

HELLO, FPGA

Document my FPGA learning journey

APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL VERSION) Project XV ZYNQ side PS accesses the reg register on the PL side to realize PS and PL data interaction

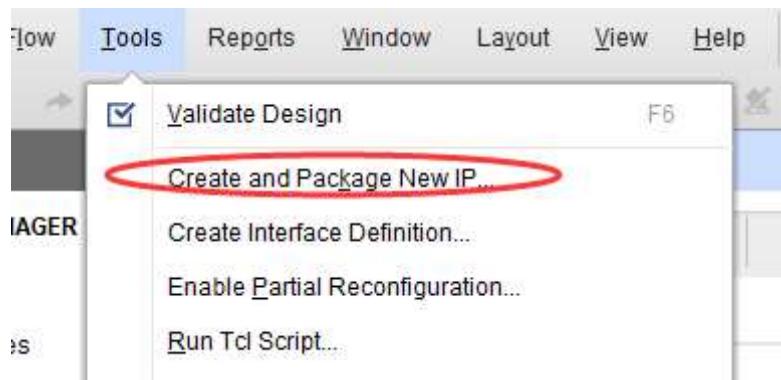
The PS side and the PL side are independent of each other in hardware, and the GPIO port of the ZYNQ PS side can be mapped to the PL side through EMIO and AXI GPIO, but only the GPIO can be controlled. This article introduces a new approach to simple data interaction (small data volume) between ARM and FPGA by giving the PS side access to the PL-side registers

This article is demonstrated on vivado2018.3, please research for other versions

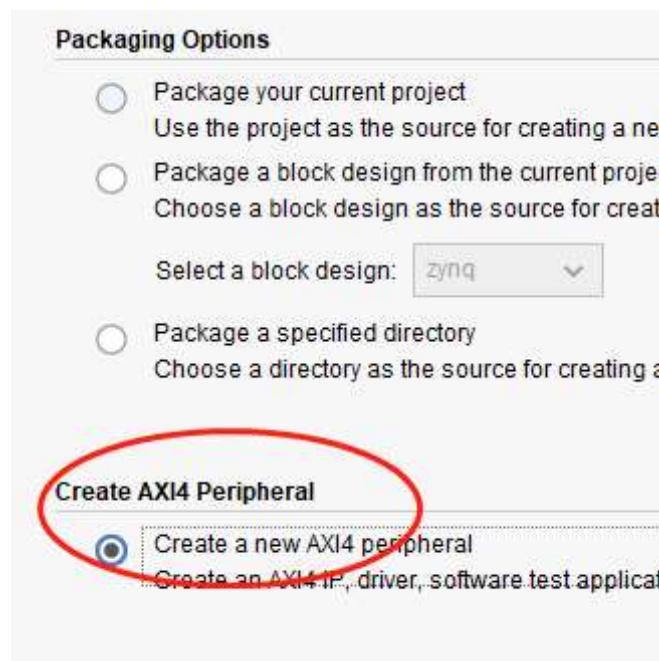
(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory.)

A Create an IP with an AXI interface

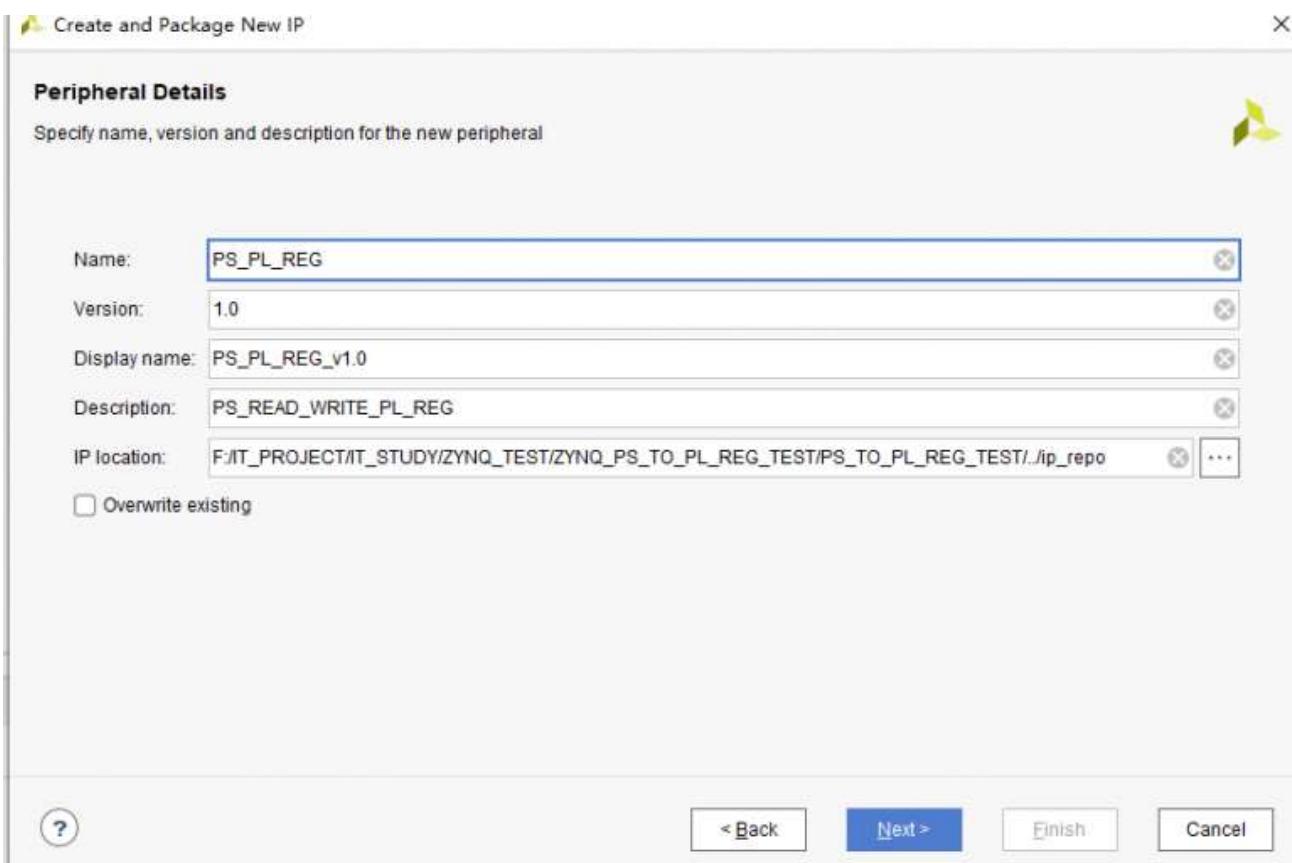
1. The process to recreate and encapsulate an IP with an AXI interface is as follows, TOOLS->Create and Package New IP



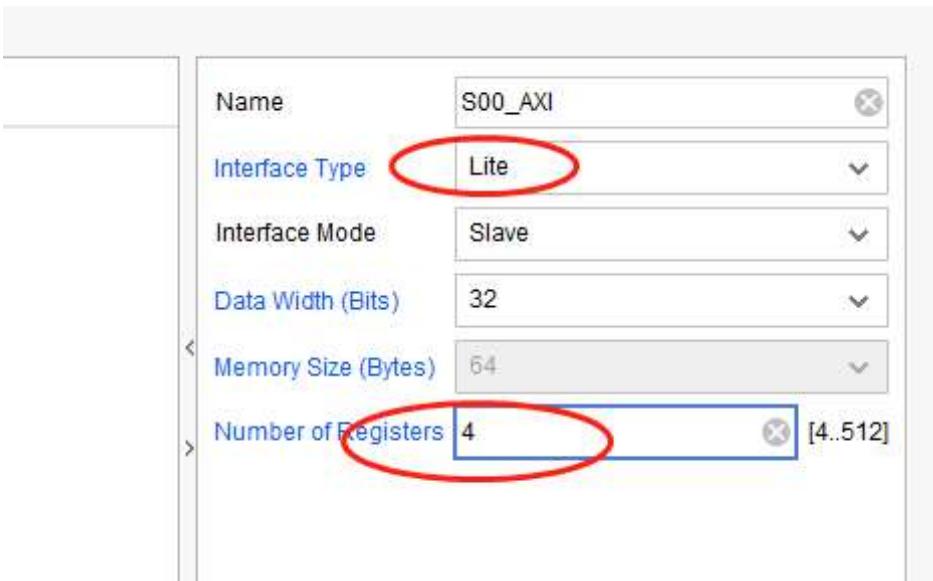
2. Select the package with AXI4 bus



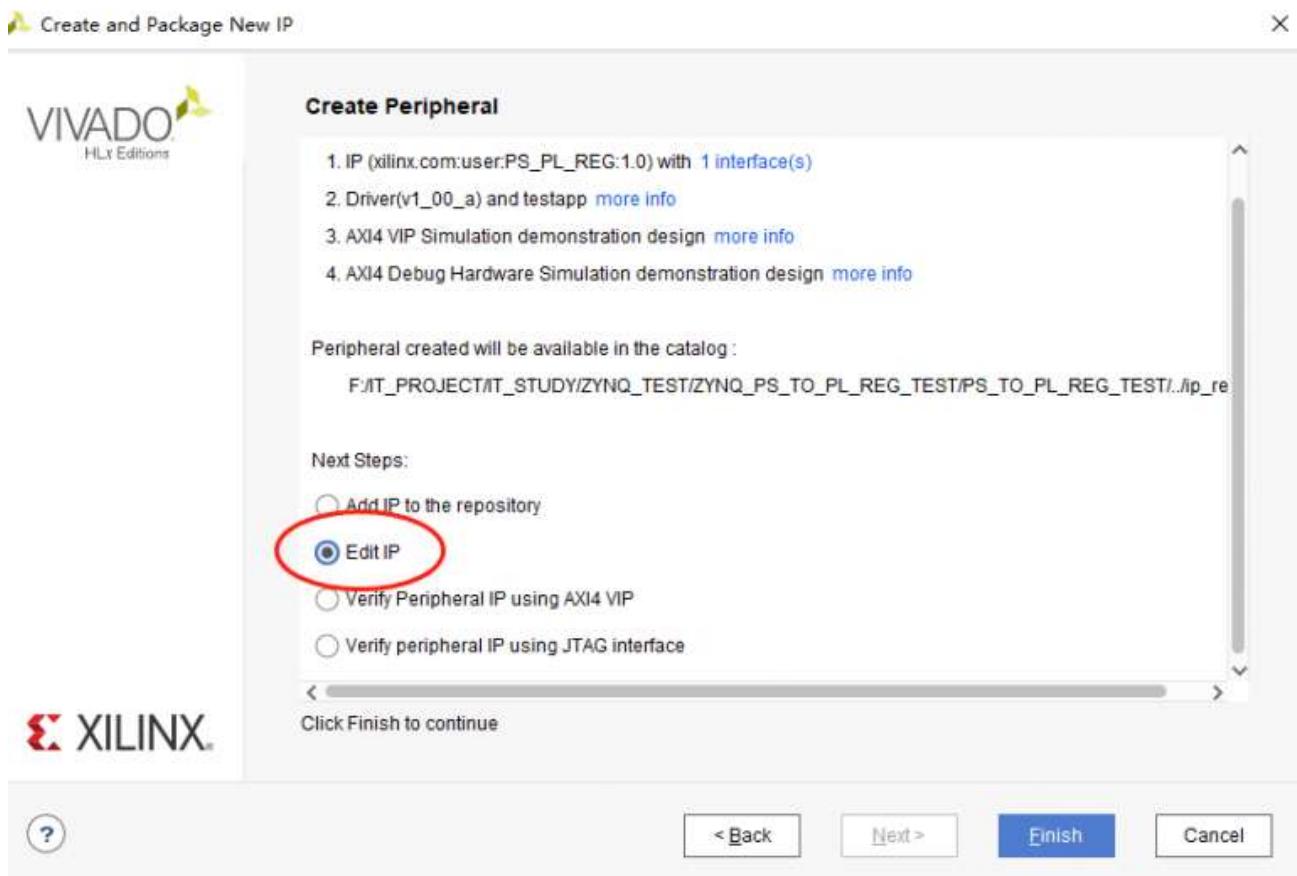
3. Next, fill in the name and other information, pay attention to the IP saving path (save to the project directory). Here the IP name PS_PL_REG i.e. PS and PL REG access each other



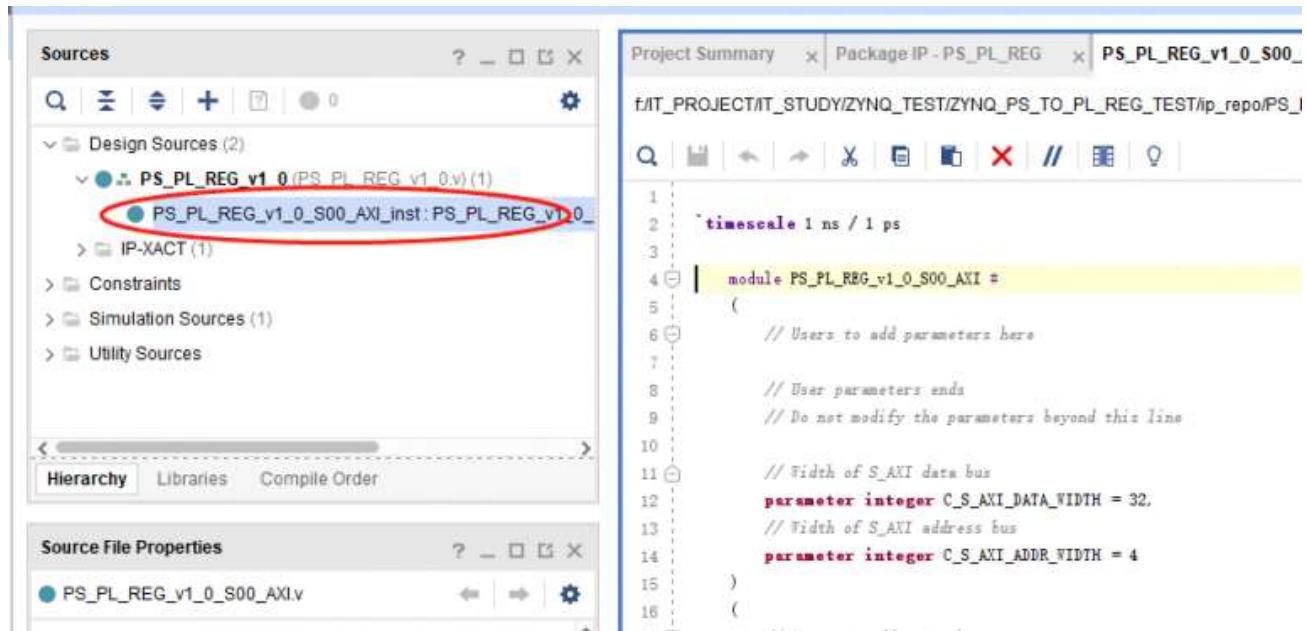
4, next, select bus-related information, here AXI's protocol selects LITE light, and then number of registers according to the needs to choose, 4 means there are four 32-bit registers The minimum selection is 4



5. Select Next EDIT IP and click FINISH to complete (be sure to select "Edit IP" here, otherwise, the software will only generate the framework file, and will not create a project about the IP core)



6. After that, the system will automatically generate a project of the IP. Here we double-click to open the IP code generated by default by the system



7. Study the code, you can see that when we set the IP, the 4 registers for PS access on the PL side are slv_reg0, slv_reg1, slv_reg2, slv_reg3, and these 4 registers can be read and written by the ARM PS side through the AXI interface.

```

//-- Number of Slave Registers 4
reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg0;
reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg1;
reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg2;
reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg3;

```

8. In order to make the interaction process more intuitive, we add an LED indicator to the original IP code to observe the result of the interaction

output reg LED;

```

    // accept the read data and response
    input wire S_AXI_RREADY,
    output reg LED//FOR TEST
);

```

并在代码中增加以下代码用来将灯和数据进行对应

```

always@(posedge S_AXI_ACLK)begin
    if(slv_reg0==32'd0)LED<=0;
    else if(slv_reg0<=32'd1)LED<=1;
end

```

代码解释，当slv_reg0等于32位的0时 LED 熄灭，当slv_reg0等于32'd1时，即十进制1时，LED点亮 (slv_reg0后面可由ARM端的程序直接读写)

9.对系统生成的IP的上层模块，也相应的增加LED 接口，并与内部的模块相连接

```

67     .S_AXI_ARVALID(s00_axi_arvalid),
68     .S_AXI_ARREADY(s00_axi_arready),
69     .S_AXI_RDATA(s00_axi_rdata),
70     .S_AXI_RRESP(s00_axi_rresp),
71     .S_AXI_RVALID(s00_axi_rvalid),
72     .S_AXI_RREADY(s00_axi_rready),
73     .LED(LED)
74 );
75

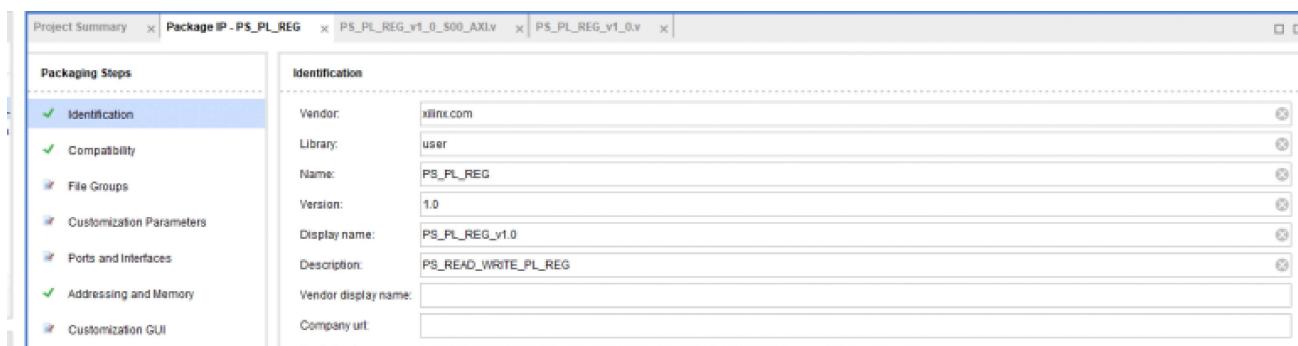
```

```

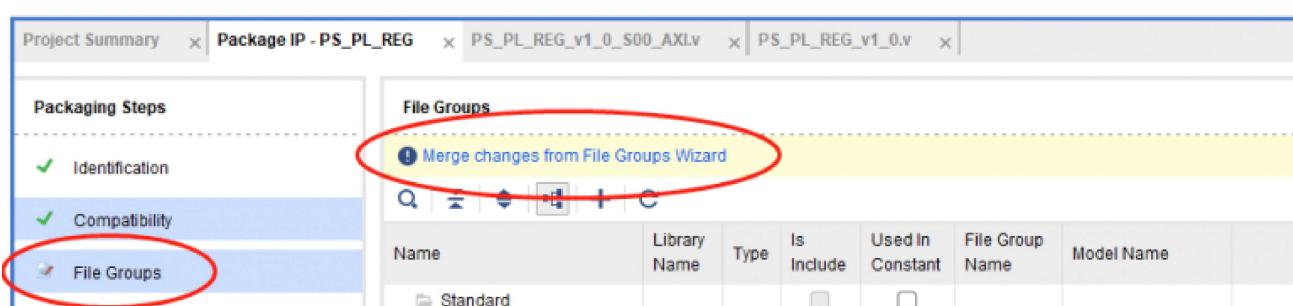
40     output wire  s00_axi_arready,
41     output wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_rdata,
42     output wire [1 : 0] s00_axi_rresp,
43     output wire  s00_axi_rvalid,
44     input wire   s00_axi_rready,
45     output wire LED//FOR TEST
46   );
47 // Instantiation of Axi Bus Interface S00_AXI
48 PS_PL_REG v1_0 S00_AXI #(

```

10. After editing, save the modifications, and then package the modified IP (before encapsulation, compile the synthesis to see if the code reports an error), return to the Package IP page to set the IP, enter the relevant information as shown below, or do not modify



Click Merge changes from File Groups Wizard in the file Groups item



According to the above operation, all the options that are not ticked in the software interface are also merged

The screenshot shows the 'Packaging Steps' on the left and the 'Customization Parameters' on the right.

Packaging Steps:

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✗ Customization Parameters (highlighted with a red oval)
- ✗ Ports and Interfaces (highlighted with a red oval)
- ✓ Addressing and Memory
- ✗ Customization GUI (highlighted with a red oval)
- ✗ Review and Package (highlighted with a red oval)

Customization Parameters:

Name	Description
Customization Parameters	
C_S00_AXI_DATA_WIDTH	Width of S_AXI data bus
C_S00_AXI_ADDR_WIDTH	Width of S_AXI address bus
C_S00_AXI_BASEADDR	
C_S00_AXI_HIGHADDR	

A red arrow points from the 'Merge changes from Customization Parameters Wizard' button in the customization parameters interface to the 'Review and Package' step in the packaging steps list.

When all but the last item is checked and updated, click Repackage Encapsulation IP in the last column to complete the IP packaging

The screenshot shows the 'Review and Package' step in the IP Integrator.

Packaging Steps:

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- ✗ Review and Package (highlighted with a red oval)

Review and Package:

IP has been modified.

Summary:

- Display name: PS_PL_REG_v1.0
- Description: My new AXI IP
- Root directory: e:/3_EBAZ4205/ZYNQ_PROJECT/EBAZ4205/PS_TO_PL_REG/ip_repo/PS_PL_REG_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](#)

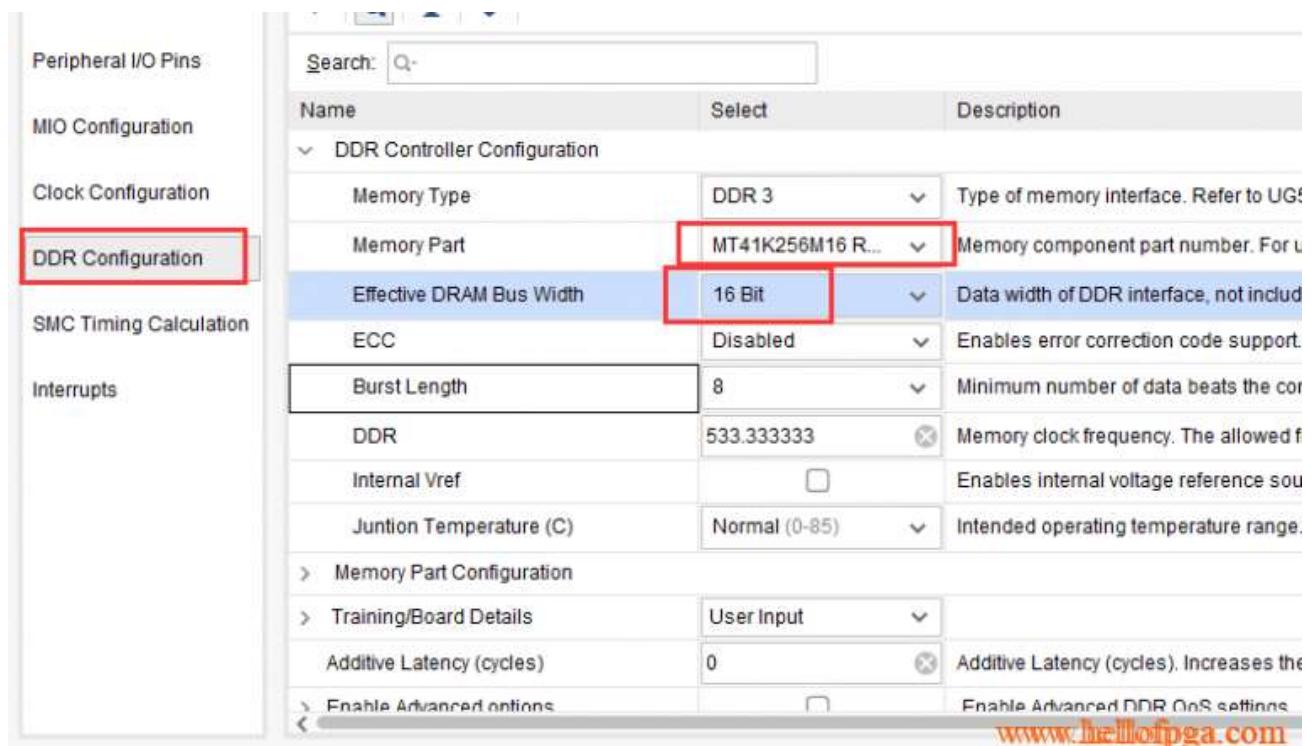
A red oval highlights the 'Re-Package IP' button at the bottom right of the interface.

After the IP is packaged, the project that modifies the IP will be automatically closed by the system

2. Add ZYNQ module to the vivado project

1. Add ZYNQ module

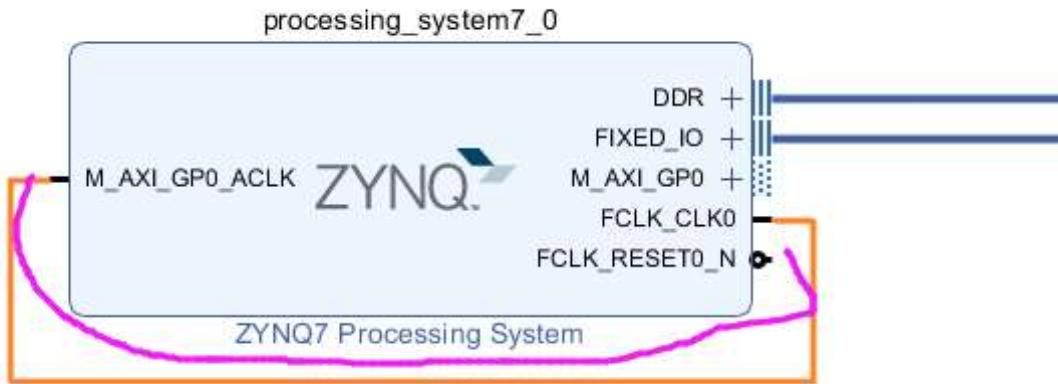
- 1) Add the ZYNQ module in Block Design, and set the DDR option to MT41K256M16RE-125 bit wide in the setting interface to select 16bit



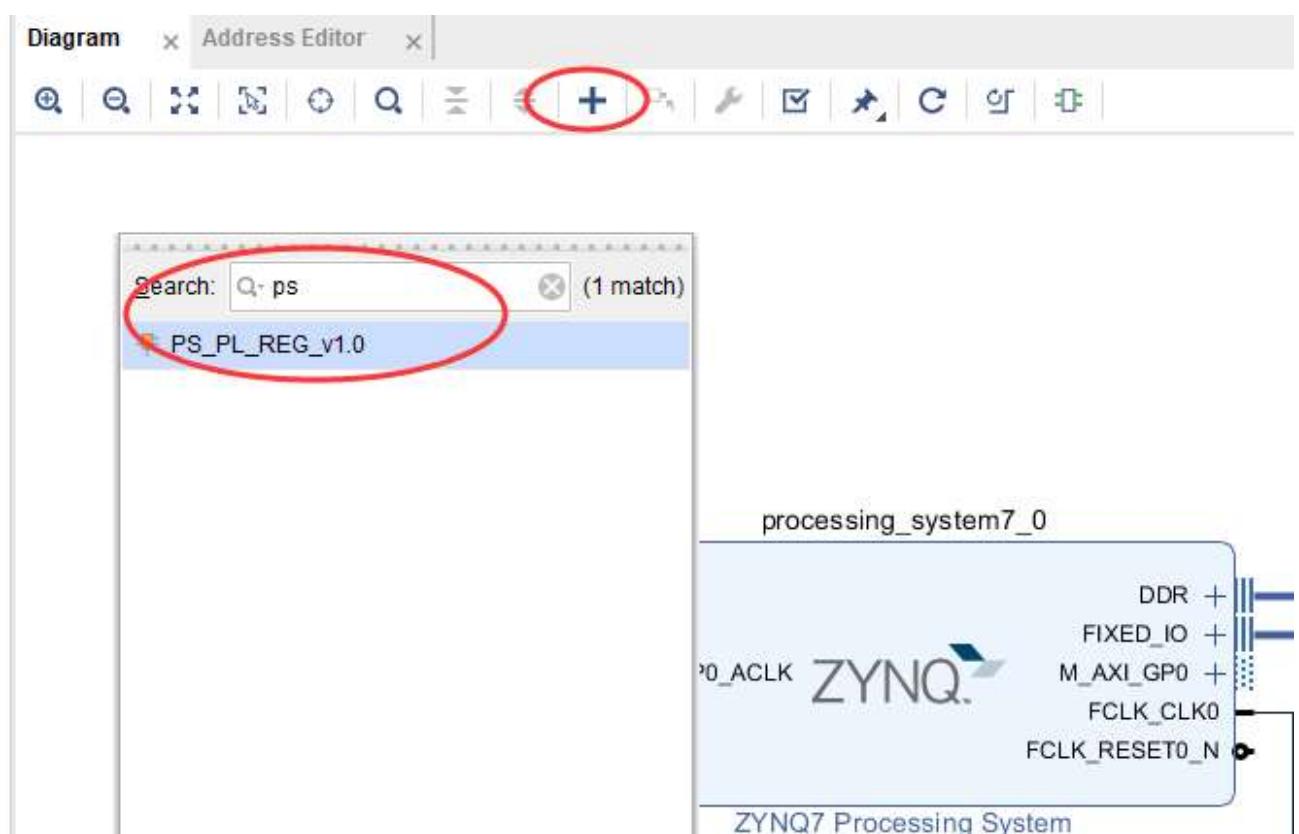
- 2) Select Run Block Automation to complete the port autorouting



- 3) Manually connect the FCLK_CLK0 to the M_AXI_GP0_ACLK

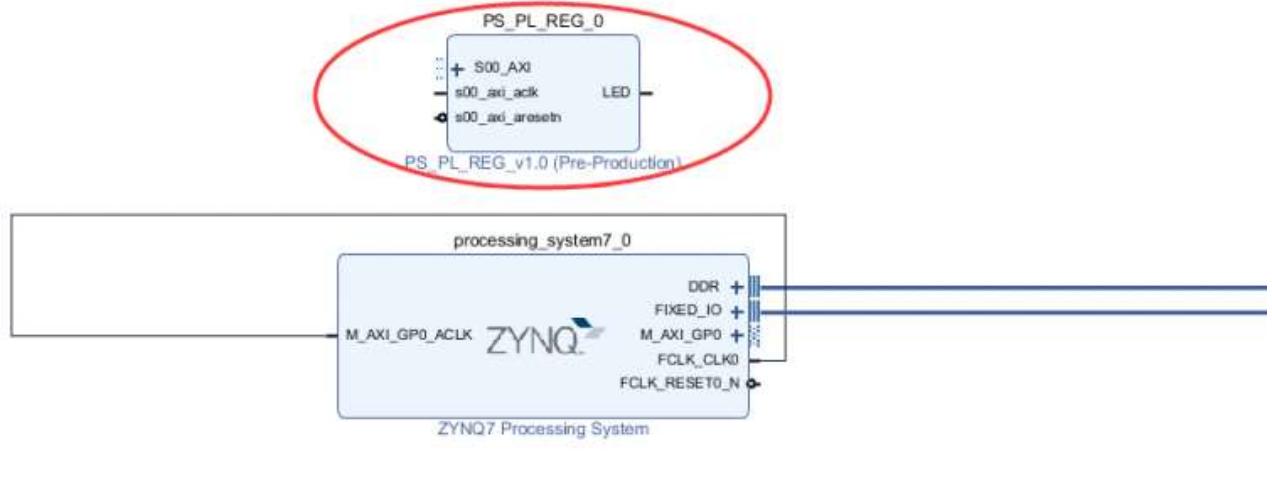


4) Add the PS_TO_PL_REG IP module we just created in the block design

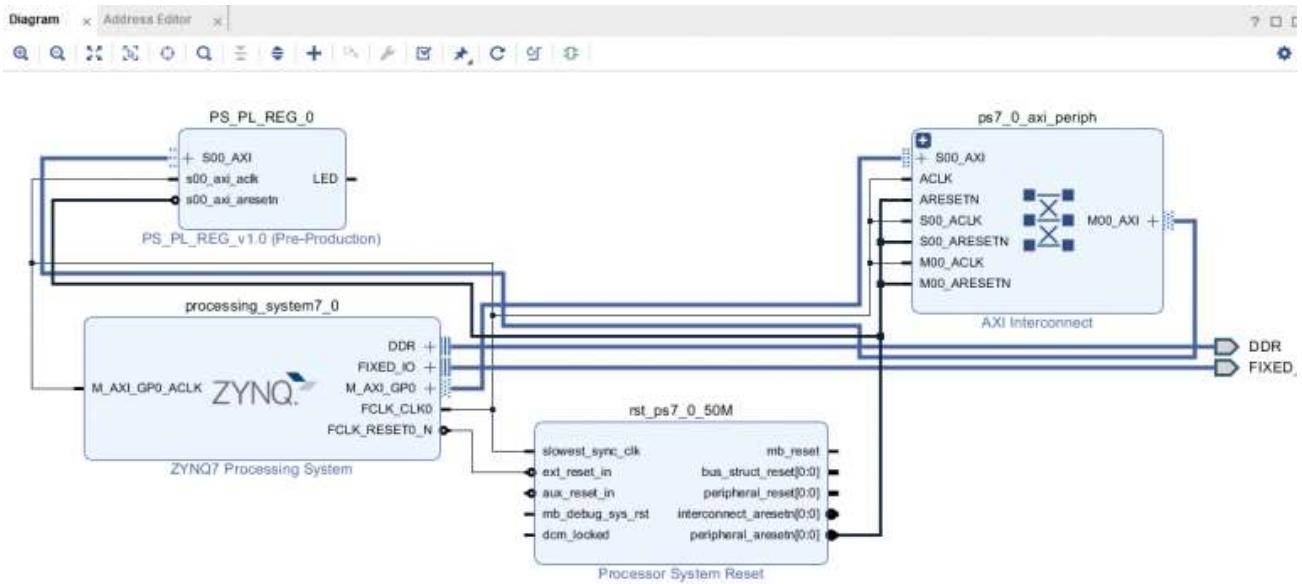


5) As shown in the figure below The PS_PL_REG module we added has been added to the design, click Run Connection Automation again to try the automatic connection

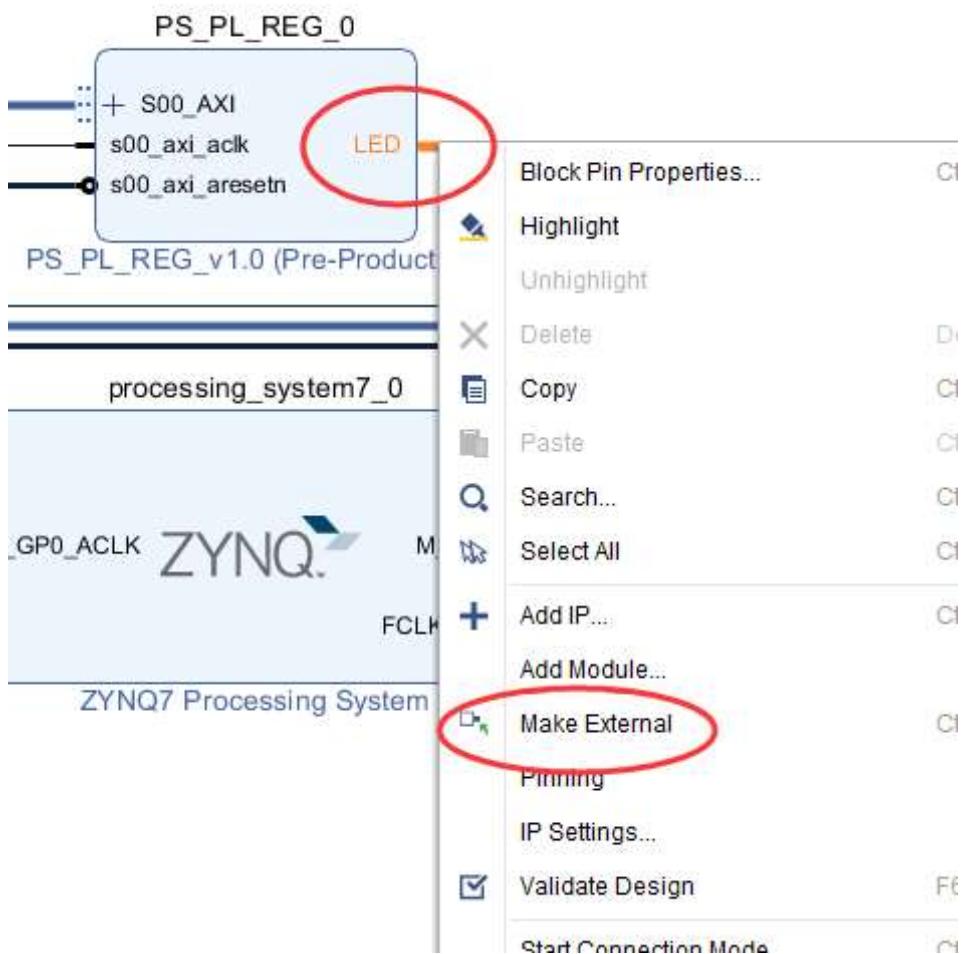
Designer Assistance available. Run Connection Automation



As shown in the figure below, the system automatically adds the modules that need to be added for us, and automatically connects each module

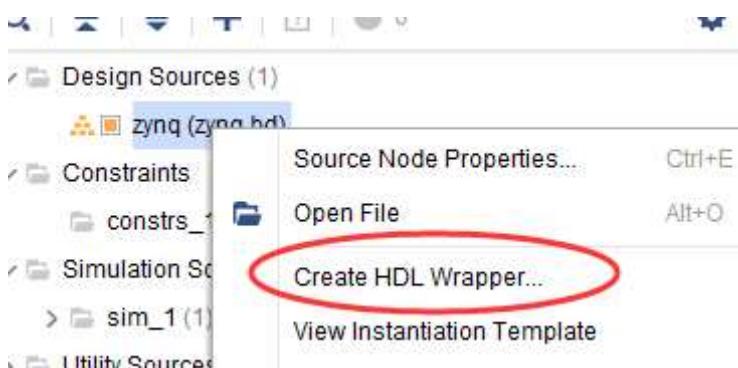


6) Because our demo requires the LED to observe the results, right-click the LED here to print the LED window



In this way, our entire hardware system is laid out.

- 7) In Design Sources, right-click on the ZYNQ module we just designed and click Create HDL Wrapper



- 8) Compile, assign pins for LEDs, synthesis

Compile RUN



Assign pins to the LEDs

The screenshot shows a table titled "Find Results" with various columns: Name, Direction, Interface, Neg Diff Pair, Package Pin, Fixed, Bank, I/O Std, and V_{cc}. The table lists several pins and their properties. The row for "LED_0" is highlighted with a red box around its entire row. The "Package Pin" column for "LED_0" contains a dropdown menu with "P20" selected. The "I/O Std" column for "LED_0" also has a dropdown menu with "LVCMOS33*" selected.

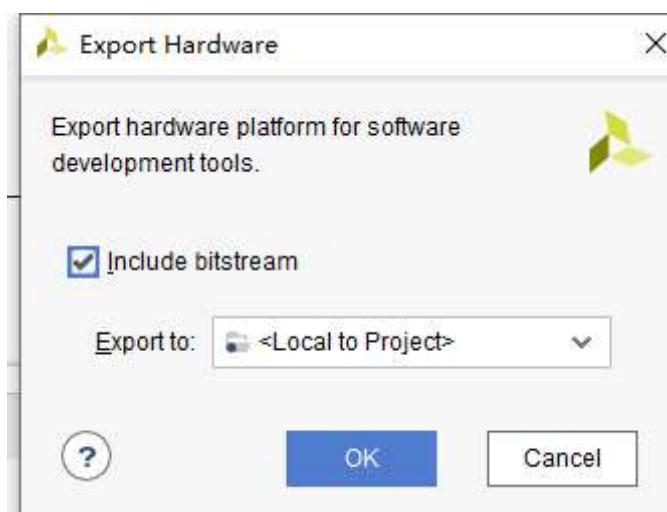
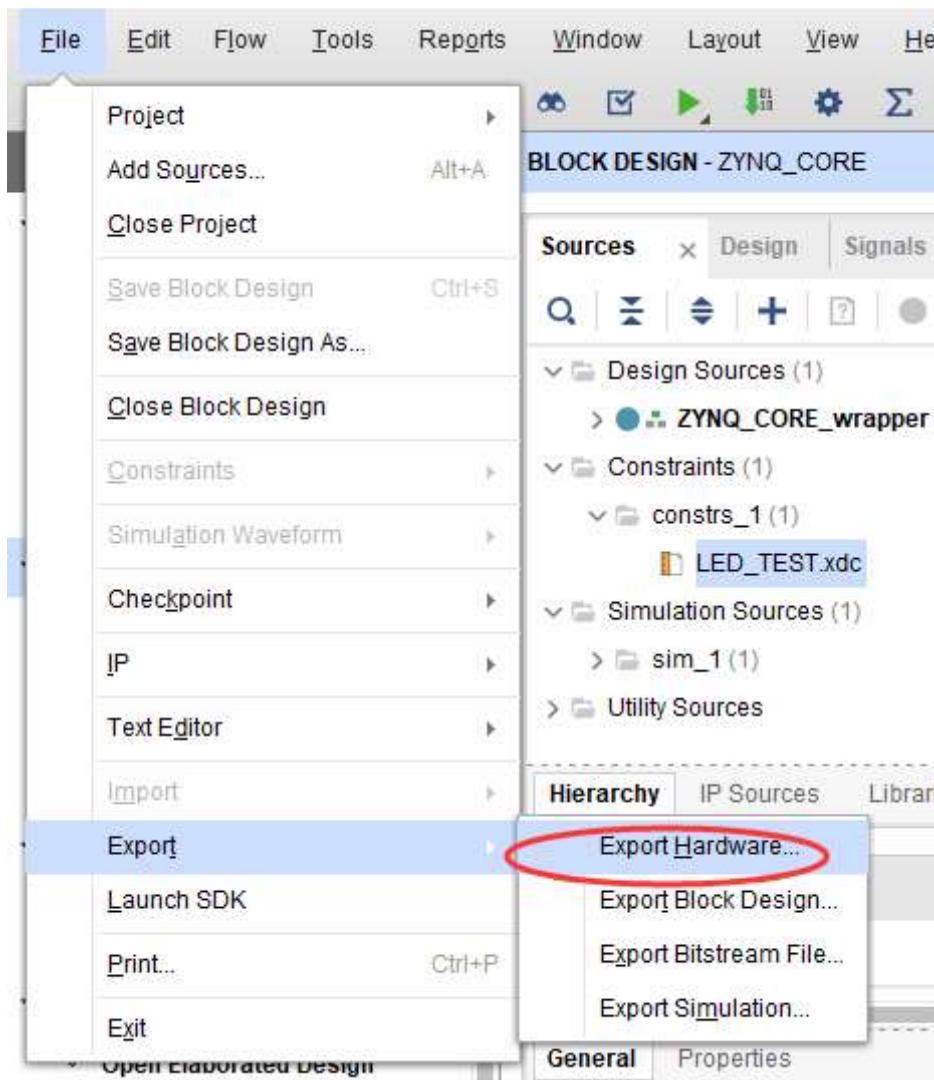
Name	Direction	Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	V _{cc}
FIXED_IO_mio[53]	INOUT	FIXED_IO_32035		C12	✓	501	default (LVCMOS18)	1
FIXED_IO_ps_clk	INOUT	FIXED_IO_32035		F7	✓	500	LVCMOS33*	3
FIXED_IO_ps_porb	INOUT	FIXED_IO_32035		B5	✓	500	LVCMOS33*	3
FIXED_IO_ps_srstb	INOUT	FIXED_IO_32035		C9	✓	501	LVCMOS33*	3
LED_0	DUT			P20	▼	34	LVCMOS33*	3

Synthesize and generate a bit file

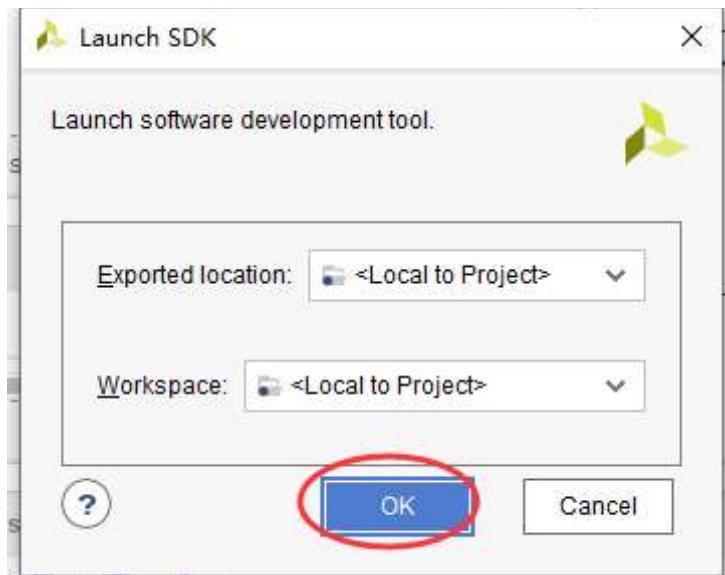


3. PS part of the project creation

- 1) File→Export→Export hardware..., check "include bitstream" in the pop-up dialog box, and click "OK" to confirm, as shown in the figure below.



2) File→ Launch SDK, in the pop-up dialog box, save the default, click "OK", as shown in the figure below.

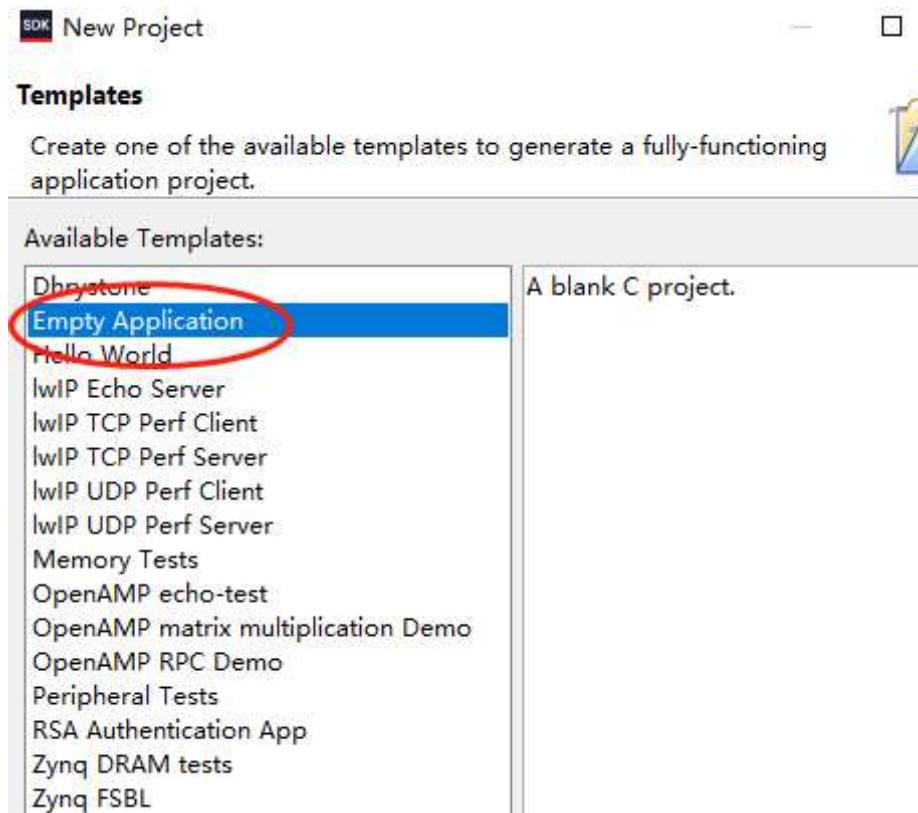


The SDK development environment will automatically open

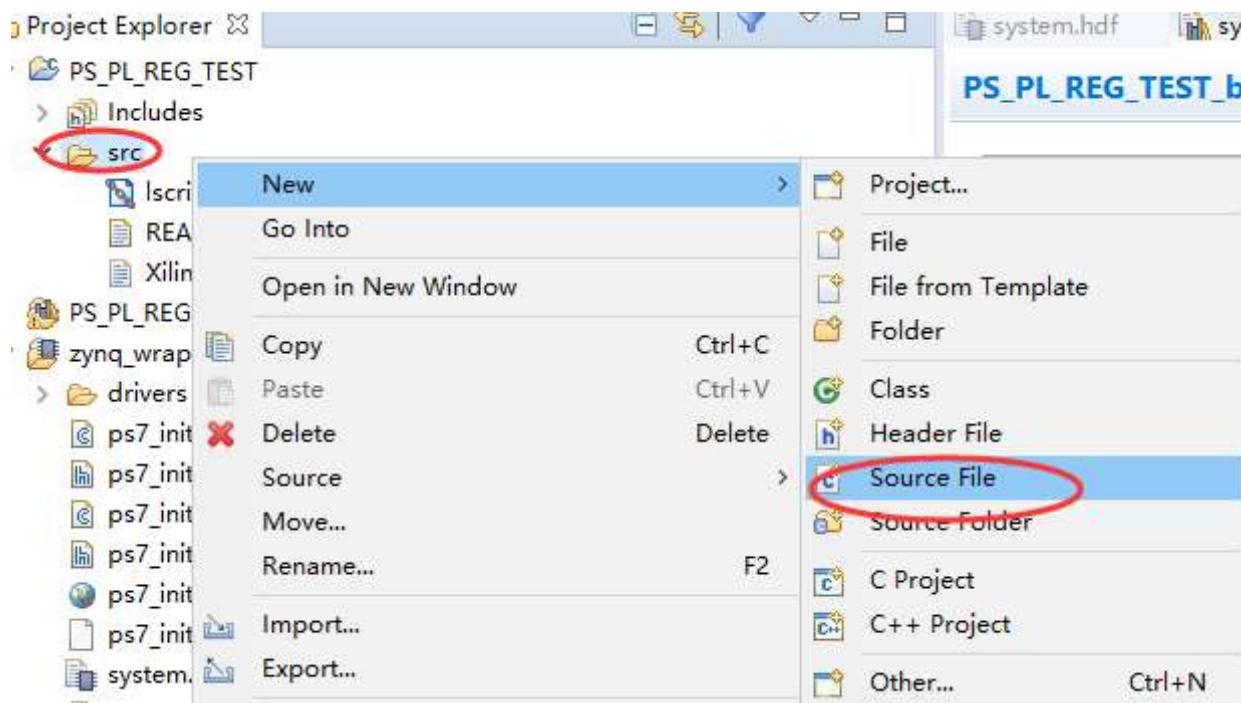
The SDK development environment will automatically open

3) Create another LED project, responsible for the flashing of LED indicators

a. Create a new, empty project, which can be named PS_PL_REG_TEST



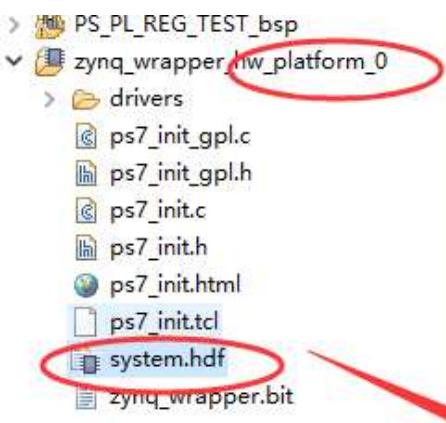
b. Right-click the SRC directory of the project and create a new SOURCE FILE



c. Take the name main.c

4. Writing code on the PS side

1. Click system.hdf in the platform Here all resources will be displayed here, find the PS_PL_REG we created earlier, you can see that the base address of this peripheral is 0X43C00000, we will use this address later



Design Information

Target FPGA Device: 7z010
 Part: xc7z010clg400-1
 Created With: Vivado 2018.3
 Created On: Wed Feb 16 19:39:50 2022

Address Map for processor ps7_cortexa9 [0-1]

Cell	Base Addr	High Addr
ps7_intc_dist_0	0xf8f01000	0xf8f01fff
PS_PL_REG_0	0x43c00000	0x43c0ffff
ps7_scutimer_0	0xf8f00600	0xf8f0061f
ps7_slcr_0	0xf8000000	0xf8000fff
ps7_scuwdt_0	0xf8f00620	0xf8f006ff
ps7_l2cachec_0	0xf8f02000	0xf8f02fff
ps7_scuc_0	0xf8f00000	0xf8f000fc

2. Add the following code in main.c

```
#include "xparameters.h"
#include "xil_io.h"

#define LED_BASE_ADDR 0x43c00000

int main()
{
    volatile int Delay;

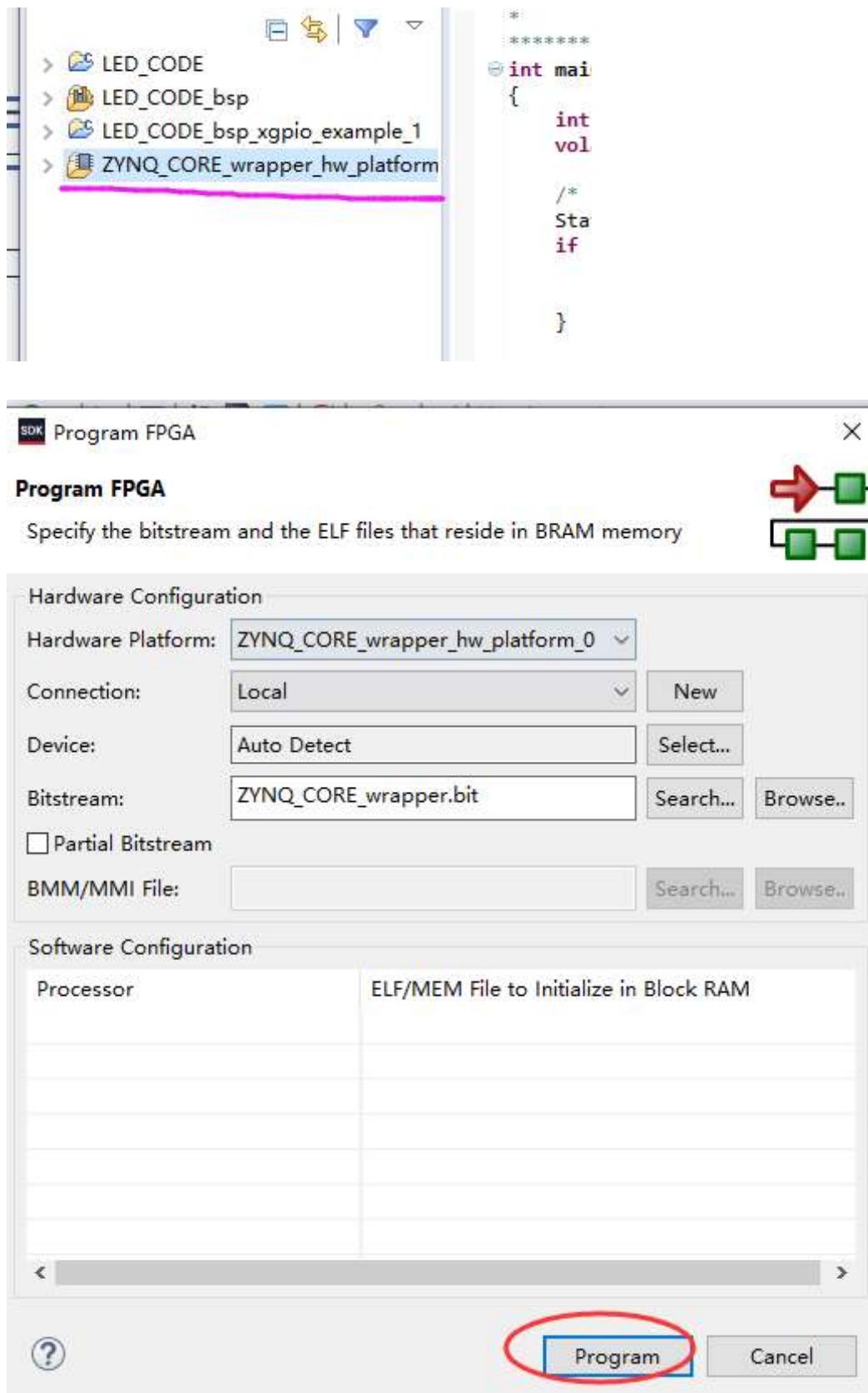
    while(1)
    {
        Xil_Out32(LED_BASE_ADDR, 1);
        for (Delay = 0; Delay < 10000000; Delay++);
        Xil_Out32(LED_BASE_ADDR, 0);
        for (Delay = 0; Delay < 10000000; Delay++);
    }

    return 0;
}
```

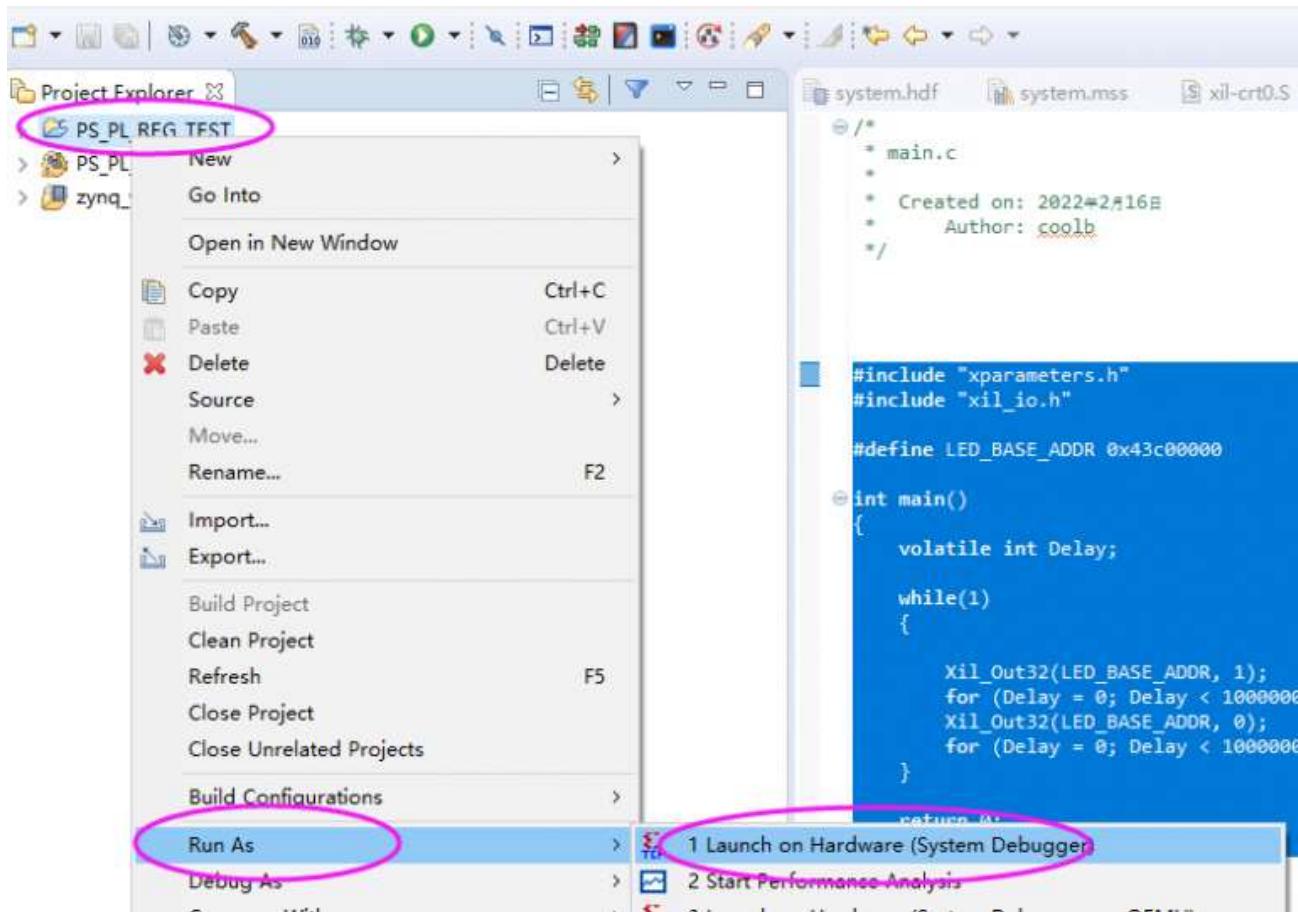
3. Download the program to the board for verification

- a) Select the hardware platform in the project, right-click → Program FPGA, select Default in the pop-up dialog box, click "program" to complete the Program work in the FPGA PL

part

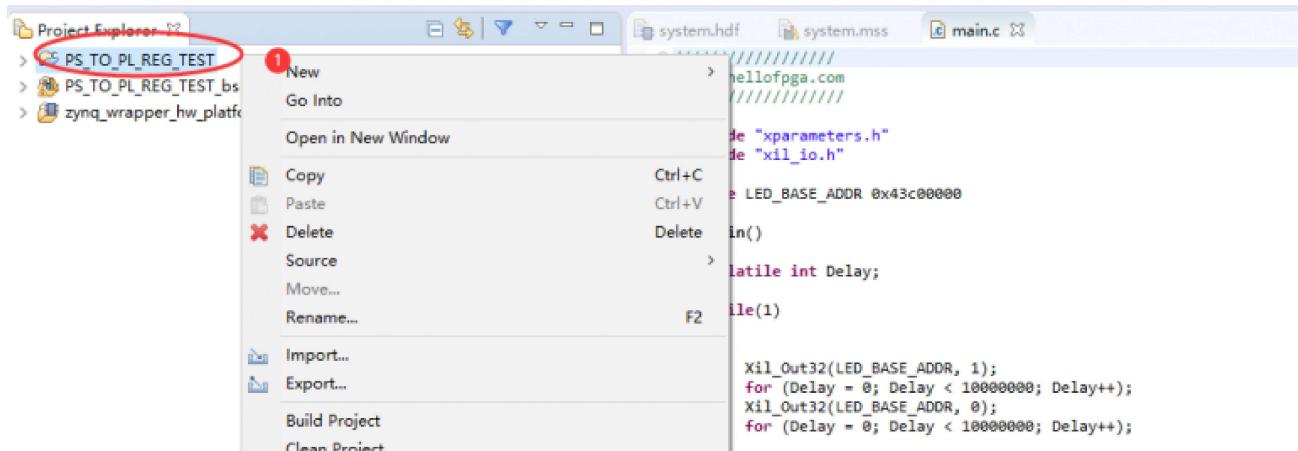


- b) Right-click on the project we created and select Run As→1 Launch on Hardware (System Debugger)

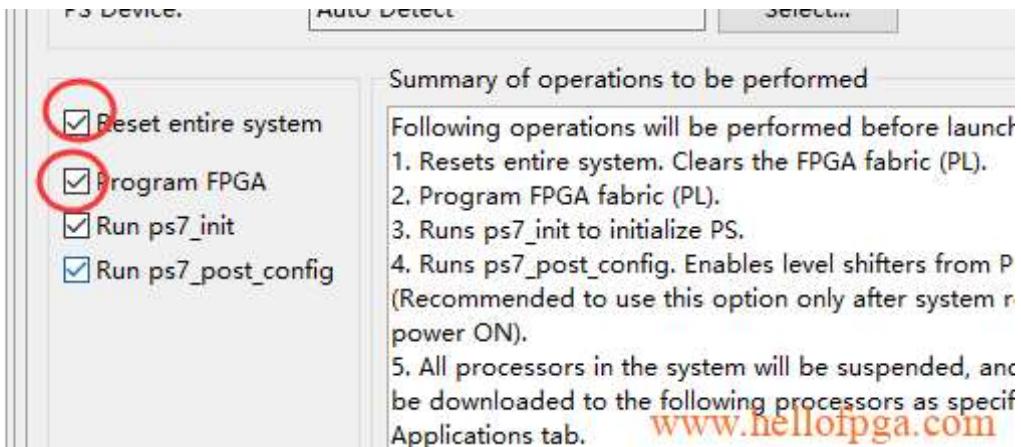


You can see the LED1 light on the board blinking

If you want to automatically reload the FPGA bit when DEBUG, see the following operation to set the right-click project →RUN AS→ Run Configurations



Check the Program FPGA and Reset Entire System options



After the previous two steps of setup, each time you click the green arrow DEBUG, the system will reset the FPGA and re-program the PL part

d) Code interpretation

```
#define LED_BASE_ADDR 0x43c00000
```

The 0X43C00000 here is the module address we saw earlier

Xil_Out32(LED_BASE_ADDR, 1); This is equivalent to writing 1 to the module address above

Xil_Out32(LED_BASE_ADDR,0) ; This is equivalent to writing 0 to the base address of the module

Because of the 4 registers in the module created on the PL side, only register 0 corresponds to the LED, so we can control the LED indicator by simply writing information to the base address.

If you want to control register 1, register 2, register 3 only need to +1 on the basis of the base address

In addition, we can also use PS in reverse to read the register value of the PL side, the specific process is not described in expansion, the actual PS access can be Xil_In32 through instructions (LED_BASE_ADDR); (LED_BASE_ADDR the base address defined for us) to read.

Through the above method, we have successfully realized the interaction between PS and PL through registers, which is relatively simple and easy to start, of course, there are many ways to interact, and we will continue to sort it out later.

AXI's newsletter is also a big topic, and I will continue to organize the content when I have time

The complete project of this article is as follows:

[15_PS_TO_PL_REG_XC7Z020](#)

Download

 [SMART ZYNQ SP & SL](#)