

# Vivado & Vitis — Add Your Own IP Workflow

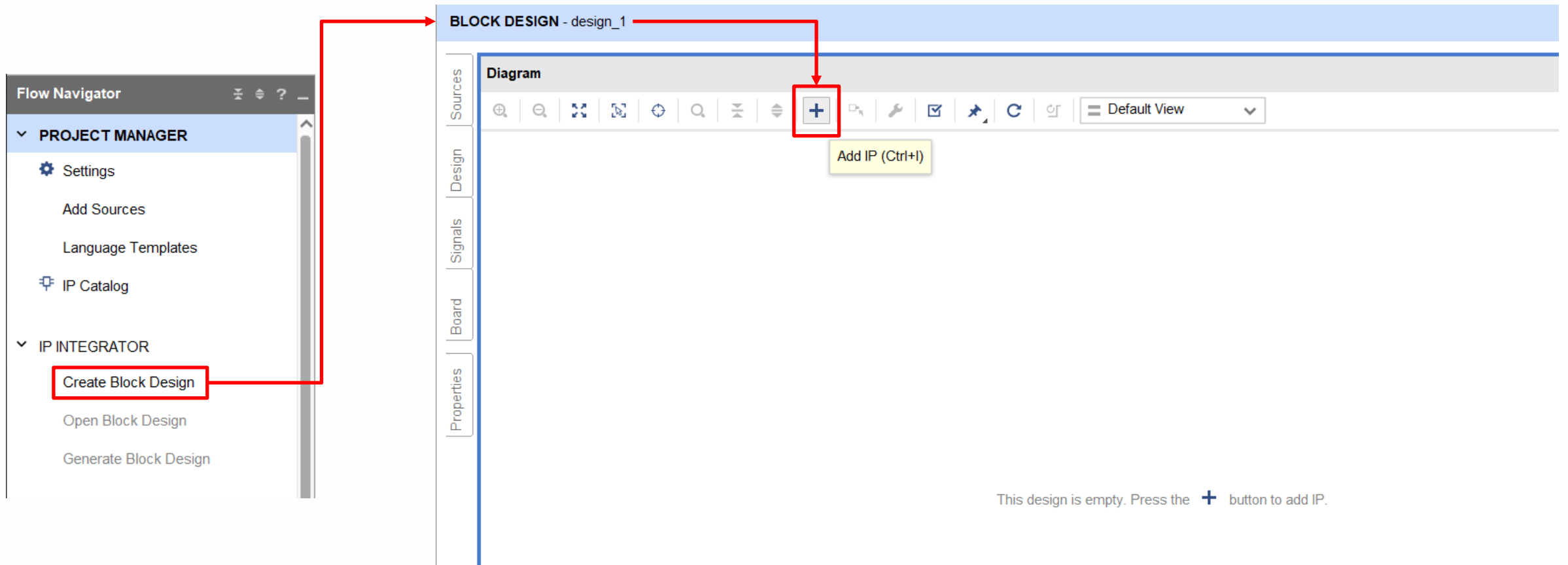
# Agenda

- Vivado — Add Your Own IP
- Vitis — A Sample for Driving Own IP

# Vivado — Add Your Own IP

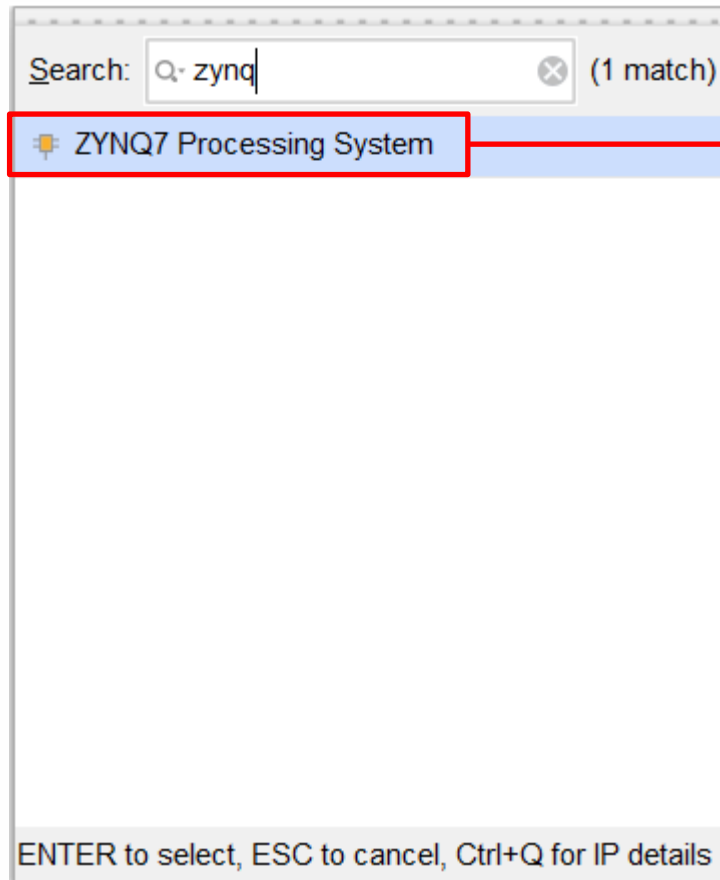
# Block Design

- Step 1: Add Zynq Processing System IP



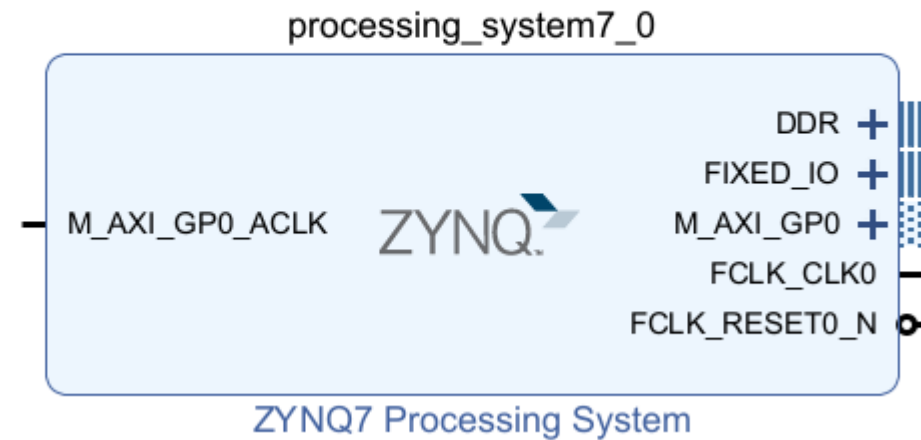
# Block Design

- Step 1: Add Zynq Processing System IP



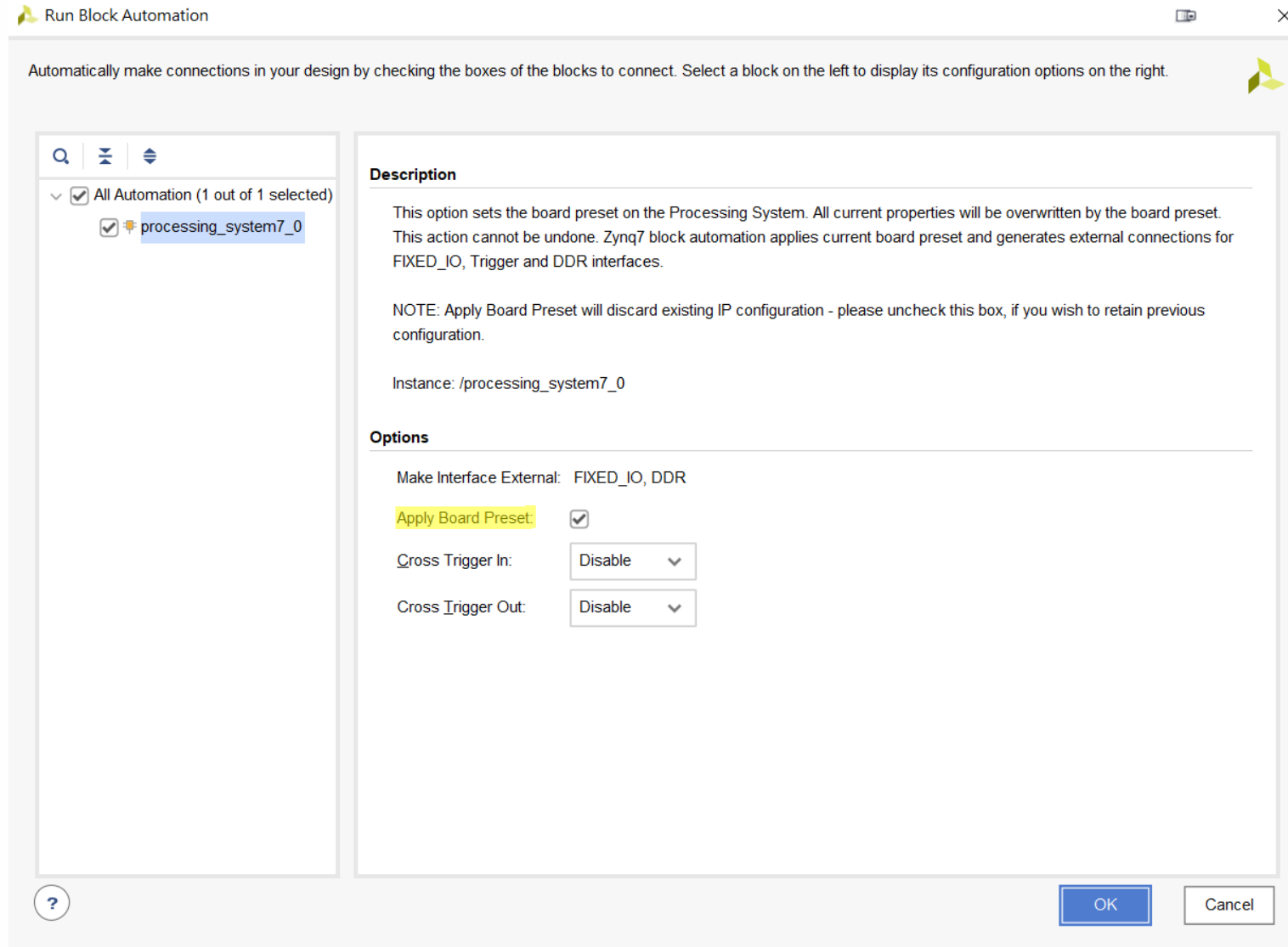
Click here

Designer Assistance available. [Run Block Automation](#)



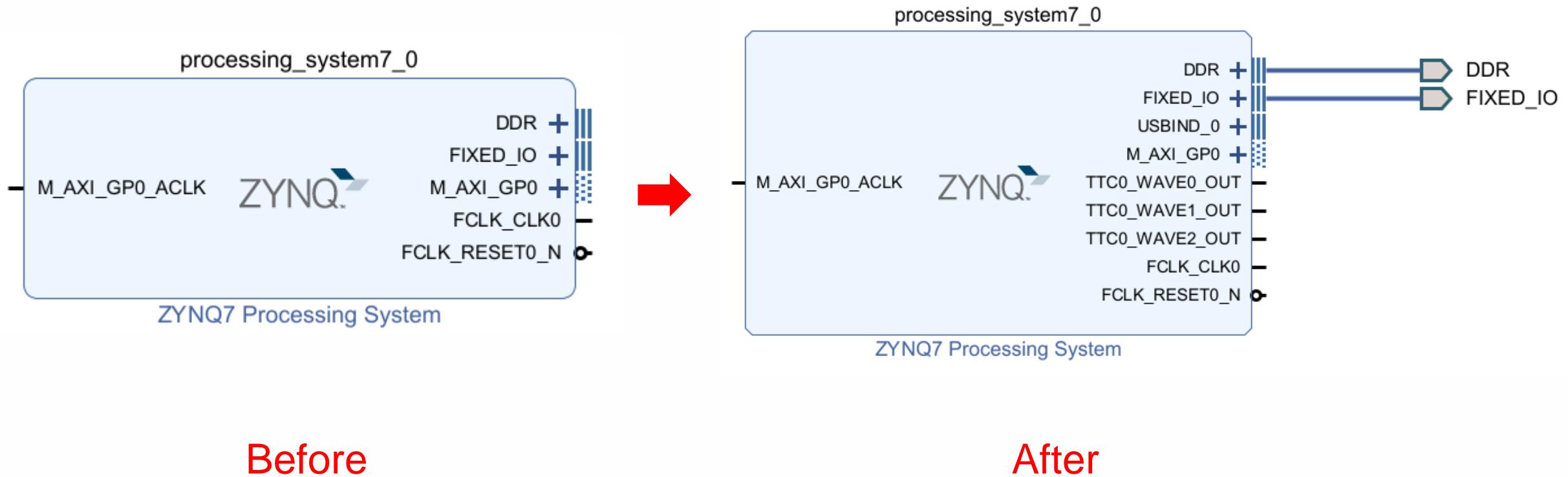
# Block Design

- Step 1: Add Zynq Processing System IP



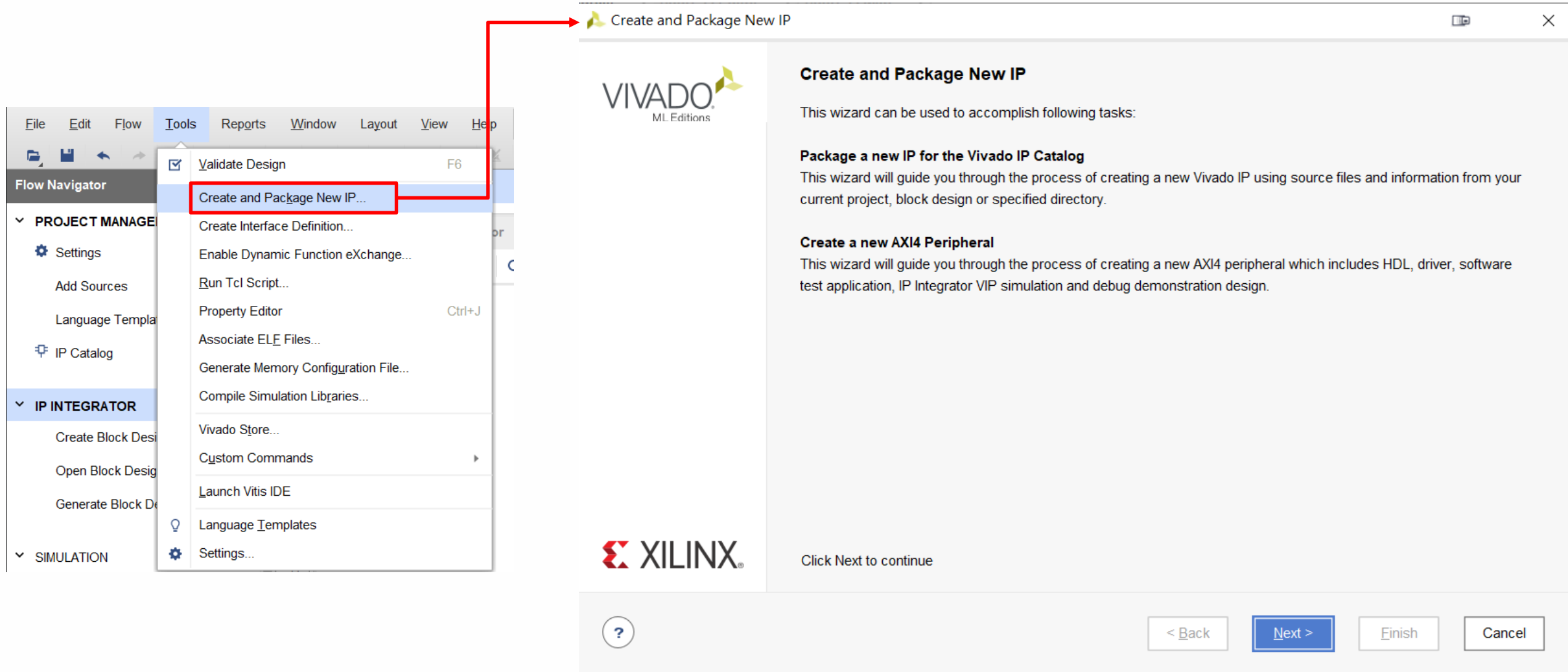
# Block Design

- Step 1: Add Zynq Processing System IP



# Block Design

- Step 2: Create Your Own IP



The image shows the Vivado ML Editions interface. On the left, the 'Tools' menu is open, and 'Create and Package New IP...' is highlighted with a red box. A red arrow points from this menu item to the 'Create and Package New IP' wizard window on the right.

**Create and Package New IP**

This wizard can be used to accomplish following tasks:

- Package a new IP for the Vivado IP Catalog**  
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.
- Create a new AXI4 Peripheral**  
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Click Next to continue

The wizard window includes a 'Next >' button, which is highlighted in blue, and 'Back <', 'Finish', and 'Cancel' buttons.



# Block Design

- Step 2: Create Your Own IP

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:
- ☐ Package a specified directory  
Choose a directory as the source for creating a new IP Definition.

- Package current project to a new IP
- Package current block design to a new IP
- Package specified directory to a new IP

### 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.



< Back

Next >

Finish

Cancel

# Block Design

- Step 2: Create Your Own IP

Create and Package New IP

**Peripheral Details**

Specify name, version and description for the new peripheral

Name: myip

Version: 1.0

Display name: myip\_v1.0

Description: My new AXI IP

IP location: C:/Users/User/Desktop/Lab/Test/./ip\_repo

☐ Overwrite existing

? < Back Next > Finish Cancel

# Block Design

- Step 2: Create Your Own IP

Create and Package New IP

## Add Interfaces

Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

+ -

Interfaces

S00\_AXI

S00\_AXI

myip\_v1.0

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

Lite

Lite

Full

Stream

Slave

Master

Slave

?

< Back

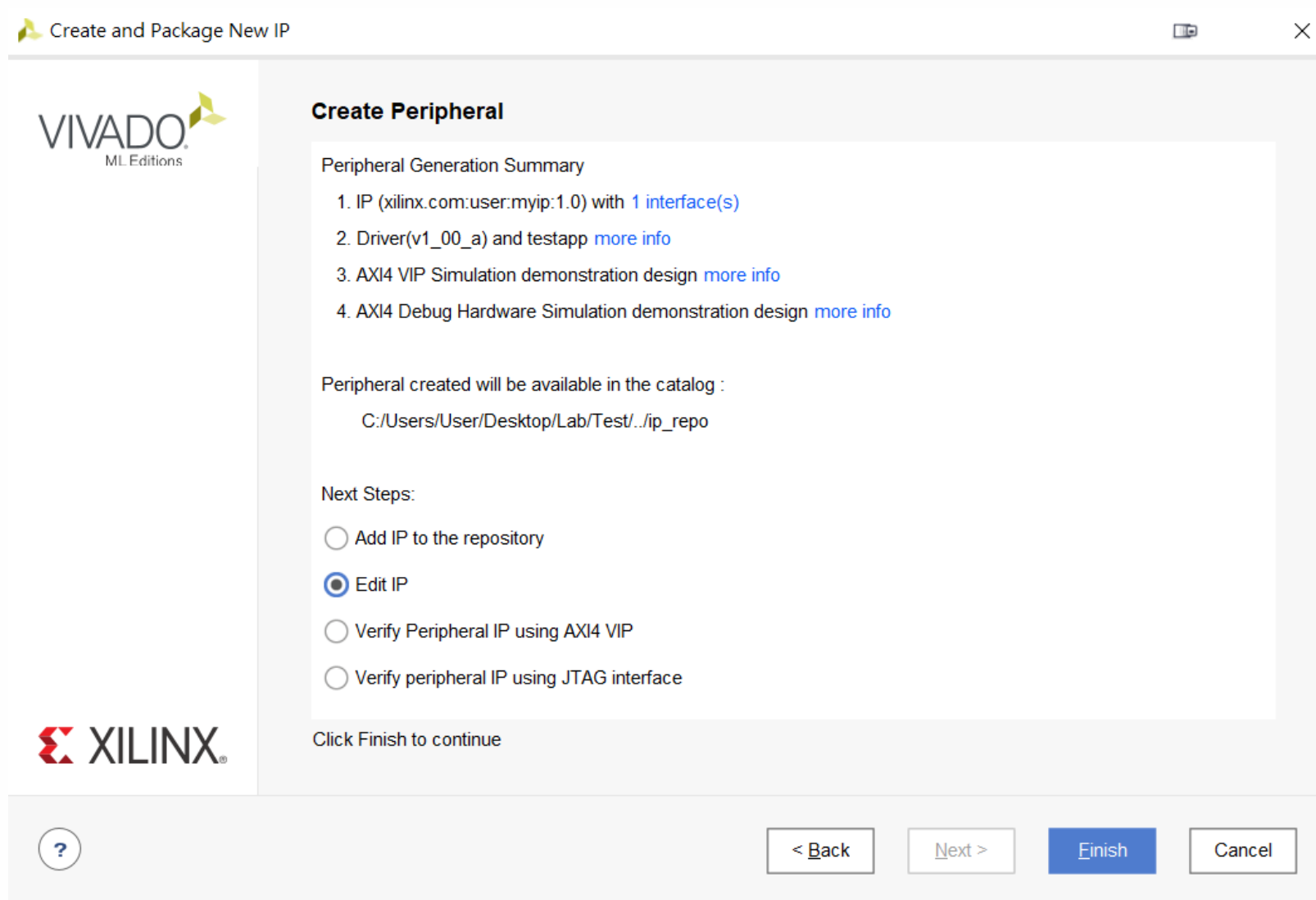
Next >

Finish

Cancel

# Block Design

- Step 2: Create Your Own IP



# Block Design

- Step 3: Edit Your Own IP

The screenshot displays the Xilinx Project Manager interface for a project named 'myip\_v1\_0\_project'. The main window is titled 'Package IP - myip' and is divided into two panes: 'Packaging Steps' and 'Identification'.

**Packaging Steps:** A list of steps with green checkmarks indicating completion:

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

**Identification:** A form for configuring the IP package:

- Vendor: xilinx.com
- Library: user
- Name: myip
- Version: 1.0
- Display name: myip\_v1.0
- Description: My new AXI IP
- Vendor display name: (empty)
- Company url: (empty)
- Root directory: c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0
- Xml file name: c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0/component.xml

**Categories:** A list of categories with a plus button to add more:

- AXI\_Peripheral

**Sources:** A sidebar on the left showing the project hierarchy:

- Design Sources (2)
  - myip\_v1\_0 (myip\_v1\_0.v) (1)
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)
- Utility Sources

**Properties:** A bottom pane with the text 'Select an object to see properties'.

# Block Design

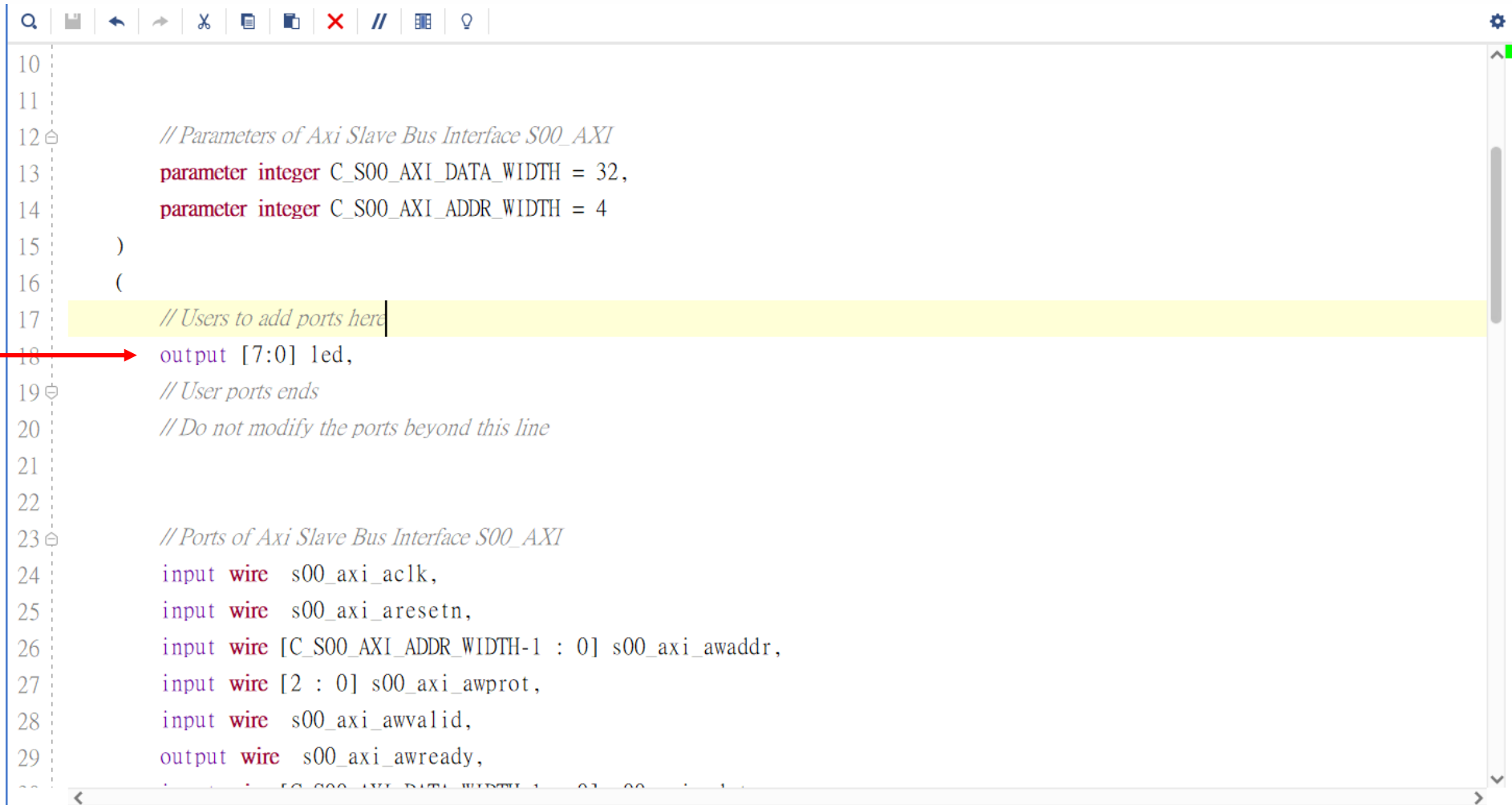
- Step 3: Edit Your Own IP

The screenshot displays the Xilinx Project Manager interface for a project named 'myip\_v1\_0\_project'. The 'Sources' pane on the left shows a tree structure with 'Design Sources (2)' expanded, containing 'myip\_v1\_0 (myip\_v1\_0.v) (1)'. A red box highlights this entry, and a red arrow points to a secondary 'Sources' pane. This secondary pane shows the same tree structure but with 'myip\_v1\_0 (myip\_v1\_0.v) (1)' selected and highlighted in blue. Below the tree, the 'Hierarchy' tab is active, showing a list of checked items: 'Ports and Interfaces', 'Addressing and Memory', 'Customization GUI', and 'Review and Package'. To the right, the 'Properties' pane is visible, showing fields for 'Description' (My new AXI IP), 'Vendor display name', 'Company url', 'Root directory' (c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0), and 'Xml file name' (c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0/component.xml). The 'Categories' section at the bottom shows a list with 'AXI\_Peripheral'.

# Block Design

- Step 3: Edit Your Own IP — add output of leds

➤ myip\_v1\_0.v



```
10
11
12 // Parameters of Axi Slave Bus Interface S00_AXI
13 parameter integer C_S00_AXI_DATA_WIDTH = 32,
14 parameter integer C_S00_AXI_ADDR_WIDTH = 4
15 )
16 (
17 // Users to add ports here
18 output [7:0] led,
19 // User ports ends
20 // Do not modify the ports beyond this line
21
22
23 // Ports of Axi Slave Bus Interface S00_AXI
24 input wire s00_axi_aclk,
25 input wire s00_axi_aresetn,
26 input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_awaddr,
27 input wire [2 : 0] s00_axi_awprot,
28 input wire s00_axi_awvalid,
29 output wire s00_axi_awready,
```

# Block Design

- Step 3: Edit Your Own IP — add output of leds

➤ myip\_v1\_0.v

```
46 // Instantiation of Axi Bus Interface S00_AXI
47 myip_v1_0_S00_AXI # (
48     .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
49     .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
50 ) myip_v1_0_S00_AXI_inst (
51     .led(led),|
52     .S_AXI_ACLK(s00_axi_aclk),
53     .S_AXI_ARESETN(s00_axi_aresetn),
54     .S_AXI_AWADDR(s00_axi_awaddr),
55     .S_AXI_AWPROT(s00_axi_awprot),
56     .S_AXI_AWVALID(s00_axi_awvalid),
57     .S_AXI_AWREADY(s00_axi_awready),
58     .S_AXI_WDATA(s00_axi_wdata),
59     .S_AXI_WSTRB(s00_axi_wstrb),
60     .S_AXI_WVALID(s00_axi_wvalid),
61     .S_AXI_WREADY(s00_axi_wready),
62     .S_AXI_BRESP(s00_axi_bresp),
```



# Block Design

- Step 3: Edit Your Own IP — add output of leds

➤ myip\_v1\_0\_S00\_AXI.v



```
4 module myip_v1_0_S00_AXI #
5 (
6     // Users to add parameters here
7
8     // User parameters ends
9     // Do not modify the parameters beyond this line
10
11     // Width of S_AXI data bus
12     parameter integer C_S_AXI_DATA_WIDTH  = 32,
13     // Width of S_AXI address bus
14     parameter integer C_S_AXI_ADDR_WIDTH  = 4
15 )
16 (
17     // Users to add ports here
18     output [7:0] led,
19
20     // User ports ends
21     // Do not modify the ports beyond this line
22
23     // Global Clock Signal
```

# Block Design

- Step 3: Edit Your Own IP — add output of leds

➤ myip\_v1\_0\_S00\_AXI.v

```
231     else begin
232         if (slv_reg_wren)
233             begin
234                 case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
235                     2'h0:
236                         for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
237                             if ( S_AXI_WSTRB[byte_index] == 1 ) begin
238                                 // Respective byte enables are asserted as per write strobes
239                                 // Slave register 0
240                                 slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
241                             end
242                     2'h1:
243                         for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
244                             if ( S_AXI_WSTRB[byte_index] == 1 ) begin
245                                 // Respective byte enables are asserted as per write strobes
246                                 // Slave register 1
247                                 slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
248                             end
249                     2'h2:
250                         for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
```

AXI → Register

Register → AXI

```
366     end
367
368     // Implement memory mapped register select and read logic generation
369     // Slave register read enable is asserted when valid address is available
370     // and the slave is ready to accept the read address.
371     assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
372     always @(*)
373     begin
374         // Address decoding for reading registers
375         case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
376             2'h0 : reg_data_out <= slv_reg0;
377             2'h1 : reg_data_out <= slv_reg1;
378             2'h2 : reg_data_out <= slv_reg2;
379             2'h3 : reg_data_out <= slv_reg3;
380             default : reg_data_out <= 0;
381         endcase
382     end
383
384     // Output register or memory read data
385     always @( posedge S_AXI_ACLK )
```

# Block Design

- Step 3: Edit Your Own IP — add output of leds
  - myip\_v1\_0\_S00\_AXI.v



The screenshot shows a Verilog code editor with a toolbar at the top. The code is as follows:

```
117
118     // I/O Connections assignments
119
120     assign S_AXI_AWREADY    = axi_awready;
121     assign S_AXI_WREADY    = axi_wready;
122     assign S_AXI_BRESP     = axi_bresp;
123     assign S_AXI_BVALID    = axi_bvalid;
124     assign S_AXI_ARREADY   = axi_arready;
125     assign S_AXI_RDATA     = axi_rdata;
126     assign S_AXI_RRESP     = axi_rresp;
127     assign S_AXI_RVALID    = axi_rvalid;
128
129     assign led = slv_reg0;
130
131     // Implement axi_awready generation
132     // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
133     // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
134     // de-asserted when reset is low.
135
136     always @( posedge S_AXI_ACLK )
137     begin
```

A red arrow points to line 129, which is highlighted in yellow: `assign led = slv_reg0;`

# Block Design

- Step 3: Edit Your Own IP — merge changes and package

The screenshot shows the Xilinx IP Packaging Wizard interface. The left sidebar lists the 'Packaging Steps' in a vertical list. The main area is titled 'File Groups' and contains a table of file groups. A yellow banner at the top of the main area contains a warning icon and the text 'Merge changes from File Groups Wizard', which is highlighted by a red rectangle. Below the banner is a toolbar with icons for search, expand/collapse, refresh, and other actions. The table has columns for Name, Library Name, Type, Is Include, File Group Name, and Model Name. The 'Is Include' column contains checkboxes. The 'File Group Name' and 'Model Name' columns contain values for the file groups.

**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			<input type="checkbox"/>		
Advanced			<input type="checkbox"/>		
> Verilog Synthesis (2)			<input type="checkbox"/>		myip_v1_0
> Verilog Simulation (2)			<input type="checkbox"/>		myip_v1_0
> Software Driver (6)			<input type="checkbox"/>		
> UI Layout (1)			<input type="checkbox"/>		
> Block Diagram (1)			<input type="checkbox"/>		

# Block Design

- Step 3: Edit Your Own IP — merge changes and package

The screenshot shows the Xilinx IP Packaging tool interface. The top tabs include 'Project Summary', 'Package IP - myip', 'myip\_v1\_0.v', and 'myip\_v1\_0\_S00\_AXI.v'. The left sidebar, titled 'Packaging Steps', lists the following steps with their completion status:

- Identification (checked)
- Compatibility (checked)
- File Groups (checked)
- Customization Parameters (checked)
- Ports and Interfaces (checked and highlighted)
- Addressing and Memory (checked)
- Customization GUI (checked)
- Review and Package (unchecked)

The main area, titled 'Ports and Interfaces', contains a table with the following columns: Name, Interface Mode, Enablement Dependency, Direction, Driver Value, Size Left, Size Right, Size Left Dependency, Size Right Dependency, and Type Name. The table lists three items: 'S00\_AXI' (slave interface), 'Clock and Reset Signals' (group), and 'led' (port). The 'led' port row is highlighted with a red box.

Name	Interface Mode	Enablement Dependency	Direction	Driver Value	Size Left	Size Right	Size Left Dependency	Size Right Dependency	Type Name
> S00_AXI	slave								
> Clock and Reset Signals									
led			out		7	0			std_logic_vector

# Block Design

- Step 3: Edit Your Own IP — merge changes and package

The screenshot displays the Xilinx IP Packaging tool interface. The top tab bar shows four tabs: 'Project Summary', 'Package IP - myip', 'myip\_v1\_0.v', and 'myip\_v1\_0\_S00\_AXI.v'. The left sidebar, titled 'Packaging Steps', lists seven steps with green checkmarks: 'Identification', 'Compatibility', 'File Groups', 'Customization Parameters', 'Ports and Interfaces', 'Addressing and Memory', and 'Customization GUI'. The 'Review and Package' step is highlighted in blue. The main content area is titled 'Review and Package' and contains two sections. The 'Summary' section shows: 'Display name: myip\_v1.0', 'Description: My new AXI IP', and 'Root directory: c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0'. The 'After Packaging' section contains the text: 'An archive will not be generated. Use the settings link below to change your preference', 'Project will be removed after completion', and a blue link 'Edit packaging settings'. At the bottom right, a red-bordered button labeled 'Re-Package IP' is visible.

Project Summary x Package IP - myip x myip\_v1\_0.v x myip\_v1\_0\_S00\_AXI.v x

**Packaging Steps**

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

**Review and Package**

**Summary**

Display name: myip\_v1.0  
Description: My new AXI IP  
Root directory: c:/Users/User/Desktop/Lab/ip\_repo/myip\_1.0

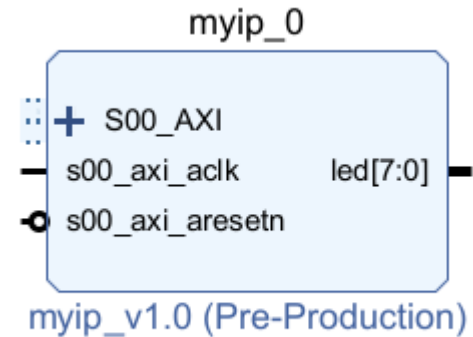
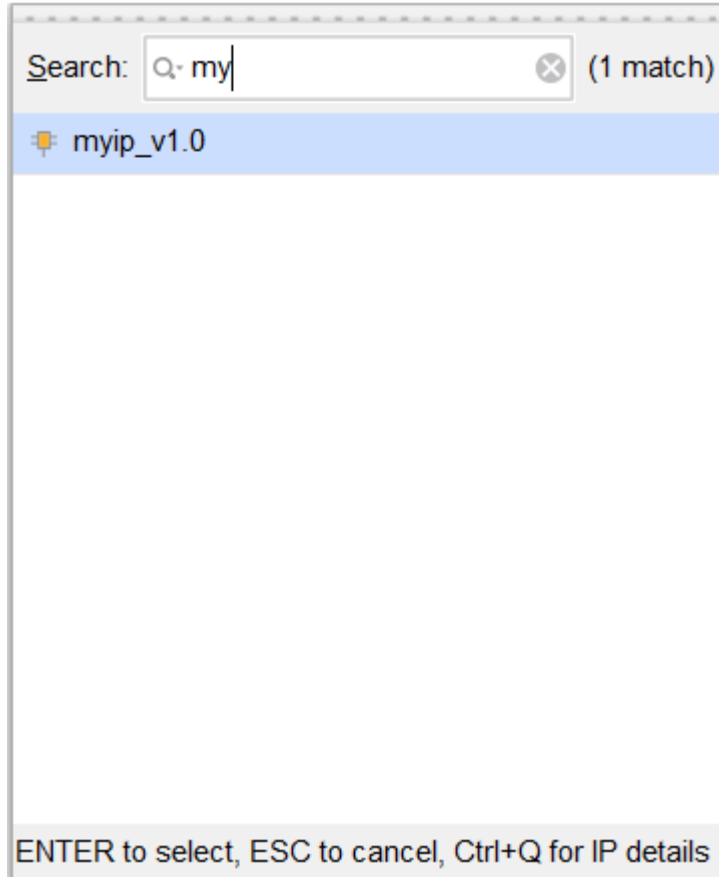
**After Packaging**

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

Re-Package IP

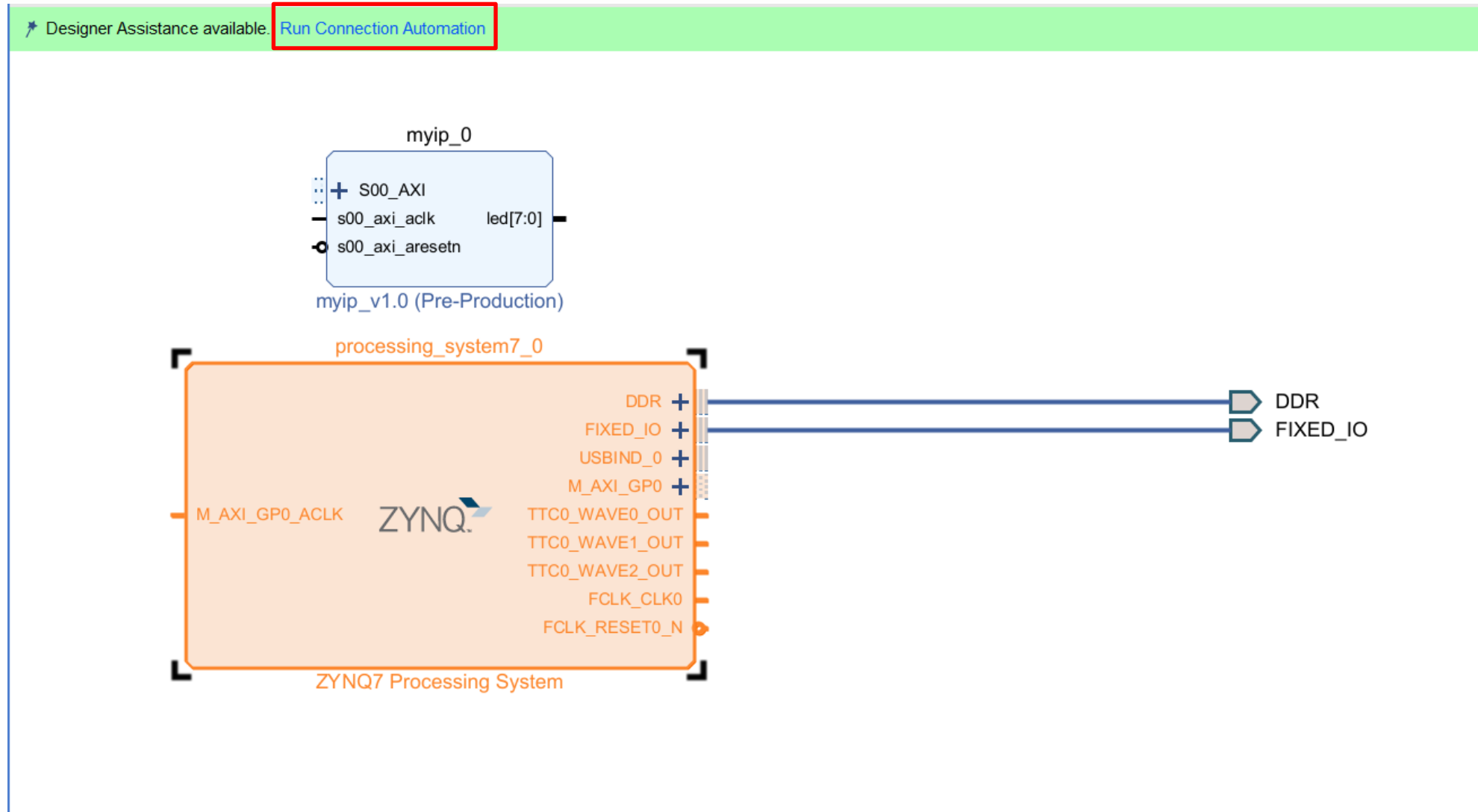
# Block Design

- Step 4: Add Own IP to Block Design



# Block Design

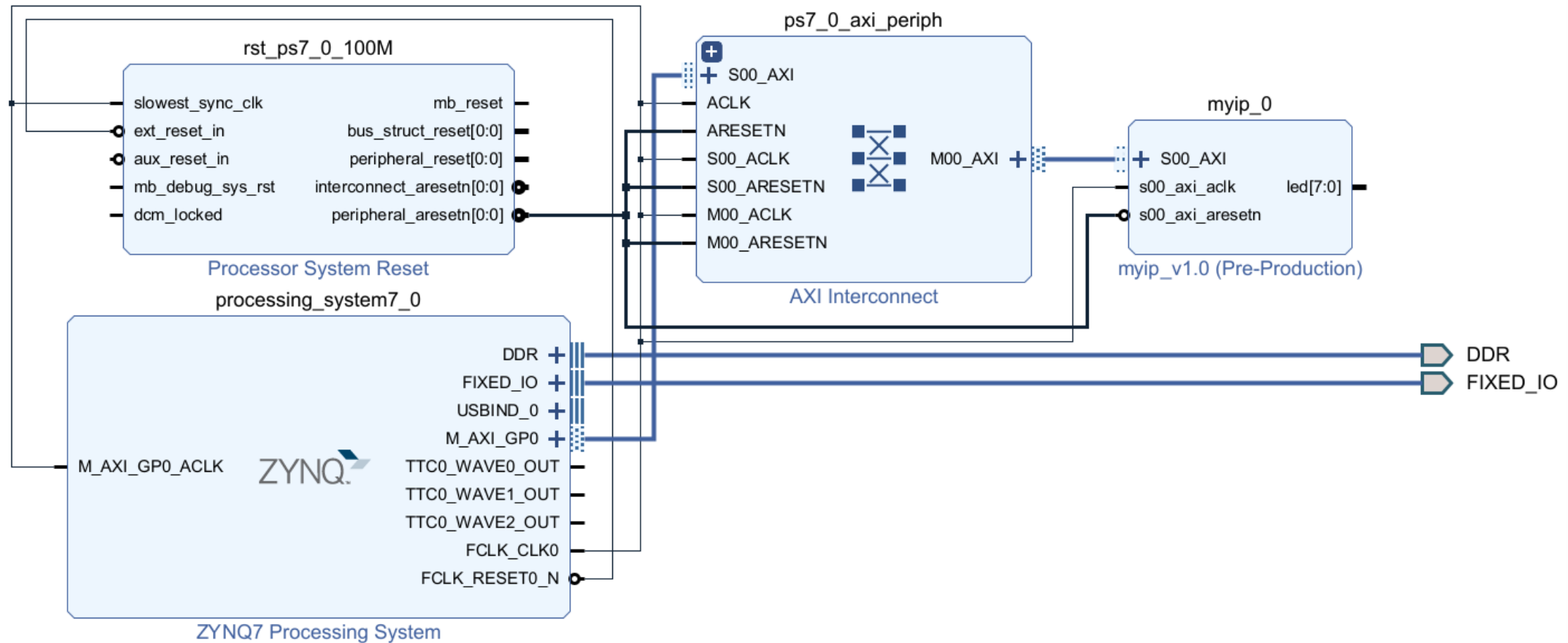
- Step 4: Add Own IP to Block Design





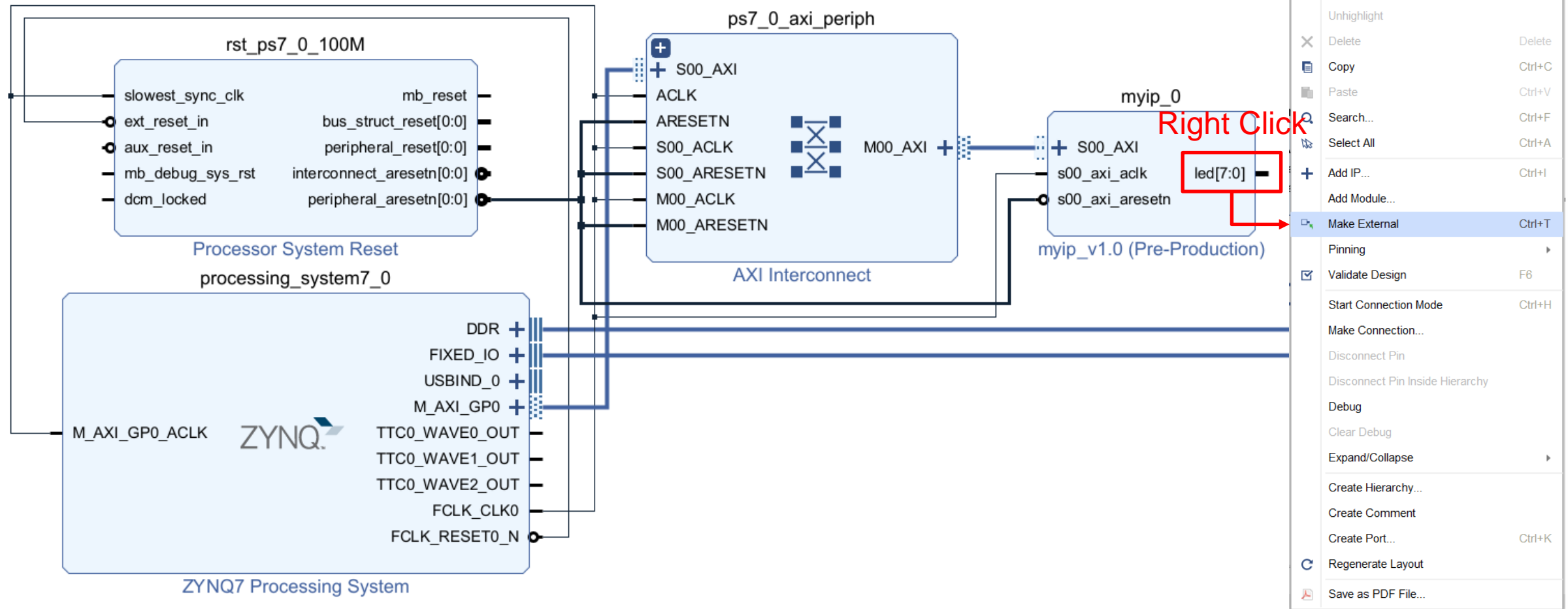
# Block Design

- Step 4: Add Own IP to Block Design



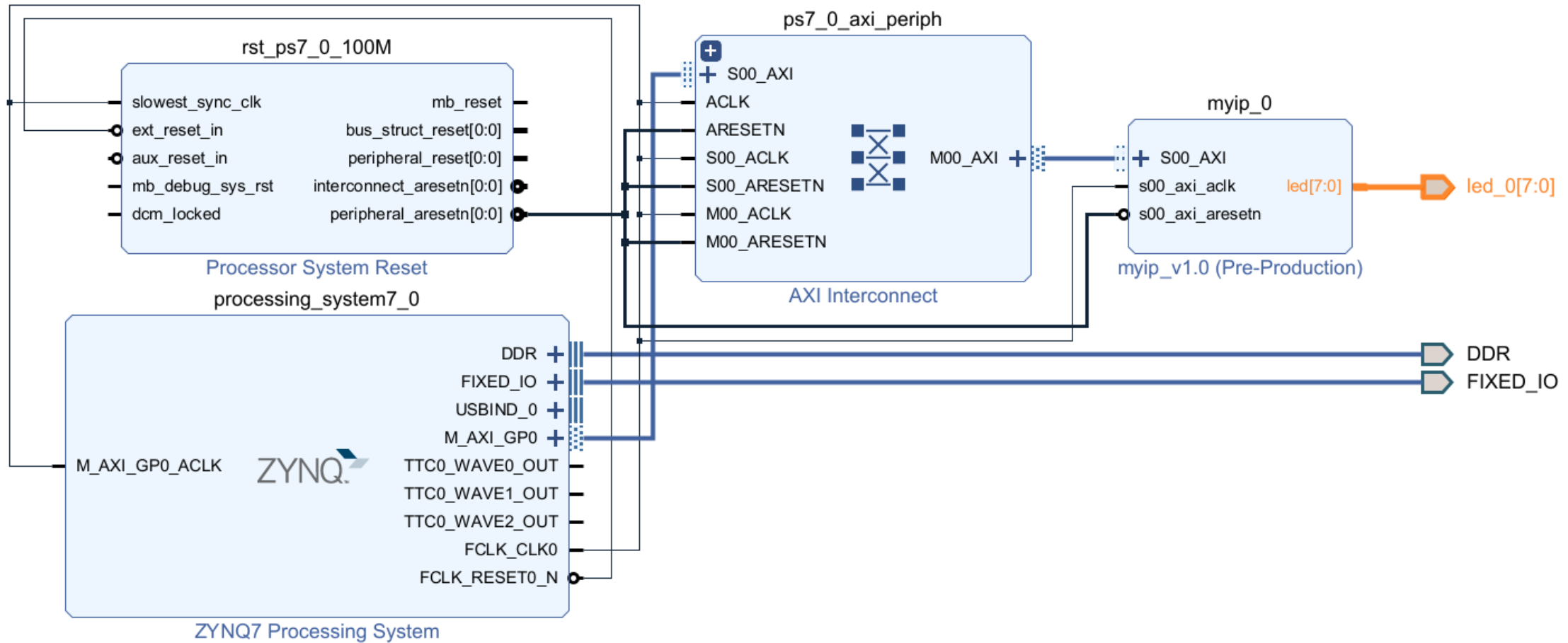
# Block Design

- Step 4: Add Own IP to Block Design



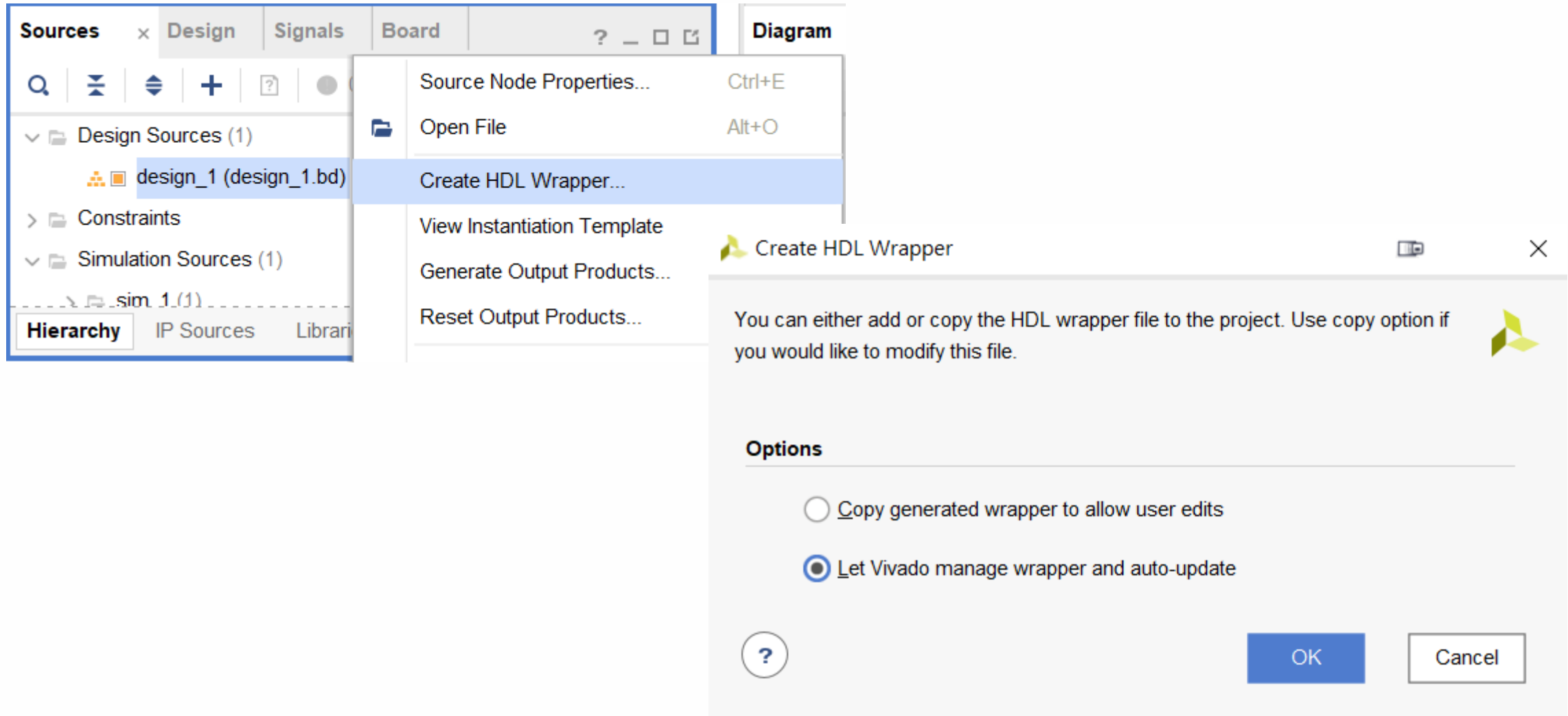
# Block Design

- Step 4: Add Own IP to Block Design



# Block Design

- Step 5: Generate XSA file for Vitis



# Block Design

- Step 5: Generate XSA file for Vitis

The screenshot displays the Xilinx Vitis IDE interface. On the left, the 'IP INTEGRATOR' sidebar is visible, with the 'Generate Block Design' option highlighted in a red box. A red arrow points from this option to the 'Generate Output Products' dialog box in the center. The dialog box shows a list of output products to be generated, including 'design\_1.bd' (OOB per IP) with sub-items for Synthesis, Implementation, and Simulation. Below this, the 'Synthesis Options' section shows 'Out of context per IP' selected. The 'Run Settings' section shows 'Number of jobs' set to 4. At the bottom of the dialog are buttons for '?', 'Apply', 'Generate', and 'Cancel'. A second red arrow points from the 'Generate' button to the 'Generate Bitstream' option in the 'PROGRAM AND DEBUG' sidebar on the right, which is also highlighted in a red box. The 'Sources' window at the top right shows the design hierarchy: 'Design Sources (1)' containing 'design\_1\_wrapper (design\_1\_wrapper.v) (1)', 'design\_1\_i : design\_1 (design\_1.bd) (1)', and 'design\_1 (design\_1.v) (5)'. The 'Hierarchy' tab is selected in the Sources window.

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

Generate Output Products

The following output products will be generated.

Preview

- design\_1.bd (OOB per IP)
  - Synthesis
  - Implementation
  - Simulation

Synthesis Options

- ☐ Global
- ☒ Out of context per IP
- ☐ Out of context per Block Design

Run Settings

Number of jobs: 4

? Apply Generate Cancel

Sources x Design Signals Board

Design Sources (1)

- design\_1\_wrapper (design\_1\_wrapper.v) (1)
  - design\_1\_i : design\_1 (design\_1.bd) (1)
    - design\_1 (design\_1.v) (5)

Constraints

Hierarchy IP Sources Libraries Compile Order

PROGRAM AND DEBUG

- Generate Bitstream**
- Open Hardware Manager

# Block Design

- Step 5: Generate XSA file for Vitis

The image shows the Xilinx IDE interface. On the left, the **File** menu is open, and the **Export** option is highlighted with a red box. A red arrow points from the **Export Hardware...** option in the submenu to the **Export Hardware Platform** dialog on the right.

The **Export Hardware Platform** dialog is titled "Output" and contains the following text: "Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design." There are two radio button options:

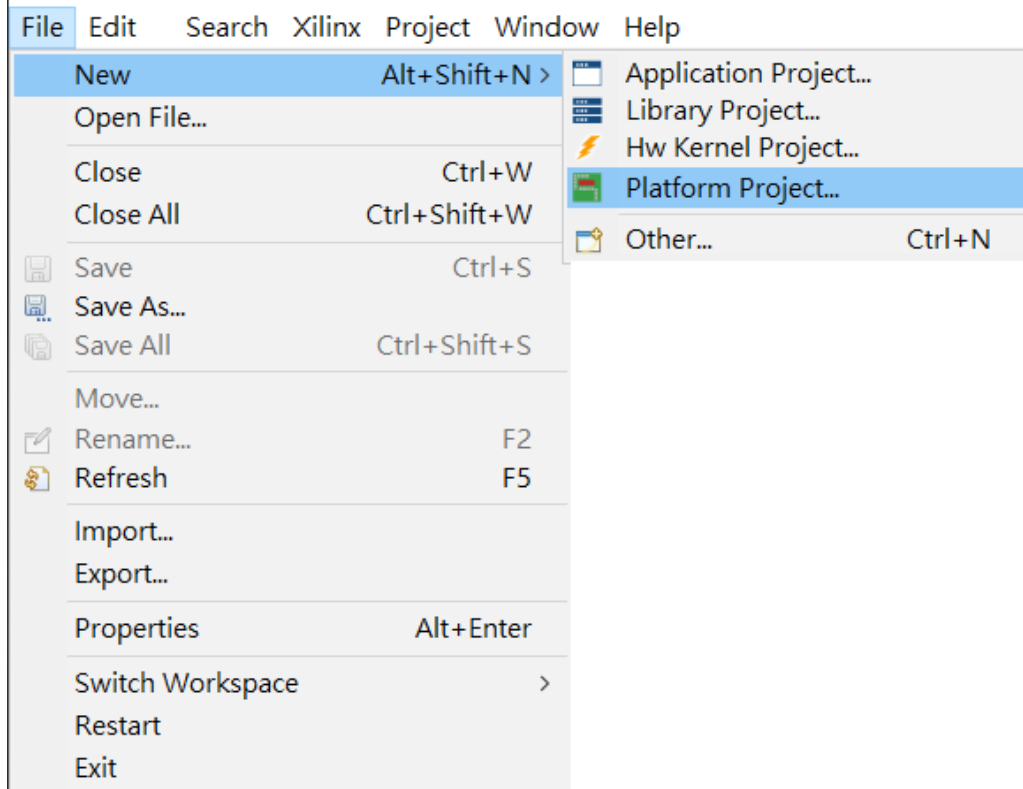
- ☐ Pre-synthesis  
This platform includes a hardware specification for downstream software tools.
- ☒ Include bitstream  
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

At the bottom of the dialog, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

# Vitis — A Sample for Driving Own IP

# Create Platform

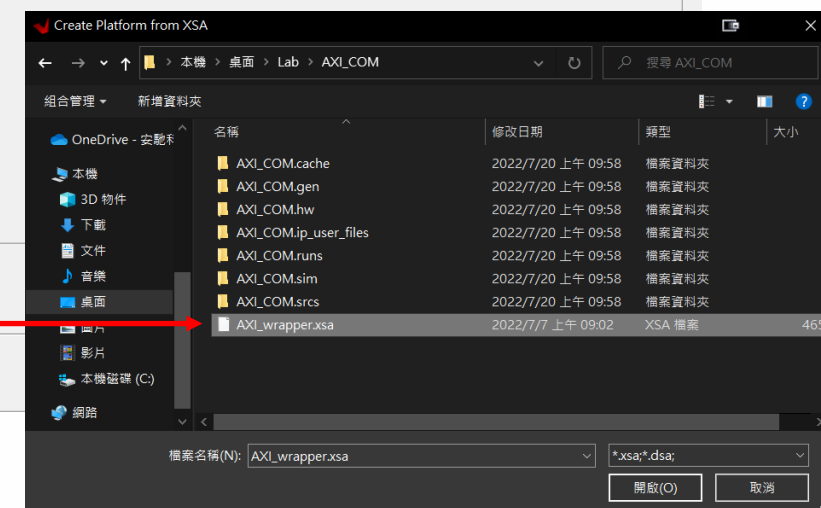
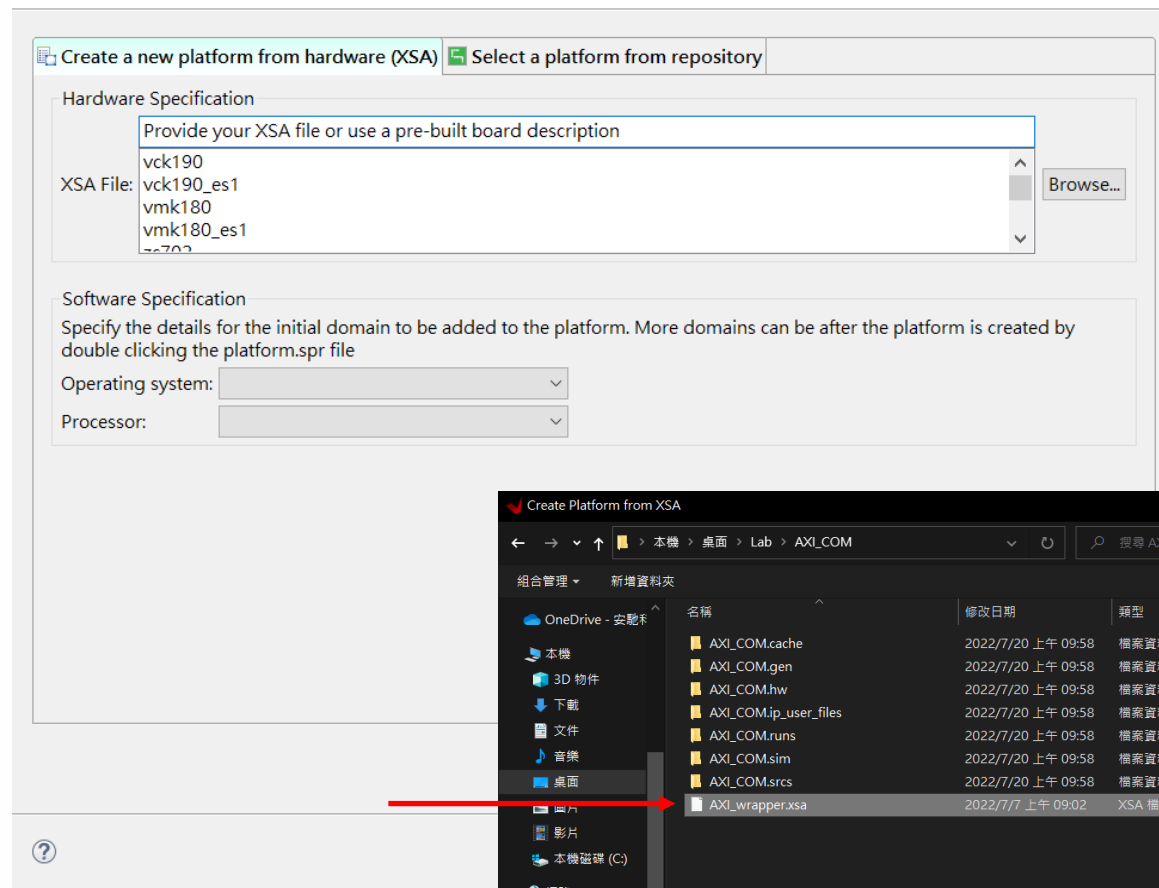
AXI\_Vitis - timer\_led/src/TimerINTC4Leds.c - Vitis IDE



New Platform Project

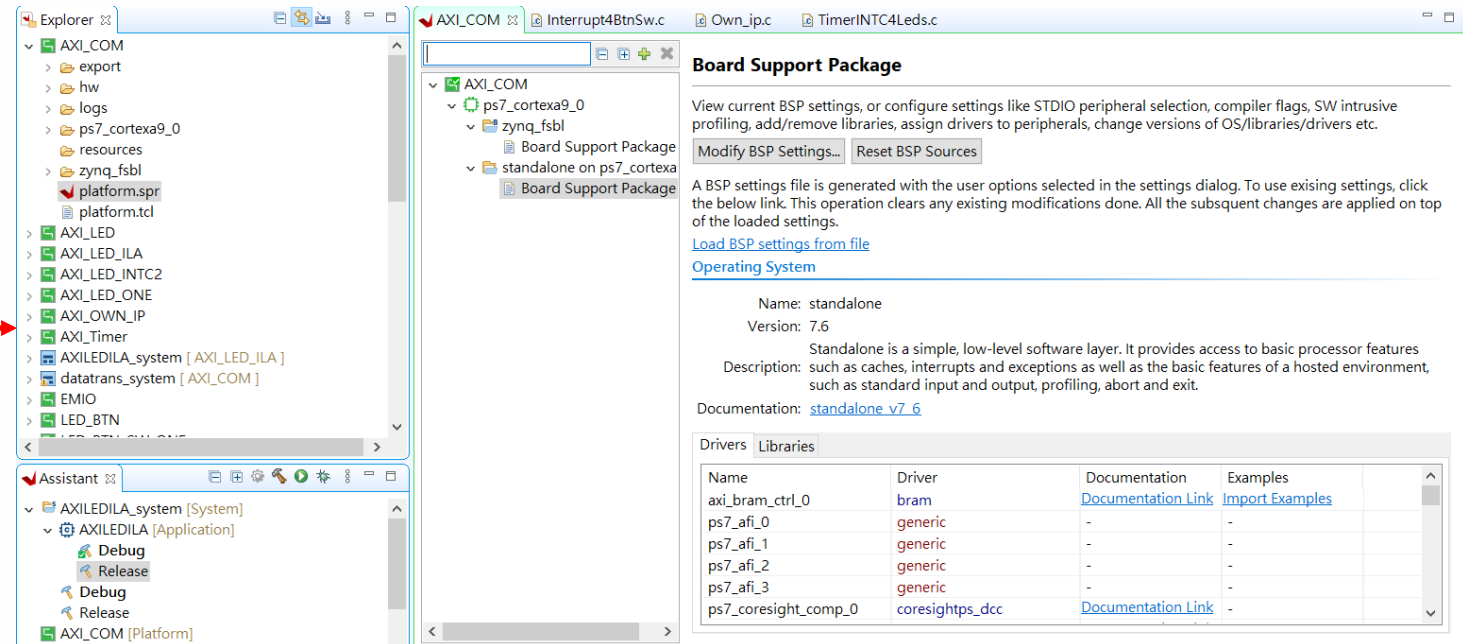
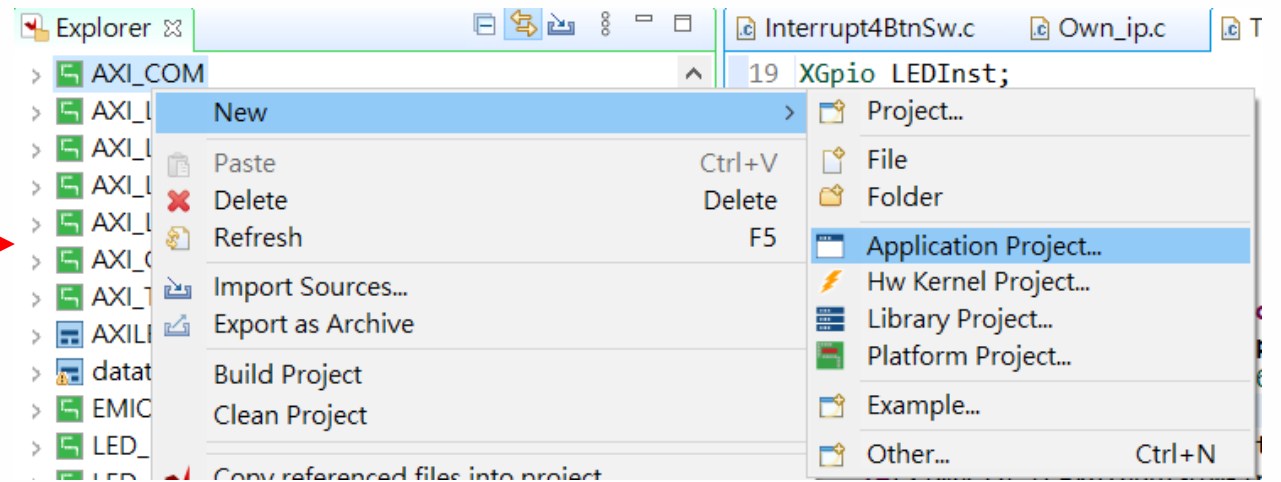
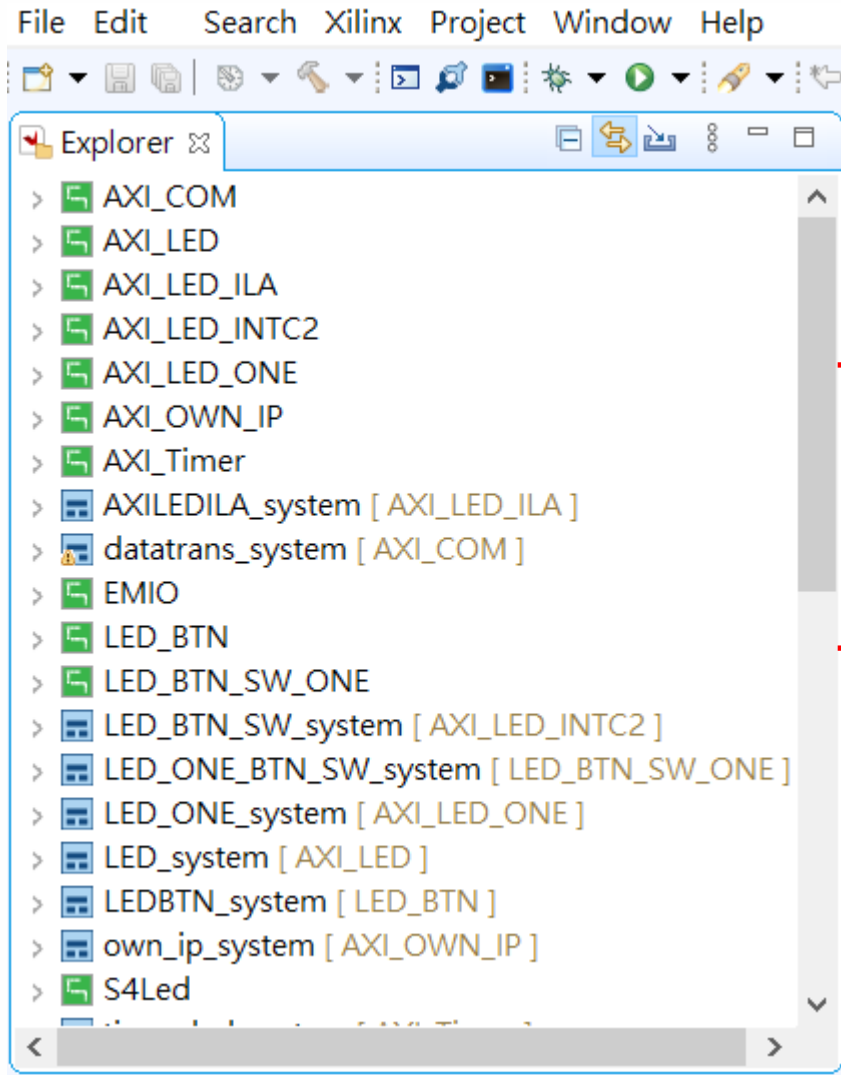
Platform

Please select a platform to create the project





# Create Application



# Vitis Build Problem

- Can't Build Own IP

```
"Compiling own_ip_for_leds..."
```

```
arm-xilinx-eabi-gcc.exe: error: *.c: Invalid argument  
arm-xilinx-eabi-gcc.exe: fatal error: no input files  
compilation terminated.
```

```
make[2]: *** [Makefile:18: libs] Error 1
```

```
make[1]: *** [Makefile:46: ps7_cortexa9_0/libsrc/own_ip_for_leds_v1_0/src/make.libs] Error 2
```

```
make: *** [Makefile:18: all] Error 2
```

```
Failed to build the bsp sources for domain - standalone_domain
```

```
Failed to generate the platform.
```

```
Reason: Failed to build the zynq_fsbl application.
```

# Vitis Build Problem

- Solution — **modify the makefile**
  1. hw\drivers\**<your IP name>**\src\Makefile
  2. ps7\_cortexa9\_0\standalone\_domain\bsp\ps7\_cortexa9\_0\libsrc\**<your IP name>**\src\Makefile
  3. zynq\_fsbl\zynq\_fsbl\_bsp\ps7\_cortexa9\_0\libsrc \**<your IP name>**\ src\Makefile

```
1 COMPILER=  
2 ARCHIVER=  
3 CP=cp  
4 COMPILER_FLAGS=  
5 EXTRA_COMPILER_FLAGS=  
6 LIB=libxil.a  
7  
8 RELEASEDIR=../../lib  
9 INCLUDEDIR=../../include  
10 INCLUDES=-I./ -I${INCLUDEDIR}  
11  
12 INCLUDEFILES=*.h  
13 LIBSOURCES=*.c  
14 OUTS = *.o  
15  
16 libs:  
17     echo "Compiling own_ip_for_leds..."  
18     $(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)  
19     $(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}  
20     make clean  
21  
22 include:  
23     ${CP} $(INCLUDEFILES) $(INCLUDEDIR)  
24  
25 clean:  
26     rm -rf ${OUTS}
```



```
1 COMPILER=  
2 ARCHIVER=  
3 CP=cp  
4 COMPILER_FLAGS=  
5 EXTRA_COMPILER_FLAGS=  
6 LIB=libxil.a  
7  
8 RELEASEDIR=../../lib  
9 INCLUDEDIR=../../include  
10 INCLUDES=-I./ -I${INCLUDEDIR}  
11  
12 INCLUDEFILES=$(wildcard *.h)  
13 LIBSOURCES=$(wildcard *.c *.cpp)  
14 OUTS = $(addsuffix .o, $(basename $(wildcard *.c)))  
15  
16 OBJECTS = $(addsuffix .o, $(basename $(wildcard *.c *.cpp)))  
17 ASSEMBLY_OBJECTS = $(addsuffix .o, $(basename $(wildcard *.S)))  
18  
19 libs:  
20     echo "Compiling myip..."  
21     $(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)  
22     $(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}  
23     make clean  
24  
25 include:  
26     ${CP} $(INCLUDEFILES) $(INCLUDEDIR)  
27  
28 clean:  
29     rm -rf ${OUTS}
```

# Example Code

- Light the Leds

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
```

```
int main()
```

```
{
```

```
    init_platform();
```

You can check this variable in "xparameters.h" to find your own ip name

```
    volatile unsigned int *led = (unsigned int *)XPAR_OWN_IP_FOR_LEDS_0_LED_AXI_BASEADDR;
```

```
    while(1) {
```

```
        *led = 255;
```

```
    }
```

```
    cleanup_platform();
```

```
    return 0;
```

```
}
```



# Results

