Vivado & Vitis — Add Your Own IP Workflow

XILINX.

Agenda

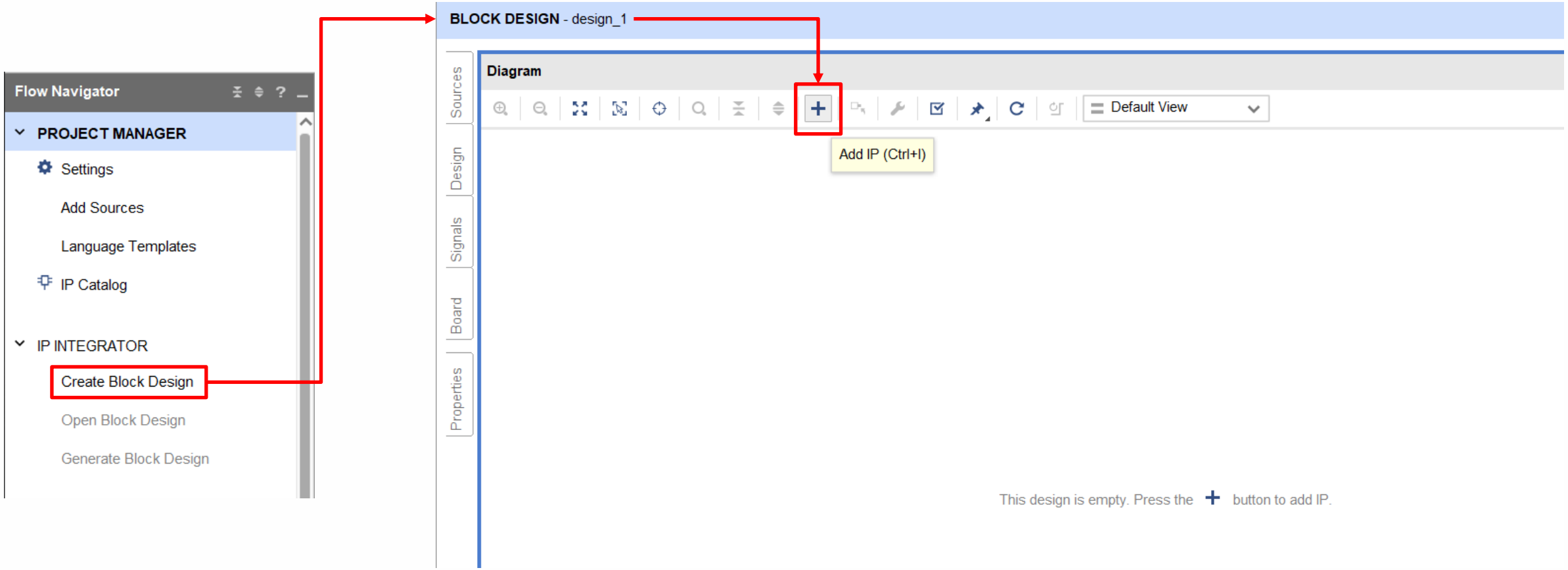
- Vivado — Add Your Own IP
- Vitis — A Sample for Driving Own IP

Vivado – Add Your Own IP

XILINX.

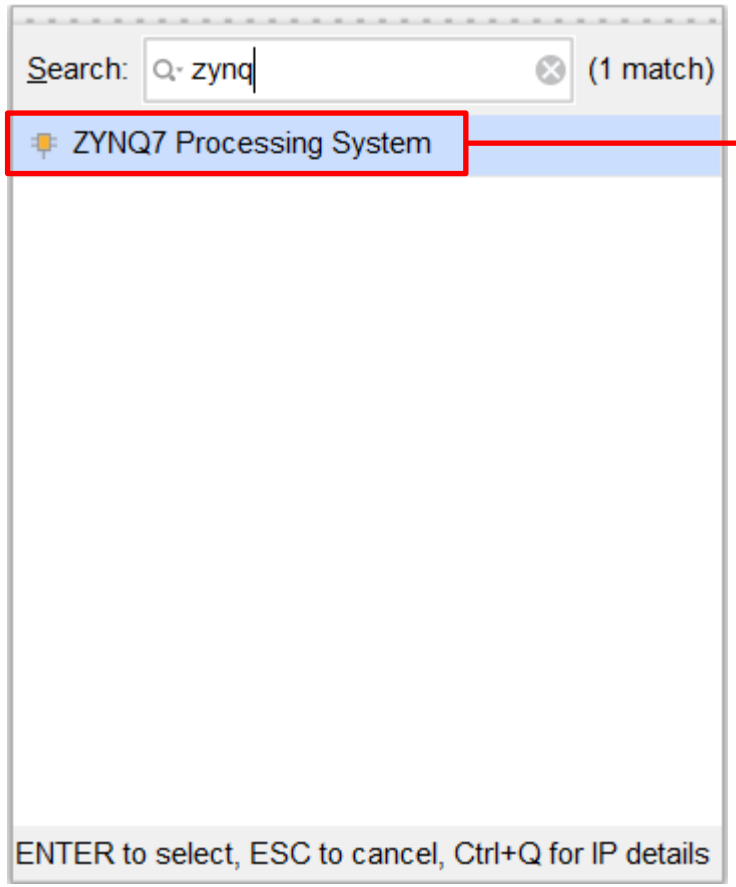
Block Design

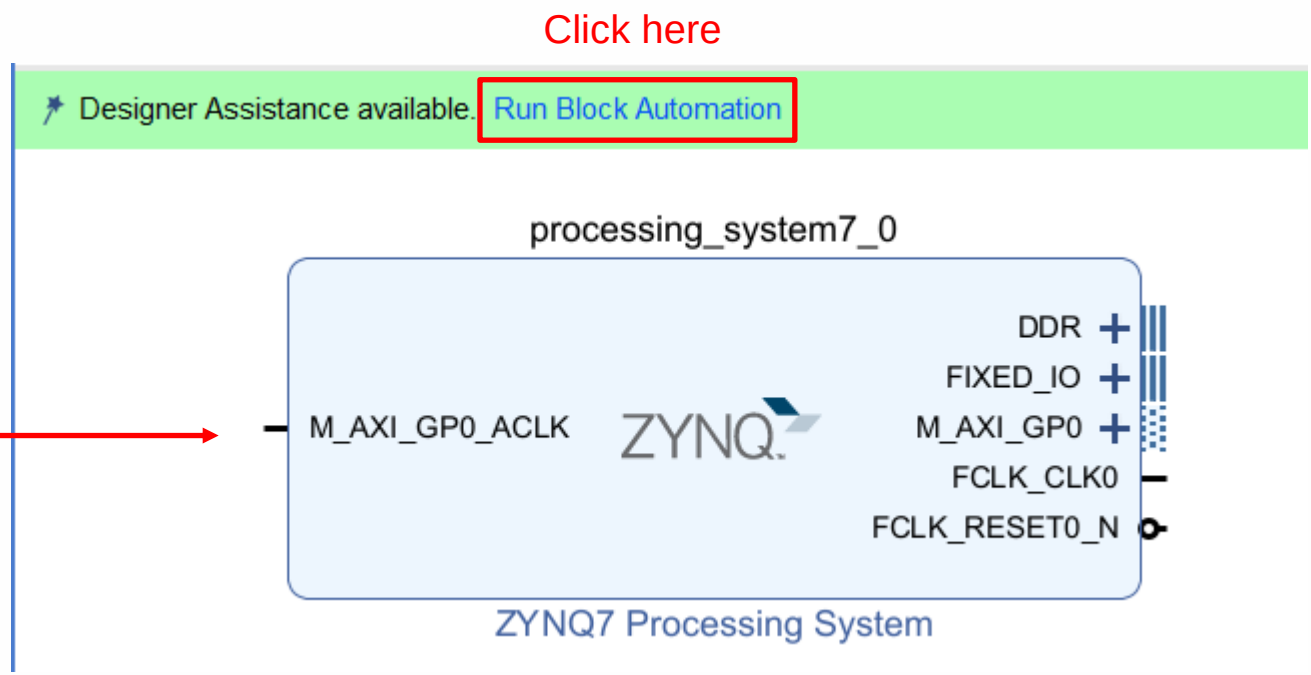
- Step 1: Add Zynq Processing System IP



Block Design

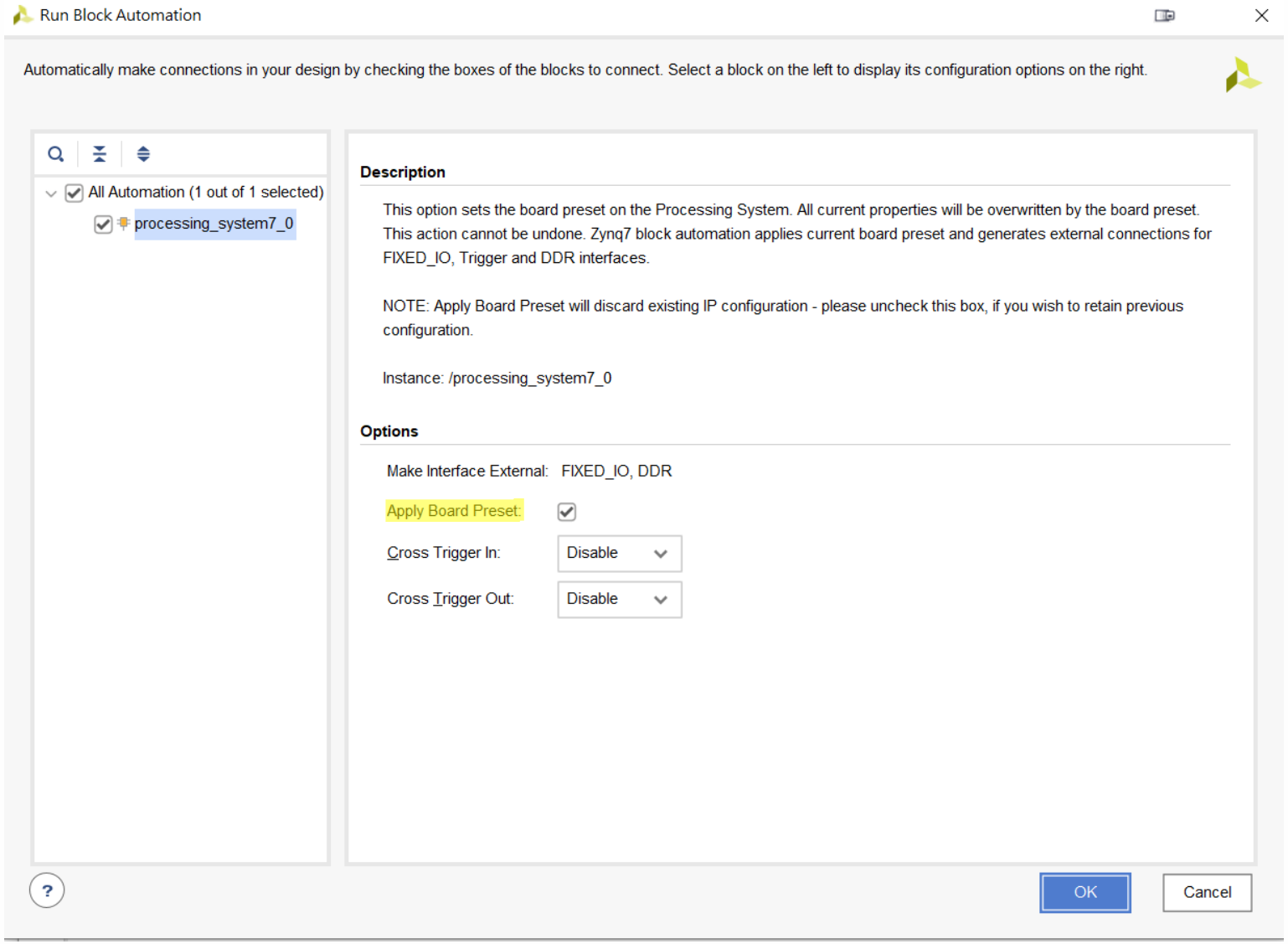
- Step 1: Add Zynq Processing System IP





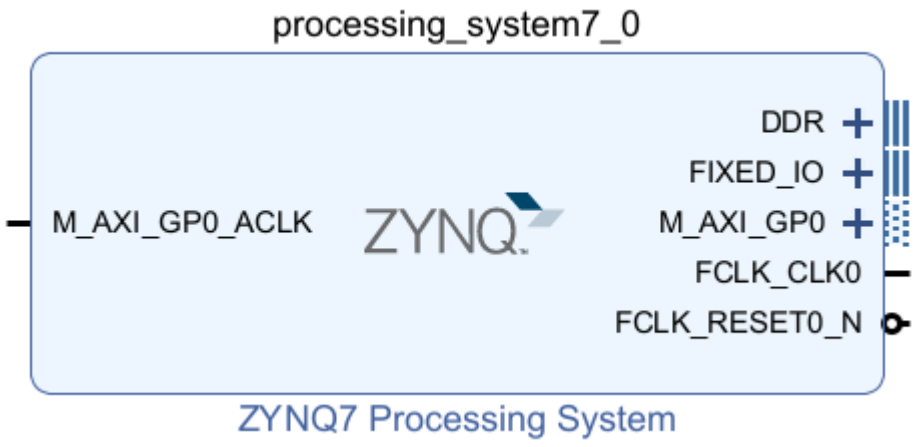
Block Design

- Step 1: Add Zynq Processing System IP

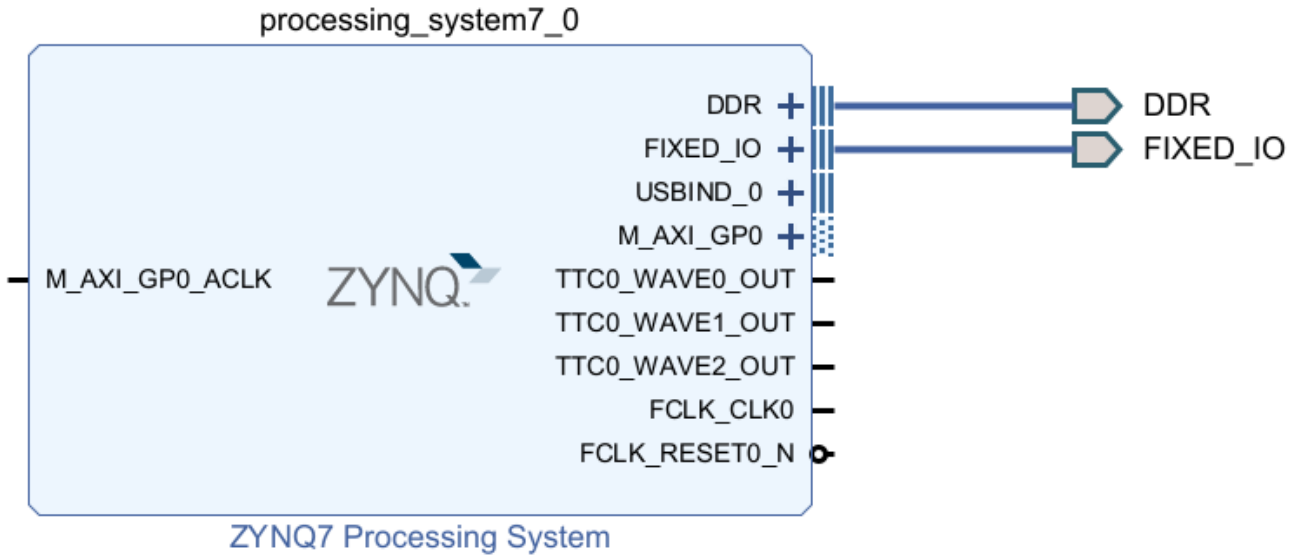


Block Design

- Step 1: Add Zynq Processing System IP



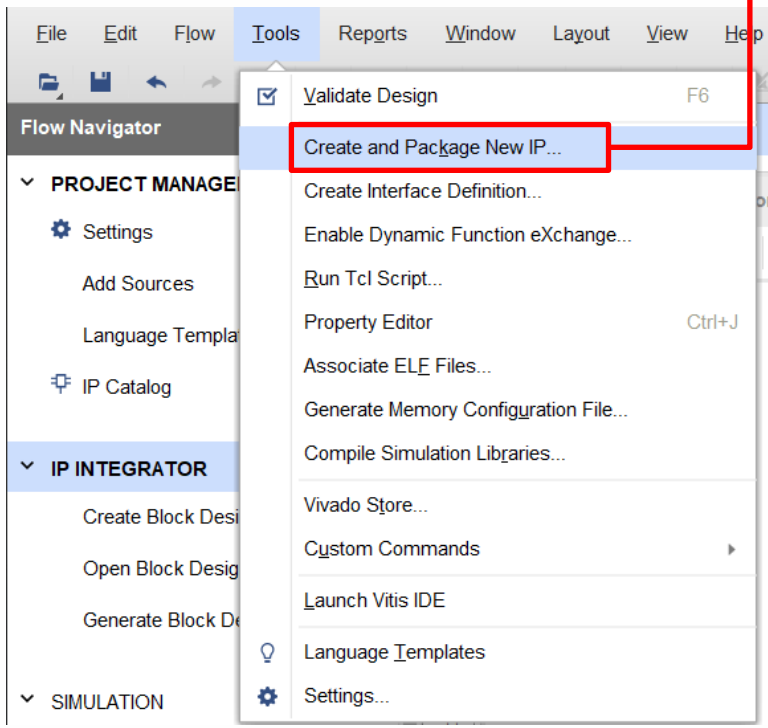
Before

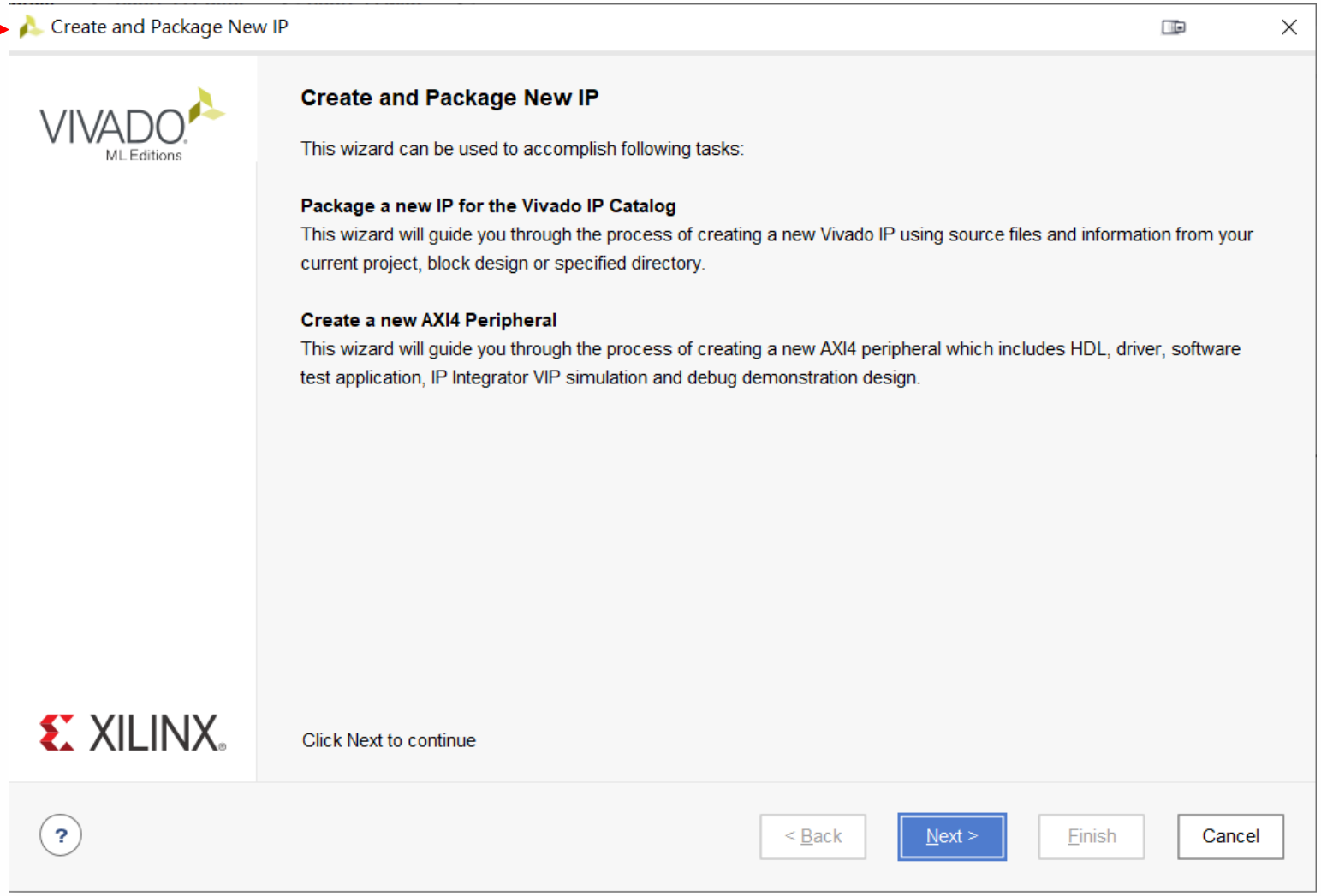


After

Block Design

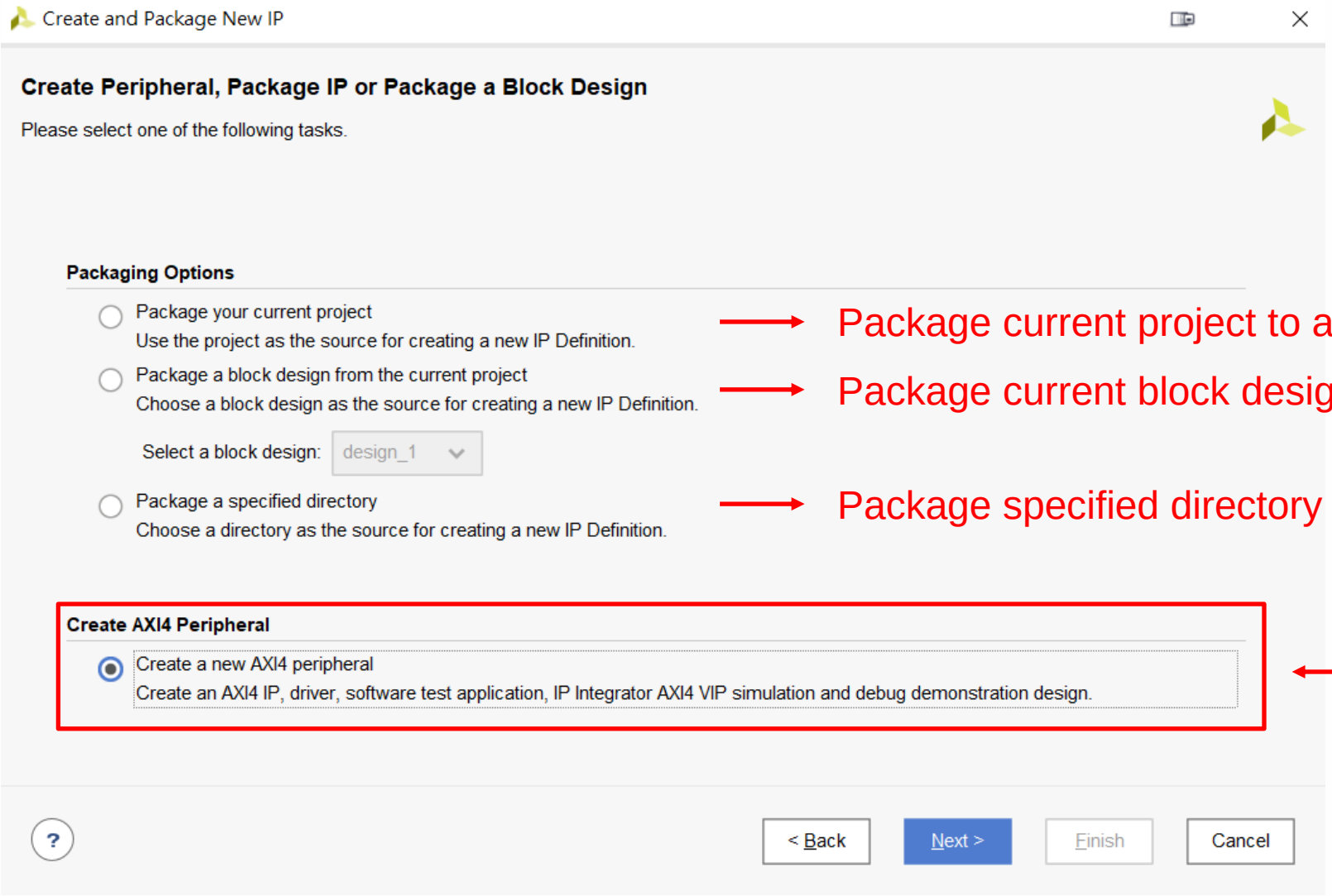
- Step 2: Create Your Own IP





Block Design

- Step 2: Create Your Own IP



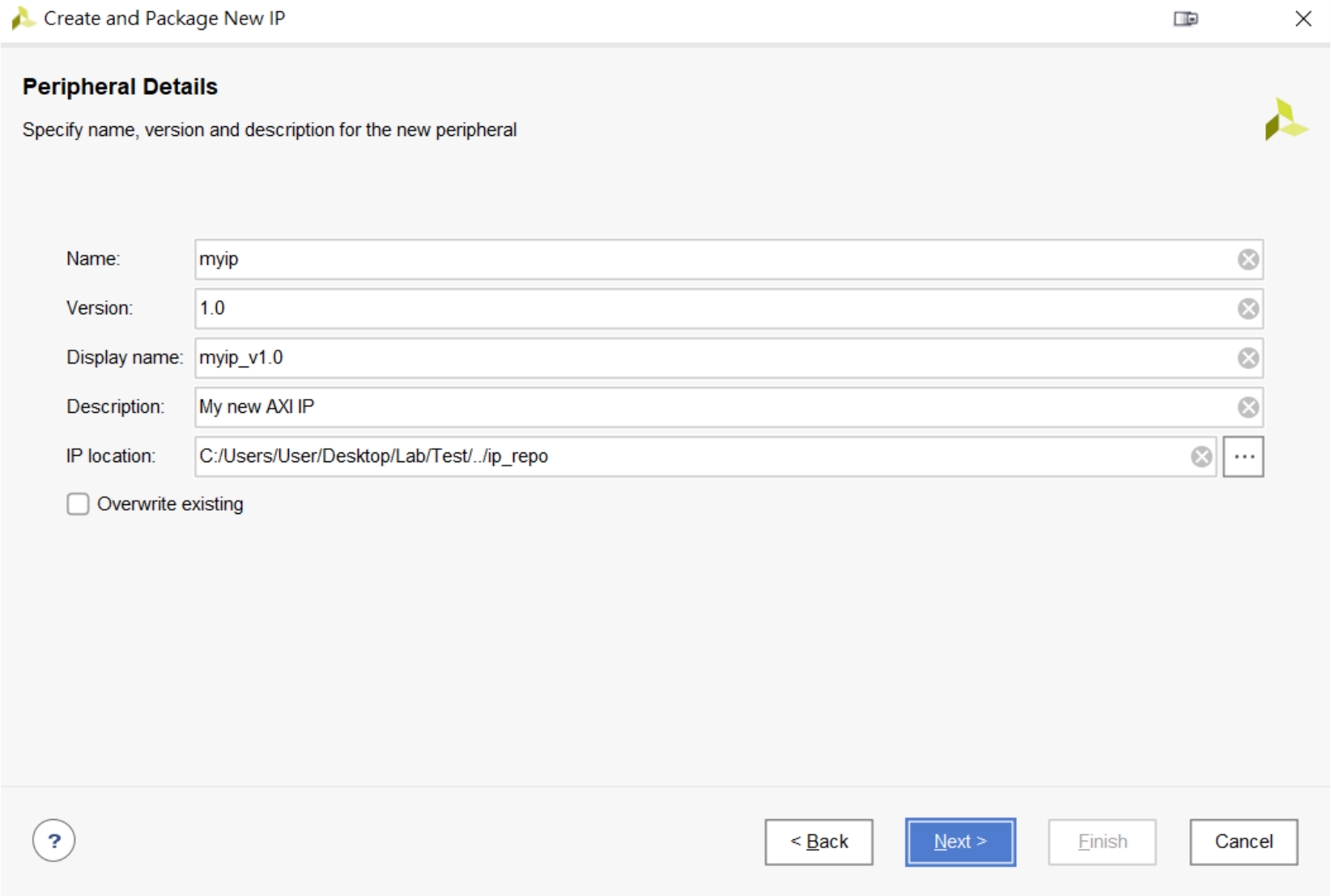
Package current project to a new IP

Package current block design to a new IP

Package specified directory to a new IP

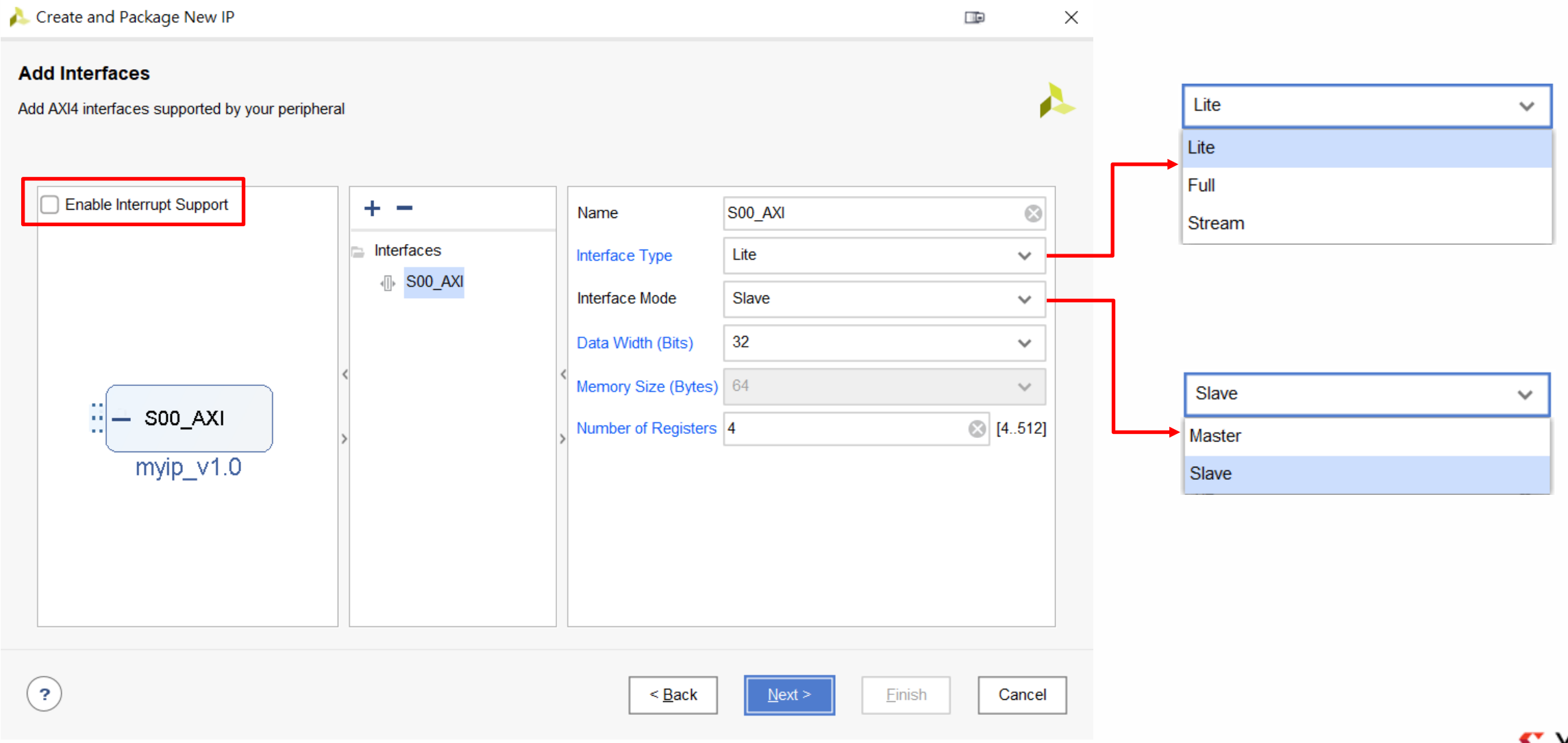
Block Design

- Step 2: Create Your Own IP



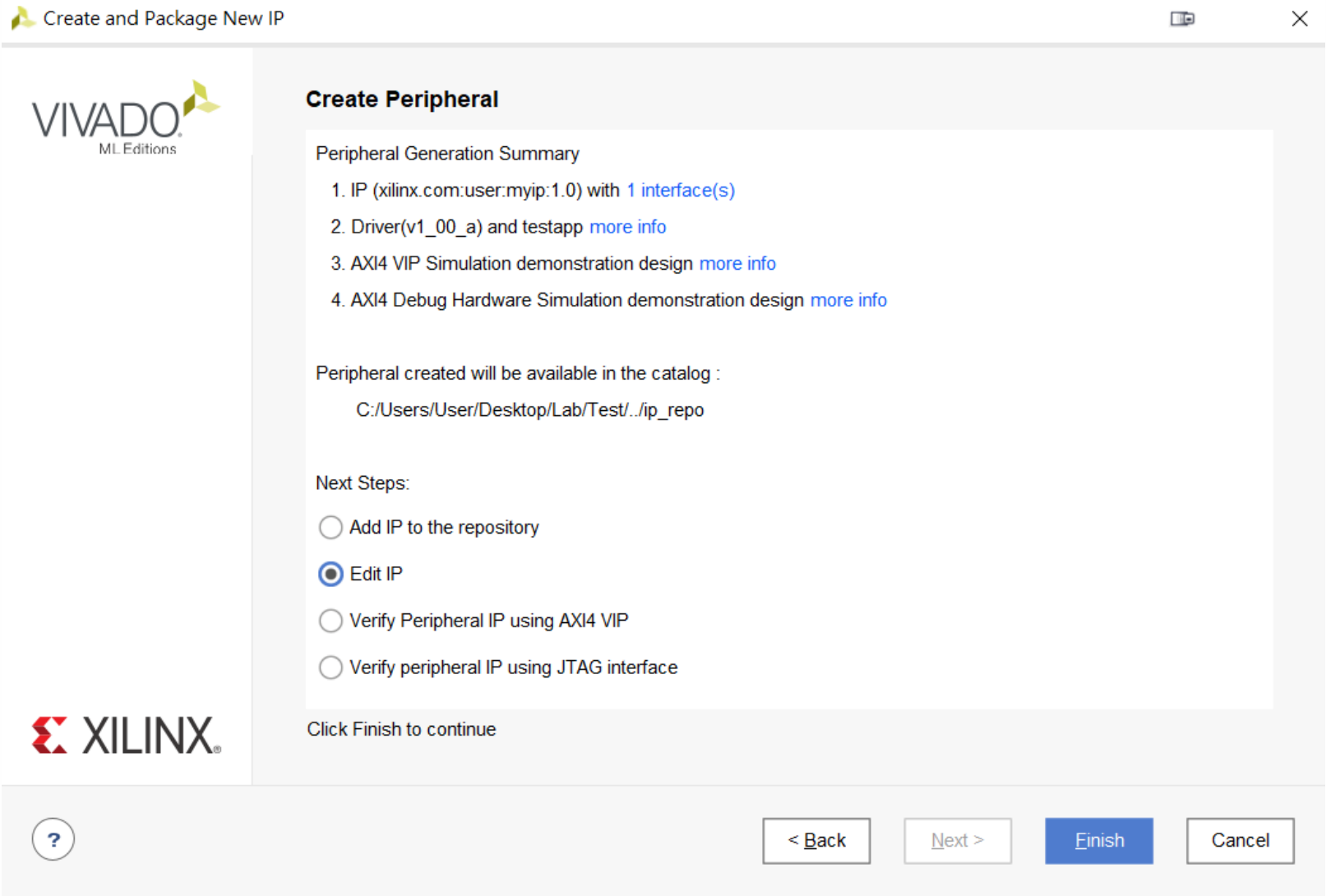
Block Design

- Step 2: Create Your Own IP



Block Design

- Step 2: Create Your Own IP



Block Design

- Step 3: Edit Your Own IP

PROJECT MANAGER - myip_v1_0_project

Sources

- Design Sources (2)
  - **myip_v1_0** (myip_v1_0.v) (1)
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)
- Utility Sources

Hierarchy | Libraries | Compile Order

Properties

Select an object to see properties

Project Summary | **Package IP - myip**

Packaging Steps

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

Identification

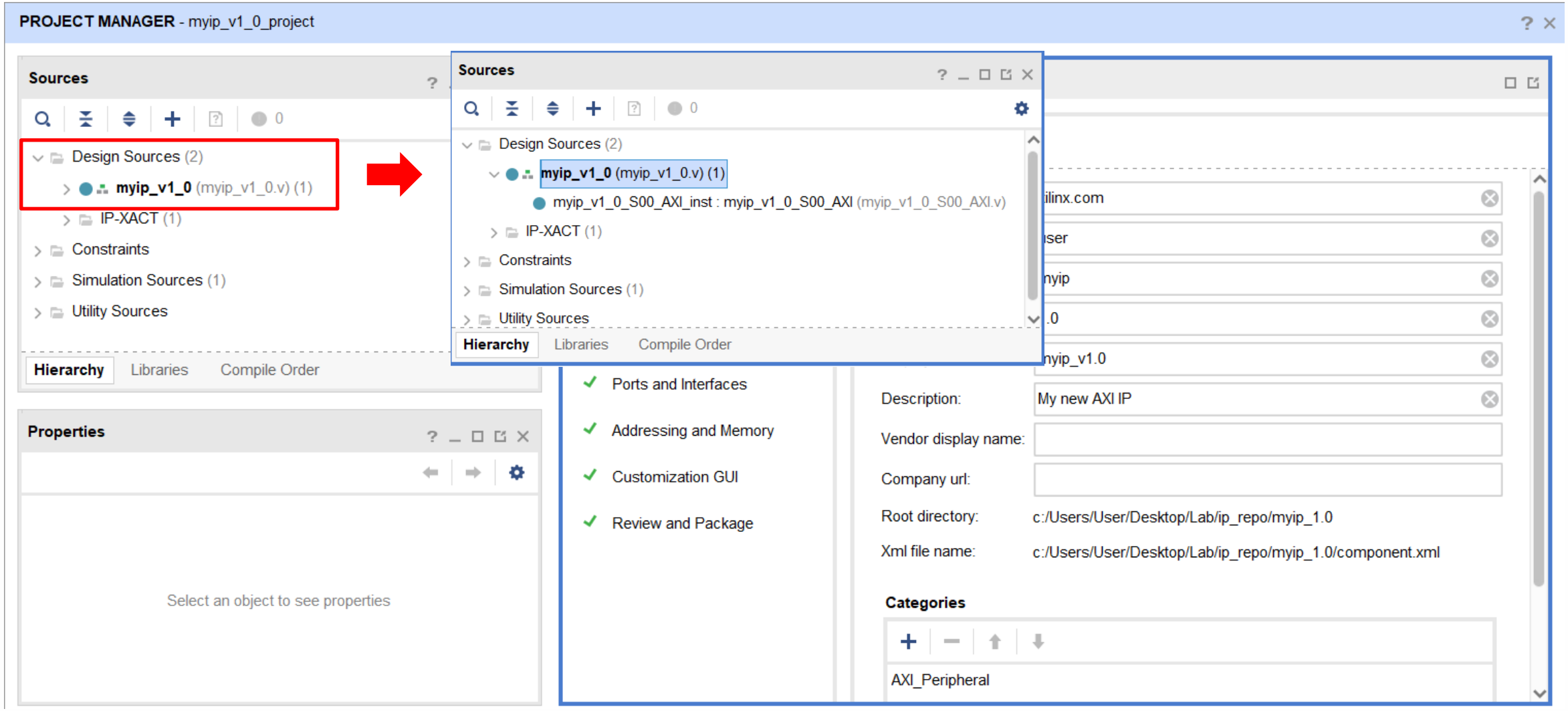
Vendor:	xilinx.com
Library:	user
Name:	myip
Version:	1.0
Display name:	myip_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	c:/Users/User/Desktop/Lab/ip_repo/myip_1.0
Xml file name:	c:/Users/User/Desktop/Lab/ip_repo/myip_1.0/component.xml

Categories

AXI_Peripheral

Block Design

- Step 3: Edit Your Own IP



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,
```

XILINX

Block Design

- Step 3: Edit Your Own IP — add output of leds
  - ➢ myip_v1_0.v

```
// Instantiation of Axi Bus Interface S00_AXI
    myip_v1_0_S00_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
    ) myip_v1_0_S00_AXI_inst (
        .led(led),
        .S_AXI_ACLK(s00_axi_aclk),
        .S_AXI_ARESETN(s00_axi_aresetn),
        .S_AXI_AWADDR(s00_axi_awaddr),
        .S_AXI_AWPROT(s00_axi_awprot),
        .S_AXI_AWVALID(s00_axi_awvalid),
        .S_AXI_AWREADY(s00_axi_awready),
        .S_AXI_WDATA(s00_axi_wdata),
        .S_AXI_WSTRB(s00_axi_wstrb),
        .S_AXI_WVALID(s00_axi_wvalid),
        .S_AXI_WREADY(s00_axi_wready),
        .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 )
```

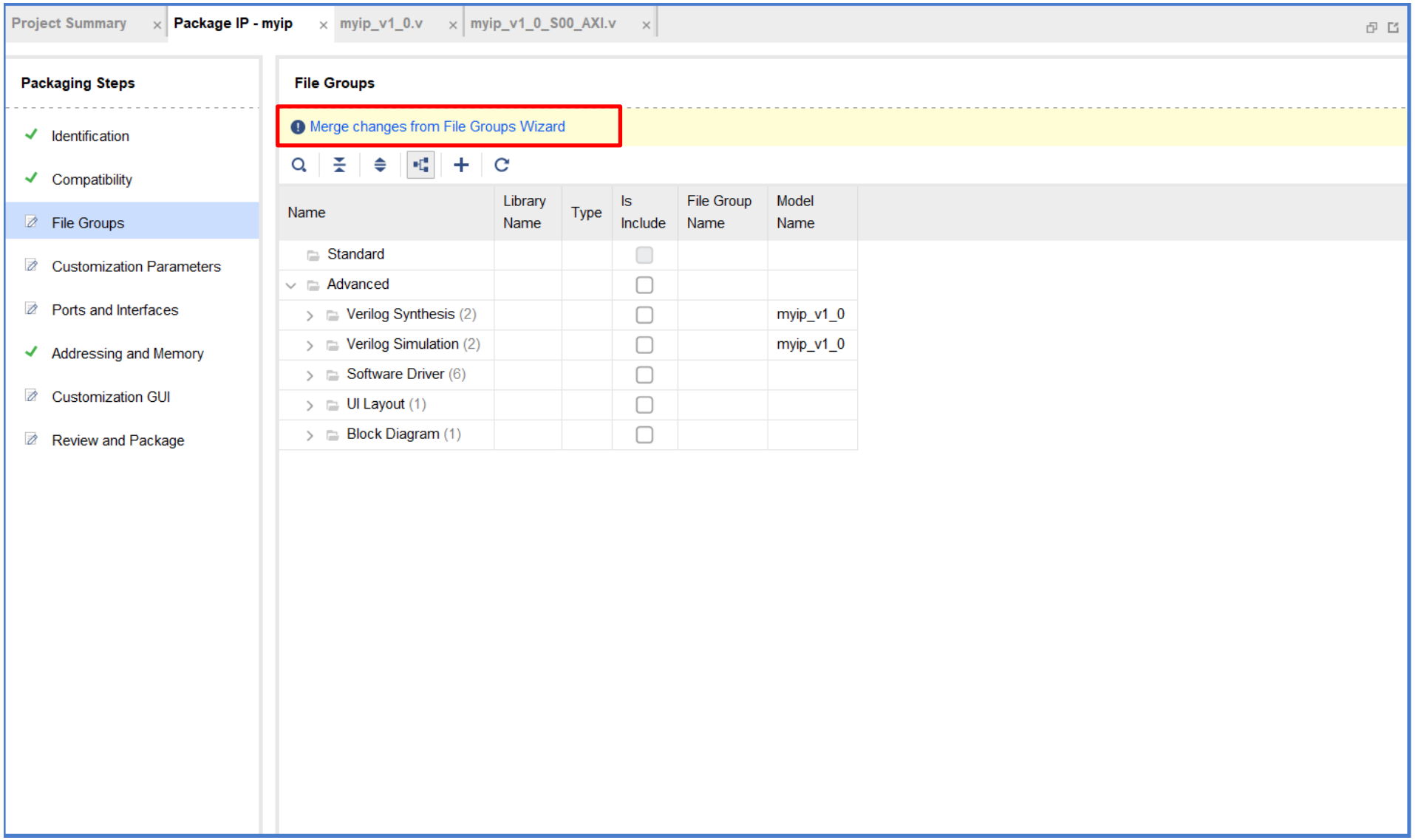
Block Design

- Step 3: Edit Your Own IP — add output of leds
  - ➢ myip_v1_0_S00_AXI.v

```
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     // Implement axi_awready generation
131     // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
132     // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
133     // de-asserted when reset is low.
134
135     always @( posedge S_AXI_ACLK )
136     begin
```

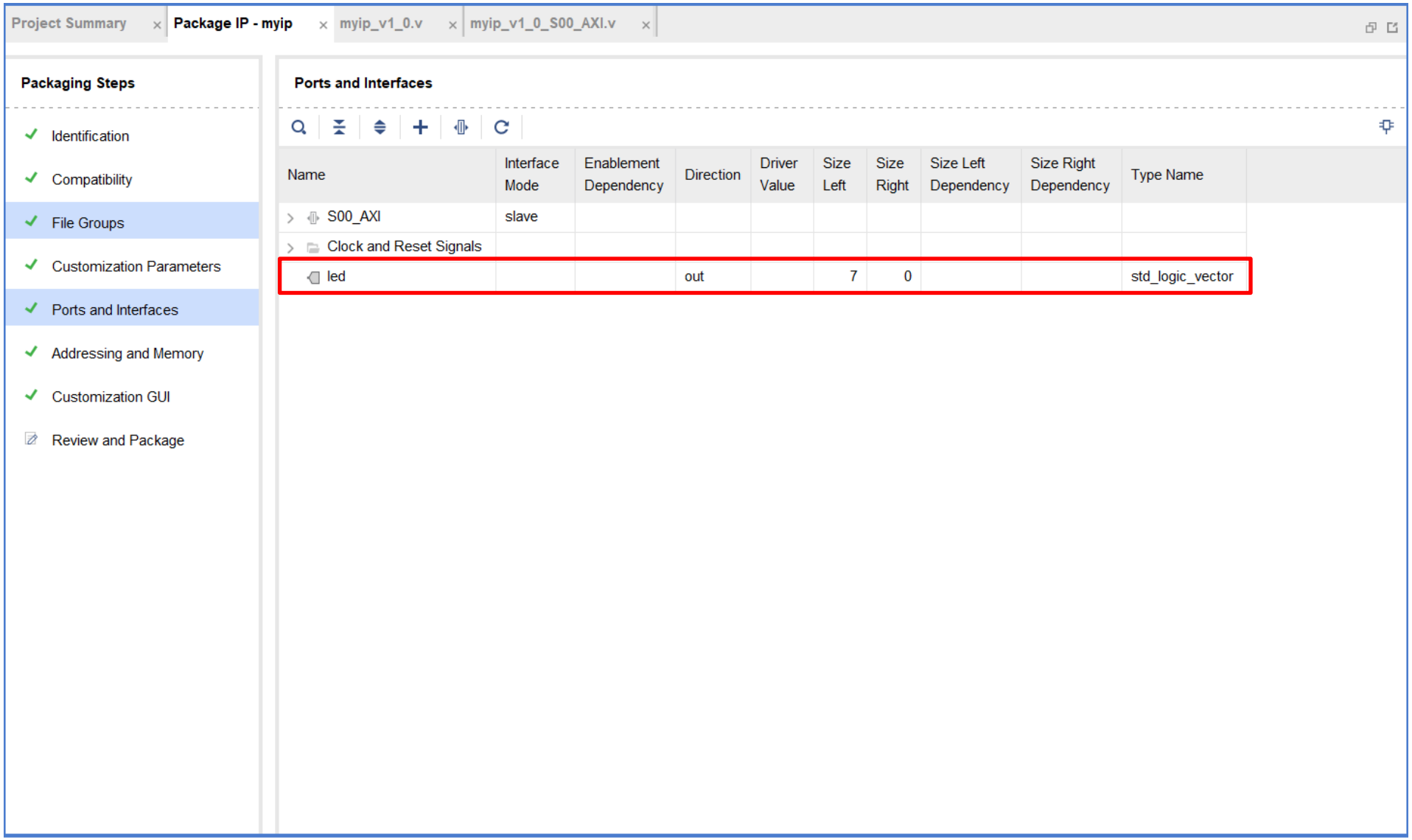
Block Design

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



Block Design

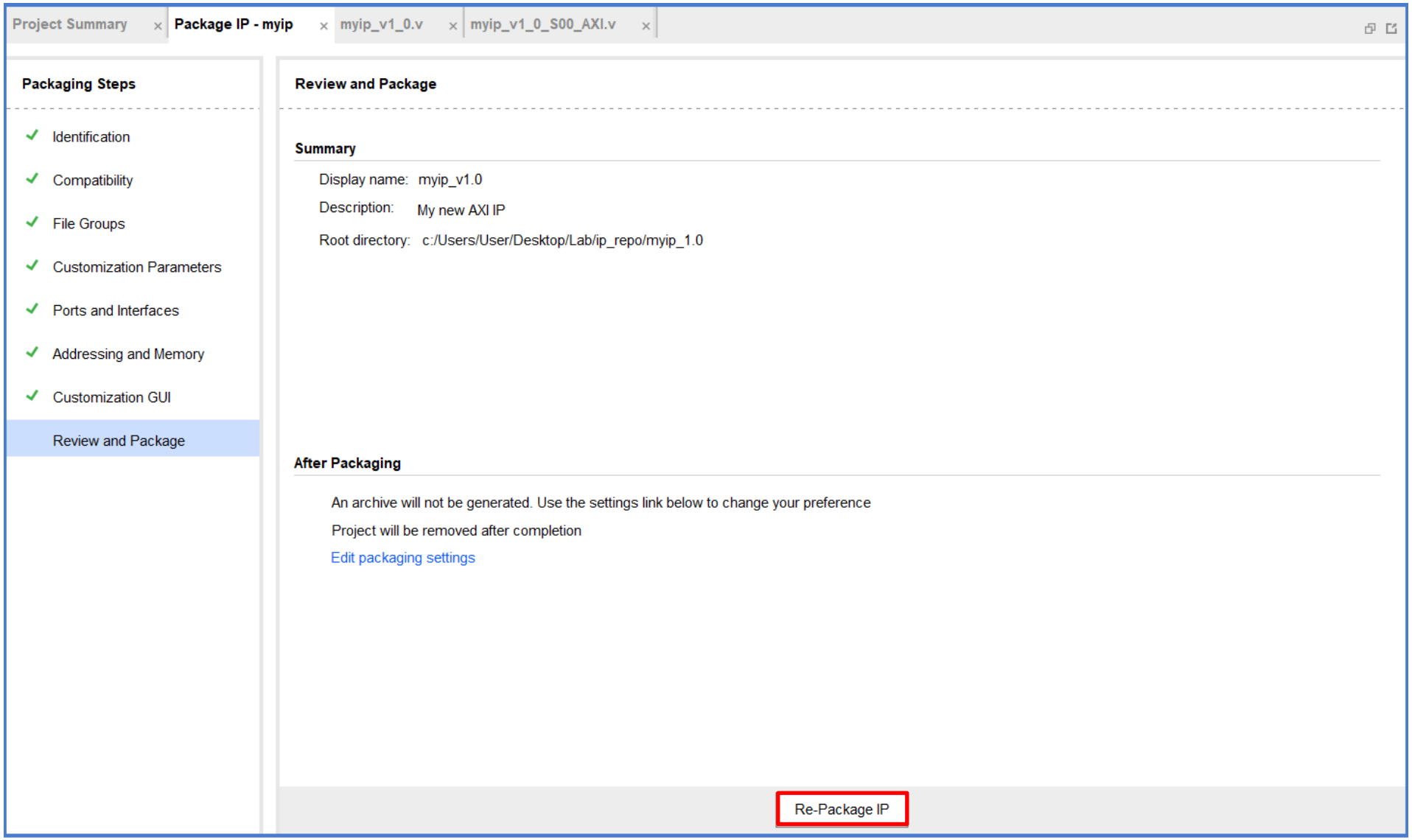
- Step 3: Edit Your Own IP — merge changes and package



XILINX

Block Design

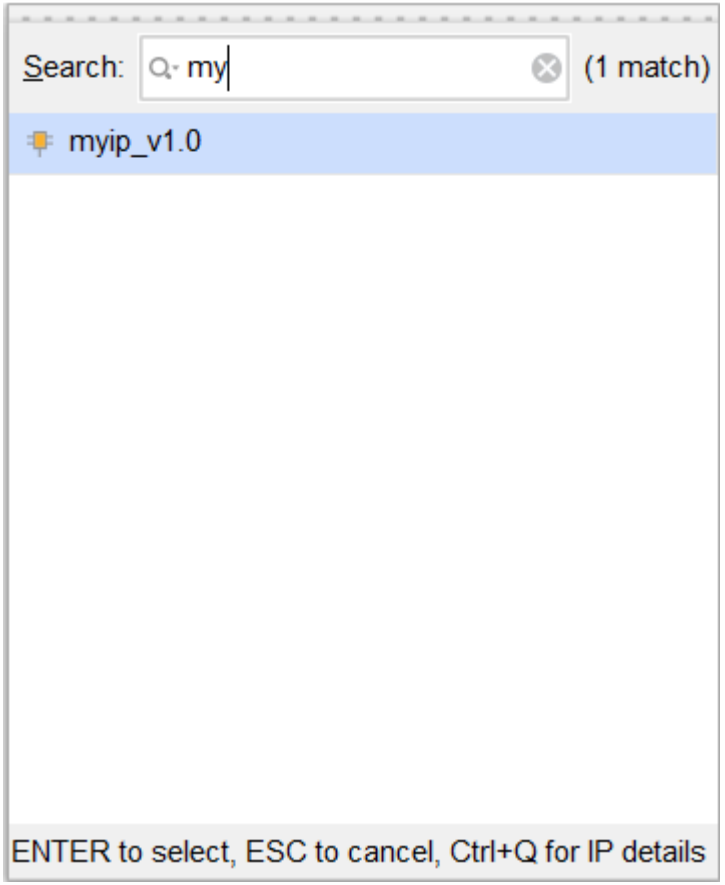
- Step 3: Edit Your Own IP — merge changes and package

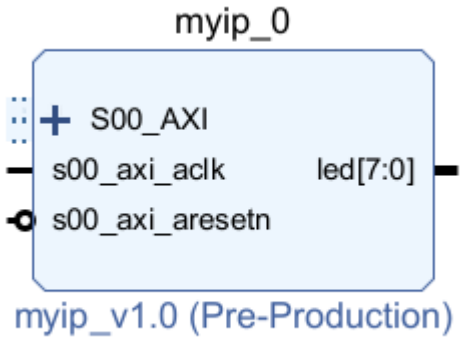


XILINX

Block Design

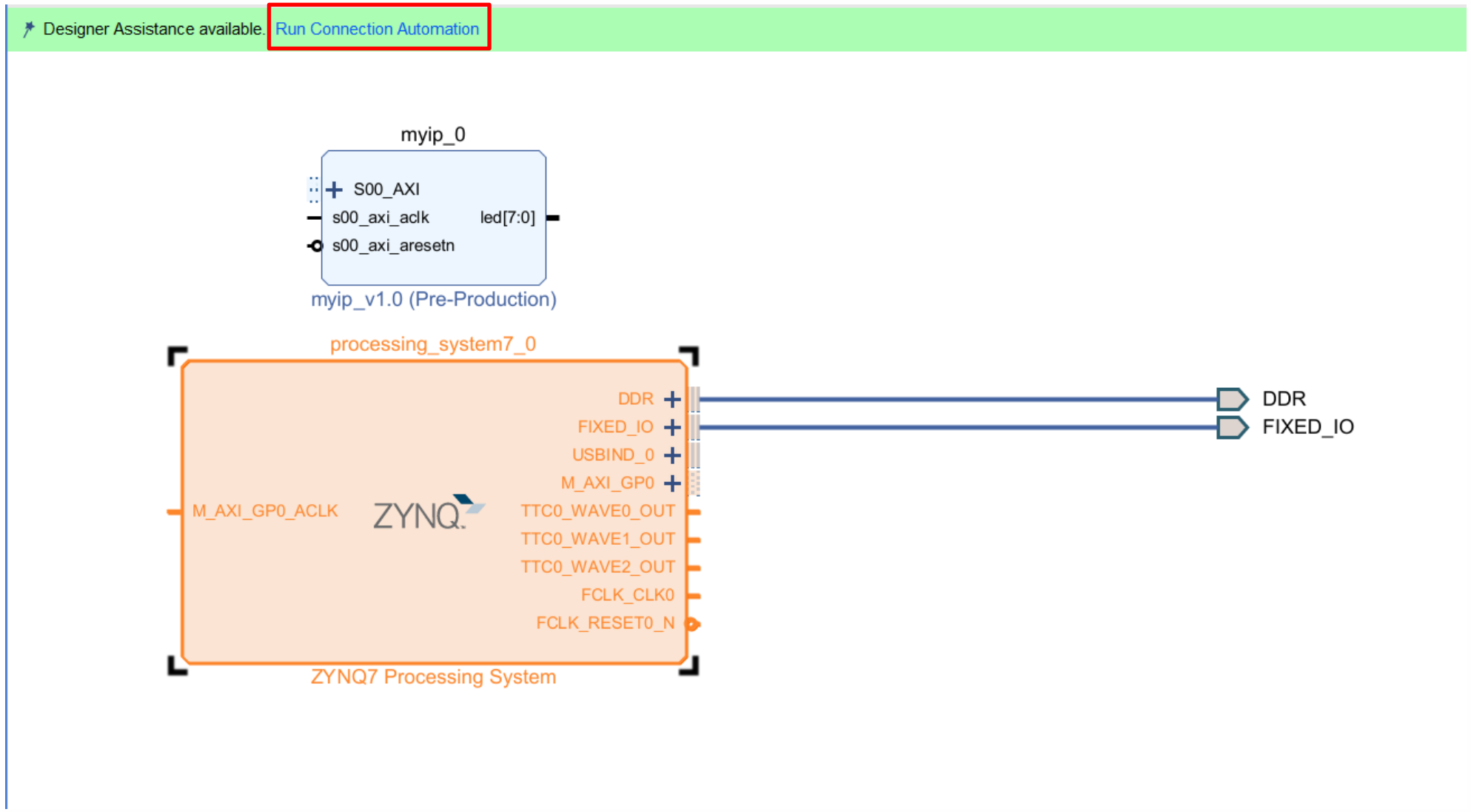
- Step 4: Add Own IP to Block Design





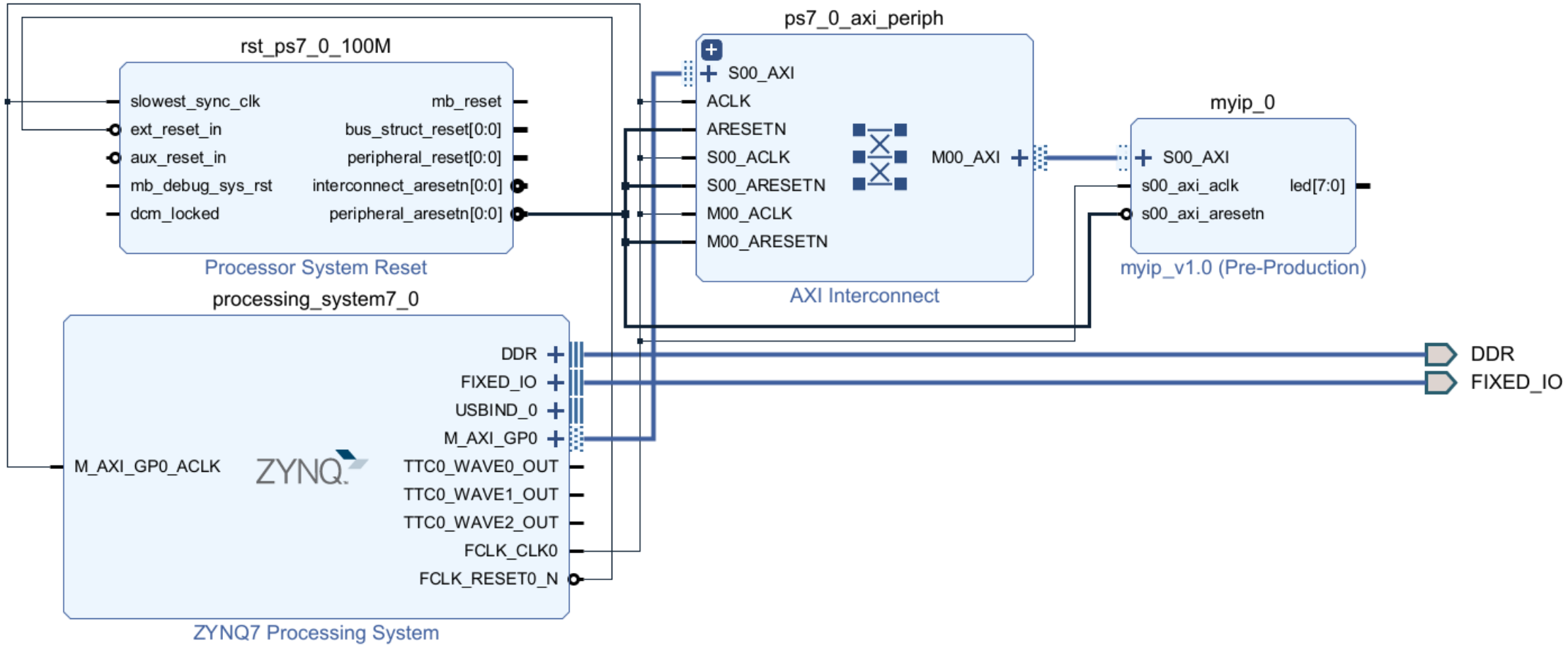
Block Design

- Step 4: Add Own IP to Block Design



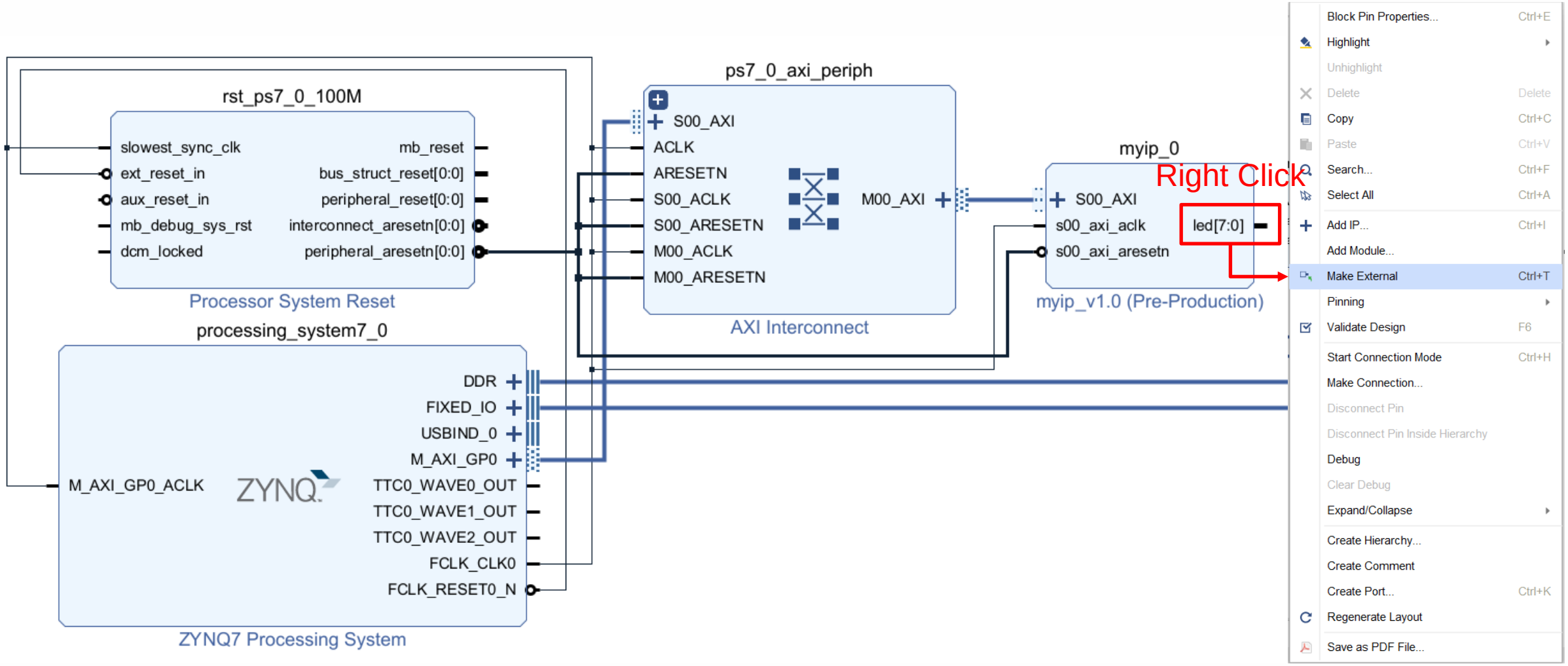
Block Design

- Step 4: Add Own IP to Block Design



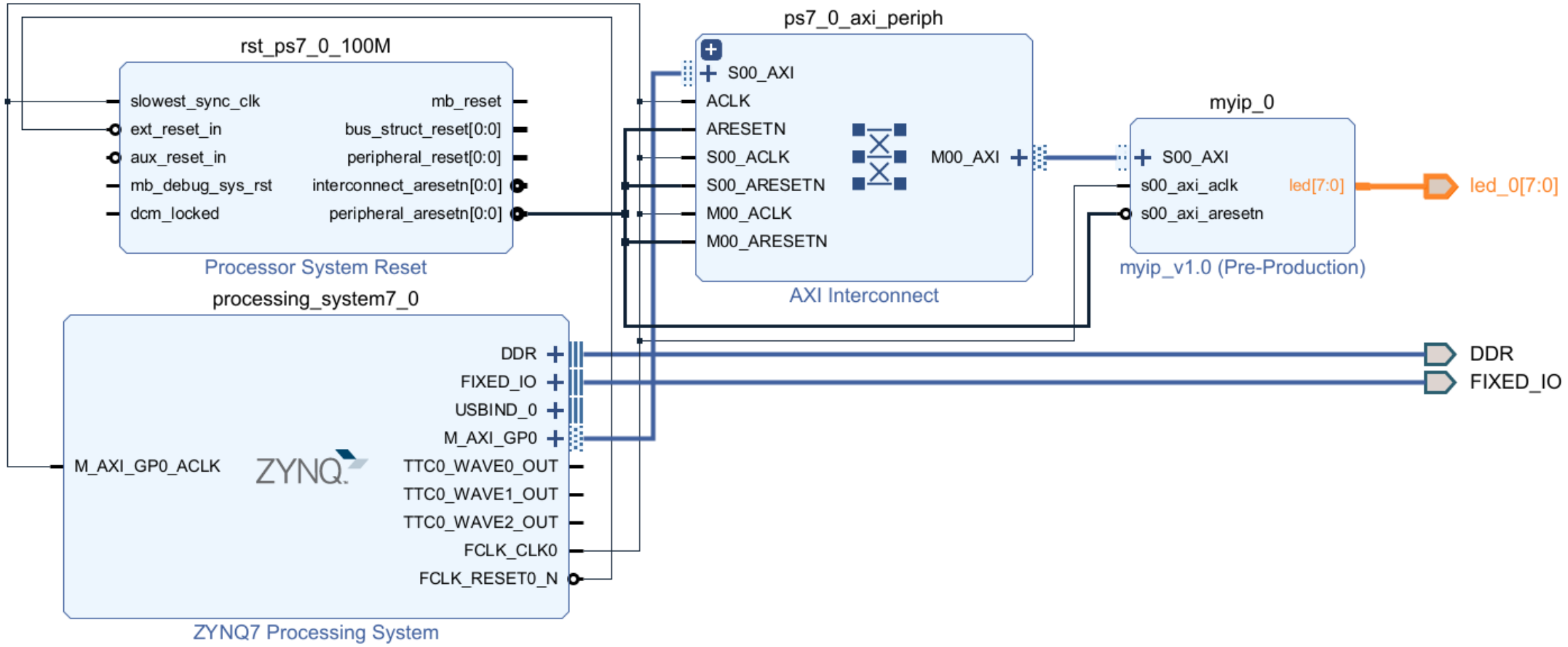
Block Design

- Step 4: Add Own IP to Block Design



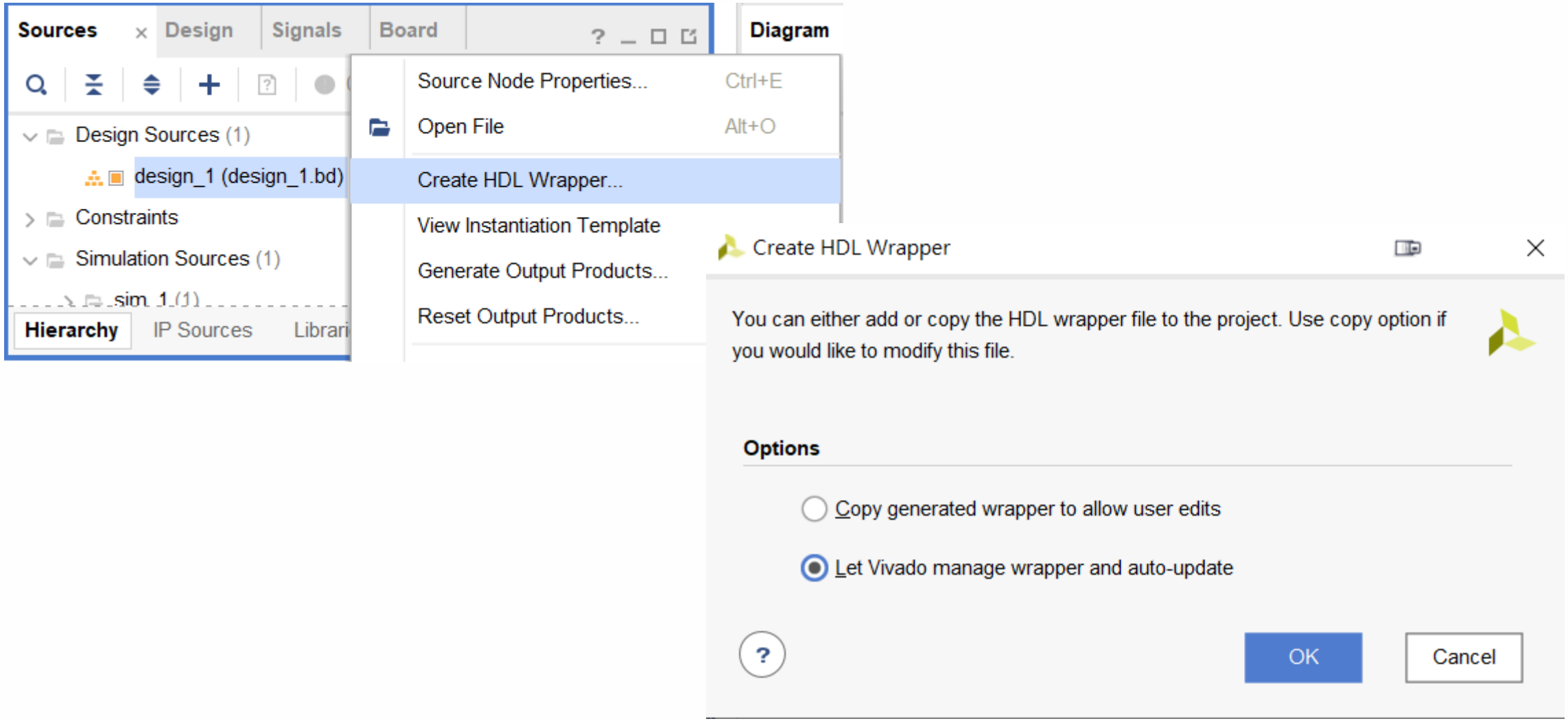
Block Design

- Step 4: Add Own IP to Block Design



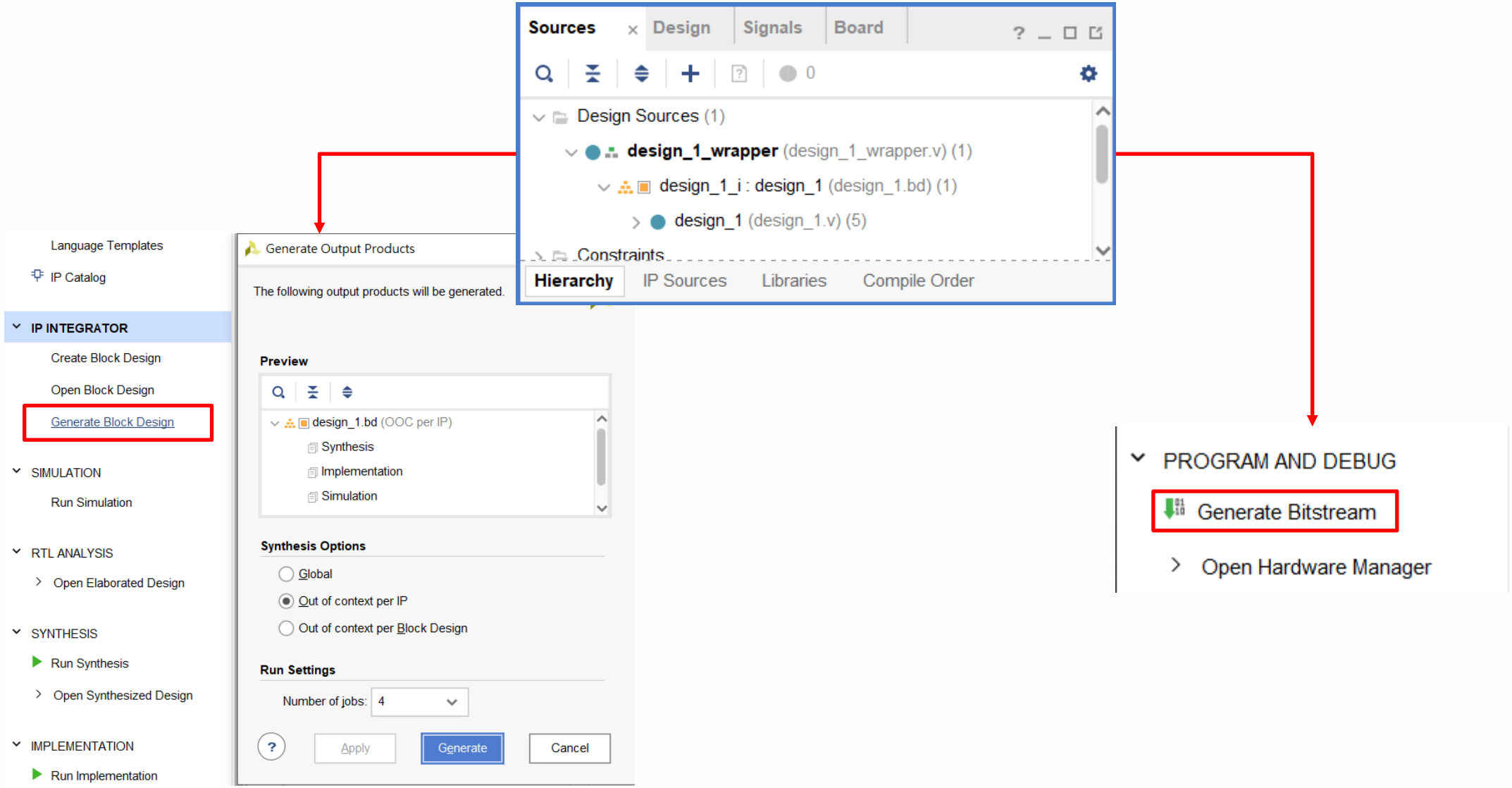
Block Design

- Step 5: Generate XSA file for Vitis



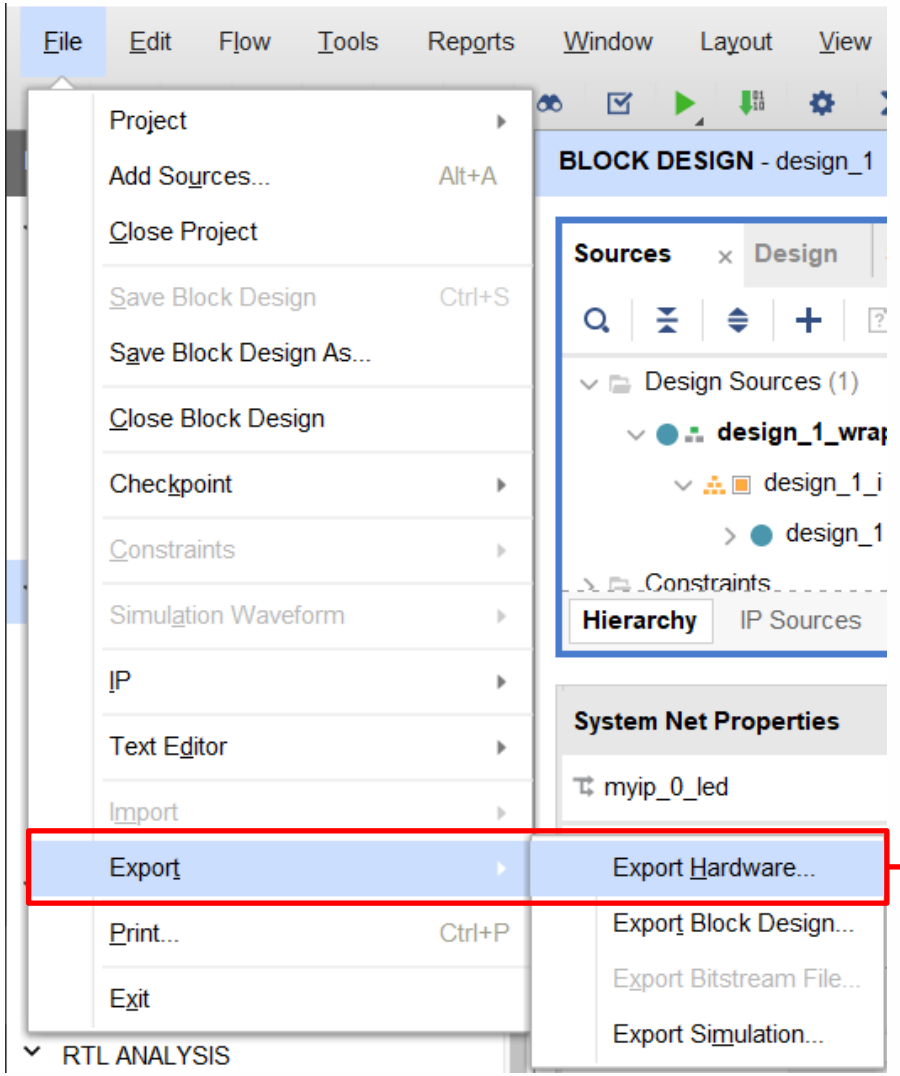
Block Design

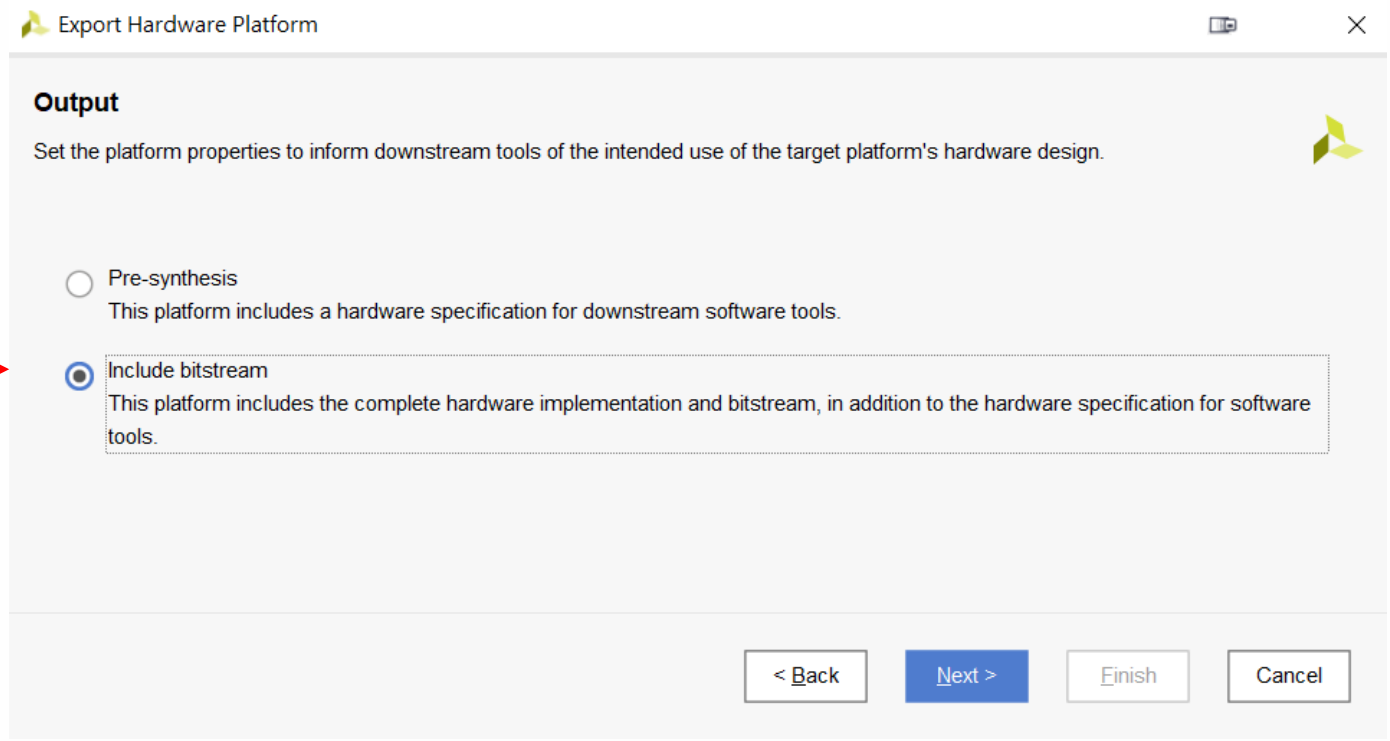
- Step 5: Generate XSA file for Vitis



Block Design

- Step 5: Generate XSA file for Vitis

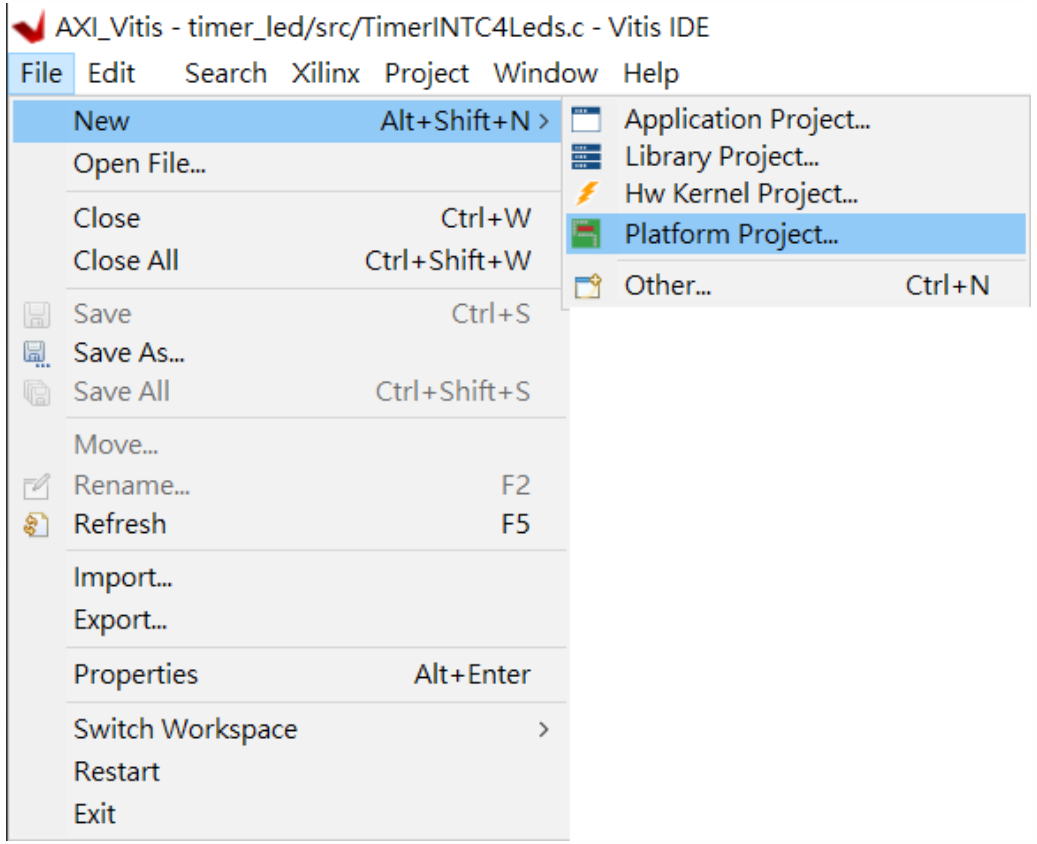


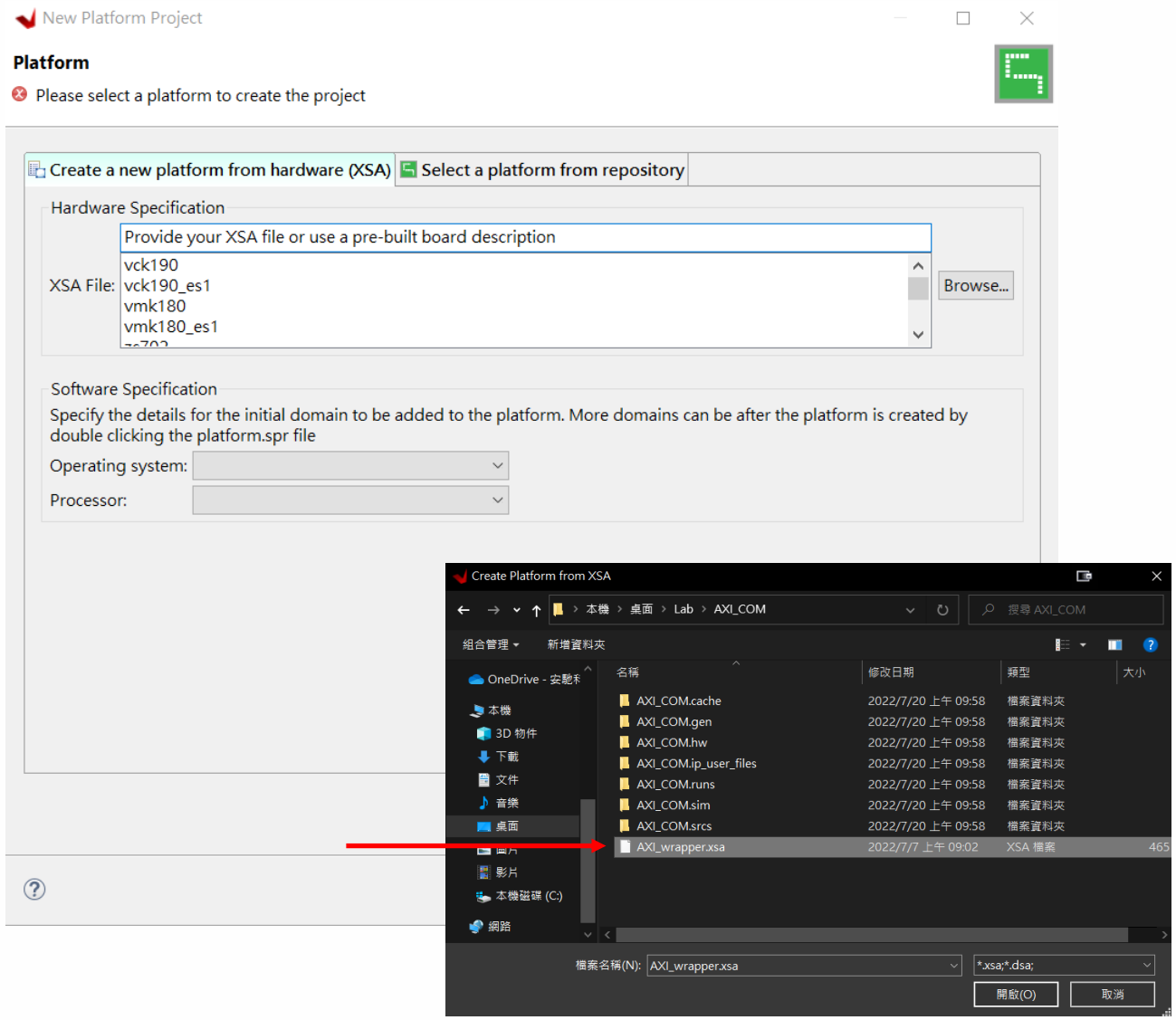


Vitis – A Sample for Driving Own IP

XILINX.

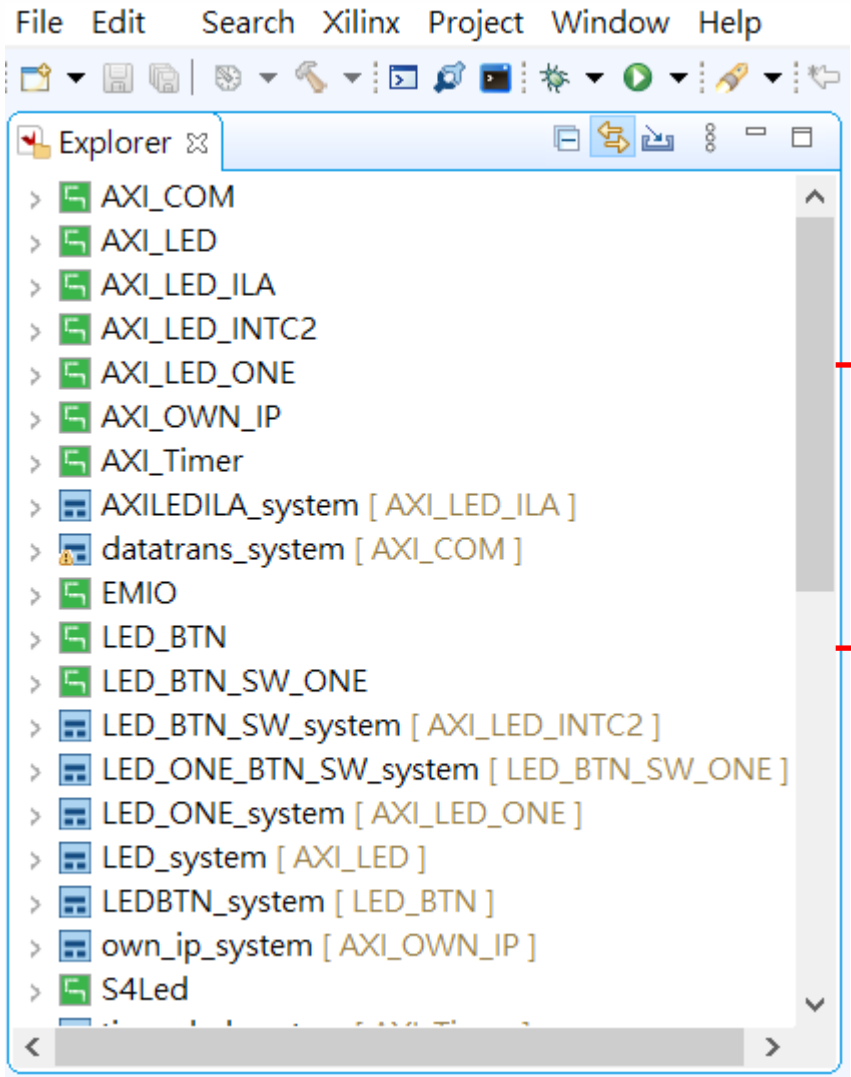
Create Platform

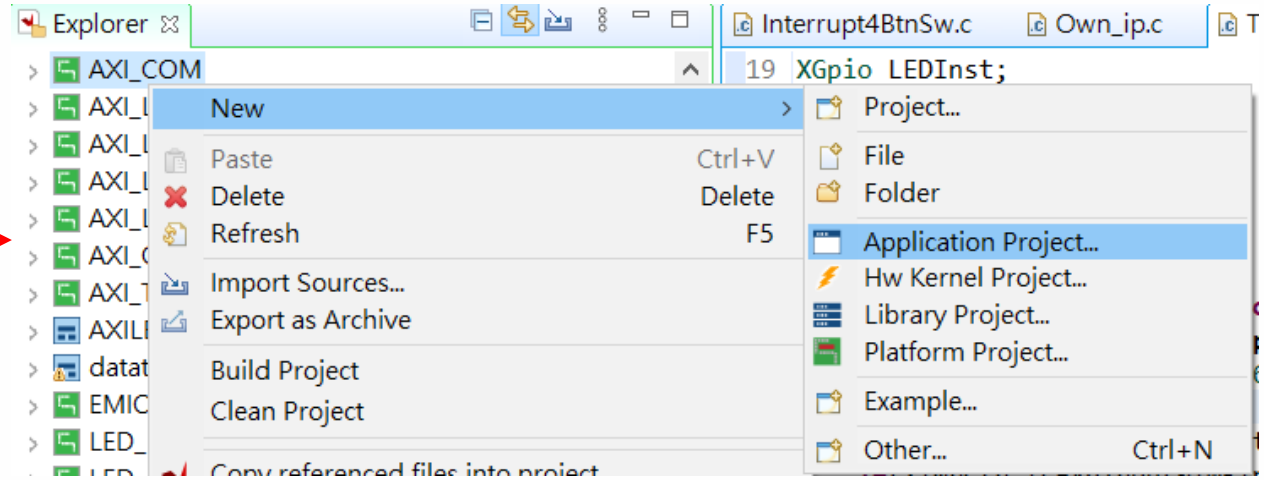


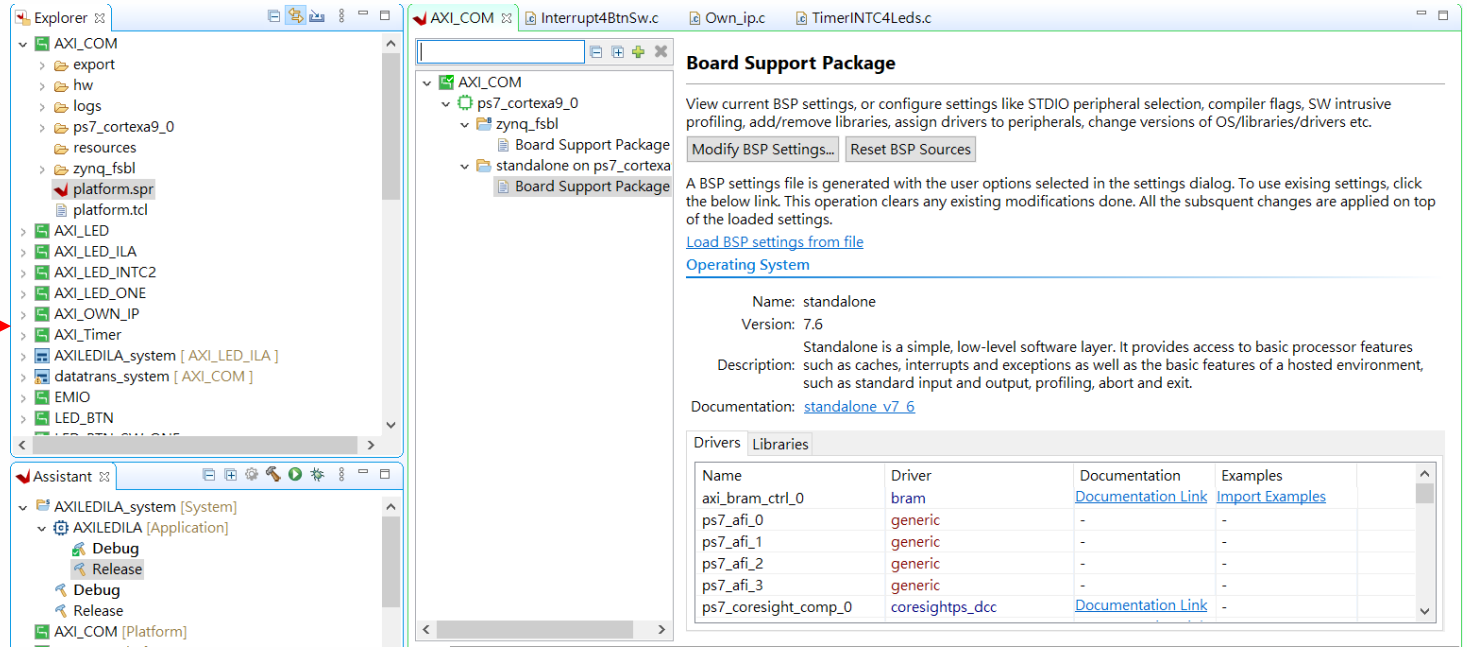


XILINX

Create Application







XILINX

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

```makefile
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

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

libs:
	echo "Compiling own_ip_for_leds..."
	$(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
	$(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
	make clean

include:
	${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
	rm -rf ${OUTS}
```

```makefile
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c *.cpp)
OUTS = $(addsuffix .o, $(basename $(wildcard *.c)))

OBJECTS = $(addsuffix .o, $(basename $(wildcard *.c *.cpp)))
ASSEMBLY_OBJECTS = $(addsuffix .o, $(basename $(wildcard *.S)))

libs:
	echo "Compiling myip..."
	$(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
	$(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
	make clean

include:
	${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
	rm -rf ${OUTS}
```

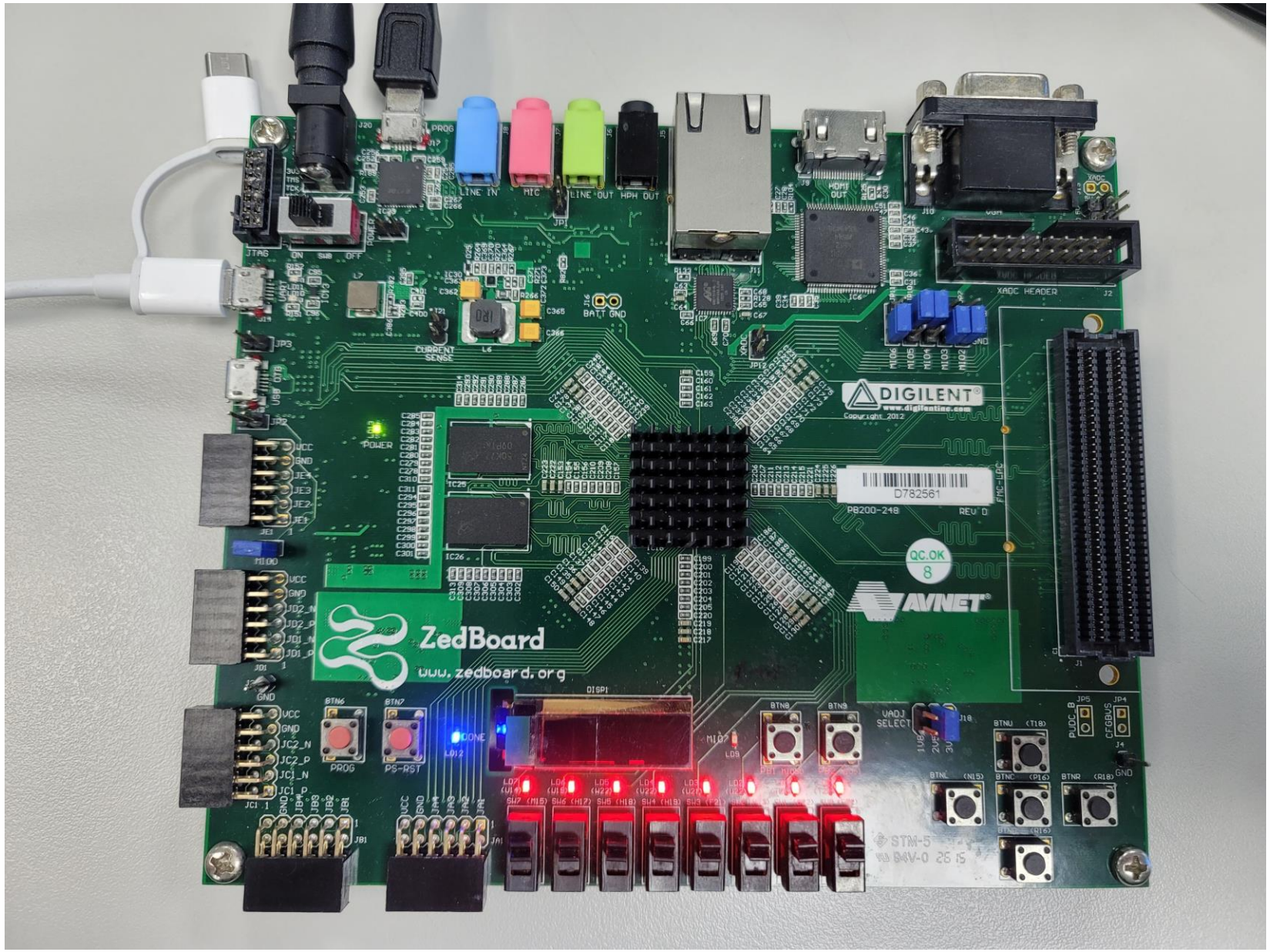
Example Code

- Light the Leds

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

int main()
{
    init_platform();

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


    while(1) {
        *led = 255;
    }

    cleanup_platform();
    return 0;
}
```

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

Results


ZedBoard
www.zedboard.org
DIGILENT
AVNET
QC.OK
D782561
PROG
PS-RST
XADC HEADER
LINE IN
MIC
LINE OUT
HPH OUT
HDMI OUT
JTAG
POWER
BATT GND
CURRENT SENSE
VADJ SELECT