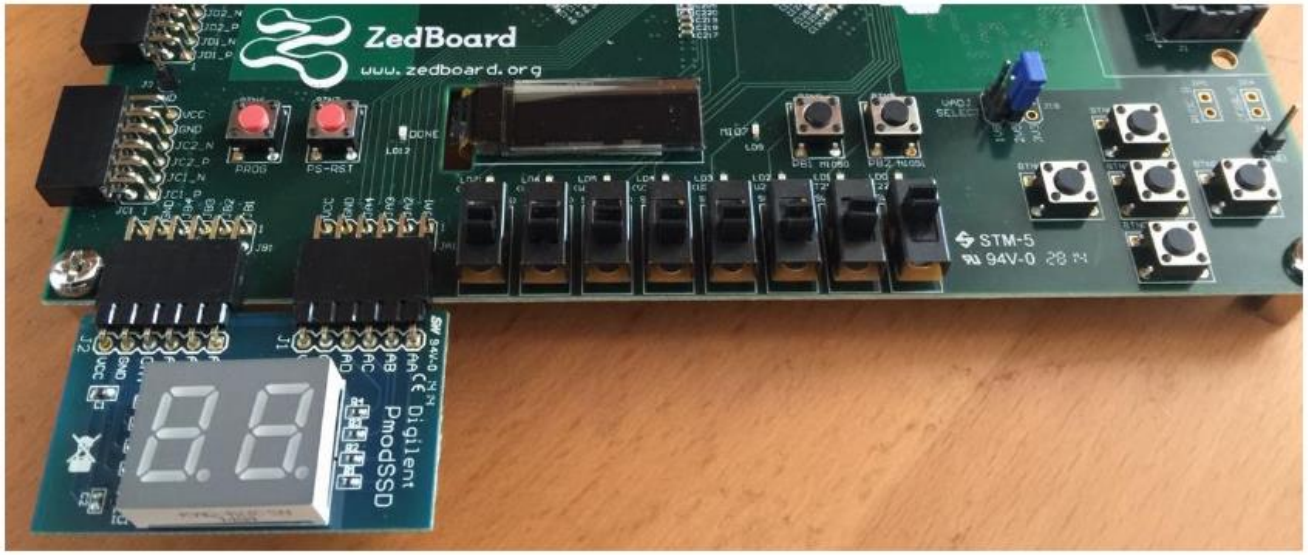
香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lab07: Integration of ARM and FPGA

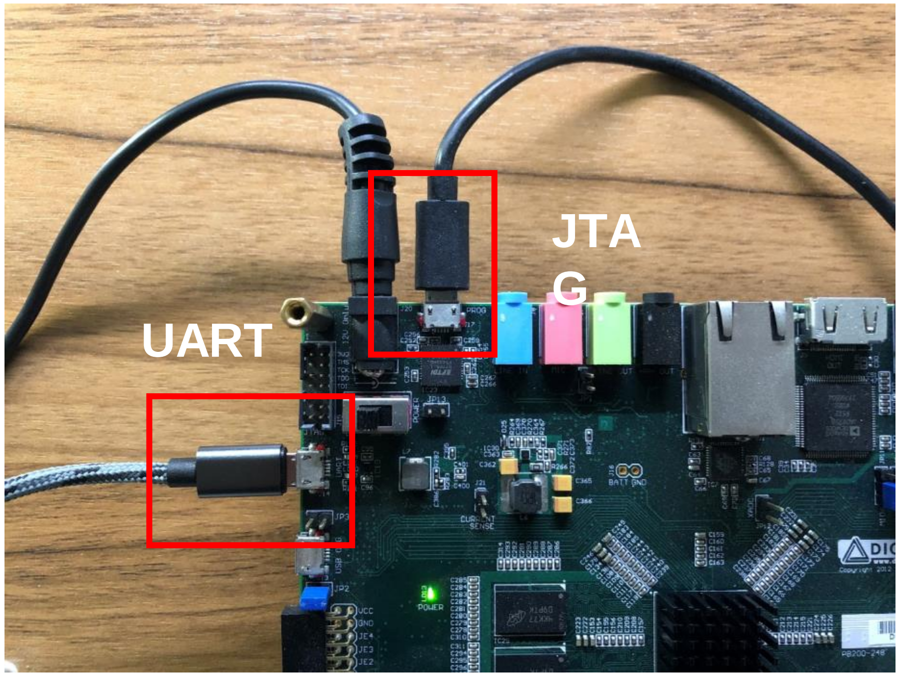
Exercise

- In this lab, you need to follow the step-by-step instructions build a Random Number Generator in software:
  - ARM processor (i.e., Processing System) is responsible for generating the random number as VHDL is not capable of.
  - FPGA (i.e., Programmable Logic) is interfacing with the per
- You need to connect the PmodSSD to the ZedBoard via the Pmod connectors as follows:

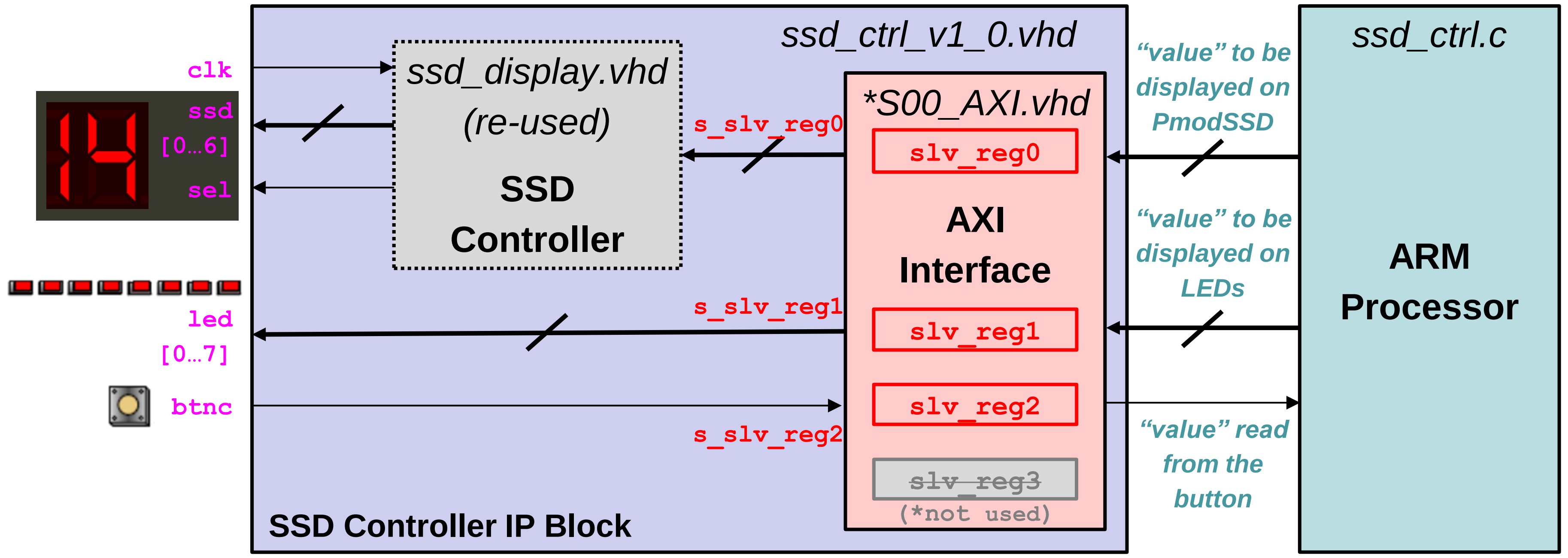


Hardware Setup

- In this lab, you need to connect two micro-b USB wires to the Zedboard.
  - Connect the JTAG port on the top. This port is used to program the hardware bitstream.
  - Connect the UART port under the power switch. This port is used to program the software binary to the board and communicate with the software program.



AXI IP Block Design Diagram

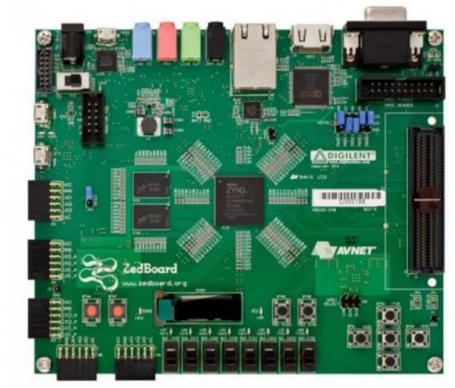


Lab07 Outline

- **PART 1: IP Block Design**

AMD Vivado

① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

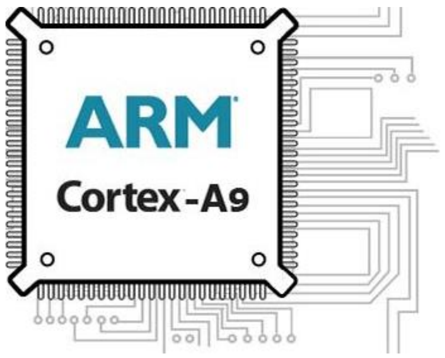


&

- **PART 2: ARM Programming**

AMD Vitis

⑤ ARM Programming
⑥ Launch on Hardware



- **PART 1: IP Block Design**

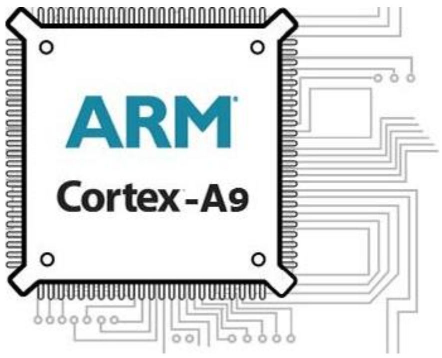
AMD Vivado

① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

- **PART 2: ARM Programming**

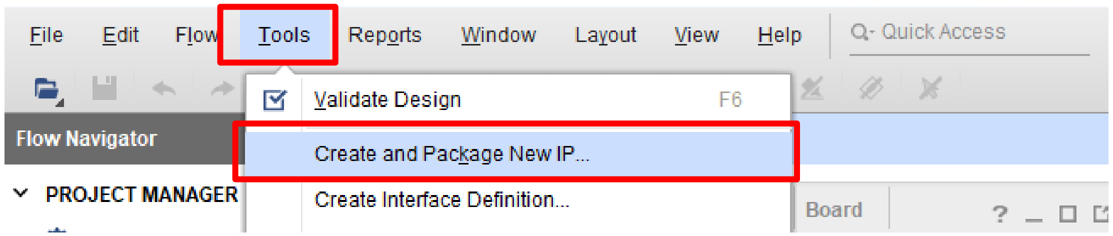
AMD Vitis

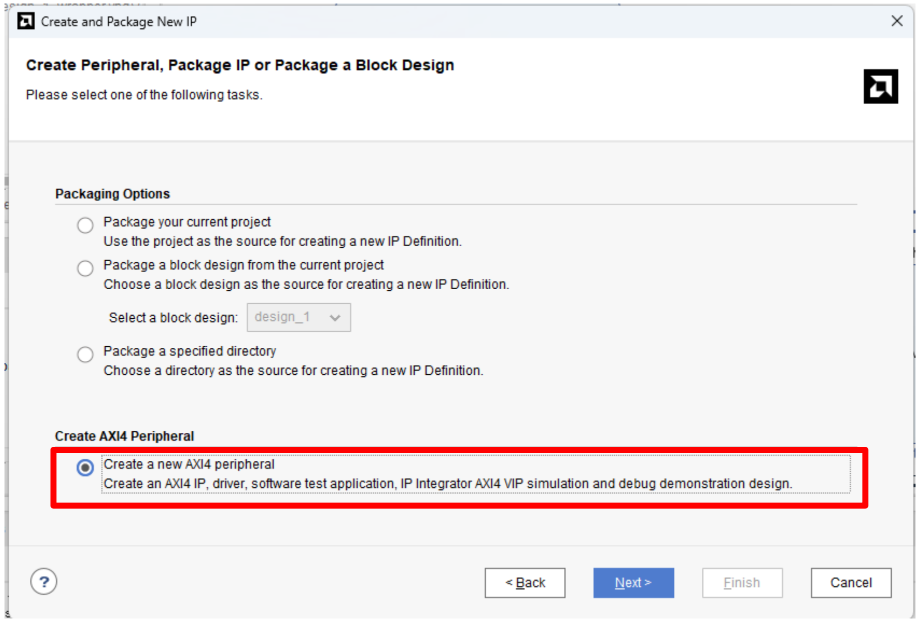
⑤ ARM Programming
⑥ Launch on Hardware

&


IP Block Creation

- Create a new Project called "Lab07"
- Click "Tools -> Create and Package New IP"
- Select "Create a new AXI4 peripheral"




Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

Packaging Options

( ) Package your current project
Use the project as the source for creating a new IP Definition.

( ) Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.

Select a block design: design_1

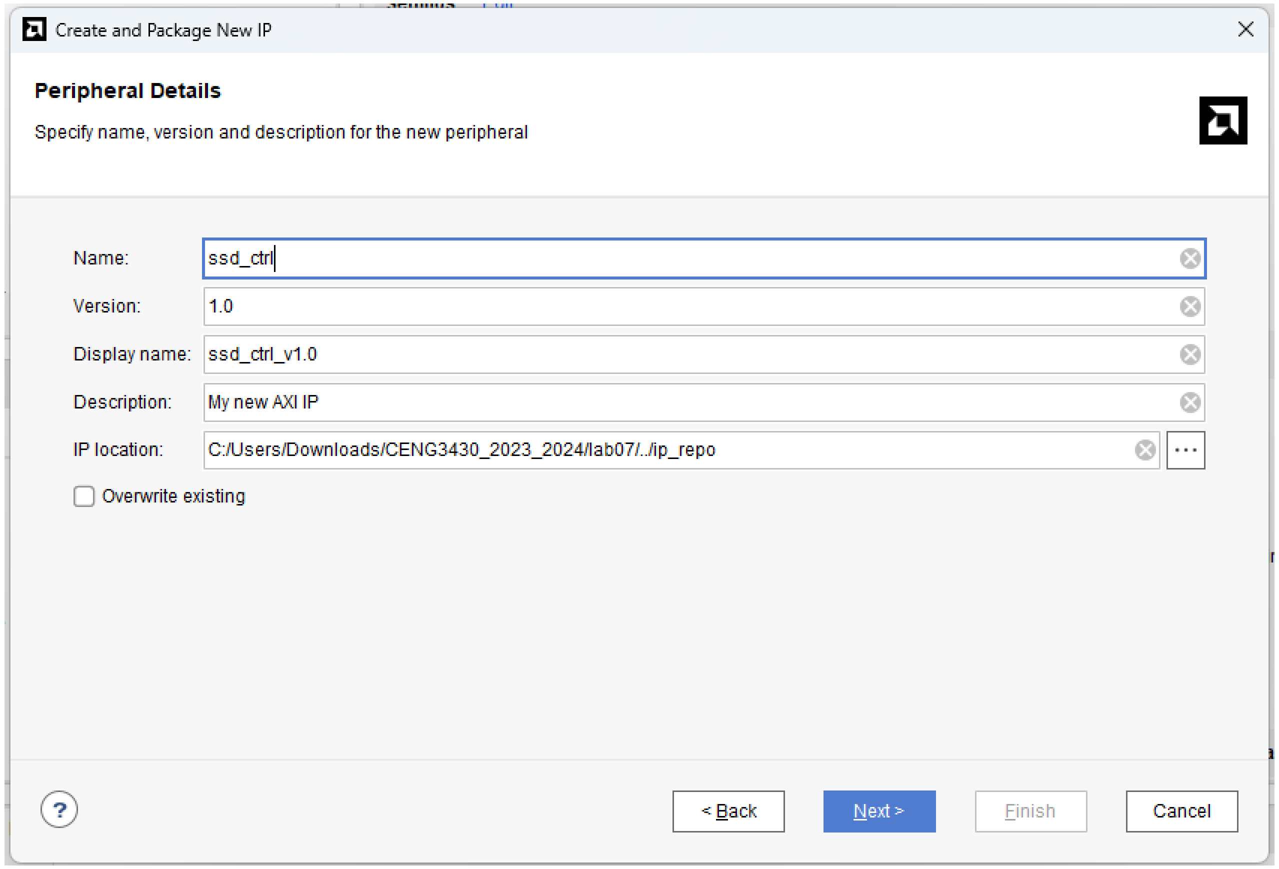
( ) Package a specified directory
Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

[✓ Create a new AXI4 peripheral]
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

IP Block Creation

- Name it "ssd_ctrl"
- Create a folder called "ip_repo" and store it
  - Remember the IP location!!



Peripheral Details

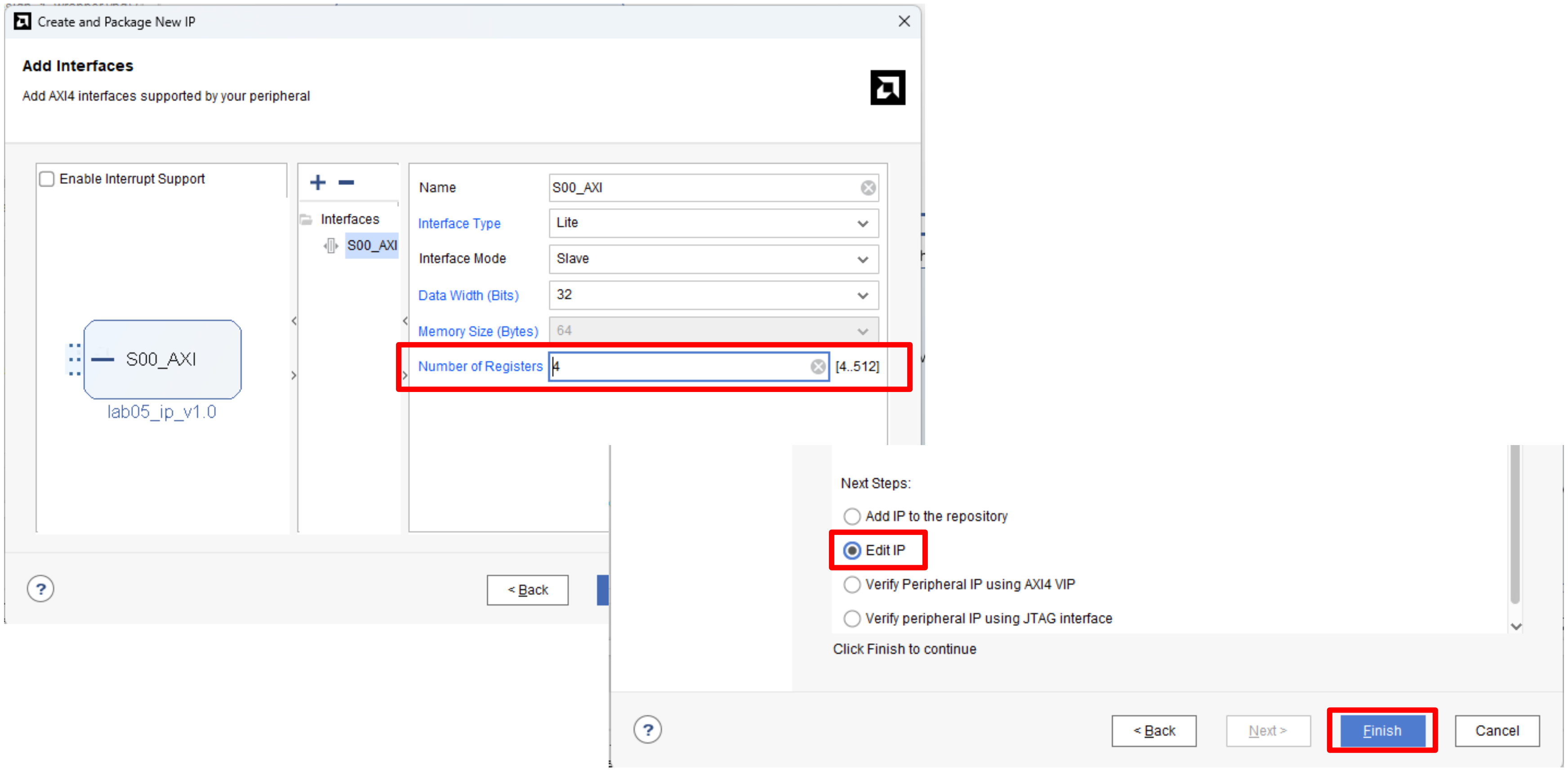
Specify name, version and description for the new peripheral

Name:	ssd_ctrl
Version:	1.0
Display name:	ssd_ctrl_v1.0
Description:	My new AXI IP
IP location:	C:/Users/Downloads/CENG3430_2023_2024/lab07/../ip_repo

☐ Overwrite existing

IP Block Creation

- Although we only require 3 registers, the minimum number is 4, so make it 4
- Finally, select "Edit IP" and click "Finish"



Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

[ ] Enable Interrupt Support

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

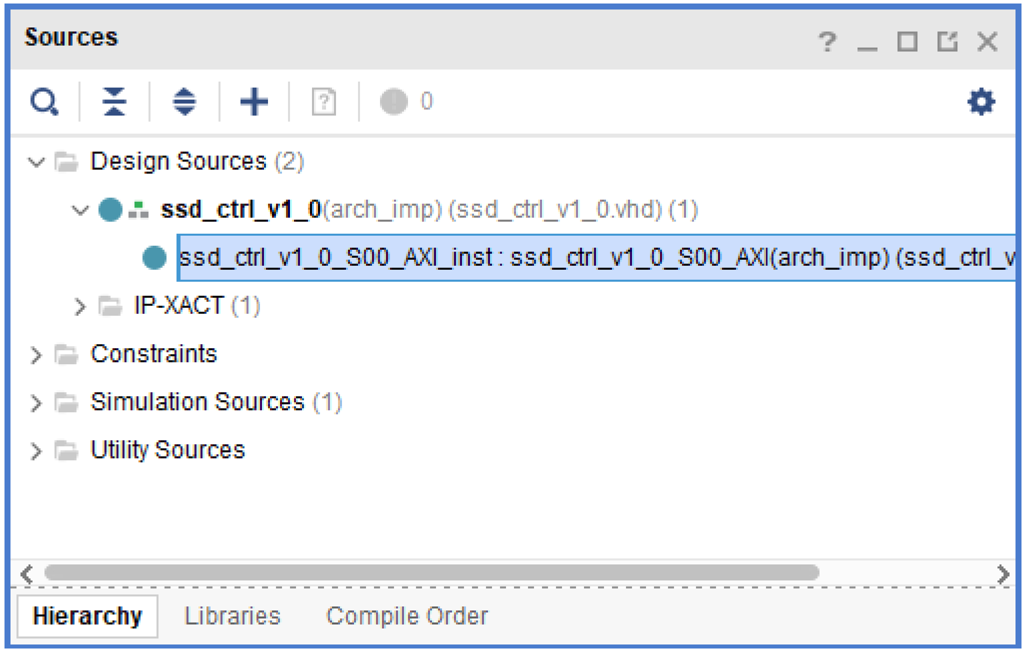
Next Steps:

- ○ Add IP to the repository
- ● Edit IP
- ○ Verify Peripheral IP using AXI4 VIP
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

IP Block Creation

- You will see the Design Sources have created two files
  - `ssd_ctrl_v1_0.vhd`
  - `ssd_ctrl_v1_0_S00_AXI.vhd`
    - We are going to work on the AXI file first



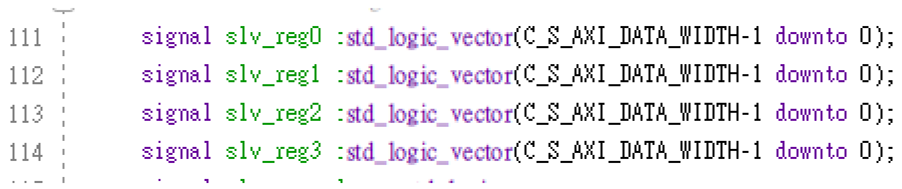
Sources

Design Sources (2)
ssd_ctrl_v1_0(arch_imp) (ssd_ctrl_v1_0.vhd) (1)
ssd_ctrl_v1_0_S00_AXI_inst : ssd_ctrl_v1_0_S00_AXI(arch_imp) (ssd_ctrl_v
IP-XACT (1)
Constraints
Simulation Sources (1)
Utility Sources

Hierarchy Libraries Compile Order

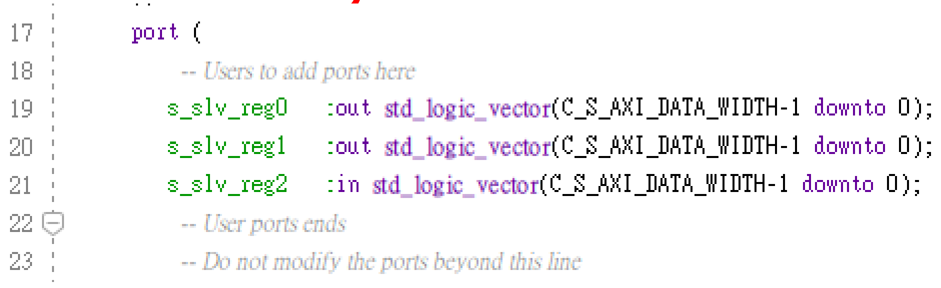
IP Block Creation

- First, edit the `ssd_ctrl_v1_0_S00_AXI.vhd`
- The slv_reg0-3 is already defined



```
signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
```

- However, the ports are not. So, we need to create the ports in the entity

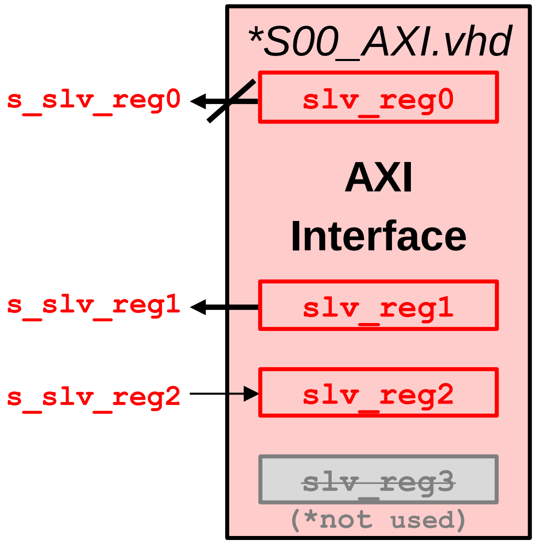


```
port (
    -- Users to add ports here
    s_slv_reg0   :out  std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    s_slv_reg1   :out  std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    s_slv_reg2   :in  std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    -- User ports ends
    -- Do not modify the ports beyond this line
```

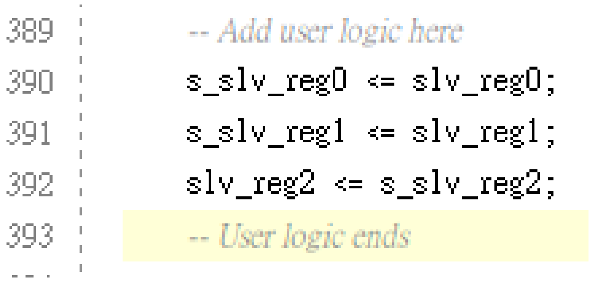
```
s_slv_reg0  :out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
s_slv_reg1  :out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
s_slv_reg2  :in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
```

You can copy this

IP Block Creation



- Connect the ports to registers after *Add user logic*



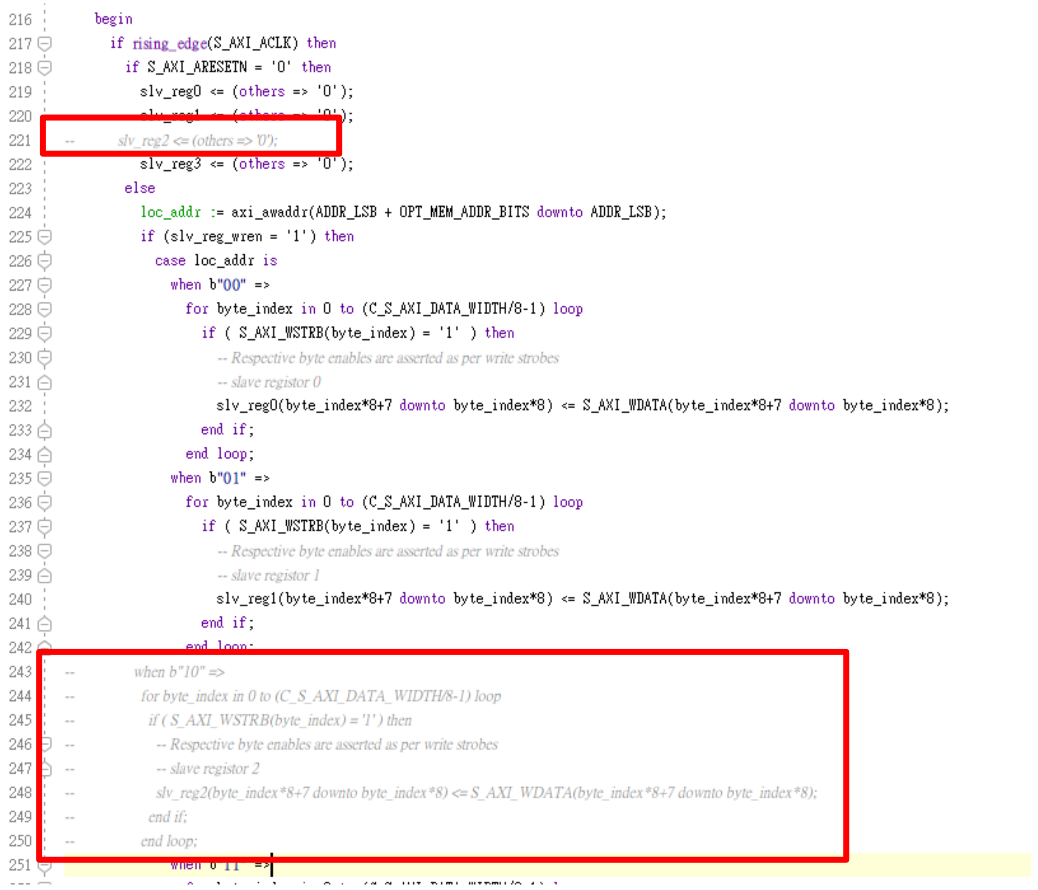
```
        -- Add user logic here
        s_slv_reg0 <= slv_reg0;
        s_slv_reg1 <= slv_reg1;
        slv_reg2 <= s_slv_reg2;
        -- User logic ends
```

```
s_slv_reg0 <= slv_reg0;
s_slv_reg1 <= slv_reg1;
slv_reg2 <= s_slv_reg2;
```

You can copy this

IP Block Creation

- For any ports then are receiving inputs from the outside, we don't need to reset **(btn -> slv_reg2)**
- **Comment the following codes**

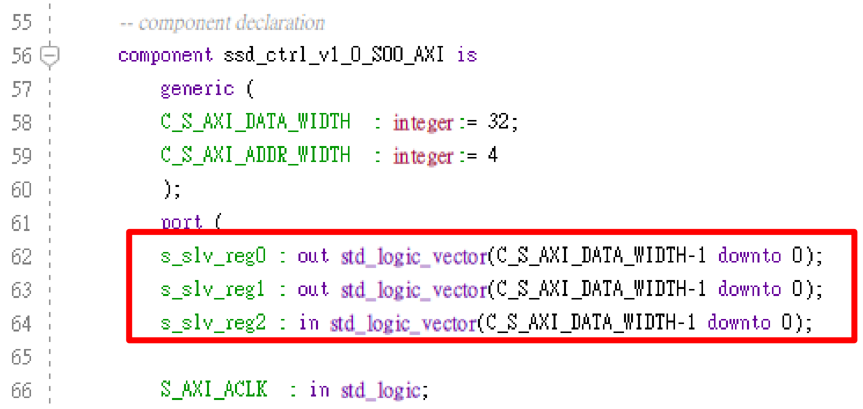


```
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
--        slv_reg2 <= (others => '0');
      slv_reg3 <= (others => '0');
    else
      loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
      if (slv_reg_wren = '1') then
        case loc_addr is
          when b"00" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 0
                slv_reg0(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
          when b"01" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 1
                slv_reg1(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
              end if;
            end loop;
--          when b"10" =>
--            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
--              if ( S_AXI_WSTRB(byte_index) = '1' ) then
--                -- Respective byte enables are asserted as per write strobes
--                -- slave registor 2
--                slv_reg2(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
--              end if;
--            end loop;
          when b"11" =>
```

1. Select the lines of code you want to comment
2. Right-click -> Select "Toggle Line Comments"

IP Block Creation

- Go back to `ssd_ctrl_v1_0.vhd`
- **Add the ports created in axi.vhd to the component**



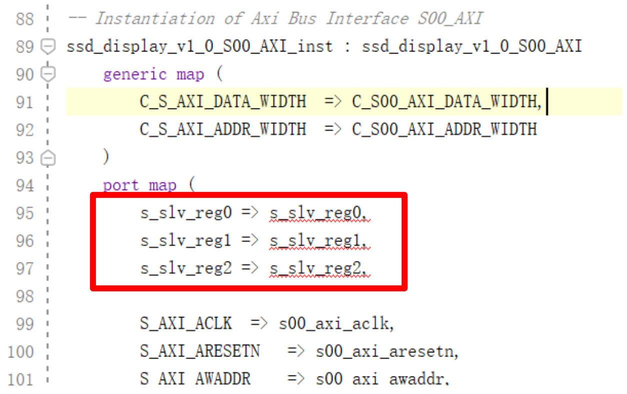
```
-- component declaration
component ssd_ctrl_v1_0_S00_AXI is
    generic (
    C_S_AXI_DATA_WIDTH  : integer := 32;
    C_S_AXI_ADDR_WIDTH  : integer := 4
    );
    port (
    s_slv_reg0 : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    s_slv_reg1 : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    s_slv_reg2 : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);

    S_AXI_ACLK  : in std_logic;
```

```
s_slv_reg0  :out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
s_slv_reg1  :out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
s_slv_reg2  :in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
```

You can copy this

IP Block Creation

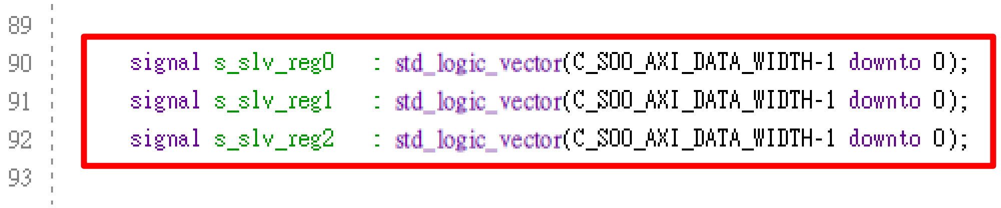
- **Then, component the port map as well**
  - **Don’t forget to create the signals called s_slv_reg0-3**



```
-- Instantiation of Axi Bus Interface S00_AXI
ssd_display_v1_0_S00_AXI_inst : ssd_display_v1_0_S00_AXI
    generic map (
        C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
        C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
    )
    port map (
        s_slv_reg0 => s_slv_reg0,
        s_slv_reg1 => s_slv_reg1,
        s_slv_reg2 => s_slv_reg2,

        S_AXI_ACLK  => s00_axi_aclk,
        S_AXI_ARESETN   => s00_axi_aresetn,
        S_AXI_AWADDR    => s00_axi_awaddr,
```

```
s_slv_reg0 => s_slv_reg0,
s_slv_reg1 => s_slv_reg1,
s_slv_reg2 => s_slv_reg2,
```

You can copy this

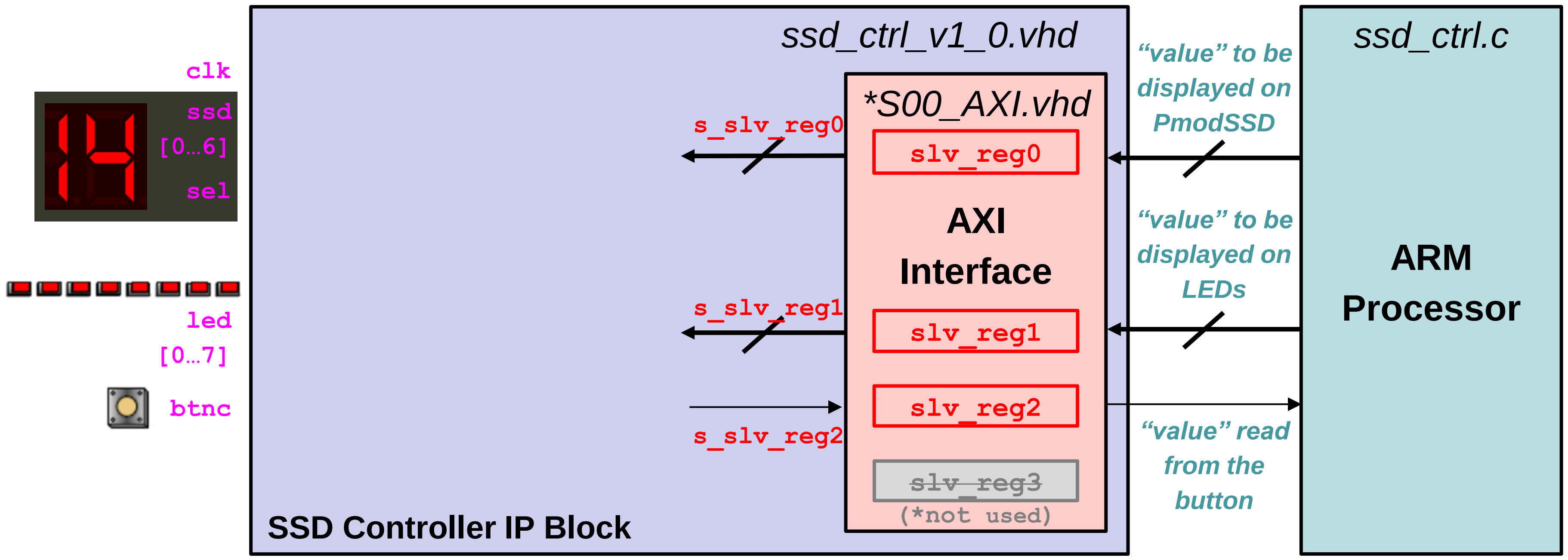


```
signal s_slv_reg0   : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
signal s_slv_reg1   : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
signal s_slv_reg2   : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
```

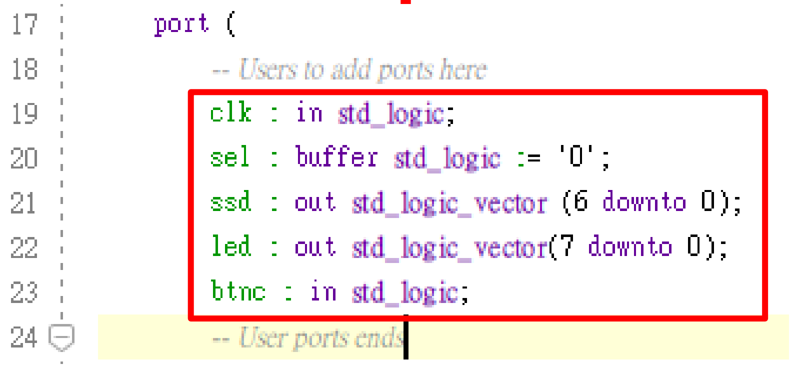
```
signal s_slv_reg0 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
signal s_slv_reg1 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
signal s_slv_reg2 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
```

You can copy this

IP Block Creation



- **Create the ports in the entity**



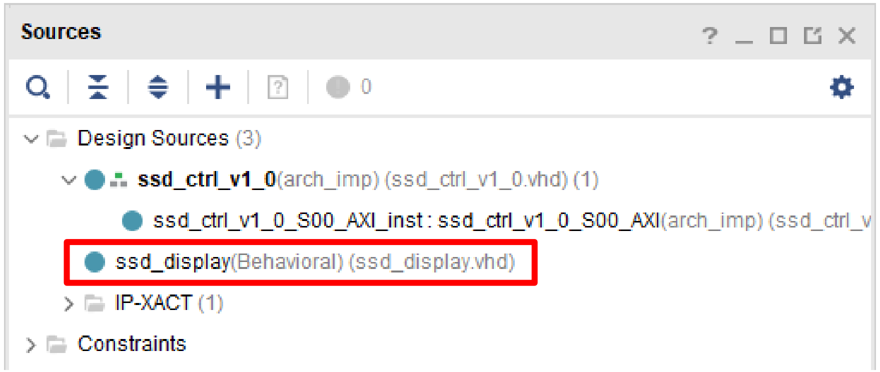
```
    port (
        -- Users to add ports here
        clk : in std_logic;
        sel : buffer std_logic := '0';
        ssd : out std_logic_vector (6 downto 0);
        led : out std_logic_vector(7 downto 0);
        btnc : in std_logic;
        -- User ports ends
```

```
clk : in std_logic;
sel : buffer std_logic := '0';
ssd : out std_logic_vector (6 downto 0);
led : out std_logic_vector (7 downto 0);
btnC : in std_logic;
```

You can copy this

Customize the AXI IP Block

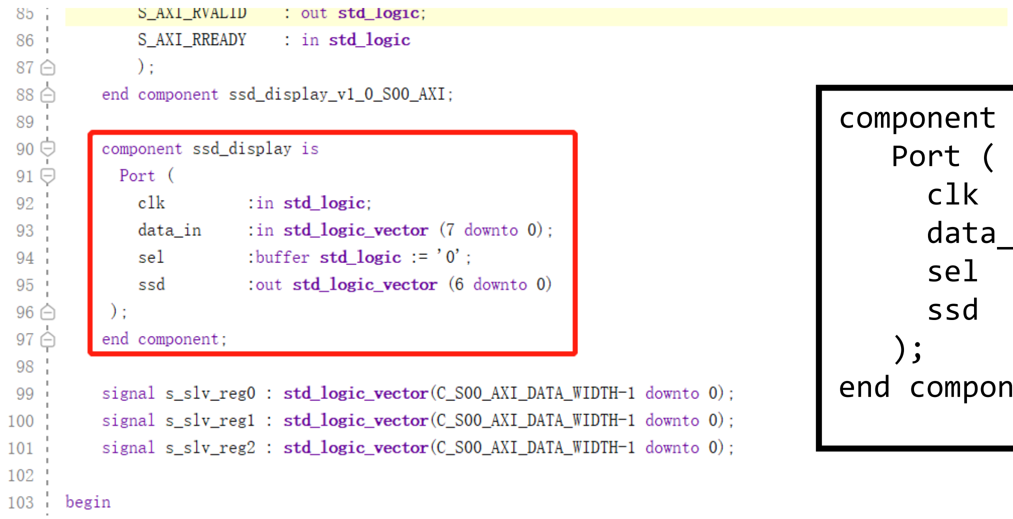
- We first **import the ssd_display.vhd file** (Download from Blackboard), then **create a component for it in ssd_ctrl_v1_0.vhd**



Sources

Design Sources (3)
- ssd_ctrl_v1_0(arch_imp) (ssd_ctrl_v1_0.vhd) (1)
  - ssd_ctrl_v1_0_S00_AXI_inst : ssd_ctrl_v1_0_S00_AXI(arch_imp) (ssd_ctrl_v
- ssd_display(Behavioral) (ssd_display.vhd)
- IP-XACT (1)

Constraints



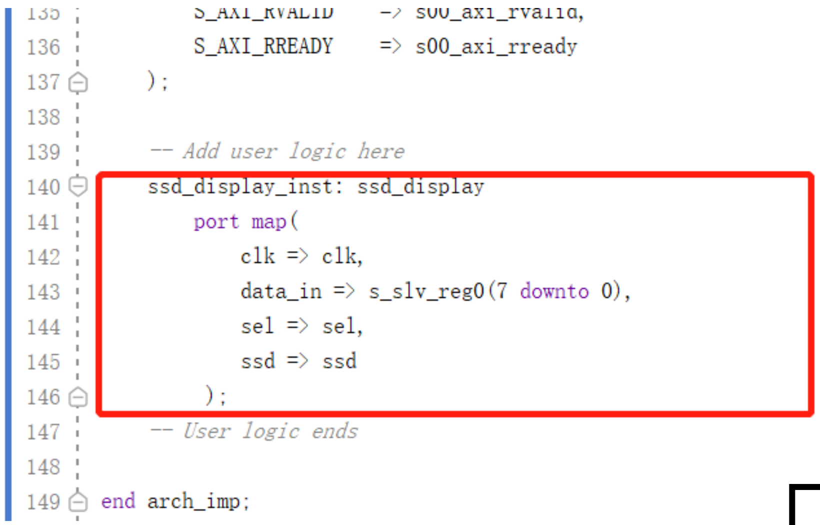
```
            S_AXI_RVALID    : out std_logic;
            S_AXI_RREADY    : in std_logic
            );
    end component ssd_display_v1_0_S00_AXI;

    component ssd_display is
      Port (
        clk         :in std_logic;
        data_in     :in std_logic_vector (7 downto 0);
        sel         :buffer std_logic := '0';
        ssd         :out std_logic_vector (6 downto 0)
      );
    end component;

    signal s_slv_reg0 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    signal s_slv_reg1 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
    signal s_slv_reg2 : std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);

begin
```

```
component ssd_display is
   Port (
     clk          :in std_logic;
     data_in      :in std_logic_vector (7 downto 0);
     sel          :buffer std_logic := '0';
     ssd          :out std_logic_vector (6 downto 0)
   );
end component;
```

You can copy this

Customize the AXI IP Block

- Then also **port map** it
  - (data_in only have 8-bit)

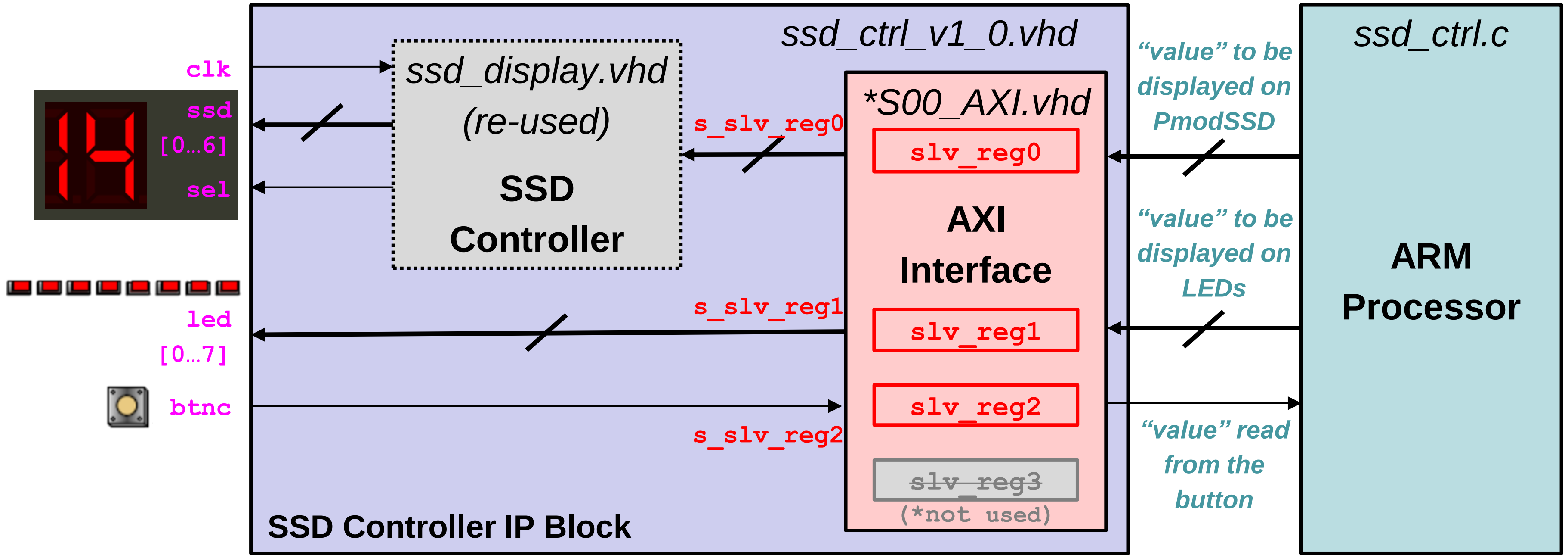


```
            S_AXI_RVALID    => s00_axi_rvalid,
            S_AXI_RREADY    => s00_axi_rready
        );

        -- Add user logic here
        ssd_display_inst: ssd_display
            port map(
                clk => clk,
                data_in => s_slv_reg0(7 downto 0),
                sel => sel,
                ssd => ssd
             );
        -- User logic ends

    end arch_imp;
```

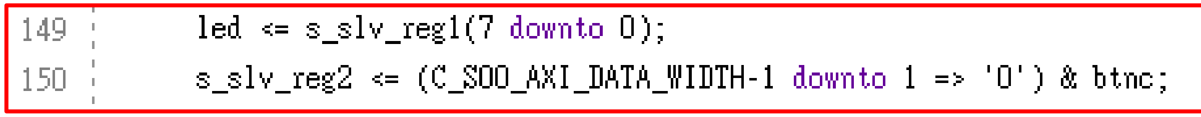
```
ssd_display_inst: ssd_display
   port map(
       clk => clk,
       data_in => s_slv_reg0(7 downto 0),
       sel => sel,
       ssd => ssd
    );
```

You can copy this

IP Block Creation



- Finally, **connect the led and btn to the s_slv_reg1-2** in the same file
  - For s_slv_reg2, we concatenate thirty-one '0' and btnc to make it 32-bit



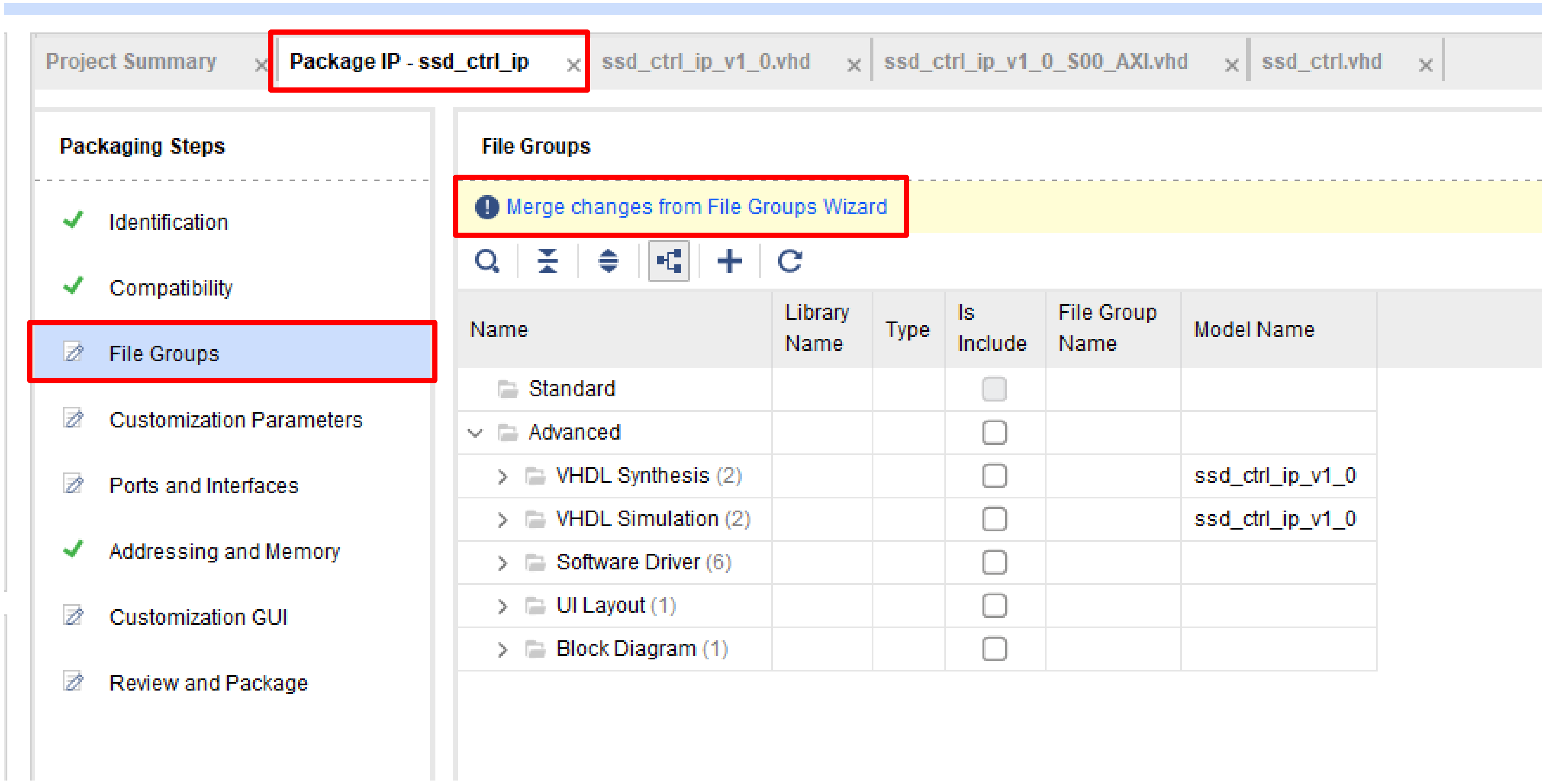
```
led <= s_slv_reg1(7 downto 0);
s_slv_reg2 <= (C_S00_AXI_DATA_WIDTH-1 downto 1 => '0') & btnc;
```

```
led <= s_slv_reg1(7 downto 0);
s_slv_reg2 <= (C_S00_AXI_DATA_WIDTH-1 downto 1 => '0') & btnc;
```

You can copy this

IP Block Creation

- Click "**Package IP**" and "**File Groups**"
  - Select "**Merge changes from File Groups Wizard**"



Project Summary | Package IP - ssd_ctrl_ip | ssd_ctrl_ip_v1_0.vhd | ssd_ctrl_ip_v1_0_S00_AXI.vhd | ssd_ctrl.vhd

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- File Groups
- Customization Parameters
- Ports and Interfaces
- ✓ Addressing and Memory
- Customization GUI
- Review and Package

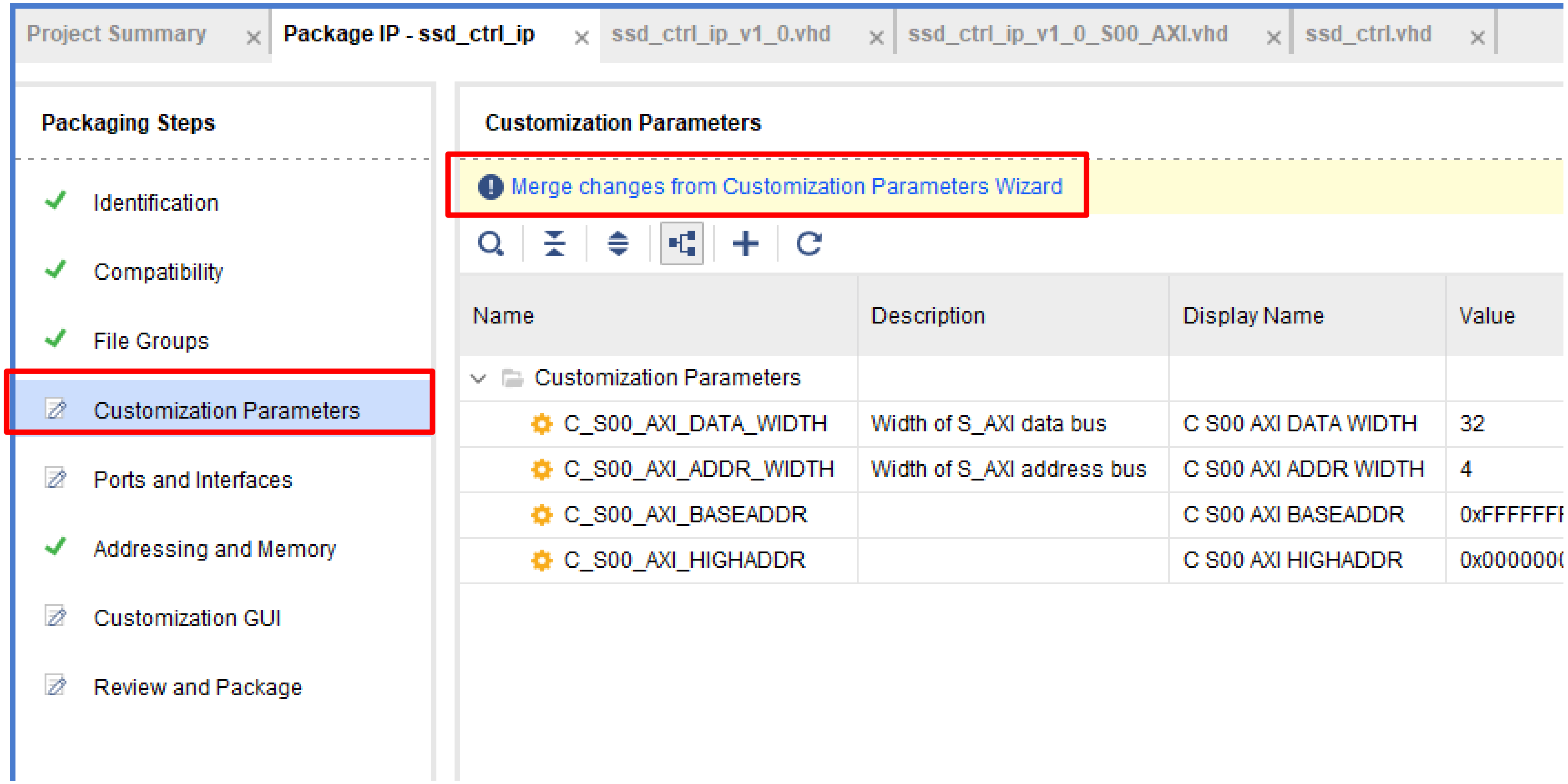
File Groups

Merge changes from File Groups Wizard

Name	Library Name	Type	Is Include	File Group Name	Model Name
Standard			☐		
Advanced			☐		
VHDL Synthesis (2)			☐		ssd_ctrl_ip_v1_0
VHDL Simulation (2)			☐		ssd_ctrl_ip_v1_0
Software Driver (6)			☐		
UI Layout (1)			☐		
Block Diagram (1)			☐		

IP Block Creation

- Similarly, Click "Customization Parameters"
  - Select "Merge changes from Customization Parameters Wizard"



Project Summary | Package IP - ssd_ctrl_ip | ssd_ctrl_ip_v1_0.vhd | ssd_ctrl_ip_v1_0_S00_AXI.vhd | ssd_ctrl.vhd

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- Customization Parameters
- Ports and Interfaces
- ✓ Addressing and Memory
- Customization GUI
- Review and Package

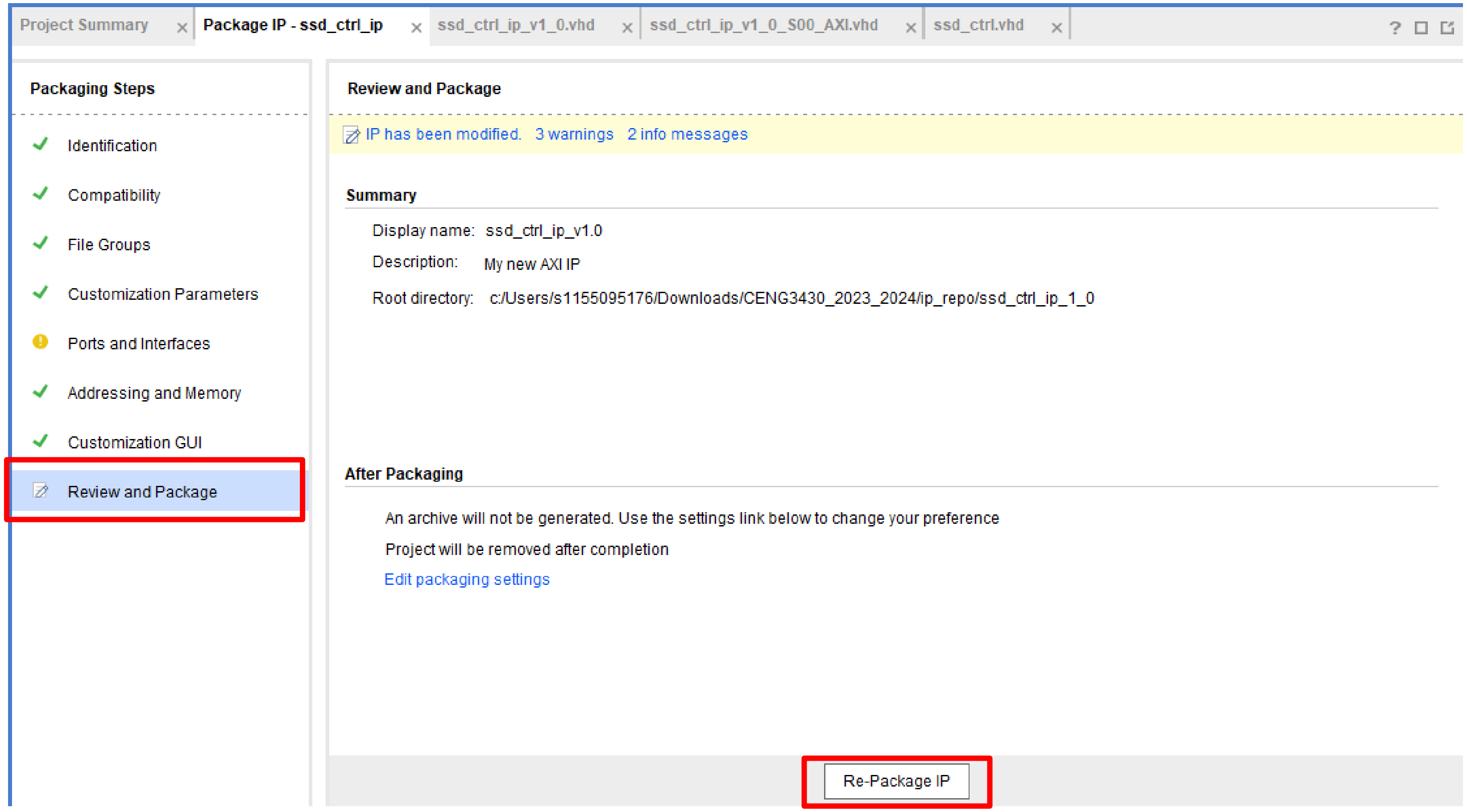
Customization Parameters

Merge changes from Customization Parameters Wizard

Name	Description	Display Name	Value
Customization Parameters			
C_S00_AXI_DATA_WIDTH	Width of S_AXI data bus	C S00 AXI DATA WIDTH	32
C_S00_AXI_ADDR_WIDTH	Width of S_AXI address bus	C S00 AXI ADDR WIDTH	4
C_S00_AXI_BASEADDR		C S00 AXI BASEADDR	0xFFFFFFF
C_S00_AXI_HIGHADDR		C S00 AXI HIGHADDR	0x0000000

IP Block Creation

- Finally, click "Review and Package"
- And click "Re-Package IP"



Review and Package

IP has been modified. 3 warnings 2 info messages

Summary

Display name: ssd_ctrl_ip_v1.0

Description: My new AXI IP

Root directory: c:/Users/s1155095176/Downloads/CENG3430_2023_2024/ip_repo/ssd_ctrl_ip_1_0

After Packaging

An archive will not be generated. Use the settings link below to change your preference

Project will be removed after completion

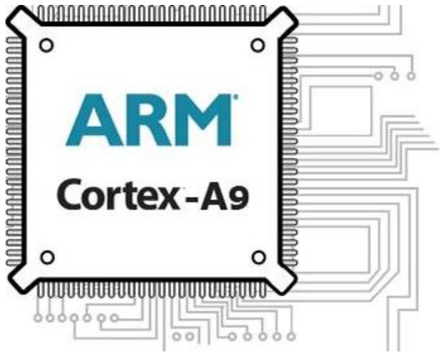
Edit packaging settings

- **PART 1: IP Block Design**

AMD Vivado

① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

&



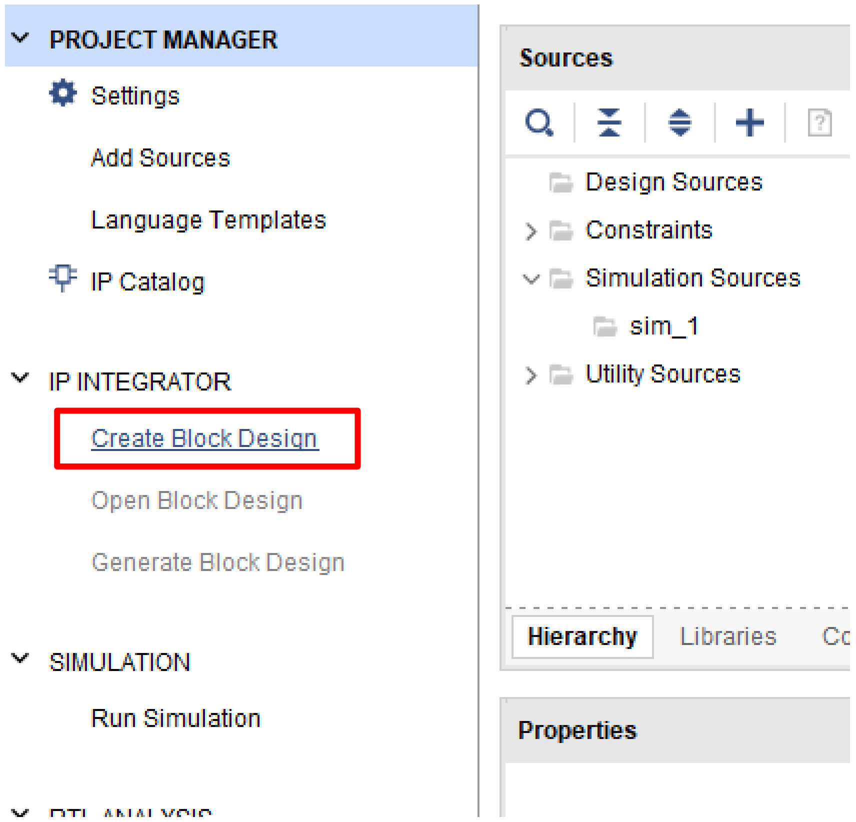
- **PART 2: ARM Programming**

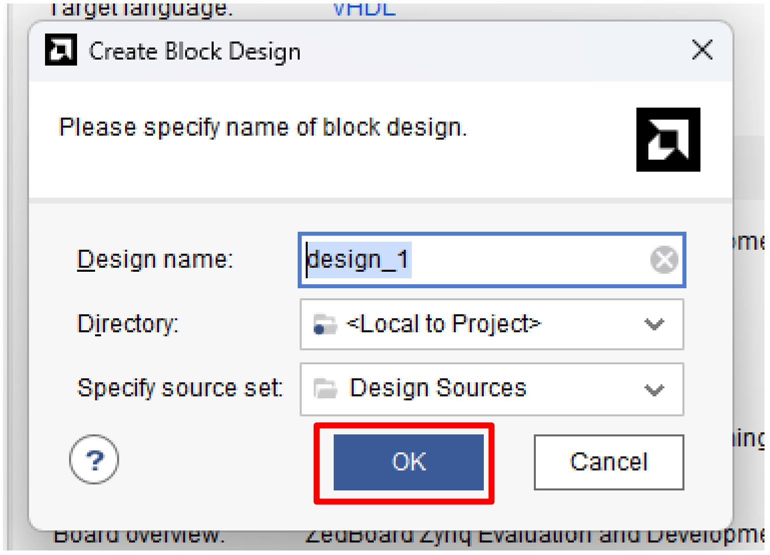
AMD Vitis

⑤ ARM Programming
⑥ Launch on Hardware

IP Integration

- Click "**Create Block Design**"
- Then, Click "**OK**"




Create Block Design

Please specify name of block design.

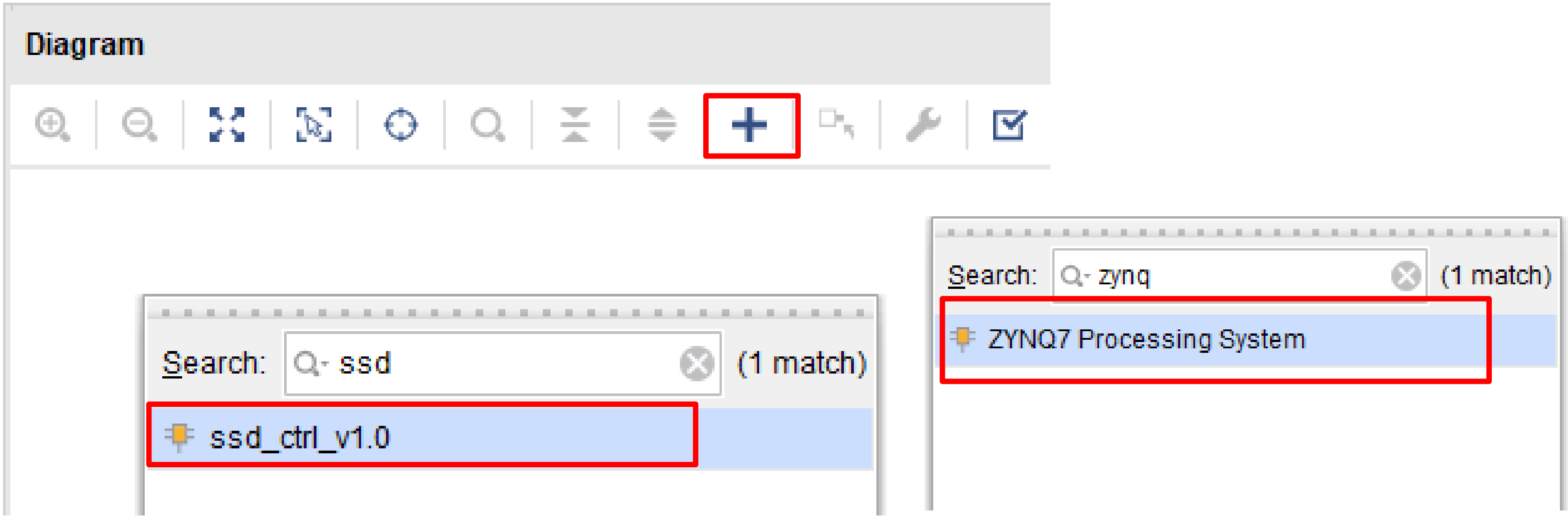
Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

IP Integration

- Click “Add IP”
- Search “ssd_display” and select “**ssd_display_v1.0**”
- Also, search “zynq” and select “**ZYNQ7 Processing System**”

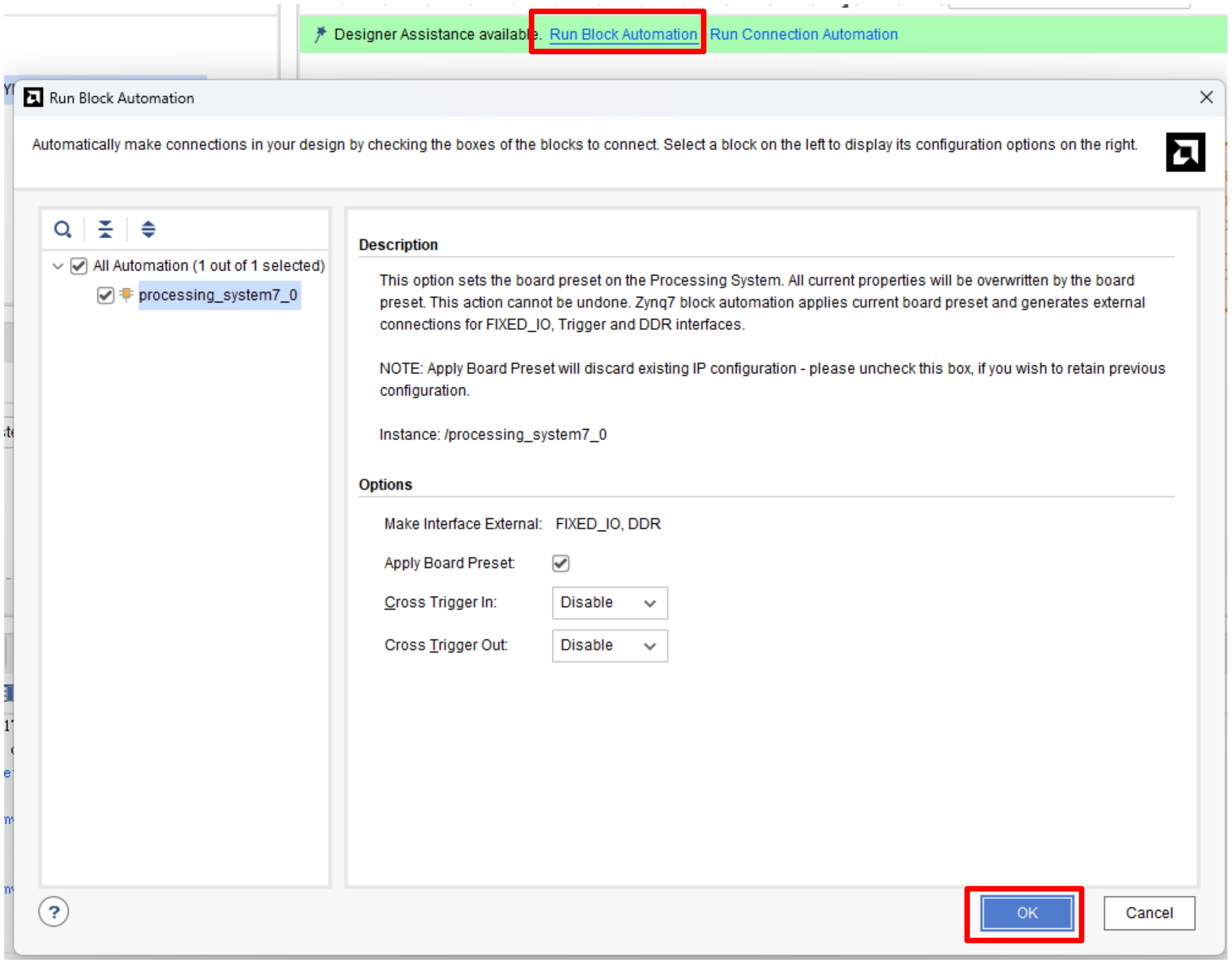


Search: ssd (1 match)
ssd_ctrl_v1.0

Search: zynq (1 match)
ZYNQ7 Processing System

IP Integration

- Click "Run Block Automation" and click "OK"



Designer Assistance available. Run Block Automation Run Connection Automation

Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 out of 1 selected)
processing_system7_0

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

Make Interface External: FIXED_IO, DDR

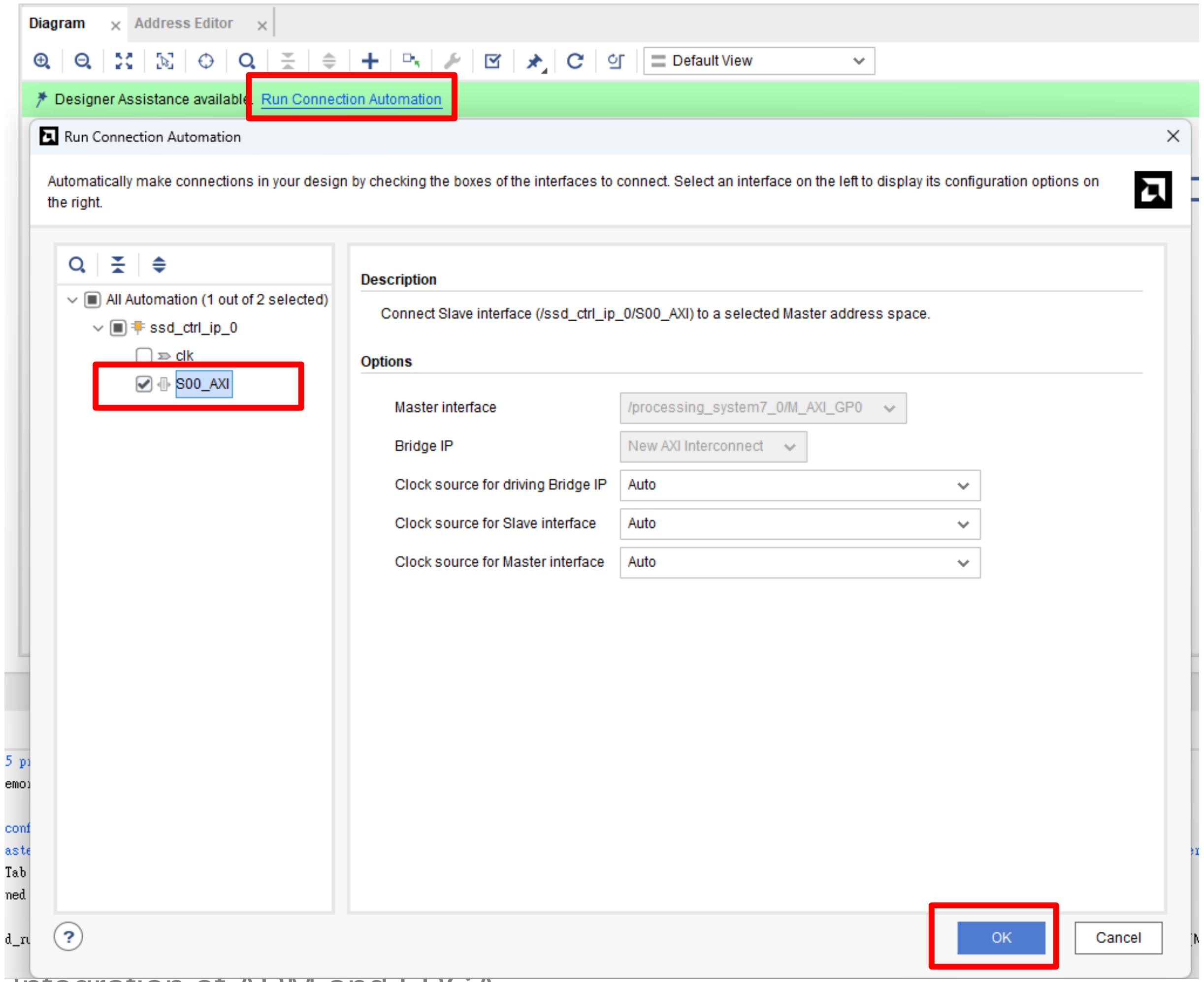
Apply Board Preset: ✓

Cross Trigger In: Disable

Cross Trigger Out: Disable

IP Integration

- Click “**Run Connection Automation**”, Select “**S_AXI**” on the left, and click “OK”



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- All Automation (1 out of 2 selected)
  - ssd_ctrl_ip_0
    - clk
    - S00_AXI

Description

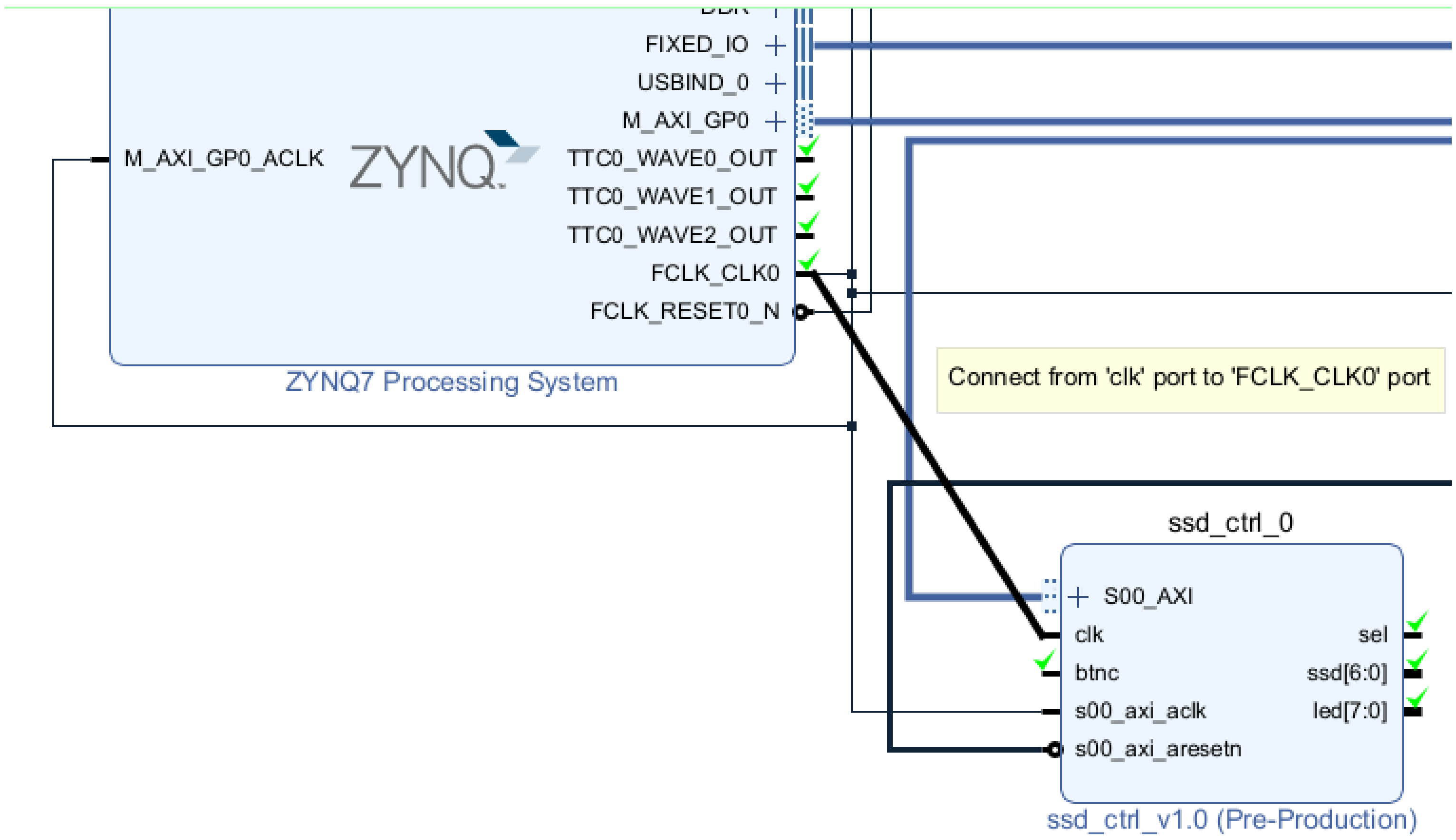
Connect Slave interface (/ssd_ctrl_ip_0/S00_AXI) to a selected Master address space.

Options

Master interface	/processing_system7_0/M_AXI_GP0
Bridge IP	New AXI Interconnect
Clock source for driving Bridge IP	Auto
Clock source for Slave interface	Auto
Clock source for Master interface	Auto

IP Integration

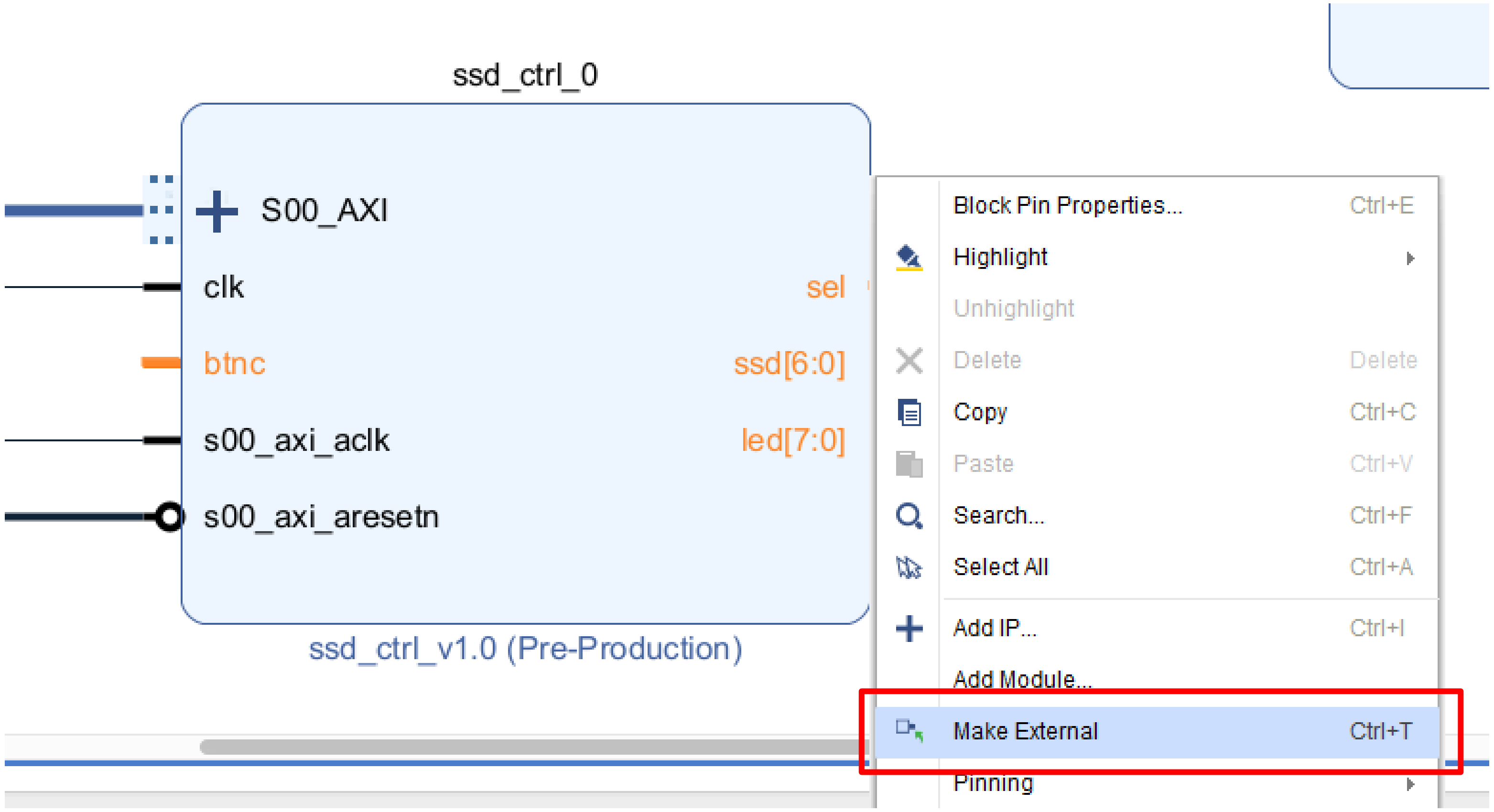
- **Drag the clk from ssd_ctrl IP to FCLK_CLK0 from ZYNQ IP**



Connect from 'clk' port to 'FCLK_CLK0' port

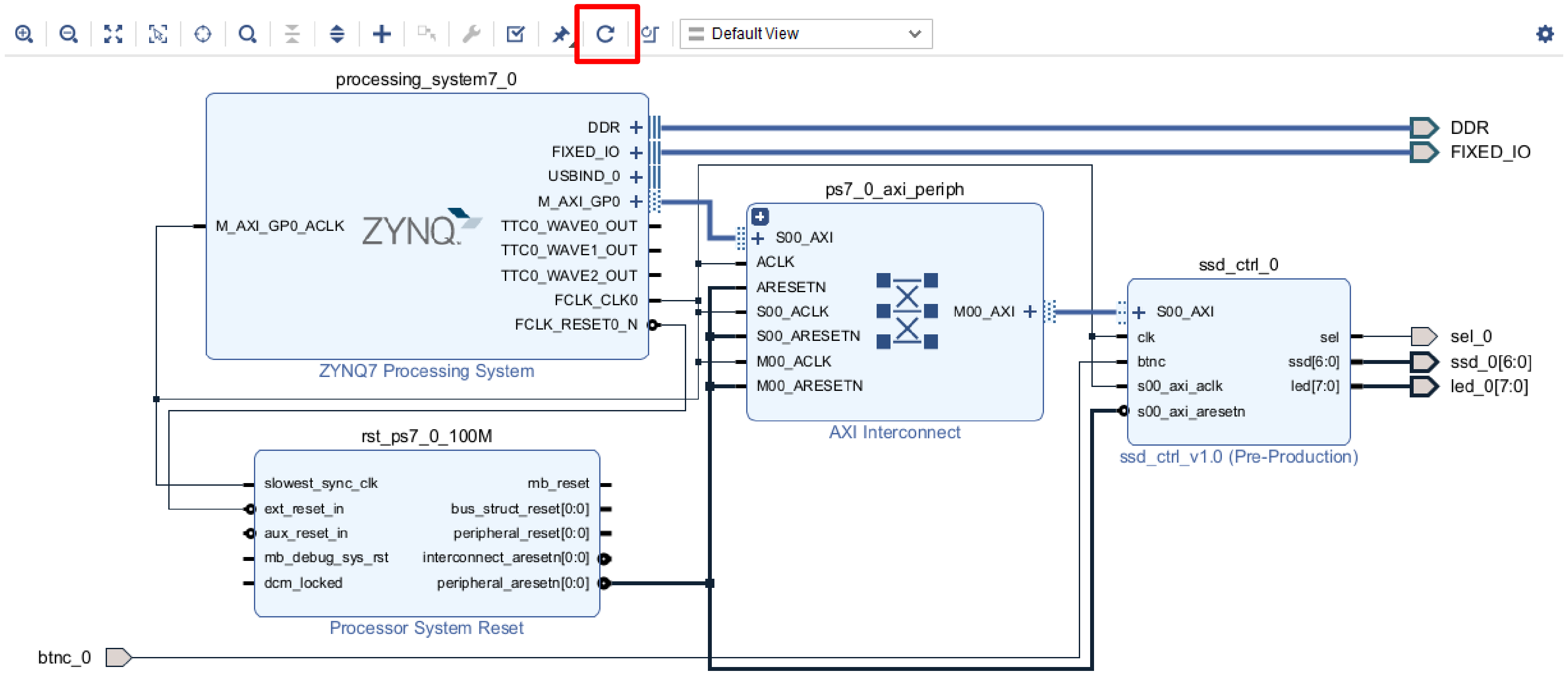
IP Integration

- Select btnc, sel, ssd, and led
- Right-click and select “Make External



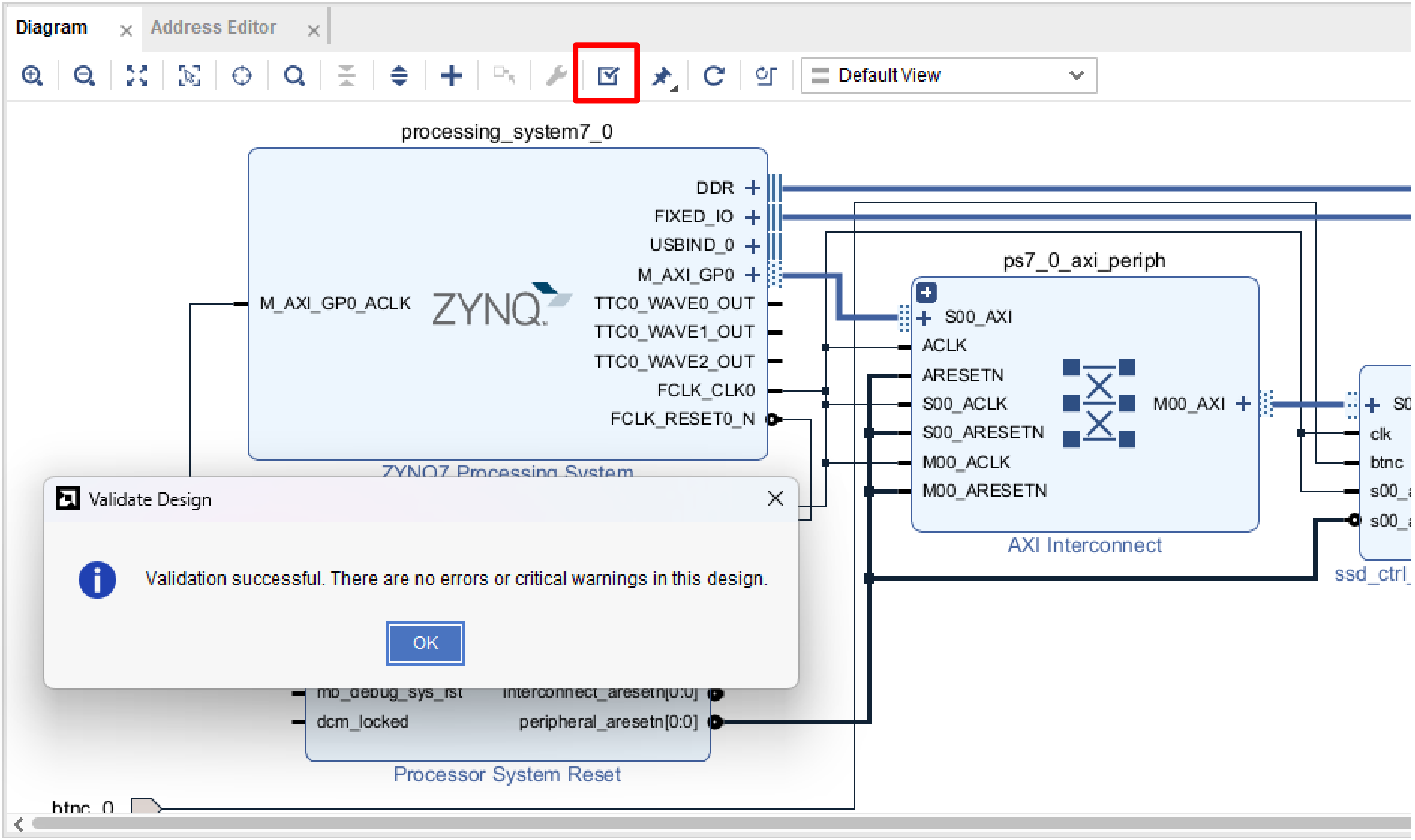
IP Integration

- Click "Regenerate Layout"
- The block diagram should look like this



IP Integration

- **Click “Validate Design”**
- **The result should have no errors**



Validate Design

Validation successful. There are no errors or critical warnings in this design.

OK

- **PART 1: IP Block Design**

AMD Vivado

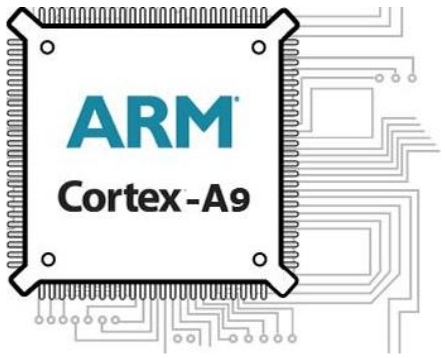
① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

&

- **PART 2: ARM Programming**

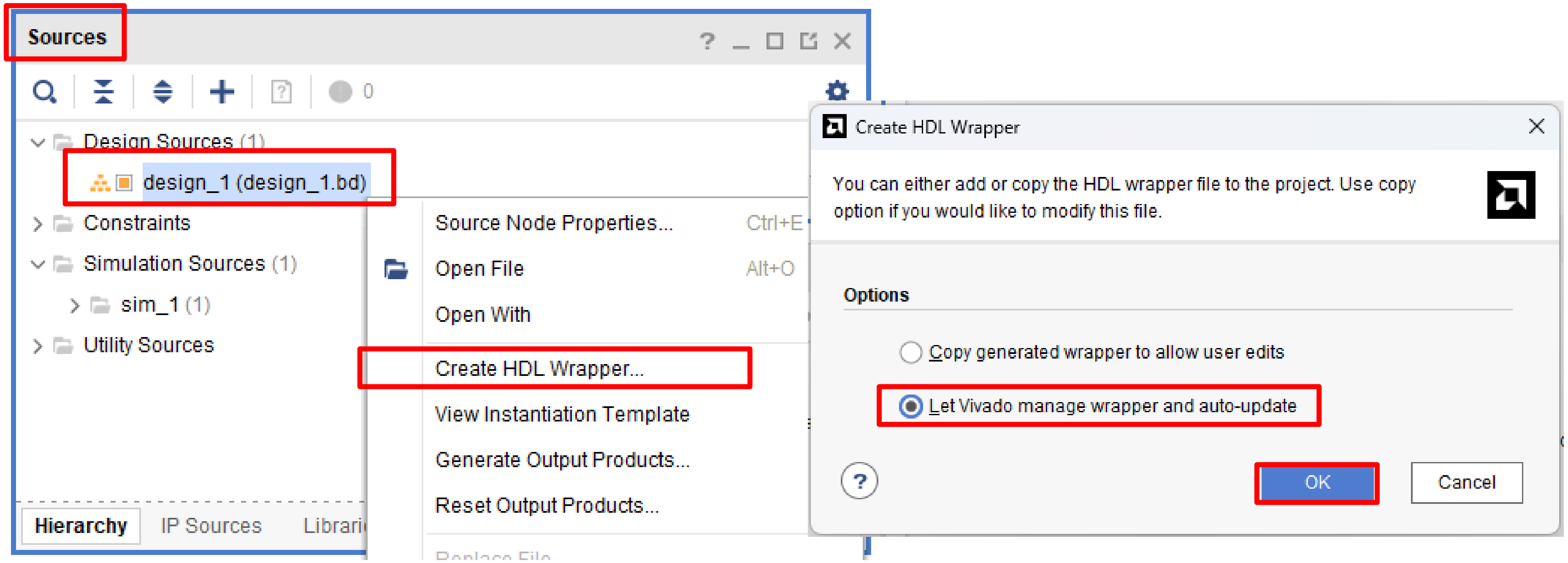
AMD Vitis

⑤ ARM Programming
⑥ Launch on Hardware



HDL Wrapper & Generate Bitstream

- **Right click the "design_1" in Design Source**
- **Select "Create HDL Wrapper"**
- **Let the Vivado create it automatically**



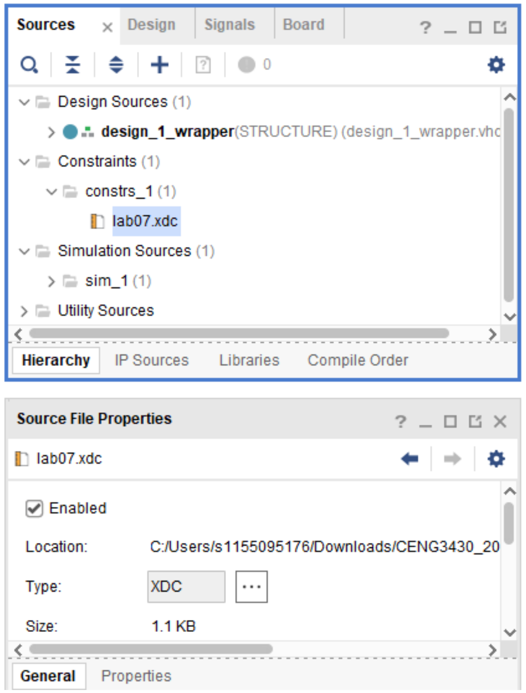
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

Copy generated wrapper to allow user edits [✓ Let Vivado manage wrapper and auto-update]

HDL Wrapper & Generate Bitstream

- We have provided the lab07.xdc (Download from blackboard)
- **Import the lab07.xdc**



- Design Sources (1)
  - design_1_wrapper(STRUCTURE) (design_1_wrapper.vhc
- Constraints (1)
  - constrs_1 (1)
    - lab07.xdc
- Simulation Sources (1)
  - sim_1 (1)
- Utility Sources

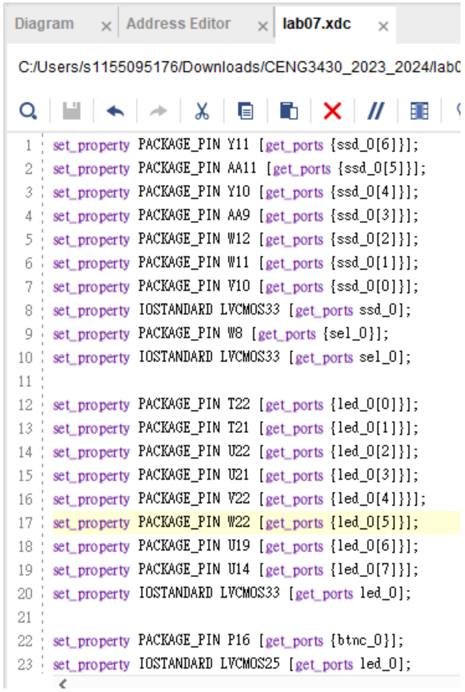
lab07.xdc

Enabled

Location: C:/Users/s1155095176/Downloads/CENG3430_20

Type: XDC

Size: 1.1 KB



C:/Users/s1155095176/Downloads/CENG3430_2023_2024/lab0

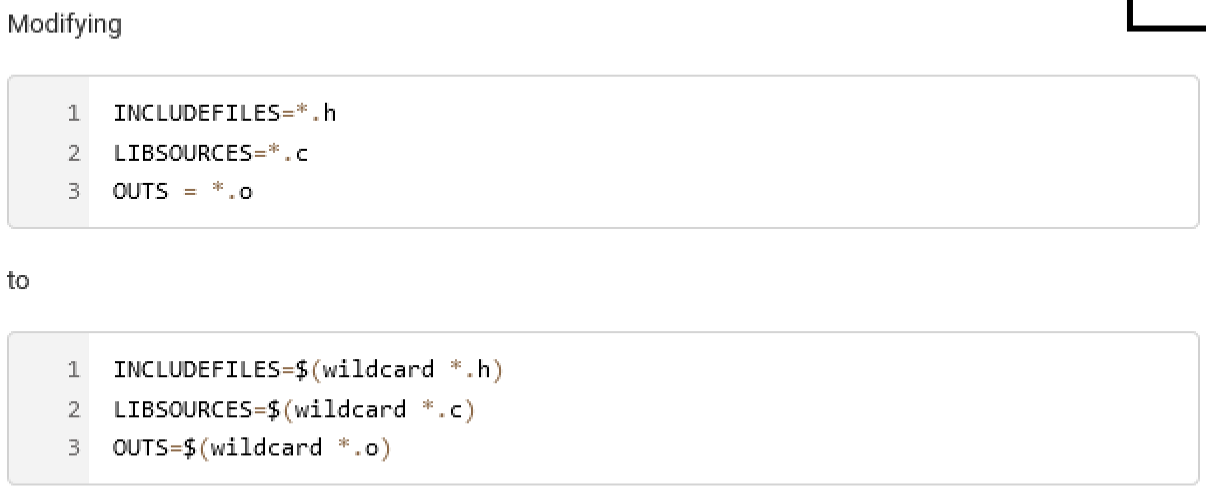
```
set_property PACKAGE_PIN Y11 [get_ports {ssd_0[6]}];
set_property PACKAGE_PIN AA11 [get_ports {ssd_0[5]}];
set_property PACKAGE_PIN Y10 [get_ports {ssd_0[4]}];
set_property PACKAGE_PIN AA9 [get_ports {ssd_0[3]}];
set_property PACKAGE_PIN W12 [get_ports {ssd_0[2]}];
set_property PACKAGE_PIN W11 [get_ports {ssd_0[1]}];
set_property PACKAGE_PIN V10 [get_ports {ssd_0[0]}];
set_property IOSTANDARD LVCMOS33 [get_ports ssd_0];
set_property PACKAGE_PIN W8 [get_ports {sel_0}];
set_property IOSTANDARD LVCMOS33 [get_ports sel_0];

set_property PACKAGE_PIN T22 [get_ports {led_0[0]}];
set_property PACKAGE_PIN T21 [get_ports {led_0[1]}];
set_property PACKAGE_PIN U22 [get_ports {led_0[2]}];
set_property PACKAGE_PIN U21 [get_ports {led_0[3]}];
set_property PACKAGE_PIN V22 [get_ports {led_0[4]}}];
set_property PACKAGE_PIN W22 [get_ports {led_0[5]}];
set_property PACKAGE_PIN U19 [get_ports {led_0[6]}];
set_property PACKAGE_PIN U14 [get_ports {led_0[7]}];
set_property IOSTANDARD LVCMOS33 [get_ports led_0];

set_property PACKAGE_PIN P16 [get_ports {btnc_0}];
set_property IOSTANDARD LVCMOS25 [get_ports led_0];
```

HDL Wrapper & Generate Bitstream

- As vivado 2023 has a bug, we need to modify a file by ourselves
- At ip_repo\ssd_ctrl_1_0\drivers\ssd_ctrl_v1_0\src\
  - ip_repo is created at here
- Modify the Makefile

```
INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS = $(wildcard *.o)
```

You can copy this



Modifying

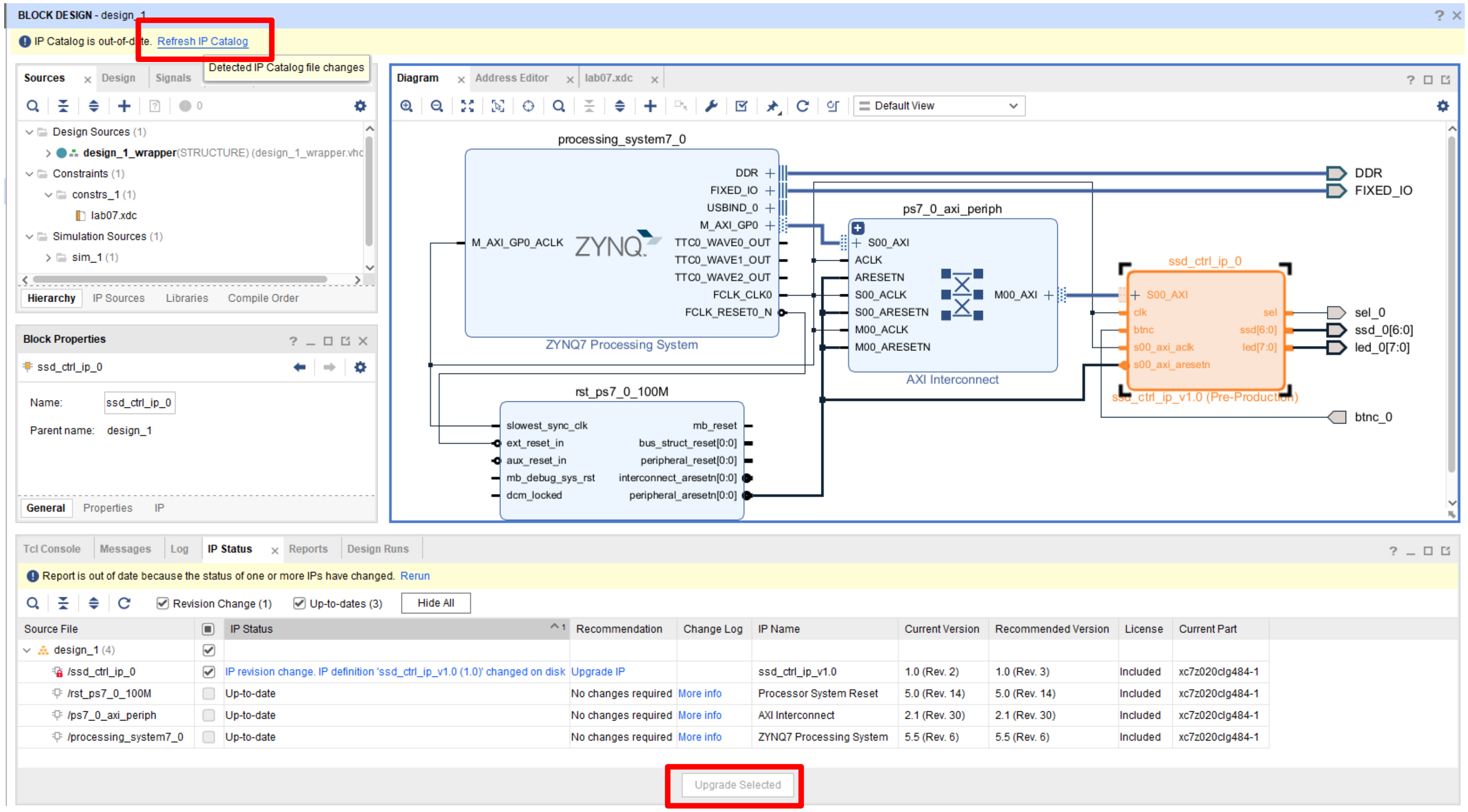
```
INCLUDEFILES=*.h
LIBSOURCES=*.c
OUTS = *.o
```

to

```
INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS=$(wildcard *.o)
```

HDL Wrapper & Generate Bitstream

- Back to Vivado, Click "Reflesh IP Catalog"
- Click "Update Selected"



IP Catalog is out-of-date. Refresh IP Catalog

Report is out of date because the status of one or more IPs have changed. Rerun

Source File		IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
design_1 (4)	✓								
/ssd_ctrl_ip_0	✓	IP revision change. IP definition 'ssd_ctrl_ip_v1.0 (1.0)' changed on disk	Upgrade IP		ssd_ctrl_ip_v1.0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7z020clg484-1
/rst_ps7_0_100M		Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 14)	5.0 (Rev. 14)	Included	xc7z020clg484-1
/ps7_0_axi_periph		Up-to-date	No changes required	More info	AXI Interconnect	2.1 (Rev. 30)	2.1 (Rev. 30)	Included	xc7z020clg484-1
/processing_system7_0		Up-to-date	No changes required	More info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020clg484-1

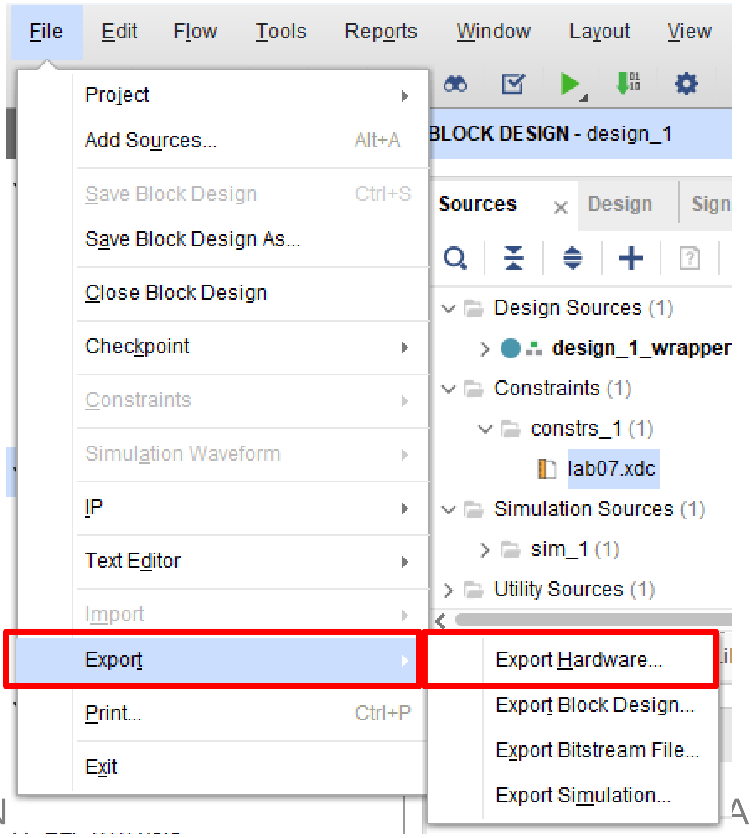
HDL Wrapper & Generate Bitstream

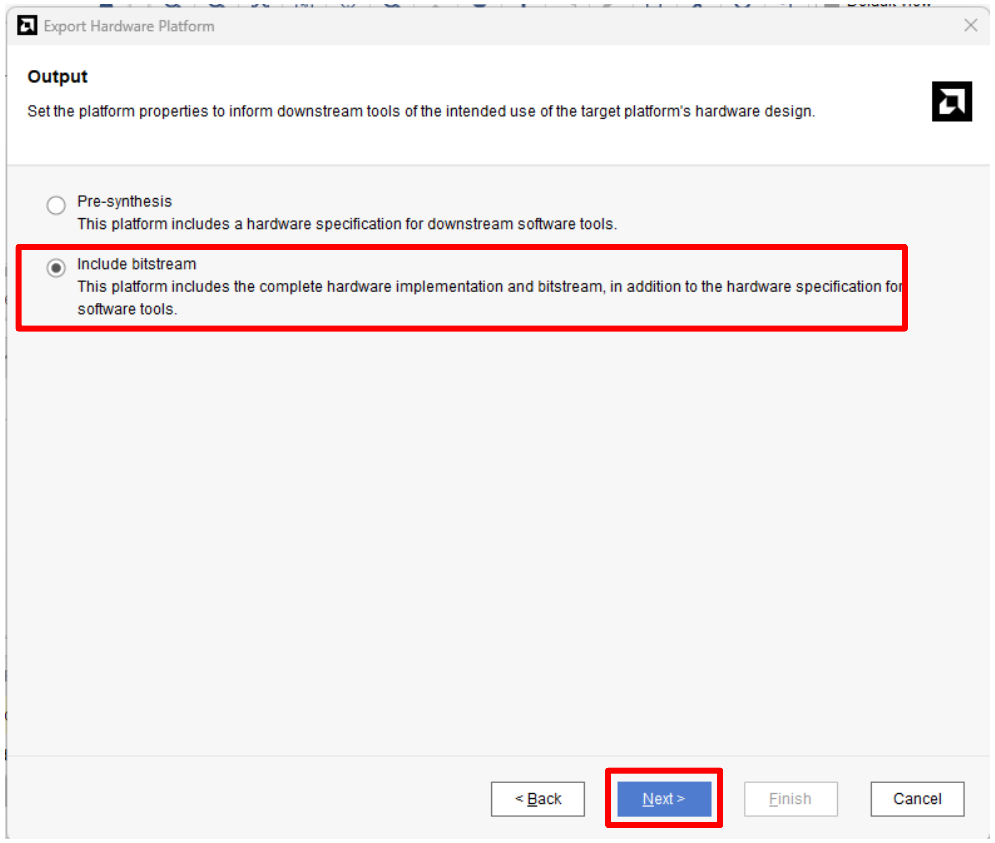
- **Run "Generate Bitstream"**

- **If you encounter any errors, please see Modifying the IP Block (Just in Case)**
  - **Except for the last step**

HDL Wrapper & Generate Bitstream

- **Then click “File -> Export”**
- **Click “Export Hardware”**
  - **Include the bitstream**




Export Hardware Platform

Output

Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

( ) Pre-synthesis
This platform includes a hardware specification for downstream software tools.

(●) Include bitstream
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

Lab07 Outline

- **PART 1: IP Block Design**

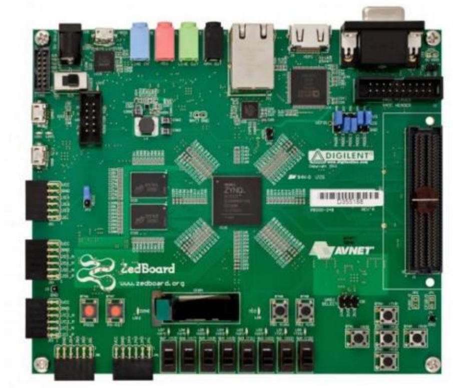
AMD Vivado

① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

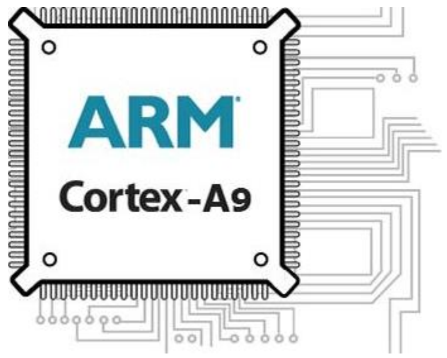
- **PART 2: ARM Programming**

AMD Vitis

⑤ ARM Programming (Set up Environment)
⑥ Launch on Hardware

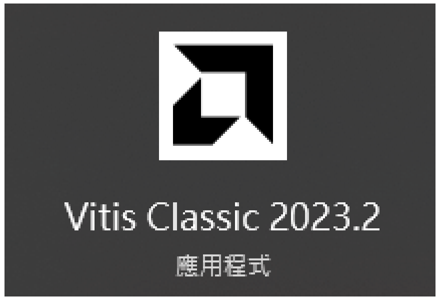


&

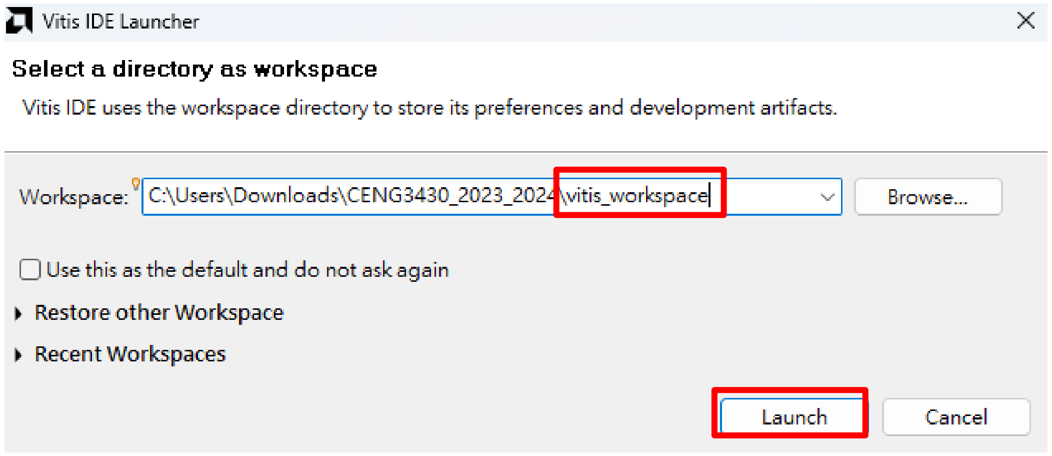


Set Up Environment

- Then open the "Vitis Classic 2023.2"



- Create a "vitis_workspace" folder and Launch


Vitis IDE Launcher

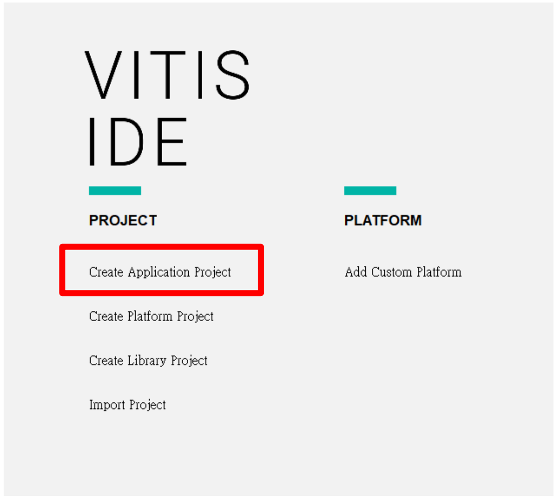
Select a directory as workspace

Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Downloads\CENG3430_2023_2024\vitis_workspace

Set Up Environment

- Click "**Create Application Project**"
- Click "**Create a new platform from hardware (XSA)**", click "**Browse**"
- Select the "**design1_wrapper.xsa**" **in the lab07 projcet folder**


VITIS IDE

PROJECT

Create Application Project

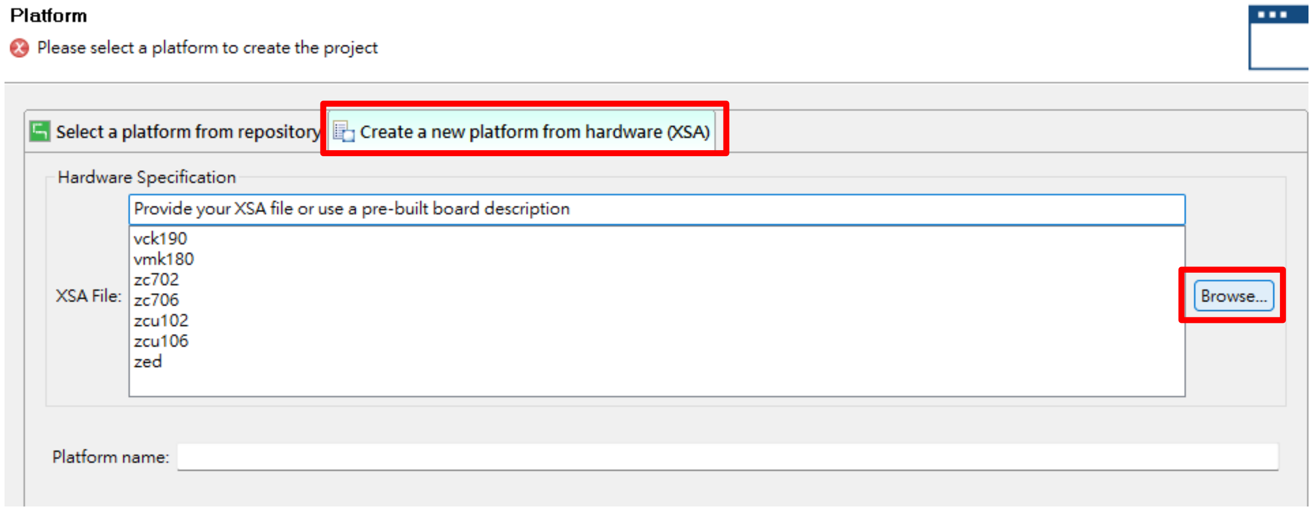
Create Platform Project

Create Library Project

Import Project

PLATFORM

Add Custom Platform


Platform

Please select a platform to create the project

Select a platform from repository | Create a new platform from hardware (XSA)

Hardware Specification

Provide your XSA file or use a pre-built board description

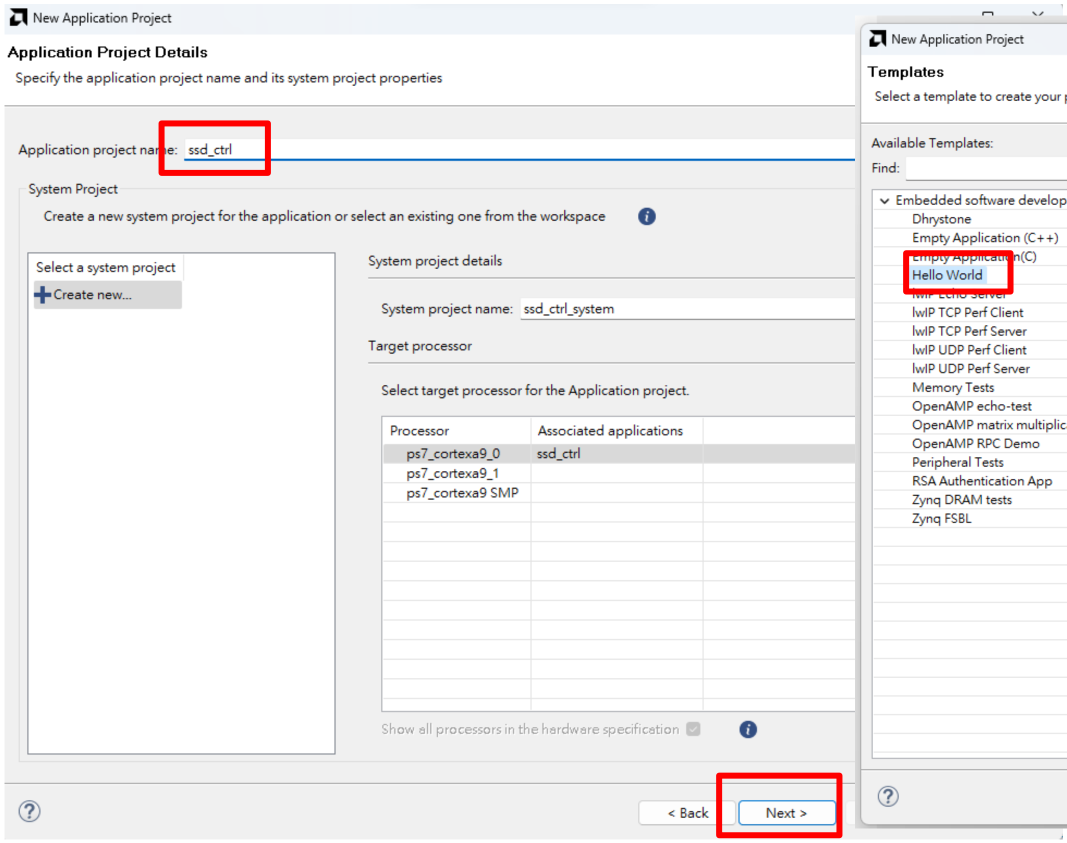
XSA File:
vck190
vmk180
zc702
zc706
zcu102
zcu106
zed

Browse...

Platform name:

Set Up Environment

- Set the project name "ssd_ctrl"
- Finally select "Hello World" templates and click "Finish"


New Application Project

Application Project Details

Specify the application project name and its system project properties

Application project name: ssd_ctrl

System Project

Create a new system project for the application or select an existing one from the workspace

Select a system project

Create new...

System project details

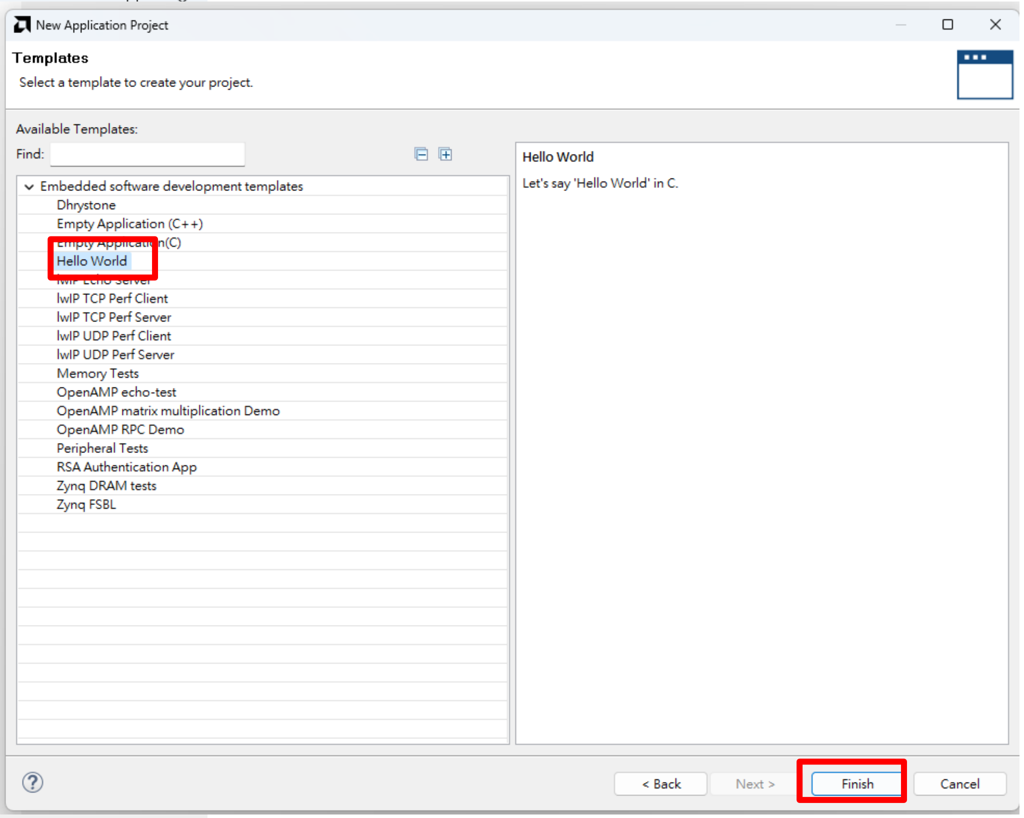
System project name: ssd_ctrl_system

Target processor

Select target processor for the Application project.

Processor	Associated applications
ps7_cortexa9_0	ssd_ctrl
ps7_cortexa9_1	
ps7_cortexa9 SMP	

Show all processors in the hardware specification


New Application Project

Templates

Select a template to create your project.

Available Templates:

Find:

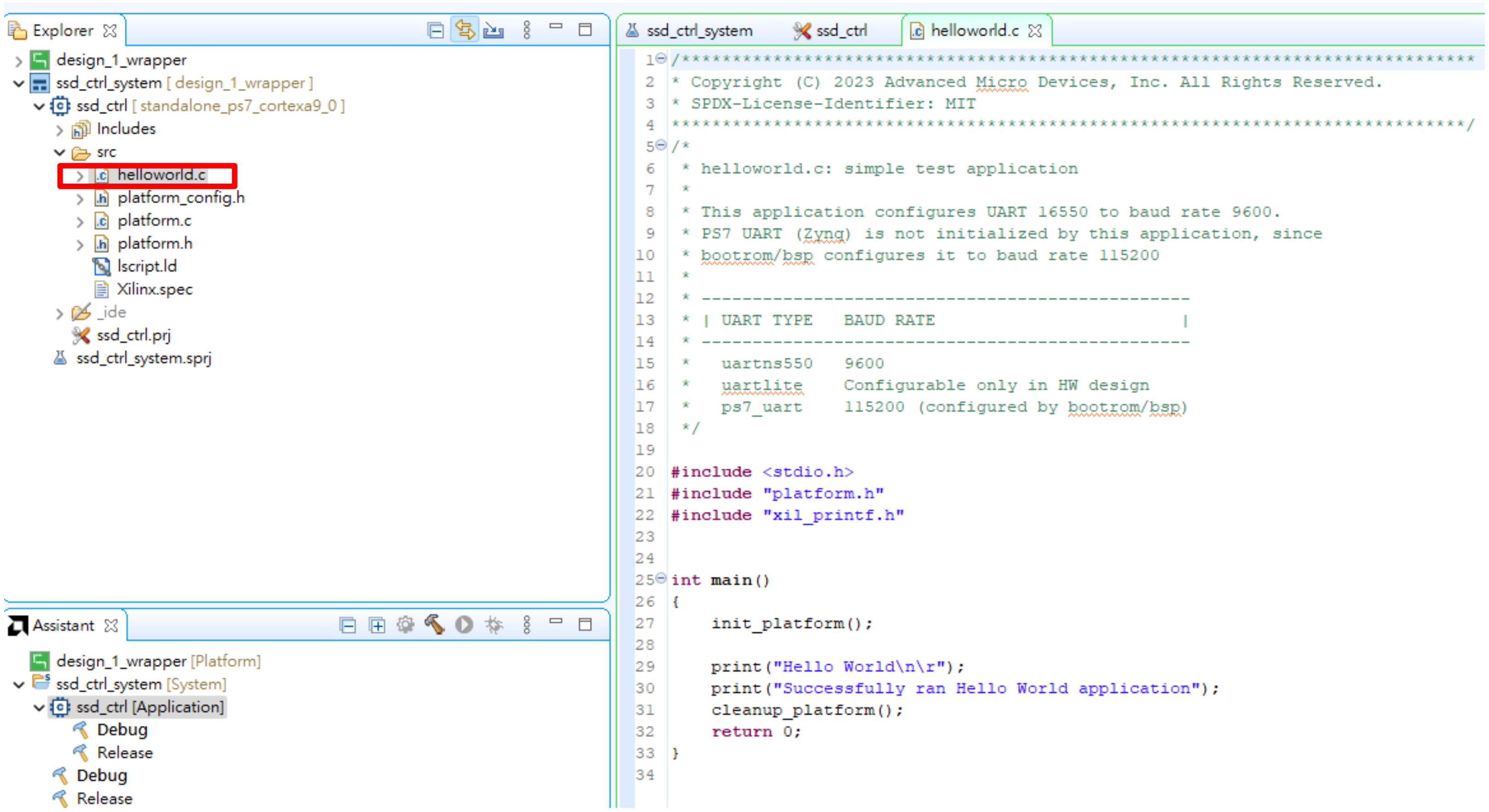
- Embedded software development templates
  - Dhrystone
  - Empty Application (C++)
  - Empty Application(C)
  - Hello World
  - lwIP Echo Server
  - lwIP TCP Perf Client
  - lwIP TCP Perf Server
  - lwIP UDP Perf Client
  - lwIP UDP Perf Server
  - Memory Tests
  - OpenAMP echo-test
  - OpenAMP matrix multiplication Demo
  - OpenAMP RPC Demo
  - Peripheral Tests
  - RSA Authentication App
  - Zynq DRAM tests
  - Zynq FSBL

Hello World

Let's say 'Hello World' in C.

Set Up Environment

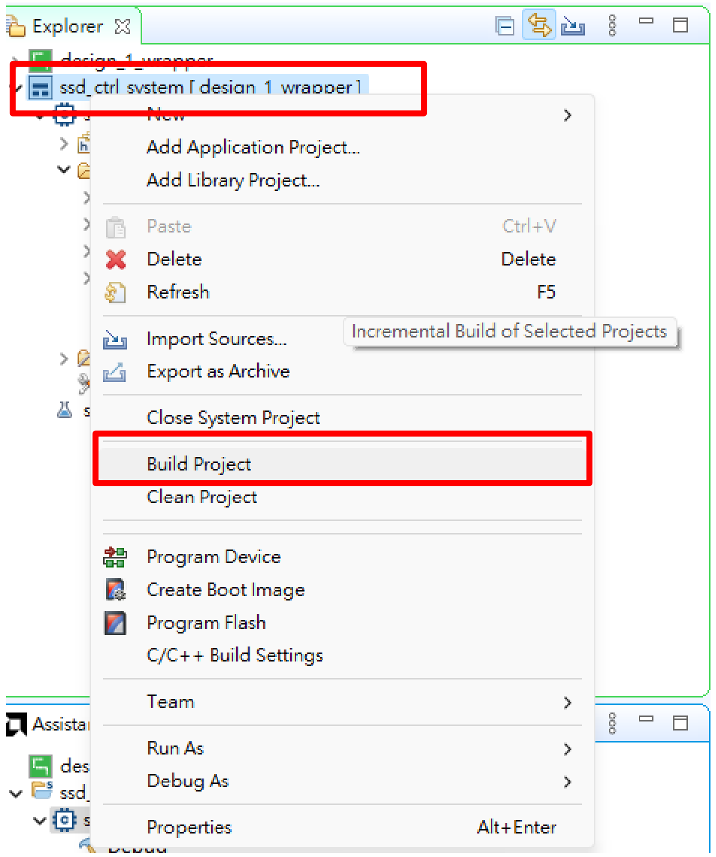
- **Open the “helloworld.c” in Explorer “ssd_ctrl_system -> ssd_ctrl -> src”**



```c
/******************************************************************************
* Copyright (C) 2023 Advanced Micro Devices, Inc. All Rights Reserved.
* SPDX-License-Identifier: MIT
******************************************************************************/
/*
 * helloworld.c: simple test application
 *
 * 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");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

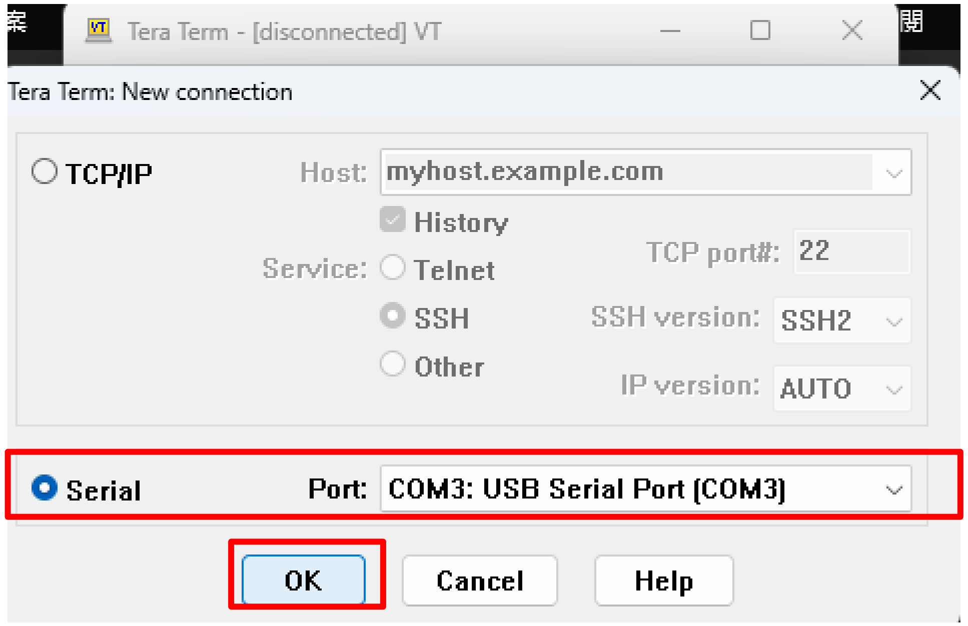
Set Up Environment

- Do a simple test using the hello world program
- Right click "ssd_ctrl_system" and **select "Build Project"**



Set Up Environment

- Open “Tera Term” and select “Serial”
- Choose the right port
  - You could turn off and on the Zedboard to find out which COM is from the UART



Tera Term - [disconnected] VT

Tera Term: New connection

TCP/IP Host: myhost.example.com

[✓] History

TCP port#: 22

Service: Telnet [✓ SSH] Other

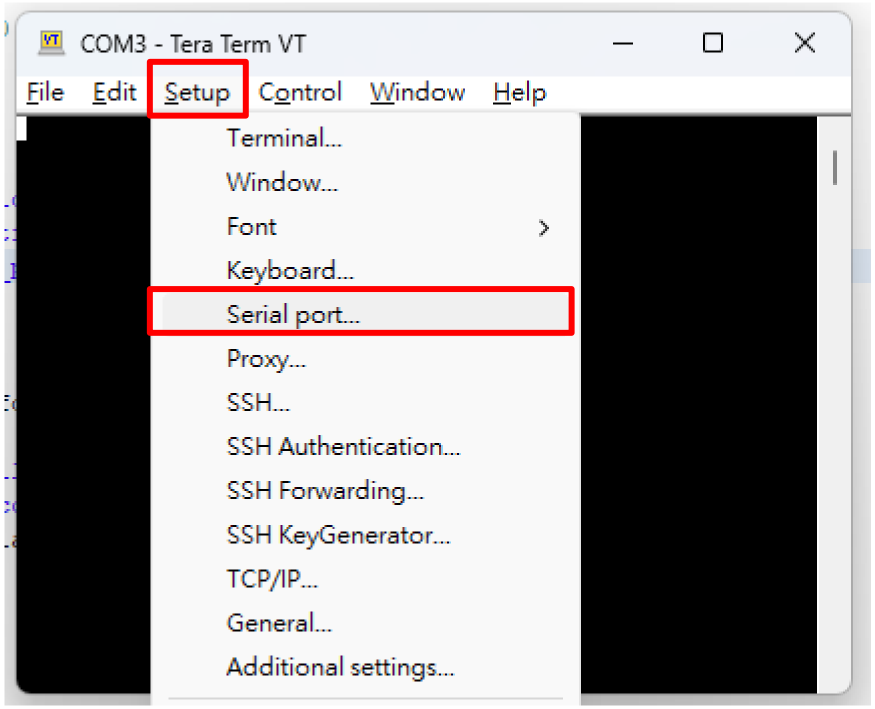
SSH version: SSH2

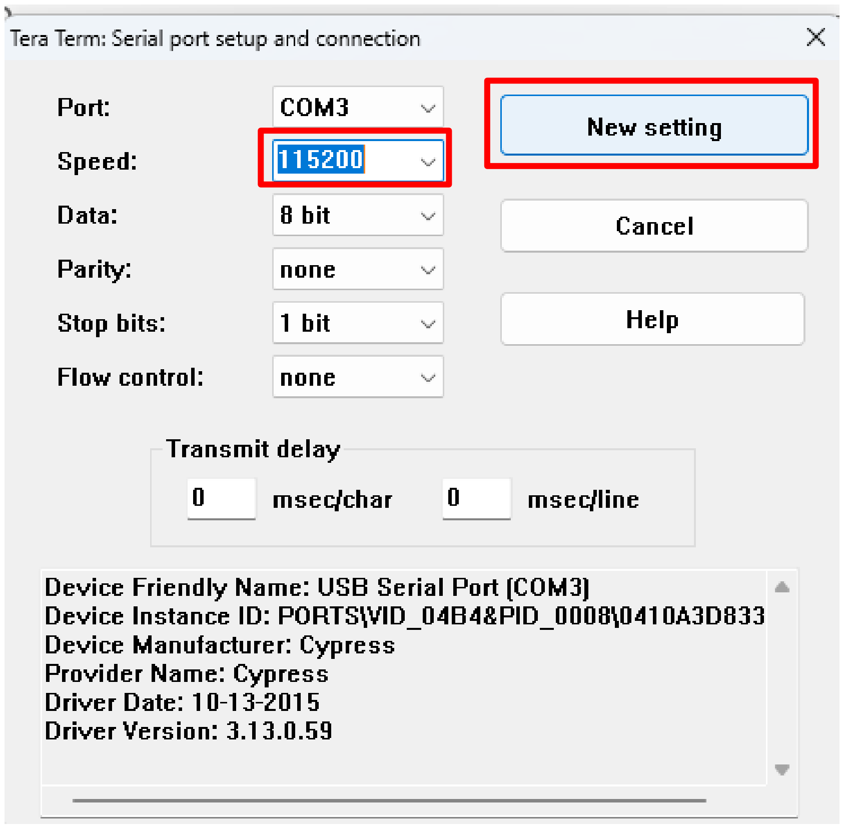
IP version: AUTO

[✓ Serial] Port: COM3: USB Serial Port (COM3)

Set Up Environment

- Click "**Setup -> Serial port**"
- Choose "**115200**" at the speed and click "**New settling**"



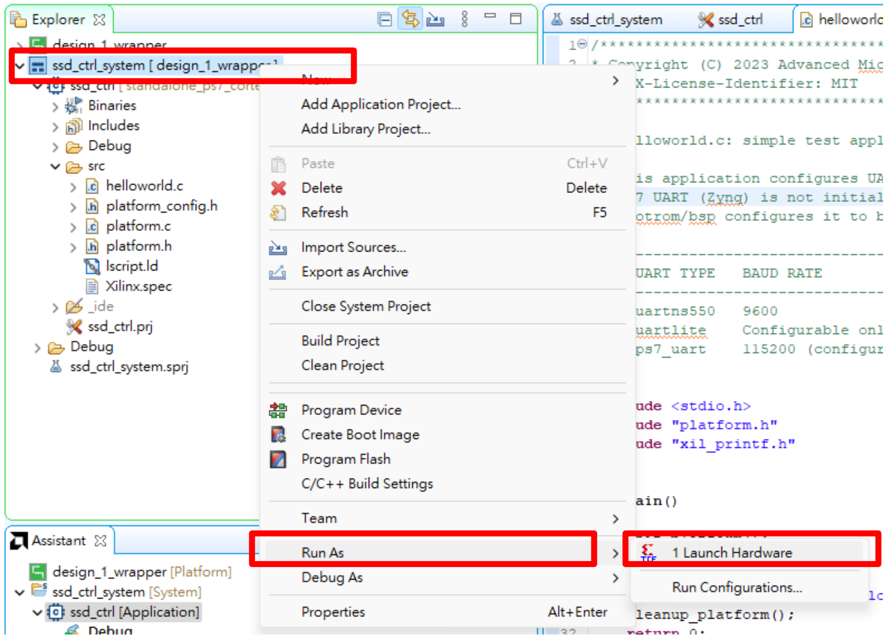


Port: COM3
Speed: 115200
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none
Transmit delay: 0 msec/char 0 msec/line

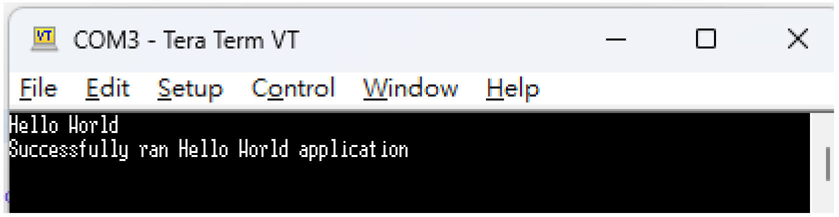
```
Device Friendly Name: USB Serial Port (COM3)
Device Instance ID: PORTS\VID_04B4&PID_0008\0410A3D833
Device Manufacturer: Cypress
Provider Name: Cypress
Driver Date: 10-13-2015
Driver Version: 3.13.0.59
```

Set Up Environment

- Back to Vitis, right click “ssd_ctrl_system” and select “**Run As -> Launch Hardware**”
- Then you should see the “Hello World” message on Tera Term if everything is right



```
/******************************
 * Copyright (C) 2023 Advanced Mic
X-License-Identifier: MIT
******************************

lloworld.c: simple test appl

is application configures UA
7 UART (Zynq) is not initial
otrom/bsp configures it to b

-----------------------------
UART TYPE   BAUD RATE
-----------------------------
uartns550   9600
uartlite    Configurable onl
ps7_uart    115200 (configur

ude <stdio.h>
ude "platform.h"
ude "xil_printf.h"

ain()

leanup_platform();
return 0;
```



```
Hello World
Successfully ran Hello World application
```

Example

Lab07 Outline

- **PART 1: IP Block Design**

AMD Vivado

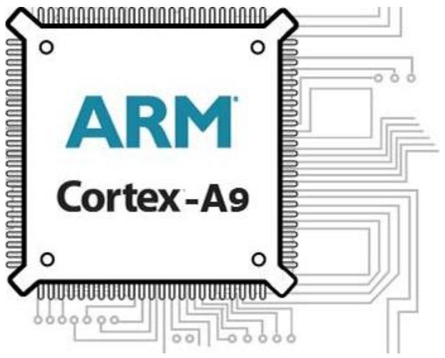
① IP Block Creation
② IP Integration
③ HDL Wrapper
④ Generate Bitstream

- **PART 2: ARM Programming**

AMD Vitis

⑤ ARM Programming
⑥ Launch on Hardware

&



ARM Programming

- First, we are going to learn how to interface with the PmodSSD, LED, and Button
- We need to include three libraries

- `#include "xparameters.h"`
- `#include "ssd_ctrl.h"  //created by the ssd_ctrl IP`
- `#include "xil_io.h"`

- The most important one is the `ssd_ctrl.h`
  - Contain the slv_reg0-3 offset
  - Defined read and write function for slv_reg0-3
  - You could click the header file on Outline (Right sidebar) to see the content

ARM Programming

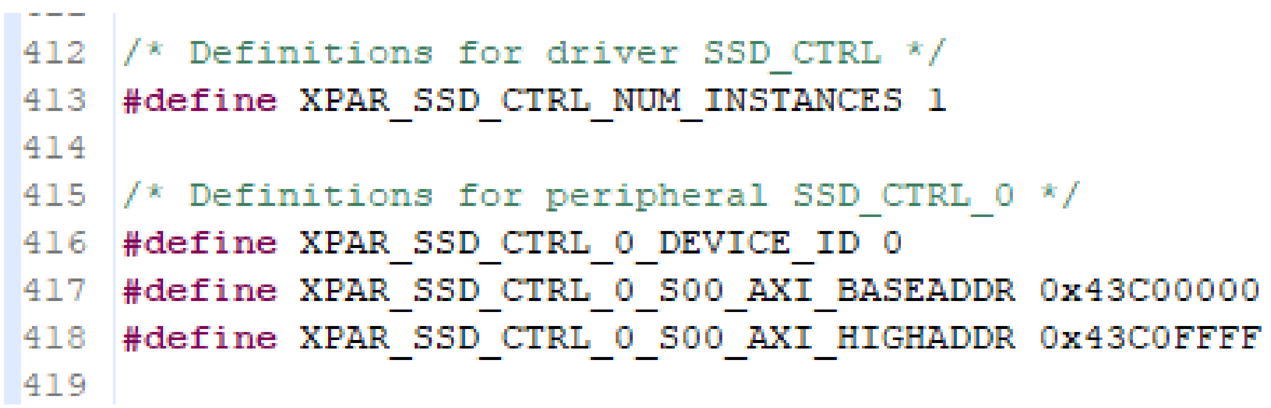
- `#define SSD_CTRL_mWriteReg(BaseAddress, RegOffset, Data)`
- `#define SSD_CTRL_mReadReg(BaseAddress, RegOffset)`

- The **RegOffset** select **which slv_reg to read / write**
  - According to the AXI IP Block Customization Section
    - `#define SSD_CTRL_S00_AXI_SLV_REG0_OFFSET 0`
      - Digit For PmodSSD
    - `#define SSD_CTRL_S00_AXI_SLV_REG1_OFFSET 4`
      - LED
    - `#define SSD_CTRL_S00_AXI_SLV_REG2_OFFSET 8`
      - Button
    - `#define SSD_CTRL_S00_AXI_SLV_REG3_OFFSET 12`
      - Not Used

ARM Programming

- `#define SSD_CTRL_mWriteReg(BaseAddress, RegOffset, Data)`
- `#define SSD_CTRL_mReadReg(BaseAddress, RegOffset)`

- **The BaseAddress could be found in `xparameters.h`**
  - `#define XPAR_SSD_CTRL_IP_0_S00_AXI_BASEADDR 0x43C00000`



```
/* Definitions for driver SSD_CTRL */
#define XPAR_SSD_CTRL_NUM_INSTANCES 1

/* Definitions for peripheral SSD_CTRL_0 */
#define XPAR_SSD_CTRL_0_DEVICE_ID 0
#define XPAR_SSD_CTRL_0_S00_AXI_BASEADDR 0x43C00000
#define XPAR_SSD_CTRL_0_S00_AXI_HIGHADDR 0x43C0FFFF
```

ARM Programming

- Example
  - State 0: The digit stop
  - State 1: The digit keep increasing

  - We want to keep reading the button
  - If the button change from 0 to 1
    - State 0 -> State 1
    - State 1 -> State 0

ARM Programming

```
int main()
{
    init_platform();
    print("Hello World\n\r");
    print("Successfully ran Hello World application\n\r");

    u32 btn_prev = 0, state = 0, digit = 0;
    while(1){
        u32 btn = SSD_CTRL_mReadReg(XPAR_SSD_CTRL_0_S00_AXI_BASEADDR,
        SSD_CTRL_S00_AXI_SLV_REG2_OFFSET);
        if(btn != btn_prev && btn == 1)
            state = 1 - state;
        if(state == 1){
            if(digit == 0xFF)
                digit = 0;
            else
                digit++;
        }
        SSD_CTRL_mWriteReg(XPAR_SSD_CTRL_0_S00_AXI_BASEADDR,
        SSD_CTRL_S00_AXI_SLV_REG0_OFFSET, digit);

        btn_prev = btn;
        for(volatile int i=0; i < 10000000; i++){}
    }

    cleanup_platform();
    return 0;
}
```

ARM Programming

```c
int main()
{
    init_platform();
    print("Hello World\n\r");
    print("Successfully ran Hello World application\n\r");

    u32 btn_prev = 0, state = 0, digit = 0;
    while(1){
        u32 btn = SSD_CTRL_mReadReg(XPAR_SSD_CTRL_0_S00_AXI_BASEADDR,
        SSD_CTRL_S00_AXI_SLV_REG2_OFFSET);
        if(btn != btn_prev && btn == 1)
            state = 1 - state;
        if(state == 1){
            if(digit == 0xFF)
                digit = 0;
            else
                digit++;
        }
        SSD_CTRL_mWriteReg(XPAR_SSD_CTRL_0_S00_AXI_BASEADDR,
        SSD_CTRL_S00_AXI_SLV_REG0_OFFSET, digit);

        btn_prev = btn;
        for(volatile int i=0; i < 10000000; i++){}
    }

    cleanup_platform();
    return 0;
}
```

Base Address

Register 2 -> Btn

Checking Btn event from 0 to 1

Register 0 -> PmodSSD

Wait for certain cycles

Task

ARM Programming

- Task for lab07 – Random Number Generator
  - **Press BTNC to switch state**
  - For state 0 (Initial state), the digit stop
    - Initial value is 0
  - For state 1, keep generate random number in hex from 0 - FF
  - For LED0-7
    - If the current state is state 0
      - Only LED {0,1,2,3} are on
    - Else if the current state is state 1
      - Only LED {4,5,6,7} are on

ARM Programming

- How to generate random number?

- You need to **include these two libraries**
  - `#include <xtime_l.h>`
  - `#include <stdlib.h>`

- First, you need to **set up a seed** for random number (Before the while loop)
  - `XTime T;`
  - `XTime_GetTime(&T);`
  - `srand(T);`

- Then you could **generate a random number**
  - `u32 random = rand()`

ARM Programming

- The slv_reg1 is LED

- You could write hex value to LED
- Such as 0xFF -> 1111 1111b

- **You just need to figure out the hex value of**
  - **State 0: 1111 0000b**
  - **State 1: 0000 1111b**

Lab07: Submission

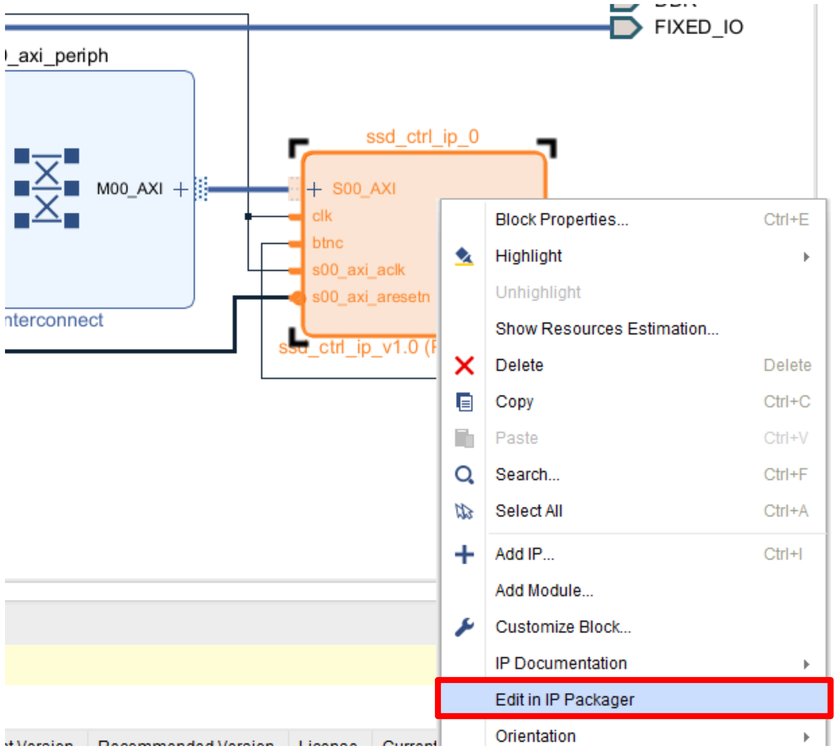
- **Demo Video Requirement:**
- In the demo video, you should clearly display
  - Generate 10 random hex numbers
  - Stay at each random number for 1-2 seconds

- **Submission Rule:**
  - Submit your (1) source code (.c), (2) a screen shot of your block design, and (3) a short demo video to blackboard
  - Deadline: 12:30 on 19 March 2025 (Late submission is NOT acceptable)

Modifying the IP Block (Just in Case)

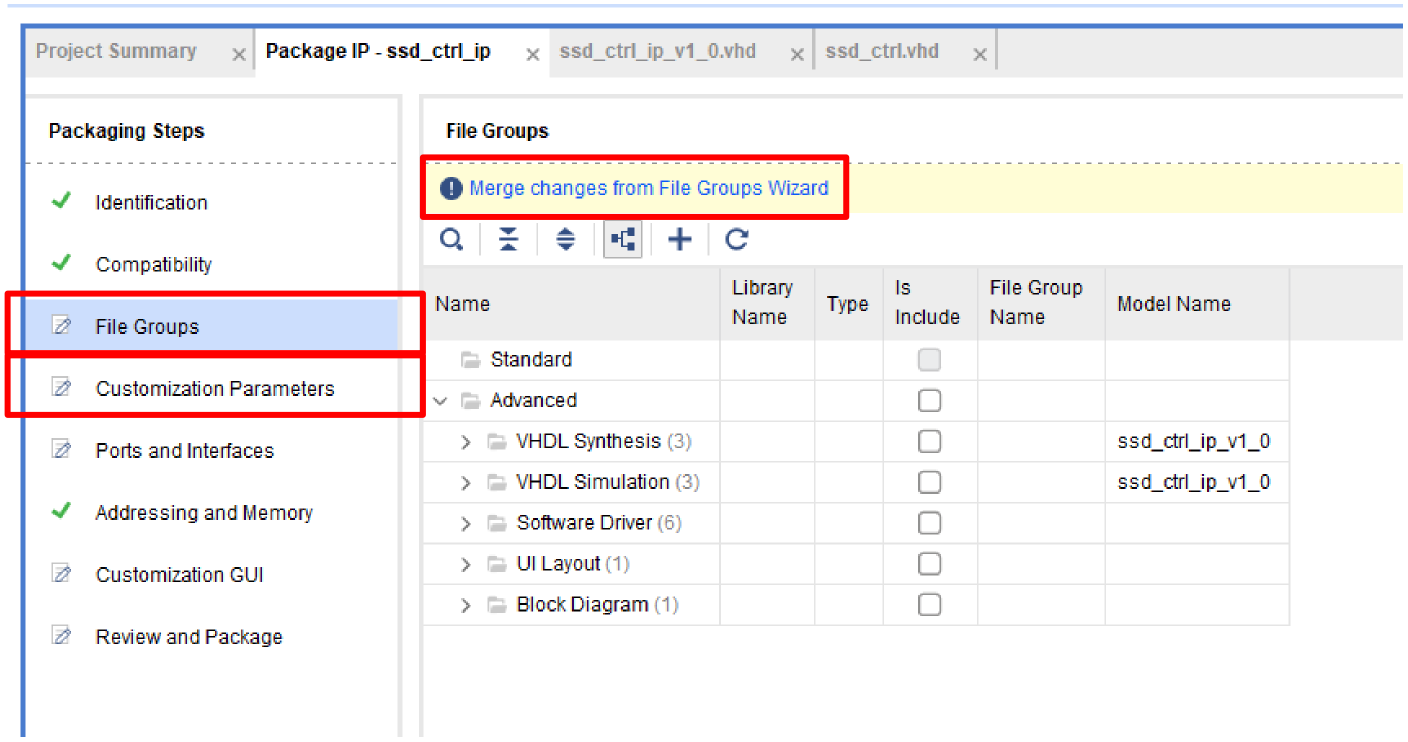
Modifying the IP Block

- If you need to modify the ssd_ctrl ip block, here is the guide for you
- Right-click the IP block and **Select "Edit in IP Packager" and Click "OK"**



Modifying the IP Block

- **Do the modification**
- **Again Click "Merge changes" in both "File Groups" and "Customization Parameters"**



Project Summary | Package IP - ssd_ctrl_ip | ssd_ctrl_ip_v1_0.vhd | ssd_ctrl.vhd

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- File Groups
- Customization Parameters
- Ports and Interfaces
- ✓ Addressing and Memory
- Customization GUI
- Review and Package

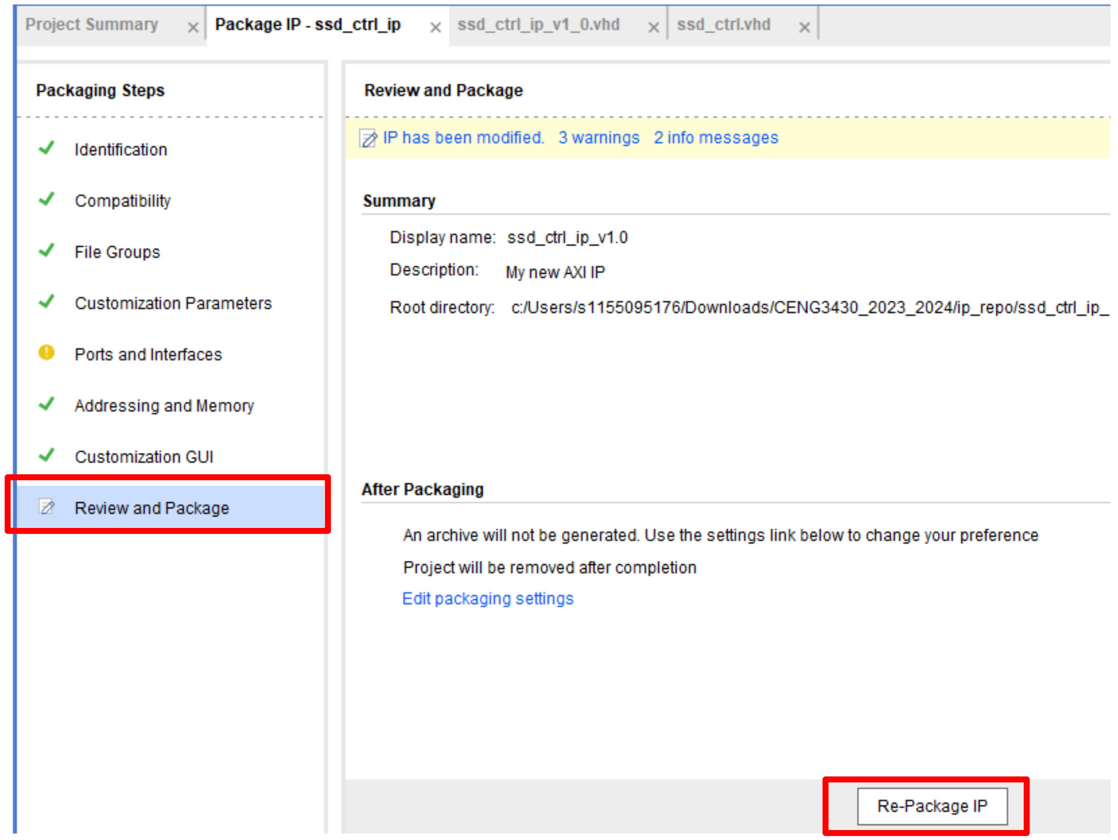
File Groups

Merge changes from File Groups Wizard

Name	Library Name	Type	Is Include	File Group Name	Model Name
Standard			☐		
Advanced			☐		
VHDL Synthesis (3)			☐		ssd_ctrl_ip_v1_0
VHDL Simulation (3)			☐		ssd_ctrl_ip_v1_0
Software Driver (6)			☐		
UI Layout (1)			☐		
Block Diagram (1)			☐		

Modifying the IP Block

- Select "Review and Package"
- Click "Re-Package IP"



Review and Package

IP has been modified. 3 warnings 2 info messages

Summary

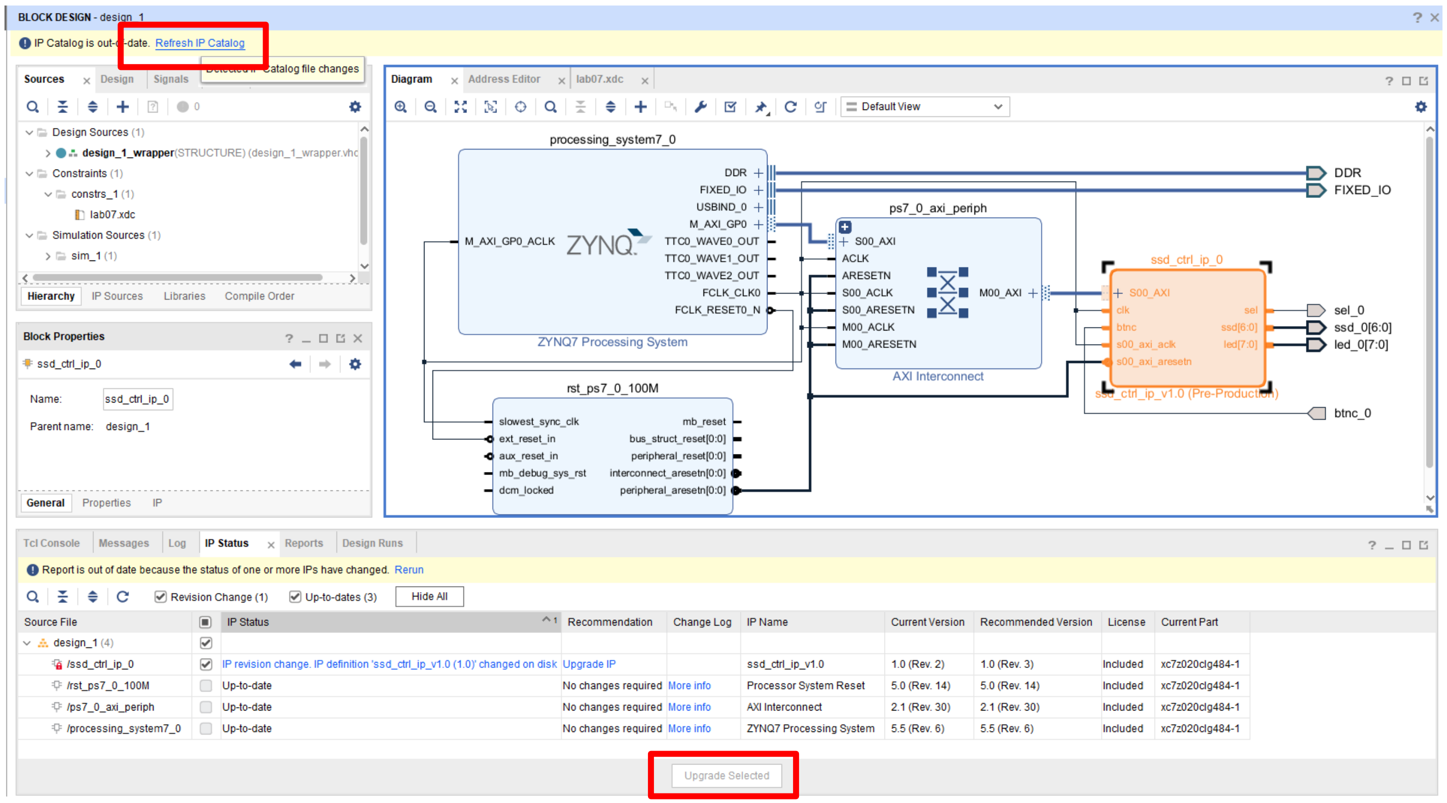
Display name: ssd_ctrl_ip_v1.0
Description: My new AXI IP
Root directory: c:/Users/s1155095176/Downloads/CENG3430_2023_2024/ip_repo/ssd_ctrl_ip_

After Packaging

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

Modifying the IP Block

- Click “Reflesh IP Catalog”
- Click “Update Selected”
- Then “Generate Bitstream”:



Tcl Console | Messages | Log | IP Status | Reports | Design Runs

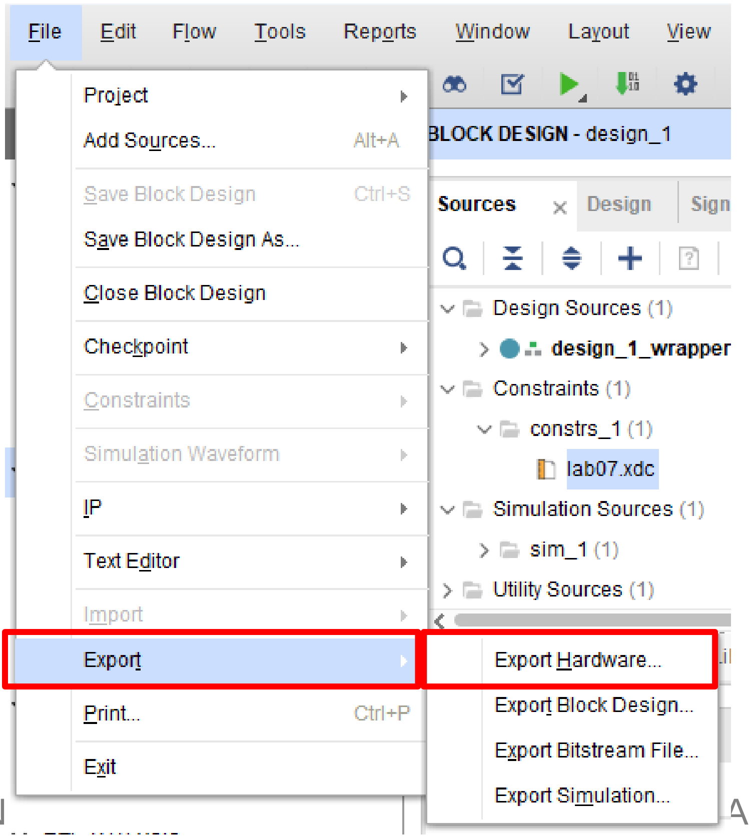
Report is out of date because the status of one or more IPs have changed. Rerun

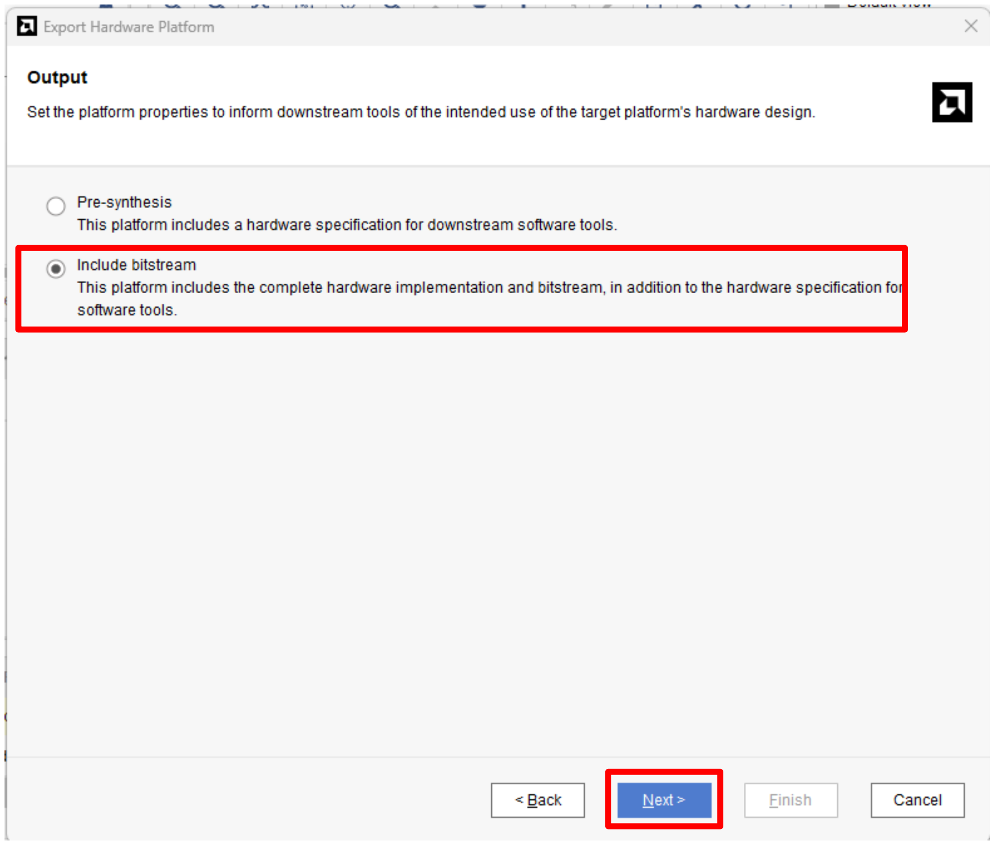
Revision Change (1) | Up-to-dates (3) | Hide All

Source File		IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
design_1 (4)	✓								
/ssd_ctrl_ip_0	✓	IP revision change. IP definition 'ssd_ctrl_ip_v1.0 (1.0)' changed on disk	Upgrade IP		ssd_ctrl_ip_v1.0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7z020clg484-1
/rst_ps7_0_100M		Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 14)	5.0 (Rev. 14)	Included	xc7z020clg484-1
/ps7_0_axi_periph		Up-to-date	No changes required	More info	AXI Interconnect	2.1 (Rev. 30)	2.1 (Rev. 30)	Included	xc7z020clg484-1
/processing_system7_0		Up-to-date	No changes required	More info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020clg484-1

Modifying the IP Block

- Then click "File -> Export"
- Click "Export Hardware"
  - Include the bitstream





Output

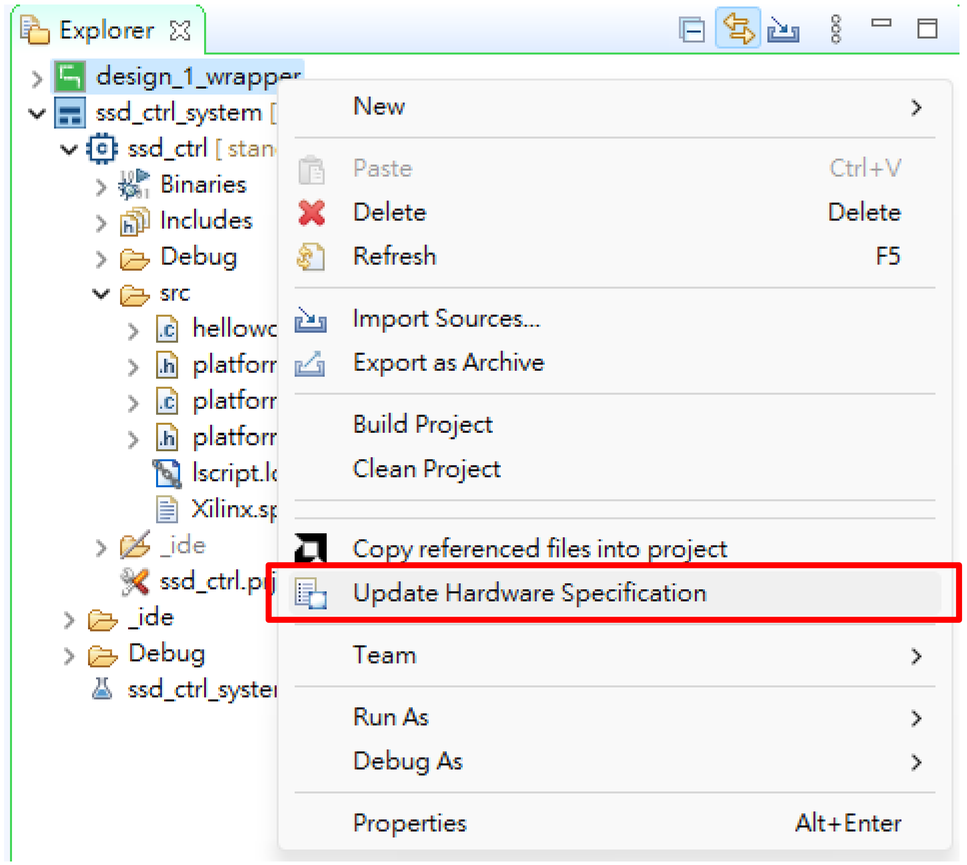
Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

( ) Pre-synthesis
This platform includes a hardware specification for downstream software tools.

(●) Include bitstream
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

Modifying the IP Block

- At vitis, **right-click the “design_1_wrapper”**
- **And click “Update Hardware Specification”**
- Then you could rebuild and run



香 港 中 文 大 學
The Chinese University of Hong Kong

Thank You!