

Vivado HLS

(version 2014.1)

Example of Matrix Multiplication using Zynq

Gustavo Sutter

gustavo.sutter@uam.es

June 2014 - draft 0.3

Introduction

- Implement Matrix Multiplication example as a Zynq coprocessor connected to ACP (Accelerator Coherency Port) via DMA (Direct Memory Access)
- Our Target board is the Zedboard (Xilinx Zynq 7000 AP SoC XC7Z020 CLG484 package)
<http://www.zedboard.org/>



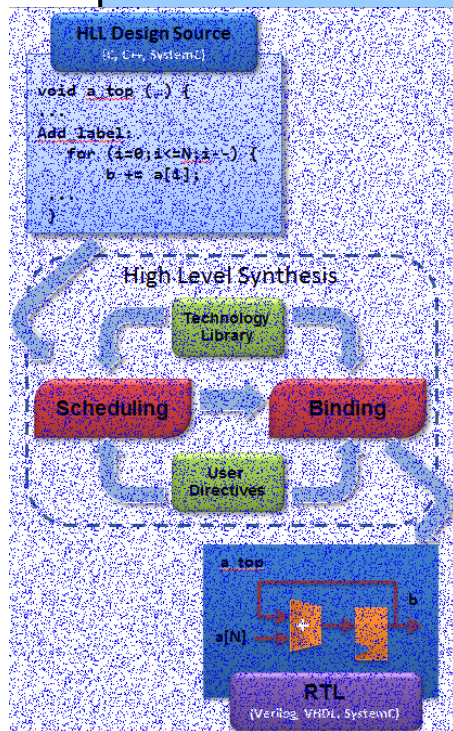
Introduction

- This work is based in a previous example from ElectraTraining and UAM courses, which uses Microblaze and the Matrix multiplications as a FSL coprocessor
- There are some enhancement inspired in the Xilinx XAPP1170



Disclaimer: This is a simple and incomplete implementation guide of Matrix Multiplication as a Zynq coprocessor. The concepts necessary to understand the solution are not explained here.

Agenda



- Setup and introduction
 - Zynq, ACP, DMA project
- Create HLS project and export rtl
 - Improve solution, Add AXI4 interfaces, export XACT
- Generate the Zynq project
 - Generate the design, Add the peripheral, connect, Export to SDK
- Generate the Zynq SW in SDK
 - Create a sw project, evaluate results
- Another more efficient example

Matrix Multiplication in Zynq

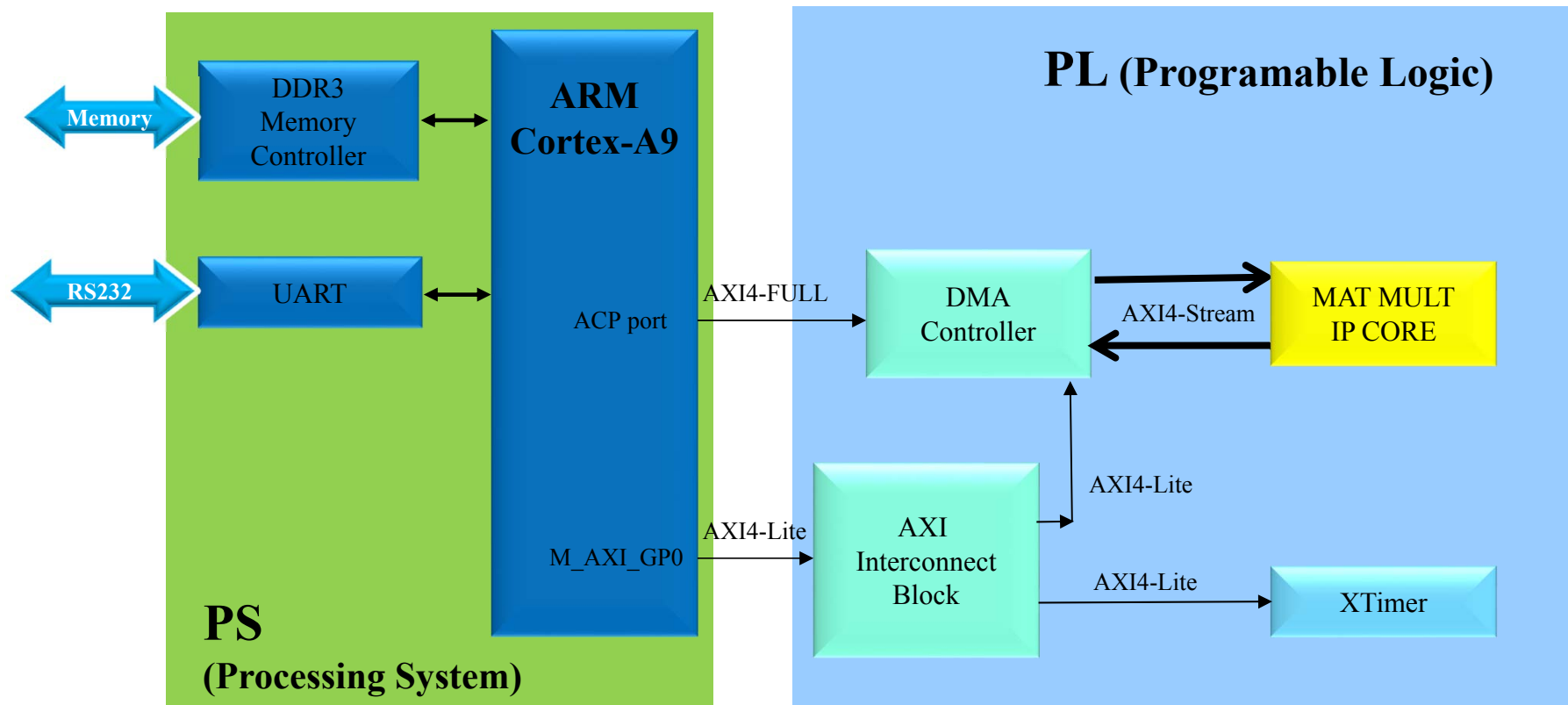
- Objective: Implement Matrix Multiplication example as a Zynq coprocessor
- We start with the previous example
- Create the project matrixmult_zynq_prj
 - Using the command console and running a script
 - vivado_hls -f scripts/matrimult_zynq.tcl
 - vivado_hls -p matrixmul_edk_prj
 - Or open the vivado-hls GUI
 - Create a new project
 - Add src/matrimult_zynq.cpp as source
 - Add src/matrixmult_zynq_test.cpp as testbench
 - Use same Zynq device as in previous lab with 100 MHz.

Matrix Multiplication: preliminaries

- We will multiply 32 by 32 floating point matrixes (instead of 4x4 8-bit matrixes as in previous) in order to tackle a bigger problem.
- We will start with a modification of our previous example.
- We simplify and assume square matrixes, replacing MAT_A_ROWS, MAT_A_COLS, MAT_B_ROWS, MAT_B_COLS by MAT_DIM

The Final Architecture

- We will use an ARM CPU and the coprocessor designed in Vivado-HLS will be implemented in PL connected through ACP port using DMA



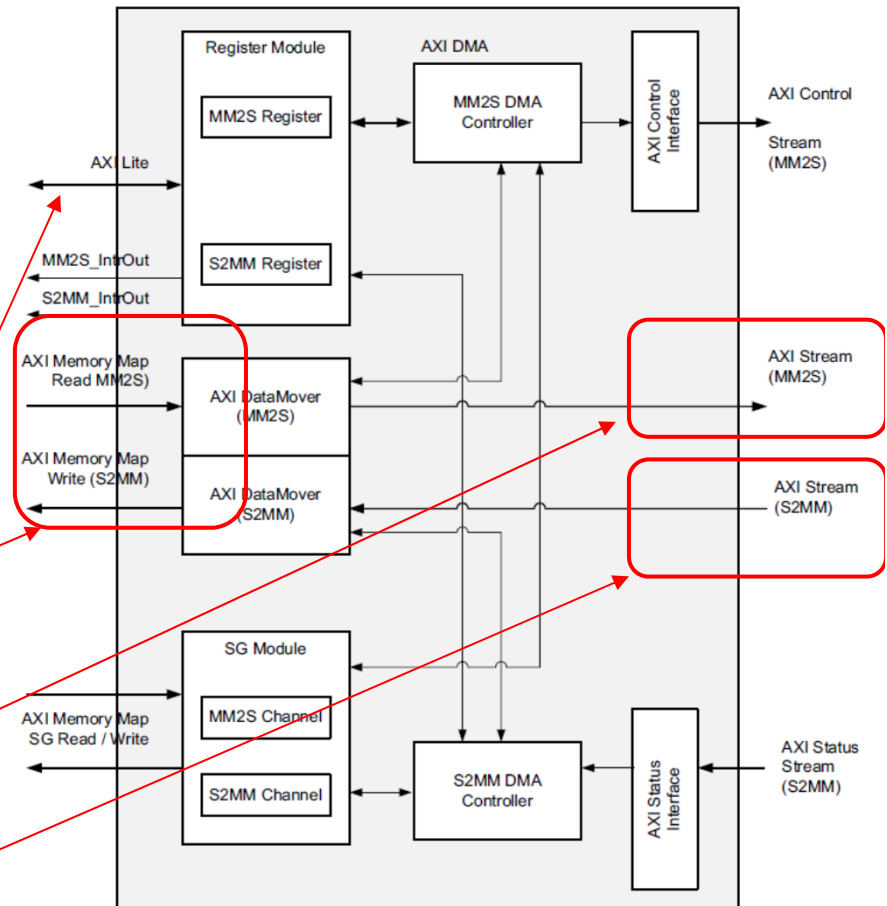
The Final System: ACP and DMA

- The ACP (Accelerator Coherency Port) is a 64-bit AXI slave interface on the Snoop Control Unit (SCU) that provides an asynchronous cache-coherent access point directly from the Zynq PL to the PS.
- The Matrix Multipliers designed in Vivado-HLS in PL will be connected through ACP port
- We will use DMA (Direct Memory Access) in order to transfer data from Memory to the accelerator and back to the memory.
 - We will use simple DMA (not Scatter/Gather) using pooling.

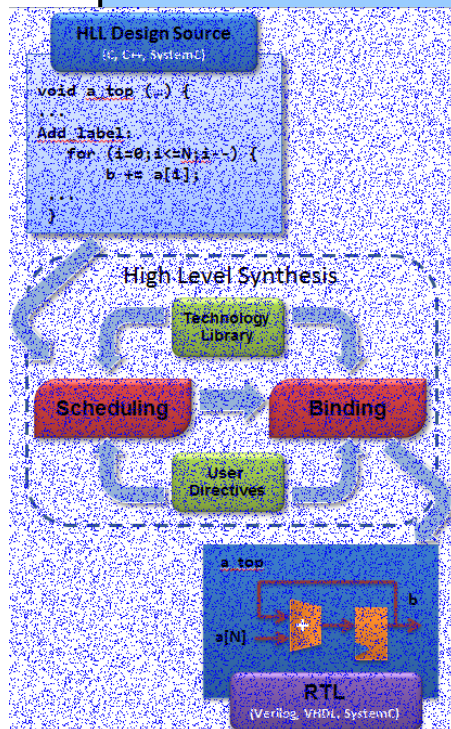
The Final System: DMA details

- The AXI DMA core provides direct memory access between memory and peripherals **through an AXI4-Stream interface**. The core design has the following AXI4 interfaces:

- AXI4-Lite Slave (for configuration)
- AXI4 Memory Map Read Master/Write Master
- AXI4 to AXI4-Stream (MM2S) Stream Master
- AXI4-Stream to AXI4 (S2MM) Stream Slave



Agenda



- Setup and introduction
 - Zynq, ACP, DMA project
- Create HLS project and export rtl
 - Improve solution, Add AXI4 interfaces, export XACT
- Generate the Zynq project
 - Generate the design, Add the peripheral, connect, Export to SDK
- Generate the Zynq SW in SDK
 - Create a sw project, evaluate results
- Another more efficient example

Matrix Multiplication: Simple Implementation

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.09	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
362561	362561	362562	362562	none

Detail

+ Instance

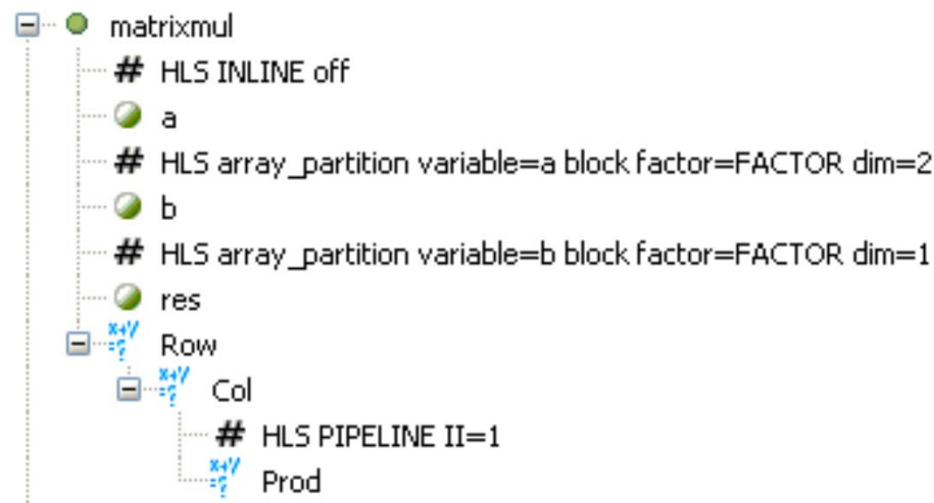
[-] Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- Row	362560	362560	11330	-	-	32	no
+ Col	11328	11328	354	-	-	32	no
++ Prod	352	352	11	-	-	32	no

- We start with a naive implementation

Improve the matrix multiplication

- Make a new solution (solution 2)
- Add pipeline and array_partition directives



Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.09	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1190	1190	1191	1191	none

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	1188	1188	166	1	1	1024	yes

Improve the matrix multiplication

- An important latency improvement (around x300)
- But uses much more resources (32 FP multiplier in parallel and also 32 FP adders!!!)

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U0	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U1	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U2	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U3	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U4	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U5	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fadd_32ns_32ns_32_5_full_dsp_U6	matrixmul_fadd_32ns_32ns_32_5_full_dsp	0	2	205	390
matrixmul_fmulo_32ns_32ns_32_4_max_dsp_U60	matrixmul_fmulo_32ns_32ns_32_4_max_dsp	0	3	143	321
matrixmul_fmulo_32ns_32ns_32_4_max_dsp_U61	matrixmul_fmulo_32ns_32ns_32_4_max_dsp	0	3	143	321
matrixmul_fmulo_32ns_32ns_32_4_max_dsp_U62	matrixmul_fmulo_32ns_32ns_32_4_max_dsp	0	3	143	321
matrixmul_fmulo_32ns_32ns_32_4_max_dsp_U63	matrixmul_fmulo_32ns_32ns_32_4_max_dsp	0	3	143	321
Total		64	0	160	22752

Memory

Performance Estimates

Timing (ns)

Clock		solution1	solution2
default	Target	10.00	10.00
	Estimated	8.09	8.09

Latency (clock cycles)

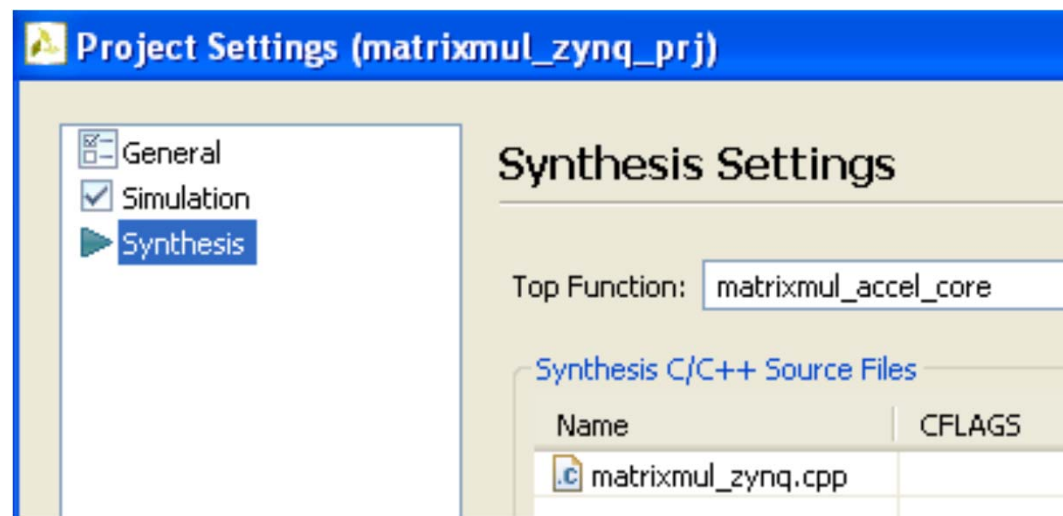
		solution1	solution2
Latency	min	362561	1190
	max	362561	1190
Interval	min	362562	1191
	max	362562	1191

Utilization Estimates

	solution1	solution2
BRAM_18K	0	0
DSP48E	5	160
FF	506	13413
LUT	836	23123

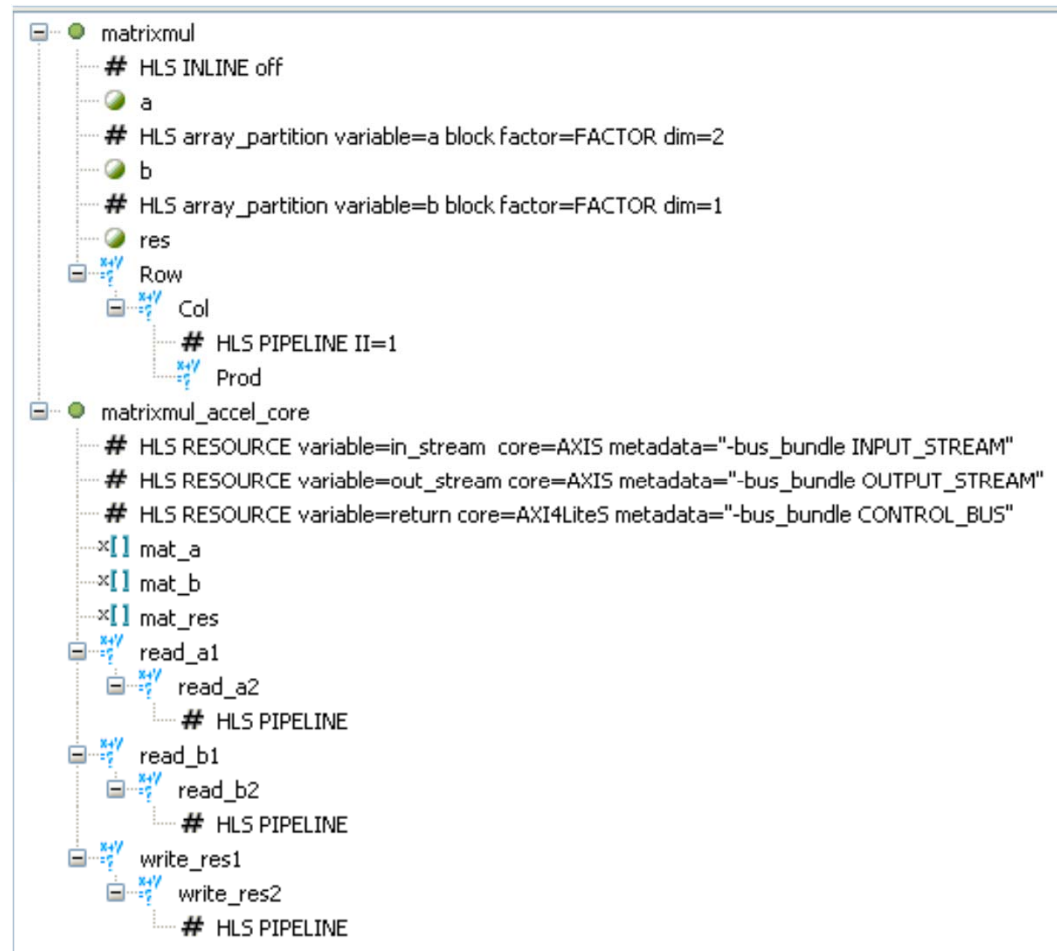
Connect the matrix multiplication with AXI-stream

- Since the DMA gives us a AXI-Stream interface, we will add a “wrapper” that connects to it.
- Make a new solution and change the top function to `matrix_accel_core`
 - Observe the acceleration core (remember AXI protocol)



Connect the matrix multiplication with AXI-stream

- matrix_accel_core reads streamed matrixes A and B and saves them in BRAMs
- Calls matrixmul function
- Transforms the matrix result in an output stream.



Connect the matrix multiplication with AXI-stream

- We multiply by 4 the total latency.
 - Observe the internal memory used.

Make sense? Verify it.

Name	BRAM_18K	DSP48E	FF	LUT
Total	34	160	14274	24787

Instance

Instance	Module	Latency		Interval		Type
		min	max	min	max	
grp_matrixmul_fu_928	matrixmul	1190	1190	1190	1190	none

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- read_a1_read_a2	1030	1030	8	1	1	1024	yes
- read_b1_read_b2	1030	1030	8	1	1	1024	yes
- write_res1_write_res2	1028	1028	6	1	1	1024	yes

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.09	1.25

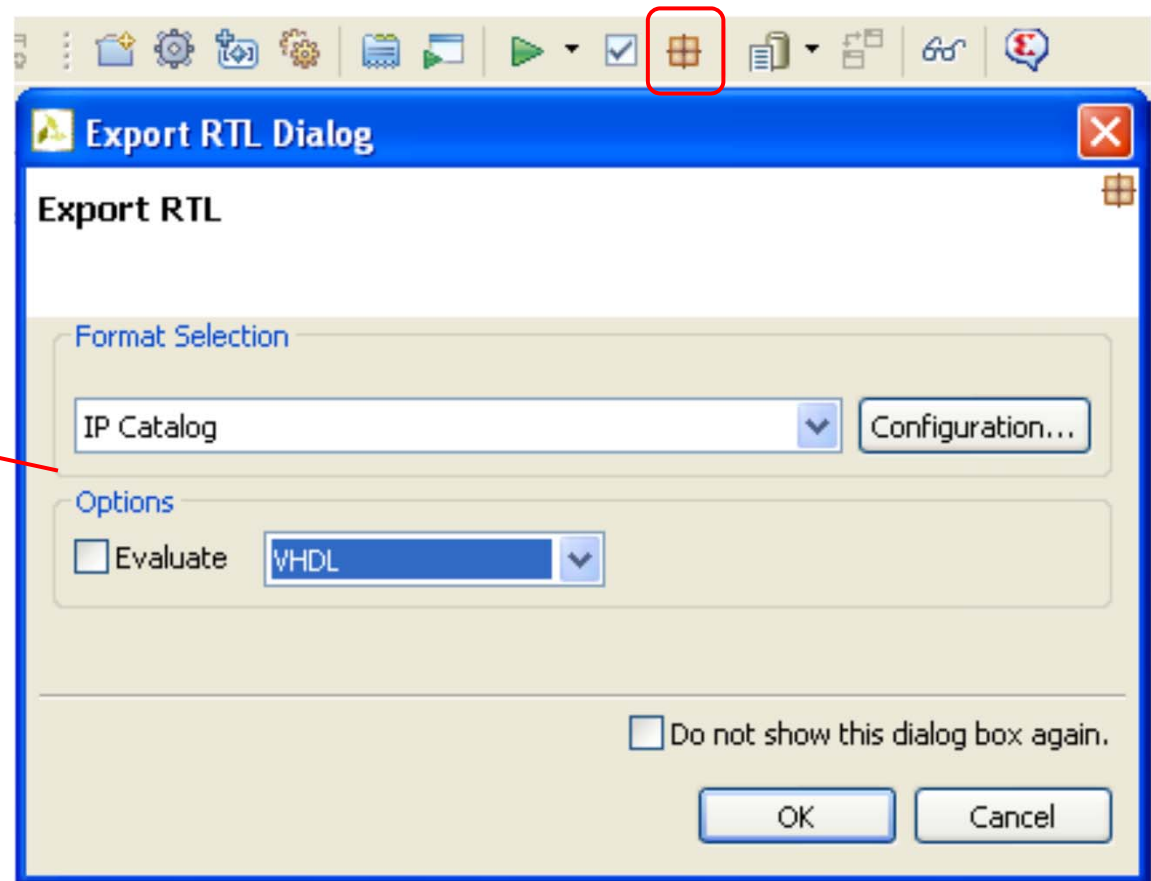
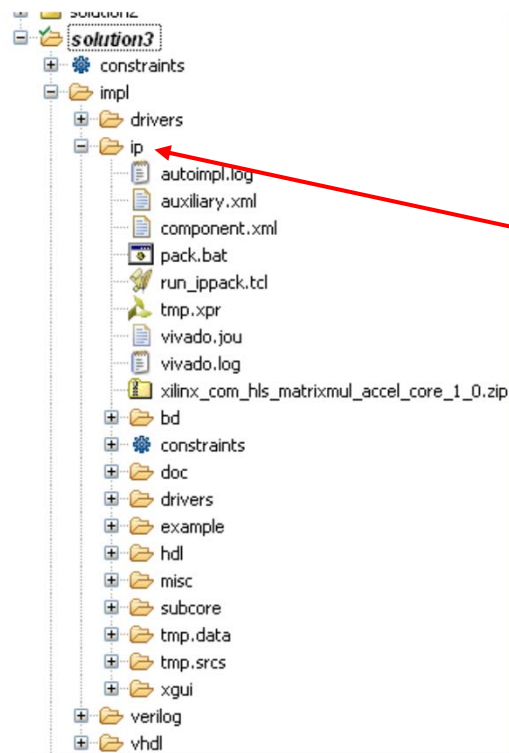
Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
4284	4284	4285	4285	none

Export matrix_accel_core

- Export the RTL to IP Catalog (IP-XACT)



Simulation and co-simulating

- The testbench is ready to simulate and cosimulate both codes (the matrixmult and the matrixmult_accel_core).
- Comment and uncomment the lines in main() at matrixmult_zynq_test.cpp

Cosimulation Report for 'matrixmul_accel_core'

Result

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA
SystemC	Pass	4284	4284	4284	4285	4285	4285

Export the report(.html) using the [Export Wizard](#)

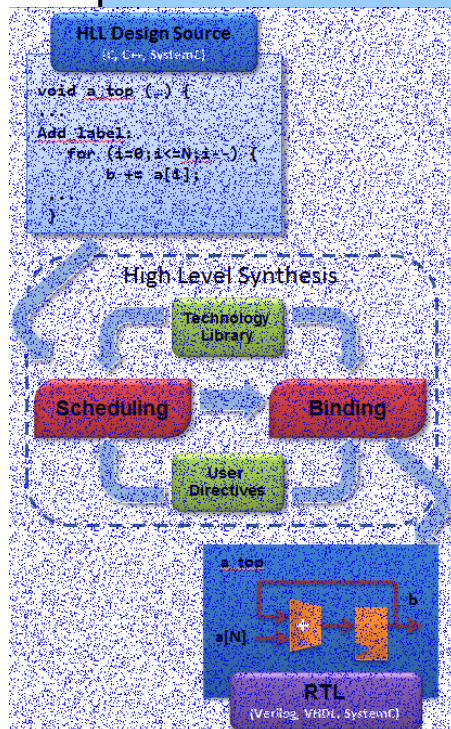
```
{0,25792,51584,77376,103168,128960,154  
}
```

Test passed. No errors

@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***

@I [LIC-101] Checked in feature [VIVADO_HLS]

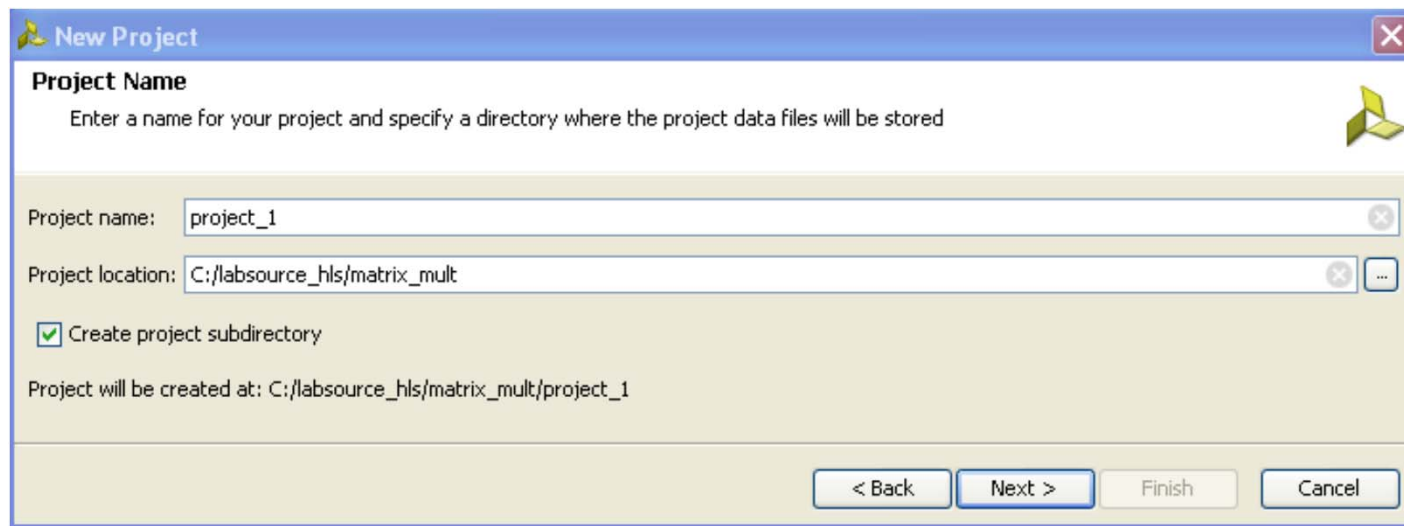
Agenda



- Setup and introduction
 - Zynq, ACP, DMA project
- Create HLS project and export rtl
 - Improve solution, Add AXI4 interfaces, export XACT
- Generate the Zynq project
 - Generate the design, Add the peripheral, connect, Export to SDK
- Generate the Zynq SW in SDK
 - Create a sw project, evaluate results
- Another more efficient example

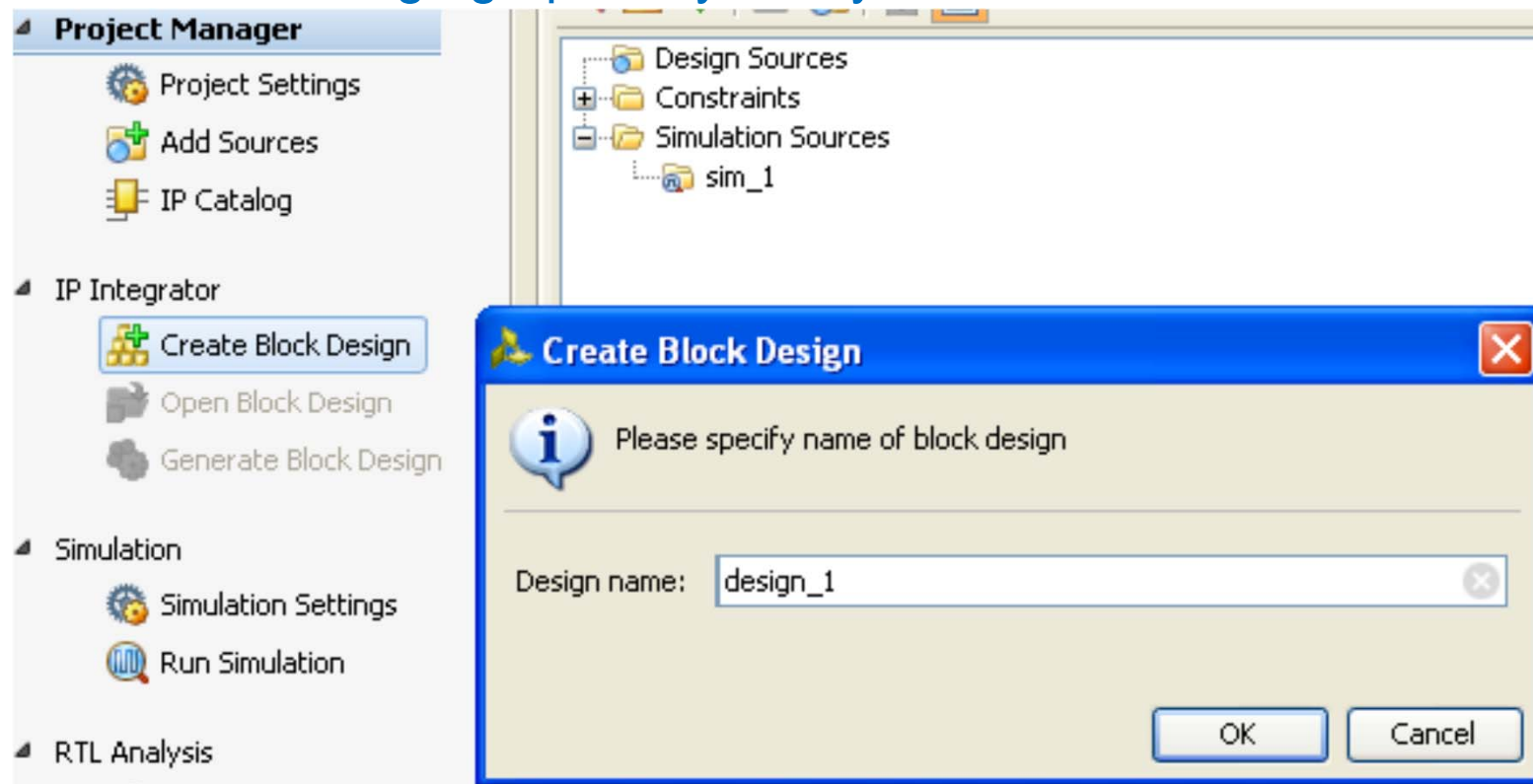
Create Zynq system with the Matrix Multiplier coprocessor

- We will use Vivado (not Vivado-HLS)
- Create a new Vivado Project (RTL Project)
 - Do not add sources, nor existing IP, nor constraints
 - Select Zedboard Zynq board



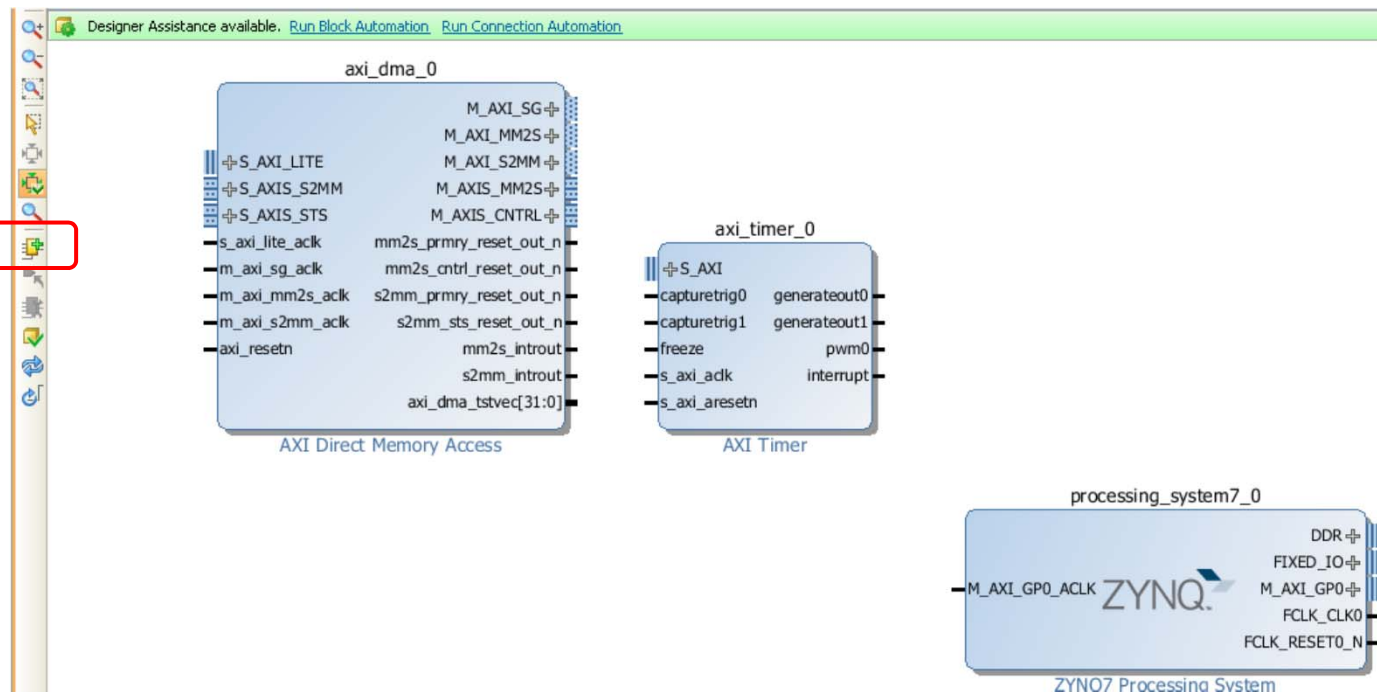
Creating a Vivado Project

- Create a Block Design.
 - We will design graphically the system



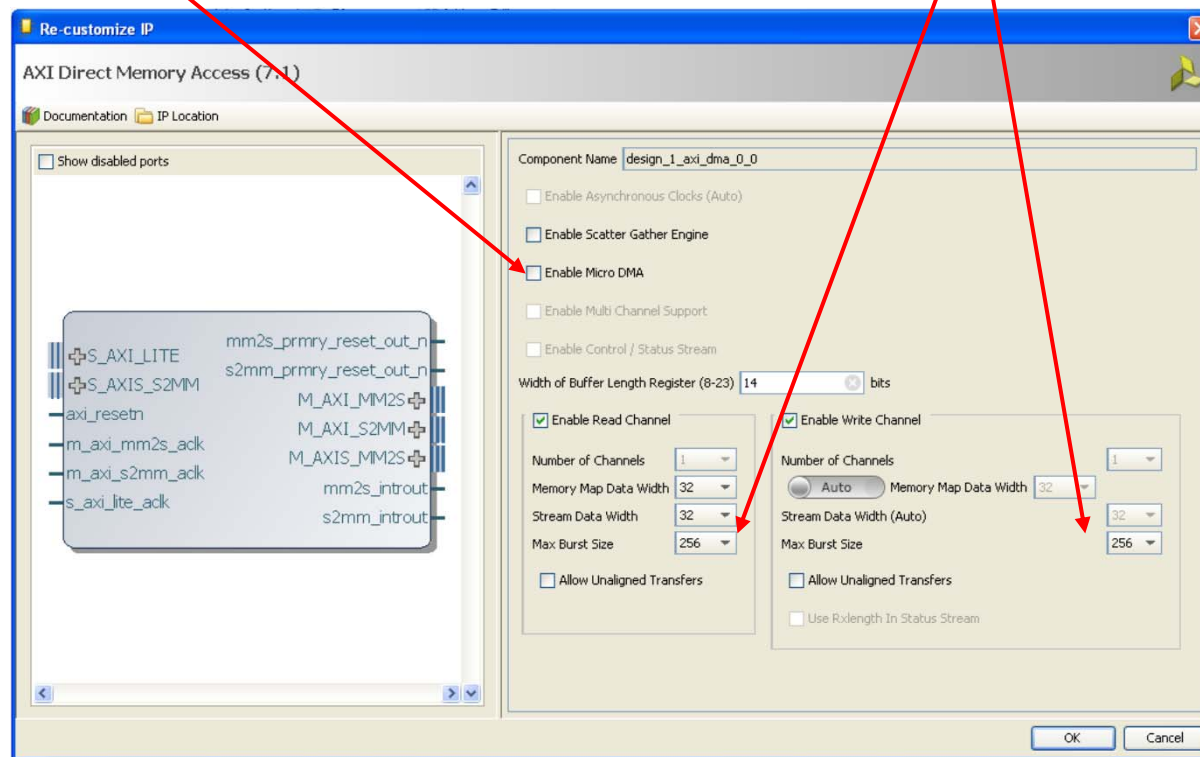
Create a Vivado Project

- Add IPs
 - Zynq processing system, AXI timer, and AXI Direct Memory Access (DMA)



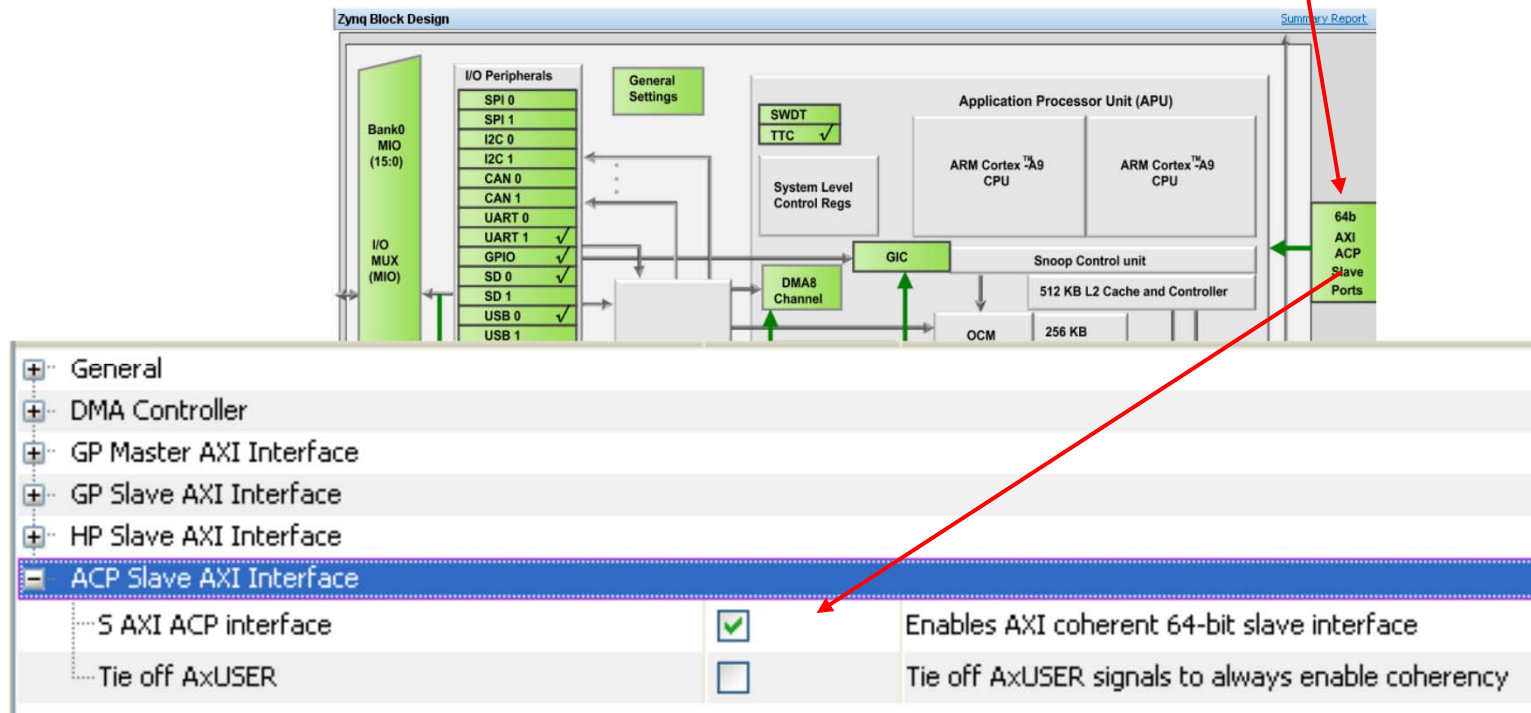
Create a Vivado Project

- Configure DMA (right click, customize block)
 - Disable Scatter Gather Engine, Increase the burst size (256).



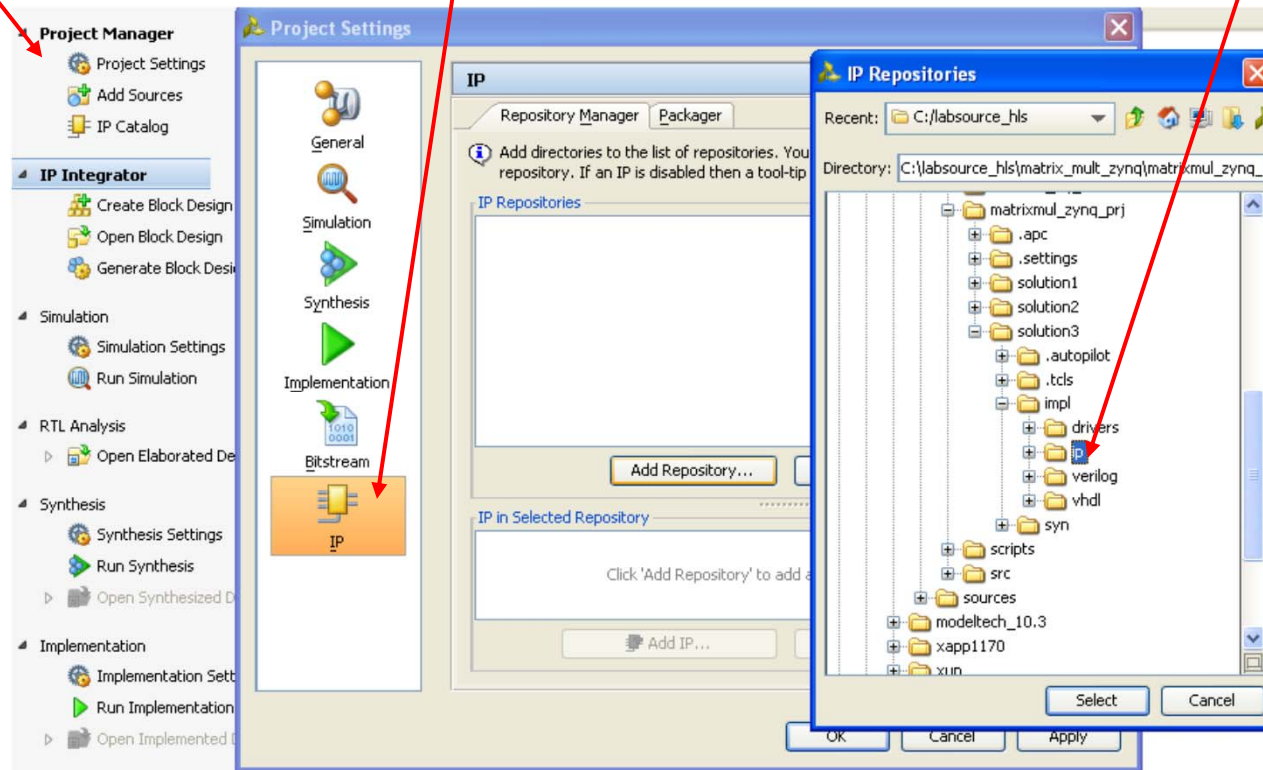
Create a Vivado Project

- Configure Zynq system
 - Enable the ACP interface. Double click AXI ACP port



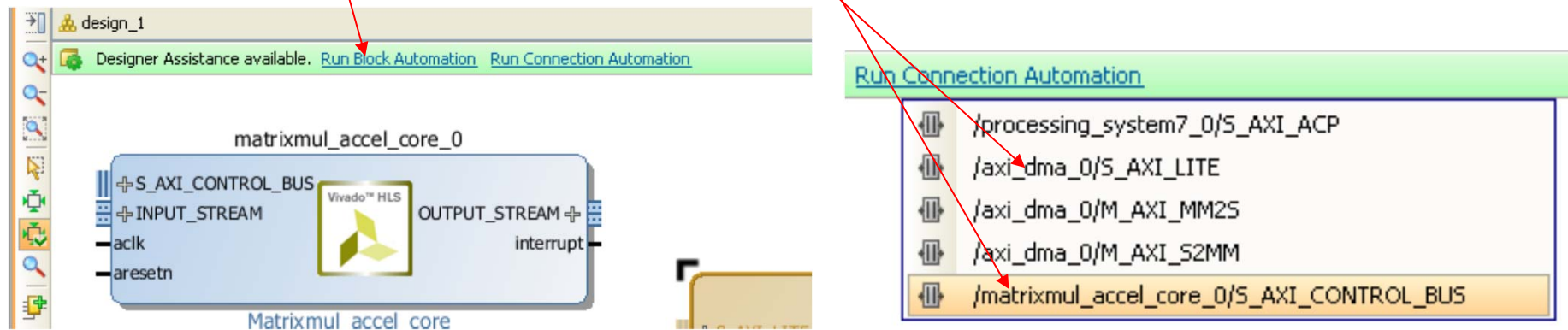
Add Matrix Mult to IP Catalog

- Project settings -> IP
 - Select the IP folder from your Vivado-hls implementation



Start connecting the block design

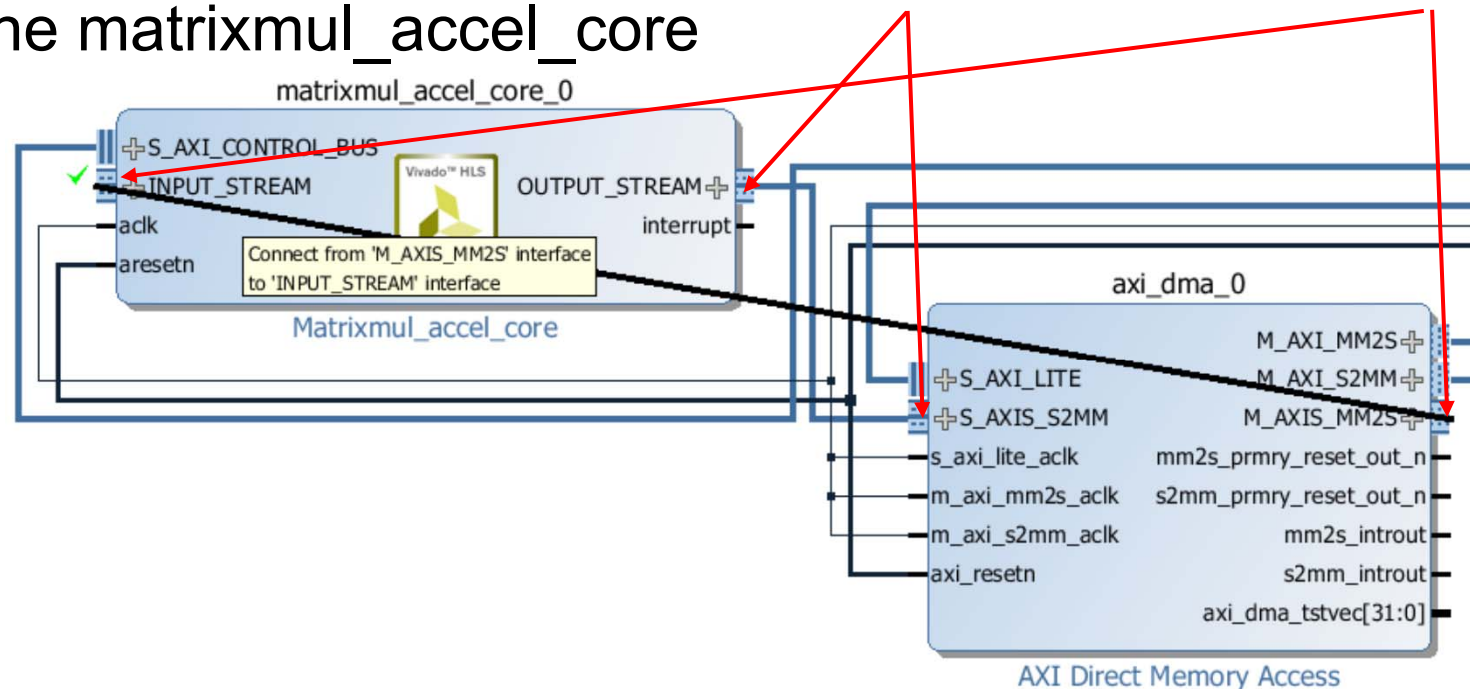
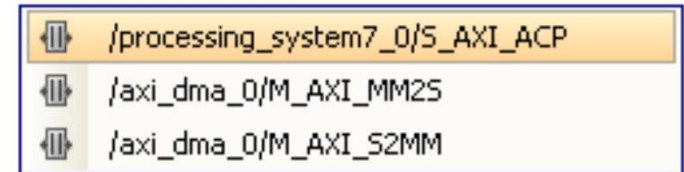
- Add the matrixmul_accel_core
- Start with autoconnections
 - Run block automation (for Zynq device)
 - Run connection automation
 - for AXI Timer
 - for matrix mul_accel_core
 - for axi_dma_0/S_AXI_LITE



Finishing the block design

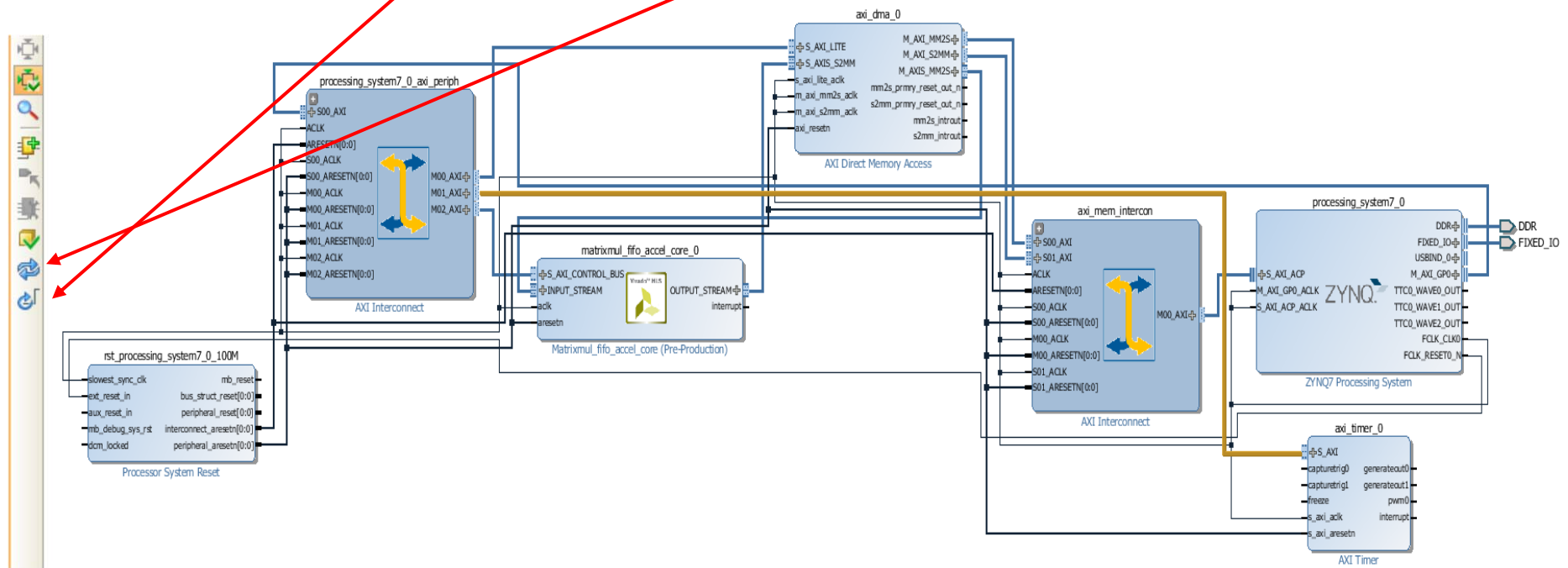
- Connect ACP to M_AXI_MM2S and M_AXI_S2MM
- Connect the stream interface of axi_dma_0 to the matrixmul_accel_core

Run Connection Automation



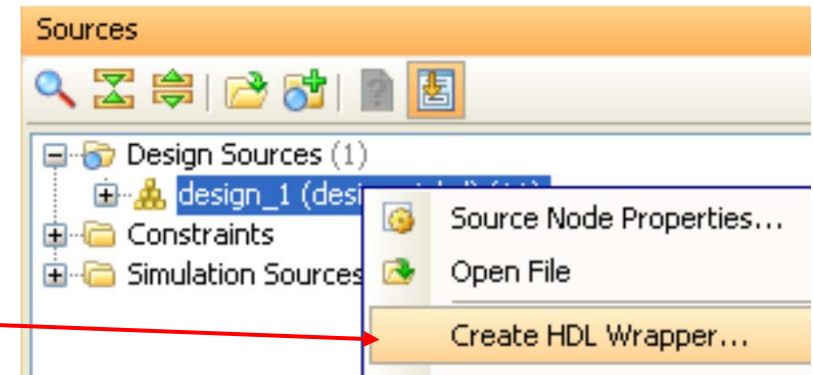
Finish the block design

- Regenerate the layout and validate the design (F6)



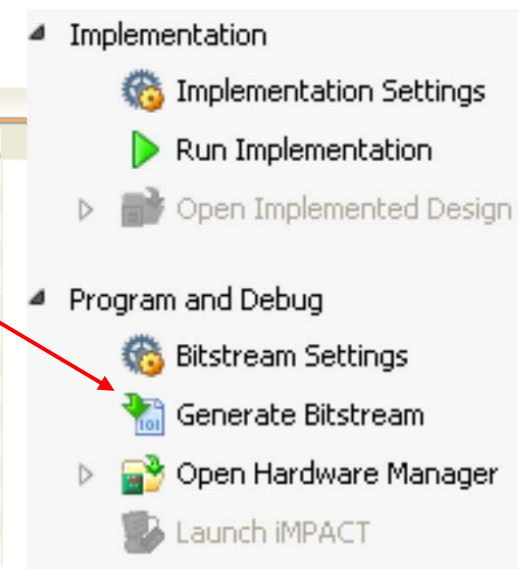
Check Address, create HDL and generate bitstream

- Check the generated addresses
 - It should be automatic
- Create the HDL Wrapper of black diagram
- Generate the bit-stream



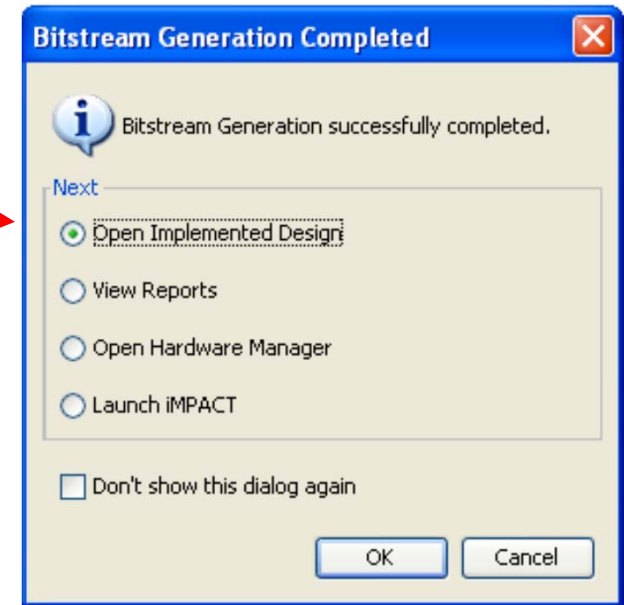
The 'Address Editor' window displays a table of memory addresses for various components. The table has columns: Cell, Interface Pin, Base Name, Offset Address, Range, and High Address.

Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_timer_0	S_AXI	Reg	0x42800000	64K	0x4280FFFF
matrixmul_accel_core_0	S_AXI_CONTROL...	Reg	0x43C00000	64K	0x43C0FFFF
axi_dma_0	S_AXI_LITE	Reg	0x40400000	64K	0x4040FFFF
axi_dma_0					
Data_SG (32 address bits : 4G)					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE0000000	4M	0xE03FFFFF
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x00000000	512M	0x1FFFFFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC000000	16M	0xFCFFFFFF
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x00000000	512M	0x1FFFFFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC000000	16M	0xFCFFFFFF
Unmapped Slaves (1)					

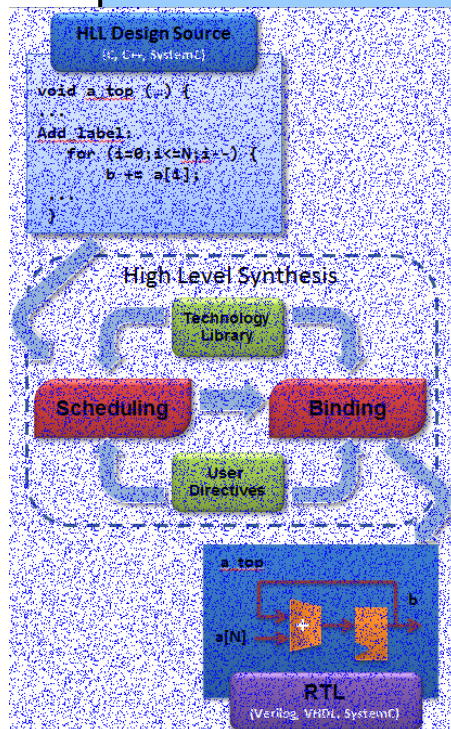


Export design to Xilinx SDK

- After bitstream generation
 - Open Implemented design
- File->Export->Export HW for SDK
 - Select all check boxes



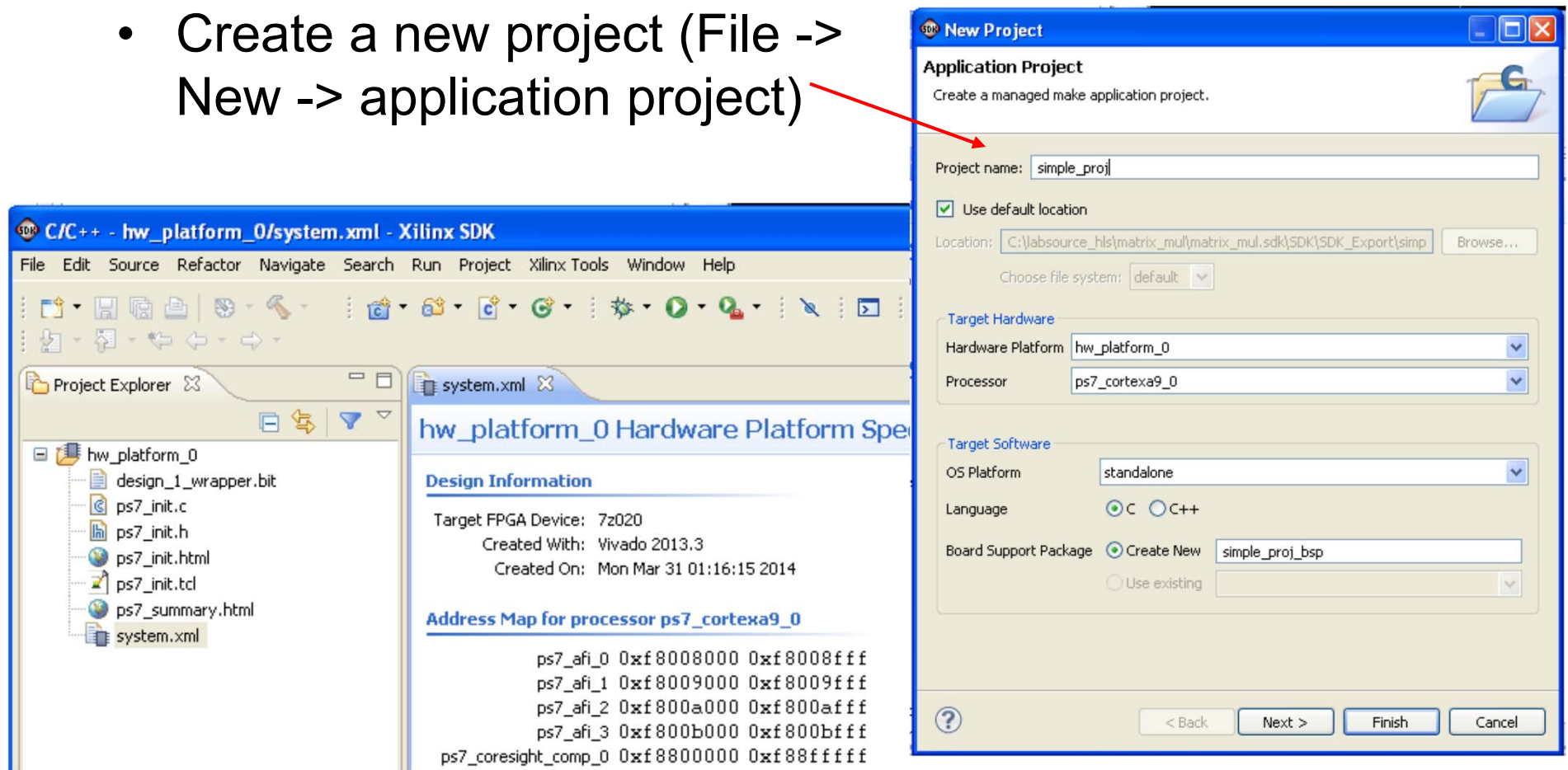
Agenda



- Setup and introduction
 - Zynq, ACP, DMA project
- Create HLS project and export rtl
 - Improve solution, Add AXI4 interfaces, export XACT
- Generate the Zynq project
 - Generate the design, Add the peripheral, connect, Export to SDK
- Generate the Zynq SW in SDK
 - Create a sw project, evaluate results
- Another more efficient example

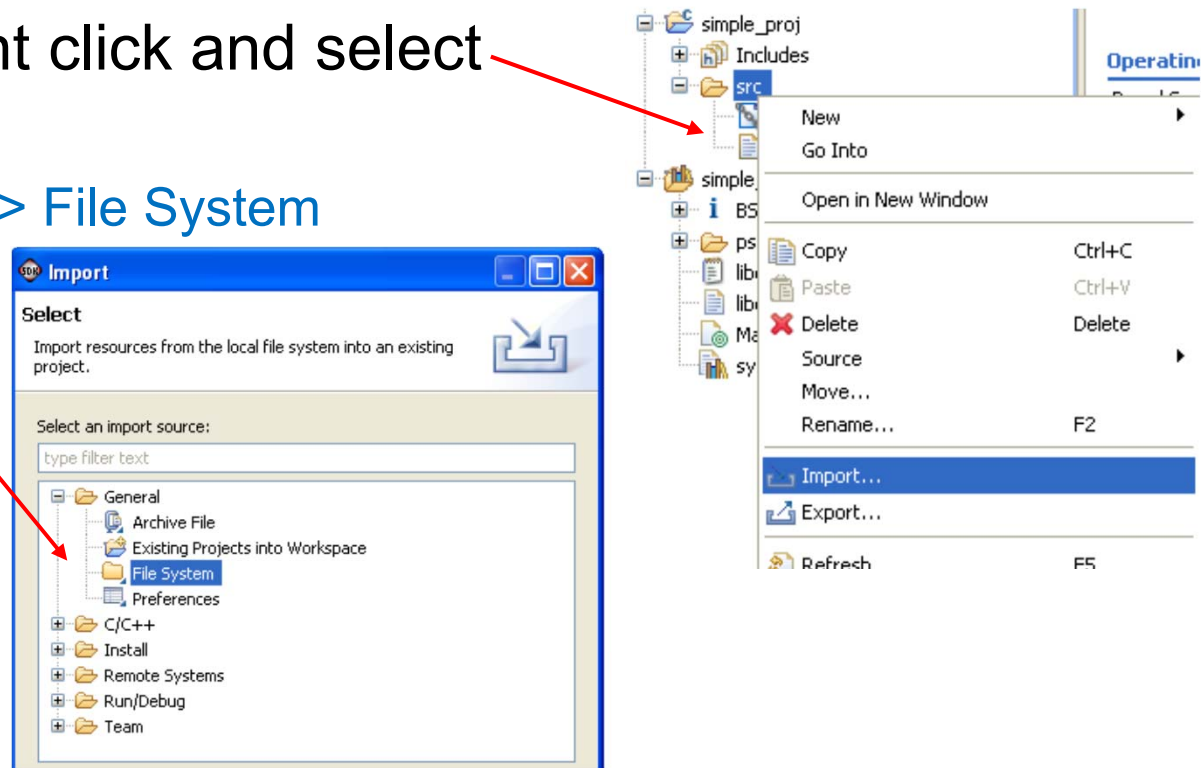
Xilinx SDK (Software Development Kit)

- Create a new project (File -> New -> application project)



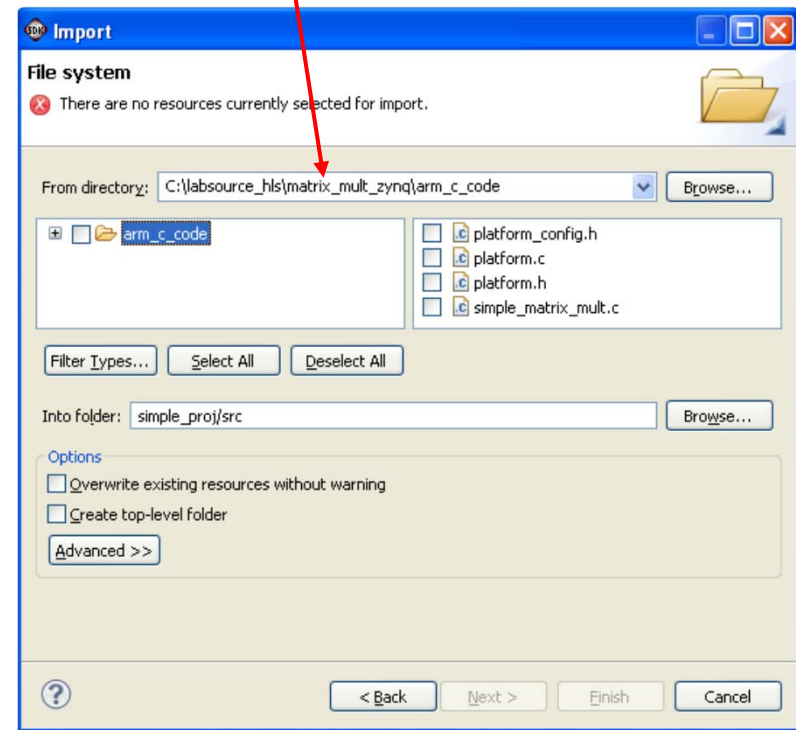
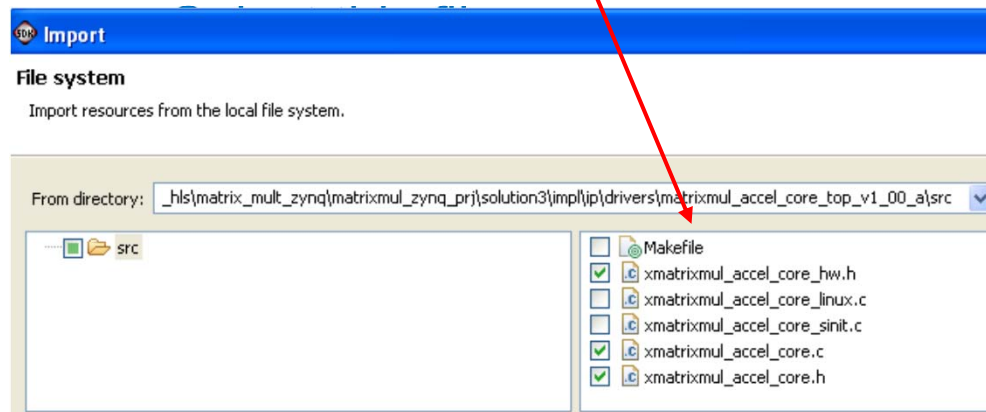
Xilinx SDK (Software Development Kit)

- Select an “Empty Application” and finish
 - The libraries will be created
- In src folder right click and select import.
 - Then General -> File System



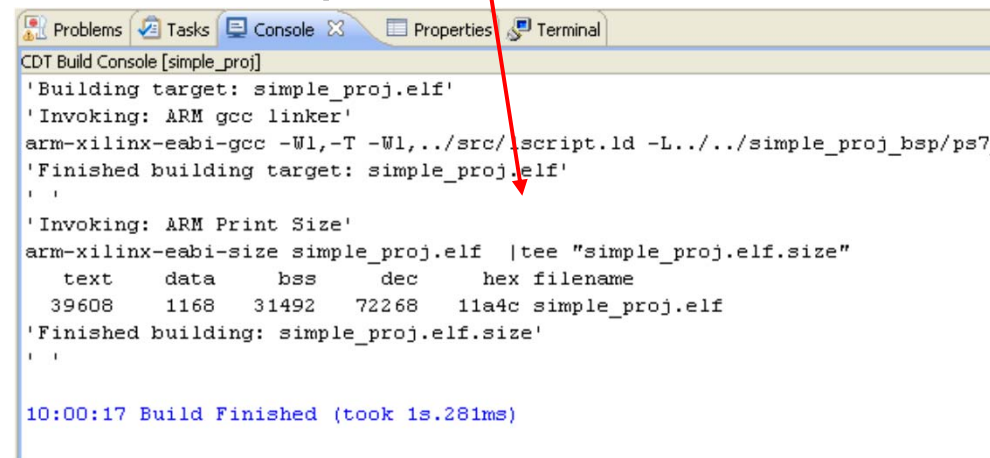
Xilinx SDK (Software Development Kit)

- Select the files from arm_c_code provided
 - Select all files
- Add files from drivers folder in accelerator implementation



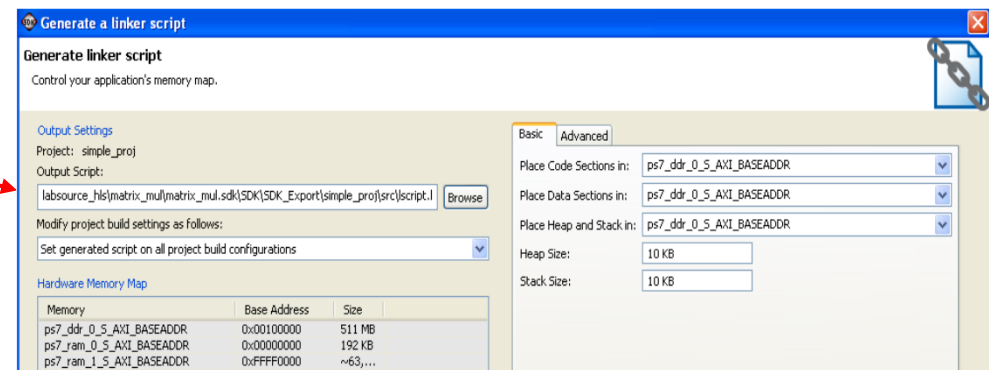
Xilinx SDK (Software Development Kit)

- As a result the code should be compiled.



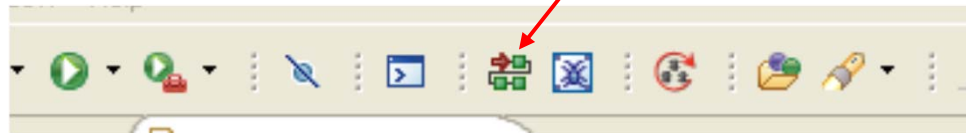
```
CDT Build Console [simple_proj]
'Building target: simple_proj.elf'
'Invoking: ARM gcc linker'
arm-xilinx-eabi-gcc -Wl,-T -Wl,../src/script.ld -L../simple_proj_bsp/ps7_
'Finished building target: simple_proj.elf'
'
'Invoking: ARM Print Size'
arm-xilinx-eabi-size simple_proj.elf |tee "simple_proj.elf.size"
text data bss dec hex filename
39608 1168 31492 72268 11a4c simple_proj.elf
'Finished building: simple_proj.elf.size'
'
10:00:17 Build Finished (took 1s.281ms)
```

- Generate linker script and increase heap and stack size
 - Its important!, otherwise will fail

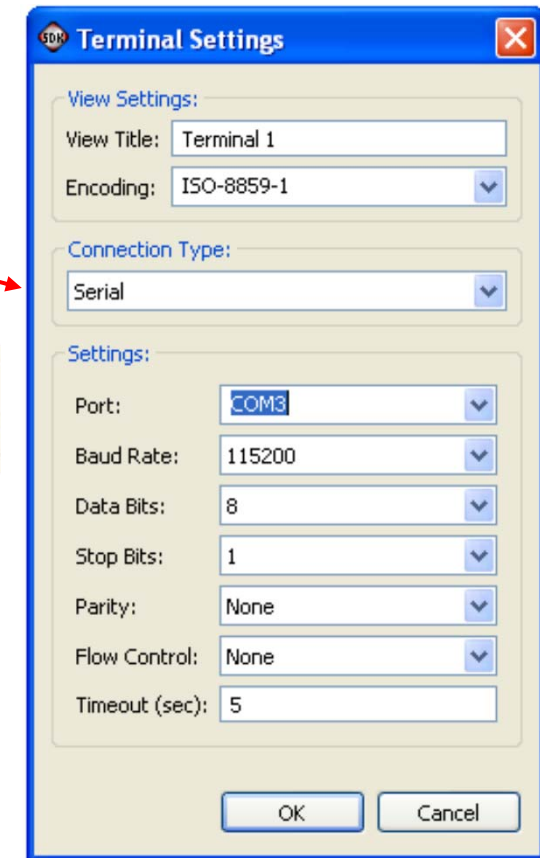
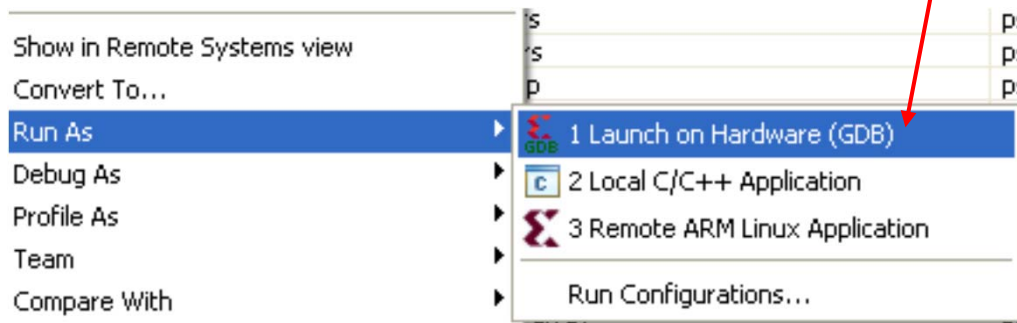


Xilinx SDK: Configure Terminal

- Configure the terminal (115200 bps)
- Program the FPGA
 - You are downloading only the HW



- Select the SW project. Right Click and select Run As -> Launch on HW (GDB)



Xilinx SDK: Result

- You can see the results in console

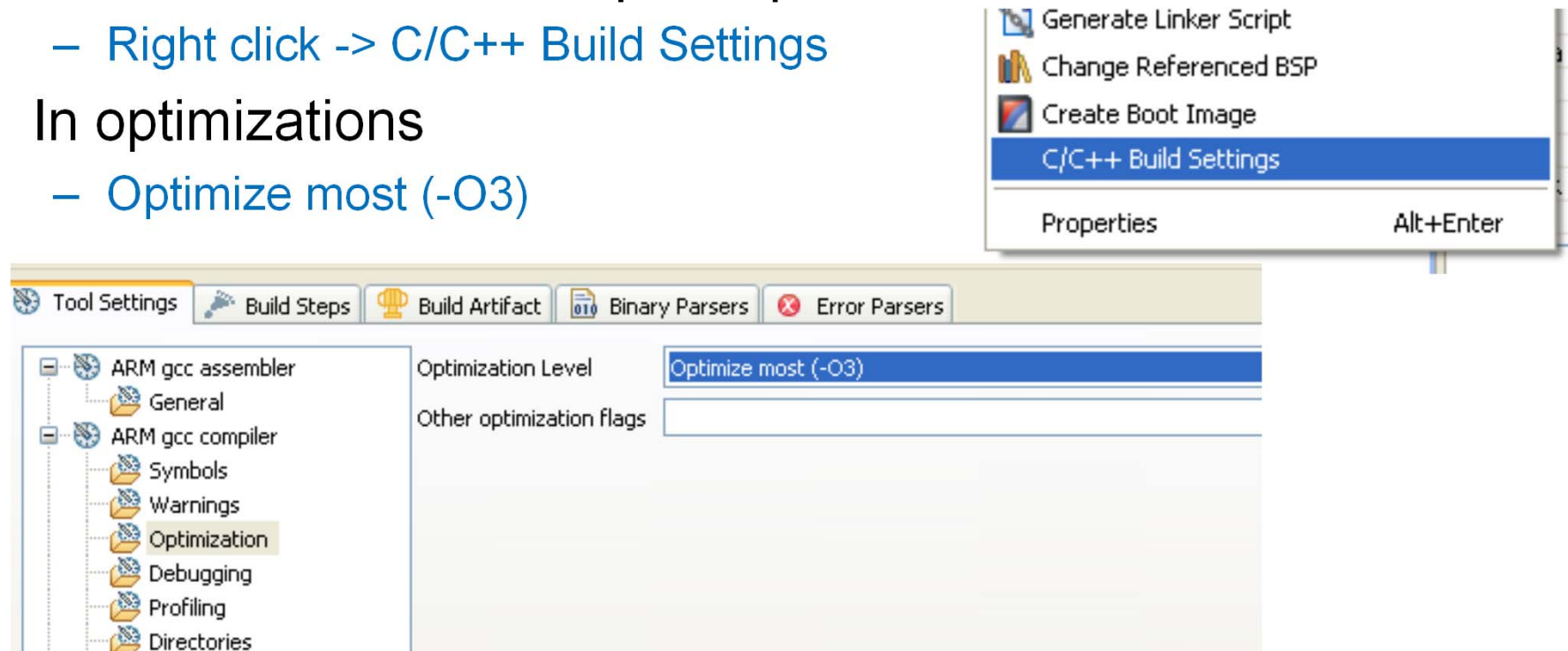
```
*****
FP MATRIX MULT in Vivado HLS + DMA

DMA Init done
Calibrating the timer:
init_time: 29 cycles.
curr_time: 51 cycles.
calibration: 22 cycles.
Loop time for 10000 iterations is 15005 cycles.
Before Start

Total run time for SW on ARM Processor (bare metal) is 152841 cycles.
MatMultAccel Status: isDone 0, isIdle 1, isReady1
MatMultAccel Status: isDone 0, isIdle 0, isReady0
Cache cleared
Setup HW accelerator done
Total run time for AXI DMA + HW accelerator is 4961 cycles.
MatMultAccel Status: isDone 1, isIdle 1, isReady1
SW and HW results match!
Acceleration factor 30.808
```

Xilinx SDK: Result

- This is a quite unfair comparison.
- We will enable the compiler optimizations
 - Right click -> C/C++ Build Settings
- In optimizations
 - Optimize most (-O3)



Xilinx SDK: Result

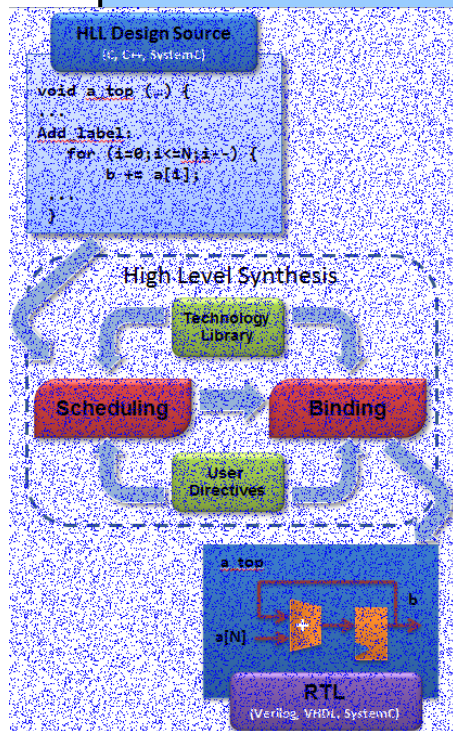
- Now results are less impressive, but not bad ;)

```
*****
FP MATRIX MULT in Vivado HLS + DMA

DMA Init done
Calibrating the timer:
init_time: 27 cycles.
curr_time: 49 cycles.
calibration: 22 cycles.
Loop time for 10000 iterations is 0 cycles.
Before Start

Total run time for SW on ARM Processor (bare metal) is 26238 cycles.
MatMultAccel Status: isDone 0, isIdle 1, isReady1
MatMultAccel Status: isDone 0, isIdle 0, isReady0
Cache cleared
Setup HW accelerator done
Total run time for AXI DMA + HW accelerator is 4954 cycles.
MatMultAccel Status: isDone 1, isIdle 1, isReady1
SW and HW results match!
Acceleration factor: 5.296
```

Agenda



- Setup and introduction
 - Zynq, ACP, DMA project
- Create HLS project and export rtl
 - Improve solution, Add AXI4 interfaces, export XACT
- Generate the Zynq project
 - Generate the design, Add the peripheral, connect, Export to SDK
- Generate the Zynq SW in SDK
 - Create a sw project, evaluate results
- Another more efficient example

Improving Results

- We can improve the results in various directions.
 - We can work at 200 MHz instead in 100 MHz in the PS
 - We will modify the core in order to receive first the B matrix and then with each A row compute a new row solution.
 - Then we use a similar approach as in streamed multiplier from previous lab.
 - We will rewrite some unrolled loops in order to use an addition tree and optimize latency.
 - It seems that Vivado-HLS has some errors in code generation. We use a workaround to generate the expected HW.
 - I will explain in more details (not yet)...

Improving Results

- Vivado-HLS results
 - Half cycles, at 200 MHz
 - Uses half BRAMs.

Name	BRAM_18K	DSP48E	FF	LUT
Total	16	158	22172	26061

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	5.00	5.03	0.63

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2139	2139	2140	2140	none

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- read_b1_read_b2	1024	1024	2	1	1	1024	yes
- Cache_Row_a_0	33	33	2	1	1	32	yes
- Row_Col	1076	1076	54	1	1	1024	yes

Improving Results

- Observe the improvements.
 - Now the Timer is also at 200 MHZ

```
*****  
FP MATRIX MULT in Vivado HLS + DMA, optimized
```

```
DMA Init done  
Calibrating the timer:  
init_time: 64 cycles.  
curr_time: 99 cycles.  
calibration: 35 cycles.  
Loop time for 10000 iterations is 30007 cycles.  
Before Start
```

```
Total run time for SW on ARM Processor (no SO) is 305680 cycles.  
MatMultAccel Status: isDone 0, isIdle 1, isReady1  
MatMultAccel Status: isDone 0, isIdle 0, isReady0  
Cache cleared  
Setup HW accelerator done  
Total run time for AXI DMA + HW accelerator is 3128 cycles.  
MatMultAccel Status: isDone 1, isIdle 1, isReady1  
SW and HW results match!  
Acceleration factor: 97.723
```

```
*****  
FP MATRIX MULT in Vivado HLS + DMA, optimized
```

```
DMA Init done  
Calibrating the timer:  
init_time: 75 cycles.  
curr_time: 106 cycles.  
calibration: 31 cycles.  
Loop time for 10000 iterations is -1 cycles.  
Before Start
```

```
Total run time for SW on ARM Processor (no SO) is 52527 cycles.  
MatMultAccel Status: isDone 0, isIdle 1, isReady1  
MatMultAccel Status: isDone 0, isIdle 0, isReady0  
Cache cleared  
Setup HW accelerator done  
Total run time for AXI DMA + HW accelerator is 3109 cycles.  
MatMultAccel Status: isDone 1, isIdle 1, isReady1  
SW and HW results match!  
Acceleration factor: 16.895
```

gcc -o0

gcc -o3

Questions?