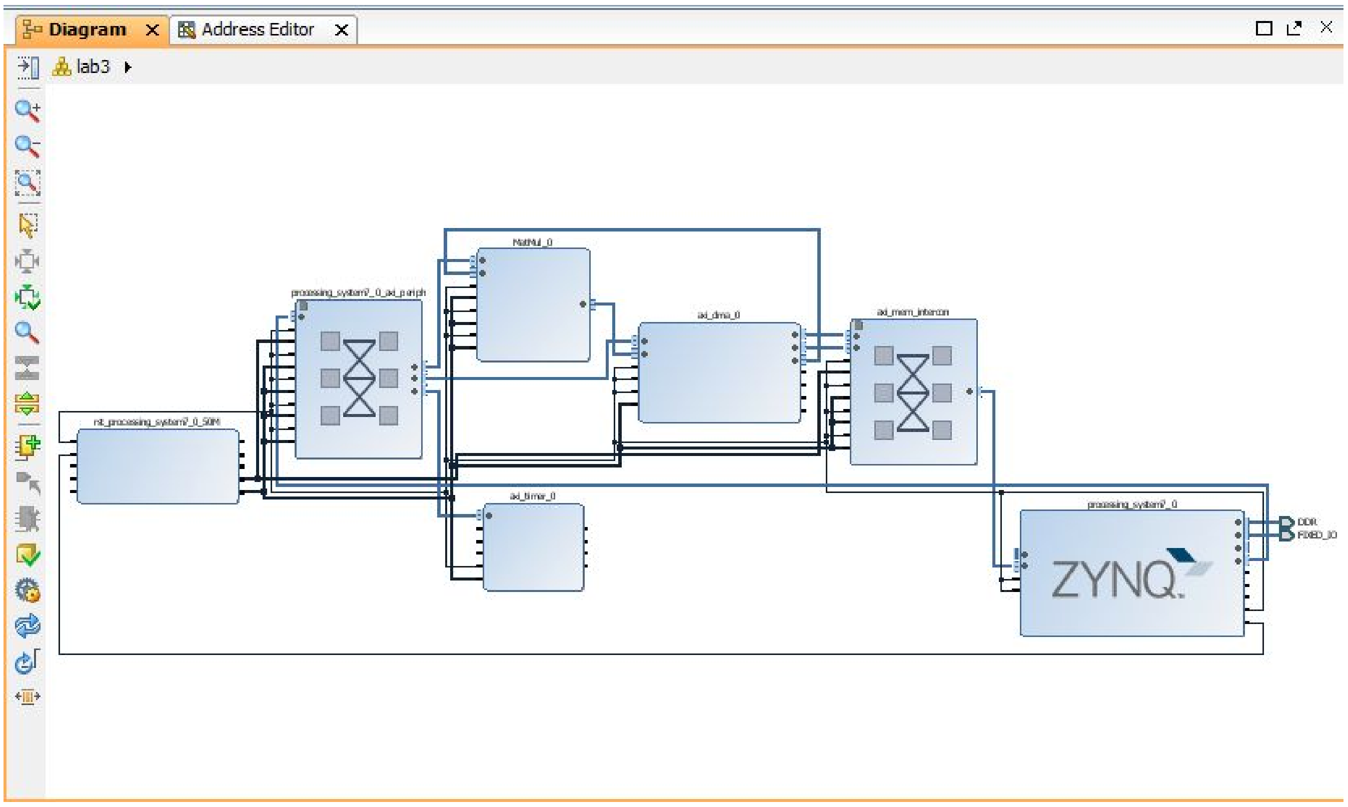
Zoom into AXI Matrix Multiplier and AXI DMA. You will need to complete the routing manually, as the automation is unable to finish the routing. Besides "reset" and "clock" in the multiplier IP, you need to fix the following connections:

Multiplier M00_AXIS → DMA S_AXIS
DMA M_AXIS → Multiplier S00_AXIS

Hover the mouse pointer over the interface. The pointer changes to a pen. Click-and-hold and drag the pen to the desired interface. Also a green check mark will appear next to the matching interfaces. Release the pen to complete the interconnection.

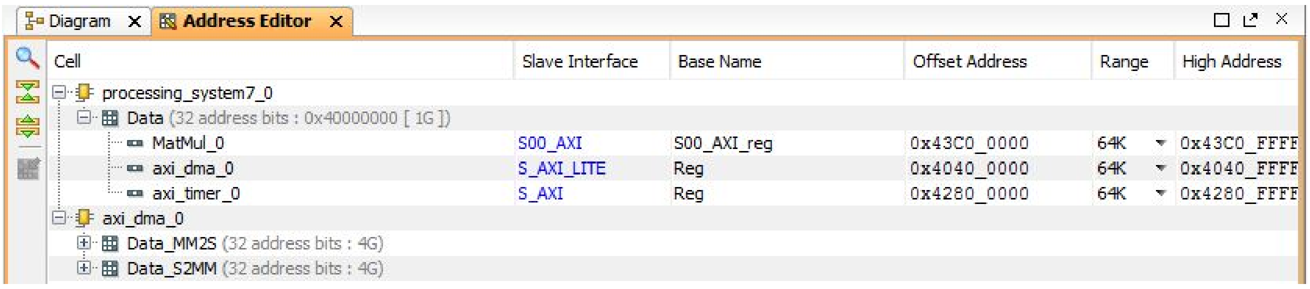
Similarly connect the unattached "clock" and "reset" pins of the Multiplier IP.

Select "Regenerate Layout" to see the whole design. Save the design and select "Validate Design" (F6).



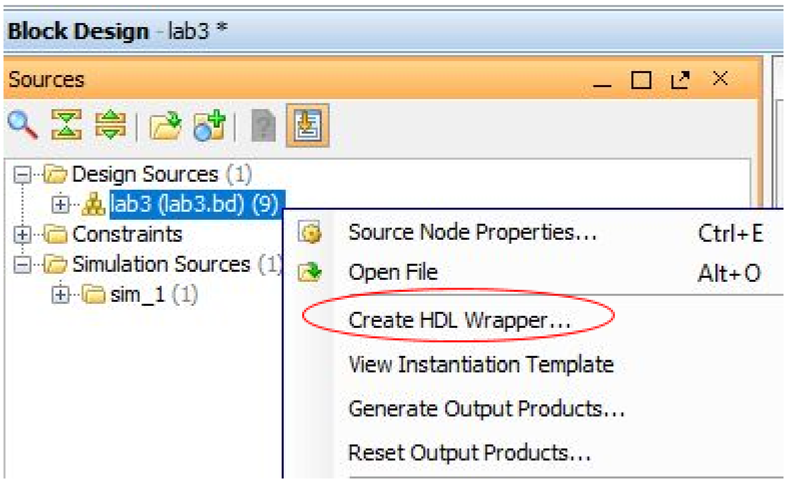
Ignore the warning about "axis_tstrb" signal.

Also have a look at the "Address Editor". These are the base addresses of the IPs that are used by the PS to communicate with them. These addresses together with device drivers are exported to the SDK to develop software that runs on the PS and communicates with the IPs.



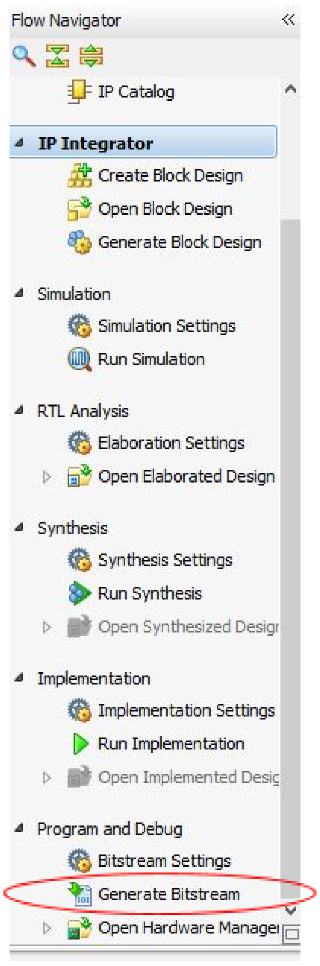
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
MatMul_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_timer_0	S_AXI	Reg	0x4280_0000	64K	0x4280_FFFF
axi_dma_0					
Data_MM2S (32 address bits : 4G)					
Data_S2MM (32 address bits : 4G)					

Now we need to create an HDL Wrapper that contains the whole design. Right-click on the block design (.bd) in the "Design Sources" and select "Create HDL Wrapper". From the window that appears select "Let Vivado Manage the Wrapper". Ignore the warning message concerning the s_axis_tstrb signal.

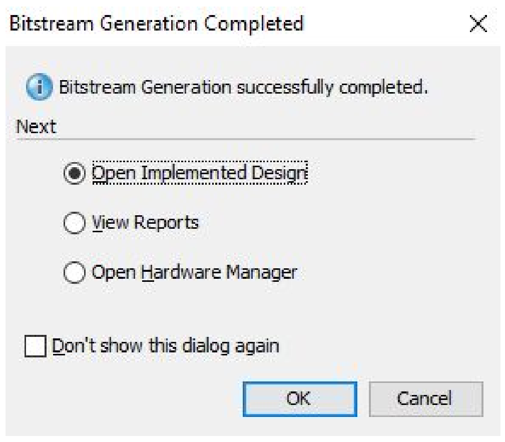


We are now ready to implement the design and create a bitstream. We will then export the bitstream to the SDK and write software to be run on the PS. Via the SDK we will then programme the FPGA using the bitstream and run the software on the PS.

In the "Flow Navigator" click on the "Generate Bitstream". Choose "Yes" to run the implementation first. Ignore the warning about the s_axis_tstrb signal.



It takes a while for the bitstream to be created. Once completed, click “Cancel” to close the window.


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

(●) Open Implemented Design

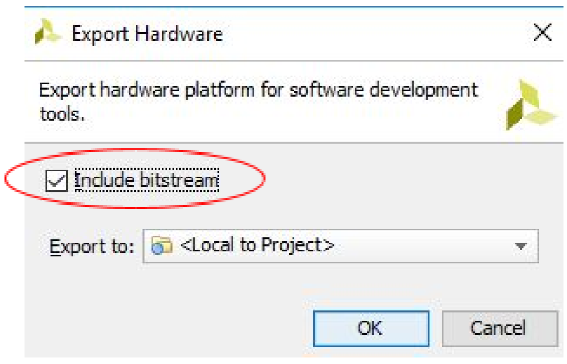
( ) View Reports

( ) Open Hardware Manager

☐ Don't show this dialog again

OK Cancel

Choose “File → Export → Export Hardware”. Select “Include bitstream” and do not change “Export to: <Local to Project>”.


Export Hardware

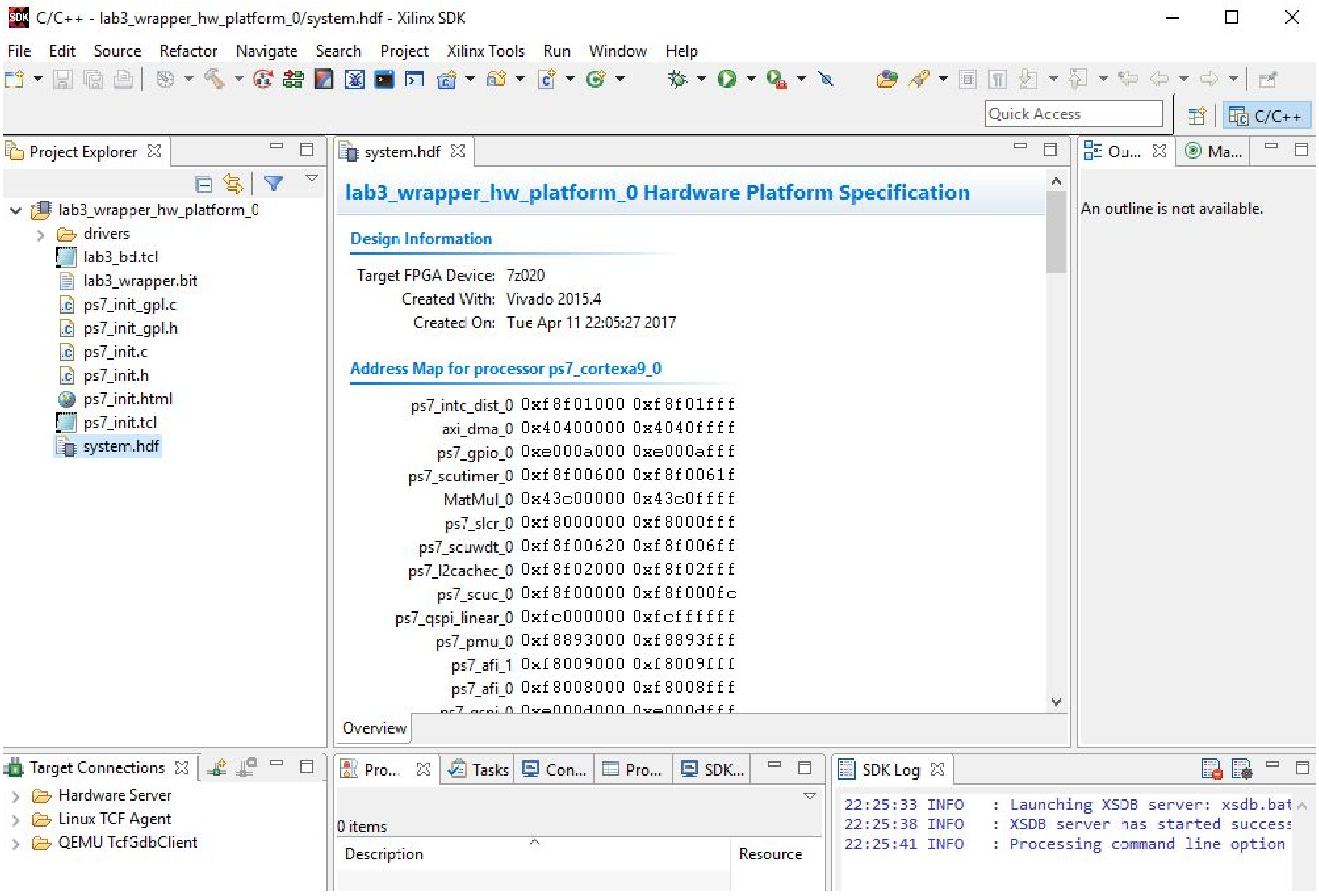
Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

OK Cancel

Choose “File → Launch SDK” and click OK. The Eclipse-based IDE of the Xilinx SDK opens.



lab3_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device:	7z020
Created With:	Vivado 2015.4
Created On:	Tue Apr 11 22:05:27 2017

Address Map for processor ps7_cortexa9_0

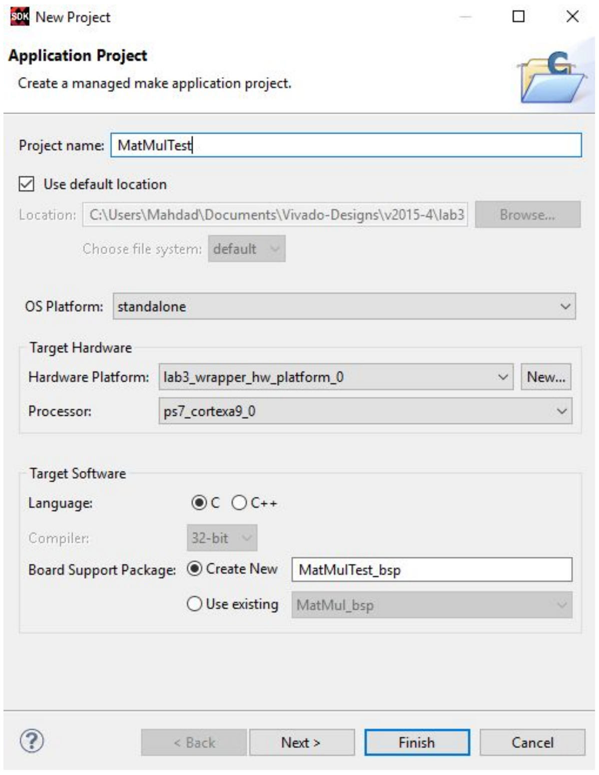
```
   ps7_intc_dist_0 0xf8f01000 0xf8f01fff
         axi_dma_0 0x40400000 0x4040ffff
         ps7_gpio_0 0xe000a000 0xe000afff
     ps7_scutimer_0 0xf8f00600 0xf8f0061f
          MatMul_0 0x43c00000 0x43c0ffff
         ps7_slcr_0 0xf8000000 0xf8000fff
      ps7_scuwdt_0 0xf8f00620 0xf8f006ff
     ps7_l2cachec_0 0xf8f02000 0xf8f02fff
         ps7_scuc_0 0xf8f00000 0xf8f000fc
    ps7_qspi_linear_0 0xfc000000 0xfcffffff
         ps7_pmu_0 0xf8893000 0xf8893fff
         ps7_afi_1 0xf8009000 0xf8009fff
         ps7_afi_0 0xf8008000 0xf8008fff
```

An outline is not available.

```
22:25:33 INFO  : Launching XSDB server: xsdb.bat
22:25:38 INFO  : XSDB server has started success
22:25:41 INFO  : Processing command line option
```

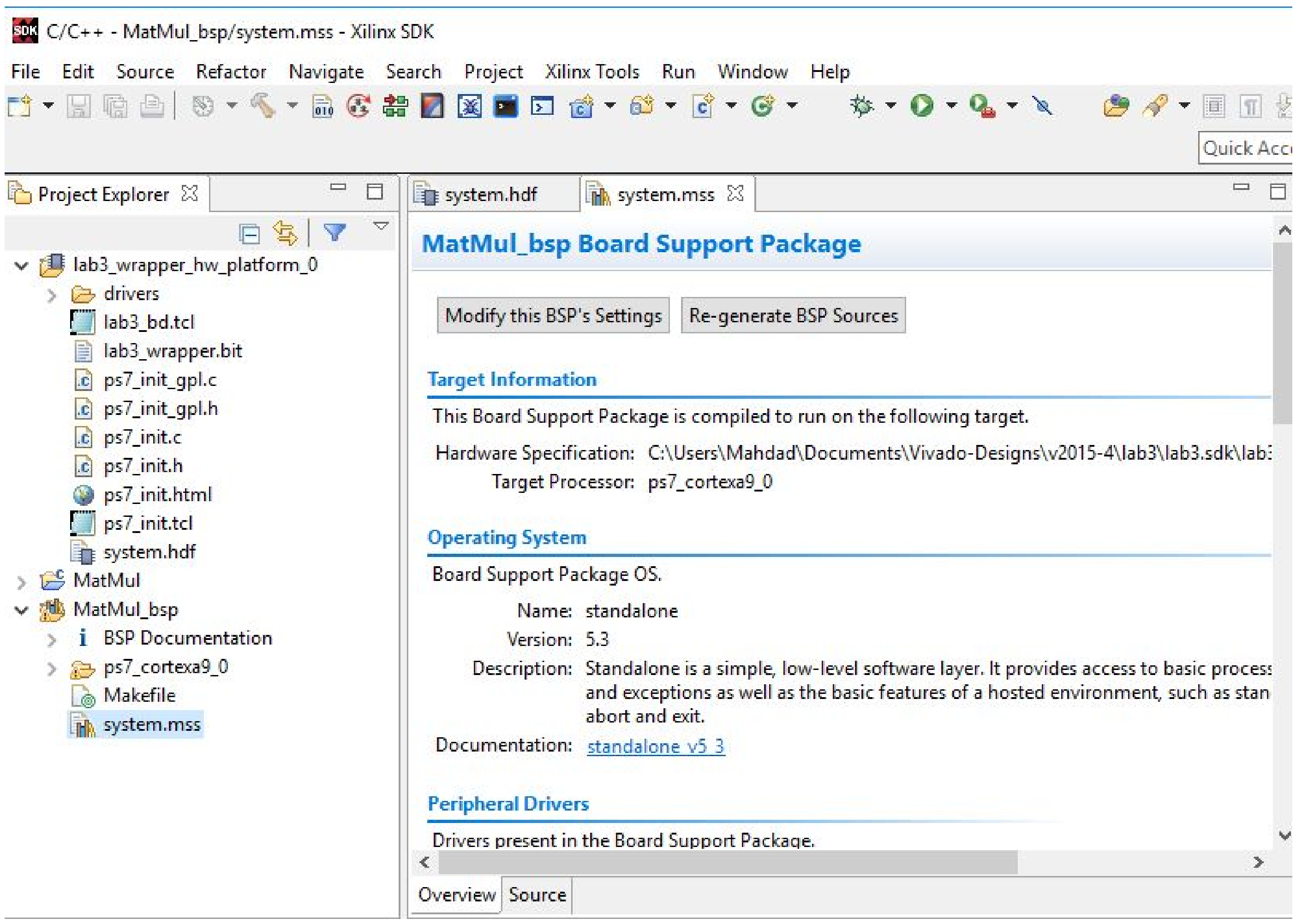
Your block design is exported to the SDK, with multiplier device driver. In the “system.hdf” (middle window) you can see the components/IPs and their memory-mapped addresses.

Select “File → New → Application Project”. Give the project a name and click “Next”. Select the default “Hello World” template. We will use this template to write our software. Click “Finish”.



Project name:	MatMulTest
Use default location	✓
Location:	C:\Users\Mahdad\Documents\Vivado-Designs\v2015-4\lab3
Choose file system:	default
OS Platform:	standalone
Target Hardware	
Hardware Platform:	lab3_wrapper_hw_platform_0
Processor:	ps7_cortexa9_0
Target Software	
Language:	[✓ C] C++
Compiler:	32-bit
Board Support Package:	[✓ Create New] MatMulTest_bsp
	Use existing MatMul_bsp

SDK creates your application project as well as the board support package (BSP). The application project contains binaries (.elf) for the ARM cores, debug files, libraries, and all the build and make scripts for your project. Also the BSP contains all the libraries and header files for all your design components and also the devices on the development board.



MatMul_bsp Board Support Package

Target Information

This Board Support Package is compiled to run on the following target.

Hardware Specification: C:\Users\Mahdad\Documents\Vivado-Designs\v2015-4\lab3\lab3.sdk\lab3
Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone
Version: 5.3
Description: Standalone is a simple, low-level software layer. It provides access to basic process and exceptions as well as the basic features of a hosted environment, such as stan abort and exit.
Documentation: standalone_v5_3

Peripheral Drivers

Drivers present in the Board Support Package.

From the “Project Explorer” open MatMul Src and double-click Helloworld.c (you can rename the file, if you wish).

Remove the “void print” and “print(Hello world)”. Add the following header files:

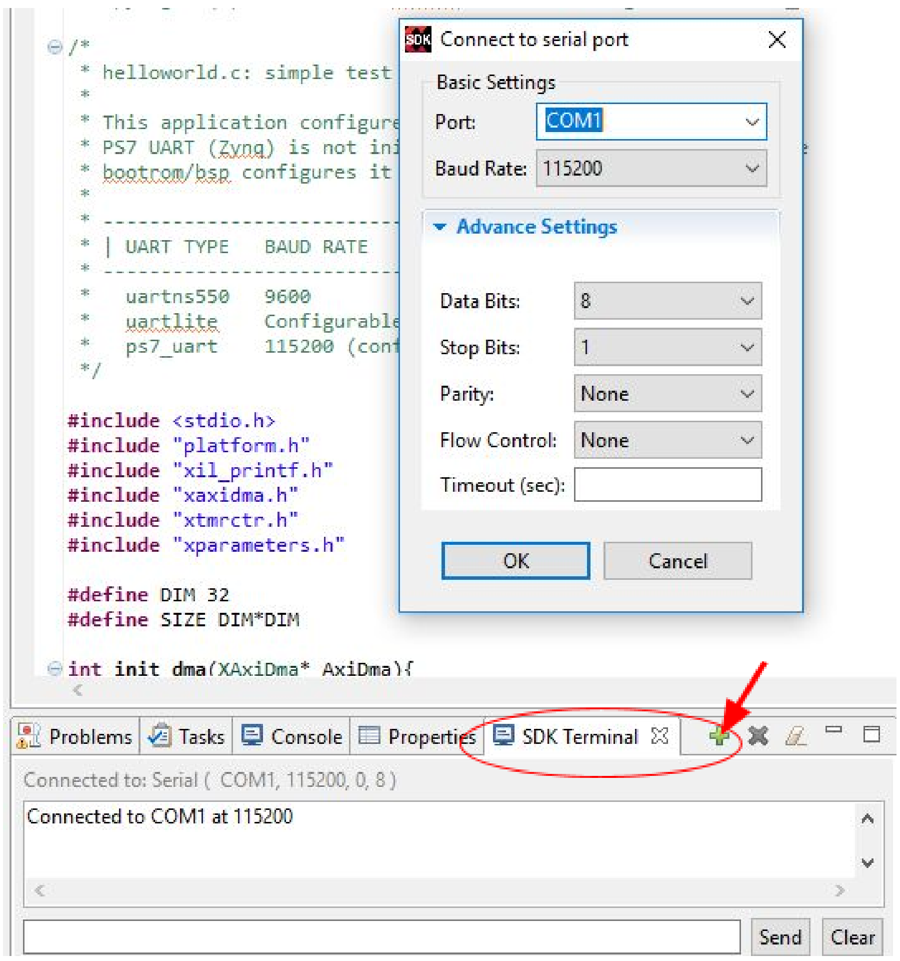
```
#include "xil_printf.h"      // function xil_printf ( )
#include "xaxidma.h"         // function calls to DMA
#include "xtmrctr.h"         // function calls to Timer
#include "xparameters.h"  // addresses of all the memory-mapped devices in the design
```

Have a look at the code provided in the lab skeleton files (`lab2/lab2.sdk`) and see how the software accesses the memory-mapped devices, including the accelerator.

Appendix II - Programming the FPGA

You can optionally test your design using an actual Zynq-7000 FPGA. Connect the JTAG and UART cables to the USB port of your computer and power on the FPGA development board.

Connect to the serial port by adding a connection from the “SDK Terminal”. Port is the COM Port number assigned to the UART USB connection (silicon labs cp210x).



```
/*
 * helloworld.c: simple test
 *
 * This application configure
 * PS7 UART (Zynq) is not ini
 * bootrom/bsp configures it
 *
 * -------------------------
 * | UART TYPE   BAUD RATE
 * -------------------------
 *   uartns550   9600
 *   uartlite    Configurable
 *   ps7_uart    115200 (conf
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xaxidma.h"
#include "xtmrctr.h"
#include "xparameters.h"

#define DIM 32
#define SIZE DIM*DIM

int init_dma(XAxiDma* AxiDma){
```

Connect to serial port

Basic Settings

Port: COM1

Baud Rate: 115200

Advance Settings

Data Bits: 8

Stop Bits: 1

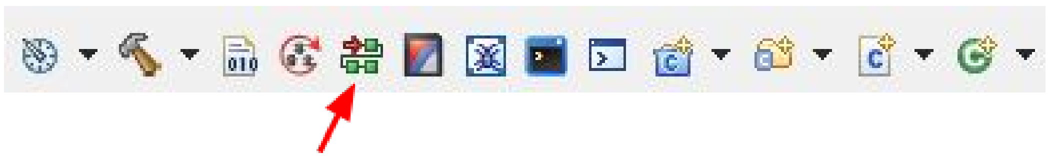
Parity: None

Flow Control: None

Timeout (sec):

```
Connected to: Serial ( COM1, 115200, 0, 8 )
Connected to COM1 at 115200
```

Choose “Xilinx Tools → Program FPGA”.



Click on the “Program” in the window that appears. When the programming is completed, right-click on the application folder in the “Project Explorer” window and select “Run As → 1 Launch on Hardware”. Your software begins to execute on the ARM PS and you will see the printouts in the “SDK Terminal”.