

# Vivado & Vitis — LED with AXI GPIO

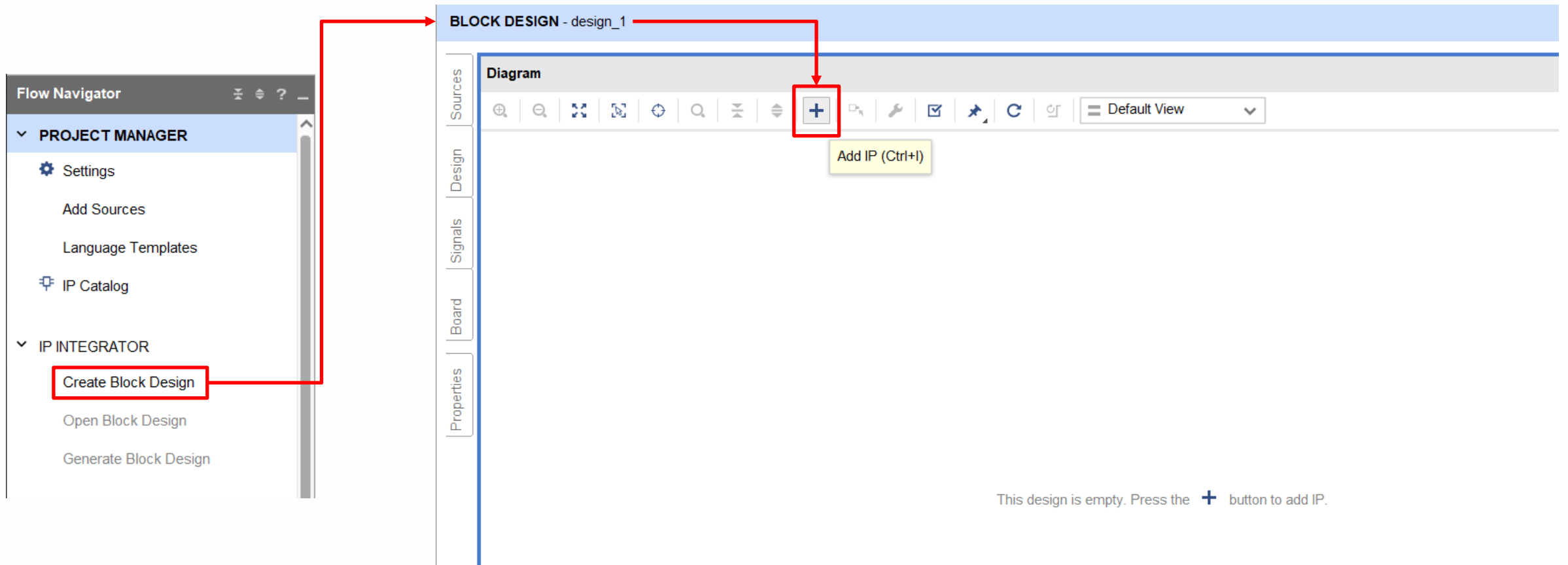
# Agenda

- Vivado — AXI GPIO Block Design
- Vitis — A Sample for Driving AXI GPIO LED

# Vivado — AXI GPIO Block Design

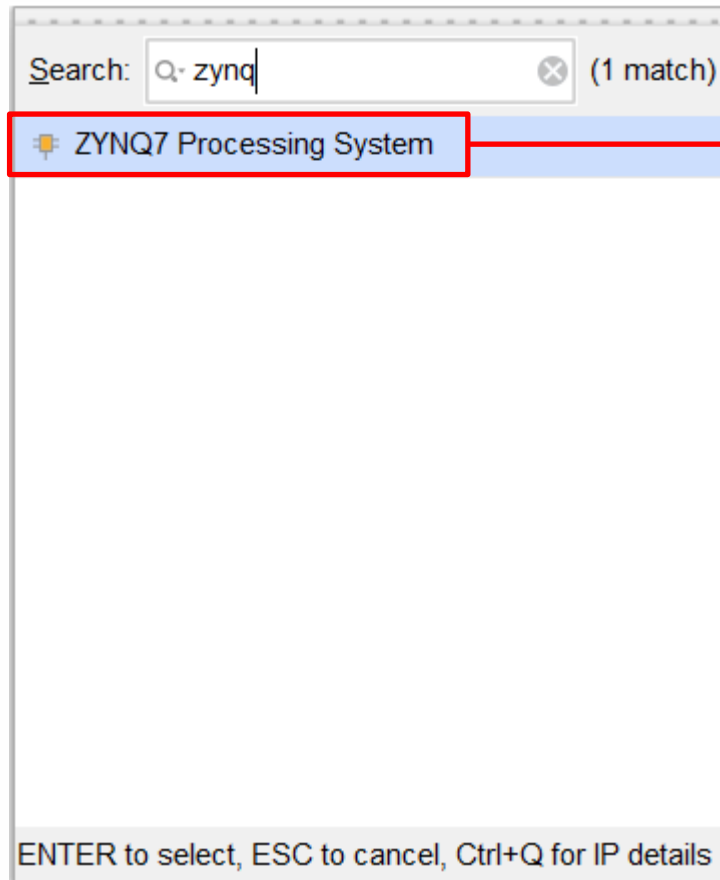
# Block Design

- Step 1: Add Zynq Processing System IP



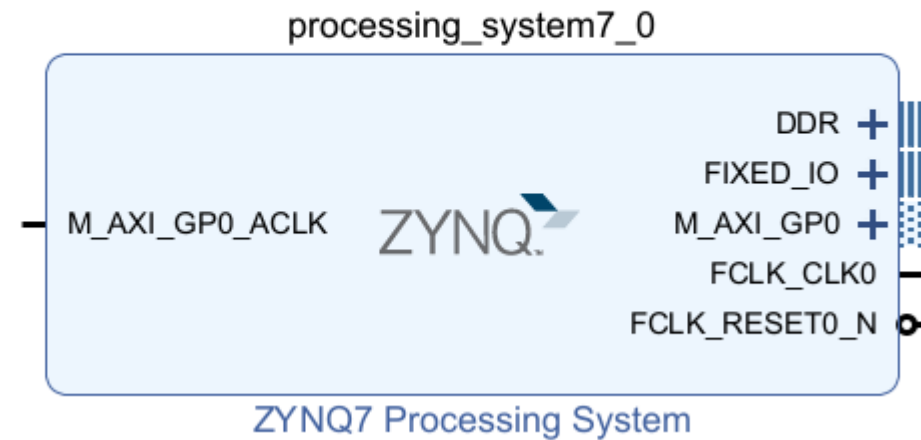
# Block Design

- Step 1: Add Zynq Processing System IP



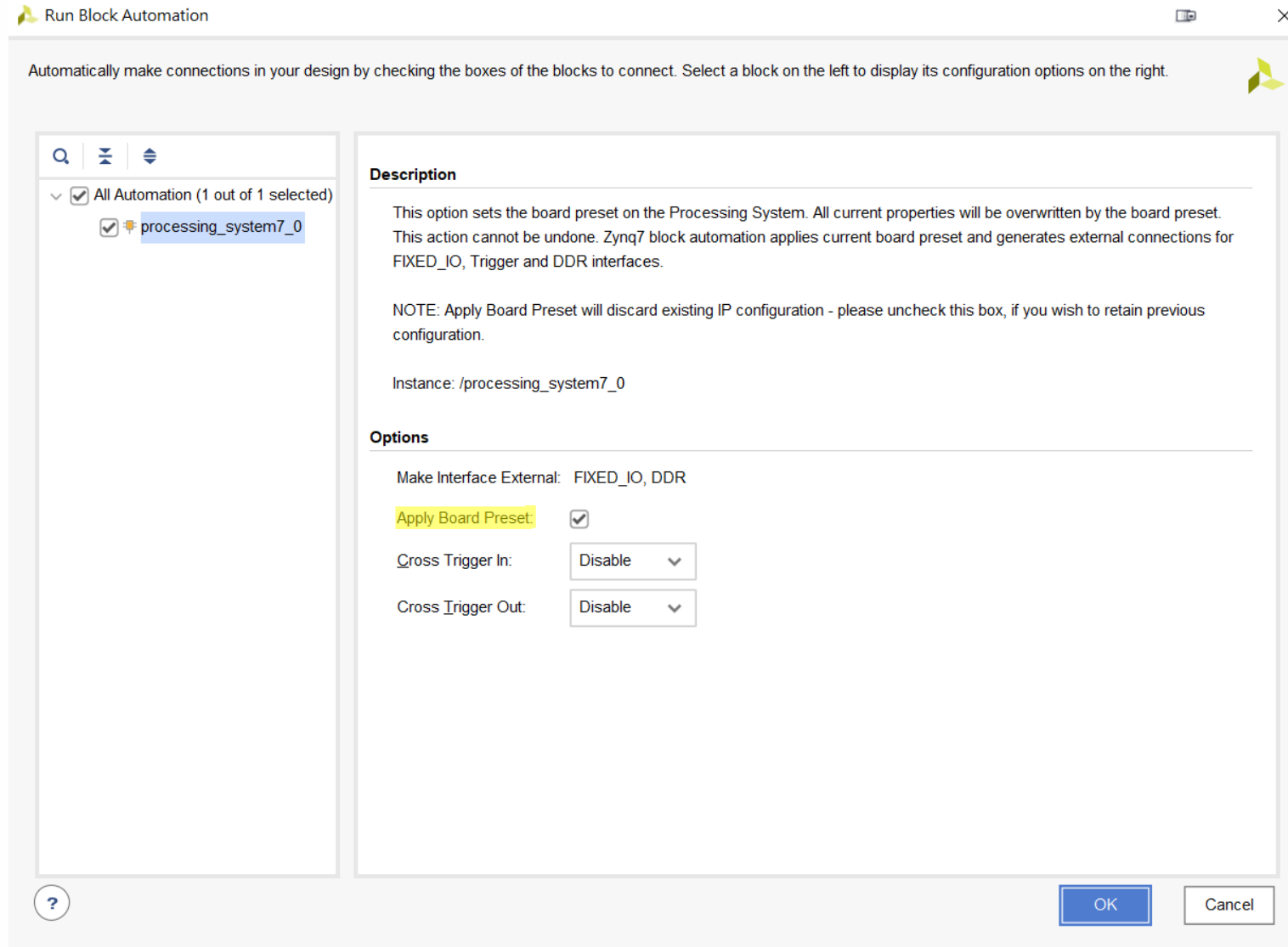
Click here

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



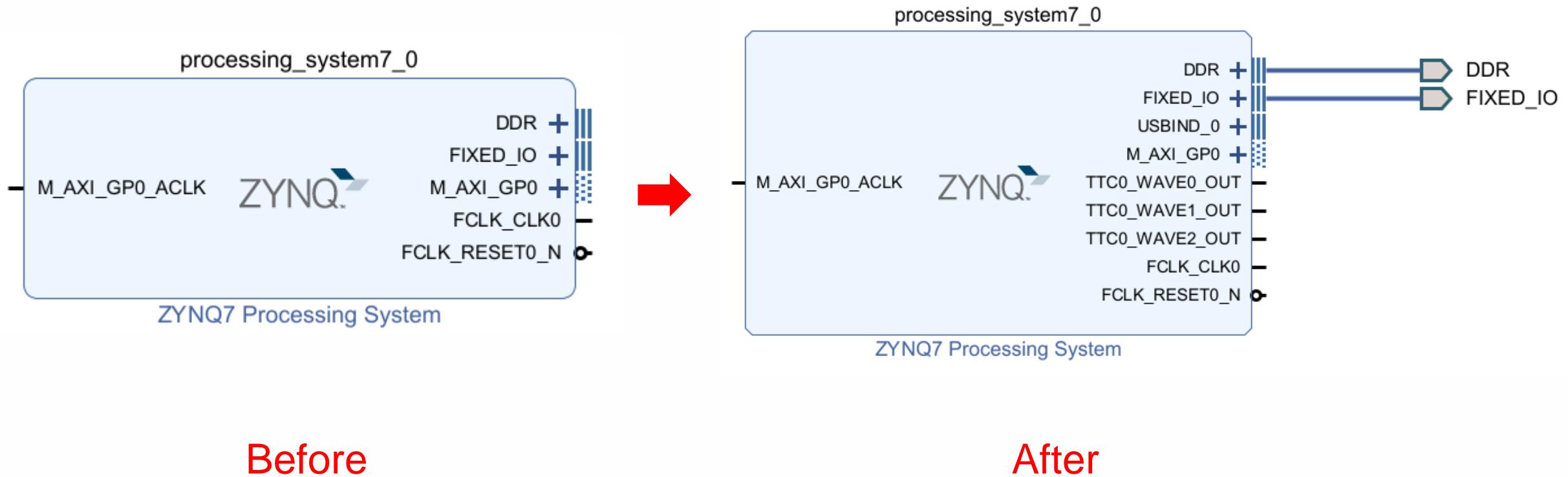
# Block Design

- Step 1: Add Zynq Processing System IP



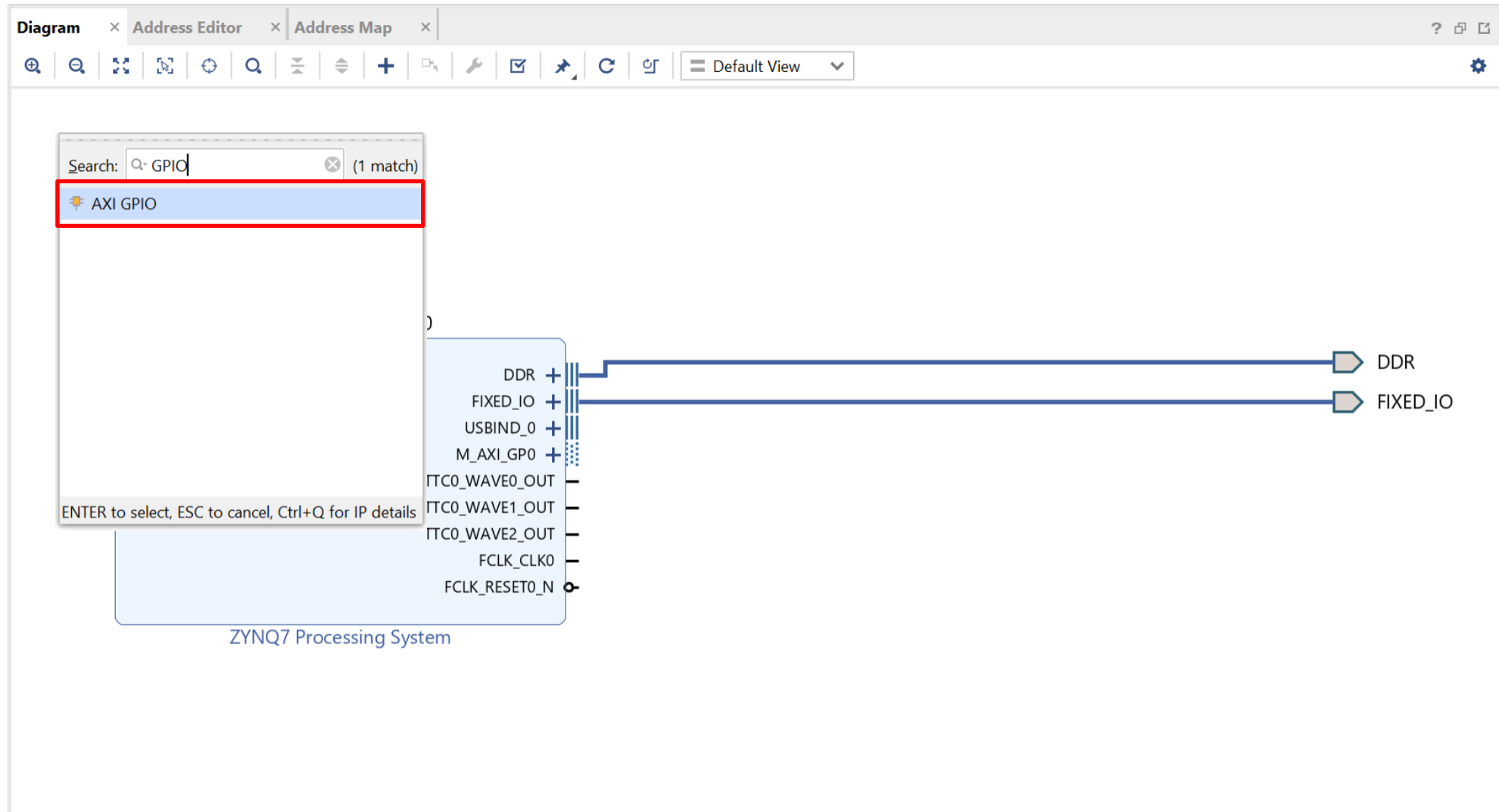
# Block Design

- Step 1: Add Zynq Processing System IP



# Block Design

- Step 2: Add AXI GPIO IP to Block Design

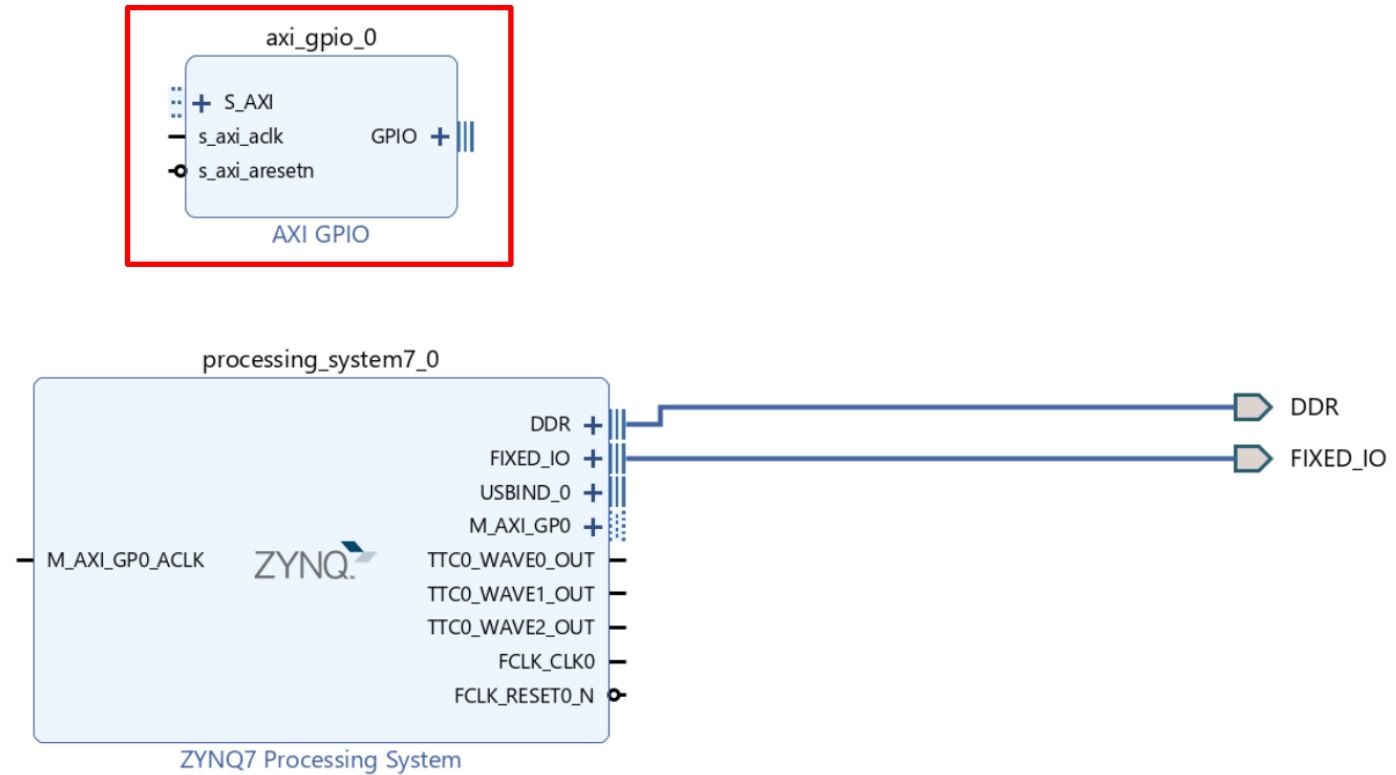




# Block Design

- Step 2: Add AXI GPIO IP to Block Design

✦ Designer Assistance available. [Run Connection Automation](#)



# Block Design

- Step 2: Add AXI GPIO IP to Block Design

☐ Show disabled ports

Component Name

**Board** **IP Configuration**

Associate IP interface with board interface


IP Interface	Board Interface
GPIO	Custom
GPIO2	Custom
	btns 5bits
	leds 8bits
	sws 8bits

**Block Design:** A block labeled "GPIO" with three input ports: `s_axi_adk`, `s_axi_aresetn`, and `S_AXI`. The `S_AXI` port is highlighted with a blue plus icon.

# Block Design

- Step 2: Add AXI GPIO IP to Block Design

☐ Show disabled ports



```

graph LR
    S_AXI[S_AXI] -- s_axi_adk --> GPIO[GPIO]
    S_AXI -- s_axi_aresetn --> GPIO
  
```

Component Name

**Board** | **IP Configuration**

**GPIO**

- ☐ All Inputs
- ☒ All Outputs

GPIO Width  [1 - 32]

Default Output Value  [0x00000000,0xFFFFFFFF]

Default Tri State Value  [0x00000000,0xFFFFFFFF]

☐ Enable Dual Channel

**GPIO 2**

- ☐ All Inputs
- ☐ All Outputs

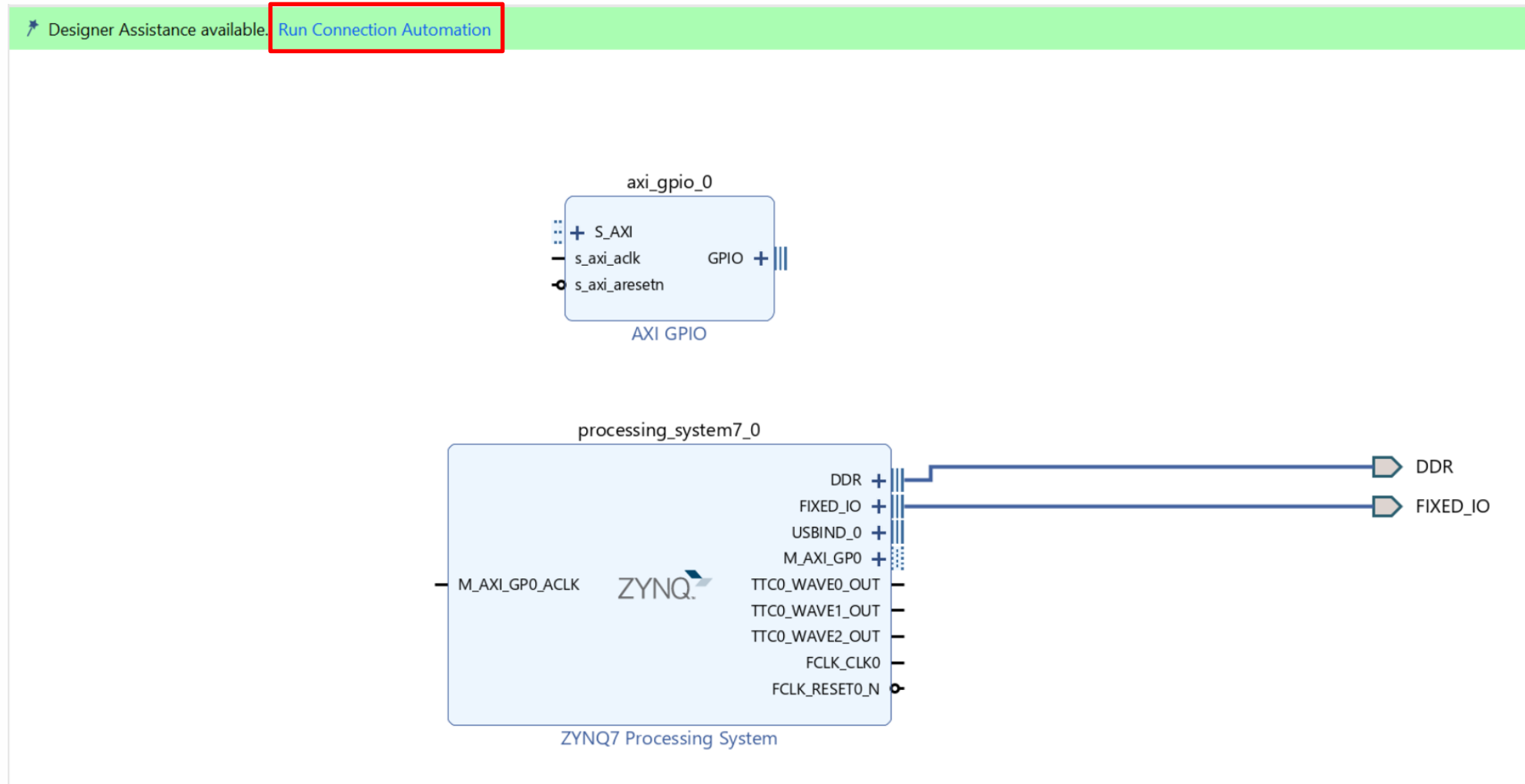
GPIO Width  [1 - 32]

Default Output Value  [0x00000000,0xFFFFFFFF]

☐ Enable Interrupt

# Block Design

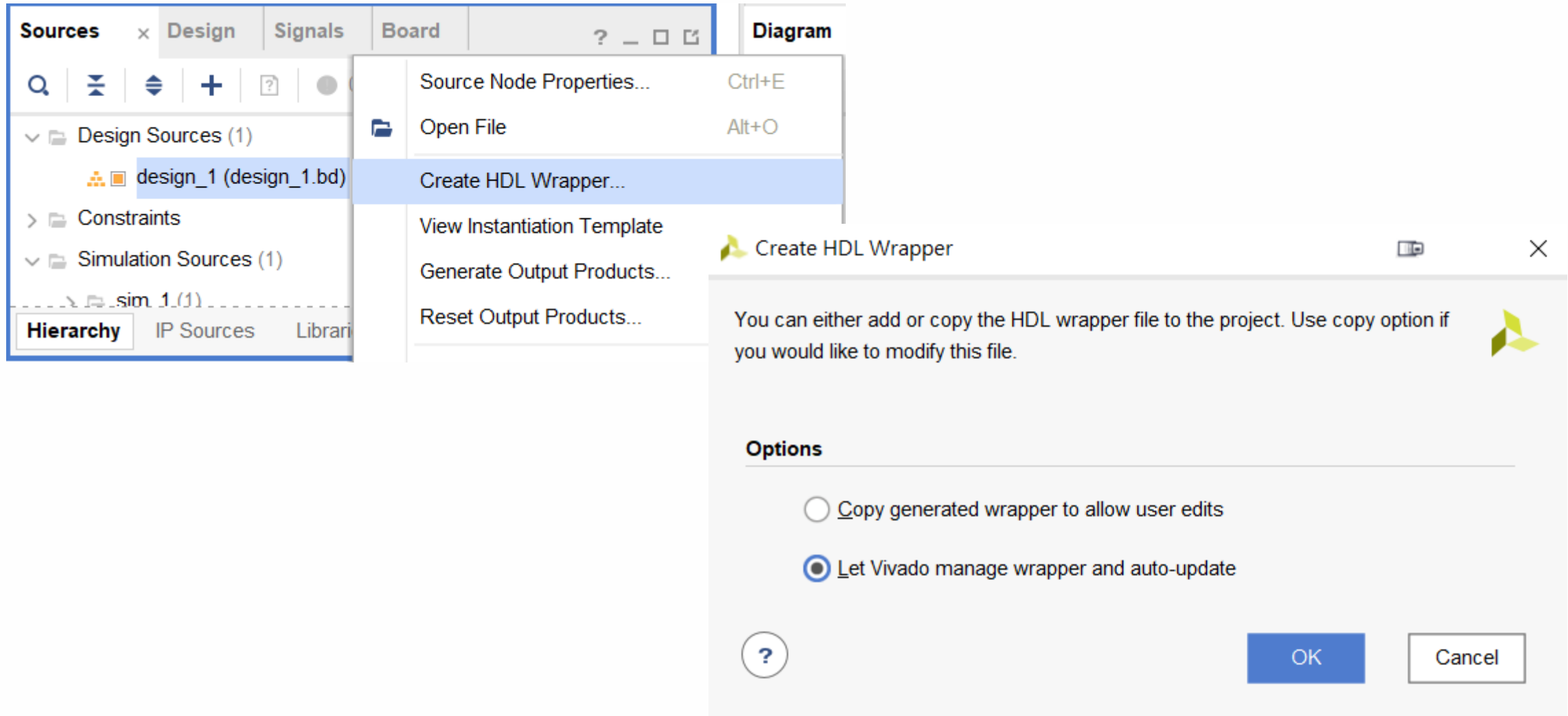
- Step 3: Run Connection Automation





# Block Design

- Step 4: Generate XSA file for Vitis



# Block Design

- Step 4: 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\_wrapper', 'design\_1\_i: design\_1', and 'design\_1'. Below this list, the 'Preview' section shows the 'design\_1.bd' file with options for 'Synthesis', 'Implementation', and 'Simulation'. The 'Synthesis Options' section includes radio buttons for 'Global', 'Out of context per IP' (selected), and 'Out of context per Block Design'. 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 red arrow points from the 'Generate' button to the 'PROGRAM AND DEBUG' sidebar on the right. In this sidebar, the 'Generate Bitstream' option is highlighted in a red box, and a red arrow points from it to the 'Open Hardware Manager' option below it. The 'Sources' window at the top right shows the design hierarchy, including 'design\_1\_wrapper', 'design\_1\_i: design\_1', and 'design\_1'.

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 (OOC 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

- 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 4: 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 'Export' submenu to the 'Export Hardware Platform' dialog on the right.

The 'Export Hardware Platform' dialog is titled 'Export Hardware Platform'. It has a tab labeled 'Output'. The text inside the dialog says: '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.

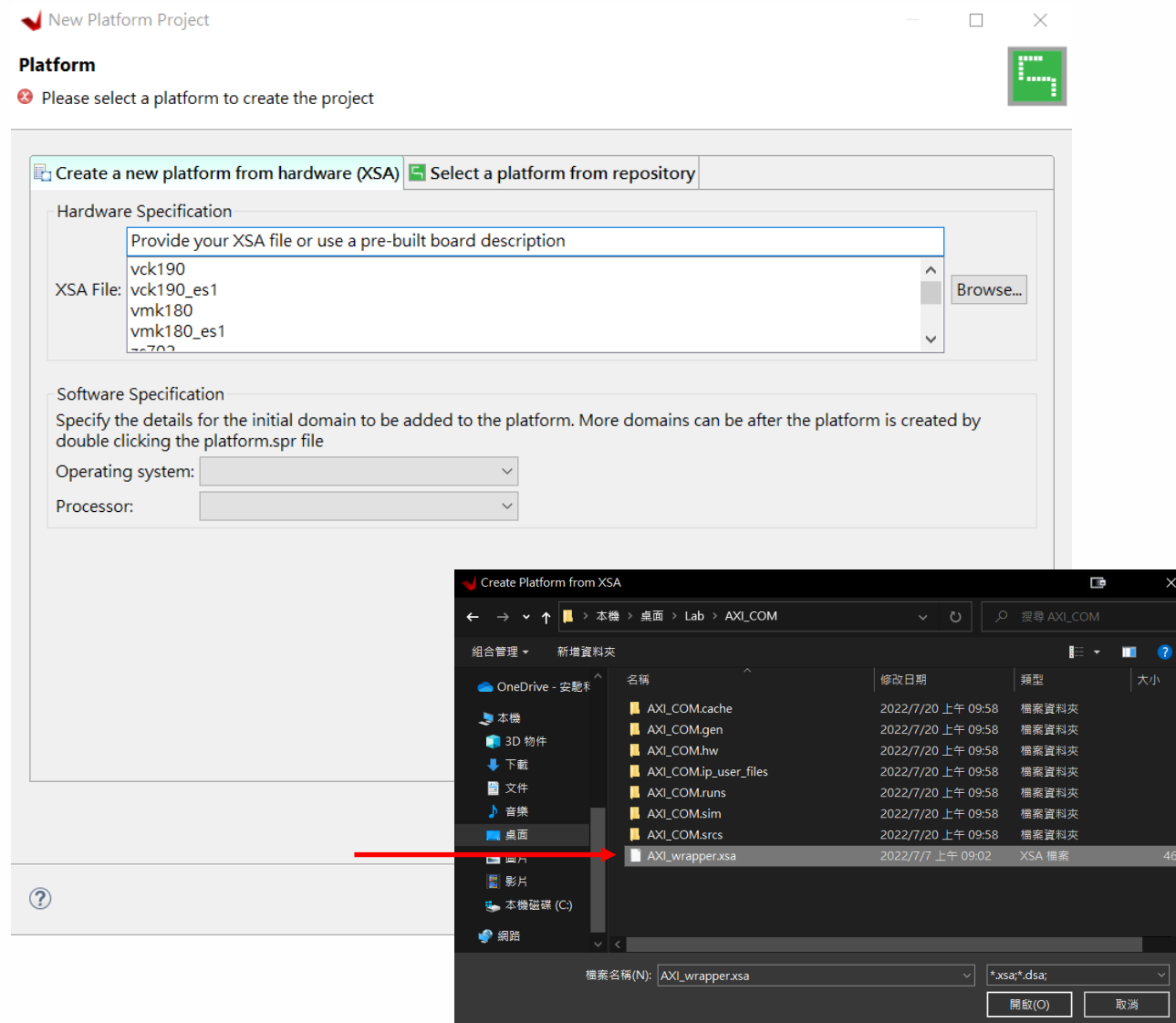
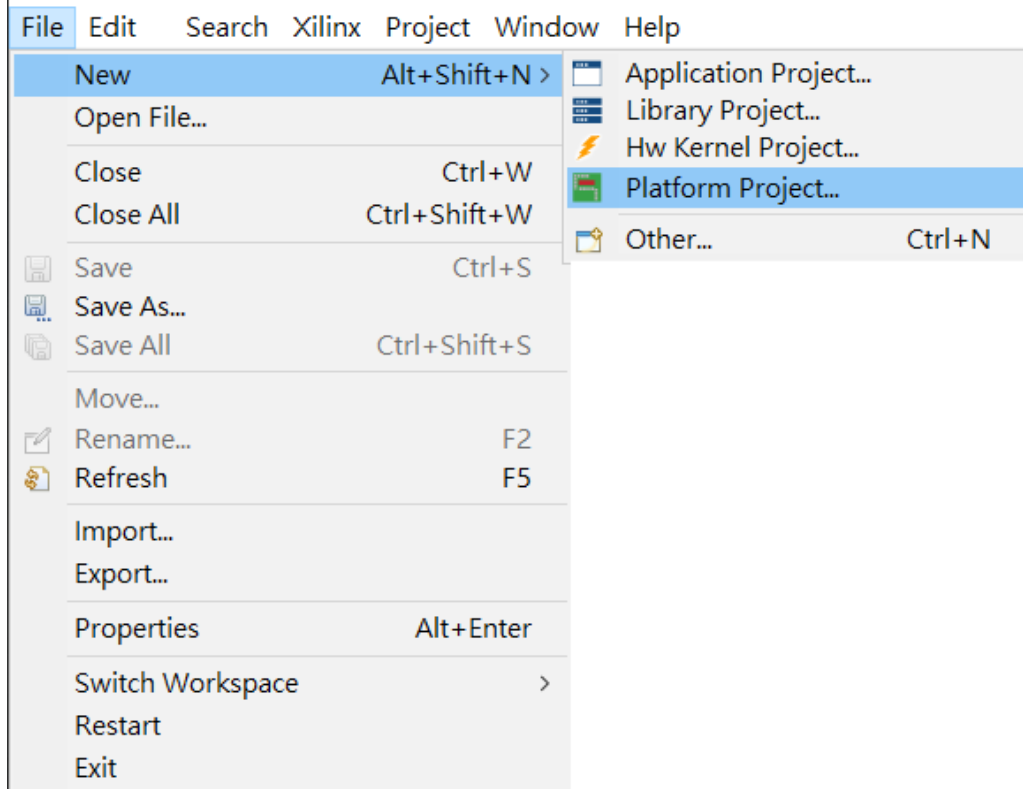
At the bottom of the dialog, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.



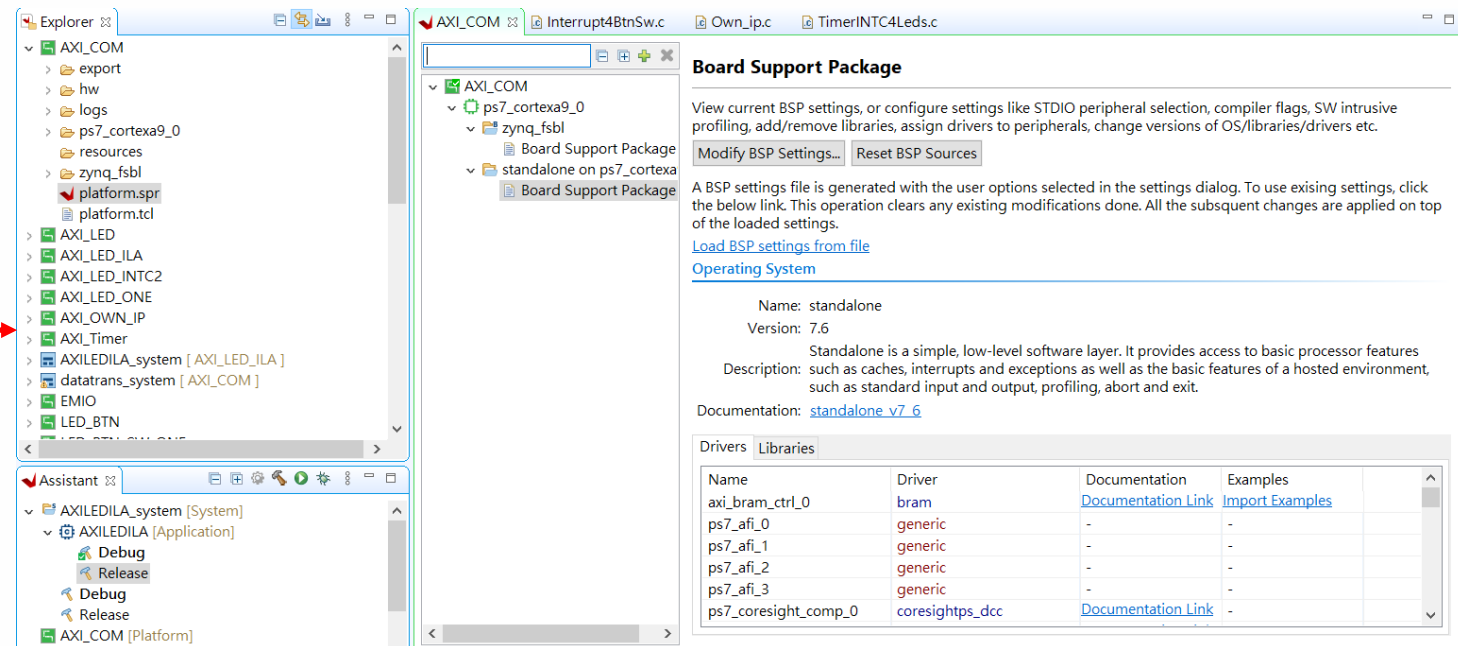
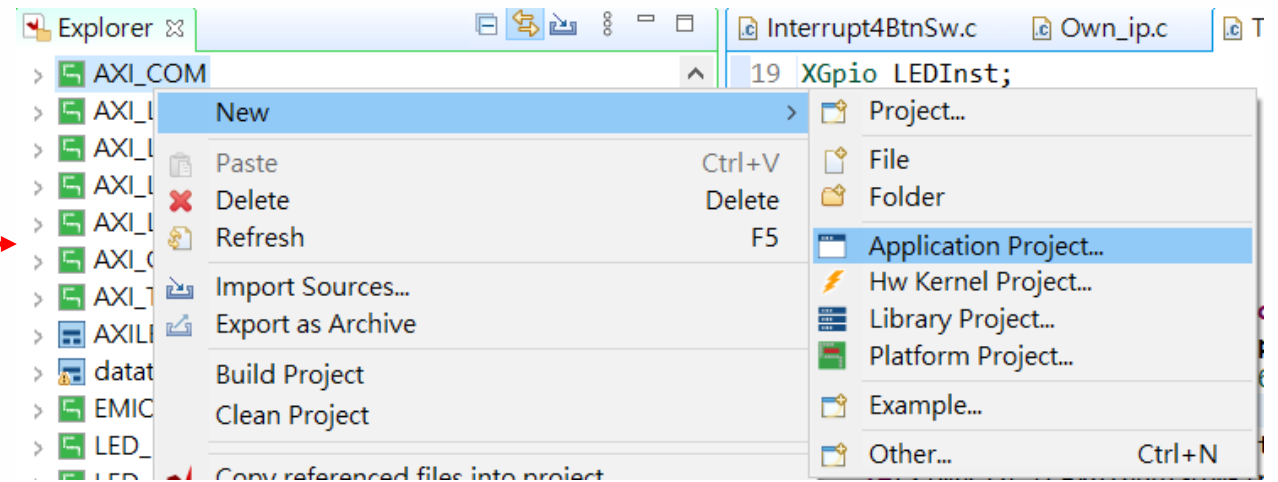
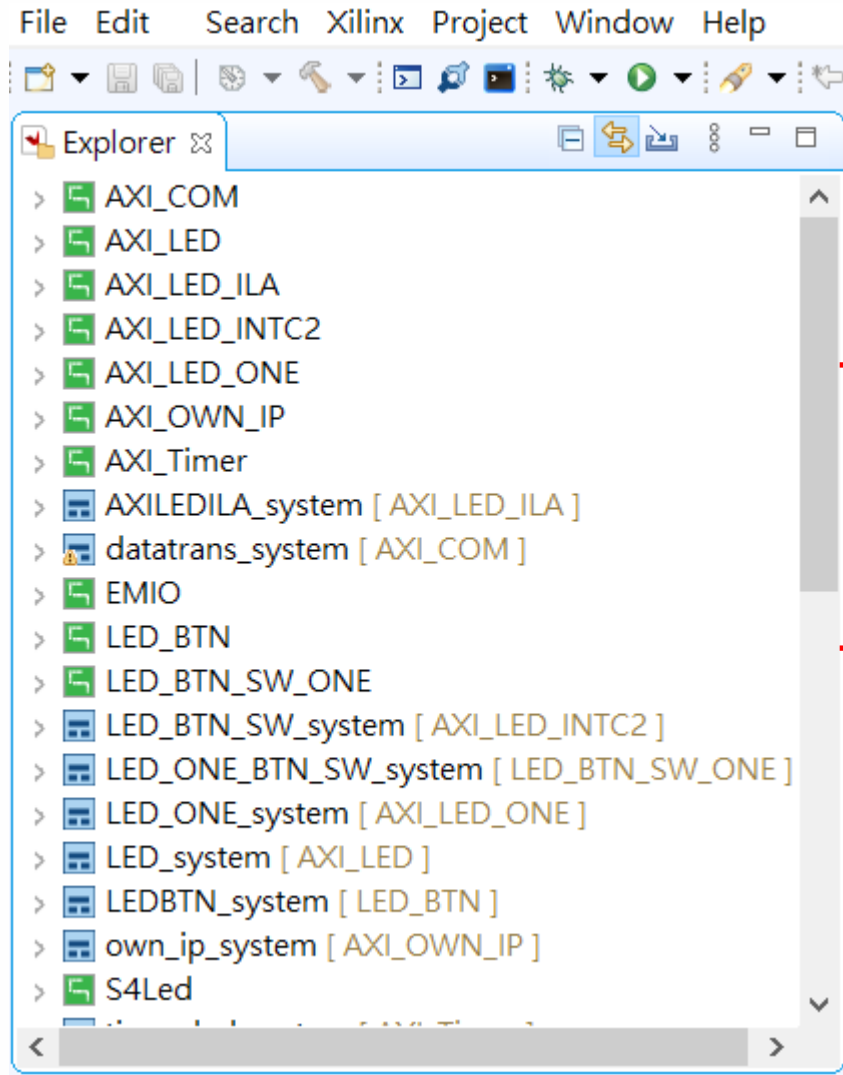
# Vitis — A Sample for Driving AXI GPIO LED

# Create Platform

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

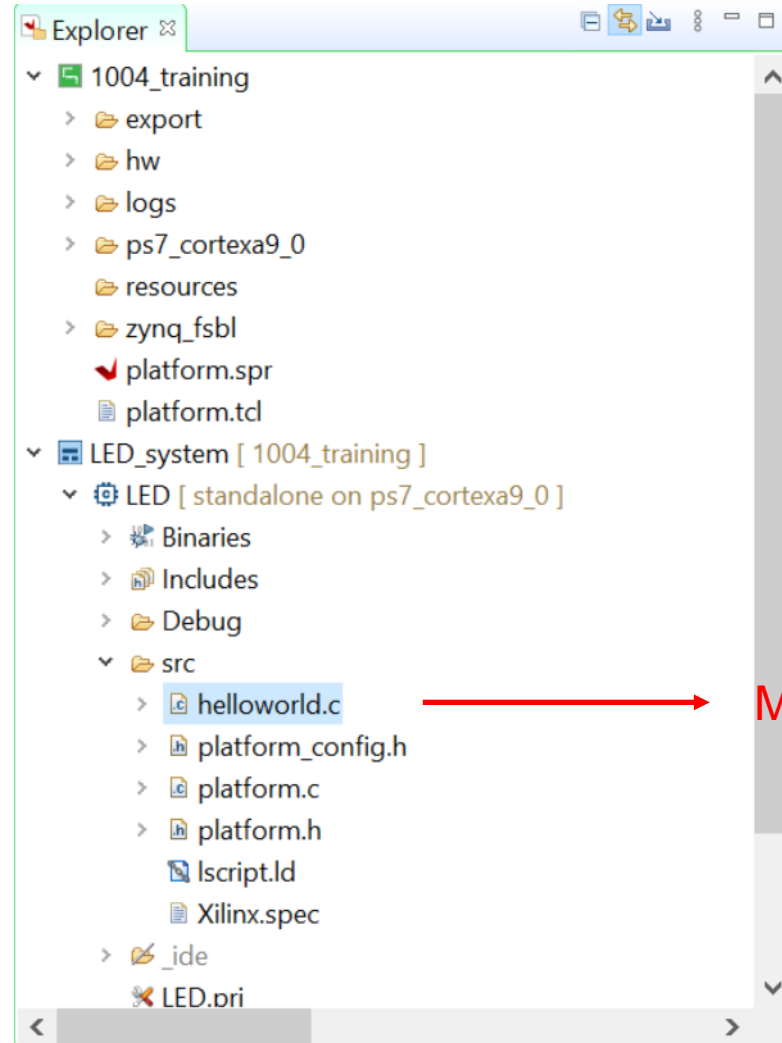


# Create Application



# Example Code

- Light the Leds



Modify this file for your own code

# Example Code

- Light the Leds

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "sleep.h"
#include <math.h>

#define LED_ID XPAR_GPIO_0_DEVICE_ID
#define LED_CHANNEL 1
XGpio Gpio;

int main(void){
    int i=0;

    xil_printf("GPIO LED TEST\n\r");

    XGpio_Initialize(&Gpio, LED_ID); // 初始化 GPIO

    XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0); // 設定 GPIO 為輸入輸出, 0 is output

    while (1) {
        for(i=7;i>-1;--i){ // 八顆 LED 輪流亮滅

            XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, pow(2,i)); // 寫值給 LED

            usleep(100000);

            XGpio_DiscreteClear(&Gpio, LED_CHANNEL, pow(2,i)); // 等於 XGpio_DiscreteWrite(&Gpio,LED_CHANNEL,0x00), 即全清零

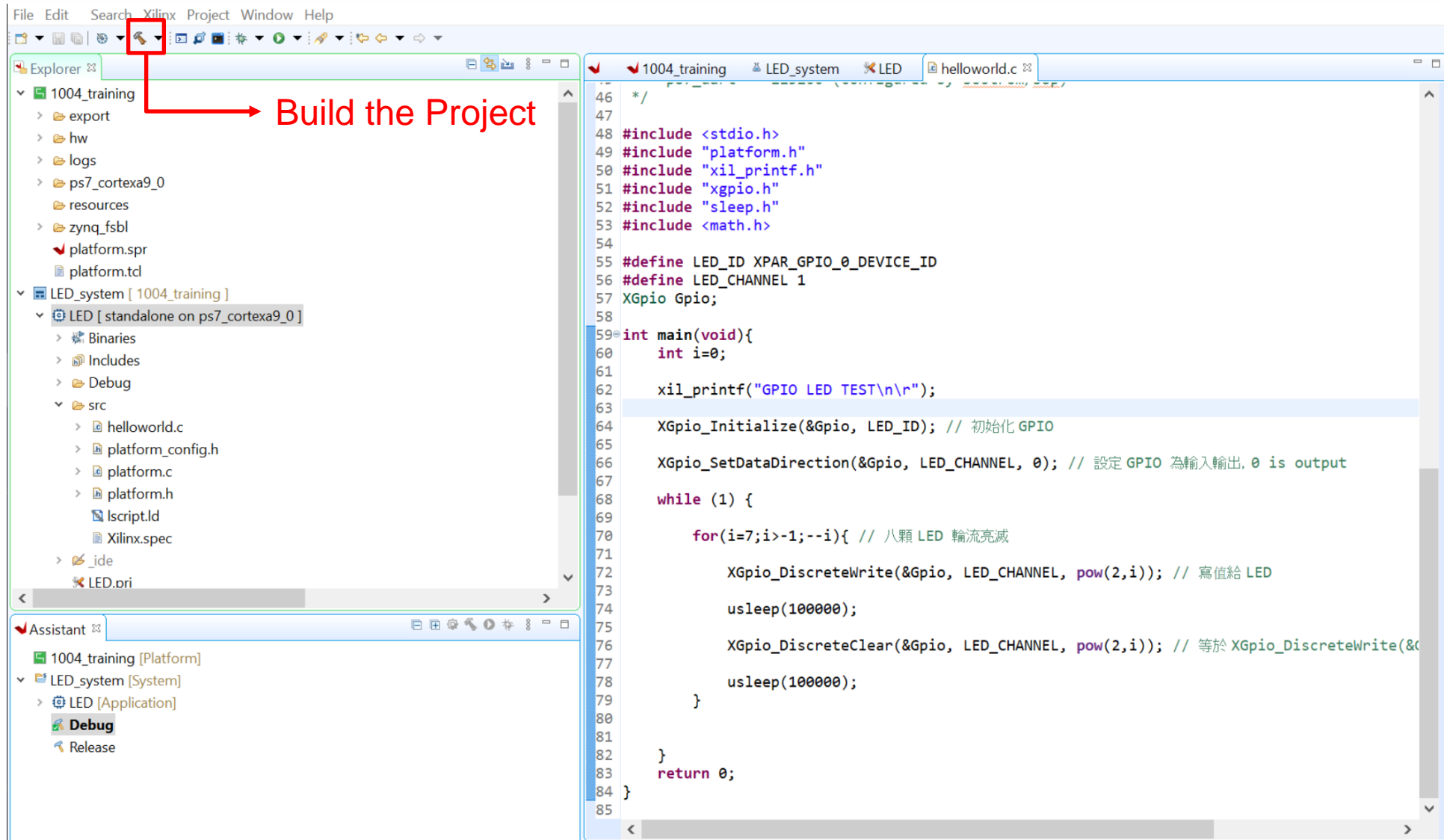
            usleep(100000);
        }

    }

    return 0;
}
```

# Example Code

- Light the Leds



# Vitis Build Problem

- Build Error



The screenshot shows the Vitis IDE interface with the 'Console' tab selected. The console output displays the build process for a project named 'LED'. It starts with a pre-build step and then proceeds to the main build. The linker command is shown, and the error message 'undefined reference to `pow`' is highlighted in red. The build process ends with an error status.

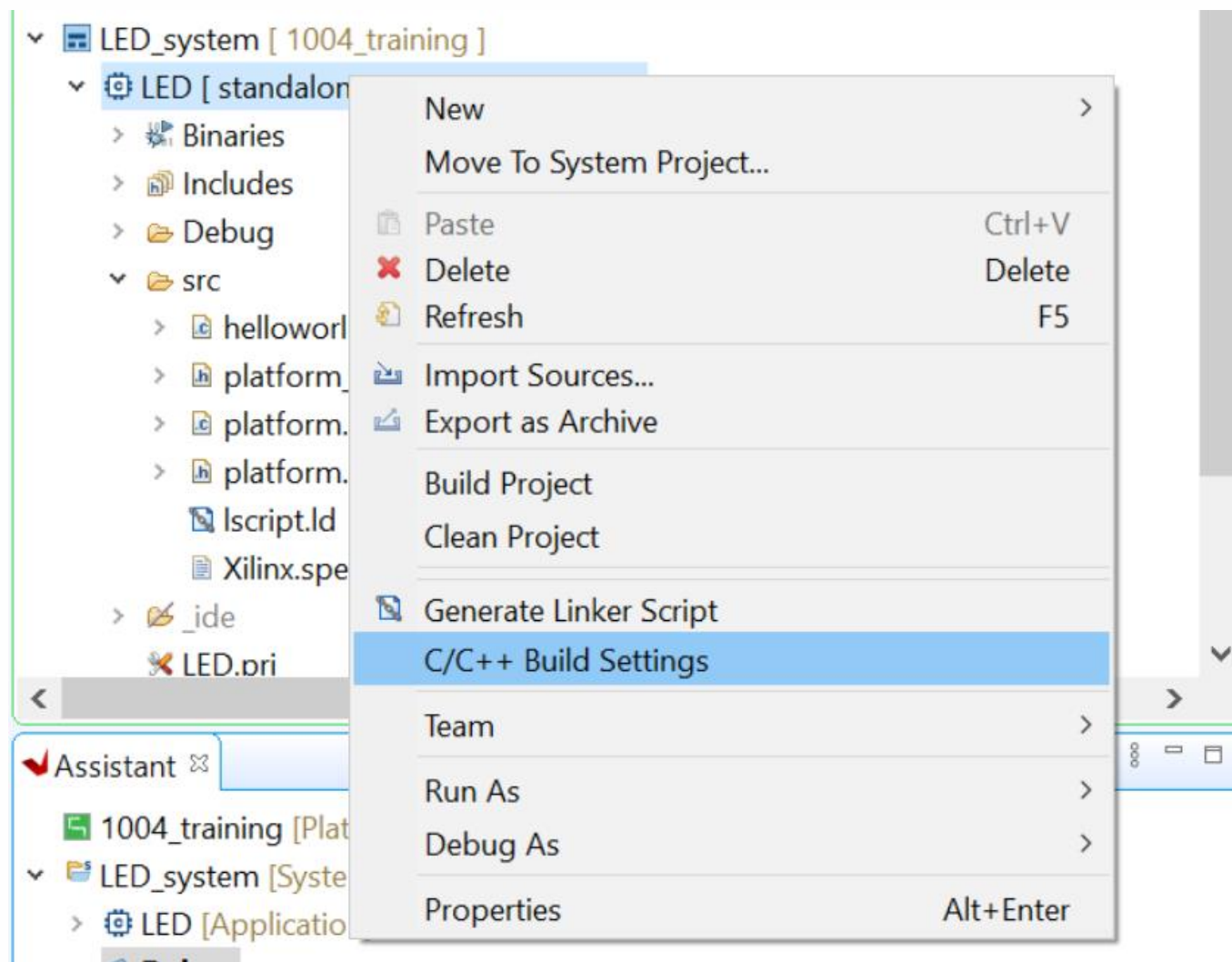
```
Build Console [LED, Debug]
13:52:59 **** Incremental Build of configuration Debug for project LED ****
make all
make --no-print-directory pre-build
a9-linaro-pre-build-step
, ,

make --no-print-directory main-build
'Building target: LED.elf'
'Invoking: ARM v7 gcc linker'
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpv=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript.ld -LC:/Users/User,
c:/xilinx/vitis/2021.1/gnu/aarch32/nt/gcc-arm-none-eabi/x86_64-oesdk-mingw32/usr/bin/arm-xilinx-eabi/../../libexec/arm-xilinx-eabi/gcc/arm-xilinx
C:\Users\User\Desktop\AXI_Vitis_211\LED\Debug\../src/helloworld.c:72: undefined reference to `pow'
c:/xilinx/vitis/2021.1/gnu/aarch32/nt/gcc-arm-none-eabi/x86_64-oesdk-mingw32/usr/bin/arm-xilinx-eabi/../../libexec/arm-xilinx-eabi/gcc/arm-xilinx
collect2.exe: error: ld returned 1 exit status
make[1]: *** [makefile:43: LED.elf] Error 1
make: *** [makefile:34: all] Error 2

13:53:00 Build Finished (took 306ms)
```

# Vitis Build Problem

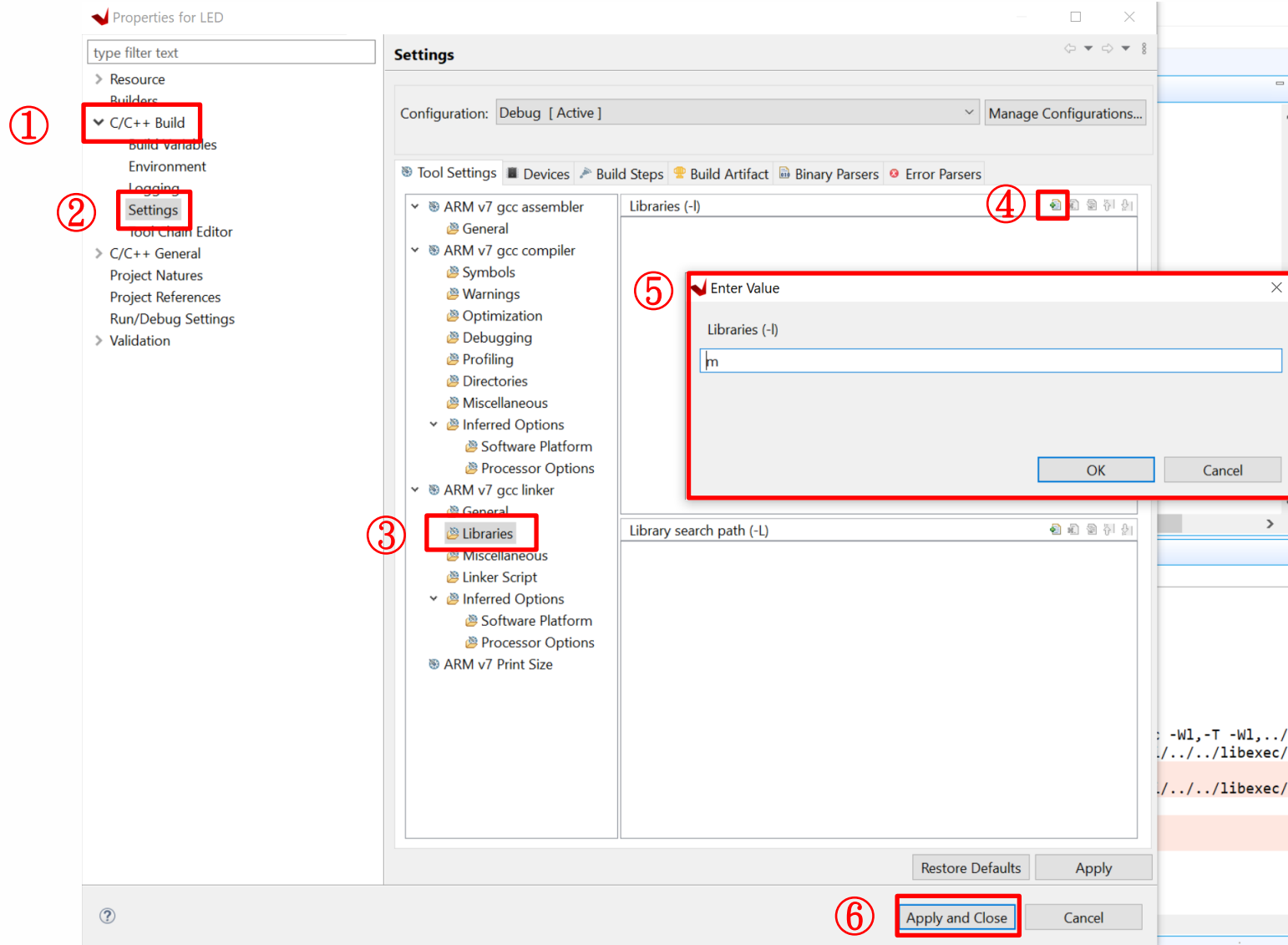
- Build Error - Solution





# Vitis Build Problem

- Build Error - Solution



# Run on Hardware

The screenshot shows the Xilinx IDE interface. On the left, a project tree displays the structure of the 'LED\_system' project, including sub-projects like 'LED' and 'src'. A context menu is open over the 'LED' sub-project, listing various actions such as 'New', 'Move To System Project...', 'Paste', 'Delete', 'Refresh', 'Import Sources...', 'Export as Archive', 'Build Project', 'Clean Project', 'Generate Linker Script', 'C/C++ Build Settings', 'Team', 'Run As', 'Debug As', and 'Properties'. The 'Run As' option is selected, which has opened a sub-menu. This sub-menu contains three options: '1 Launch Hardware (Single Application Debug)', '2 Launch SW Emulator (Single Application Debug)', and '3 Launch Hardware (Single Application Debug (GDB))'. The first option, '1 Launch Hardware (Single Application Debug)', is highlighted. In the background, a code editor shows C code for GPIO initialization and setting data direction.

```
57 XGpio Gpio;

(void){
    i=0;

    printf("GPIO LED TEST\n\r");

    io_initialize(&Gpio, LED_ID); // 初始化 GPIO

    io_setDataDirection(&Gpio, LED_CHANNEL, 0); // 設
```

# Results

