

Parallel and Reconfigurable VLSI Computing (5)

# Walk Through FPGA Design

Hiroki Nakahara

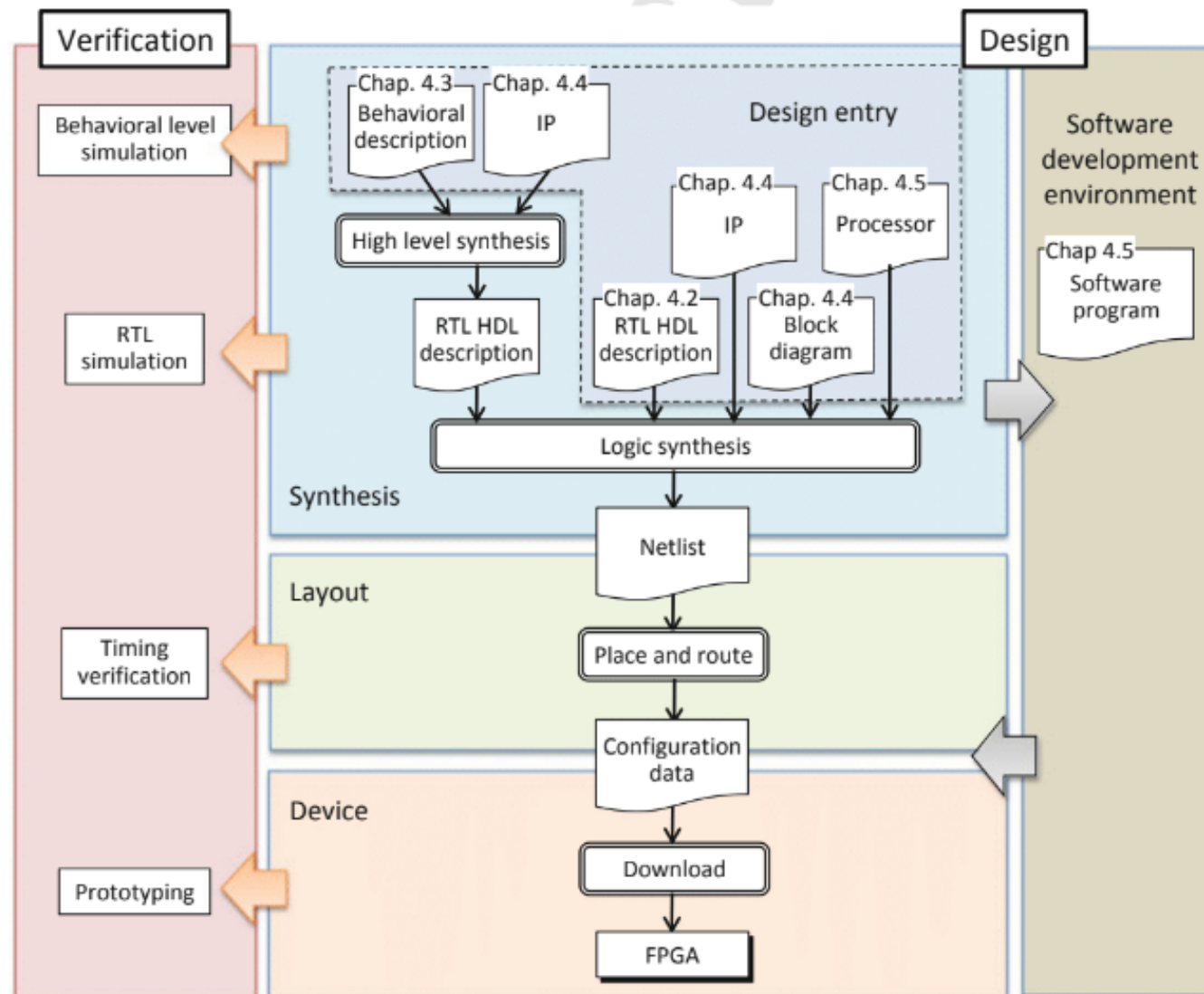
Tokyo Institute of Technology

# Outline

- Design flow, design tool, development environment for implementation of target system on FPGA
- Target board: Digilent Zybo-Z7
- Design tool: Xilinx Vivado 2017.4, XSDK 2017.4
  - 1 Design flow overview
  - 2 HDL design flow with logic simulation
  - 3 Processor design flow

# Conventional FPGA Design Flow

# FPGA Design Flow





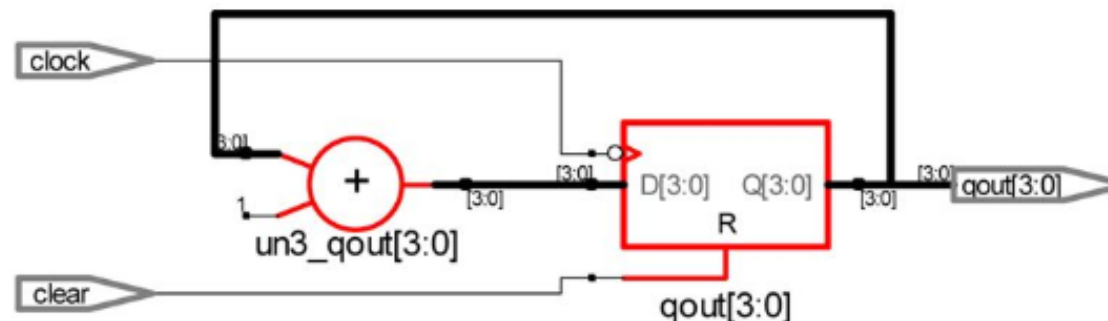
# Register-Transfer Level (RTL) Design

- Combinational Logic
  - Un-timed behavior (equation or truth table)
- Sequential Logic
  - Timed one (Finite state machine (FSM))
  - Register + Combinational logic
- RTL design
  - Design behavior using high-level state machine (to be explained)
  - Remaining: Convert to sequential circuit

# Hardware Description Language (HDL)

- Represents hardware structure and behavior
  - Can be processed by computers (as a software)
- VHDL, Verilog-HDL, AHDL, myHDL, MODAL
- Used for simulation/synthesis

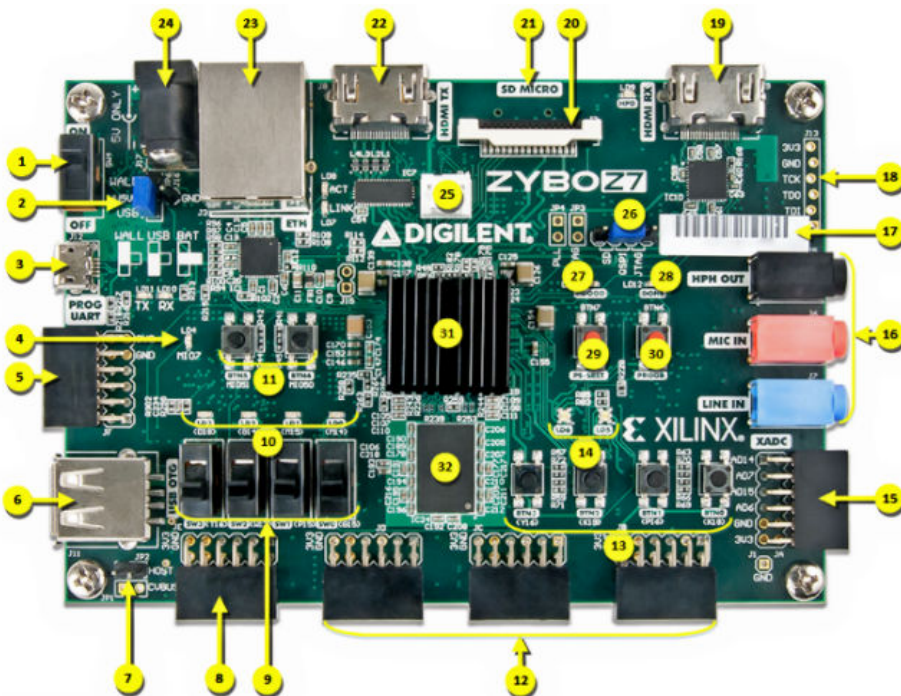
```
// the body of the 4-bit counter.  
always @(negedge clock or posedge clear)  
  if (clear)  
    qout <= 4'd0;  
  else  
    qout <= (qout + 1); // qout = (qout + 1) % 16;
```



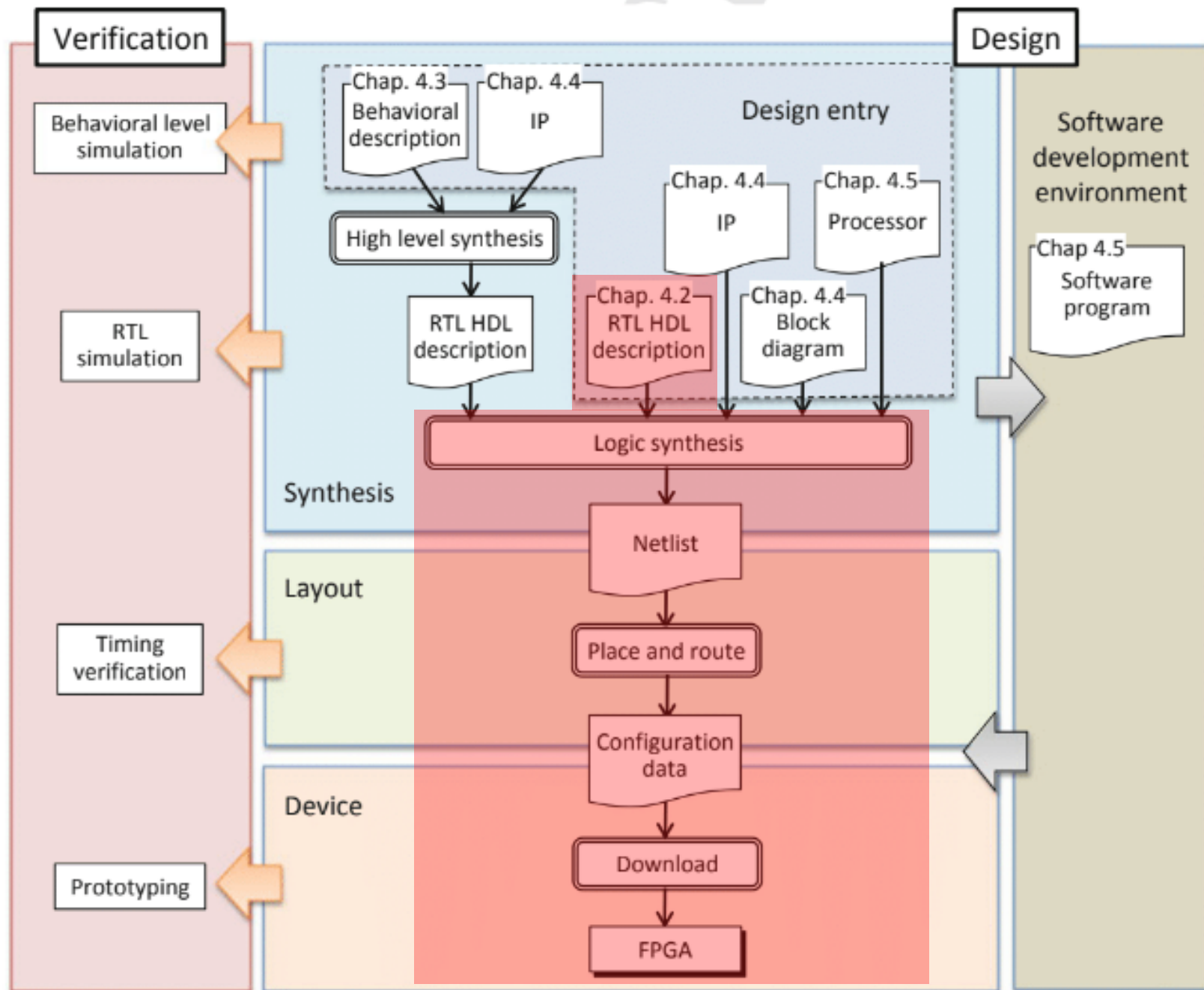
# HDL Design Flow with Simulation

# Target FPGA Board

- Digilent Zybo Z7-10/Z7-20
- Resource: <https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/start>
- Manual: [https://reference.digilentinc.com/\\_media/reference/programmable-logic/zybo-z7/zybo-z7\\_rm.pdf](https://reference.digilentinc.com/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf)



Call out	Description	Call out	Description	Call out	Description
1	Power Switch	12	High-speed Pmod ports *	23	Ethernet port
2	Power select jumper	13	User buttons	24	External power supply connector
3	USB JTAG/UART port	14	User RGB LEDs *	25	Fan connector (5V, three-wire) *
4	MIO User LED	15	XADC Pmod port	26	Programming mode select jumper
5	MIO Pmod port	16	Audio codec ports	27	Power supply good LED
6	USB 2.0 Host/OTG port	17	Unique MAC address label	28	FPGA programming done LED
7	USB Host power enable jumper	18	External JTAG port	29	Processor reset button
8	Standard Pmod port	19	HDMI input port	30	FPGA clear configuration button
9	User switches	20	Pcam MIPI CSI-2 port	31	Zynq-7000
10	User LEDs	21	microSD connector (other side)	32	DDR3L Memory
11	MIO User buttons	22	HDMI output port	* denotes difference between Z7-10 and Z7-20	



# Run Vivado (Not HLS!!)

```
# source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

```
# vivado &
```

File -> New Project

In "Create a New Vivado Project" window, Click "Next"

In "Project Name" window, set followings:

Project name: hello\_hw\_1

Project location: C:/FPGA/lec2 (Windows)

/root/FPGA/lec2 (Unix)

Then, project will be created at: C:/FPGA/lec2/hello\_hw\_1

(/root/FPGA/lec2/hello\_hw\_1)

Next, Click "Next"

# Settings (Cont'd)

In "Project Type", check "RTL Project", then "Next"

In "Add sources", just click "Next", and in "add constraints", click "Next"

In "Default Part", **carefully choose your FPGA!!**, then "Next"

Finally, in "New Project Summary", then click "Finish"

Select: ☒ Parts ☐ Boards

▼ Filter

Product category: General Purpose ▼ Speed grade: -1

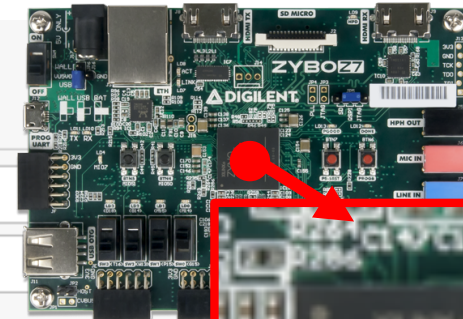
Family: Zynq-7000 ▼ Temp grade: All Remaining

Package: clg400 ▼

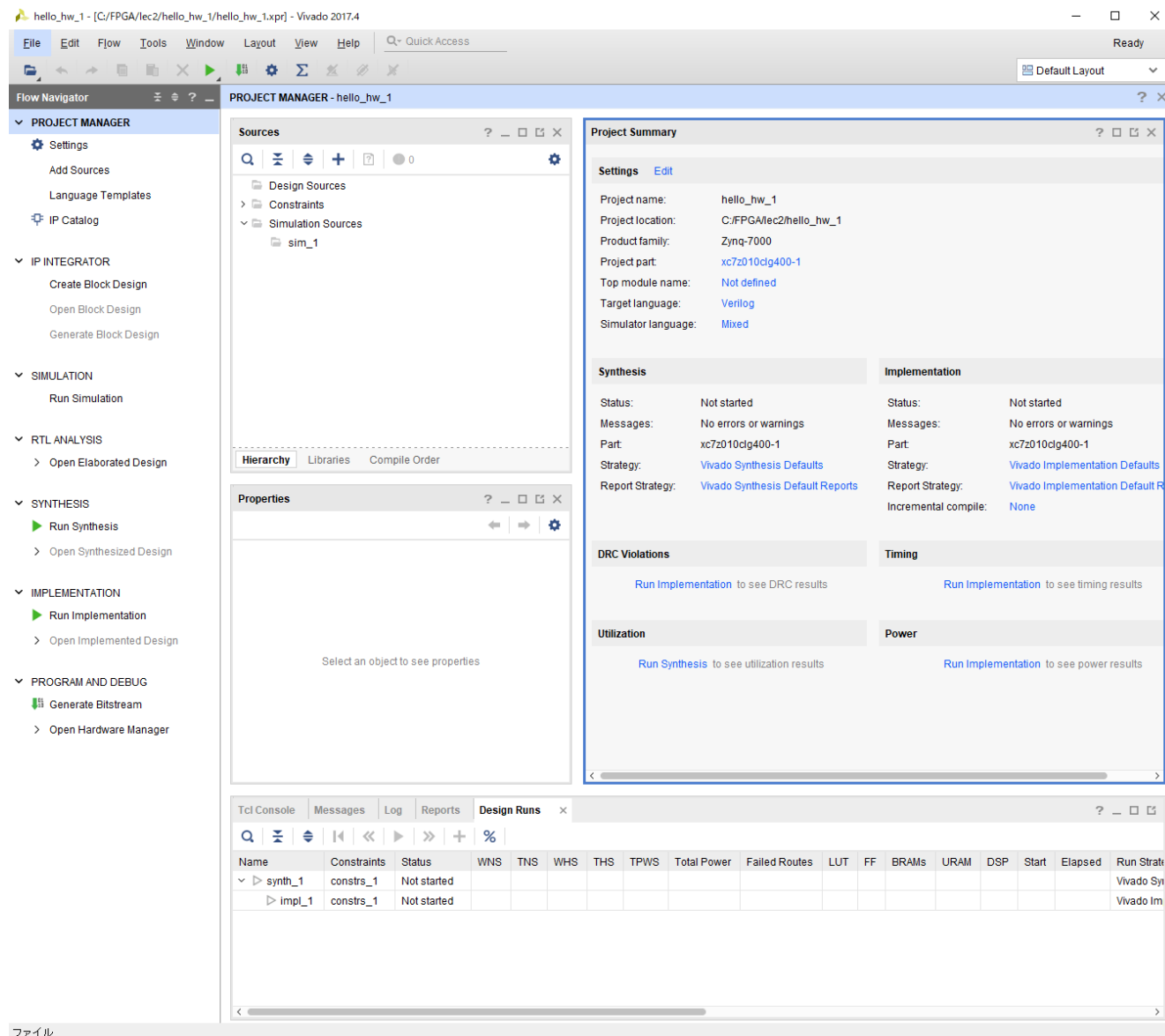
Reset All Filters

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers
<input checked="" type="radio"/> xc7z007sc1g400-1	400	100	14400	28800	50	0	66	0
<input checked="" type="radio"/> xc7z010clg400-1	400	100	17600	35200	60	0	80	0
<input type="radio"/> xc7z014sc1g400-1	400	125	40600	81200	107	0	170	0
<input type="radio"/> xc7z020clg400-1	400	125	53200	106400	140	0	220	0

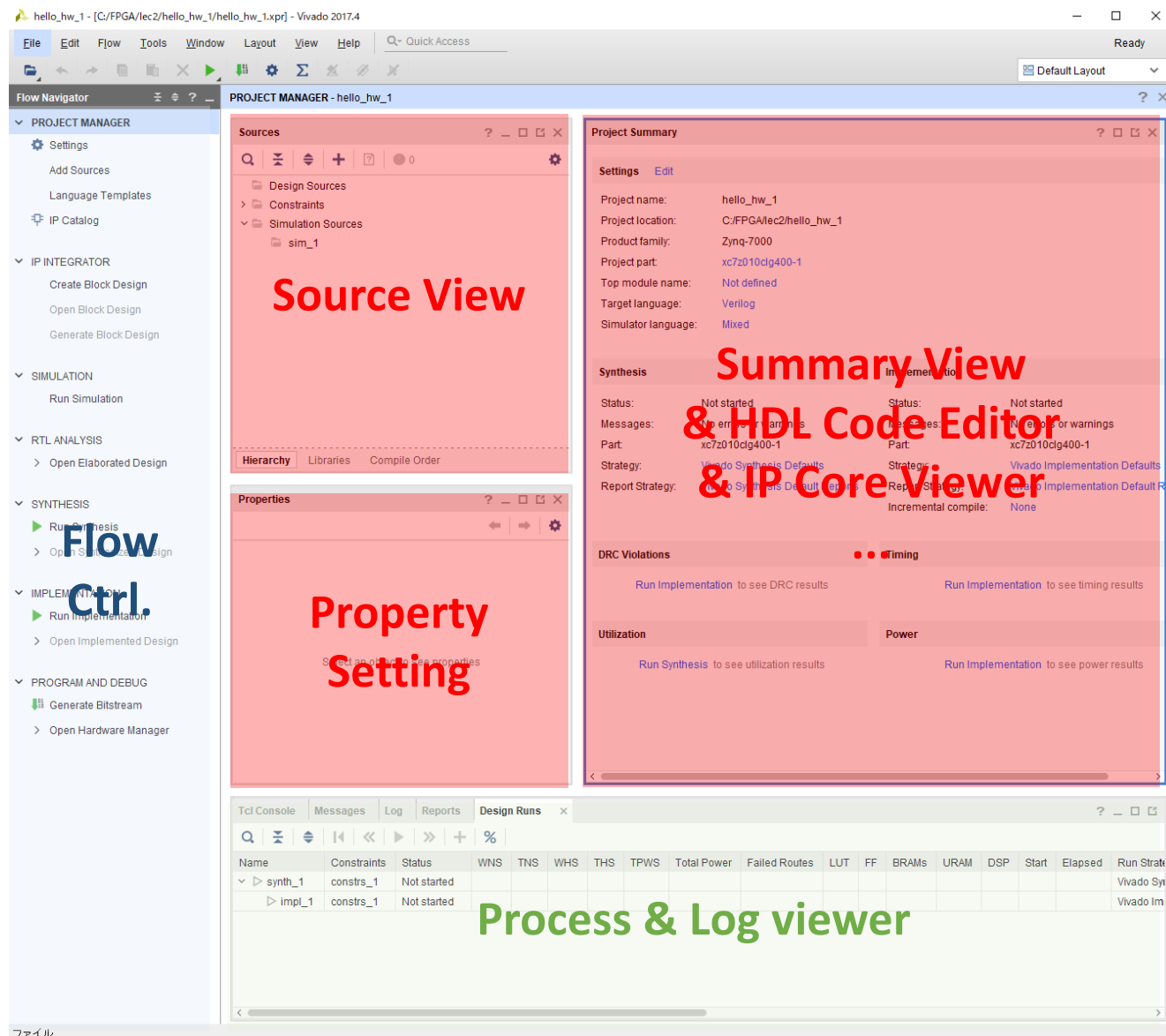


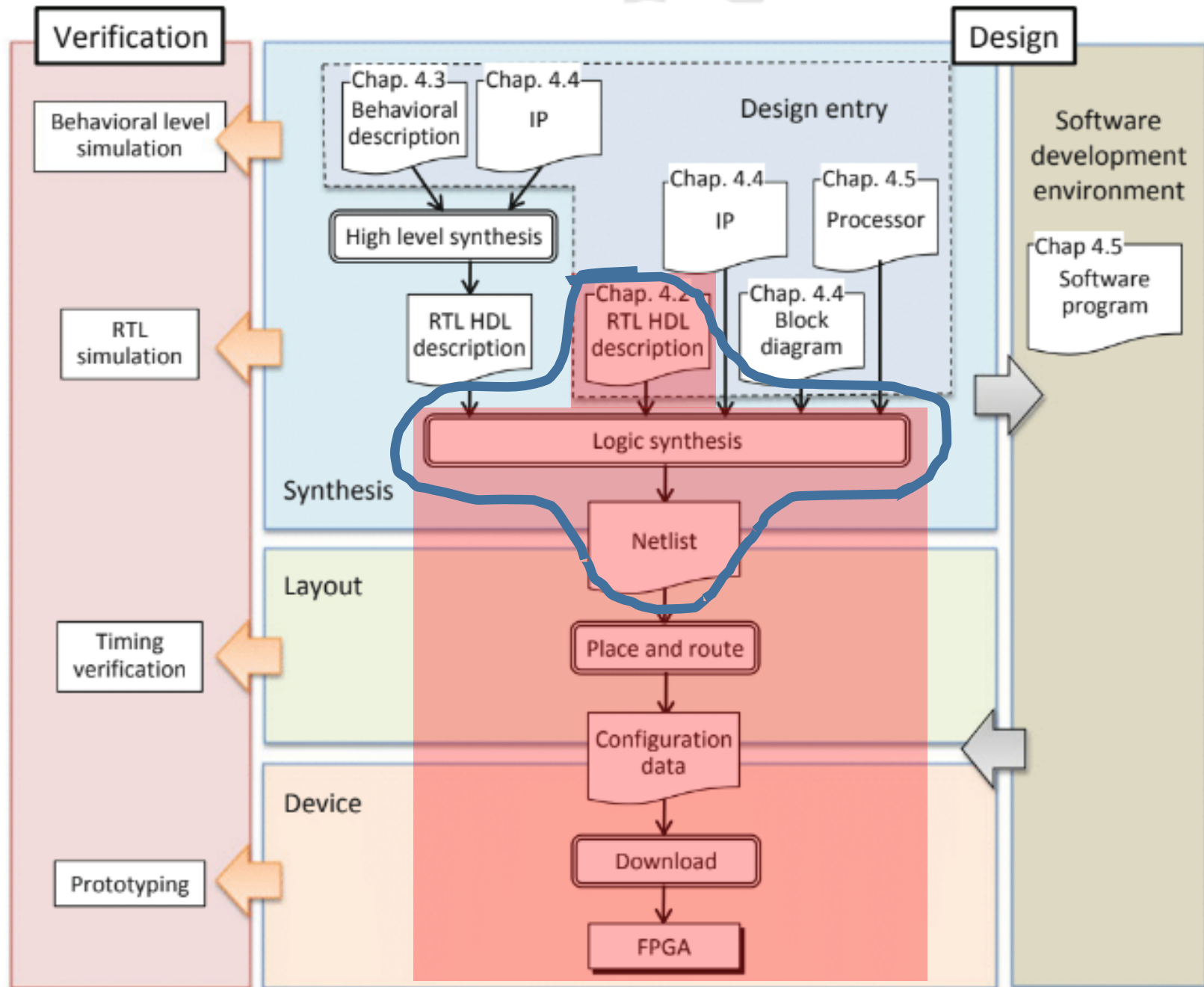
# Vivado 2017.4 GUI





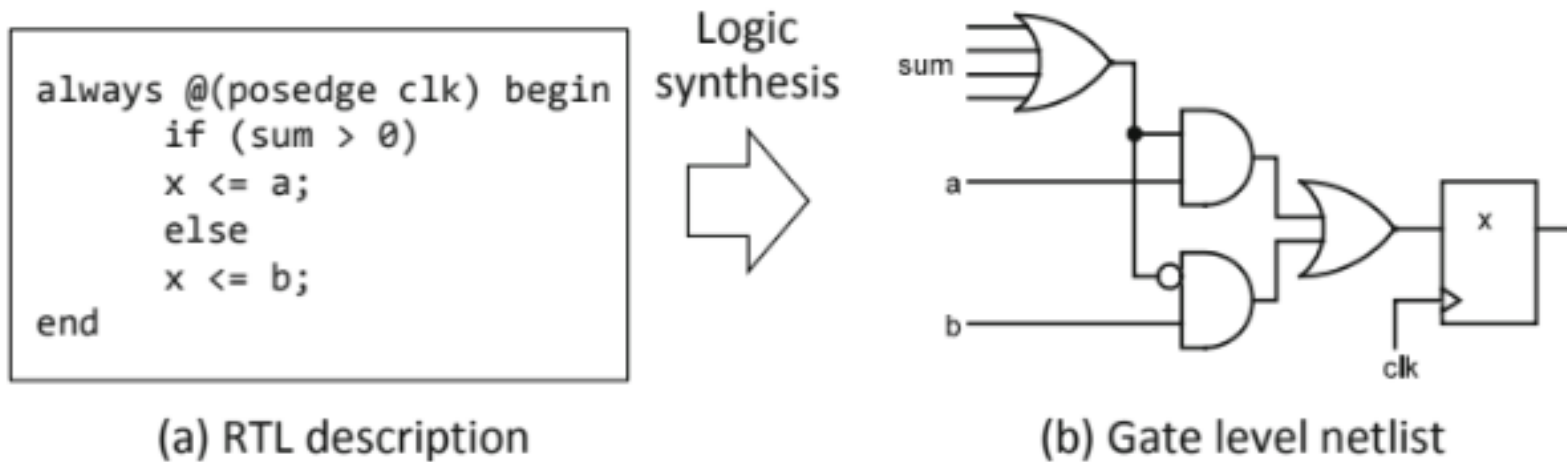
# Vivado 2017.4 IDE





# Logic Synthesis

- Synthesis of logic circuit (Mainly, sequential circuit) from RTL description
- Output: Netlist
- Netlist: Represents a set of logic elements such as primitive gates and flip-flops and their connections



# Create HDL Source File

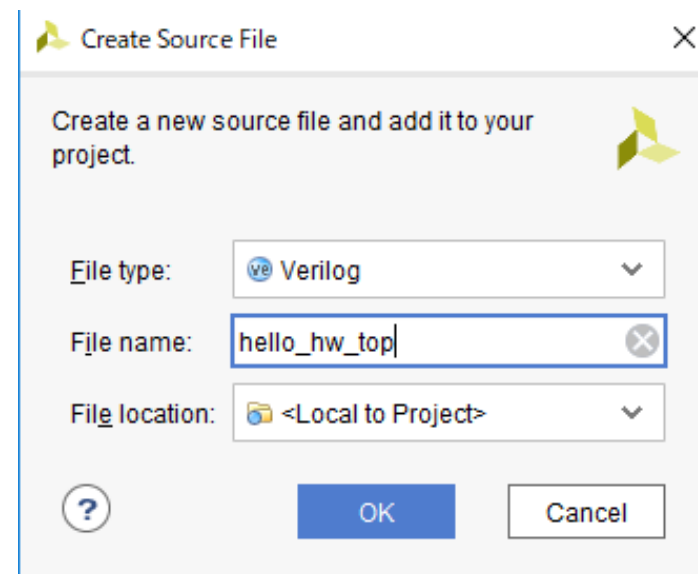
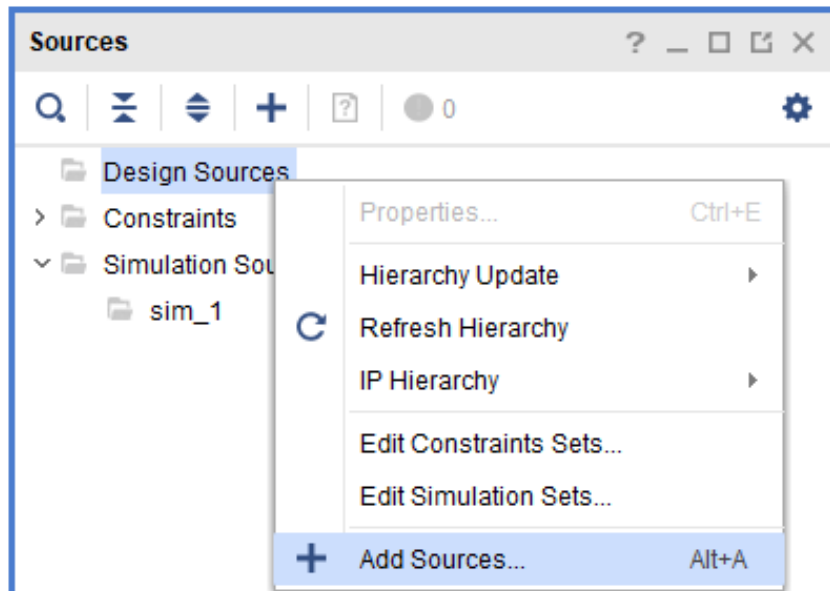
In "Sources", right click "Design Sources", then click "Add Sources..."

In "Add Sources", confirm "Add or create design sources", then "Next"

In "Add or Create Design Sources", click "Create File"

In "Create Source File", enter "hello\_hw\_top" to "File name:", then "OK"

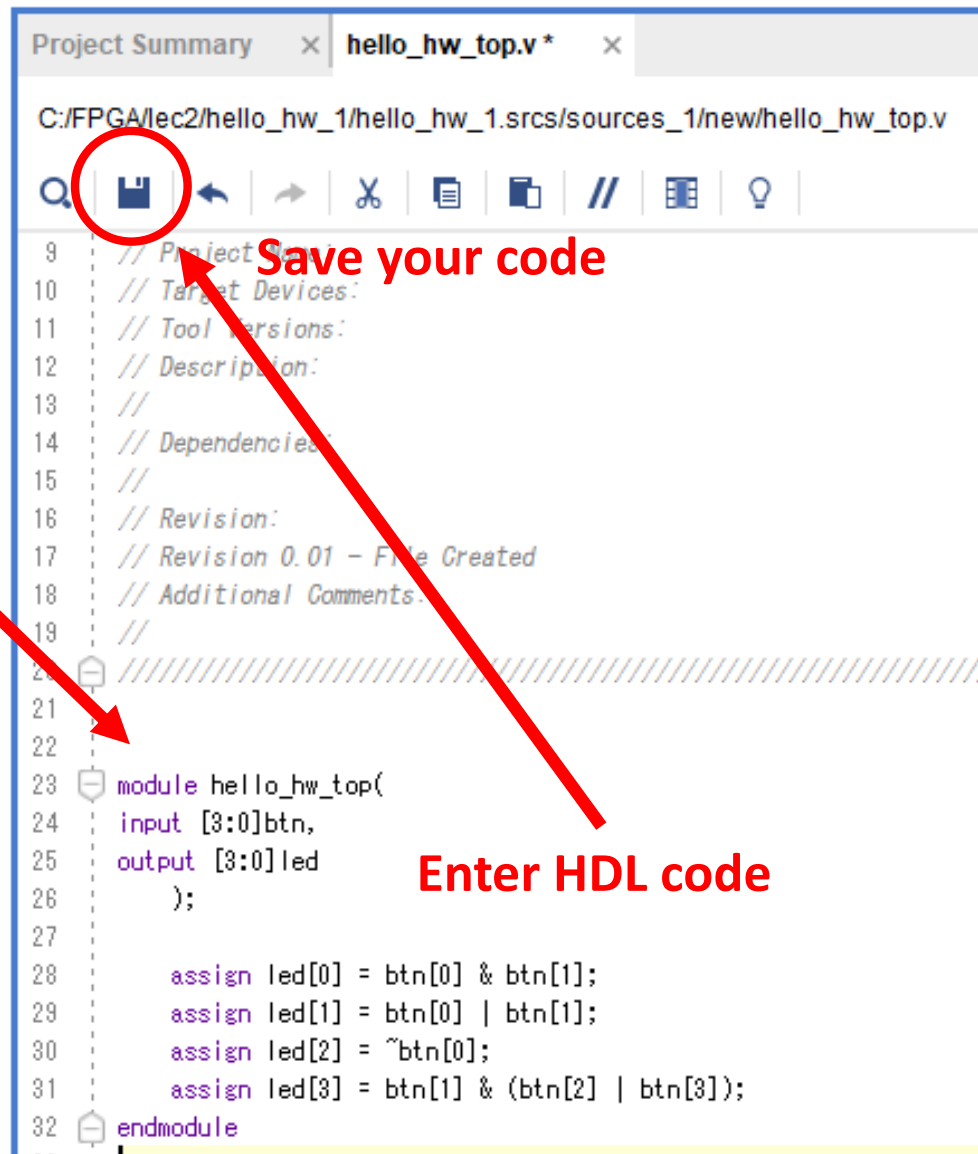
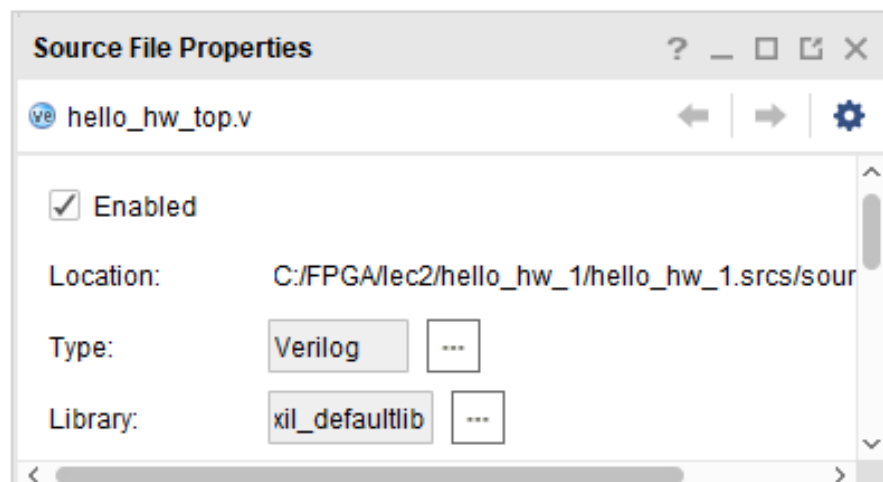
Back to "Add or Create Design Sources", then "Finish", "OK", and "Yes"



# Write Your First HDL



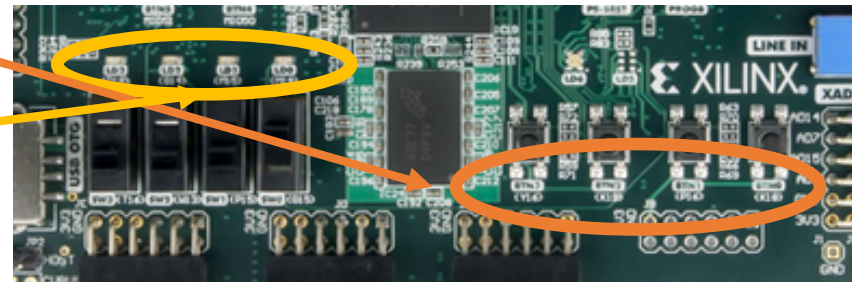
Double click  
"hello\_hw\_top"



# First Hardware

hardware must be  
specified by  
these words

```
module hello_hw_top(  
    input [3:0]btn,  
    output [3:0]led  
);
```



Assignment of a combinational logic

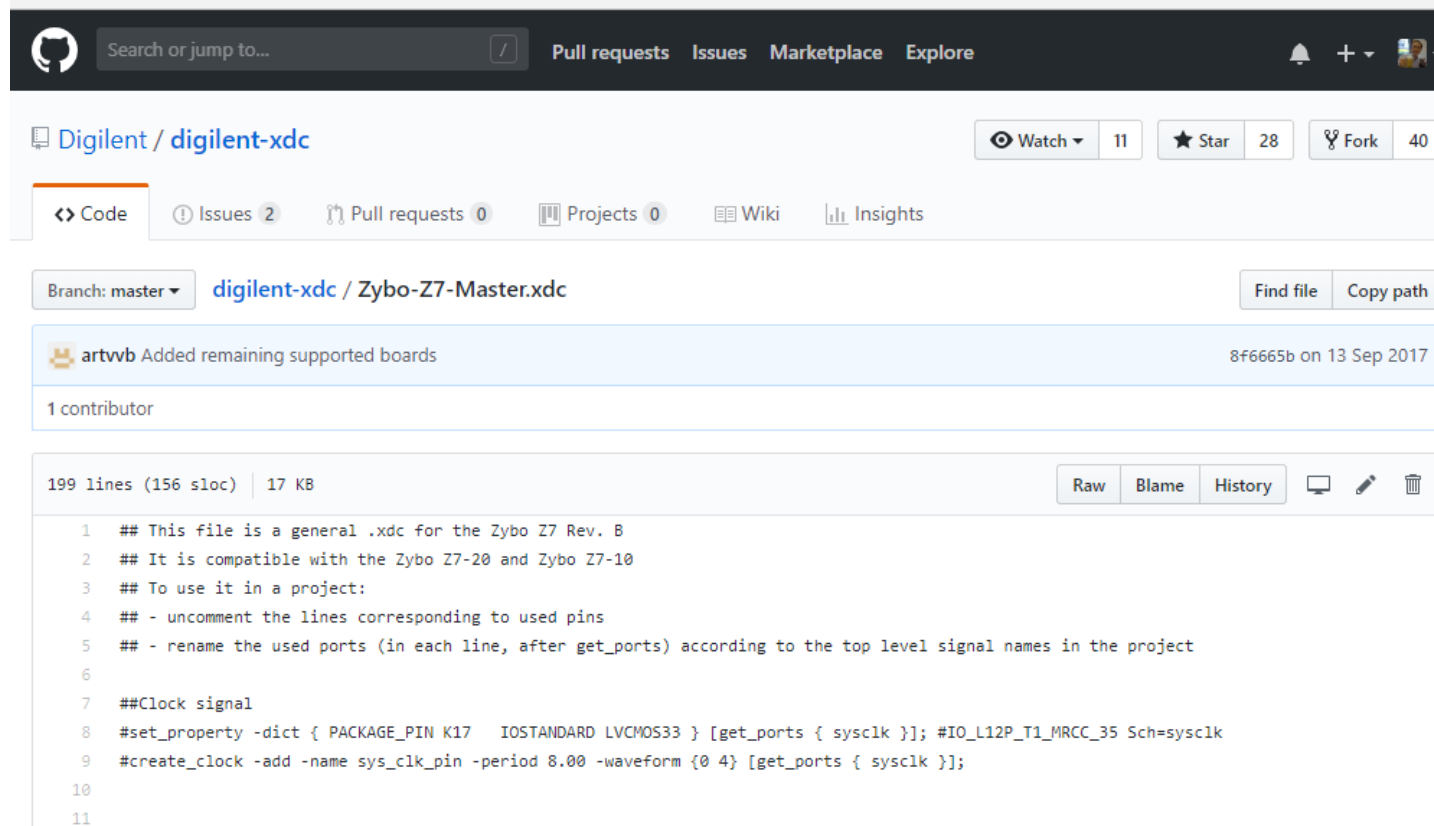
```
    assign led[0] = btn[0] & btn[1];  
    assign led[1] = btn[0] | btn[1];  
    assign led[2] = ~btn[0];  
    assign led[3] = btn[1] & (btn[2] | btn[3]);
```

```
endmodule
```

# Specify Constraint

- Pins (Location), and its direction(Input/Output), scandalization (LVDS, LVCMOS33,...)
- **Download** Digilent "Zybo-Z7-Master.xdc" from GitHub

<https://github.com/Digilent/digilent-xdc/blob/master/Zybo-Z7-Master.xdc>



The screenshot shows the GitHub interface for the repository `Digilent/digilent-xdc`. The file `Zybo-Z7-Master.xdc` is selected, and the commit history shows a commit by `artwvb` on 13 Sep 2017. The file content is displayed in a code editor with line numbers 1 through 11.

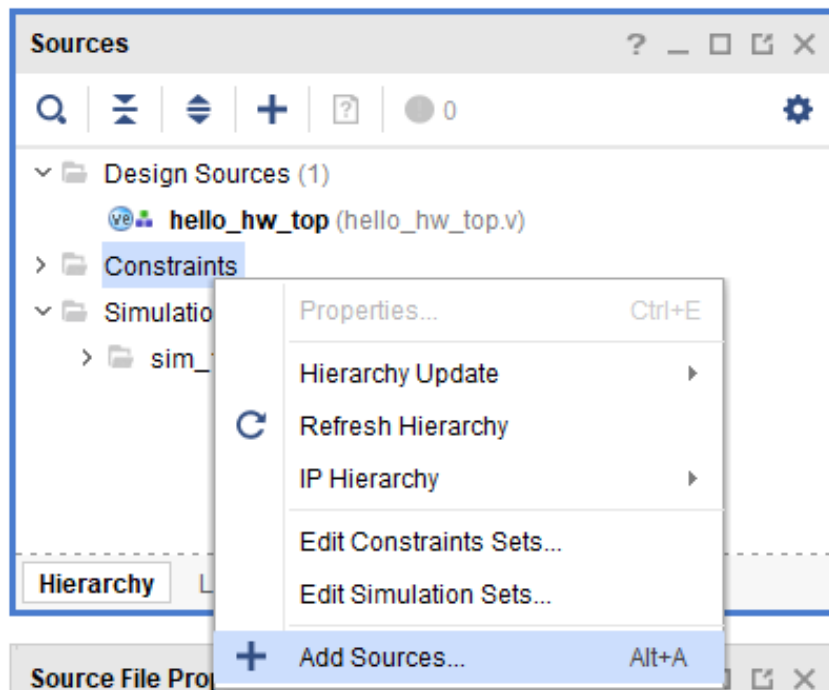
```
1  ## This file is a general .xdc for the Zybo Z7 Rev. B
2  ## It is compatible with the Zybo Z7-20 and Zybo Z7-10
3  ## To use it in a project:
4  ## - uncomment the lines corresponding to used pins
5  ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
6
7  ##Clock signal
8  #set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports { sysclk }]; #IO_L12P_T1_MRCC_35 Sch=sysclk
9  #create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { sysclk }];
10
11
```

# Add Constraint File

Right click on "Constraints", and select "Add Sources..."

Make sure "Add or create constraints", then "Next"

In "Add or Create Constraints", click "Add Files", then load "Zybo-Z7-Master.xdc" from Digilent's GitHub repository, and "Finish"





# Modify Constraint File

**Sources**

- Design Sources (1)
  - hello\_hw\_top (hello\_hw\_top.v)
- Constraints (1)
  - constrs\_1 (1)
    - Zybo-Z7-Master.xdc**
- Simulation Sources (1)
  - sim\_1 (1)

**Source File Properties**

Zybo-Z7-Master.xdc

**Project Summary** | hello\_hw\_top.v \* | **Zybo-Z7-Master.xdc \***

C:/Users/hirok/Desktop/Zybo-Z7-Master.xdc

**Save your constraint**

```
16 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { sw[3,
17
18
19 ##Buttons
20 set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMOS33 } [get_ports { bt
21 set_property -dict { PACKAGE_PIN P16 IOSTANDARD LVCMOS33 } [get_ports { bt
22 set_property -dict { PACKAGE_PIN K19 IOSTANDARD LVCMOS33 } [get_ports { bt
23 set_property -dict { PACKAGE_PIN Y16 IOSTANDARD LVCMOS33 } [get_ports { bt
24
25
26 ##LEDs
27 set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { le
28 set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { le
29 set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { le
30 set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { le
31
32
```

**Double click .xdc file**

**Comment of "btn"s and "led"s**

# Do Logic Synthesis

Flow Navigator

- PROJECT MANAGER
  - Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP INTEGRATOR
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- SIMULATION
  - Run Simulation
- RTL ANALYSIS
  - Open Elaborated Design
- SYNTHESIS**
  - Run Synthesis**
  - Open Synthesized Design
- IMPLEMENTATION
  - Run Implementation
  - Open Implemented Design

PROJECT MANAGER - hello\_hw\_1

Sources

- Design Sources (1)
- Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

- ☒ Launch runs on local host: Number of jobs: 4
- ☐ Generate scripts only
- ☐ Don't show this dialog again

OK

Synthesis Completed

Synthesis successfully completed.

Next

- ☒ Run Implementation
- ☐ Open Synthesized Design
- ☐ View Reports
- ☐ Don't show this dialog again

OK Cancel

# Schematic View

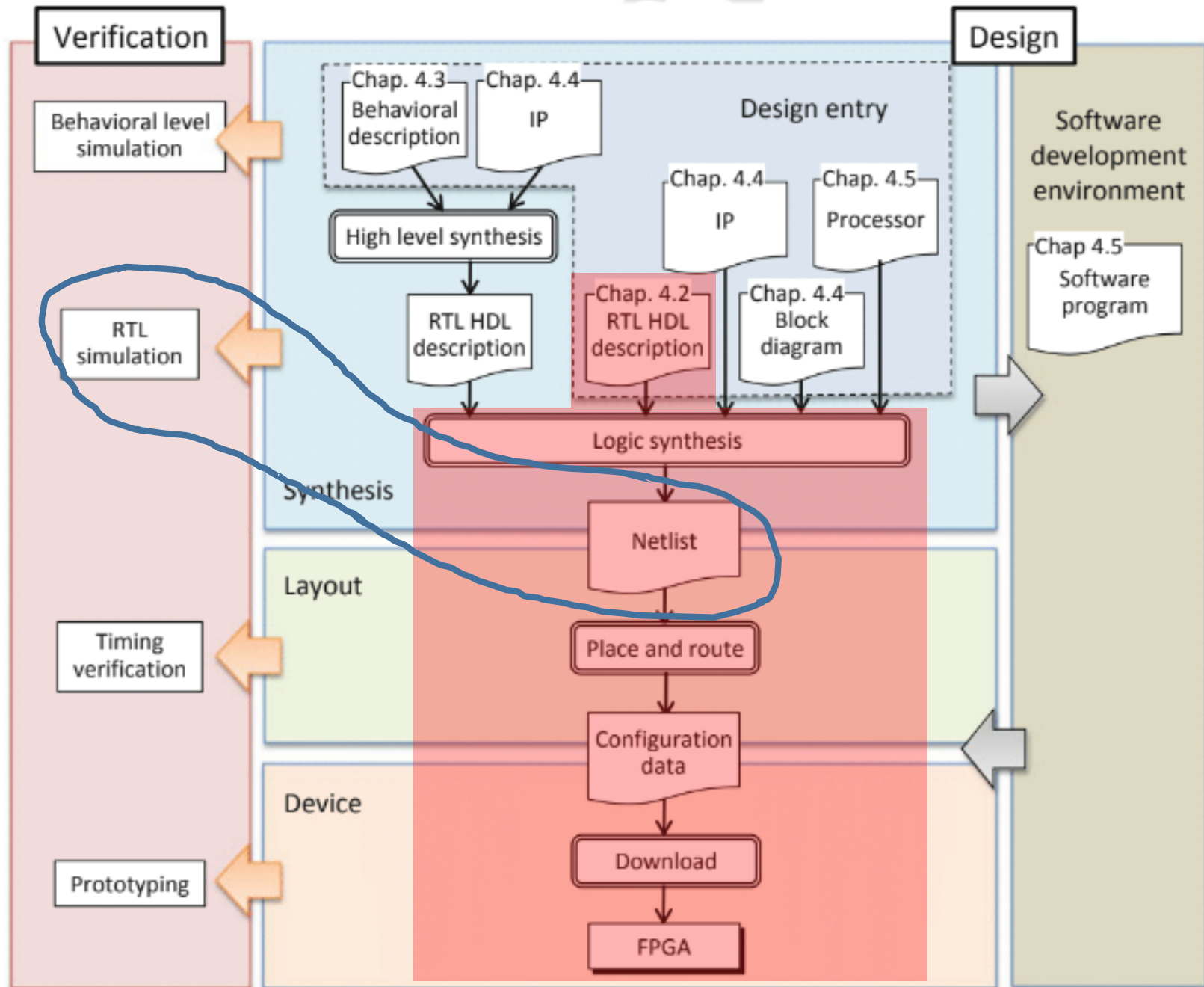
The image shows the Xilinx IDE interface with the Schematic View selected in the left-hand menu. The main window displays a schematic diagram of a 4-bit LED driver circuit. The circuit includes four input buffers (btn\_IBUF[0:3]\_inst), four LUTs (LUT1, LUT2, LUT3, and LUT4), and four output buffers (led\_OBUF[0:3]\_inst). The LUTs are configured to implement a 4-bit counter or similar logic, with LUT3 highlighted by a red circle.

A "Specify LUT Equation" dialog box is open, showing the LUT equation for 'led\_OBUF[3]\_inst\_i\_1'. The equation is  $O = I0 \& I1 + I0 \& I2$ . The dialog also lists legal operators and examples. A red arrow points from the equation input field to the "Truth Table" tab in the Cell Properties window below.

The Cell Properties window for 'led\_OBUF[3]\_inst\_i\_1' is shown, displaying the truth table for the LUT equation. The truth table is as follows:

I2	I1	I0	O = I0 & I1 + I0 & I2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The "Truth Table" tab is highlighted with a red circle. The "Schematic" view is also highlighted with a red circle in the left-hand menu.



# RTL Simulation

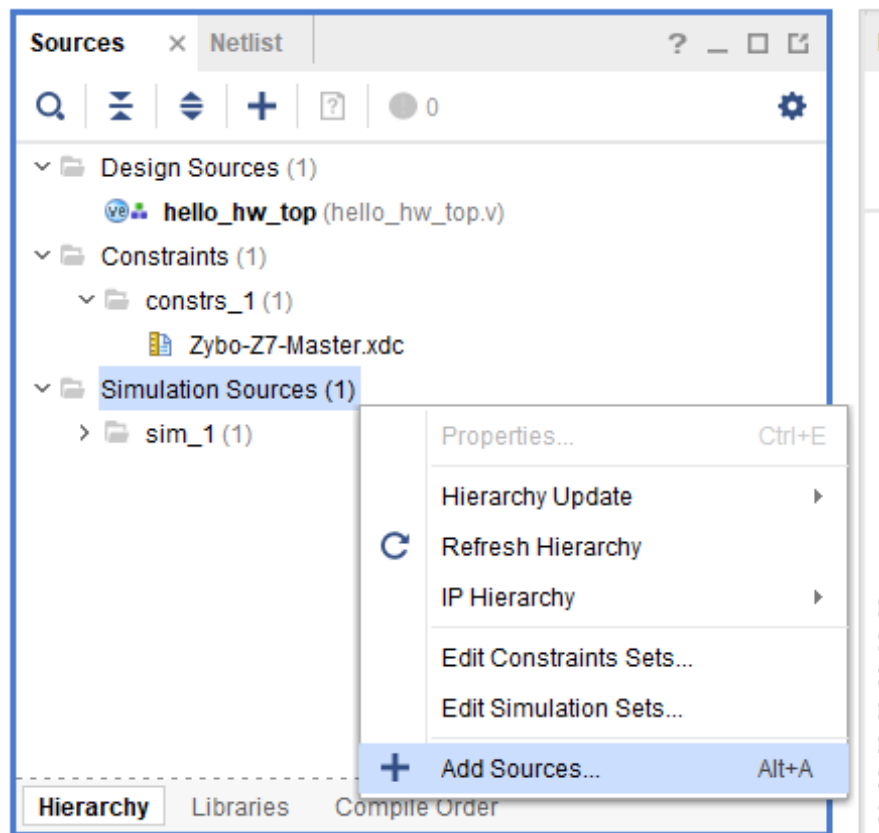
- Confirm the designed circuit meets the required specifications (or not)
- Simulation model
  - Behavior of RTL description
    - Verify correctness of functions and operations
  - Synthesized netlist
    - Timing and delay of signal change based on the delay time of the assigned logic/memory element, analyze the state of signal transition with propagation delay
    - Allows analysis of power consumption
  - Post placement and routing netlist
    - Estimate wiring delay time and enable the most detailed timing /power analysis

# Set Simulation Source

Make sure "Sources tab" is selected

Right click on "Simulation Sources", then select "Add Sources..."

In "Add Sources", confirm "Add or create simulation sources", then "Next"

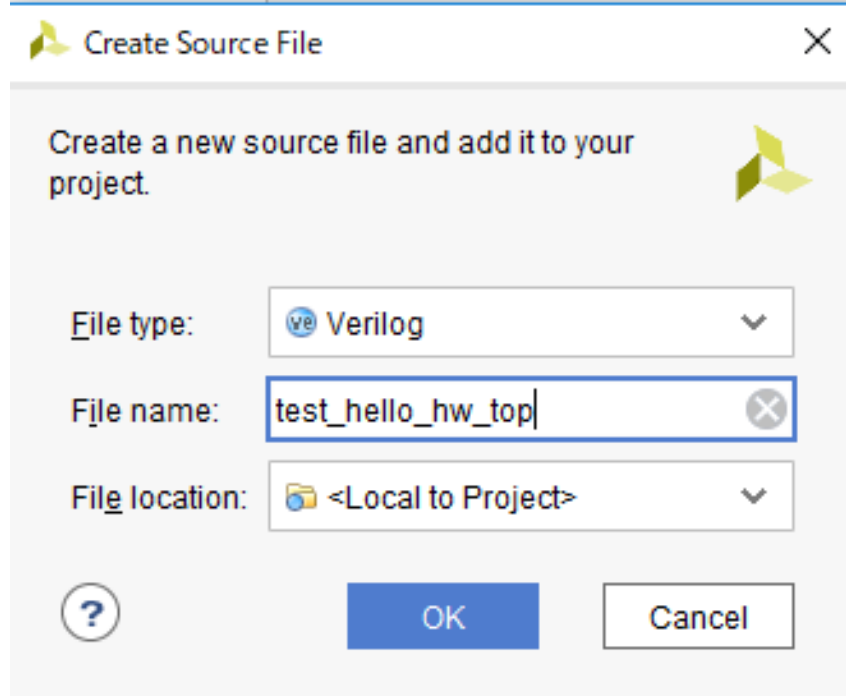


# Cont'd

In "Add or Create Simulation Sources", click "Create File"

In "Create Source File", enter "test\_hello\_hw\_top" in "File name", then "OK", and "Finish"

In Define module, just "OK"



# Write TestBench

The image shows the Vivado IDE interface. On the left is the 'Sources' window with the 'Netlist' tab selected. It displays a project hierarchy: 'Design Sources (1)' containing 'hello\_hw\_top (hello\_hw\_top.v)'; 'Constraints (1)' containing 'Zybo-Z7-Master.xdc'; and 'Simulation Sources (2)' containing 'sim\_1 (2)' which includes 'hello\_hw\_top (hello\_hw\_top.v)' and 'test\_hello\_hw (test\_hello\_hw\_top.v)'. A red arrow points from 'test\_hello\_hw' to the code editor on the right. The code editor shows the testbench code for 'test\_hello\_hw'. A red circle highlights the 'Save' icon in the toolbar, with the text 'Save your testbench' next to it. The code itself is as follows:

```
1 module test_hello_hw;
2   reg [3:0] test_in;
3   wire [3:0] test_out;
4
5   hello_hw_top hello_hw_top_inst(
6     .btn( test_in),
7     .led( test_out)
8   );
9
10  initial begin
11    test_in = 4'b0000;
12    #10 test_in = 4'b0001;
13    #10 test_in = 4'b0010;
14    #20 test_in = 4'b0011;
15  end
16 endmodule
```

The last line of the code, 'endmodule', is highlighted in yellow, with the text 'Write testbench' below it.



# Run Behavioral Simulation

The screenshot shows the Vivado IDE interface. On the left, the 'SYNTHESIS' tab is selected in the Project Navigator. A context menu is open over the 'Run Synthesis' button, with 'Run Behavioral Simulation' highlighted. A red arrow points from this menu item to the simulation results window. The simulation results window displays a table of signal values and a corresponding timing diagram.

Name	Value
test_in[3:0]	3
test_in[3]	0
test_in[2]	0
test_in[1]	1
test_in[0]	1
test_out[3:0]	3
test_out[3]	0
test_out[2]	0
test_out[1]	1
test_out[0]	1

The timing diagram on the right shows the signal values over time. A yellow vertical line indicates the current simulation time at 45.400 ns. The signal values are: test\_in[3:0] = 3, test\_out[3:0] = 3, test\_in[3] = 0, test\_in[2] = 0, test\_in[1] = 1, test\_in[0] = 1, test\_out[3] = 0, test\_out[2] = 0, test\_out[1] = 1, test\_out[0] = 1.

```
assign led[0] = btn[0] & btn[1];  
assign led[1] = btn[0] | btn[1];  
assign led[2] = ~btn[0];  
assign led[3] = btn[1] & (btn[2] | btn[3]);
```

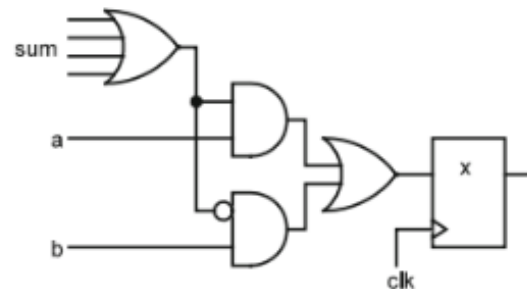
**Verify your description with simulation result**

# Simulation Level

```
always @(posedge clk) begin
    if (sum > 0)
        x <= a;
    else
        x <= b;
end
```

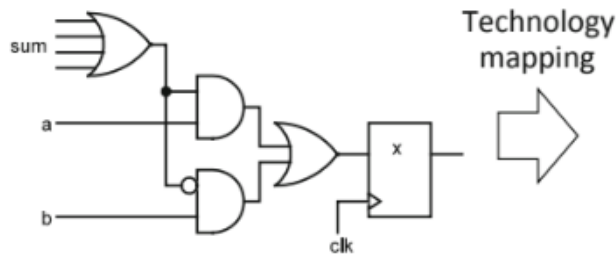
(a) RTL description

Logic  
synthesis



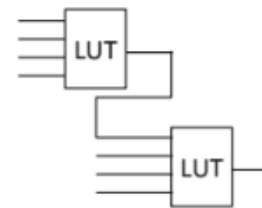
(b) Gate level netlist

Behavioral Simulation:  
Sim. Speed is faster,  
but, low accurate



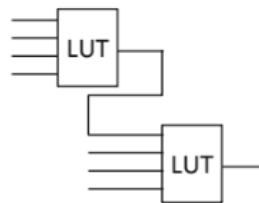
(b) Gate level netlist

Technology  
mapping

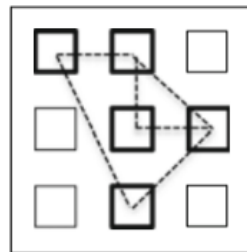


(c) LUT level netlist

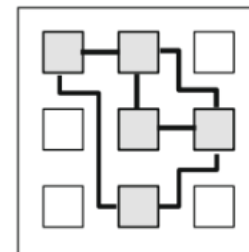
Post-Synthesis:  
Sim. Speed is middle,  
but, relatively accurate



(a) LUT level netlist

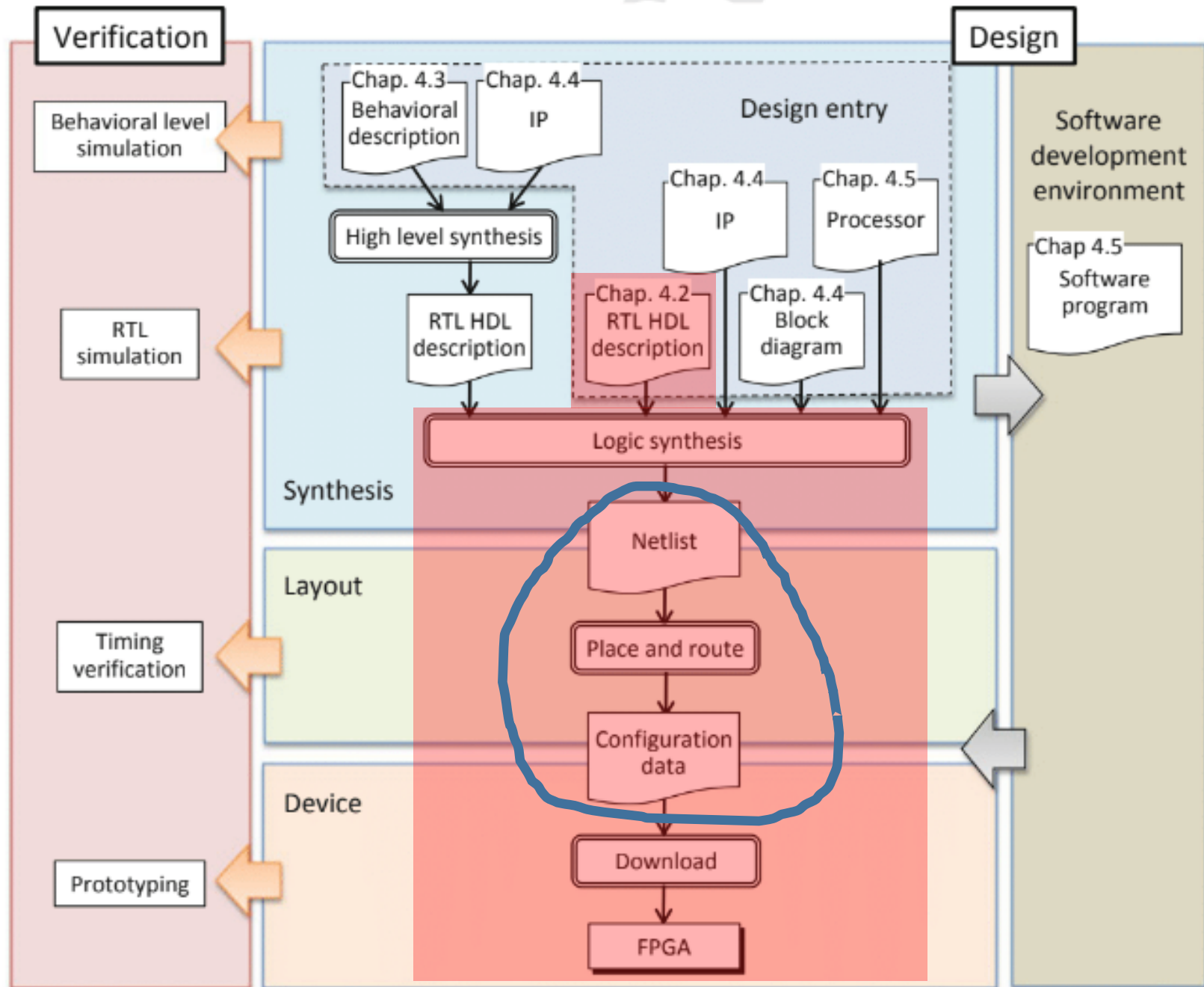


(b) Place



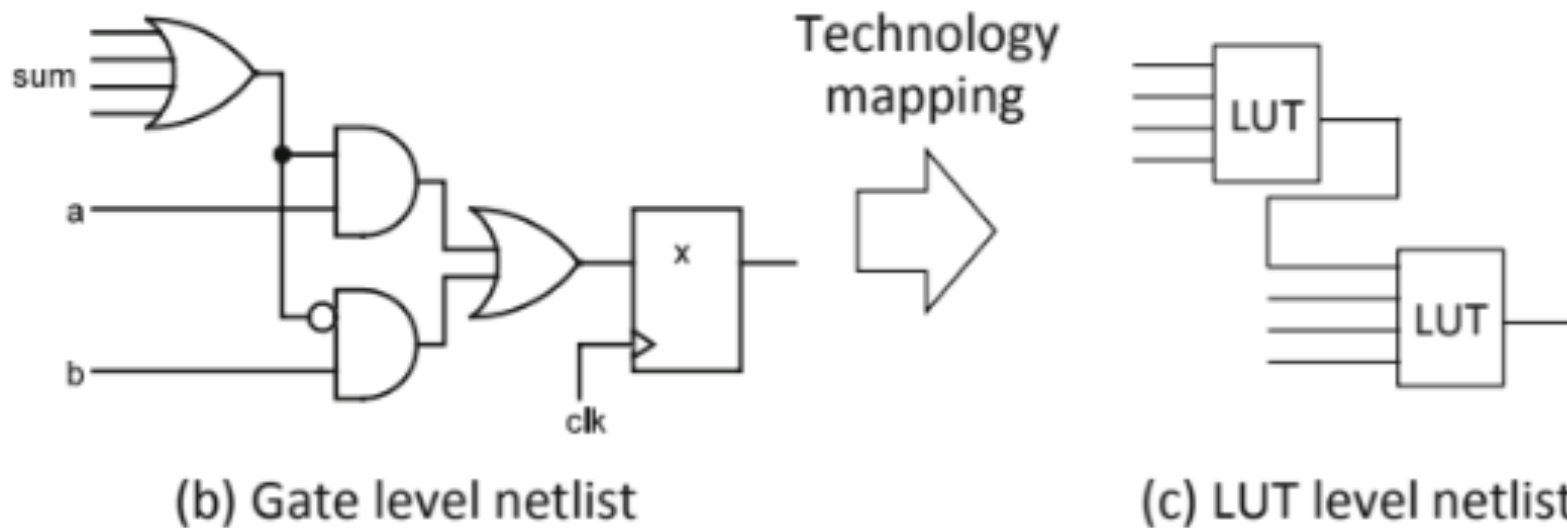
(c) Route

Post-Implementation:  
Sim. Speed is slower,  
but, more accurate



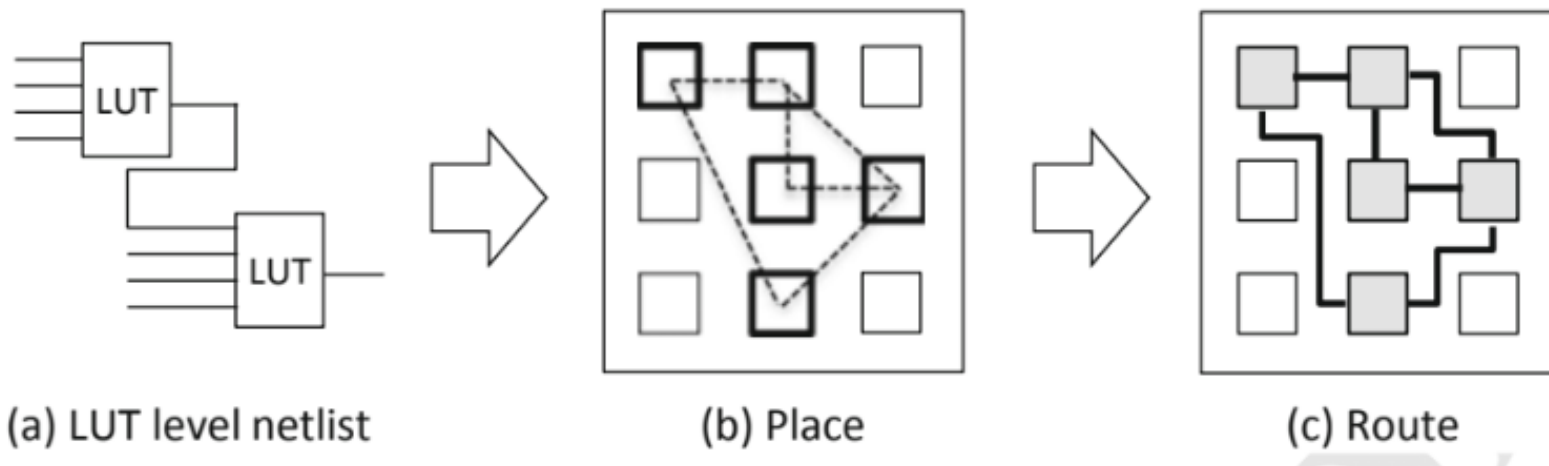
# Technology Mapping

- Assign the synthesized netlist to the logic elements of the FPGA
- Rewritable logic elements called LUT(Look-Up Table) are used



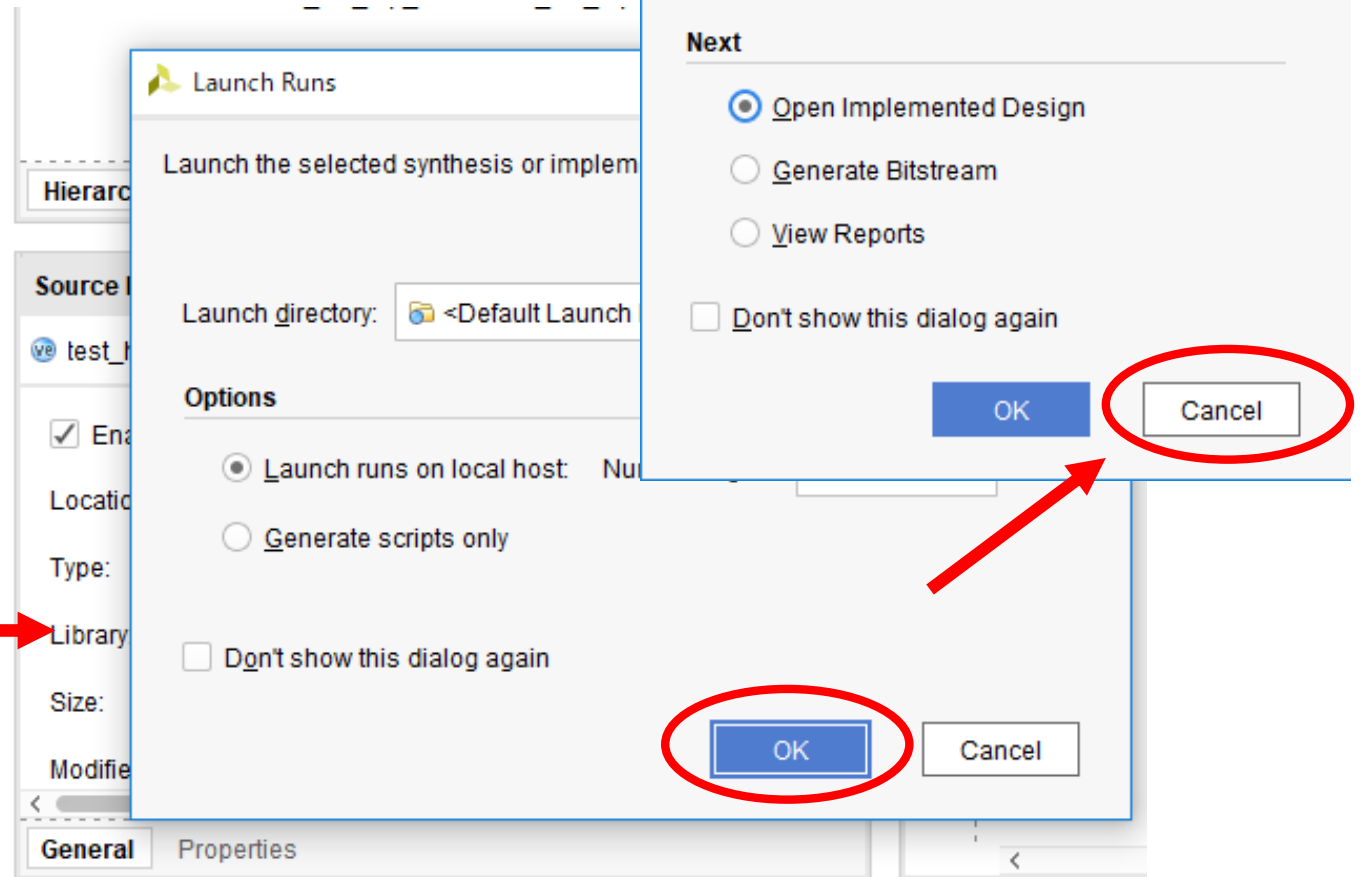
# Place-and-Routing

- Assign LUT-based netlist to logical resources and routing resources on FPGA
- Generally, after allocate logical resources and then perform routing



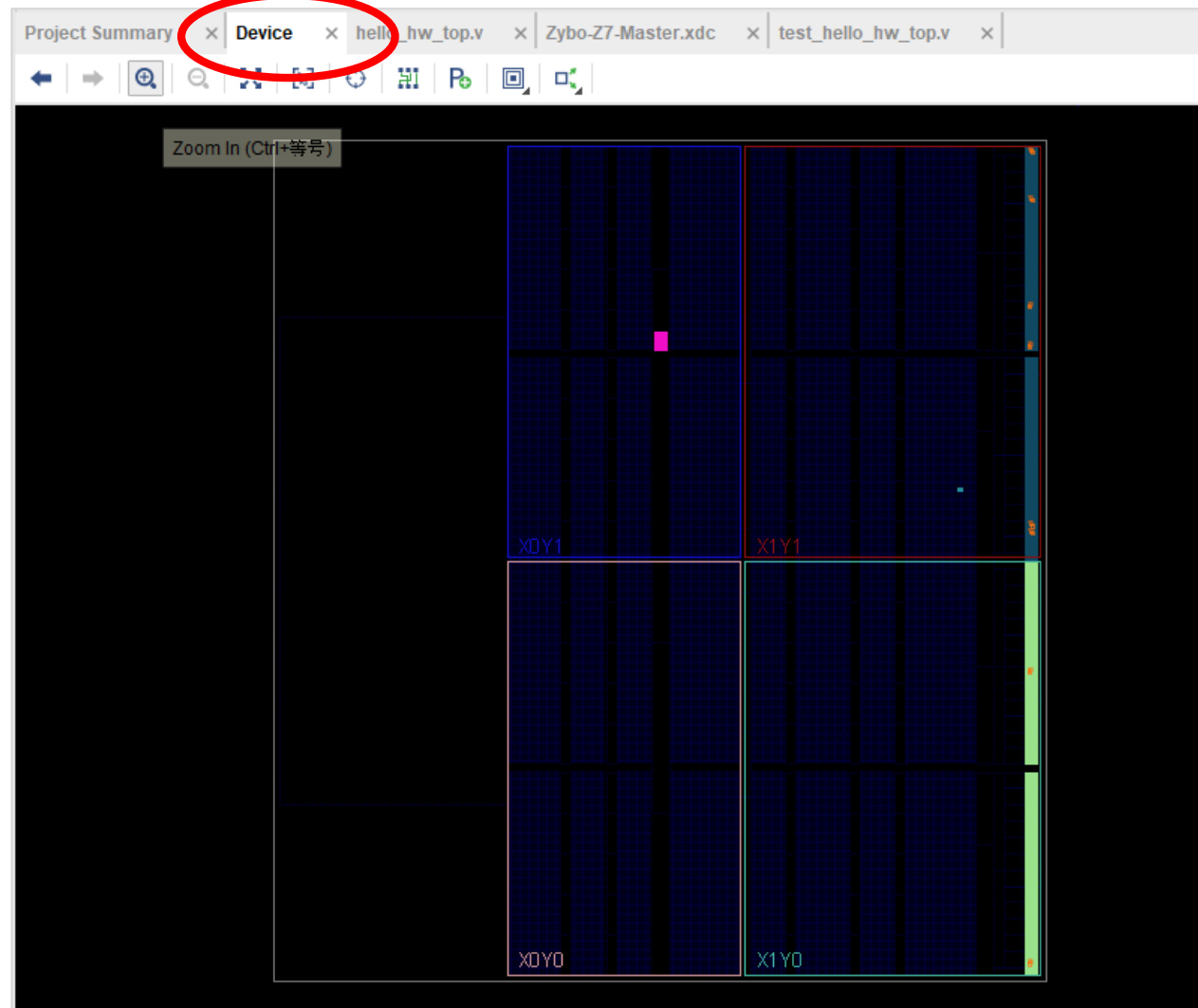
# Launch Place-and-Routing

- ✓ SIMULATION
  - Run Simulation
- ✓ RTL ANALYSIS
  - > Open Elaborated Design
- ✓ SYNTHESIS
  - ▶ Run Synthesis
  - > Open Synthesized Design
- ✓ IMPLEMENTATION
  - ▶ Run Implementation
  - > Open Implemented Design
- ✓ PROGRAM AND DEBUG
  - Generate Bitstream



# Investigate FPGA Architecture

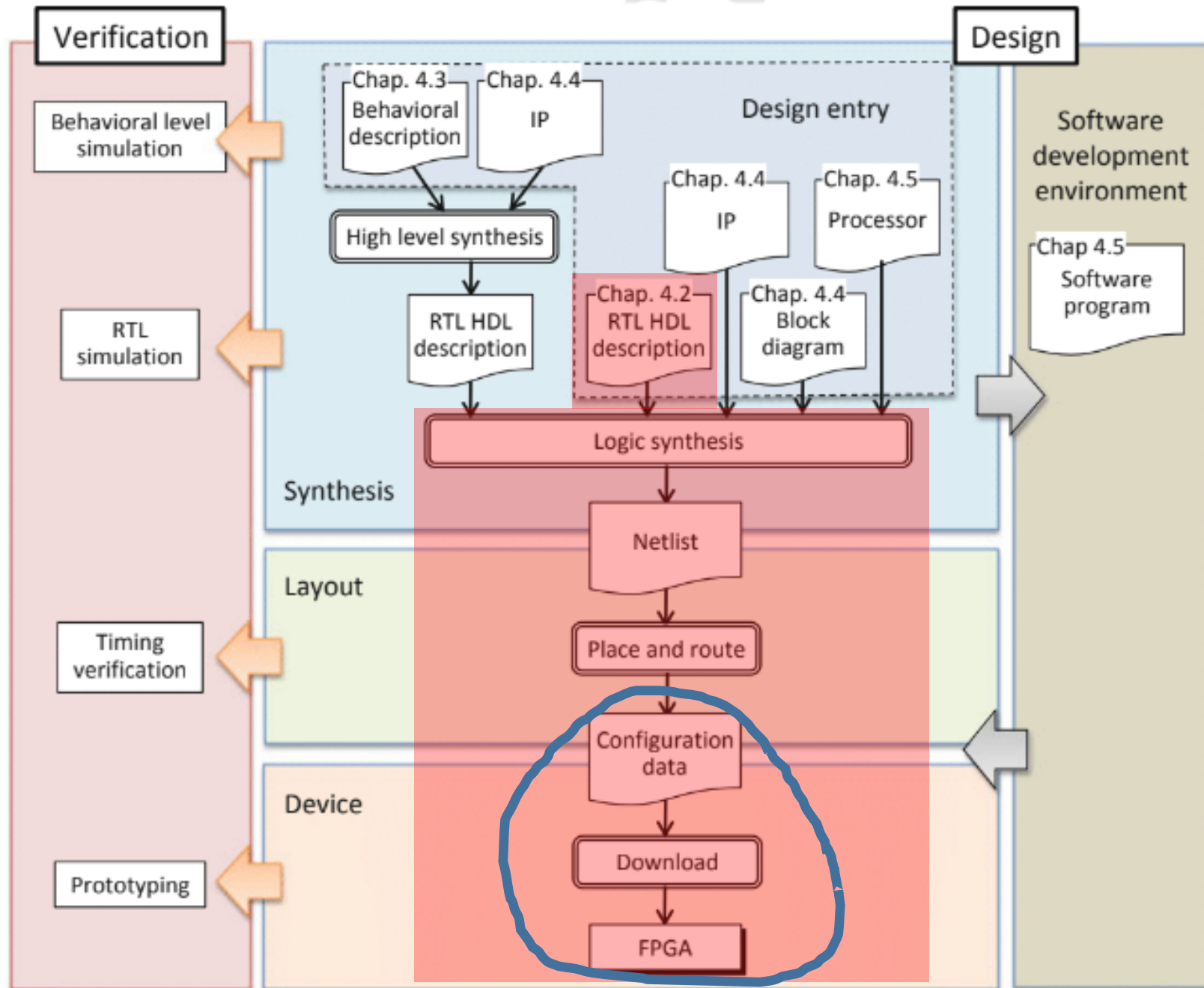
Device



# Programming

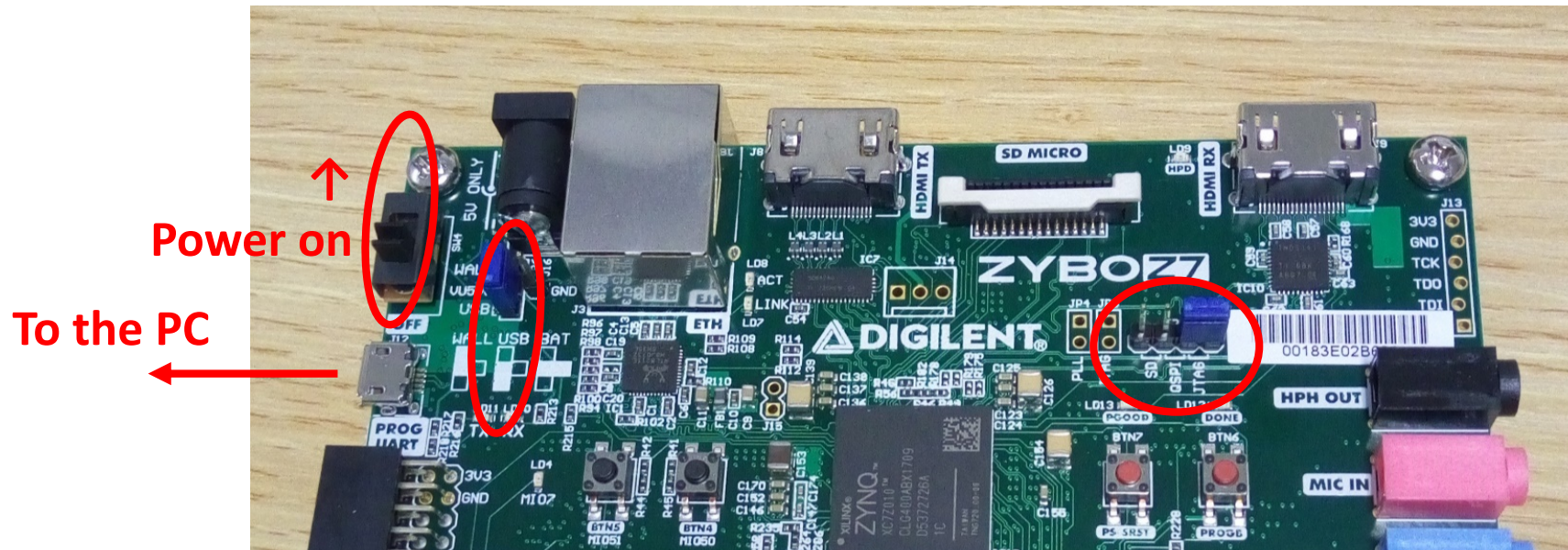
- The completed circuit is converted as bitstream (configuration data or program file) for programming the logical and wiring resources in the FPGA
- Send bitstream to the FPGA using the programmer
  - Direct writing by Joint Test Action Group (JTAG) to program nonvolatile memory (Flash, EEPROM)
  - Circuit configuration is erased due to power off or reset of FPGA
- After writing, the board can be started as a bootable unit



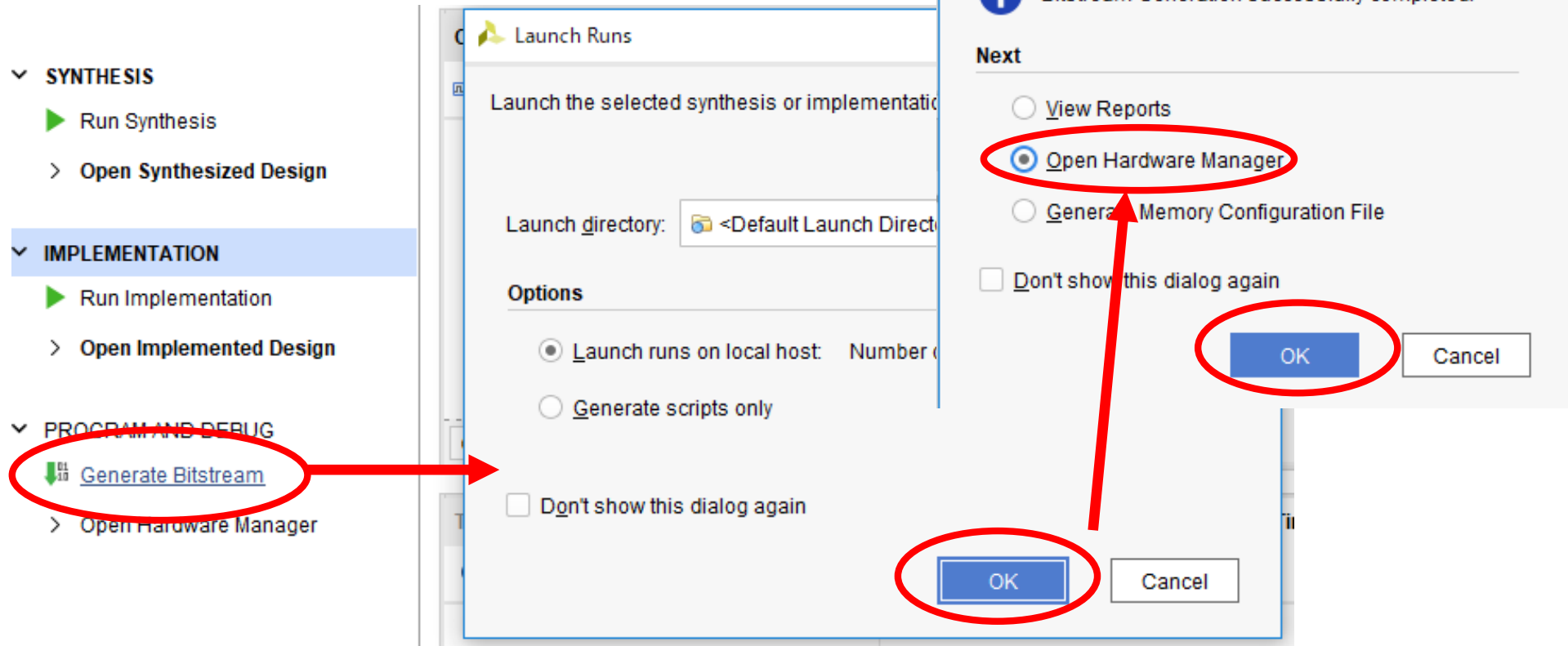


# Setup Zybo-Z7 Board

- Make sure the FPGA board is not connect to your PC
- Set jumper pins as shown in Photograph
  - USB-power and JTAG mode
- Connect to the PC via mini-USB cable
- Turn on your FPGA board

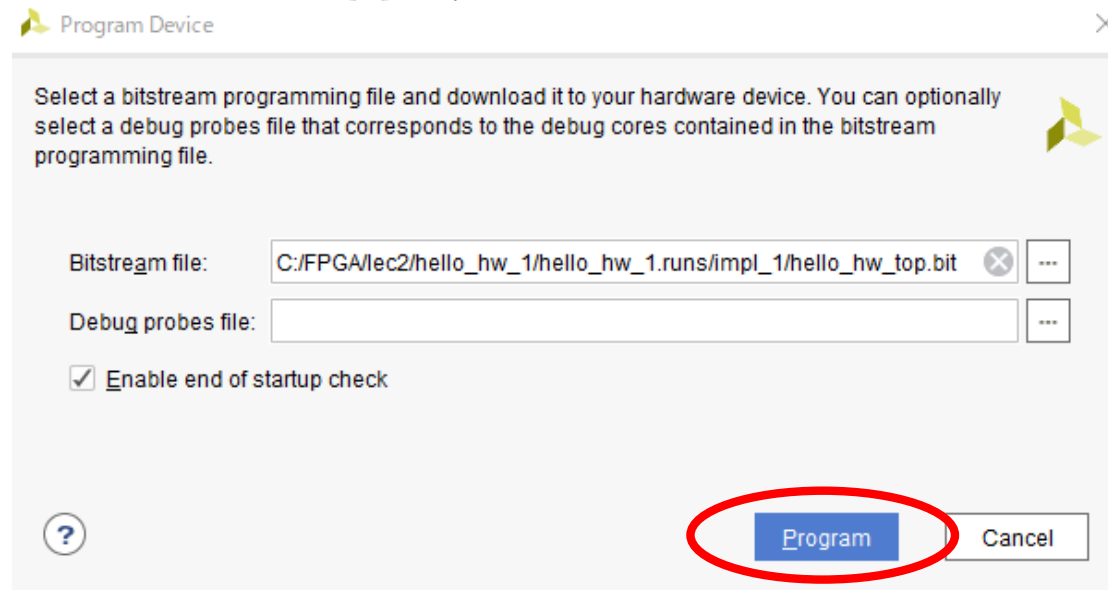
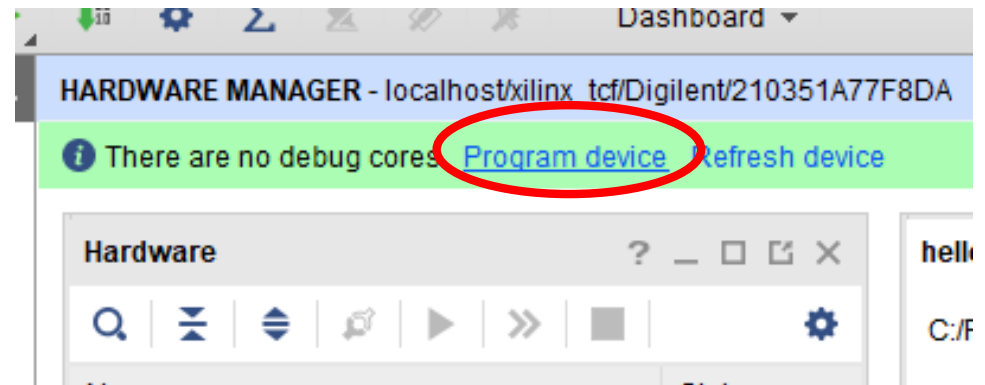
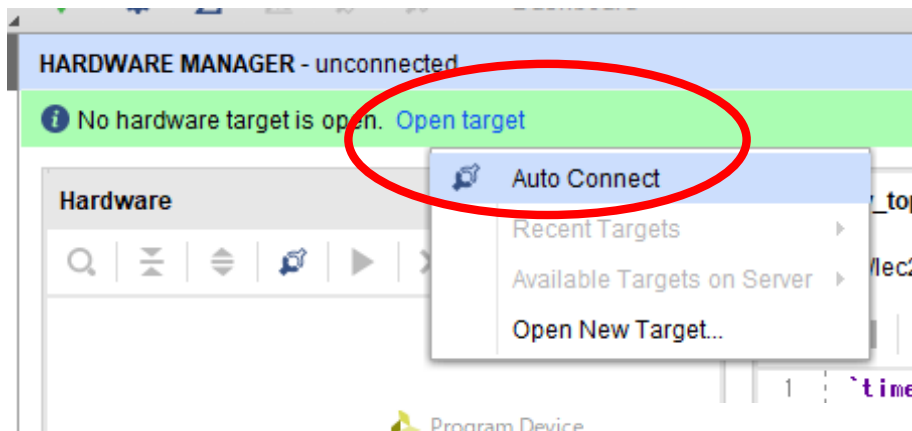


# Generate Bitstream (Configuration data)



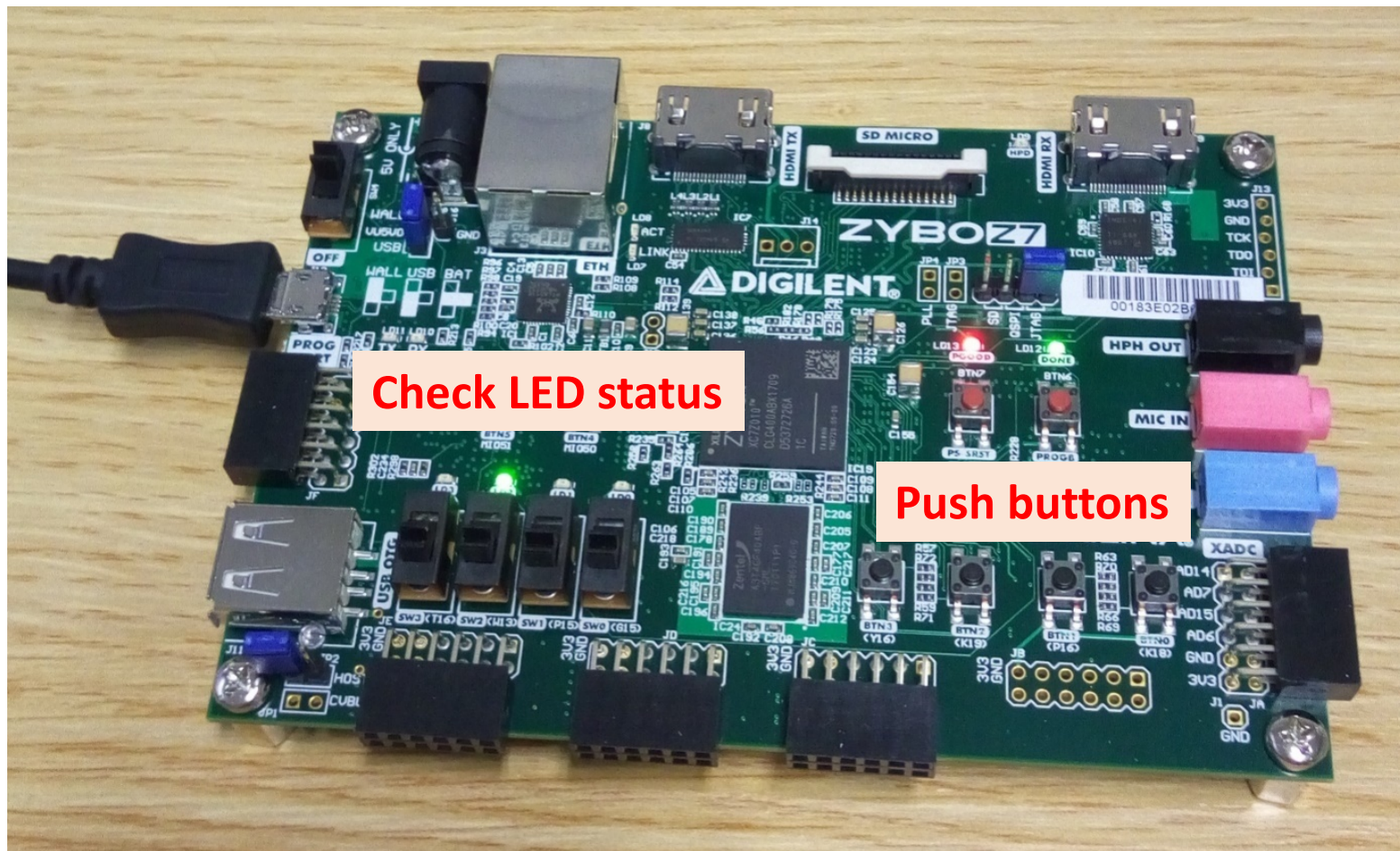
# Programming

- Click "Open target", then "Auto Connect"
- Next, click "Program device", then "Program"

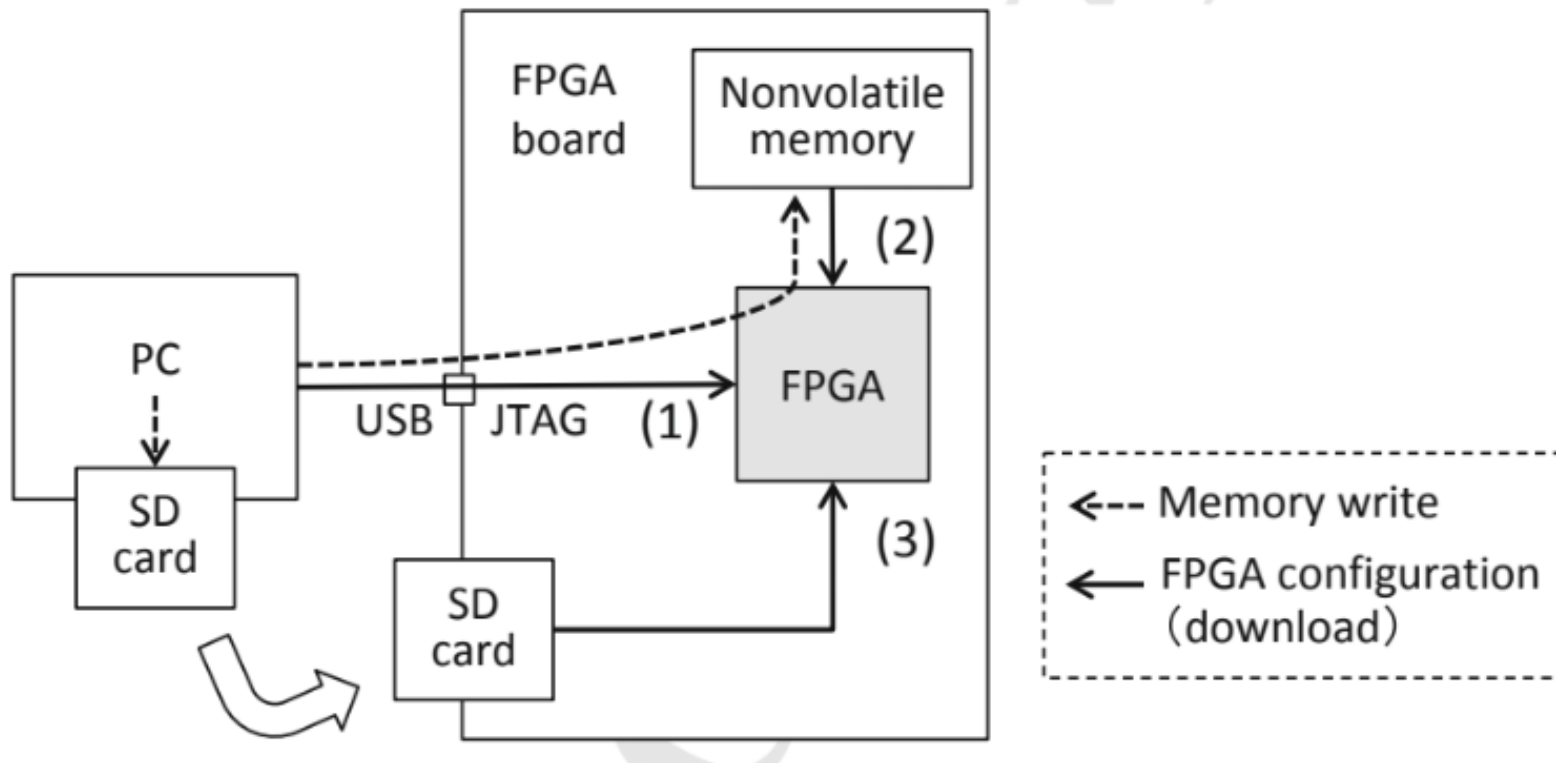




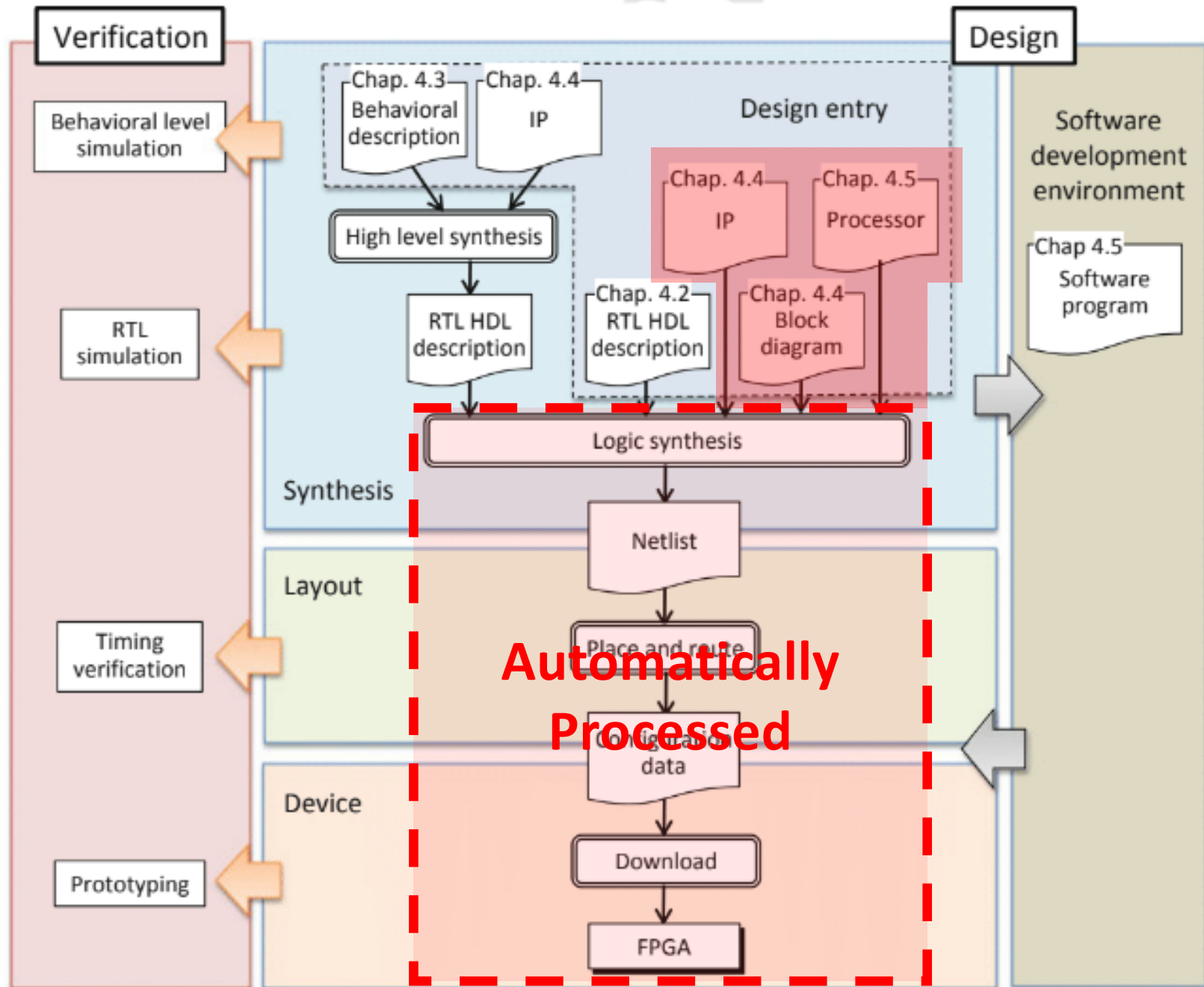
# Welcome to HW world!



# FPGA Configuration Methods



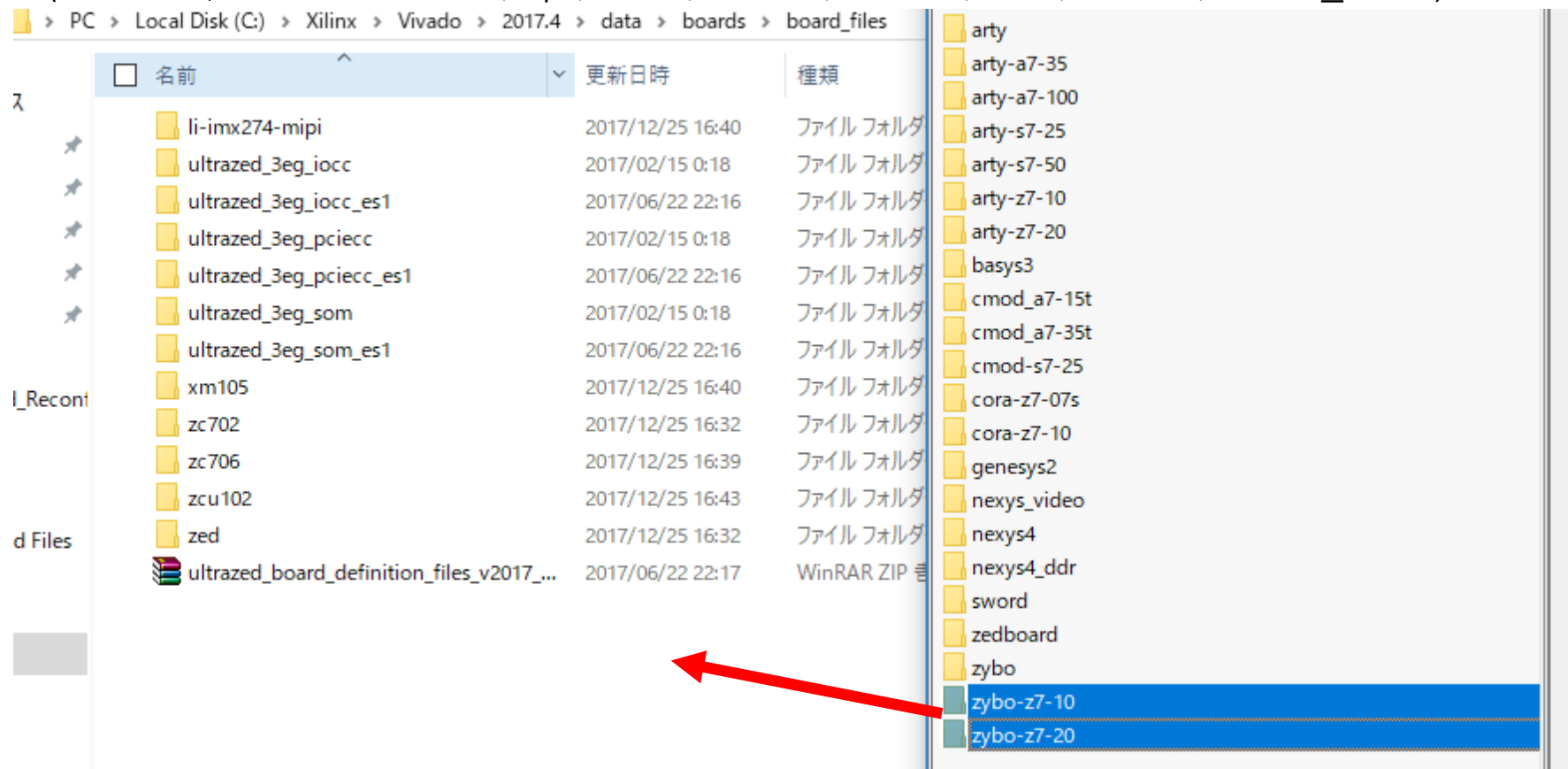
# Processor Design Flow





# Board File Definition File

- Constraint File & IP cores for specified board
- Zybo-Z7: <https://github.com/Digilent/vivado-boards/archive/master.zip>
- Unzip and store to C:/Xilinx/Vivado/2017.4/data/boards/board\_files  
(For Unix, it is located in "/opt/Xilinx/Vivado/2017.4/data/boards/board\_files")



# Hello World on Zybo-Z7

- Output of "Hello World" with UART of PS section and software on CPU
- Main part is PS section, but first we will make hardware at Vivado
- After that, we will write Hello World software on SDK

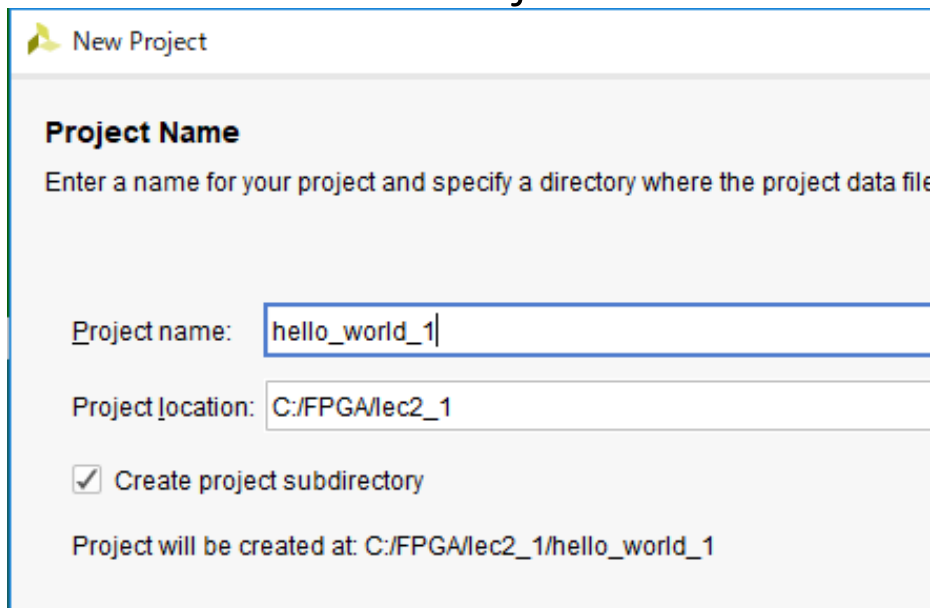
# Create Project

Run Vivado 2017.4

File->New Project, then "Next"

Specify project name "hello\_world\_1" and its location is "C:/FPGA/lec2\_1" (For Unix, it is "/root/FPGA/lec2\_1")

Select "RTL Project" with no sources and no



New Project

**Project Name**  
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

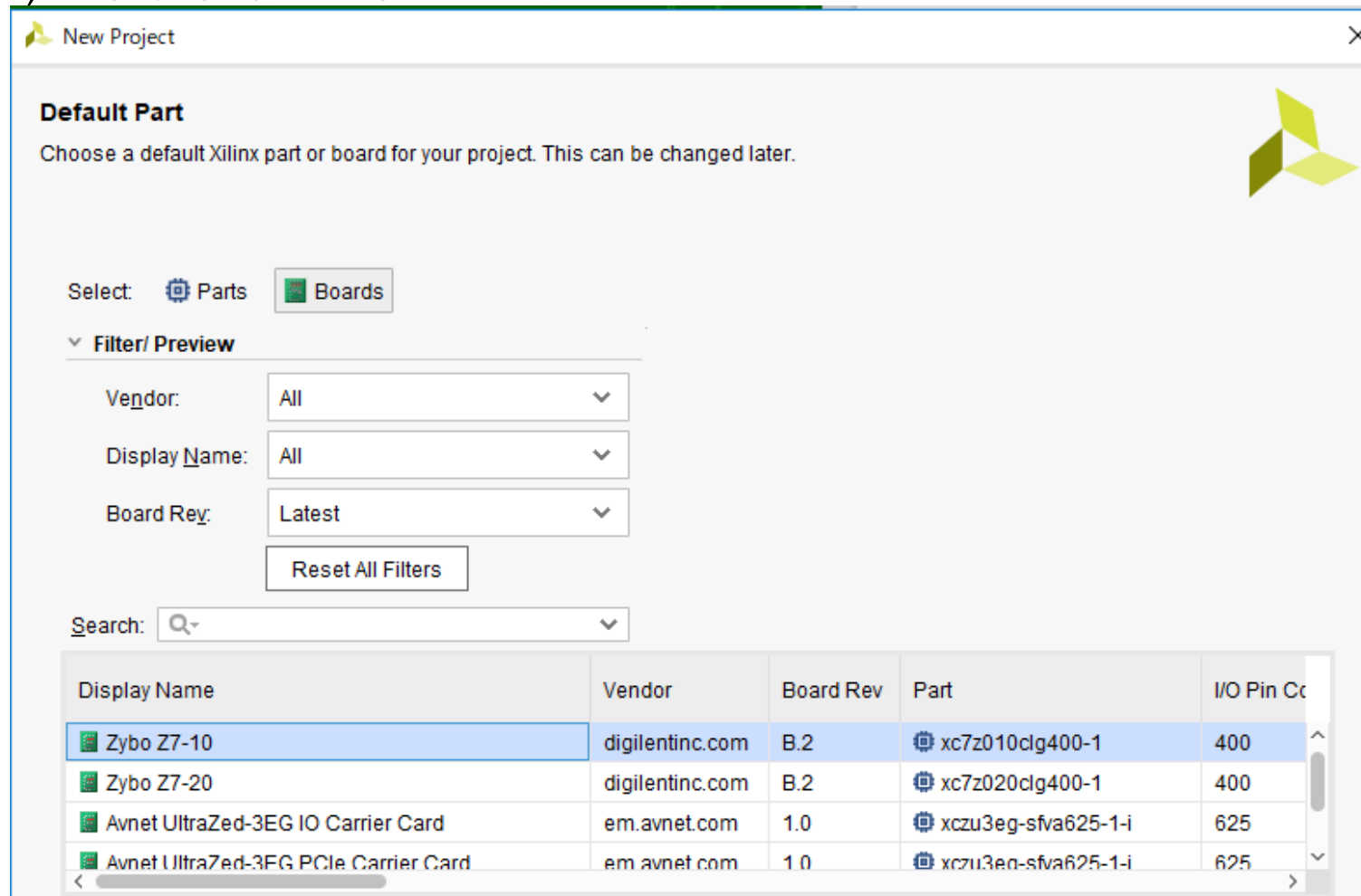
☒ Create project subdirectory

Project will be created at: C:/FPGA/lec2\_1/hello\_world\_1

# Select Zybo-Z7

In "Default Part", select "Boards" tab, and **select your Zybo! (Z7-10 or Z7-20)**

Then, "Next" and "Finish"



**New Project**

**Default Part**  
Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☐ Parts ☒ Boards









▼ **Filter/ Preview**

Vendor: All

Display Name: All

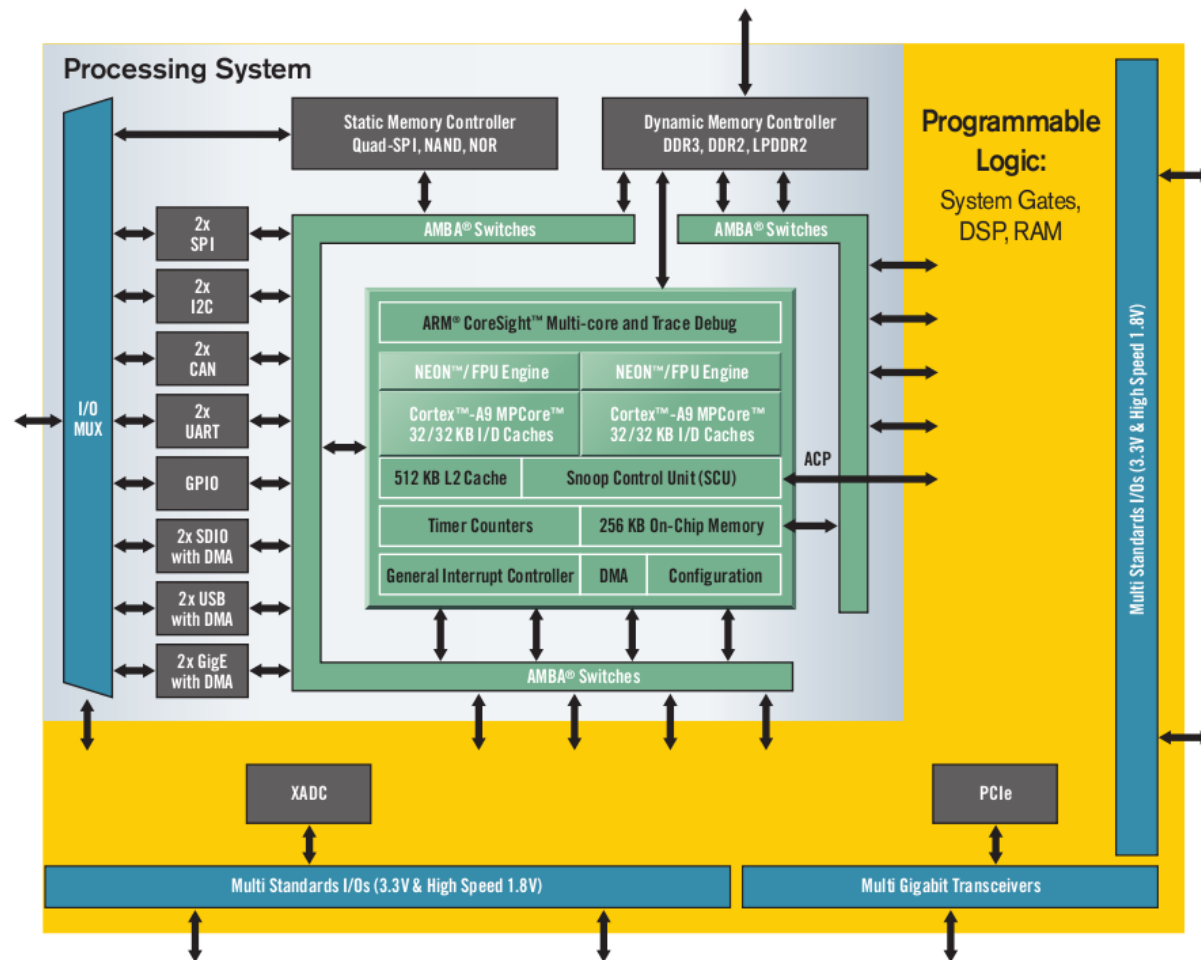
Board Rev: Latest

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count
 Zybo Z7-10	digilentinc.com	B.2	 xc7z010clg400-1	400
 Zybo Z7-20	digilentinc.com	B.2	 xc7z020clg400-1	400
 Avnet UltraZed-3EG IO Carrier Card	em.avnet.com	1.0	 xczu3eg-sfva625-1-i	625
 Avnet UltraZed-3EG PCIe Carrier Card	em.avnet.com	1.0	 xczu3eg-sfva625-1-i	625

# Modern FPGA

- Processor System (PS) + Programmable Logic (PL)
- Hard macro IPs with dedicated IP (on PL) → Short-time design

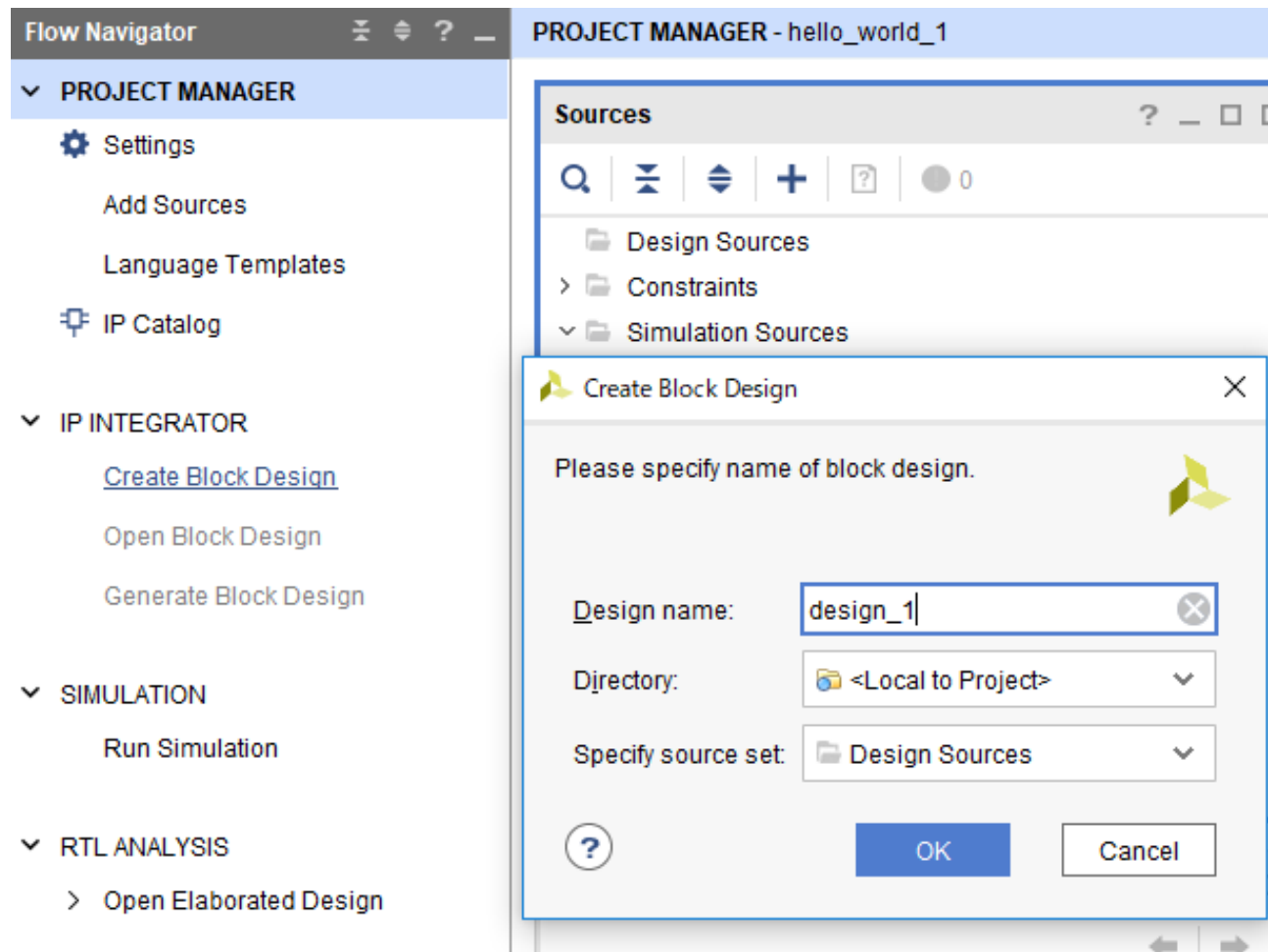


Source: Xilinx.com

# IP Design

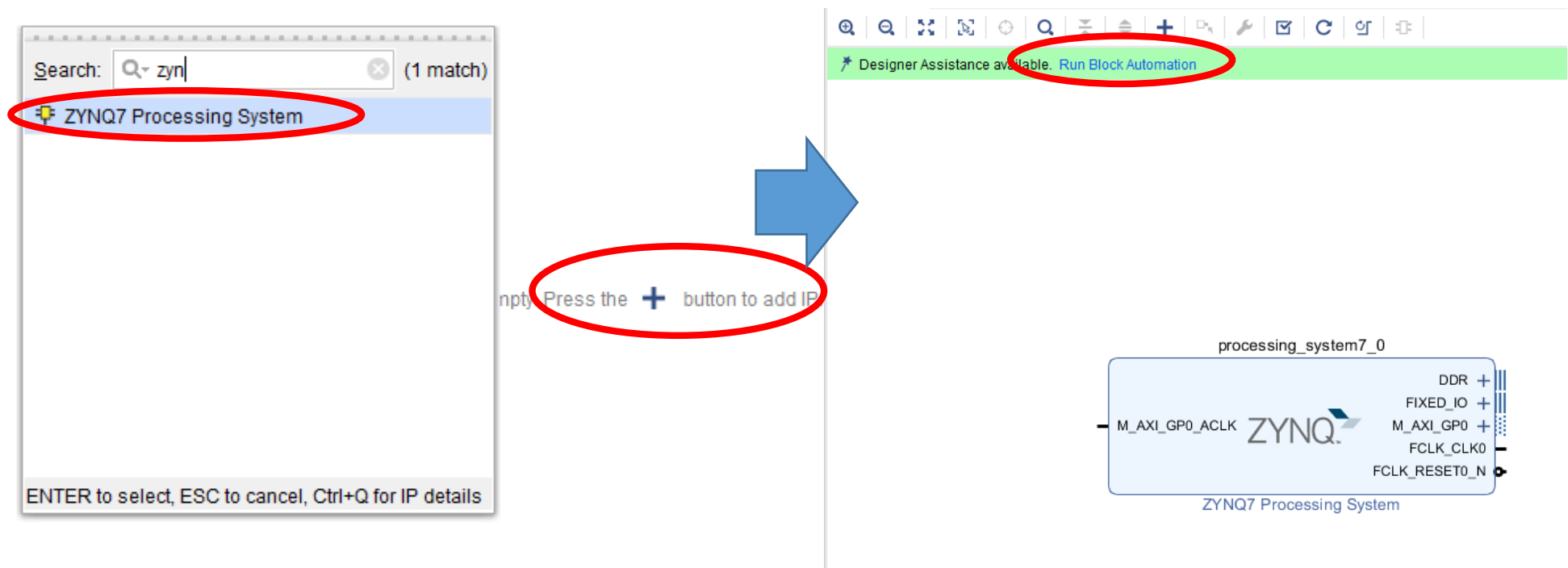
- The scale of the digital system increases day by day, the design-time is prolonged and the development cost is also increased
- Modules such as interface, control of peripheral devices, communication, encryption, compression, signal and image processing are common in many cases
  - Possible to reduce development time and cost problems by reusing them
- Commonable and reusable hardware library
  - IP (Intellectual Property)

# Launch IP Designer



# Select Zynq PS

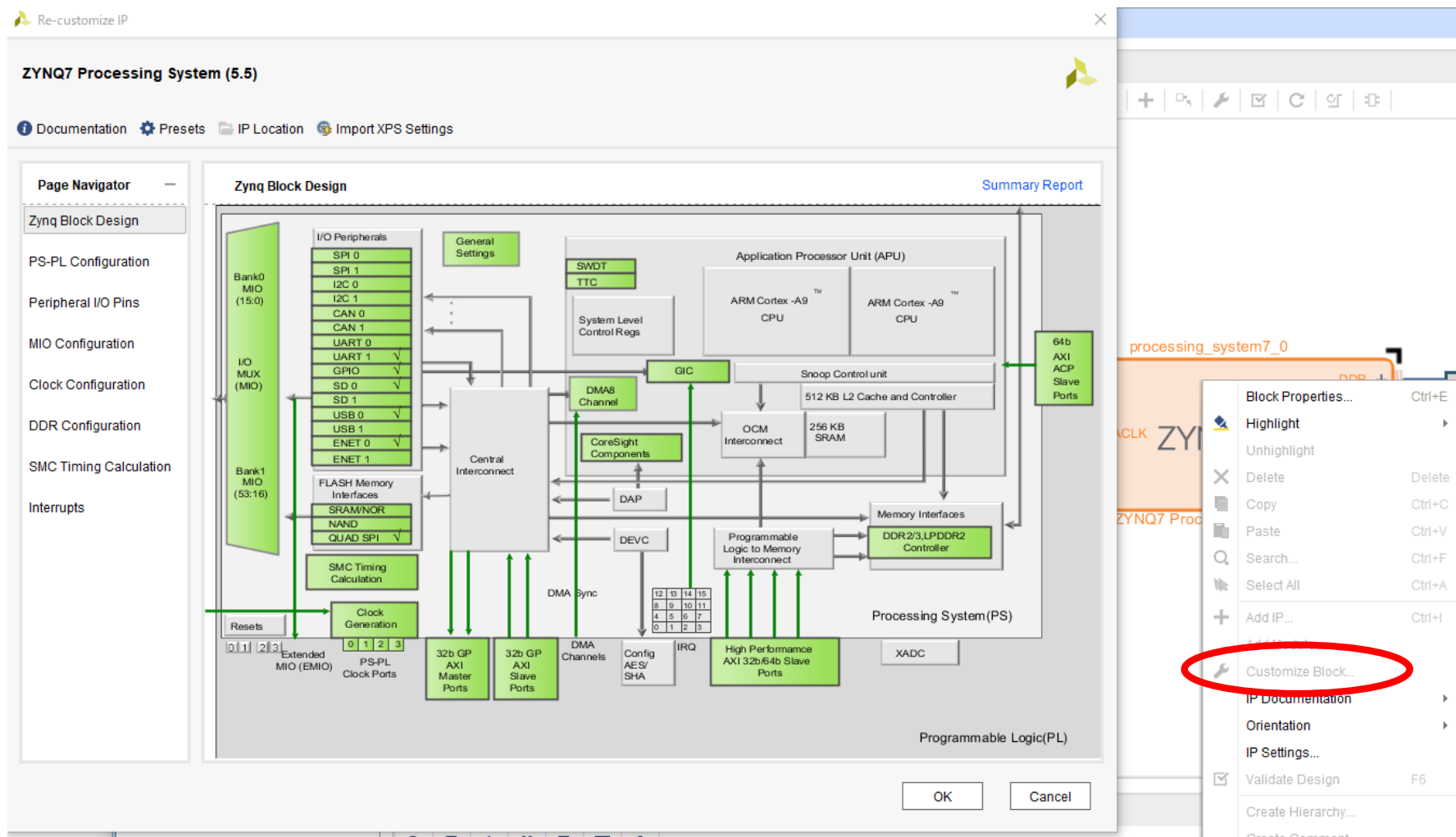
- Click "+" button and select "ZYNQ7", then ZYNQ IP core is placed to BLOCK DESIGN Editor
- Click "Run Block Automation", and make sure "Apply Board Preset" is checked, then "OK"





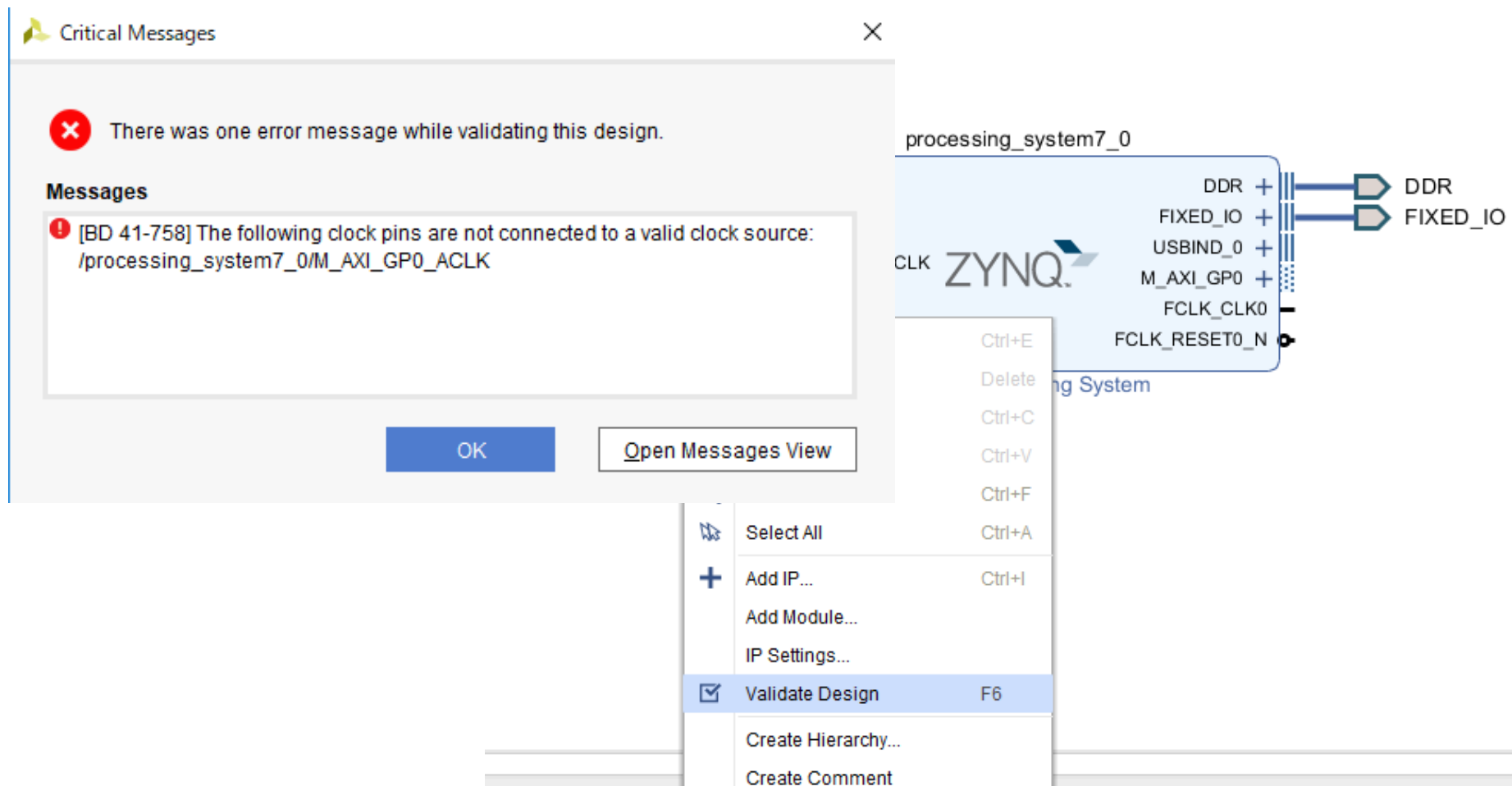
# View Re-customize IP

Right click on "ZYNQ" block and select "Customize Block ...", then we can modify the settings (However, these have already been specified by BDF)



# Valid Design

- Right click on empty space, and select "Validate Design"
- This tutorial meets critical error



# Re-use PL Fabric Clocks

- PS generates 50MHz clock signal to control PL

Re-customize IP

## ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration**
- DDR Configuration
- SMC Timing Calculation
- Interrupts

### Clock Configuration

Basic Clocking Advanced Clocking

Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Search: Q

Component	Clock Source	Requested Frequ...	Actual Frequency(...)	Range(MHz)
> Processor/Memory Clocks				
> IO Peripheral Clocks				
▼ PL Fabric Clocks				
<input checked="" type="checkbox"/> FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000

processing\_system7\_0

ZYNQ

Q7 Processing System

DDR +

FIXED\_IO +

USBIND\_0 +

M\_AXI\_GP0 +

FCLK\_CLK0 +

FCLK\_RESET0\_N

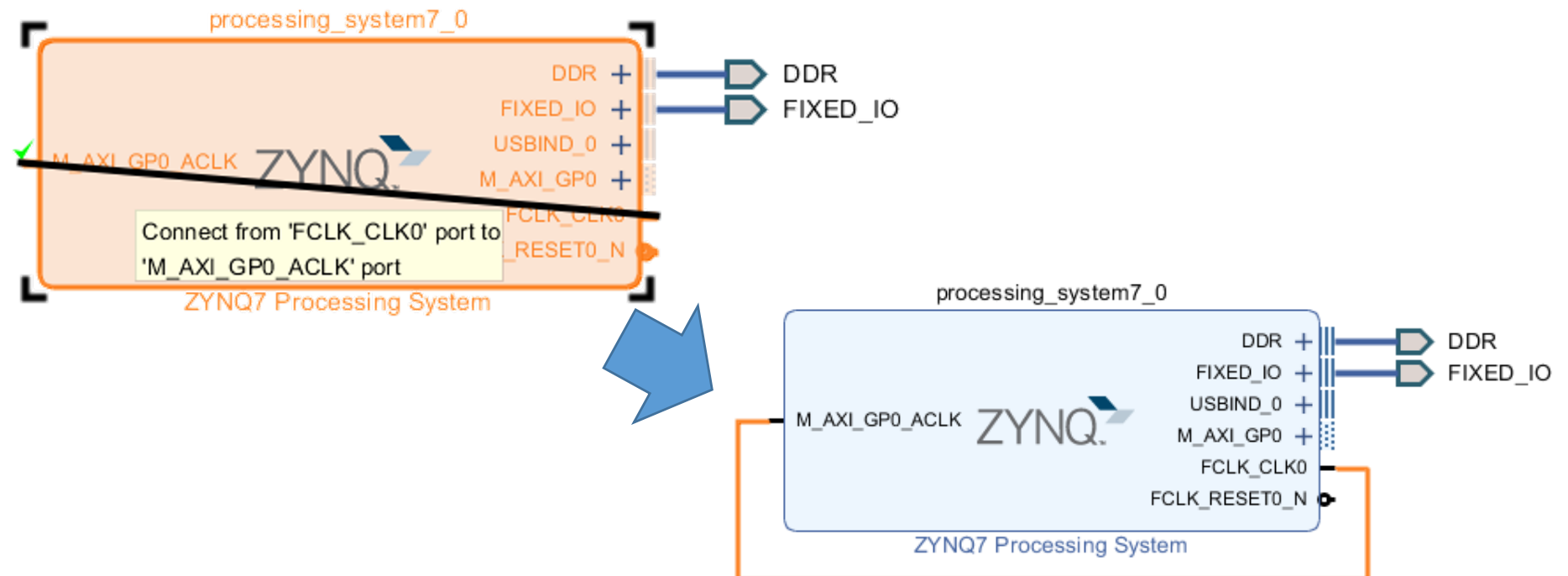
DDR

FIXED\_IO

# Connect Clock Sources

Drag "FCLK\_CLK0" and drop to "M\_AXI\_GP0\_ACLK", then it automatically connect them

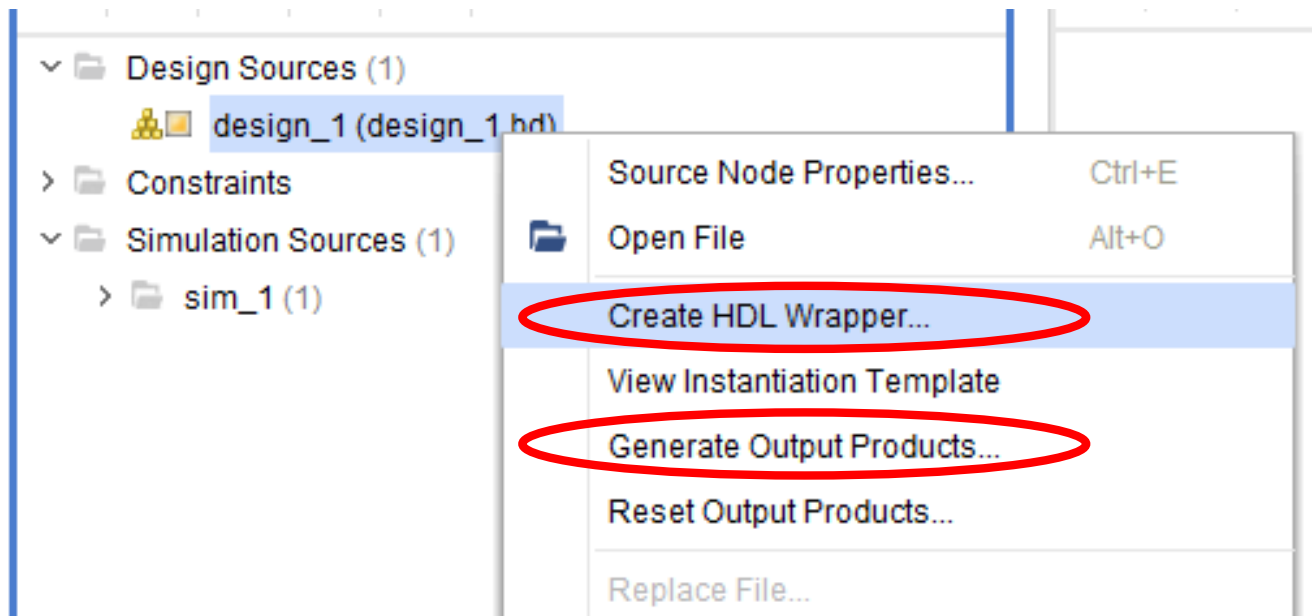
Validate design again, then there are no errors



# Generate HDL Files

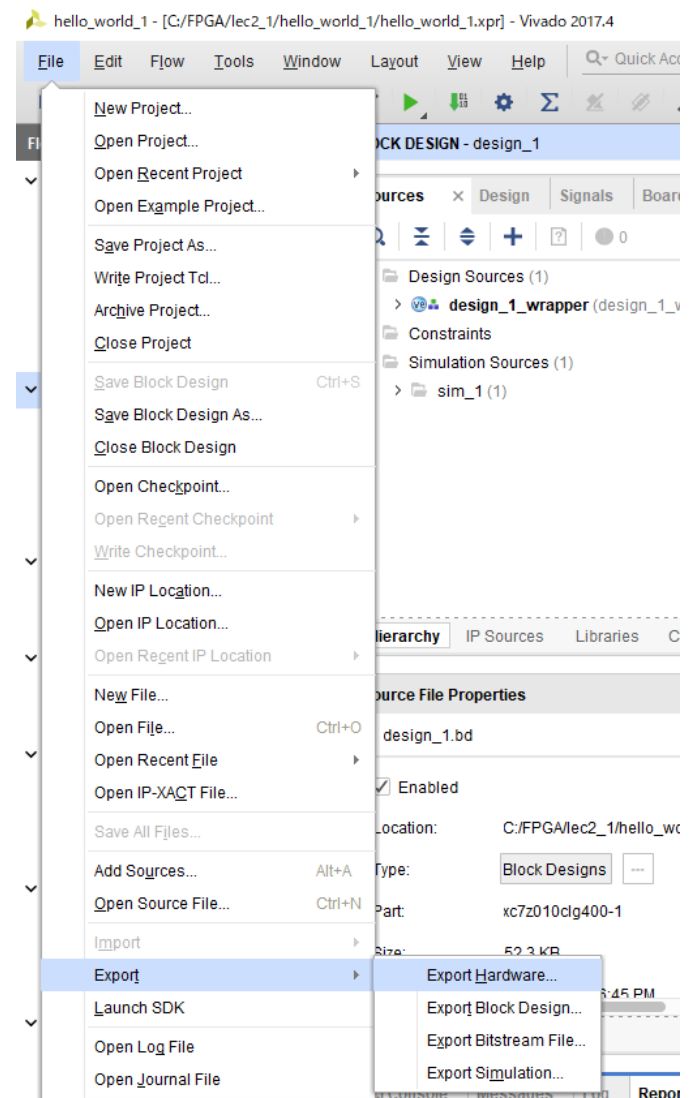
To re-use HDL design flow, do following steps:

1. Right-click on "design\_1", and select "Generate Output Products...", then "Generate"
2. Again, Right-click on "design\_1", and select "Create HDL Wrapper...", then "OK"



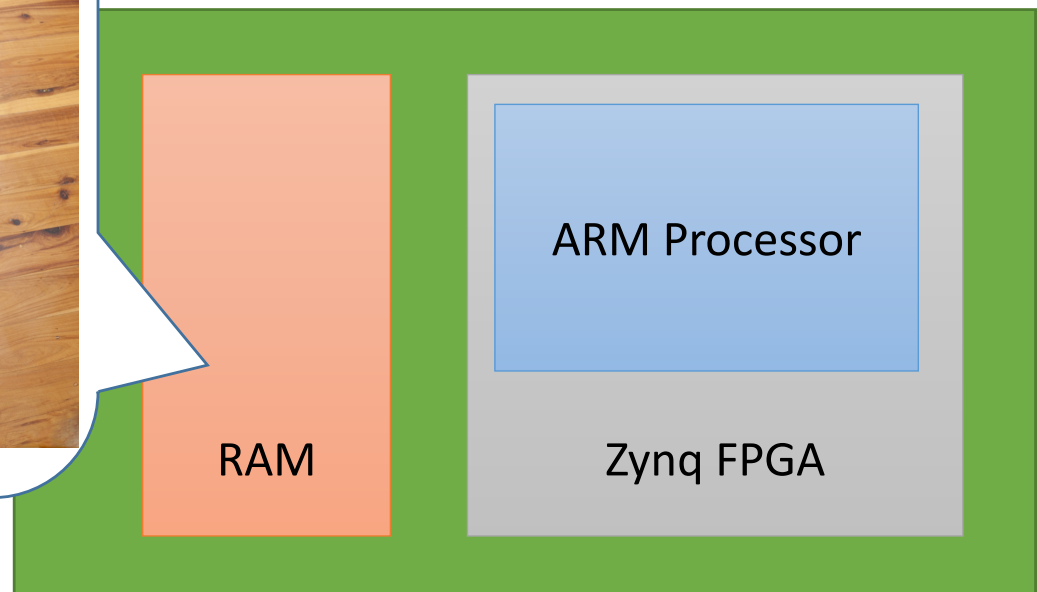
# Synthesis Hardware and Export

- Click "Generate Bitstream", then "Yes" and "OK"
- Wait few minutes...
  - FPGA design tool do logic synthesis, place-and-routing, and generating bitstream
- After finish bitstream generation, then "Cancel"
- In "Menu", select "File" and "Export", then "Export Hardware"
- **Check "Include bitstream"**, then "OK"



# We are Here...

- Hardware (Processor) has already generated, however, there are no software on the RAM
  - HDF: Hardware definition file (Bitstream + Memory I/O map)
- Write software and compile it, then execute

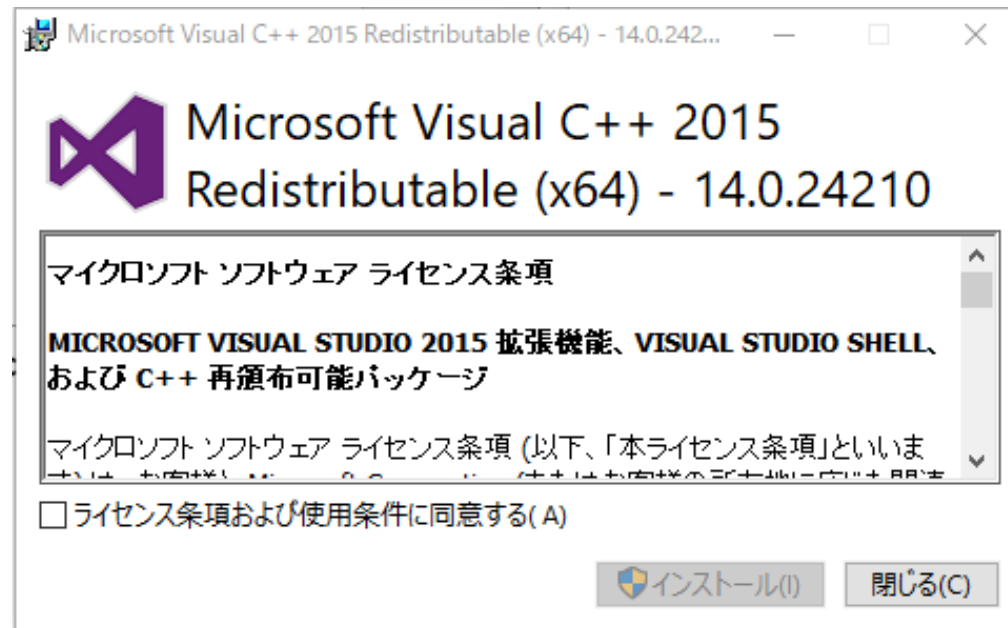


# Launch SDK

In Vivado, "Menu" -> "File" -> "Launch SDK" -> "OK"

**Note:** If you meet Visual C++ Runtime trouble, see  
<https://forums.xilinx.com/t5/Installation-and-Licensing/Vivado-Xilinx-SDK-Error-Incorrect-Visual-C-Version/td-p/442628>

-> Rename C:/Xilinx/SDK/2017.4/tips/win64/vcredist\_x64.exe  
(and xvcredist.ext both)





# SDK Launched with designed HW

The screenshot displays the Xilinx SDK interface for a project named 'hello\_world\_1.sdk'. The title bar indicates the project path: 'C/C++ - design\_1\_wrapper\_hw\_platform\_0/system.hdf - Xilinx SDK'. The menu bar includes File, Edit, Navigate, Search, Project, Run, Xilinx, Window, and Help. The toolbar contains various icons for file operations and development tools.

The **Project Explorer** on the left shows the project structure. A red circle highlights the 'design\_1\_wrapper\_hw\_platform\_0' folder, which contains the following files:

- design\_1\_wrapper.bit
- ps7\_init\_gpl.c
- ps7\_init\_gpl.h
- ps7\_init.c
- ps7\_init.h
- ps7\_init.html
- ps7\_init.tcl
- system.hdf

The **system.hdf** file is selected, displaying the **design\_1\_wrapper\_hw\_platform\_0 Hardware Platform Specification**. The **Design Information** section shows:

- Target FPGA Device: 7z010
- Part: xc7z010clg400-1
- Created With: Vivado 2017.4
- Created On: Wed Jun 13 22:17:11 2018

The **Address Map for processor ps7\_cortexa9\_0-1** is shown in a table:

Cell	Base Addr	High Addr	Slave I/f
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	
ps7_gpio_0	0xe000a000	0xe000afff	
ps7_scutimer_0	0xf8f00600	0xf8f0061f	
ps7_slcr_0	0xf8000000	0xf8000fff	
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	
ps7_scuc_0	0xf8f00000	0xf8f000fc	
ps7_qspi_linear_0	0xfc000000	0xfcffffff	
ps7_pmu_0	0xf8893000	0xf8893fff	
ps7_afi_1	0xf8009000	0xf8009fff	

The **Target Connections** pane at the bottom left shows the following connections:

- Hardware Server
- Linux TCF Agent
- QEMU TcfGdbClient

The **SDK Log** pane at the bottom right shows the following log entries:

```
22:42:16 INFO : Registering command handlers for
22:42:18 INFO : Launching XSCT server: xsct.ba
22:42:21 INFO : XSCT server has started succes
22:42:21 INFO : Successfully done setting XSCT
22:42:23 INFO : Processing command line option
22:42:23 INFO : Successfully done setting SDK
```

Already loaded

# Make Project

In "Menu", "New" ->

"Application Project"

Set project name "HelloWorld"

OS: Standalone (default)

Hardware: (default)

Processor: ps7\_cortexa9\_0

Language: C

Board Support Package (BSP):

HelloWorld\_bsp

(It is a library collection for your HW)

SDK New Project

**Application Project**  
Create a managed make application project.

Project name: HelloWorld

☒ Use default location  
Location: C:\FPGA\lec2\_1\hello\_world\_1\hello\_world\_1.sdk\HelloWorld Browse...

Choose file system: default

OS Platform: standalone

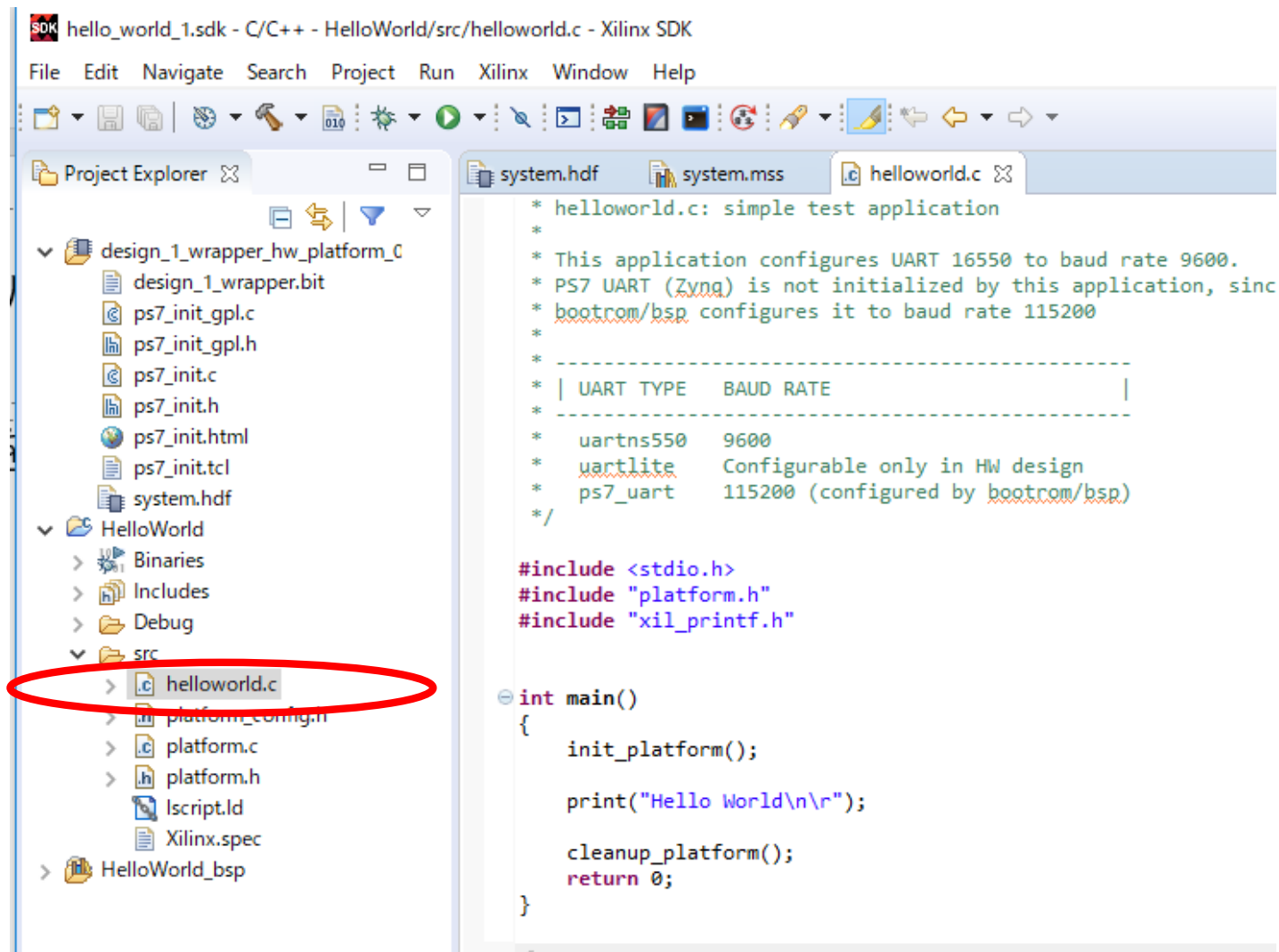
Target Hardware  
Hardware Platform: design\_1\_wrapper\_hw\_platform\_0 New...  
Processor: ps7\_cortexa9\_0

Target Software  
Language: ☒ C ☐ C++  
Compiler: 32-bit  
Hypervisor Guest: N/A  
Board Support Package: ☒ Create New HelloWorld\_bsp  
☐ Use existing

? < Back Next > Finish Cancel

# Check Source Code

"helloworld.c" has already been registered to a project



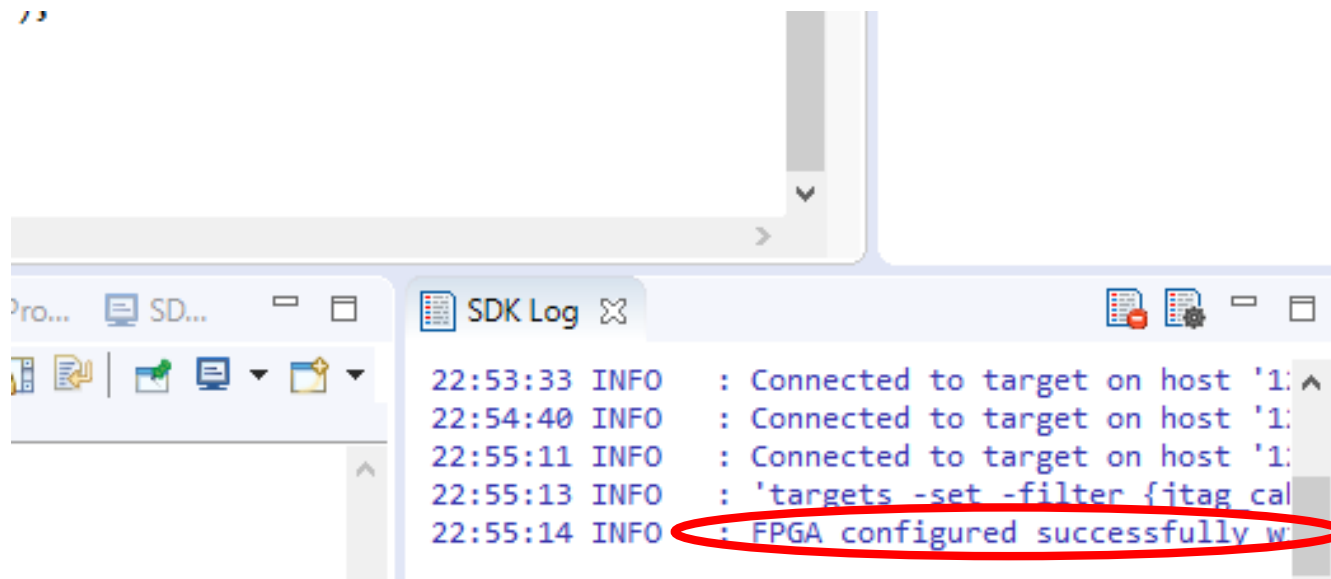
# Environment Setup

Connect the Zybo to the PC, then turn-on power switch

In "Menu", "Xilinx" -> "Program FPGA", then "Program"

- Hardware is configured until you turn-off power

In SDK log, if "FPGA configured successfully..." is showed, then go to the next slide, otherwise, turn-off and turn-on power, then try to "Program" again



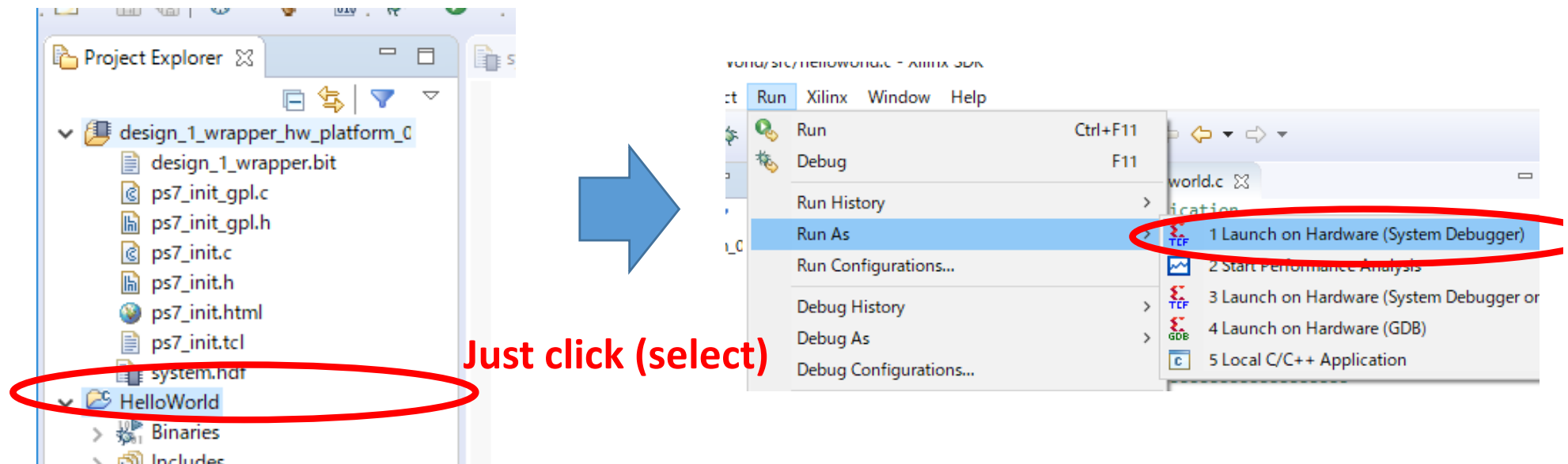
# Environment Setup & Run

Connect the Zybo to the PC

Run Terminal software (e.g. Tera Term for Windows, gtkterm for Unix)

Connect "USB Serial Port" with 115200 bps

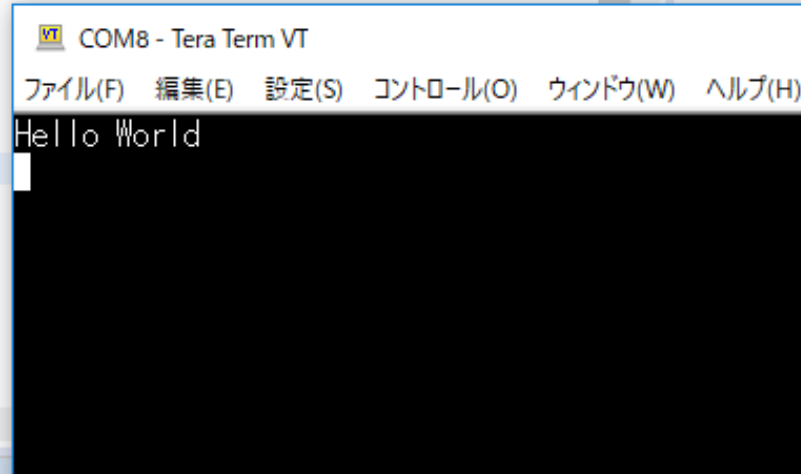
Select "HelloWorld" project in the Project Explorer, then, in "Menu", "Run As" -> "Launch on Hardware (System Debugger)"



# Welcome to SW World!

```
* This application configures UART 16550 to baud rate 9600.  
* PS7 UART (Zynq) is not initialized by this application, since  
* bootrom/bsp configures it to baud rate 115200  
*  
* -----  
* | UART TYPE   BAUD RATE  
* -----  
*   uartns550   9600  
*   uartlite    Configurable only in HW design  
*   ps7_uart    115200 (configured by bootrom/bsp)  
*/  
  
#include <stdio.h>  
#include "platform.h"  
#include "xil_printf.h"  
  
int main()  
{  
    init_platform();  
  
    print("Hello World\n\r");  
  
    cleanup_platform();  
    return 0;  
}
```

- st
- pl
- xi
- m



# Exercise

1. (Mandatory) Execute the HDL design with following this tutorial, and "hello world" software tutorial. Then, send the source code and screen shot of your execution situation by a PDF.

If you meet any troubles, don't hesitate to contact me.

[nakahara@ict.e.titech.ac.jp](mailto:nakahara@ict.e.titech.ac.jp)

Deadline is 10<sup>th</sup>, July, 2020 JST PM13:20

(At the beginning of the next lecture)