香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lab08: High Level Synthesis Exercise

Task

- **Accelerating Floating Point Matrix Multiplication with High-Level Synthesis**

mmult.h

```
template <typename T, int DIM>
void mmult_hw(T A[DIM][DIM], T B[DIM][DIM], T C[DIM][DIM])
{
  // matrix multiplication of a A*B matrix
  L1:for (int ia = 0; ia < DIM; ++ia)
  {
    L2:for (int ib = 0; ib < DIM; ++ib)
    {
      T sum = 0;
      L3:for (int id = 0; id < DIM; ++id)
      {
        sum += A[ia][id] * B[id][ib];
      }
    C[ia][ib] = sum;
  }
}
```

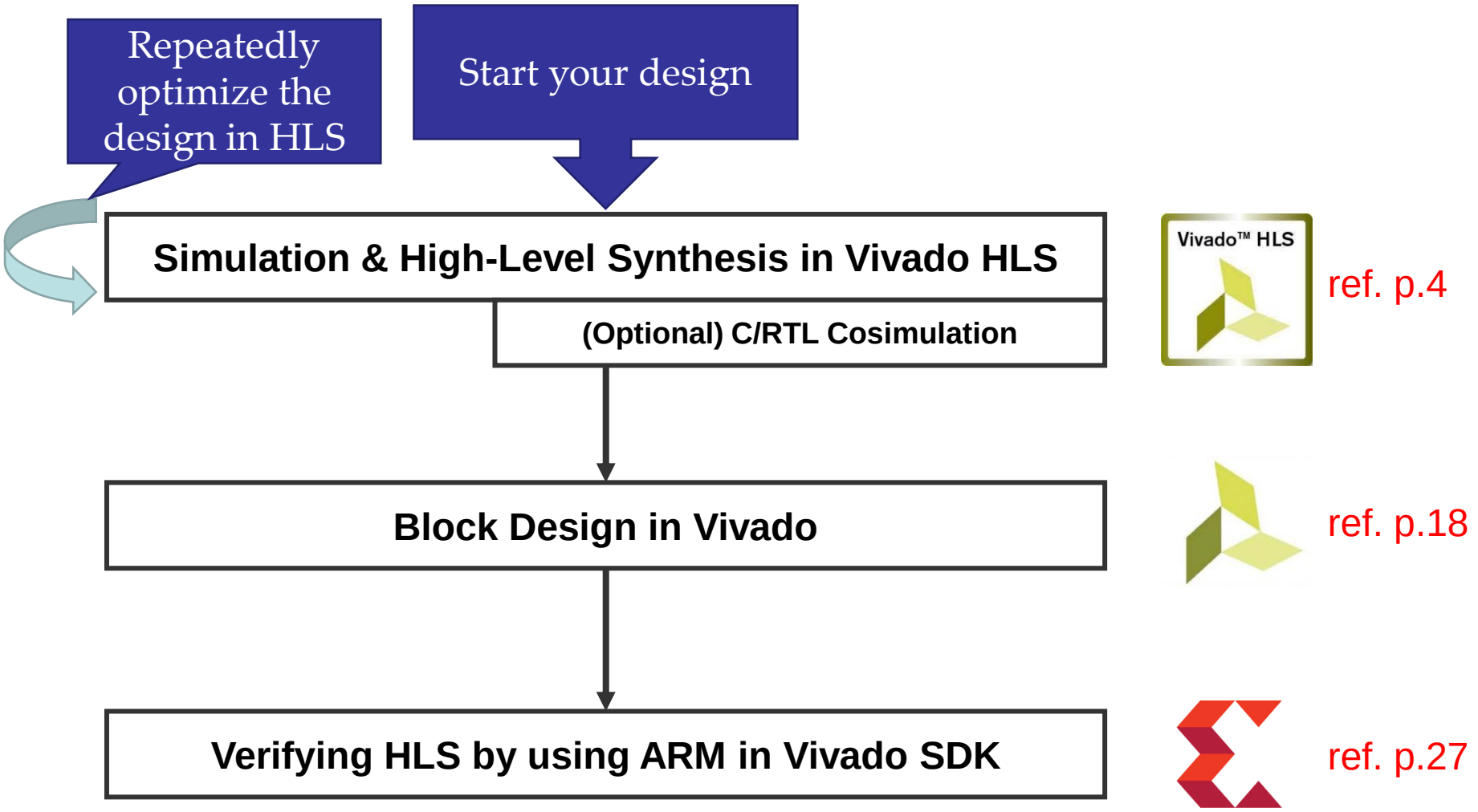
1st Loop

2nd Loop

3rd Loop

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

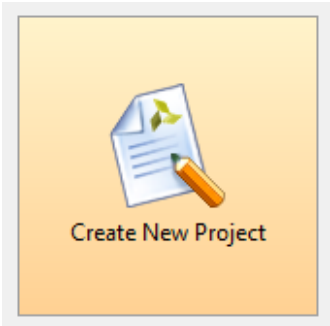
Workflow

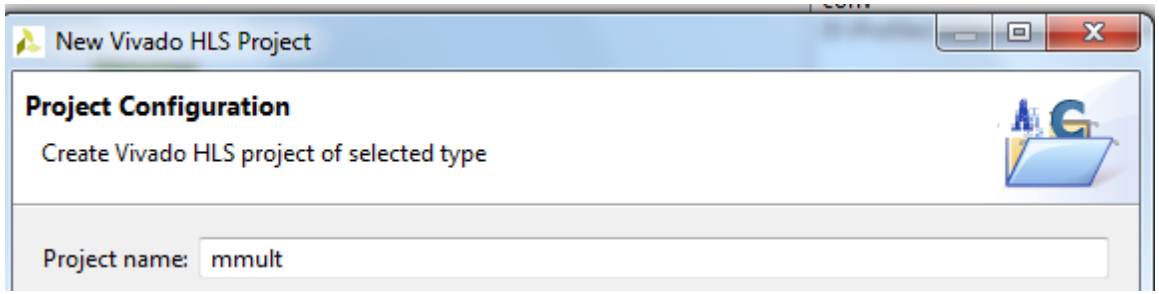

Repeatedly optimize the design in HLS
Start your design
Simulation & High-Level Synthesis in Vivado HLS
(Optional) C/RTL Cosimulation
Block Design in Vivado
Verifying HLS by using ARM in Vivado SDK
Vivado™ HLS
ref. p.4
ref. p.18
ref. p.27

High-Level Synthesis

Step 1 - High-Level Synthesis (HLS)

- **Download the lab08 material from Blackboard**
- Open **Vivado HLS 2016.3** and Create a new project called “mmult”

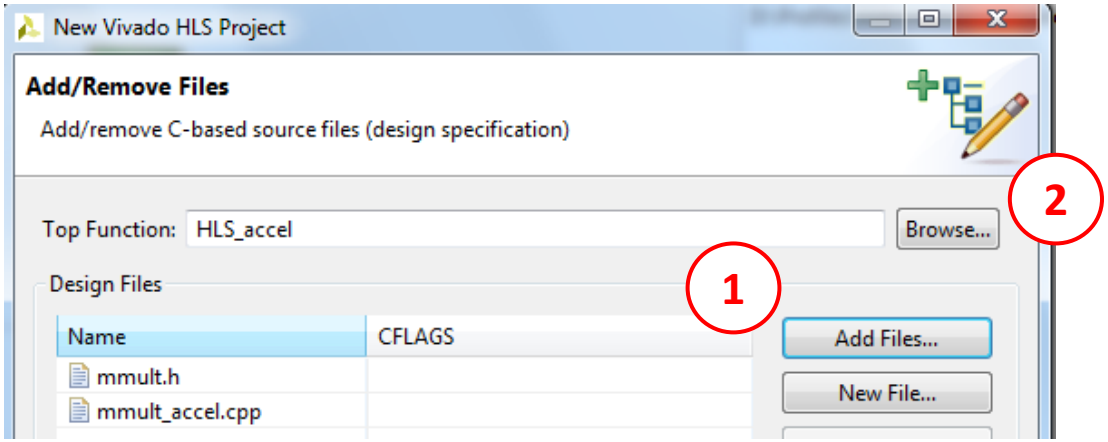


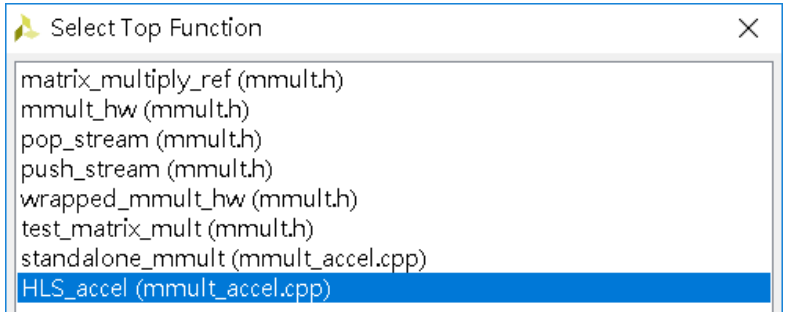


Remarks: Avoid naming your folder with space “ ”
e.g. “`C:\Users\Lab 8`” may cause some trouble
Change to `Lab_8` or `Lab08` to avoid errors from Vivado

Step 2 - High-Level Synthesis (HLS)

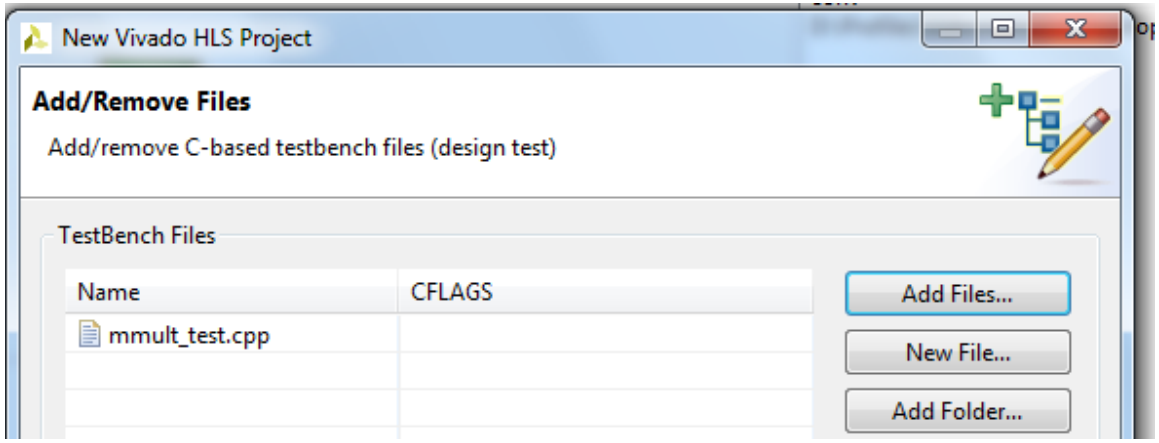
- Add the two C-based source files (**mmult.h and mmult_accel.cpp**) into the project and specify the "**HLS_accel**" as Top Function by click Browse:





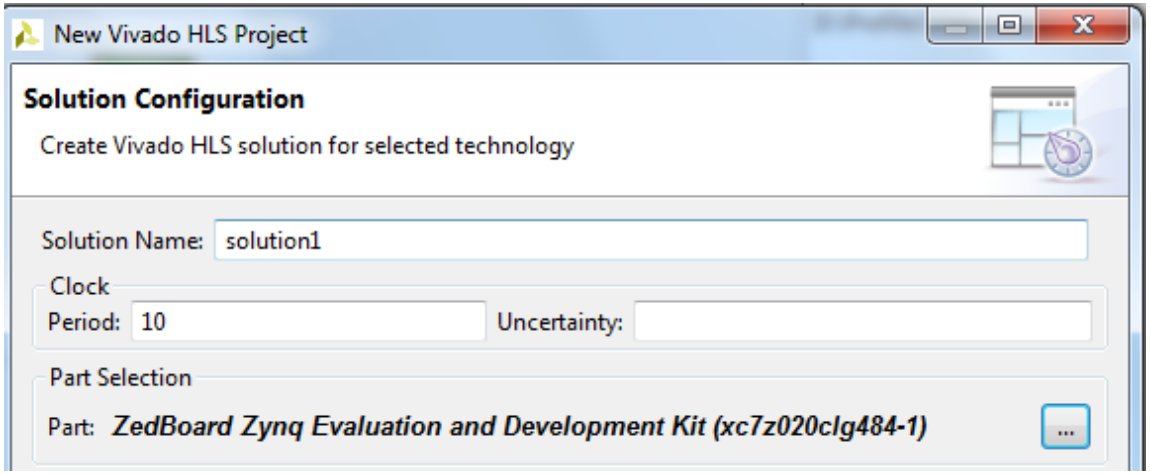
Step 3 - High-Level Synthesis (HLS)

- Add the C-based testbench file (**mmult_test.cpp**) into the project:



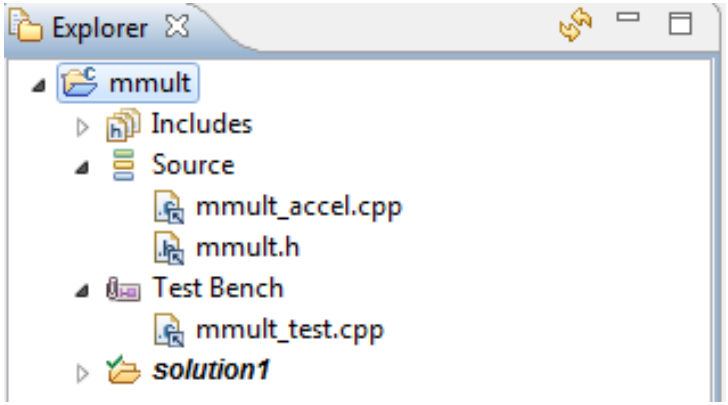
Step 4 - High-Level Synthesis (HLS)

- **Specify Zedboard as the "Part"**, and make sure that the clock period (i.e., the timing constraint) is specified as "**10**" (ns) since the on-board clock frequency is 100 MHz and the clock uncertainty is left empty.



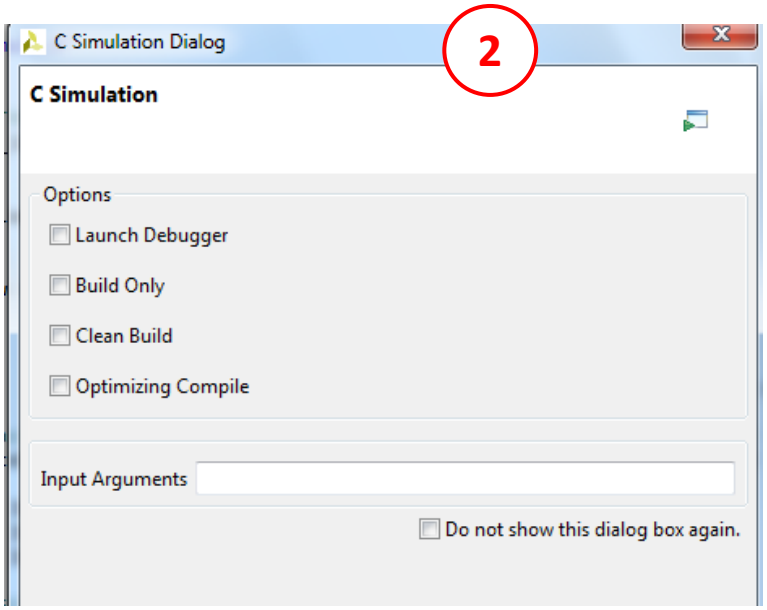
Step 5 - High-Level Synthesis (HLS)

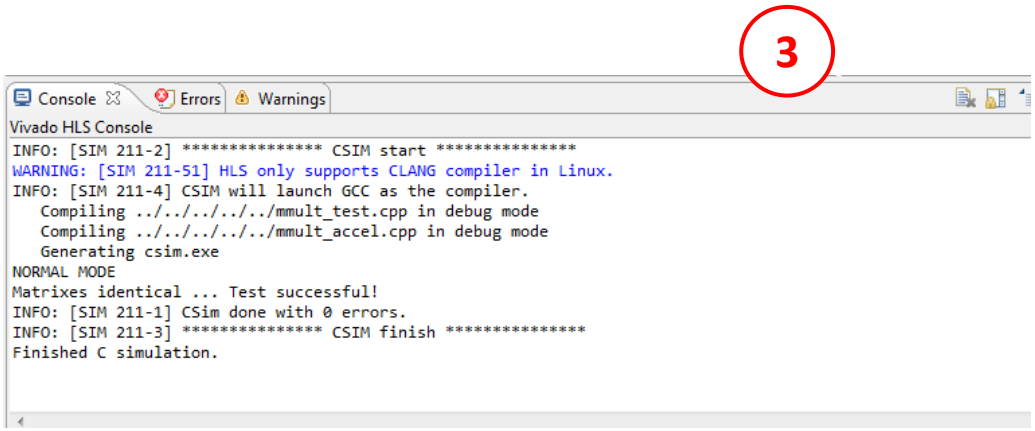
- Finish it and check the file structure:



Step 6 - High-Level Synthesis (HLS)

- Run **C Simulation**
  - Simply click OK in C Simulation Dialog

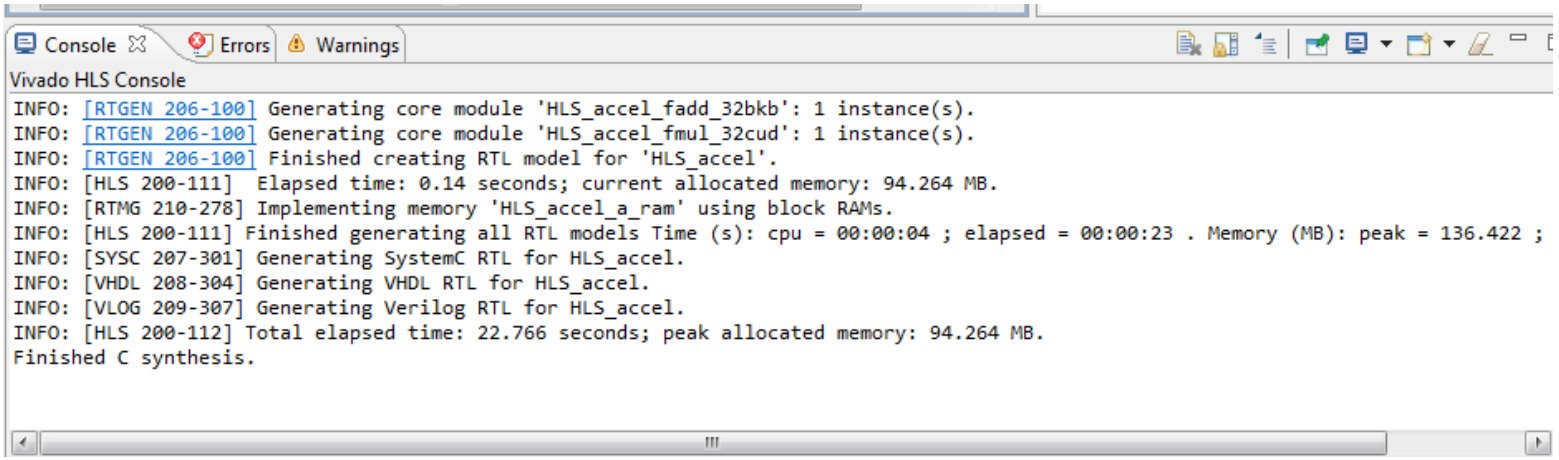




Expected Outcome

Step 7 - High-Level Synthesis (HLS)

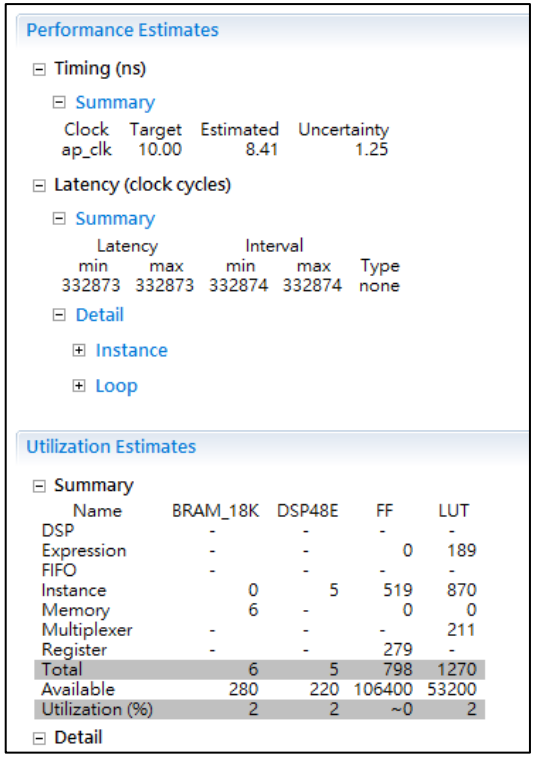
- Run **C Synthesis**



Expected Outcome

Step 8 - High-Level Synthesis (HLS)

- Check the "**Performance Estimates**" and "**Utilization Estimates**" in the "Synthesis Report for HLS_accel"
  - Solution -> Open Report -> Synthesis



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.41	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
332873	332873	332874	332874	none

Detail

Instance

Loop

Utilization Estimates

Summary

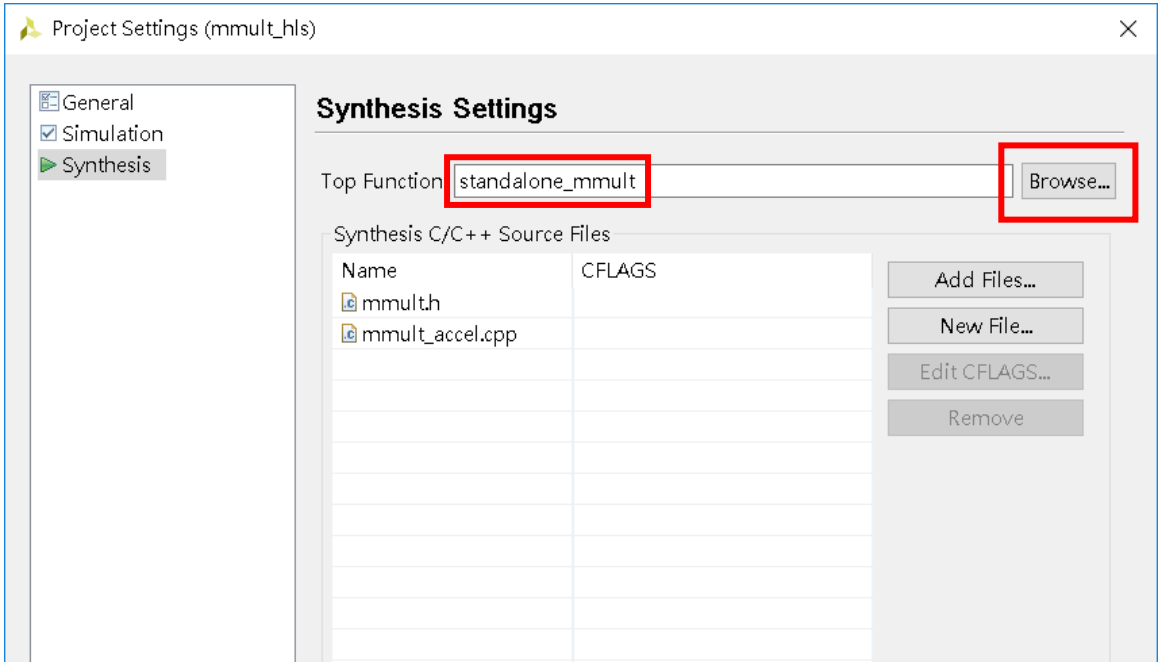
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	189
FIFO	-	-	-	-
Instance	0	5	519	870
Memory	6	-	0	0
Multiplexer	-	-	-	211
Register	-	-	279	-
Total	6	5	798	1270
Available	280	220	106400	53200
Utilization (%)	2	2	~0	2

Detail

By default, this window will pop out

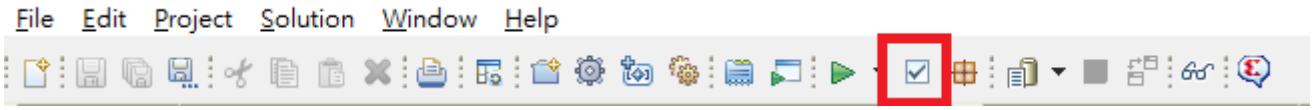
(Optional) Step 1- C/RTL Cosimulation

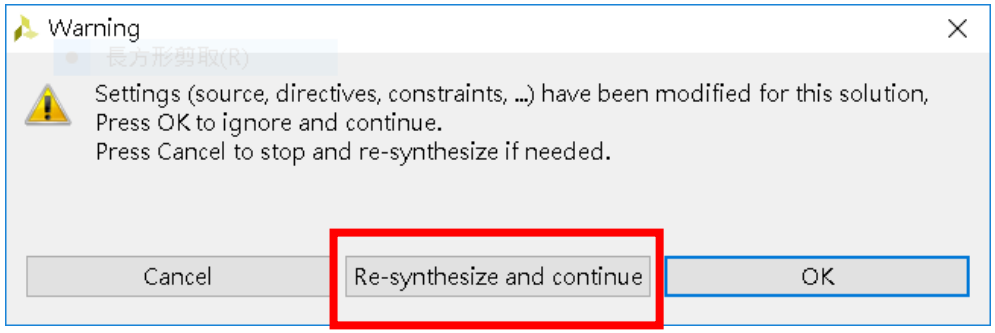
- Click Project -> Project Setting -> Synthesis -> Top Function -> Browse
  - Change Top Function to "**standalone_mmult**"

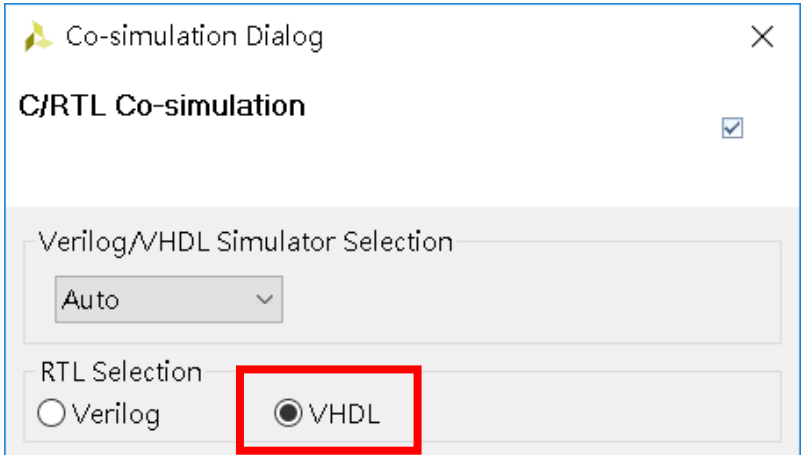


(Optional) Step 2- C/RTL Cosimulation

- Run C/RTL Cosimulation
  - Choose “Re-synthesize and Continue” in the “Warning” pop-up window
  - Choose “VHDL” in the RTL Selection of the “Co-simulation Dialog” as follows)

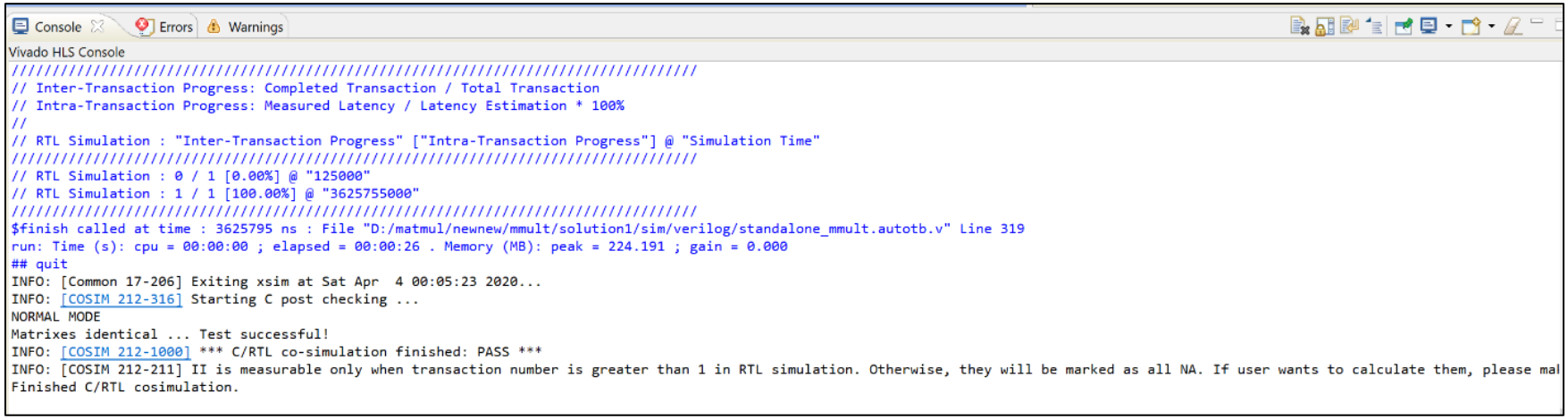






(Optional) Step 3- C/RTL Cosimulation

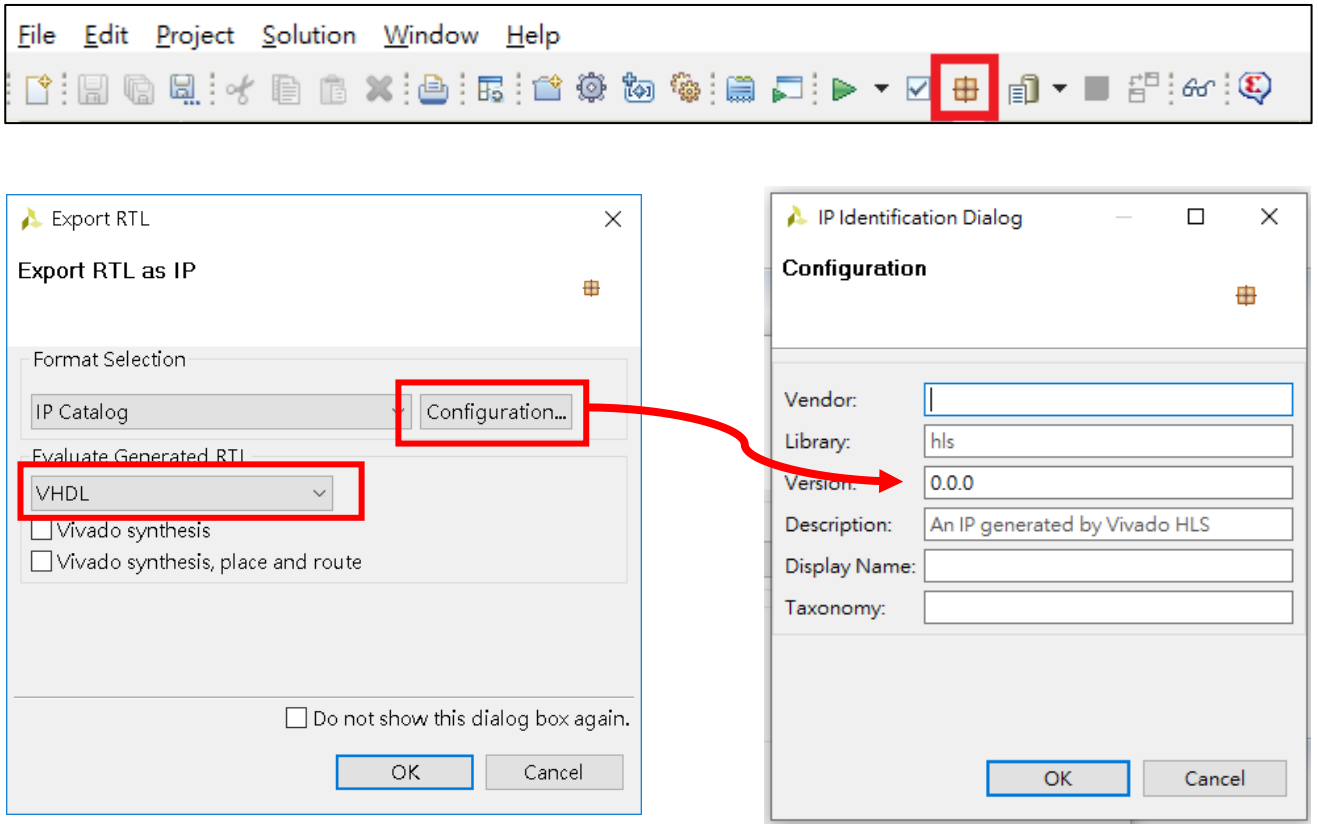
- Check the console and wait until C/RTL cosimulation is finished



- Click Project -> Project Setting -> Synthesis -> Top Function -> Browse
  - Change Top Function back to "HLS_accel" and Run C Synthesis again

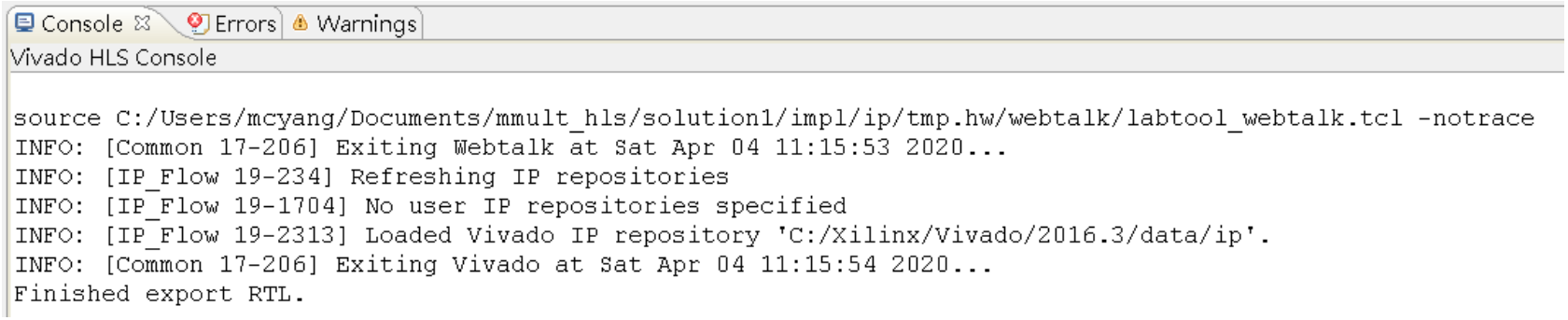
Step 9 - High-Level Synthesis (HLS)

- Export RTL
  - Choose “VHDL” in “Evaluate Generated RTL”
  - Click “Configuration”, set version to “0.0.0”



Step 9 - High-Level Synthesis (HLS)

- Export RTL
  - Wait until the export RTL is finished
  - Close Vivado HLS

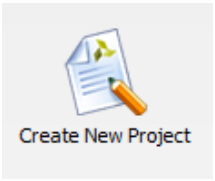


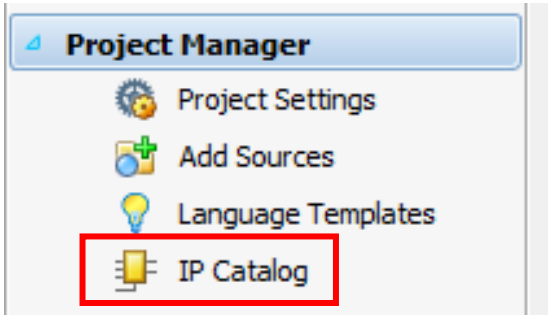
Expected Outcome

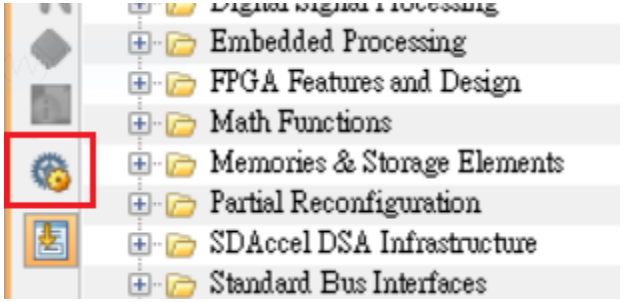
Block Design in Vivado

Step 1 - Block Design

- Open Vivado 2016.3 and Create a new project
  - Don’t need to create any files
- Click “IP Catalog” and then click “IP Settings”

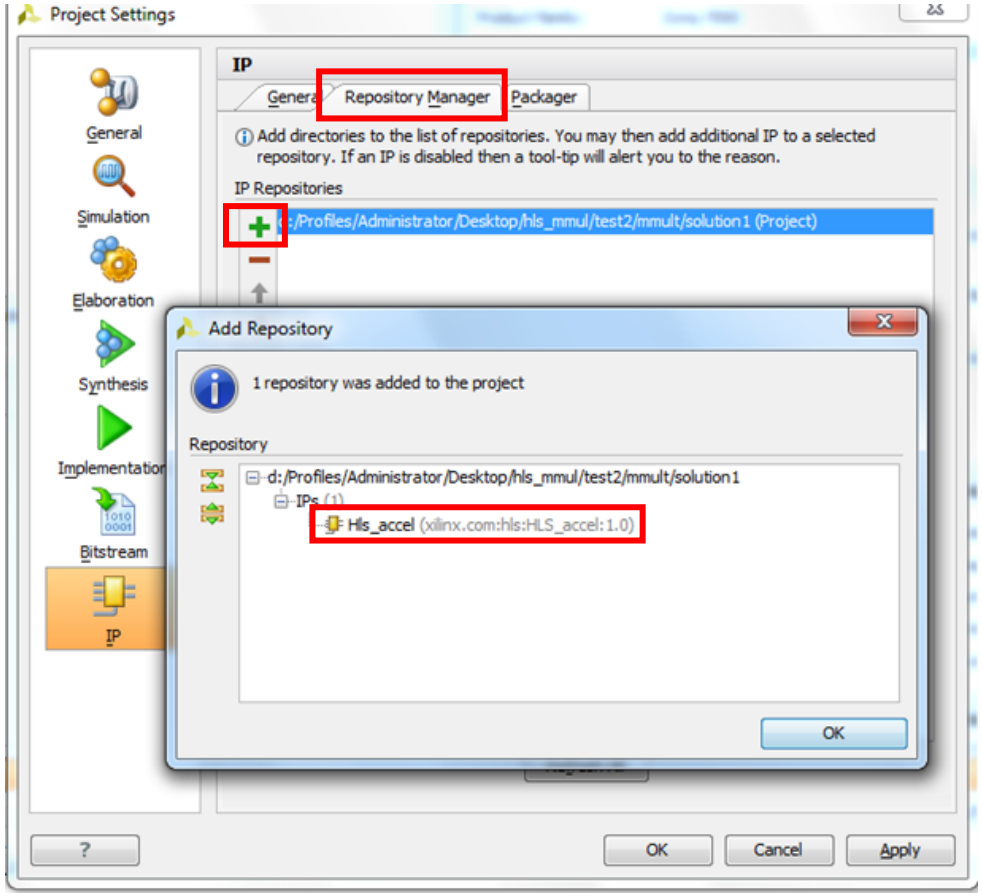






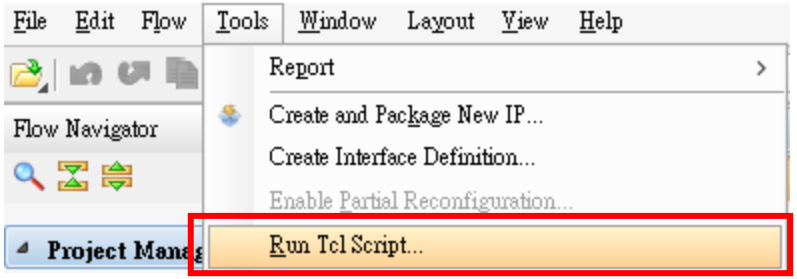
Step 2 - Block Design

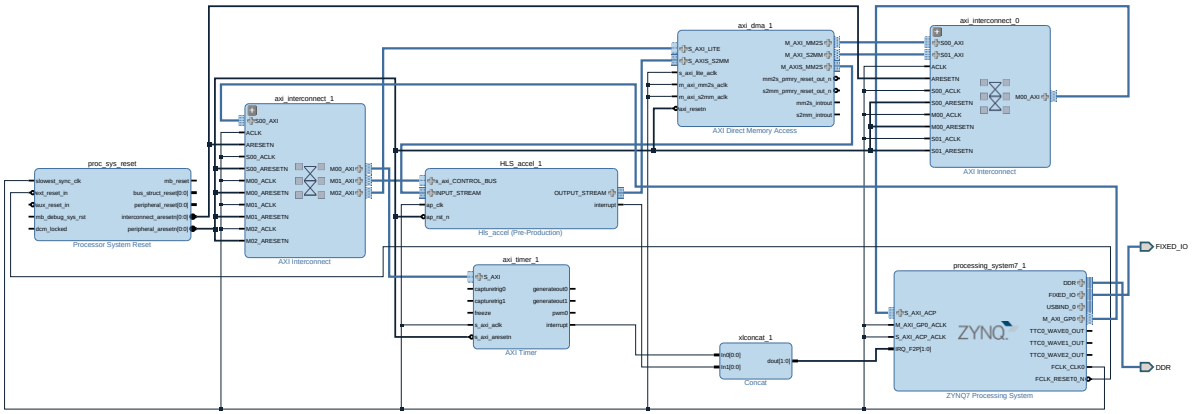
- At the "Repository Manager", click "+"
  - Add the repository of the HLS project
  - Then you can see the IP name "HLS_accel"

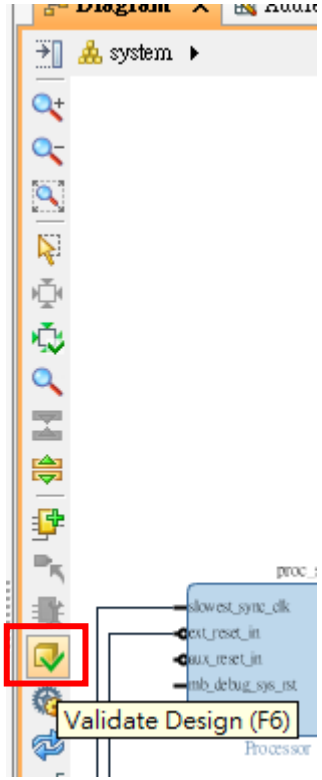


Step 3 - Block Design

- Click Tools -> Run Tcl Script
  - Select `system.tcl` from the lab material
  - The block design is generated as follow
  - Click the Validation Design to ensure the block design success

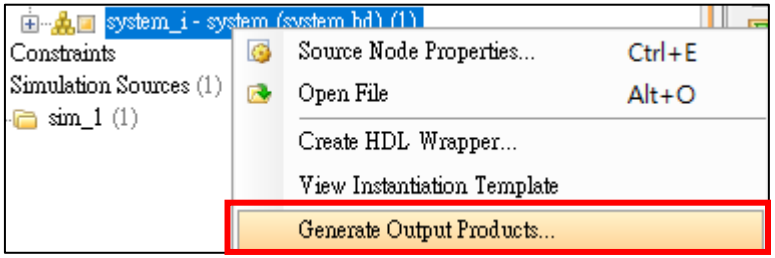


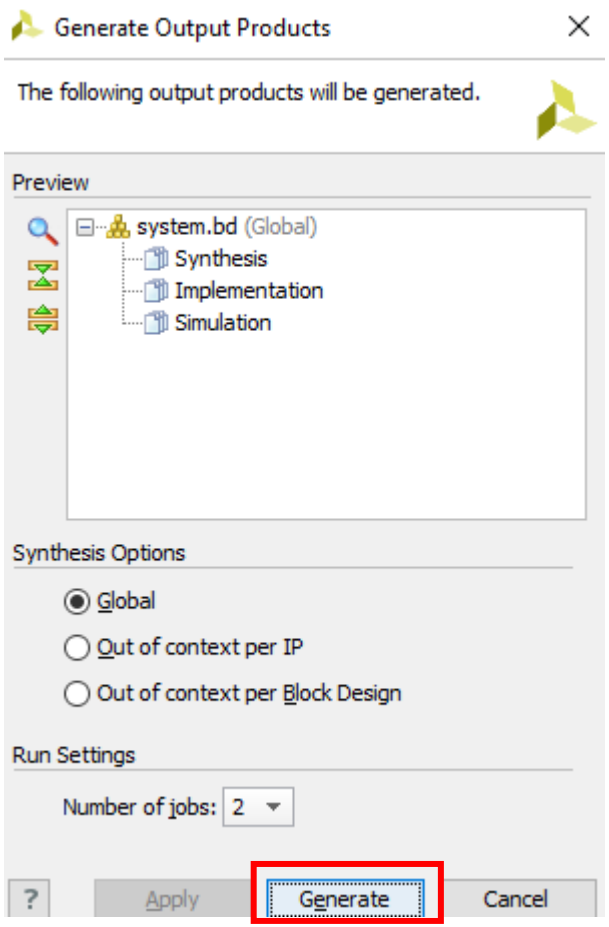




Step 4 - Block Design

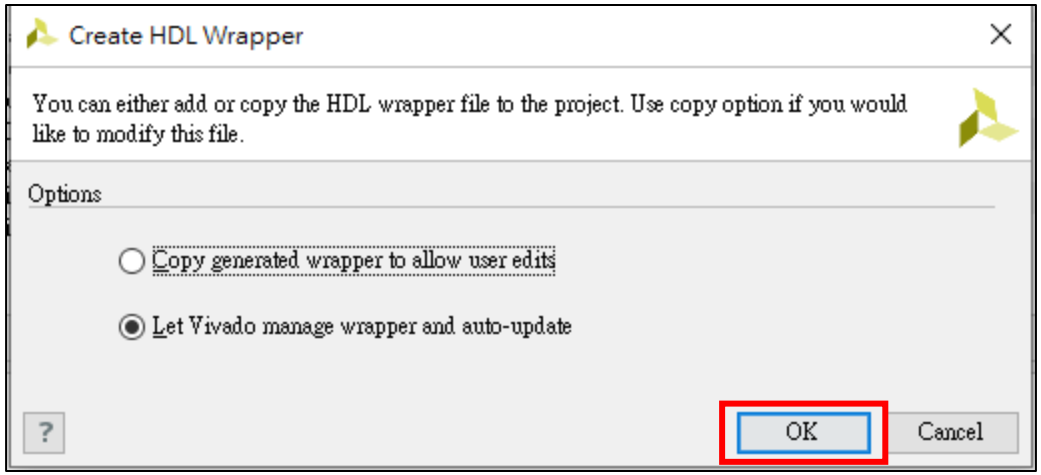
- Right click system.bd, click Generate Output Products
  - Click Generate





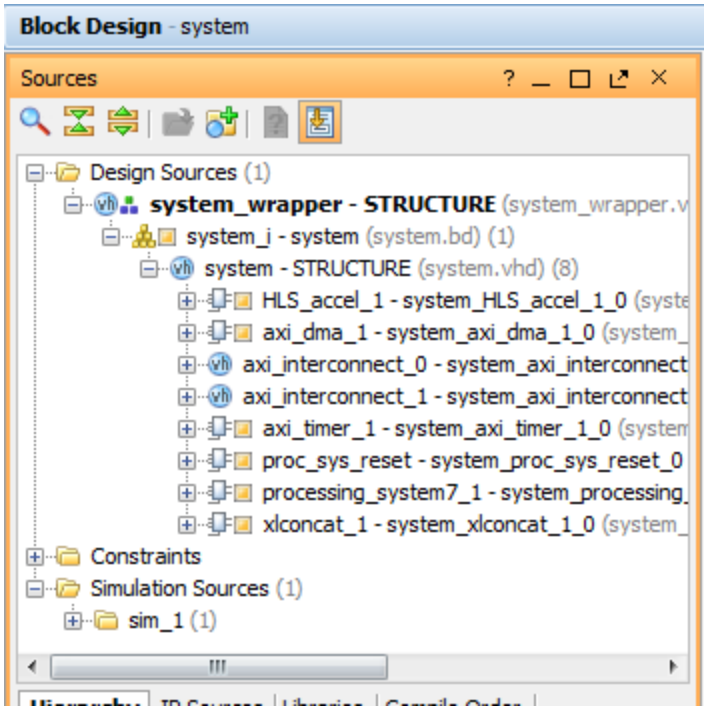
Step 5 - Block Design

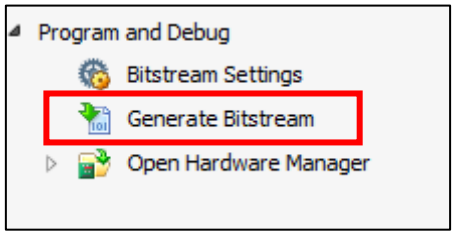
- Right click system.bd
  - Click Create HDL Wrapper and click OK

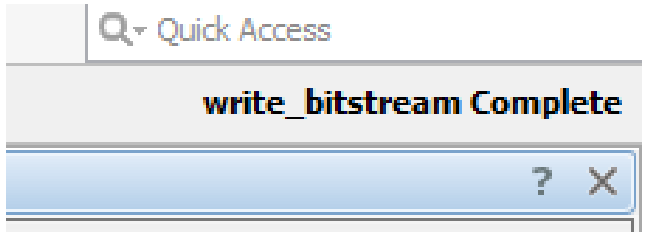


Step 6 - Block Design

- Then you can see the Design Sources like this: system_wrapper.vhd is generated
- Click Generate Bitstream and wait until the message “write_bitstream complete” is shown

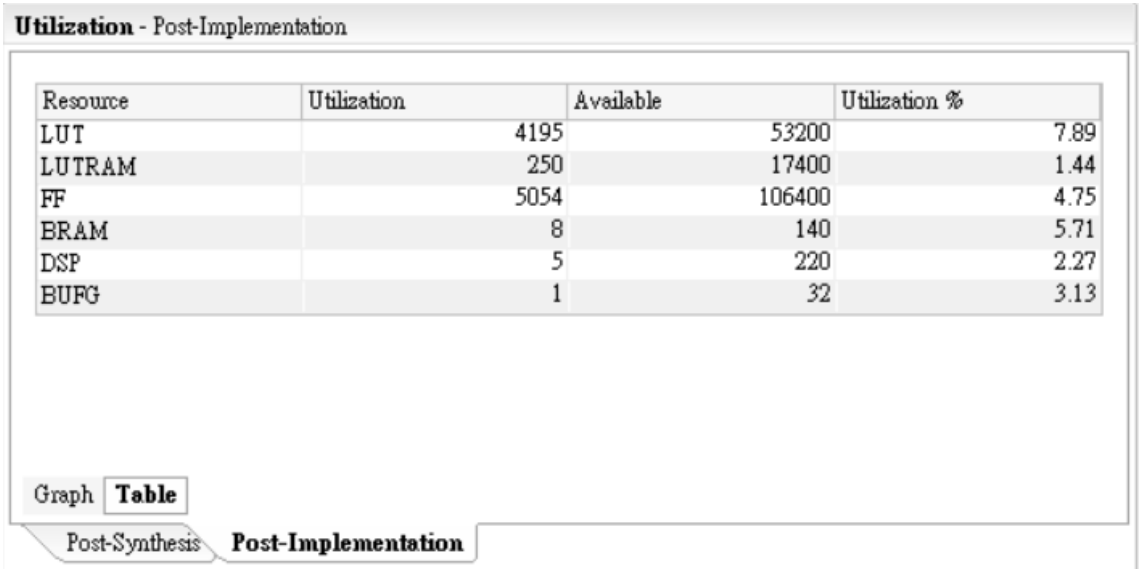






Step 7 - Block Design

- Check Utilization – Post-Implementation of your design by clicking Window -> Project Summary



Utilization - Post-Implementation

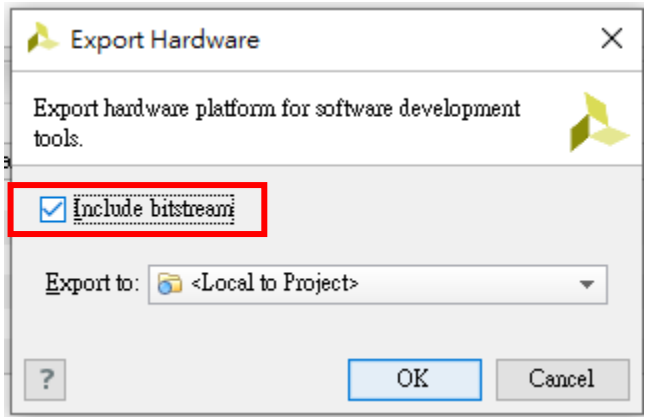
Resource	Utilization	Available	Utilization %
LUT	4195	53200	7.89
LUTRAM	250	17400	1.44
FF	5054	106400	4.75
BRAM	8	140	5.71
DSP	5	220	2.27
BUFG	1	32	3.13

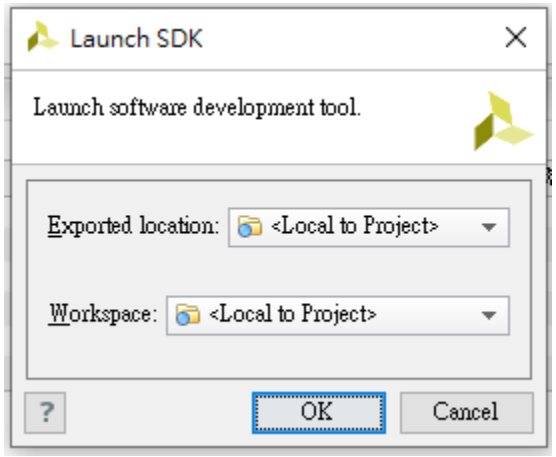
Graph **Table**

Post-Synthesis **Post-Implementation**

Step 8 - Block Design

- Click File->Export Hardware (Click “Include bitstream”)->OK
- Click Launch SDK



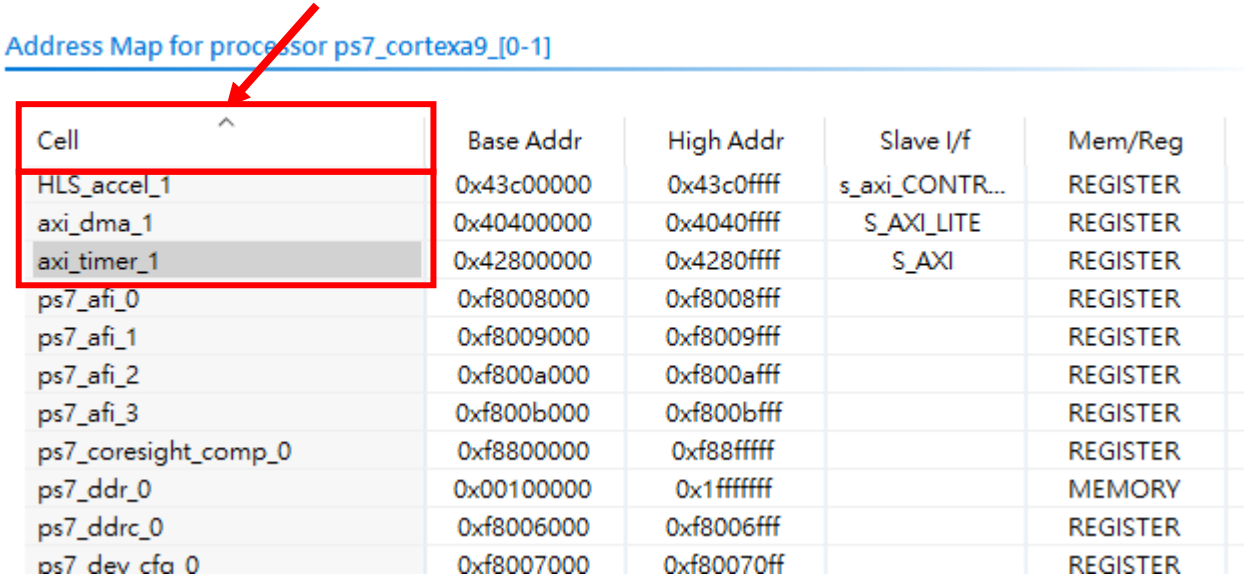


Verifying HLS by using ARM in Vivado SDK

Step 1 - Verifying HLS by using ARM

- After launch Vivado SDK, you have to check the existence of
  - HLS_accel_1
  - axi_dma_1
  - axi_timer_1

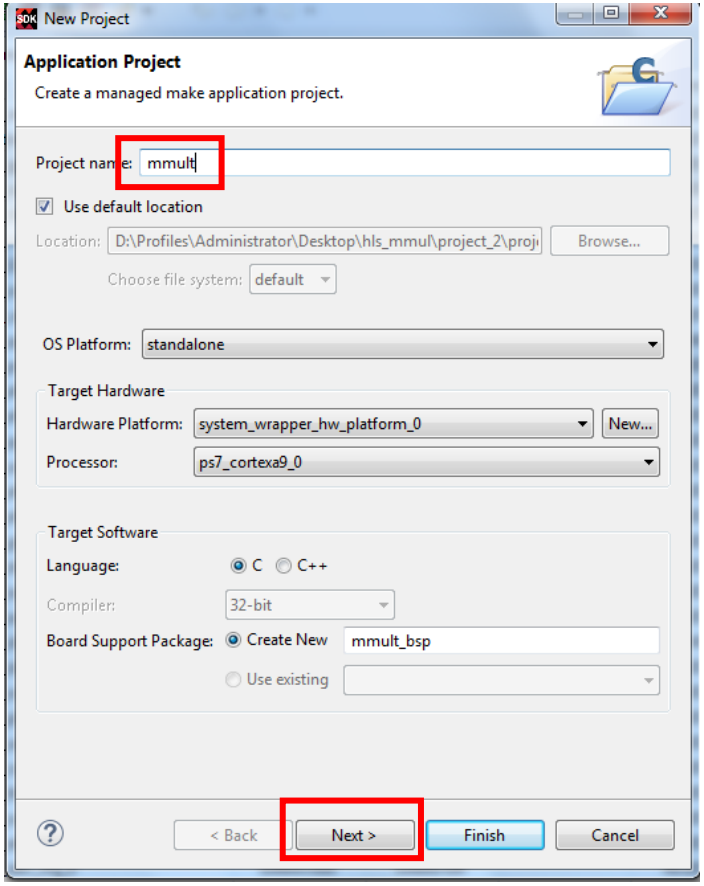
Click “Cell” to sort the table by cell name



Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
HLS_accel_1	0x43c00000	0x43c0ffff	s_axi_CONTR...	REGISTER
axi_dma_1	0x40400000	0x4040ffff	S_AXI_LITE	REGISTER
axi_timer_1	0x42800000	0x4280ffff	S_AXI	REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_coresight_comp_0	0xf8800000	0xf88fffff		REGISTER
ps7_ddr_0	0x00100000	0x1fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER

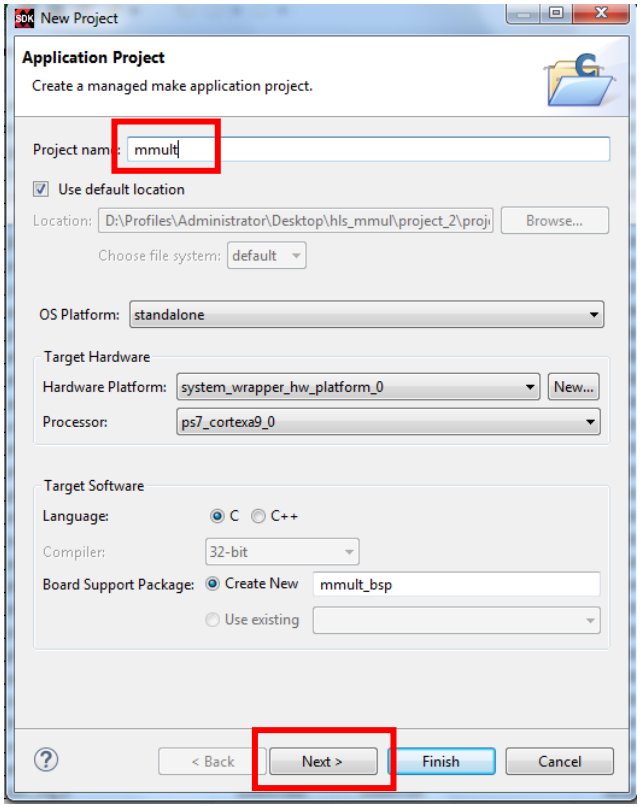
Step 2 - Verifying HLS by using ARM

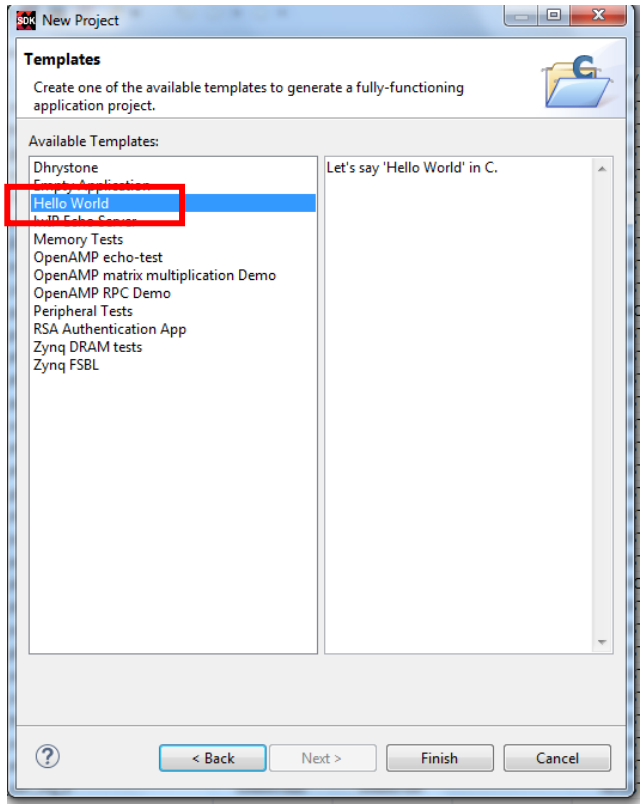
- Create a new application project by File->New->Application Project and input project name and click next:



Step 3 - Verifying HLS by using ARM

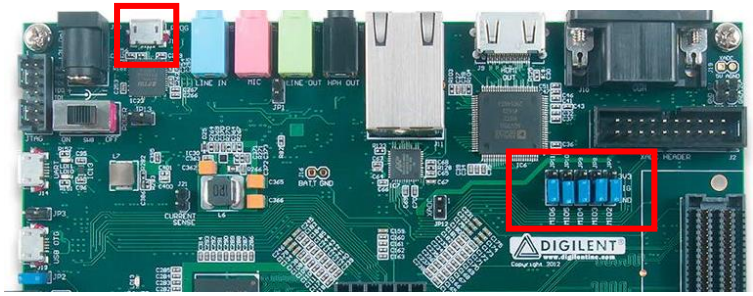
- Create a new application project by File->New->Application Project and input project name and click Next
  - Select Hello World Templates

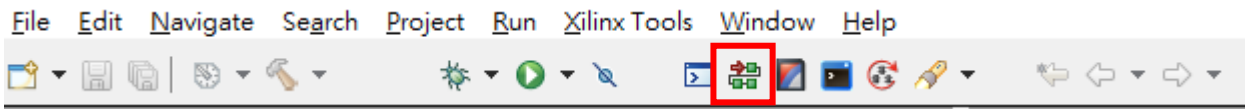




Step 4 - Verifying HLS by using ARM

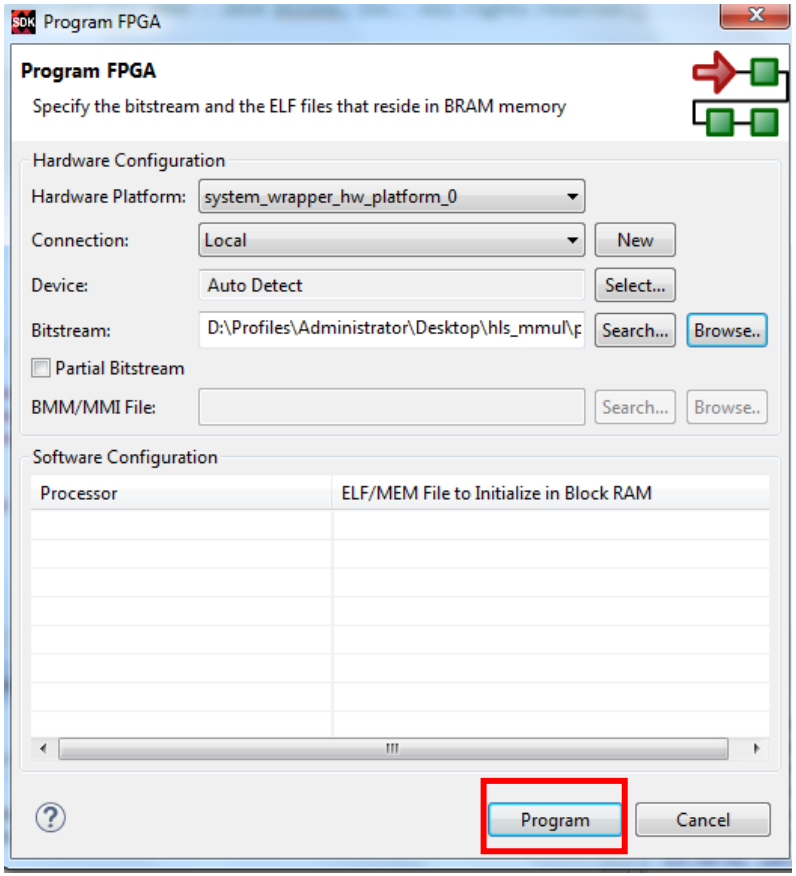
- Verify the boot mode are set as follows
- Connect the PROG port of Zed Board (J17) to a PC using the MicroUSB cable and Click Program FPGA





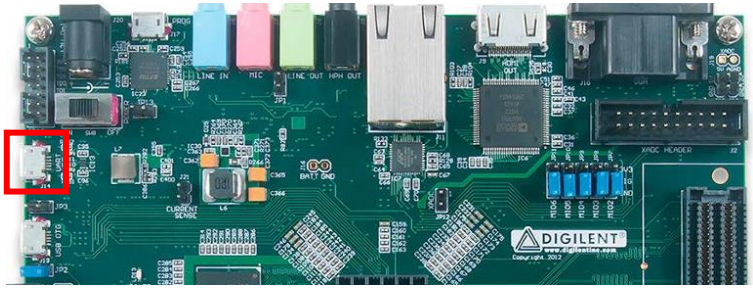
Step 5 - Verifying HLS by using ARM

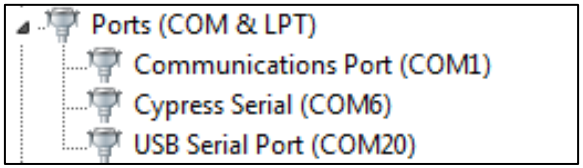
- Select bitstream file and click Program. After finish the program, you can see the green light is illumed.



Step 6 - Verifying HLS by using ARM

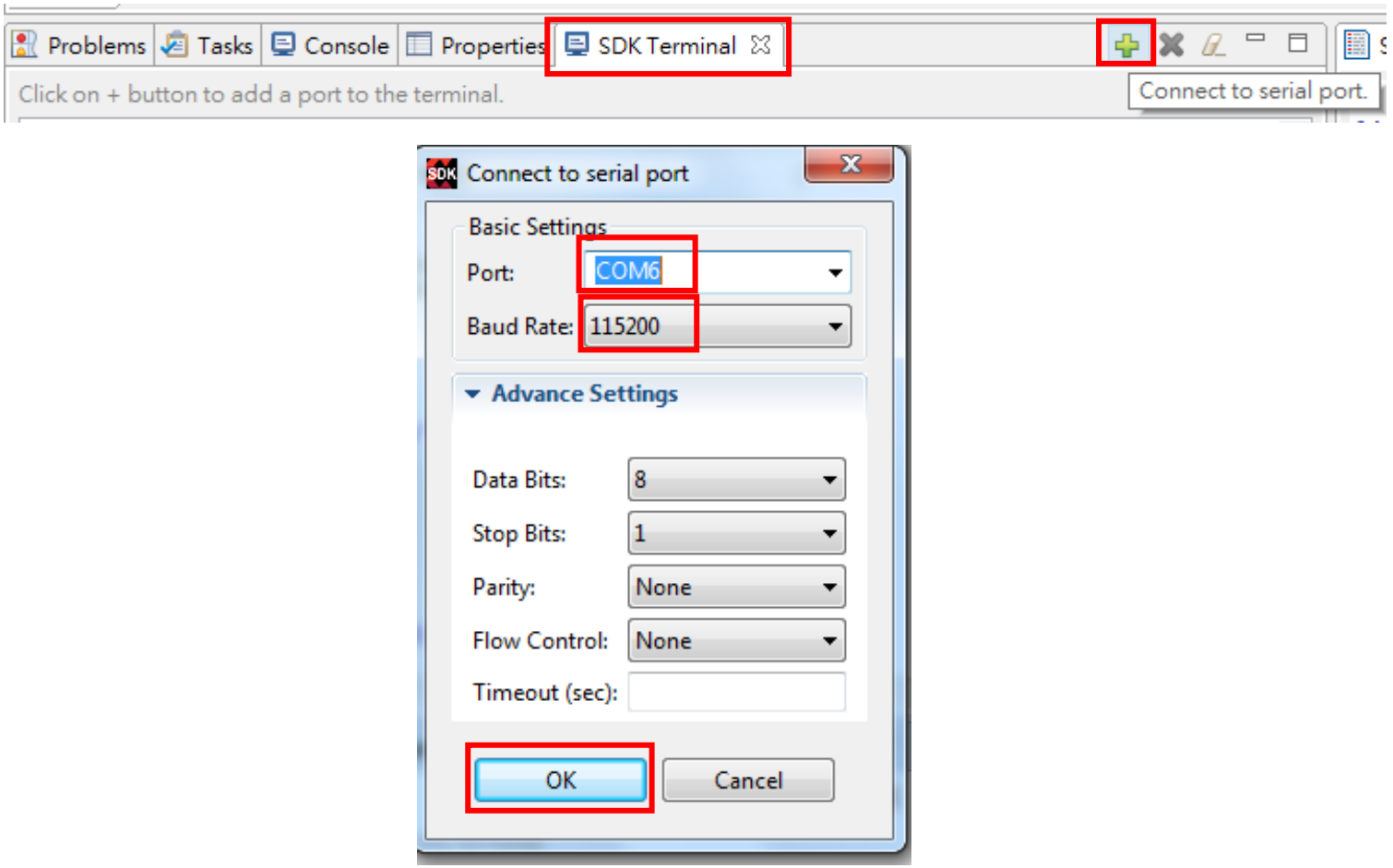
- Connect the USB-UART port of ZedBoard (J14) which is labeled UART to a PC using the MicroUSB cable
  - Check COM port at the Device Manager in Windows





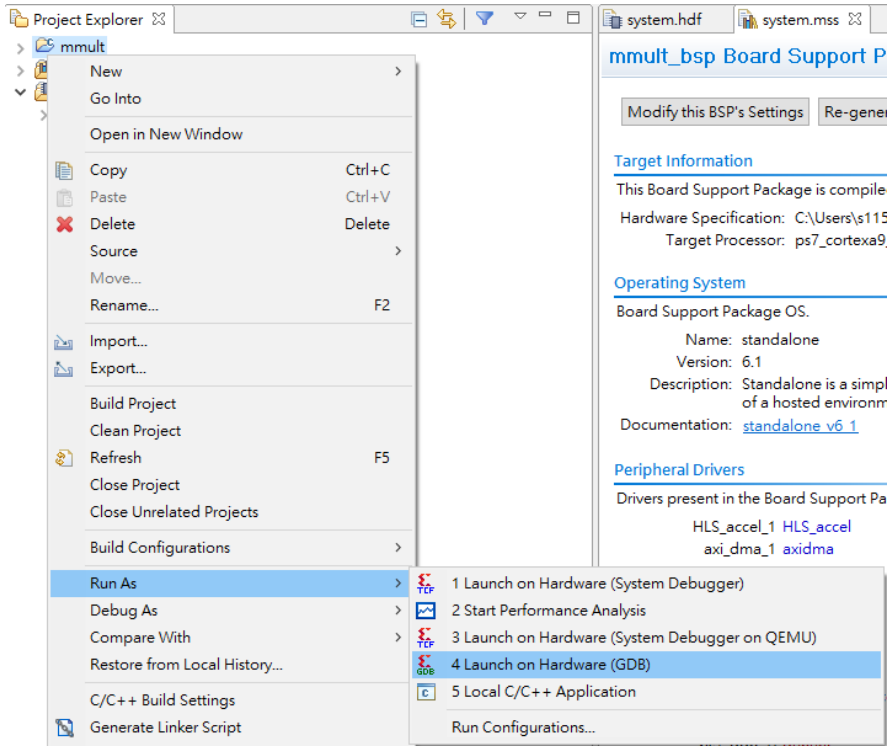
Step 7 - Verifying HLS by using ARM

- Set series setting in SDK Terminal by green “+” icon
  - Select the right Port and Baud Rate, and click OK



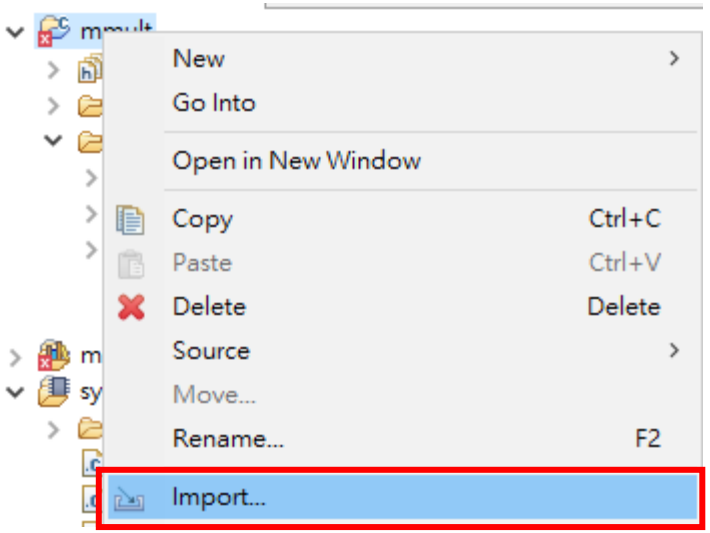
Step 8 - Verifying HLS by using ARM

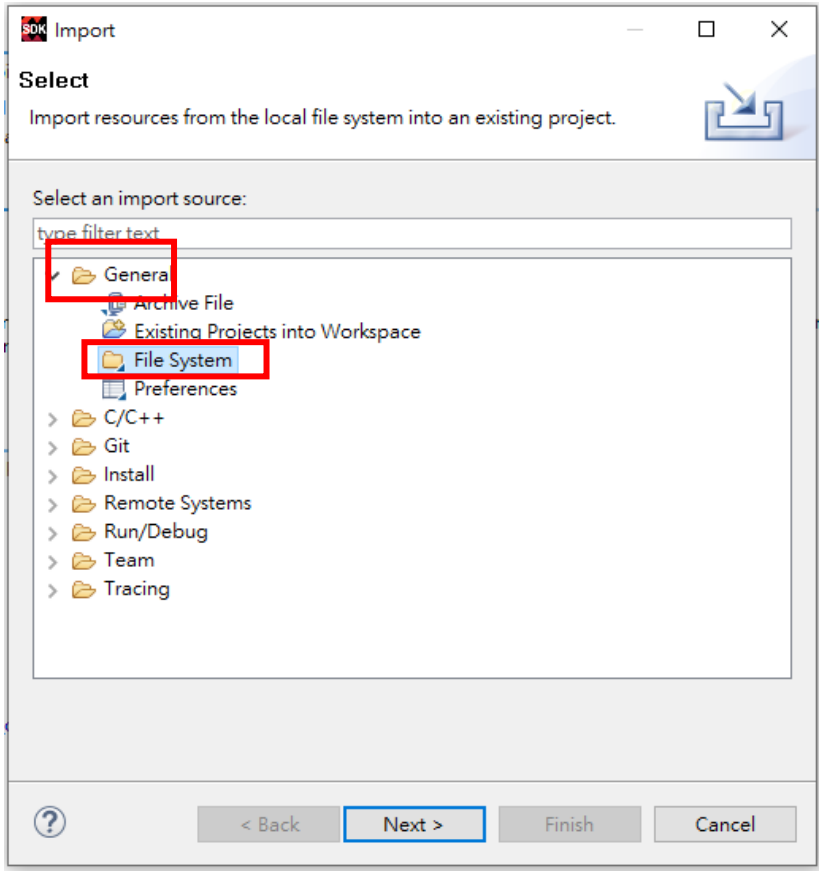
- Test COM port by right click “mmult”->Run As->Launch on Hardware (GDB). Then you can see Hello World in SDK Terminal
  - You should see the “Hello World” in SDK Terminal



Step 9 - Verifying HLS by using ARM

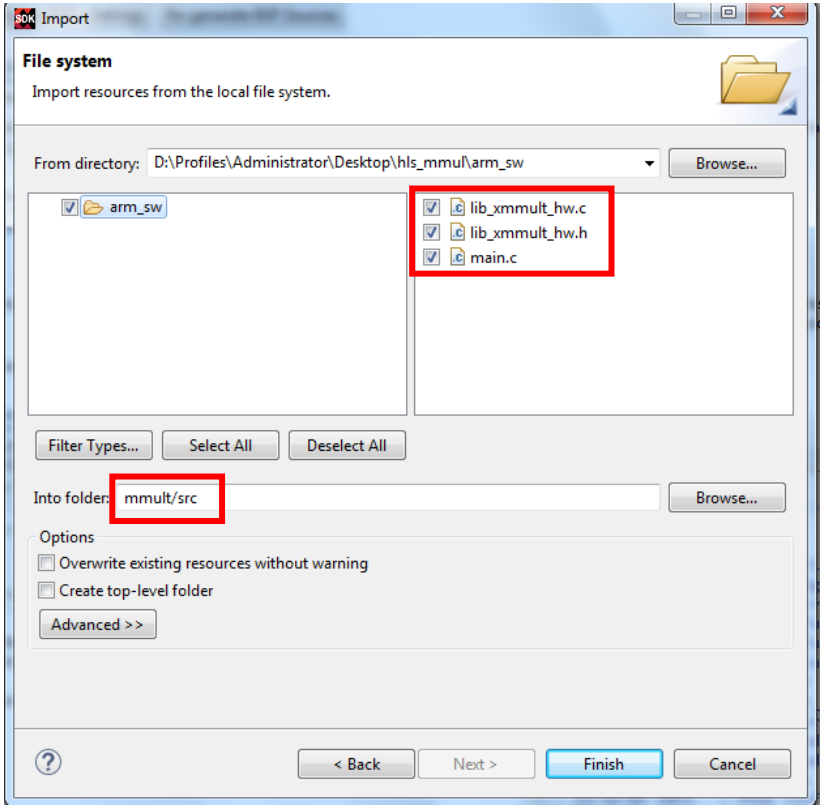
- Delete the helloworld.c file
- Right click mmult project, click “Import”
  - Select General -> File System





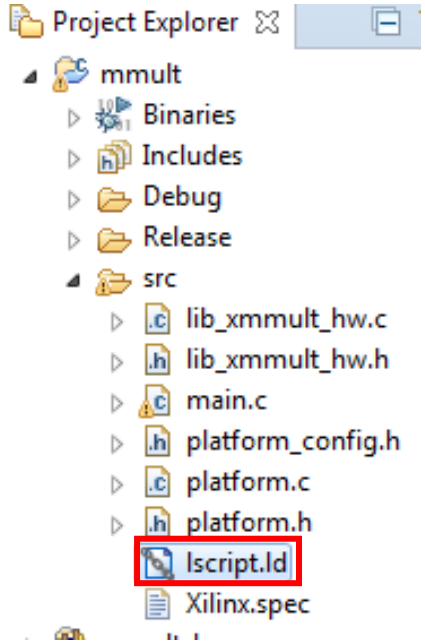
Step 10 - Verifying HLS by using ARM

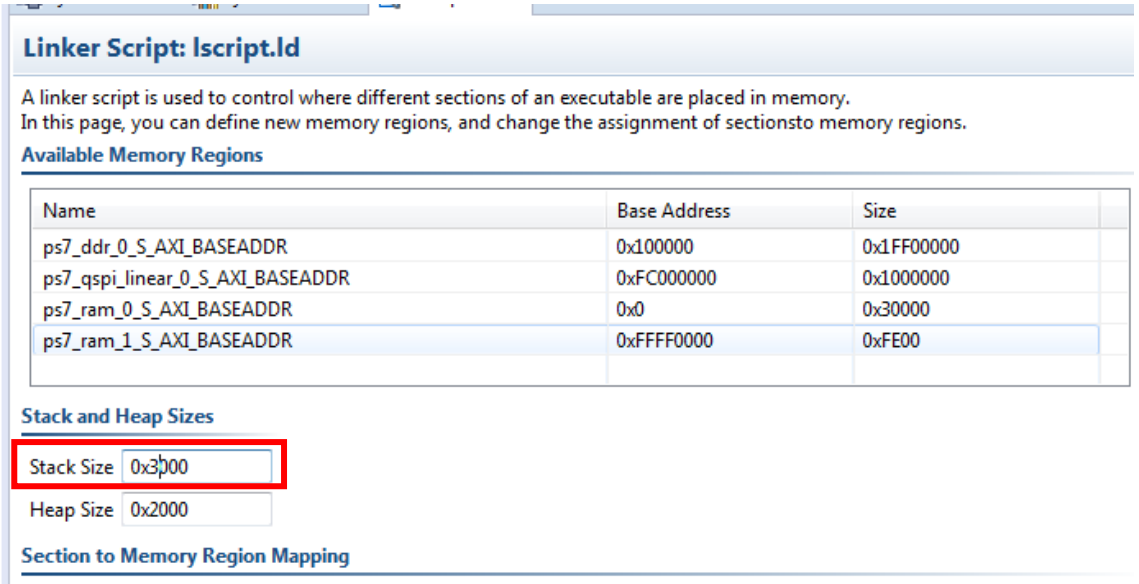
- Import the lib_xmmult_hw.h, lib_xmmult_hw.c, and main.c into mmult/src folder



Step 11 - Verifying HLS by using ARM

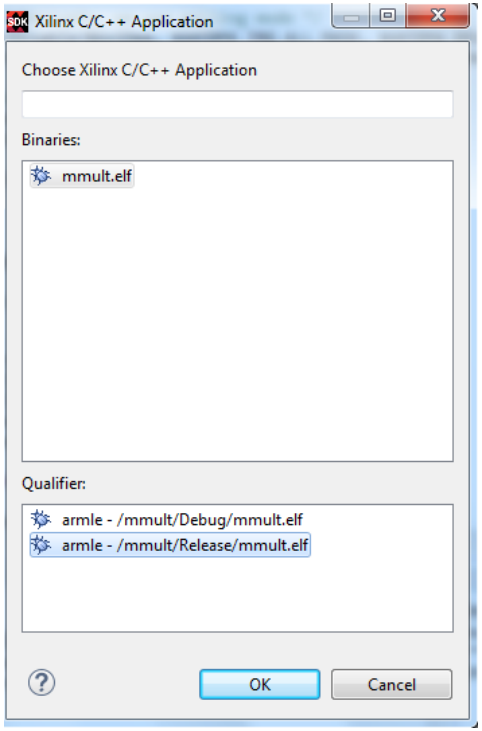
- In the Project Explorer mmult/src, double-click on the file lscript.ld.
  - Change the Stack Size from 0x2000 to 0x3000. Save the file (Ctrl + S)





Step 12 - Verifying HLS by using ARM

- Right click “mmult”
  - Select Build Configuration -> Build All
  - Select Run As -> Launch on Hardware (GDB) and select armlet-/mmult/Release/mmult.elf and click OK



Step 13 - Verifying HLS by using ARM

- Finally, you can see the SDK Terminal

SDK Log | Terminal 1

Serial: (COM6, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)

```
Hello World
************************************************************
DMA Init done
            Loop time for 1024 iterations is -2 cycles
                                                    Running Matrix Mult in SW


Total run time for SW on Processor is 25880 cycles over 1024 tests.
Cache cleared
Total run time for AXI DMA + HW accelerator is 333830 cycles over 1024 tests
                                                                        Acce
leration factor: 0.77

                    SW and HW results match!

                                              **********************************
************************************************************
DMA Init done
            Loop time for 1024 iterations is -2 cycles
                                                    Running Matrix Mult in SW


Total run time for SW on Processor is 25880 cycles over 1024 tests.
Cache cleared
Total run time for AXI DMA + HW accelerator is 333830 cycles over 1024 tests
                                                                        Acce
leration factor: 0.77

                    SW and HW results match!
```

Lab08: Your Task

- **Task: Develop/optimize the floating point matrix multiplication with HLS**
  - **Basic:** Complete the step-by-step instructions
  - **Improve Parallelism**:

  You just need to modify the mmult_hw() in mmult.h (From Line 44)

  Step 1: Try to add pipeline directive to L1, L2 or L3

  Uncomment the "#pragma HLS PIPELINE II=1" to let the L1, L2 or L3 pipeline.

  Step 2: Read and analysis the synthesis report to find out what limits parallelism. (Check the "Loop" in the report to see whether the pipeline of the loop works well.)

  Set the "#pragma HLS PIPELINE II=1" to L2 is a relatively good choice.

  The ideal initiation interval should be 1, however there are some issues make the ideal initiation interval impractical. In the next step you will solve the bottlenecks.

Lab08: Your Task

Step 3: Add or modify the directives to further increase the parallelism. Parallelism means that multiple operations occur simultaneously, sometimes the bottleneck is the inability to read the data for calculation in time, sometimes the bottleneck is that the pipeline stops because of the in/out loop.

1) Uncomment the "`#pragma HLS array_partition variable=<ARRAY> block factor=<FACTOR> dim=<DIM>`" then set the factor and dim to let the array be partitioned.
2) Remove the "`#pragma HLS loop_flatten off`", which prevents the HLS tool from automatically merging outer loops of the "pipelining" into one larger loop.

- **Submission Rule:**
  - Submit your
    1) Revised mmult.h
    2) Screen shot of "Performance Estimates" for **each** HLS design (as shown in P.12)
    3) Screen shot of SDK verification for **each** HLS design (as shown in P.40)

    In total: 3 + 3 screen shots
  - Deadline: 12:30 on 26 March 2025 (Late submission is NOT acceptable)

Hints: Workflow (p.3)

- You just need to modify the mmult_hw() in mmult.h (From Line 44)
- Run C Synthesis
- Until you got a decent result in the performance report, then you export RTL, move on to the block design and SDK verification steps

- How to read the C Synthesis report?

Hints: Performance of the solution

- How to read the C Synthesis report?

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.41	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
332873	332873	332874	332874	none

Detail

Instance

Loop

This is the latency estimation (Performance of the program)
Lower would be better

Utilization Estimates

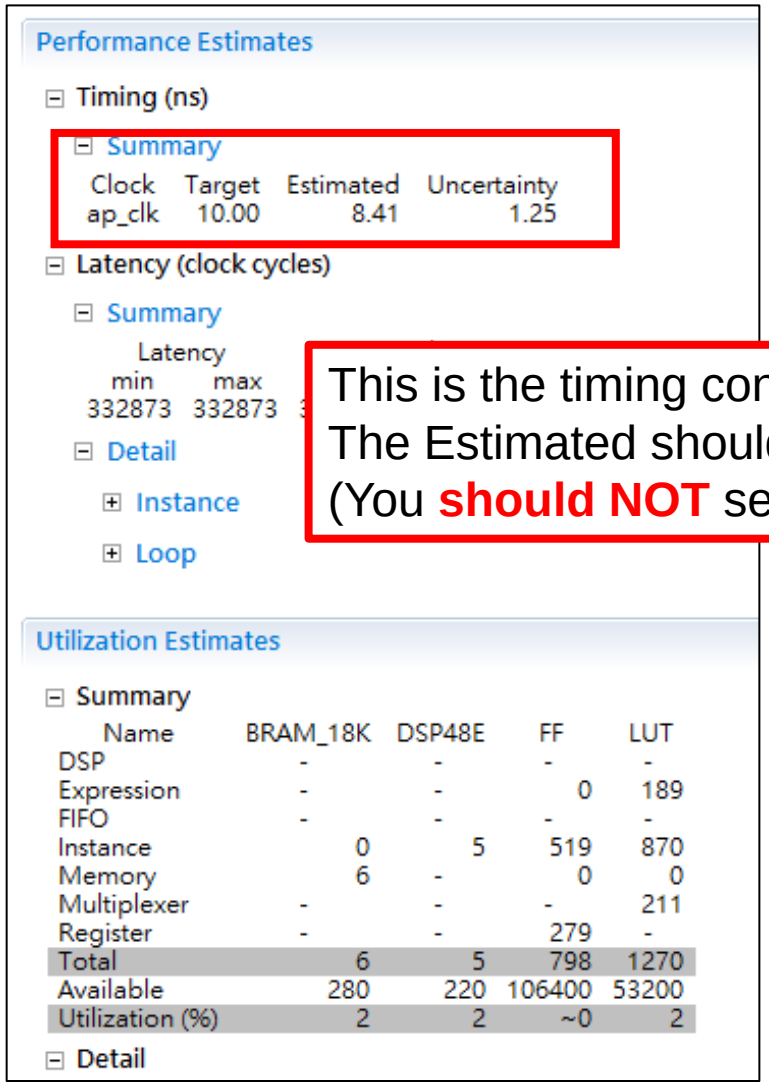
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	189
FIFO	-	-	-	-
Instance	0	5	519	870
Memory	6	-	0	0
Multiplexer	-	-	-	211
Register	-	-	279	-
Total	6	5	798	1270
Available	280	220	106400	53200
Utilization (%)	2	2	~0	2

Detail

Hints: Comply with time constraints

- How to read the C Synthesis report?



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.41	1.25

Latency (clock cycles)

Summary

Latency	
min	max
332873	332873

Detail

Instance

Loop

Utilization Estimates

Summary

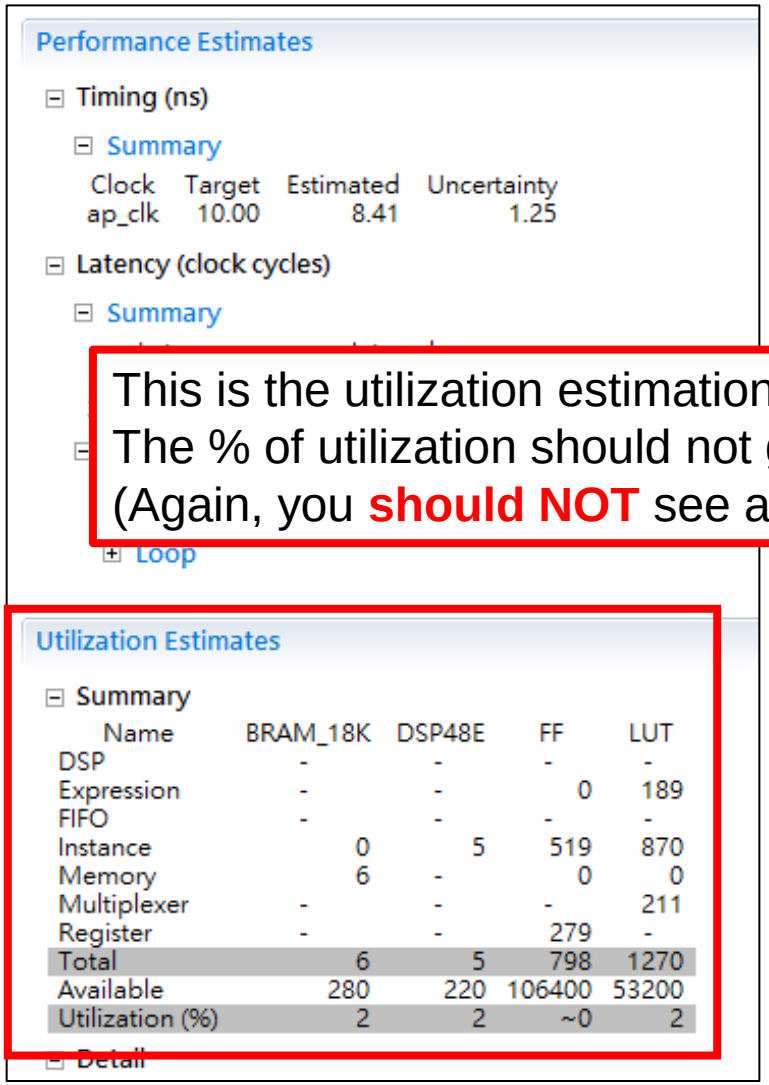
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	189
FIFO	-	-	-	-
Instance	0	5	519	870
Memory	6	-	0	0
Multiplexer	-	-	-	211
Register	-	-	279	-
Total	6	5	798	1270
Available	280	220	106400	53200
Utilization (%)	2	2	~0	2

Detail

This is the timing constraint estimation
The Estimated should not exceed the Target
(You **should NOT** see any **red** value at here)

Hints: Use resources wisely

- How to read the C Synthesis report?



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.41	1.25

Latency (clock cycles)

Summary

Loop

This is the utilization estimation
The % of utilization should not greater than 100
(Again, you **should NOT** see any **red** value at here)

Utilization Estimates

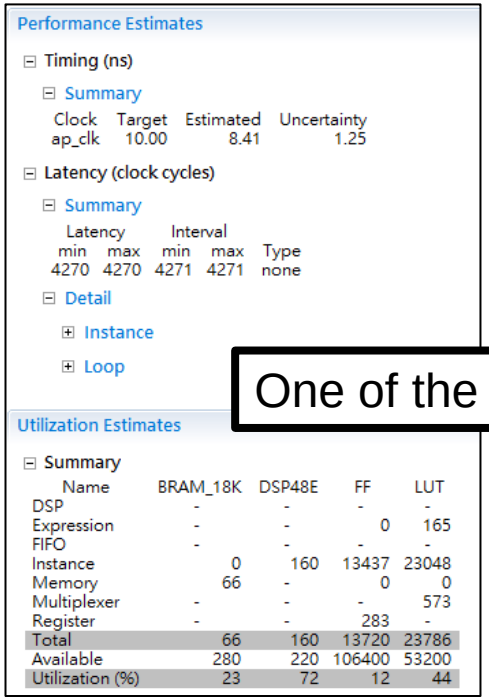
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	189
FIFO	-	-	-	-
Instance	0	5	519	870
Memory	6	-	0	0
Multiplexer	-	-	-	211
Register	-	-	279	-
Total	6	5	798	1270
Available	280	220	106400	53200
Utilization (%)	2	2	~0	2

Detail

Hints: Final performance

- For Loop Optimization, you need to choose the which loop do you want to optimize? which pragma do you use?
- For Array Partition Optimization, you need to decide the parameters, such as dimension and factor.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.41	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
4270	4270	4271	4271	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	165
FIFO	-	-	-	-
Instance	0	160	13437	23048
Memory	66	-	0	0
Multiplexer	-	-	-	573
Register	-	-	283	-
Total	66	160	13720	23786
Available	280	220	106400	53200
Utilization (%)	23	72	12	44

One of the result that we could achieve

香港中文大學
The Chinese University of Hong Kong

Thank you