
PUFacademy
A PUFsecurity Alliance

Digital Logic Design
- Lecture 7
- FPGA .

2025 Spring

PUFacademy

A PUFsecurity Alliance

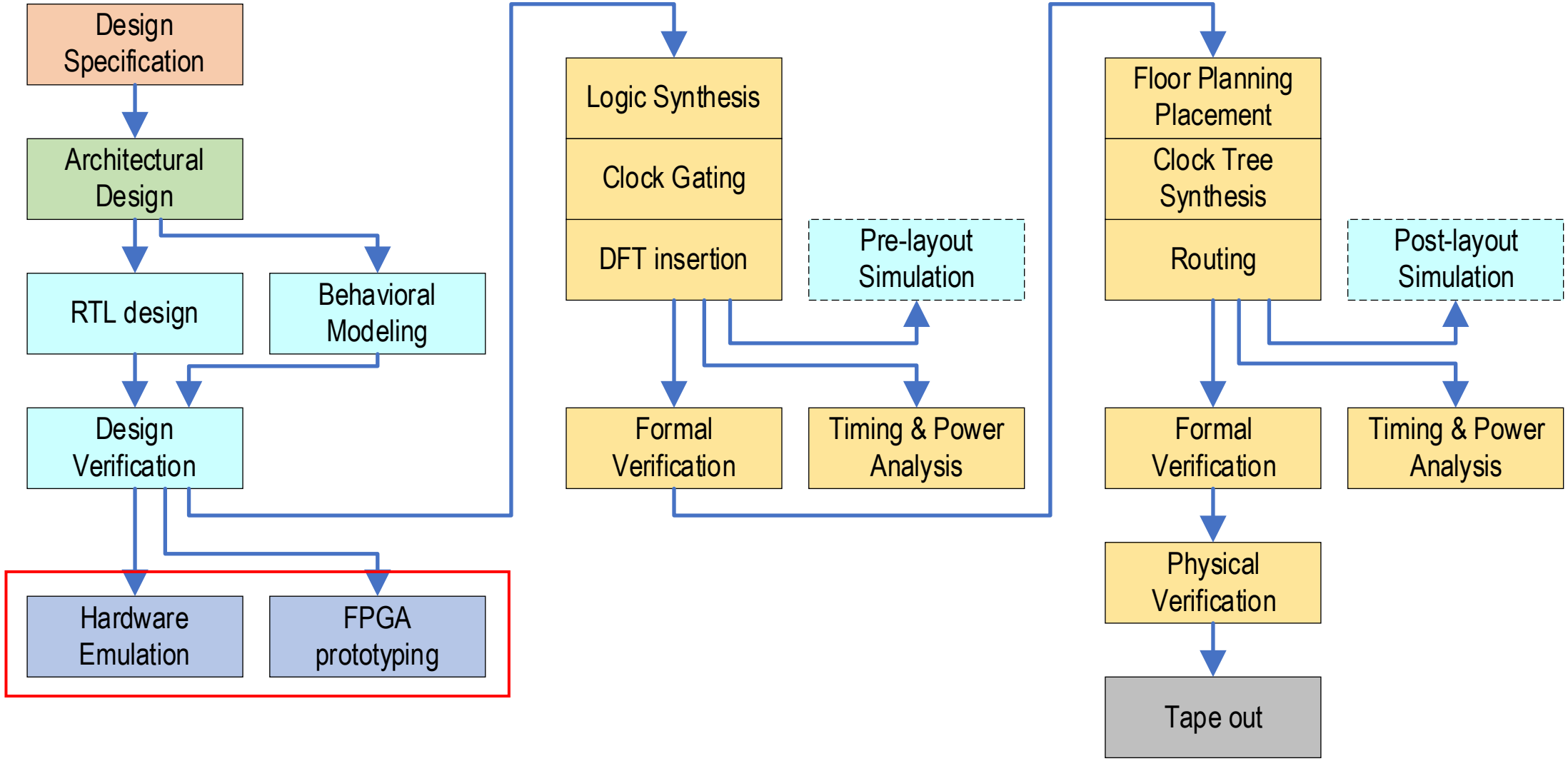
Agenda

1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

Agenda

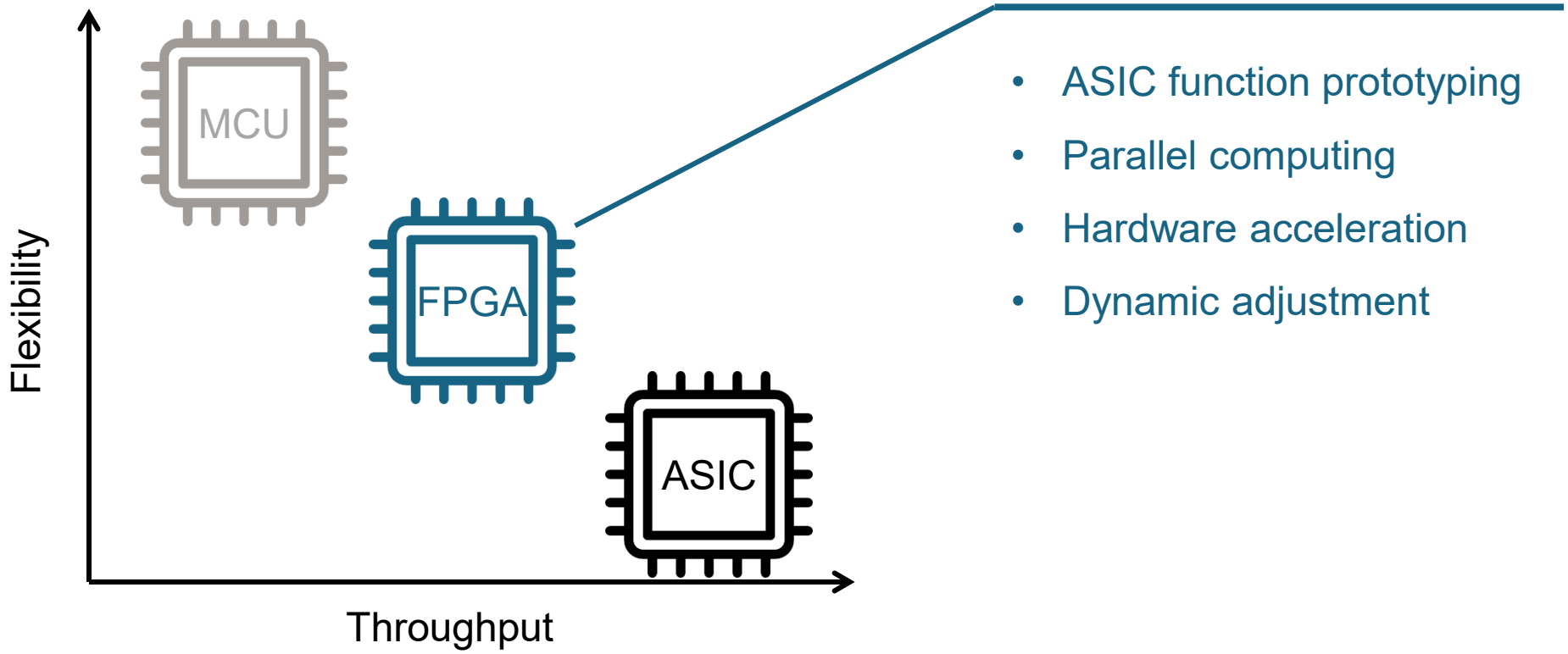
1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

From Code to Chip



Why We Need FPGA?

- Originally, for ASIC function prototyping
- The flexibility of FPGA make it proper for many commercial application

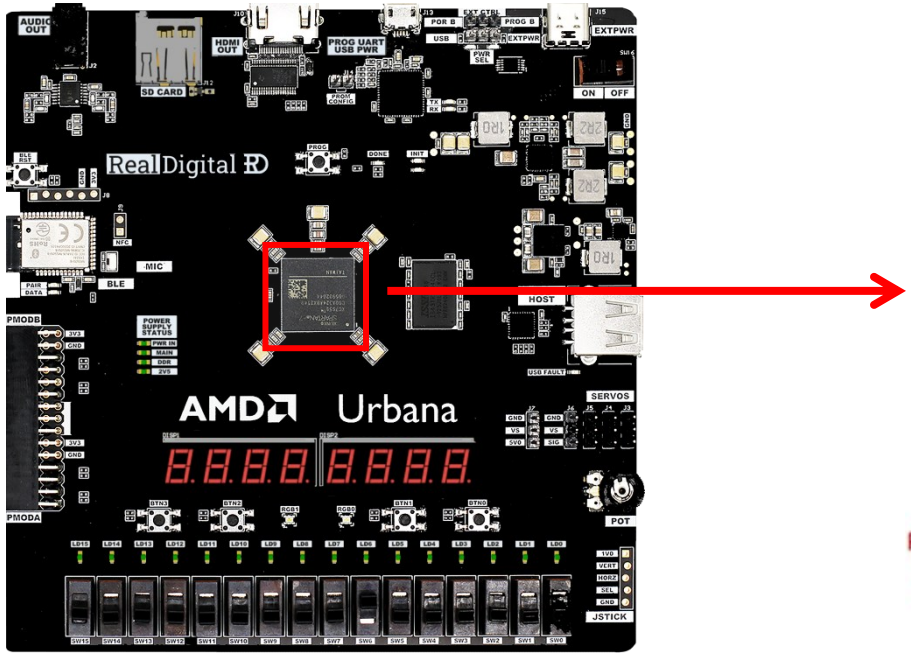


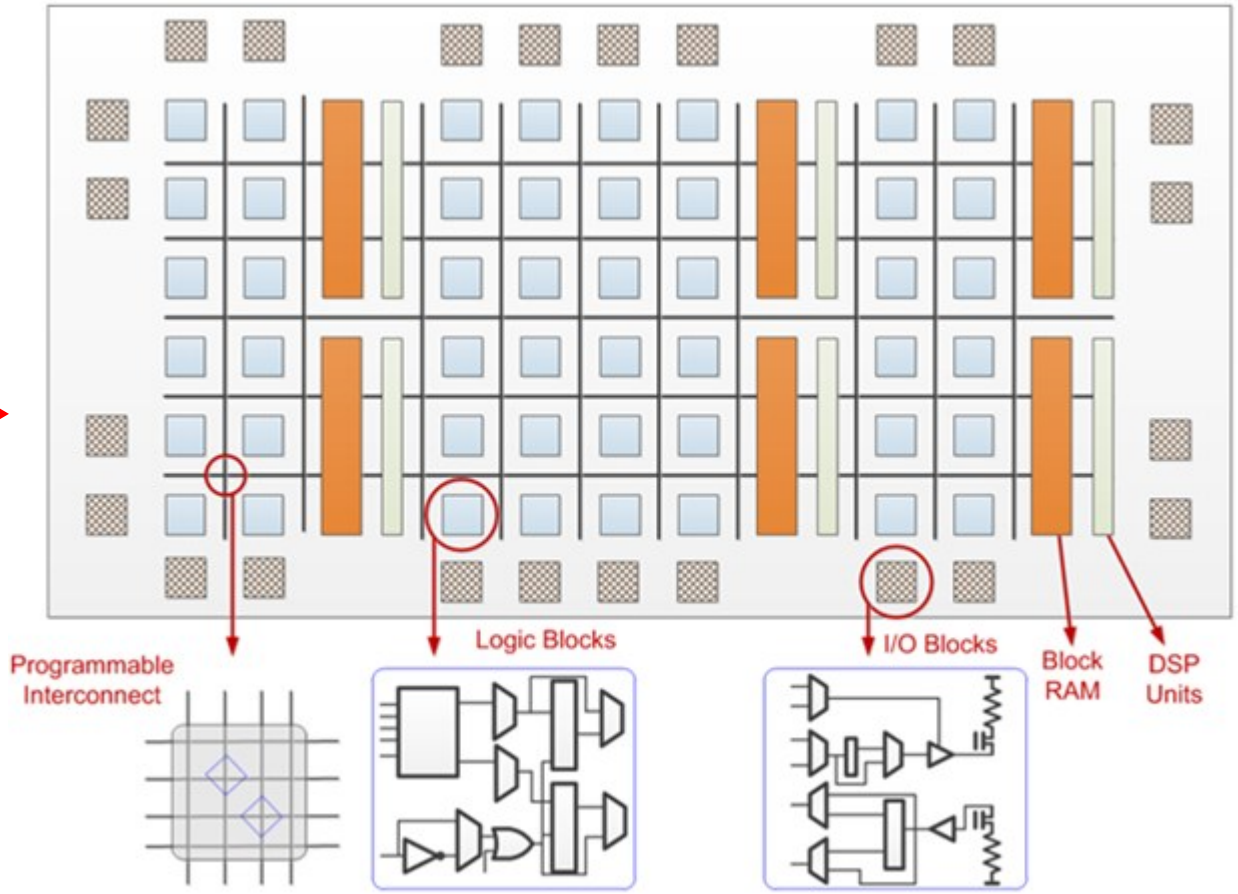
Agenda

1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

What is an FPGA?

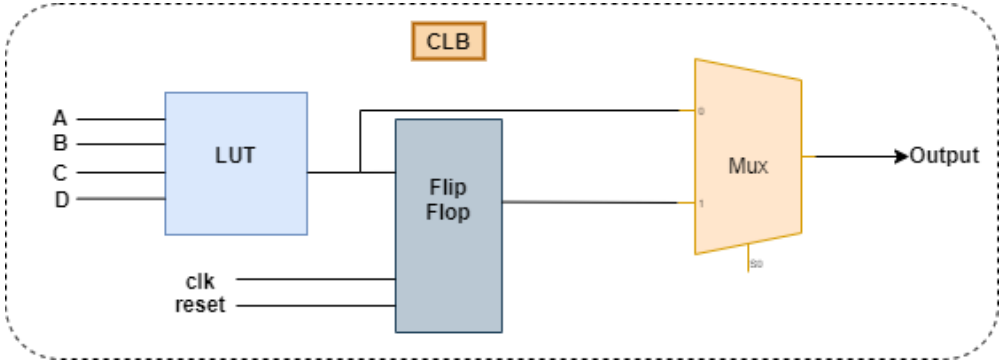
- FPGA: Field Programmable Gate Array
  - Gate Array (composed of configurable logic blocks)
  - Programmable

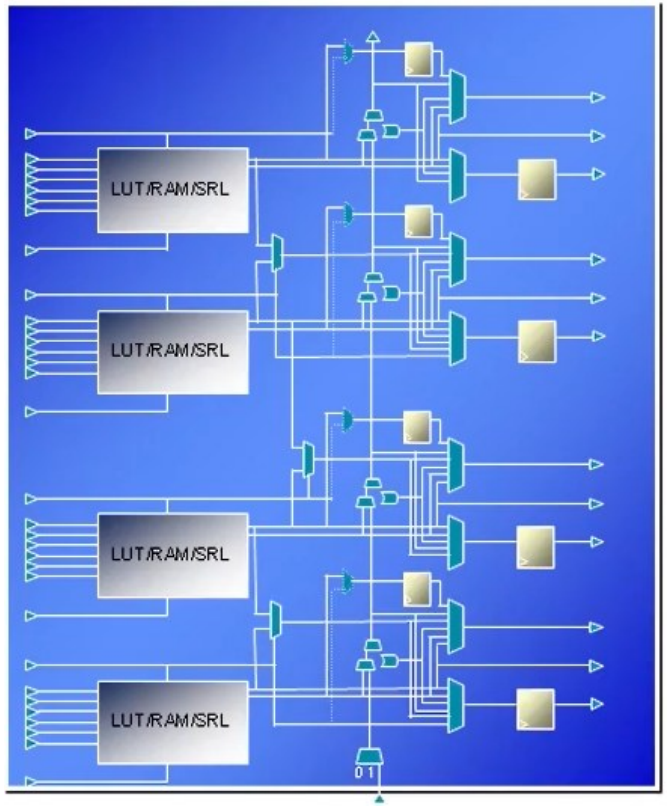




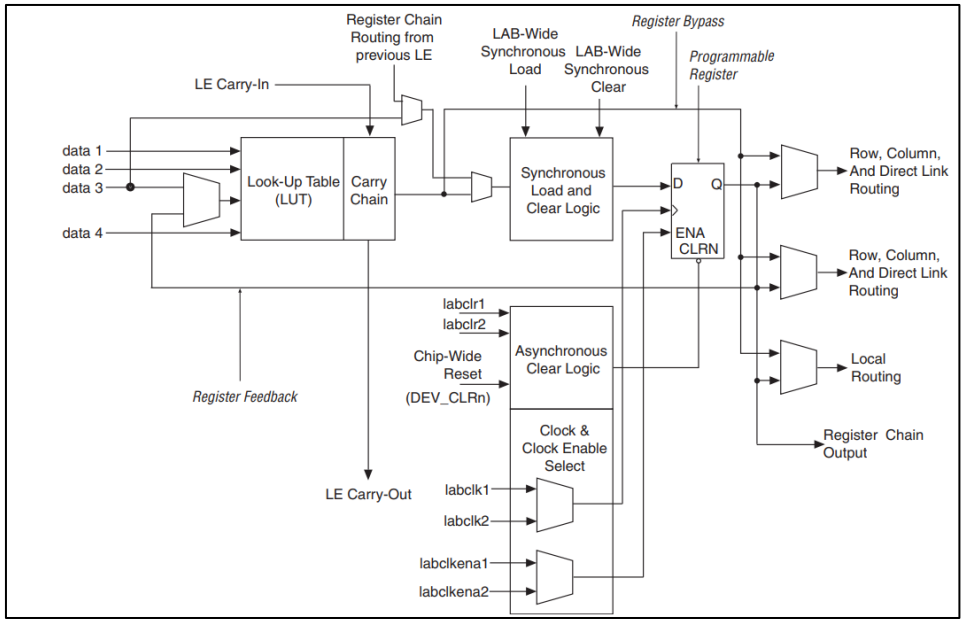
Configurable Logic Blocks

- Synonyms: Slice (Xilinx, AMD), Logic Element (Altera, Intel)
- Contain look-up tables (LUTs), flip-flops, multiplexers





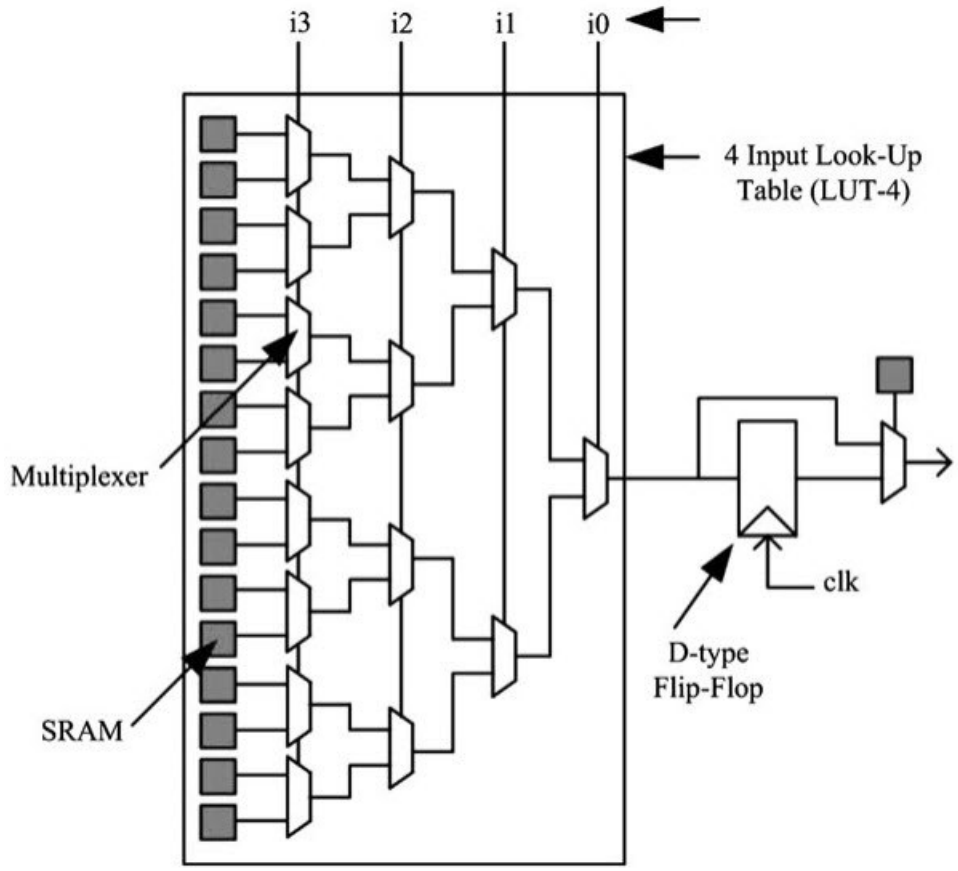
Xilinx 7-Series Slice Architecture



Altera Cyclone IV logic element

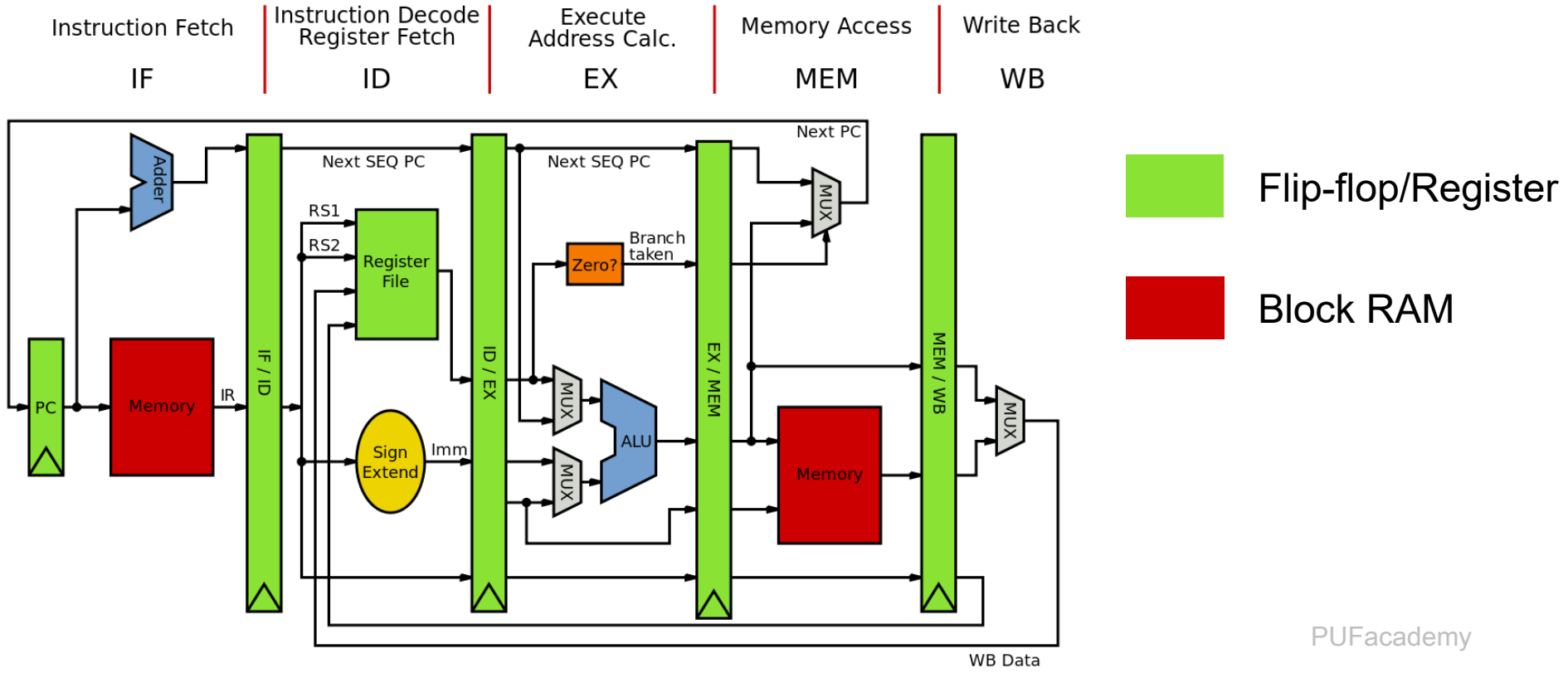
Look-up Table

- Constituted by SRAMs and multiplexers
  - SRAM: hold the output corresponding to input i3~i0
  - Multiplexer: controlled by i3~i0 to propagate output
- For different purposes, SRAMs may be replaced by Flash or anti-fuse



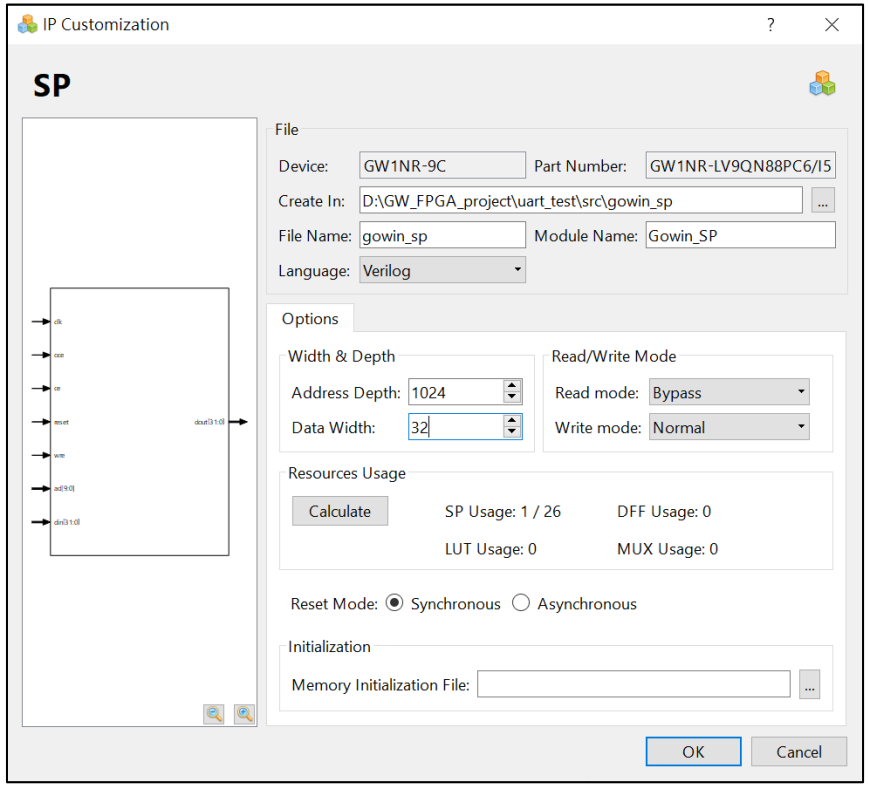
Block RAM (1/2)

- Memory unit in FPGA
- Can be utilized as SRAM, ROM, FIFO capacity
- Instance method: Block RAM generator GUI in EDA tool, or inferred RAM



Block RAM (2/2)

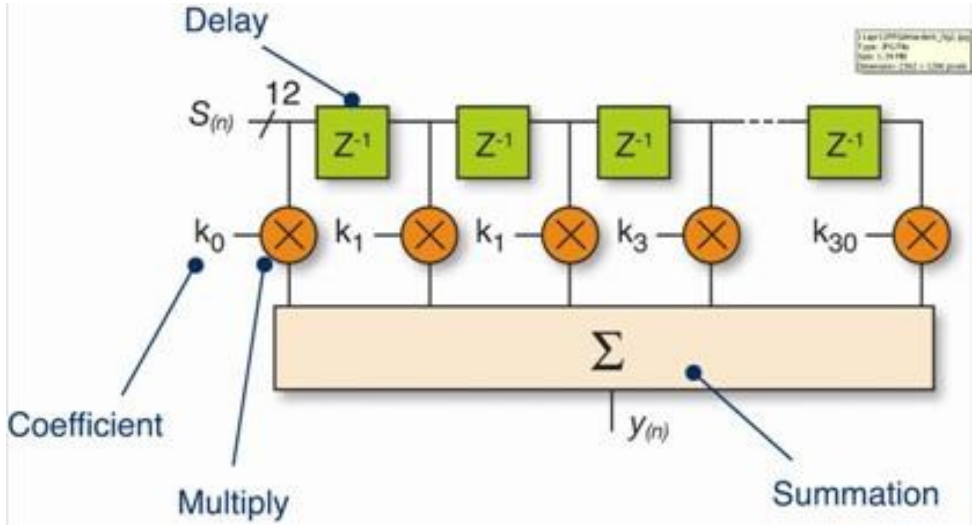
- Memory unit in FPGA
- Can be utilized as SRAM, ROM, FIFO capacity
- Instance method: Block RAM generator GUI in EDA tool, or inferred RAM(not all EDA tools support)

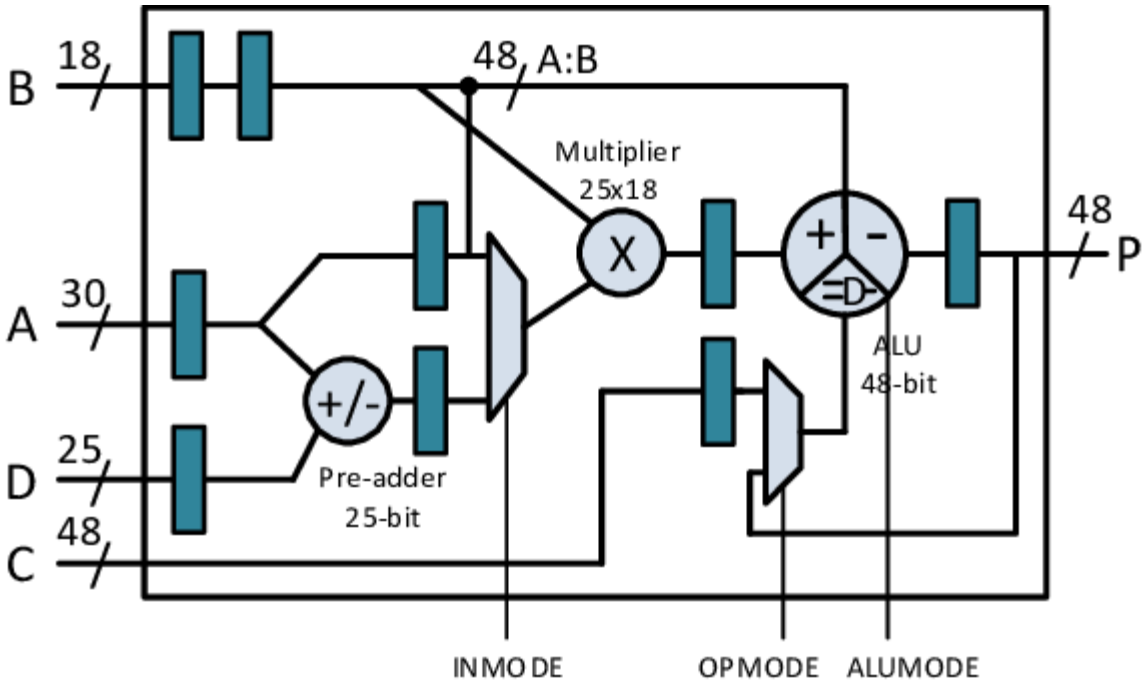



```
module inferred_ram_single_port#(
parameter DATA_DEPTH = 1024,
parameter DATA_WIDTH = 32,
parameter ADDR_WIDTH = clog2(DATA_DEPTH),
parameter DAT_FILE   = "BRAM_DATA.dat"
)(
input                          clka,
input                          ena,
input                          wea,
input      [ADDR_WIDTH  -1:0] addra,
input      [DATA_WIDTH  -1:0] dina,
output reg [DATA_WIDTH  -1:0] douta
);
reg        [DATA_WIDTH  -1:0] RAM [DATA_DEPTH-1:0];

initial begin
   $readmemh(DAT_FILE,RAM);
end

always@(posedge clka) begin
   if(ena) begin
      if(wea) RAM[addra] <= dina;
      else    douta      <= RAM[addra];
   end
end

function integer clog2;
   input integer value;
   begin
      value = value - 1;
      for(clog2=0; value>0; clog2=clog2+1) begin
         value = value >> 1;
      end
   end
endfunction
endmodule
```

Digital Signal Processing Unit

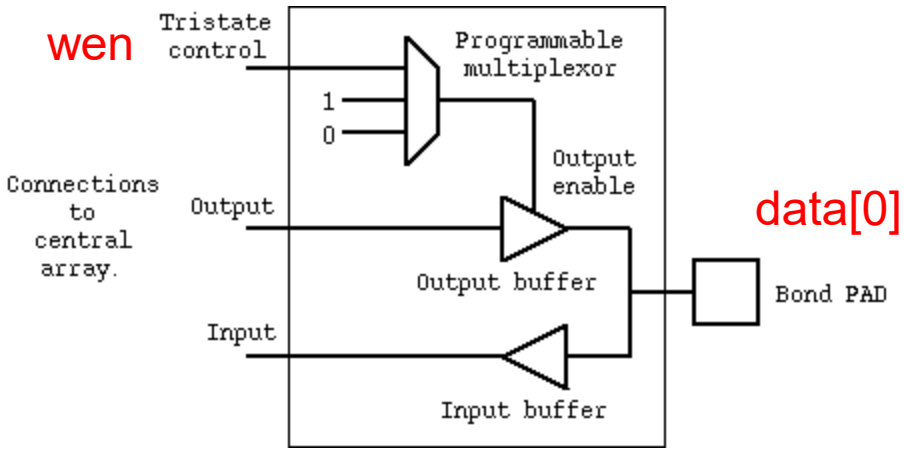
- The DSP units are usually used for signal processing or multiply-accumulate
- EDA tool may optimize your design with DSP units as it detects multiply-accumulate behavior
- The DSP optimization can be disable by Tcl command





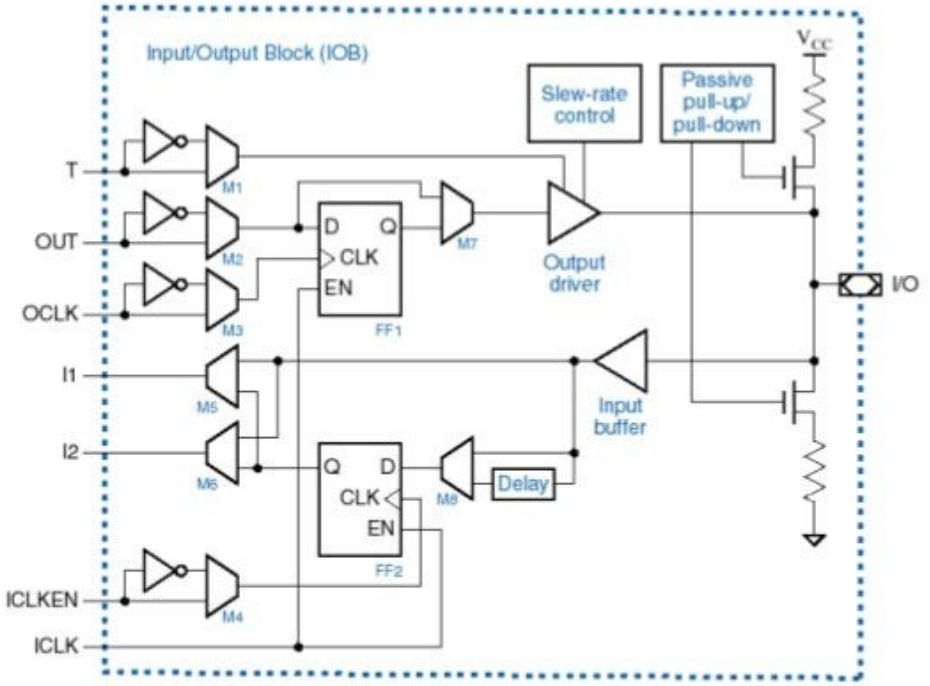
I/O Block

- The I/O Blocks on FPGA are configurable (via constraints) to fit different I/O requirement
  - Input, output, inout (tristate control)
  - Voltage level
  - Driving ability (Slew-rate)
  - Pull-up/pull-down resistor




```
input          clk,
output         wen,
inout  [7:0]   data
```

I/O Blocks



Reference

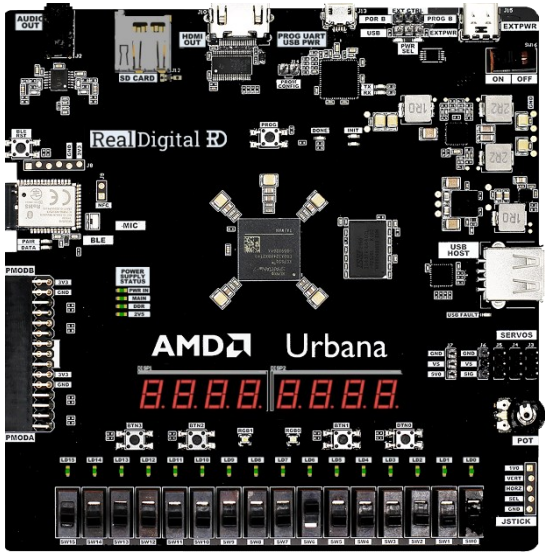
- AMD What is an FPGA: https://www.youtube.com/watch?v=2maAkrQzCkk
- AMD Xilinx 7-Series CLB Architcture: https://www.xilinx.com/video/fpga/7-series-clb-architecture.html
- AMD Xilinx 7-Series DSP resources: https://www.xilinx.com/video/fpga/7-series-dsp-resources.html

Agenda

1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

Urbana FPGA Board

- For Lab03, the Urbana FPGA and Xilinx electronic design automation (EDA) tool are introduced
  - FPGA board: Urbana board
    - https://www.realdigital.org/doc/496fed57c6b275735fe24c85de5718c2#setting-up-the-urbana-board
  - EDA: Vivado, Software Development Kit (SDK)



Urbana board	
FPGA device	Xilinx Spartan-7 XC7S50-C SGA324
LUT element	32600
Registers(Flip-Flop)	65200
Crystal oscillator	100MHz
Debugger	Onboard USB-JTAG & USB-UART

EDA Installation

- EDA: Vivado 2017.4 & SDK 2017.4
  - Please follow “113 Digital Logic Design - Lecture 7.1 - Download & installation of Vivado”

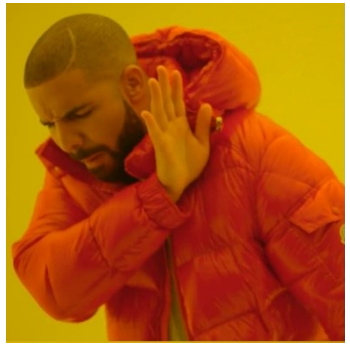
Vivado & SDK

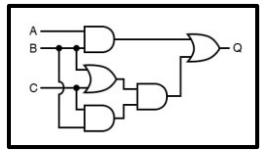
- Build an environment fully based on your RTL design
- Build an environment with CPU(harden/soft core) and your RTL design

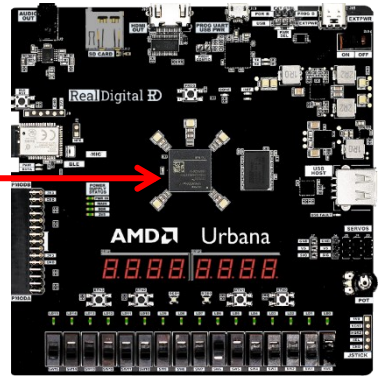
- Run application on your environment (an CPU is required)

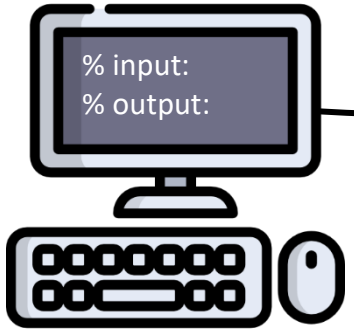
Design Verification and Demonstration with FPGA

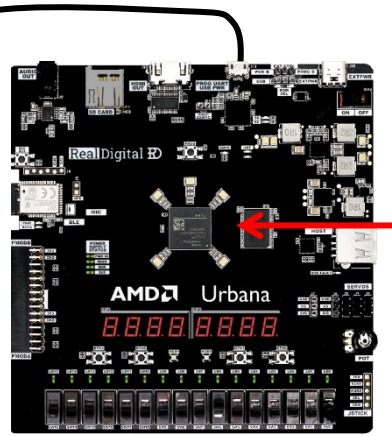
- “A demonstration is more than worth a thousand words”
- A user-friendly interface for verifying/demonstrating your design is usually essential

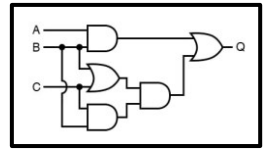




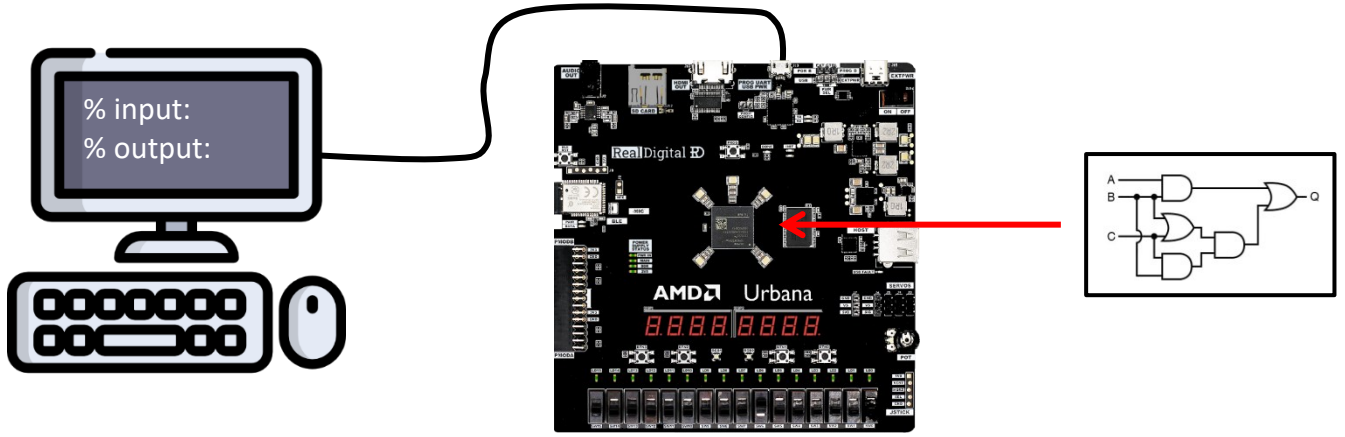


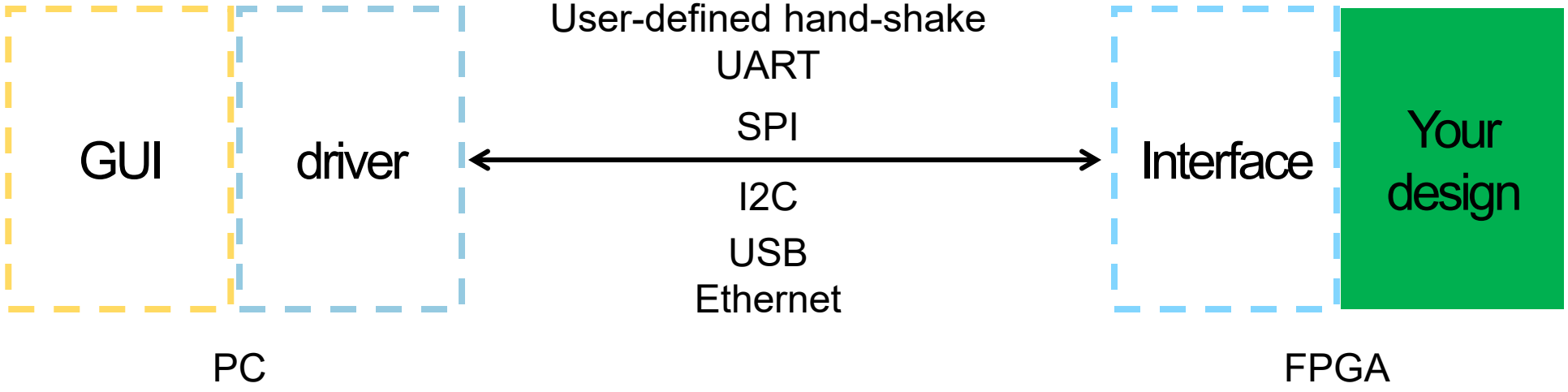




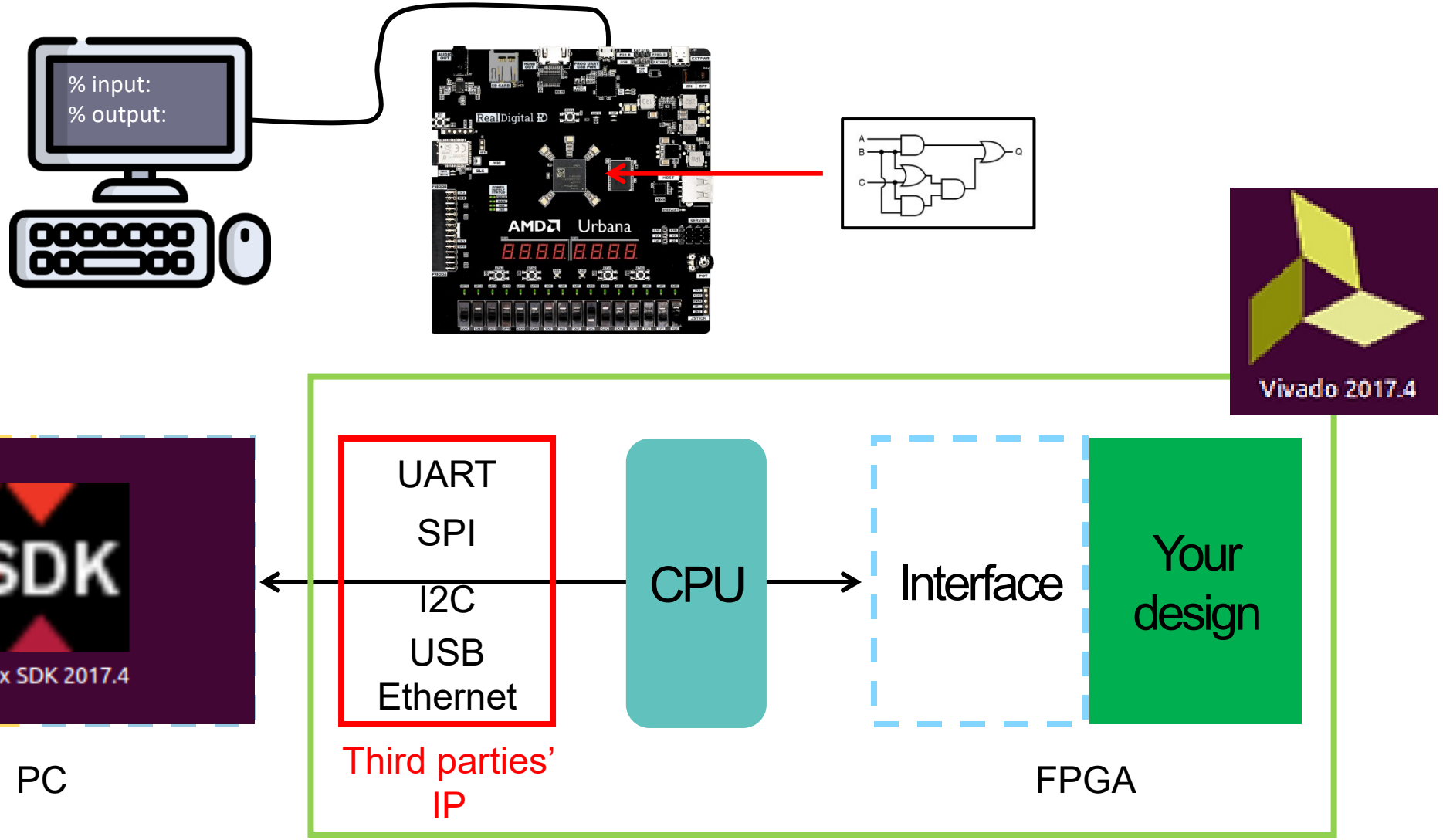


What do You Need for Demonstration?



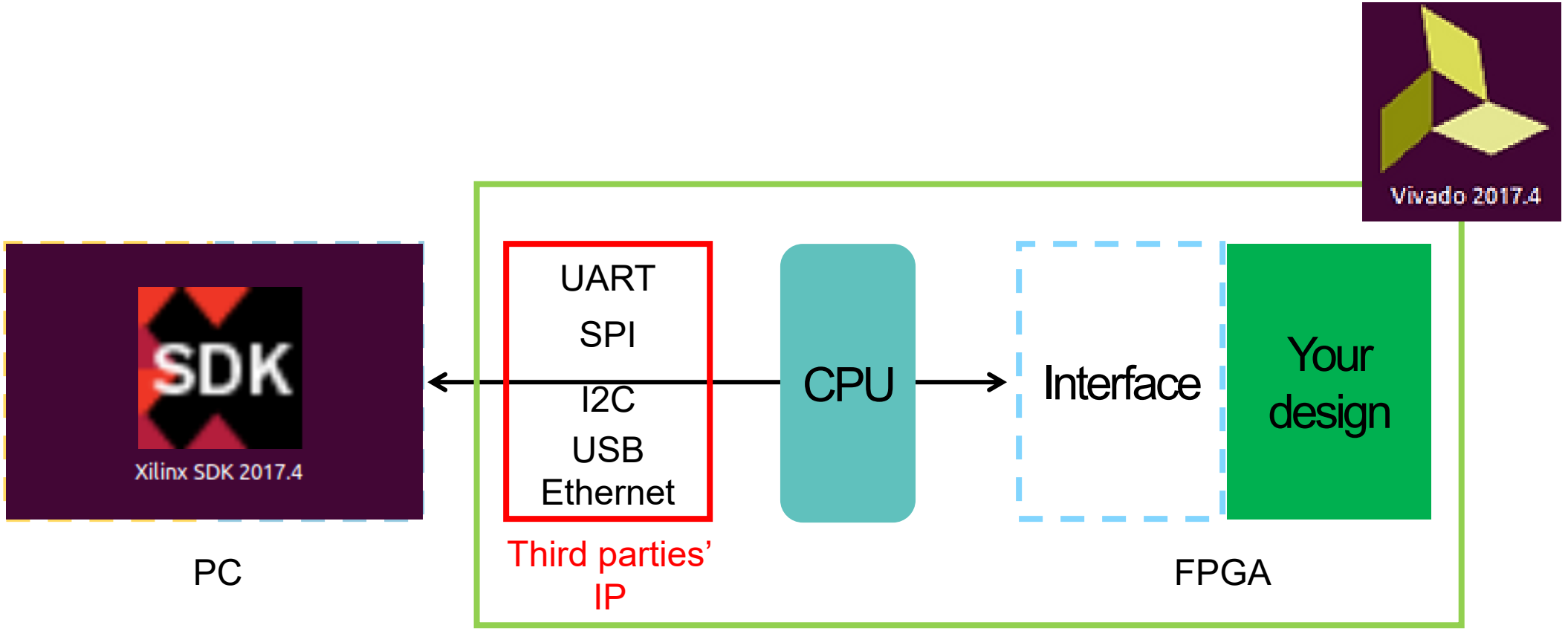


What do You Need for Demonstration?



Build up These Blocks for Time-to-Market

- In this lab, the CPU environment and SDK help us to build a demonstration GUI efficiently
- We just need to pack our design into a AXI compatible IP

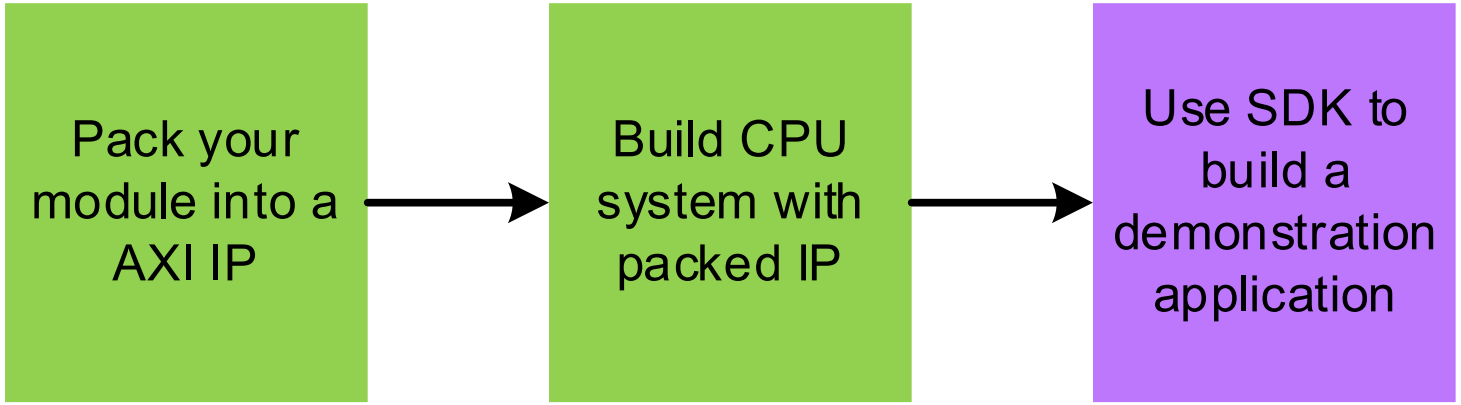


Agenda

1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

Practice: Create an AXI IP and Access It on SDK

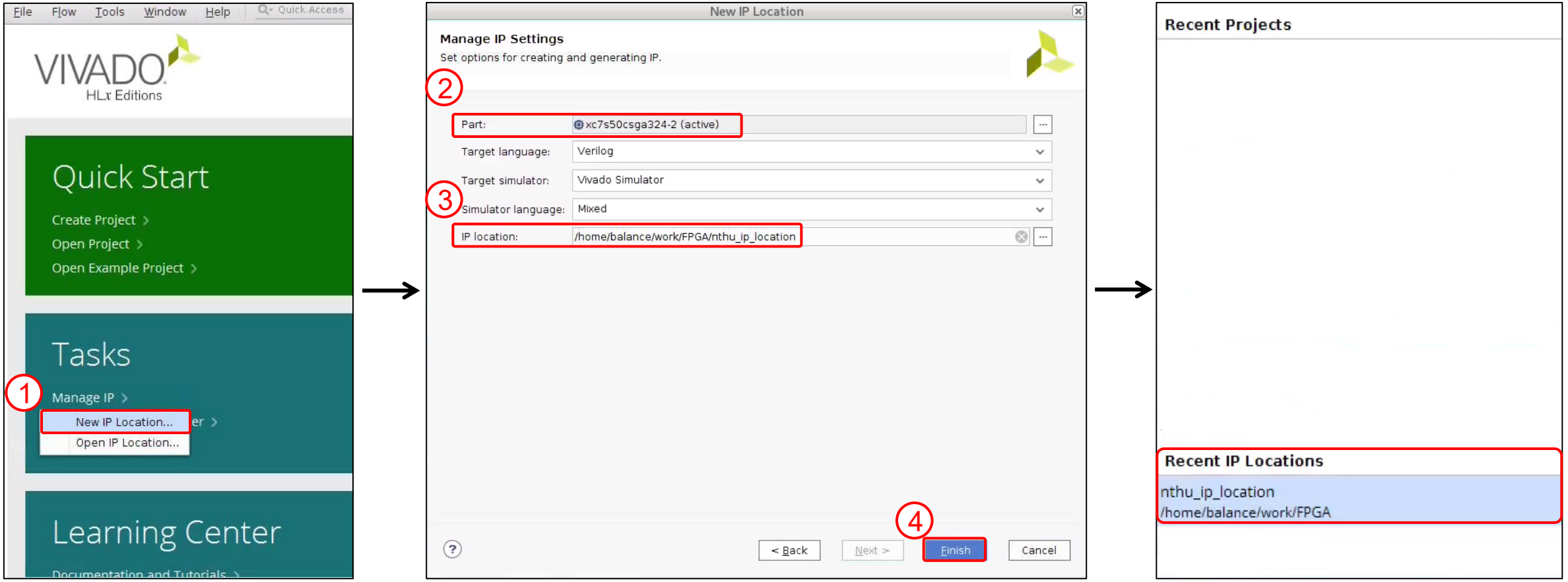
- This practice is used to make you familiar with the Urbana FPGA board and the development flow on EDA tool



Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

New IP Location for IP Management

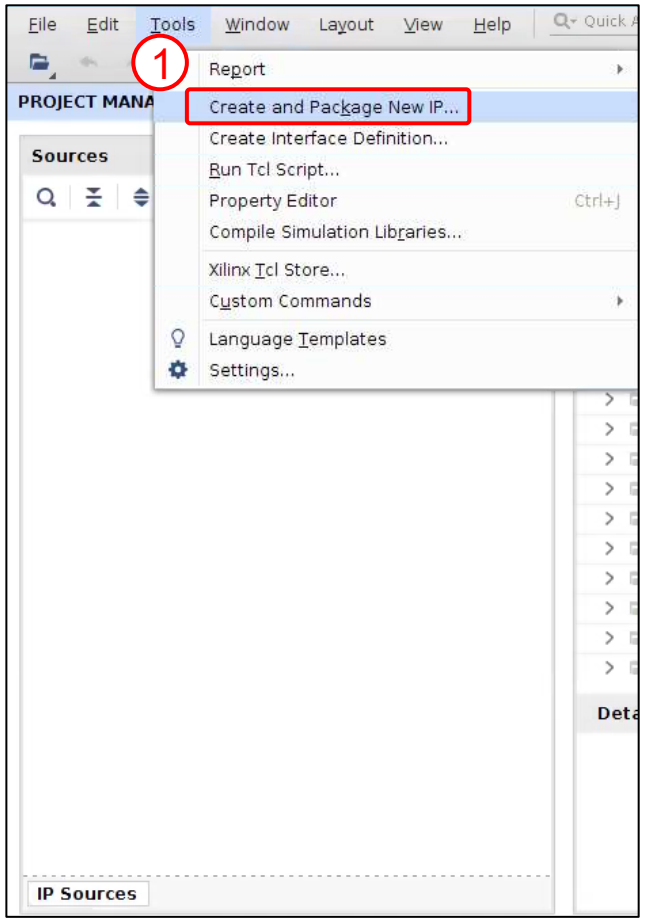
- Manage IP -> New IP Location -> Next -> Set “Part” as “xc7s50csga324-2” -> select IP location -> Finish

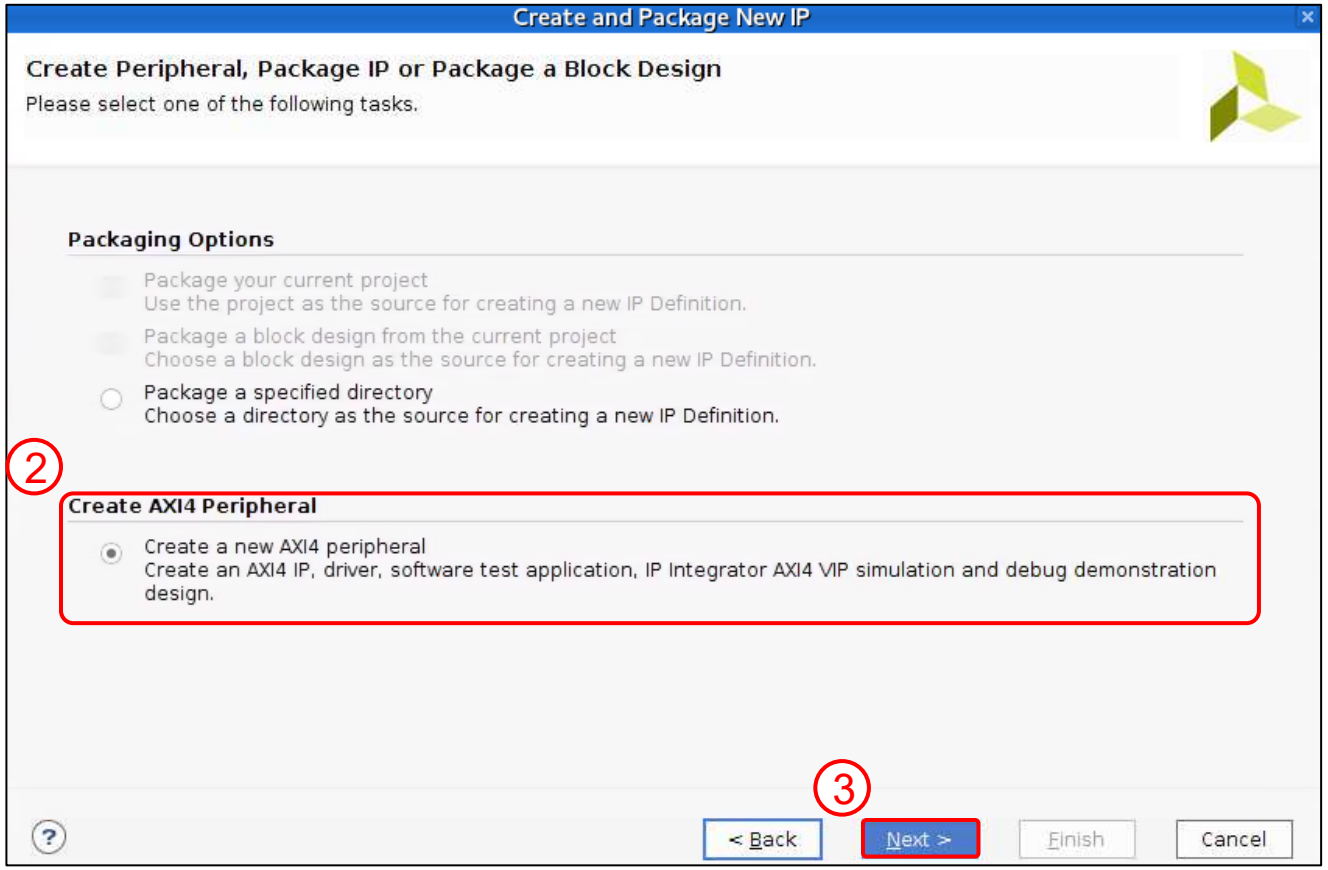


Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Create an AXI IP (1/3)

- Tools -> Create and Package New IP… -> Next -> Create a new AXI4 peripheral -> Next…

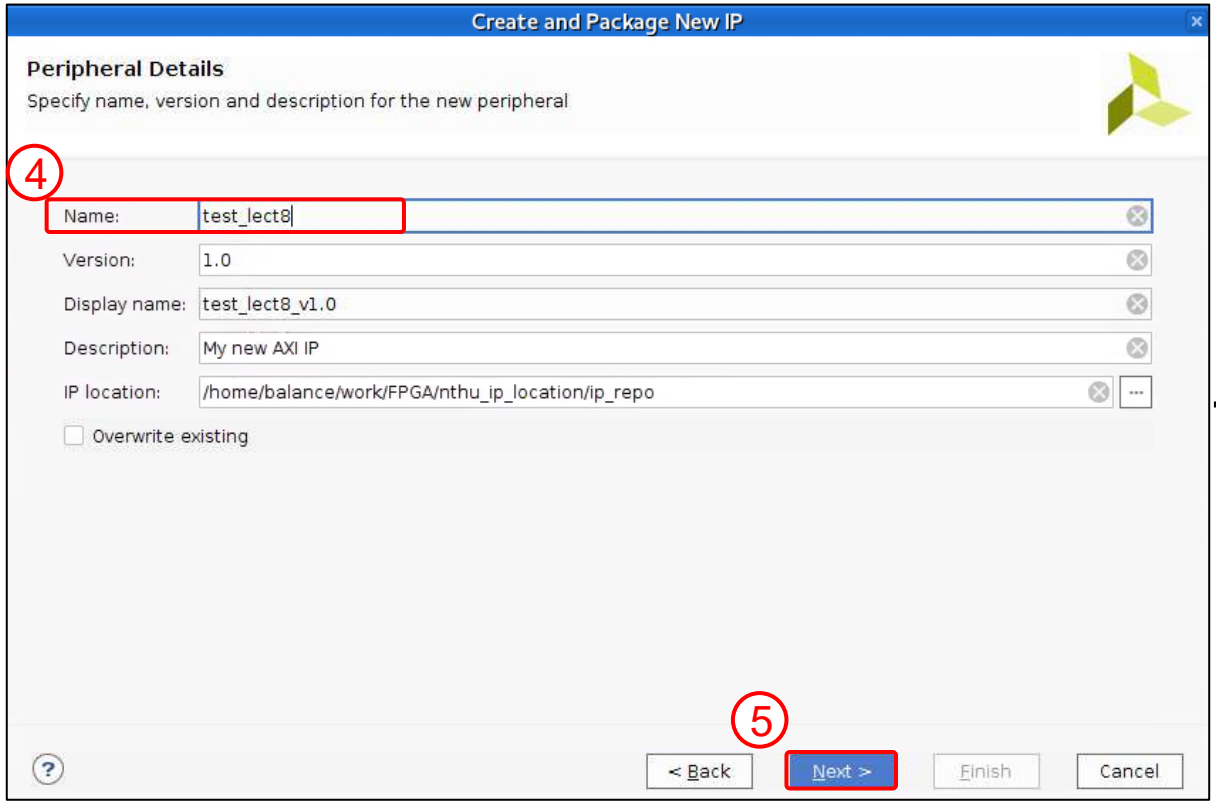


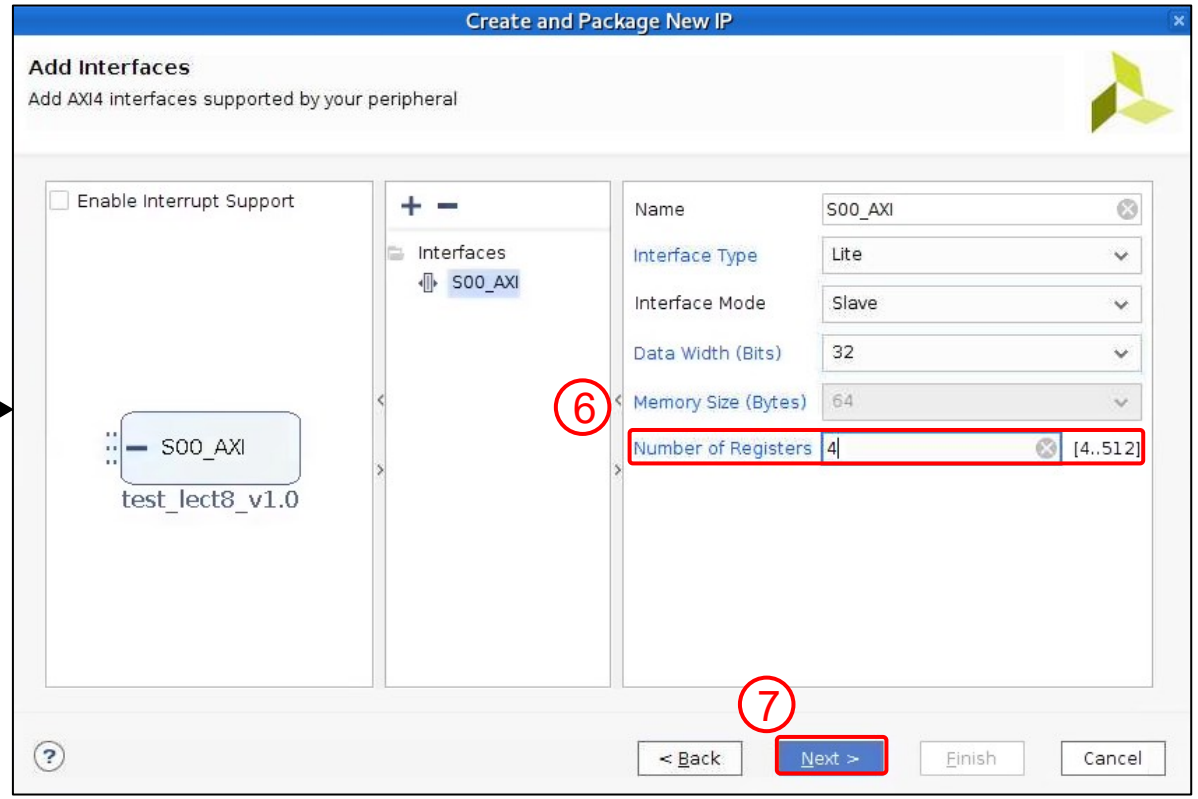


Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Create an AXI IP (2/3)

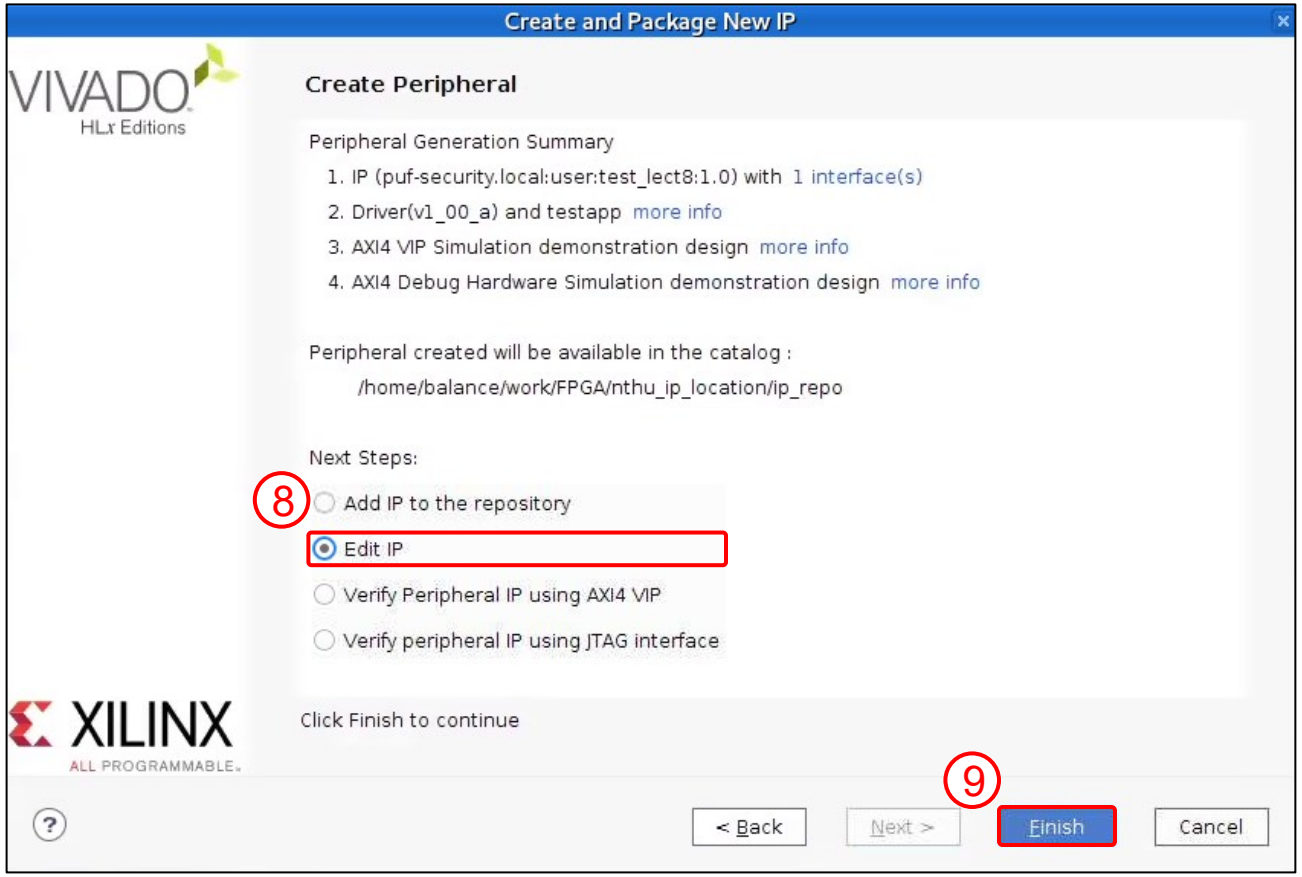
- Modify your IP name -> Next -> Modify “Number of Registers (Lab03)” -> Next





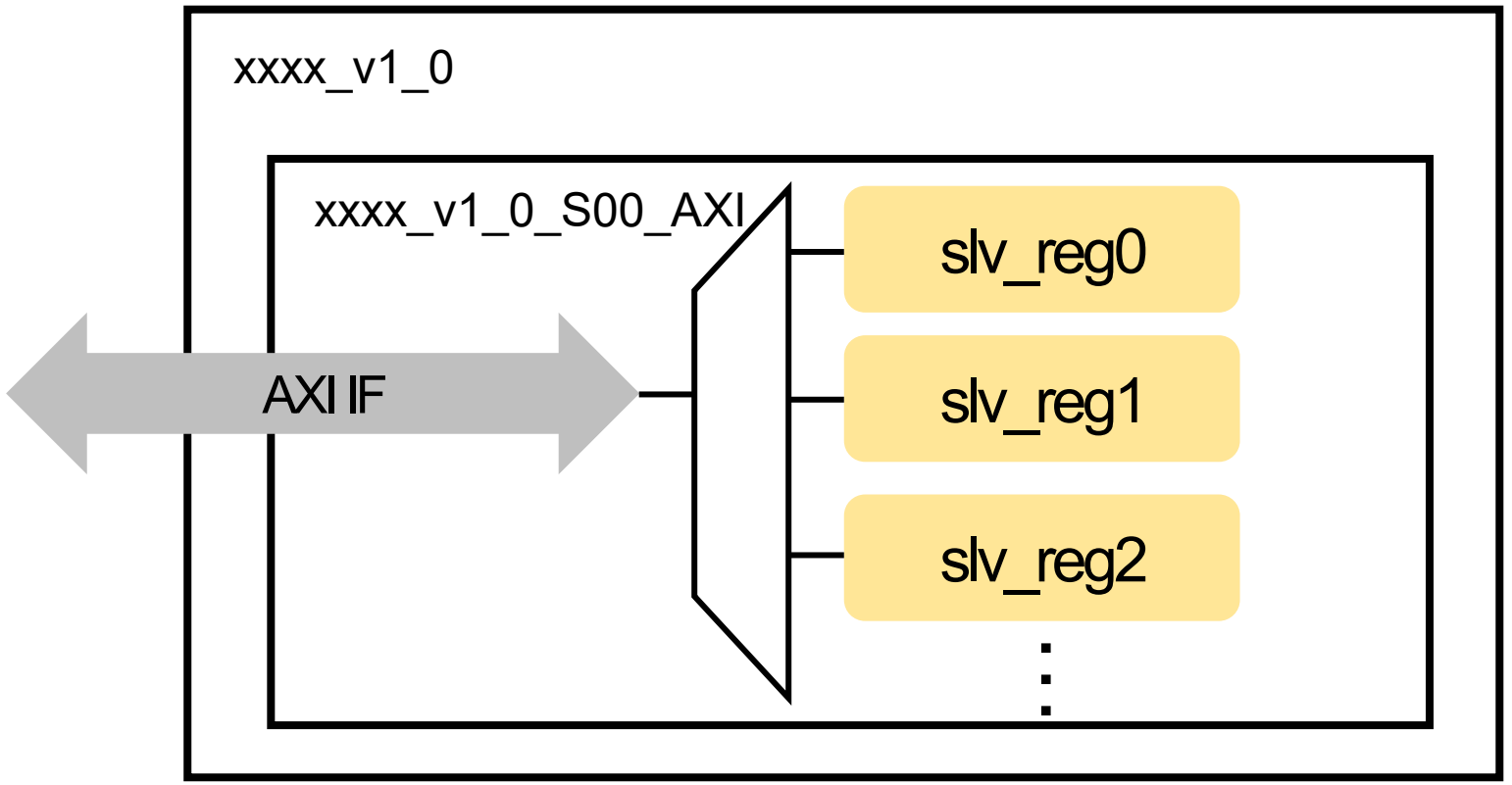
Create an AXI IP (3/3)

- Select “Edit IP” -> Finish



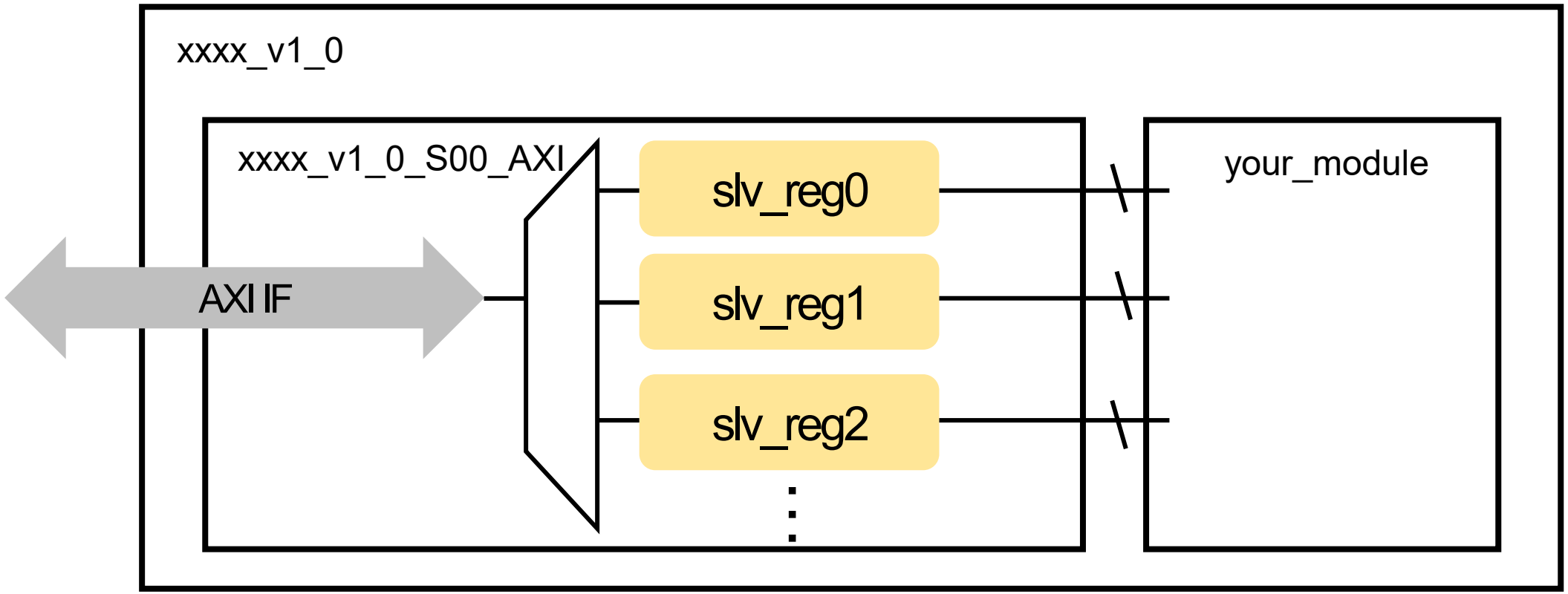
Structure of AXI IP Template

- AXI bus reference:
  - https://developer.arm.com/documentation/ihi0022/latest/



Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Add Your Module into the AXI IP Template



Practice: Modify the AXI IP Template

- Force the output to 0x2025_0516 as a READ request to the “0x4” address of this IP

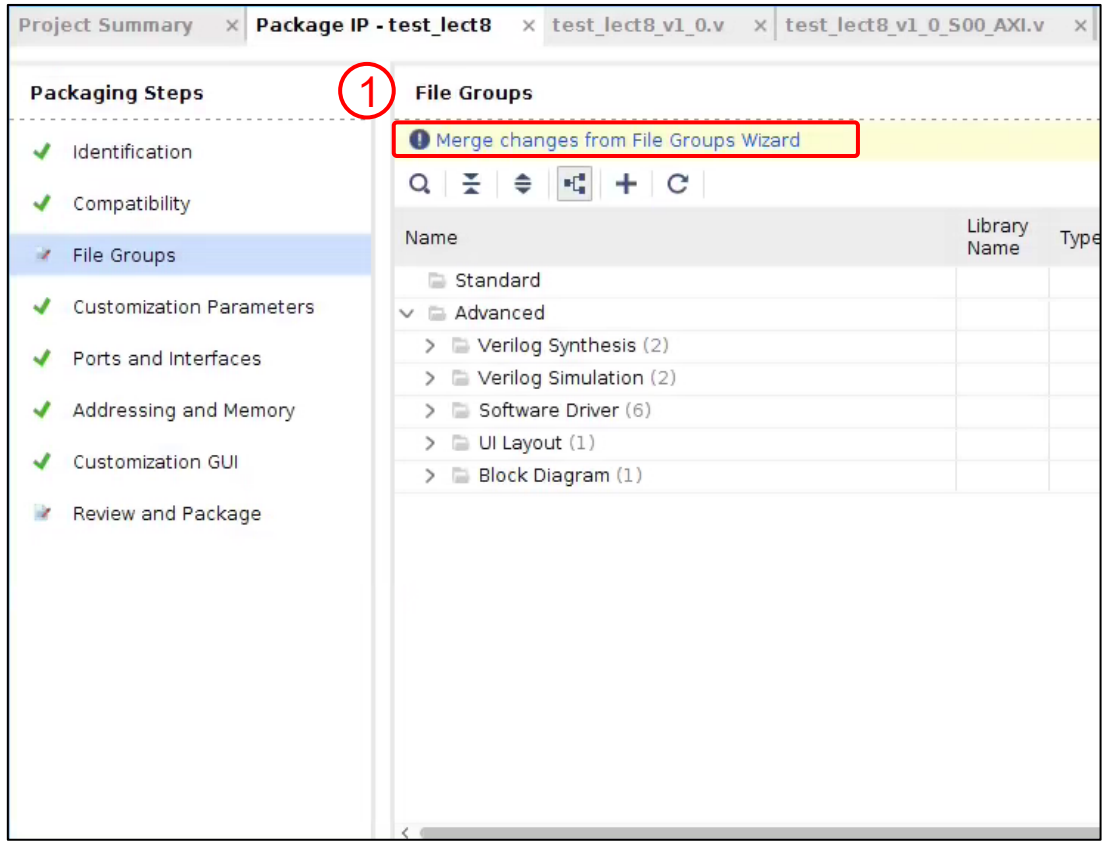
```
367  // Implement memory mapped register select and read logic generation
368  // Slave register read enable is asserted when valid address is available
369  // and the slave is ready to accept the read address.
370  assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
371  always @(*)
372  begin
373        // Address decoding for reading registers
374        case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
375          2'h0   : reg_data_out <= slv_reg0;
376          2'h1   : reg_data_out <= slv_reg1;
377          2'h2   : reg_data_out <= slv_reg2;
378          2'h3   : reg_data_out <= slv_reg3;
379          default : reg_data_out <= 0;
380        endcase
381  end
```

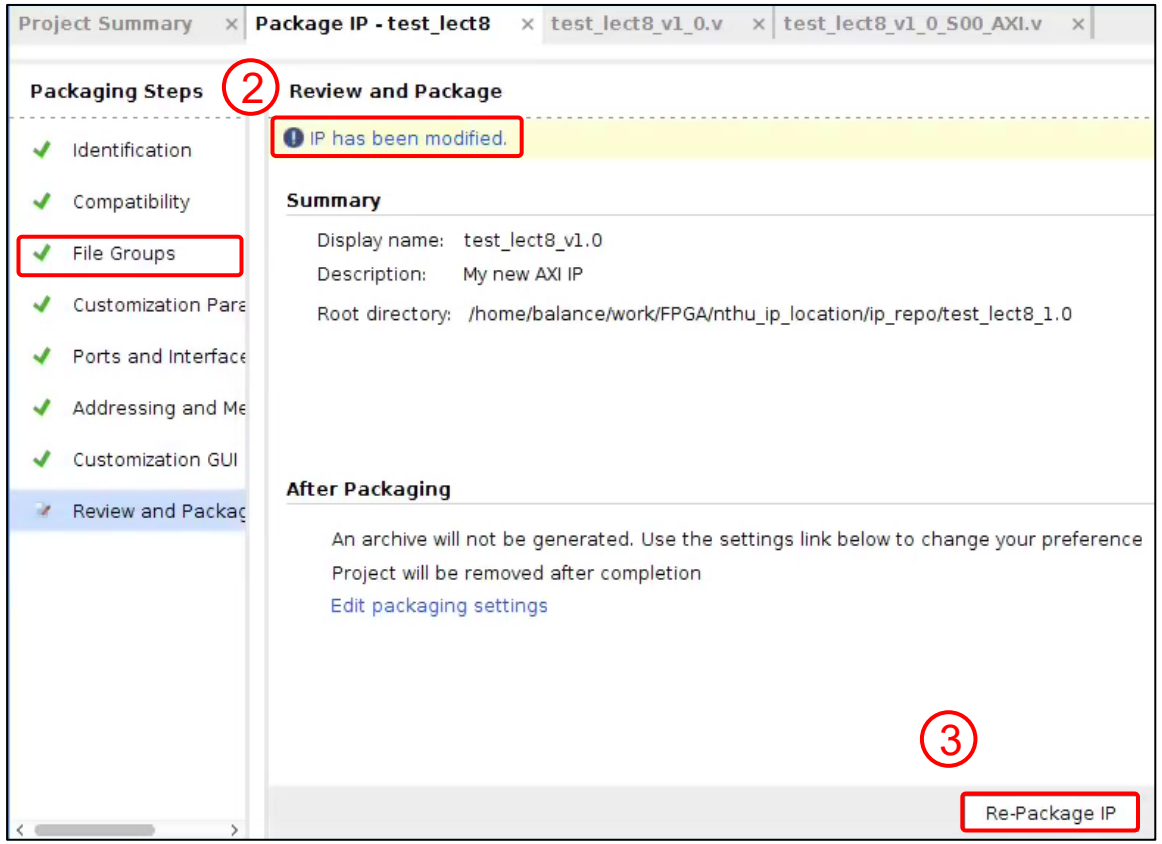
```
// Implement memory mapped register select and read logic generation
// Slave register read enable is asserted when valid address is availableble
// and the slave is ready to accept the read address.
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
      // Address decoding for reading registers
      case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
        2'h0   : reg_data_out <= slv_reg0;
        2'h1   : reg_data_out <= 32'h2025_0516;
        2'h2   : reg_data_out <= slv_reg2;
        2'h3   : reg_data_out <= slv_reg3;
        default : reg_data_out <= 0;
      endcase
end
```

Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Re-pack the modified AXI IP

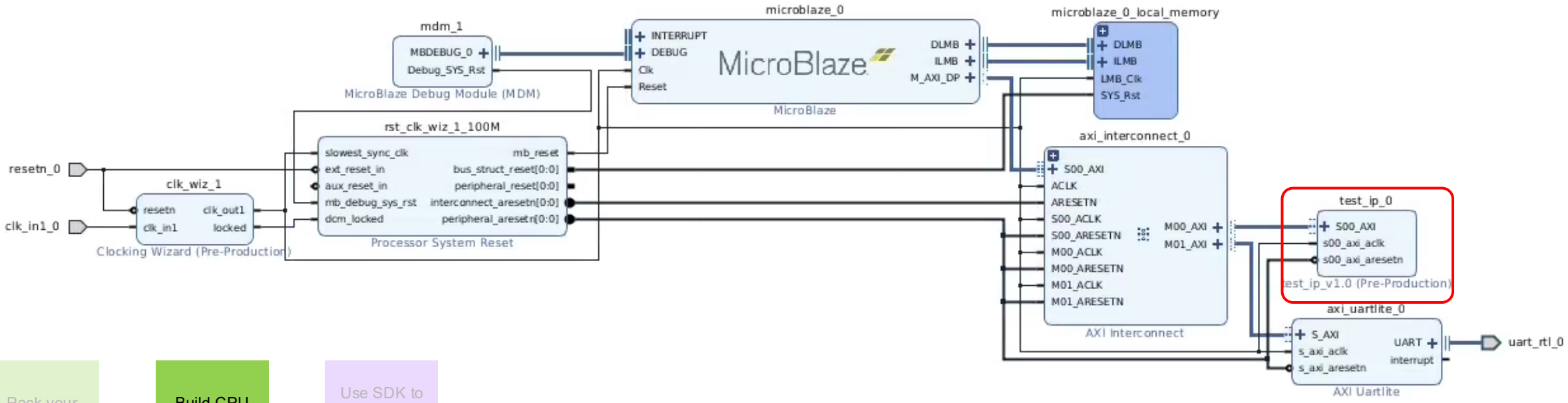
- As you modify the IP, you need to re-pack it for merging its change for projects using it
- Before you run the re-pack process, make sure your RTL code is correct
- In the re-pack process, you need to confirm all steps in Packaging Steps





Open Pre-built CPU System Project

- Let's replace the test_ip_0 with your own IP
- How to see this block diagram?
  - Flow Navigator -> IP INTEGRATOR -> Open Block Design



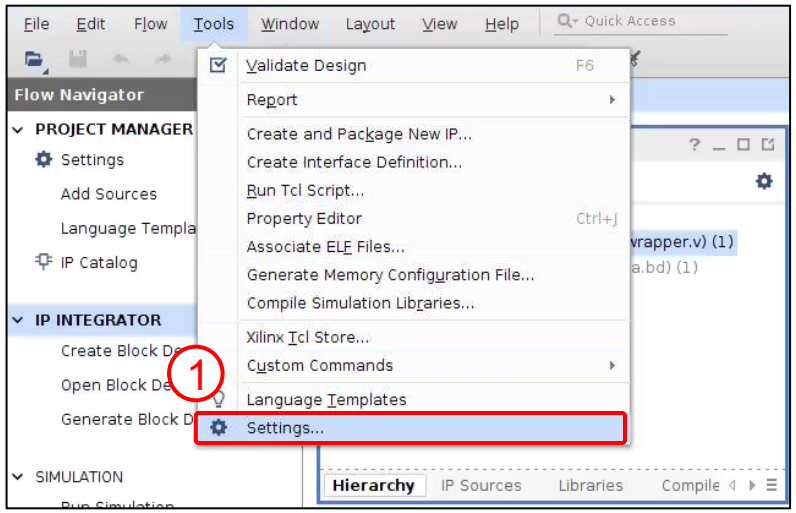
Pack your module into a AXI IP → **Build CPU system with packed IP** → Use SDK to build a demonstration application

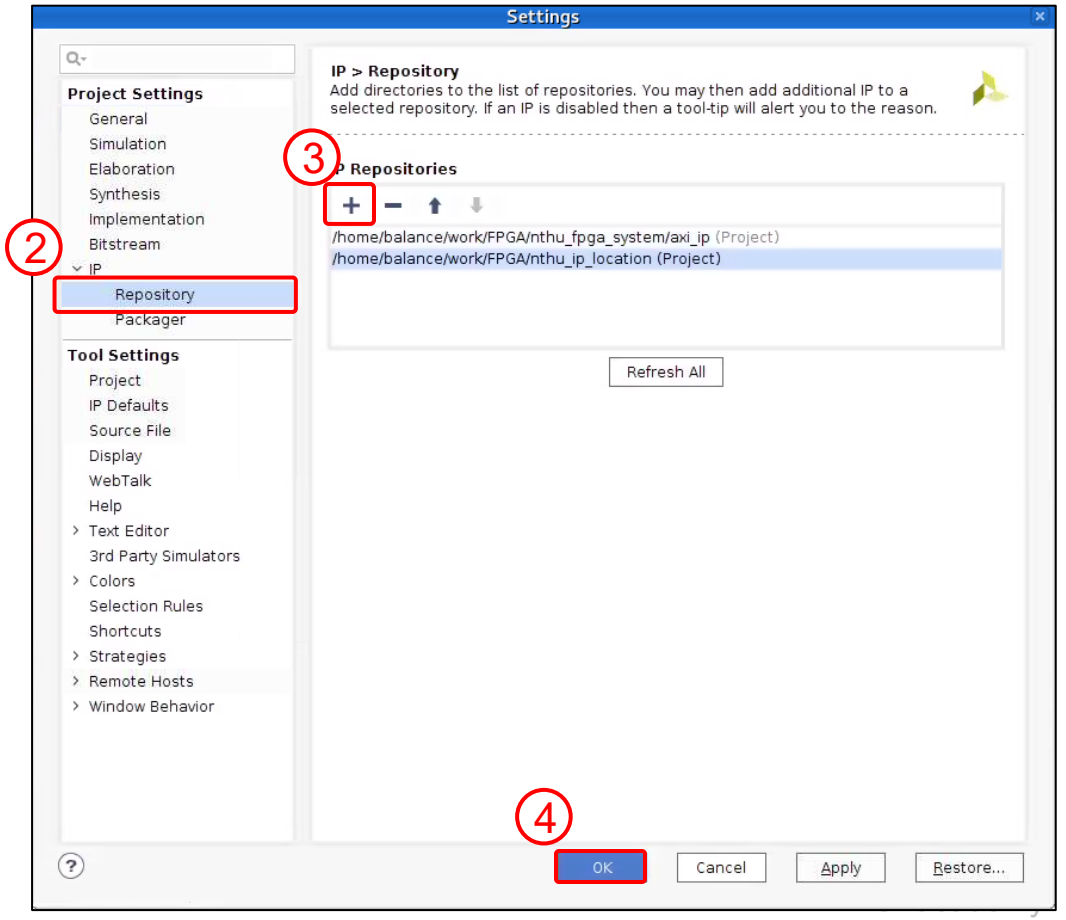
Reference of block design: https://fpga.eetrend.com/blog/2022/100562171.html

Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Import Your Packed IP

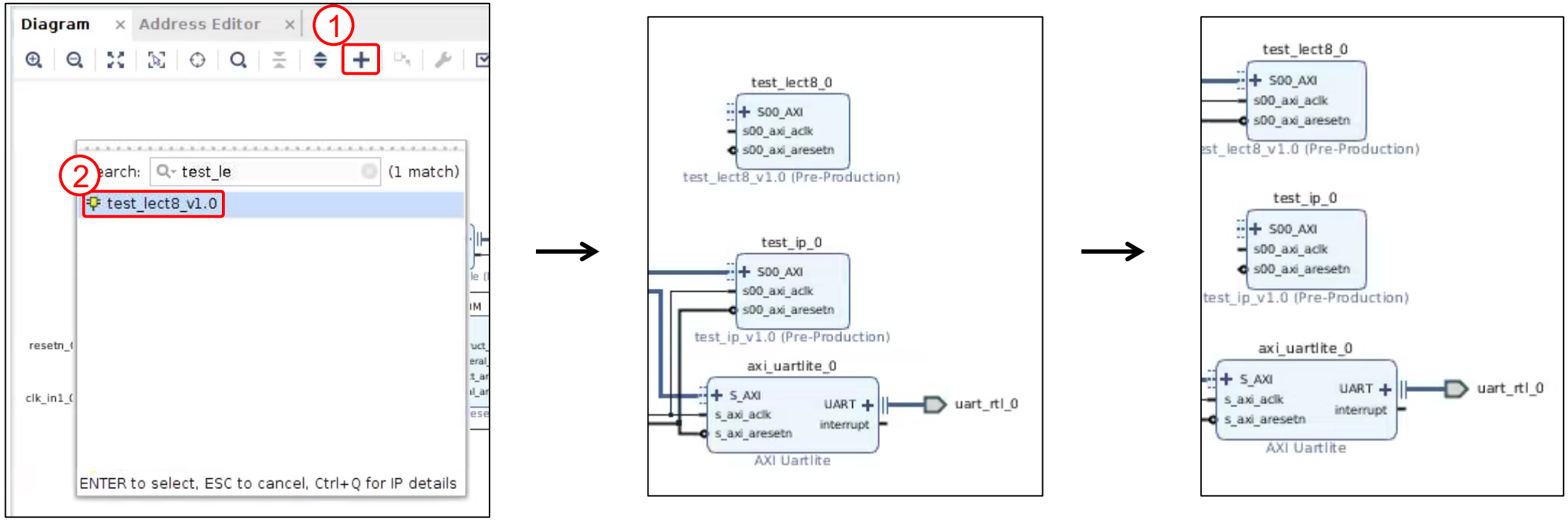
- Tools -> Settings -> IP -> Repository -> “+” -> select the location of your IP(p.25) -> OK





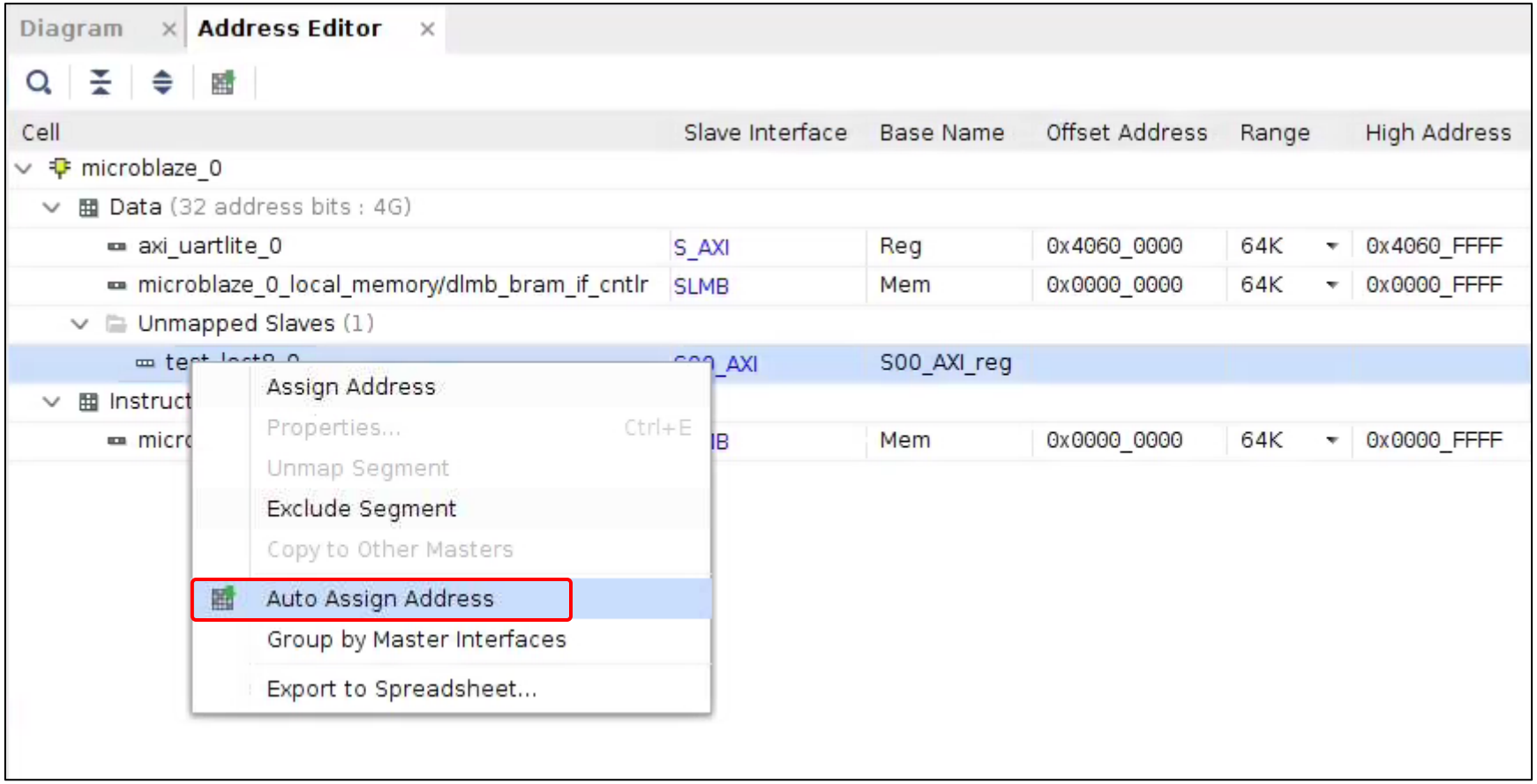
Replace the Old IP with Yours

- Diagram -> “+” -> search your IP -> click -> move the connection from Old IP to yours
- Don’t forget remove Old IP



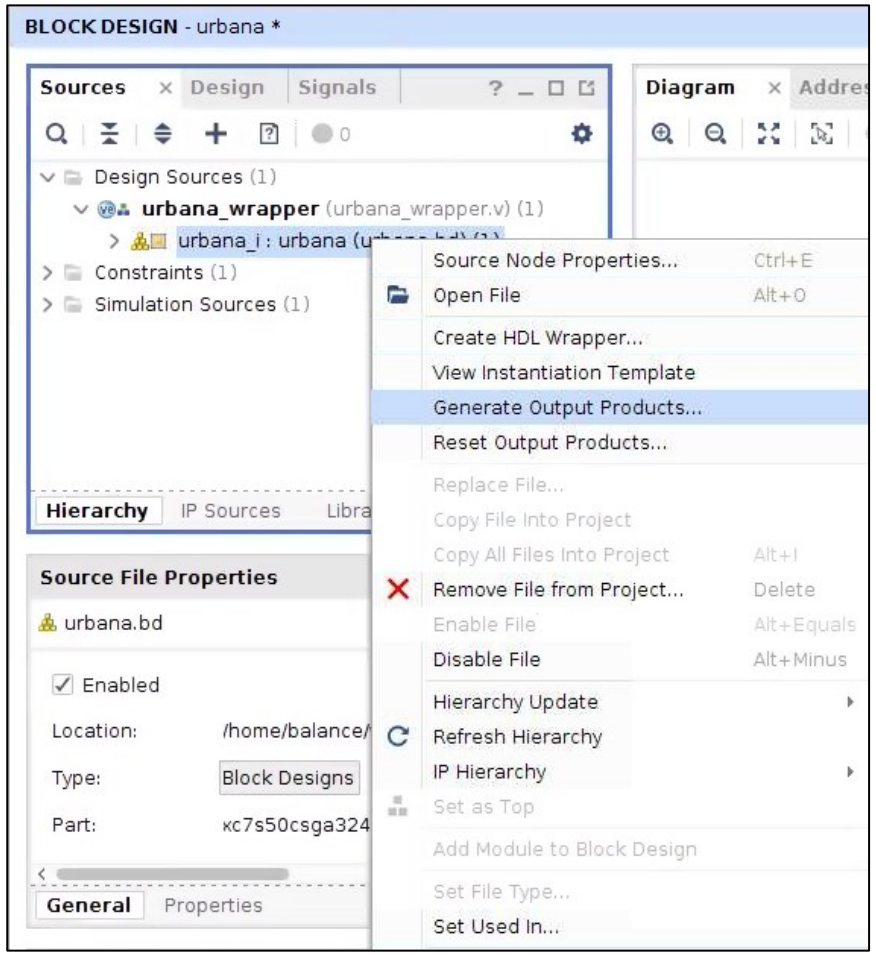
Assign Address for your IP

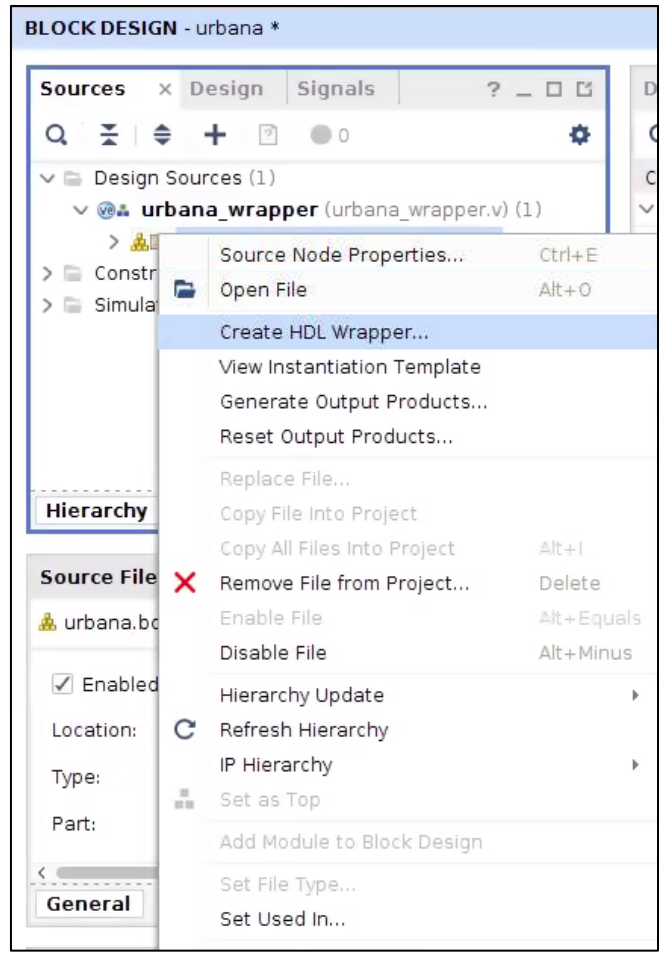
- Address Editor -> right click your IP -> Auto Assign Address

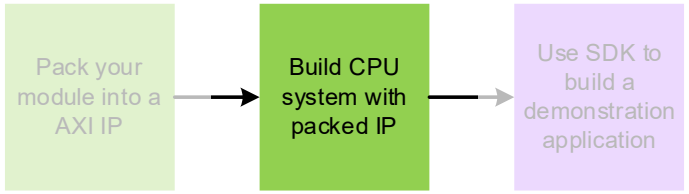


Update the Block Design

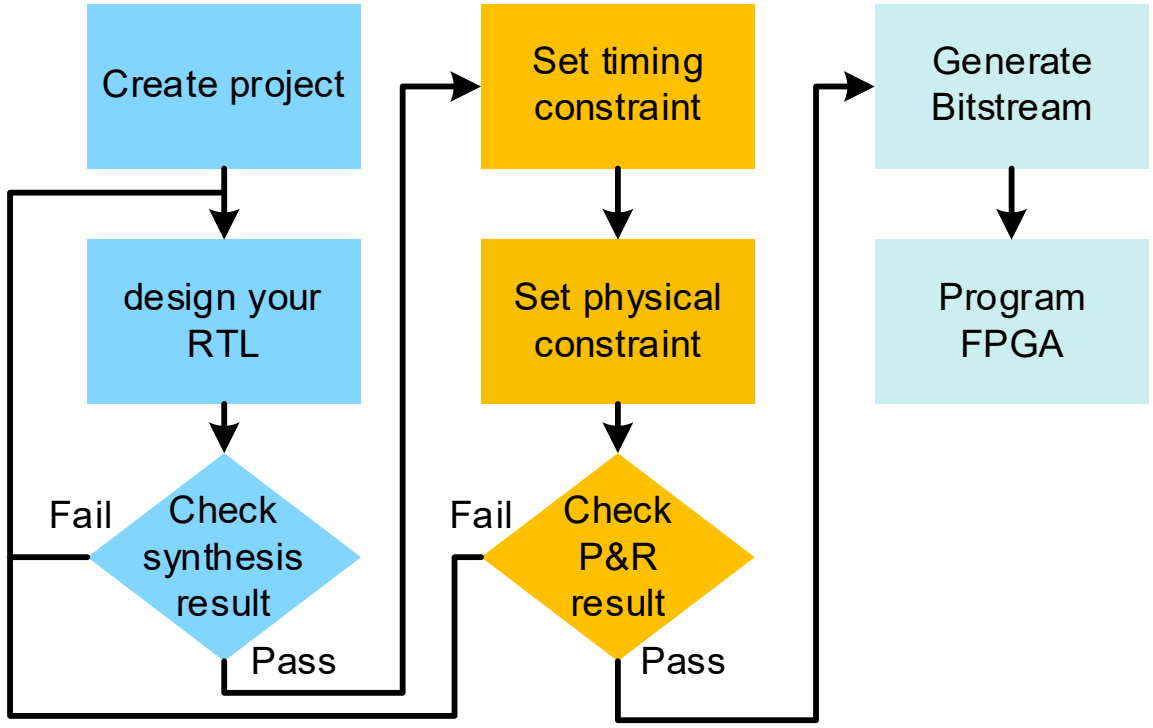
- Generate Output Product -> Create HDL Wrapper…



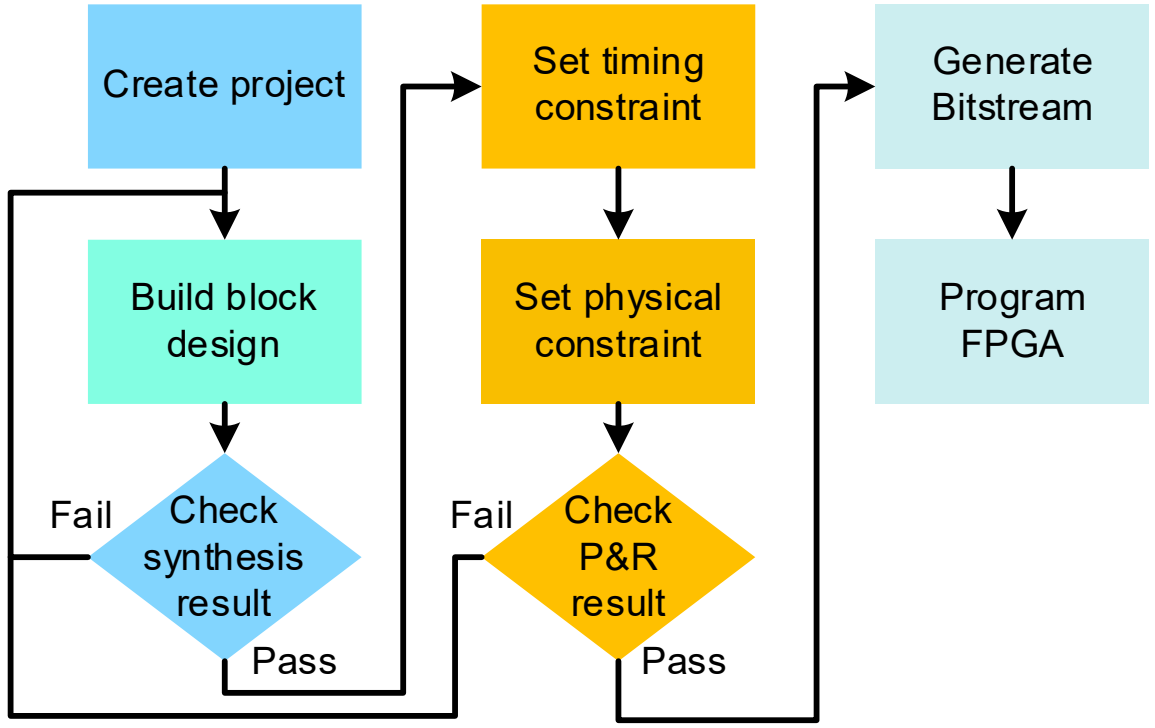




Vivado FPGA Design Flow



Pure RTL Flow



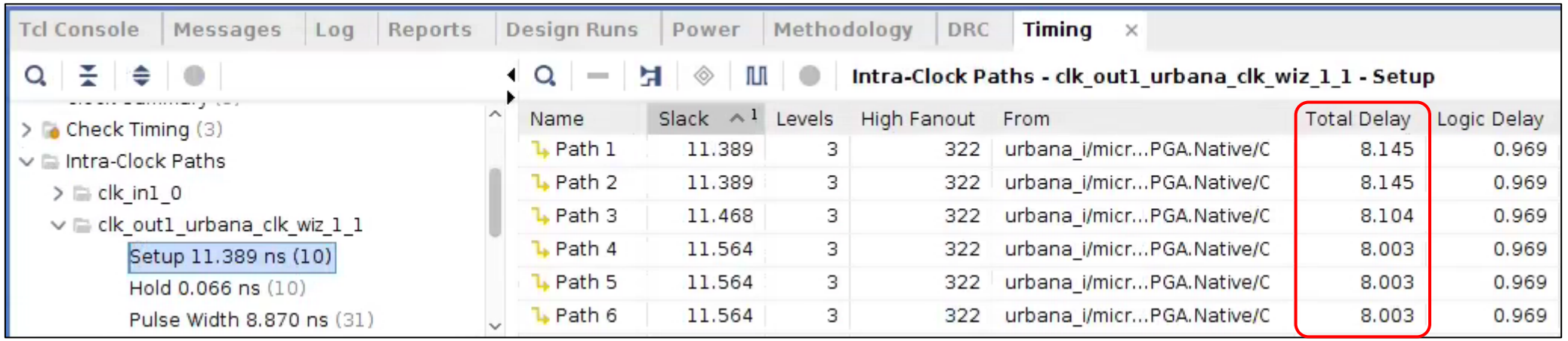
Block design Flow

Check Messages

- “Error” is not allowed for bitstream generation
- “Critical warning” and “Warning” should be checked carefully

Check Timing

- Make sure your timing is safe for operating on FPGA
- Open Implementation Design -> Report Timing Summary -> Timing

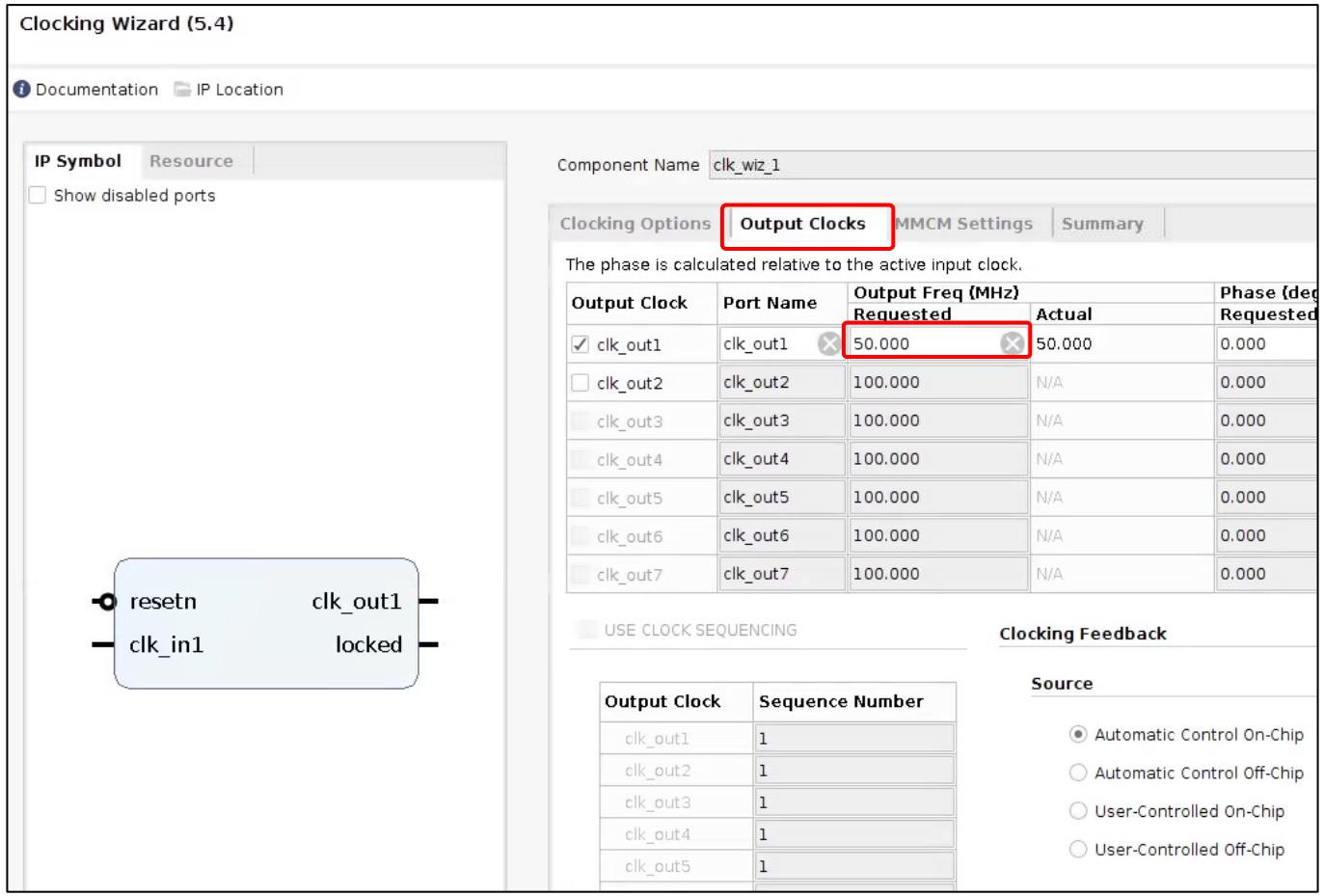


Name	Slack	Levels	High Fanout	From	Total Delay	Logic Delay
Path 1	11.389	3	322	urbana_i/micr...PGA.Native/C	8.145	0.969
Path 2	11.389	3	322	urbana_i/micr...PGA.Native/C	8.145	0.969
Path 3	11.468	3	322	urbana_i/micr...PGA.Native/C	8.104	0.969
Path 4	11.564	3	322	urbana_i/micr...PGA.Native/C	8.003	0.969
Path 5	11.564	3	322	urbana_i/micr...PGA.Native/C	8.003	0.969
Path 6	11.564	3	322	urbana_i/micr...PGA.Native/C	8.003	0.969

F_{MAX} = 1000/Total Delay MHz

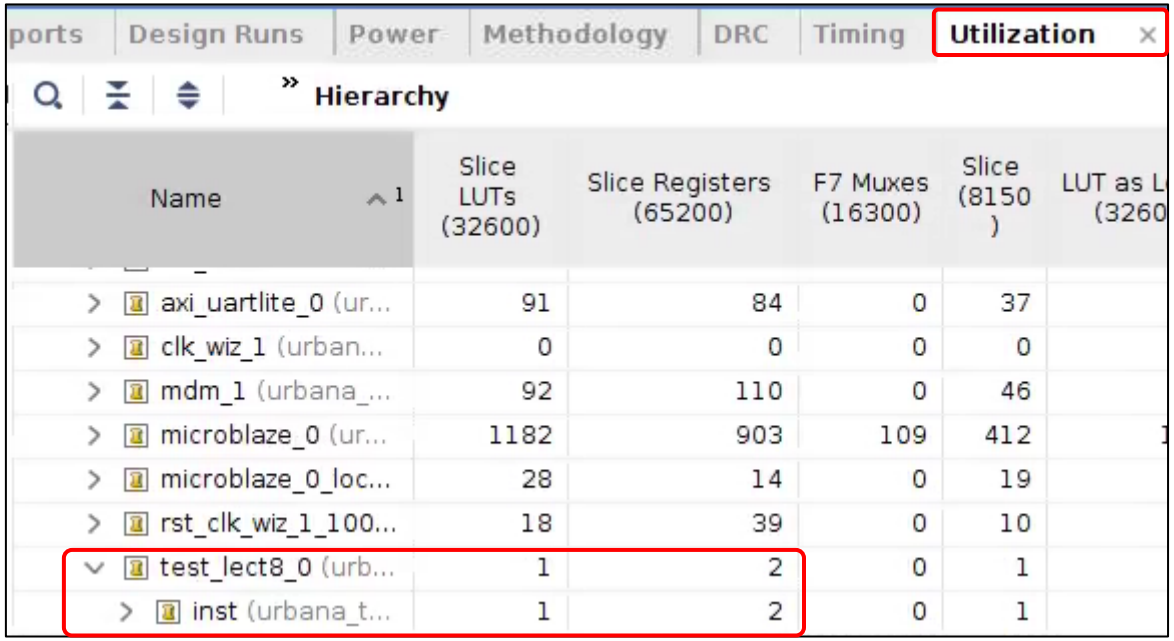
Modify Clock Frequency

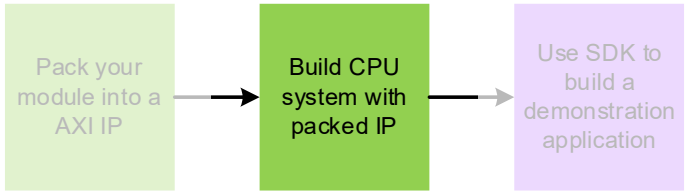
- The clock from “clk_wiz_1” is 50MHz
- You can modify it by double click on the block



Check Utilization

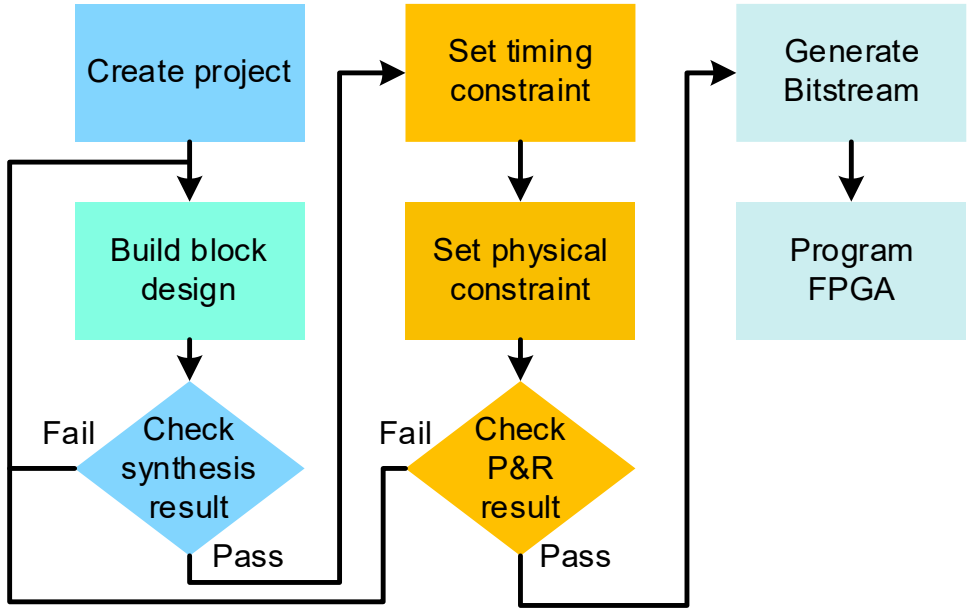
- “Utilization” in Vivado is similar with “gate count/area” in synthesis tool
- Open Implementation Design -> Report Utilization -> Utilization

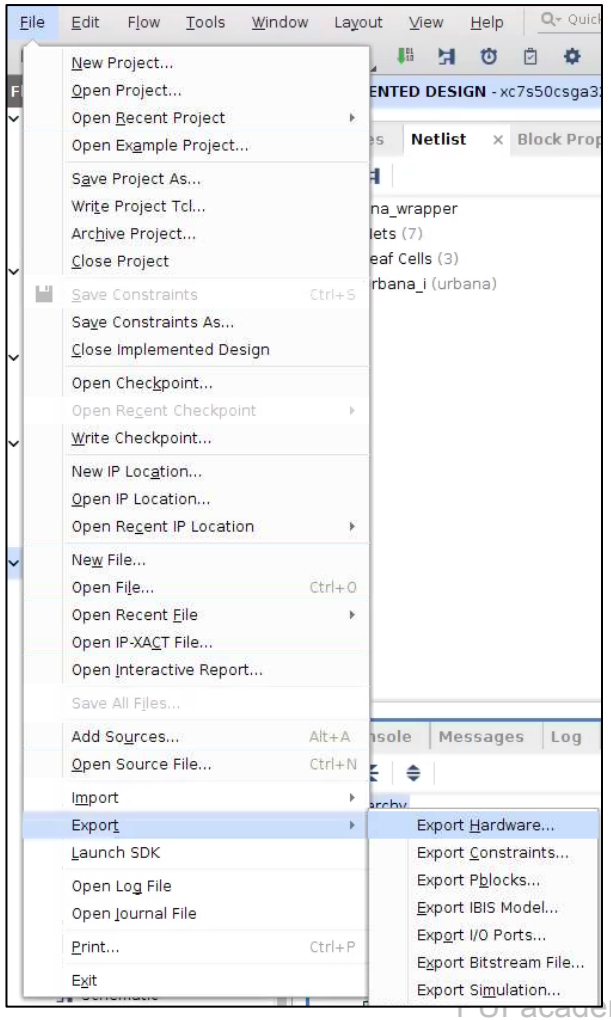


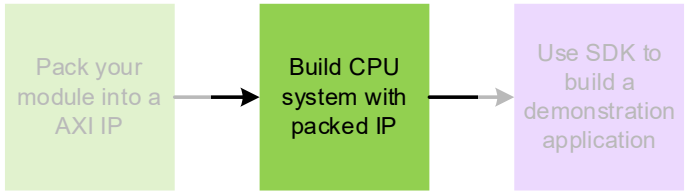


Generate HDF File for SDK

- As “Generate Bitstream” is done, you will find a bit file (urbana_wrapper.bit) under
  - <your_path>/nthu_sys_lcm_only/nthu_sys_lcm_only.runs/impl_1
- Generate HDF file (urbana_wrapper.hdf) by following step:
  - File -> Export -> Export Hardware -> Include bitstream ->OK





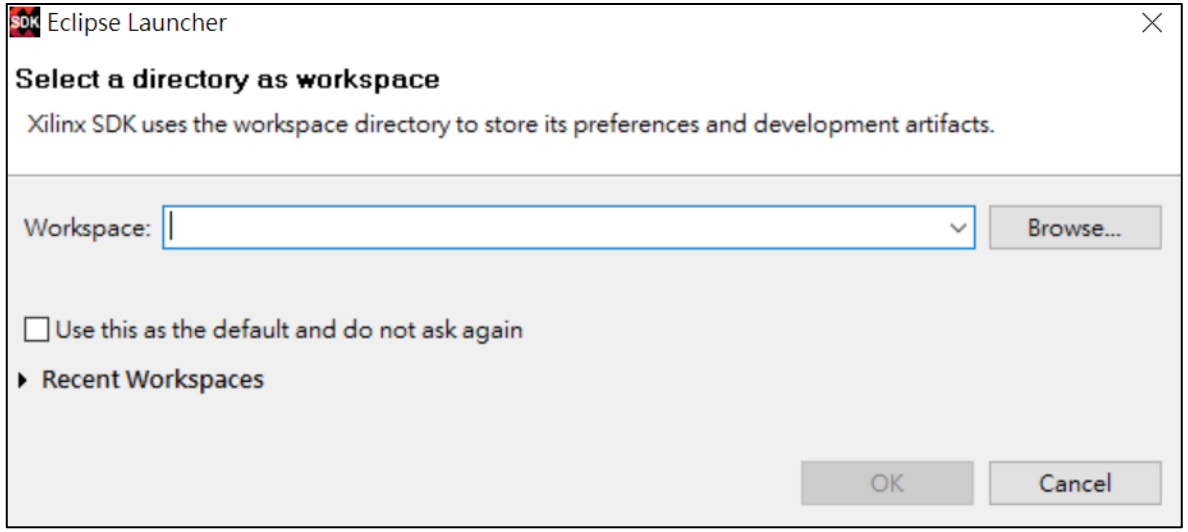


Program FPGA via bitstream

- Follow the p.14-16 in “113 Digital Logic Design - Lecture 7.1 - Urbana FPGA board power on testing”
- The CPU system start to operate as SW0 switch to “ON”

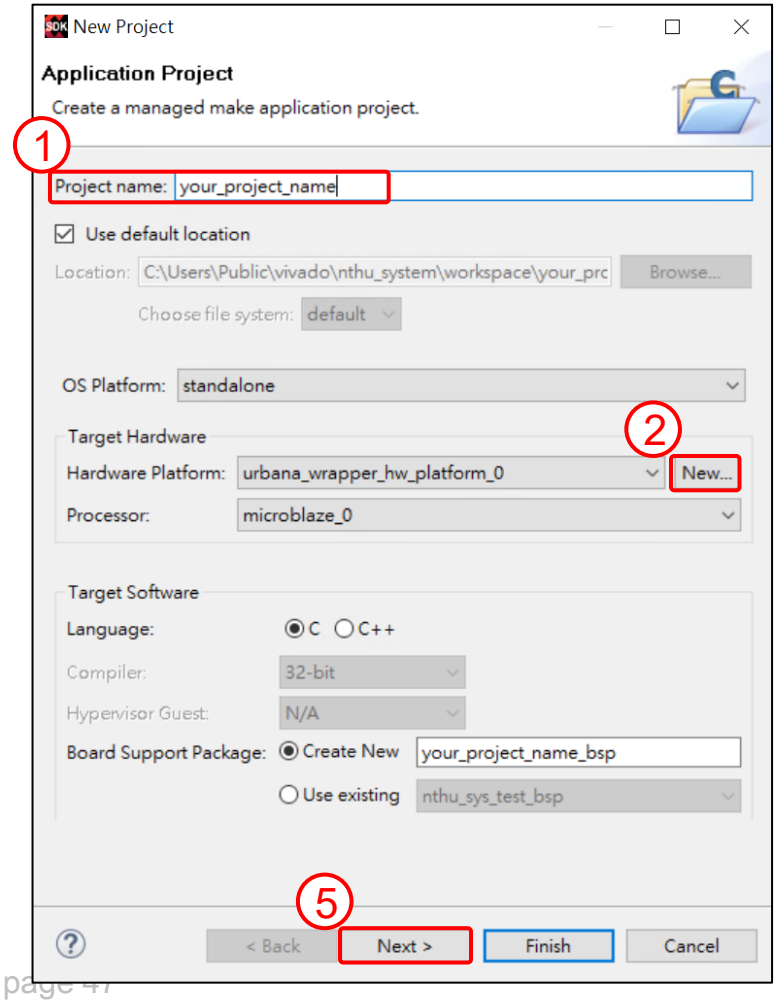
Open SDK

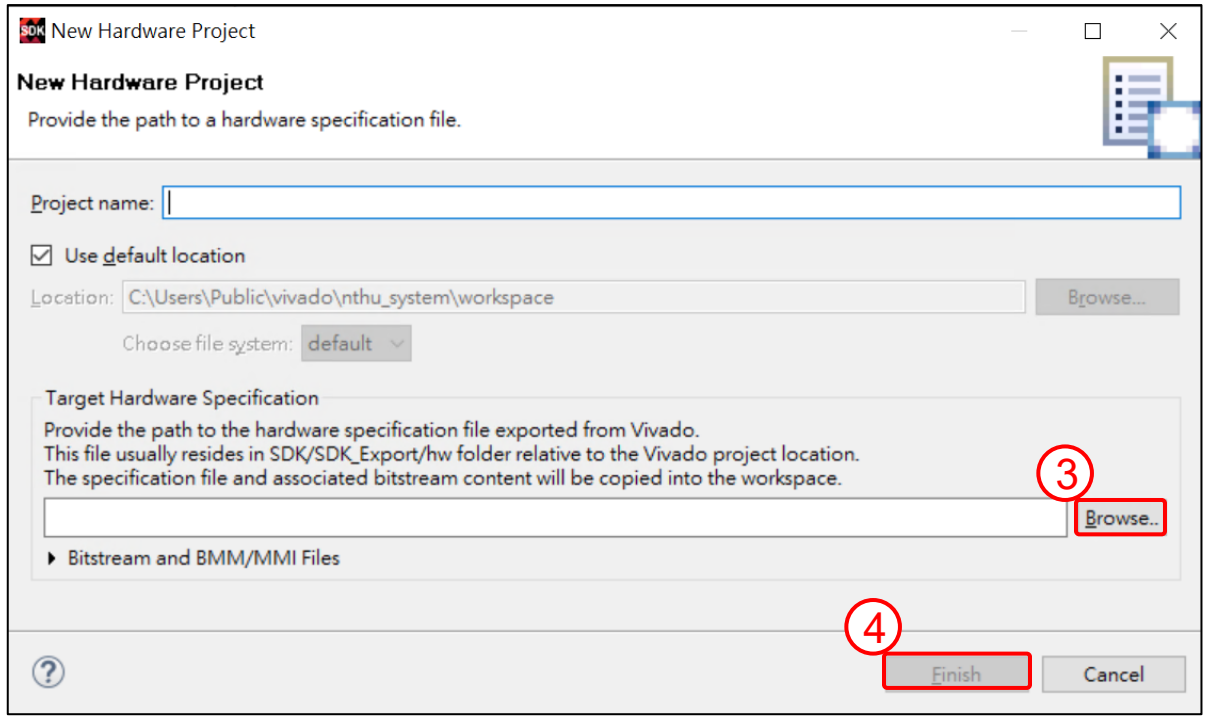
- Give a path of folder as the workspace



New Application Project (1/2)

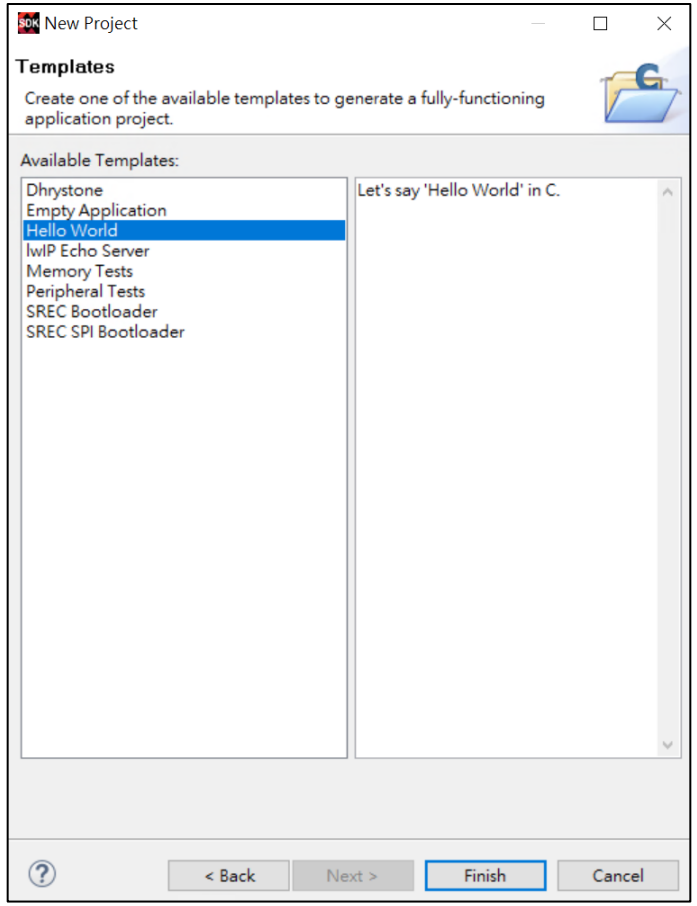
- Give a project name -> New Hardware platform -> select your HDF file -> Finish -> Next





New Application Project (2/2)

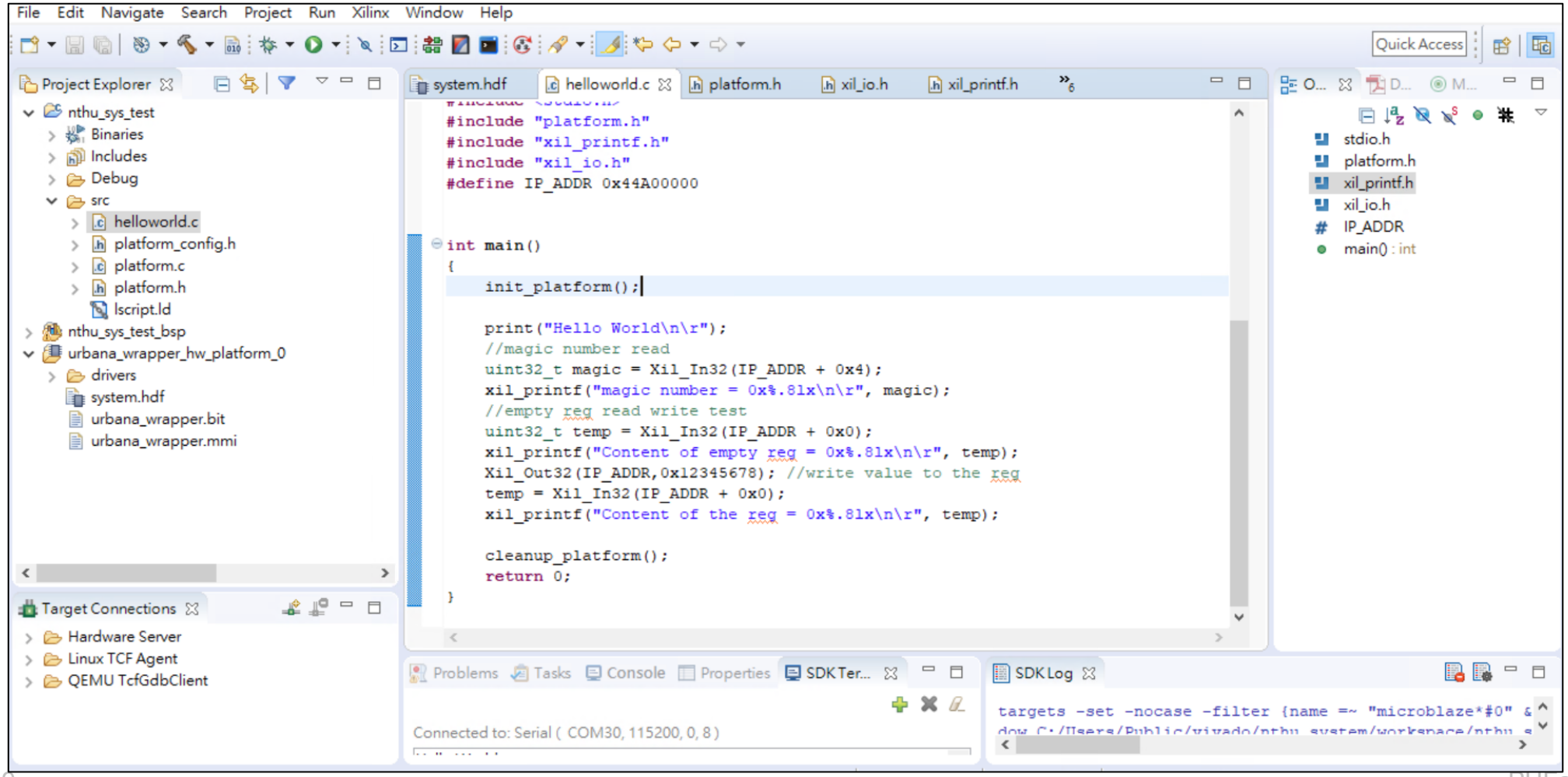
- Select “Hello World” -> Finish



Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Find the main Function

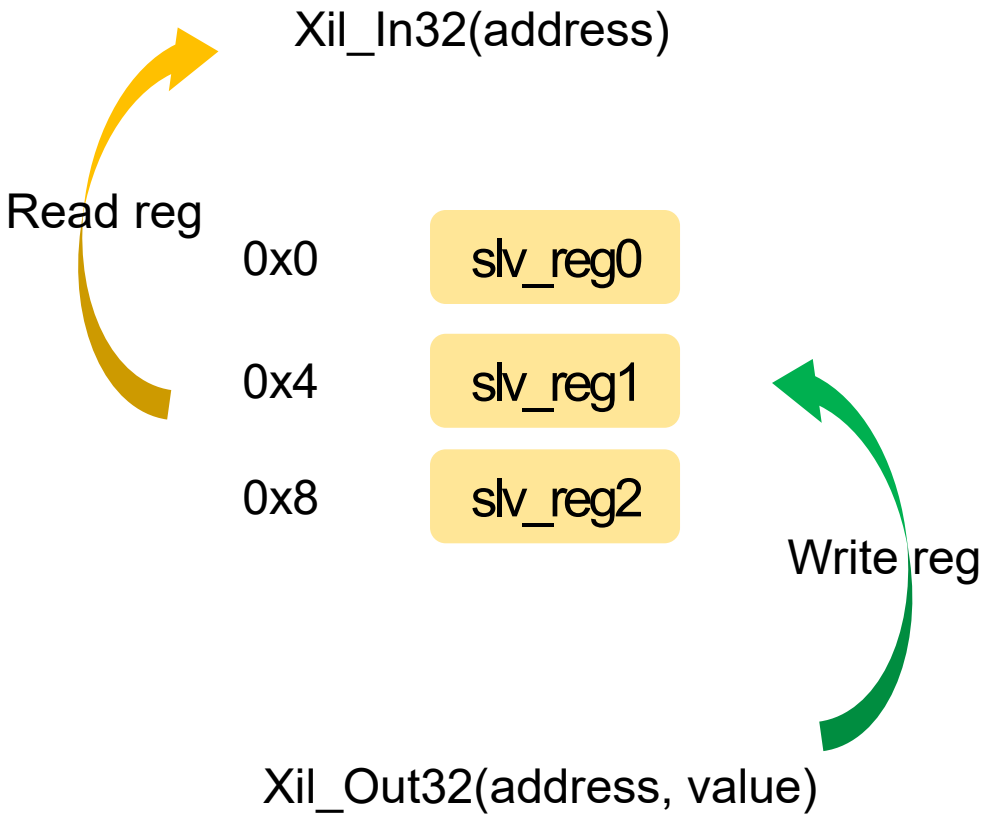
- Project Explorer -> helloworld.c



Modify main Function

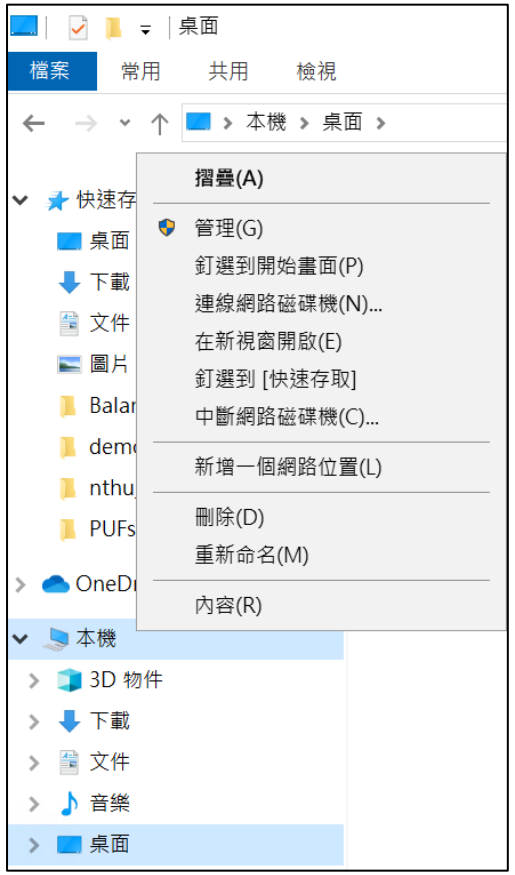
```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_io.h"
#define IP_ADDR 0x44A00000

int main()
{
    init_platform();

    print("Hello World\n\r");
    //magic number read
    uint32_t magic = Xil_In32(IP_ADDR + 0x4);
    xil_printf("magic number = 0x%.8lx\n\r", magic);
    //empty reg read write test
    uint32_t temp = Xil_In32(IP_ADDR + 0x0);
    xil_printf("Content of empty reg = 0x%.8lx\n\r", temp);
    Xil_Out32(IP_ADDR,0x12345678); //write value to the reg
    temp = Xil_In32(IP_ADDR + 0x0);
    xil_printf("Content of the reg = 0x%.8lx\n\r", temp);

    cleanup_platform();
    return 0;
}
```

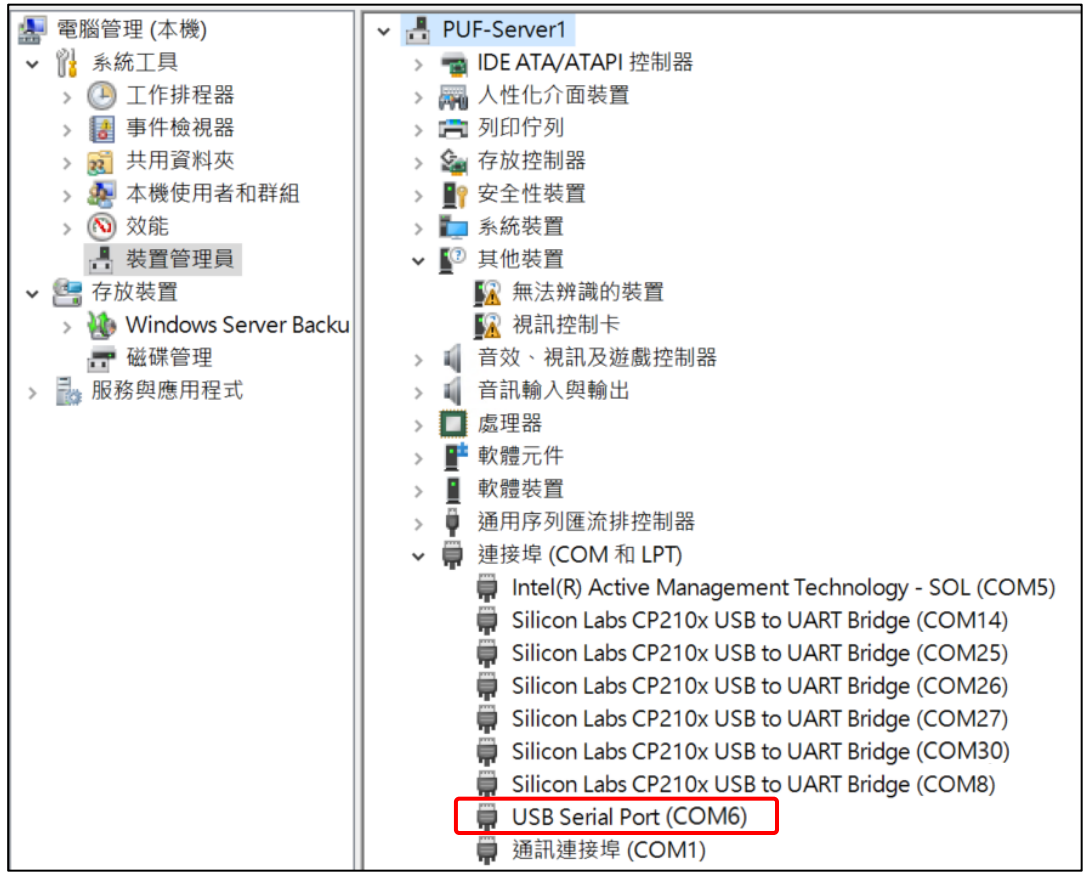
Your IP base address



Connect UART to Terminal (1/2)

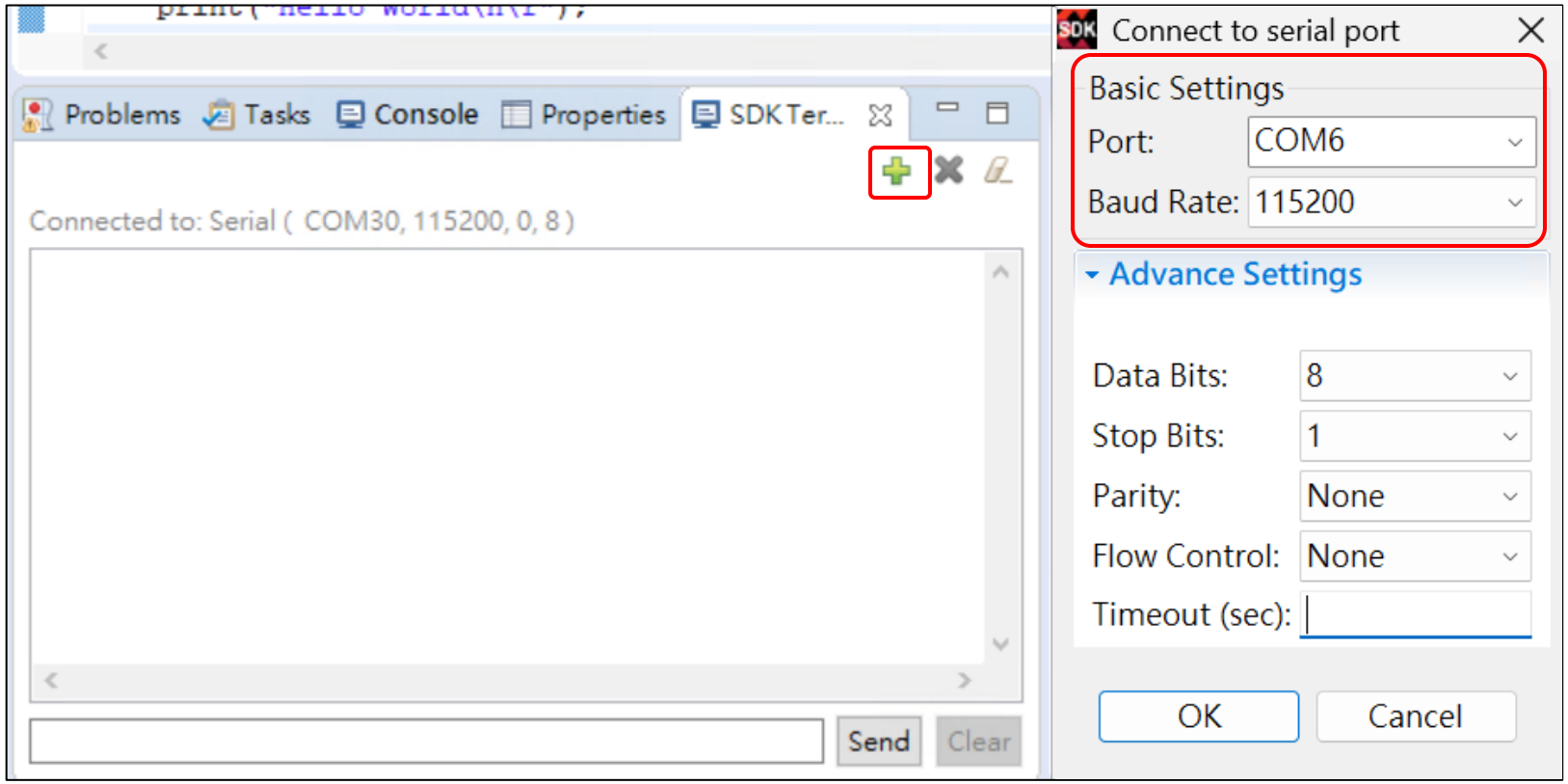
- Find the COM port number for UART via:
  - 本機 -> 管理 ->裝置管理員





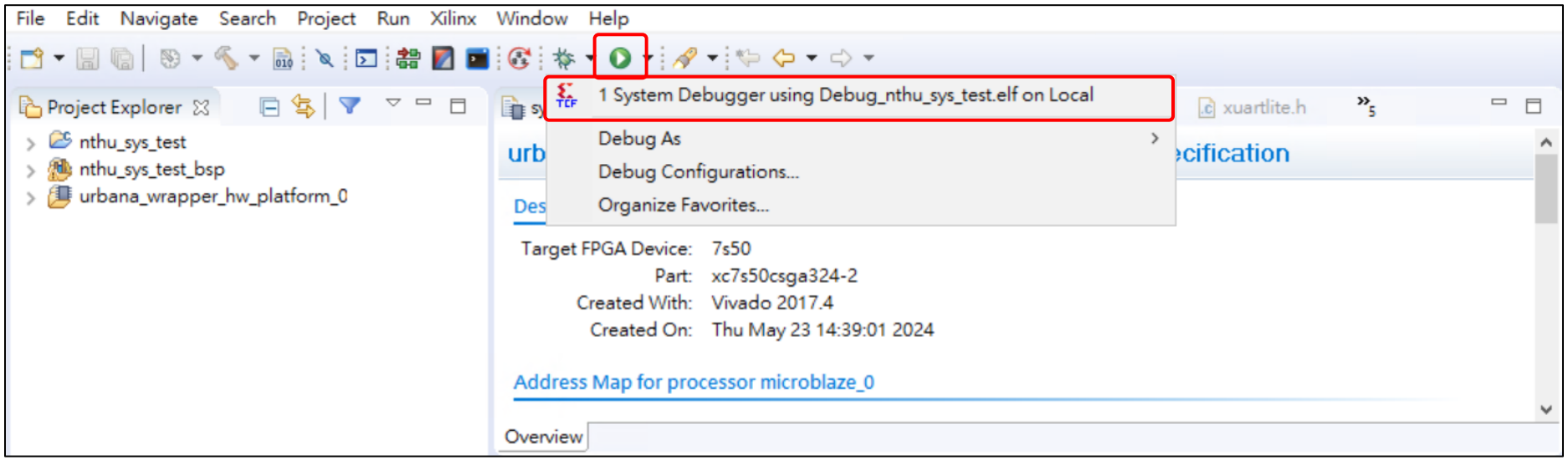
Connect UART to Terminal (2/2)

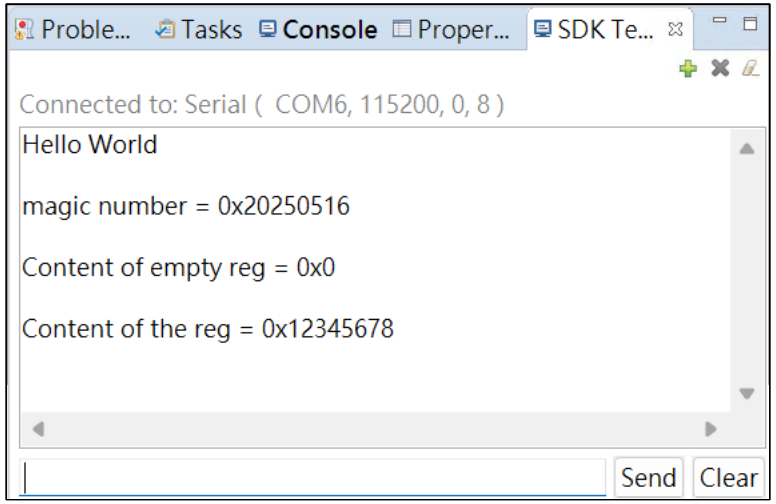
- SDK Terminal -> “+” -> set COM? -> Baud Rate 115200



Pack your module into a AXI IP → Build CPU system with packed IP → Use SDK to build a demonstration application

Run main Function on FPGA





Agenda

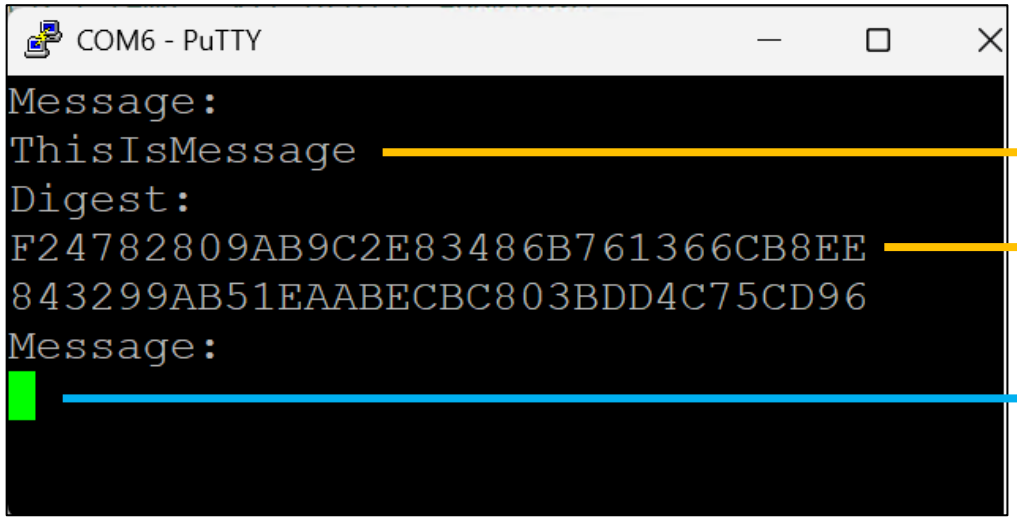
1. Why we need FPGA
2. What is FPGA
3. Introduction to Xilinx FPGA
4. Practice: create an AXI IP and access it on SDK
5. Lab 03 assignment

Lab 03 Requirements (For Demonstration)

- Your FPGA demonstration should be able to let user enter message via keyboard, and then pop out corresponding digest via your SHA-256 module repeatedly
- The order of input and output is shown in the figure below
- Size of input message change between 0~32 bytes of character

input	start_p	
input [63:0]	msg_len	//Unit: bit
input	msg_vld	
input [31:0]	msg_dat	//Total size is N * 8 bits
input [3:0]	msg_be	//Byte enable
input	msg_lst	//The last word of messages
output	msg_rdy	
output	dgst_done	
output [255:0]	dgst	
input	clk	//Periodic clock, 50MHz
input	rst_n	//Asynchronous reset, active low

Lab 02 IO SPEC

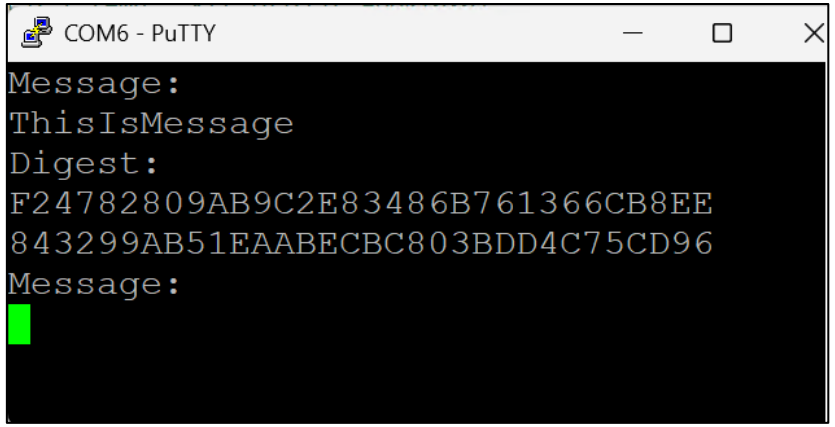


Message Input: *0~32-byte characters

Digest output: 256 bits expressed in hexadecimal

An addition “Enter” output is required for user to key in new inputs from new line

Specification for Input/Output Format on Terminal



Message(given by keyboard):

m0	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
T	h	i	s	l	s	M	e	s	s	a	g	e

Digest(given by your design):

d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15
0xF2	0x47	0x82	0x80	0x9A	0xB9	0xC2	0xE8	0x34	0x86	0xB7	0x61	0x36	0x6C	0xB8	0xEE

d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15
0x84	0x32	0x99	0xAB	0x51	0xEA	0xAB	0xEC	0xBC	0x80	0x3B	0xDD	0x4C	0x75	0xCD	0x96

Lab 03 Requirements (For Presentation)

- Block diagram of your IP design in FPGA
- Total number of registers that you used and corresponding purpose in packing your SHA-256 module
- Resource usage summary table (in Vivado -->IMPLEMENTATION-->Report Utilization)
- Flow chart of your C code for fulfilling the Lab 03 requirements
- Problems you encounter/solve
- Others…
- Time for presentation: 6 minutes

113學年度 課程大綱

Date	w	Content	host	作業	Location
2月21日	1	學期內容介紹。 晶片安全介紹	MY		資電館202
2月28日	2	AES演算法介紹 (線上課程) SHA演算法介紹 (線上課程)			放假
3月7日	3	Linux 指令介紹; RTL 語言設計 (Introduction, RTL & test bench語法) RTL 語言設計 (Combinational Logic: blocking);	CW		台達館219
3月14日	4	RTL 語言設計 (Sequential Logic: non-blocking)	Danny		台達館219
3月21日	5	RTL 語言設計 (FSM: moore, mealy) (架構)	Danny		台達館219
3月28日	6	RTL 語言設計 (Test Bench: input, DUT, output, compare)	CY	Lab1/compressor	台達館219
4月4日	7	PKC演算法介紹 (線上課程)			放假
4月11日	8	RTL 語言設計 (Digital Flow and Synthesis)	CY	Lab2/SHA2 with padding	台達館219
4月18日	9	期中報告 Lab1	助教群		台達館219
4月25日	10	補教周/助教時間(null)	助教群		台達館219
5月2日	11	硬體安全設計與操作	MY		台達館219
5月9日	12	期中報告 Lab2 (SHA2 with padding)	助教群		台達館219
5月16日	13	FPGA 進行設計開發與驗證。	Balance	Lab3/SHA2 FPGA	台達館219
5月23日	14	助教時間	助教群		台達館219
5月30日	15	Entropy Introduction; 抗攻擊設計 (線上課程)			放假
6月6日	16	期末報告 Lab3 (FPGA)	助教群		台達館219

請善用eeclass討論區!

- 有Lab 03作業上的問題，請多使用討論區提問，讓其他有相同疑問的同學也能獲得解答!

Tips

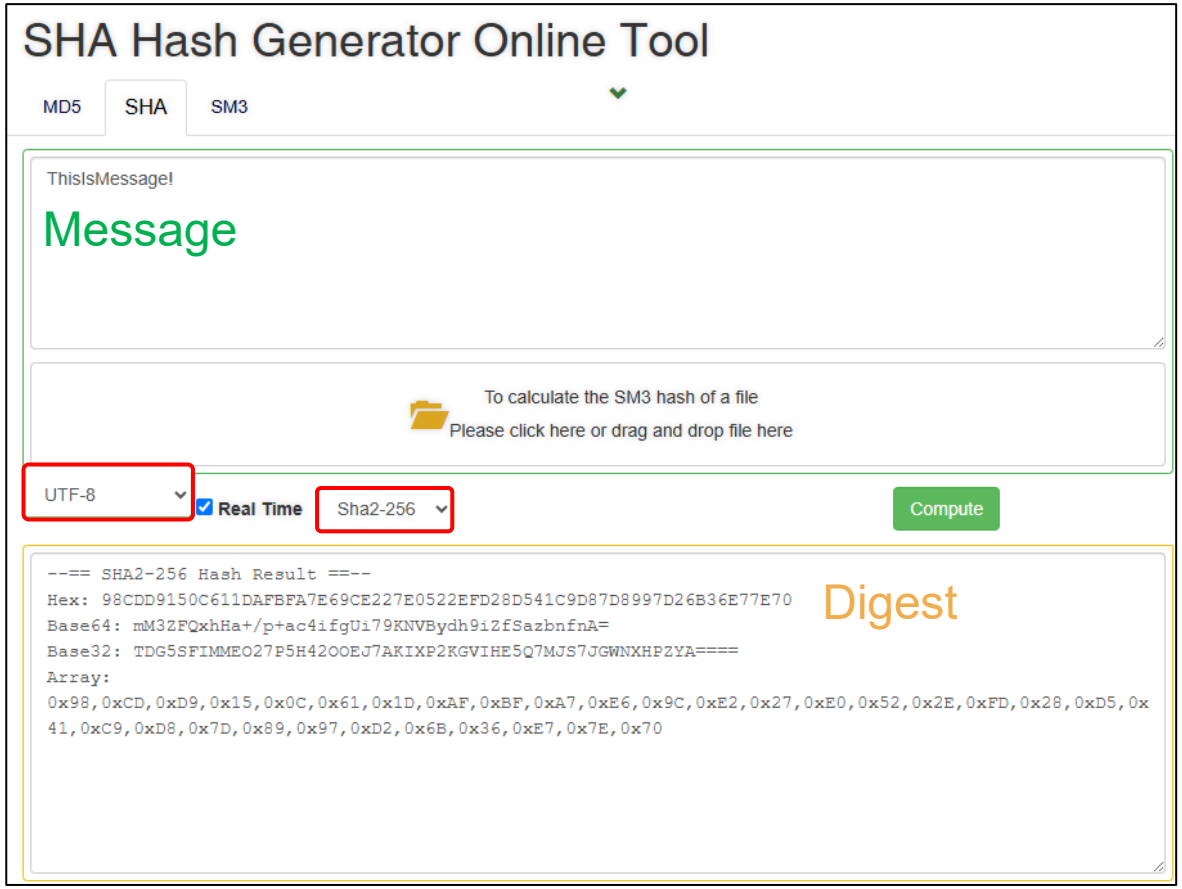
- I/O function you may need to finish Lab03:
  - Interact with your IP registers:
    - uint32_t Xil_In32(uint32_t address): return the value in the “address”
    - void Xil_Out32(uint32_t address, uint32_t value): write “value into the “address”
  - Interact with UART:
    - void xil_printf(const char8 *ctrl1, ...): same as printf
    - char8 inbyte(void): collect char from terminal

```
char p[16];
int i = 0;
p[i] = inbyte();
while(p[i]!='\r' && i!=15){
    i++;
    p[i] = inbyte();
}
xil_printf("get word: %s\n\r", p);
```

Example code for using inbyte() to build scanf() function

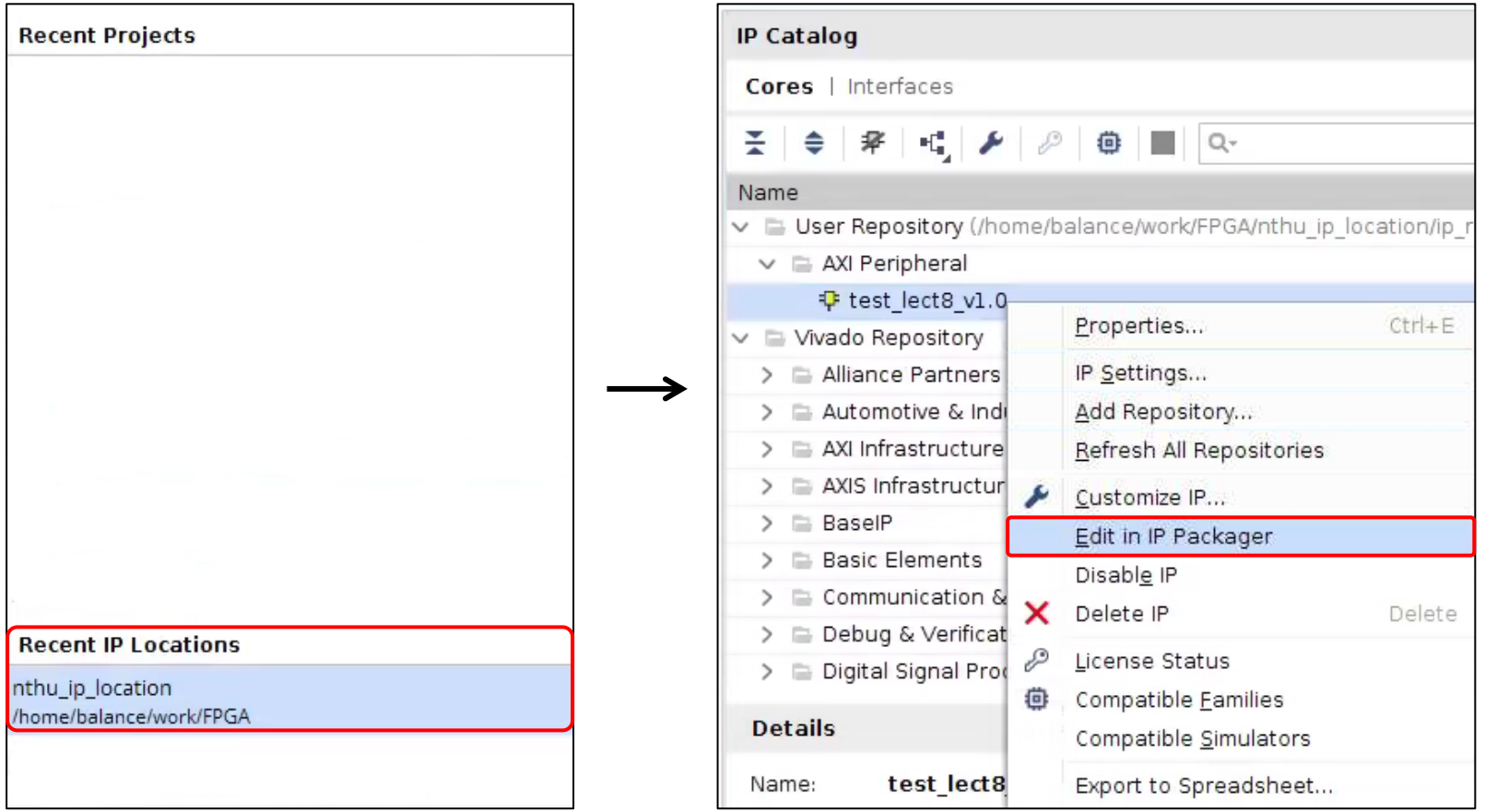
Tips

- You need to check
  - what byte values you receive from your keyboard (e.g., press “Enter”, and what is the corresponding byte value)?
  - The conversion between ciphertext to hexadecimal on terminal (e.g., your Digest is 0x92, how to show “92” on terminal)
- For you to check your demonstration
  - https://the-x.cn/en-US/hash/SecureHashAlgorithm.aspx



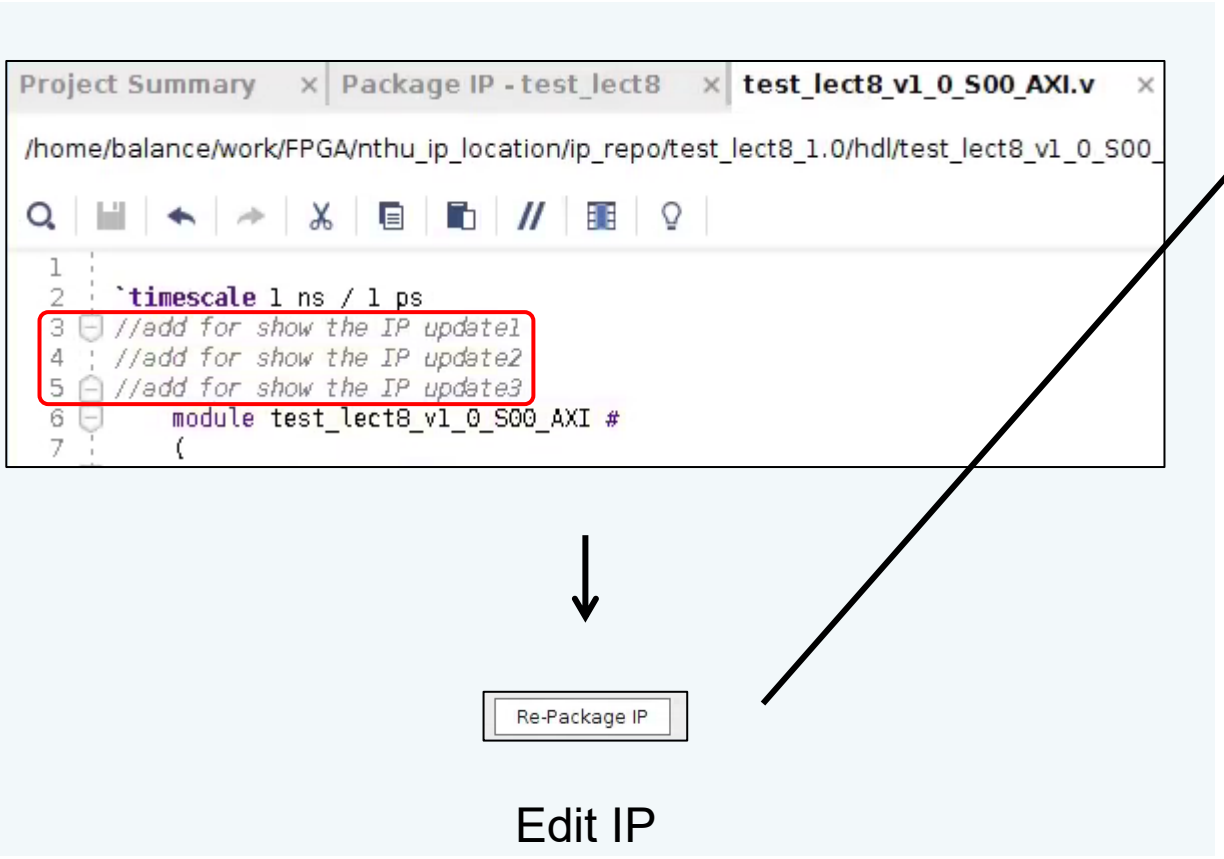
Tips – Open Your Existed IP for Modification

- Your IP location -> Find your IP in “IP Catalog” -> right click -> Edit in Packager

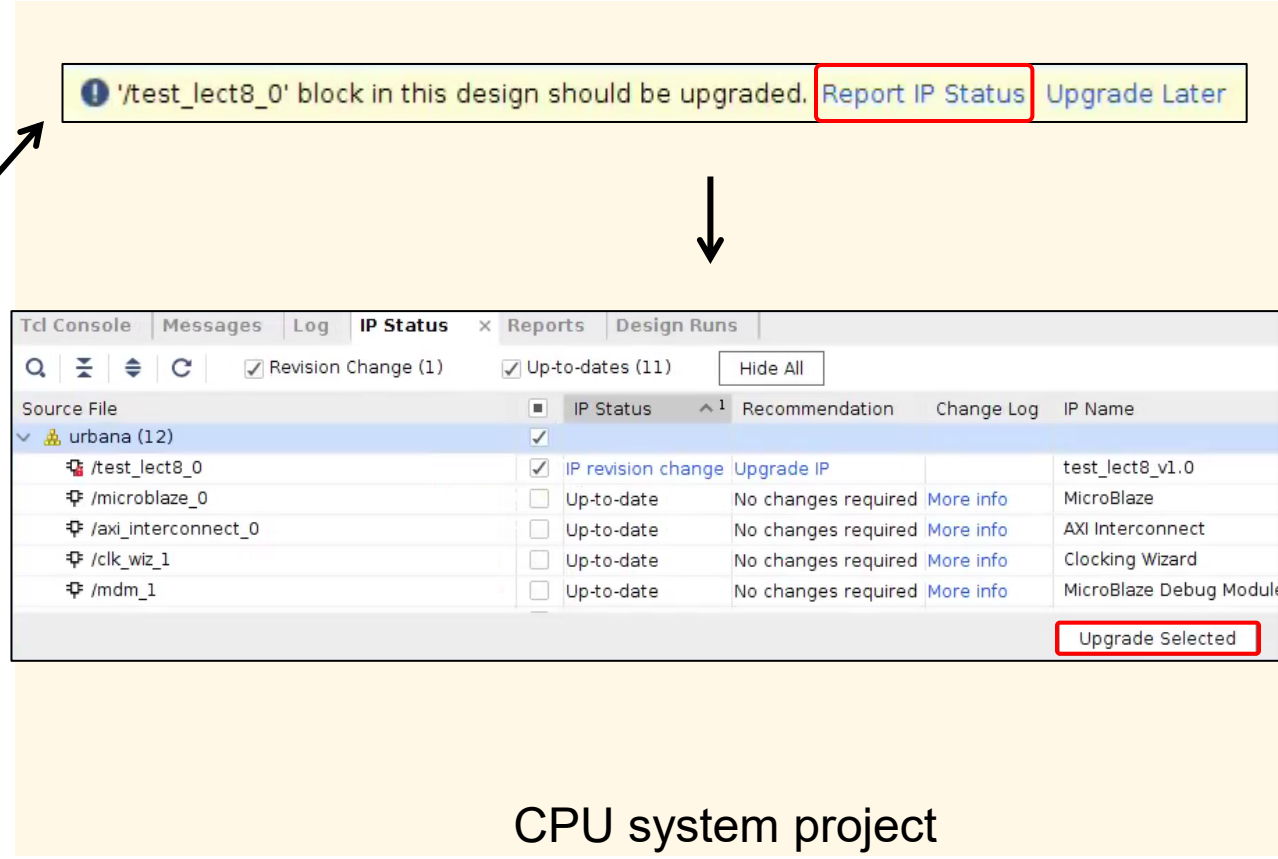


Tips – Update Your Modified IP in CPU System Project

- As you re-pack your IP, the CPU system project will jump out a message about IP upgraded
- Click Report IP Status -> Upgrade Selected -> OK -> Generate



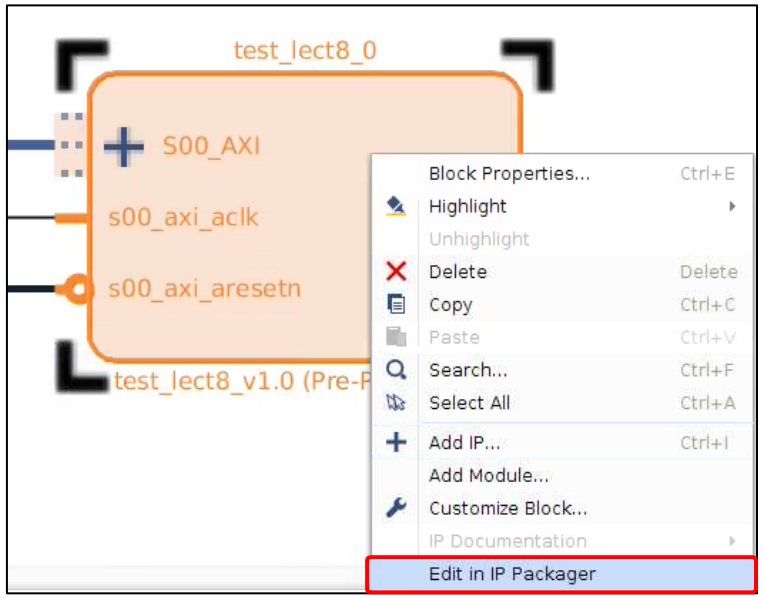
Edit IP

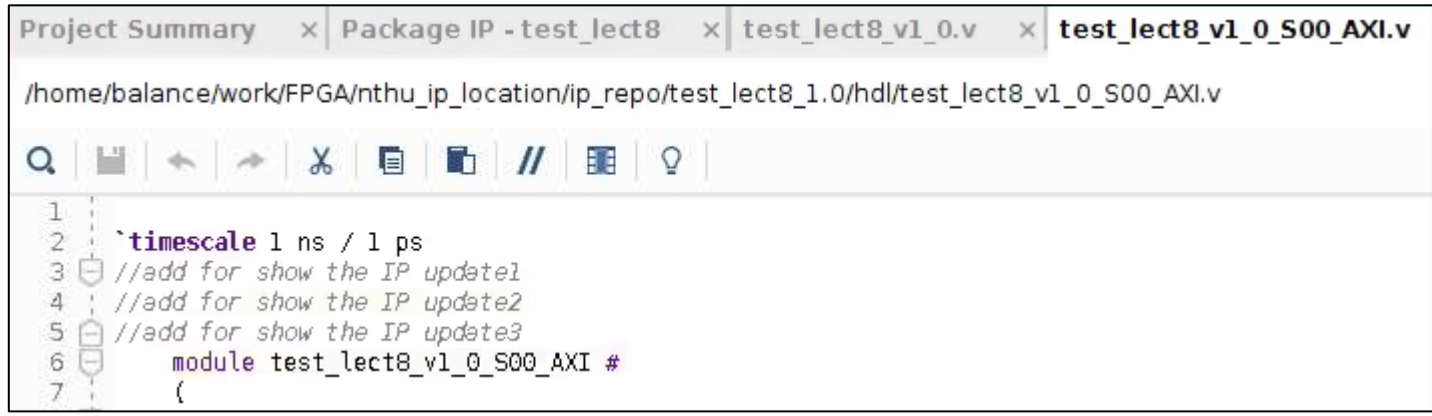


CPU system project

Tips – Verify Your Modified IP is Updated

- You may want to check if your modification has updated in the CPU system project after the step in previous page
- Right click on your IP block -> Edit in PI Packager -> check the RTL code
- Strongly recommend you modify your RTL throughput “IP location” rather than “right click the block in CPU system project”, sometimes they are saved in different path.





Feedback to us

https://forms.office.com/r/DYDu8vLaWN

Thank you!

PUFacademy
A PUFsecurity Alliance

Visit our website: pufacademy.com

Contact us: PUFacademy@pufsecurity.com