ALINX Vivado-base MicroBlaze

Basic Tutorial

ALINX

Version Record

Version	Date	Release By	Description
Rev1.01	2020-11-03	Rachel Zhou	First Release

We promise that this tutorial is not a permanent, consistent document. We will continue to revise and optimize the tutorial based on the feedback of the forum and the actufal development experience.

The English version was translated by Shanghai Tianhui Trading Company. They has not been officially Review by ALINX and are for reference only. If there are any errors, please send email to alinx@aithtech.com for correction.

Amazon Store: https://www.amazon.com/alinx

Aliexpress Store:

https://alinxfpga.aliexpress.com/store/911112202?spm=a2g0o.detail.1000007.1.704e2bedqLBW90

Ebay Store: https://www.ebay.com/str/alinxfpga

Customer Service Information

Wechat/Skype: 15026866269

Technical Email:alinx@aithtech.com

Sales Email: rachel.zhou@alinx.com

Content

Preparation and precautions

Software Environment

The software development environment is based on Vivado 2019.1

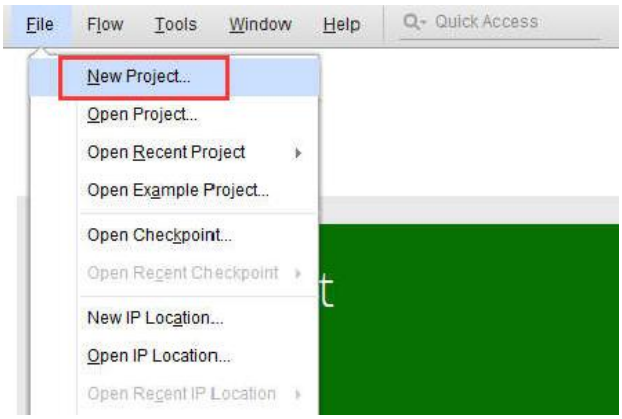
Part 1: Simple MicroBlaze System Setup

This chapter uses Vivado software to build a MicroBlaze minimum system.Use the FPGA internal block RAM as the CPU to run RAM, The main peripherals are UART and GPIO

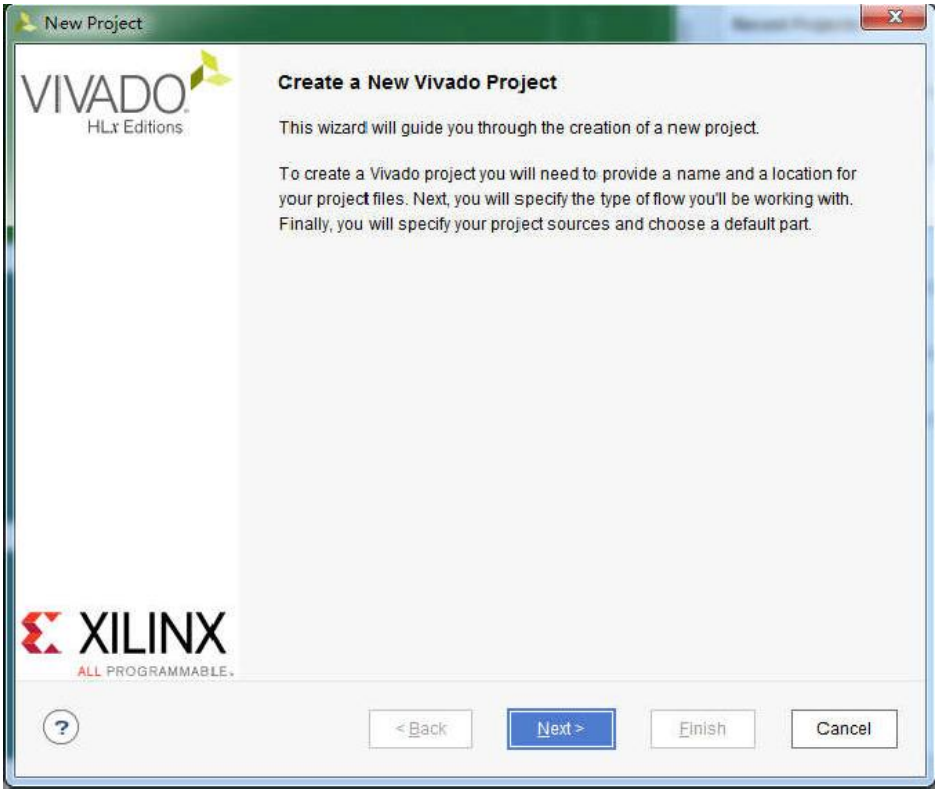
Part 1.1: Create a VIVADO project

Part 1.1.1: Create a new project

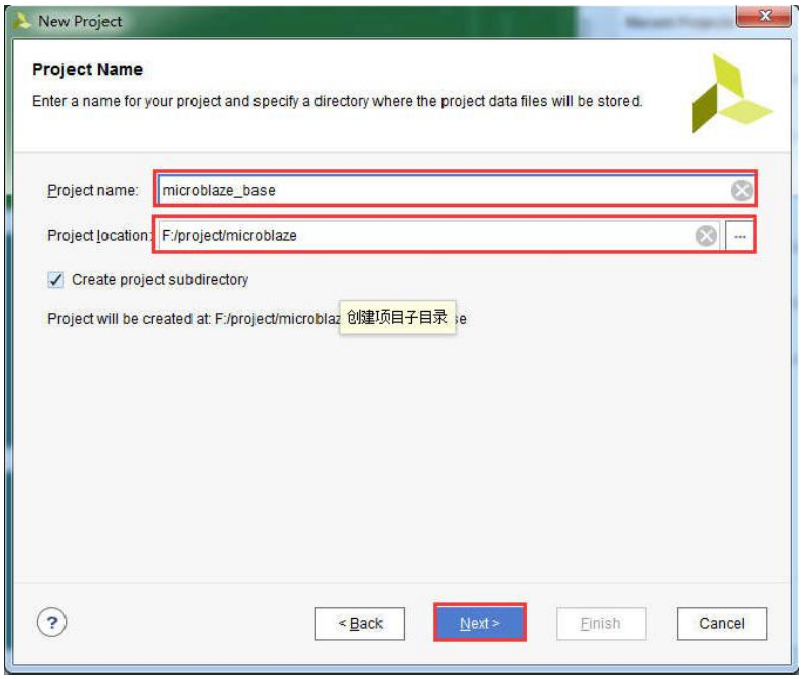
1) Open Vivado software, create a new project



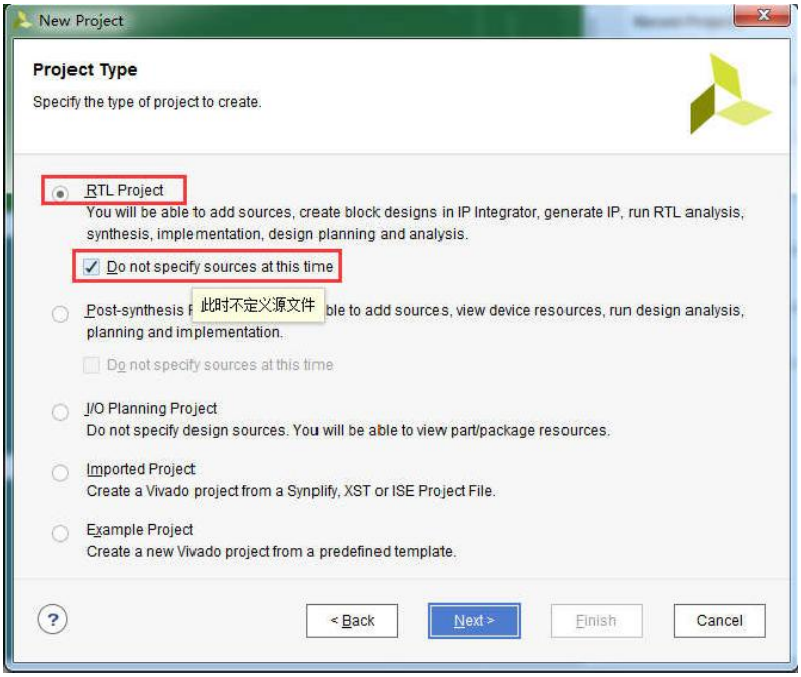
2) Click "Next" to create a new Vivaado project



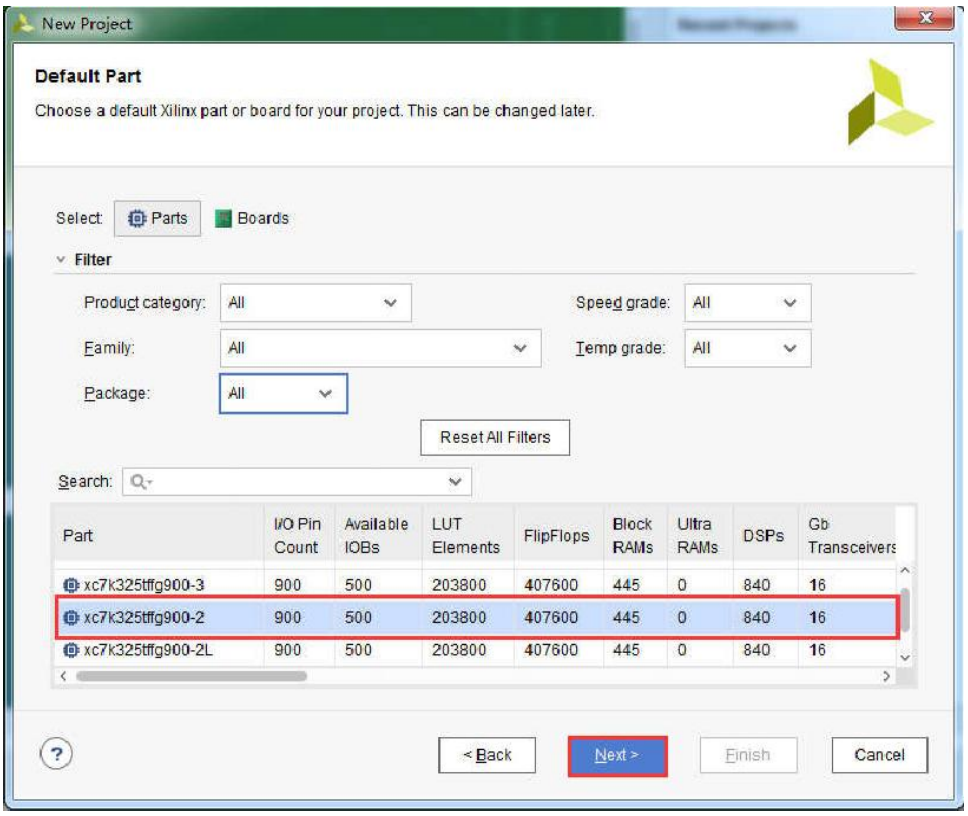
3) Fill in the project name "microblaze_base", select the project path, and keep the default project subdirectory option. The project name and project path should not have Chinese spaces. The project path should be as short as possible, otherwise compilation errors may occur. Click "Next"



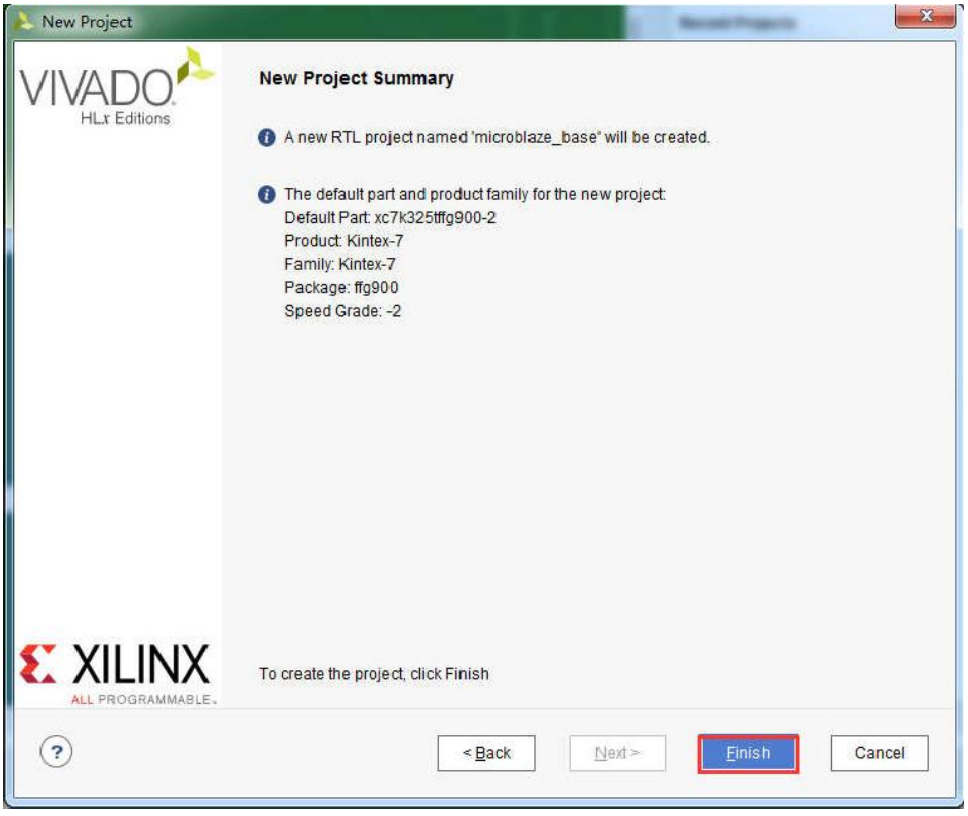
4) Choose to create an RTL project and check "Do not specify sources at this time"



5) Select the FPGA model according to the FPGA development board and click "Next"

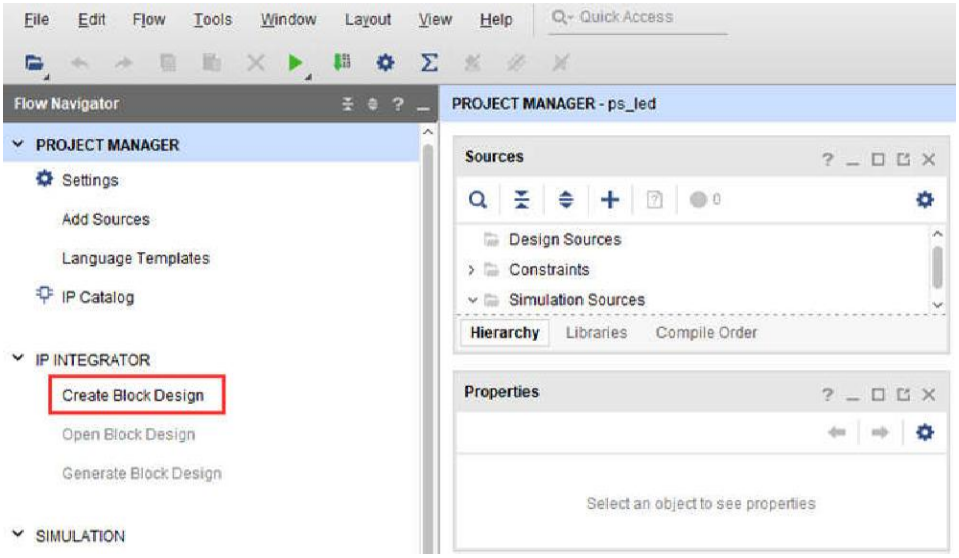


6) Click "Finish" to complete the project creation wizard

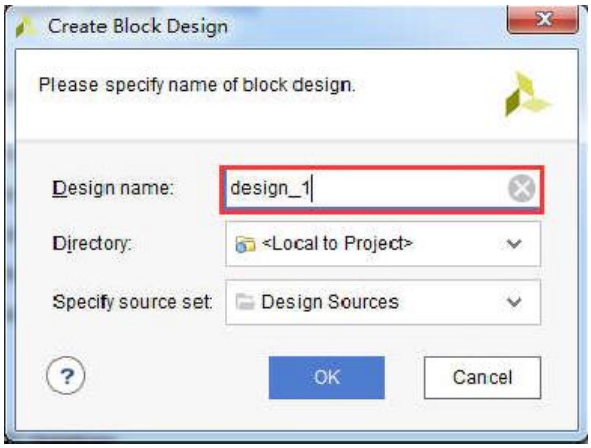


Part 1.1.2: Create a block design

7) Click “Create Block Design”

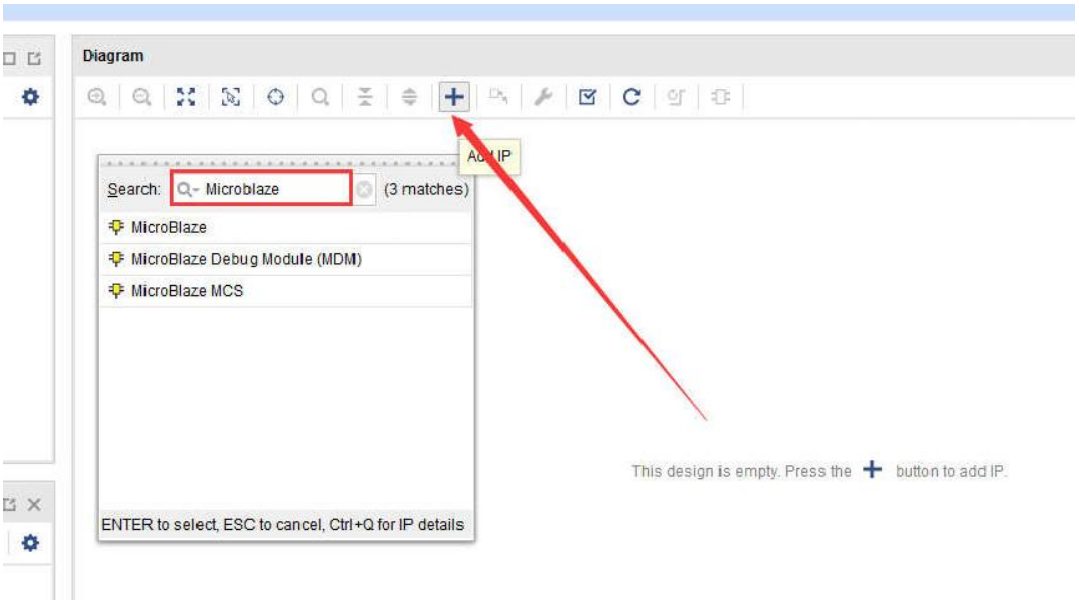


8) Keep the design name as the default and click "OK"

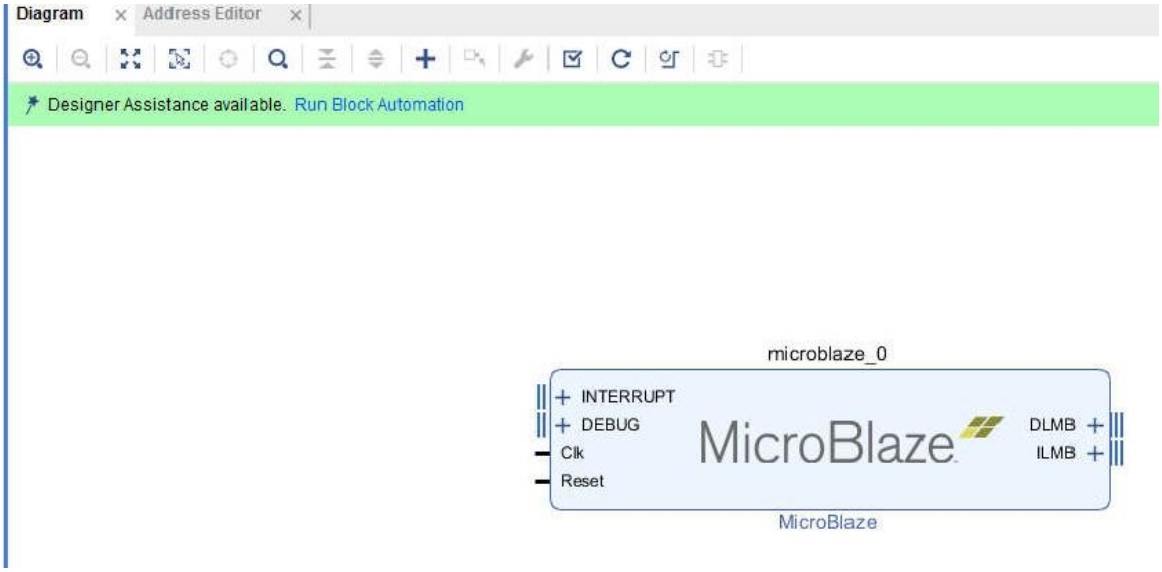


Part 1.1.3: Add soft core

9) Click the Add IP shortcut button to search for “Microblaze” and double-click “Microblaze” in the drop-down list.

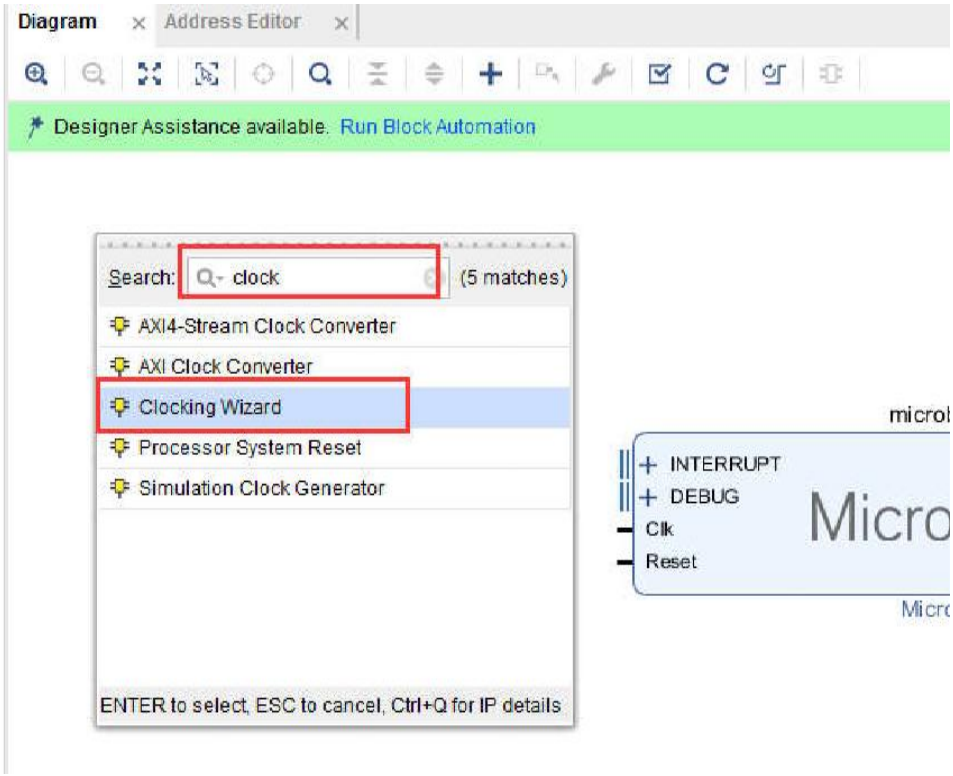


10)A Microblaze core is added to the Block design



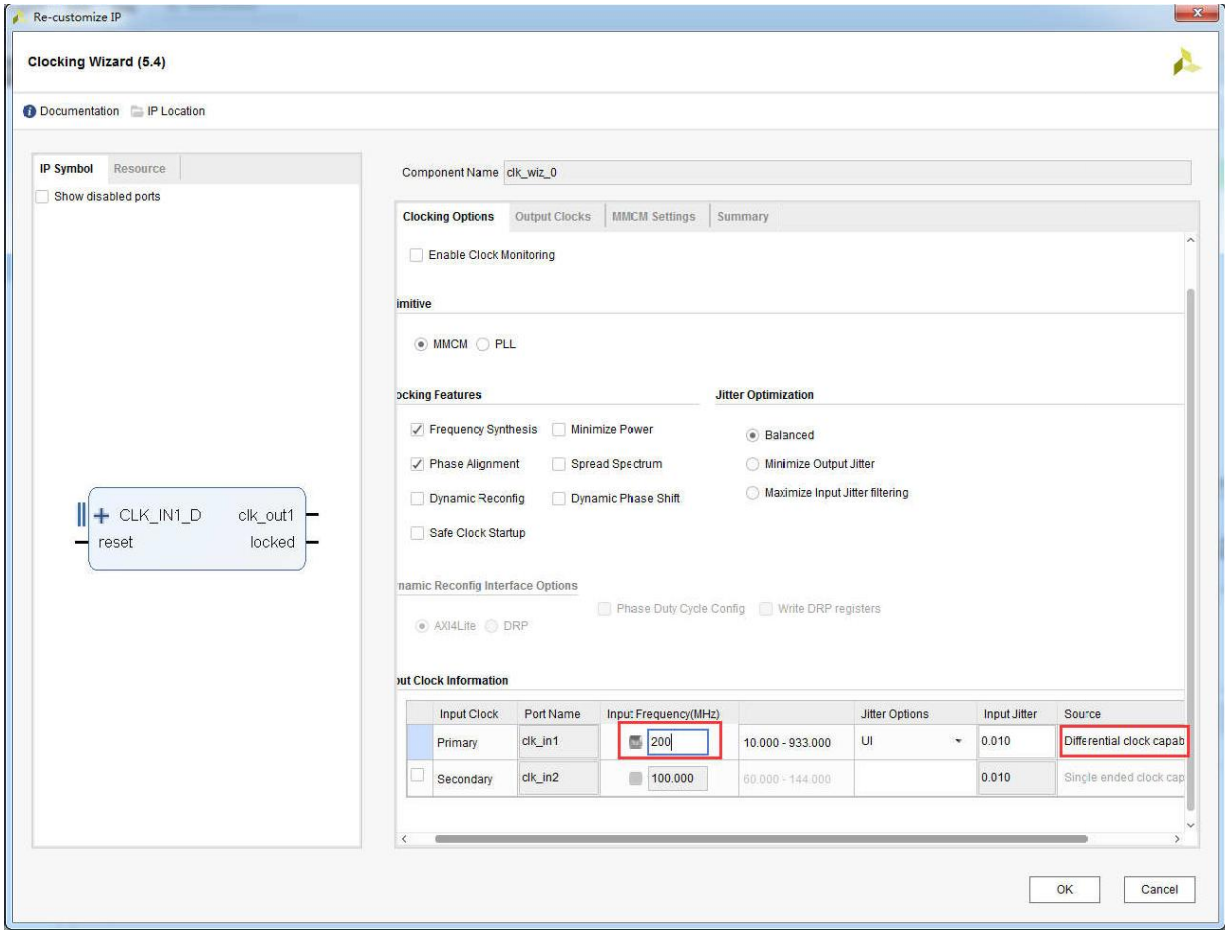
Part 1.1.4: Add clock

11)Search for "clock" and double-click "Clocking Wizard" to add a clock wizard IP core (PLL)

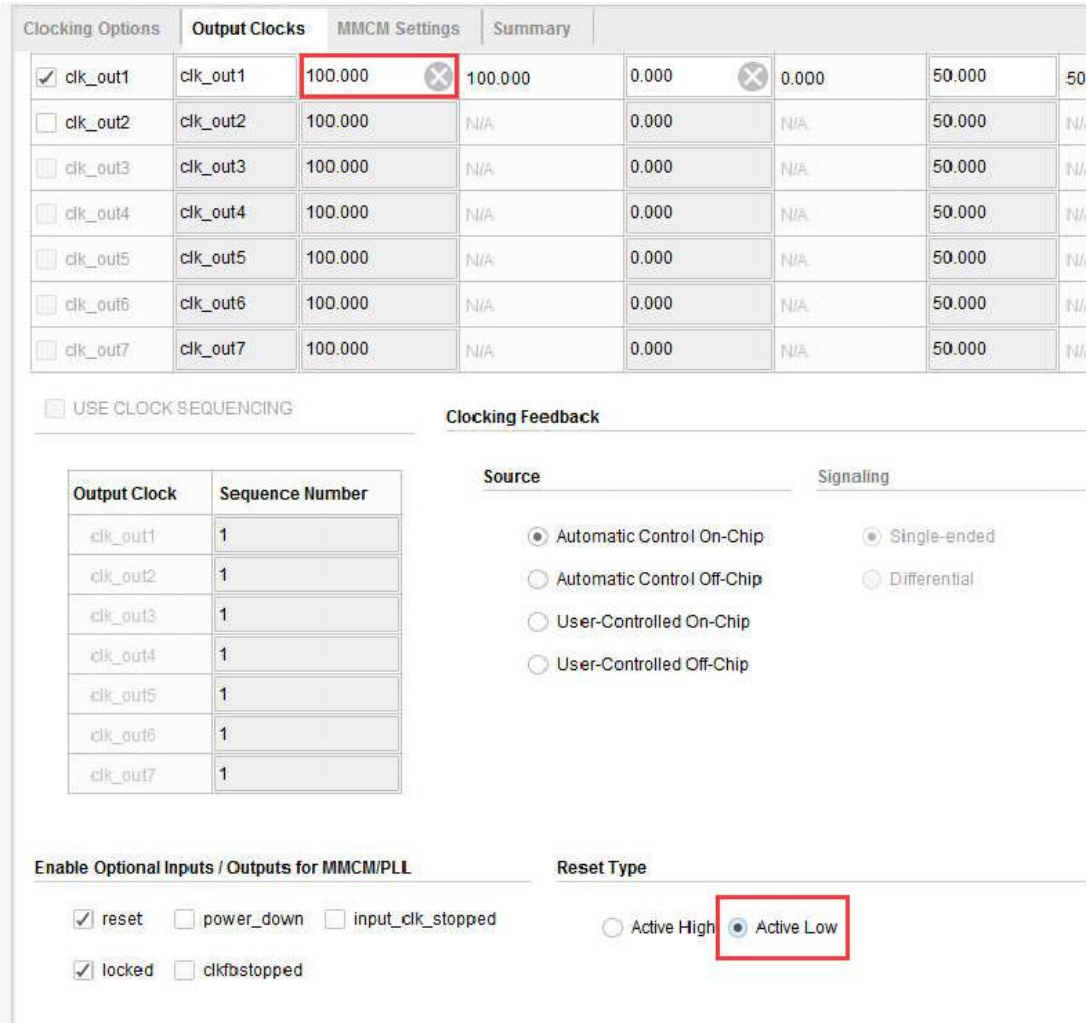


12) Double-click the clock wizard IP core you just added to set the parameters.

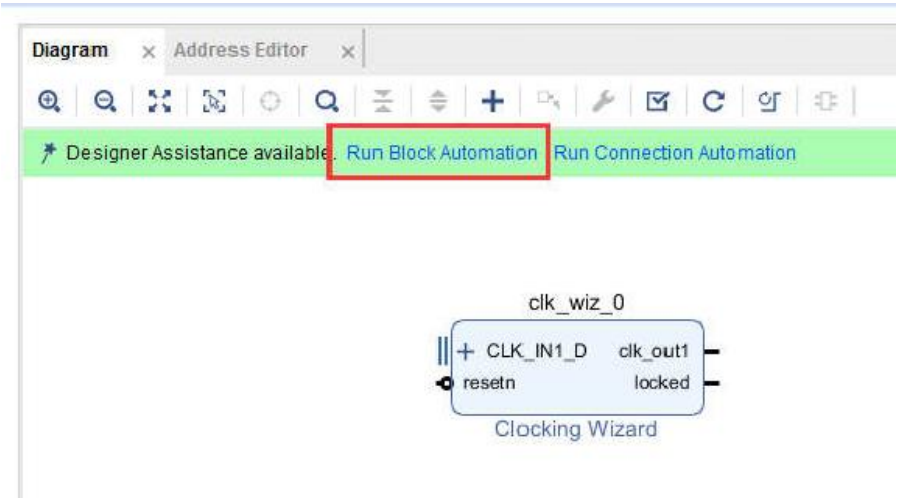
13) In the “Clocking Options” tab, the clk_in1 frequency is filled in 200, and the Source selects the differential clock. The settings here are based on the board's specific conditions.



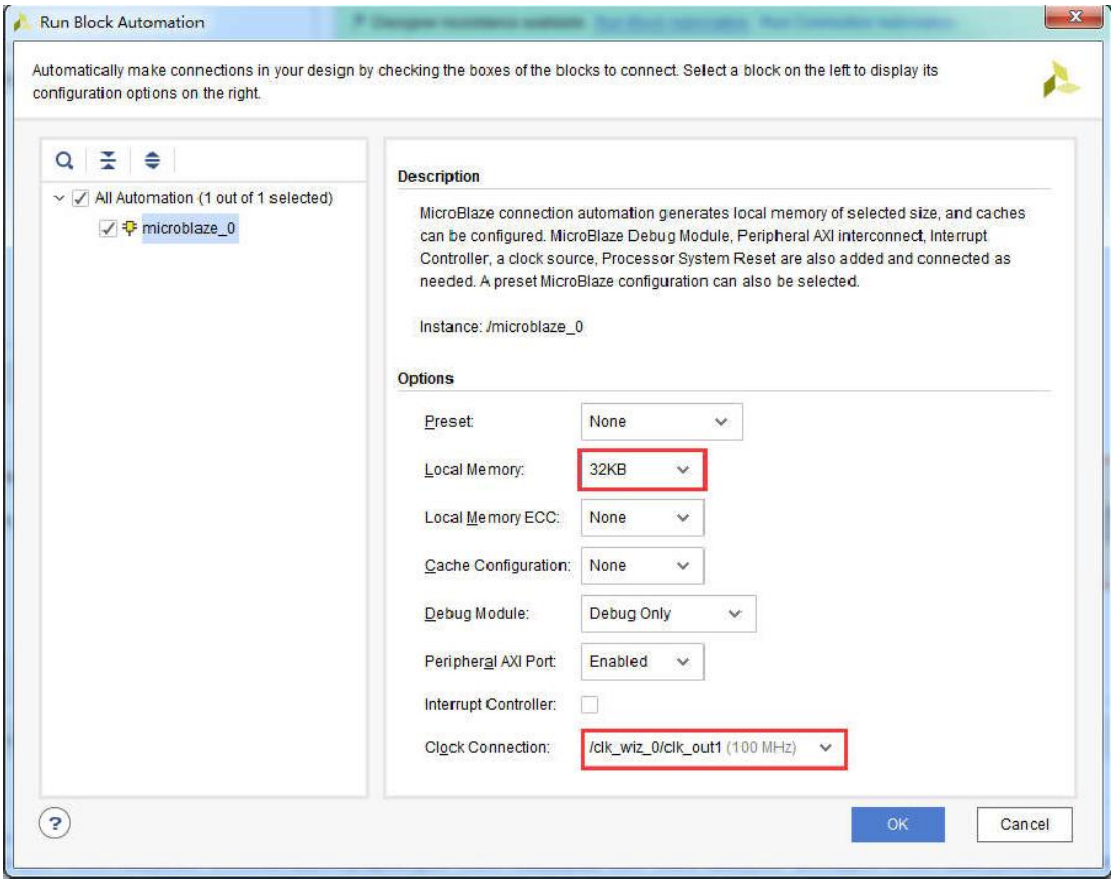
14) Clk_out1 runs as the Microblaze clock, where 100Mh is not modified, and the reset type selects “Active Low”. Low level reset, because the board's key is low when pressed.



15) Click "Run Block Automation" to complete some automatic settings. Note that this option will not always be available. Once the settings are completed, there may be no chance to perform some automatic settings in this way.

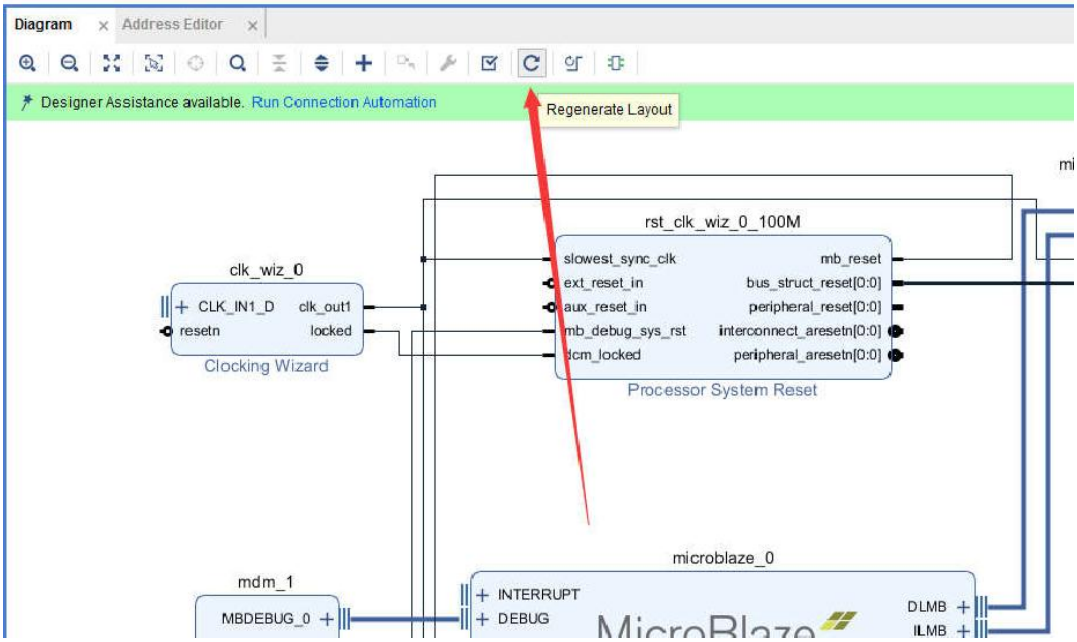


16) Complete some microblaze parameter settings in the pop-up window and keep the default "None" in the "Preset" option. This option is equivalent to presetting some microblaze configuration parameters, depending on the application. "Local Memory" selects 32KB, because there is no external memory, you can't choose too small here, otherwise the program will not run. The “Clock Connection” clock selects the 100Mhz clock output by the Clock Wizard as the CPU clock.



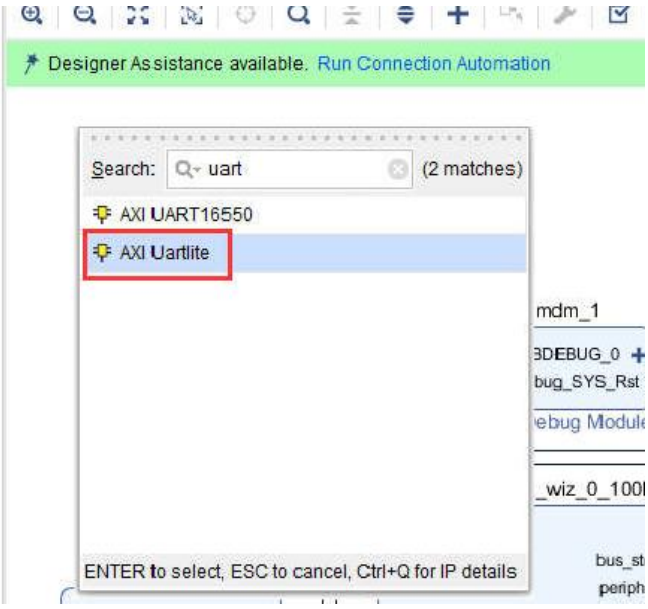
17) Click OK to complete the configuration and automatically generate some modules and complete some connections.

18) Click the re-layout button to complete the re-layout of each module.

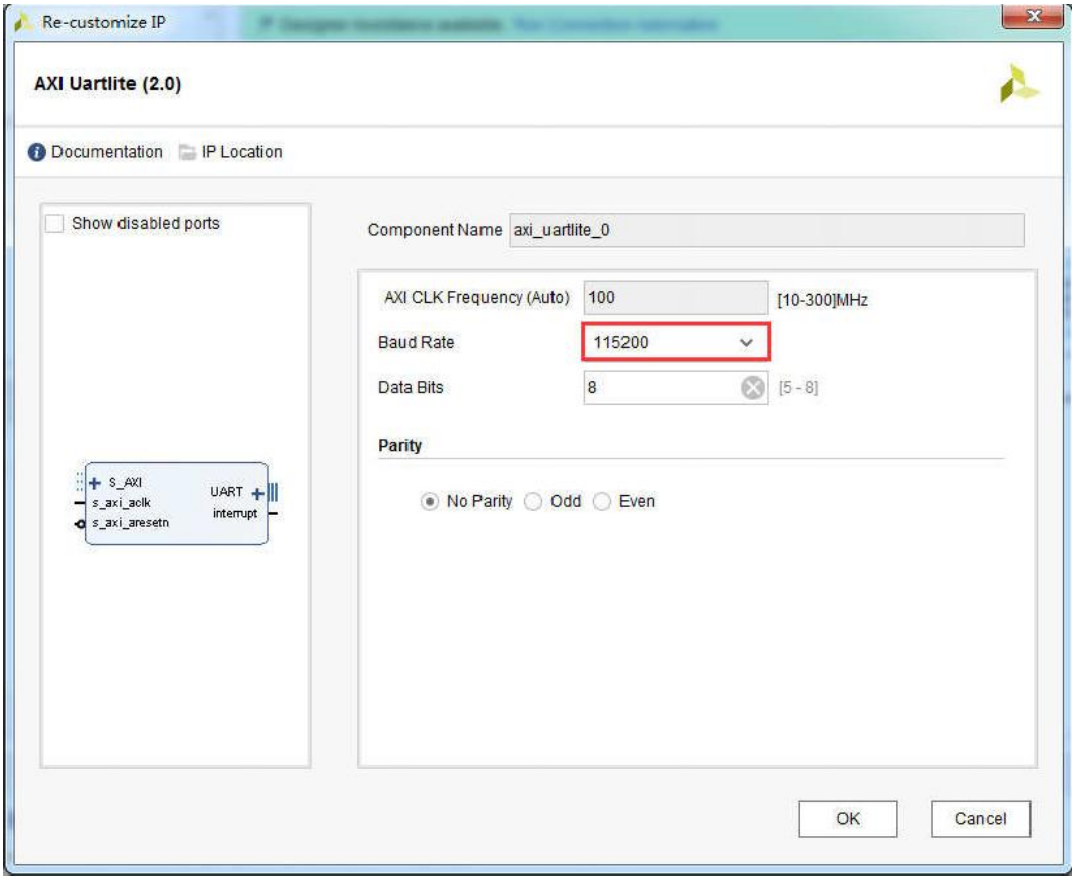


Part 1.1.5: Add peripherals

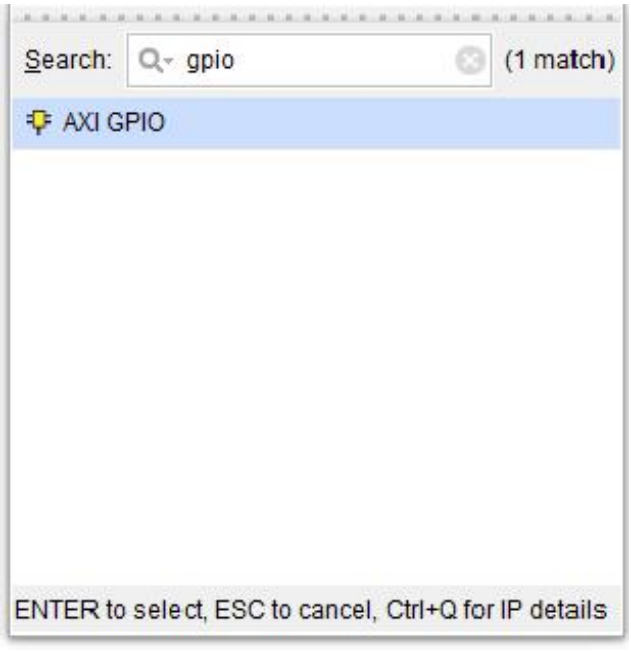
19) Search for "uart" and double-click "AXI Uartlite" to add a Uart serial IP core



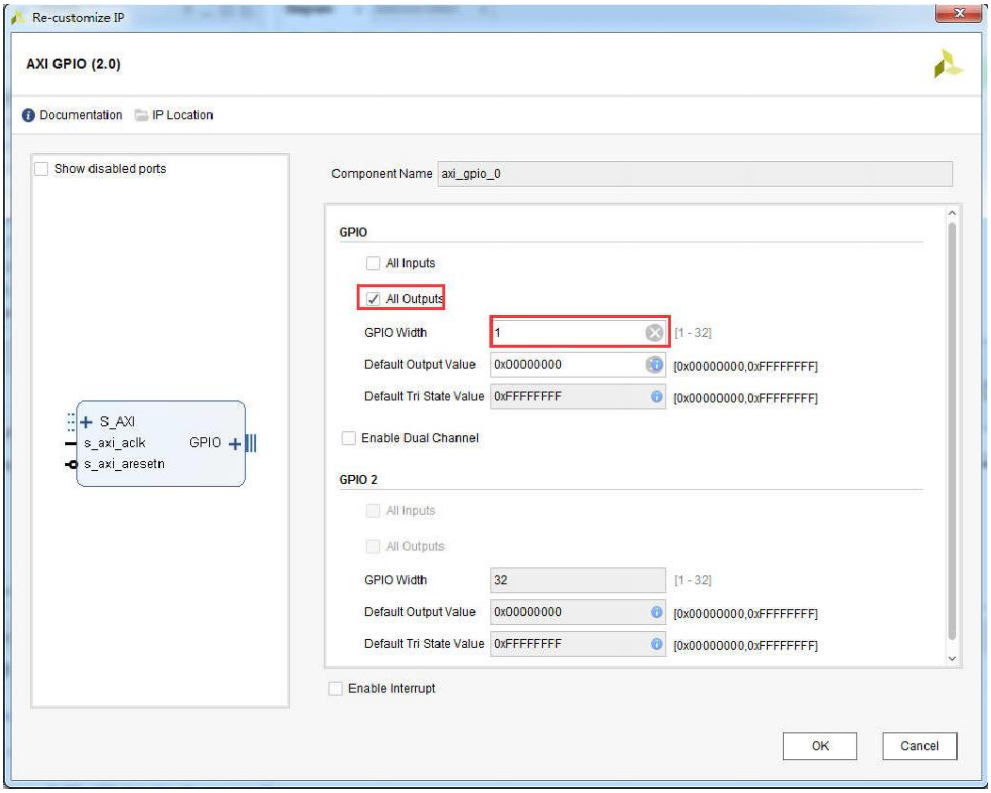
20) Double-click the Uartlite core you just added to modify the baud rate to 115200. This is the commonly used baud rate for embedded systems. The baud rate is a fixed value and cannot be dynamically modified in the program. Click OK to complete the configuration.



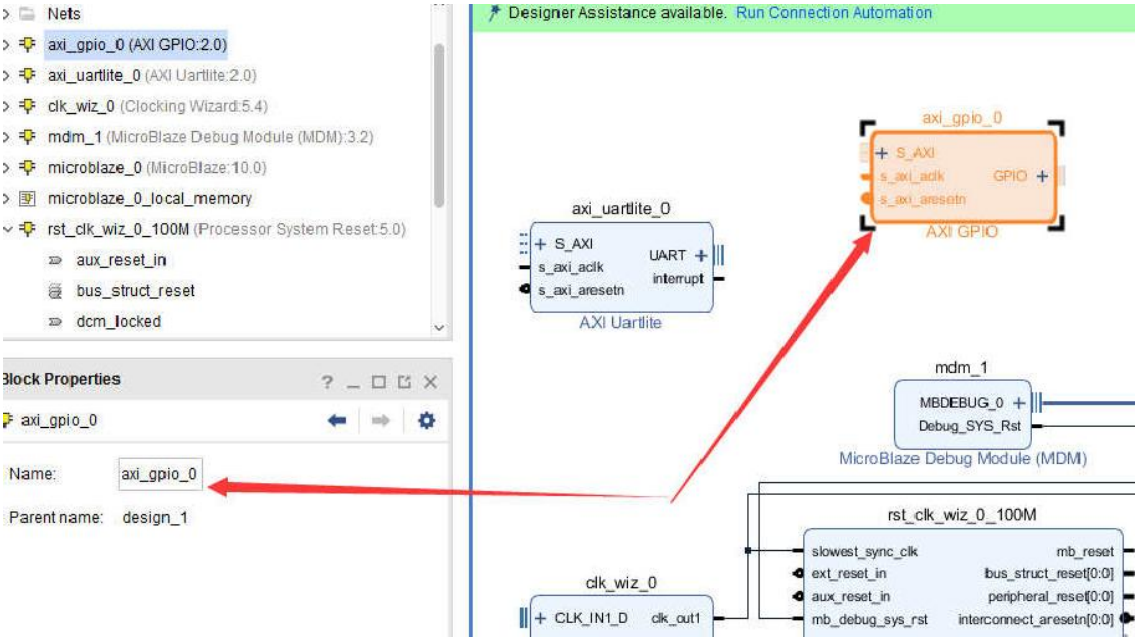
21) Search for gpio and double-click on “AXI GPIO” to add an AXI GPIO for controlling the LEDs.



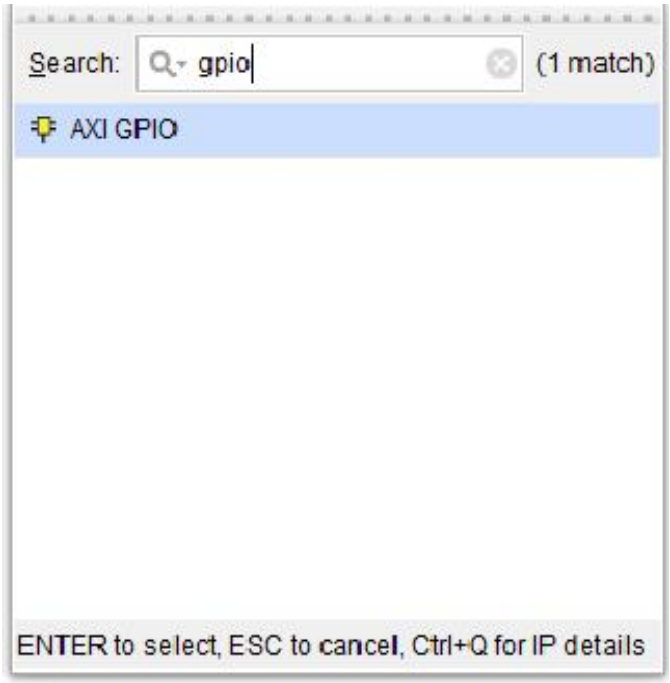
22) Double-click the AXI GPIO you just added to modify the parameters, check “All Outputs”, set to full output, “GPIO Width” fill in 1, here only control 1 bit LED, click “OK” to complete the configuration



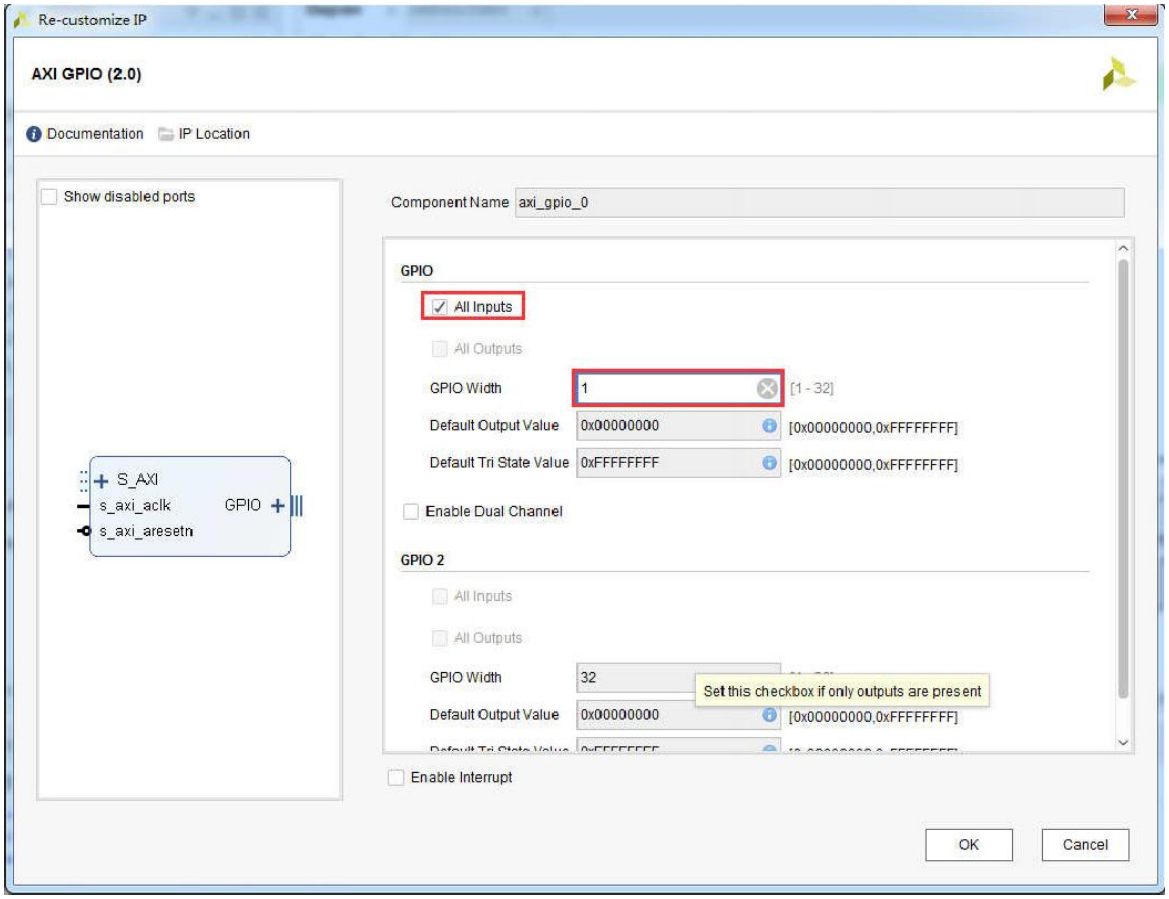
23) Select the AXI GPIO you just added, you can modify the module name in the Block Properties interface, here changed to "axi_led", need to note that if the name and the program are different, the software programming may need to modify the C language code of the routine, it is recommended for beginners to keep Consistent with the tutorial.



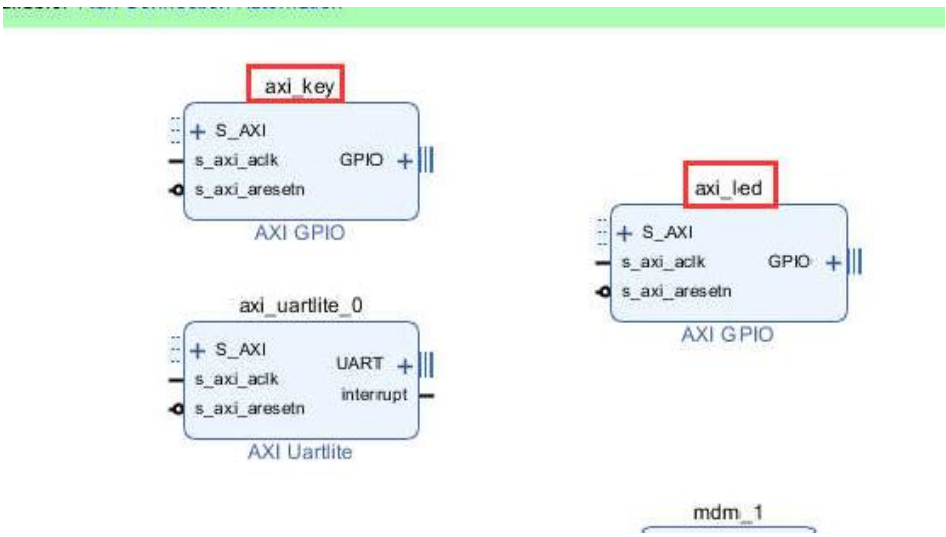
24)Add an AXI GPIO for key input



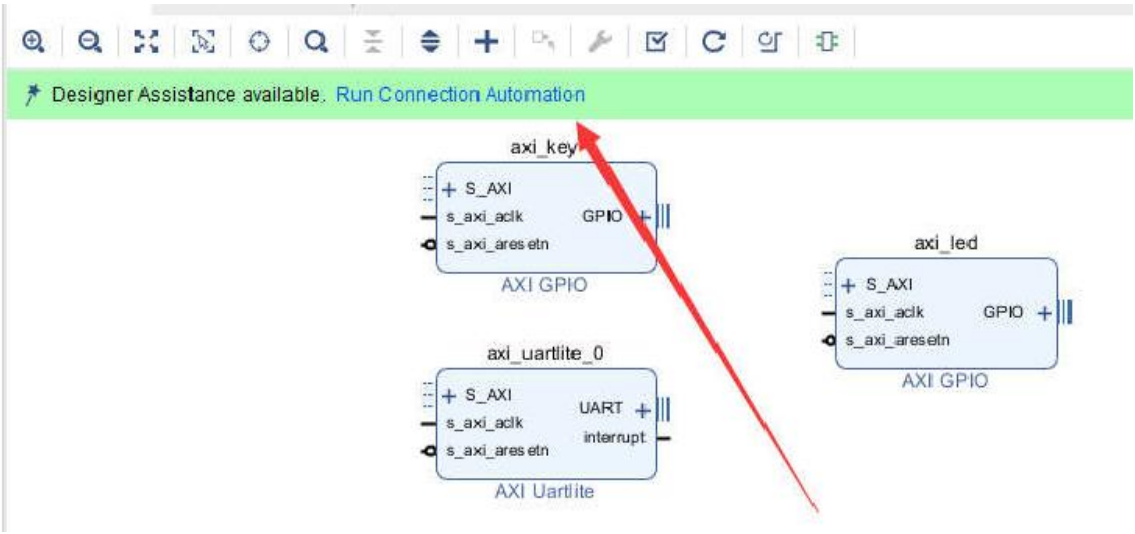
25)Modify the AXI GPIO parameters, check "All Inputs", fill in "GPIO Width", and only control the 1-bit button, click "OK" to complete the configuration.



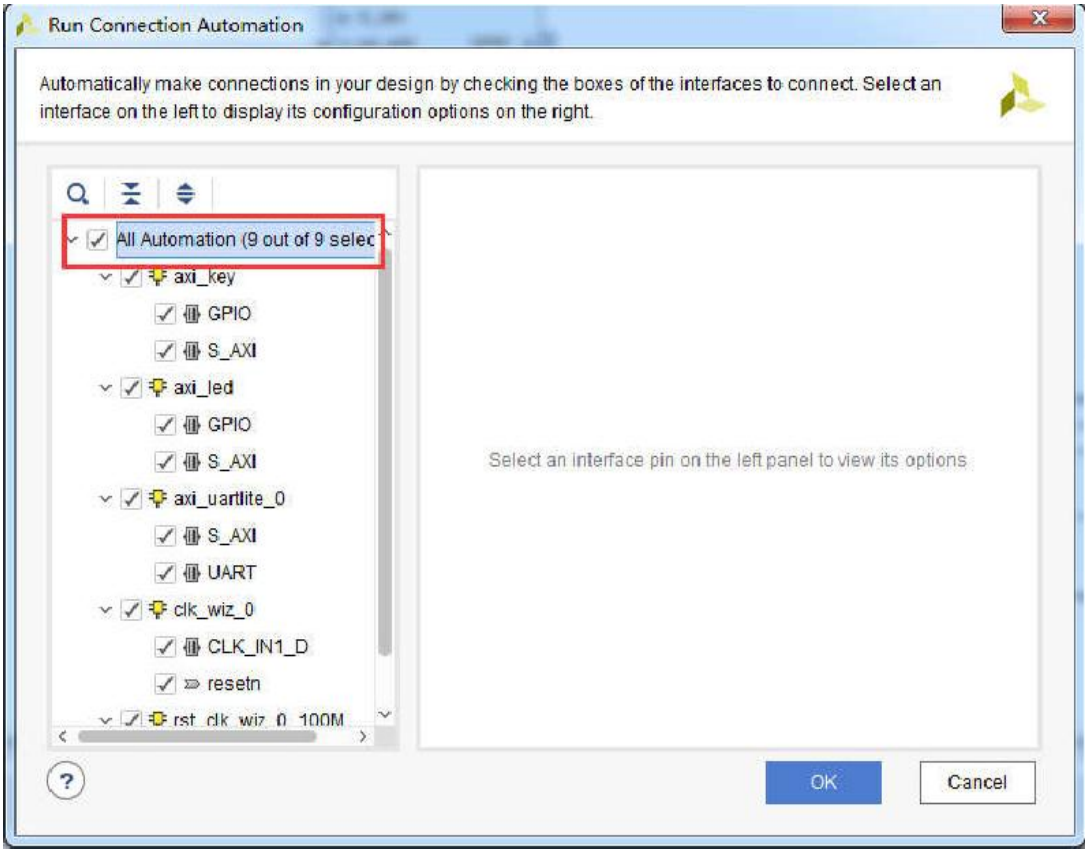
26) Select the AXI GPIO you just added. In the Block Properties interface, you can change the module name. This is changed to “axi_key”. Note that if the name and the routine are different, the C language code of the routine may need to be modified during software programming. It is recommended for beginners to keep Consistent with the tutorial.



27)Click "Run Connection Automation" to automatically connect

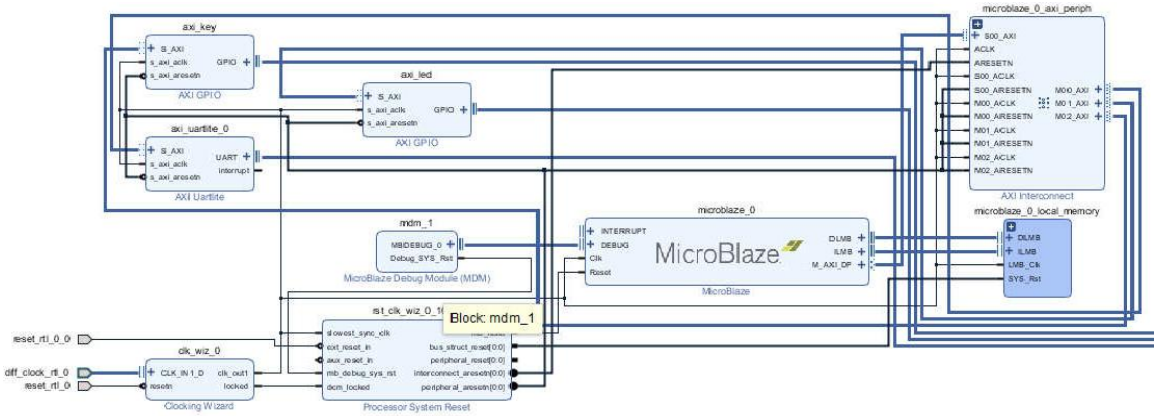


28)Select “All Automation” to automatically connect all the lines, click “OK” to start the automatic connection.

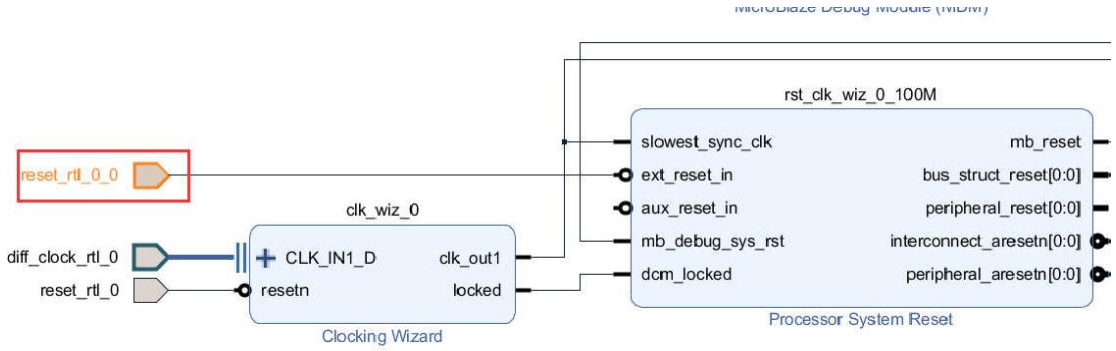


29)Auto-wiring completes the automatic connection of the newly added Uartlite and AXI GPIO, and automatically adds an AXI

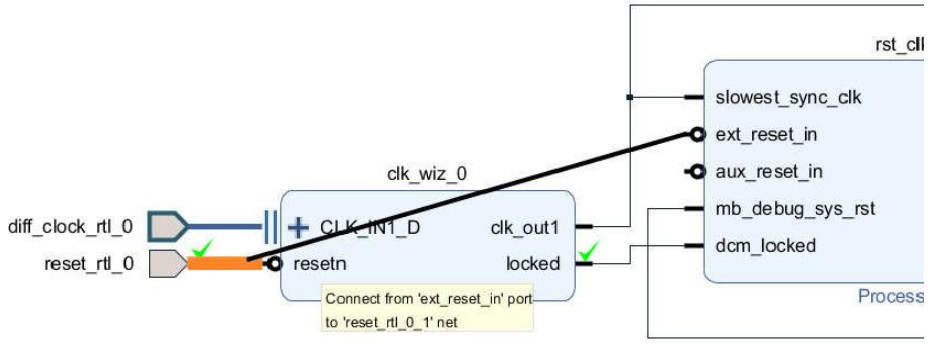
bus crossover module. The AXI cross-module can complete the multi-master multi-slave AXI device cross-connect.



30) Delete the port "reset_rtl_0_0" on the "rst_clk_wiz_0_100M" module, select it, press the "delete" button to delete



31) Connect "ext_reset_in" on the "rst_clk_wiz_0_100M" module to "resetn" in "clk_wiz_0", so that the two modules share a reset. Note that these two modules are active low-level reset.

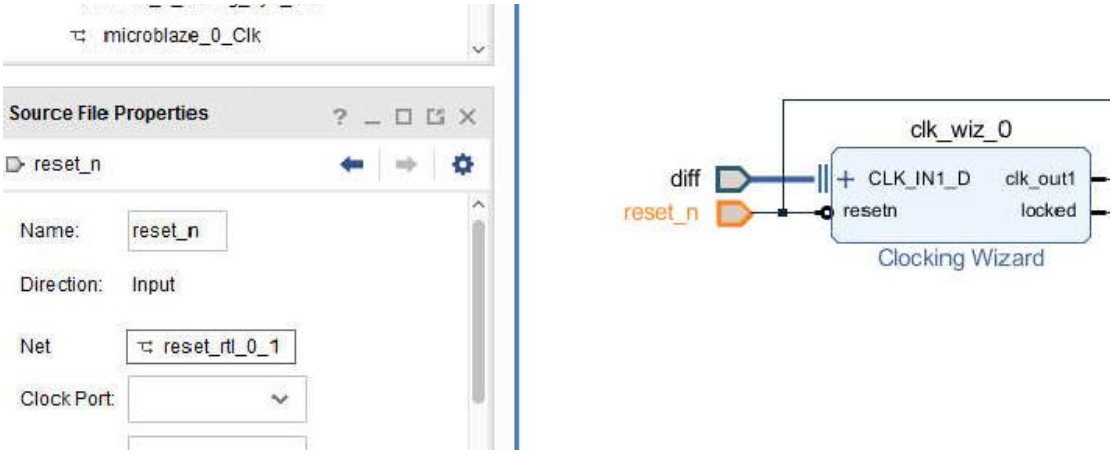


Part 1.1.6: Modify the port name

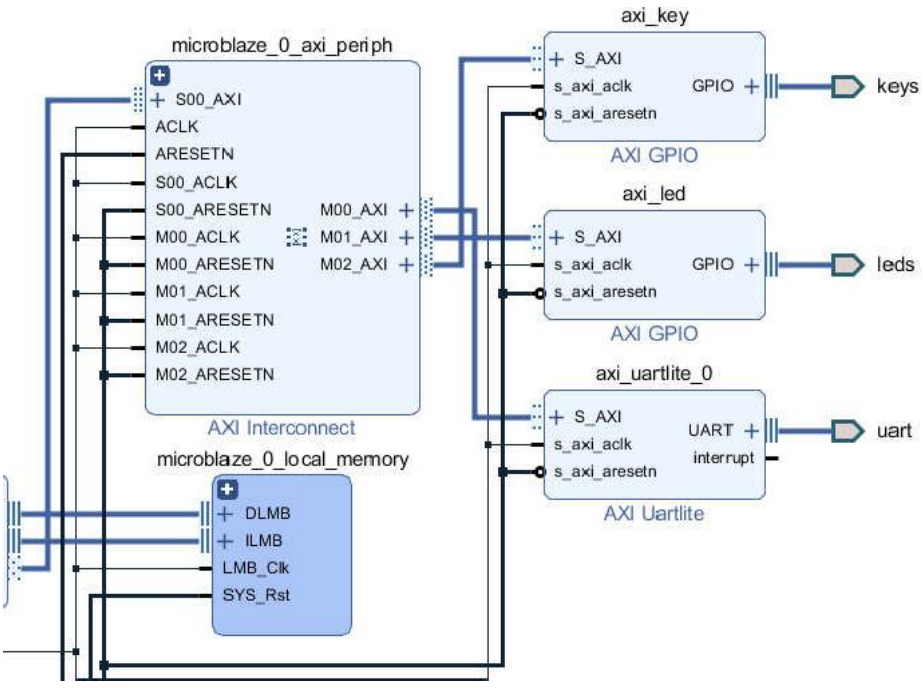
32) Modify each port name, the default port name is relatively long,

it is not easy to remember, this step is also a step that many beginners can easily forget, if this step is not modified, it will lead to inconsistency with the routine xdc file port name, the compiler will report an error.

33)"diff_clock_rtl_0" is changed to "diff", "reset_rtl_0" is changed to "reset_n"

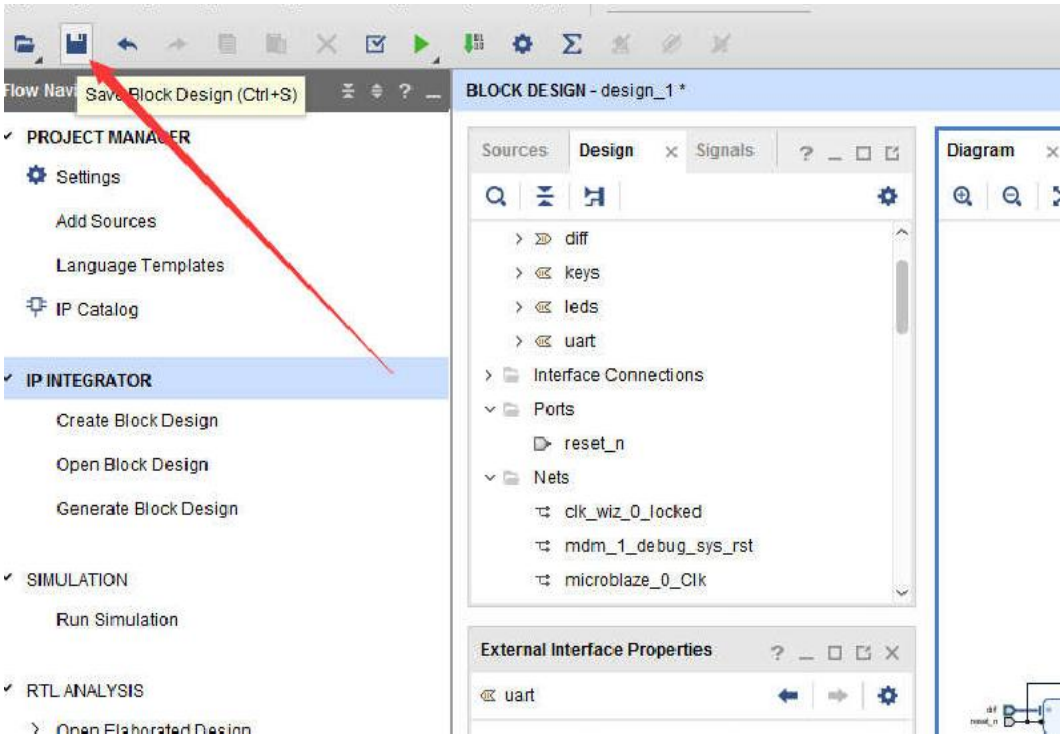


34)"gpio_rtl_1" is changed to "keys", "gpio_rtl_0" is changed to "leds", and "uart_rtl_0" is changed to "uart"



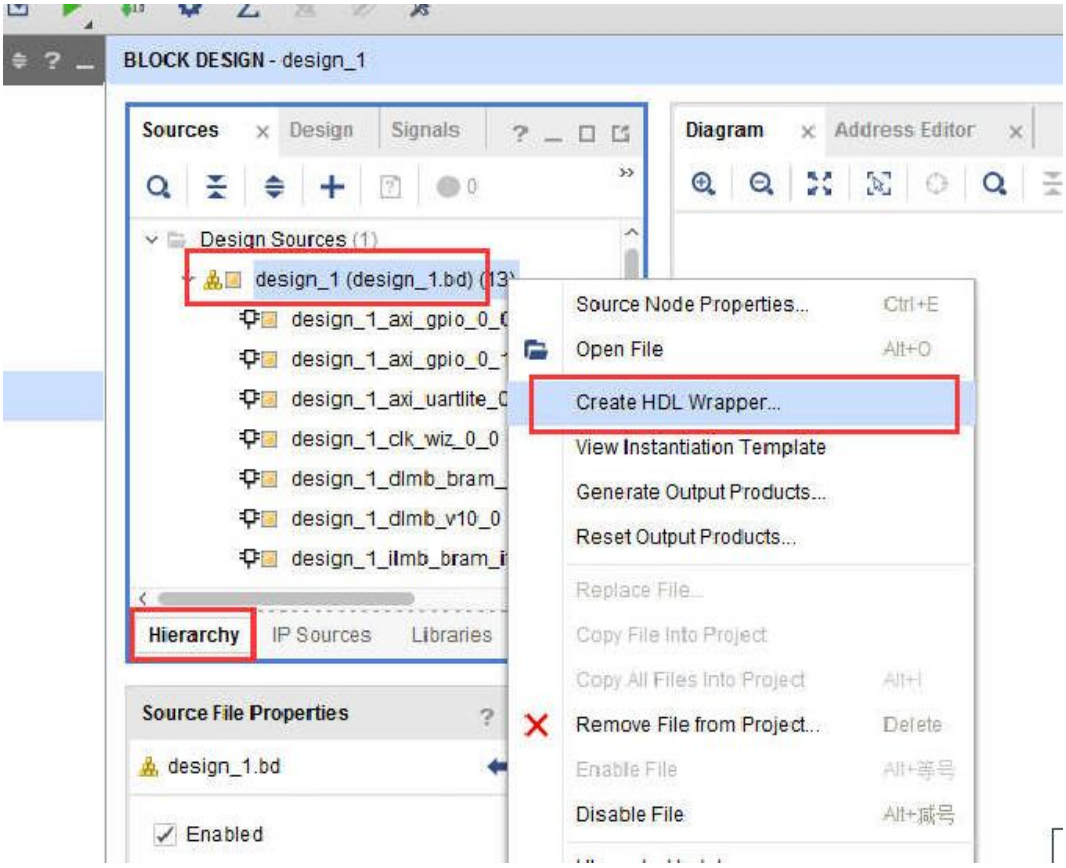
Part 1.1.7: Save and check for errors

35)Save the design and press "F6" to check the design

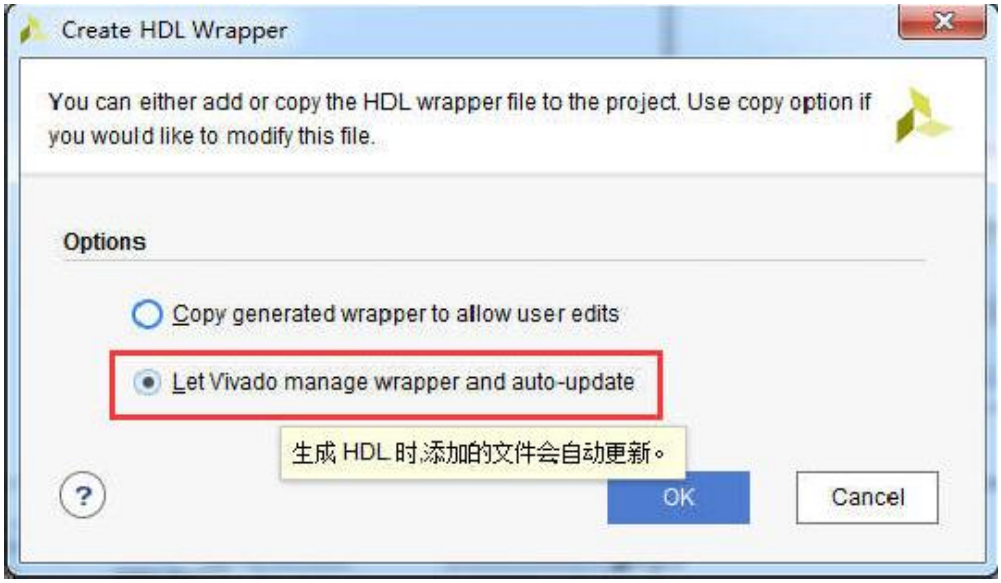


36)No errors or warnings in the design

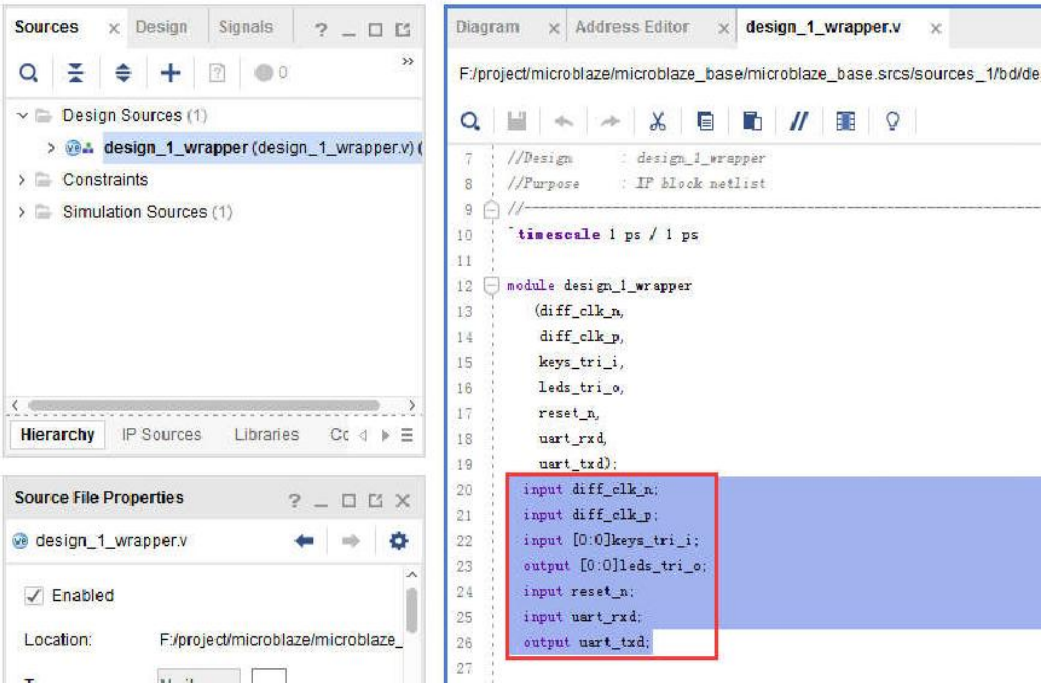
37)Select the "design_1.bd" file and right click on "Create HDL Wrapper..." to create a Verilog or VHDL top-level file.



38)Select "Let Vivado manage wrapper and auto-update" to have Vivado automatically update Verilog or VHDL files

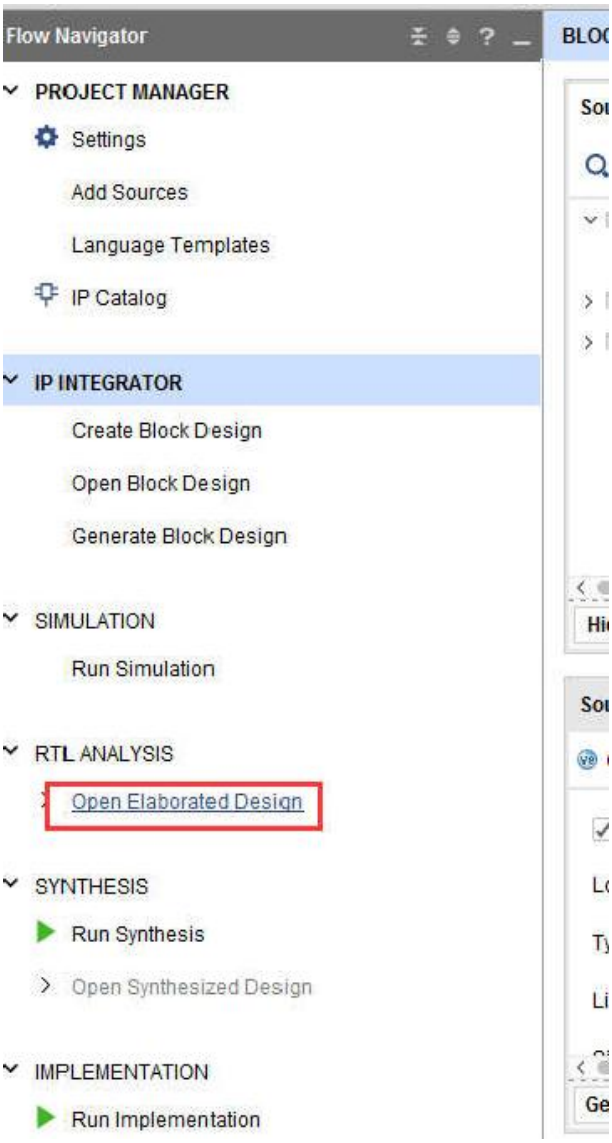


39)Double-click to open the generated "design_1_wrapper.v", you can see some ports, and assign IO to these ports below

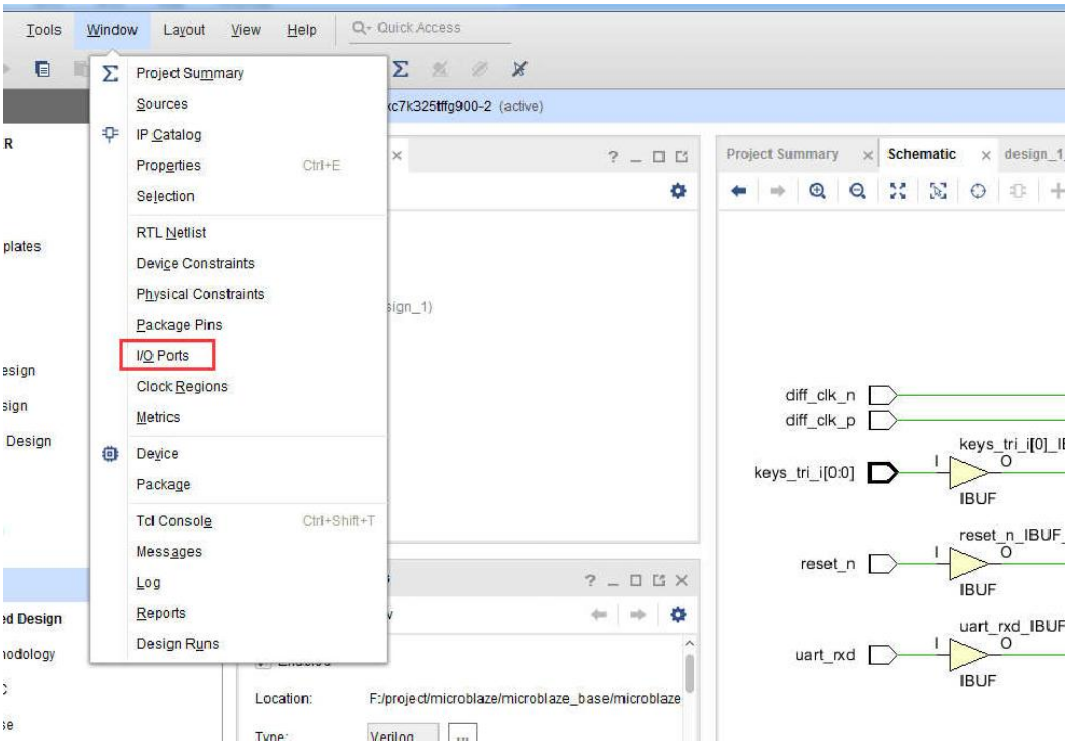


Part 1.1.8: Assign IO

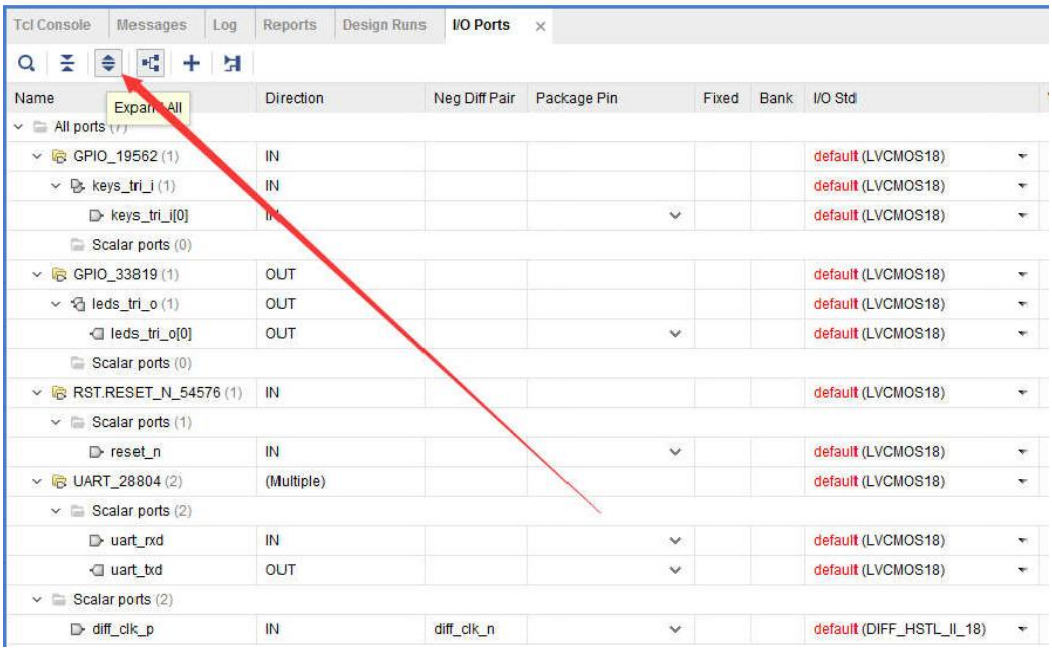
40) Click on "Open Elaborated Design" and click "OK" in the pop-up window.



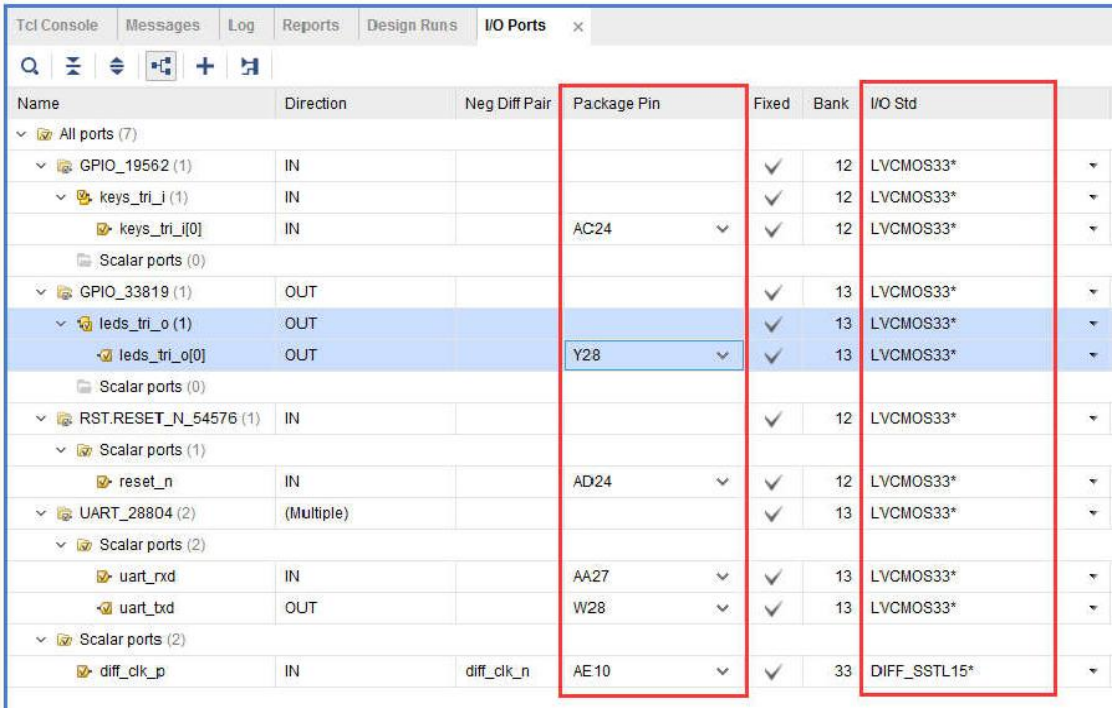
41)Click on "Windows -> I/O Ports"



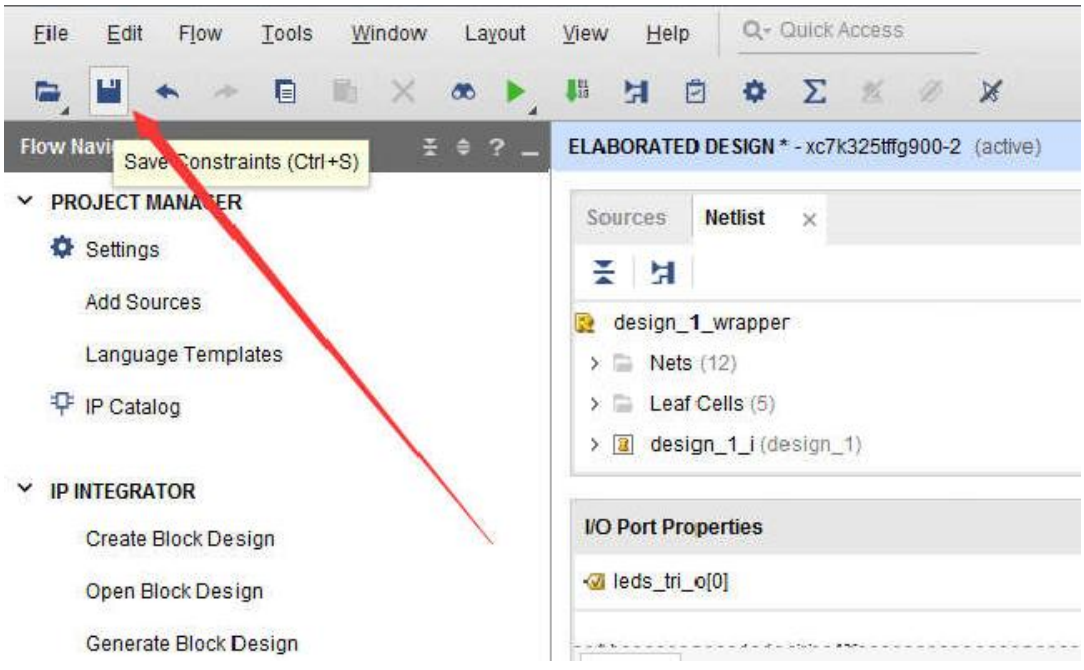
42) Click on “Expand All”, you can see that IO is not assigned, assign IO in the “Package Pin” column, and assign level standards in the “I/O Std” column.



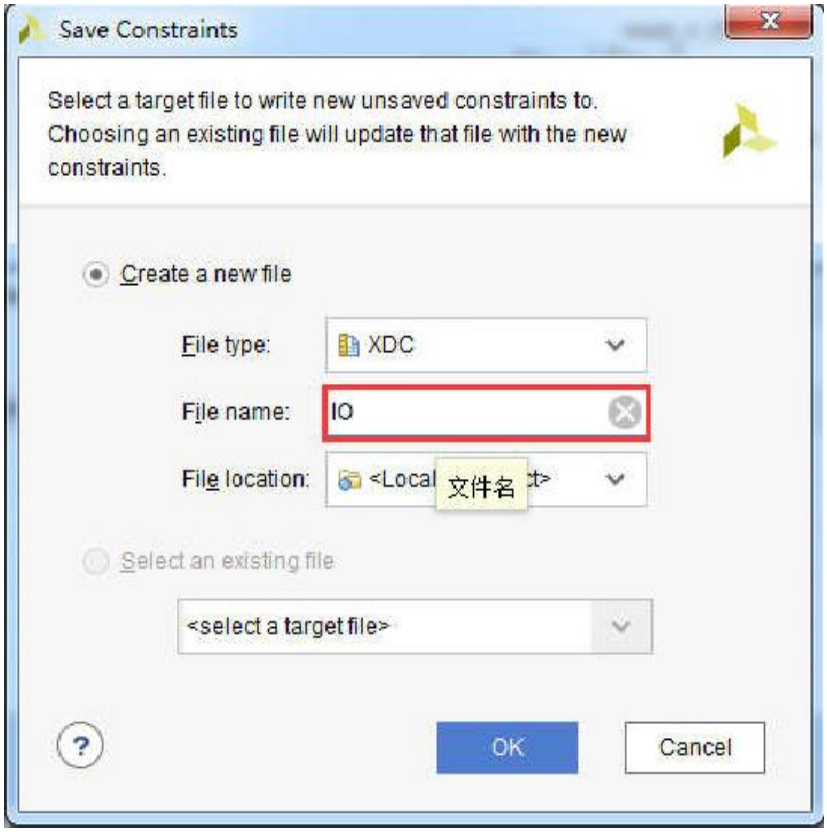
43) After allocation, you need to pay attention to the fact that the IO allocation must be allocated according to the actual FPGA development board.



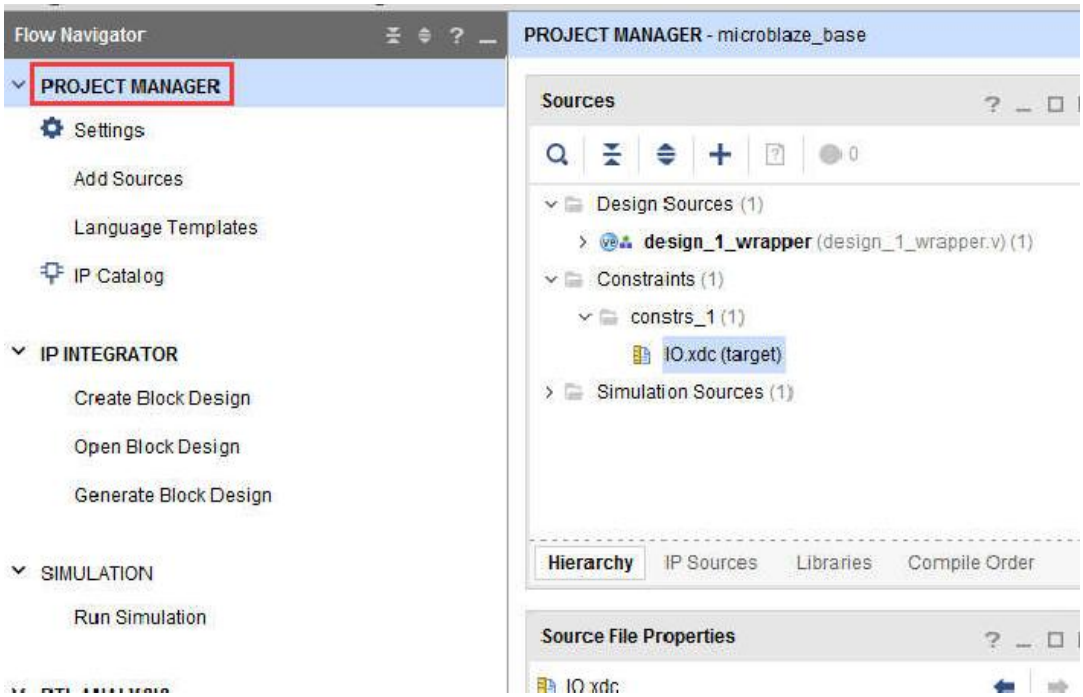
44)Click the save button



45)Prompt to create a new xdc file, fill in the IO here, click OK

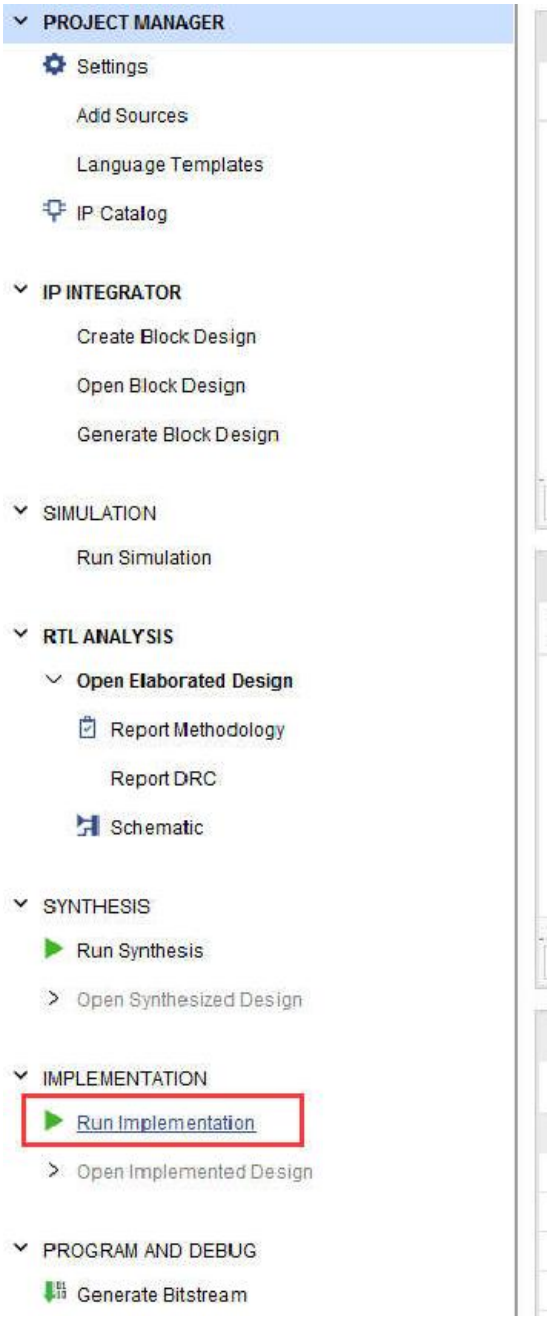


46) Click "PROJECT MANAGER" to return to the project management window, double-click the "IO.xdc" file to view the content.

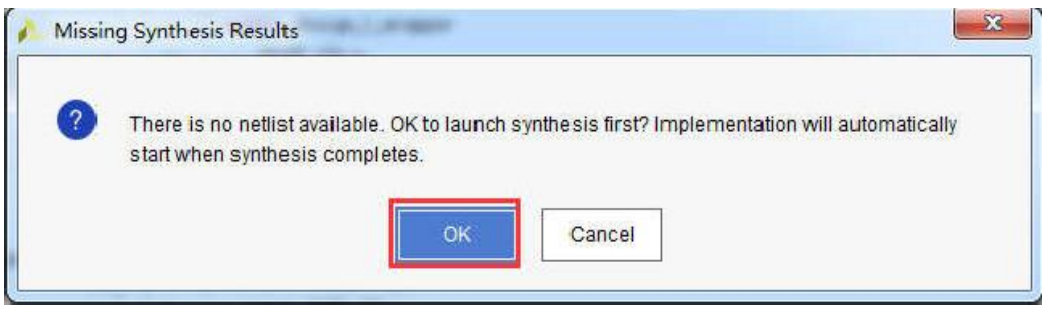


Part 1.1.9: Start to Implementation

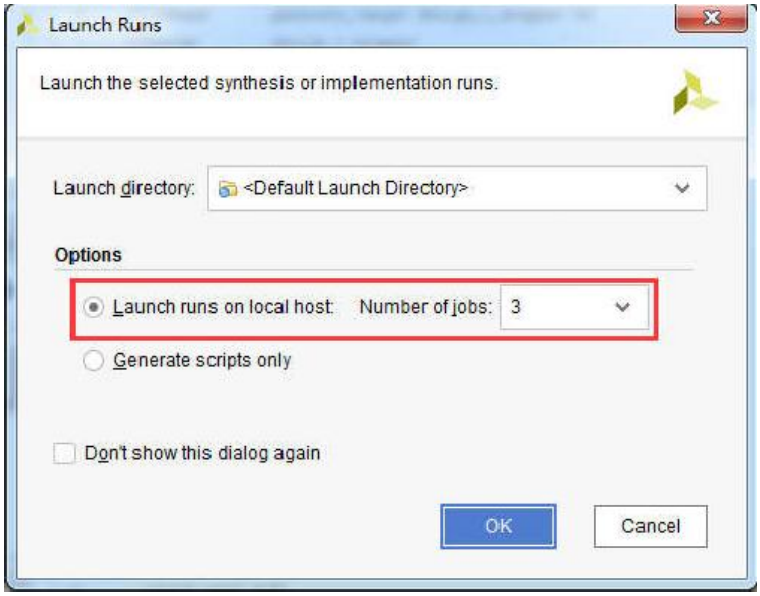
47)Click on "Run Implementation"



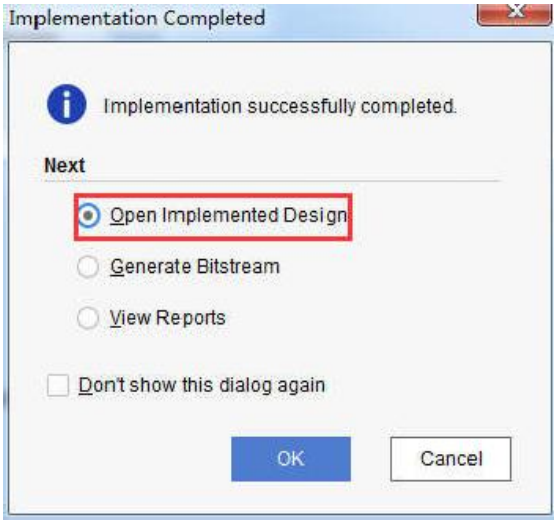
48)If the window below appears, click "OK"



49) There is a running path selection in the pop-up window. By default, "Number of jobs" can be selected according to its own CPU. The larger the number, the faster the compilation.

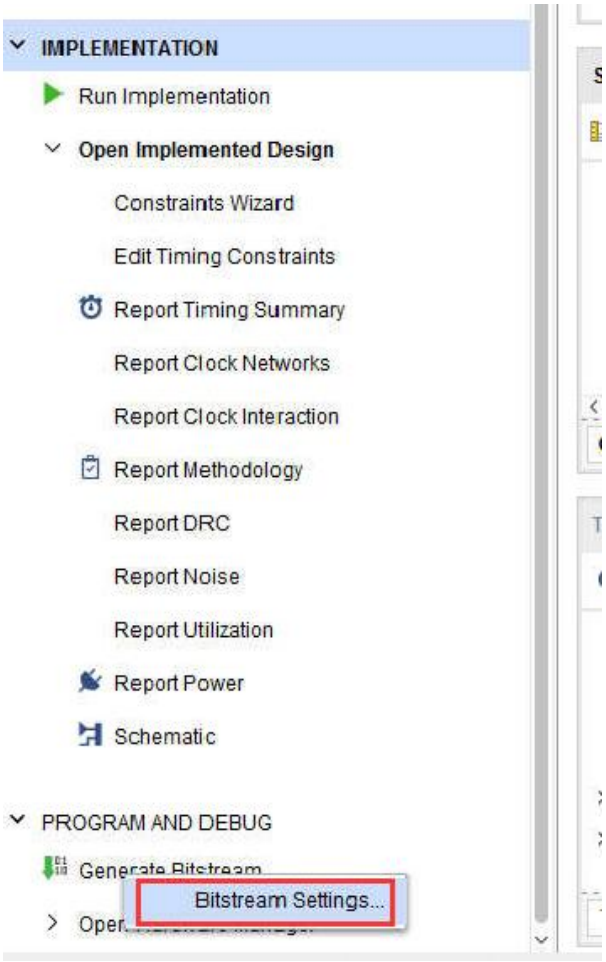


50) Select "Open Implemented Design" after compilation is complete

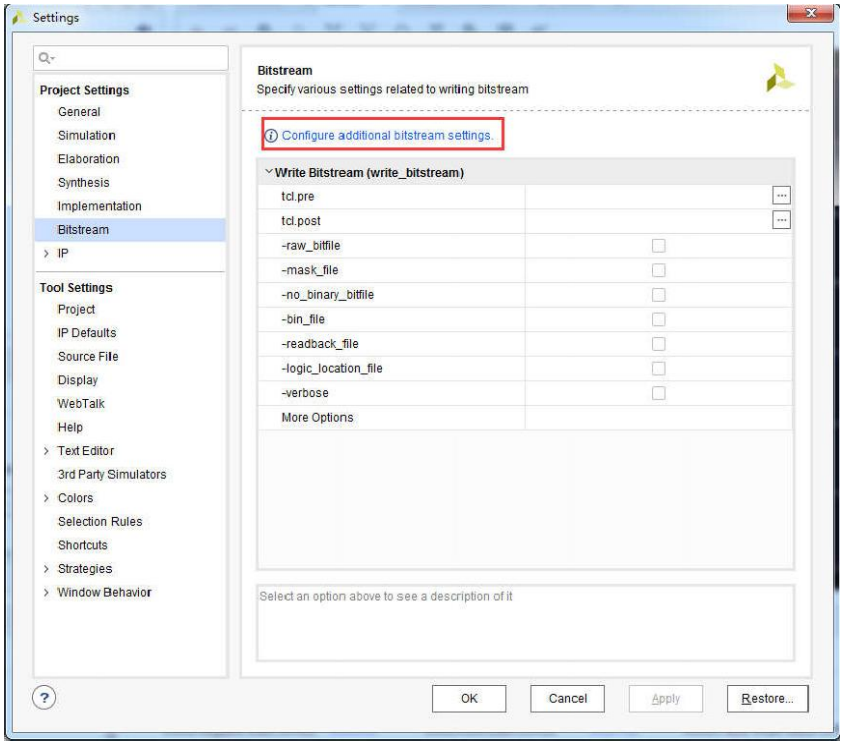


Part 1.1.10: Bitstream settings

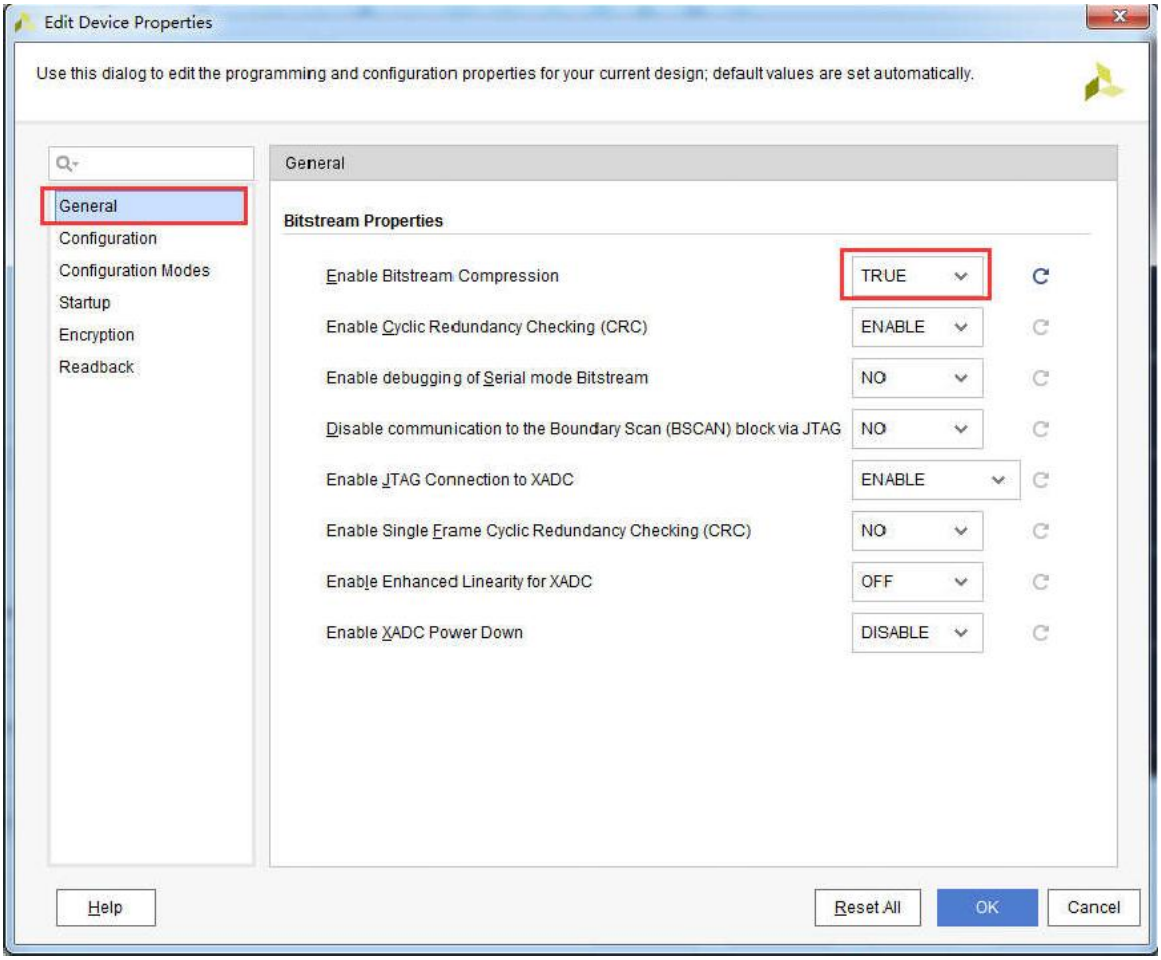
In the "PROGRAM AND DEBUG" option, right click on "Bitstream Setting..."



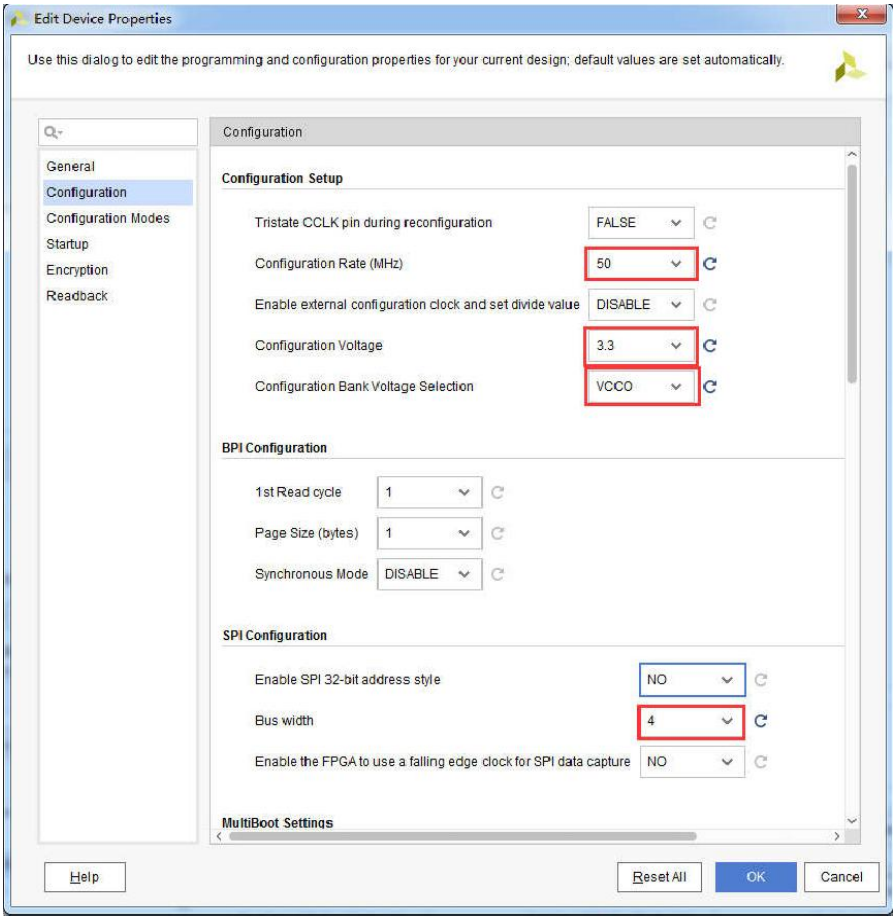
51)Click on "Configure additional bitstream settings" in the "Setting" window.



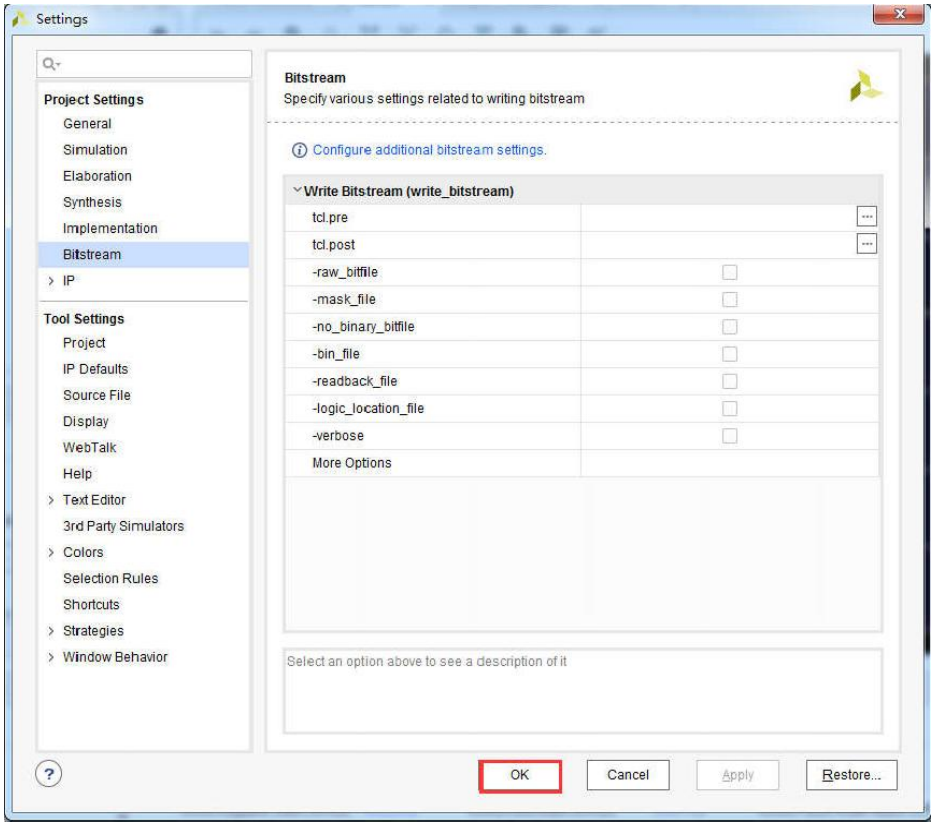
52) In the "General" tab, "Enable Bitstream Compression" selects "TRUE", compresses the Bitstream file, and makes the Bitstream file smaller.



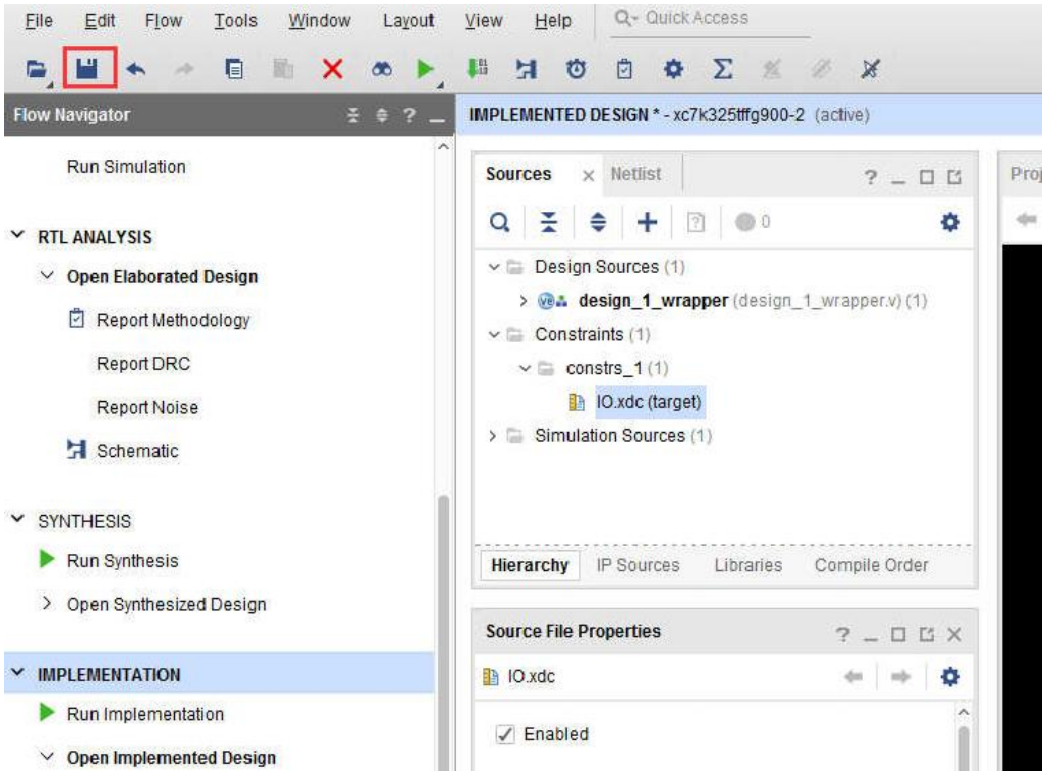
53) On the Configuration tab, modify the Configuration Rate (MHz) to 50. If the value is too large, the configuration file may not be loaded correctly. If the value is too small, the loading time may be too long. “Configuration Voltage” selects 3.3, “Configuration Bank Voltage Selection” selects VCCO. These configurations are hardware design decisions and are not explained here. In the "SPI Configuration" select "Bus width" to 4, the other keep the default, click "OK"



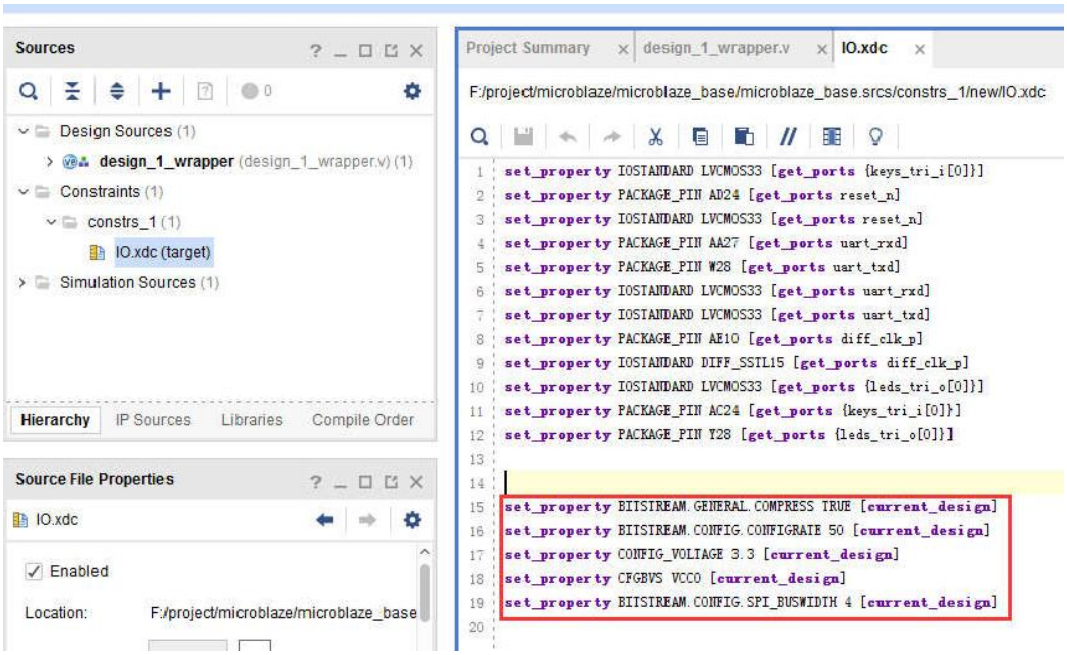
54)Click "OK" to complete the Bitstream setup



55) Click the Save button to save the configuration information to the xdc file.

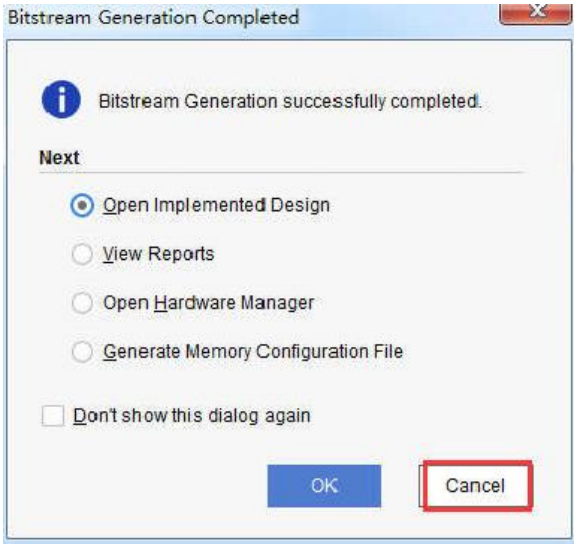


56) Open the xdc file to find out more about Bitstream configuration related code.



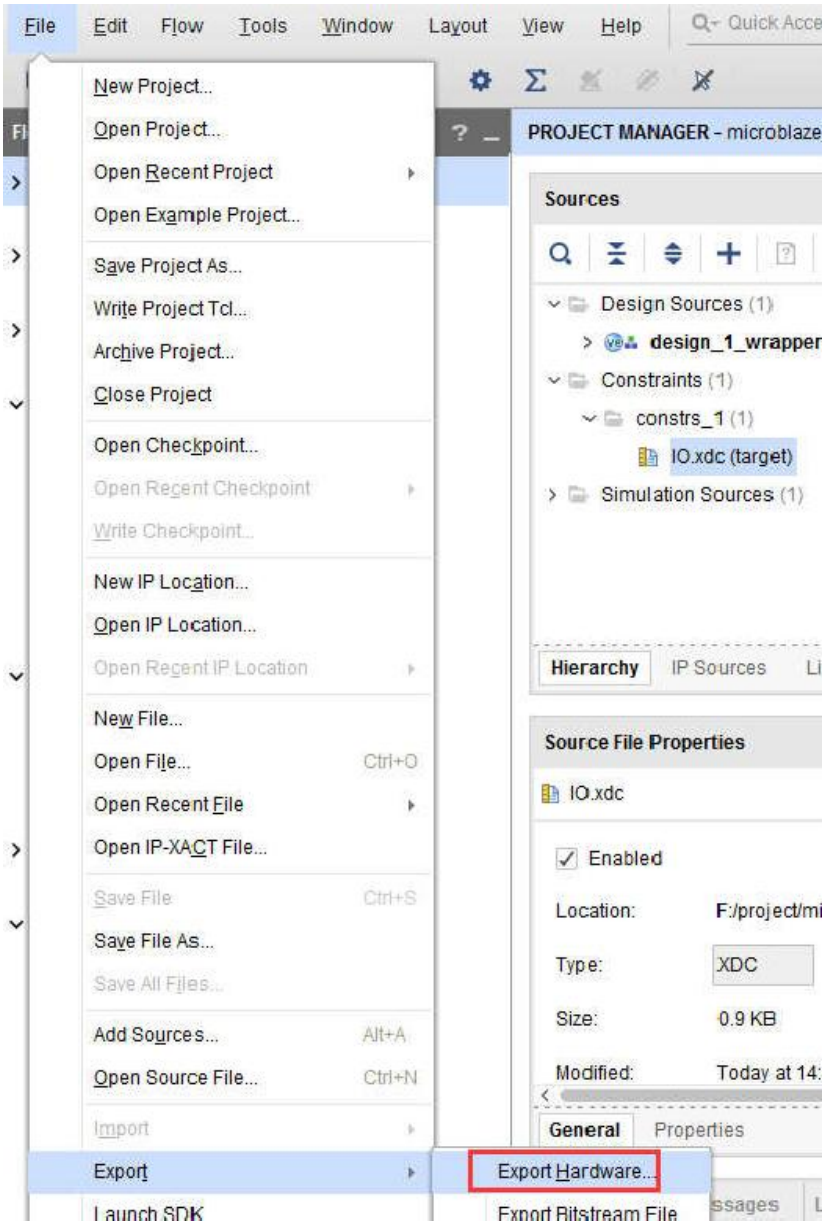
57) Click "Generate Bitstream" to generate a bit stream file.

58)After the generation, click "Cancel", do nothing

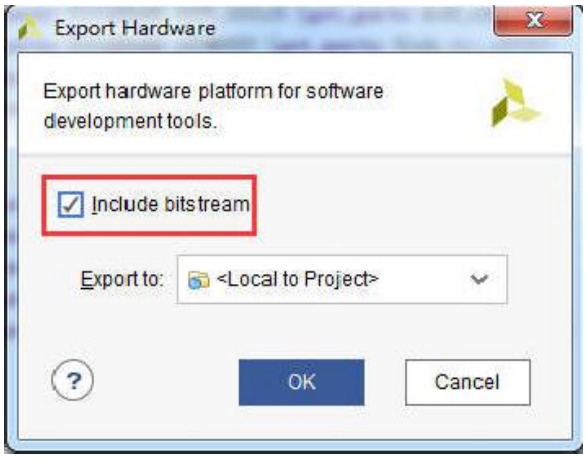


Part 1.1.11: Export hardware

59)Export hardware in "File -> Export -> Export Hardware..."

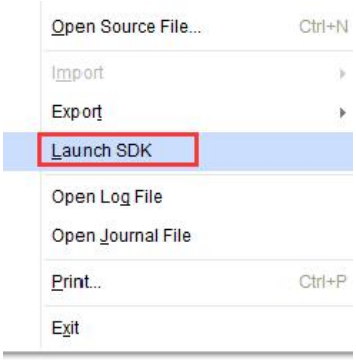


60) Select "Include bitstream" in the pop-up window to include the Bitstream file.

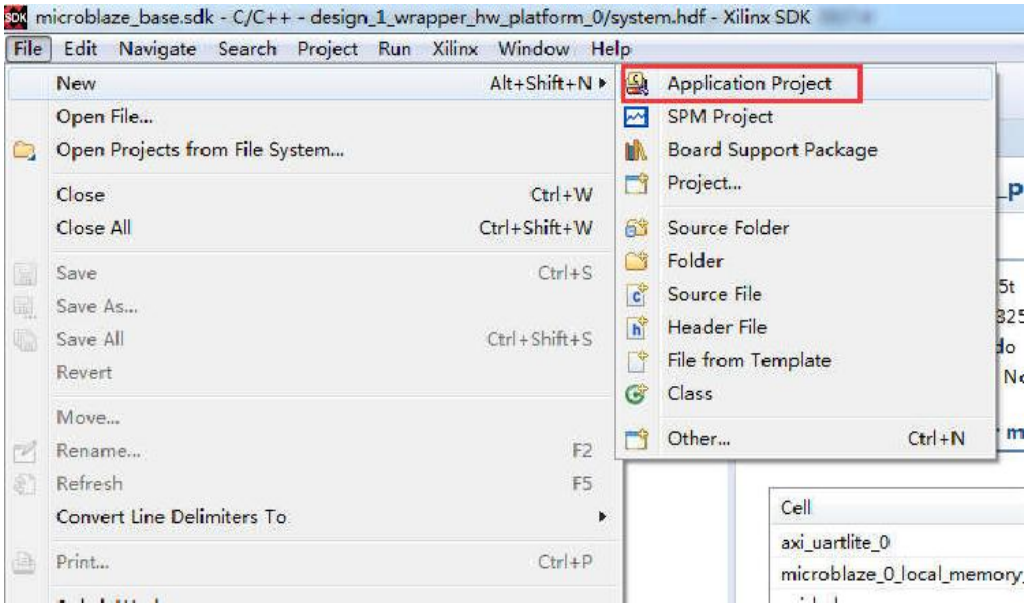


Part 1.2: SDK development "HelloWorld"

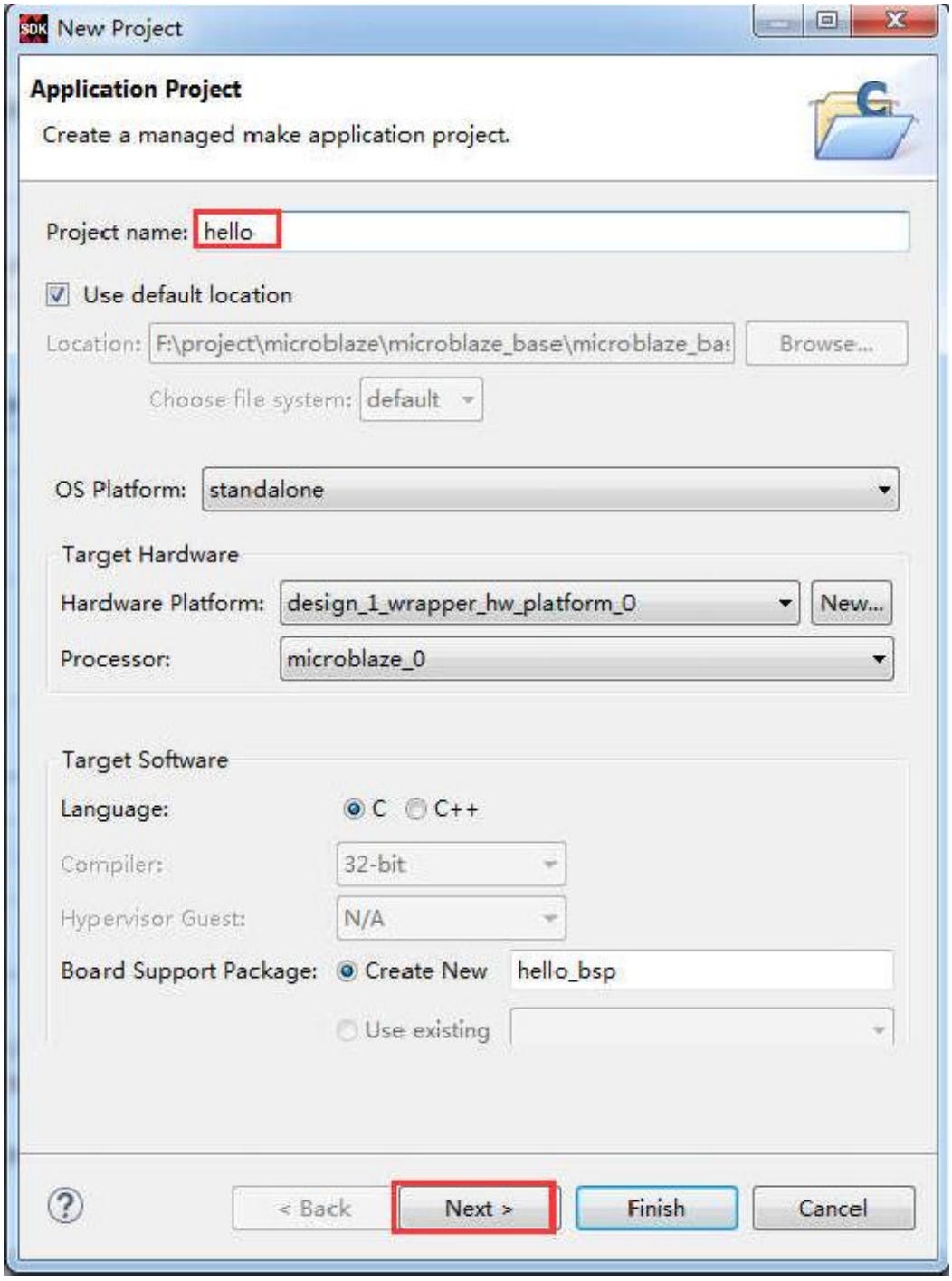
1) Click File→Launch SDK to enter the SDK operation.



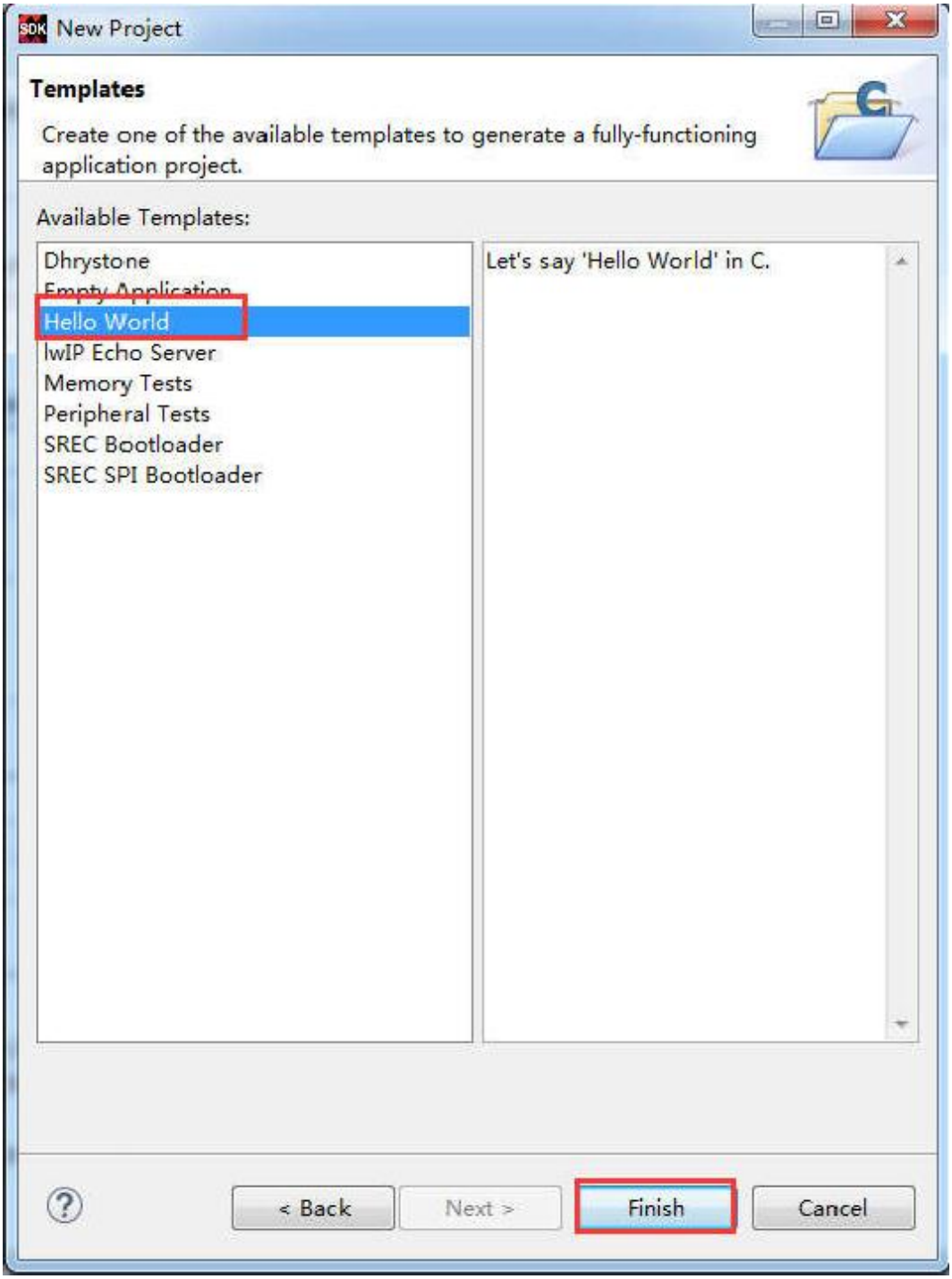
2) Create a new project



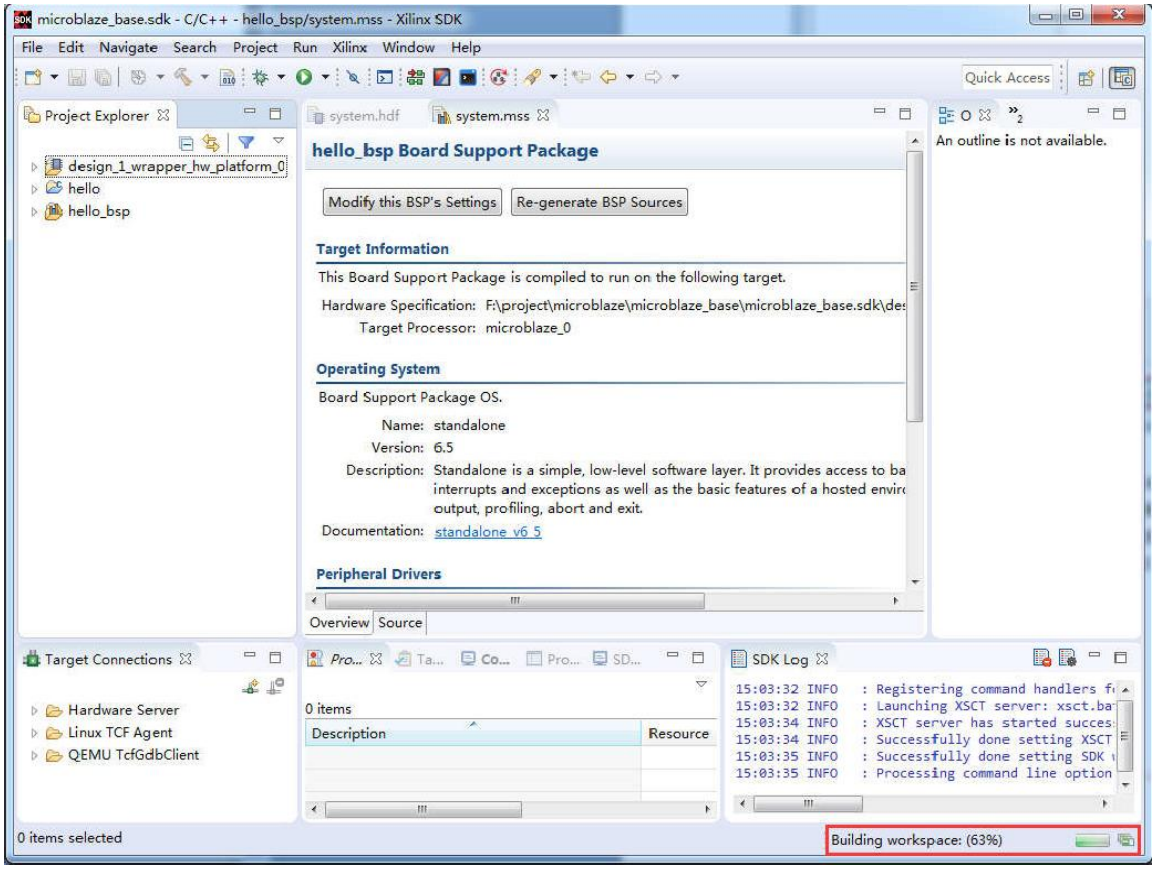
3) Set the project name to "hello" and click "Next"



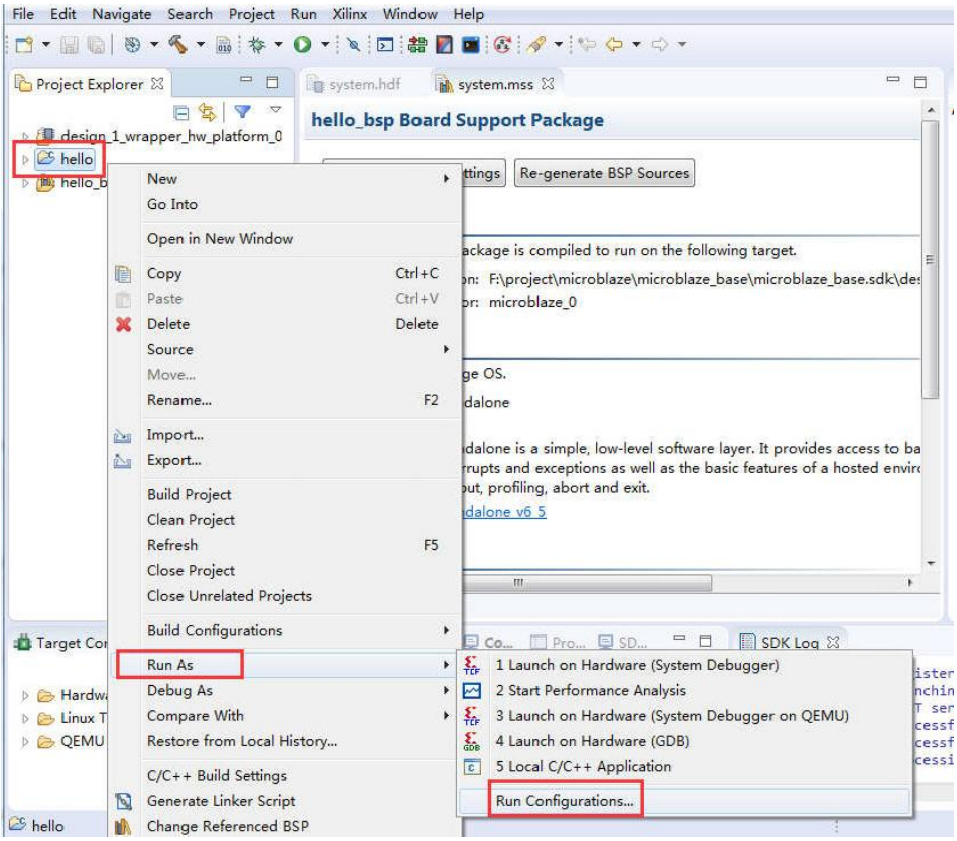
4) Select "Hello World" for the template and click "Finish"



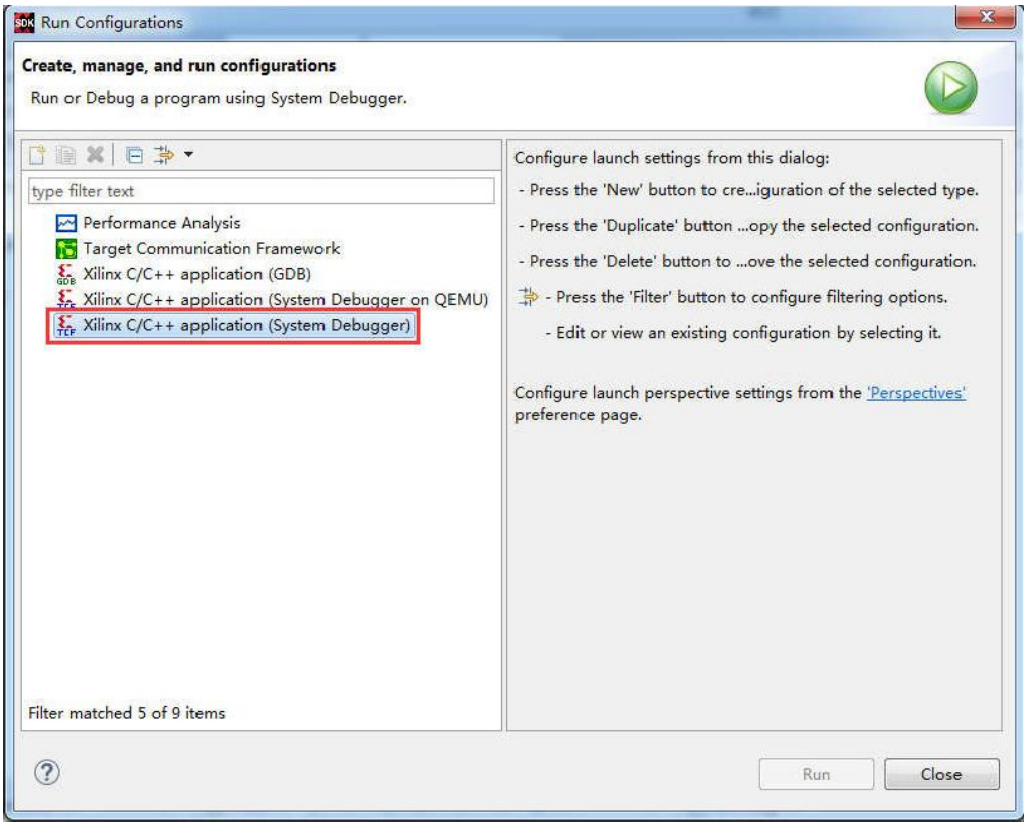
5) Automatically compile after the project is built, and wait for the compilation to complete before debugging and running.



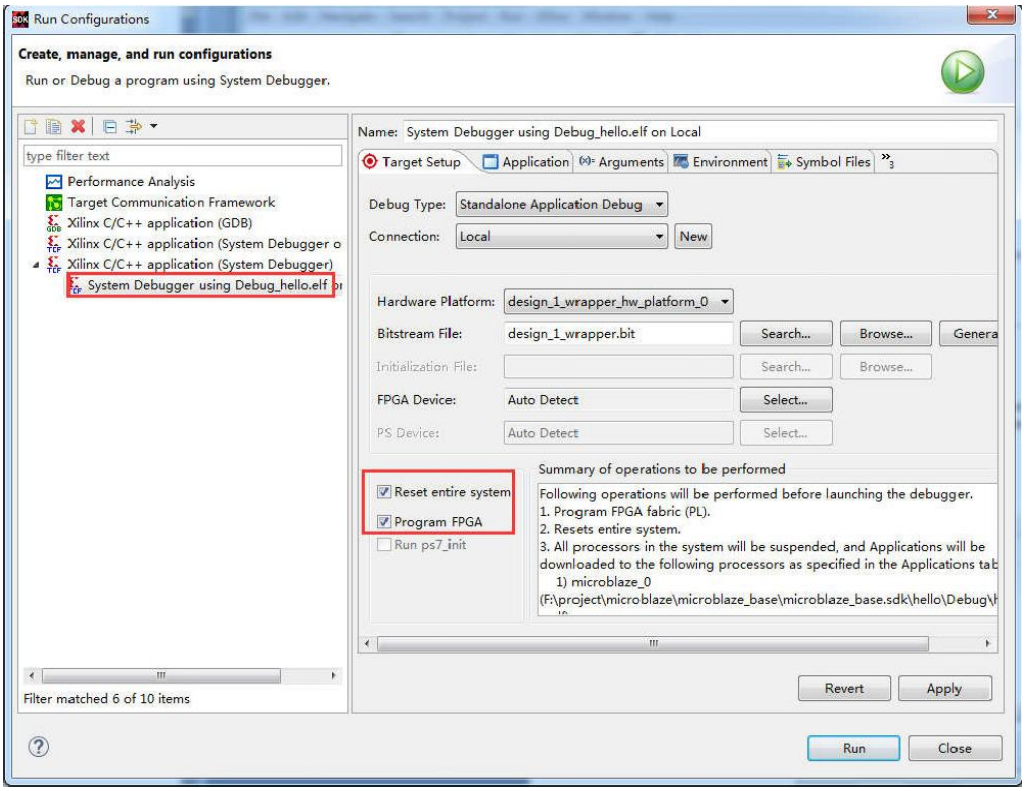
Select the "Hello" project and right click on "Run As -> Run Configuration..."



6) Double click on "Xilinx C/C++ application(System Debugger)"

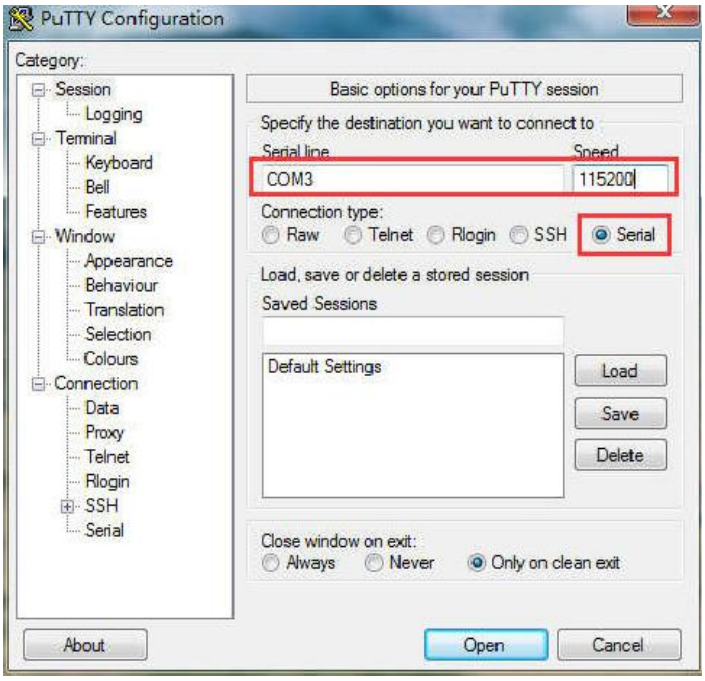


7) Check "Reset entire system" and "Program FPGA" to reset the entire system and download the FPGA

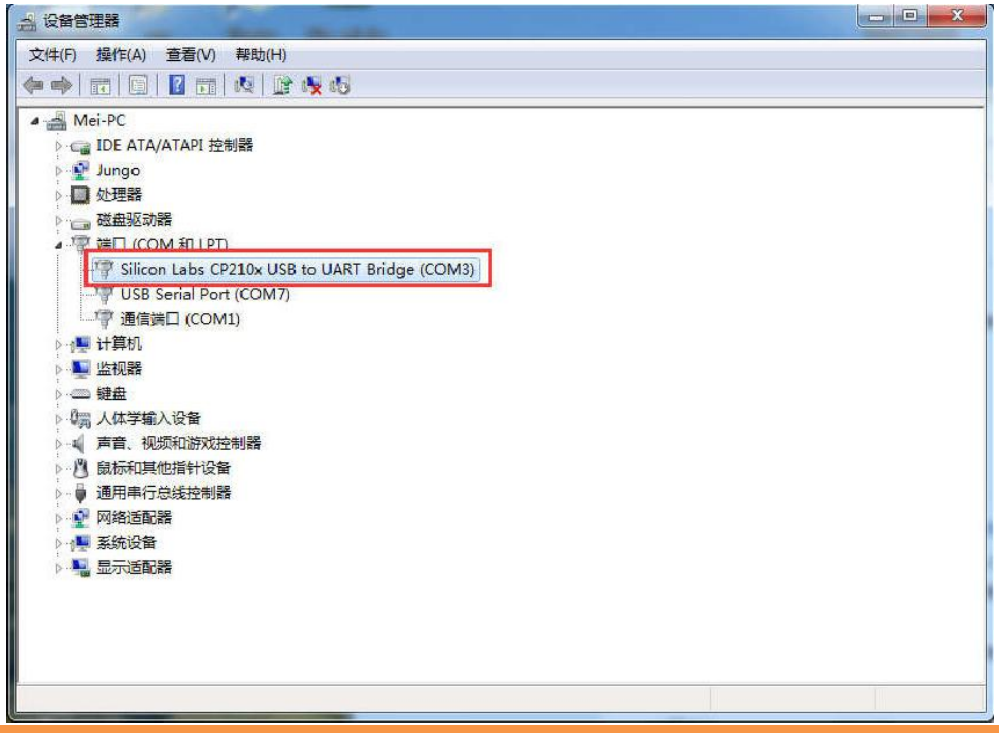


8) Connect the JTAG cable to the development board, UART USB cable to PC

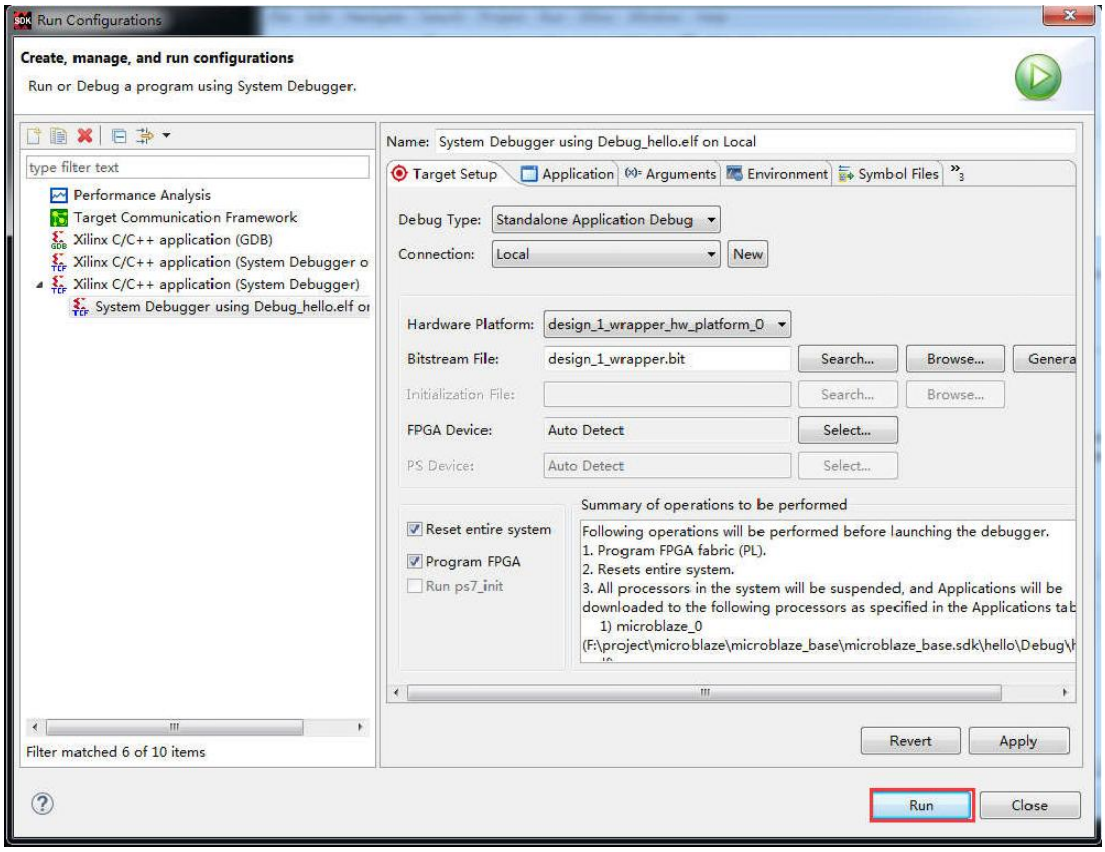
9) Using PuTTY software as a serial terminal debugging tool, PuTTY is an installation-free small software.



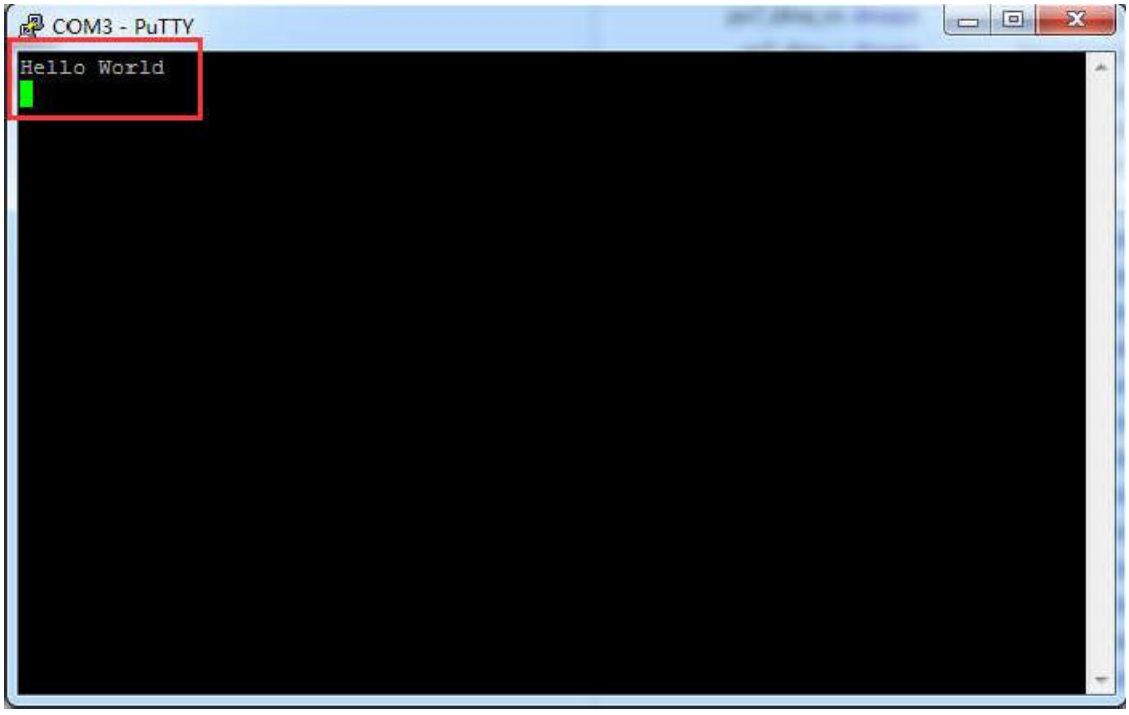
10) Select Serial, Serial line to fill in COM3, Speed fill in 115200, COM3 serial port number is filled in according to the display in the device manager, click "Open"



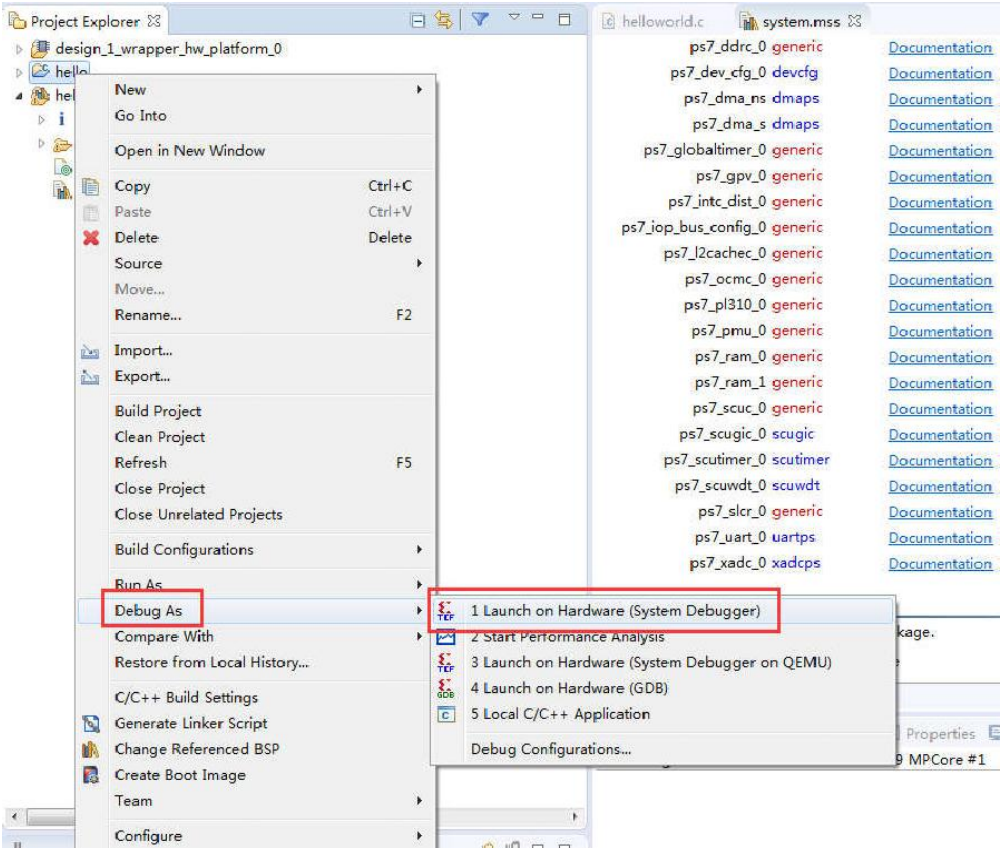
11)Click “Run”



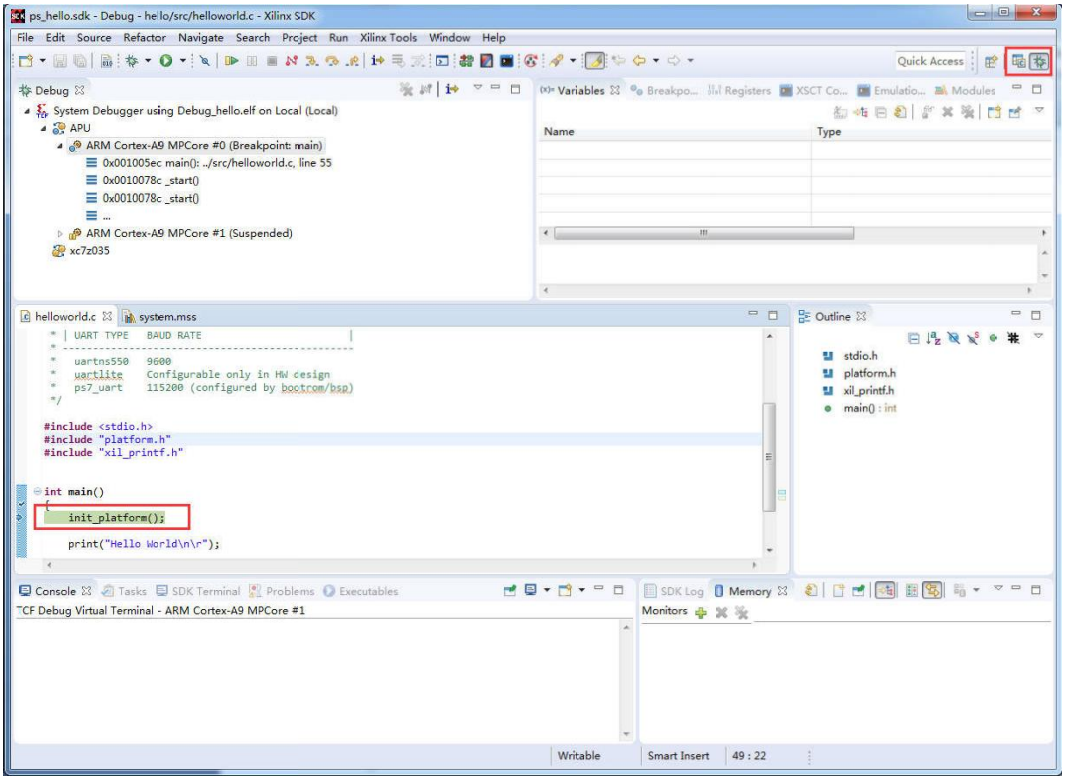
12)"Hello World" is displayed



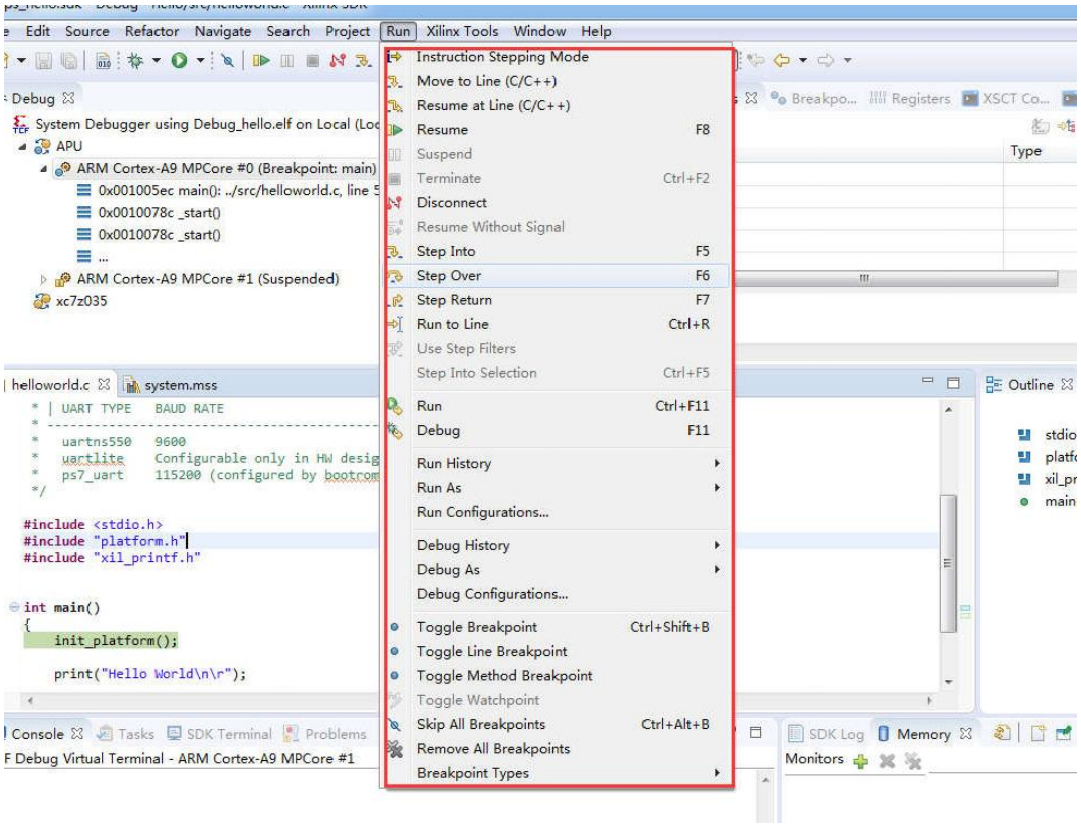
13)In addition to "Run As", you can also "Debug As", which can set breakpoints and single step



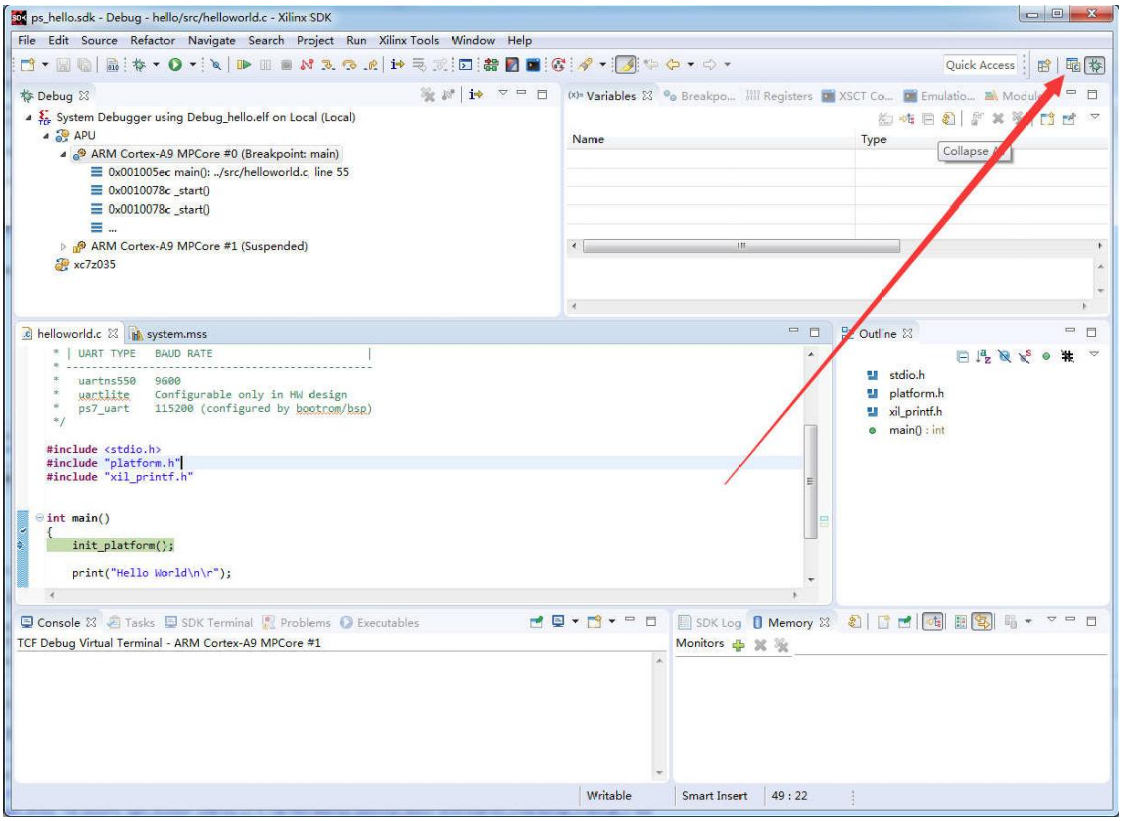
14)Enter Debug mode



15)Like other C language development IDEs, you can run it step by step, set breakpoints, etc.

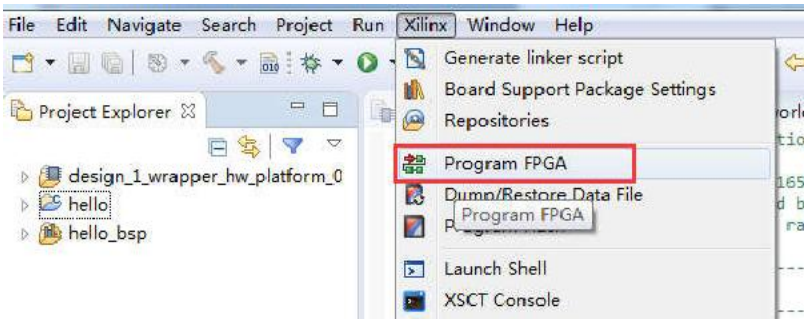


16)The IDE mode can be switched in the upper right corner

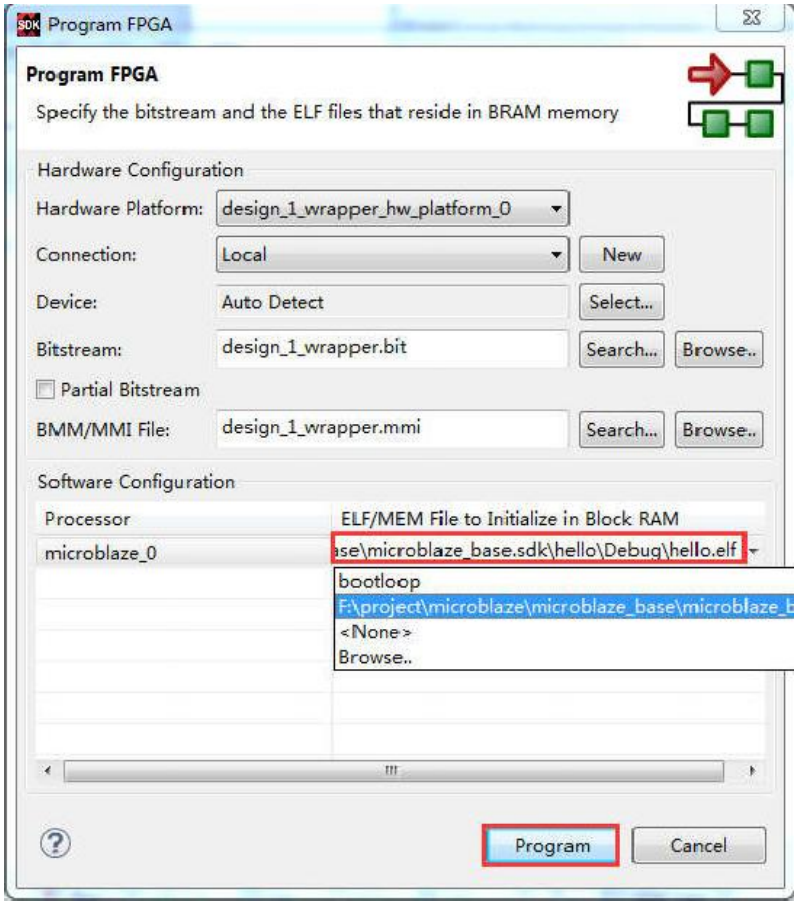


Part 1.3: Programming program to Flash

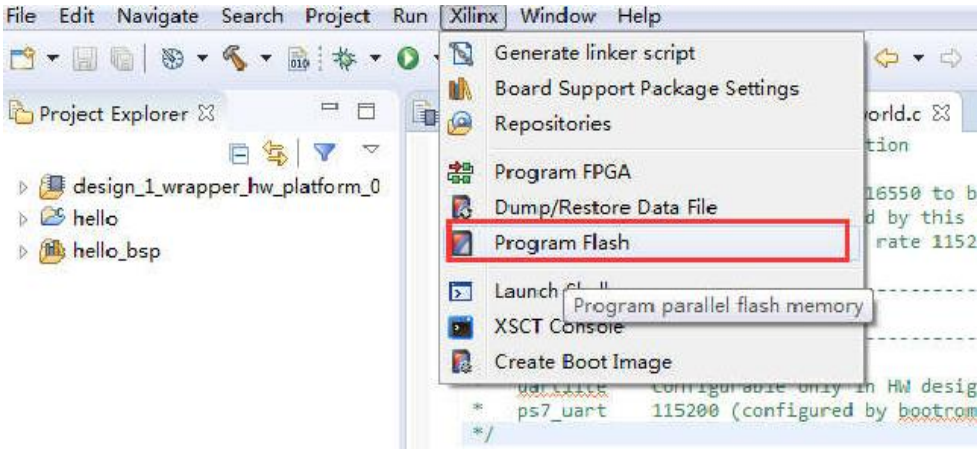
1) Select "Xilinx -> Program FPGA"



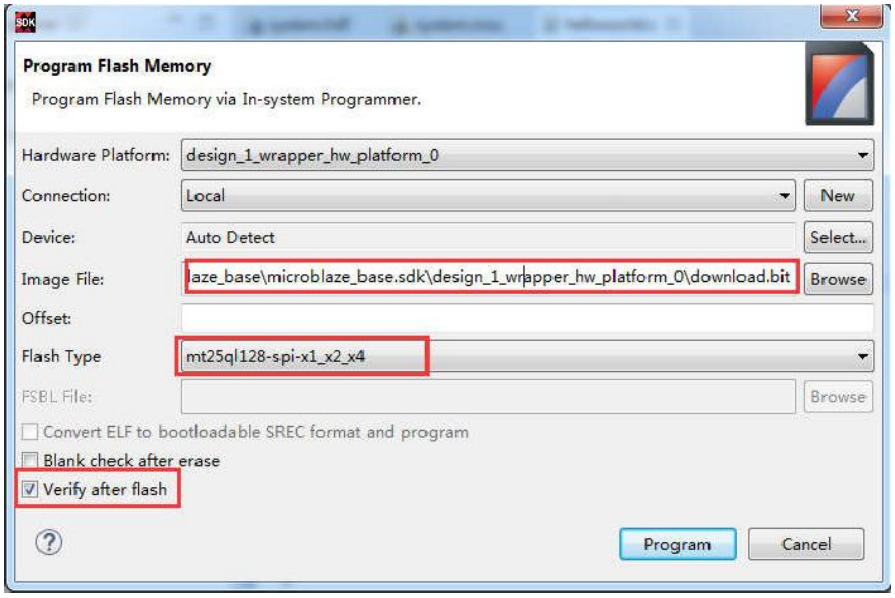
2) In the "microblaze_0" of the "Software Configuration" option of the pop-up window, select the elf file of the project "\microblaze_base.sdk\hello\Debug\hello.elf", merge the elf file and the bit file, and click "Program".



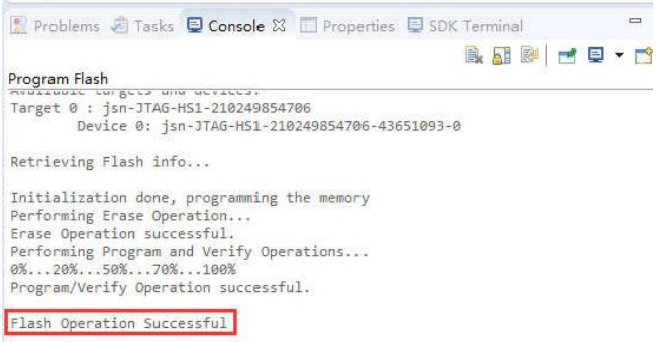
3) Click on "Xilinx -> Program Flash"



4) "Image File" selects the "download.bit" file generated in the previous step, located in the directory "microblaze_base.sdk/design_1_wrapper_hw_platform_0", "Flash Type" selects "mt25ql128-spi-x1_x2_x4", check "Verify after flash", click " Program" starts to program Flash



5) After the programming is completed, it is best to unplug the JTAG cable and restart it.



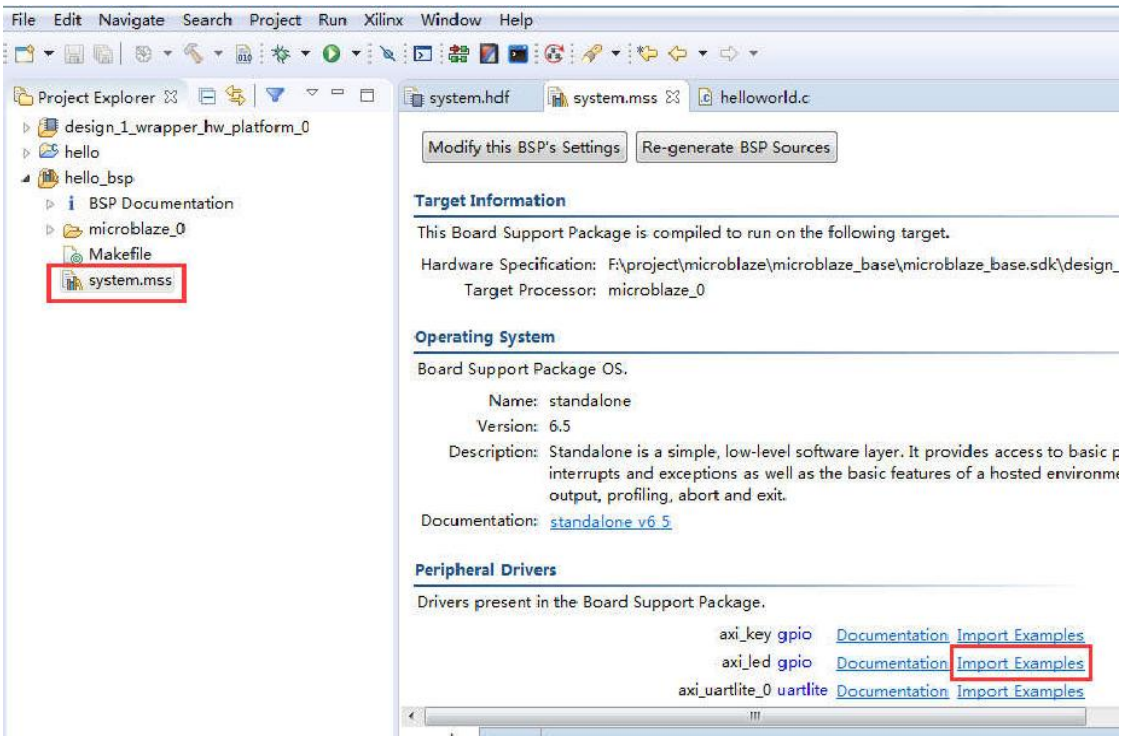
Part 2: SDK programming development

The previous chapters showed how to build a simple Microblaze hardware system and do a simple test with the Helloworld program. This chapter learns how to use the Xilinx BSP to write SDK programs, how to find related documents, but if you don't have a good C language foundation, you can't Good to master the contents of this chapter, it is recommended that learners master the C language to learn.

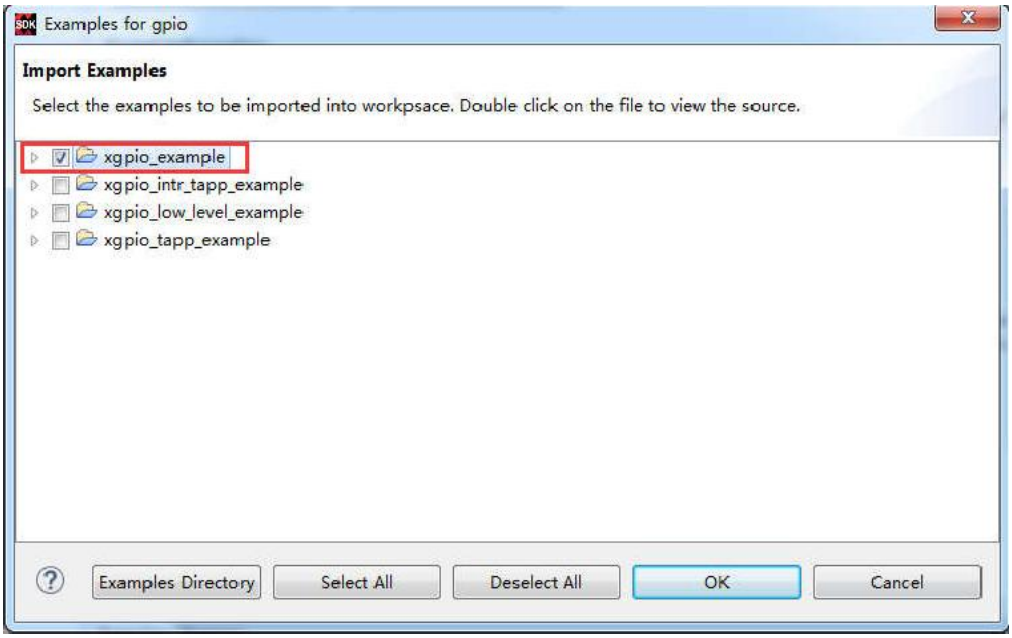
Part 2.1: GPIO control lighting LED

If we write a program that uses GPIO to drive an LED, an engineer who has learned a microcontroller might think of a library function, or think of using register direct control. Let's take a look at how to use the SDK?

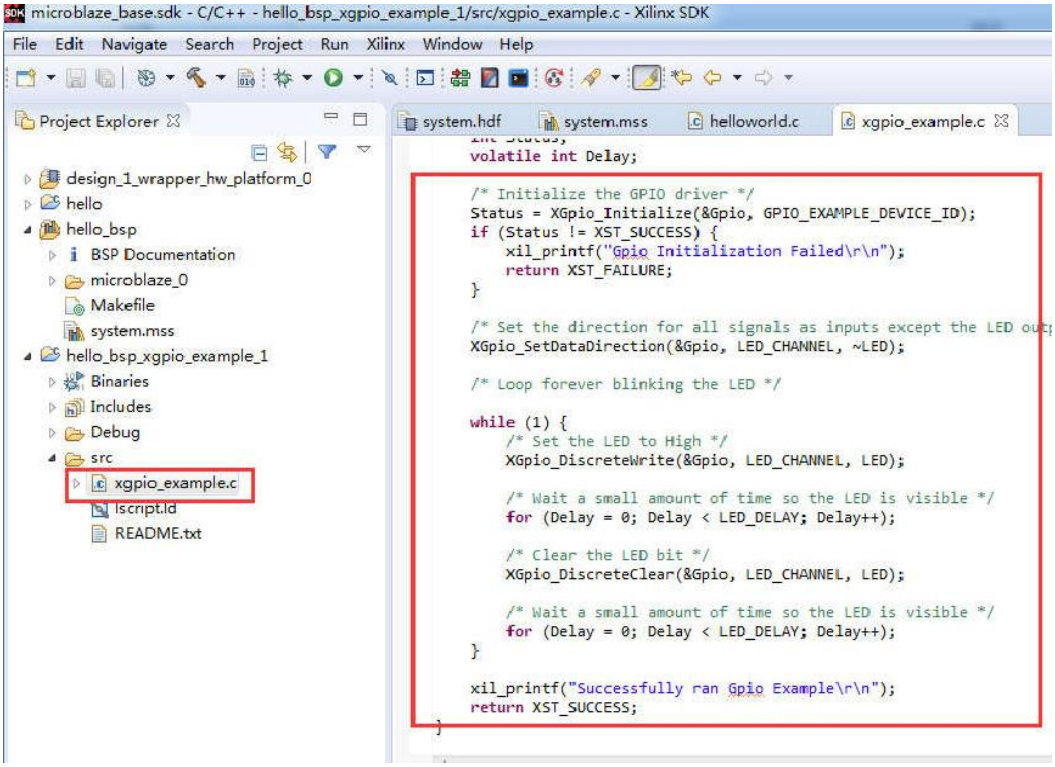
1) Double-click the "system.mss" file in the BSP directory and click "Import Examples" next to axi_led



2) Select "xgpio_example"



3) A "hello_bsp_xgpio_example_1" project is generated. Double-click "xgpio_example.c" to view the source code. You can judge whether you need to continue learning C language by reading the code. This routine is easy to understand. You can directly judge the function function by function name



4) The code needs to be modified to run. The code defines

"#define GPIO_EXAMPLE_DEVICE_ID XPAR_GPIO_0_DEVICE_ID" to determine which is the GPIO of the led and which is the GPIO of the key. These IDs are defined in the header file "xparameters.h".

```
/************************** Constant Definitions *****************************/

#define LED 0x01   /* Assumes bit 0 of GPIO is connected to an LED  */

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define GPIO_EXAMPLE_DEVICE_ID  XPAR_GPIO_0_DEVICE_ID

/*
 * The following constant is used to wait after an LED is turned on to make
 * sure that it is visible to the human eye.  This constant might need to be
 * tuned for faster or slower processor speeds.
 */
#define LED_DELAY     10000000

/*
 * The following constant is used to determine which channel of the GPIO is
 * used for the LED if there are 2 channels supported.
```

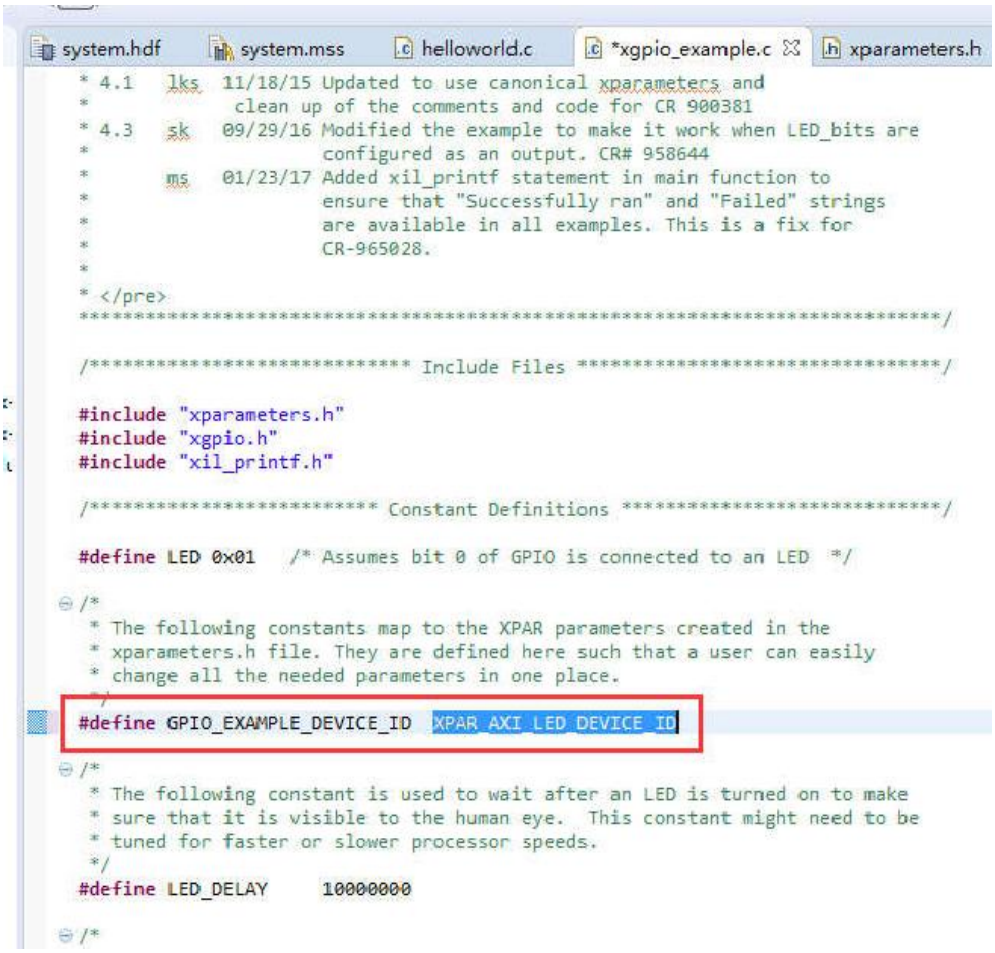
5) Move the cursor to "xparameters.h" and press the "F3" button to quickly open the header file. The same method can find a variable definition.

```
 *
 * </pre>
 ***************************************************

 /*************************** Include Files **
-xilinx-
-xilinx-
elf/inclu
 #include "xparameters.h"
 #include "xgpio.h"
 #include "xil_printf.h"

 /************************** Constant Definitior

 #define LED 0x01   /* Assumes bit 0 of GPIO is

 /*
  * The following constants map to the XPAR para
  * xparameters.h file. They are defined here su
```

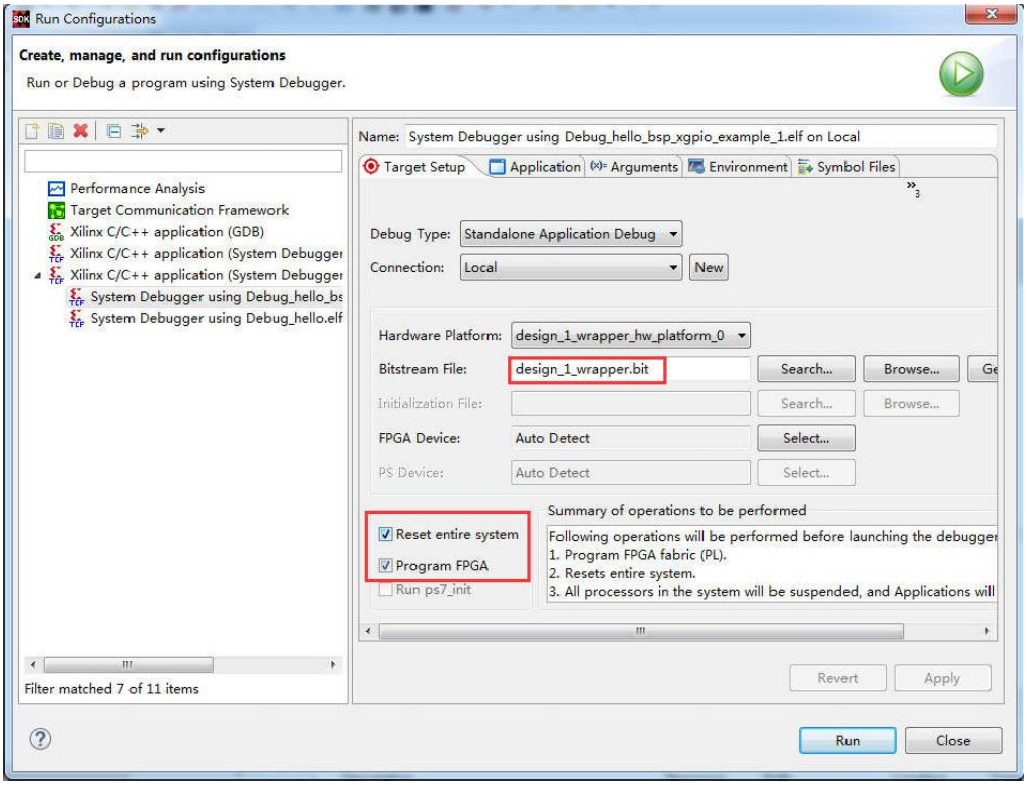
6) The ID macro found for the LED in the "xparameters.h" file is defined as "XPAR_AXI_LED_DEVICE_ID"

```
/* Canonical definitions for peripheral AXI_KEY */
#define XPAR_GPIO_0_BASEADDR 0x40010000
#define XPAR_GPIO_0_HIGHADDR 0x4001FFFF
#define XPAR_GPIO_0_DEVICE_ID XPAR_AXI_KEY_DEVICE_ID
#define XPAR_GPIO_0_INTERRUPT_PRESENT 0
#define XPAR_GPIO_0_IS_DUAL 0

/* Canonical definitions for peripheral AXI_LED */
#define XPAR_GPIO_1_BASEADDR 0x40000000
#define XPAR_GPIO_1_HIGHADDR 0x4000FFFF
#define XPAR_GPIO_1_DEVICE_ID XPAR_AXI_LED_DEVICE_ID
#define XPAR_GPIO_1_INTERRUPT_PRESENT 0
#define XPAR_GPIO_1_IS_DUAL 0
```

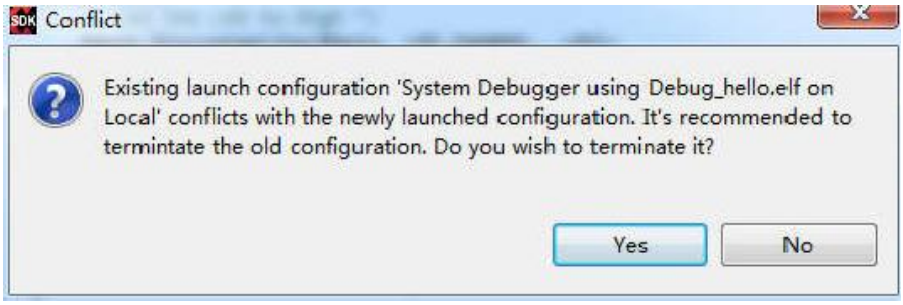
7) Modify the GPIO macro in the xgpio_example.c file to be defined as "#define GPIO_EXAMPLE_DEVICE_ID XPAR_AXI_LED_DEVICE_ID"



8) Run the configuration, you need to pay attention to Bitstream File, do not choose the wrongThe default may be the previously generated download.bit. This file is incorrect. To select the design_1_wrapper.bit file.



9) If the following window prompt appears, select "Yes"

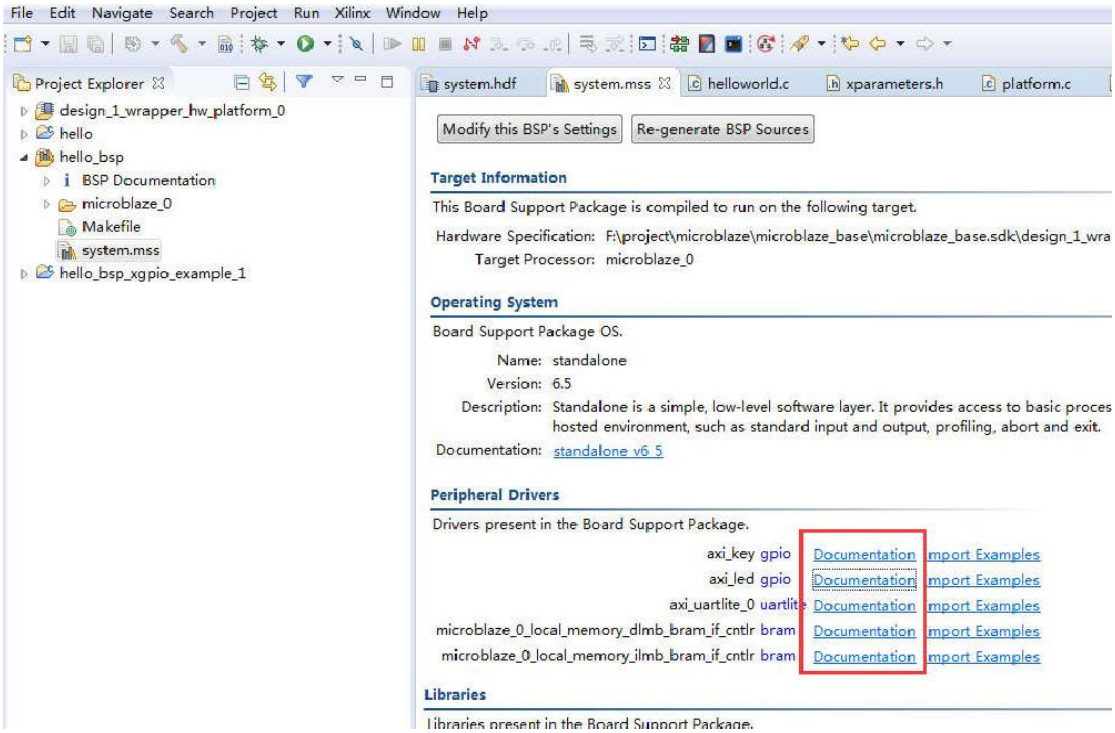


10) After running, you can see that the LED is blinking continuously.

Part 2.2: SDK software development skills

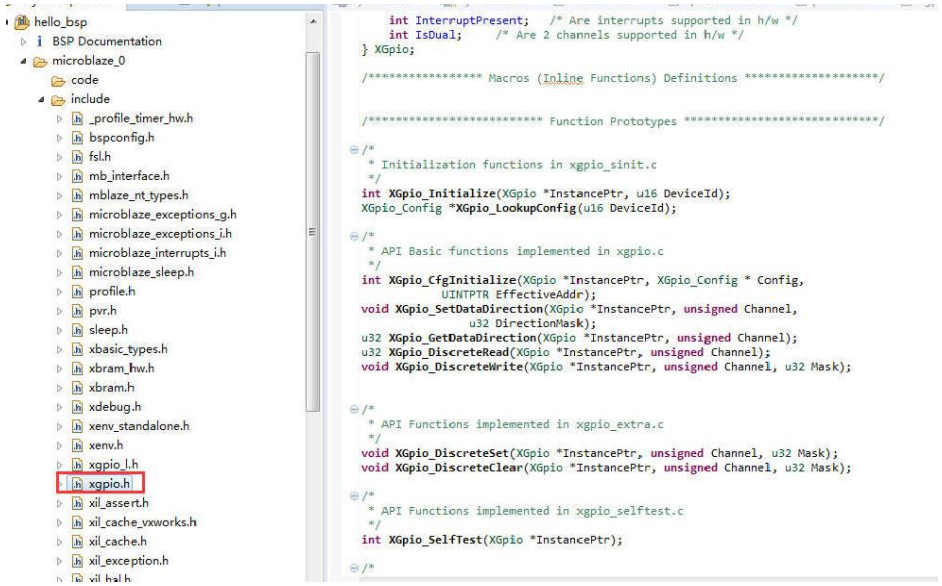
Part 2.2.1: Get help documentation

1) Double-click the system.mss file in the bsp directory to select the IP core document you want to read.

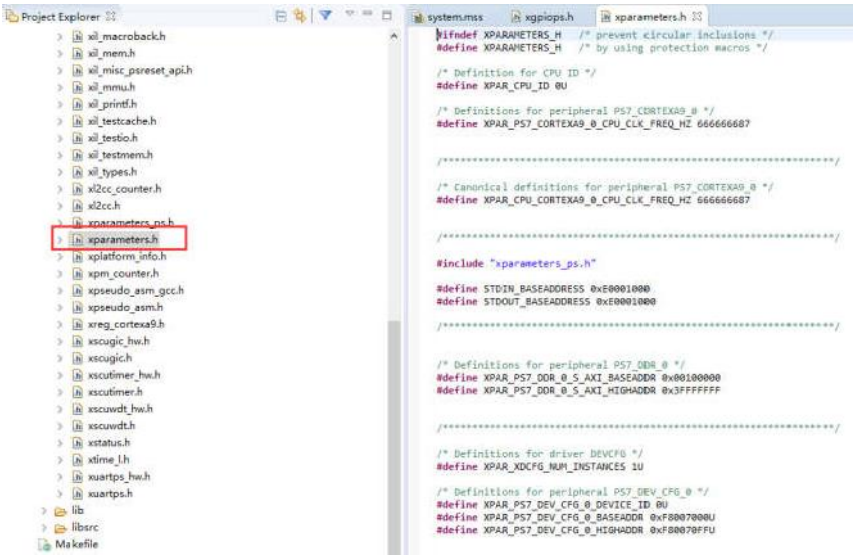


Part 2.2.2: View API function prototypes

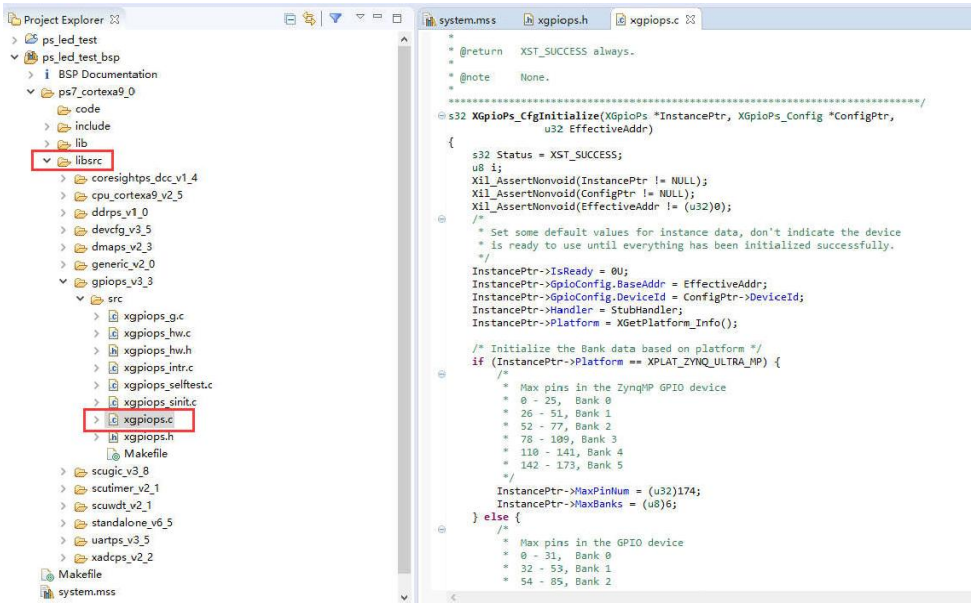
2) The various header files of xilinx are included in the bsp include folder. For example, GPIO used in this chapter uses xgpio.h, in which you can see various macro definitions.These macro definitions can be used when calling GPIO functions to improve readability. Also contains the function declarations that come with the peripherals.



3) The base address, device ID, interrupt, etc. of each peripheral are defined in the “xparameters.h” header file.

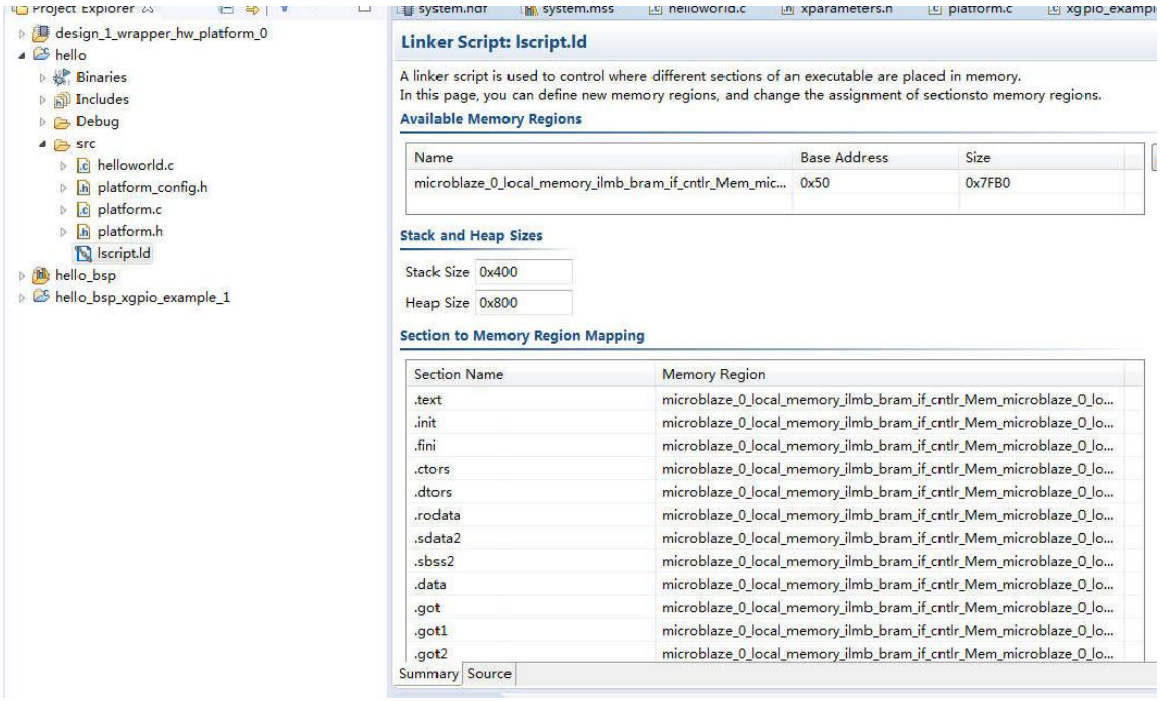


4) In the libsrc folder, contains the definition of the peripheral function, instructions for use



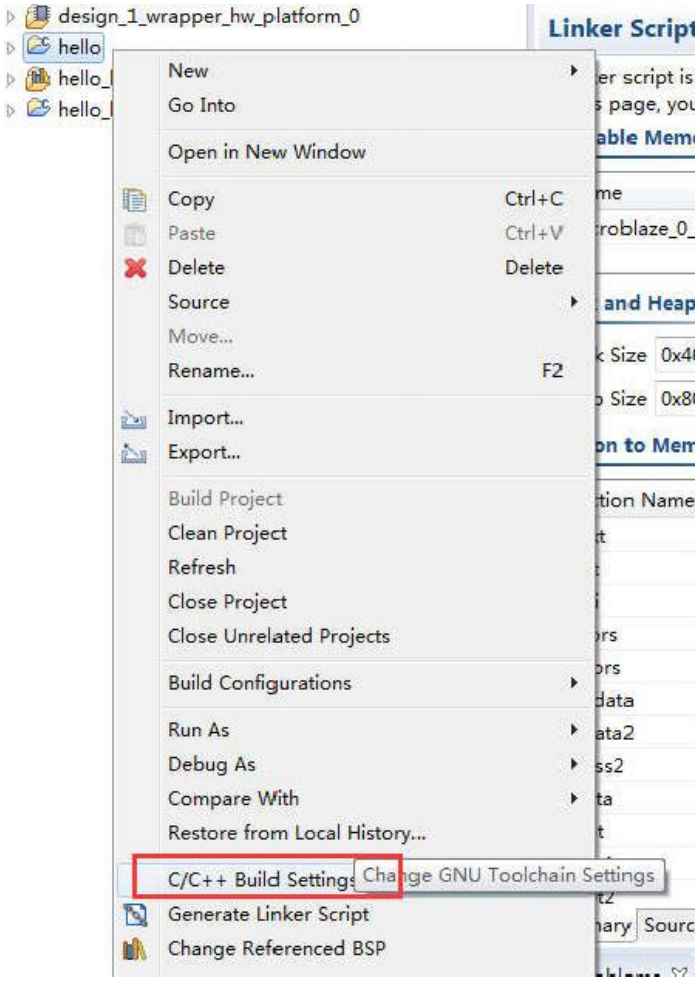
Part 2.2.3: Stack size setting

5) In lscript.ld under the src folder, defines available memory space, stack and heap size, etc., can be modified as needed

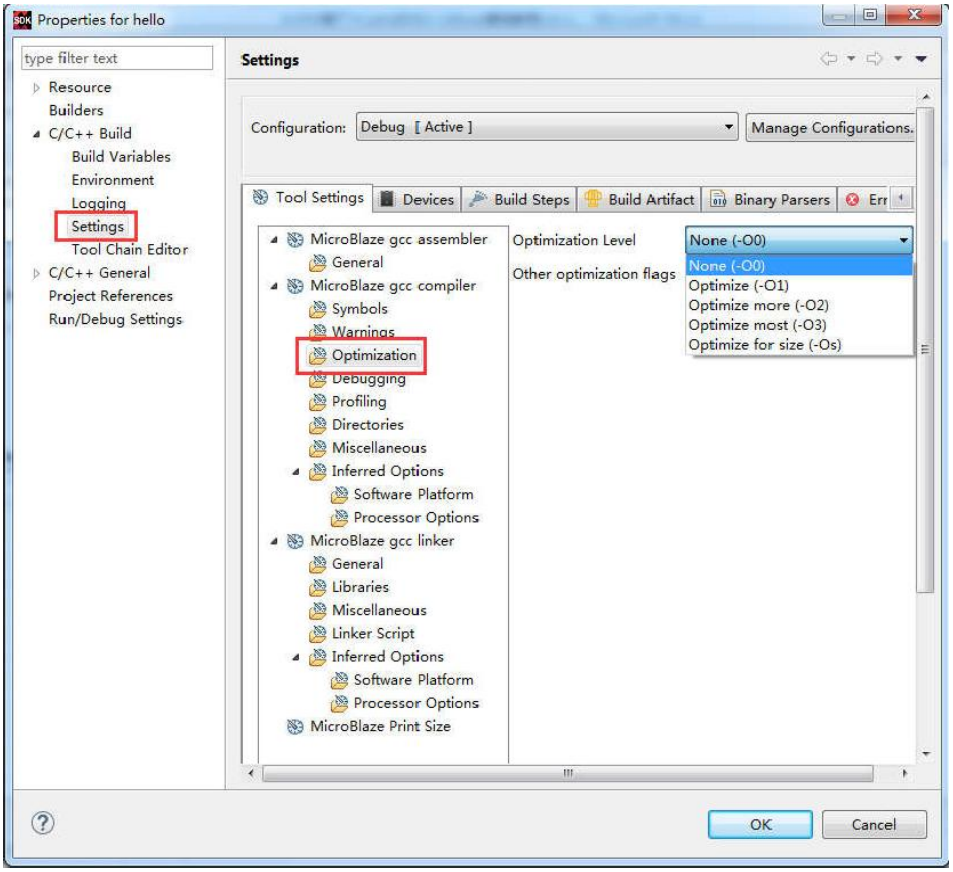


Part 2.2.4: Optimization level and debug level settings

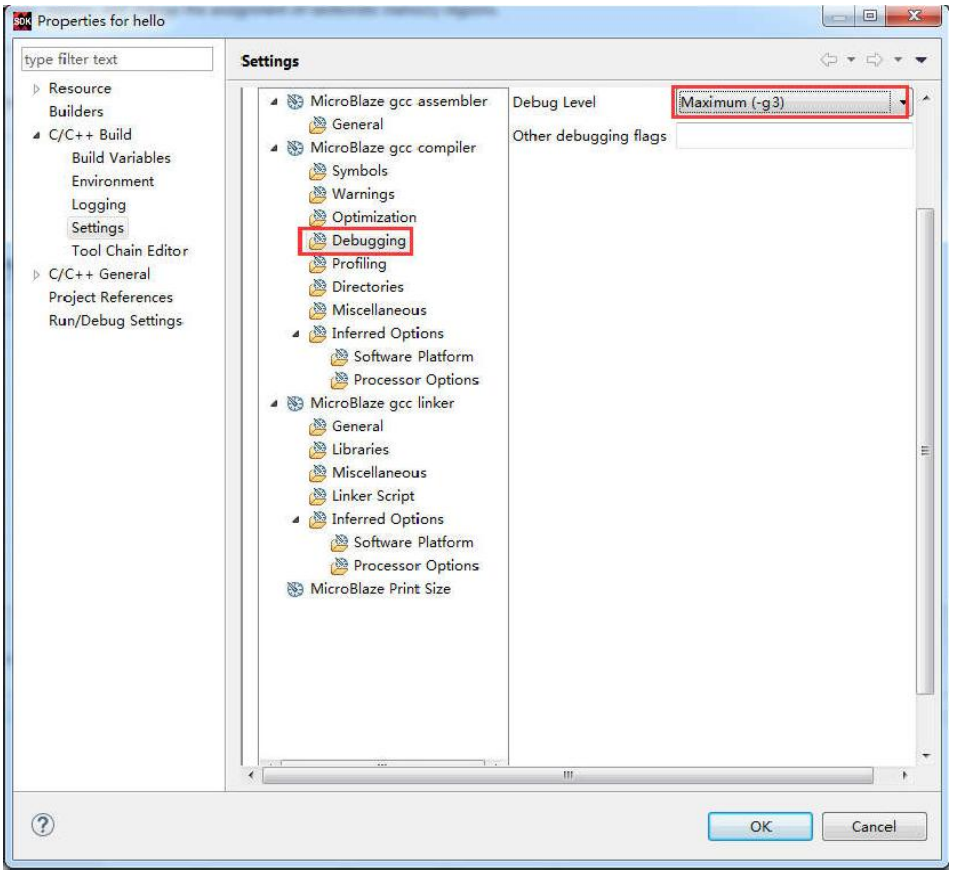
6) Select the project right click and click "C/C++ Build Settings"



7) Set "Optimization Level" in the "Optimization" option, select None (-O0) to indicate no optimization, Optimize for size (-Os) to compile the smallest code.

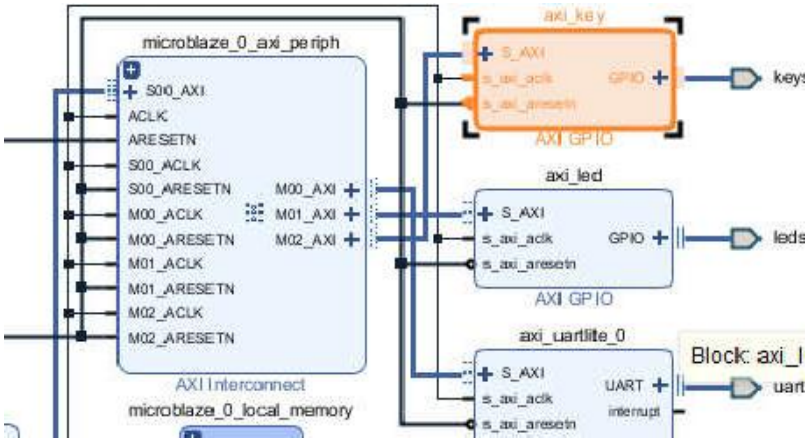


8) The debug level can be set in the "Debuging" setting.

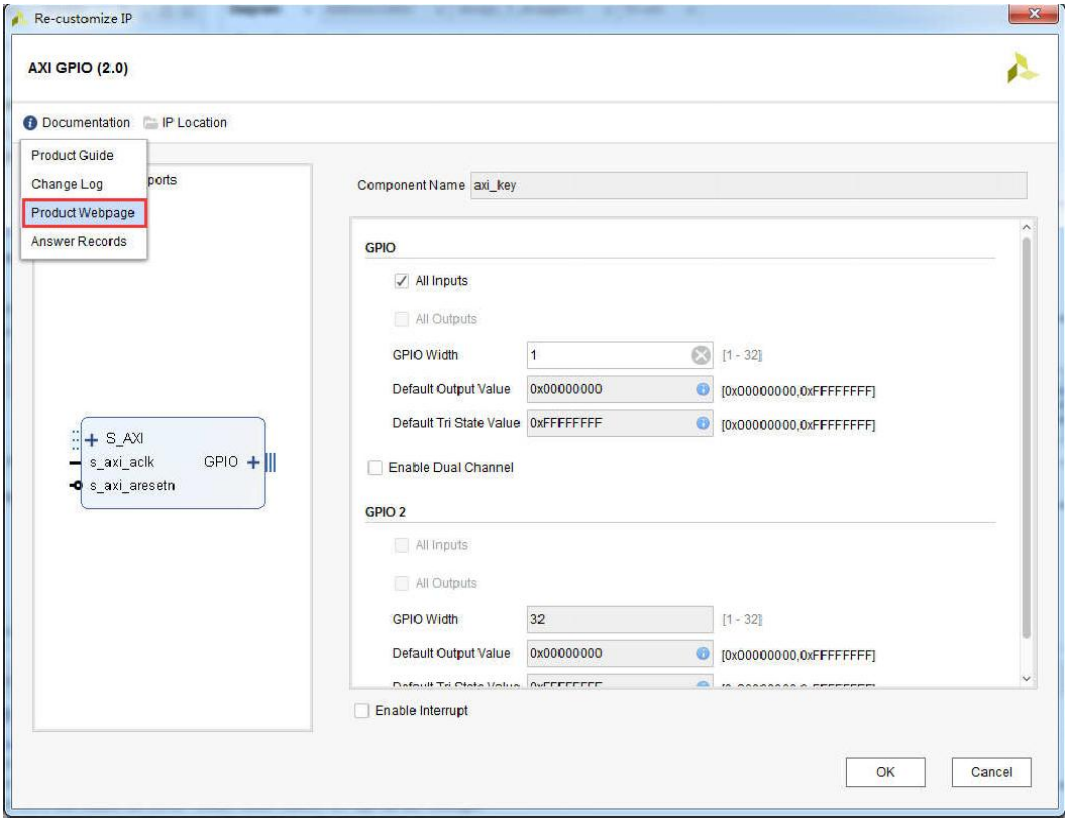


Part 2.2.5: Find the IP core register design instructions

9) Double-click the IP core in vivado



10) Click "Documentation -> Product Webpage" to open the IP core related webpage, you can find the IP core details.



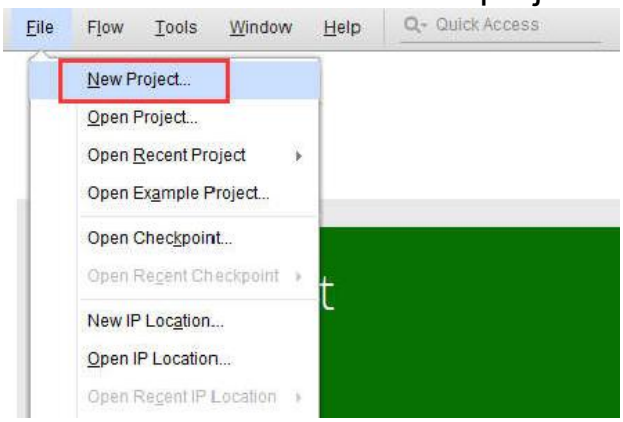
Part 3: MicroBlaze system with mig(ddr3)

This chapter uses Vivado software to build a MicroBlaze minimum system that uses the FPGA internal block RAM as the CPU to run RAM.The main peripherals are UART and GPIO. In order to balance those who do not follow the tutorial sequence, this chapter is slightly embarrassing.

Part 3.1: Create a VIVADO project

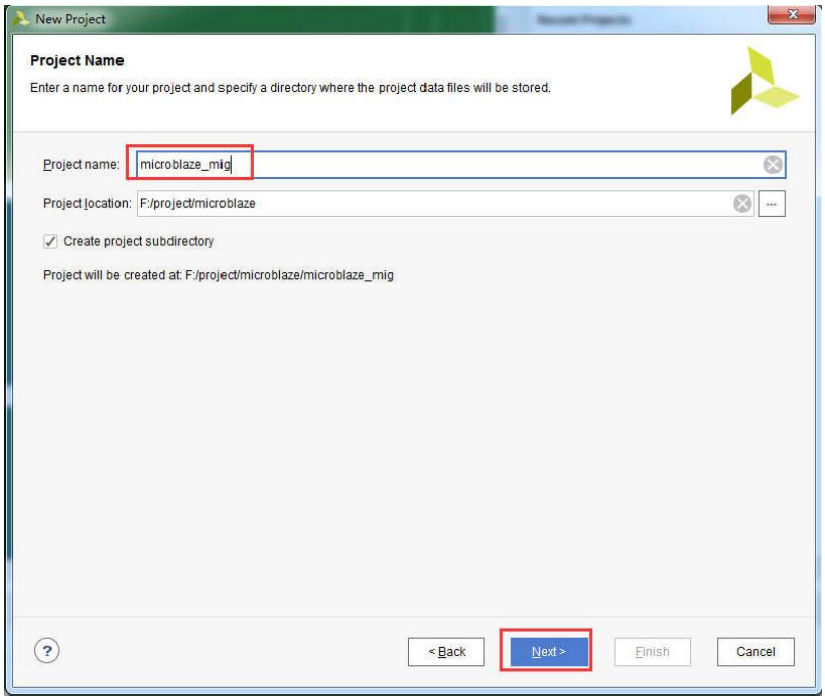
Part 3.1.1: Create a new project

1) Open Vivado software and create a new project

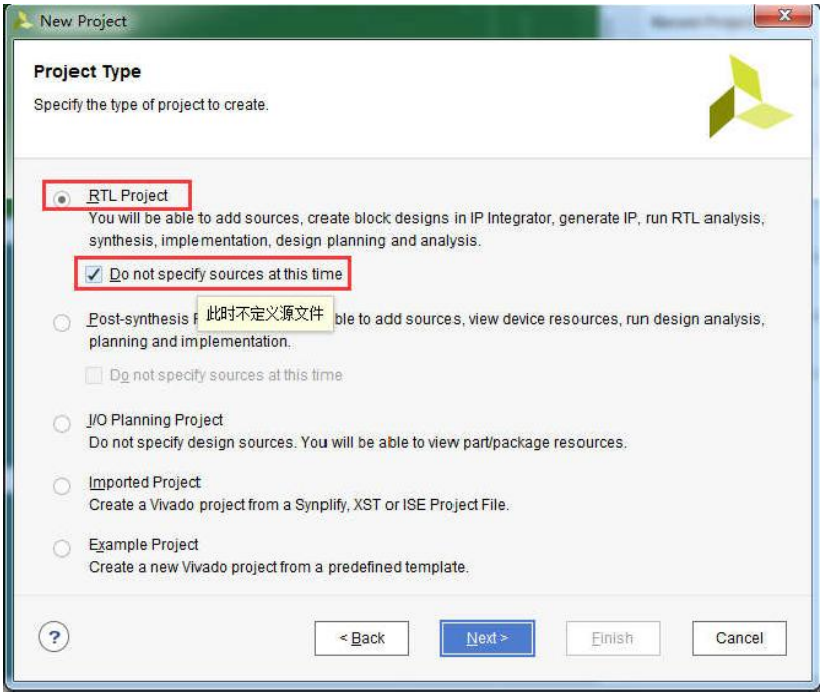


2) Click "Next" to create a new Vivado project

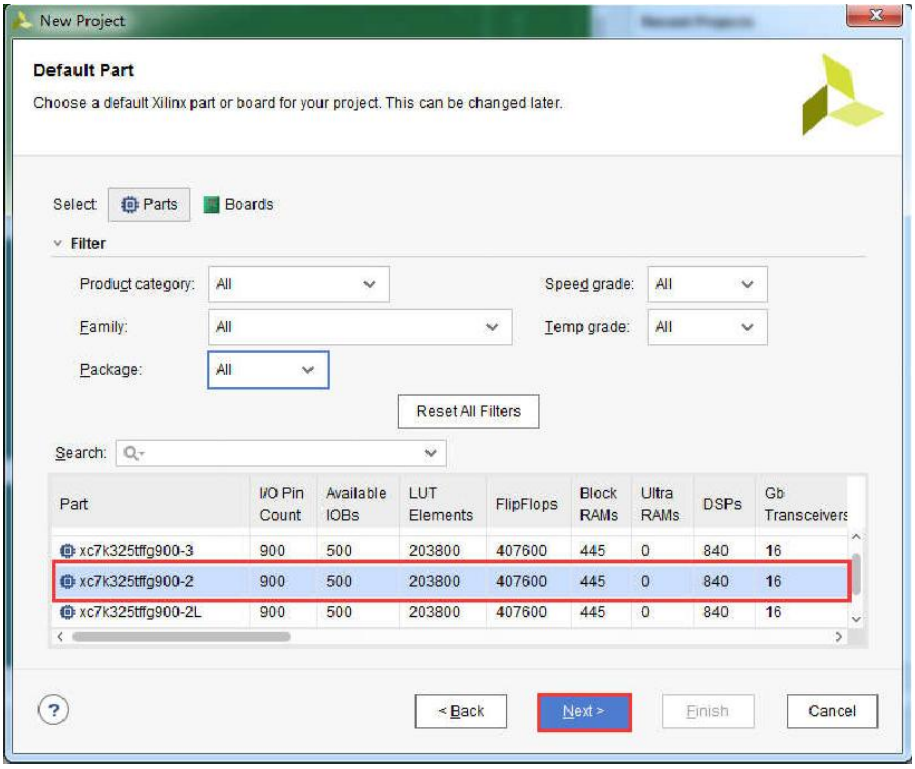
3) Fill in the project name "microblaze_mig", select the project path, and keep the default project subdirectory option. The project name and project path should not have Chinese spaces. The project path should be as short as possible, otherwise compilation errors may occur. Click "Next"



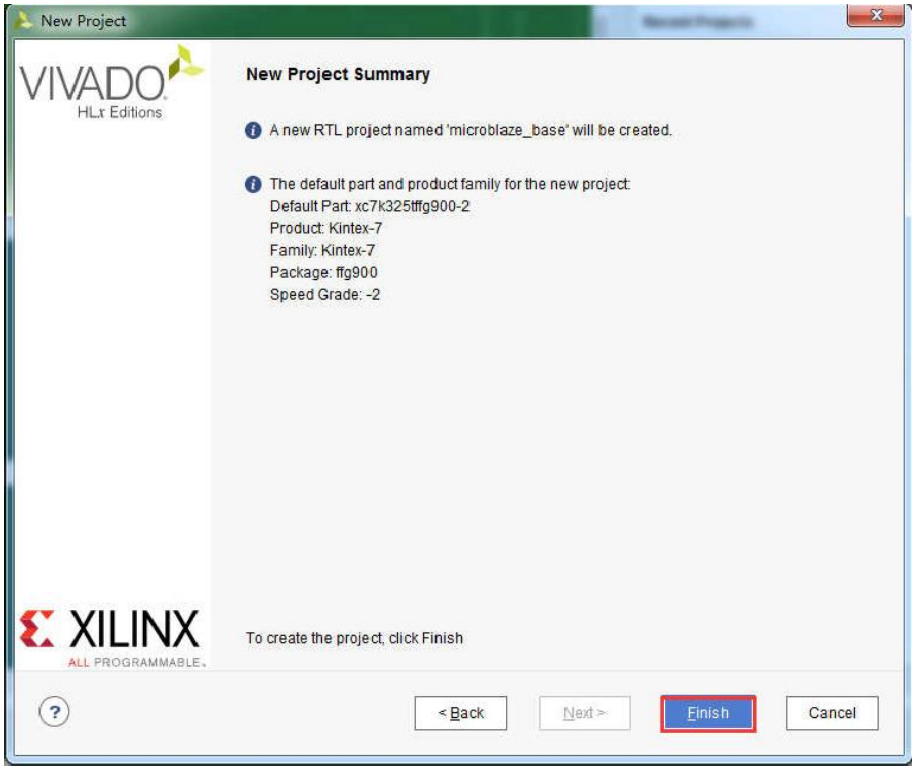
4) Choose to create an RTL project and check "Do not specify sources at this time"



5) Select the FPGA model according to the FPGA development board and click "Next"

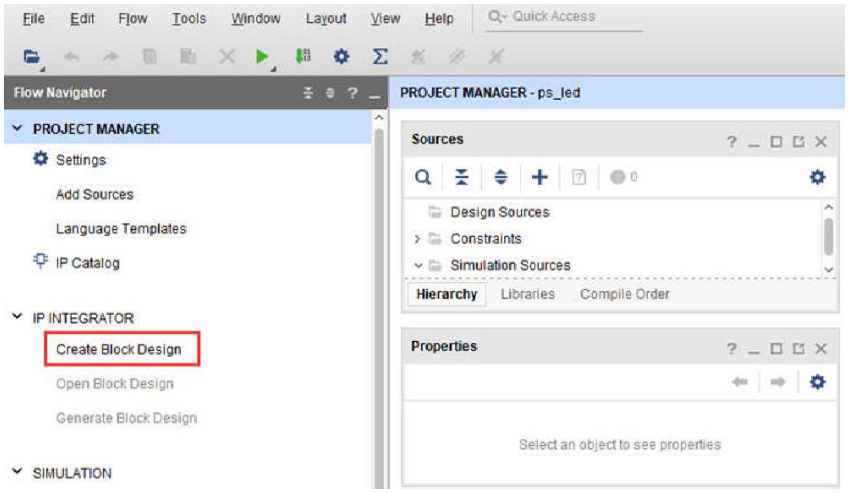


6) Click "Finish" to complete the project creation wizard.

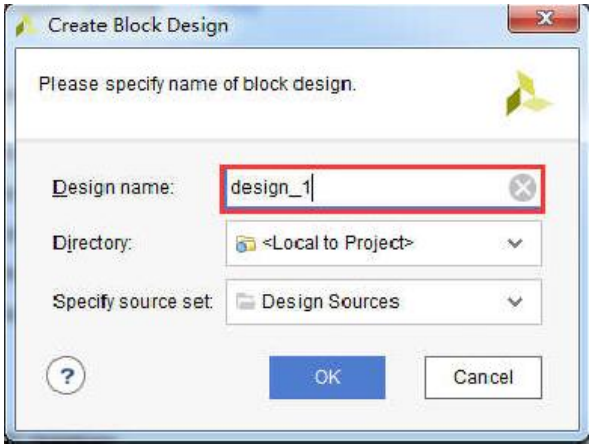


Part 3.1.2: Create Block Design

7) Click Create Block Design

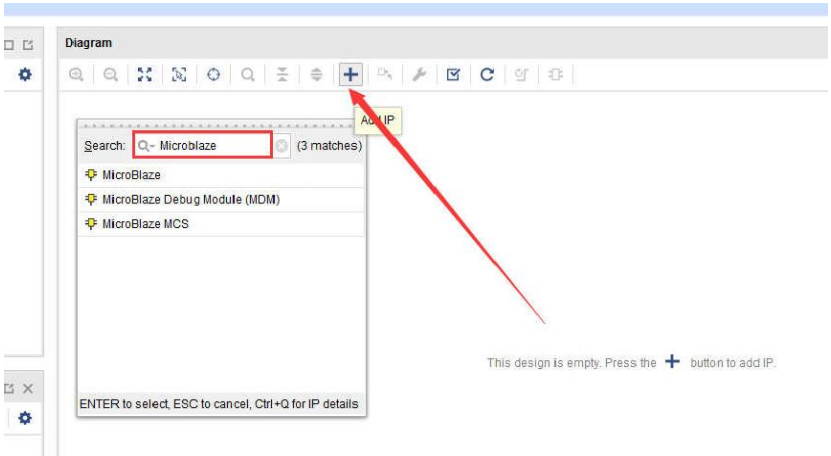


8) Keep the design name as the default and click "OK"

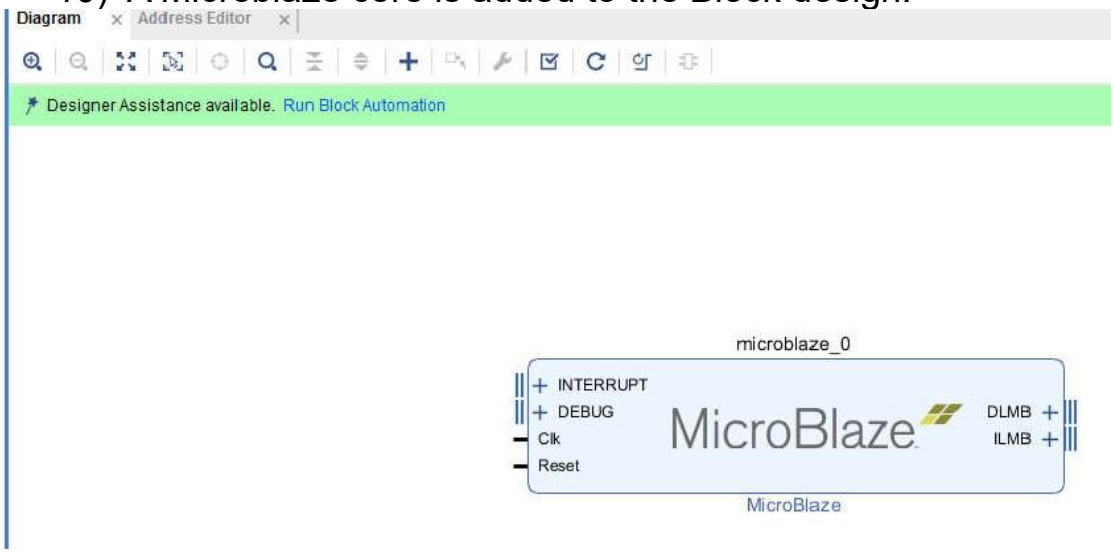


Part 3.1.3: Add soft core

9) Click the Add IP shortcut button to search for “Microblaze” and double-click “Microblaze” in the drop-down list

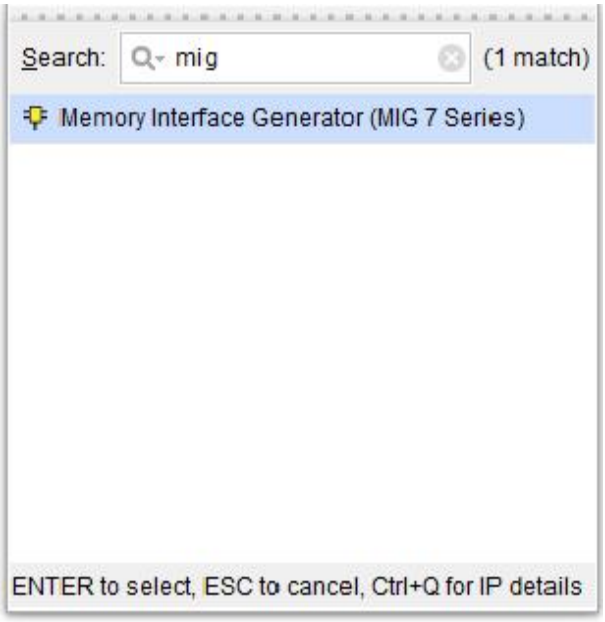


10) A Microblaze core is added to the Block design.

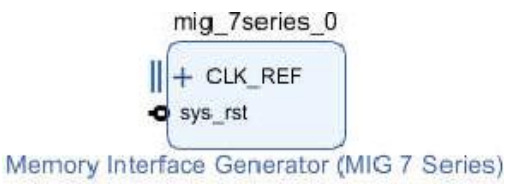


Part 3.1.4: Add mig core

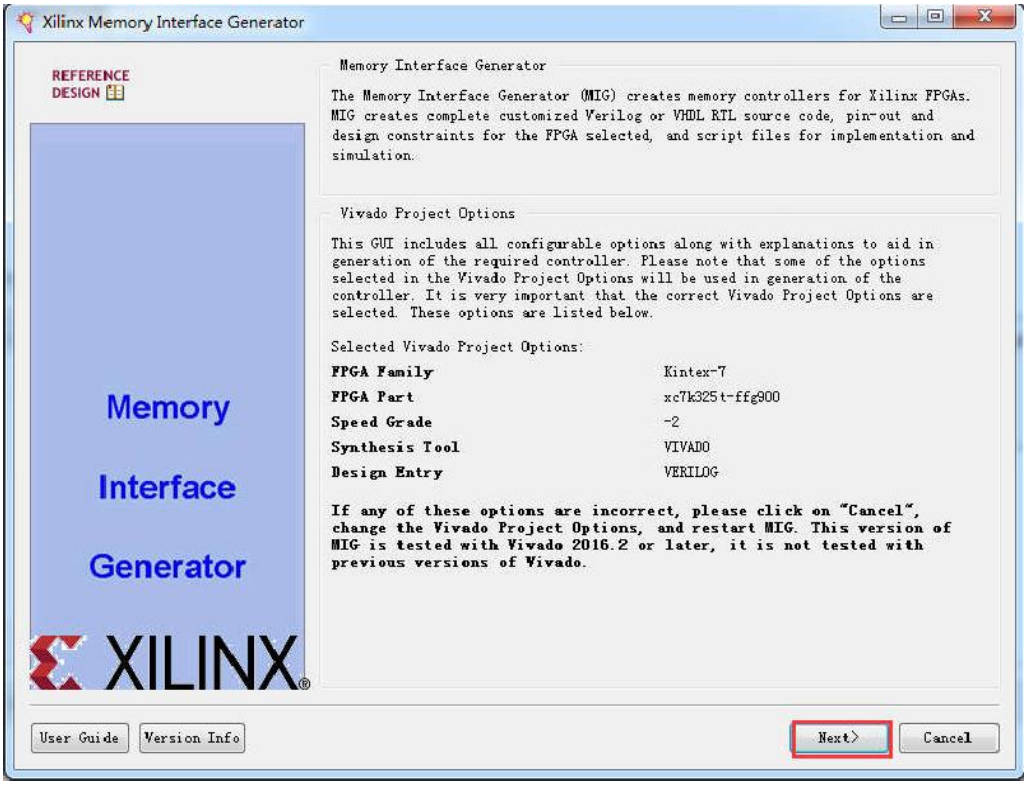
11) Search for "mig" and double click on "Memory Interface Generator (MIG 7 Series)" to add a mig core.



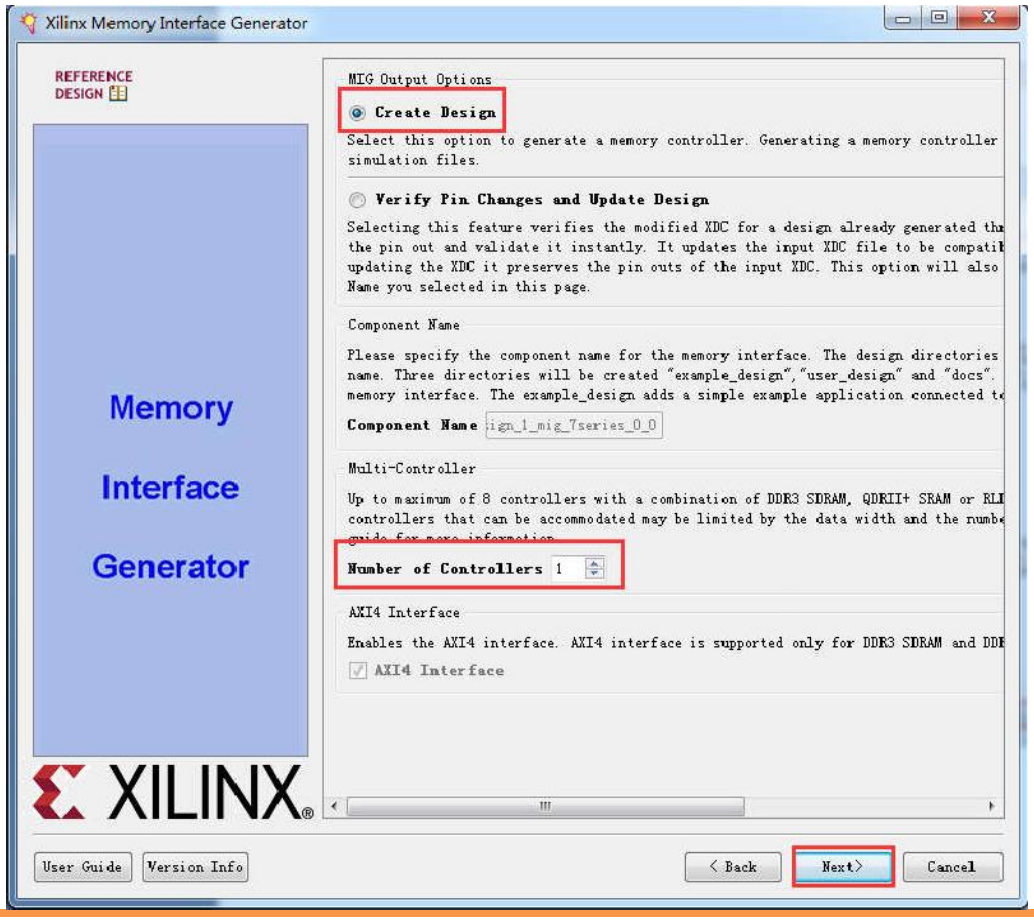
12) Double-click the mig core you just added to configure



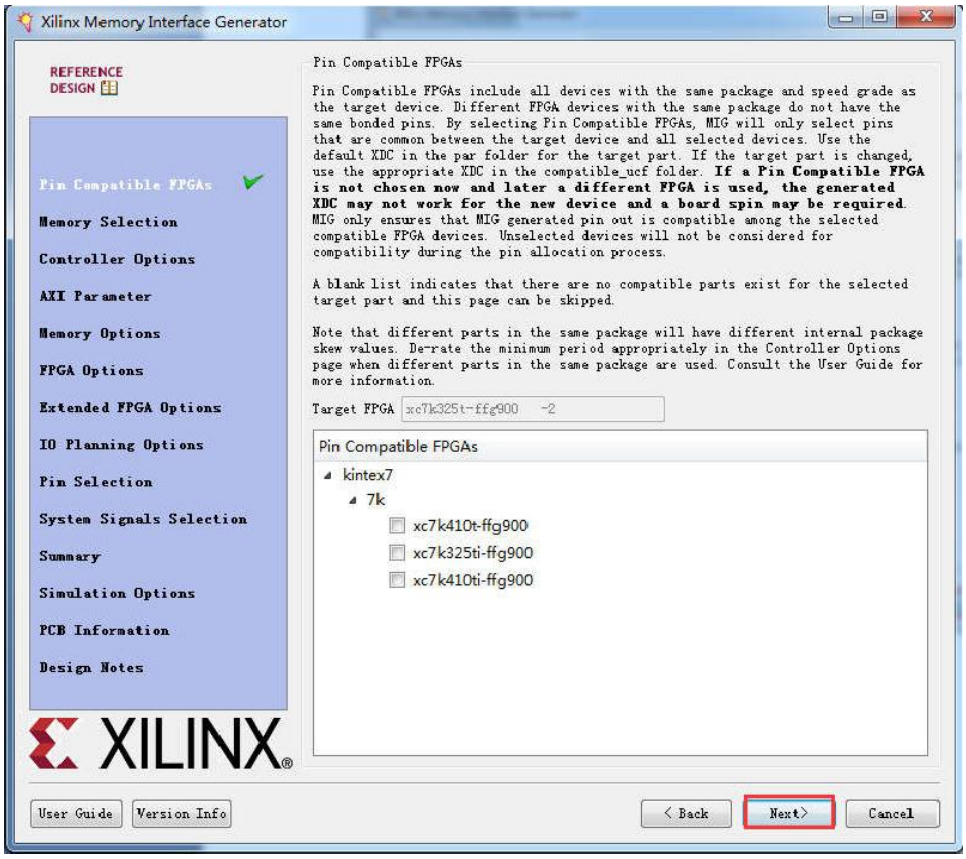
13)Click "Next"



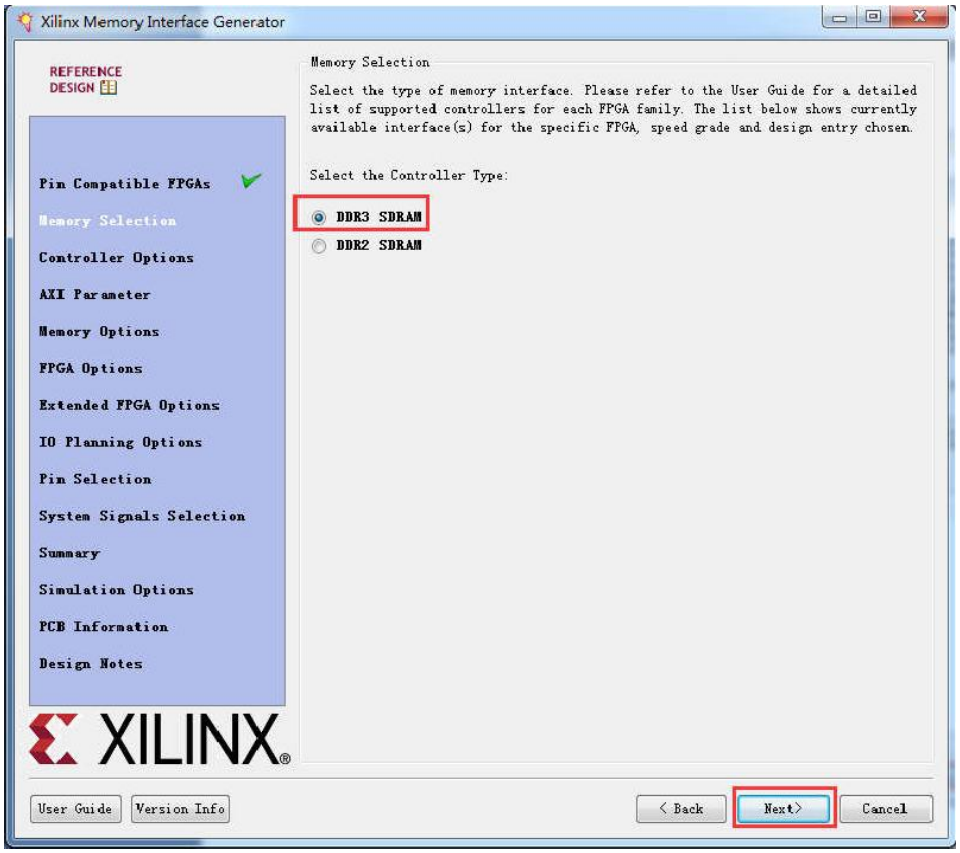
14)Keep the default value and click "Next"



15)Click "Next"

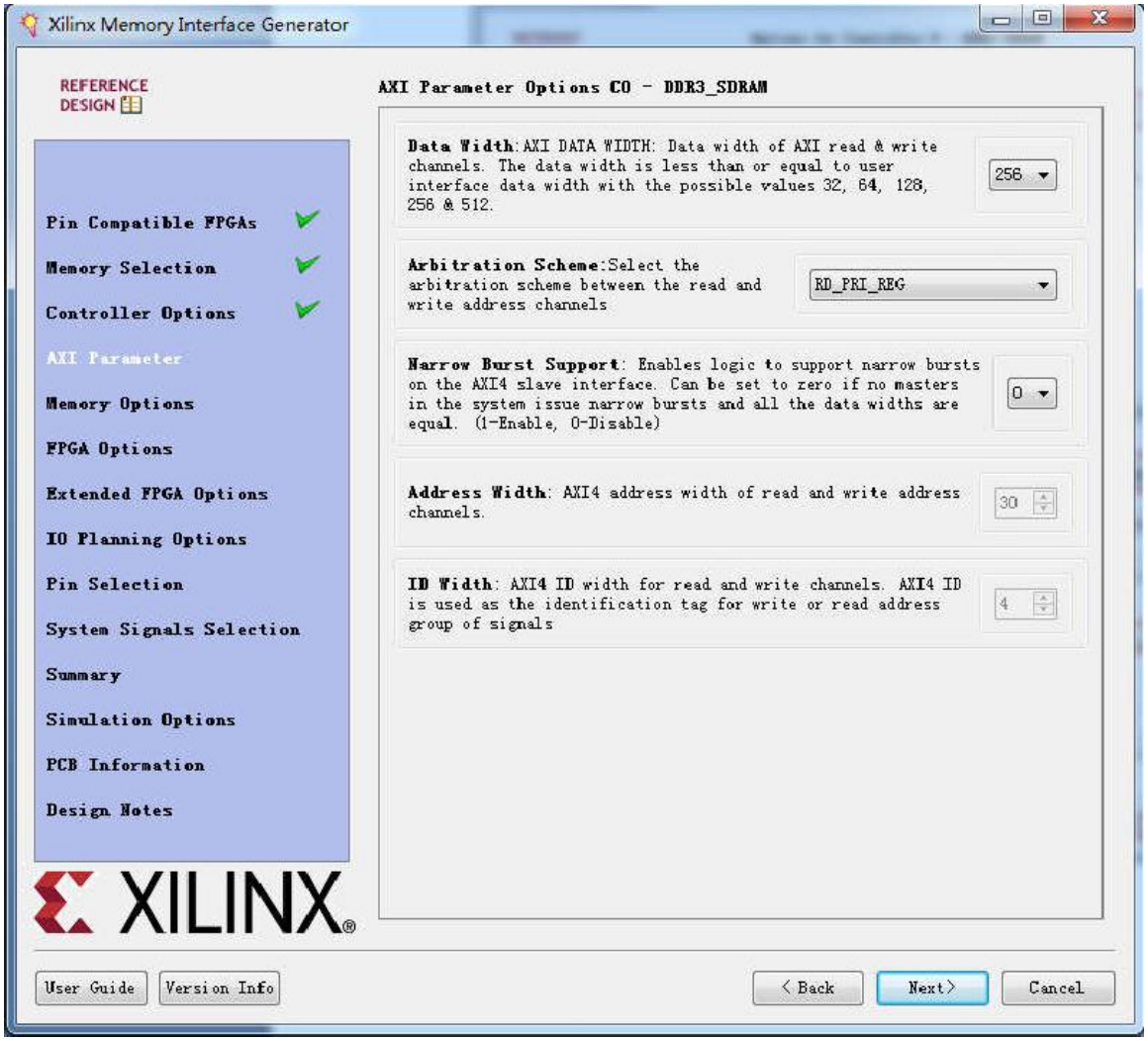


16)Click "Next"

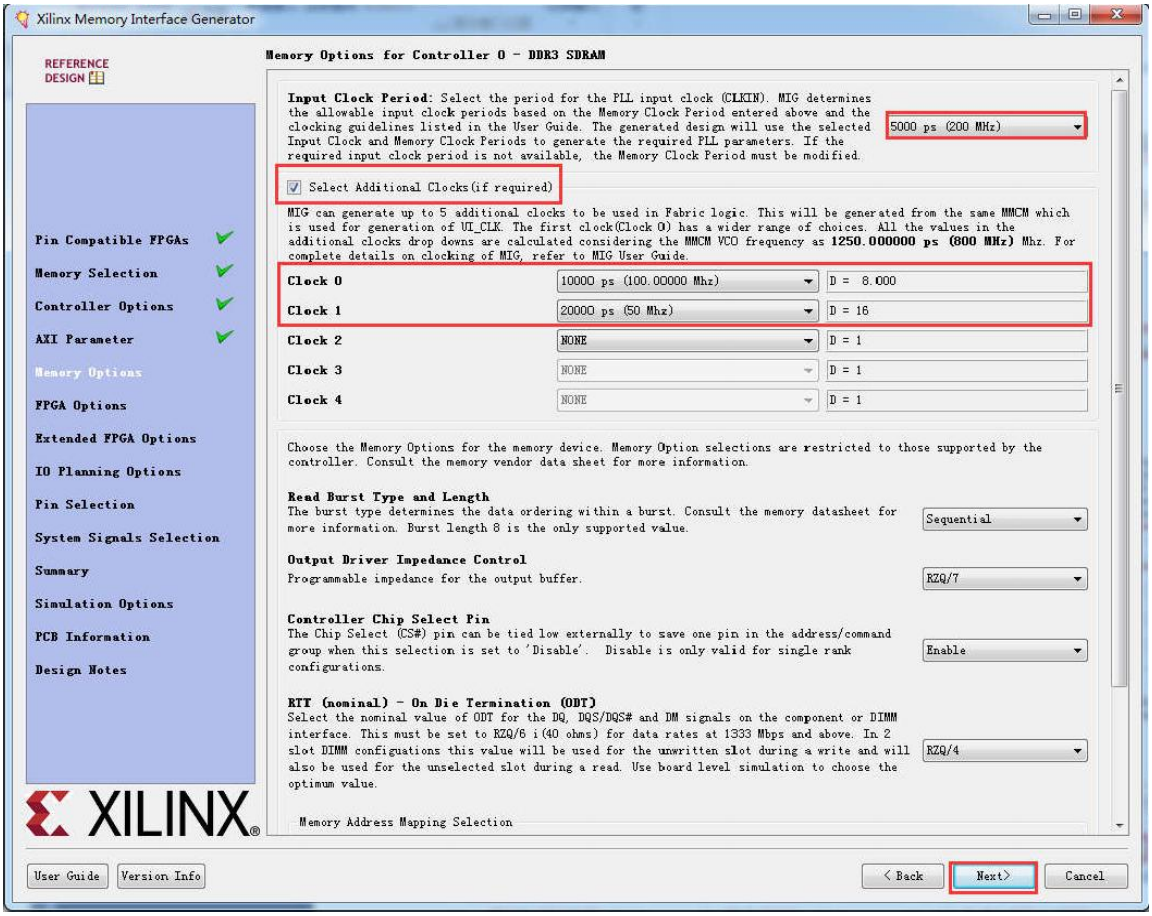


17) The Clock Period is set to 1250ps, that is, the ddr3 clock is 800Mhz, the Memory Part selects “MT41K256M16XX-125”, and the Data Width selects 32 bits. The above settings must be adjusted according to the specific development board. Click "Next"

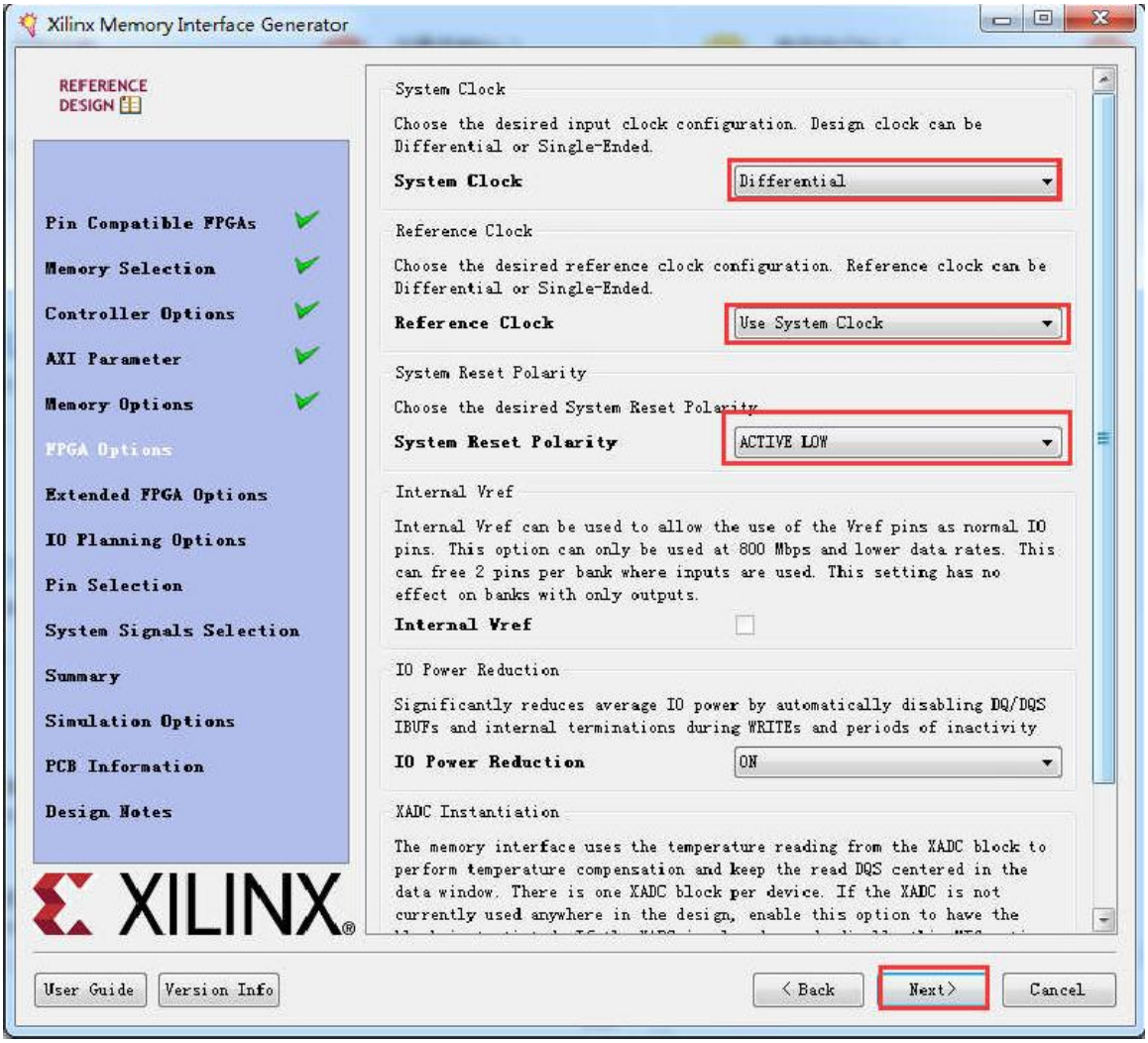
18) Click on "Next"



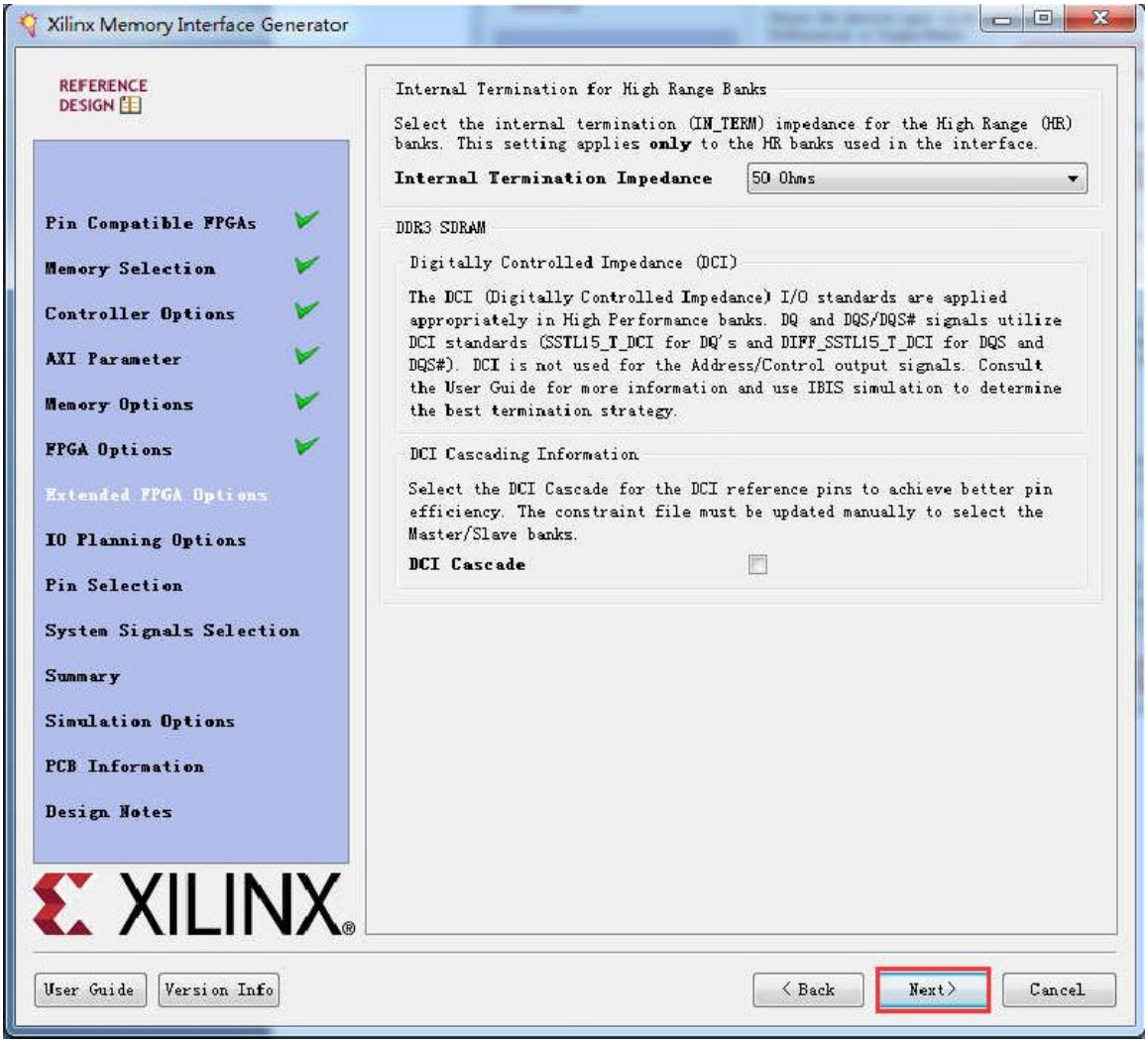
19) Input Clock Period Select 5000ps, that is, external input 200Mhz as the system clock, check “Select Additional Clocks(if required)”, select an additional output clock to Microblaze, which can save a PLL. Clock 0 selects 100Mhz and Clock 1 selects 50Mhz.



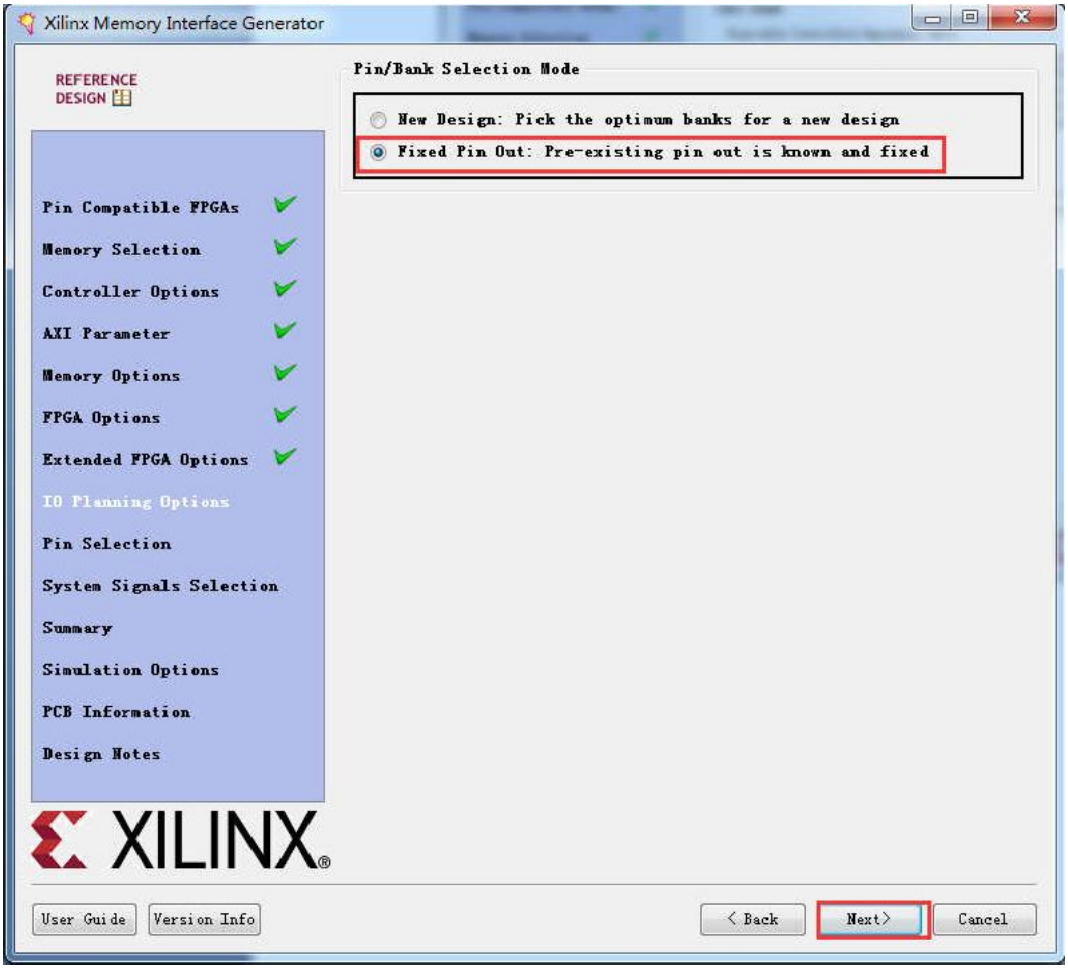
20) The system clock selects the "Differential" type, the "Reference Clock" must be 200Mhz, because the system clock is exactly 200Mhz, here you can select "Use System Clock", the reset polarity remains low reset. Click "Next"



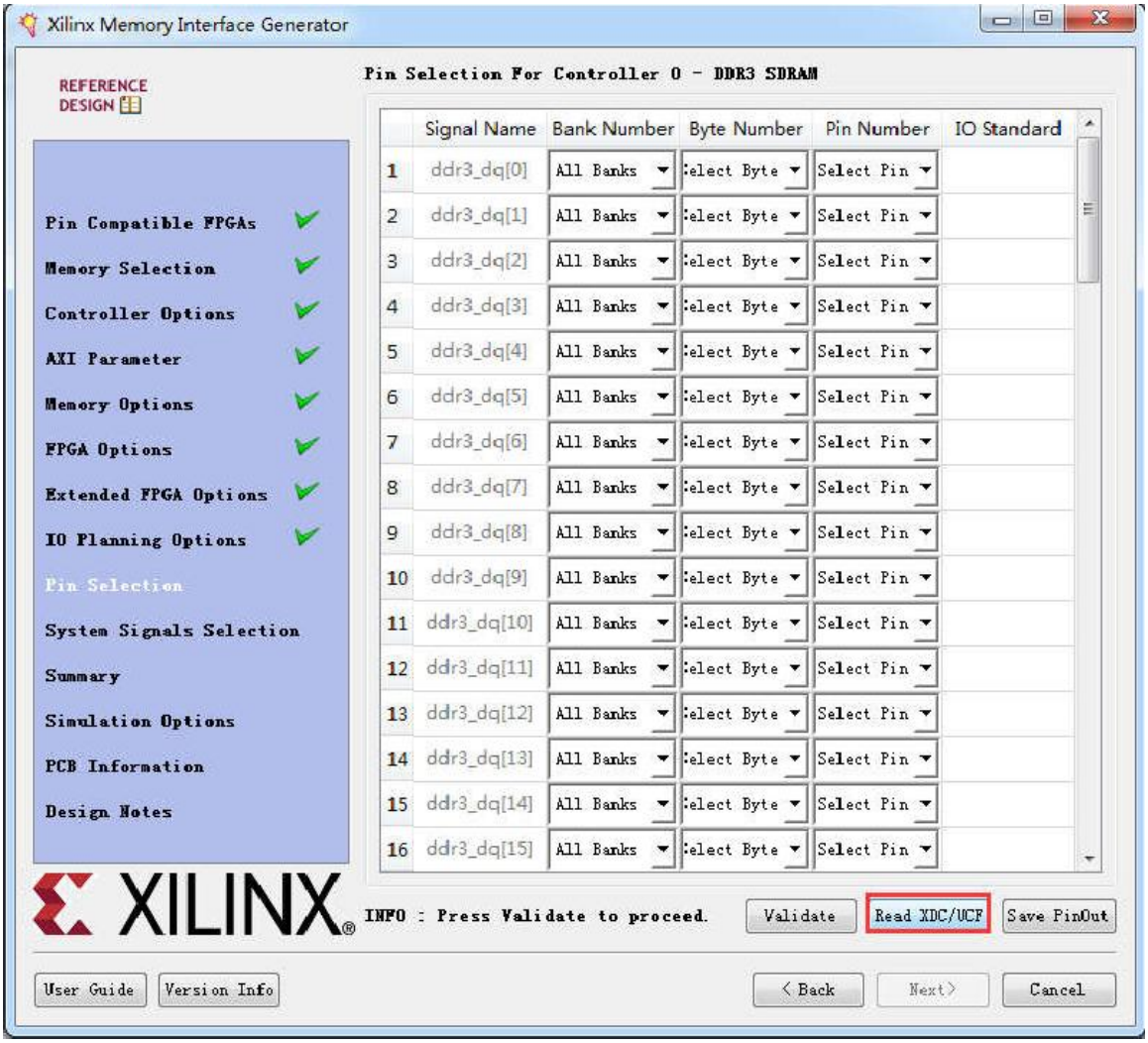
21)Click "Next"



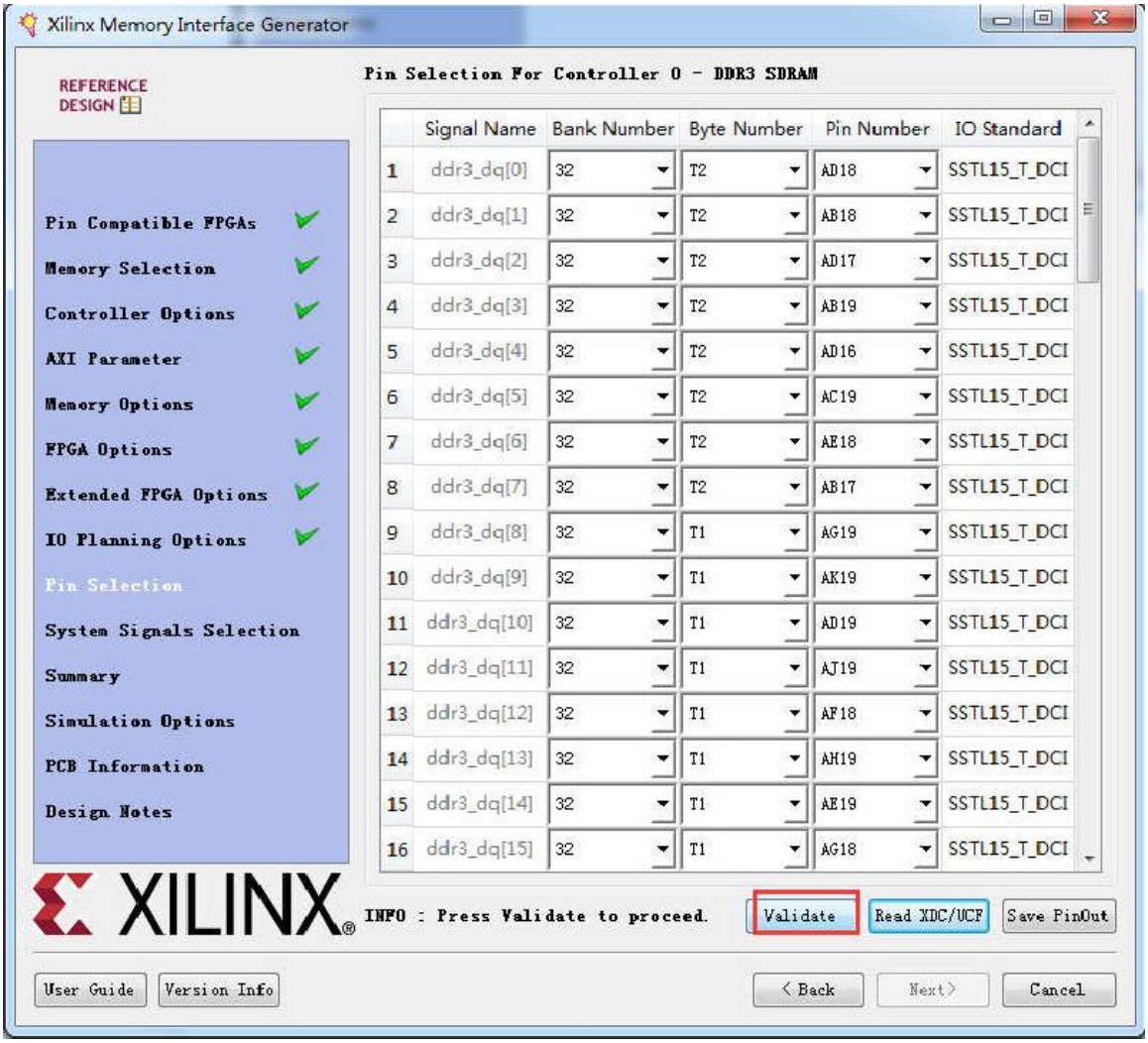
22) Select "Fixed Pin Out: Pre-existing pin out is known and fixed", because it is the development board, the ddr pin has been determined.



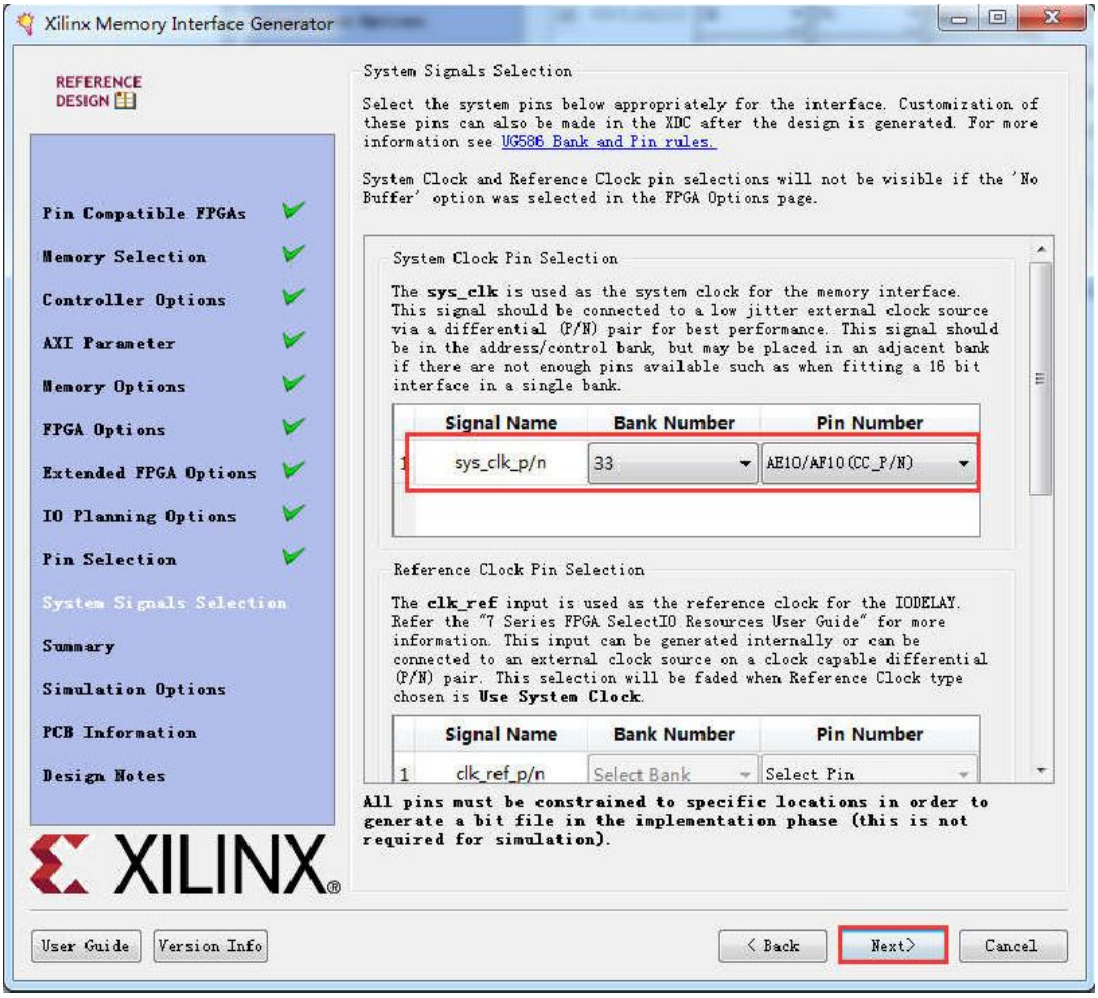
23) Click on "Read XDC/UCF" and select the ddr3.ucf file provided by the development board



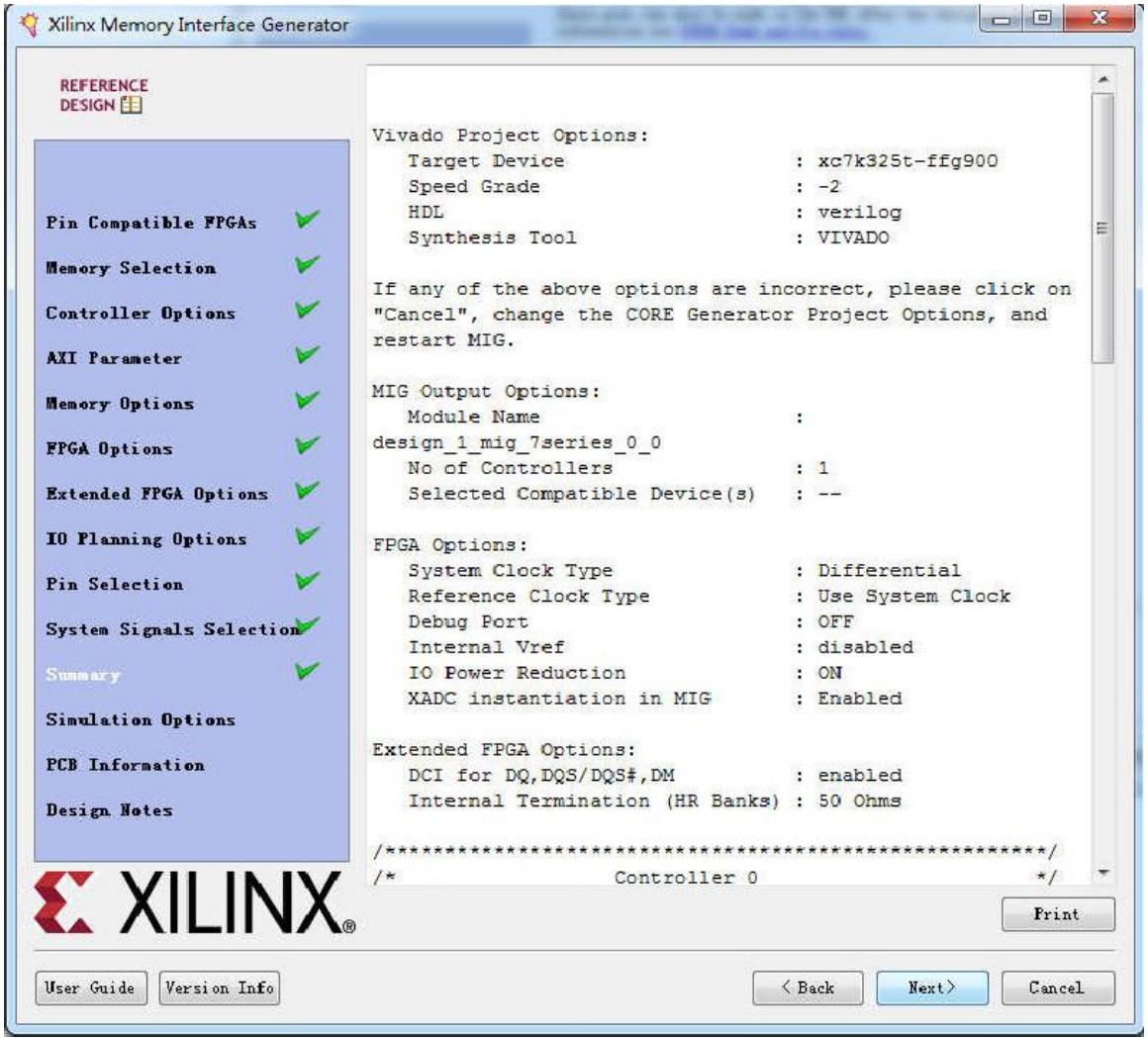
24)Click the "Validate" button to check for errors. Click "Next" if there are no errors.



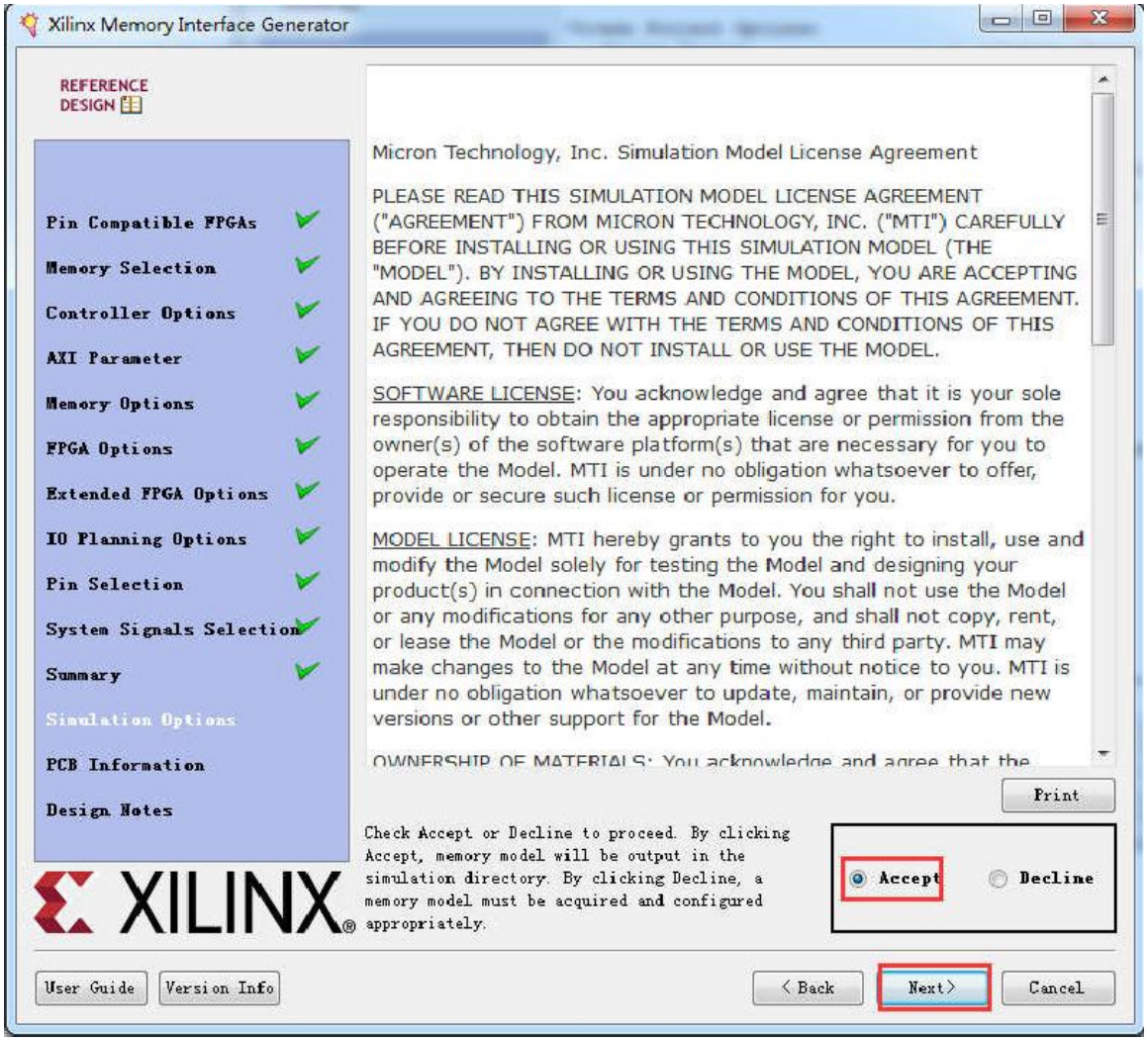
25)System clock pin assignment, according to the specific conditions of the development board, click "Next"



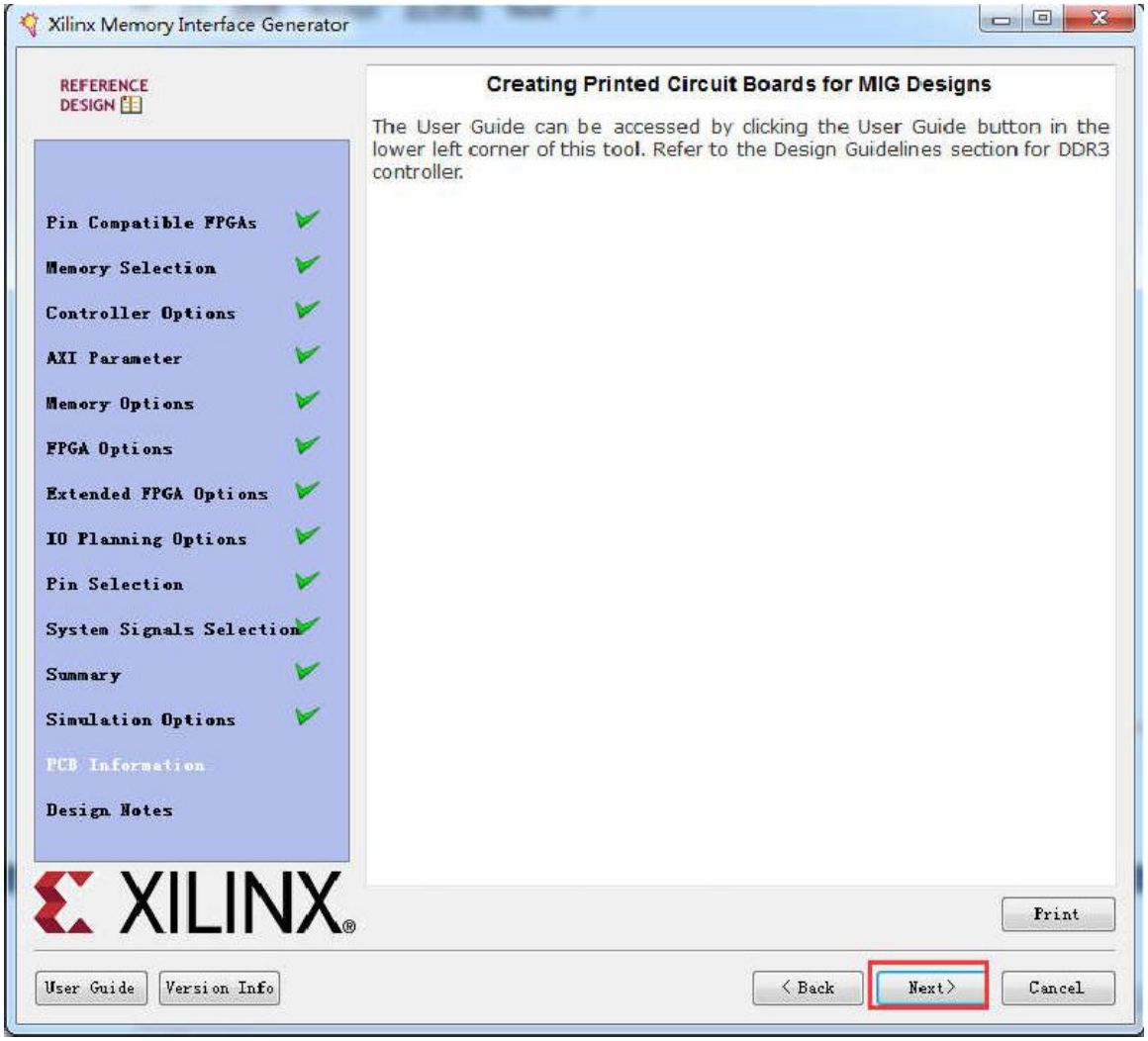
26) Click "Next"



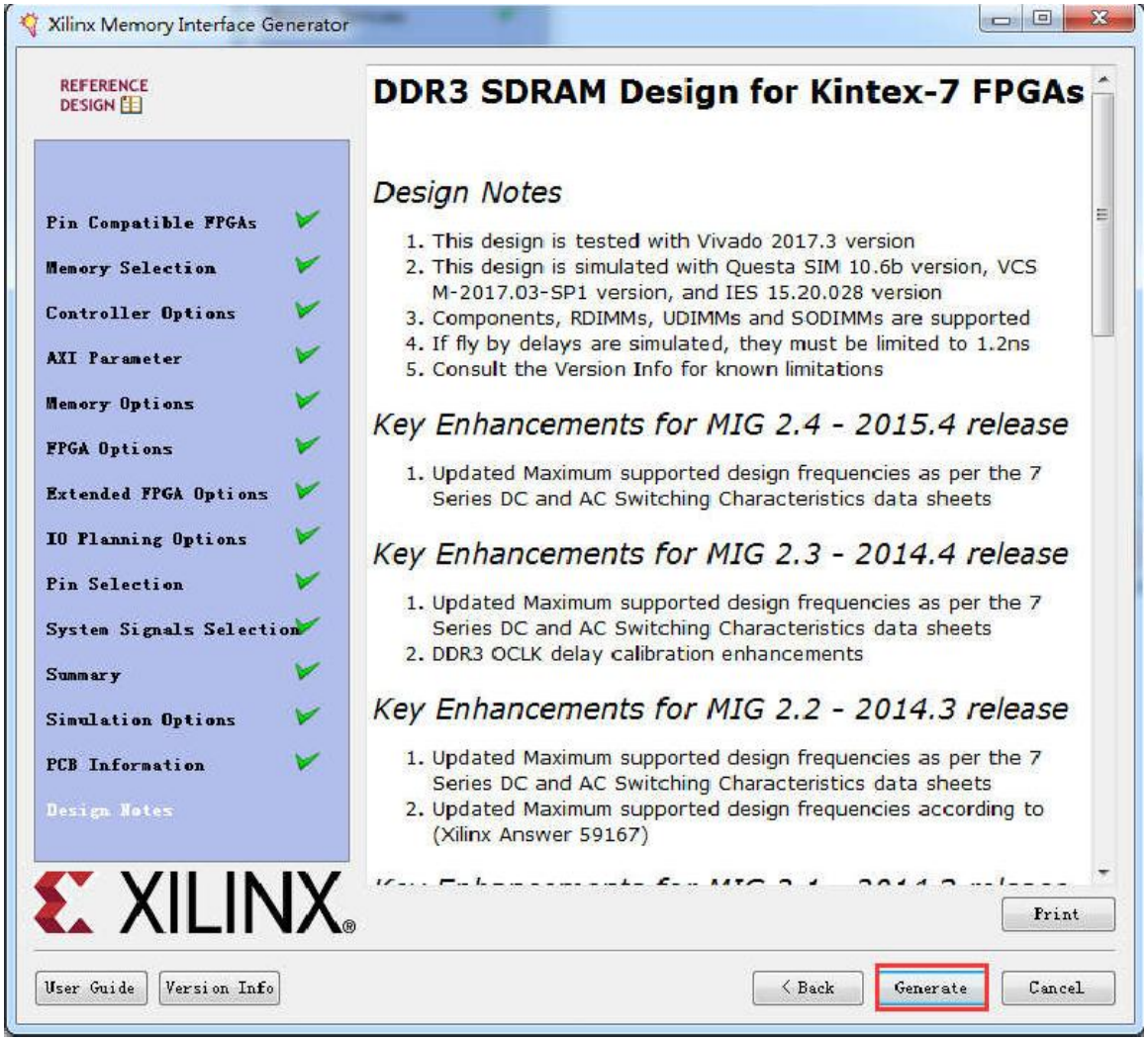
27)Select "Accept" and click "Next"



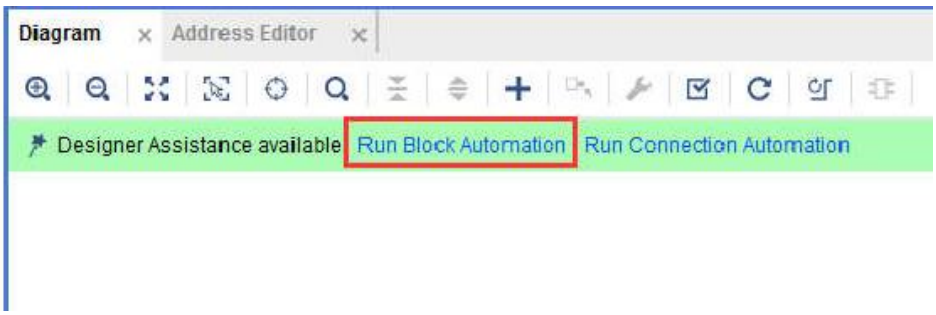
28)Click on "Next"



29)Click "Generate" to generate mig IP



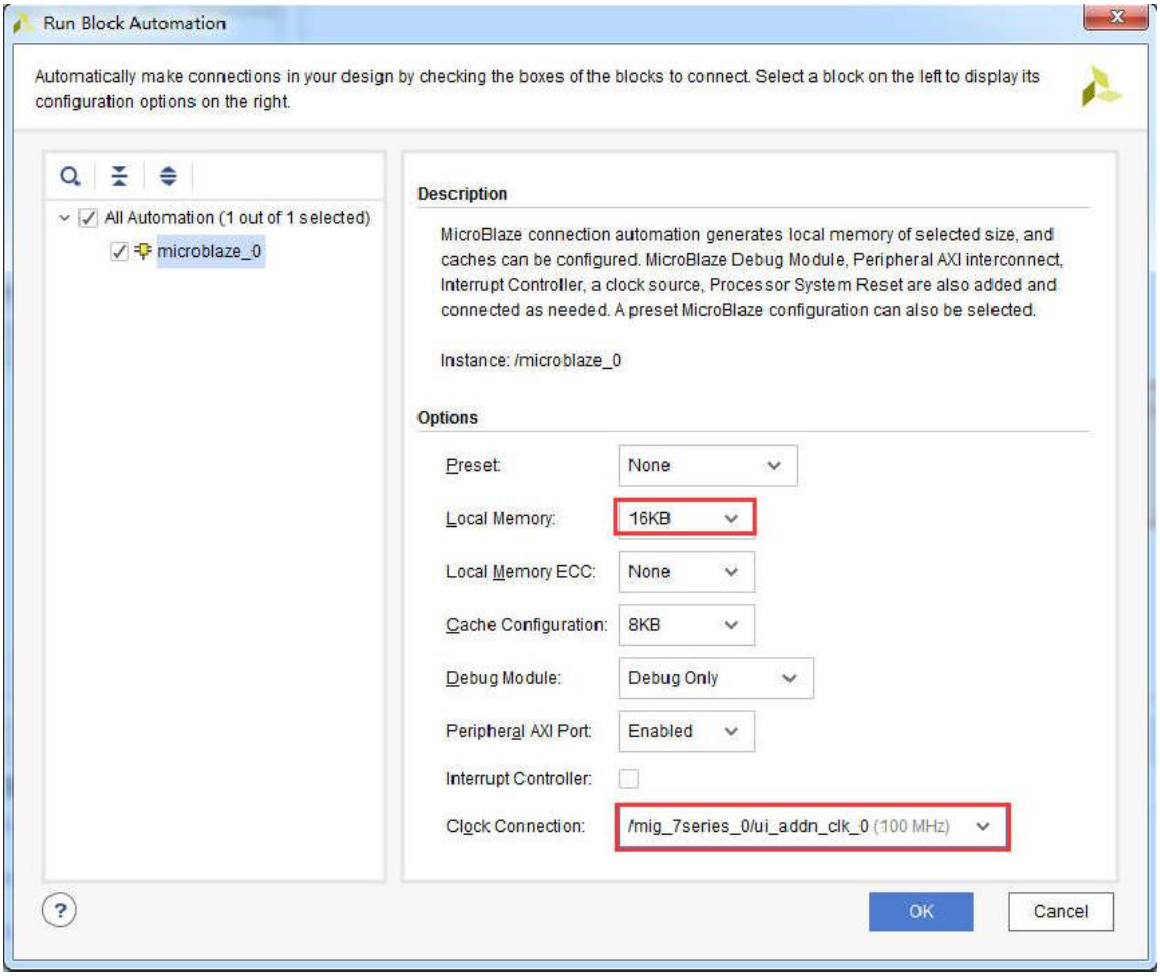
30) Click "Run Block Automation" to complete some automatic settings. Note that this option will not always be available.



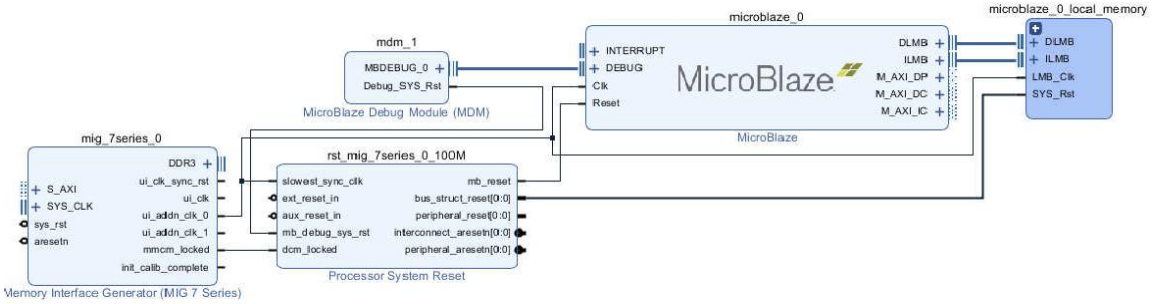
Once the settings are completed, there may be no chance to perform some automatic settings in this way.

31) Complete some microblaze parameter settings in the pop-up window and keep the default "None" in the "Preset" option.

This option is equivalent to presetting some microblaze configuration parameters, depending on the application. “Local Memory” selects 16KB, “Clock Connection” clock selects 100Mhz clock output from mig core as CPU clock. If the clock frequency is too high, it may cause timing problems. 100Mhz can meet most requirements.

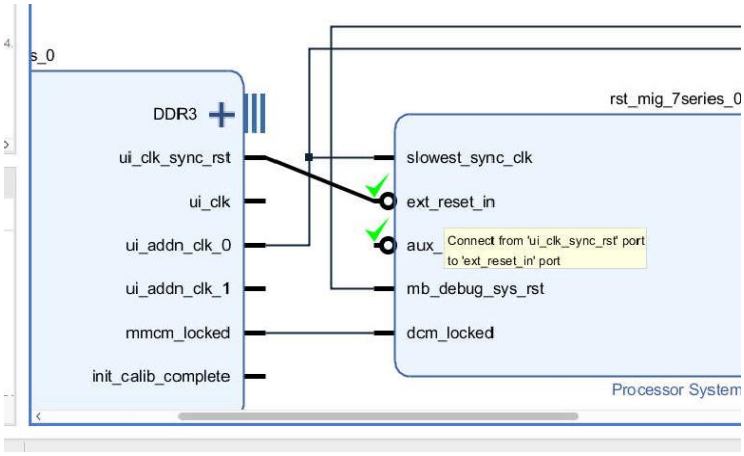


32) Some modules and connections have been added automatically



33) Connect ui_clk_sync_rst of mig_7series_0 to ext_reset_in of

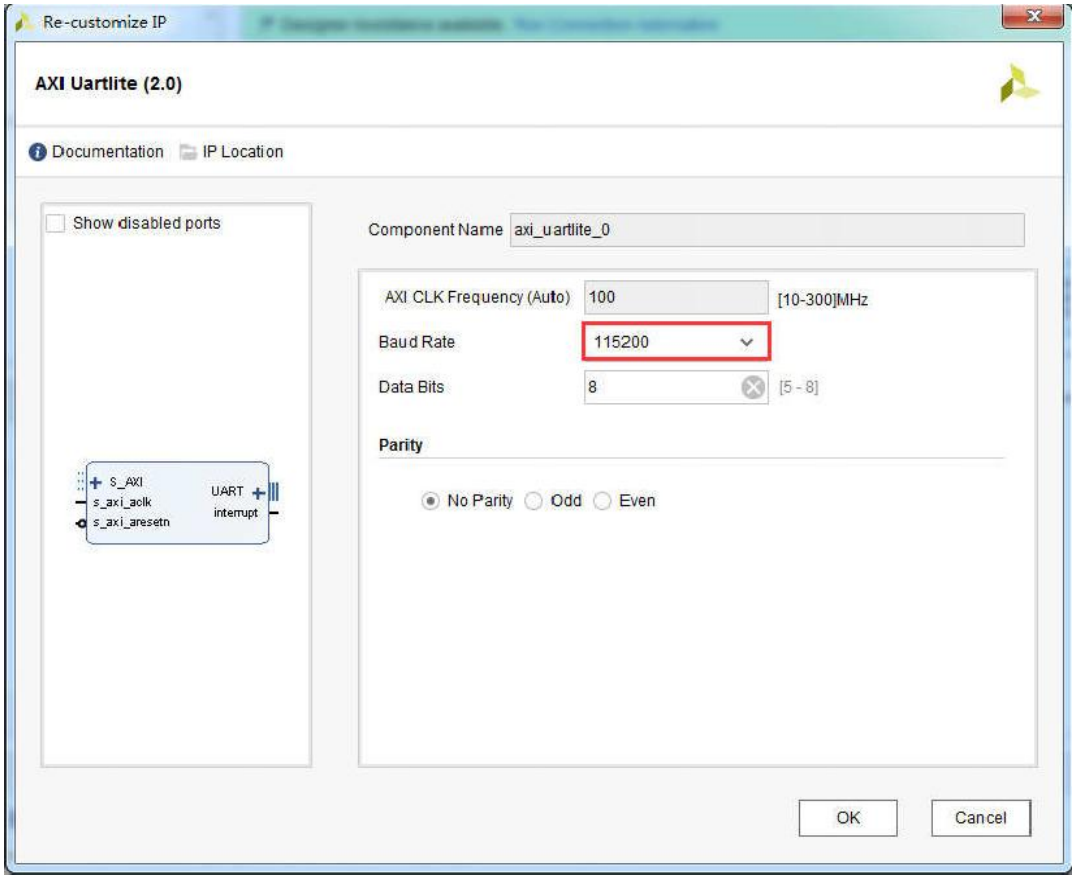
rst_mig_7series_0_100M



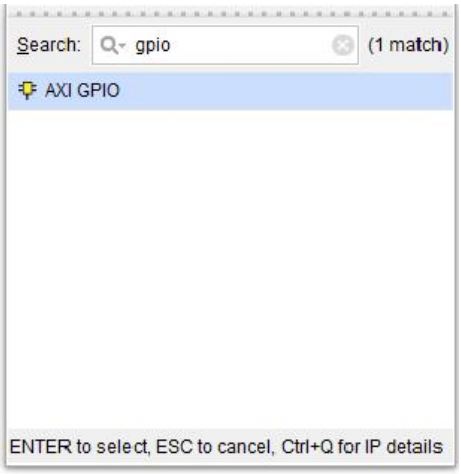
Part 3.1.5: Add peripherals

34) Search for "uart" and double-click "AXI Uartlite" to add a Uart serial IP core

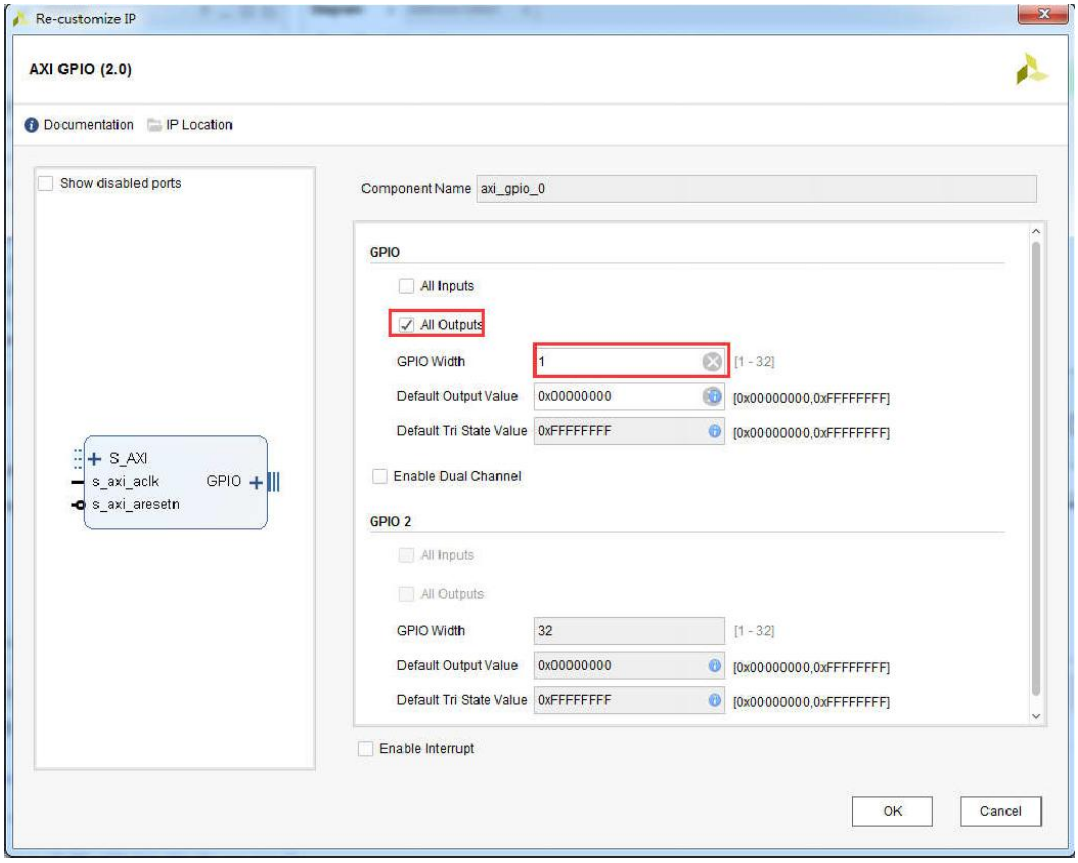
35) Double-click the Uartlite core you just added to modify the baud rate to 115200. This is the commonly used baud rate for embedded systems. The baud rate is a fixed value and cannot be dynamically modified in the program. Click OK to complete the configuration.



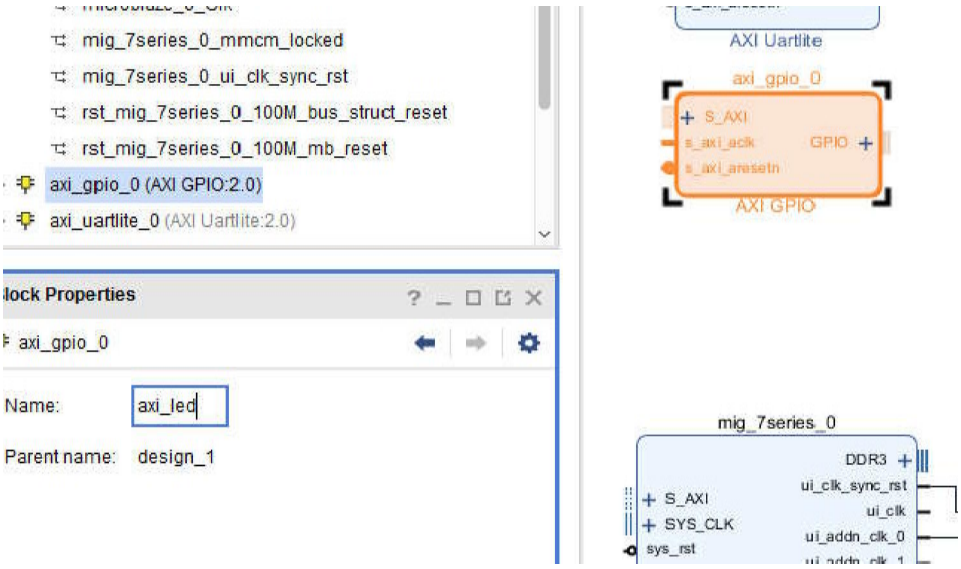
36) Search for gpio and double-click on “AXI GPIO” to add an AXI GPIO for controlling the LEDs.



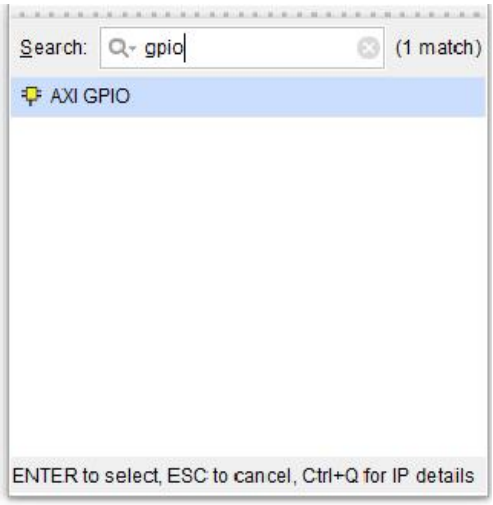
37) Double-click the AXI GPIO you just added to modify the parameters, check “All Outputs”, set to full output, “GPIO Width” fill in 1, here only control 1 bit LED, click “OK” to complete the configuration



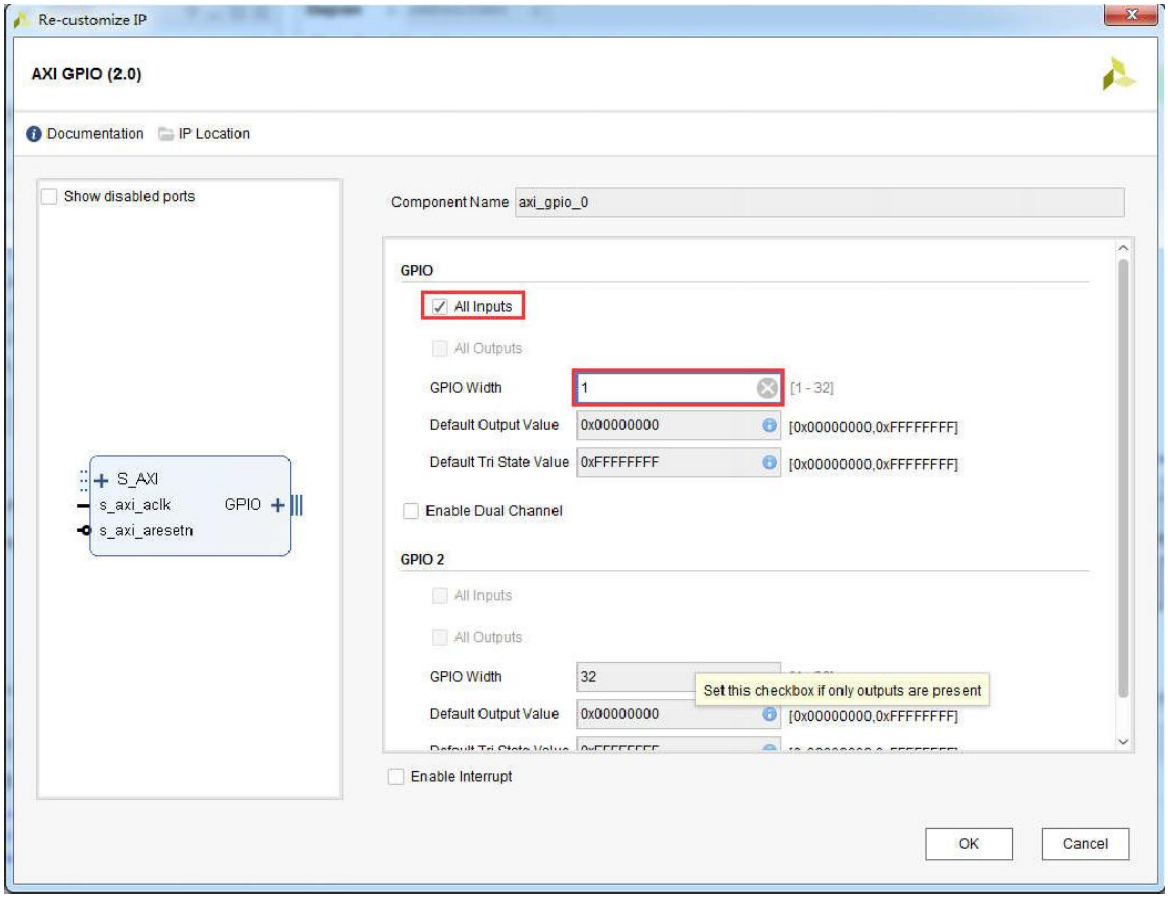
38) Select the AXI GPIO you just added, you can modify the module name in the Block Properties interface, here changed to "axi_led", need to note that if the name and the program are different, the software programming may need to modify the C language code of the routine, it is recommended for beginners to keep Consistent with the tutorial.



39)Add an AXI GPIO for key input

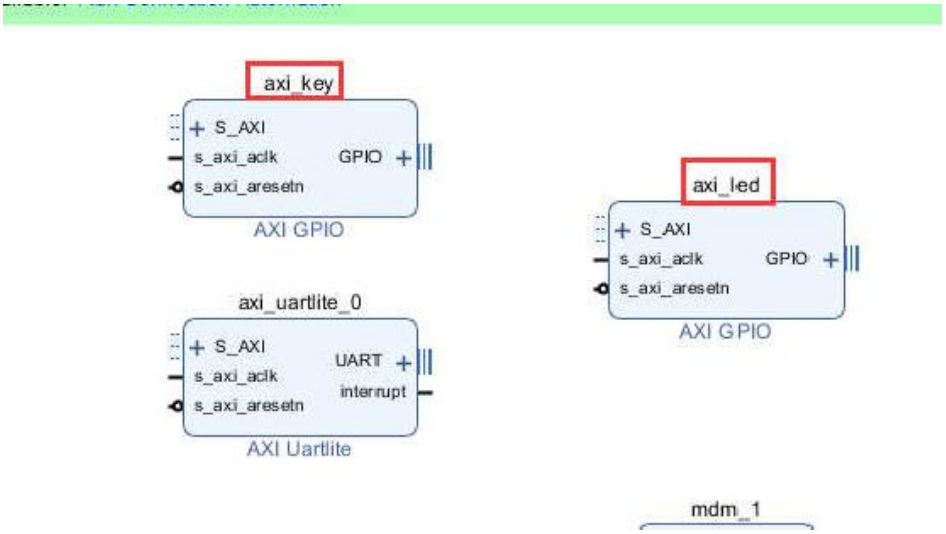


40)Modify the AXI GPIO parameters, check "All Inputs", fill in "GPIO Width", and only control the 1-bit button, click "OK" to complete the configuration.

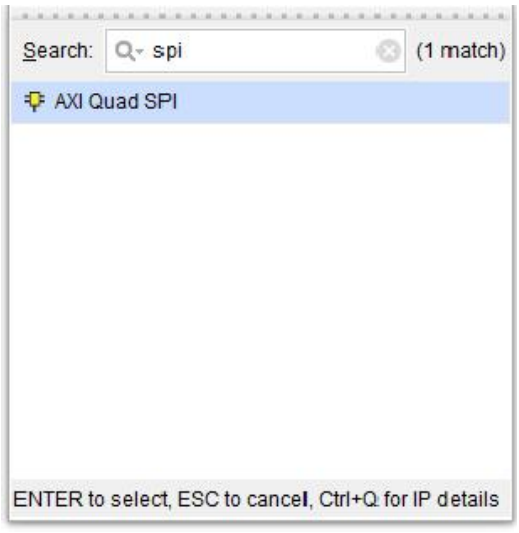


41)Modify the AXI GPIO parameters and check "All Inputs".Fill in

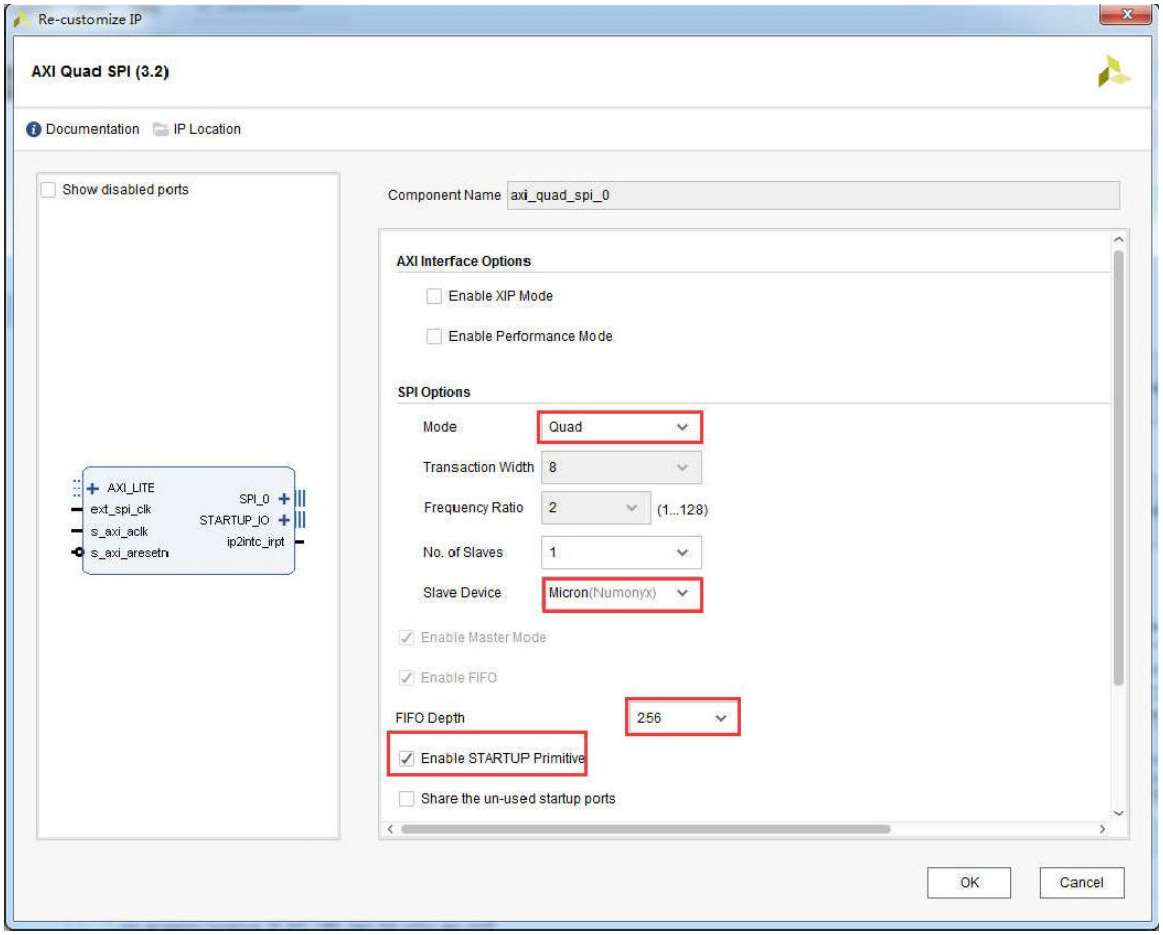
"GPIO Width" 1. Only control 1 button, click "OK" to complete the configuration.



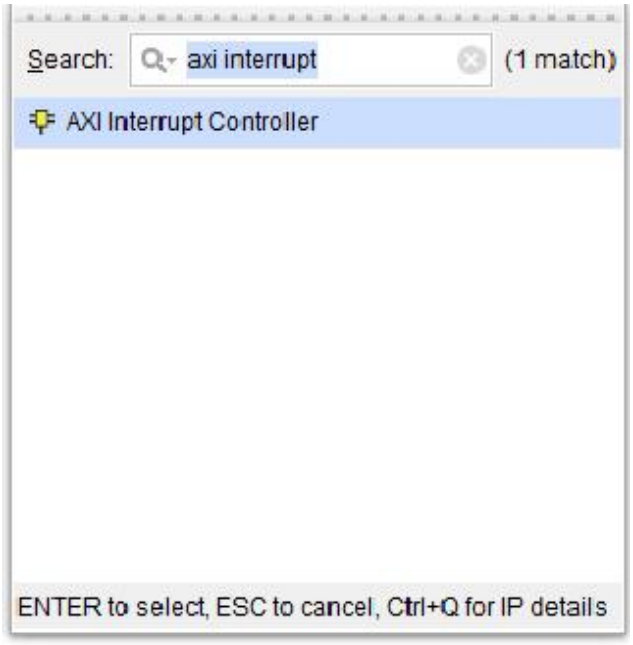
42) So spi, add "AXI Quad SPI" core for QSPI Flash on the control board



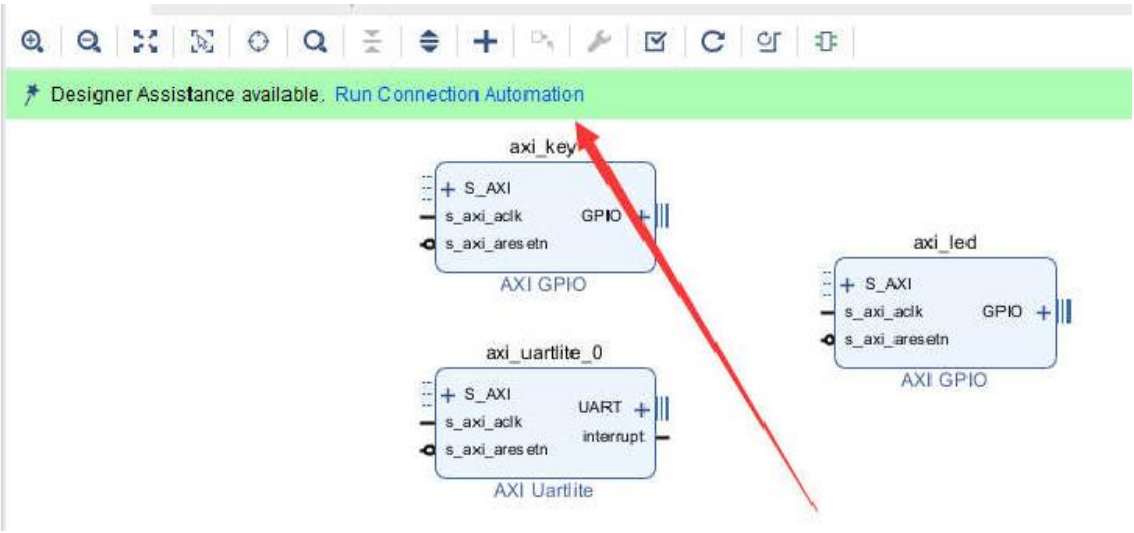
43) Double-click the IP core you just added to configure the parameters, select "Quad" for "Mode", "Micron" for "Slave Device", and "256" for "FIFO Depth", and check "Enable STARTUP Primitive"



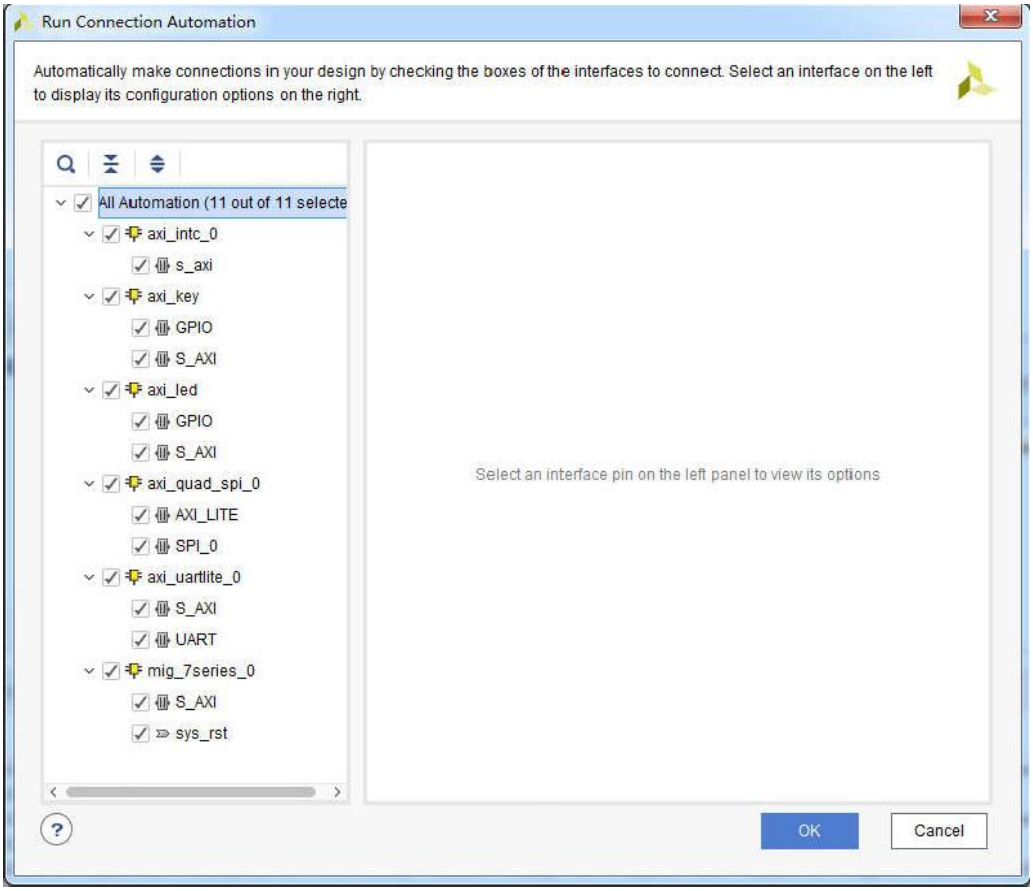
44)Search for "axi interrupt" and add "AXI Interrupt Controller"



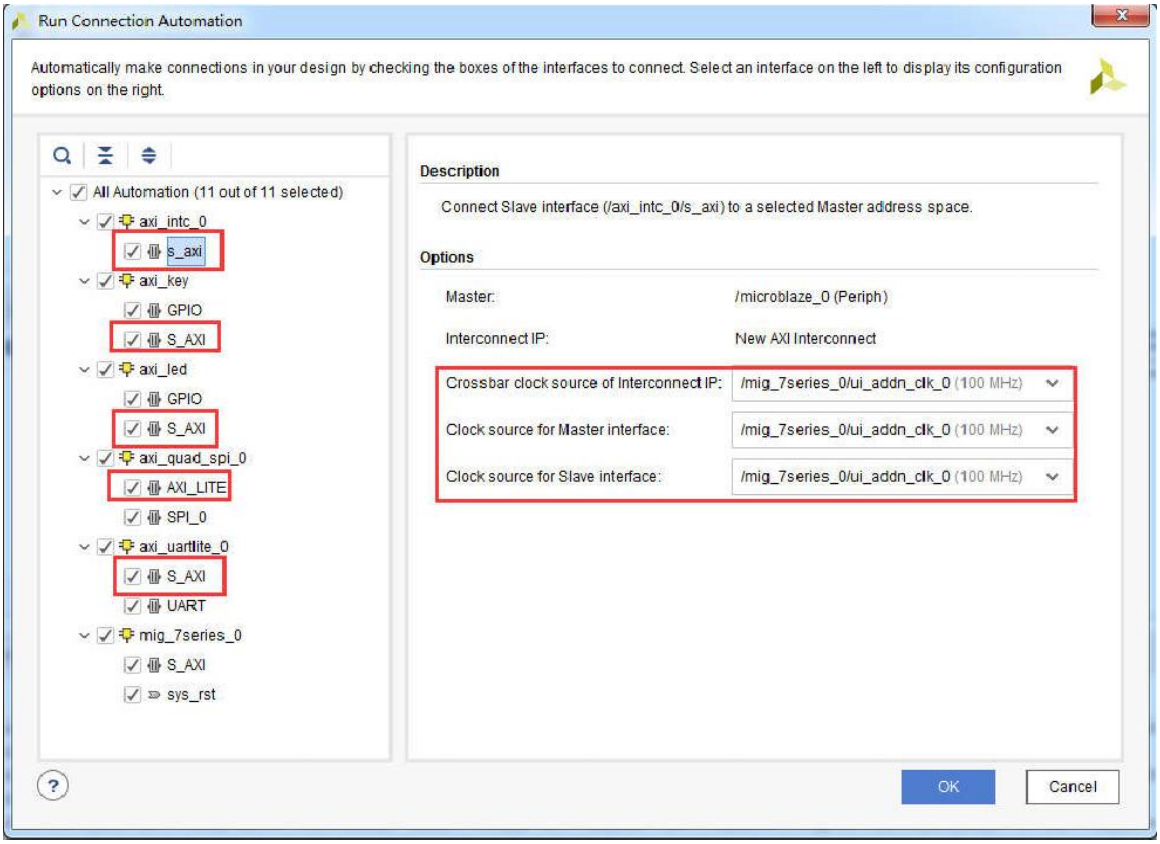
45)Click "Run Connection Automation" to automatically connect



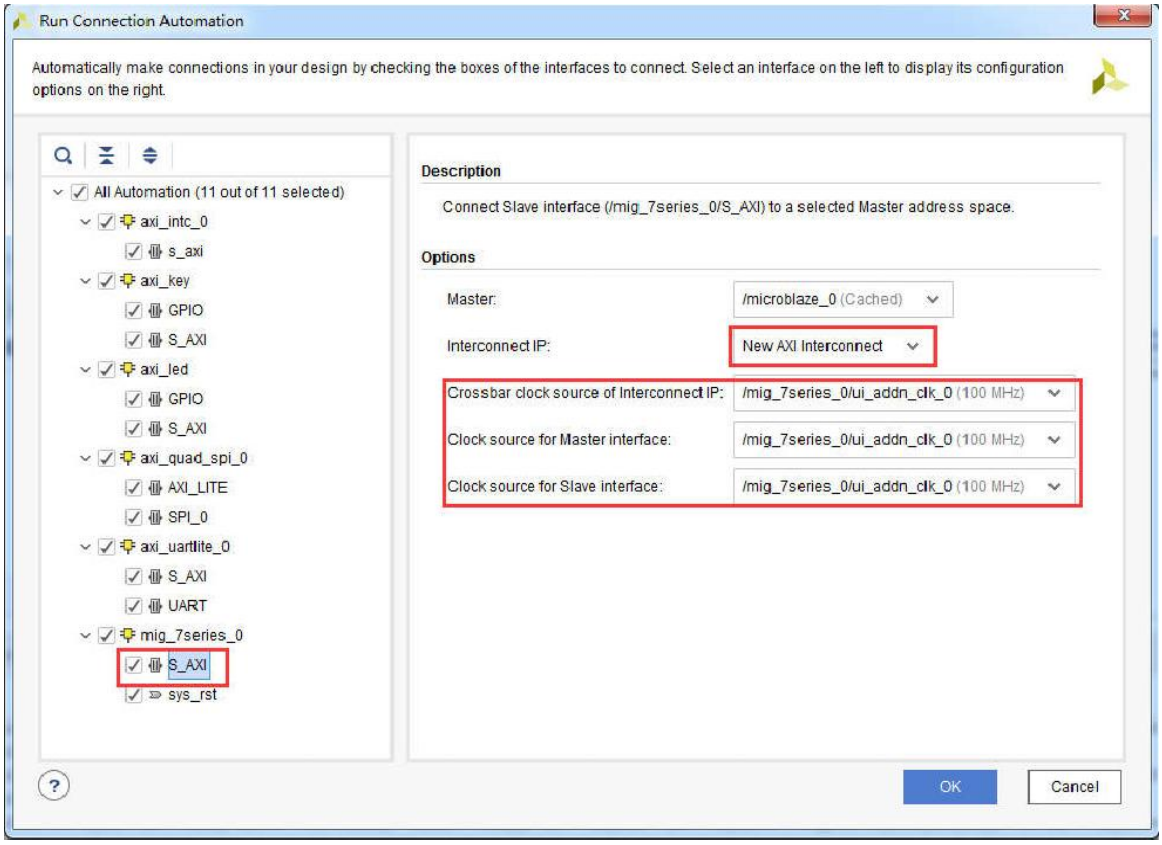
46) Select “All Automation” to automatically connect all the lines, Click “OK” to start the automatic connection.



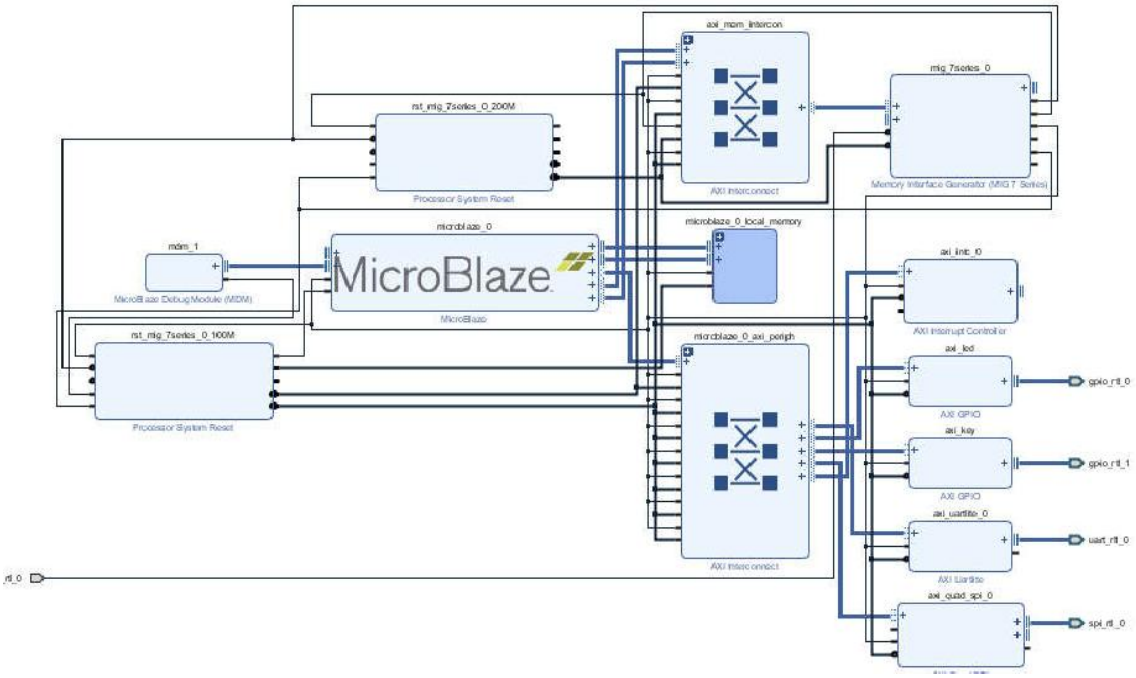
47) Manually select the clock, "Axi_intc_0", "axi_key" and other common AXI interface peripheral clocks "Crossbar clock source of interconnect IP", "Clock source for Master interface" and "Clock source for Slave interface" clocks select mig extra output 100Mh clock



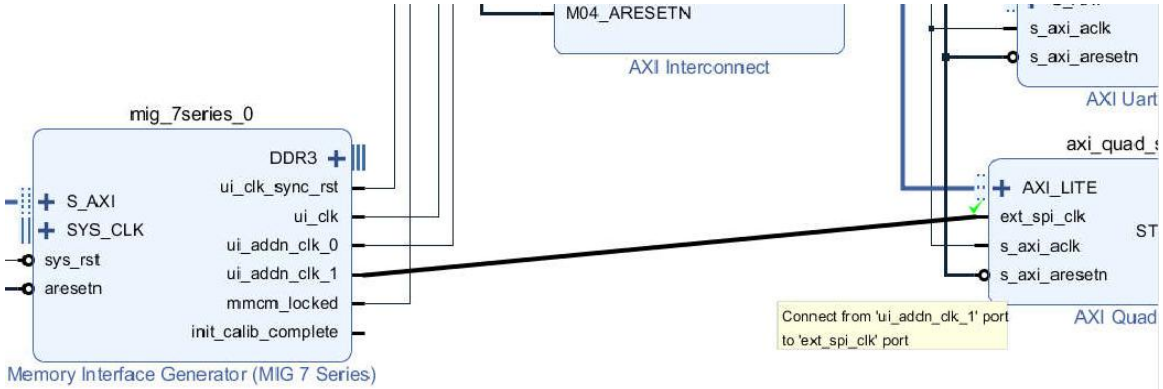
48) The "Mir_7series_0" AXI interface clock "Crossbar clock source of interconnect IP", "Clock source for Master interface" and "Clock source for Slave interface" clocks select the 100Mh clock for the extra output of mig, and "Interconnect IP" selects "New AXI Interconnect".



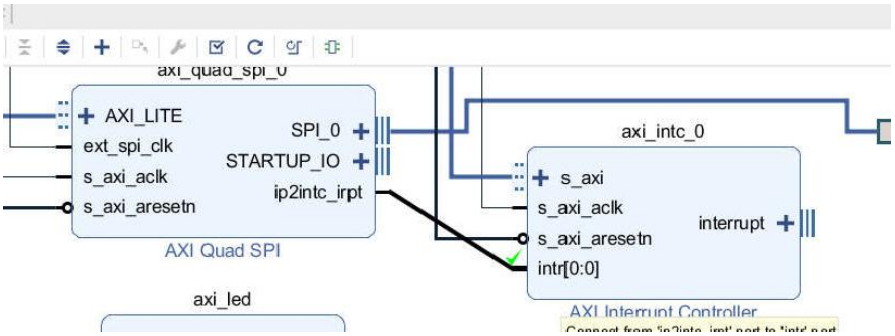
49)Automatic connection completed



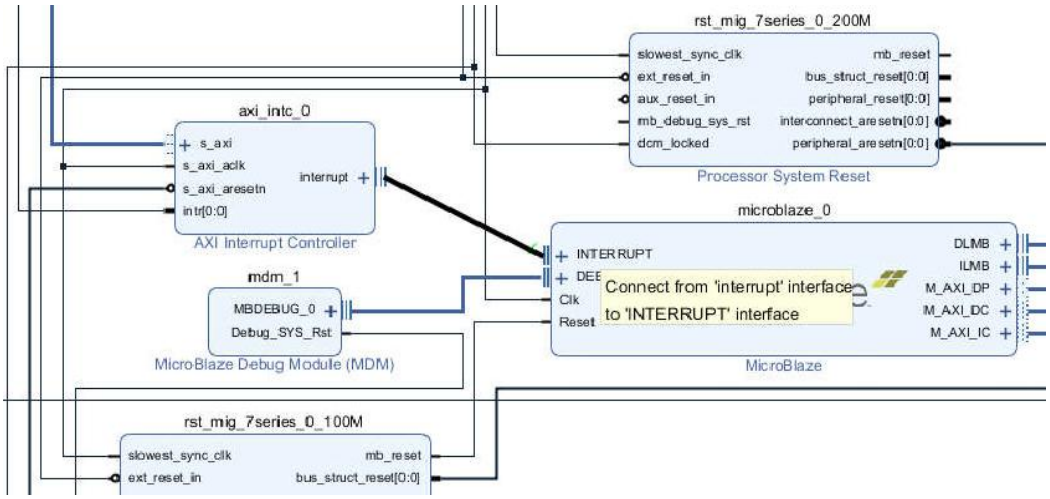
50)Connect ui_addn_clk_1 of mig_7series_0 to ext_spi_clk of axi_quad_spi_0



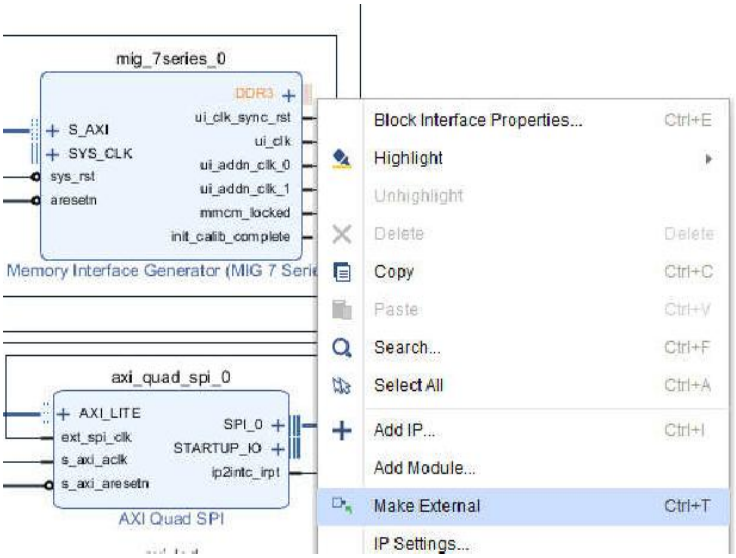
51)Connect ip2intc_irpt of axi_quad_spi_0 to intr of axi_intc_0



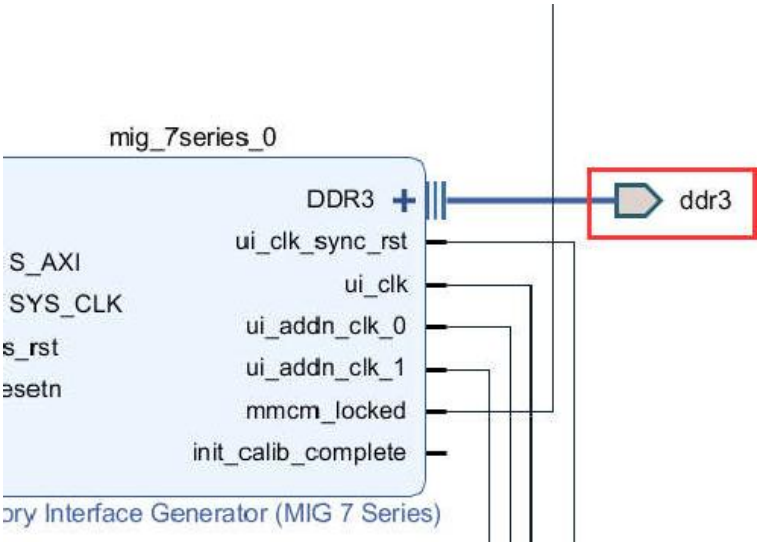
52)Connect the interrupt of axi_intc_0 to the INTERRUPT of microblaze_0



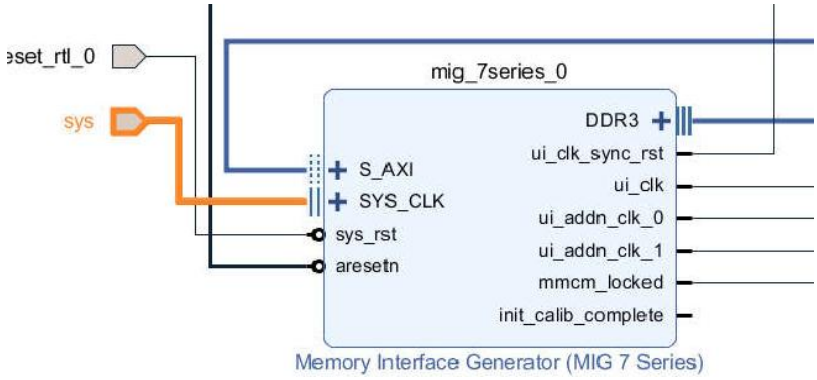
53)Export the DDR3 port of mig_7series_0



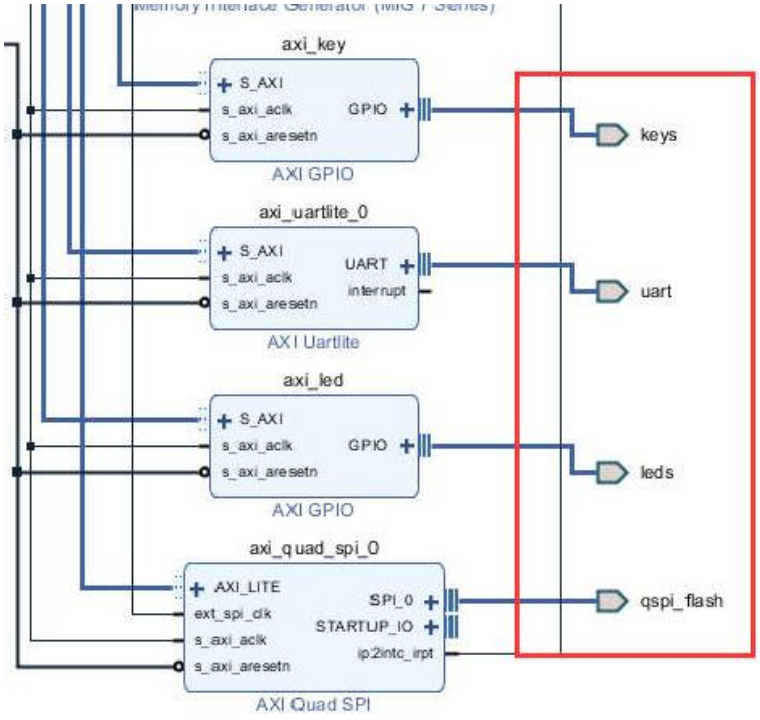
54) The port name is changed to lowercase "ddr3". If it is changed to other, it may not be compiled.



55) Export SYS_CLK of mig_7series_0 and modify the name to "sys". Other names may not compile.

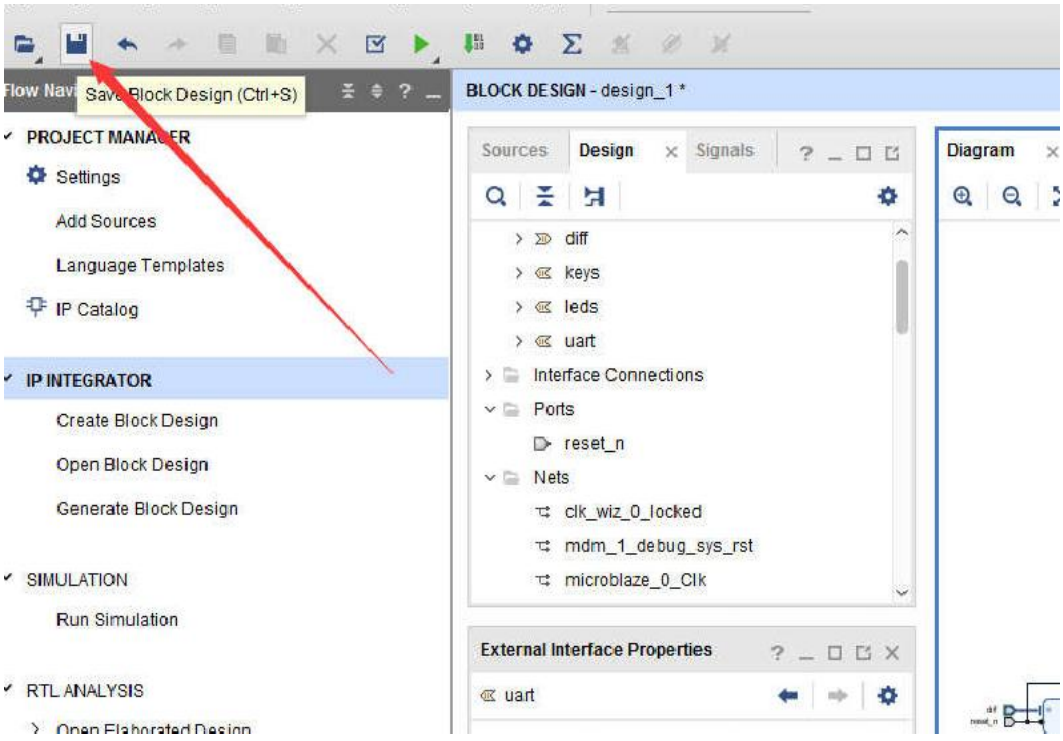


56)Modify other port names, change "reset_rtl_0" to "reset_n", "gpio_rtl_1" to "keys", "gpio_rtl_0" to "leds", "uart_rtl_0" to "uart", "spi_rtl_0" to "qspi_flash"

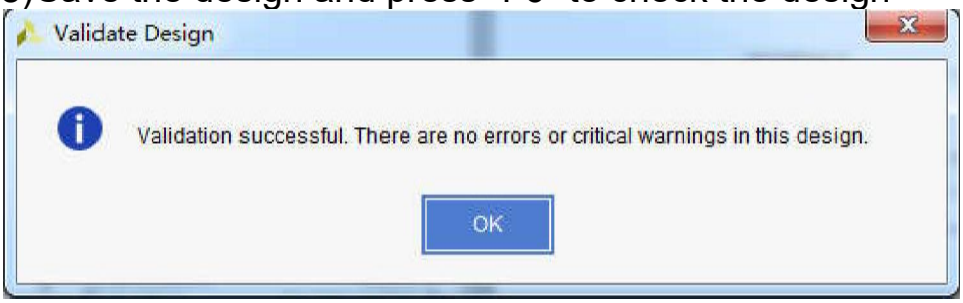


Part 3.1.6 Save and check for errors

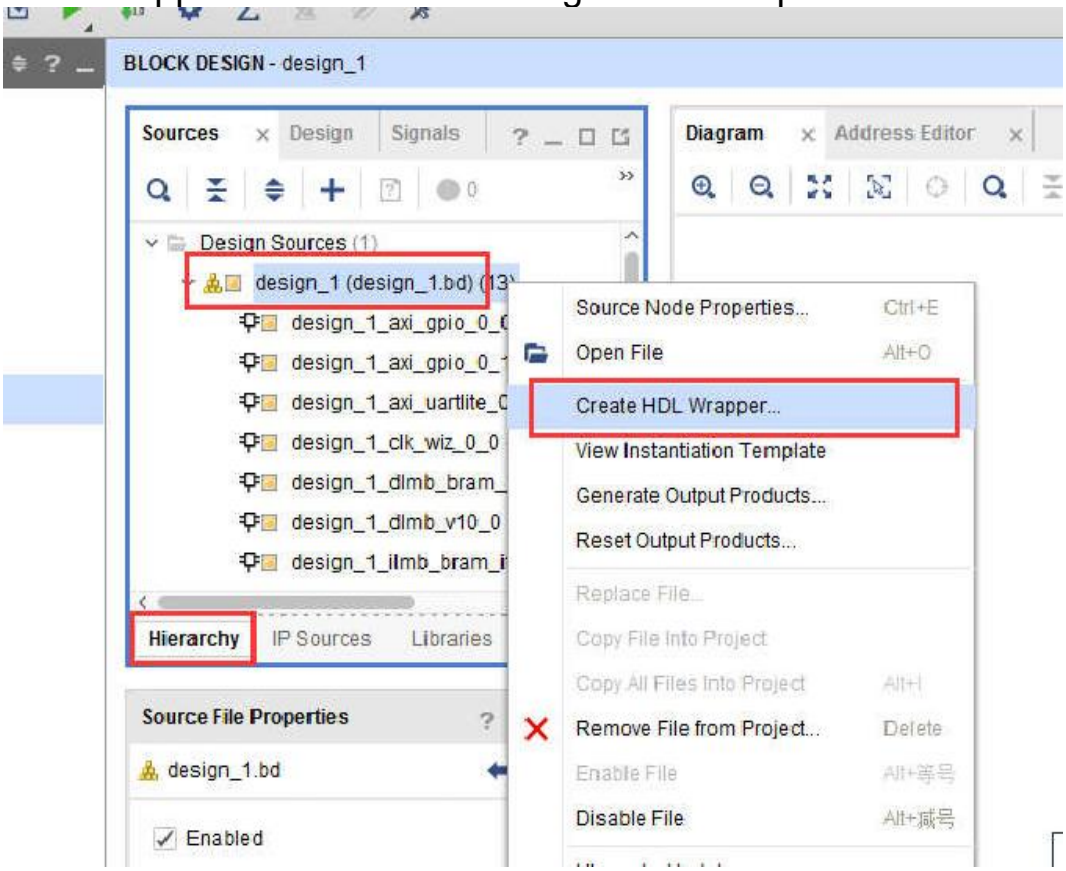
57)Save the design and press "F6" to check the design



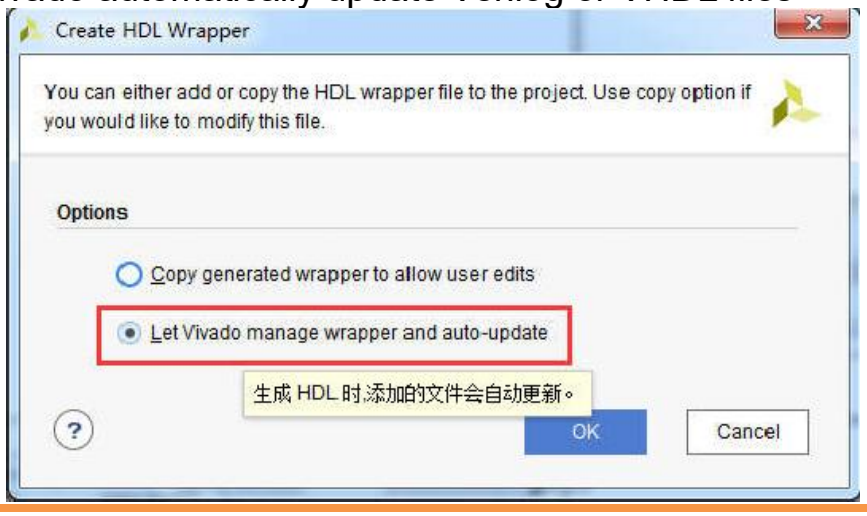
58)Save the design and press "F6" to check the design



59)Select the "design_1.bd" file and right click on "Create HDL Wrapper..." to create a Verilog or VHDL top-level file.



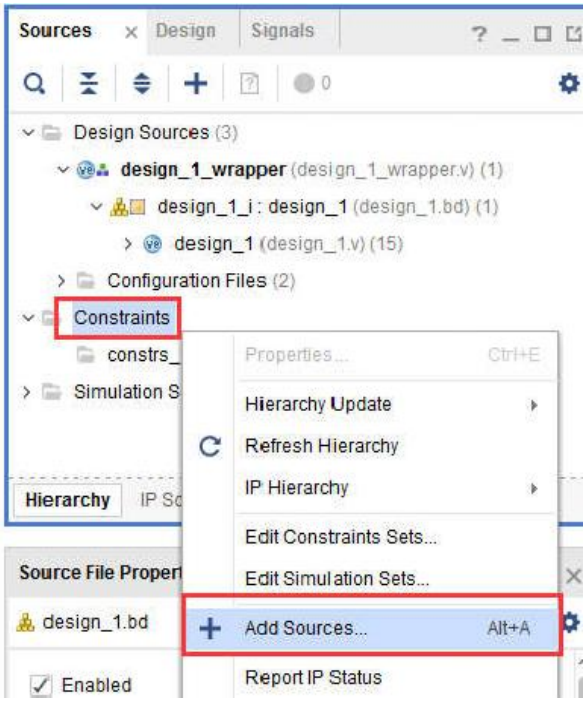
60)Select "Let Vivado manage wrapper and auto-update" to have Vivado automatically update Verilog or VHDL files

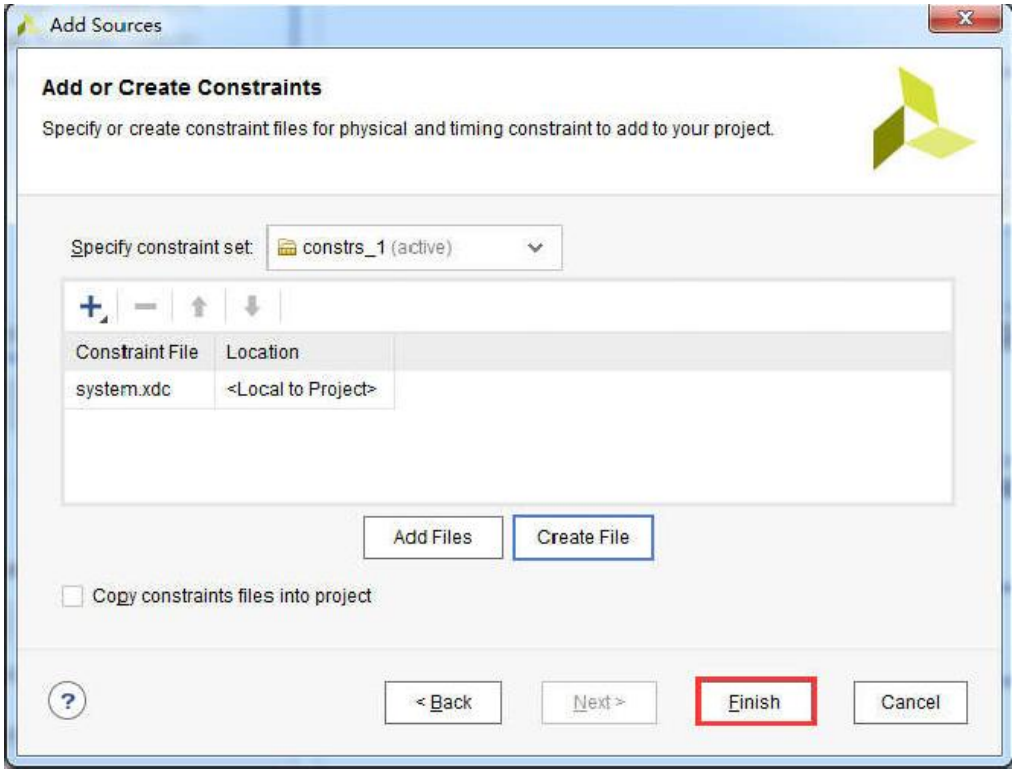


61) Double-click to open the generated "design_1_wrapper.v", you can see some ports, and assign IO to these ports below.

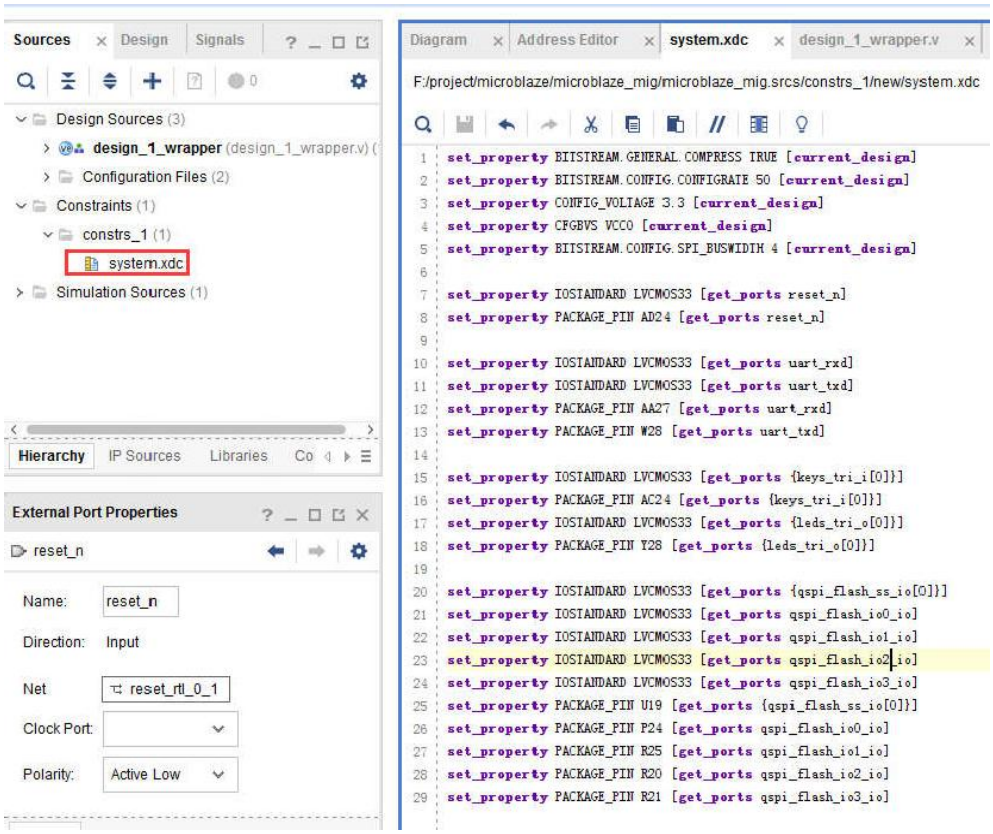
Part 3.1.7: Add an xdc constraint file

62) Select "Constrains" and right click on "Add Sources..."





63)Modify the contents of the xdc file, please double check that the port name in xdc is consistent with the port in "design_1_wrapper.v

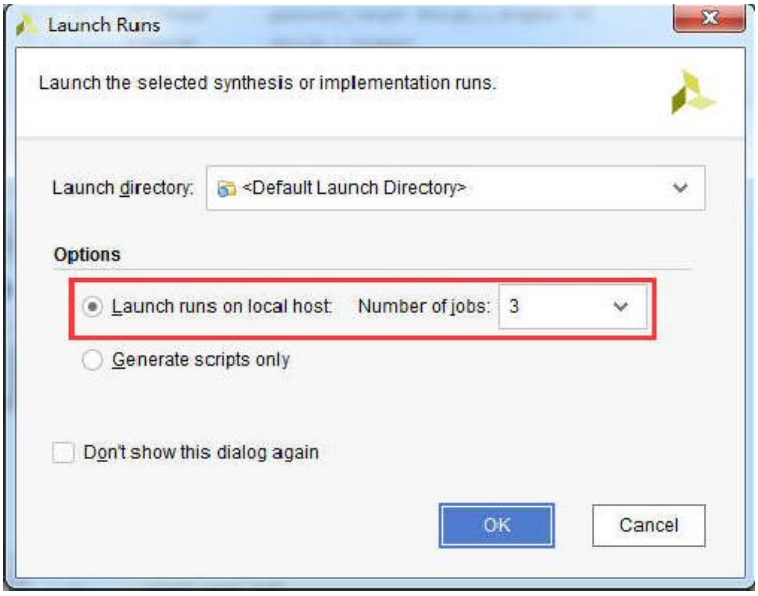


```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]

set_property IOSTANDARD LVCMOS33 [get_ports reset_n]
set_property PACKAGE_PIN AD24 [get_ports reset_n]

set_property IOSTANDARD LVCMOS33 [get_ports uart_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_txd]
set_property PACKAGE_PIN AA27 [get_ports uart_rxd]
set_property PACKAGE_PIN W28 [get_ports uart_txd]

set_property IOSTANDARD LVCMOS33 [get_ports {keys_tri_i[0]}]
set_property PACKAGE_PIN AC24 [get_ports {keys_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds_tri_o[0]}]
set_property PACKAGE_PIN Y28 [get_ports {leds_tri_o[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {qspi_flash_ss_io[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports qspi_flash_io0_io]
set_property IOSTANDARD LVCMOS33 [get_ports qspi_flash_io1_io]
set_property IOSTANDARD LVCMOS33 [get_ports qspi_flash_io2_io]
set_property IOSTANDARD LVCMOS33 [get_ports qspi_flash_io3_io]
set_property PACKAGE_PIN U19 [get_ports {qspi_flash_ss_io[0]}]
set_property PACKAGE_PIN P24 [get_ports qspi_flash_io0_io]
set_property PACKAGE_PIN R25 [get_ports qspi_flash_io1_io]
set_property PACKAGE_PIN R20 [get_ports qspi_flash_io2_io]
set_property PACKAGE_PIN R21 [get_ports qspi_flash_io3_io]
```

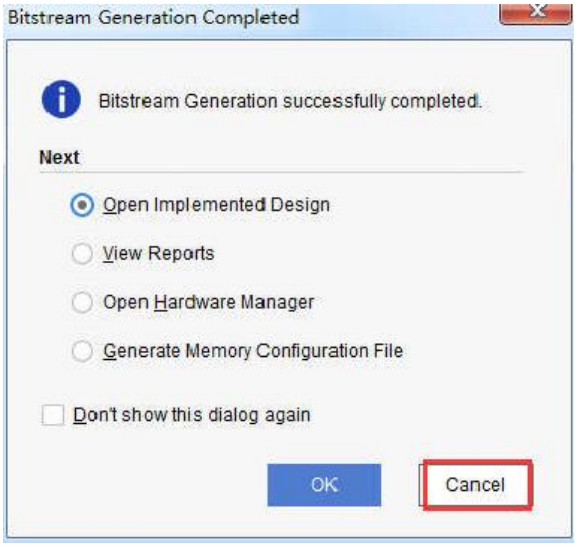
Part 3.1.8: Generate Bitstream File

64) Modify the contents of the xdc file, please double check that the port name in xdc is consistent with the port in "design_1_wrapper.v"

65) In the pop-up window, there is a running path selection. By default, "Number of jobs" can be selected according to its own CPU. The larger the number, the faster the compilation.

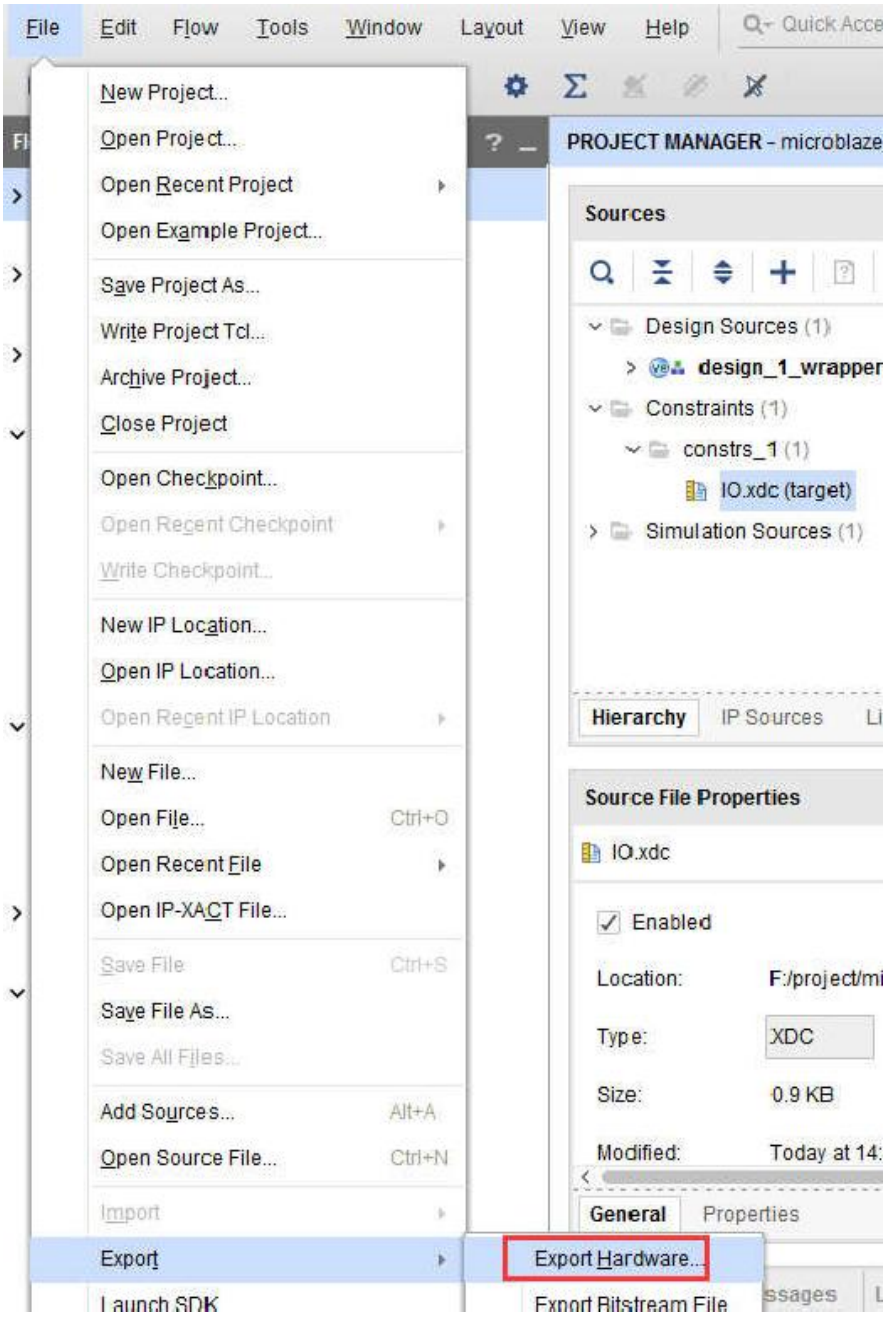


66)After generating Bitstream, click "Cancel" and do nothing.

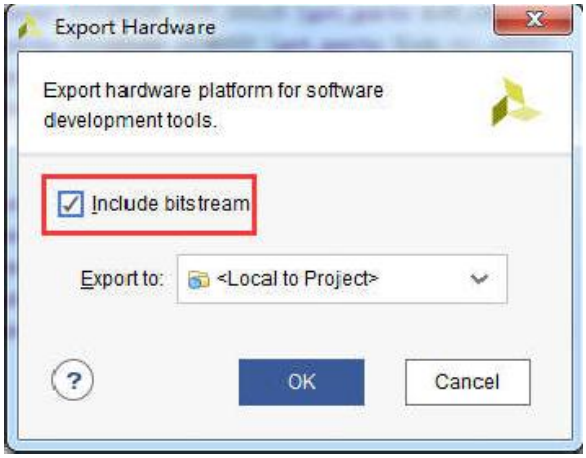


Part 3.1.9: Export hardware

67)Export hardware in "File -> Export -> Export Hardware..."

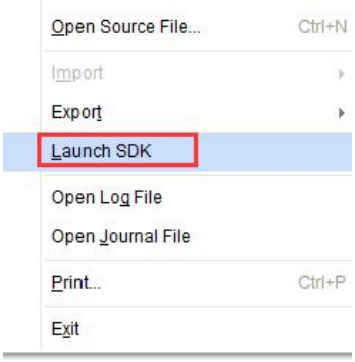


68) Select "Include bitstream" in the pop-up window to include the Bitstream file.

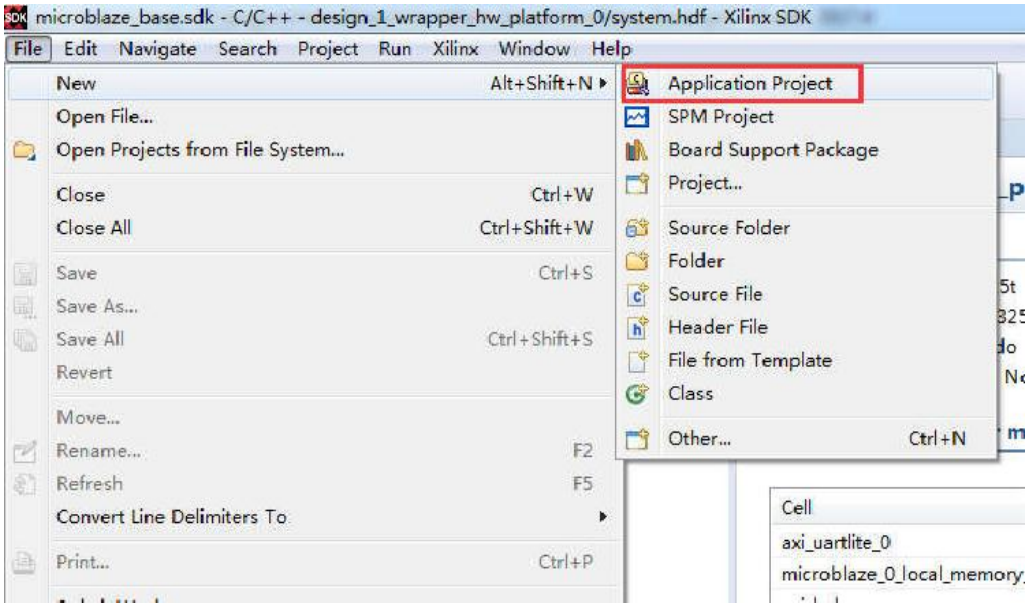


Part 3.2: SDK development "HelloWorld"

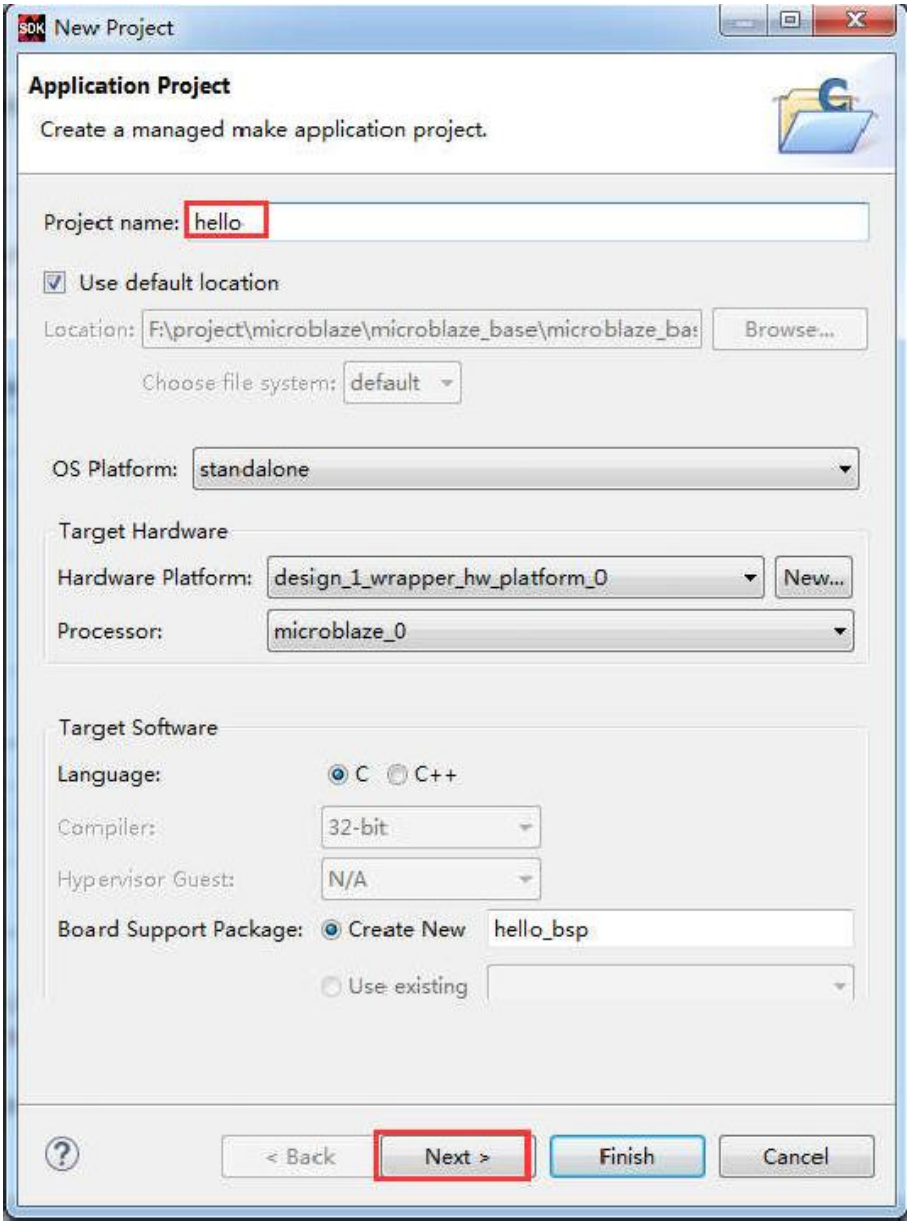
1) Click File→Launch SDK to enter the SDK operation



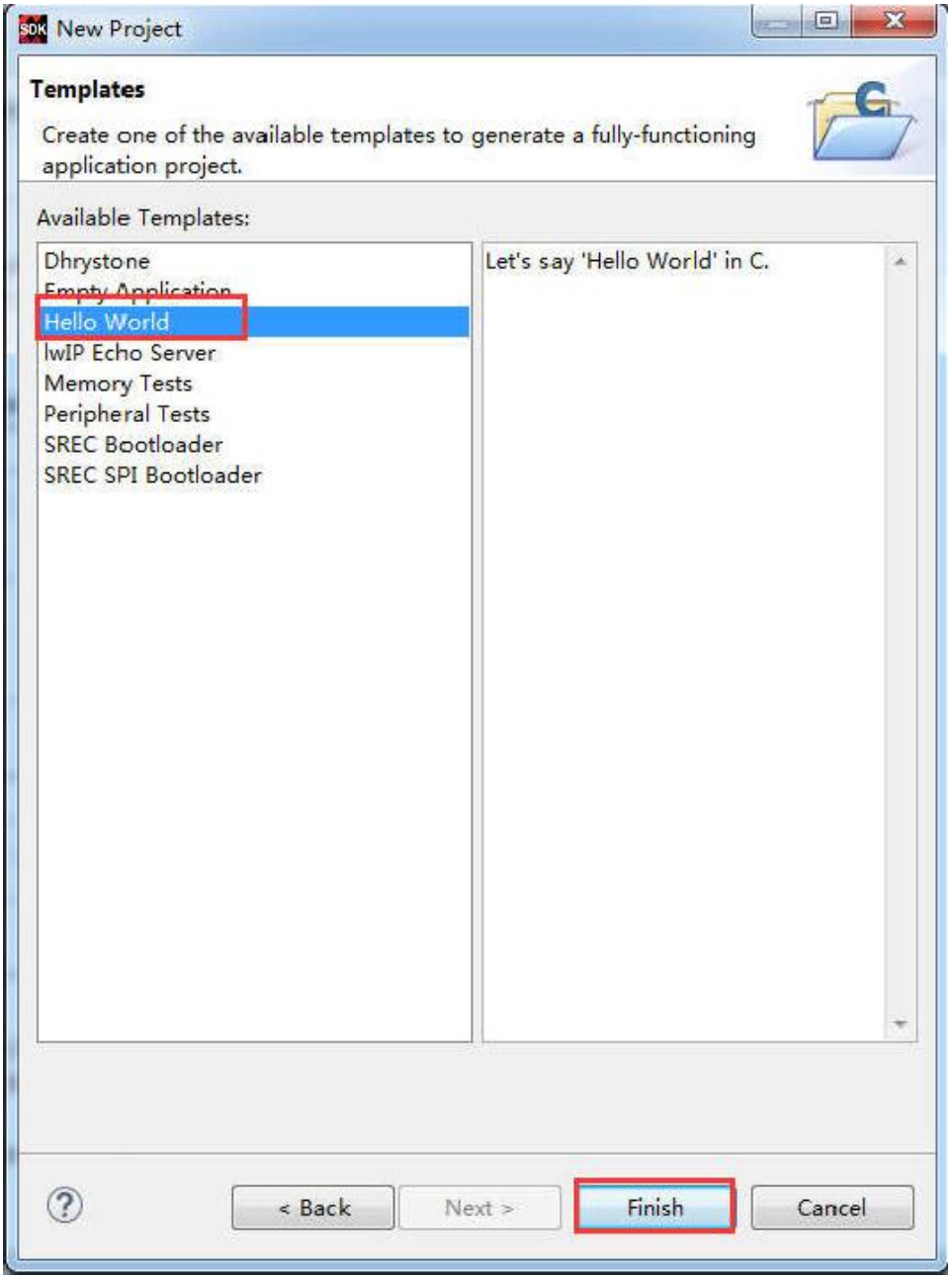
2) Create a new project



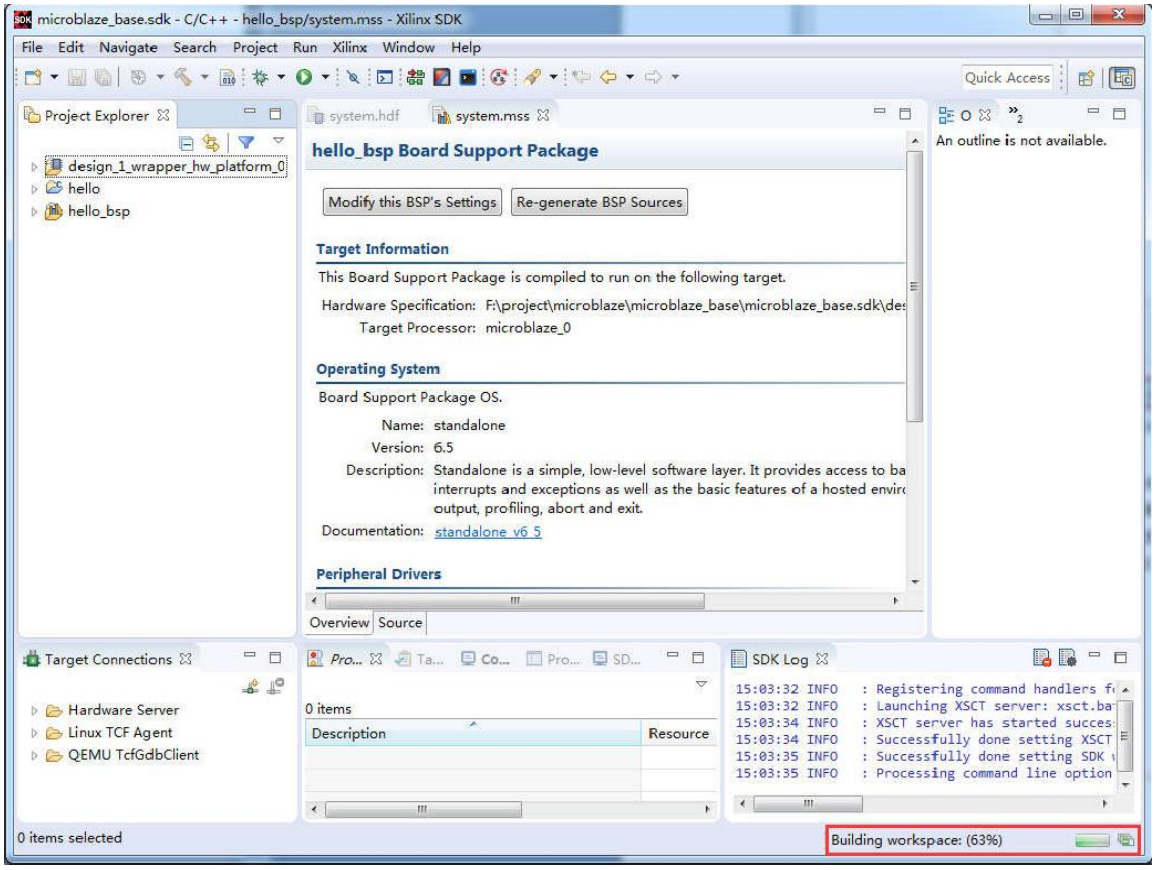
3) Set the project name to "hello" and click "Next"



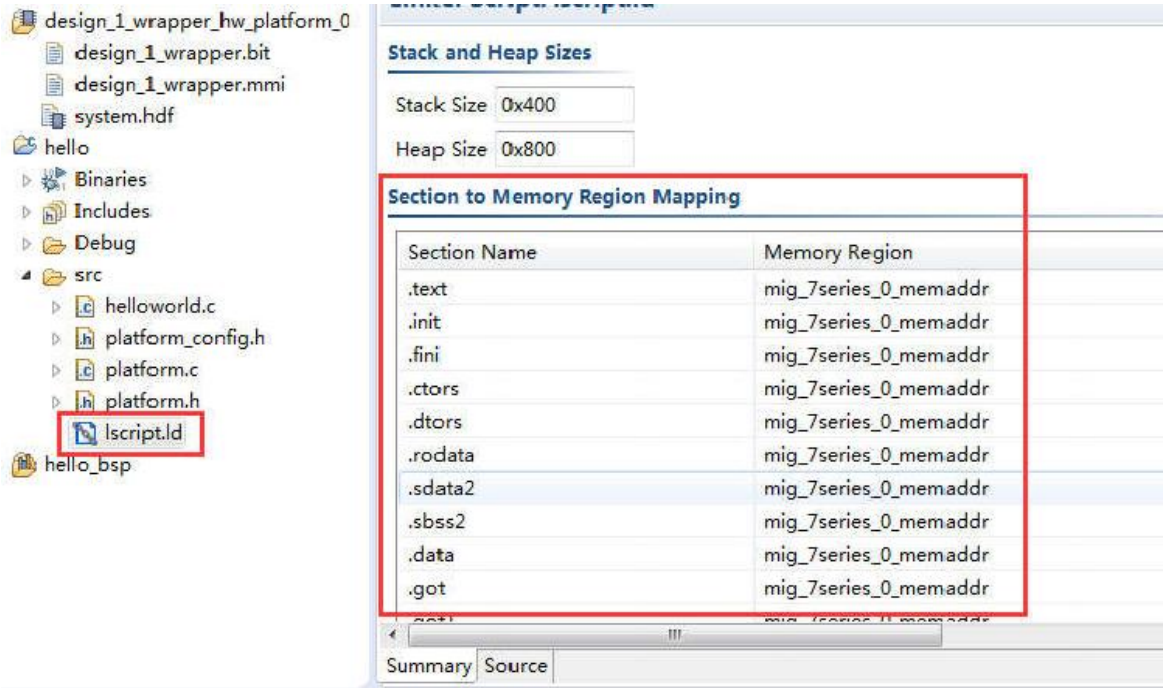
4) Select "Hello World" for the template and click "Finish"



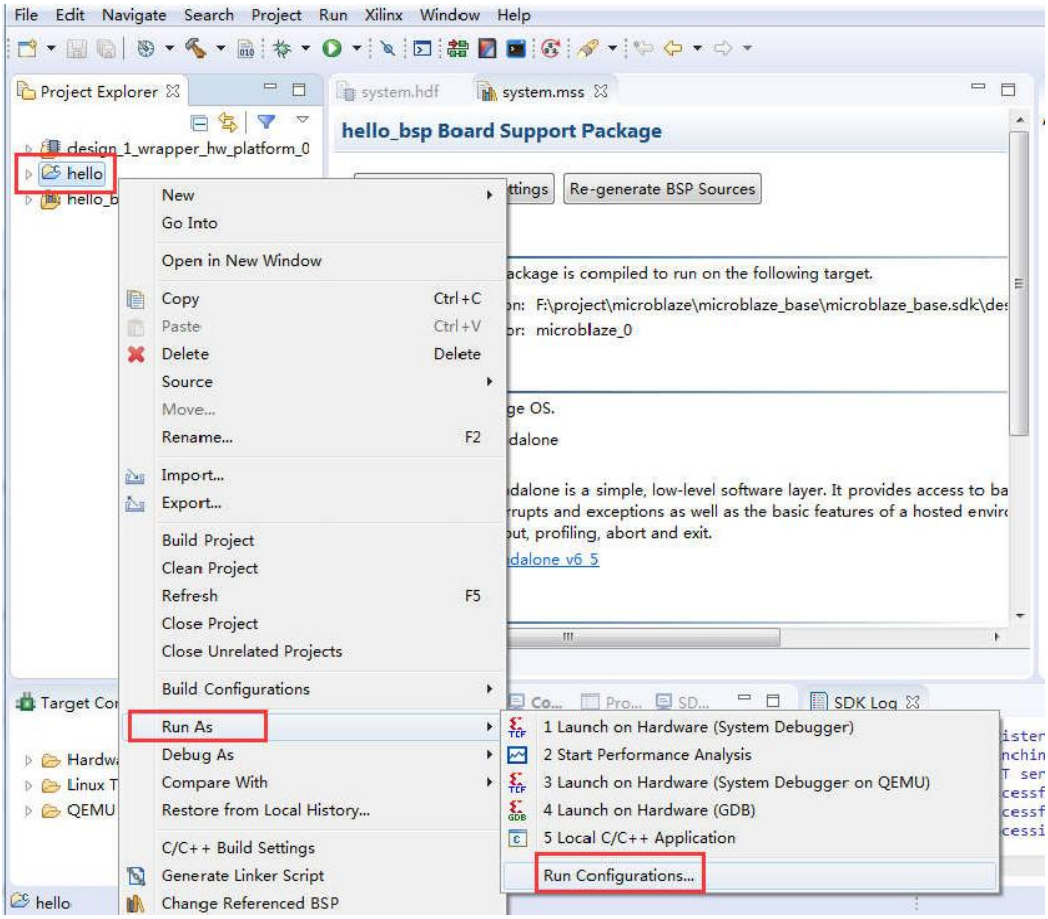
5) After the project is built, it will be compiled automatically, and it can be debugged after waiting for the compilation to complete.



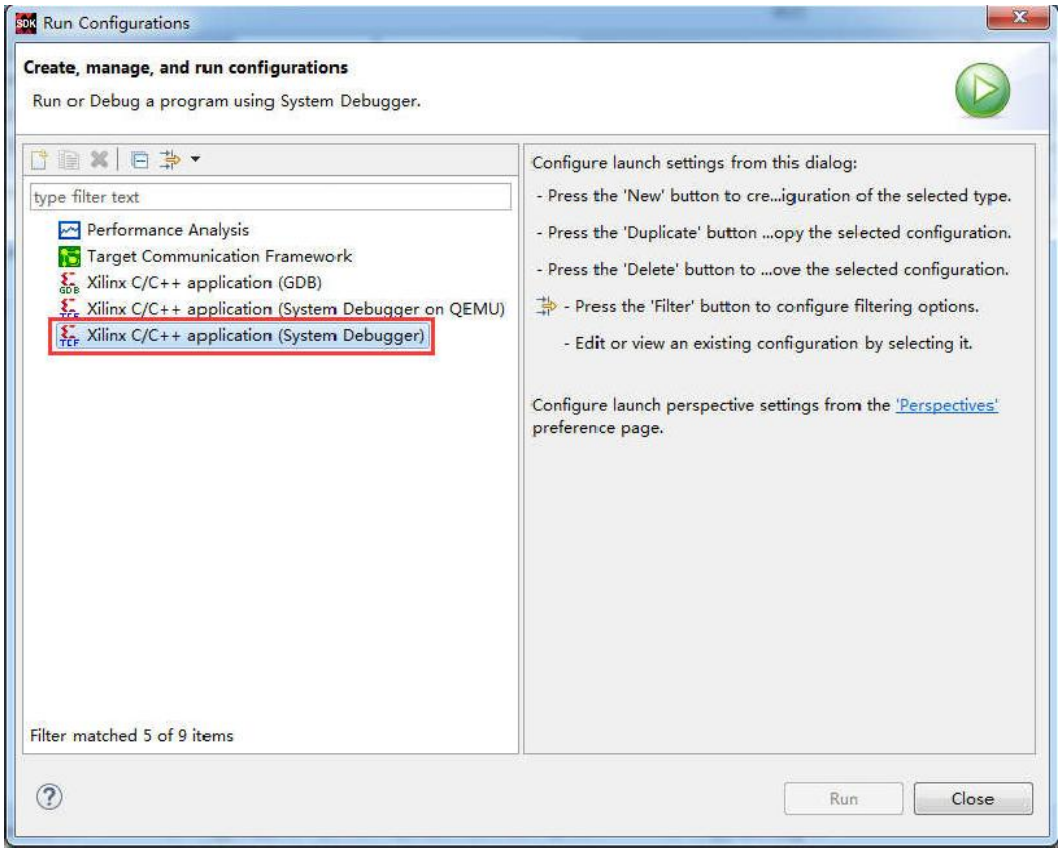
6) Double-click the "lscript.ld" file to view the stack settings. The Hello project runs on ddr by default.



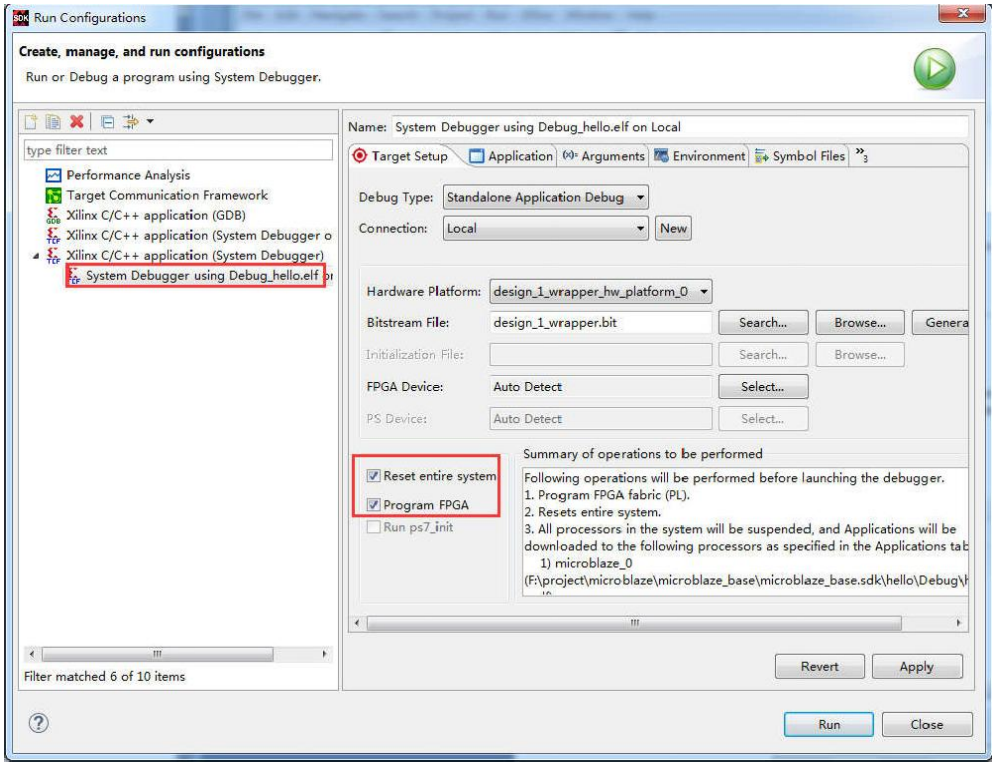
Select the "Hello" project and right click on "Run As -> Run Configuration..."



7) Double click on "Xilinx C/C++ application(System Debugger)"

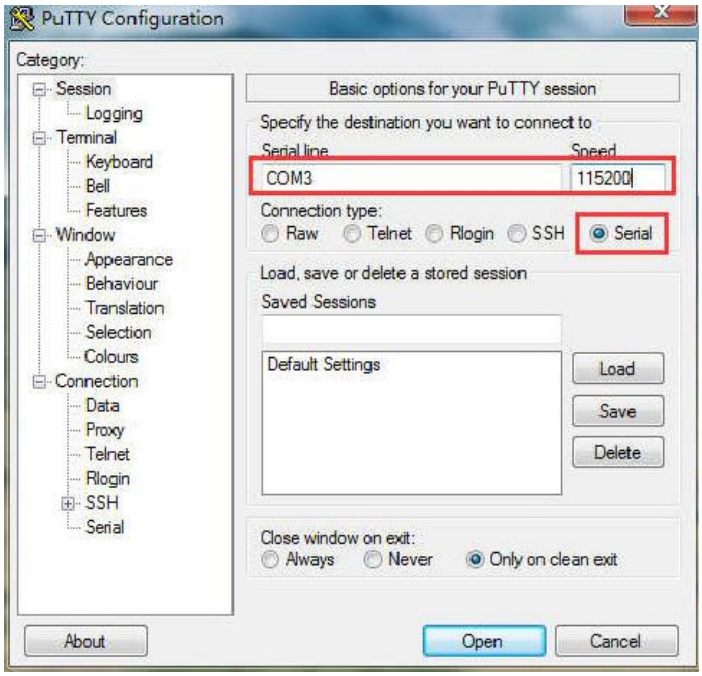


8) Check "Reset entire system" and "Program FPGA" to reset the entire system and download the FPGA

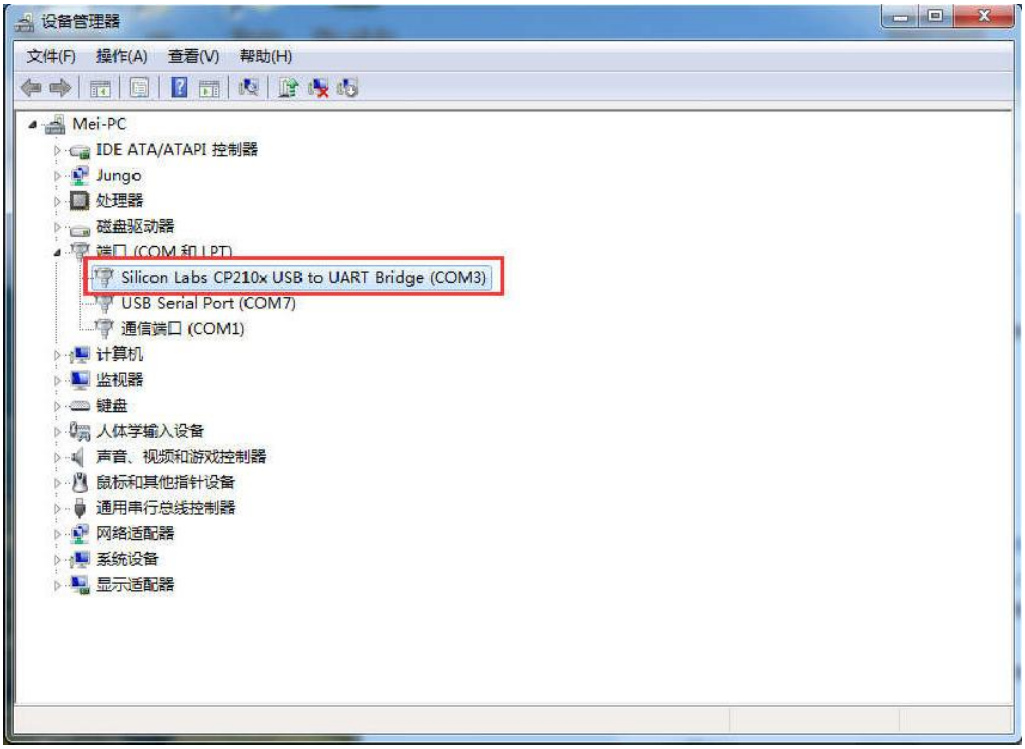


9) Connect the JTAG cable to the development board, UART USB cable to PC

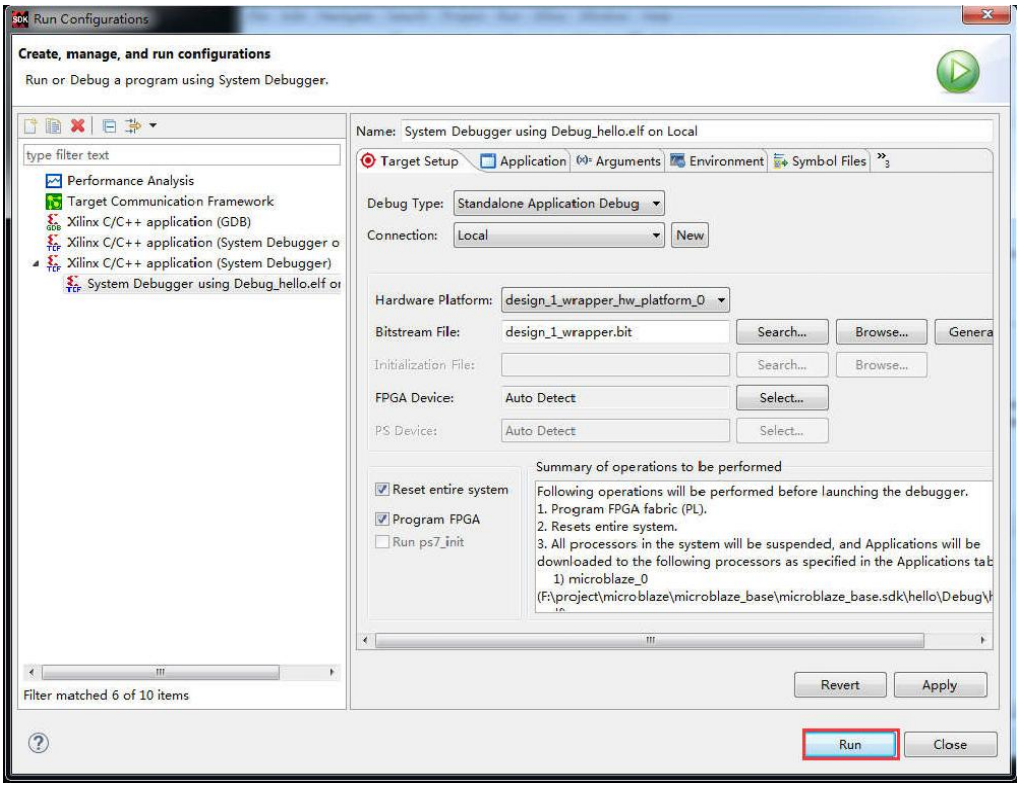
10) Using PuTTY software as a serial terminal debugging tool, PuTTY is an installation-free small software.



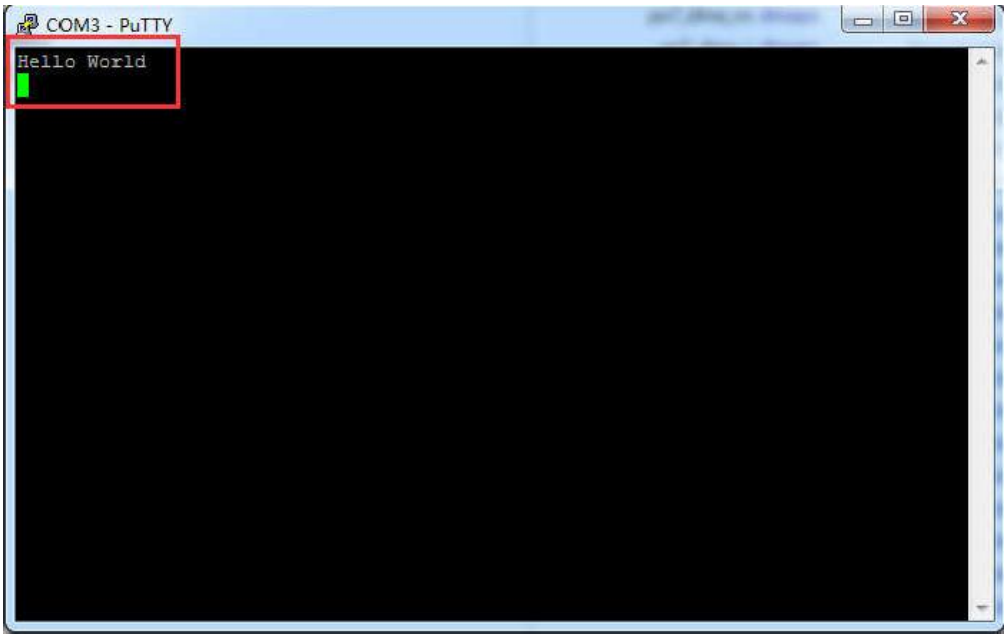
11)Select Serial, Serial line to fill in COM3, Speed fill in 115200, COM3 serial port number is filled in according to the display in the device manager, click "Open"



12)Click on "Run"



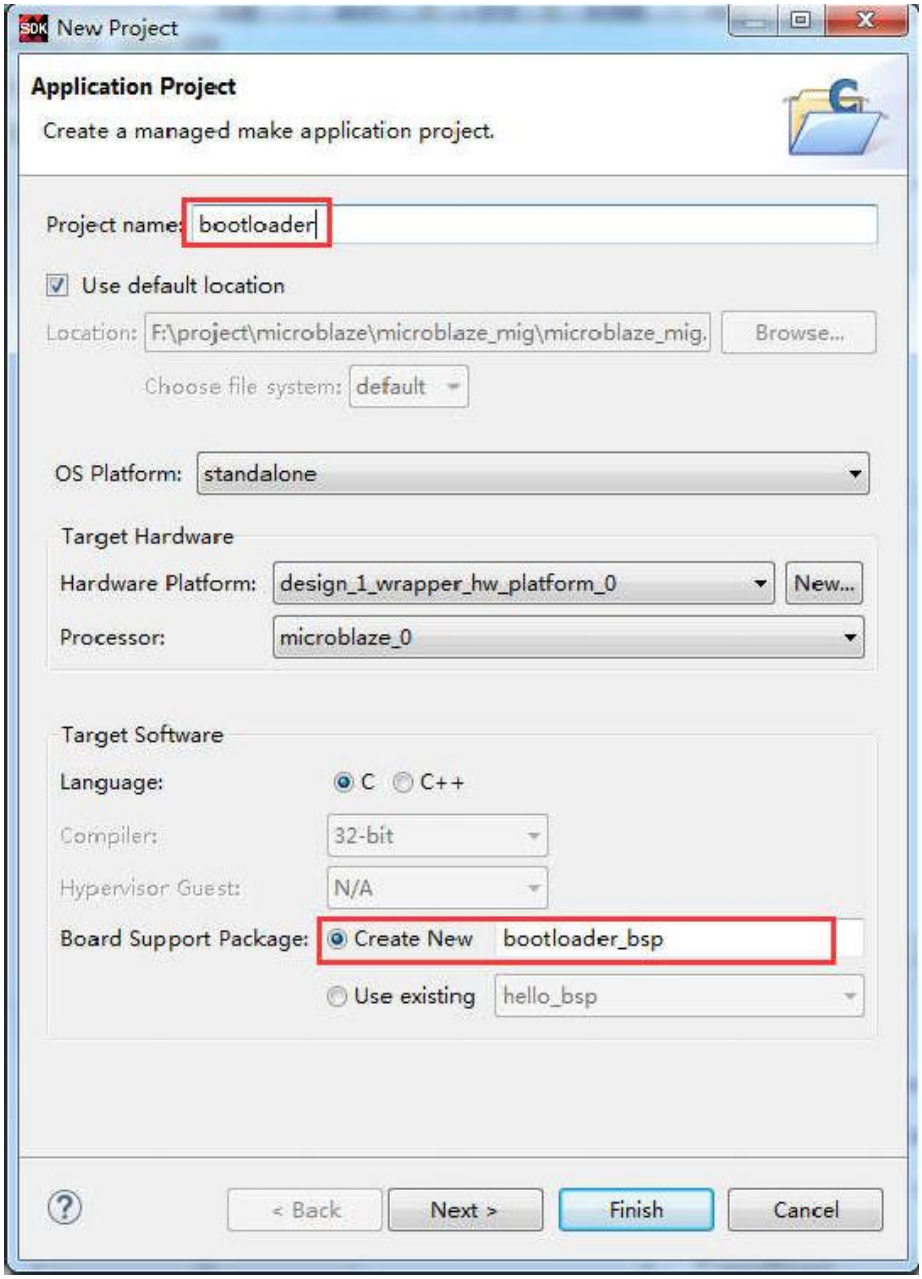
13) "Hello World" is displayed



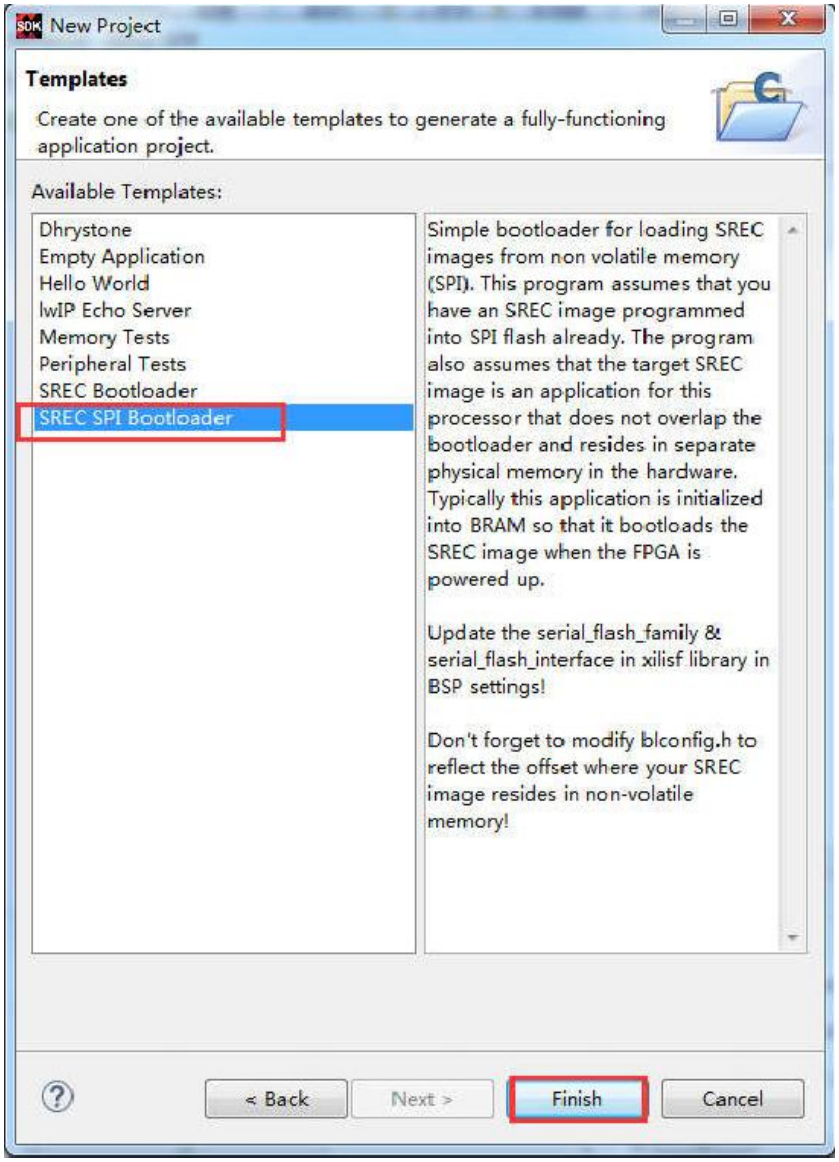
Part 3.3: Programming program to Flash

Part 3.3.1: BootLoader project

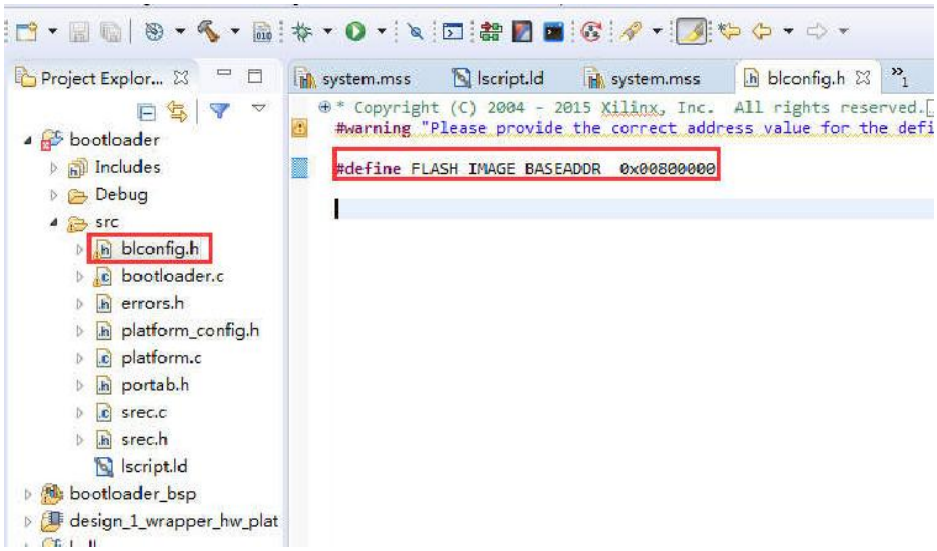
1) Create a bootloader project



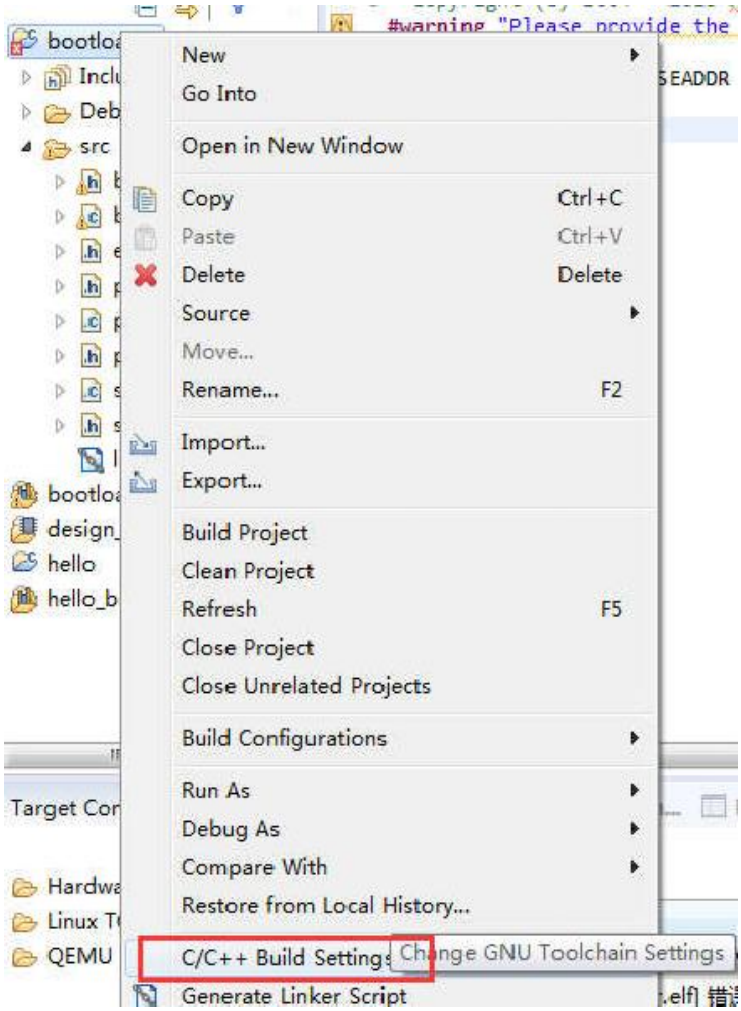
2) Select the "SREC SPI Bootloader" template and click "Finish"



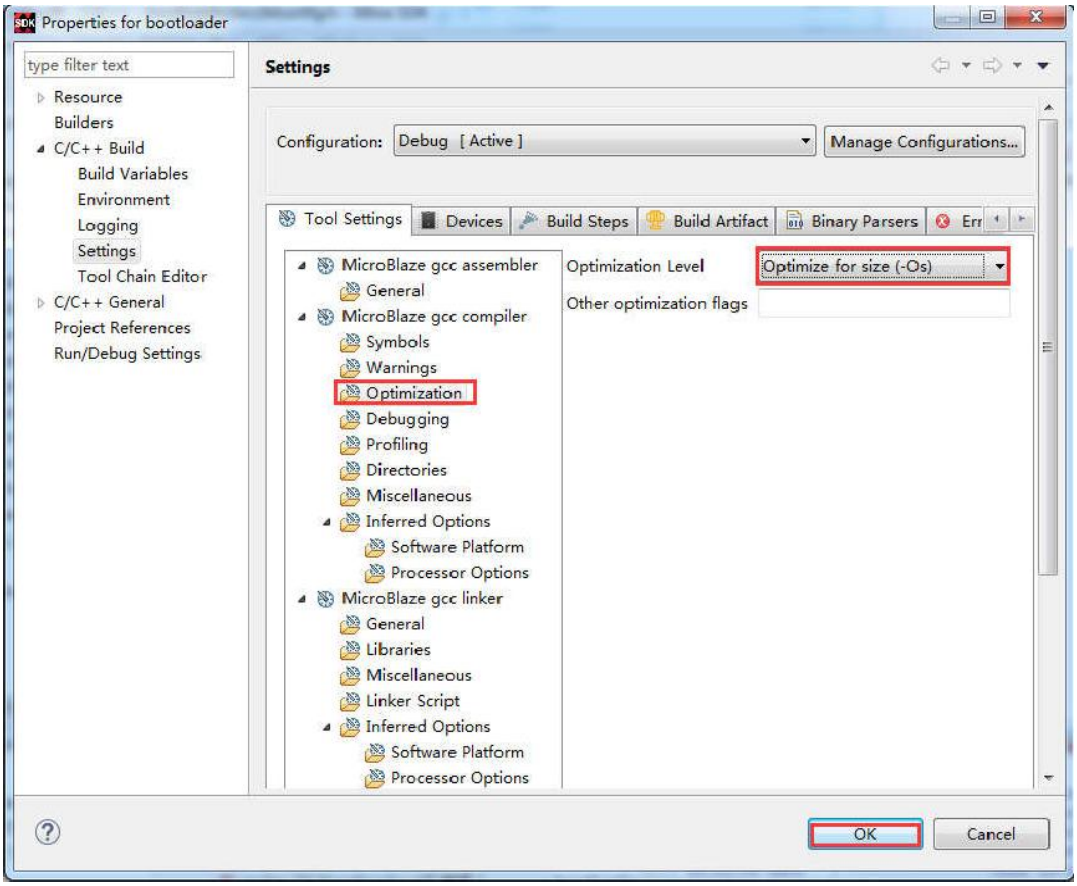
3) Modify the blconfig.h file to change the load point of the Microblaze software file in Flash to "0x00800000", which is 8MB in front of the Flash to save the Bitstream file, and 8MB to start saving software



4) Bootloader auto-compilation error, because the on-chip RAM is allocated 16K, to configure optimization options, reduce the code size, select the project right button, click "C / C + + Build Settings"

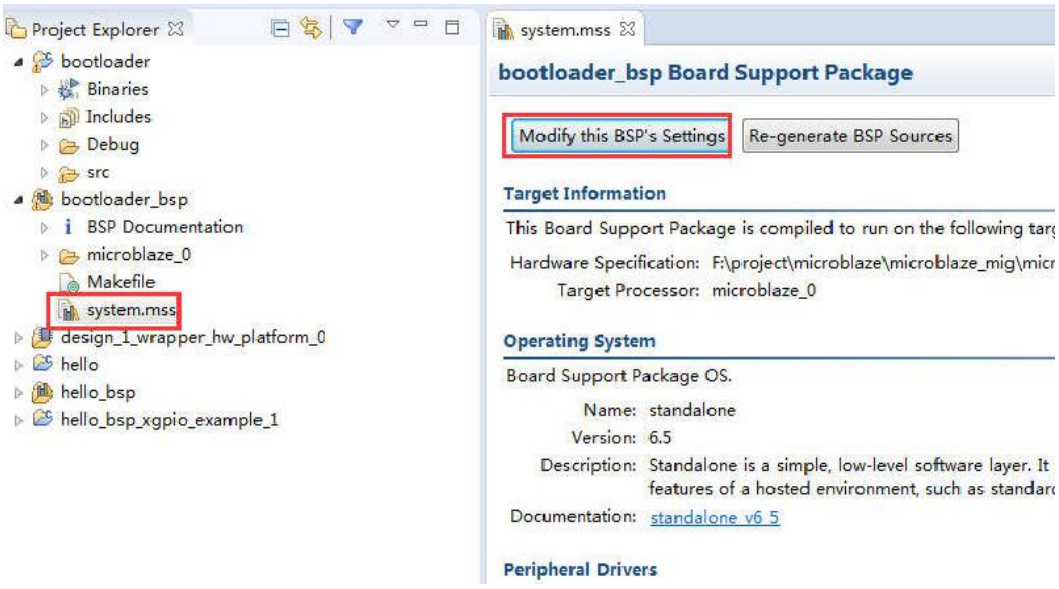


5) The optimization level is set to –Os

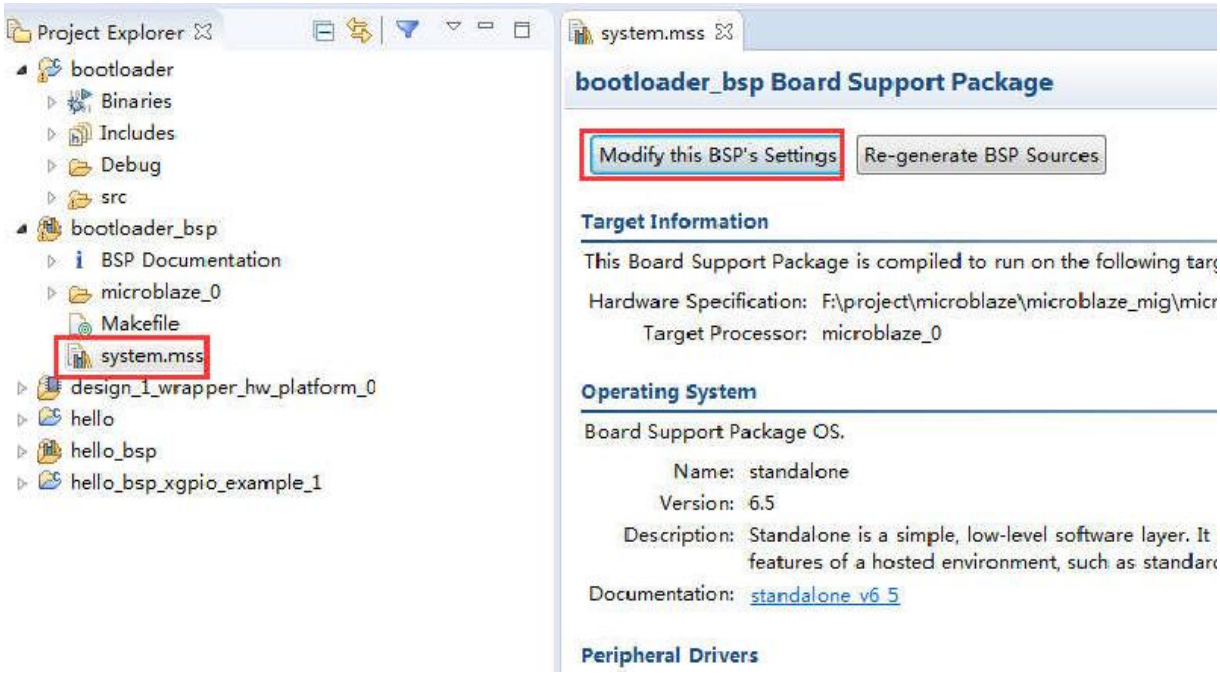


Part 3.3.2: Modify bootloader bsp

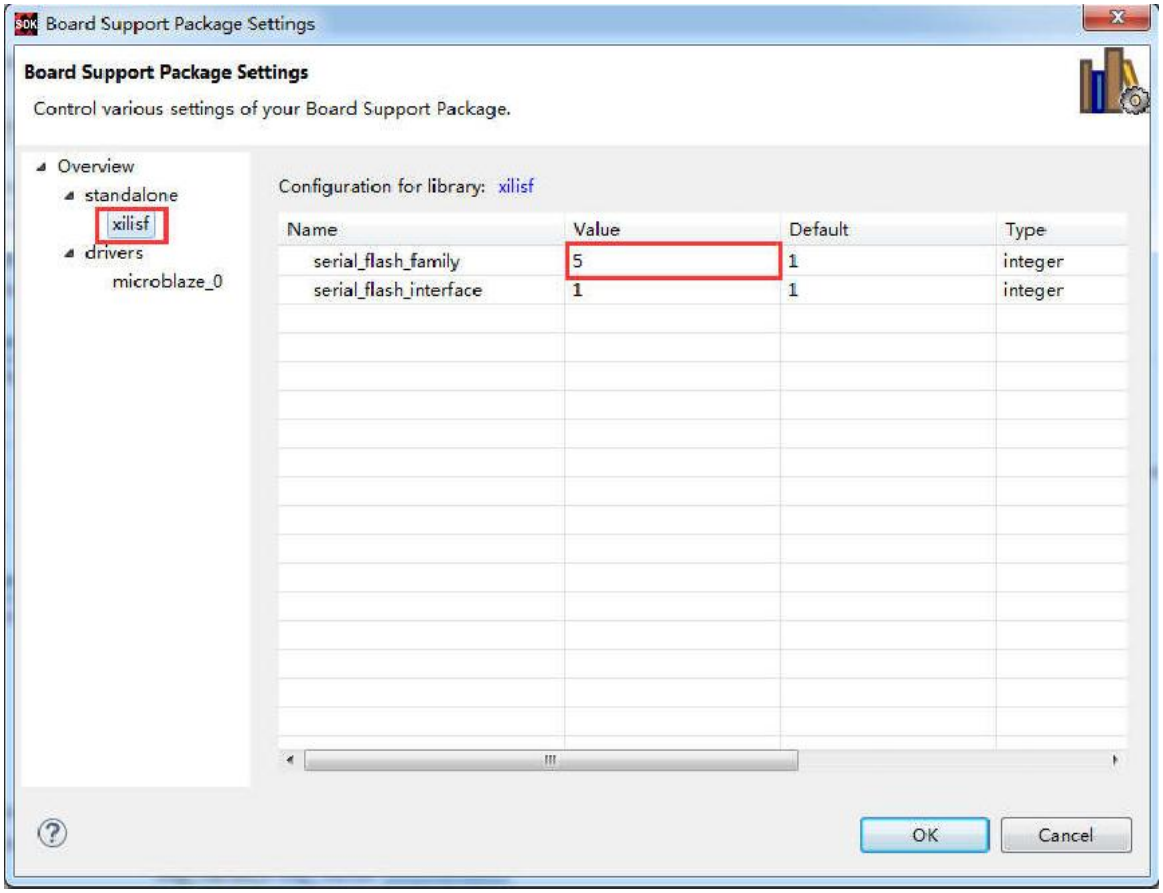
6) Double-click "system.mss" in the "bootloader_bsp" directory and click "Modify this BSP's Settings"



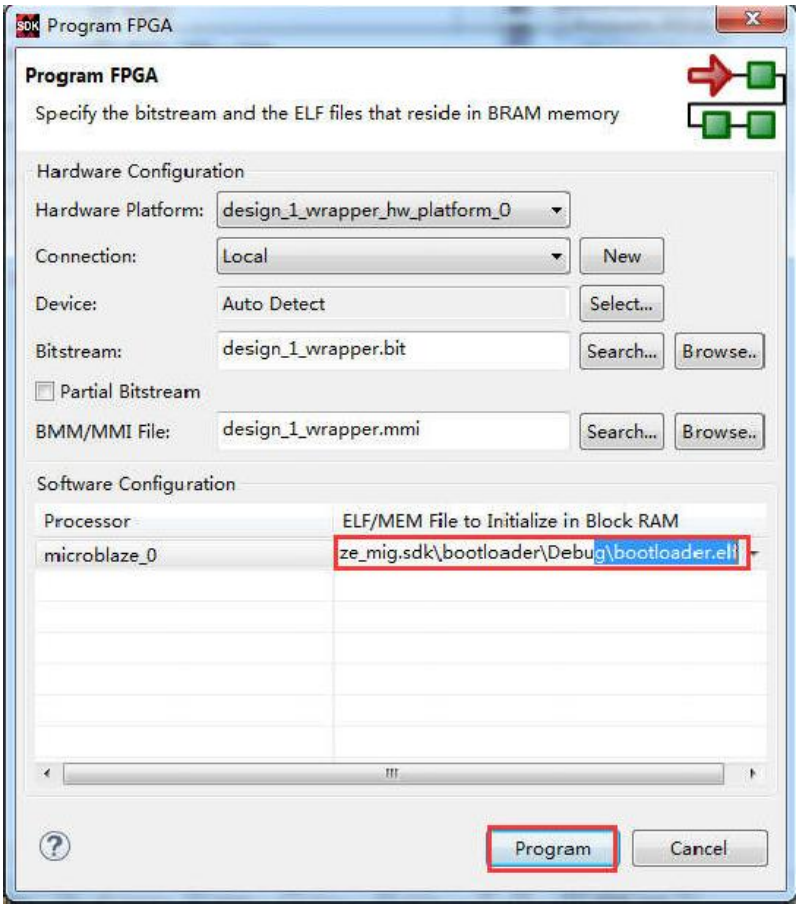
7) Double-click "system.mss" in the "bootloader_bsp" directory and click "Modify this BSP's Settings"



8) Select "xilisf" and modify "serial_flash_family" to 5

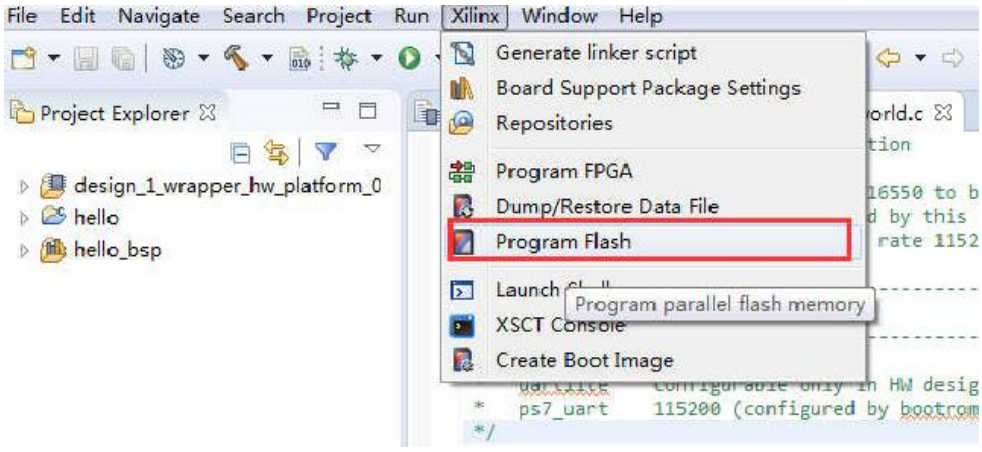


9) In the pop-up window "Software Configuration" option, "microblaze_0" select the elf file of the project "*.sdk\bootloader\Debug\bootloader.elf", merge the elf file and the bit file, click "Program"



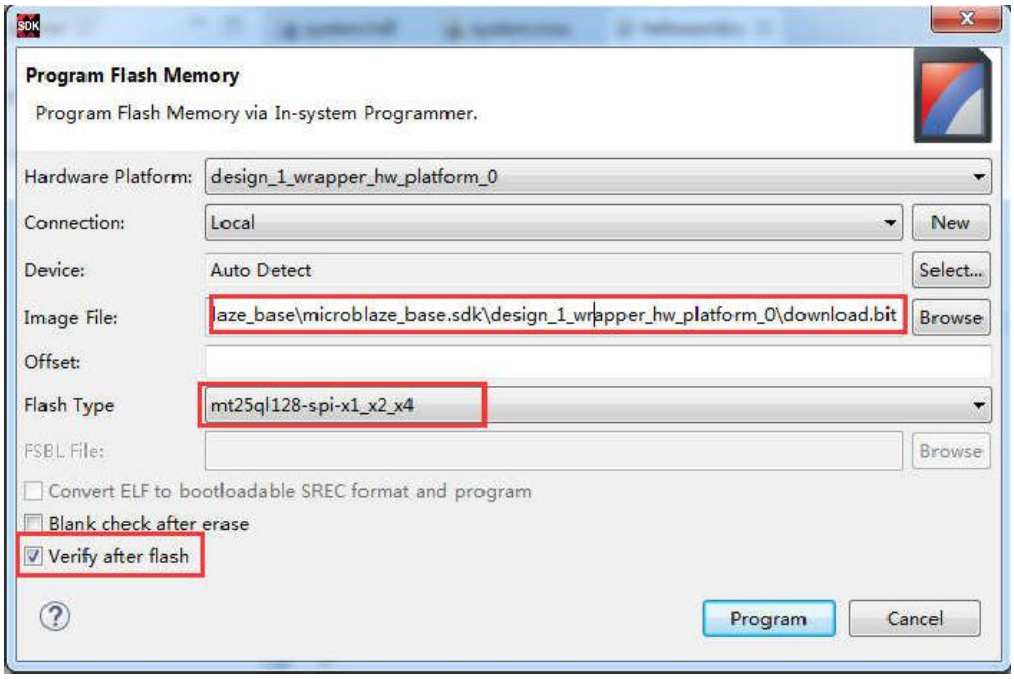
Part 3.3.3.: Programming Bitstream and BootLoader

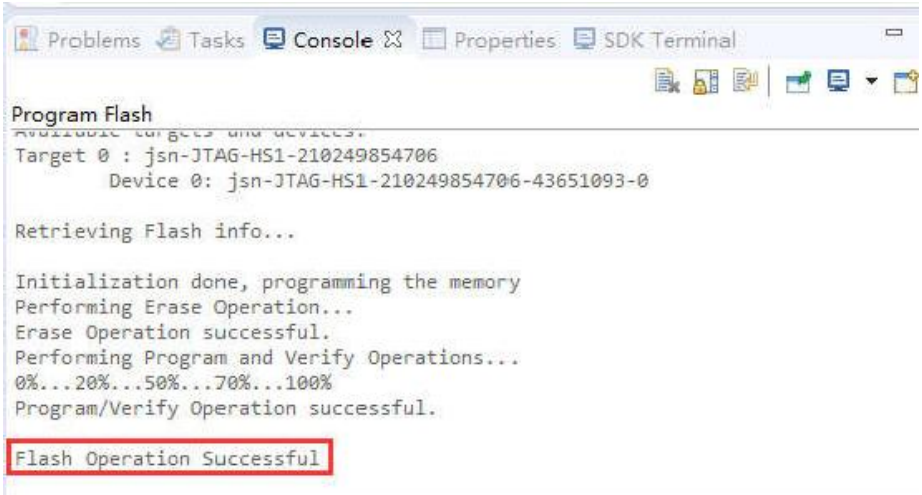
10) Click on "Xilinx -> Program Flash"



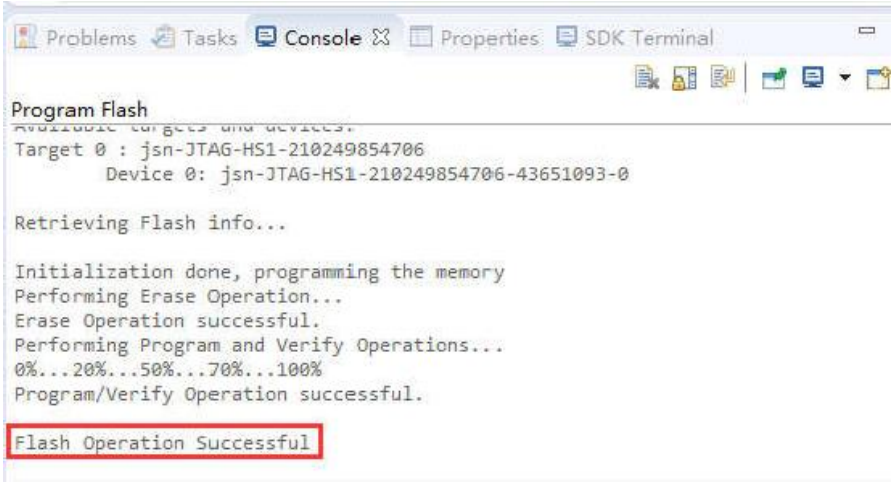
11) "Image File" selects the previous step, the generated

"download.bit" file is located in the directory.In "*.sdk/design_1_wrapper_hw_platform_0", select "mt25ql128-spi-x1_x2_x4" for "Flash Type", check "Verify after flash", click "Program" to start flashing Flash.



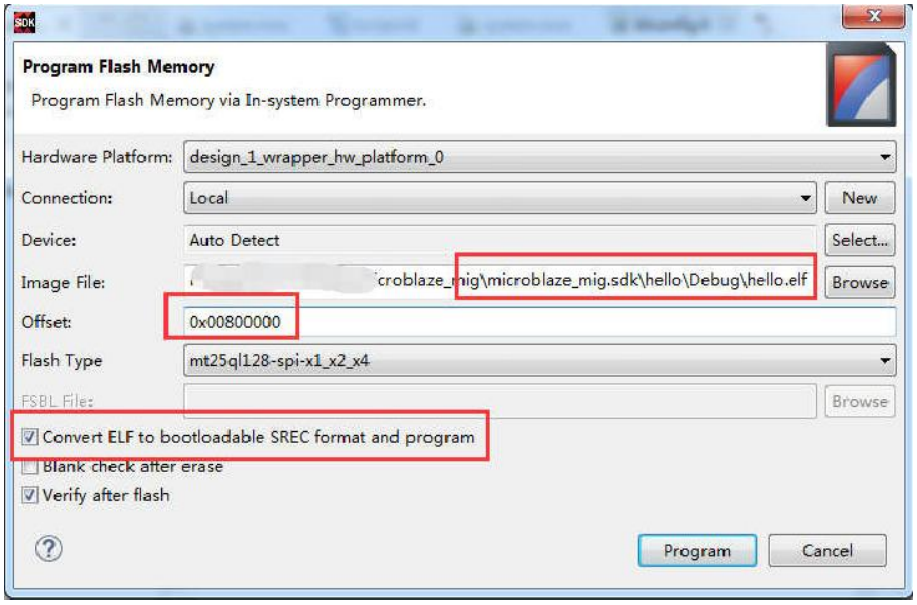


12)Prgrammed finished



Part 3.3.4: Programming application

13)Write "hello.elf" to 0x00800000. This address is set in the previous BootLoader project. Check "Convert ELF to bootloadable SREC format and program". After the programming is completed, it is best to unplug the JTAG cable and restart. Verify that the programming is successful



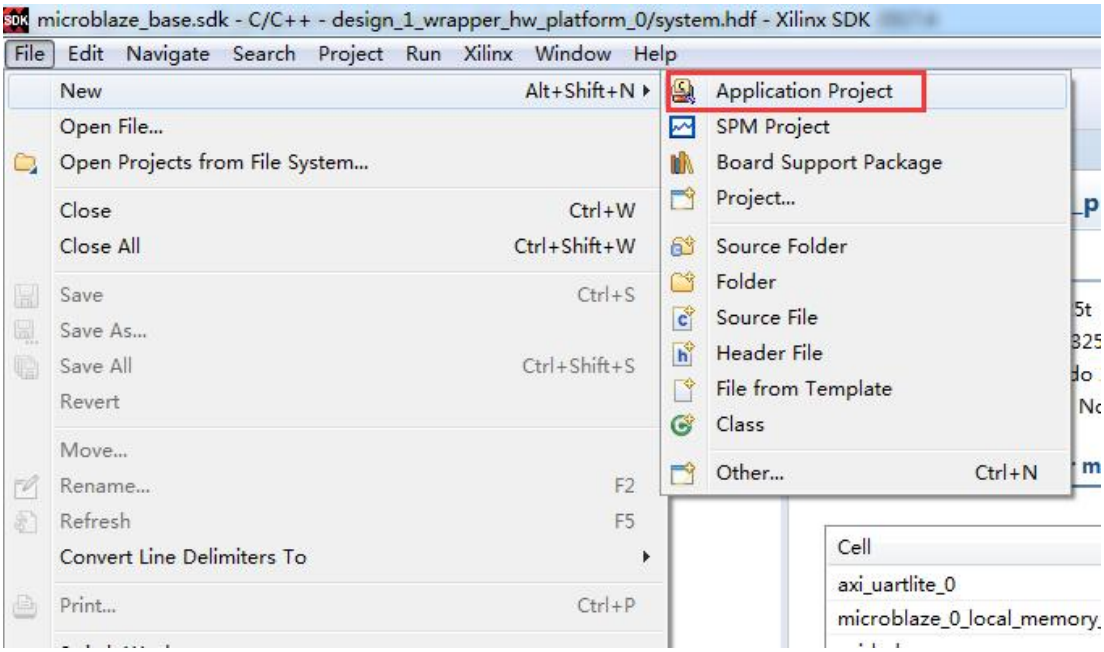
14)Running results

Part 4：Keys Controll LEDs Experiment

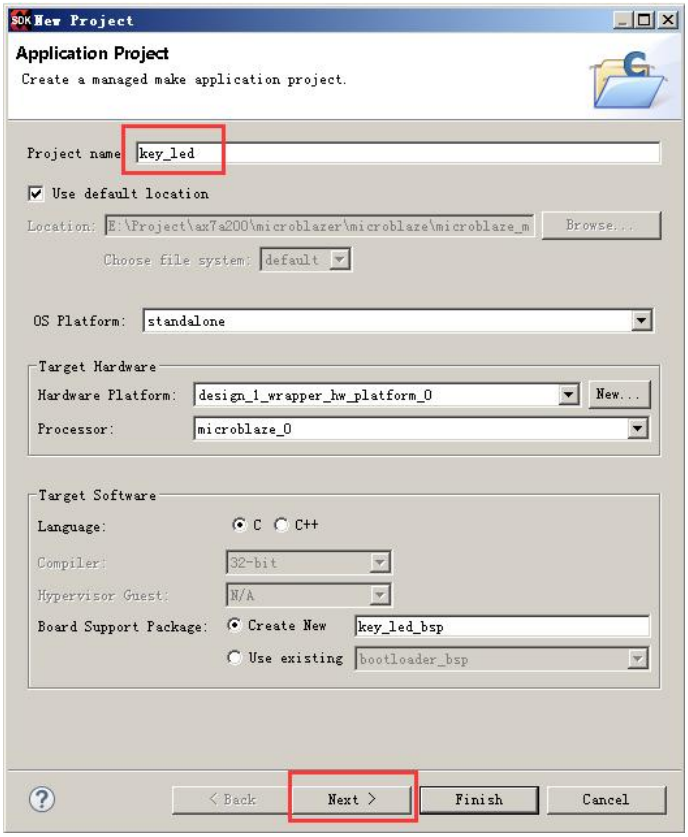
This chapter uses the Vivado project configuration in Part 3, because two GPIO controllers **axi_led** and **axi_key** have been added to the VIVADO project in Part 3. Here we only need SDK software programming to realize the key control LED light, the key is pressed, the LED light is on, the key is released, the LED light is off. In addition, the key debounce function will be added to the program to make the button function more reliable.

Part 4.1: SDK Software Programming

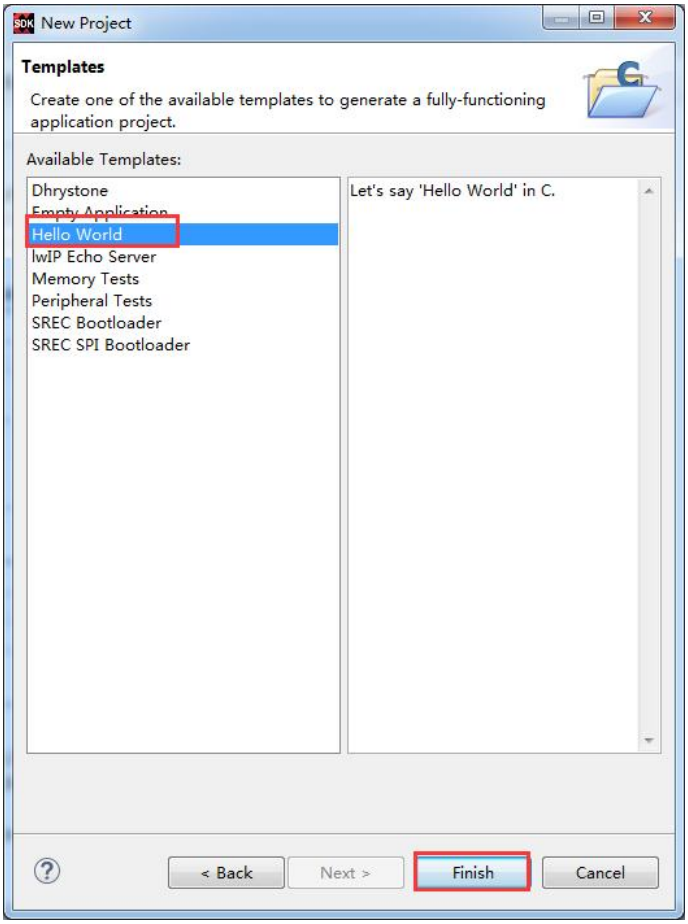
1) Created a project



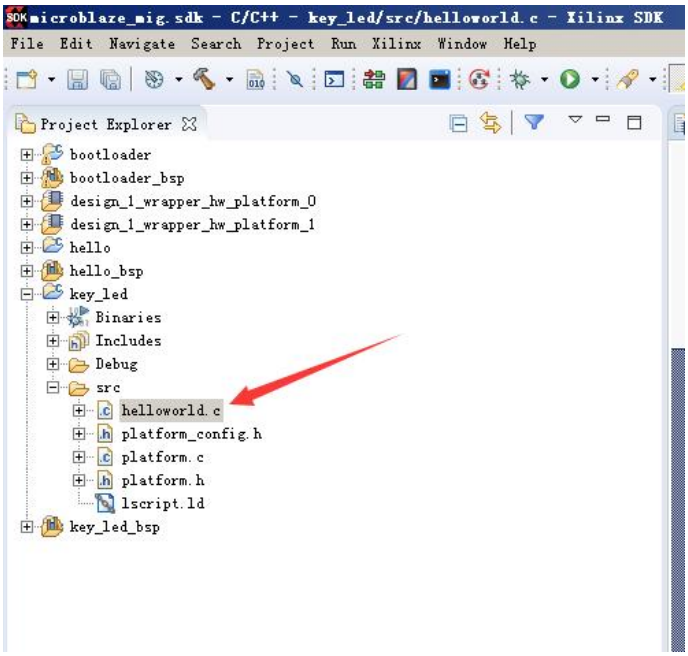
2) Set the project name as "**key_led**", and click "**Next**"



3) Module Select "**Hello World**", and click "**Finish**"



4) Modify the "helloworld.c" file after the project is established.



Modify "helloworld.c" to the following code.

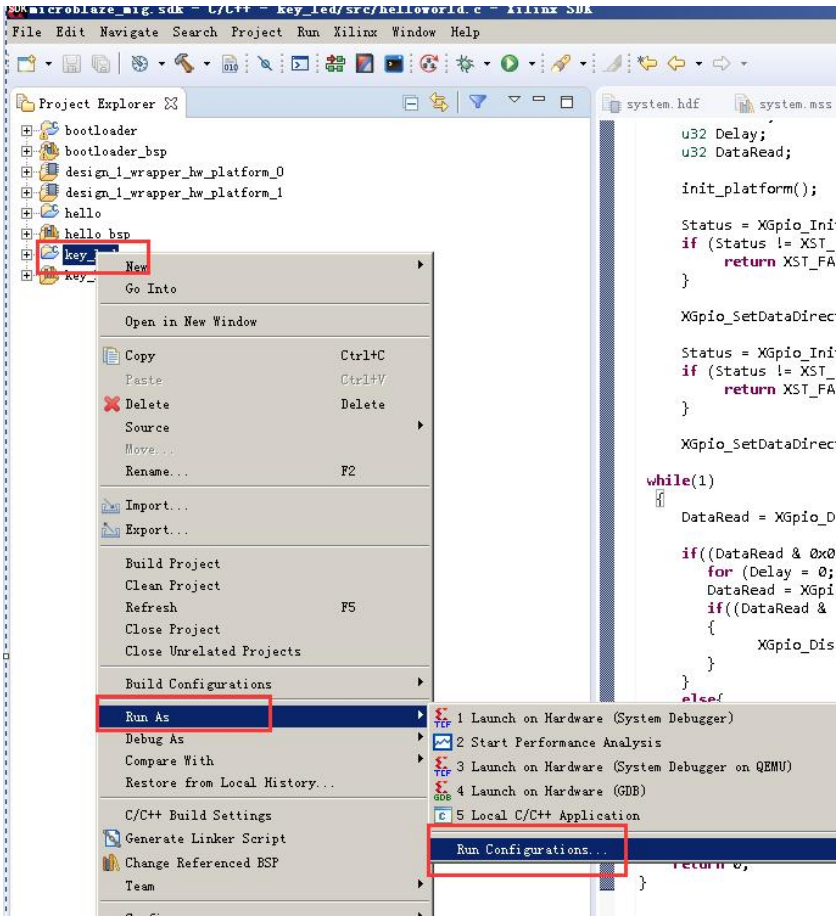
```
/***************************** Include Files *********************************/
#include <stdio.h>
#include "xparameters.h"
#include "xgpio.h"
#include "platform.h"

/************************** Variable Defintions ****************************/
XGpio Gpio_leds;
XGpio Gpio_keys;

int main()
{
     int Status;
     u32 Delay;
     u32 DataRead;

     init_platform();

     Status = XGpio_Initialize(&Gpio_keys, XPAR_AXI_KEY_DEVICE_ID);
     if (Status != XST_SUCCESS) {
          return XST_FAILURE;
     }

     XGpio_SetDataDirection(&Gpio_keys, 1, 0xFFFFFFFF);    //set Gpio_keys is
input

     Status = XGpio_Initialize(&Gpio_leds, XPAR_AXI_LED_DEVICE_ID);
     if (Status != XST_SUCCESS) {
          return XST_FAILURE;
     }

     XGpio_SetDataDirection(&Gpio_leds, 1, 0x0);         //set Gpio_leds is output

 while(1)
  {
     DataRead = XGpio_DiscreteRead(&Gpio_keys, 1);
```

```
    if((DataRead & 0x0f)!=0x0f){                    //if key is pushed
       for (Delay = 0; Delay < 2000; Delay++);
       DataRead = XGpio_DiscreteRead(&Gpio_keys, 1);  //read again
       if((DataRead & 0x0f)!=0x0f)
       {
            XGpio_DiscreteWrite(&Gpio_leds, 1, DataRead);
       }
    }
    else{                                           //if key is not pushed
          for (Delay = 0; Delay < 2000; Delay++);
          DataRead = XGpio_DiscreteRead(&Gpio_keys, 1);  //read again
          if((DataRead & 0x0f)==0x0f)
          {
               XGpio_DiscreteWrite(&Gpio_leds, 1, DataRead);
          }
    }
  }
    return 0;
}
```

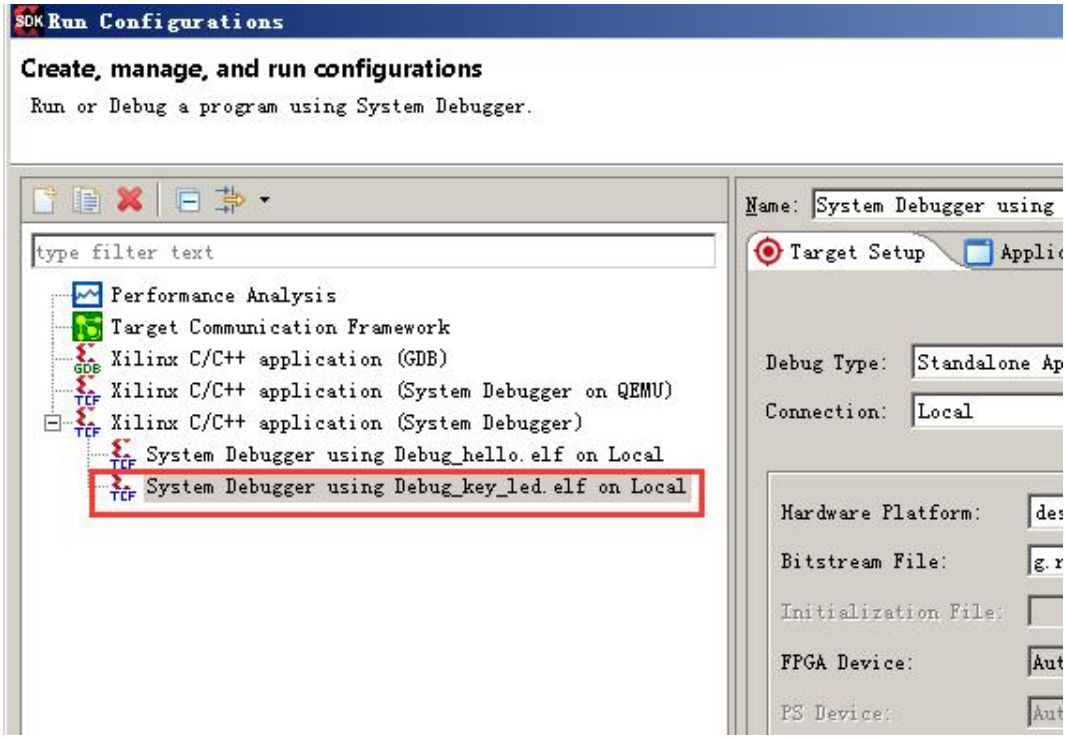
The program detects the input value of the button **Gpio_keys** in the while loop. If it finds that the key is pressed, it will wait for a while before judging the value of the key. If there is still a key pressed, write the value of **Gpio_leds**. The LED on the FPGA development board will indicate which key is pressed. The same program also judges the release of the key. When the key is released, the LED light on the FPGA development board will also go out.

Part 4.2: Download the Programs and Testing

1) Select the project "**key_led**", right click and select "**Run As -> Run Configuration...**"

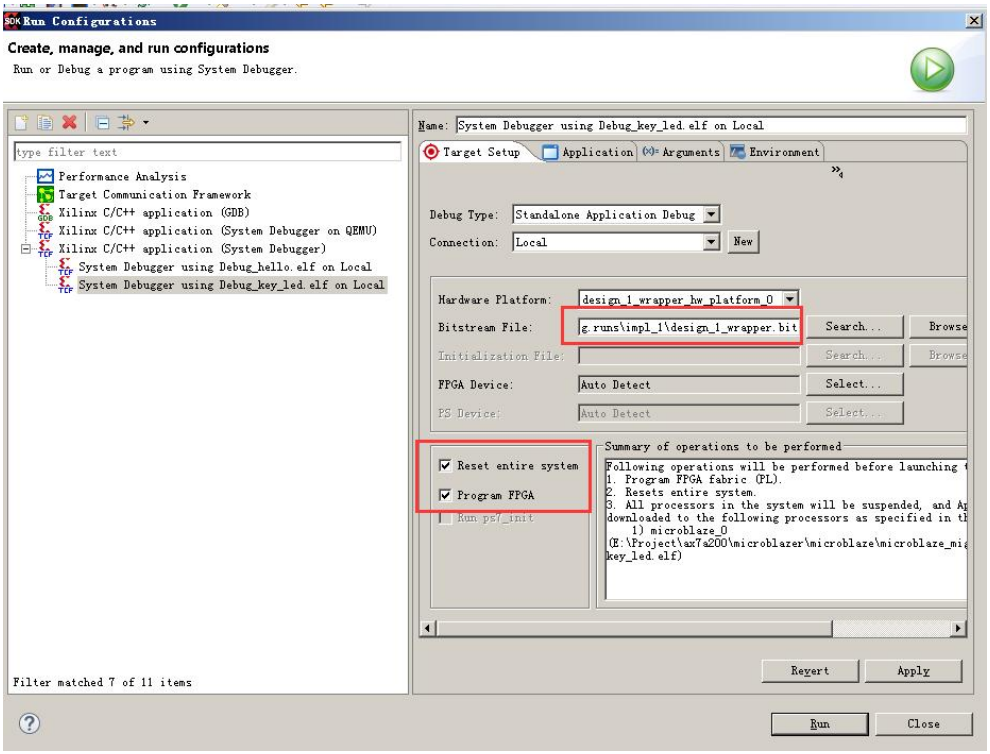


2) Double Click "**Xilinx C/C++ application(System Debugger)**", and Select "**System Debugger using Debug_key_led.elf on Local**".



3) Choose "**design_1_wrapper.bit**", and select "**Reset entire**

system" and "Program FPGA"



4) Connect the JTAG line to the FPGA development board and Run to download. After downloading, we click **KEY1** on the FPGA development board, and then the **LED1** light will light up, release the key, and the LED light will go out.

Part 5: Key interrupt Experiment

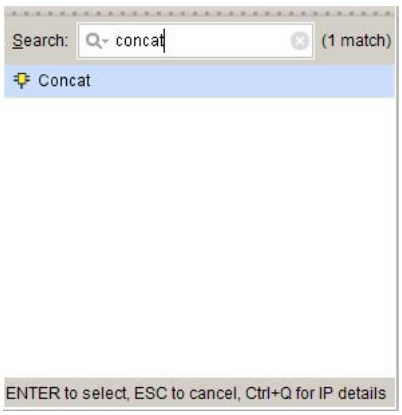
In the previous chapter Part 4, we introduced key detection, which requires the program to check the GPIO status in real time. This chapter introduces the realization of the key interrupt. The interrupt is generated through GPIO. The user program only needs to generate some corresponding operations when the interrupt is generated.

Part 5.1: Vivado Project Modification

The Vivado project in this chapter is modified on the basis of the Vivado project configuration in Part 3, and the SDK directory is deleted.

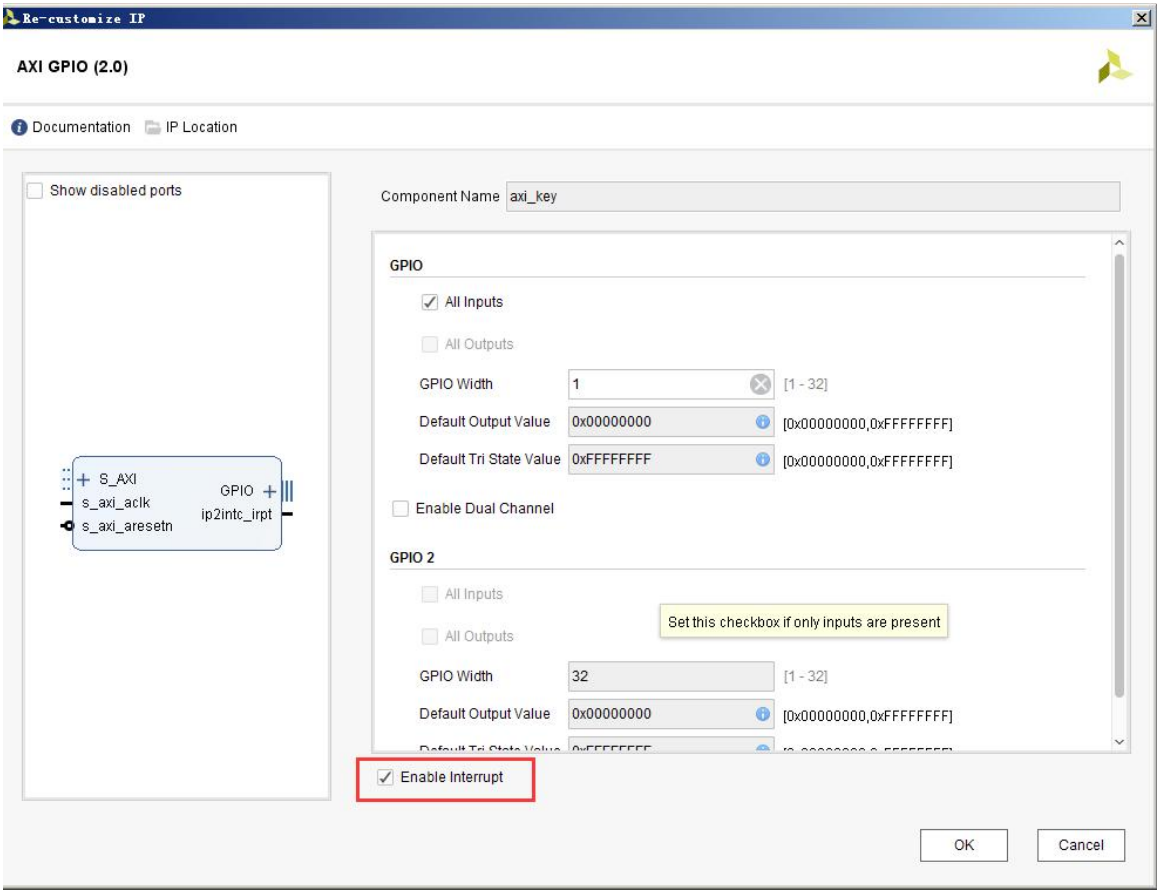
Part 5.1.1: Add Constant

1) Search “**Concat**”, and double click “**Constant**” to add a **Concat IP**



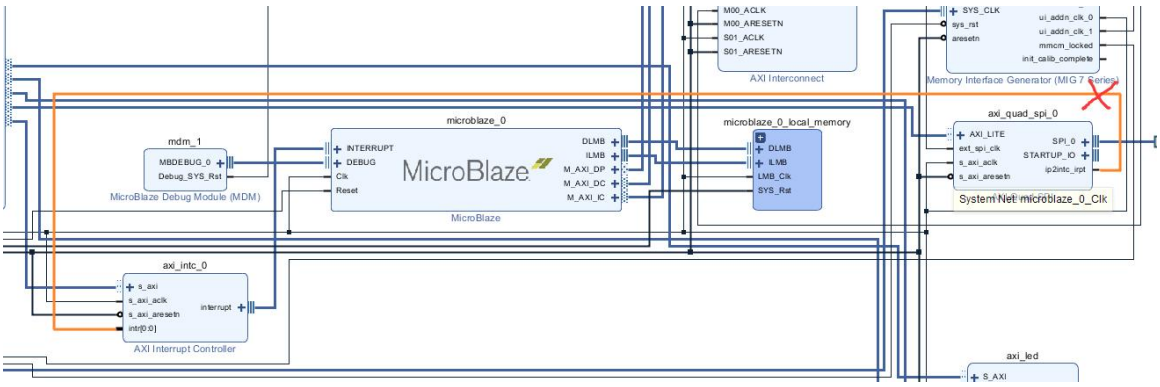
Part 5.1.2: Modify axi_key

2) Double Click "**axi_key IP**", in the configuration interface select "**Enable Interrupt**"

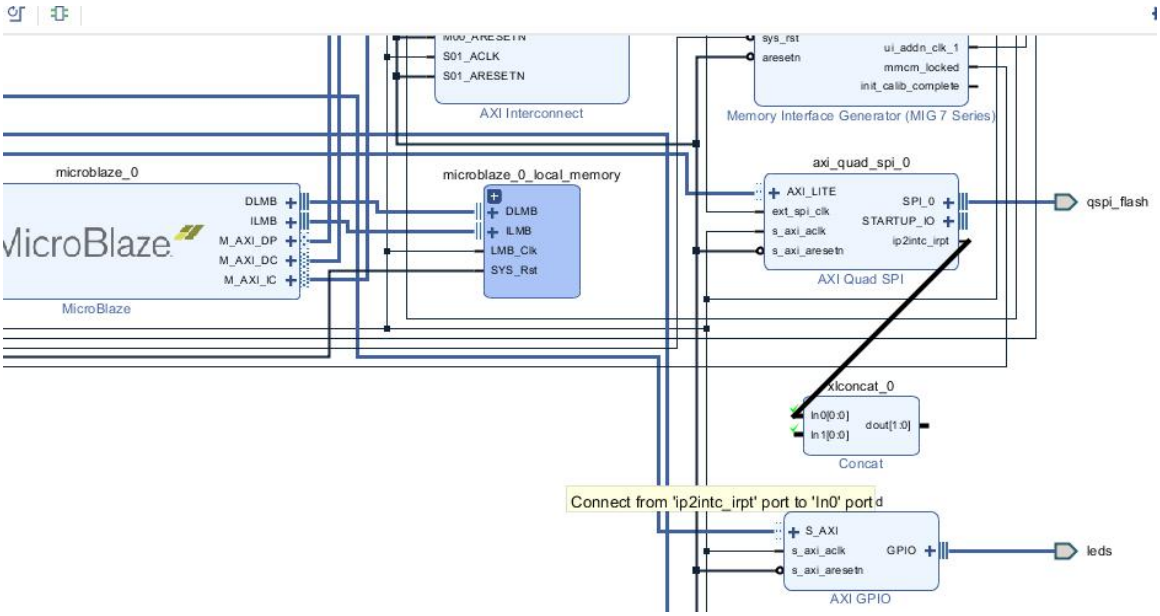


Part 5.1.3: Connect the interrupt signals

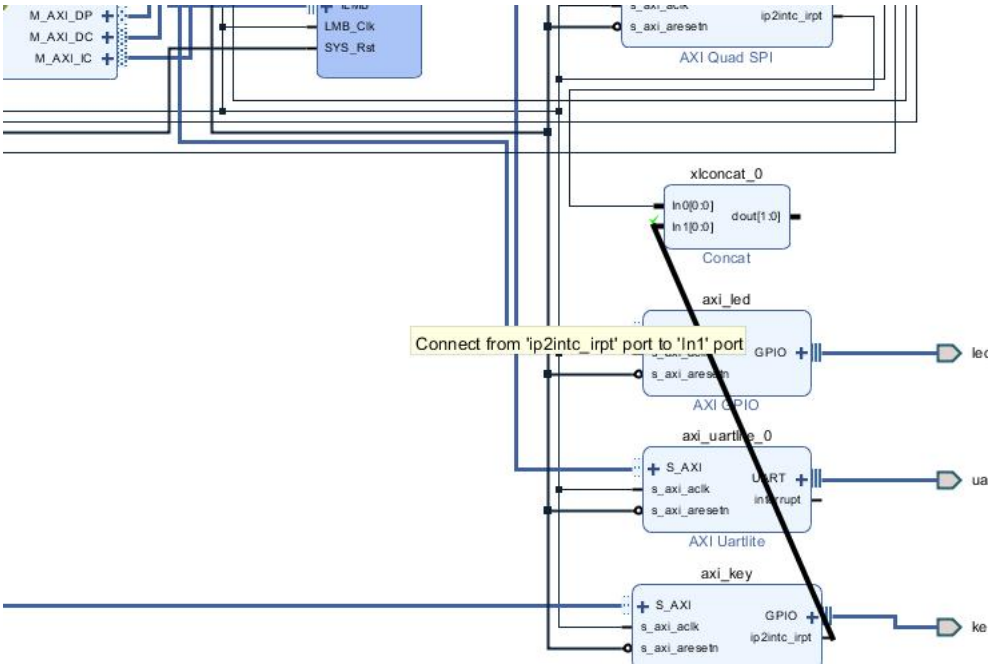
3) Delete the connection line between the interrupt output signals of **axi_quad_spi_0** and **axi_intc_0**, select the line marked in the figure below and press "delete" to delete it.



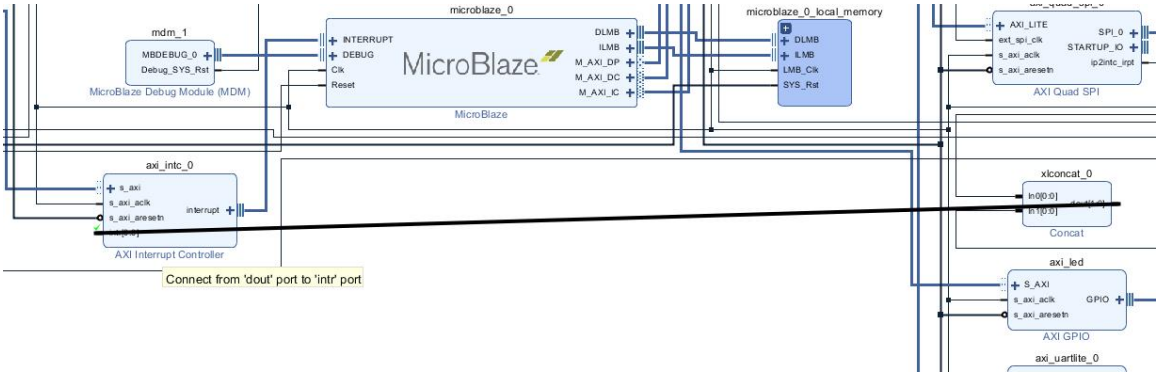
4) Connect the interrupt output signals of **axi_quad_spi_0** and **axi_intc_0**, with the input of the "**xlcontact**" just added.



5) The Connect the interrupt output signals of **axi_key** with the input of the "**xlcontact**".

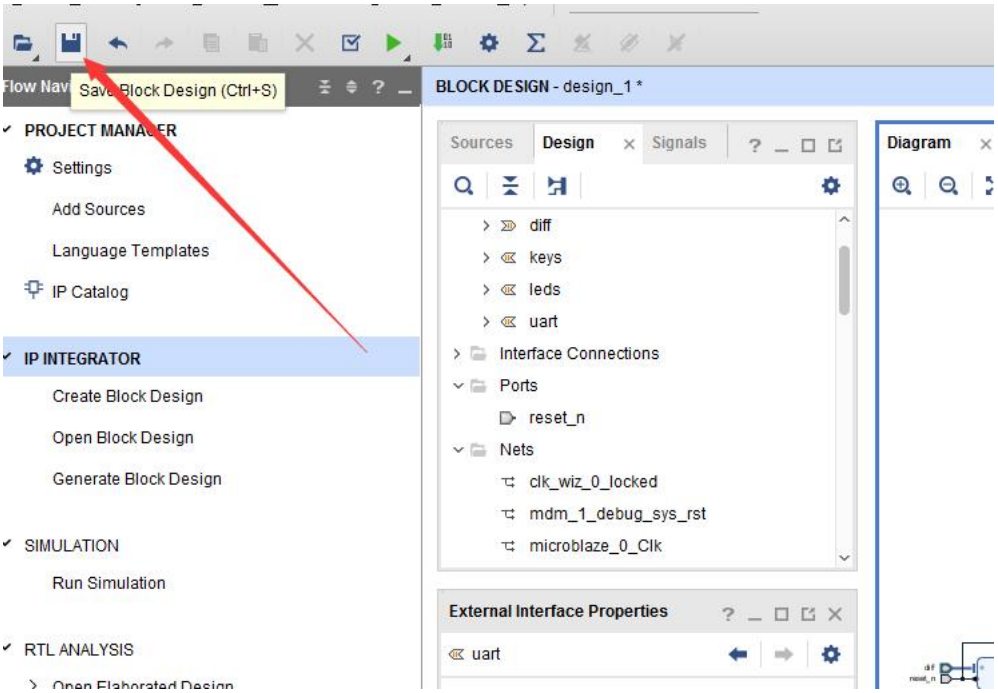


6) Connect the output signals of "xlconcat" with the interrupt input of "axi_intc_0", that equals to 2 interrupt (Flash Interrupt and Key Interrupt) all connected with the interrupt controller.



Part 5.1.4: Save the design and check errors

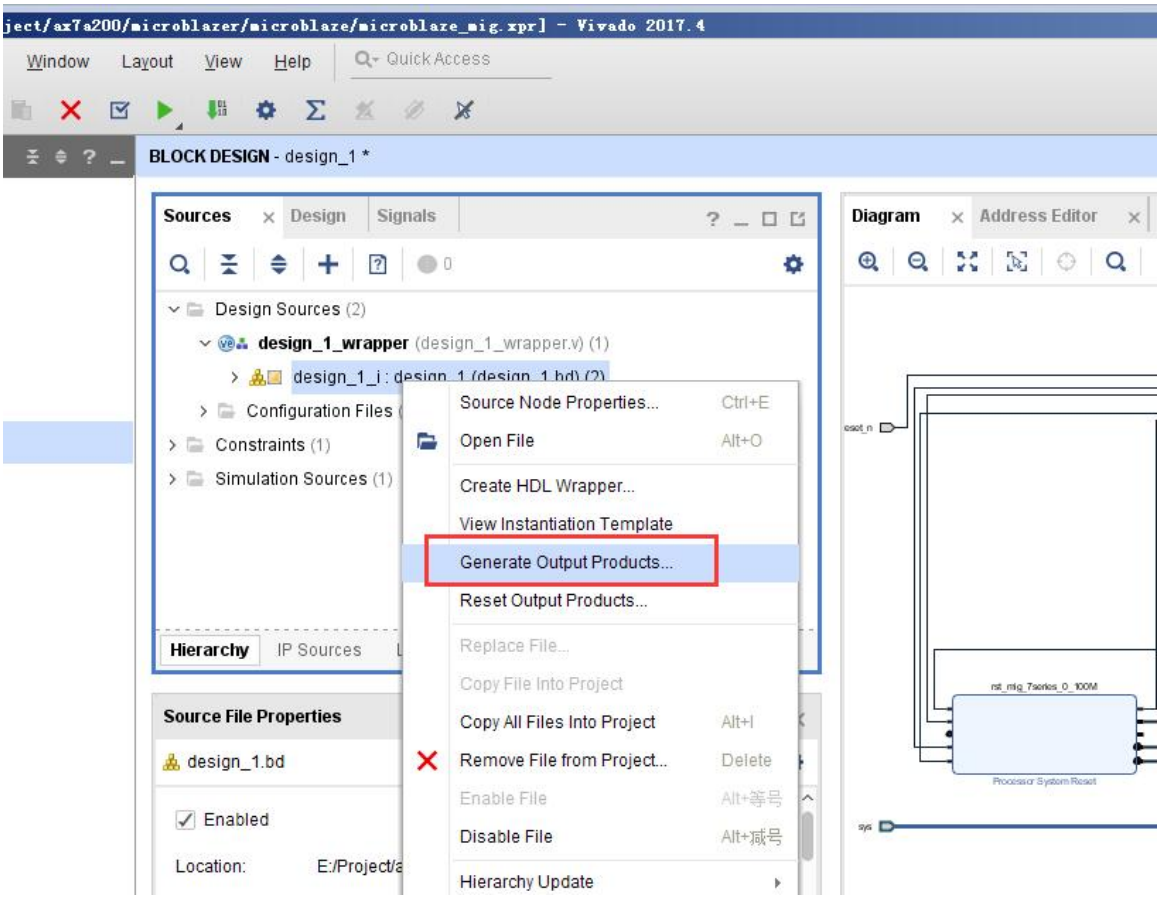
7) Save the design, and press F6 to check the design.



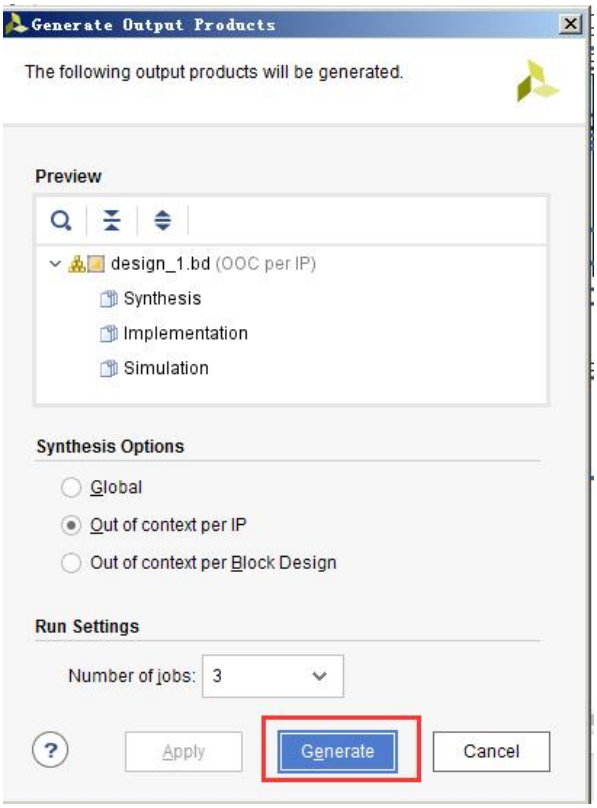
8) There is no errors or critical warning in the design.

9) Select the file “design_1.bd”, right click to choose “Generate Output Products”, update the IP parameters and connection

information to the project.



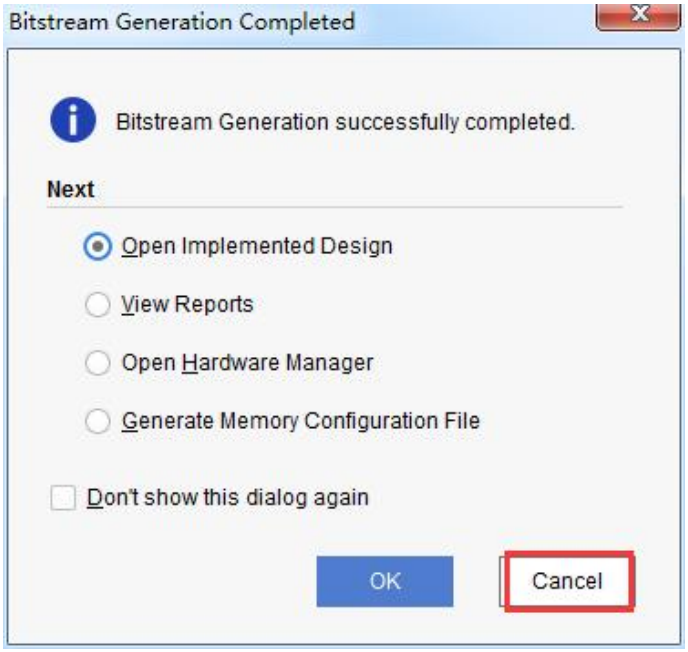
10)Select “Generate” button.



Part 5.1.5: Generate Bitstream file

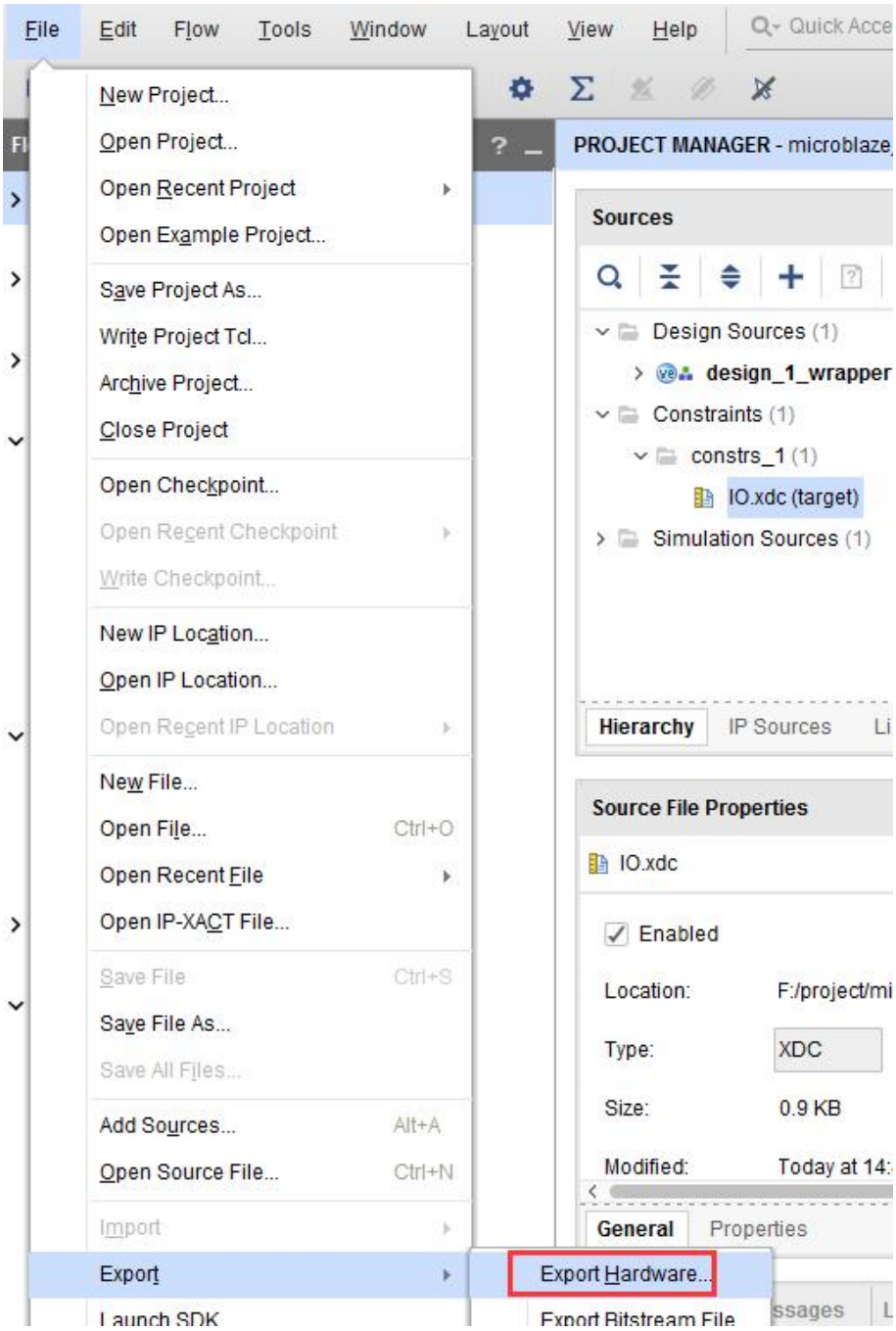
11)Click “Generate Bitstream” to generate bit stream file.

12)After generation, click “Cancel”, no need to do any action.

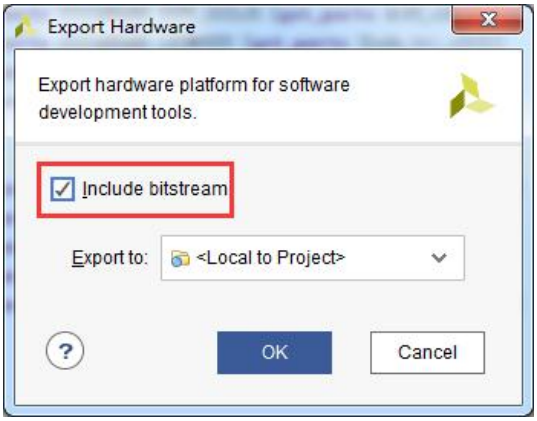


Part 5.1.6: Export Hardware

13)Export Hardware by “**File -> Export -> Export Hardware...**”

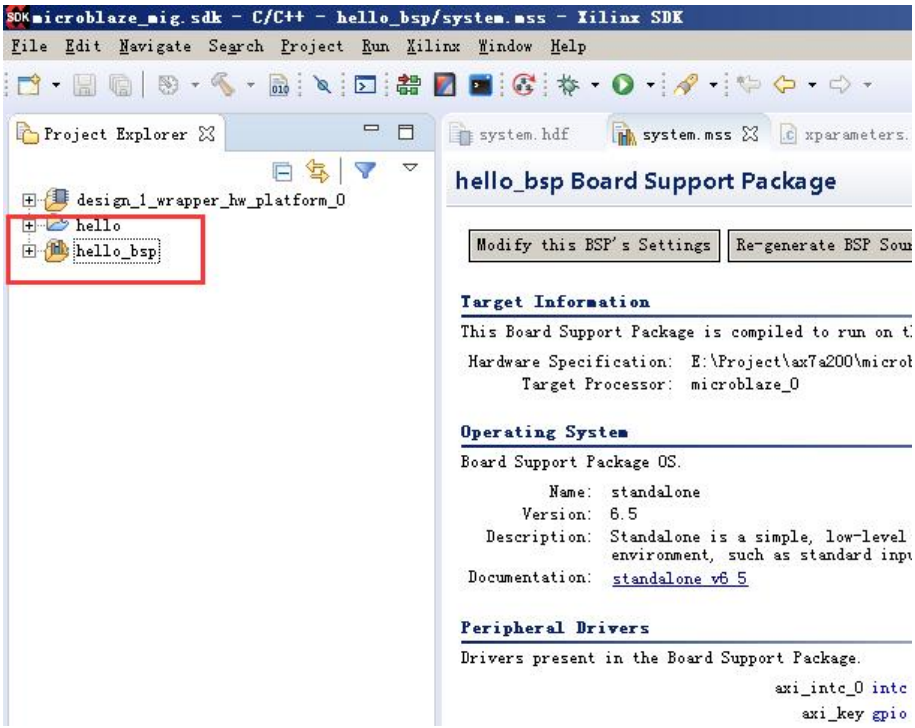


14)In the pop-up window, select "**Include bitstream**"

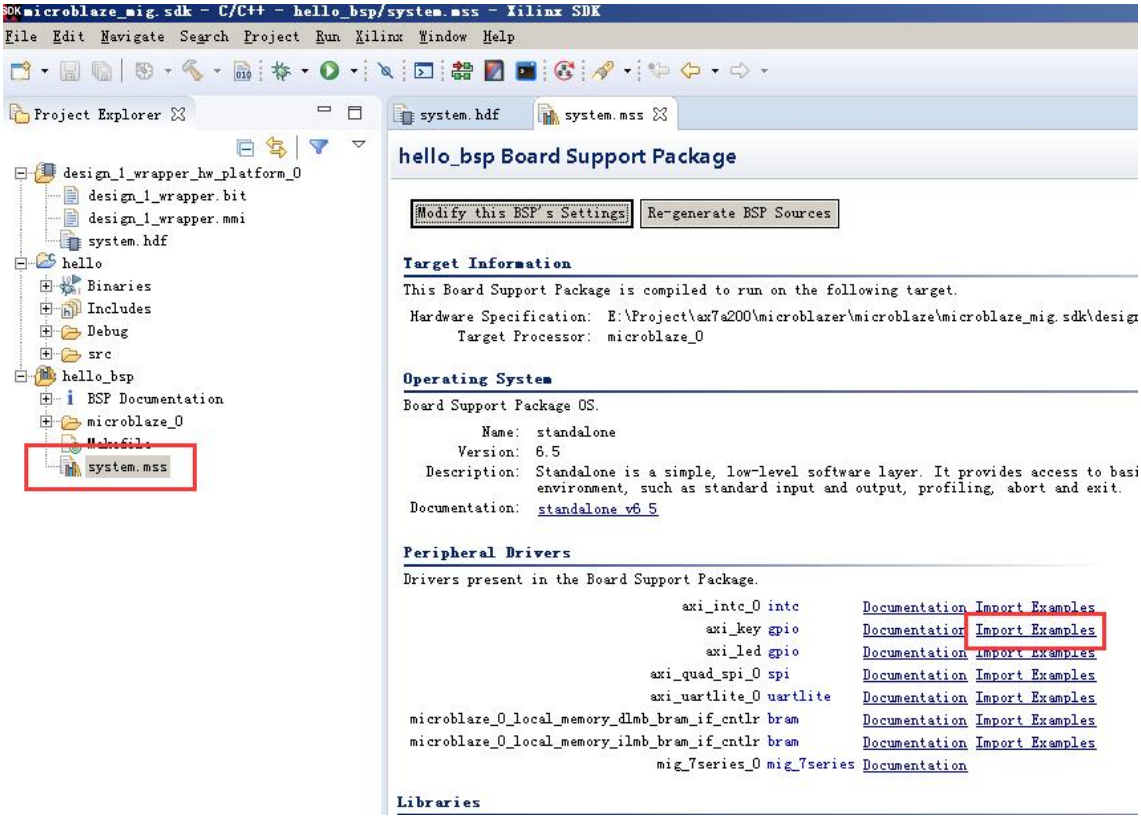


Part 5.2: Interrupt programs in the SDK Development

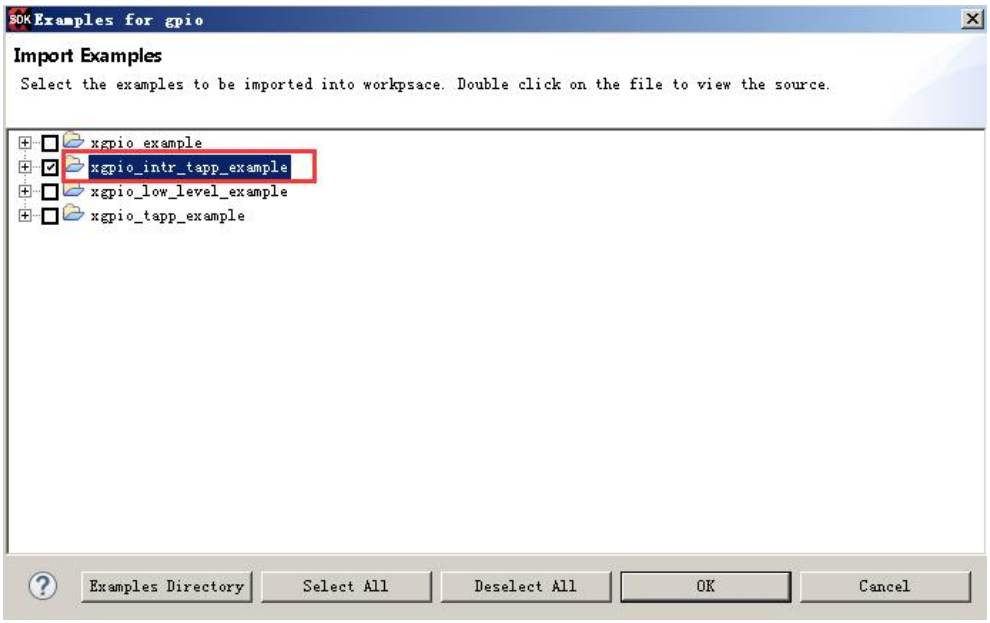
15)Create a new project "**helloworld**"



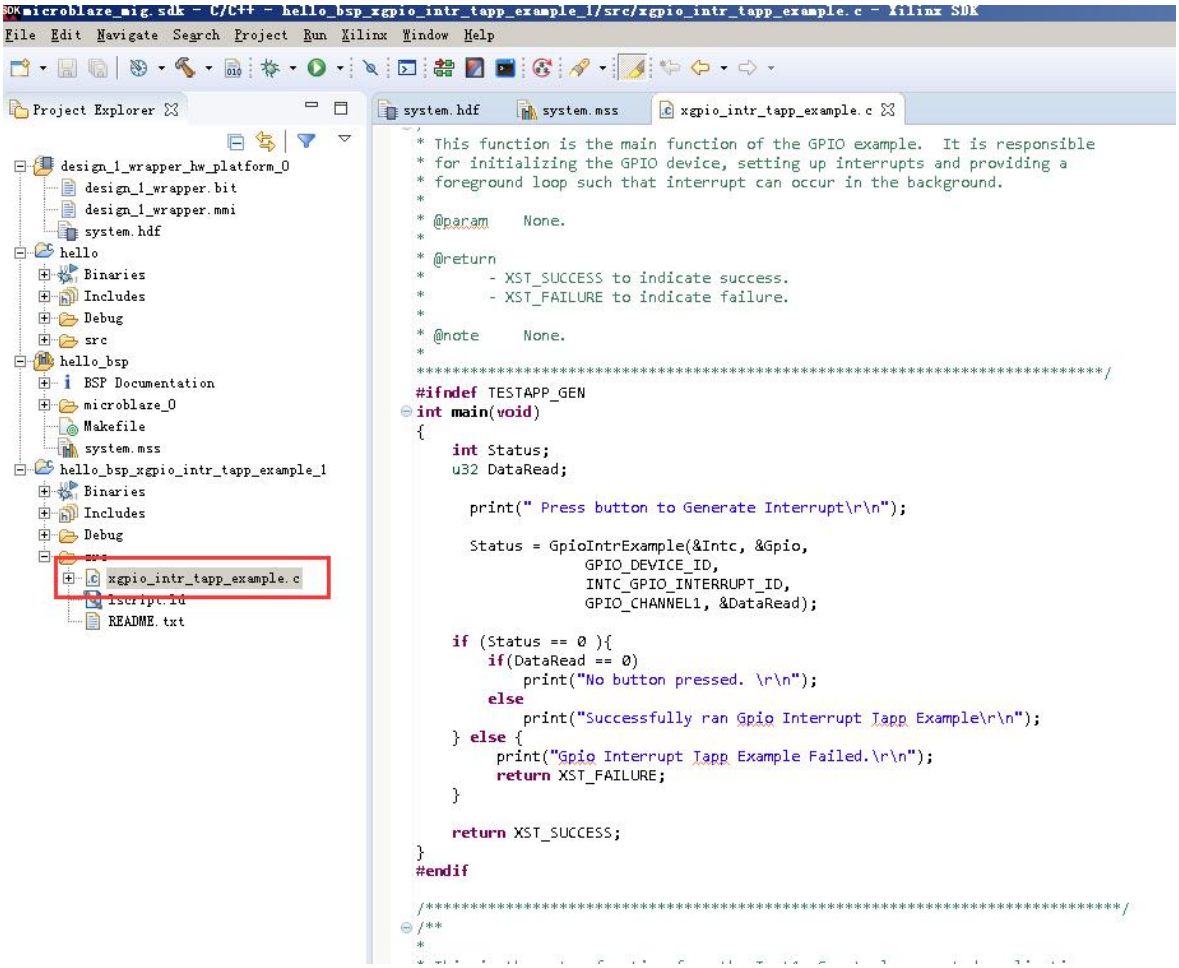
16)Double-click the "system.mss" file in the BSP directory, and click "Import Examples" next to axi_key



17)Select “xgpio_intr_tapp_example”



18)Open the generated “xpio_intr_tapp_example.c”, first look at the program inside



In the program, the **GpioSetupIntrSystem** program is used to realize the interrupt initialization of **GPIO**. The main interrupt functions used in it are:

XIntc_Initialize interrupt initialization

XIntc_Connect Set the entry address of the interrupt service program

XIntc_Enable interrupt enable

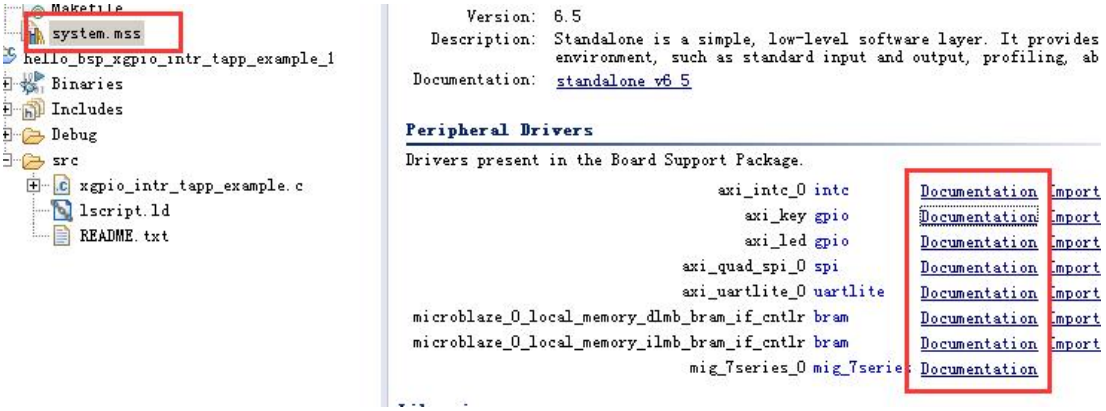
XIntc_Start interrupt start

XGpio_InterruptEnable corresponding GPIO interrupt enable

XGpio_InterruptGlobalEnable GPIO global interrupt enable

Xil_ExceptionInit abnormal Processing function

For the definition and description of these functions, you can refer to the document, and the register definition can be viewed in the IP document in the vivado software.



19)In order to make the program more concise, we modified the main function here and merged a gpioIntrExample in the main function

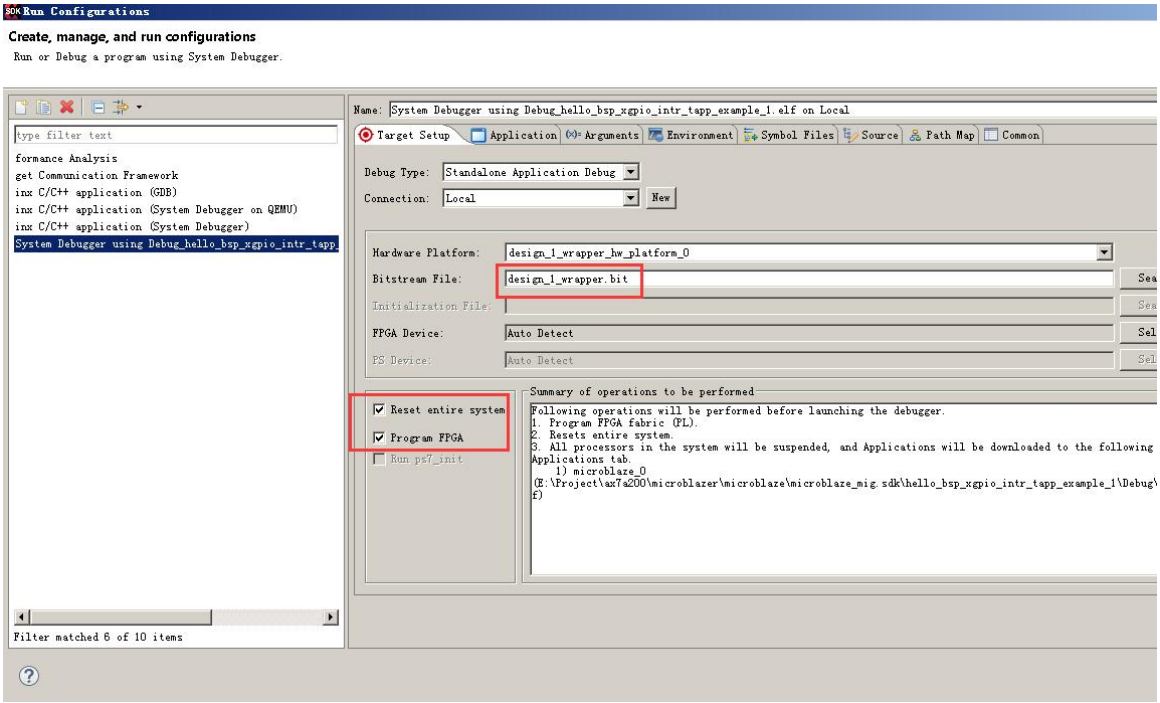
```
/************************** Include Files *******************************/

#include "xparameters.h"
#include "xgpio.h"
#include "xil_exception.h"
#include "xintc.h"
#include <stdio.h>


#define GPIO_DEVICE_ID          XPAR_AXI_KEY_DEVICE_ID
#define GPIO_CHANNEL1           1

 #define INTC_GPIO_INTERRUPT_ID     XPAR_INTC_0_GPIO_0_VEC_ID
 #define INTC_DEVICE_ID   XPAR_INTC_0_DEVICE_ID

 #define INTC_DEVICE_ID   XPAR_INTC_0_DEVICE_ID
 #define INTC        XIntc
 #define INTC_HANDLER     XIntc_InterruptHandler


/************************** Function Prototypes *****************************/
void GpioHandler(void *CallBackRef);

int GpioIntrExample(INTC *IntcInstancePtr, XGpio *InstancePtr,
                u16 DeviceId, u16 IntrId,
                u16 IntrMask, u32 *DataRead);

int GpioSetupIntrSystem(INTC *IntcInstancePtr, XGpio *InstancePtr,
                u16 DeviceId, u16 IntrId, u16 IntrMask);

void GpioDisableIntr(INTC *IntcInstancePtr, XGpio *InstancePtr,
                u16 IntrId, u16 IntrMask);

/************************** Variable Definitions ****************************/

XGpio Gpio; /* The Instance of the GPIO Driver */

INTC Intc; /* The Instance of the Interrupt Controller Driver */

static u16 GlobalIntrMask; /* GPIO channel mask that is needed by
                    * the Interrupt Handler */

static volatile u32 IntrFlag; /* Interrupt Handler Flag */


int main(void)
{
     int Status;

     print(" Press button to Generate Interrupt\r\n");

     Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
     if (Status != XST_SUCCESS) {
          return XST_FAILURE;
     }

     Status = GpioSetupIntrSystem(&Intc, &Gpio, GPIO_DEVICE_ID,INTC_GPIO_INTERRUPT_ID, GPIO_CHANNEL1);
     if (Status != XST_SUCCESS) {
          return XST_FAILURE;
```

```
	}

	IntrFlag = 0;

	while (1) {
		if (IntrFlag) {
			IntrFlag =0;
			print("KEY1 is pressed, interrupt is happen \r\n");
		}
	}


	GpioDisableIntr(&Intc, &Gpio, INTC_GPIO_INTERRUPT_ID, GPIO_CHANNEL1);


return XST_SUCCESS;
}


/*****************************************************************************/
/**   GPIO interrupt setup
*****************************************************************************/
int GpioSetupIntrSystem(INTC *IntcInstancePtr, XGpio *InstancePtr,
			u16 DeviceId, u16 IntrId, u16 IntrMask)
{
	int Result;

	GlobalIntrMask = IntrMask;


	/*
	 * Initialize the interrupt controller driver so that it's ready to use.
	 * specify the device ID that was generated in xparameters.h
	 */
	Result = XIntc_Initialize(IntcInstancePtr, INTC_DEVICE_ID);
	if (Result != XST_SUCCESS) {
		return Result;
	}

	/* Hook up interrupt service routine */
	XIntc_Connect(IntcInstancePtr, IntrId,
			(Xil_ExceptionHandler)GpioHandler, InstancePtr);

	/* Enable the interrupt vector at the interrupt controller */
	XIntc_Enable(IntcInstancePtr, IntrId);

	/*
	 * Start the interrupt controller such that interrupts are recognized
	 * and handled by the processor
	 */
	Result = XIntc_Start(IntcInstancePtr, XIN_REAL_MODE);
	if (Result != XST_SUCCESS) {
		return Result;
	}

	/*
	 * Enable the GPIO channel interrupts so that push button can be
	 * detected and enable interrupts for the GPIO device
	 */
	XGpio_InterruptEnable(InstancePtr, IntrMask);
	XGpio_InterruptGlobalEnable(InstancePtr);

	/*
	 * Initialize the exception table and register the interrupt
	 * controller handler with the exception table
	 */
	Xil_ExceptionInit();

	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
			(Xil_ExceptionHandler)INTC_HANDLER, IntcInstancePtr);

	/* Enable non-critical exceptions */
	Xil_ExceptionEnable();

	return XST_SUCCESS;
}

/*****************************************************************************/
/* This is the interrupt handler routine for the GPIO for this example.
```

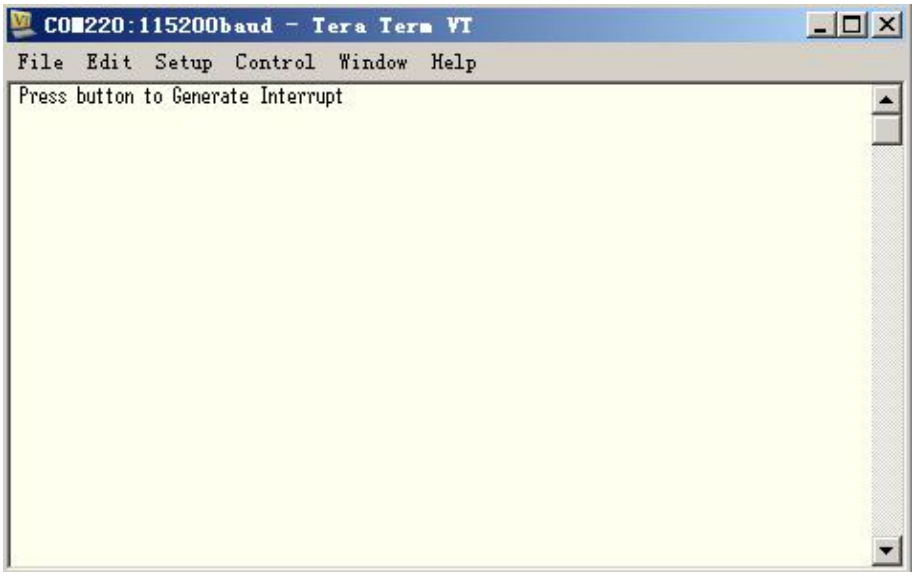
```
*****************************************************************************/
void GpioHandler(void *CallbackRef)
{
	XGpio *GpioPtr = (XGpio *)CallbackRef;

	IntrFlag = 1;

	/* Clear the Interrupt */
	XGpio_InterruptClear(GpioPtr, GlobalIntrMask);

}

/*****************************************************************************/
/**This function disables the interrupts for the GPIO
*****************************************************************************/
void GpioDisableIntr(INTC *IntcInstancePtr, XGpio *InstancePtr,
		u16 IntrId, u16 IntrMask)
{
	XGpio_InterruptDisable(InstancePtr, IntrMask);
#ifdef XPAR_INTC_0_DEVICE_ID
	XIntc_Disable(IntcInstancePtr, IntrId);
#else
	/* Disconnect the interrupt */
	XScuGic_Disable(IntcInstancePtr, IntrId);
	XScuGic_Disconnect(IntcInstancePtr, IntrId);
#endif
	return;
}
```

Part 5.3: Program download test

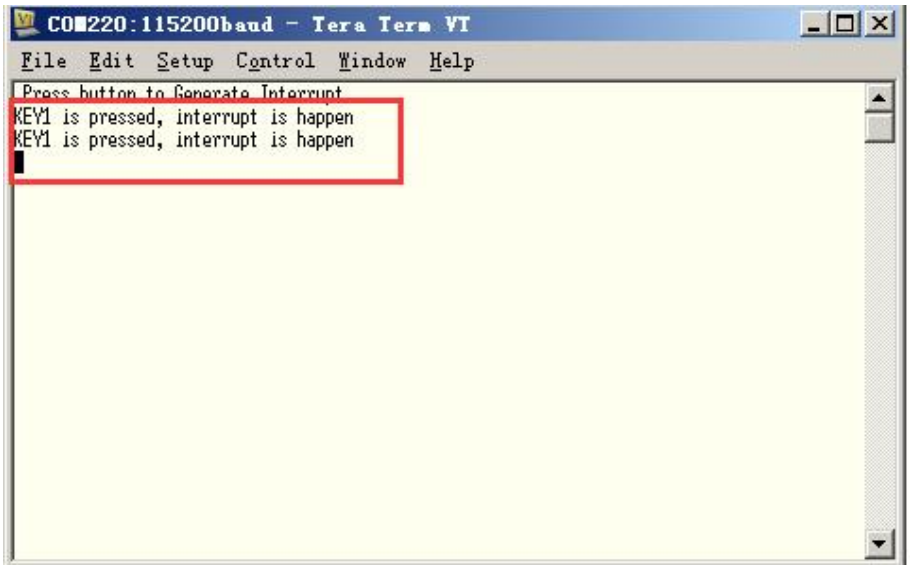
20) Select the "hello_bsp_xgpio_intr_tapp_example" project, right-click and select "Run As -> Run Configuration...", and configure the following in the pop-up interface.



21) Connect the JTAG cable to the FPGA development board and Run to download. After downloading, the serial port displays print information.



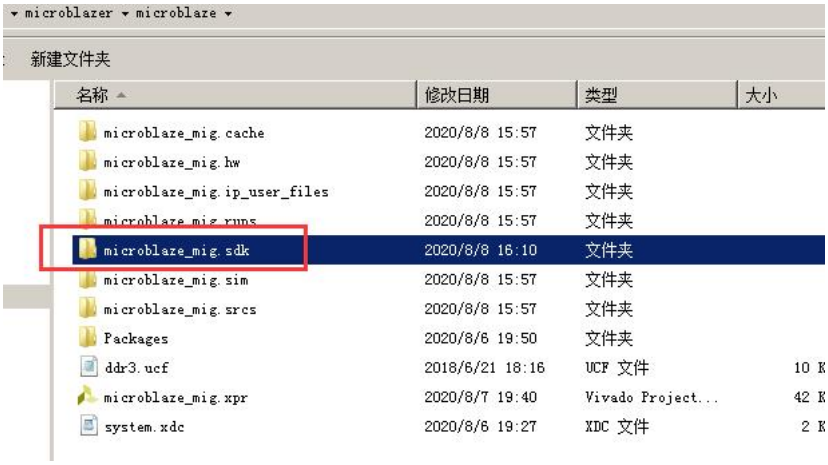
22)Press KEY1, the serial port will print 2 messages, indicating that the interrupt has been triggered twice. Because the GPIO input changes, it will generate an interrupt. So when the key is pressed, it will trigger an interrupt, and when the key is released, it will trigger another interrupt.



23)if the user wants to determine whether the interrupt was caused by a key press or a release? Then you need to read the GPIO value when the interrupt is generated. Regarding how to read the value of the key,refer to Part 4, which is not repeated here.

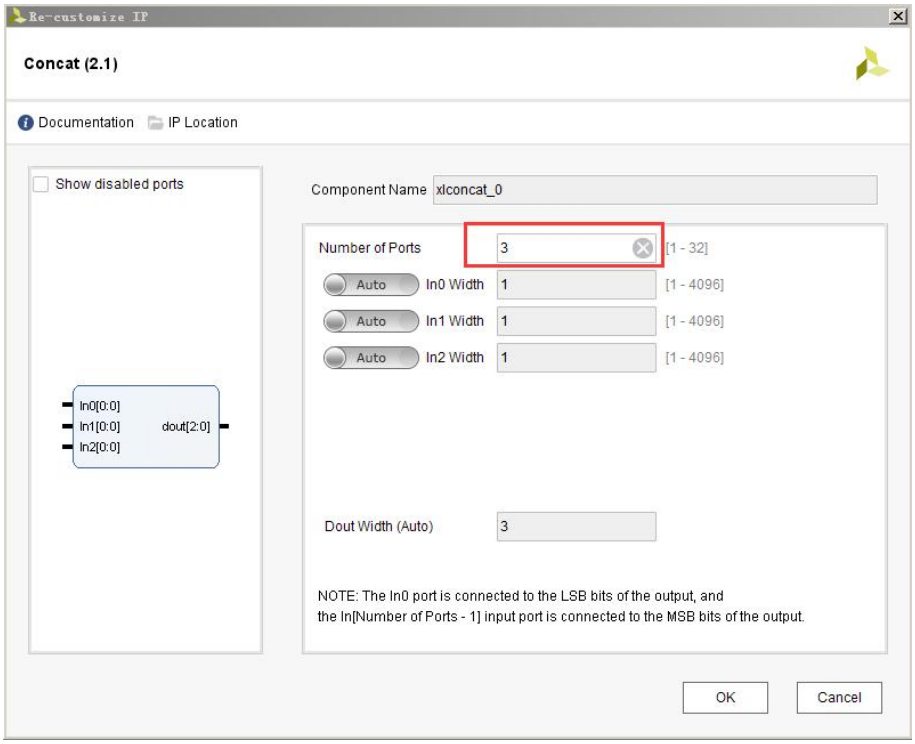
Part 6: Serial Communication Experiment

This chapter introduces how to implement serial communication in Microblaze using query mode or interrupt mode. The Vivado project here is modified on the basis of the Vivado project configuration in Part 5, and the SDK directory is deleted.



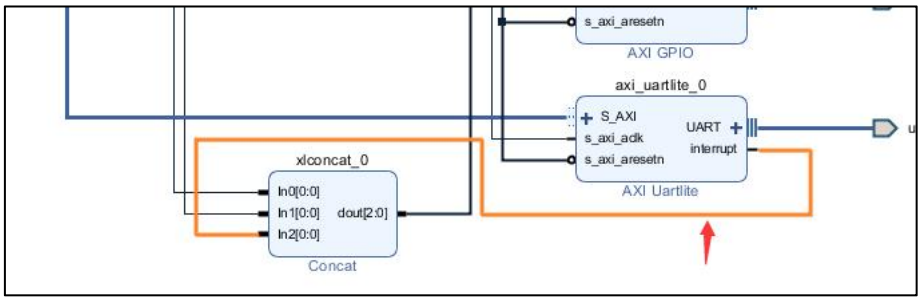
Part 6.1: Vivado Project ModificationPart 6.1.1: Modify Constant

1) Double-click the "**Constant**" IP, the number of input channels is changed to 3



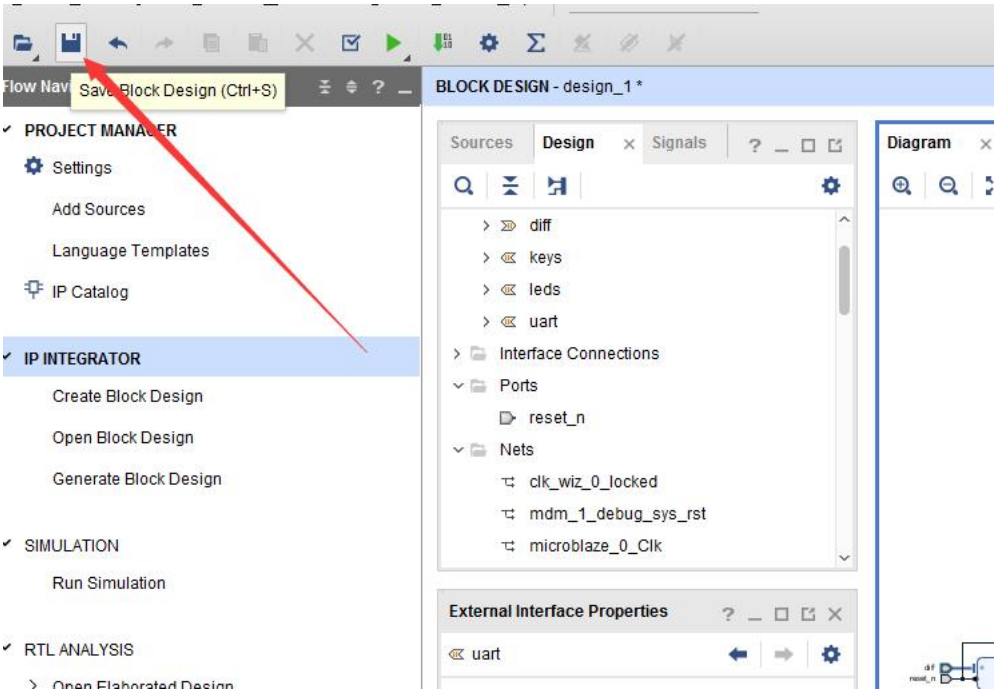
Part 6.1.2: Connect the Interrupt signals

2) The interrupt output of **axi_uartlite_0** is connected to the input pin of **xlconcat**.

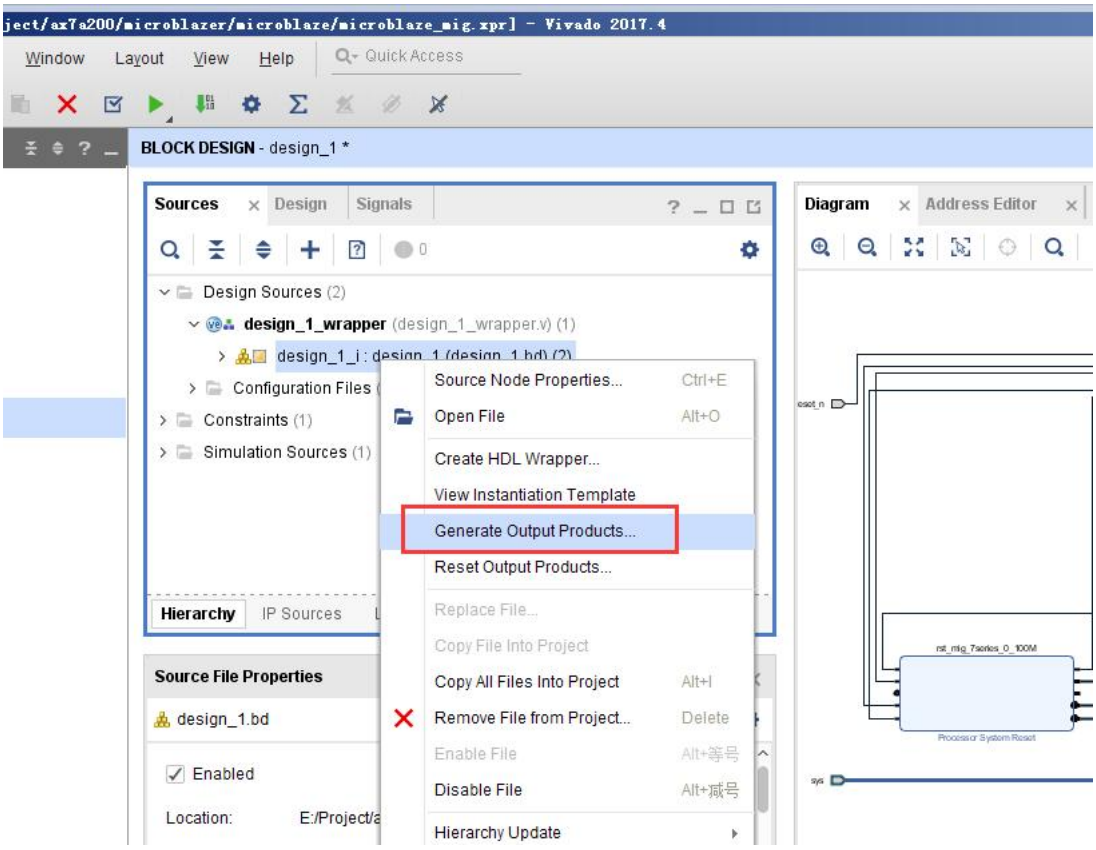


Part 6.1.3: Save and check for errors

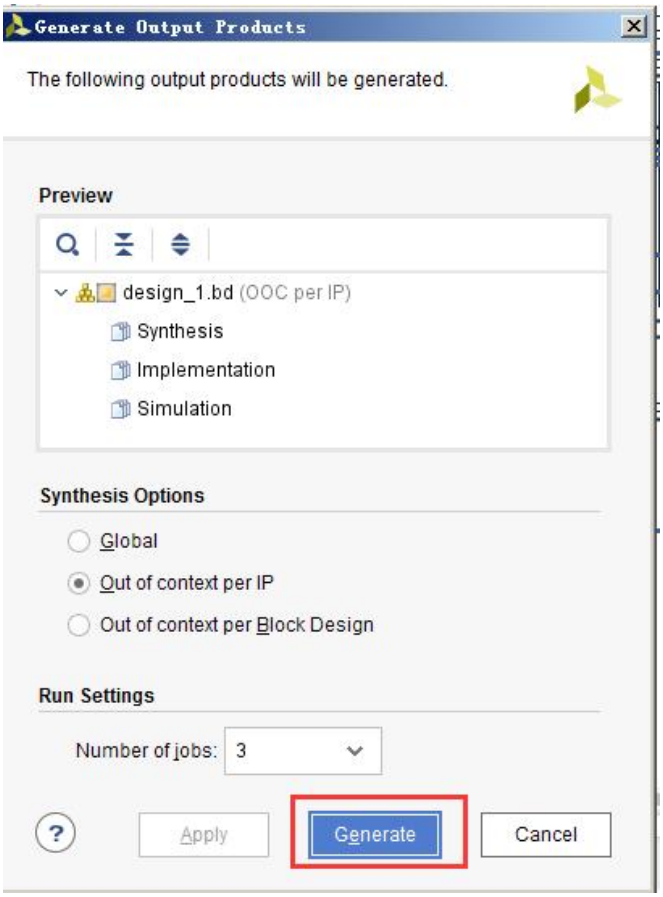
3) Save the design, and then press "F6" to check the design.



4) Select the "**design_1.bd**" file, right-click and select "**Generate Output Products**", update the IP parameters and connection information to the project.



5) Select the Generate button.

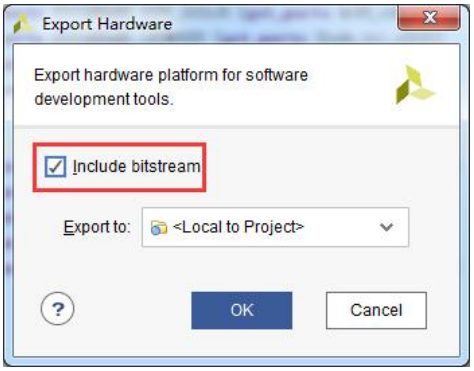


Part 6.1.4: Generate Bitstream file

6) Click "**Generate Bitstream**" to generate bit stream file.

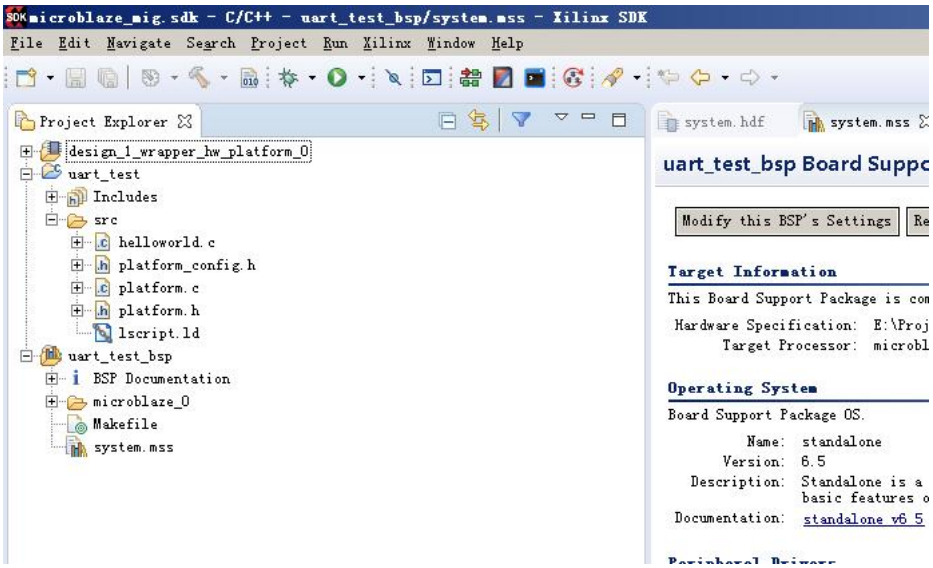
Part 6.1.5: Export hardware

7) Export the hardware in "**File -> Export -> Export Hardware...**" and select "**Include bitstream**" in the pop-up window to include the Bitstream file.



Part 6.2: SDK development of serial communication program (query method)

8) Create a **uart_test** project under the SDK.



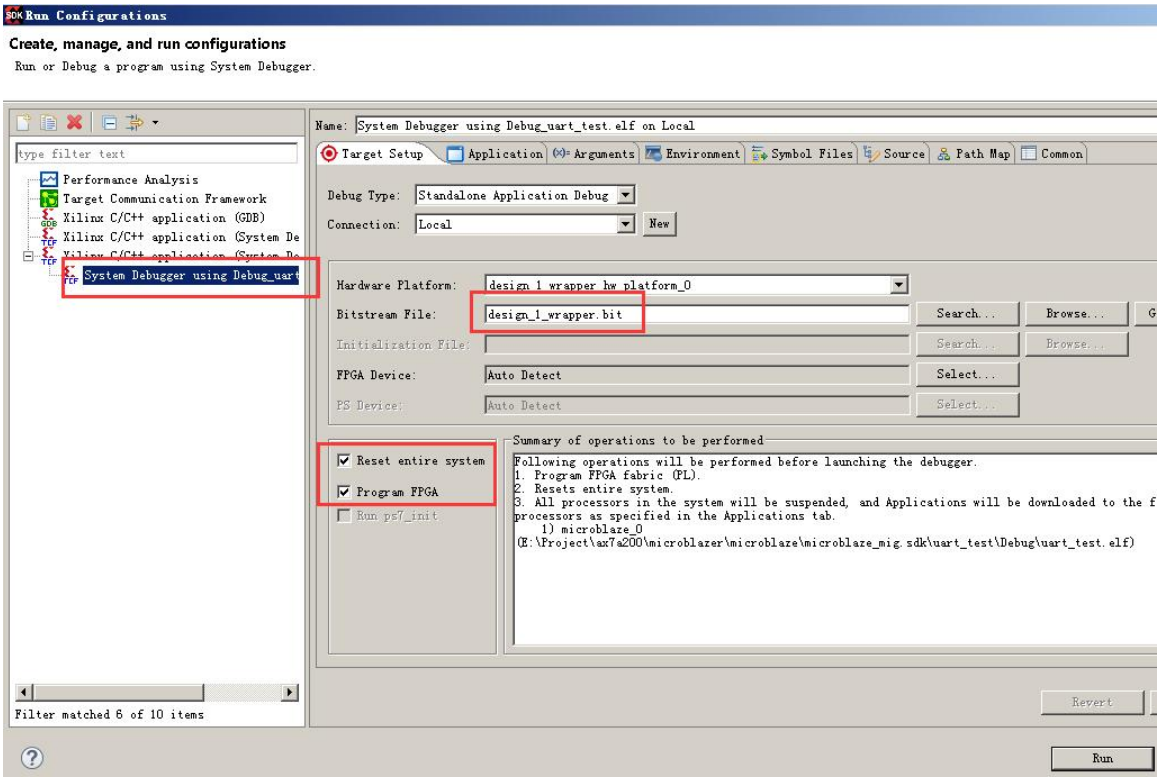
9) Modify the **helloworld.c** file as follows

```
/*************************** Include Files *******************************/

#include <stdio.h>
#include "xparameters.h"
#include "xuartlite_l.h"
#include "xuartlite.h"
#include "platform.h"

/* ------------------------------------------------------------ */
/*             main function                                    */
/* ------------------------------------------------------------ */

#define uart_baseaddr XPAR_AXI_UARTLITE_0_BASEADDR

int main()
{
    unsigned char rxdb;
    unsigned char i;
```

```
    init_platform();

   while(1)
   {

       if(!XUartLite_IsReceiveEmpty(uart_baseaddr))   //check receiver buffer is empty or
not
      {
        for (i=100;i>0;i--);
        rxdb = XUartLite_RecvByte(uart_baseaddr);    //receive one byte
        for (i=100;i>0;i--);
        XUartLite_SendByte(uart_baseaddr,rxdb);      //send one byte
        for (i=100;i>0;i--);
      }
   }
   return 0;
}
```

The program realizes serial communication by controlling the 3 registers of the **uartlite** controller. Refer to "**pg142-axi-uartlite.pdf**" for the definition of registers.

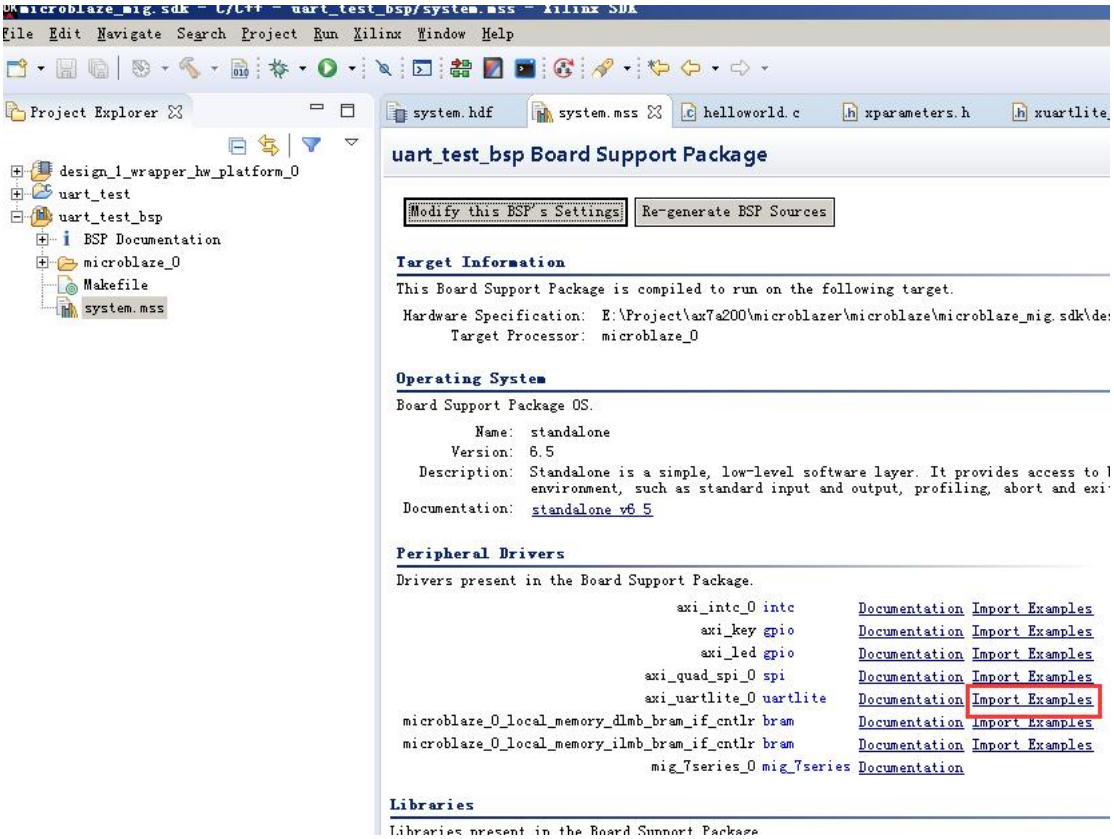
10) Select the "**uart_test**" project, right-click and select "**Run As -> Run Configuration...**", and configure the following in the pop-up interface.



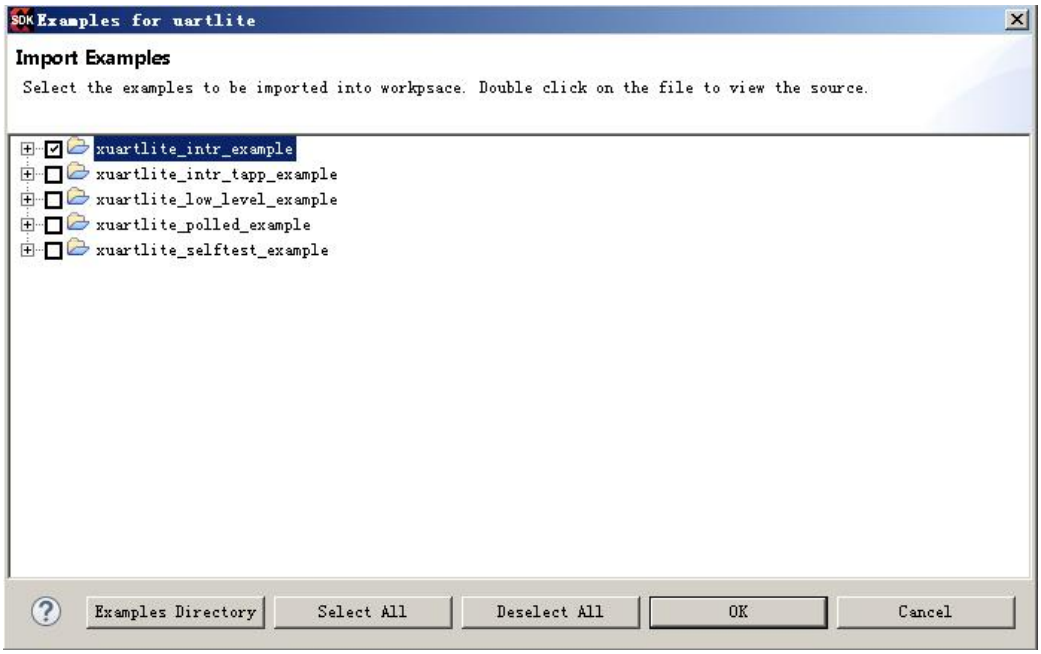
Connect the JTAG cable to the FPGA development board and Run to download. After downloading, enter the "**hello alinx**" character in the serial debugging tool, and then transmit, the same character will be returned in the receiving window.

Part 6.4: SDK development of serial communication program (interrupt mode)

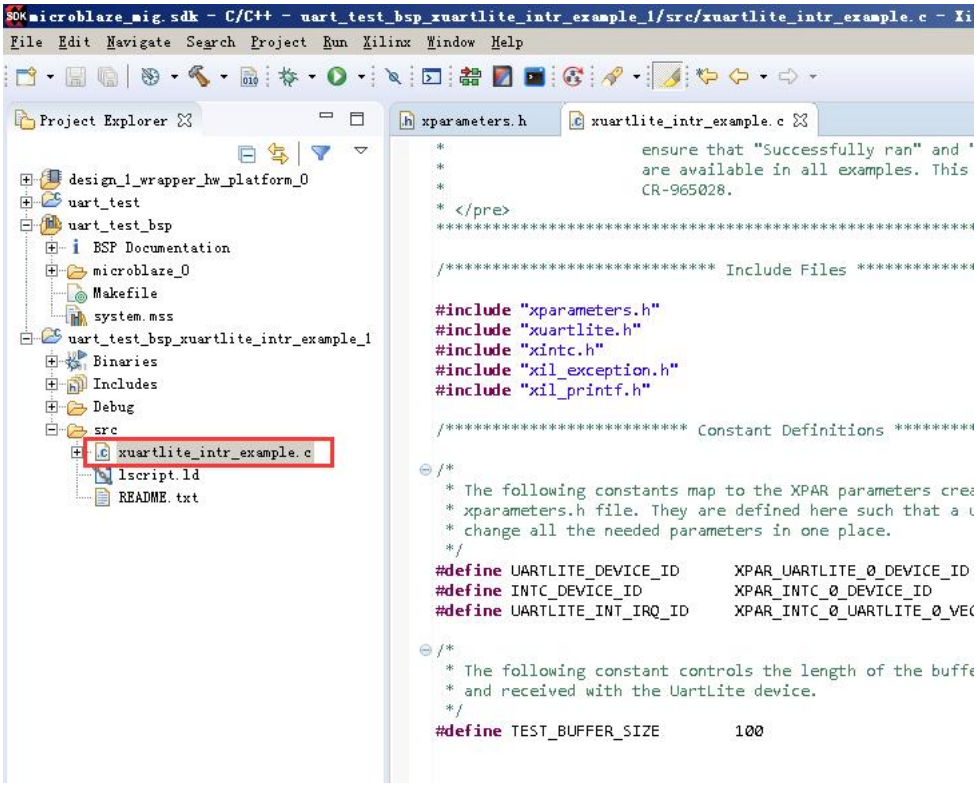
11)Double-click the "**system.mss**" file in the **BSP** directory, and click "**Import Examples**" next to "**axi_uartlite_0**".



12)Select "**uart_test_bsp_xuartlite_intr_example_1**"



13)Open the generated **xuartlite_intr_example** and take a look at the program inside.



In the program, the **uart** interrupt test is implemented through the **UartLiteIntrExample** program. The main functions used in it are:

XUartLite_Initialize (&UartLite, DeviceId) Serial port initialization

XUartLite_SelfTest (&UartLite) Serial port self-test

SetupInterruptSystem (&UartLite) creates an interrupt, this is also used in key interrupt

XUartLite_SetSendHandler(&UartLite, SendHandler, &UartLite) Send interrupt call function

XUartLite_SetRecvHandler(&UartLite, RecvHandler, &UartLite) Receive interrupt call function

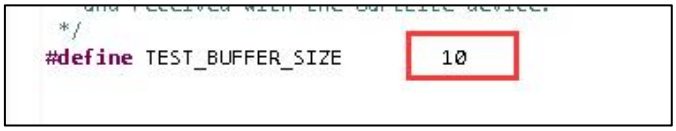
XUartLite_EnableInterrupt(&UartLite) Serial port interrupt enable

XUartLite_Recv(&UartLite, ReceiveBuffer, TEST_BUFFER_SIZE) Receive a certain amount of data to the buffer

XUartLite_Send(&UartLite, SendBuffer, TEST_BUFFER_SIZE)

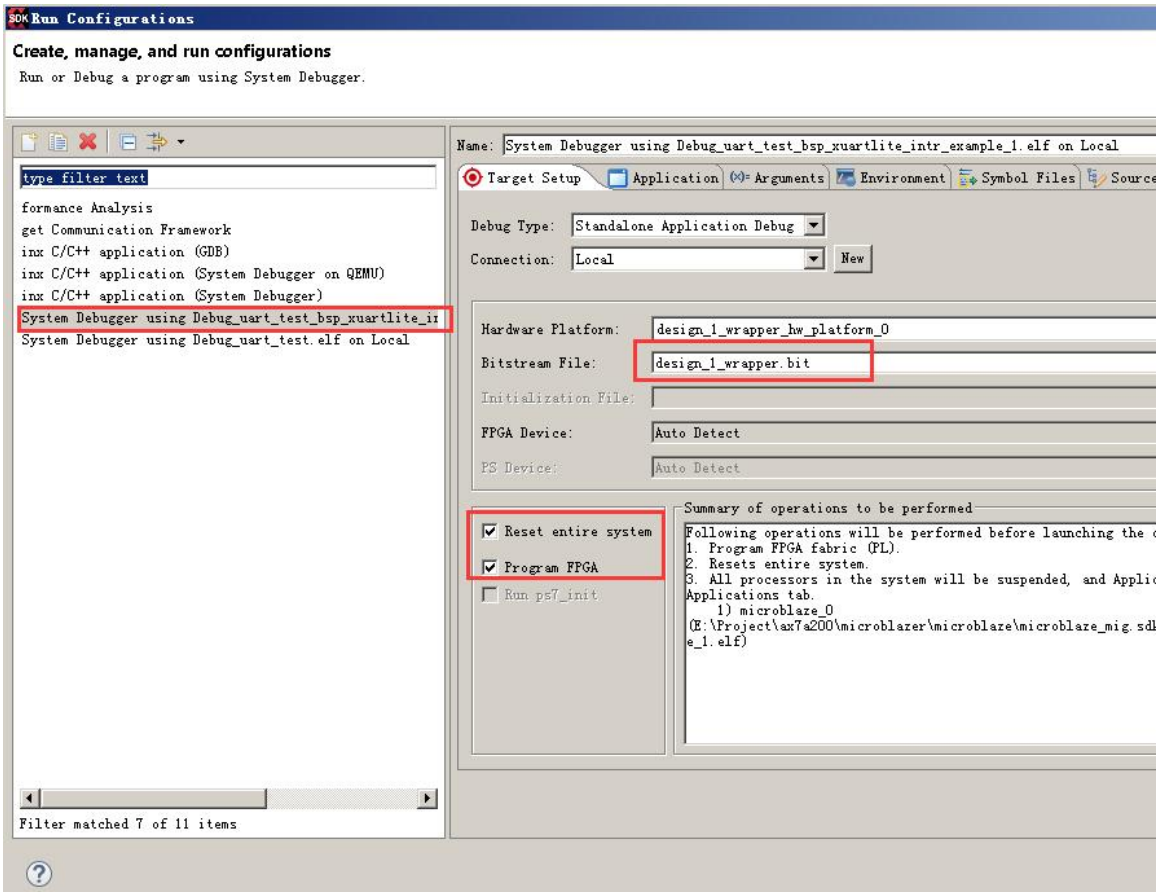
Send a certain amount of data in the buffer

14)In order to facilitate the test, we modify the number of test data to 10 here.



Part 6.5: Program download test

15)Select the "**uart_test_bsp_xuartlite_intr_example_1**" project, right-click and select "**Run As -> Run Configuration...**", and configure the following in the pop-up interface.



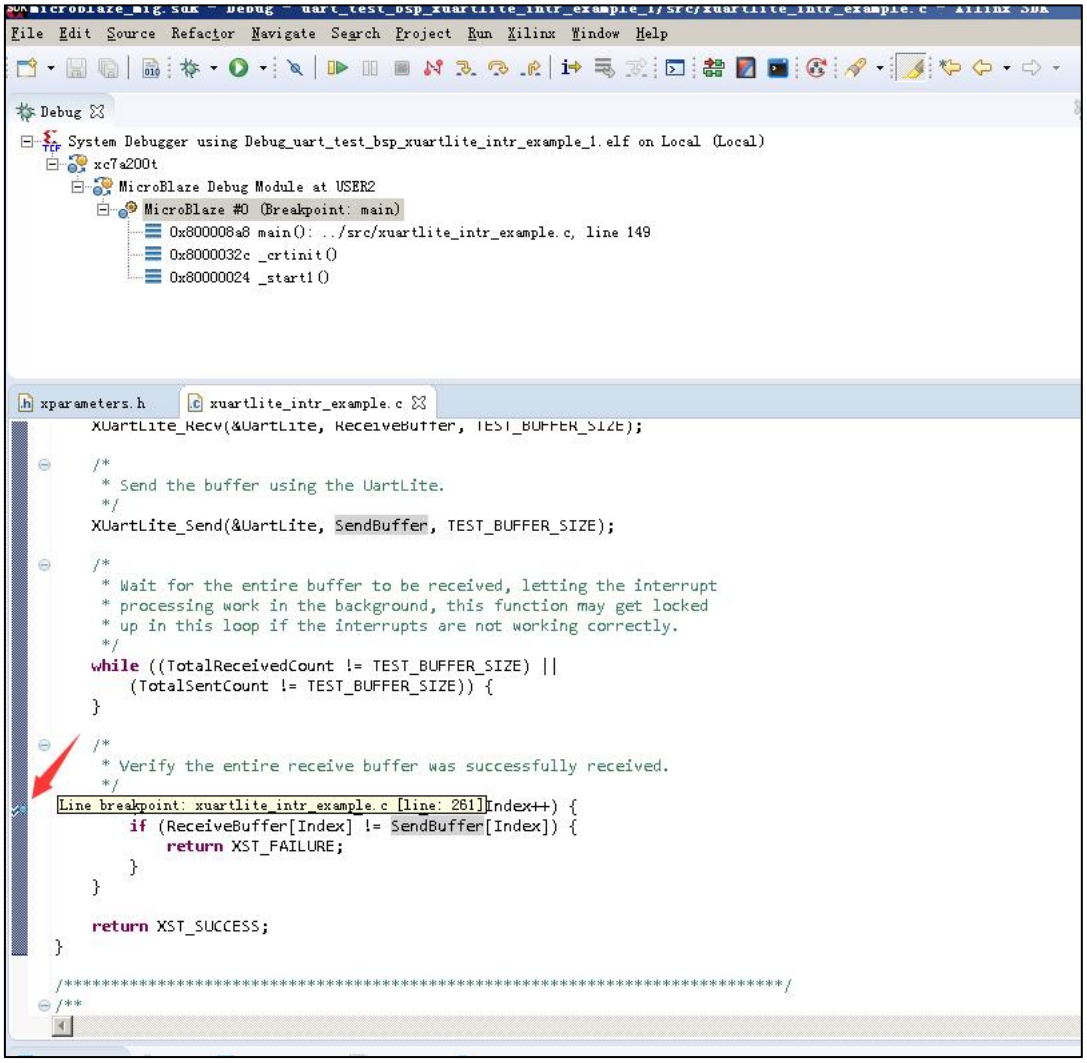
16)The serial port tool is set to hexadecimal display.

17)Connect JTAG and serial cable to FPGA development board, Run download. After downloading, the serial port displays 10 test data received.

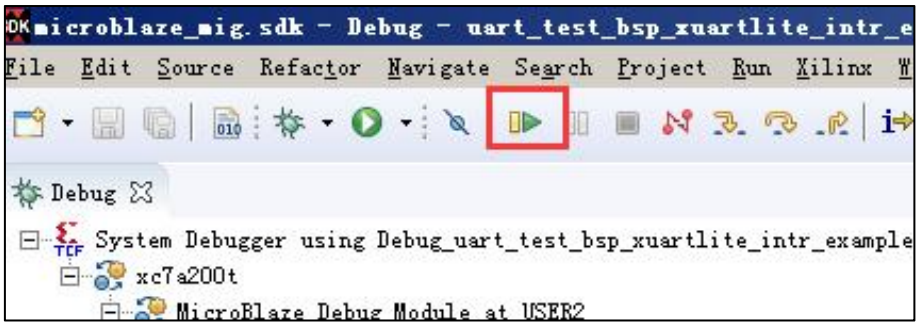
18)Then we need to copy these 10 data here and choose hexadecimal to transmit. In addition, remove the hexadecimal display tick so that you can continue to receive the character string sent by the subsequent program. Click to transmit manually.

19)After the program receives 10 data, it will compare with the transmit data, and if it is correct, it will transmit a successful message.

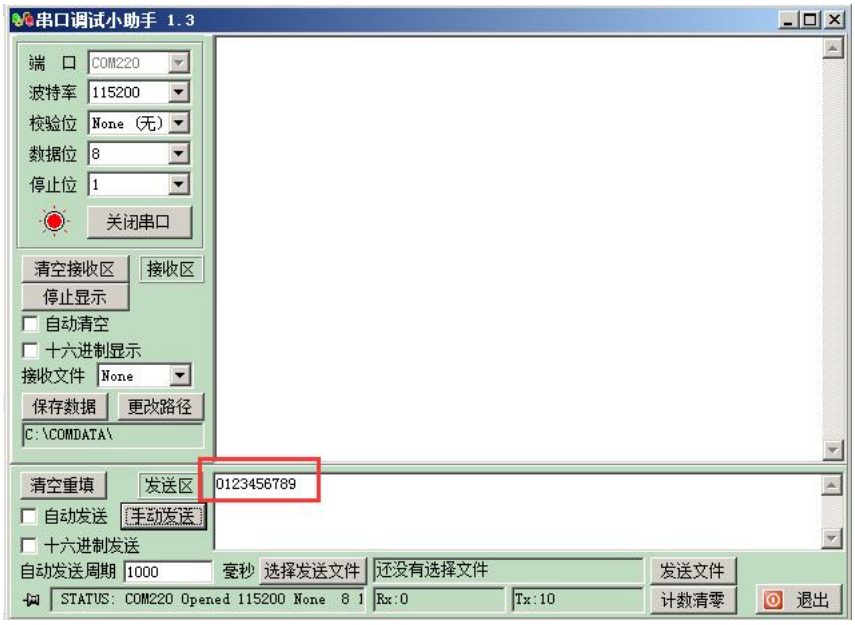
20)We can also view the received data in the program through **Debug**. After entering "**Debug as hardware**", add a breakpoint to the following statement to stop the program.



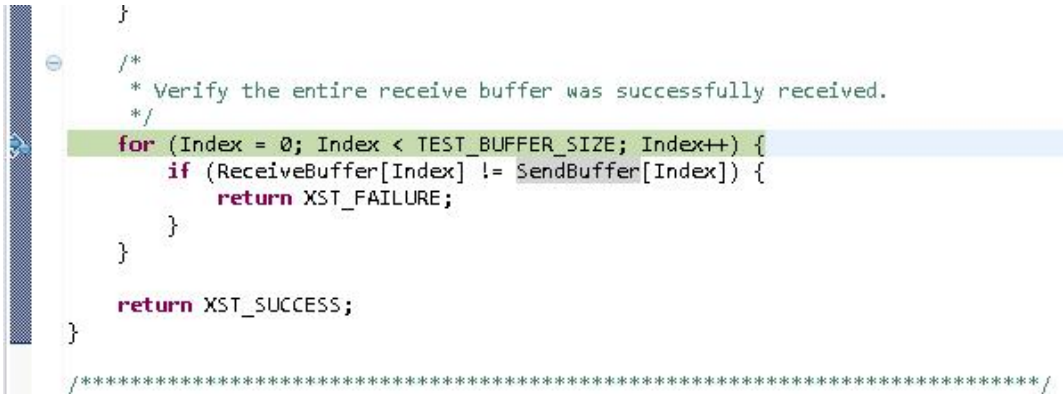
21)Click the **run** button.



22)Transmit 10 characters of data through the serial port, such as "**0123456789**", do not choose hexadecimal.

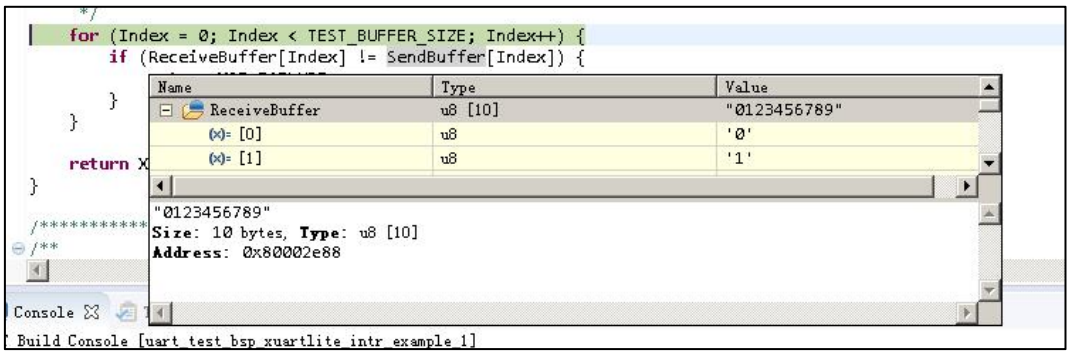


23)When the data is received, the program runs to the breakpoint.



```
    }

    /*
     * Verify the entire receive buffer was successfully received.
     */
    for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
        if (ReceiveBuffer[Index] != SendBuffer[Index]) {
            return XST_FAILURE;
        }
    }

    return XST_SUCCESS;
}

/*****************************************************************************/
```

24)View the data of **ReceiveBuffer**, which is exactly the same as the data transmit by our serial port.



Part 7: Ethernet Experiment (LWIP)

The experimental Vivado project is "net_test".

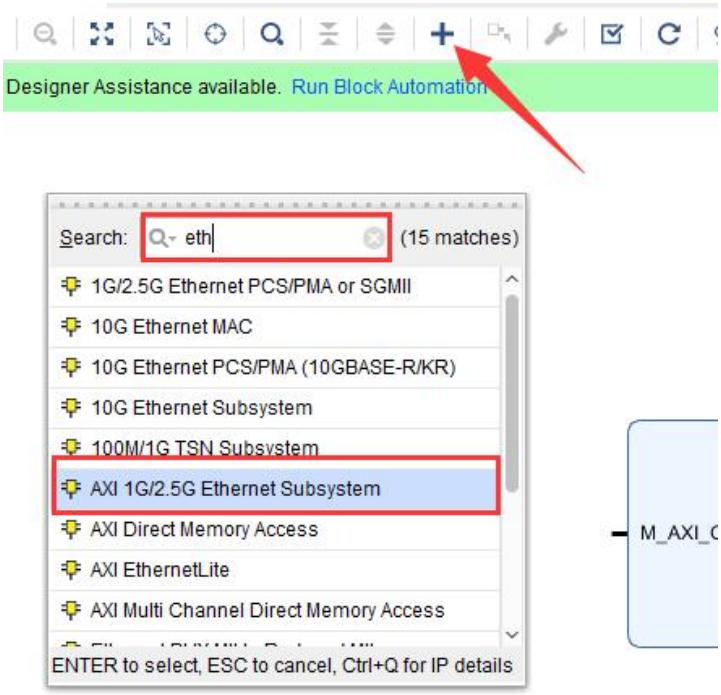
Take the AX7A200 FPGA development board as an example. The board has 1 Gigabit Ethernet, which can be connected through the RGMII interface and can be controlled by FPGA. This experiment demonstrates how to use the LWIP template that comes with the SDK for Gigabit Ethernet TCP communication. Although LWIP is a lightweight protocol stack, if you have never used it, it will be difficult to use. It is recommended to be familiar with the relevant knowledge of LWIP.

Part 7.1: Vivado Project Modification

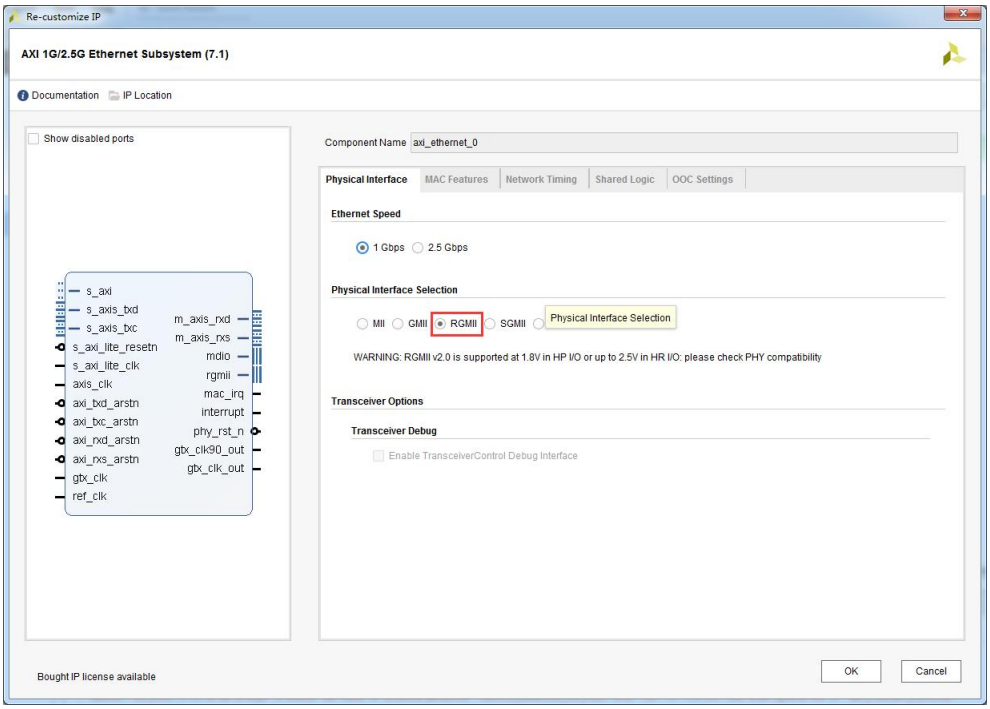
The Vivado project is modified on the basis of the Vivado project configuration in Part 6, and the SDK directory is deleted.

Part 7.1.1: Add Ethernet IP

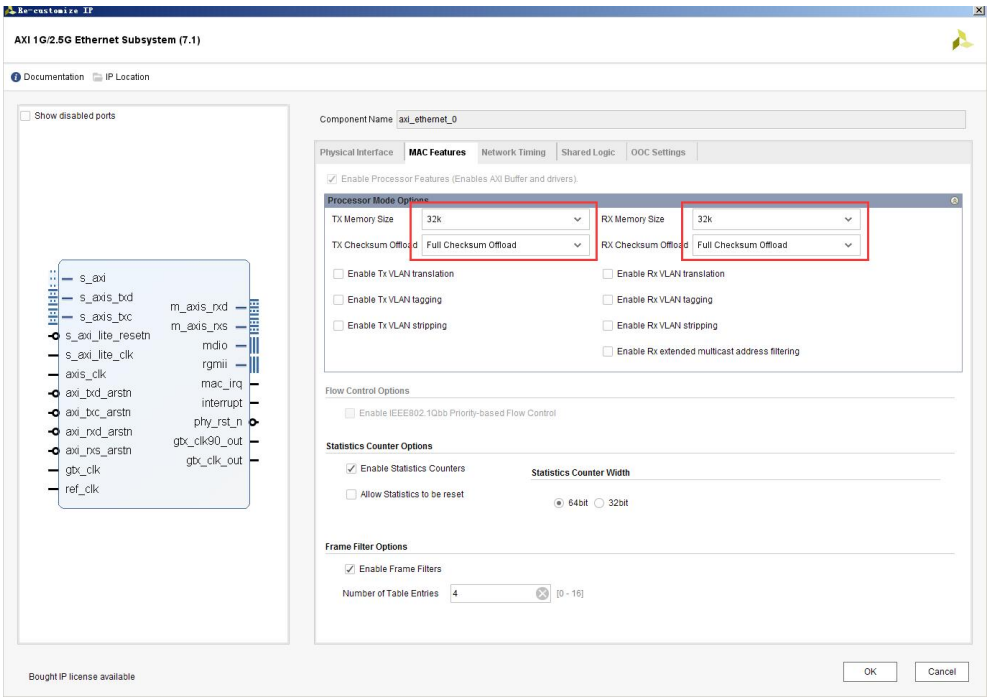
1) Search for "**eth**" and add an "**AXI 1G/2.5G Ethernet Subsystem**"



2) Double-click the module just added, modify the parameters, and select "**RGMII**" for the physical interface (if it is an FPGA development board with other **GMII** interfaces, save the default **GMII** interface)

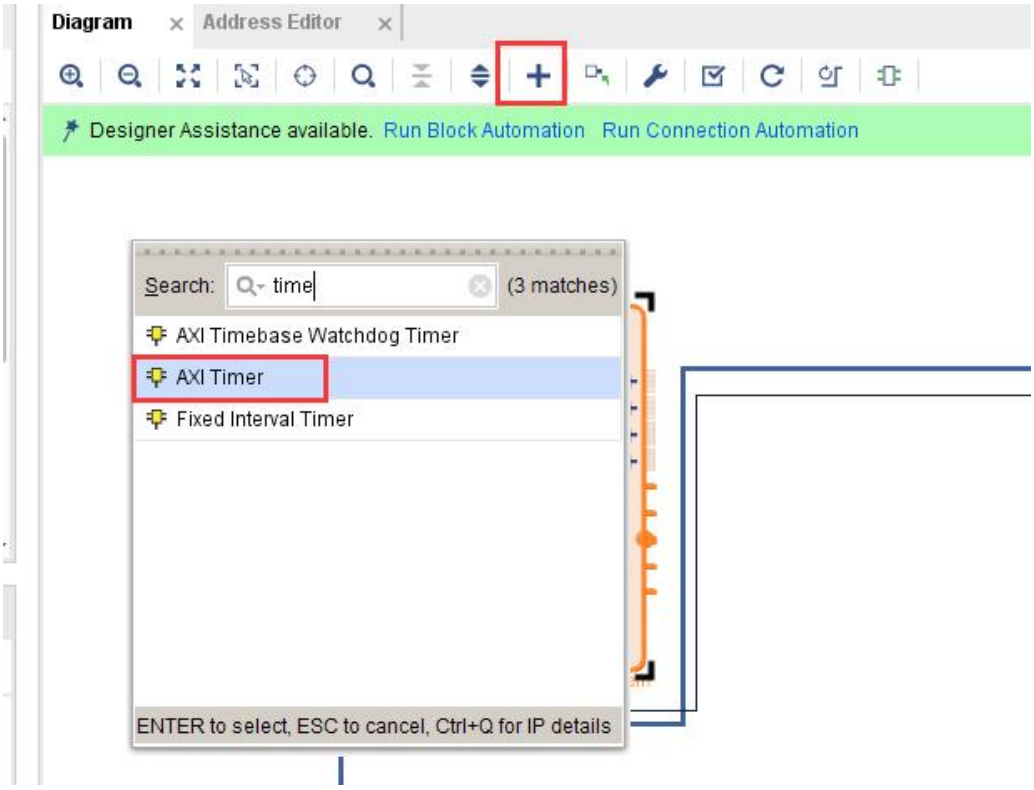


3) Select the largest **32K** capacity and "**full Checksum Offload**" in the **MAC feature** page.

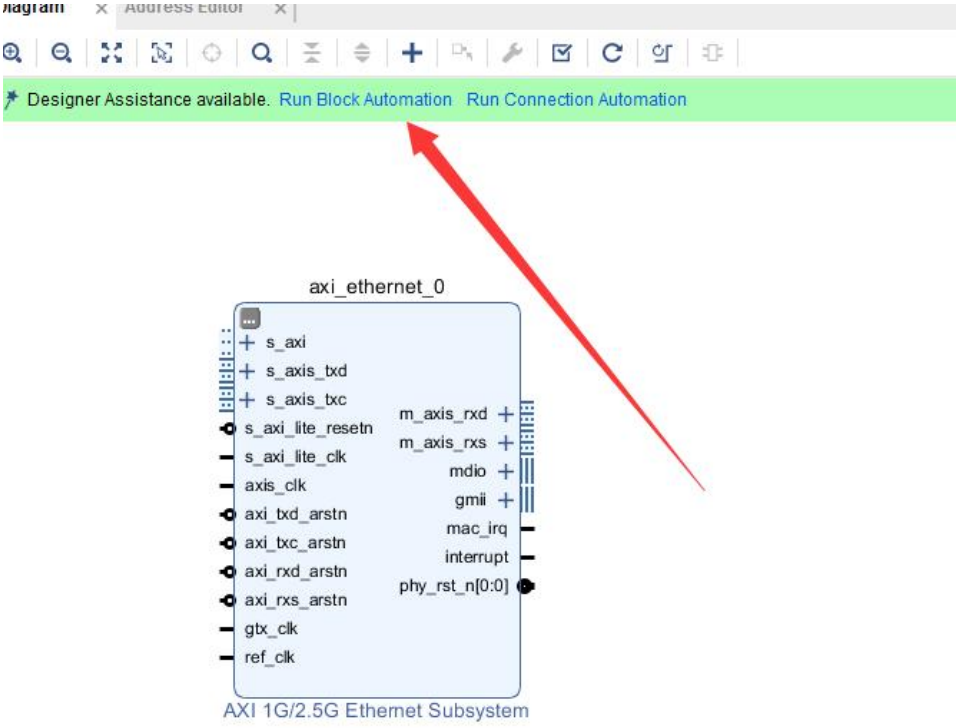


Part 7.1.2: Add Timer

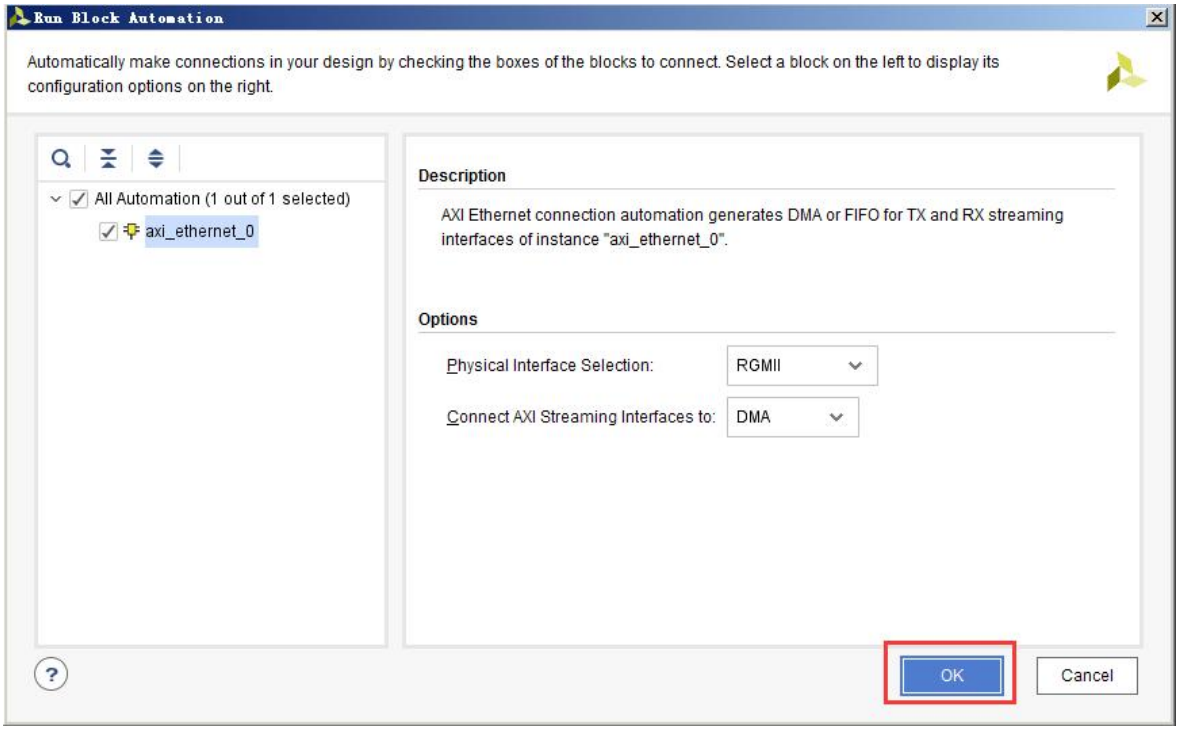
4) Ethernet communication requires an **axi timer IP**. Search for "**timer**" and select AXI Timer.



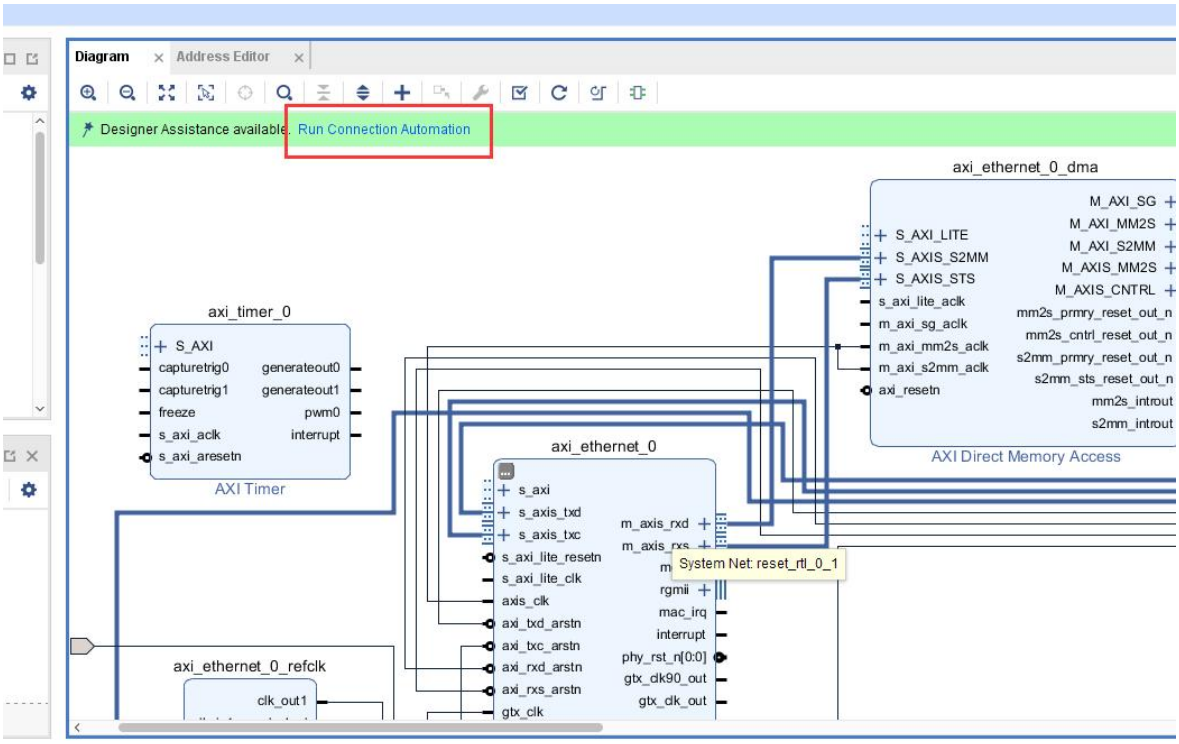
5) Click "**Run Block Automation**"



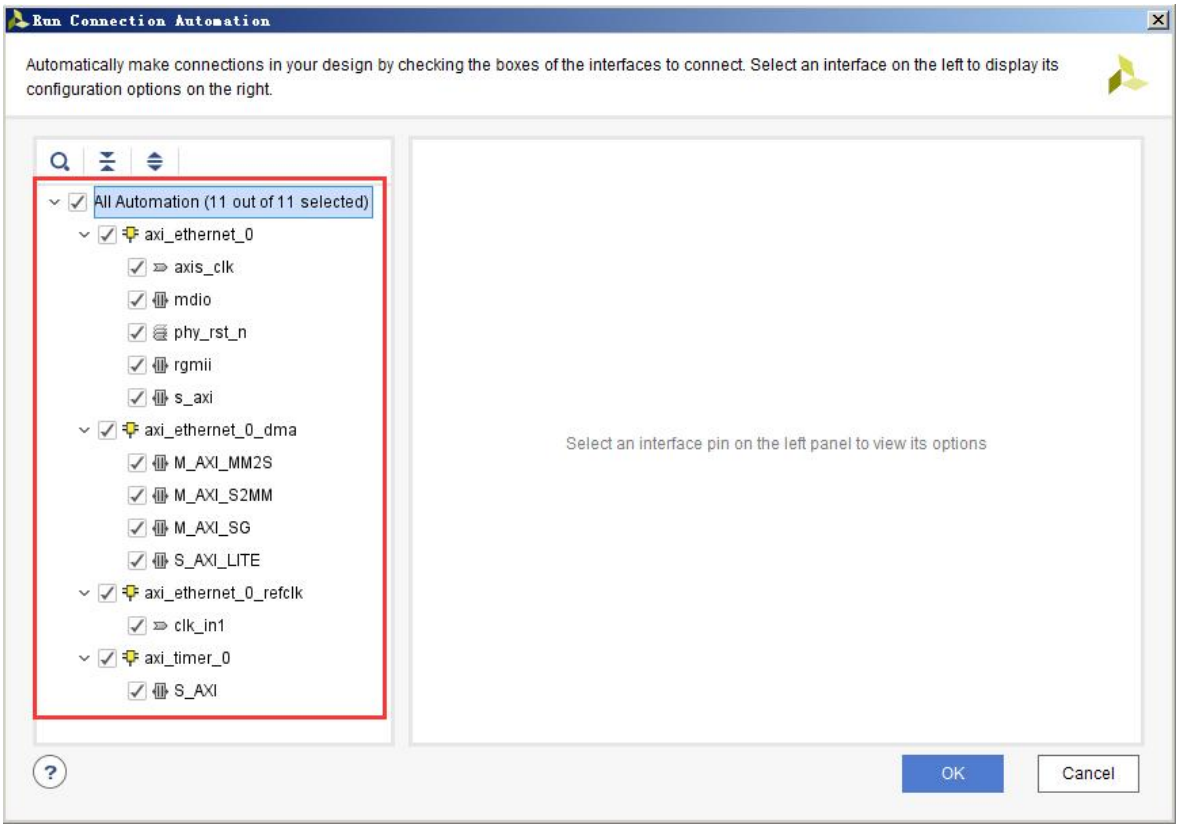
6) Select the automatic connection of **ethernet IP**, the software will automatically add **DMA IP**.



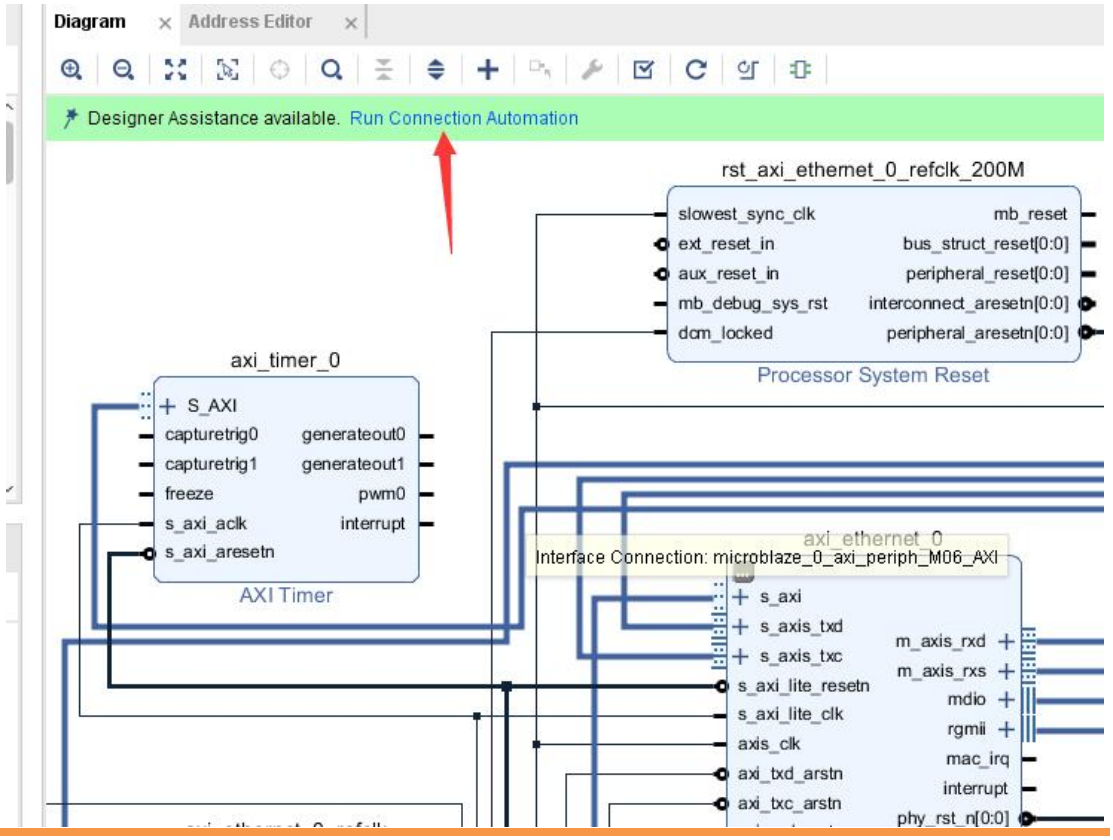
7) At this time, Vivado can automatically insert many modules, such as **DMA IP** and **clock IP**, click "**Run Connection Automation**"



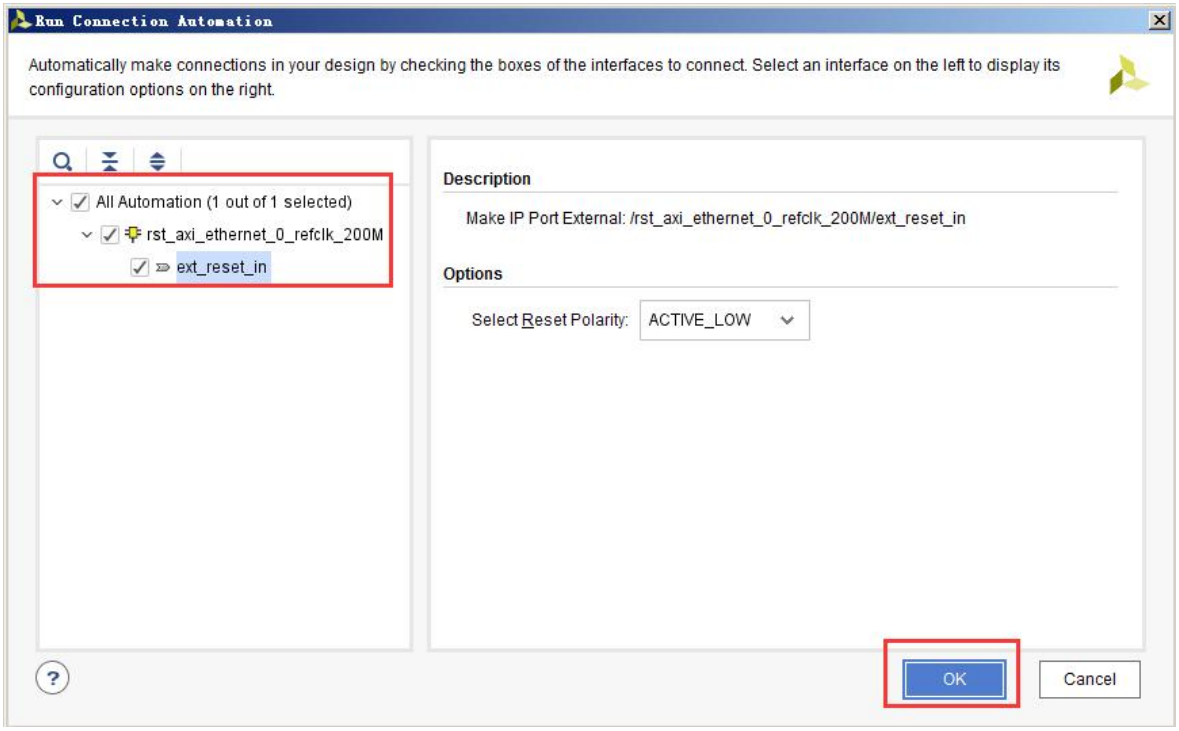
8) Select all automation



9) Then you did not complete the connection at one time, click "**Run Connection Automation**" again

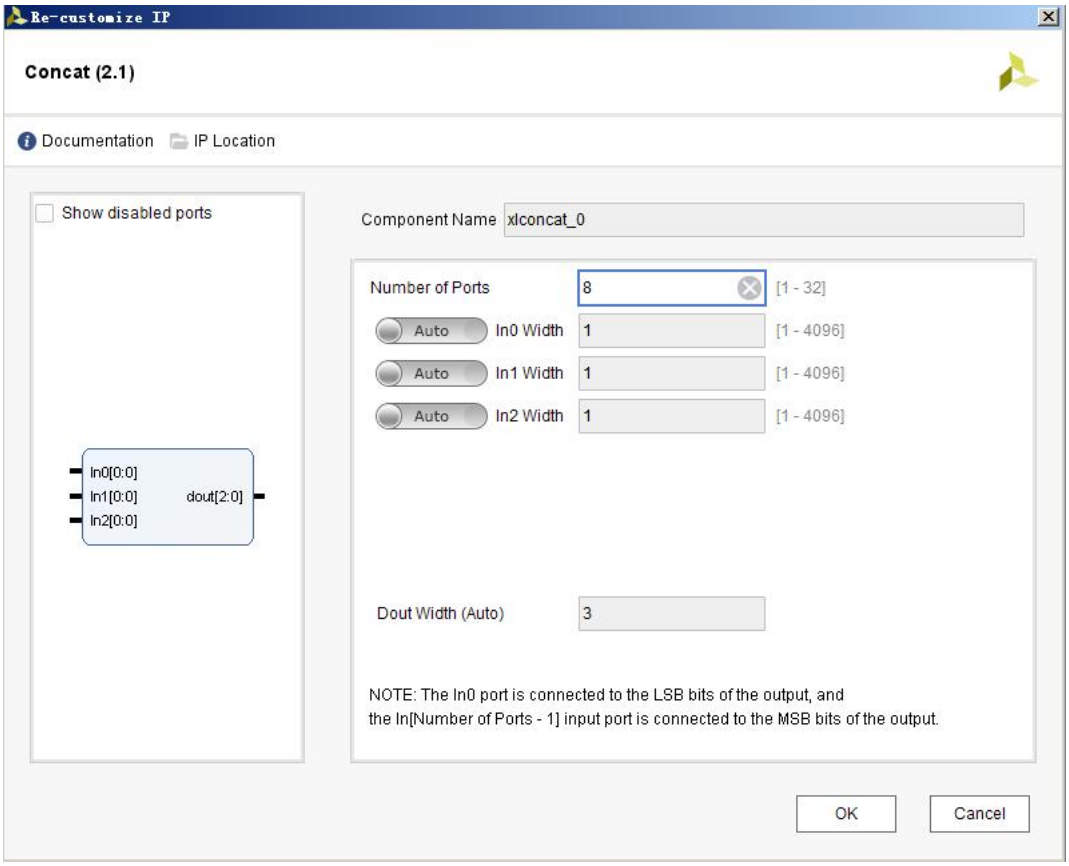


10)Select all



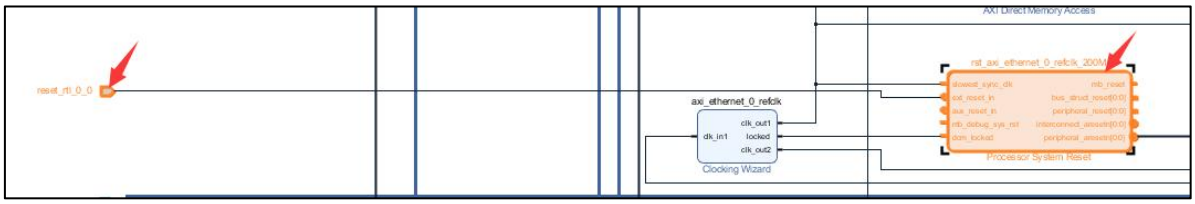
Part 7.1.3: Modify Constant

11)Double-click the "Concat" icon, the number of input ports is 8



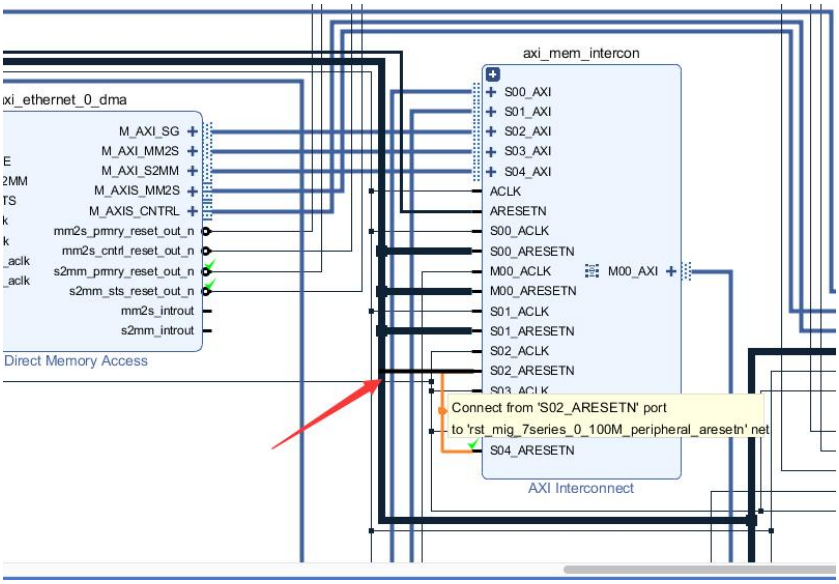
Part 7.1.4: tDelete rst_axi_ethernet_0_refclk_200M

12)Delete rst_axi_ethernet_0_refclk_200M and input port reset_rtl_0_0

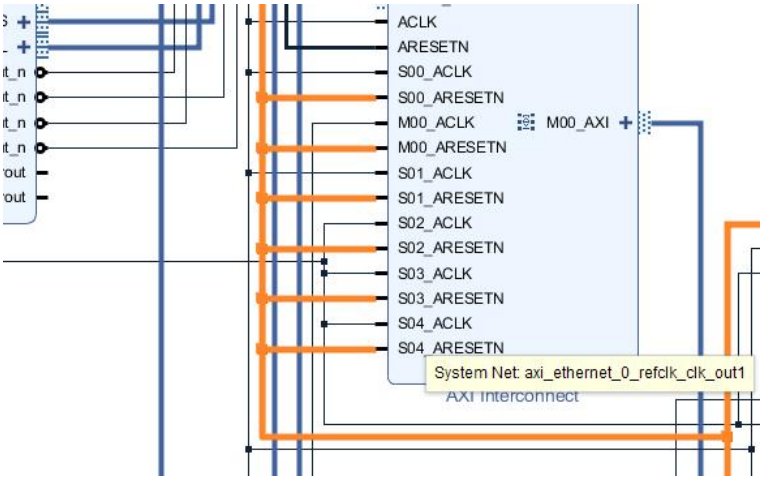


Part 7.1.5: Signal connection

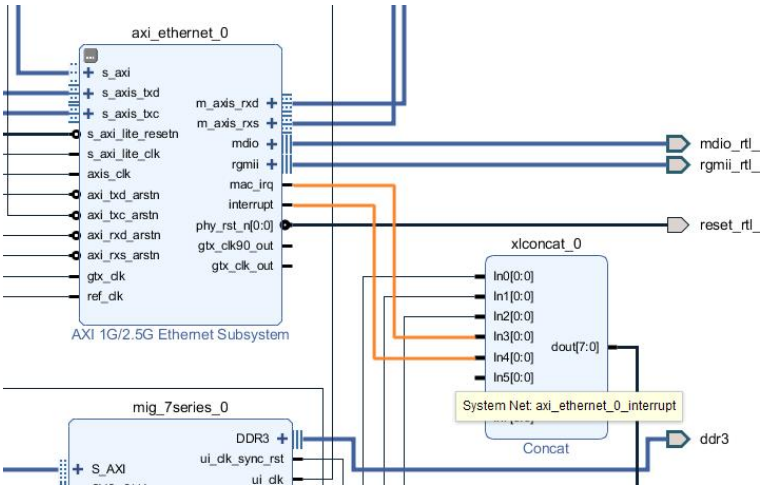
13)S02_ARESETN, S03_ARESETN, S04_ARESETN and S01_ARESETN of axi_mem_intercon are connected together.



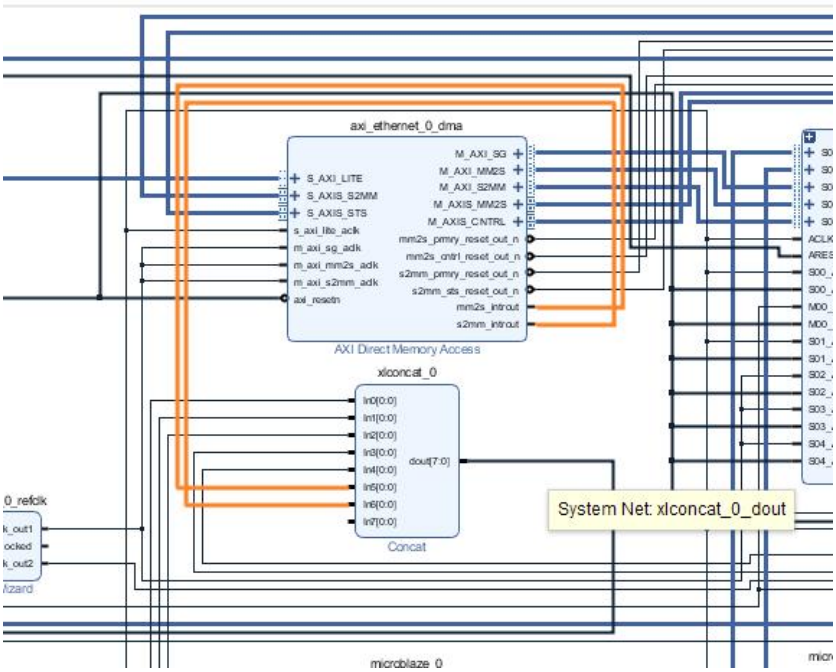
After the connection is as follows:



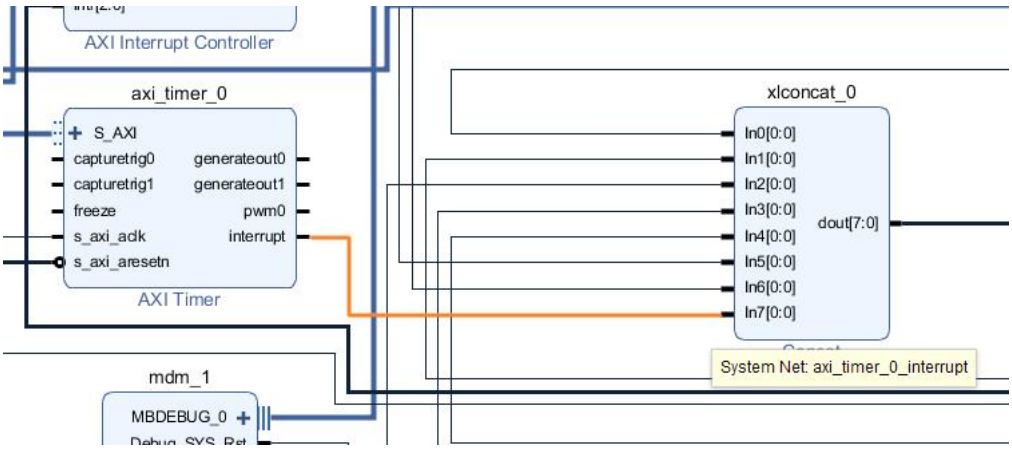
14)Connect the 2 interrupt signals of axi_ethernet_0



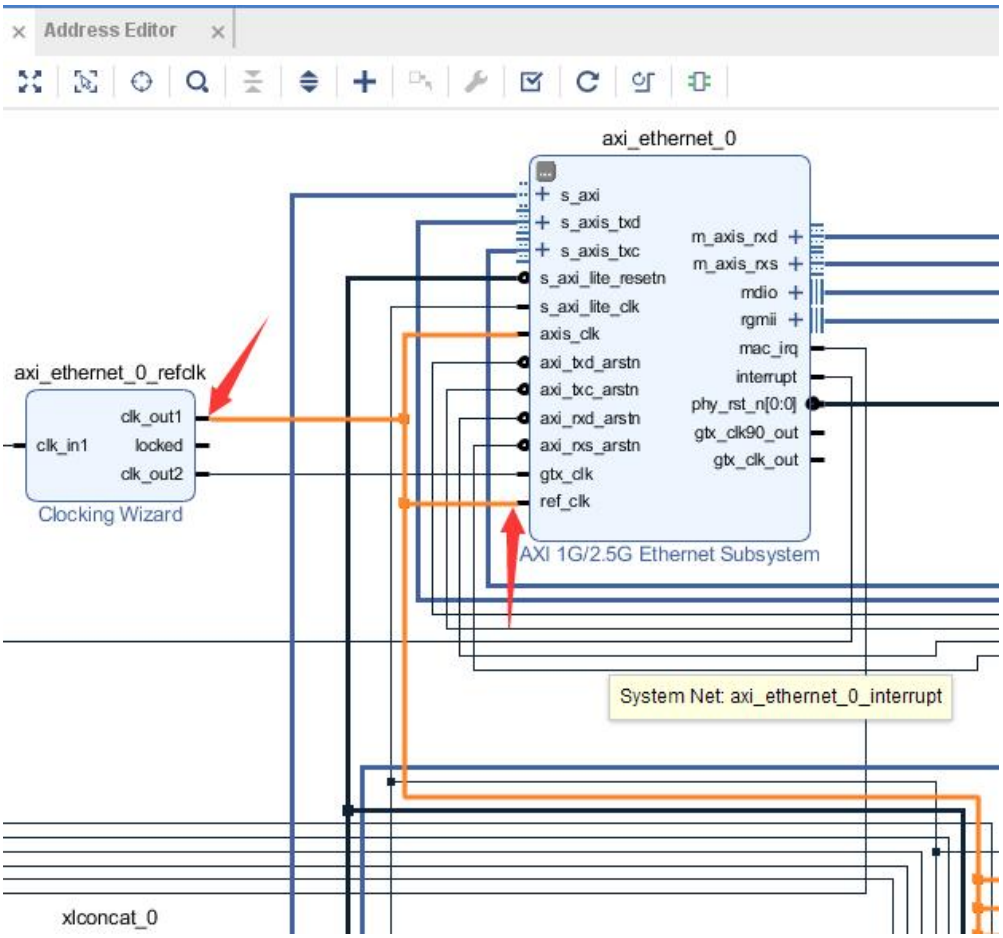
15)Connect the 2 interrupt signals of axi_ethernet_0_dma



16)Connect the interrupt signal of **axi_timer_0**

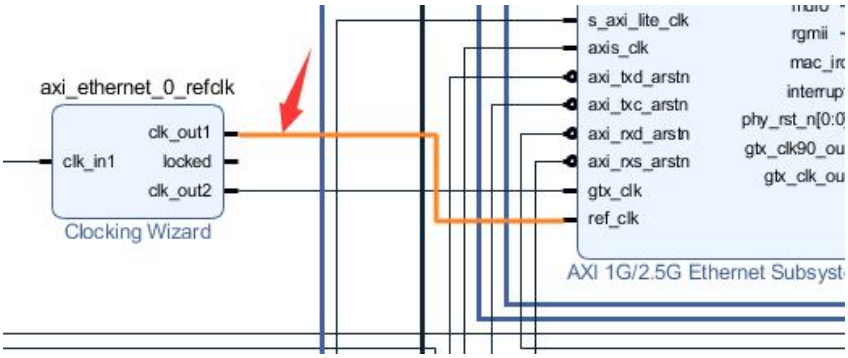


17)Disconnect the following 2 pins

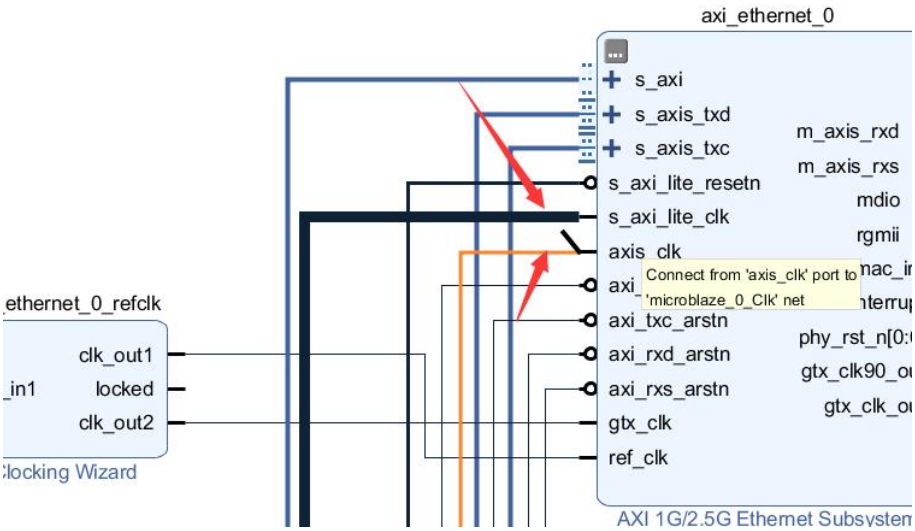


Select **Pin** respectively, right click and select "Disconnect Pin"

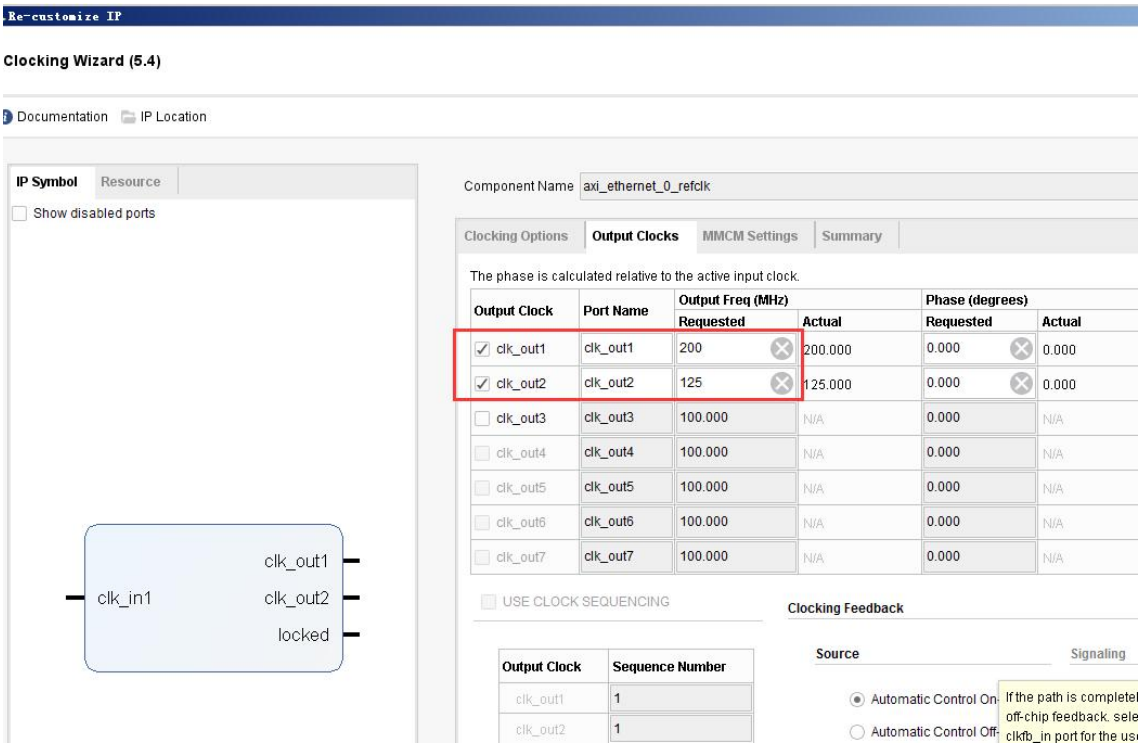
18)Then single-ended connection.



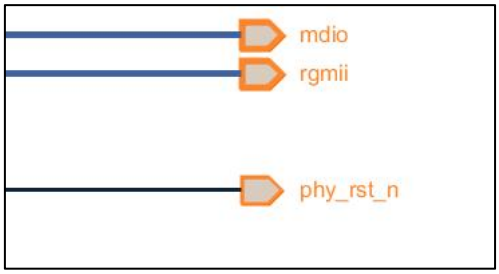
19)Connect the **axis_clk** clock to the **s_axi_lite_clk clock** connector



20)Confirm that one of the output clocks of **axi_ethernet_0_refclk** is **200M** and the other is **125M**.

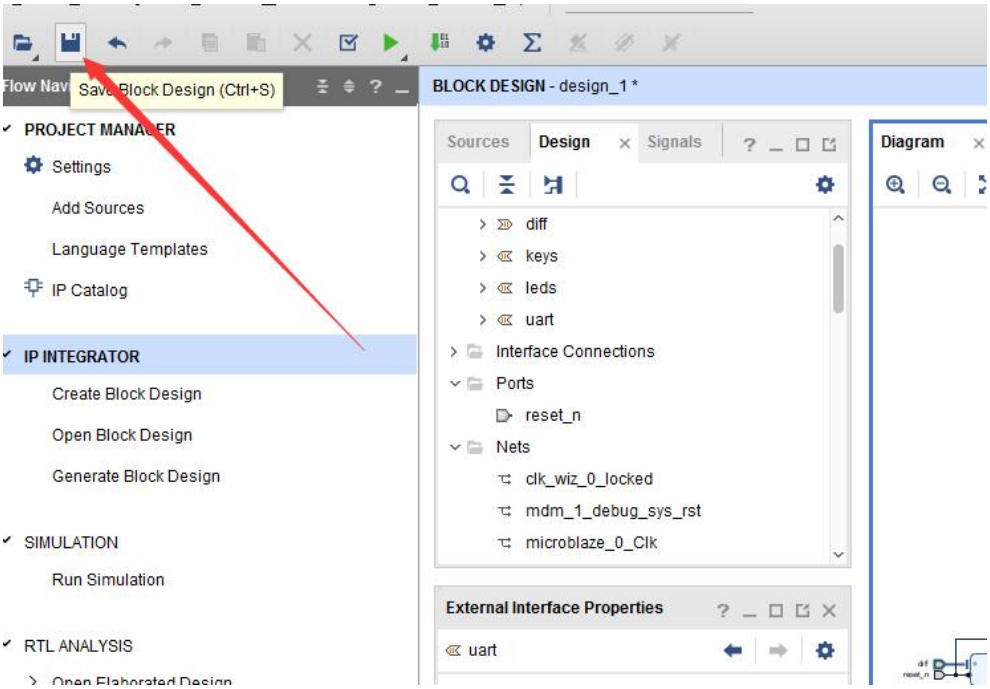


21)Modify the names of other ports

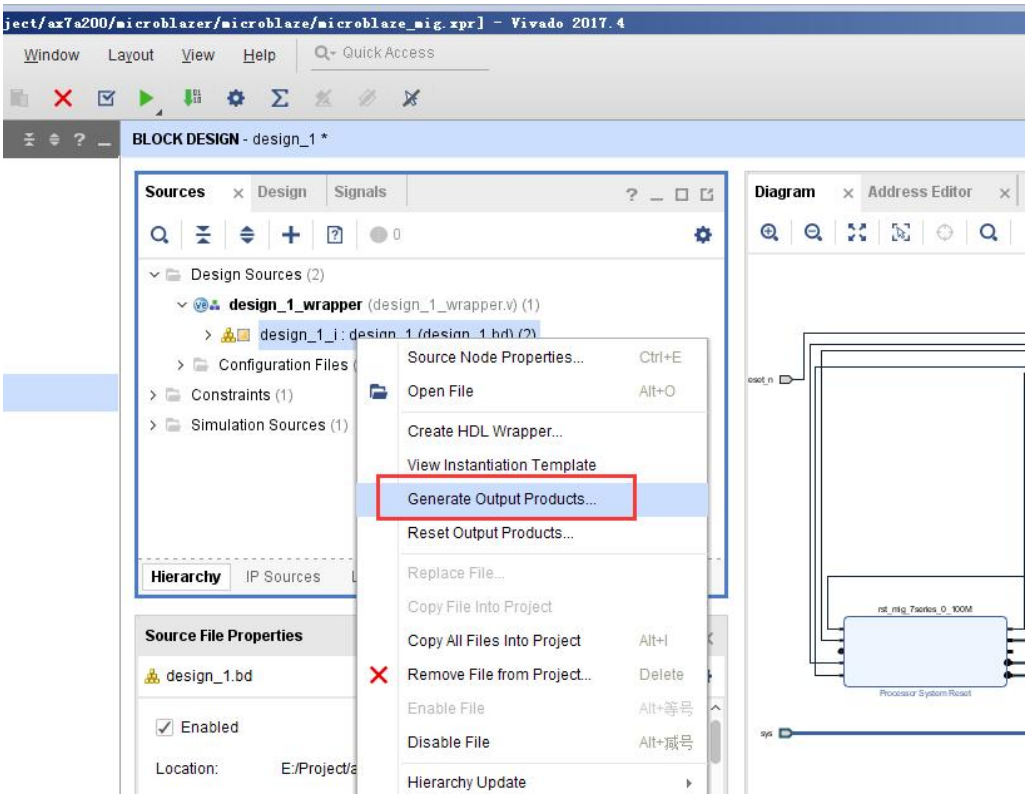


Part 7.1.6: Save and check for errors

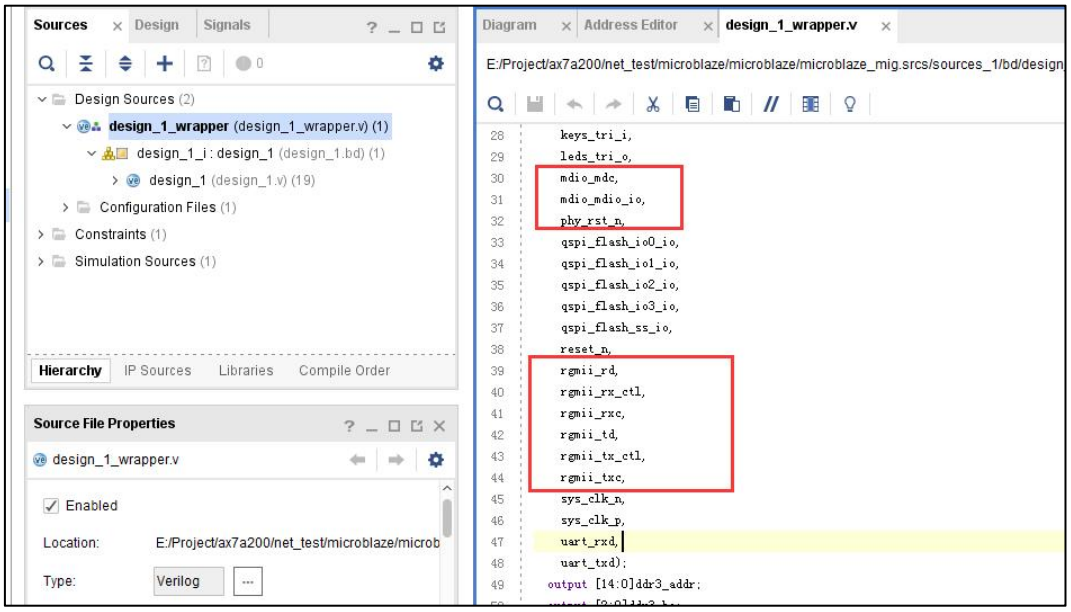
22)Save the design, and then press "**F6**" to check the design



23)Select the "**design_1.bd**" file, right-click and select "**Generate Output Products**", update the IP parameters and connection information to the project.



After the update, the port of the design_1_wrapper.v file will have more Ethernet interface signals.



Part 7.1.7: Add the restriction file of Ethernet (take ax7a200 as an example)

```
set_property PACKAGE_PIN N13 [get_ports {mdio_mdc                }]
set_property PACKAGE_PIN P14 [get_ports {mdio_mdio_io          }]
set_property PACKAGE_PIN R14 [get_ports {phy_rst_n         }]
set_property PACKAGE_PIN V18 [get_ports {rgmii_rxc               }]
set_property PACKAGE_PIN R19 [get_ports {rgmii_rx_ctl          }]
set_property PACKAGE_PIN P19 [get_ports {rgmii_rd[0]           }]
```

```
set_property PACKAGE_PIN U18 [get_ports {rgmii_rd[1]           }]
set_property PACKAGE_PIN U17 [get_ports {rgmii_rd[2]           }]
set_property PACKAGE_PIN P17 [get_ports {rgmii_rd[3]           }]
set_property PACKAGE_PIN P15 [get_ports {rgmii_txc             }]
set_property PACKAGE_PIN N17 [get_ports {rgmii_tx_ctl          }]
set_property PACKAGE_PIN N14 [get_ports {rgmii_td[0]           }]
set_property PACKAGE_PIN P16 [get_ports {rgmii_td[1]           }]
set_property PACKAGE_PIN R17 [get_ports {rgmii_td[2]           }]
set_property PACKAGE_PIN R16 [get_ports {rgmii_td[3]           }]

set_property IOSTANDARD LVCMOS33 [get_ports {mdio_mdc            }]
set_property IOSTANDARD LVCMOS33 [get_ports {mdio_mdio_io        }]
set_property IOSTANDARD LVCMOS33 [get_ports {phy_rst_n       }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rxc           }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rx_ctl        }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rd[0]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rd[1]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rd[2]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_rd[3]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_txc           }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_tx_ctl        }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_td[0]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_td[1]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_td[2]         }]
set_property IOSTANDARD LVCMOS33 [get_ports {rgmii_td[3]         }]
```

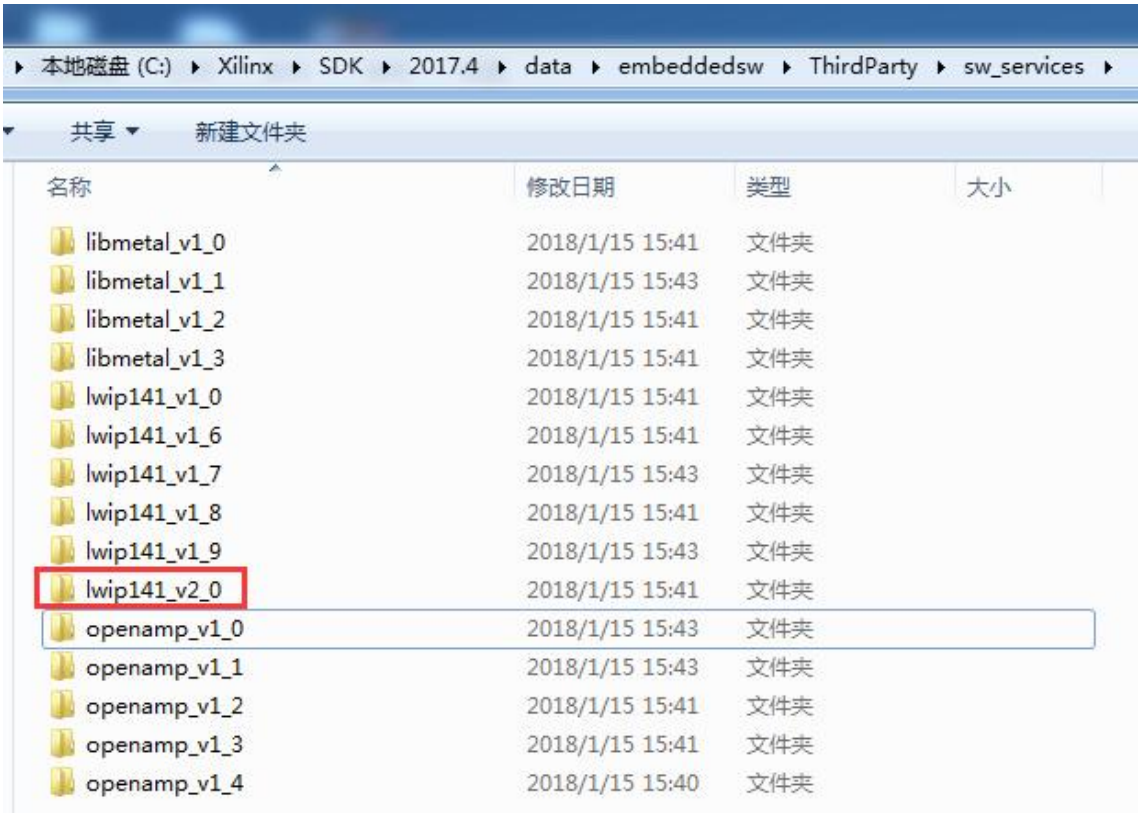
24)Compile and generate bit file, then export hardware information, start SDK

Part 7.2: SDK program

Part 7.2.1: LWIP library modification

Since the built-in LWIP library can only recognize some phy chips, if the phy chip used by the FPGA development board is not within the default support range, the library file must be modified. You can also directly use the modified library to replace the original library.

1) Find the library file directory "**X:\Xilinx\SDK\2017.4\data\embeddedsw\ThirdParty\sw_services**"



2) Find the files "**xaxiemacif_physpeed.c**" and "**xemacpsif_physpeed.c**" in the directory "**lwip141_v2_0\src\contrib\ports\xilinx\netif**" to be modified.

3) Modify the "**xaxiemacif_physpeed.c**" file and add related macro definitions

```
#define IEEE_MMD_ACCESS_CTRL_DEVAD_MASK          0x1F
#define IEEE_MMD_ACCESS_CTRL_PIDEVAD_MASK        0x801F
#define IEEE_MMD_ACCESS_CTRL_NOPIDEVAD_MASK      0x401F

#define PHY_R0_ISOLATE                           0x0400
#define PHY_DETECT_REG                           1
#define PHY_IDENTIFIER_1_REG                     2
#define PHY_IDENTIFIER_2_REG                     3
#define PHY_DETECT_MASK                          0x1808
#define PHY_MARVELL_IDENTIFIER                   0x0141
#define PHY_TI_IDENTIFIER                        0x2000

/* Marvel PHY flags */
#define MARVEL_PHY_IDENTIFIER                    0x141
#define MARVEL_PHY_MODEL_NUM_MASK                0x3F0
#define MARVEL_PHY_88E1111_MODEL                 0xC0
#define MARVEL_PHY_88E1116R_MODEL                0x240
#define PHY_88E1111_RGMII_RX_CLOCK_DELAYED_MASK  0x0080

/* TI PHY Flags */
#define TI_PHY_DETECT_MASK                       0x796D
#define TI_PHY_IDENTIFIER                        0x2000
#define TI_PHY_DP83867_MODEL                     0xA231
#define DP83867_RGMII_CLOCK_DELAY_CTRL_MASK      0x0003
#define DP83867_RGMII_TX_CLOCK_DELAY_MASK        0x0030
#define DP83867_RGMII_RX_CLOCK_DELAY_MASK        0x0003

/* TI DP83867 PHY Registers */
#define DP83867_R32_RGMIICTL1                    0x32
#define DP83867_R86_RGMIIDCTL                    0x86

#define MICREL_PHY_IDENTIFIER                    0x22
#define MICREL_PHY_KSZ9031_MODEL                 0x220

#define TI_PHY_REGCR            0xD
#define TI_PHY_ADDDR            0xE
```

4) Add phy speed acquisition function

```
unsigned int get_phy_speed_ksz9031(XAxiEthernet *xaxiemacp, u32 phy_addr)
{
     u16 control;
     u16 status;
     u16 partner_capabilities;
     xil_printf("Start PHY autonegotiation \r\n");

     XAxiEthernet_PhyWrite(xaxiemacp,phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 2);
     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_CONTROL_REG_MAC, &control);
     //control |= IEEE_RGMII_TXRX_CLOCK_DELAYED_MASK;
     control &= ~(0x10);
     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_CONTROL_REG_MAC, control);

     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 0);

     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_AUTONEGO_ADVERTISE_REG, &control);
     control |= IEEE_ASYMMETRIC_PAUSE_MASK;
     control |= IEEE_PAUSE_MASK;
     control |= ADVERTISE_100;
     control |= ADVERTISE_10;
     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_AUTONEGO_ADVERTISE_REG, control);

     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_1000_ADVERTISE_REG_OFFSET,
                                                                    &control);
     control |= ADVERTISE_1000;
     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_1000_ADVERTISE_REG_OFFSET,
                                                                    control);

     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 0);
     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_COPPER_SPECIFIC_CONTROL_REG,
                                                                    &control);
     control |= (7 << 12);  /* max number of gigabit attempts */
     control |= (1 << 11);  /* enable downshift */
     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_COPPER_SPECIFIC_CONTROL_REG,
                                                                    control);
     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
     control |= IEEE_CTRL_AUTONEGOTIATE_ENABLE;
```

```
     control |= IEEE_STAT_AUTONEGOTIATE_RESTART;

     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_CONTROL_REG_OFFSET, control);

     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
     control |= IEEE_CTRL_RESET_MASK;
     XAxiEthernet_PhyWrite(xaxiemacp, phy_addr, IEEE_CONTROL_REG_OFFSET, control);

     while (1) {
          XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
          if (control & IEEE_CTRL_RESET_MASK)
               continue;
          else
               break;
     }
     xil_printf("Waiting for PHY to complete autonegotiation.\r\n");

     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_STATUS_REG_OFFSET, &status);
     while ( !(status & IEEE_STAT_AUTONEGOTIATE_COMPLETE) ) {
          sleep(1);
          XAxiEthernet_PhyRead(xaxiemacp, phy_addr, IEEE_STATUS_REG_OFFSET,
                                                                    &status);
          }

     xil_printf("autonegotiation complete \r\n");

     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, 0x1f, &partner_capabilities);

     if ( (partner_capabilities & 0x40) == 0x40)/* 1000Mbps */
          return 1000;
     else if ( (partner_capabilities & 0x20) == 0x20)/* 100Mbps */
          return 100;
     else if ( (partner_capabilities & 0x10) == 0x10)/* 10Mbps */
          return 10;
     else
          return 0;
}
```

5) Modify the function "get_IEEE_phy_speed" to add support for KSZ9031.

```
unsigned get_IEEE_phy_speed(XAxiEthernet *xaxiemacp)
{
     u16 phy_identifier;
     u16 phy_model;
     u8 phytype;

#ifdef XPAR_AXIETHERNET_0_BASEADDR
     u32 phy_addr = detect_phy(xaxiemacp);

     /* Get the PHY Identifier and Model number */
     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, PHY_IDENTIFIER_1_REG, &phy_identifier);
     XAxiEthernet_PhyRead(xaxiemacp, phy_addr, PHY_IDENTIFIER_2_REG, &phy_model);

/* Depending upon what manufacturer PHY is connected, a different mask is
 * needed to determine the specific model number of the PHY. */
     if (phy_identifier == MARVEL_PHY_IDENTIFIER) {
          phy_model = phy_model & MARVEL_PHY_MODEL_NUM_MASK;

          if (phy_model == MARVEL_PHY_88E1116R_MODEL) {
               return get_phy_speed_88E1116R(xaxiemacp, phy_addr);
          } else if (phy_model == MARVEL_PHY_88E1111_MODEL) {
               return get_phy_speed_88E1111(xaxiemacp, phy_addr);
          }
     } else if (phy_identifier == TI_PHY_IDENTIFIER) {
          phy_model = phy_model & TI_PHY_DP83867_MODEL;
          phytype = XAxiEthernet_GetPhysicalInterface(xaxiemacp);

          if (phy_model == TI_PHY_DP83867_MODEL && phytype == XAE_PHY_TYPE_SGMII) {
               return get_phy_speed_TI_DP83867_SGMII(xaxiemacp, phy_addr);
          }

          if (phy_model == TI_PHY_DP83867_MODEL) {
               return get_phy_speed_TI_DP83867(xaxiemacp, phy_addr);
          }
     }
     else if(phy_identifier == MICREL_PHY_IDENTIFIER)
     {
          xil_printf("Phy %d is KSZ9031\n\r", phy_addr);
          get_phy_speed_ksz9031(xaxiemacp, phy_addr);
     }
     else {
          LWIP_DEBUGF(NETIF_DEBUG, ("XAxiEthernet get_IEEE_phy_speed: Detected PHY with unknown
identifier/model.\r\n"));
     }
#endif
#ifdef PCM_PMA_CORE_PRESENT
     return get_phy_negotiated_speed(xaxiemacp, phy_addr);
#endif
}
```

6) Modify "xemacpsif_physpeed.c" file to add macro definition

```
#define IEEE_PAGE_ADDRESS_REGISTER              22
#define IEEE_CTRL_1GBPS_LINKSPEED_MASK      0x2040
#define IEEE_CTRL_LINKSPEED_MASK            0x0040
#define IEEE_CTRL_LINKSPEED_1000M           0x0040
#define IEEE_CTRL_LINKSPEED_100M            0x2000
#define IEEE_CTRL_LINKSPEED_10M             0x0000
#define IEEE_CTRL_RESET_MASK                0x8000

#define IEEE_SPEED_MASK     0xC000
#define IEEE_SPEED_1000     0x8000
#define IEEE_SPEED_100      0x4000

#define IEEE_CTRL_RESET_MASK                0x8000
#define IEEE_CTRL_AUTONEGOTIATE_ENABLE      0x1000
#define IEEE_STAT_AUTONEGOTIATE_COMPLETE    0x0020
#define IEEE_STAT_AUTONEGOTIATE_RESTART     0x0200
#define IEEE_RGMII_TXRX_CLOCK_DELAYED_MASK  0x0030
#define IEEE_ASYMMETRIC_PAUSE_MASK          0x0800
#define IEEE_PAUSE_MASK                     0x0400
#define IEEE_AUTONEG_ERROR_MASK             0x8000

#define PHY_DETECT_REG                          1
#define PHY_IDENTIFIER_1_REG                    2
#define PHY_IDENTIFIER_2_REG                    3
#define PHY_DETECT_MASK                     0x1808
#define PHY_MARVELL_IDENTIFIER              0x0141
#define PHY_TI_IDENTIFIER                   0x2000
#define PHY_XILINX_PCS_PMA_ID1          0x0174
#define PHY_XILINX_PCS_PMA_ID2          0x0C00

#define MICREL_PHY_IDENTIFIER                   0x22
#define MICREL_PHY_KSZ9031_MODEL                0x220

#define XEMACPS_GMII2RGMII_SPEED1000_FD     0x140
#define XEMACPS_GMII2RGMII_SPEED100_FD      0x2100
#define XEMACPS_GMII2RGMII_SPEED10_FD       0x100
```

7) Add phy speed acquisition function

```
static u32_t get_phy_speed_ksz9031(XEmacPs *xemacpsp, u32_t phy_addr)
{
     u16_t temp;
     u16_t control;
     u16_t status;
     u16_t status_speed;
     u32_t timeout_counter = 0;
     u32_t temp_speed;
     u32_t phyregtemp;

     xil_printf("Start PHY autonegotiation \r\n");

     XEmacPs_PhyWrite(xemacpsp,phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 2);
     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_CONTROL_REG_MAC, &control);
     control |= IEEE_RGMII_TXRX_CLOCK_DELAYED_MASK;
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_CONTROL_REG_MAC, control);

     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 0);

     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_AUTONEGO_ADVERTISE_REG, &control);
     control |= IEEE_ASYMMETRIC_PAUSE_MASK;
     control |= IEEE_PAUSE_MASK;
     control |= ADVERTISE_100;
     control |= ADVERTISE_10;
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_AUTONEGO_ADVERTISE_REG, control);
```

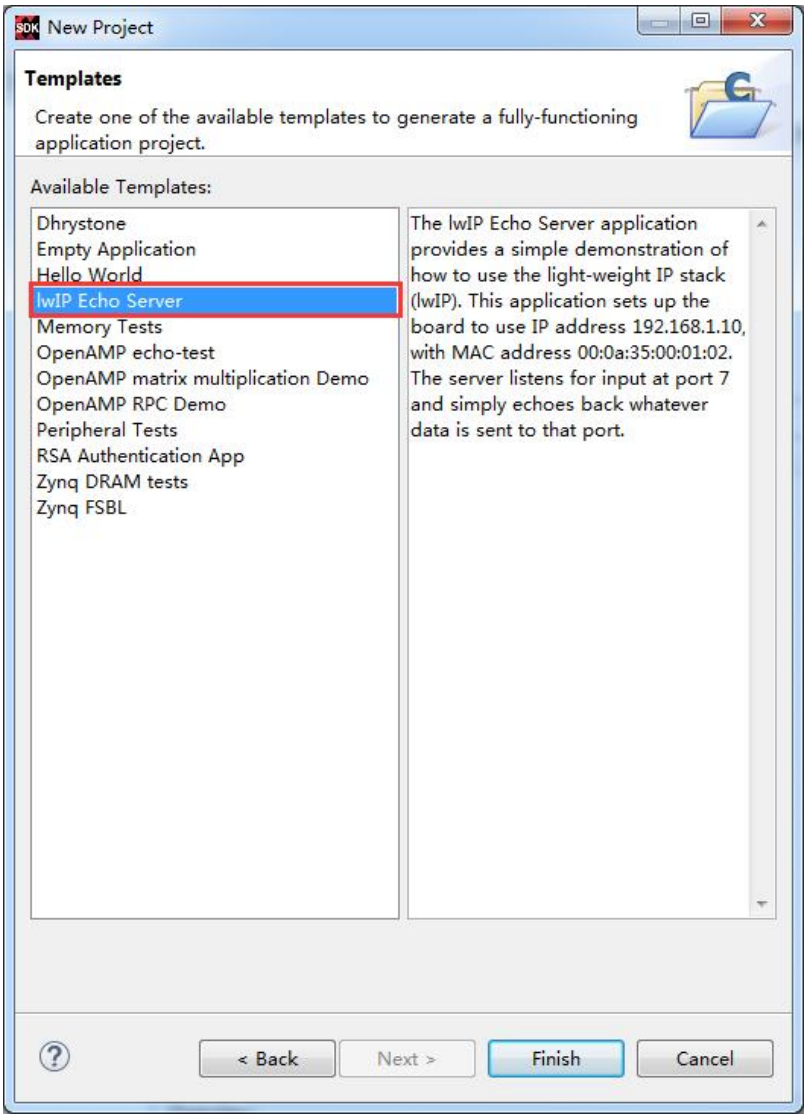
```
     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_1000_ADVERTISE_REG_OFFSET,
                     &control);
     control |= ADVERTISE_1000;
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_1000_ADVERTISE_REG_OFFSET,
                     control);

     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_PAGE_ADDRESS_REGISTER, 0);
     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_COPPER_SPECIFIC_CONTROL_REG,
                                                                  &control);
     control |= (7 << 12);  /* max number of gigabit attempts */
     control |= (1 << 11);  /* enable downshift */
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_COPPER_SPECIFIC_CONTROL_REG,
                                                                  control);
     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
     control |= IEEE_CTRL_AUTONEGOTIATE_ENABLE;
     control |= IEEE_STAT_AUTONEGOTIATE_RESTART;
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_CONTROL_REG_OFFSET, control);

     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
     control |= IEEE_CTRL_RESET_MASK;
     XEmacPs_PhyWrite(xemacpsp, phy_addr, IEEE_CONTROL_REG_OFFSET, control);

     while (1) {
          XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_CONTROL_REG_OFFSET, &control);
          if (control & IEEE_CTRL_RESET_MASK)
               continue;
          else
               break;
     }

     XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_STATUS_REG_OFFSET, &status);

     xil_printf("Waiting for PHY to complete autonegotiation.\r\n");

     while ( !(status & IEEE_STAT_AUTONEGOTIATE_COMPLETE) ) {
          sleep(1);
          XEmacPs_PhyRead(xemacpsp, phy_addr,
                          IEEE_COPPER_SPECIFIC_STATUS_REG_2,  &temp);
          timeout_counter++;

          if (timeout_counter == 30) {
               xil_printf("Auto negotiation error \r\n");
               return;
          }
          XEmacPs_PhyRead(xemacpsp, phy_addr, IEEE_STATUS_REG_OFFSET, &status);
     }
     xil_printf("autonegotiation complete \r\n");

     XEmacPs_PhyRead(xemacpsp, phy_addr,0x1f,
                     &status_speed);

     if ( (status_speed & 0x40) == 0x40)/* 1000Mbps */
          return 1000;
     else if ( (status_speed & 0x20) == 0x20)/* 100Mbps */
          return 100;
     else if ( (status_speed & 0x10) == 0x10)/* 10Mbps */
          return 10;
     else
          return 0;
     return XST_SUCCESS;
}
```

8) Modify function "get_IEEE_phy_speed", add support for KSZ9031

```
static u32_t get_IEEE_phy_speed(XEmacPs *xemacpsp, u32_t phy_addr)
{
     u16_t phy_identity;
     u32_t RetStatus;

     XEmacPs_PhyRead(xemacpsp, phy_addr, PHY_IDENTIFIER_1_REG,
                     &phy_identity);

     if(phy_identity == MICREL_PHY_IDENTIFIER)
     {
          RetStatus = get_phy_speed_ksz9031(xemacpsp, phy_addr);
     }
     else if (phy_identity == PHY_TI_IDENTIFIER) {
          RetStatus = get_TI_phy_speed(xemacpsp, phy_addr);
     } else {
          RetStatus = get_Marvell_phy_speed(xemacpsp, phy_addr);
     }
```

```
     return RetStatus;
}
```

Part 7.2.2: Create an APP based on the LWIP template under the SDK

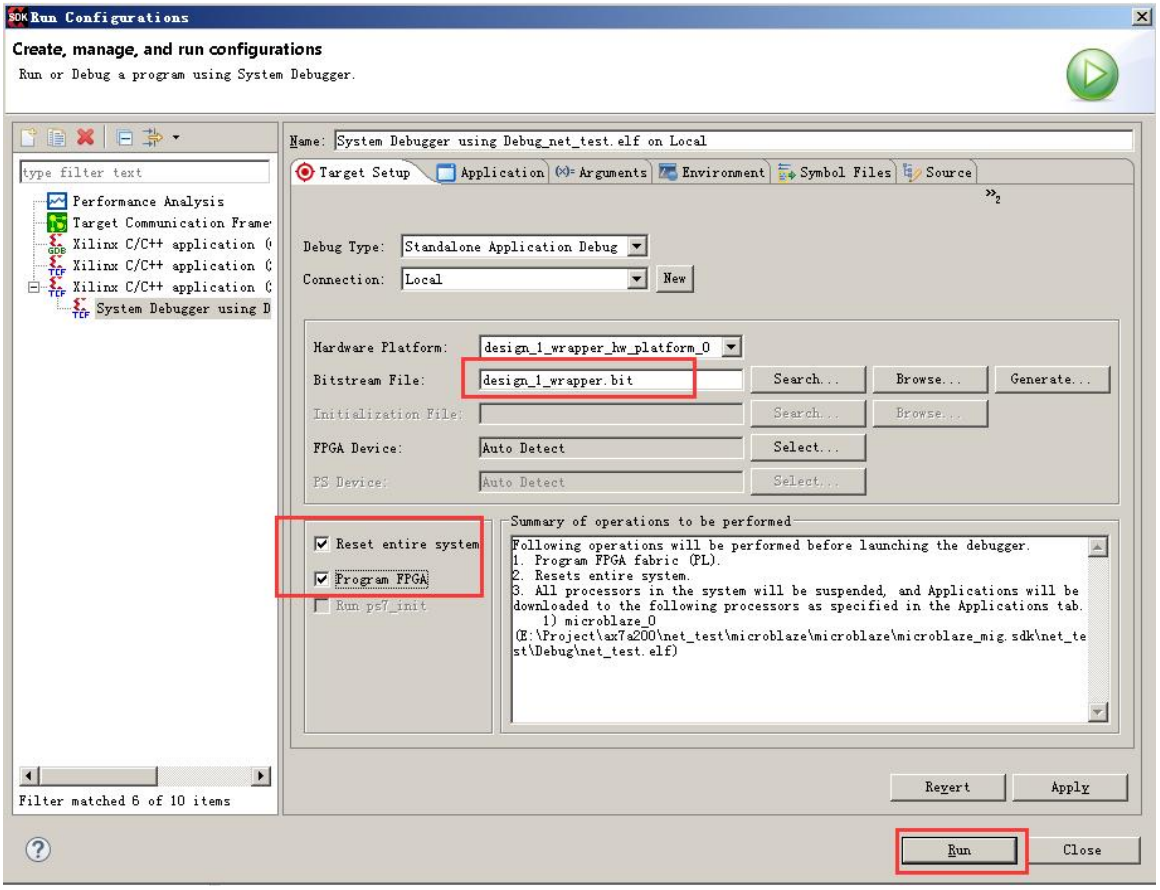


7.3: Download and debug

The test environment requires a router that supports dhcp. The FPGA development board can automatically obtain an IP address by connecting the router with a network cable. The experiment host and the FPGA development board are on the same network and can communicate with each other.

Part 7.3.1: Ethernet test

1) Connect the serial port to open the serial debugging terminal, connect the Ethernet cable to the router (**ETH2**)
2) Run **SDK**



3) You can see that the serial port prints out some information, you can see that the automatically obtained address is "**192.168.1.100**", the connection speed is **1000Mbps**, and the **tcp** port is **7**.

```
-----lwIP TCP echo server ------
TCP packets sent to port 6001 will be echoed back
WARNING: Not a Marvell or TI Ethernet PHY. Please verify the initialization sequence
Phy 1 is KSZ9031
Start PHY autonegotiation
Waiting for PHY to complete autonegotiation.
autonegotiation complete
auto-negotiated link speed: 1000
Board IP: 192.168.1.100
Netmask : 255.255.255.0
Gateway : 192.168.1.1
TCP echo server started @ port 7
```

4) At this time, if the computer is also connected to the same router, you can ping the IP address of the FPGA development board

Part 8: Custom IP Experiment

The experimental Vivado project is "**custom_pwm_ip**".

Xilinx officially provides a lot of IP cores for everyone. You can view these IP cores in the IP Catalog of Vivado. It is impossible for users to use Xilinx official free IP cores when building their own systems. They need to create their own user IP cores. There are many advantages to creating your own IP core, such as system design customization; Design reuse, you can add a license to the IP core and provide it to others for a fee; Simplify system design and shorten design time. When designing IP core with Microblaze system, the most commonly used is to use AXI bus to connect Microblaze with the IP core of PL part. This experiment will introduce how to build an AXI bus type IP core in Vivado, this IP core is used to generate a PWM, and use this to control the LED on the FPGA development board to make a breathing light effect.

Part 8.1: Introduction to PWM

We often use PWM to control LEDs, buzzers, etc., and adjust the brightness of LEDs by adjusting the duty cycle of pulses. A pwm module we have used in other FPGA development boards is as follows:

```
//////////////////////////////////////////////////////////////////////////////////
//                                                                              //
//                                                                              //
//  Author: meisq                                                               //
//          msq@qq.com                                                          //
//          ALINX(shanghai) Technology Co.,Ltd                                  //
//          heijin                                                              //
//    WEB: http://www.alinx.cn/                                                 //
//    BBS: http://www.heijin.org/                                               //
//                                                                              //
//////////////////////////////////////////////////////////////////////////////////
//                                                                              //
// Copyright (c) 2017,ALINX(shanghai) Technology Co.,Ltd                        //
//                    All rights reserved                                       //
//                                                                              //
// This source file may be used and distributed without restriction provided    //
// that this copyright statement is not removed from the file and that any      //
// derivative work contains the original copyright notice and the associated    //
// disclaimer.                                                                  //
//                                                                              //
//////////////////////////////////////////////////////////////////////////////////

//================================================================================
//  Description:  pwm model
//  pwm out period = frequency(pwm_out) * (2 ** N) / frequency(clk);
//
//================================================================================
//  Revision History:
//  Date         By          Revision    Change Description
//--------------------------------------------------------------------------------
//  2017/5/3    meisq        1.0         Original
//*******************************************************************************/
`timescale 1ns / 1ps
module ax_pwm
#(
  parameter N = 32 //pwm bit width
)
(
    input        clk,
    input        rst,
    input[N - 1:0]period,
    input[N - 1:0]duty,
    output       pwm_out
    );

reg[N - 1:0] period_r;
reg[N - 1:0] duty_r;
reg[N - 1:0] period_cnt;
```

```
reg pwm_r;
assign pwm_out = pwm_r;
always@(posedge clk or posedge rst)
begin
    if(rst==1)
    begin
        period_r <= { N {1'b0} };
        duty_r <= { N {1'b0} };
    end
    else
    begin
        period_r <= period;
        duty_r  <= duty;
    end
end

always@(posedge clk or posedge rst)
begin
    if(rst==1)
        period_cnt <= { N {1'b0} };
    else
        period_cnt <= period_cnt + period_r;
end

always@(posedge clk or posedge rst)
begin
    if(rst==1)
    begin
        pwm_r <= 1'b0;
    end
    else
    begin
        if(period_cnt >= duty_r)
            pwm_r <= 1'b1;
        else
            pwm_r <= 1'b0;
    end
end

endmodule
```

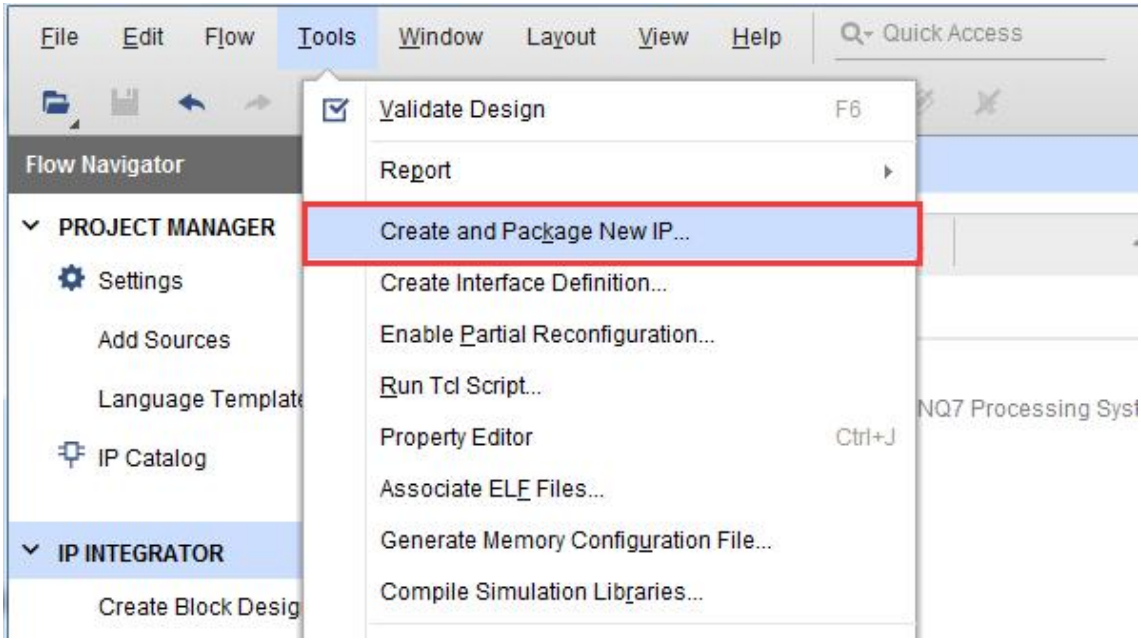
It can be seen that this PWM module needs two parameters "period" and "duty" to control the frequency and duty cycle. We need to design some registers to control these parameters. Here we need to use the AXI bus, and the PS uses the AXI bus to read and write registers.

Part 8.2: Create a Vivado Project

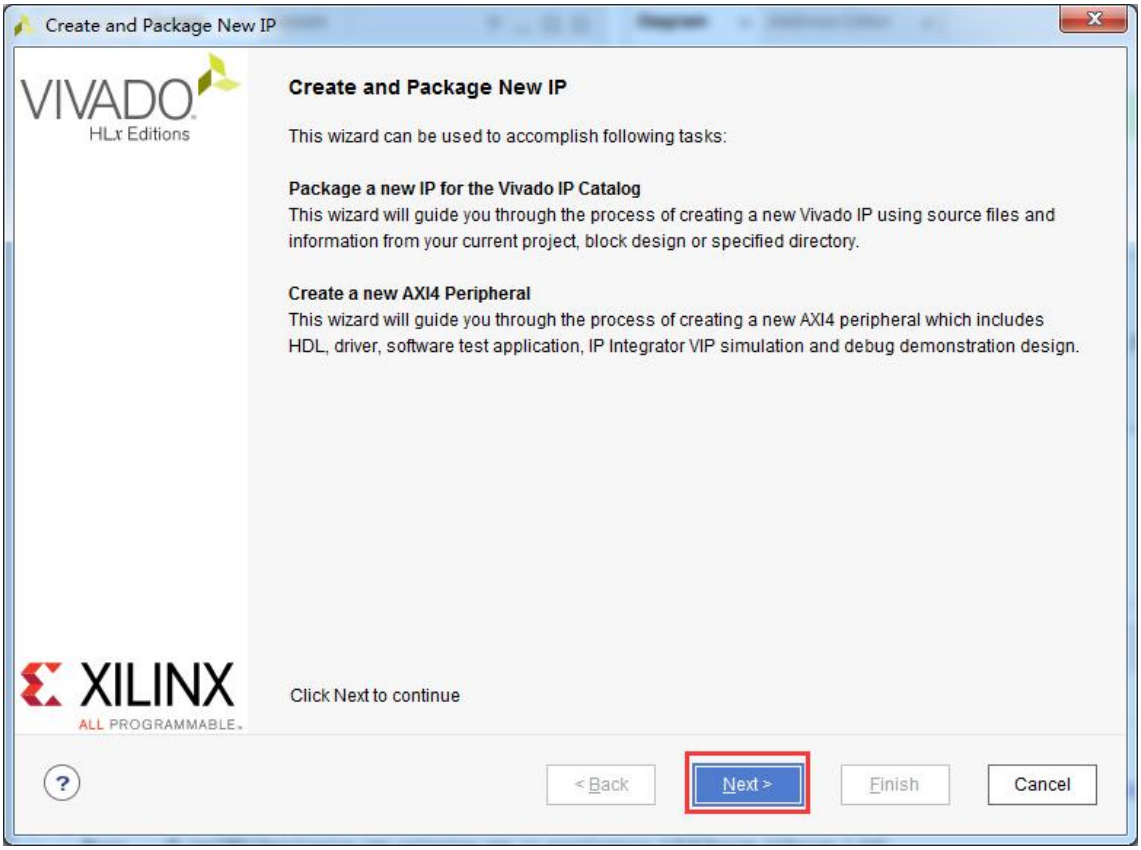
In order to avoid duplication of work, we continue to experiment on the basis of the Ethernet Vivado project in Part 7. Just delete the SDK directory.

Part 8.2.1: Create a custom IP

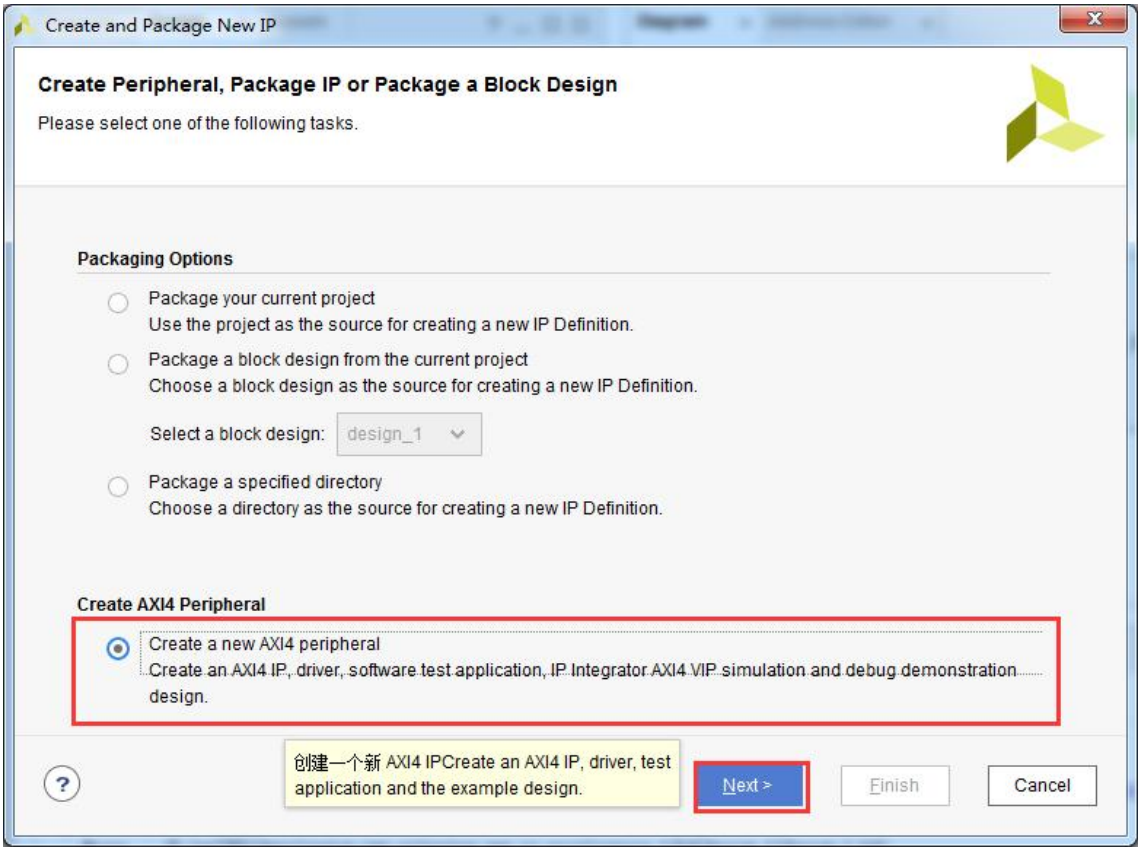
1) Open the vivado project, click the menu "Tools->Create and Package IP..."



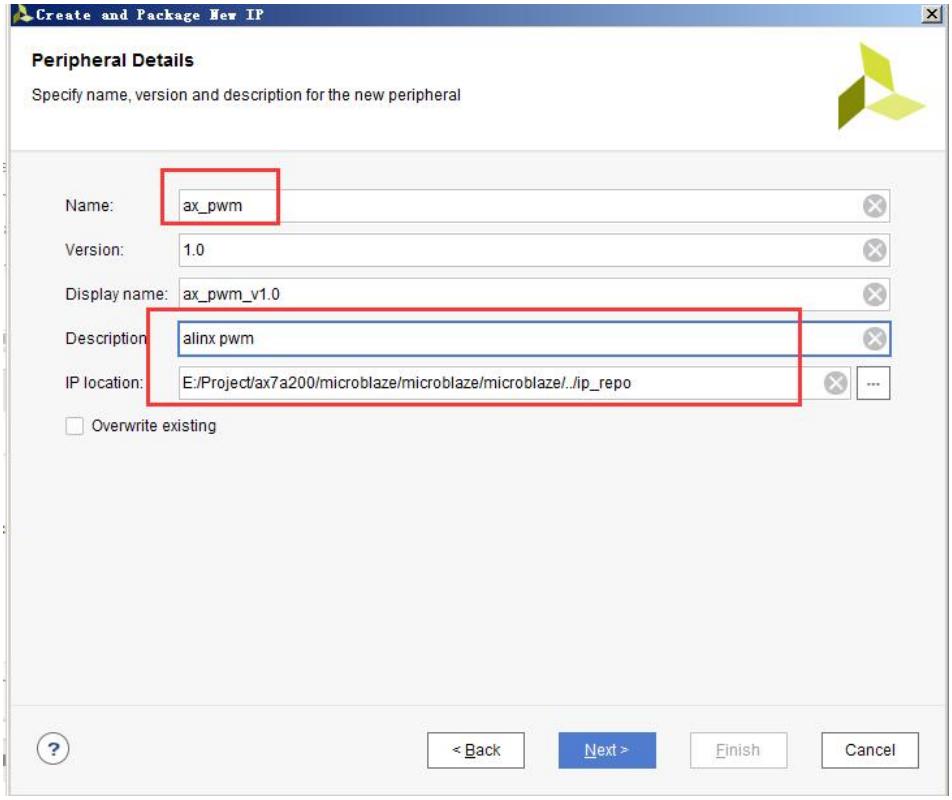
2) Select "Next"



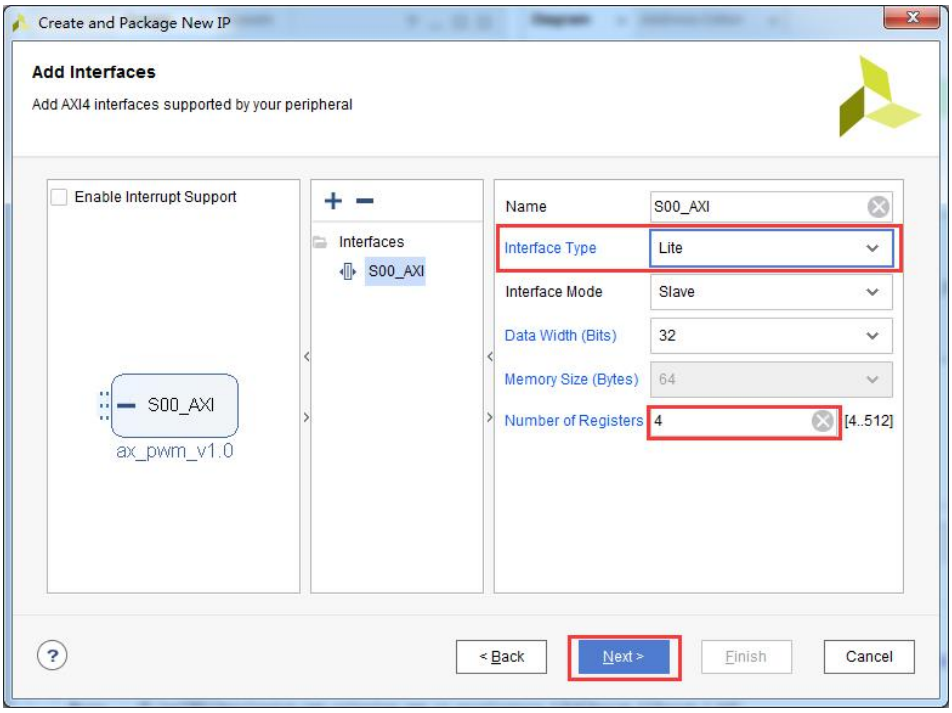
3) Choose to create a new **AXI4** device



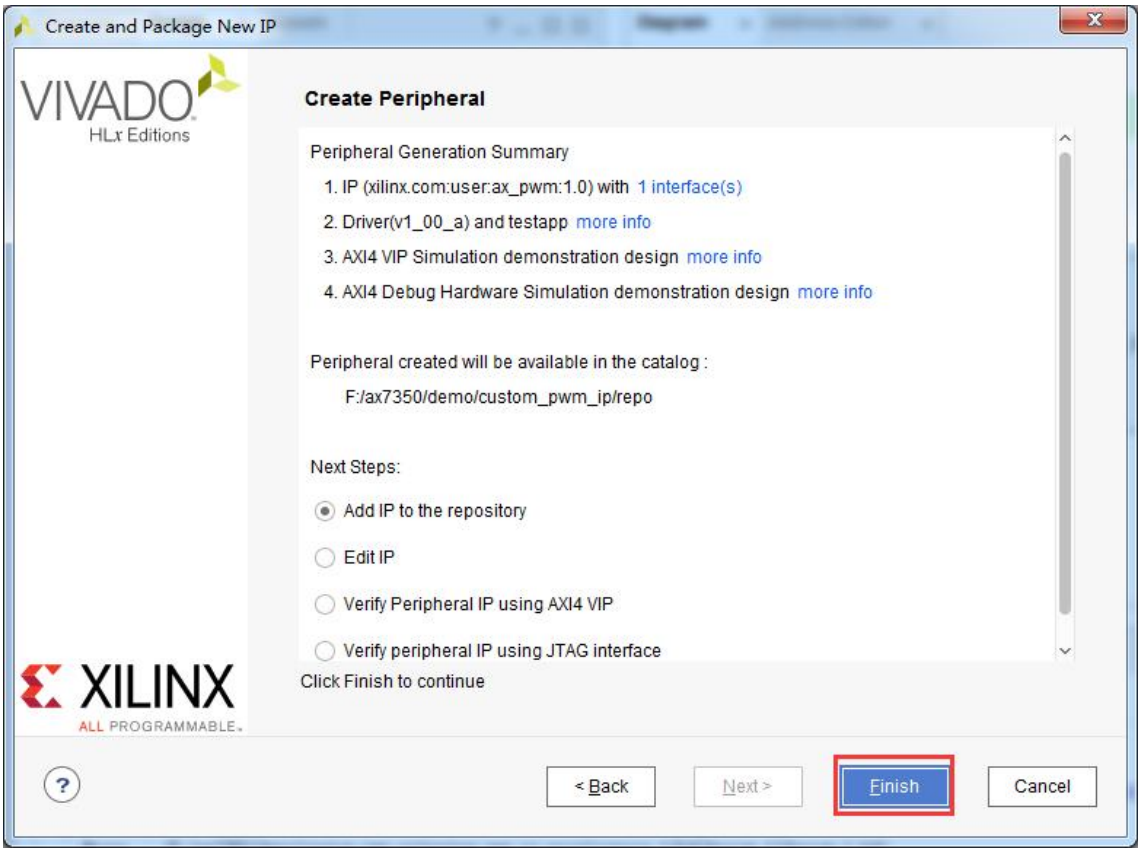
4) Fill in "**ax_pwm**" in the name, fill in "**alinx pwm**" in the description, and then choose a suitable location to put the IP



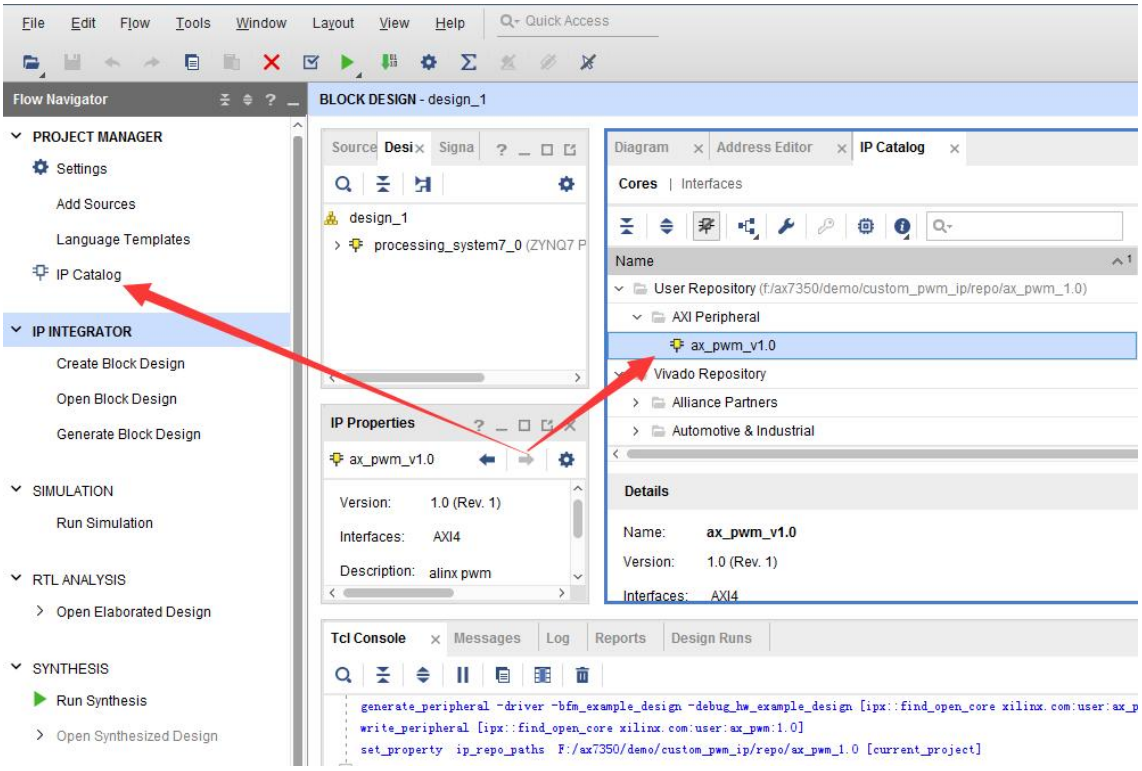
5) The following parameters can specify the interface type, the number of registers, etc. There is no need to modify it here, using the **AXI Lite Slave** interface with **4** registers.



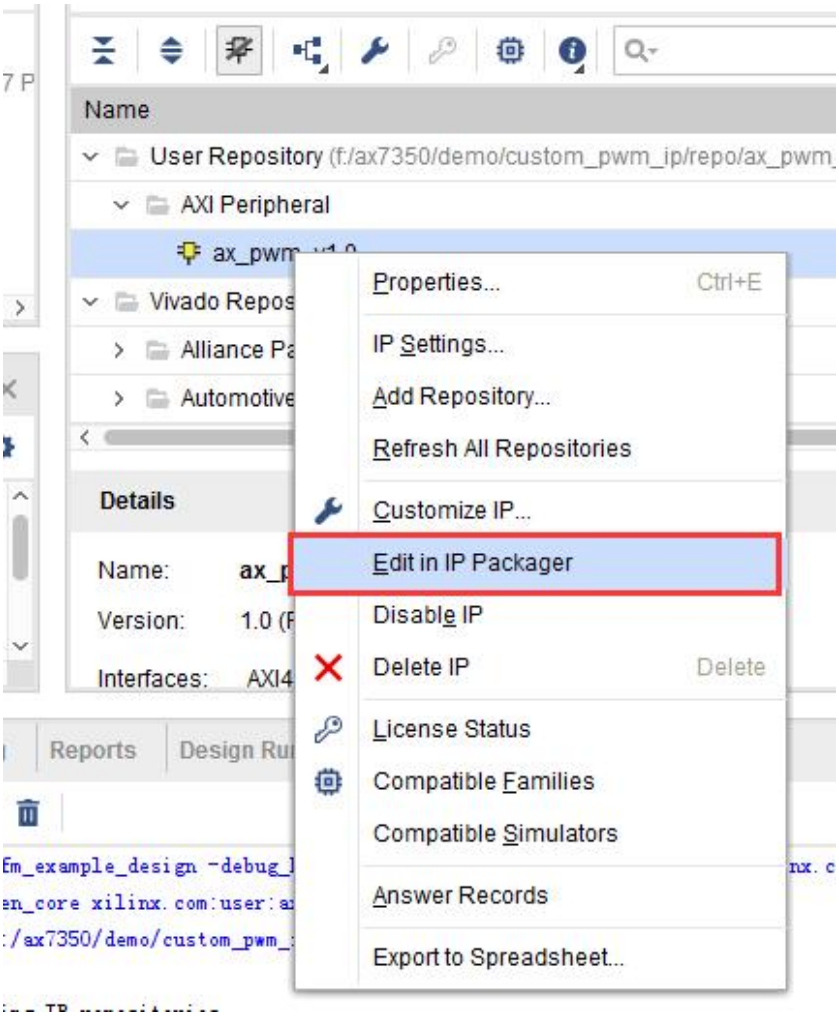
6) Click "**Finish**" to complete the creation of the **IP**



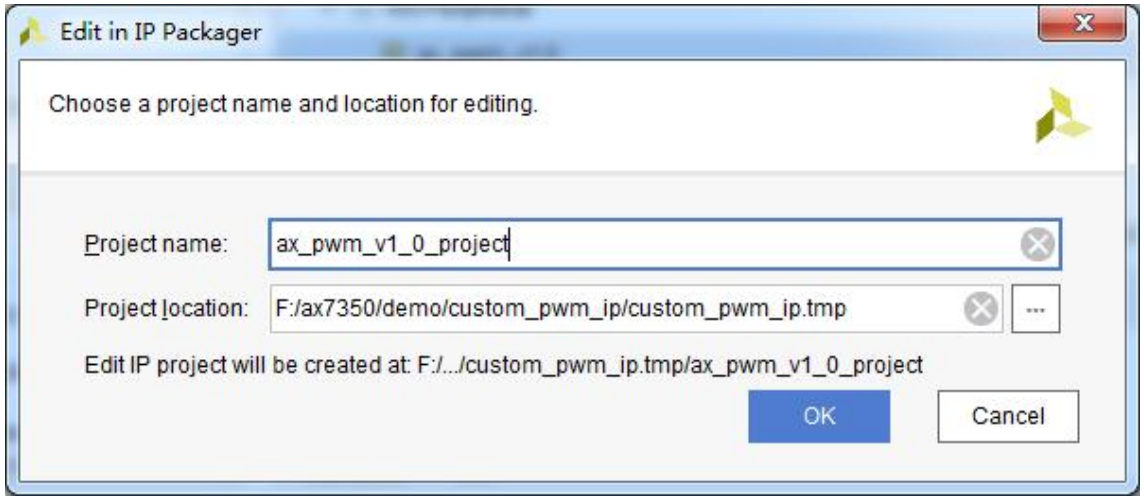
7) You can see the **IP** just created in the "**IP Catalog**"



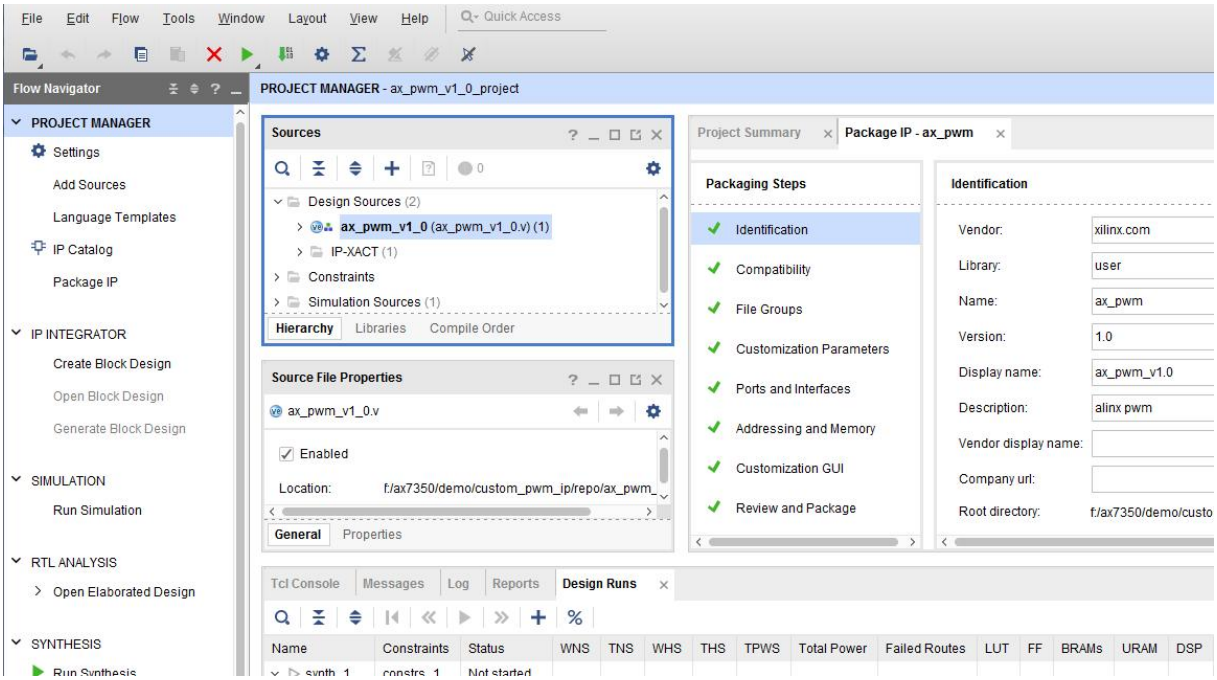
8) The IP at this time has only simple register read and write functions, we need to modify the IP, select IP, and right click "Edit in IP Packager"



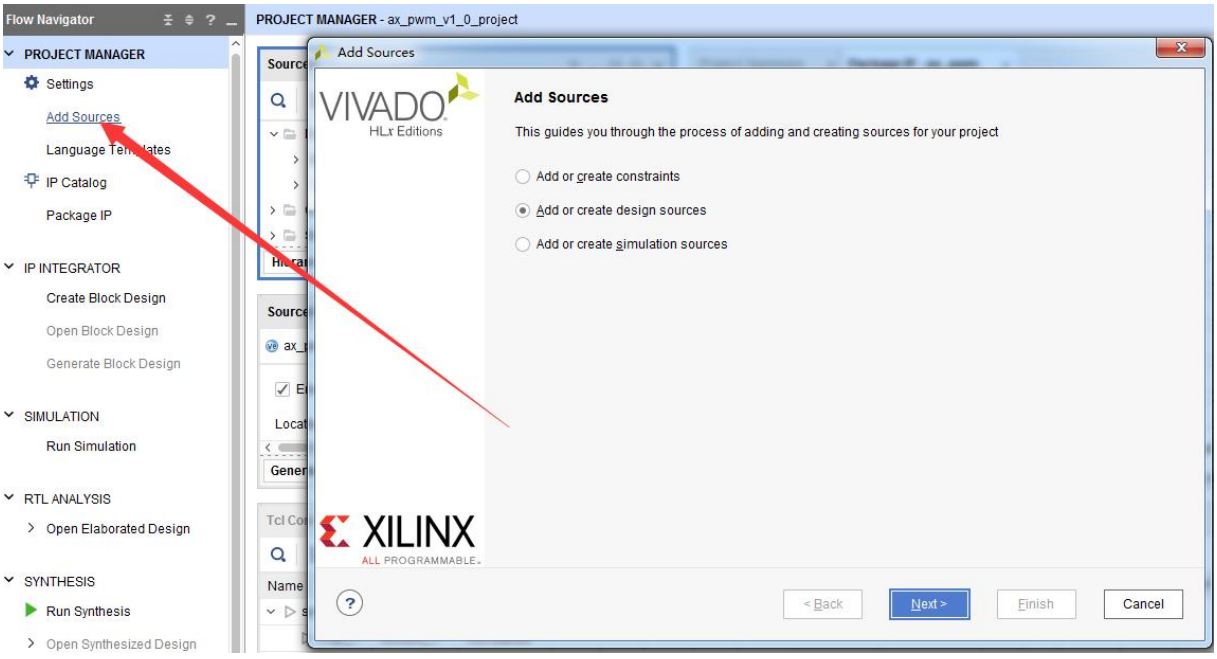
9) Then a dialog box pops up, you can fill in the project name and path, here is the default, click "OK"



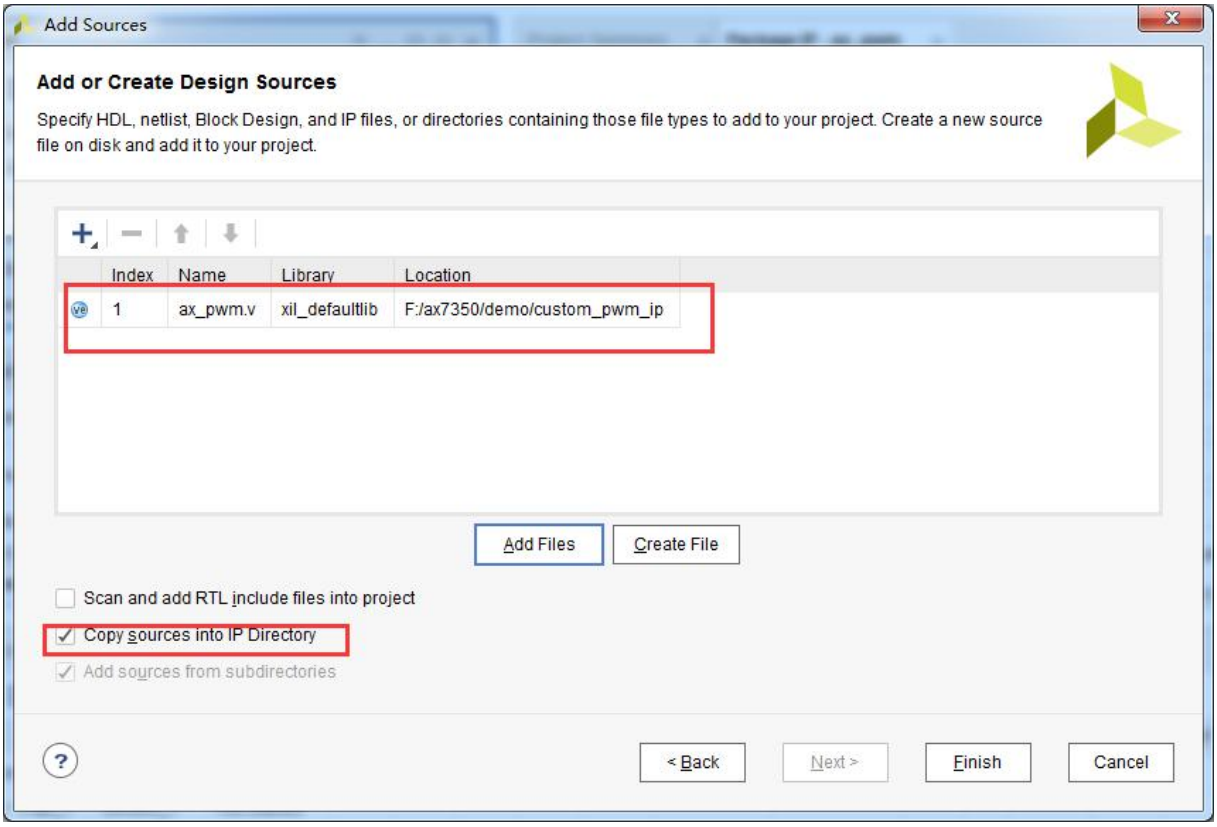
10)Vivado opened a new project



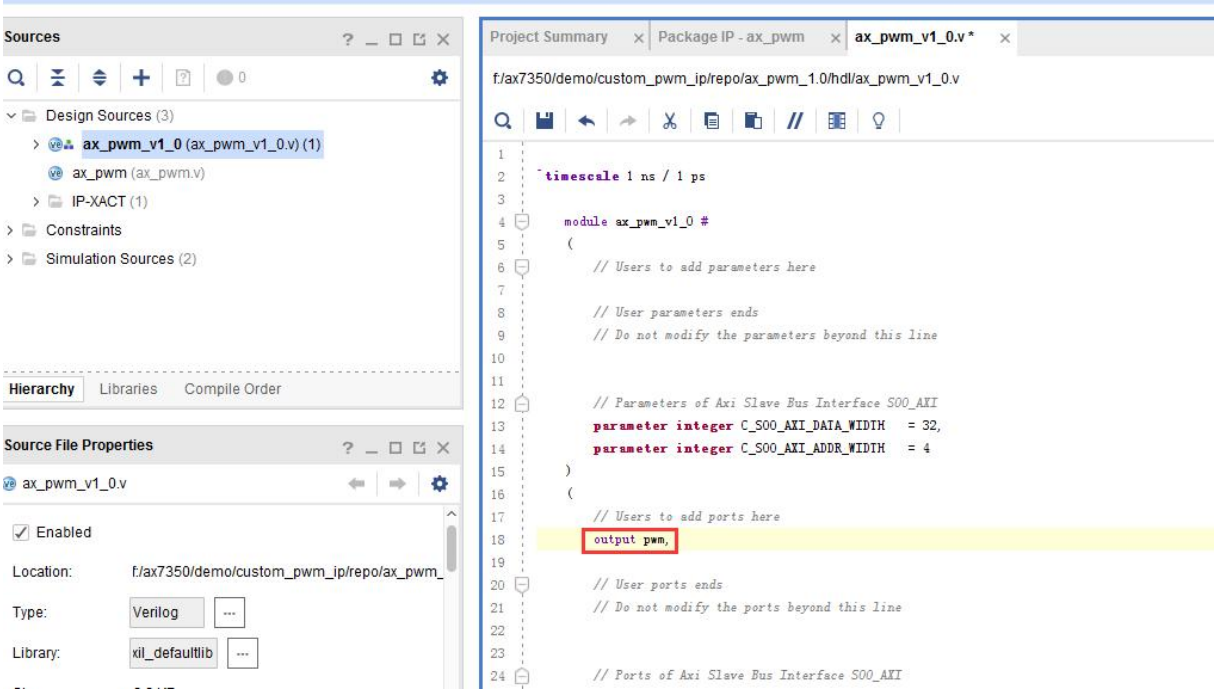
11)Add the core code of the **PWM** function



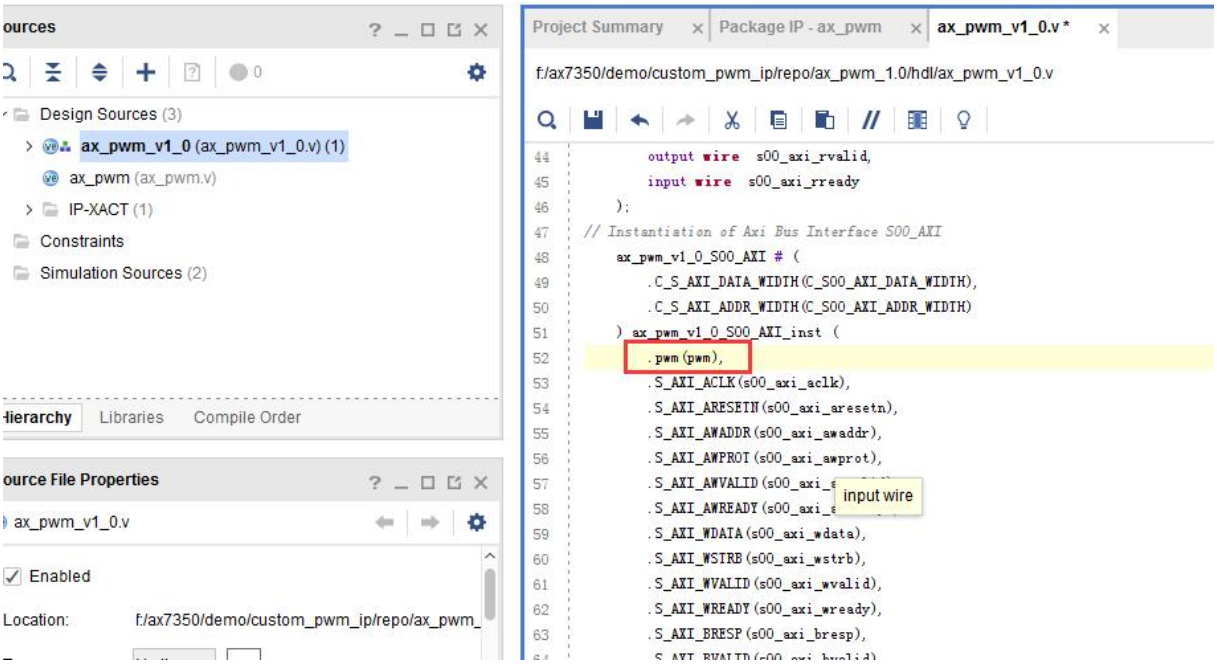
12)When adding the code, choose to copy the code to the IP directory.



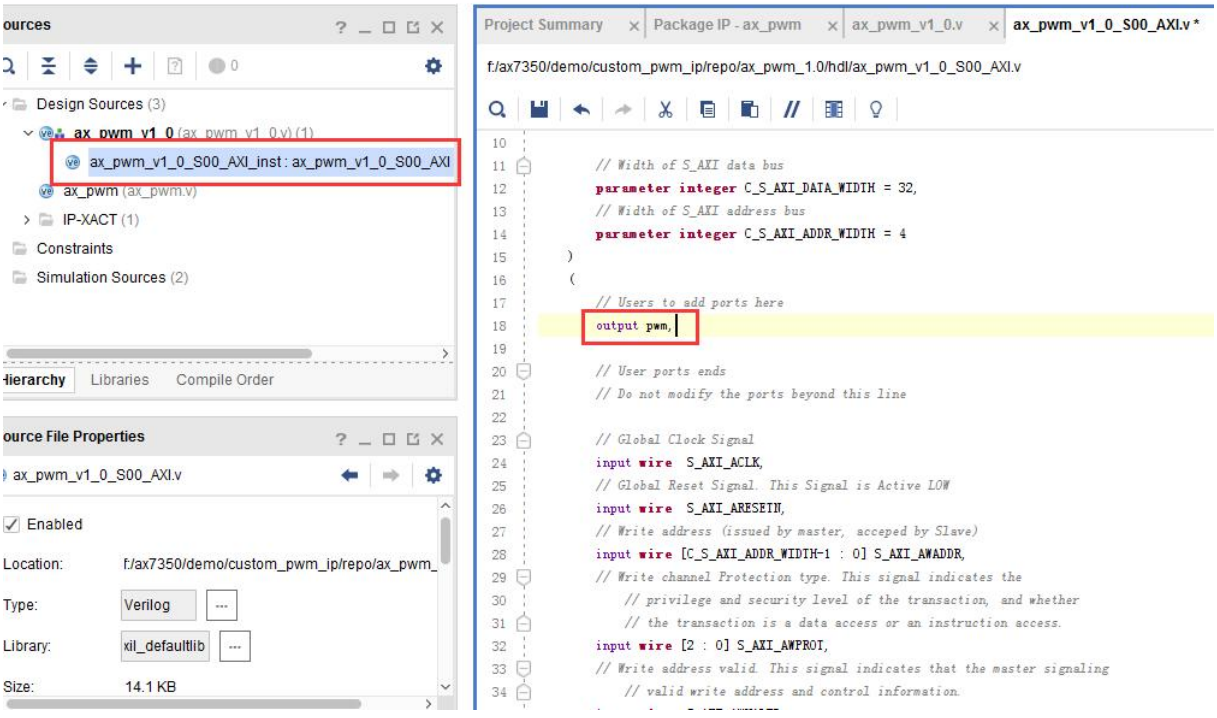
13)Modify "ax_pwm_v1_0.v" and add a pwm output port.



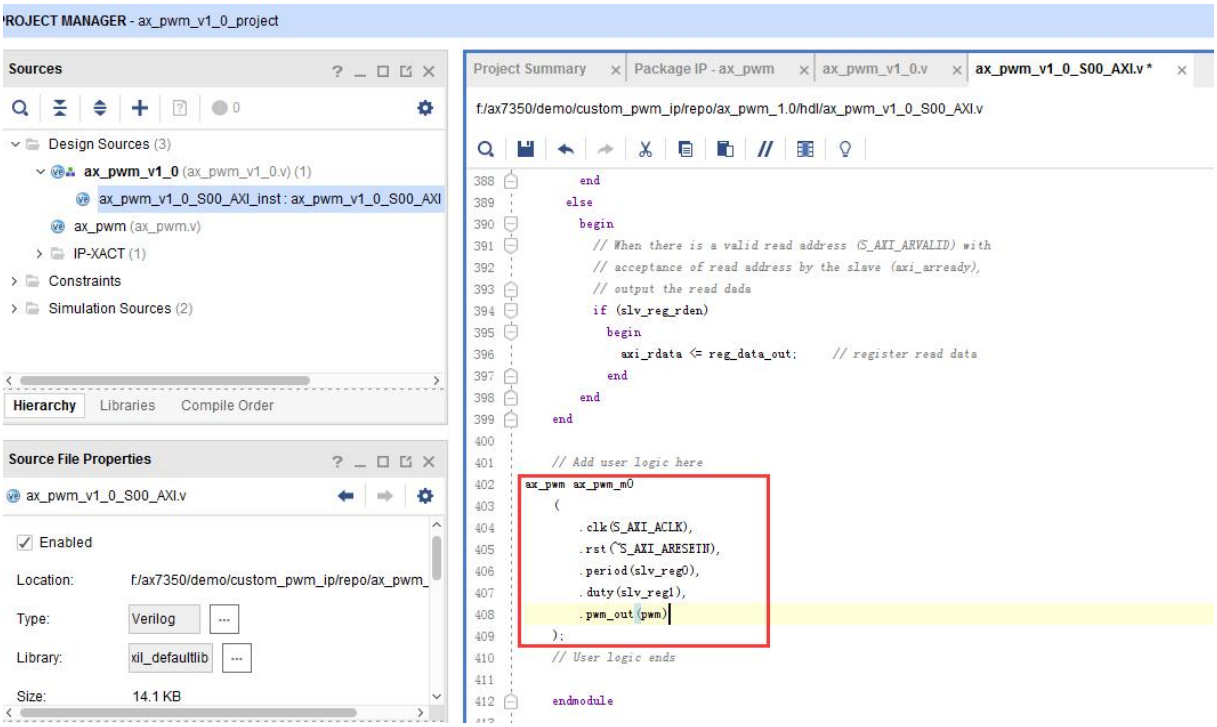
14)Modify "ax_pwm_v1_0.v", add the instantiation of pwm port in the instantiation "ax_pwm_V1_0_S00_AXI"



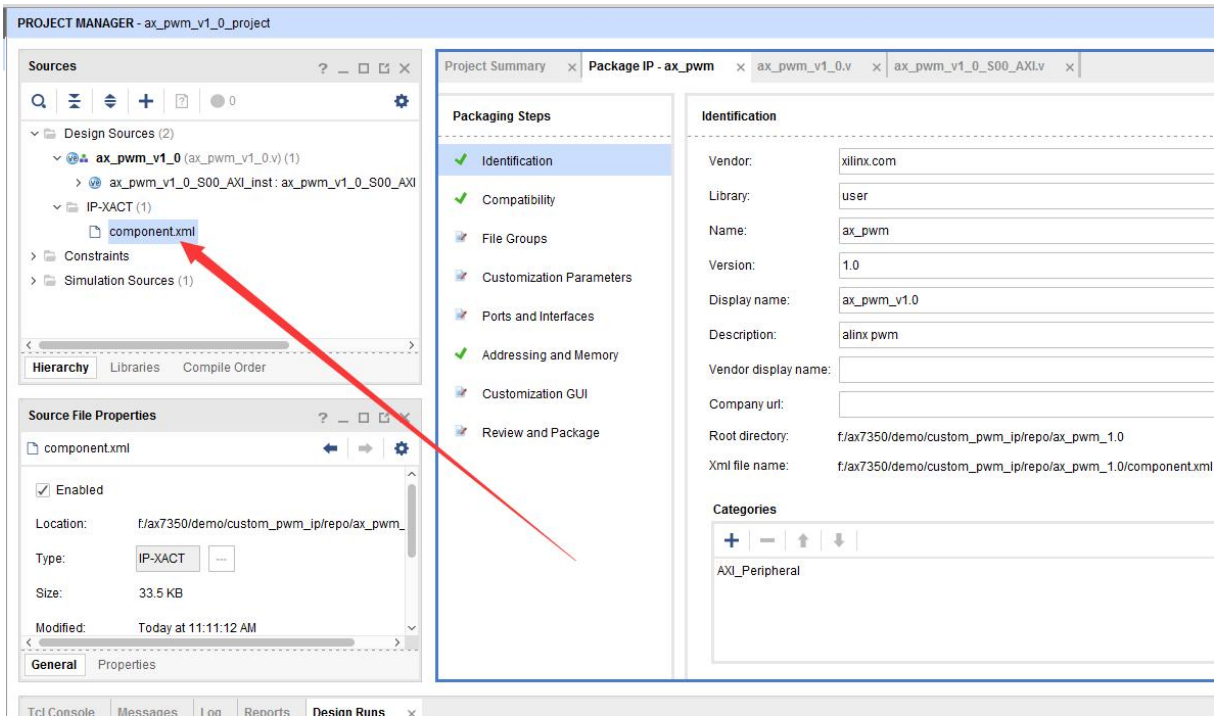
15)Modify the "**ax_pwm_v1_0_s00_AXI.v**" file, Instantiate **pwm** core function code. Add **pwm** port, this file is the core code to realize **AXI4 Lite Slave**



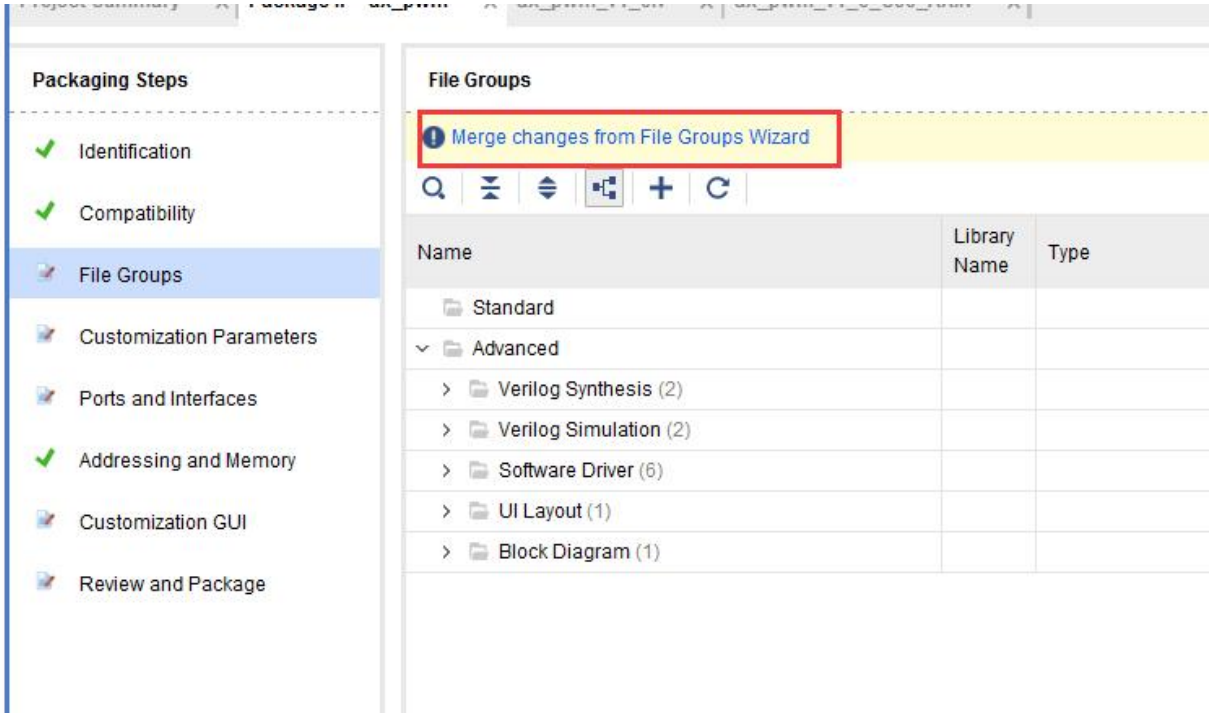
16)Modify the "**ax_pwm_v1_0_s00_AXI.v**" file, Instantiate **pwm** core function code. The registers **slv_reg0** and **slv_reg1** are used for parameter control of the **pwm** module.



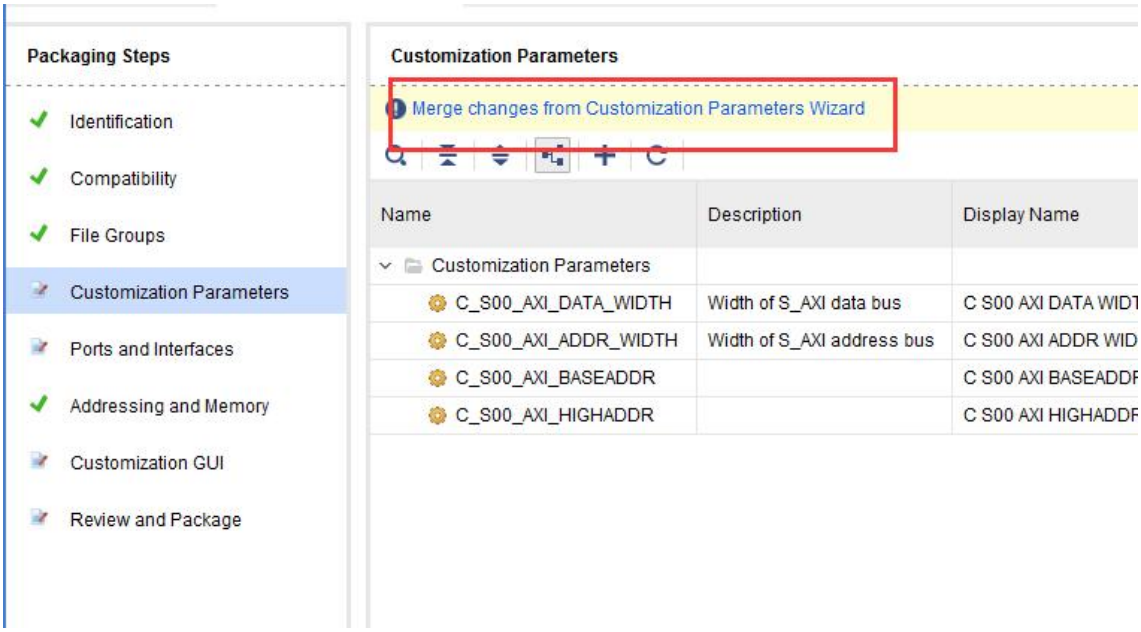
17)Double-click the "**component.xml**" file.



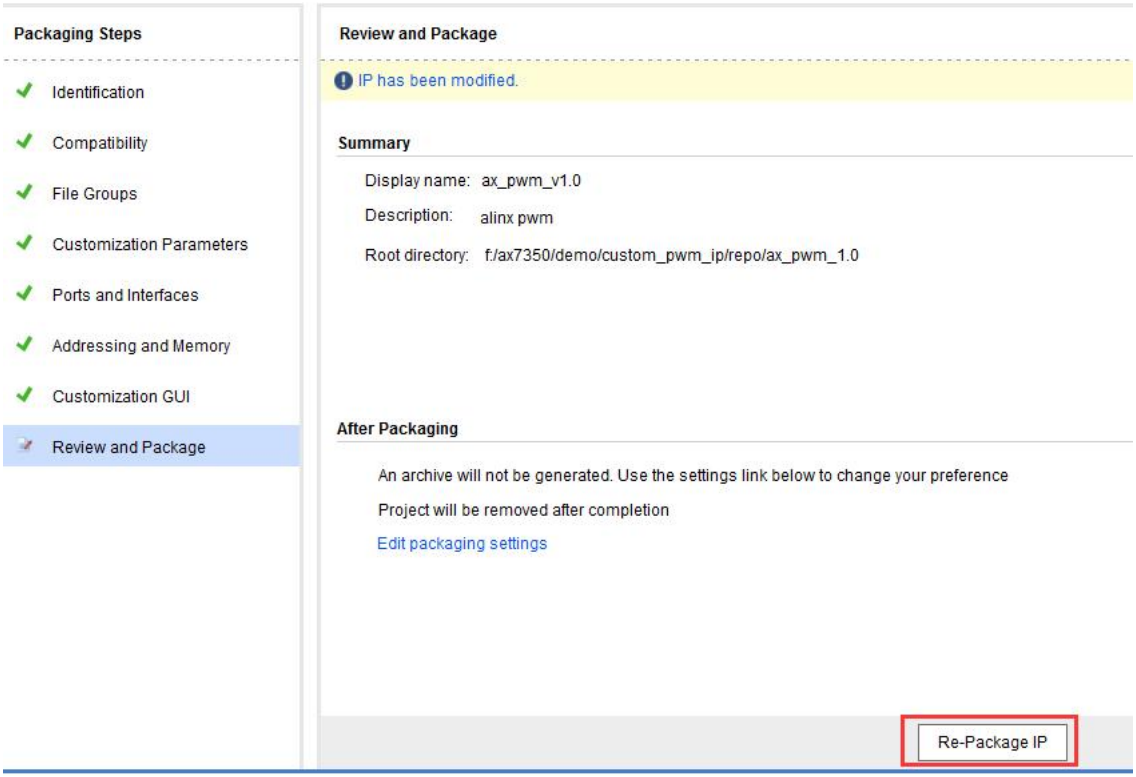
18)Click "**Merge changers from File Groups Wizard**" in the "**File Groups**" option



19)Click "**Merge changes form Customization Parameters Wizard**" in the "**Customization Parameters**" option

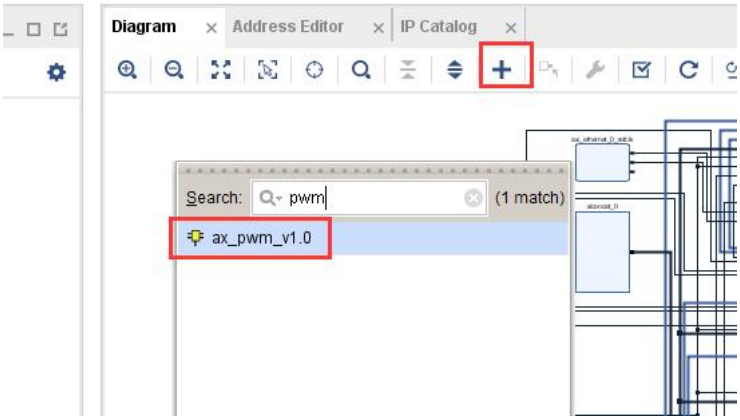


20)Click "**Re-Package IP**" to complete the IP modification

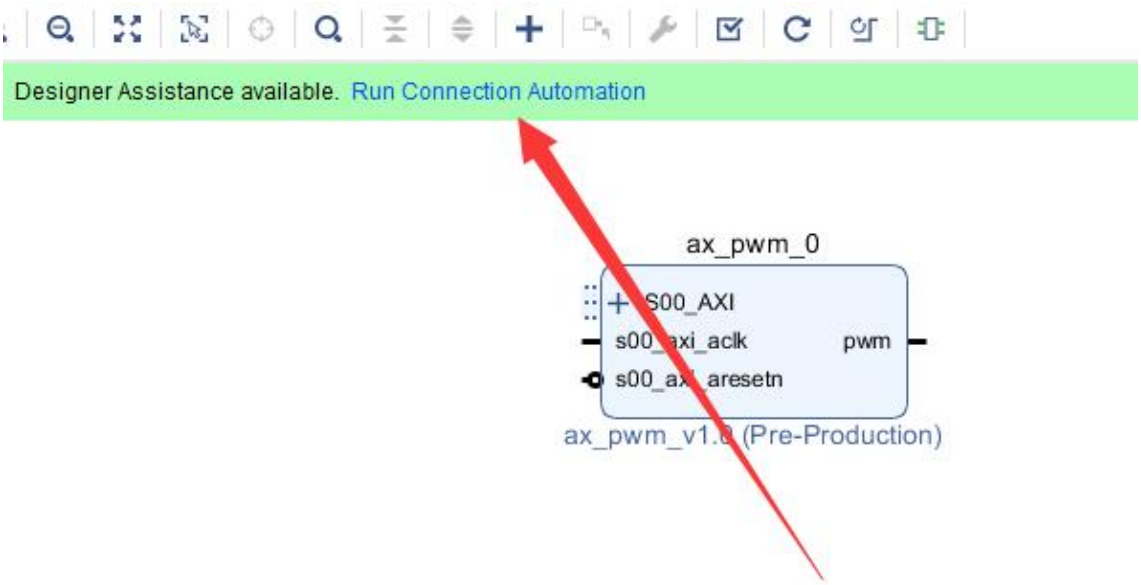


Part 8.2.2: Add custom IP to project

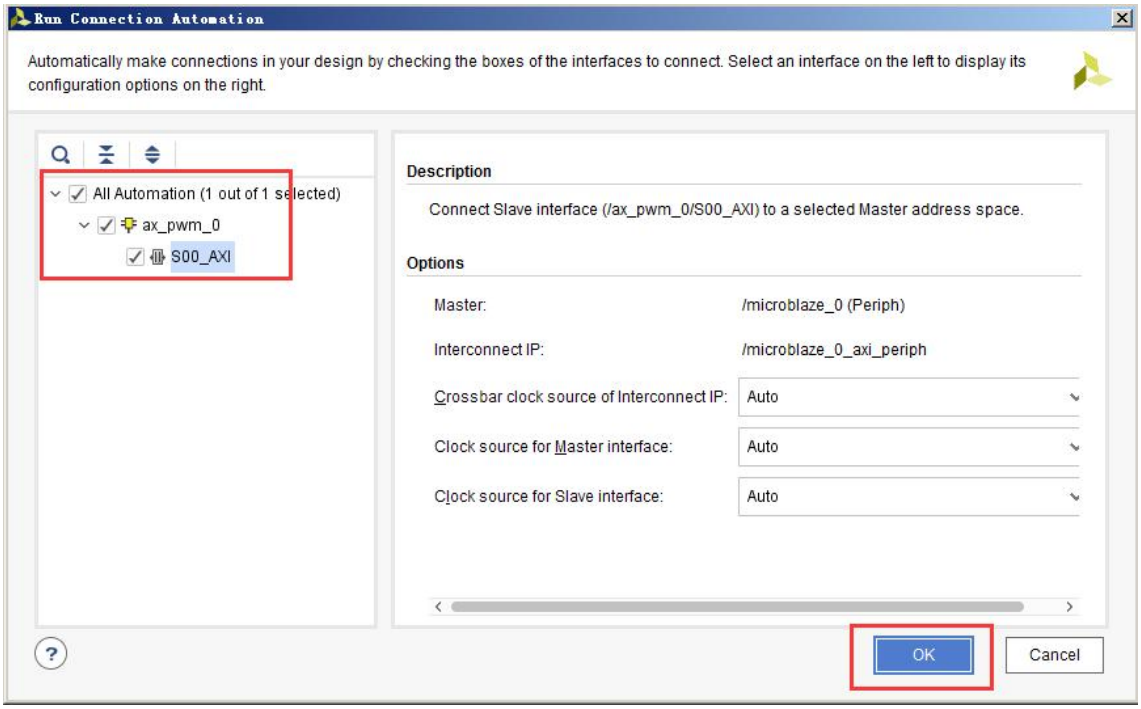
21)Search for "**pwm**" and add "**ax_pwm_v1.0**"



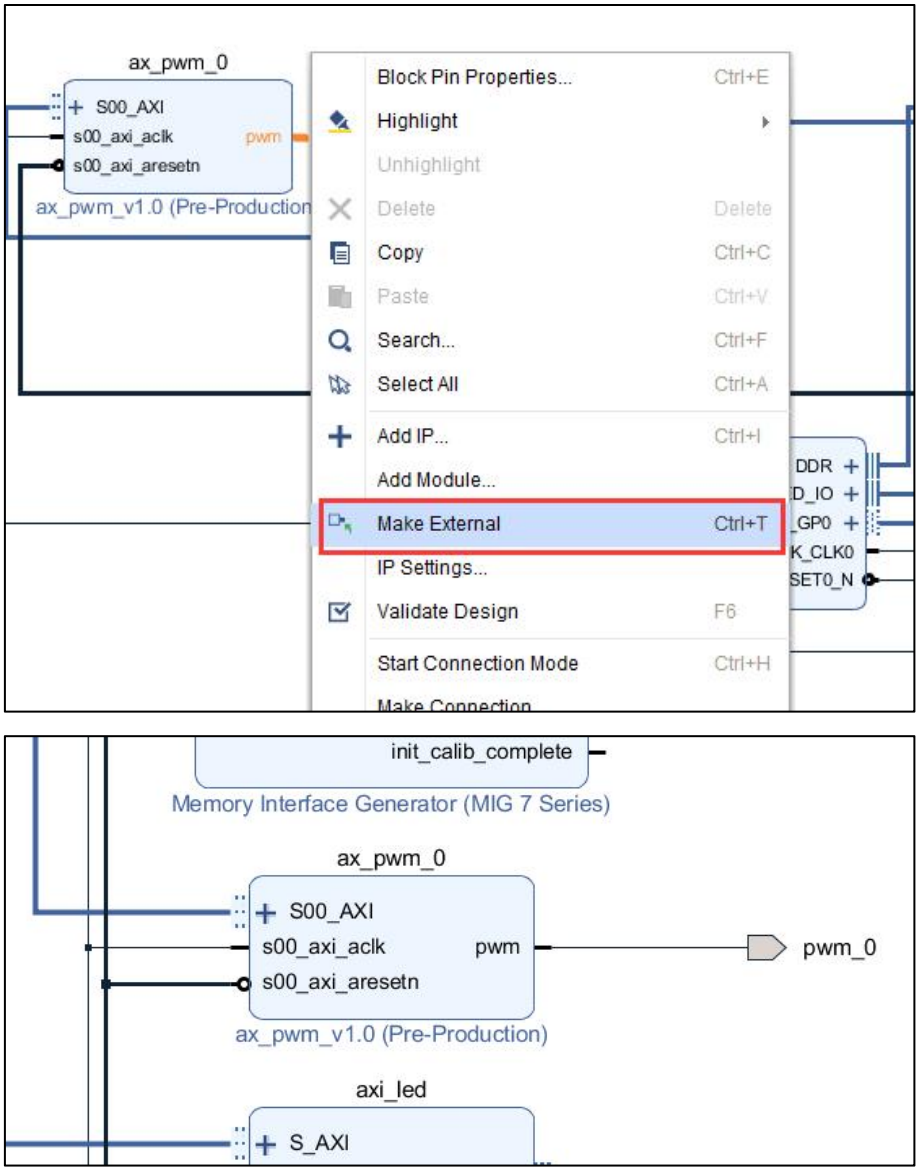
22)Click "**Run Connection Automation**"



23)Default selection, click OK

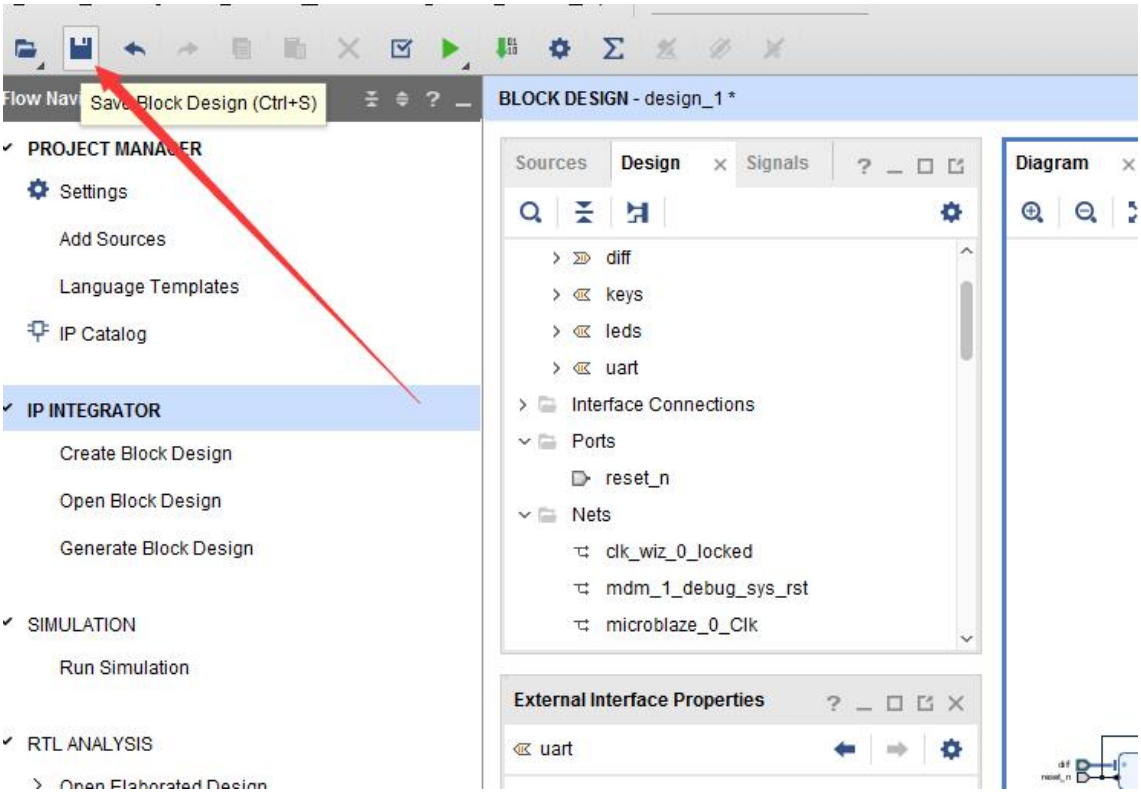


24)Export **pwm** port

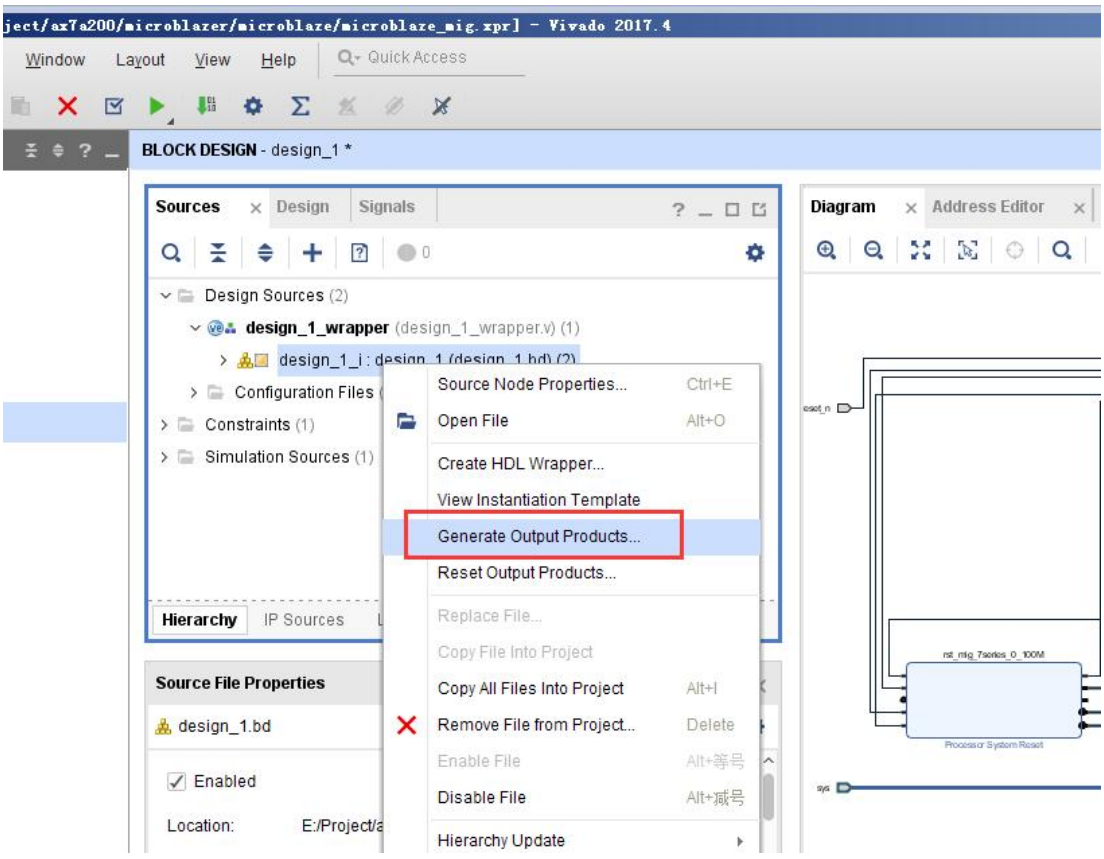


Part 8.2.3: Save and check for errors

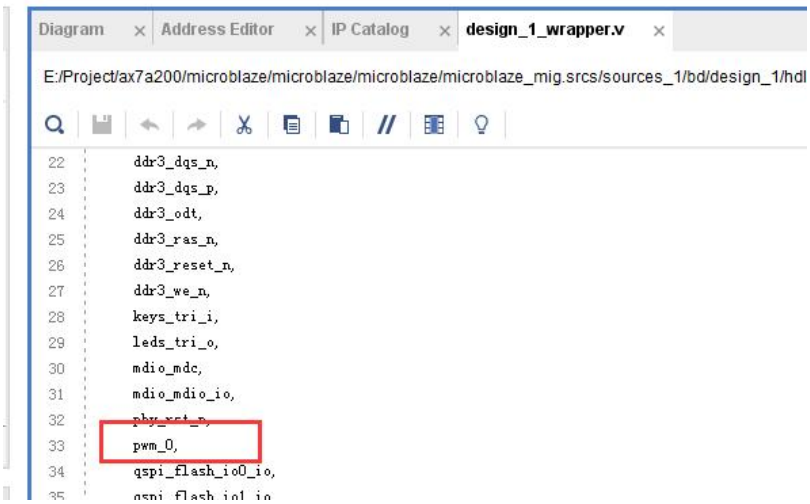
25)Save the design, and then press "**F6**" to check the design



26)Select the "**design_1.bd**" file, right-click and select "**Generate Output Products**", update the **IP** parameters and connection information to the project.



After the update, the port of the **design_1_wrapper.v** file will have an additional **pwm** interface signal.



Part 8.2.4: pwm pin constraints

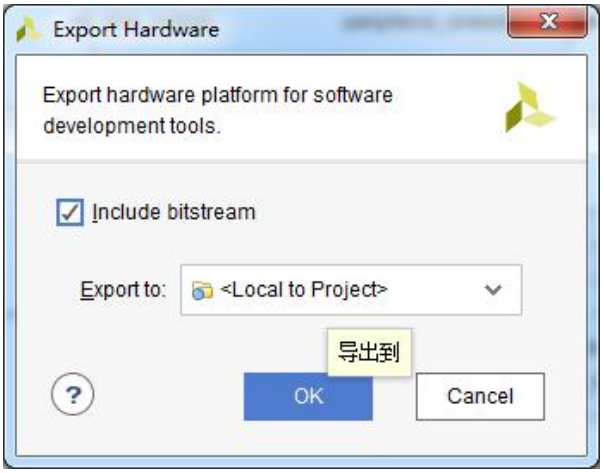
27)Add the allocation pins to the **xdc** file, allocate the **pwm_0** output port to **LED2**, and make a breathing light (take **ax7a200** as an example).

```
set_property PACKAGE_PIN M13 [get_ports {pwm_0          }]
set_property IOSTANDARD LVCMOS33 [get_ports {pwm_0      }]
```

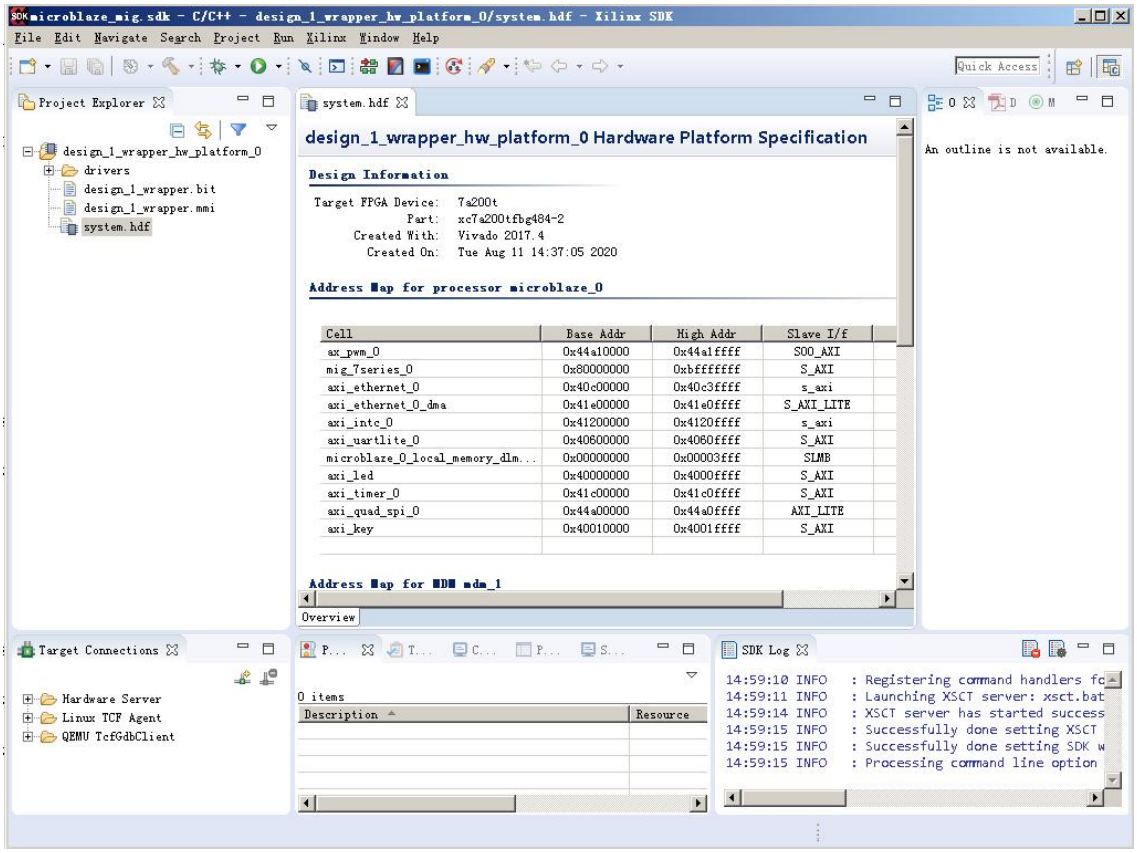
28)Compile and generate bit file, then export hardware information, start **SDK**.

Part 8.3: SDK software writing and debugging

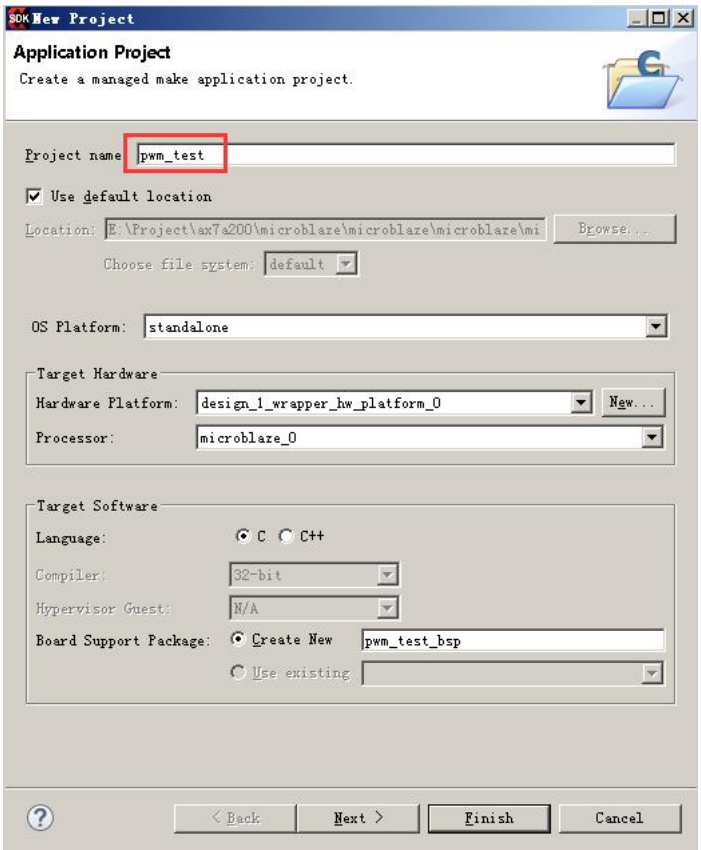
1) Export hardware



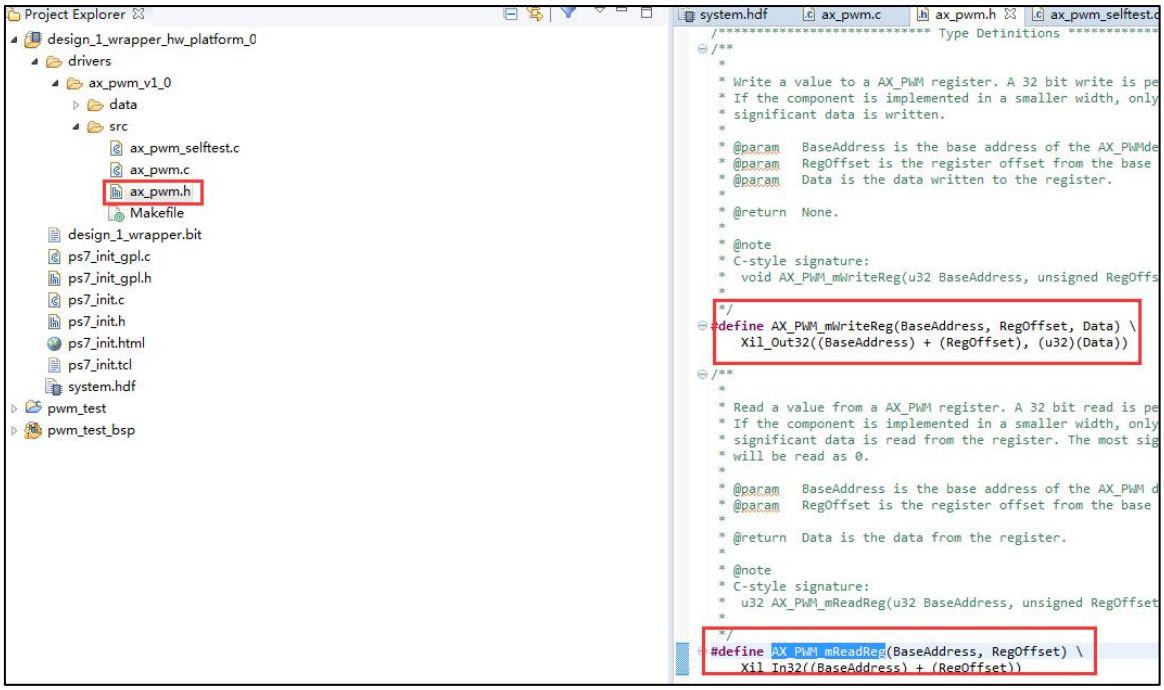
2) Start the SDK



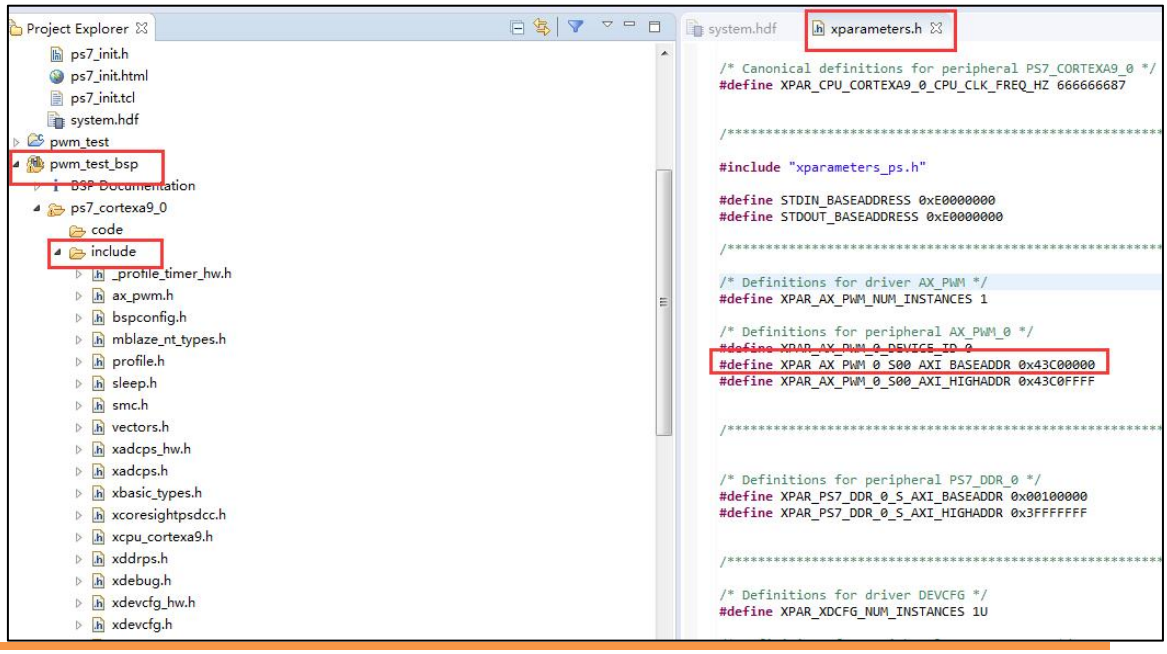
3) Create a new APP, select "**Hello World**" as the template



4) The previous examples all use the xilinx IP. Most xilinx provides a set of APIs. For this custom IP, we need to develop it ourselves. First look at the resources in the APP directory. You can find an ax_pwm.h file. This file Contains the read and write macro definitions of custom IP registers.



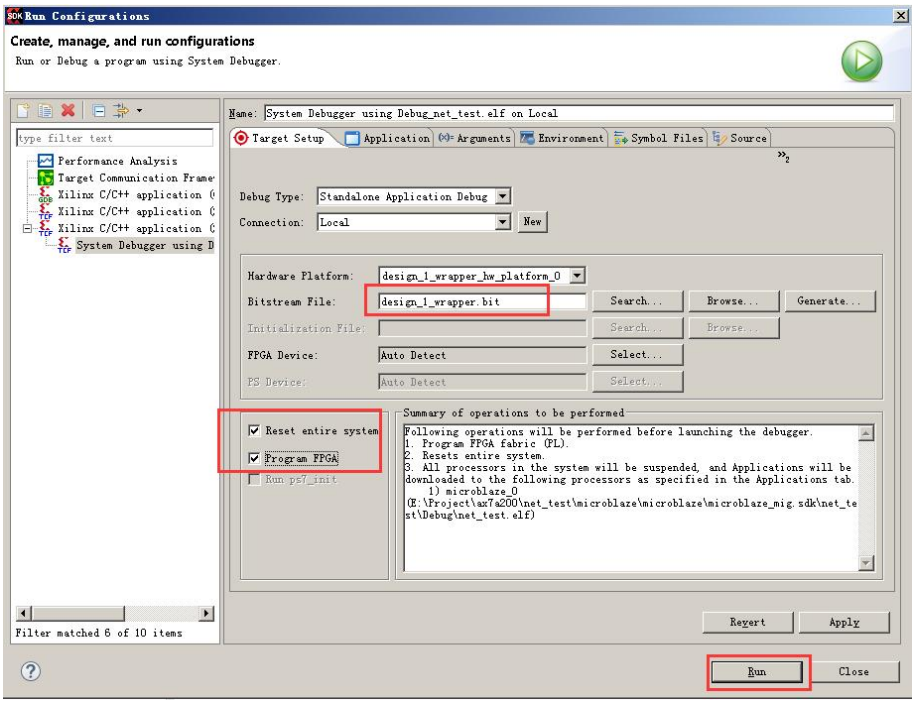
5) Find the "xparameters.h" file in the bsp, this very important file, in which the register base address of the custom IP can be found, and the base address of the custom IP can be found.



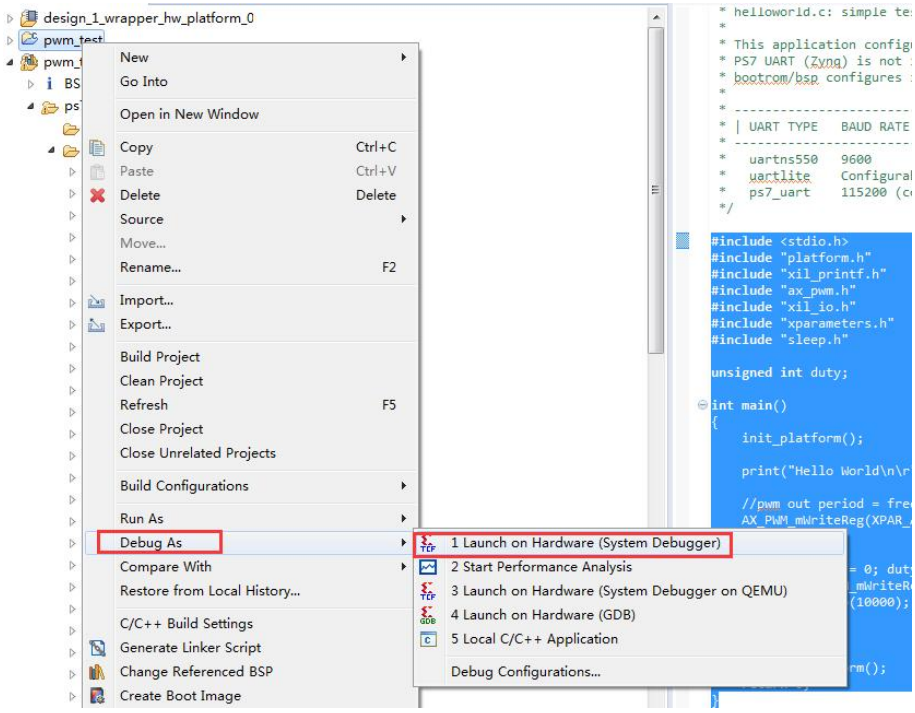
6) With the register read and write macro and the base address of the custom IP, we start to write the code and test the custom IP. We first control the PWM output frequency by writing the register AX_PWM_S00_AXI_SLV_REG0_OFFSET, and then control the duty ratio of the PWM output by writing the register AX_PWM_S00_AXI_SLV_REG1_OFFSET.

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

unsigned int duty;

int main()
{
    u32 duty;

    init_platform();

    print("Hello World\n\r");

     //pwm out period = frequency(pwm_out) * (2 ** N) / frequency(clk);
     AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR, AX_PWM_S00_AXI_SLV_REG0_OFFSET, 17179);//200hz

     while (1) {
          for (duty = 0; duty < 0xffffffff; duty = duty + 50000) {
               AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR, AX_PWM_S00_AXI_SLV_REG1_OFFSET, duty);
               usleep(20);
          }
     }

    cleanup_platform();
    return 0;
}
```

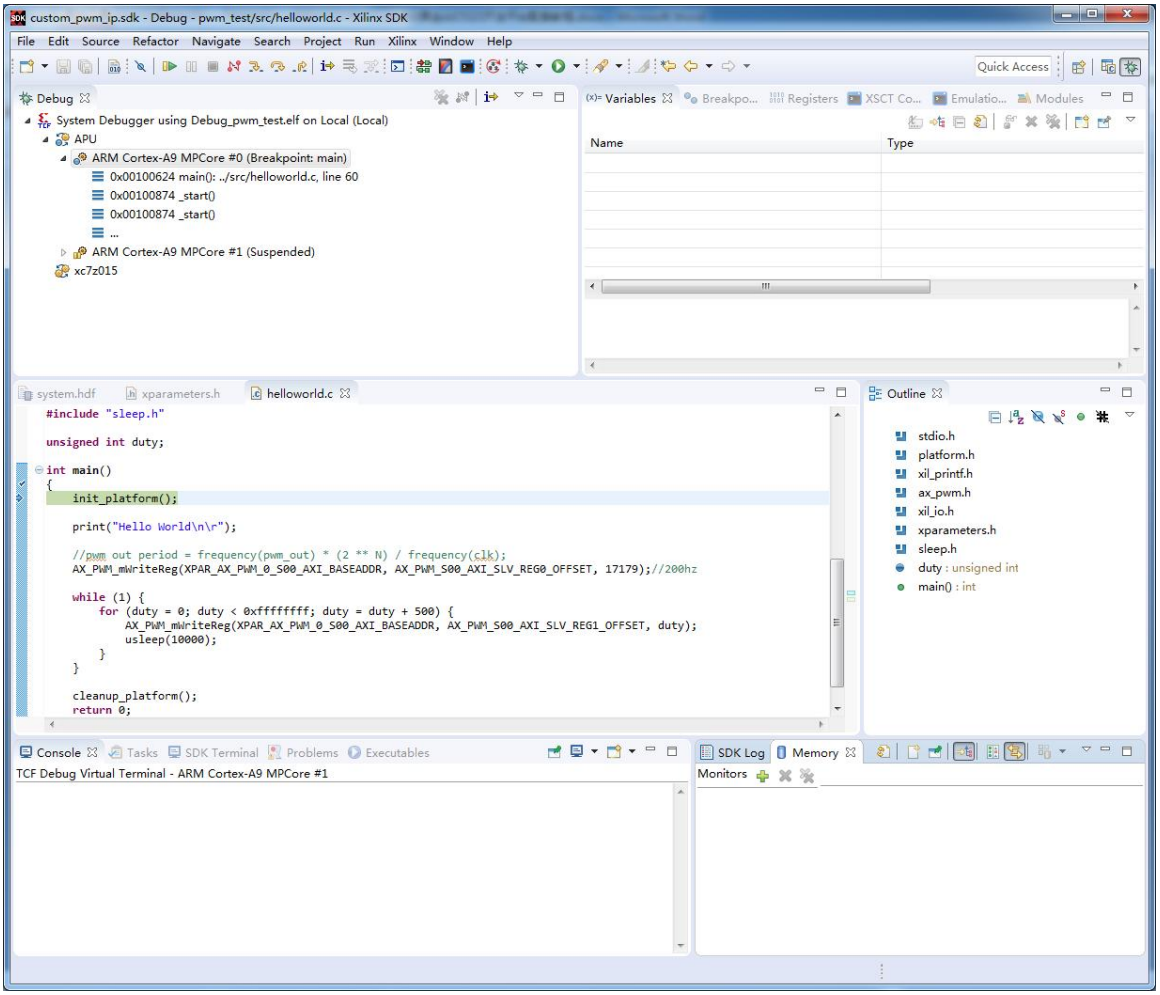
7) By running the code, we can see that LED2 presents the effect of a breathing light



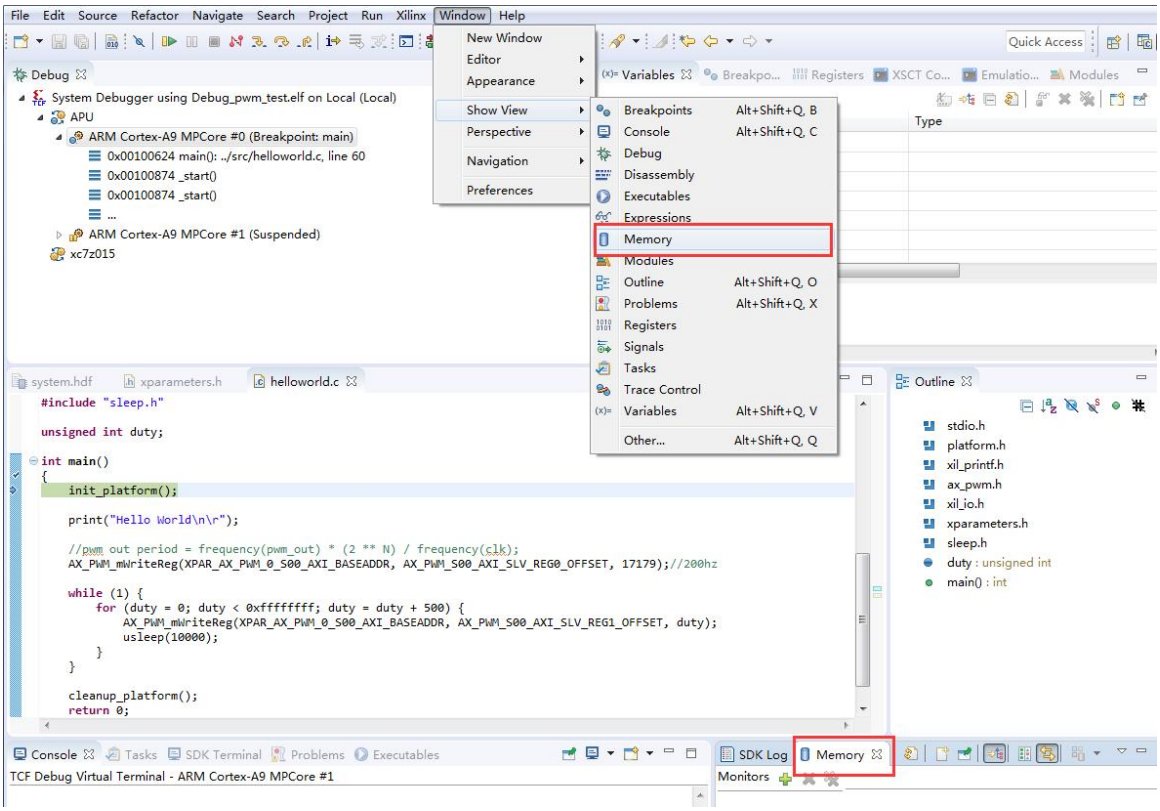
8) Through debug, let's check the register



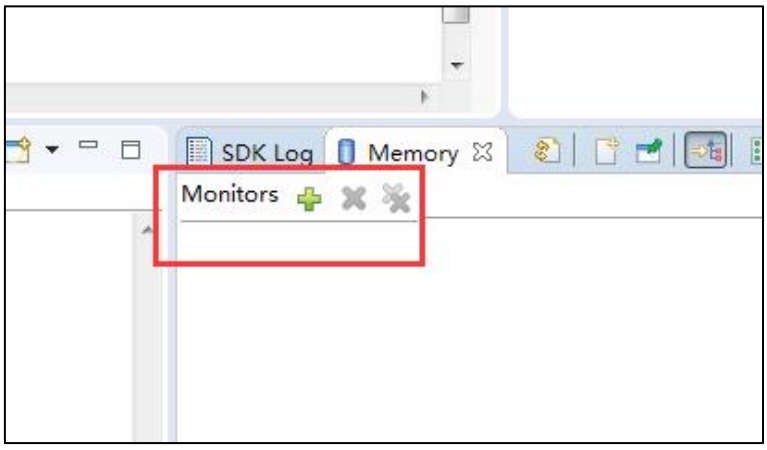
9) Enter the debug state, press "**F6**" to run in a single step.



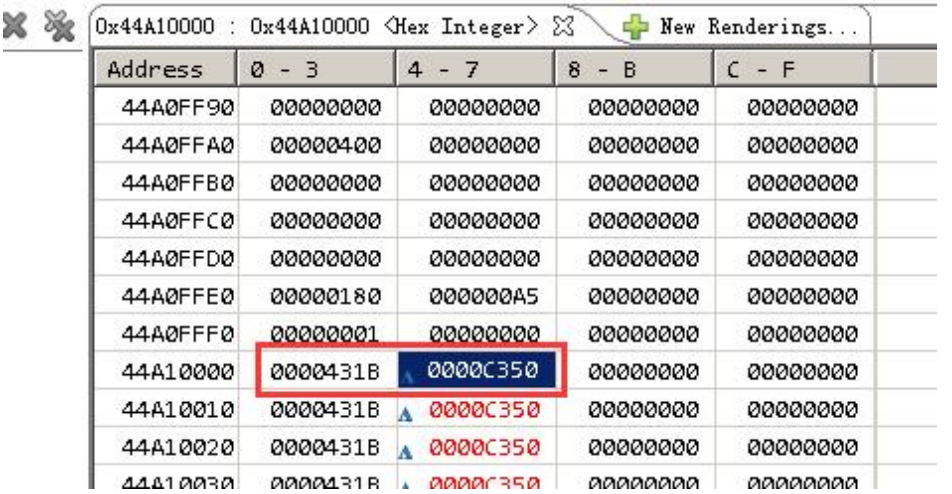
10)You can view the "Memory" window through the menu



11)Add a monitoring address "0x43c00000"



12)Single step, observe changes



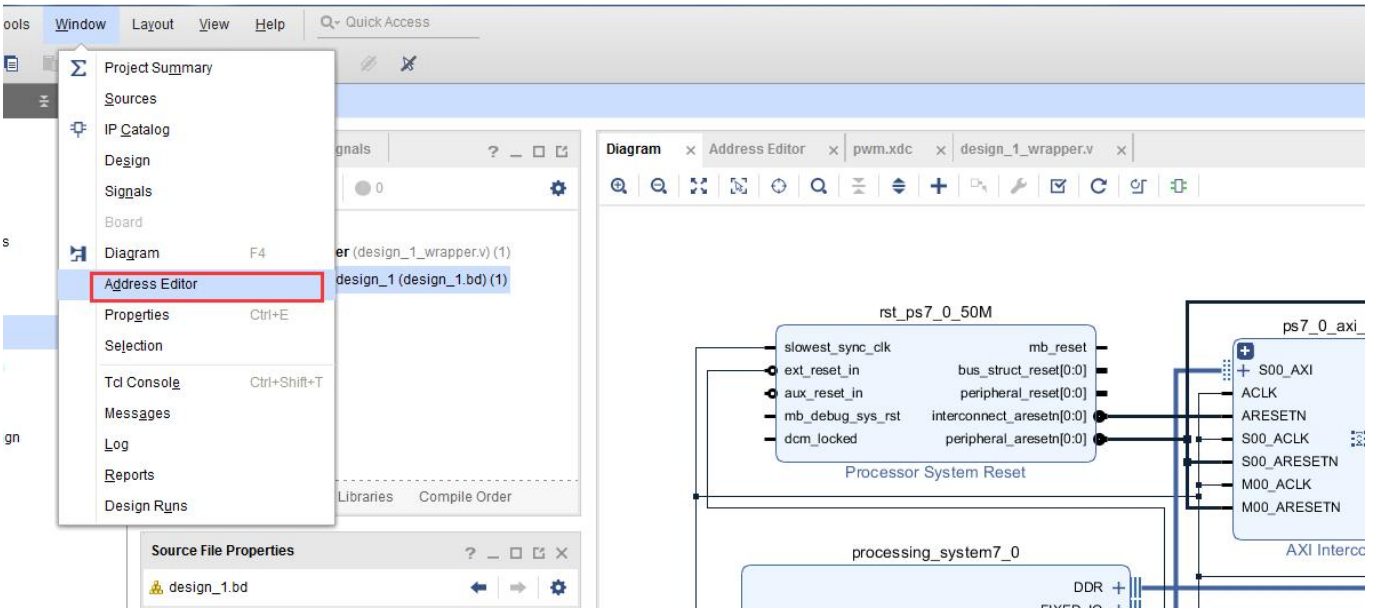
Part 8.4: Experiment Summary

Through this experiment, we have mastered more **SDK** debugging skills and the core content of **ARM + FPGA** development, which is the data interaction between ARM and FPGA.

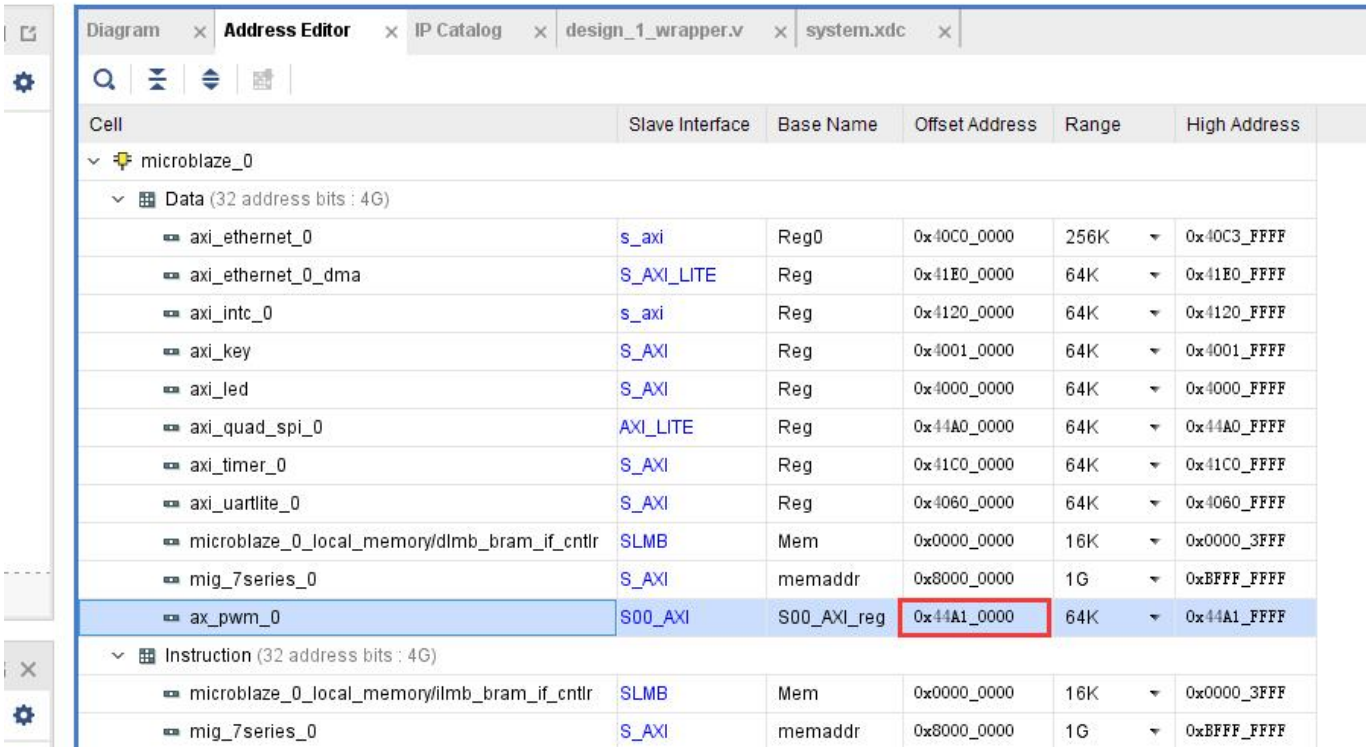
Part 8.5: Q&A

Part 8.5.1: How to know the base address of AXI IP

1) As shown in the figure below, open "**Address Editor**", you can see the address allocation



2) The address is usually assigned automatically by Vivado, we can also modify the address



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_ethernet_0	s_axi	Reg0	0x40C0_0000	256K	0x40C3_FFFF
axi_ethernet_0_dma	S_AXI_LITE	Reg	0x41E0_0000	64K	0x41E0_FFFF
axi_intc_0	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_key	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
axi_led	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	16K	0x0000_3FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	1G	0xBFFF_FFFF
ax_pwm_0	S00_AXI	S00_AXI_reg	0x44A1_0000	64K	0x44A1_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	16K	0x0000_3FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	1G	0xBFFF_FFFF

Part 9: VDMA HDMI Display

The experimental Vivado project is "**vdma_hdmi_out**".

In this experiment, we will combine the architecture of the **HMDI** display system, so that users can control the display of **HDMI** video and images in the **SDK** environment. There are many solutions to realize image display, but they are all inseparable from the **DMA** system. The **DMA** system can read the display data from the **ddr3** to the display, reducing the overhead of the **CPU**. **VDMA** is a special **DMA** developed by xilinx, dedicated to video input and output. It is an important part of learning **xilinx FPGA** video processing.

In this experiment, we will use the IP provided by Xilinx, such as **VDMA, VID_OUT, V_TC** and other video output and control **IP**. User-defined **IP**, such as **DYNCLK**, will also be used, corresponding to some FPGA development boards without **HDMI** codec chips, and a custom **RGB2DVI IP** is also required.

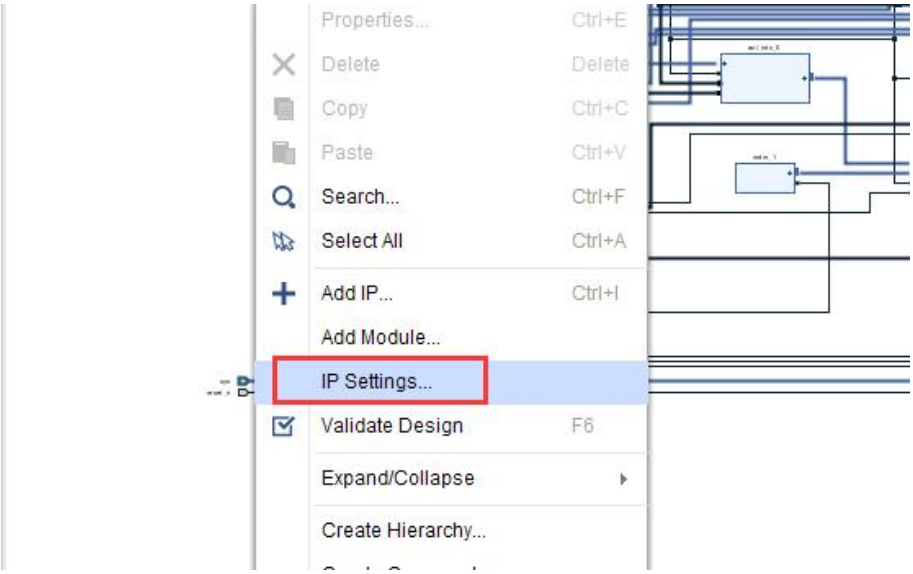
Part 9.1: Vivado Project Modification

The Vivado project is modified on the basis of the Vivado project configuration in Part 8, and the SDK directory is deleted.

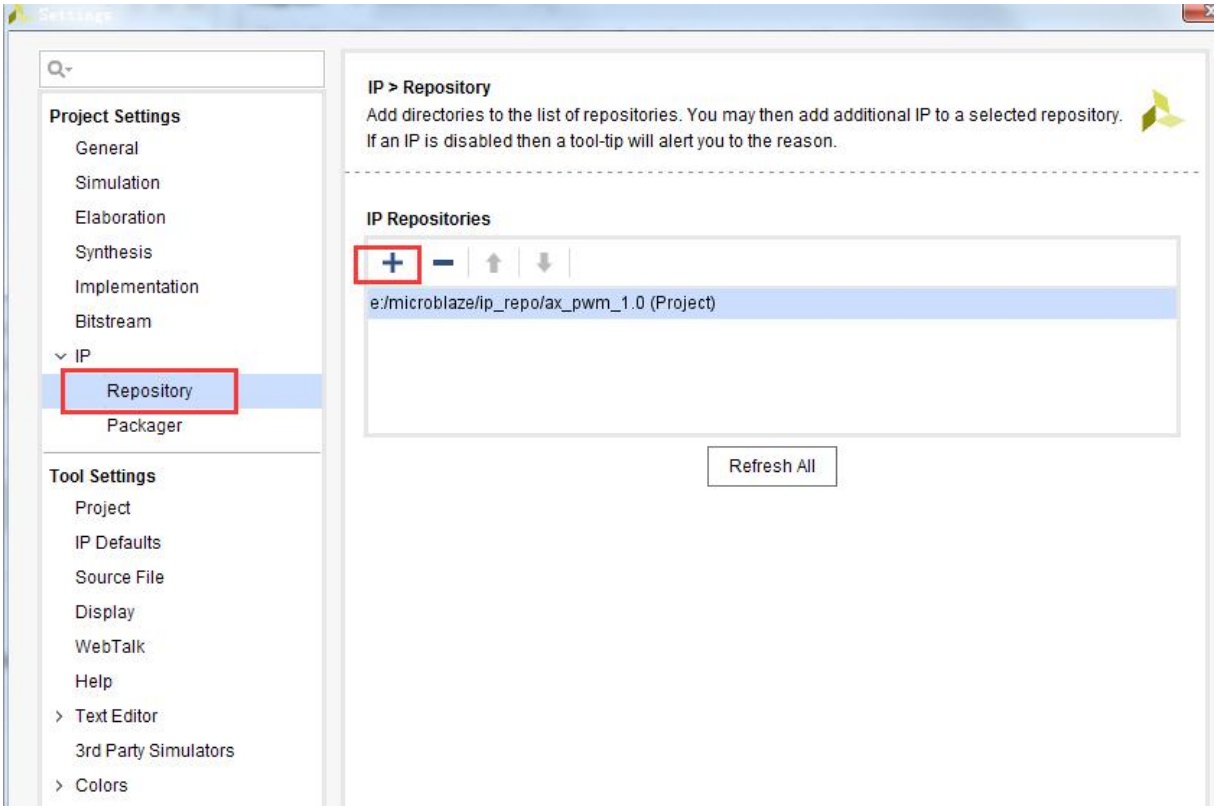
Part 9.1.1: Add custom IP

1) Copy the two IPs of **axi_dynclk** and **rgb2dvi** provided by us to the **ip_repo** directory of the project

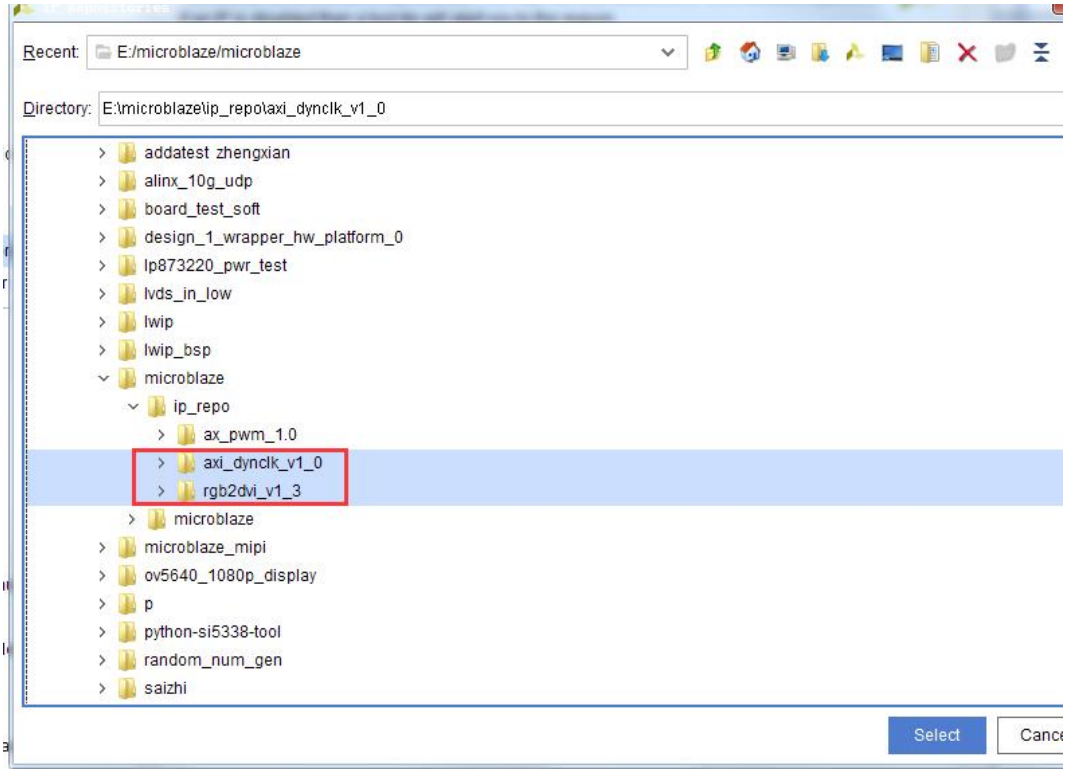
2) In the Vivado development environment, right-click on the blank space of **Diagram**, and select **IP Setting...**



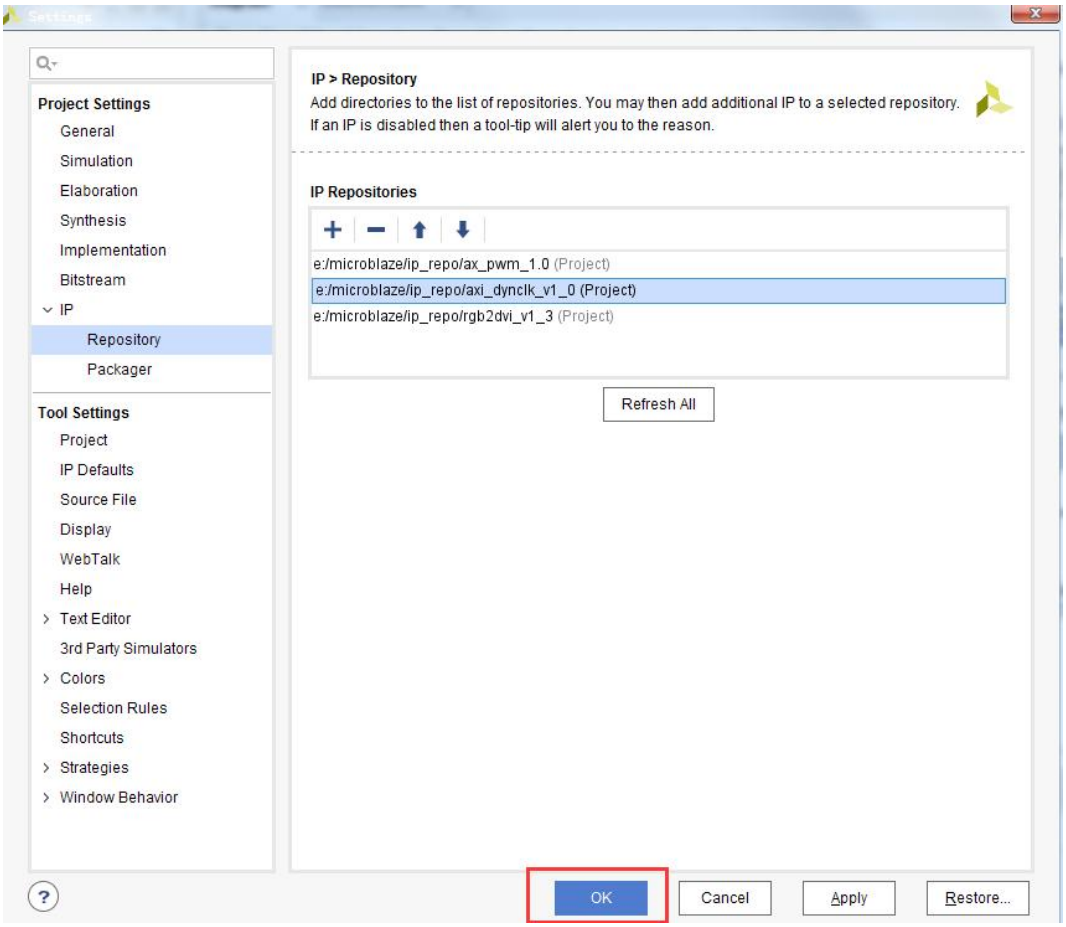
3) Select **Repository Manager** in the IP interface, and then click  to add a new **IP** library



4) Select the **axi_dynclk_v1_0** and **rgb2dvi_v1_3** file folders in the **ip_repo** directory, and click the **Select** button.

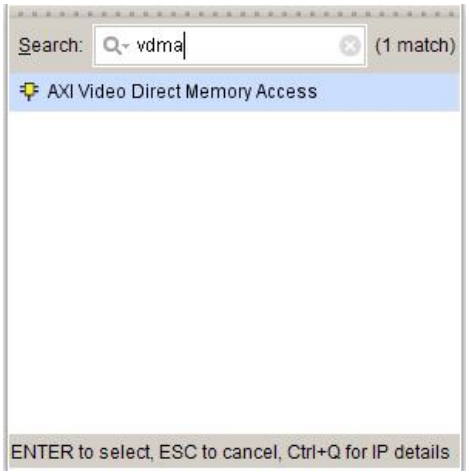


5) The custom IP has been added to the project, click OK to complete. In this way, we can use these two user-defined IPs in the project.

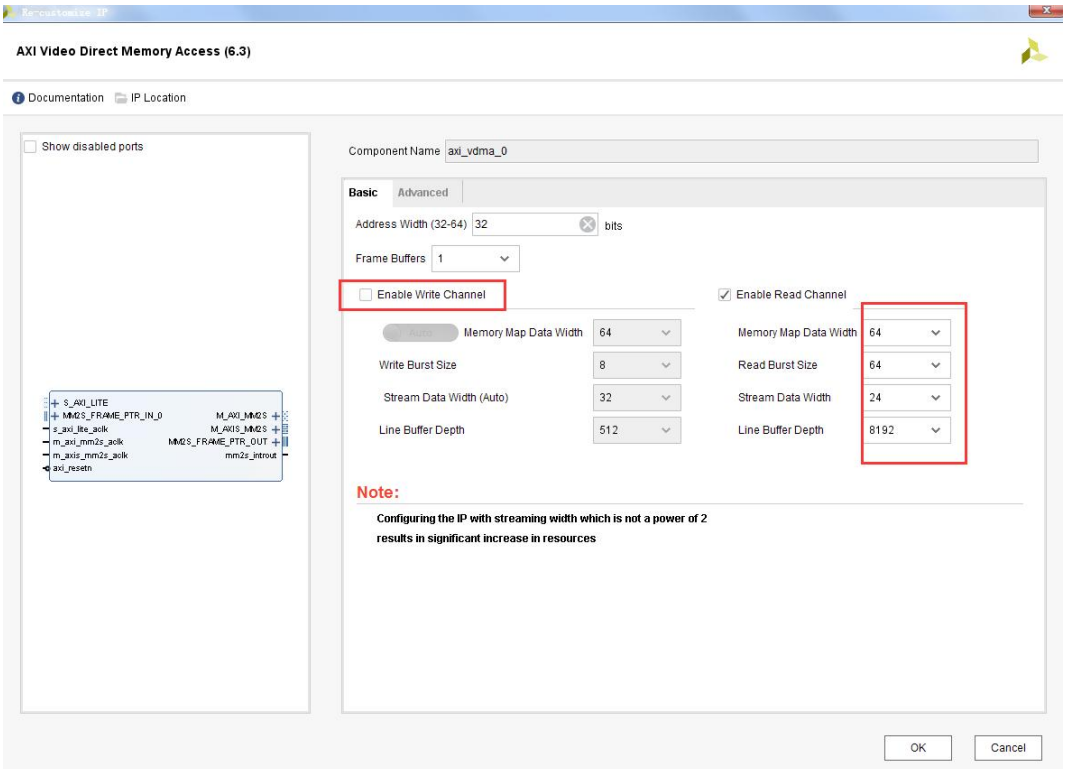


Part 9.1.2: Add VDMA

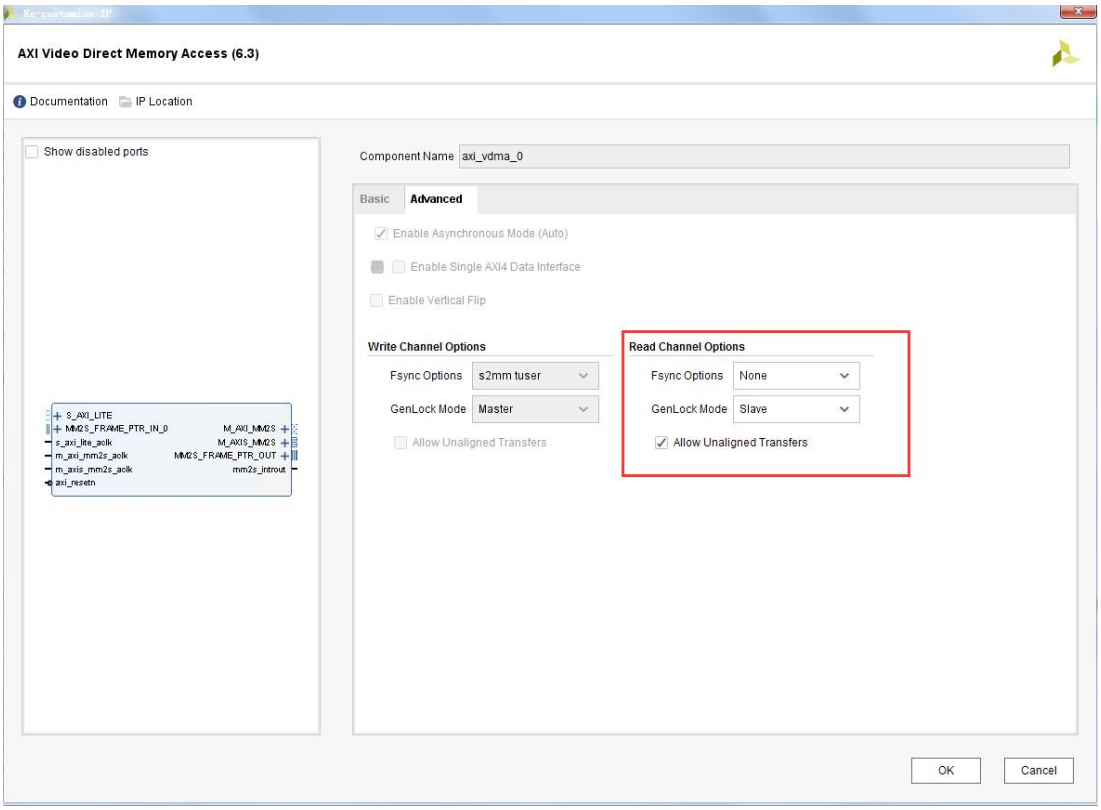
6) Add a **VDMA IP** to read data from **Microblaze** system **DDR Memory**. Search for "**vdma**" and double-click to add.



7) Double-click to open the **VDMA** configuration interface, and cancel **Enable Write Channel** in the Basic interface. Because this experiment is only to read the Memory, so just **Enable Read Channel** is fine. In addition, set the **Line Buffer Depth** to **8192**, and set the data width of the output **Stream Data Width** to **24** bits.

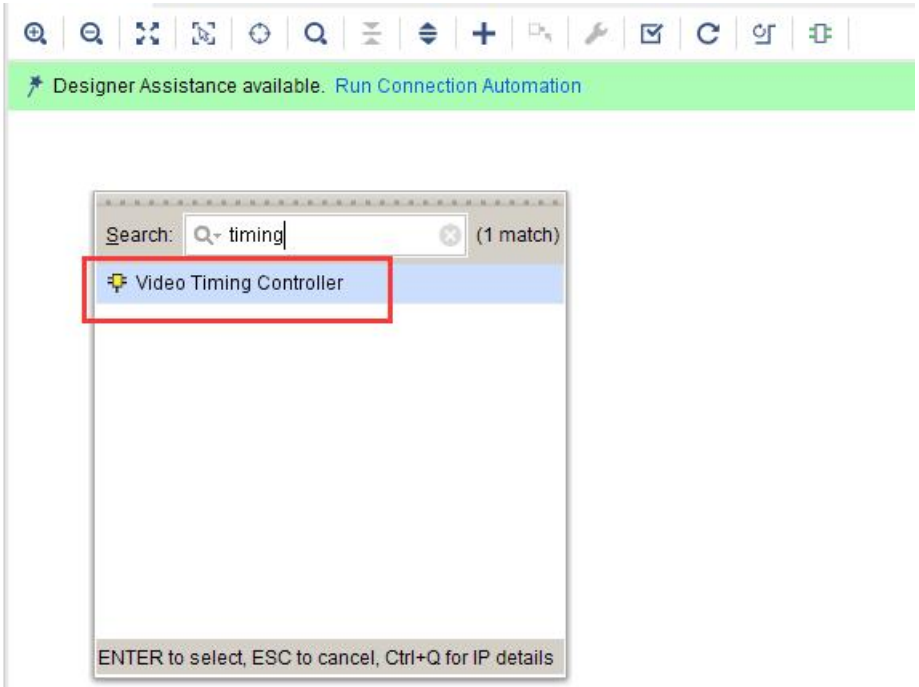


Select the **Advanced** interface and configure as follows.

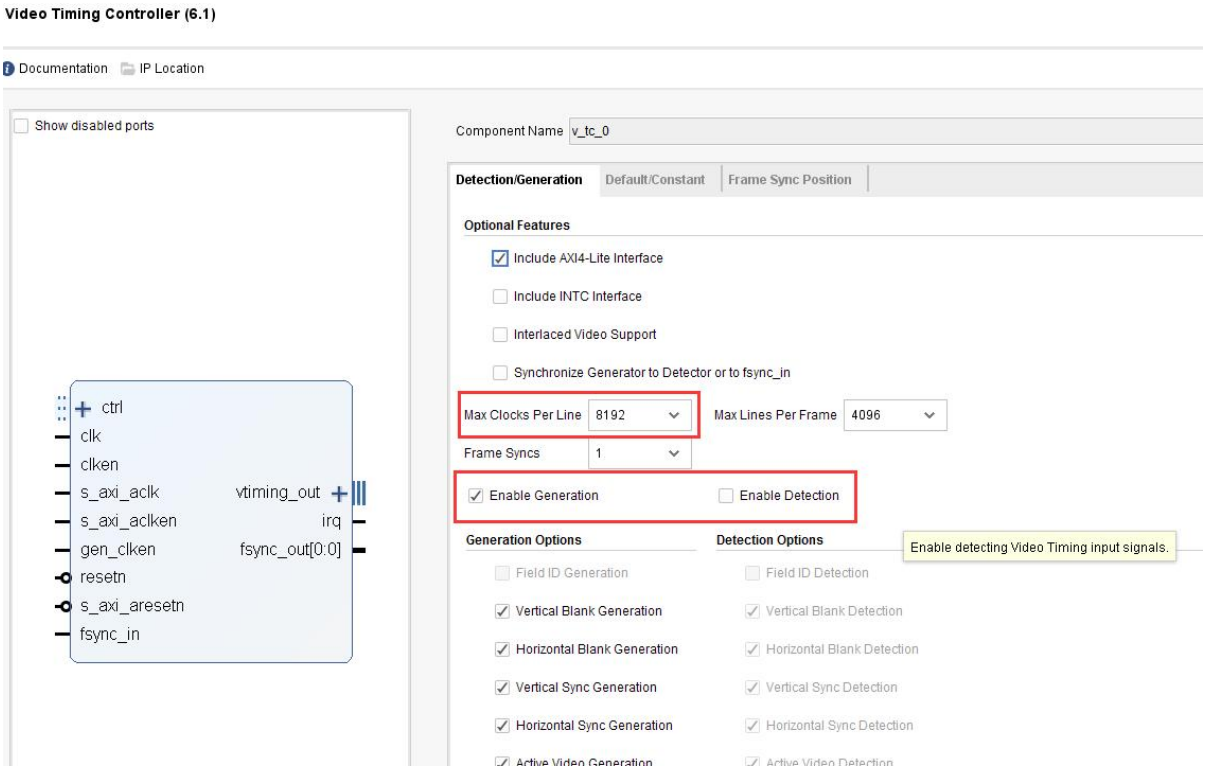


Part 9.1.3: Add Video Timing Controller

8) Add a video timing controller to generate or configure the correct **video** timing, search for "**timing**" and double-click to add.

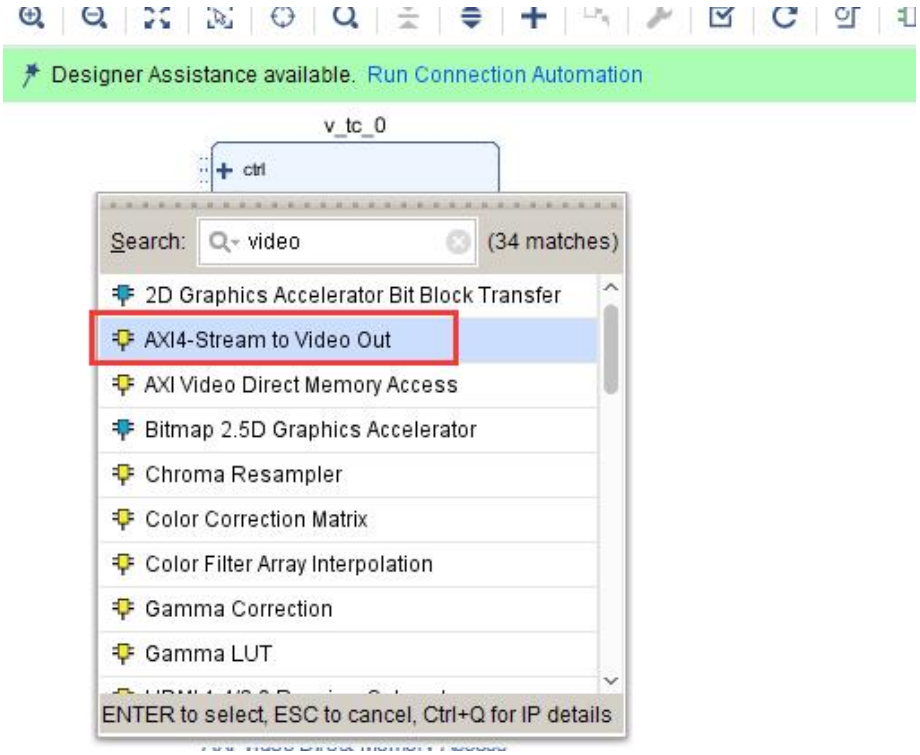


9) Configure the video timing controller parameters.

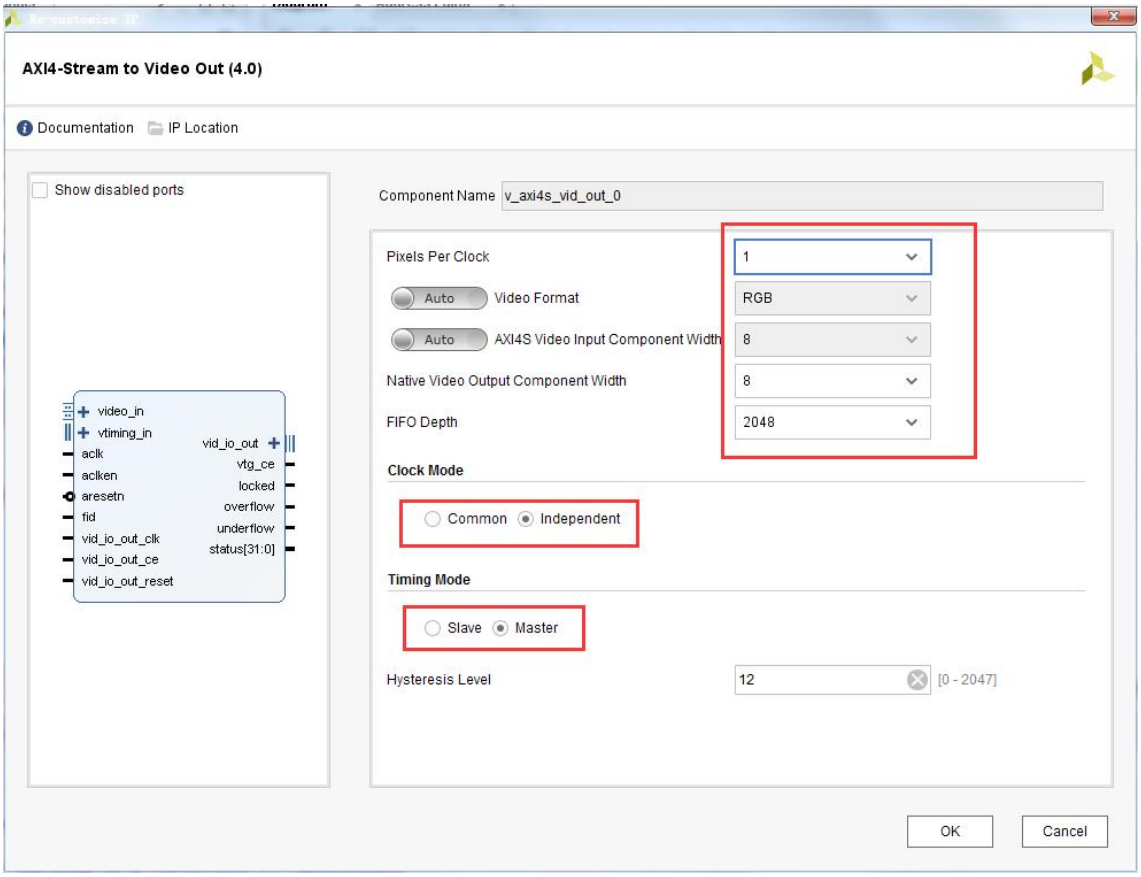


Part 9.1.4: Add AXI4-Stream to Video Out

10)AXI4-Stream to Video Out controller, search for "**video**", double click "**AXI4-Stream to Video Out**" to add.

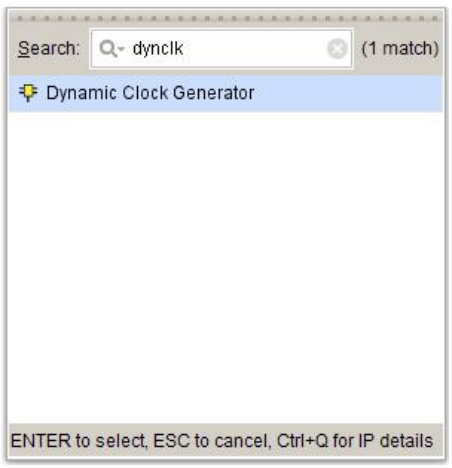


11)Configure AXI streaming video output controller parameters.



Part 9.1.5: Add axi_dynclk

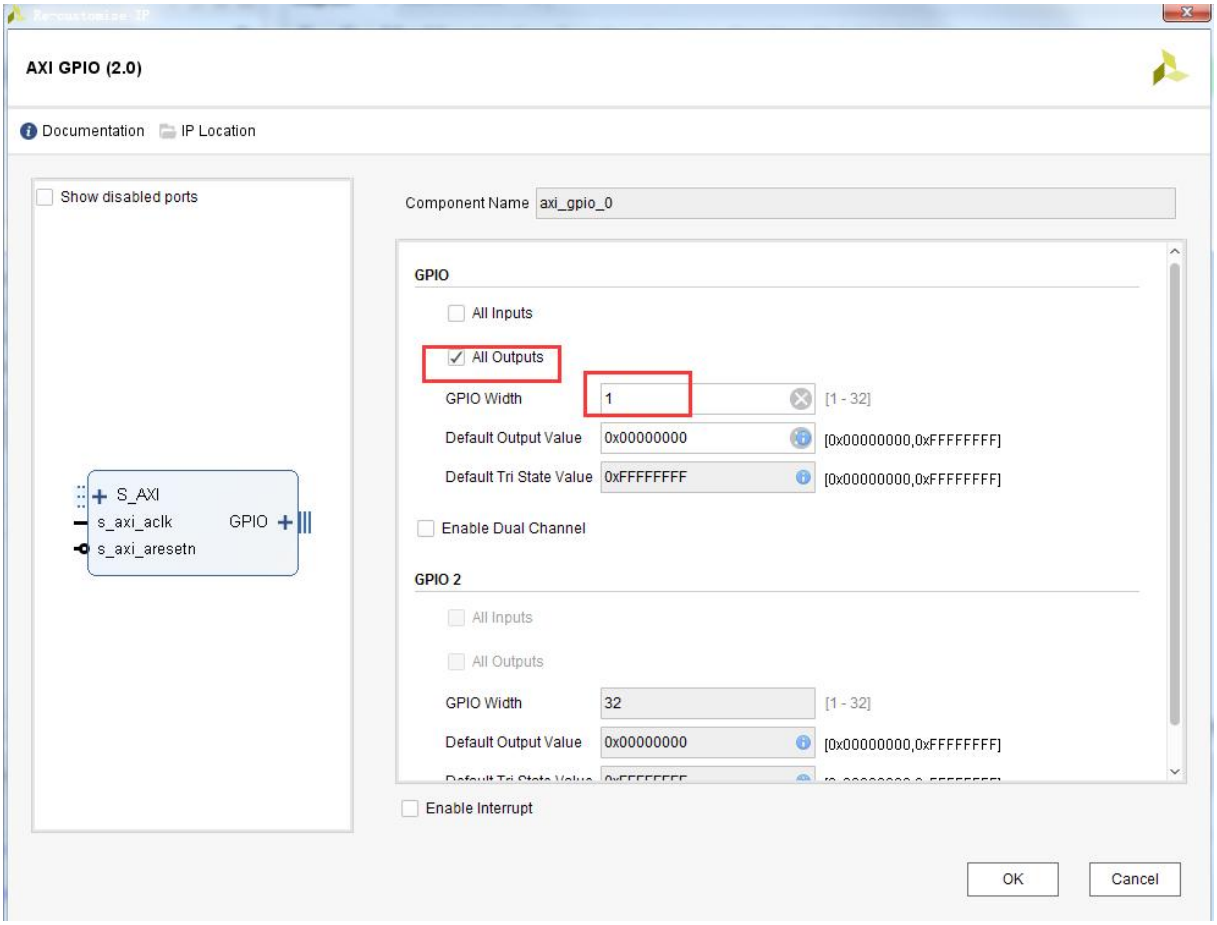
12)Since the video has many resolutions, the clock frequencies of various resolutions are different, and a dynamic clock controller needs to be used. Here we need to use the custom IP we added earlier. Search "**dynclk**", double-click "**Dynamic Clock Generator**" to add.



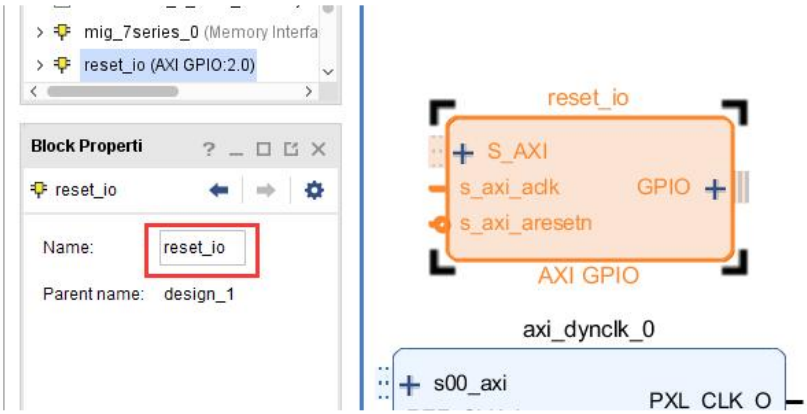
Part 9.1.6: Add I2C and GPIO

13)Add a GPIO to reset the HDMI chip (SI9134 decoder chip)

14)Configure GPIO parameters



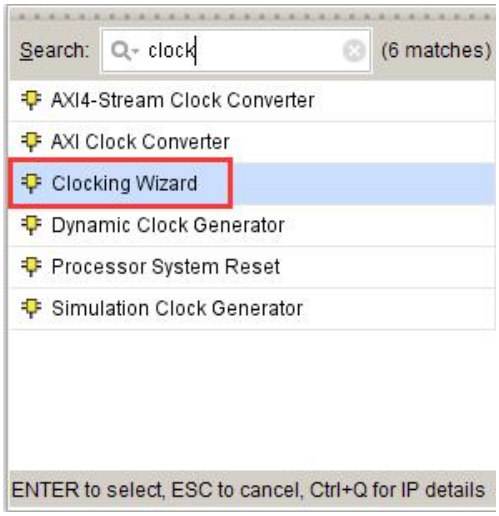
15)The IP name is taken as reset_io



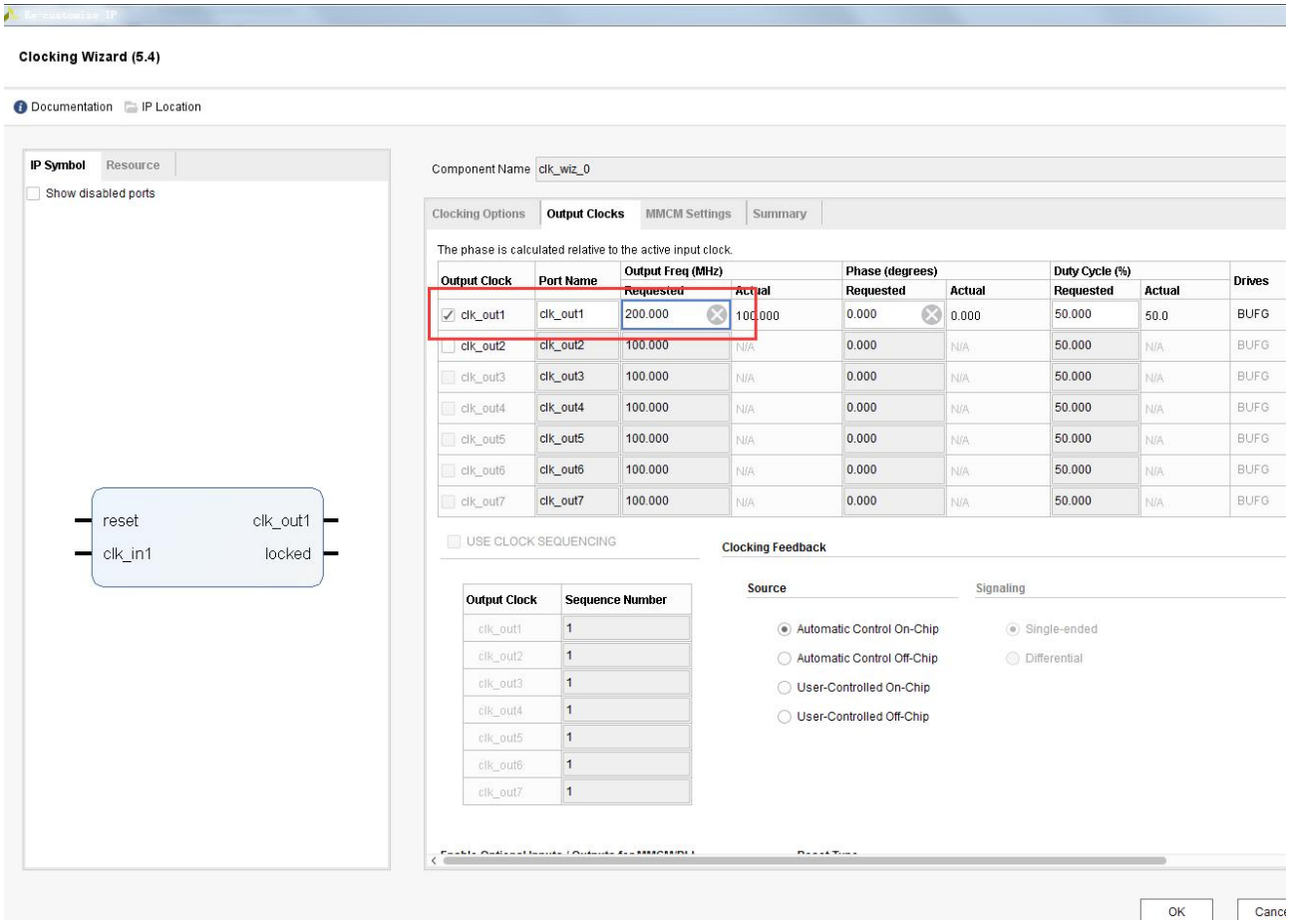
16)Add an i2c for the register configuration of the HDMI chip (SI9134 decoder chip).

Part 9.1.7: Add PII

17)Add a Clocking Wizard to output a 200Mhz clock.



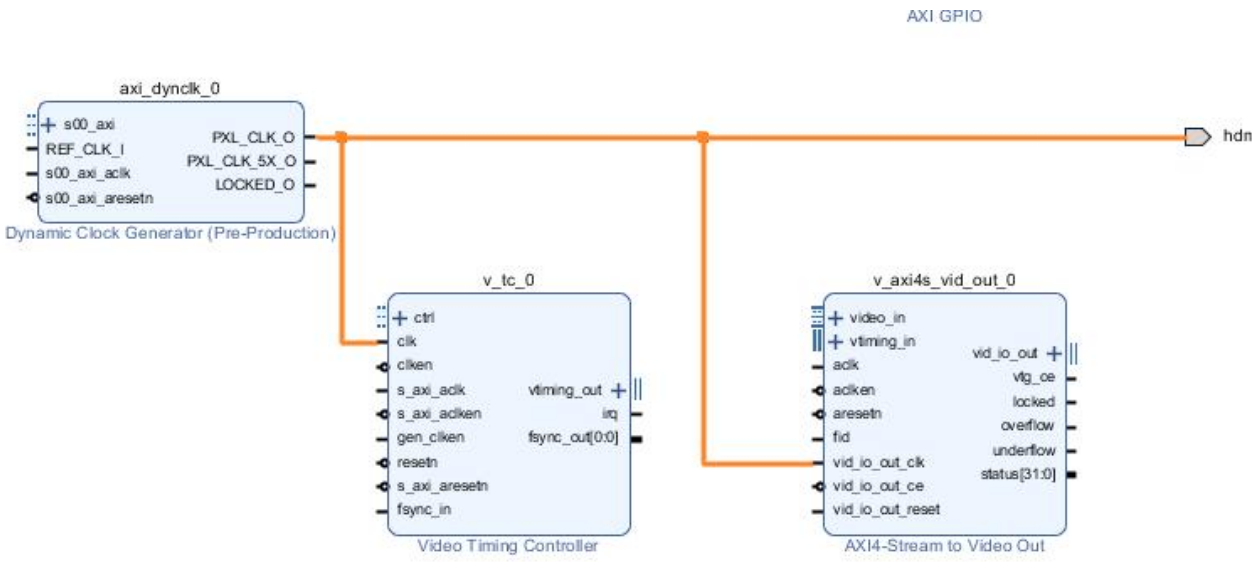
18)Configure a clock with an output frequency of **200Mhz.**



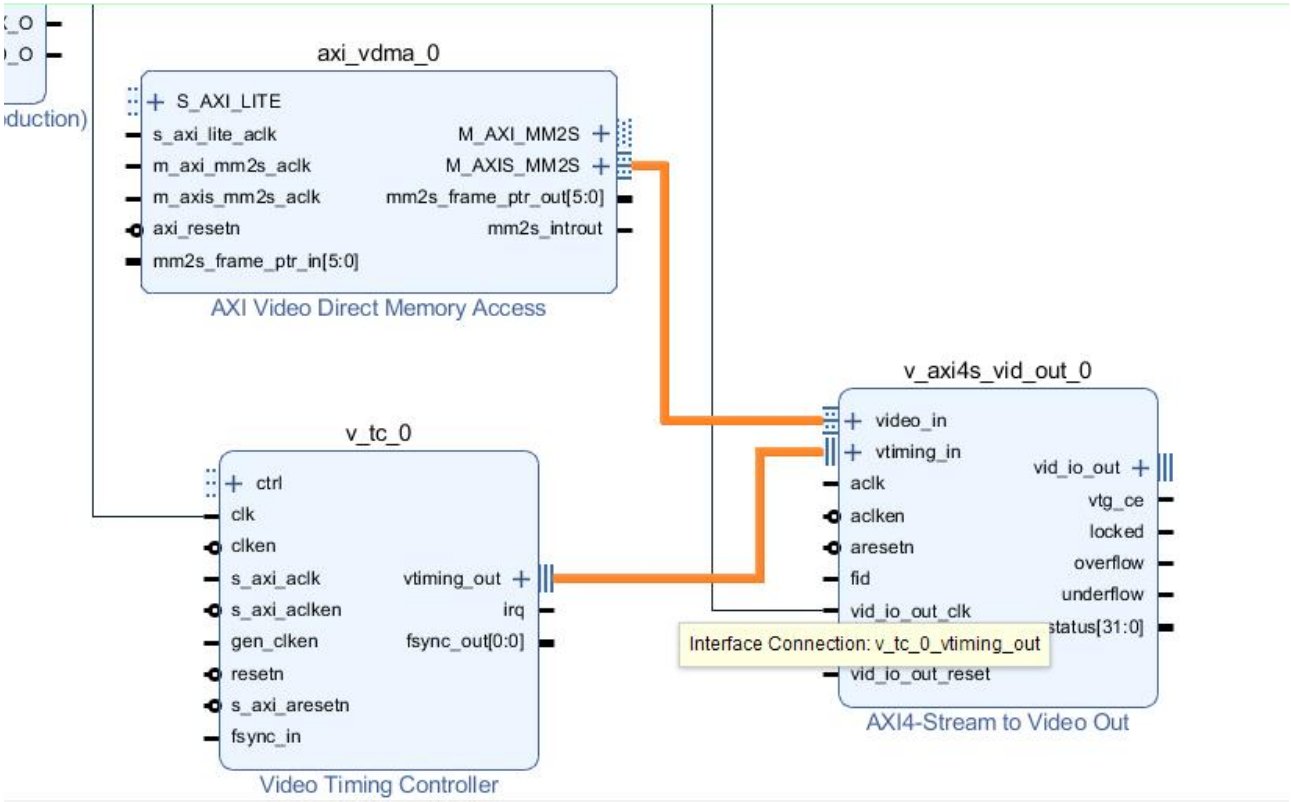
Part 9.1.8: Signal Connection

19)Set **PXL_CLK_O** of **axi_dynclk** to "**Make External**", and then rename the port to "**hdmi_out_clk**".

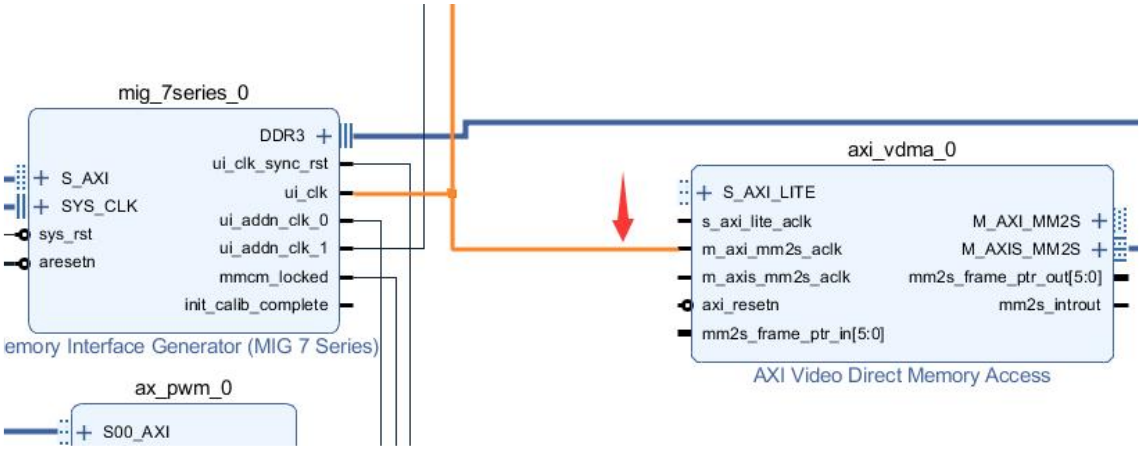
And connect to the clock pins of v_tc and v_axi4s_vid_out.



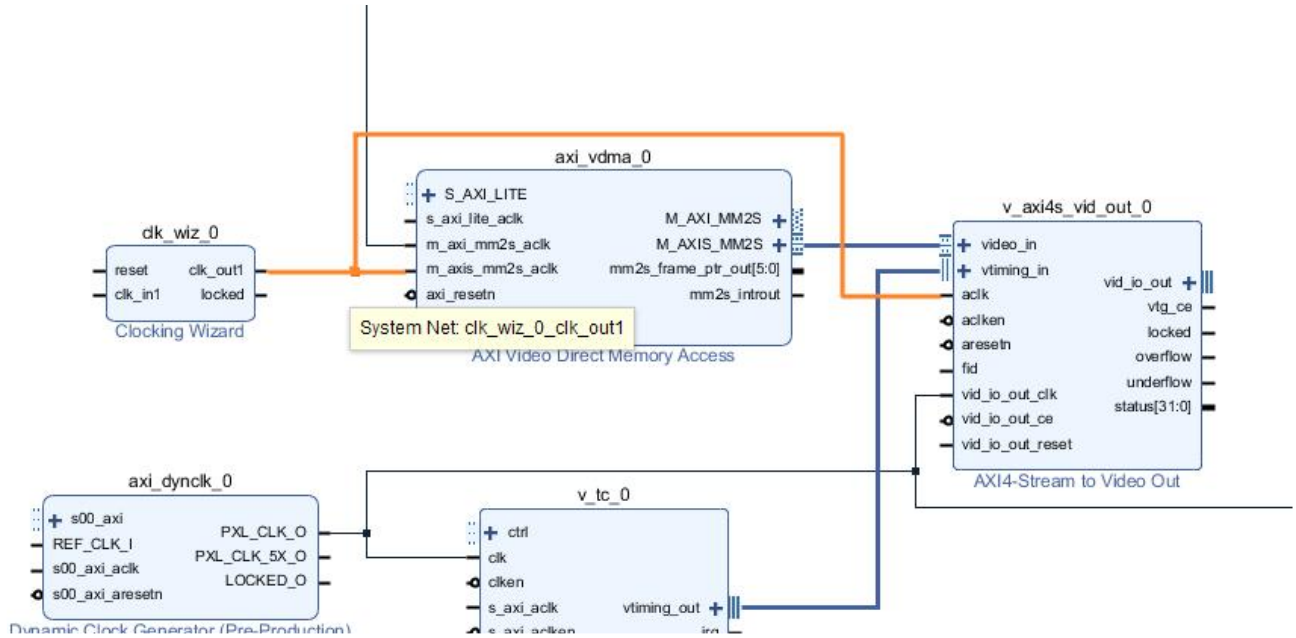
20)Connect the data stream of **vdma** and **vid_out**, and connect the timing control interface of **v_tc** and **vid_out**.



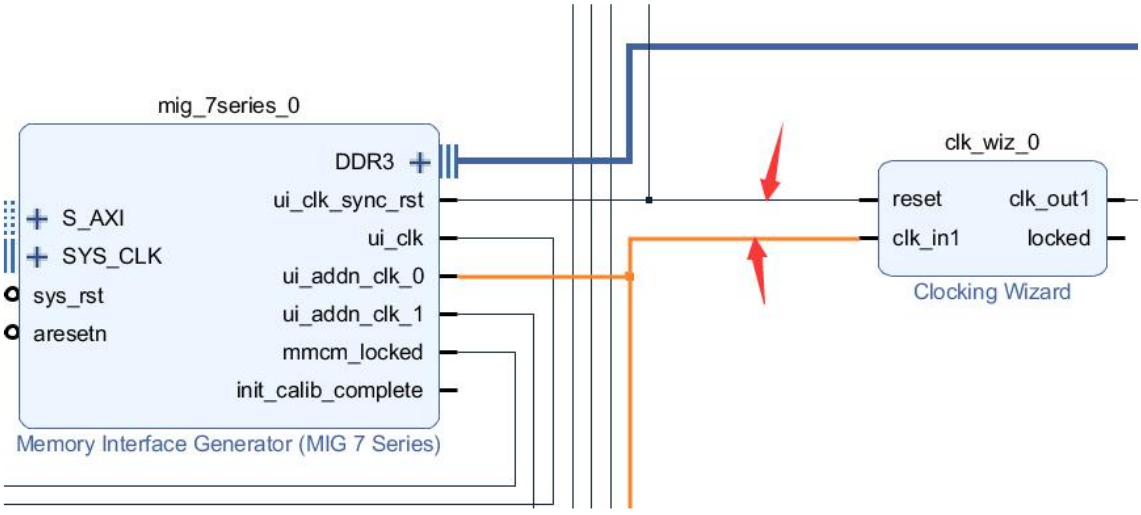
21)Connect **m_axi_mm2s_aclk** of **vdma** to **ui_clk** output of **mig_7series**, the frequency is **100Mhz**.



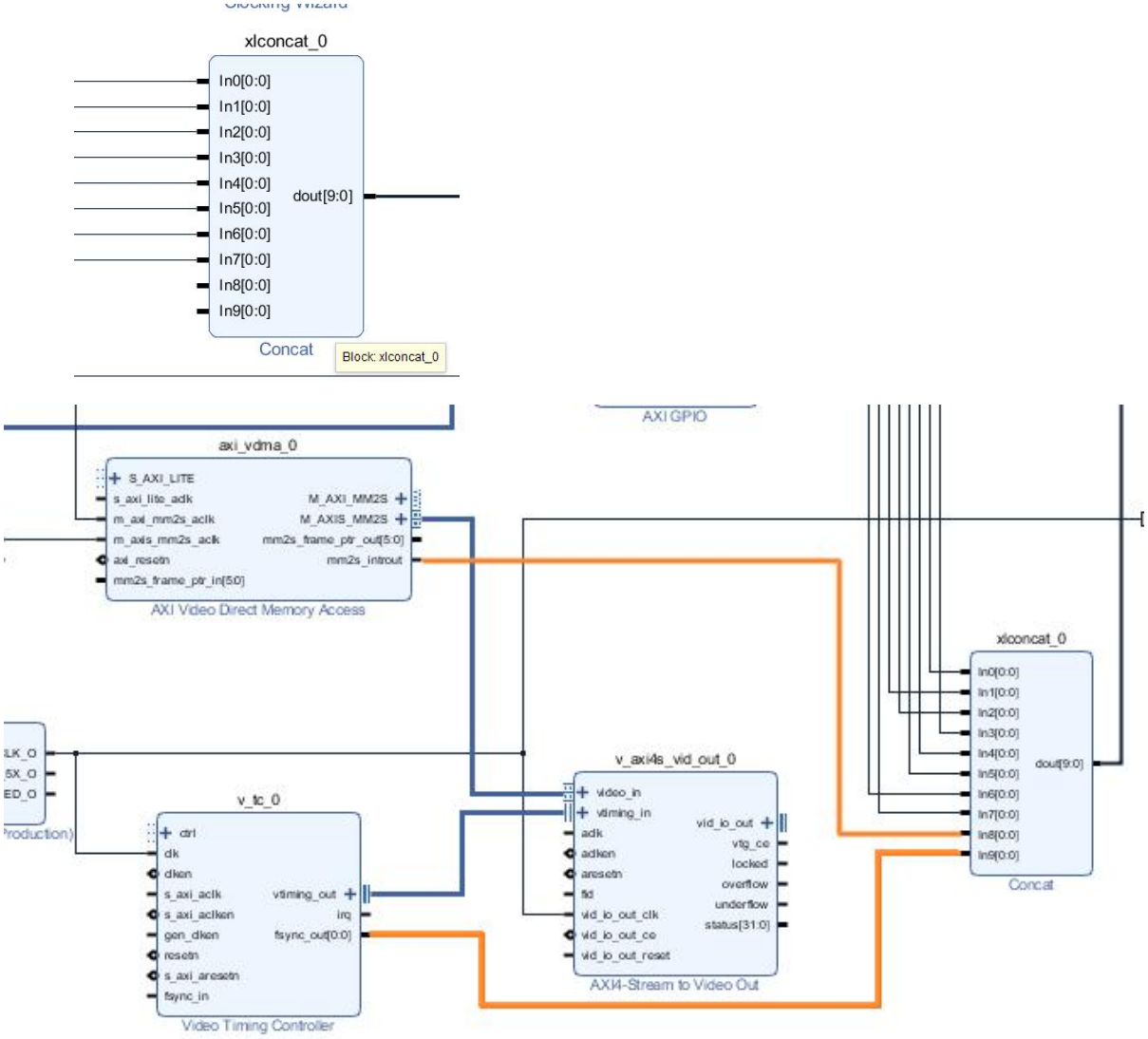
22)Connect the clock below, the frequency is **200Mhz**



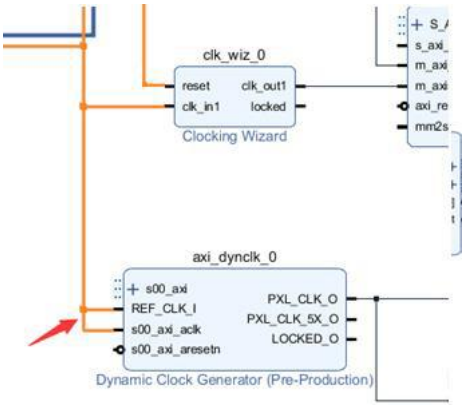
The clock input and reset input of **clk_wiz** are connected to the clock and reset of **mig_7series**, and the clock frequency is **100Mhz**.



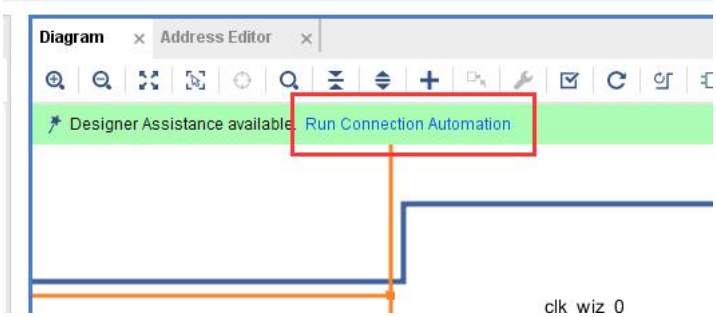
23)Modify the width of **xlconcat**, add two ports, connect the interrupt input of **vdma** and **v_tc**.



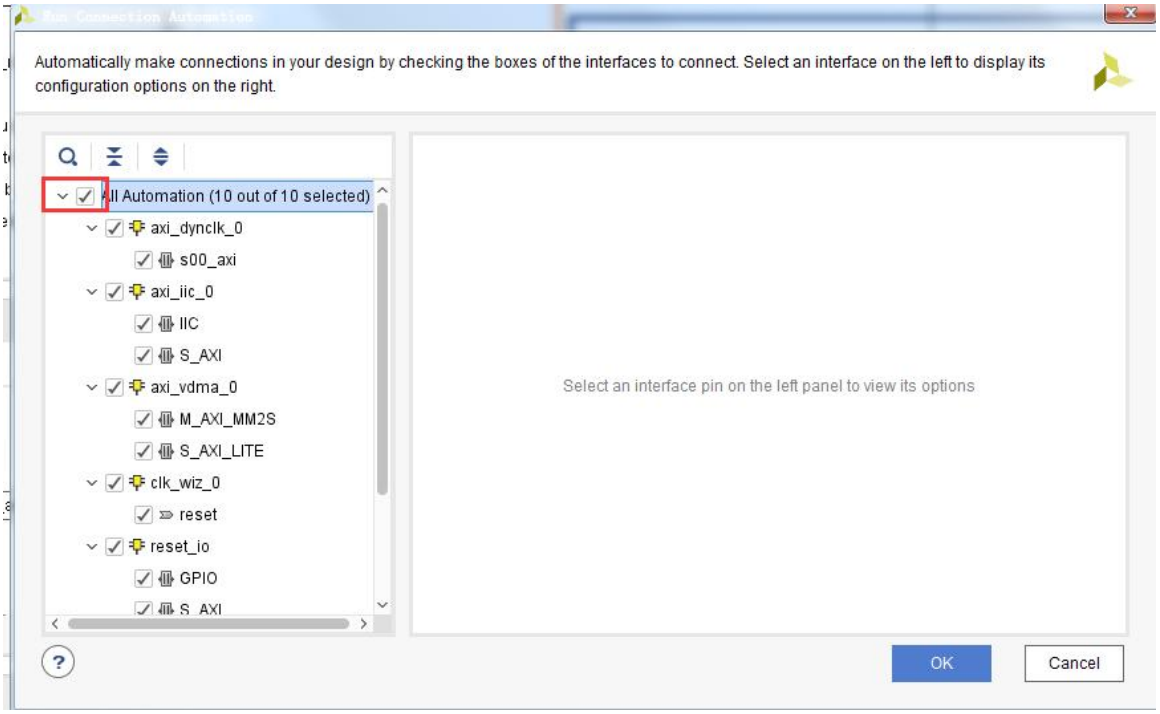
24)The input clock of **axi_dynclk** is connected to the input clock of **clk_wiz**, and the frequency is both **100mhz**.



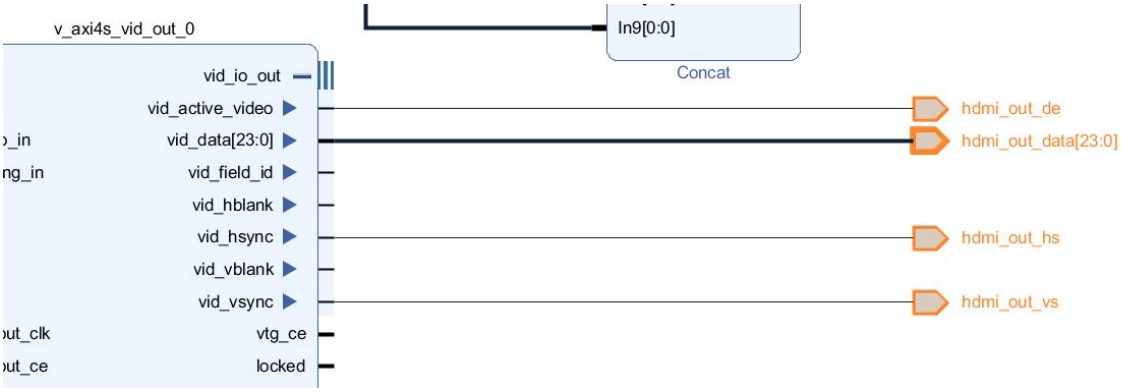
25)"**Run Block Automation**" completes other port connections.



Select all connections

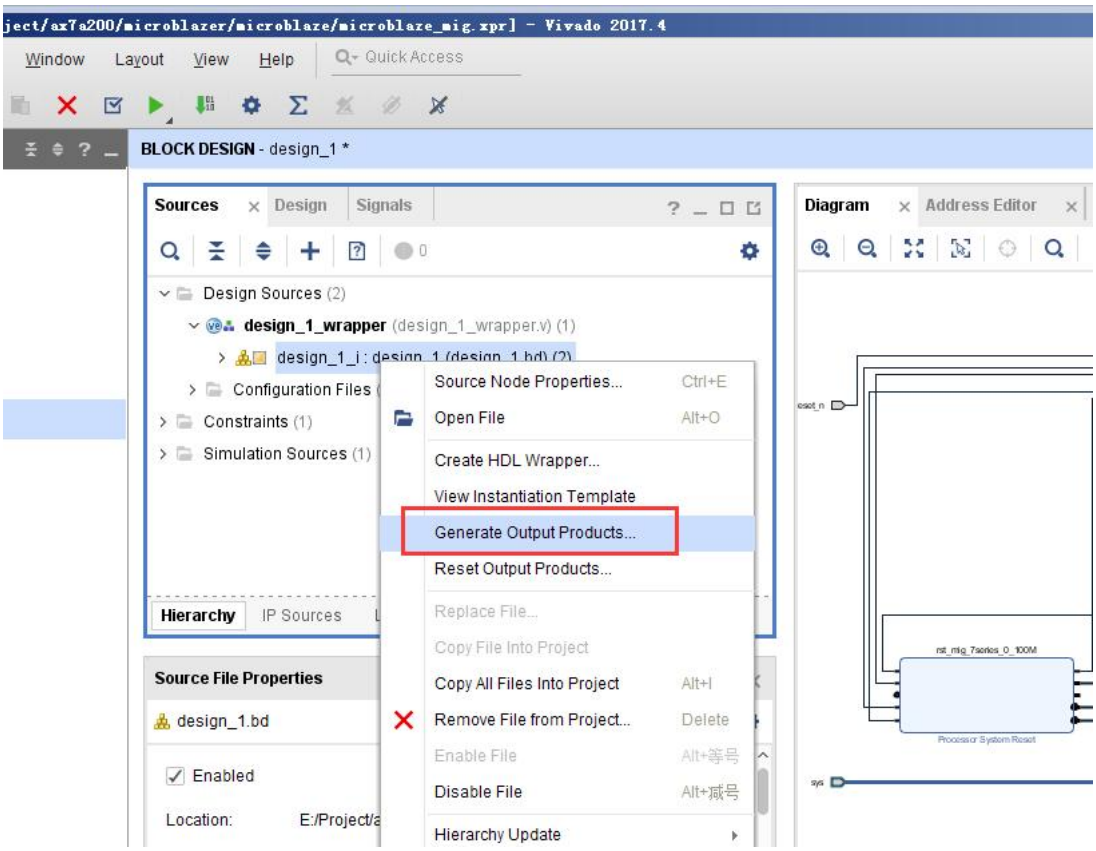


26)Set these 4 of vid_out to "**Make External**", and then change the port name as follows.



Modify the port names of **axi_iic** and **reset_io** as follows

27)Save the design, and then press "F6" to check the design. If there is no error, select the "design_1.bd" file, right-click and select "Generate Output Products", and update the IP parameters and connection information to the project



Part 9.1.10: HDMI pin constraints

28)Add the pin assignment of **hdmi** output to the *xdc* file (take **ax7a200** as an example)

```
set_property PACKAGE_PIN Y22 [get_ports hdmi_out_clk]
set_property PACKAGE_PIN V22 [get_ports {hdmi_out_data[0]}]
set_property PACKAGE_PIN Y18 [get_ports {hdmi_out_data[1]}]
set_property PACKAGE_PIN Y19 [get_ports {hdmi_out_data[2]}]
set_property PACKAGE_PIN W19 [get_ports {hdmi_out_data[3]}]
set_property PACKAGE_PIN W20 [get_ports {hdmi_out_data[4]}]
set_property PACKAGE_PIN Y21 [get_ports {hdmi_out_data[5]}]
set_property PACKAGE_PIN U21 [get_ports {hdmi_out_data[6]}]
set_property PACKAGE_PIN T21 [get_ports {hdmi_out_data[7]}]
set_property PACKAGE_PIN W21 [get_ports {hdmi_out_data[8]}]
set_property PACKAGE_PIN W22 [get_ports {hdmi_out_data[9]}]
set_property PACKAGE_PIN T20 [get_ports {hdmi_out_data[10]}]
set_property PACKAGE_PIN AB18 [get_ports {hdmi_out_data[11]}]
set_property PACKAGE_PIN AA18 [get_ports {hdmi_out_data[12]}]
set_property PACKAGE_PIN AA19 [get_ports {hdmi_out_data[13]}]
set_property PACKAGE_PIN AB20 [get_ports {hdmi_out_data[14]}]
set_property PACKAGE_PIN AA20 [get_ports {hdmi_out_data[15]}]
set_property PACKAGE_PIN AA21 [get_ports {hdmi_out_data[16]}]
set_property PACKAGE_PIN AB22 [get_ports {hdmi_out_data[17]}]
set_property PACKAGE_PIN AB21 [get_ports {hdmi_out_data[18]}]
set_property PACKAGE_PIN W17 [get_ports {hdmi_out_data[19]}]
set_property PACKAGE_PIN V17 [get_ports {hdmi_out_data[20]}]
set_property PACKAGE_PIN V20 [get_ports {hdmi_out_data[21]}]
```

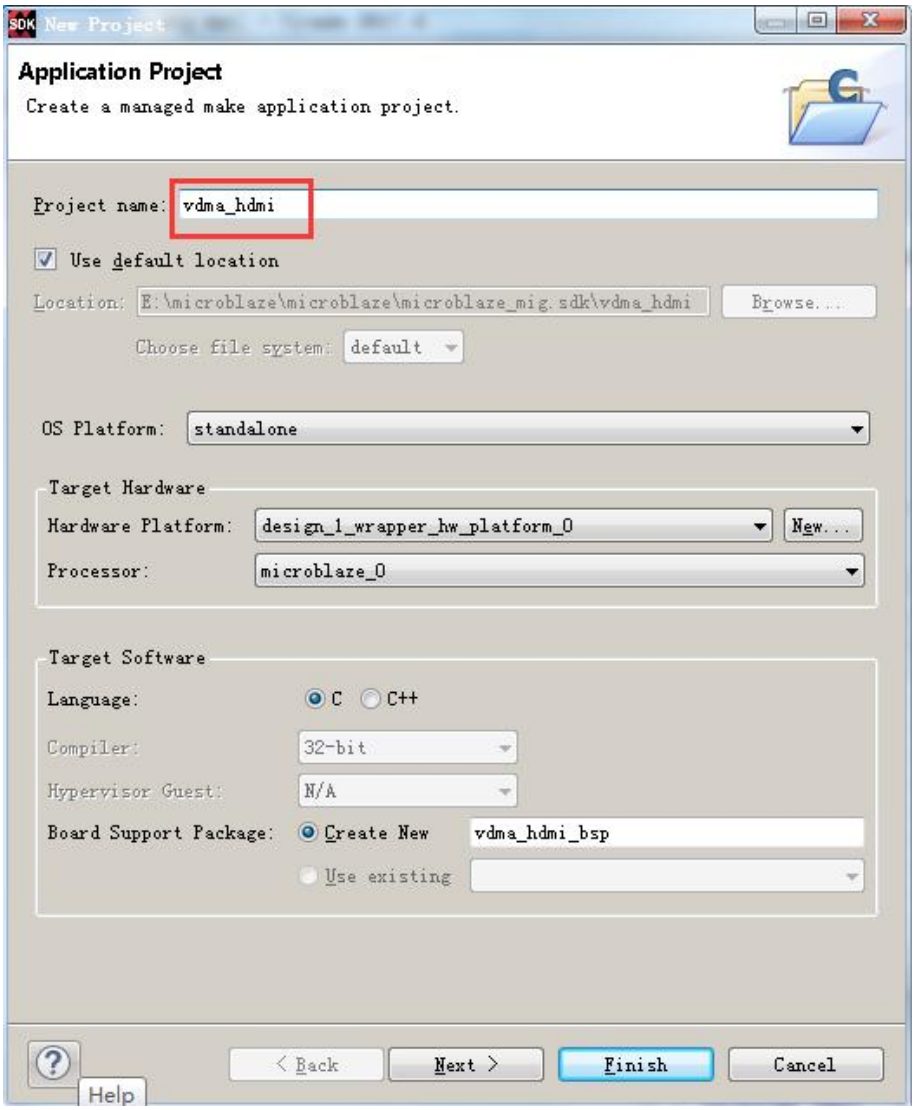
```
set_property PACKAGE_PIN U20 [get_ports {hdmi_out_data[22]}]
set_property PACKAGE_PIN V19 [get_ports {hdmi_out_data[23]}]
set_property PACKAGE_PIN U22 [get_ports hdmi_out_de]
set_property PACKAGE_PIN T18 [get_ports hdmi_out_hs]
set_property PACKAGE_PIN R18 [get_ports hdmi_out_vs]

set_property PACKAGE_PIN H13 [get_ports hdmi_i2c_scl_io]
set_property PACKAGE_PIN G13 [get_ports hdmi_i2c_sda_io]
set_property PULLUP true [get_ports hdmi_i2c_scl_io]
set_property PULLUP true [get_ports hdmi_i2c_sda_io]
set_property PACKAGE_PIN L18 [get_ports {hdmi_rst_n_tri_o[0]}]


set_property IOSTANDARD LVCMOS33 [get_ports hdmi_out_clk]
set_property IOSTANDARD LVCMOS33 [get_ports {hdmi_out_data[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_out_de]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_out_hs]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_out_vs]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_i2c_scl_io]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_i2c_sda_io]
set_property IOSTANDARD LVCMOS33 [get_ports {hdmi_rst_n_tri_o[0]}]

set_property SLEW FAST [get_ports {hdmi_out_data[*]}]
set_property SLEW FAST [get_ports hdmi_out_de]
set_property SLEW FAST [get_ports hdmi_out_hs]
set_property SLEW FAST [get_ports hdmi_out_vs]


#set_property DRIVE 24 [get_ports {hdmi_out_data[*]}]
#set_property DRIVE 24 [get_ports hdmi_out_de]
#set_property DRIVE 24 [get_ports hdmi_out_hs]
#set_property DRIVE 24 [get_ports hdmi_out_vs]
#set_property DRIVE 24 [get_ports hdmi_out_clk]
#set_property SLEW FAST [get_ports hdmi_out_clk]
```

29)Compile and generate **bit** file, then export hardware information, start **SDK**.

Part 9.2: SDK software writing and debugging

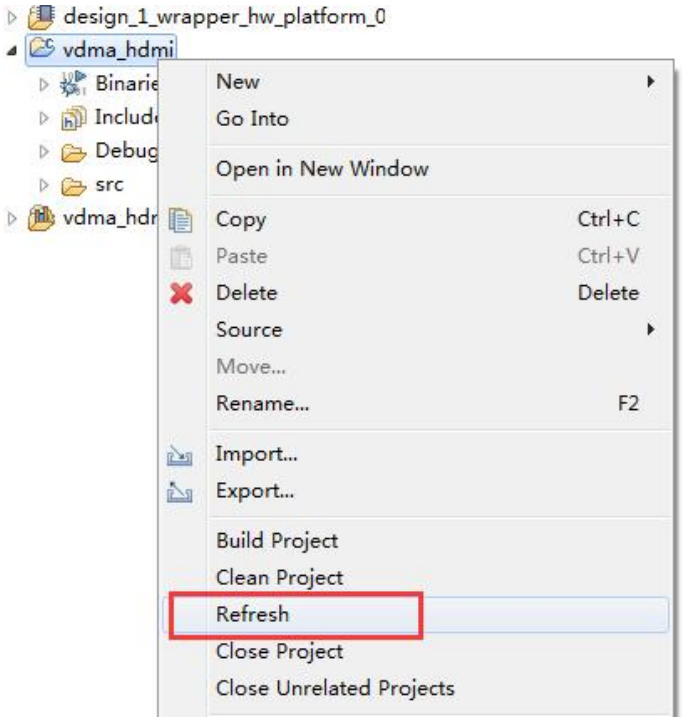
Part 9.2.1: SDK new project

30)Create a new **APP**, name the project "**vdma_hdmi**", select "**Hello World**" as the template.



31)Due to the large number of program files, we will not introduce it in detail, and copy the source code of the routine directly. Delete the files in the **src** directory and use the **src** directory file of the routine instead.

32)Refresh under **SDK**



Part 9.2.2: Program Description

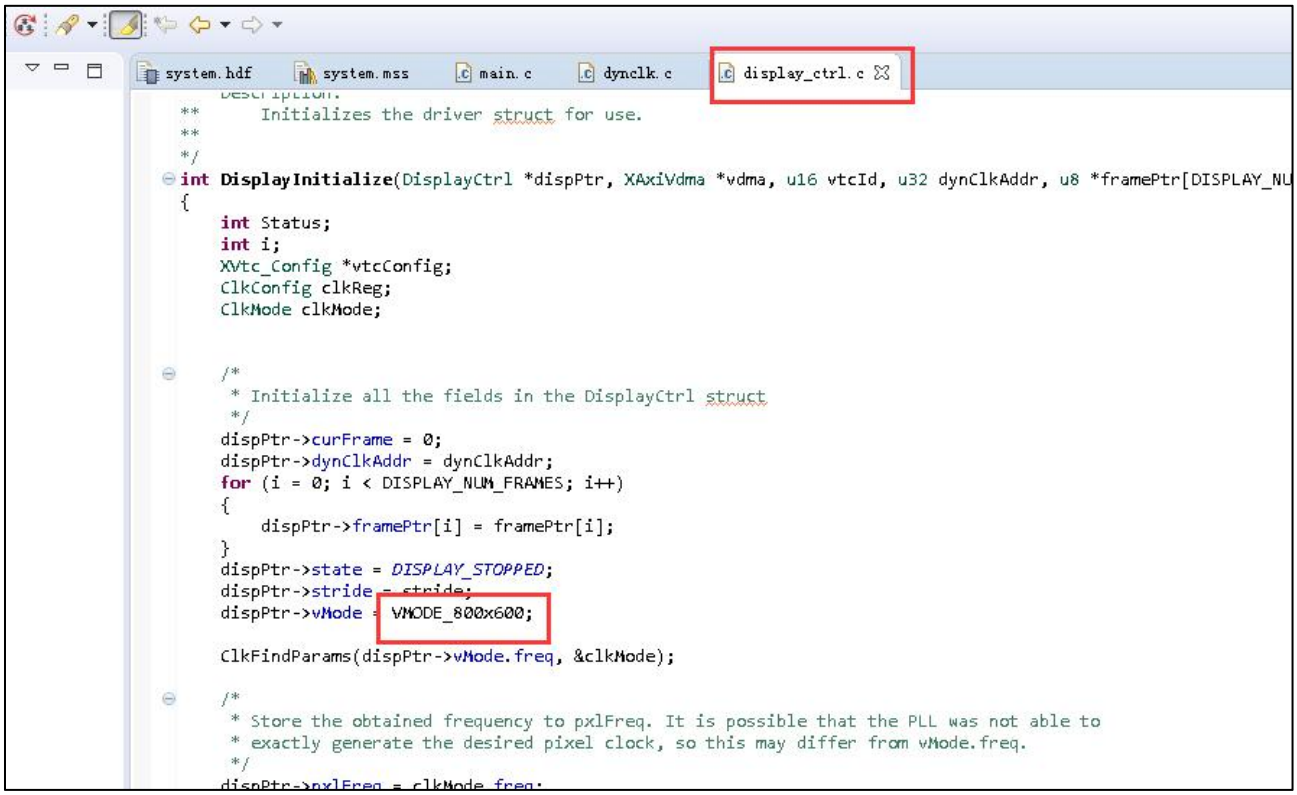
Let's take a look at the **main.c** program first. In the front of the program, we define a **DisplayCtrl** structure variable **dispCtrl**. The **DisplayCtrl** structure is defined in the **display_ctrl.h** file. The **DisplayCtrl** structure contains the base address of the **dynclk IP**, **vdma** structure, **vtc** structure, **video mode**, display image **buffer** pointer, etc. information.

First, initialize the **GPIO** in the main function and generate a reset signal to reset the **HDMI** chip.

Then call I2C to configure the registers of the HDMI chip.

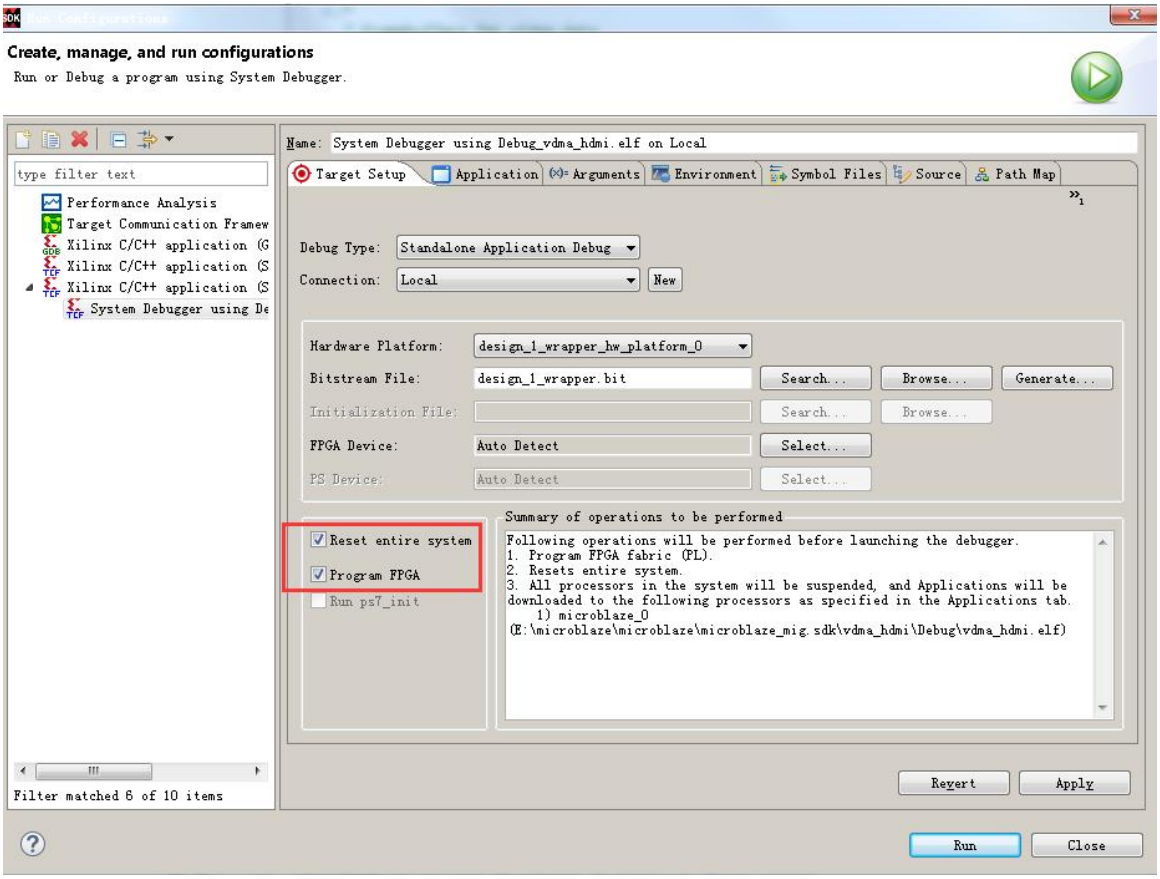
Then initialize the vdma device, then use the DisplayInitialize function to initialize the display device dispCtrl structure, and then call the DemoPrintTest function to display the Video image. The last parameter pattern of the DemoPrintTest function is used to indicate the displayed image. The default DEMO_PATTERN_0 is to display the pictures in the pic_800_600.h file. If the parameter pattern is DEMO_PATTERN_1, the grid will be displayed; if the parameter pattern is DEMO_PATTERN_2, it will display the horizontal gray gradient; if the parameter pattern is DEMO_PATTERN_3, it will display the color bars of 8 colors, which can be modified by the user.

In addition, the output video resolution is defined in the display_ctrl.c file. The user can modify this parameter to change the output image with different resolutions.

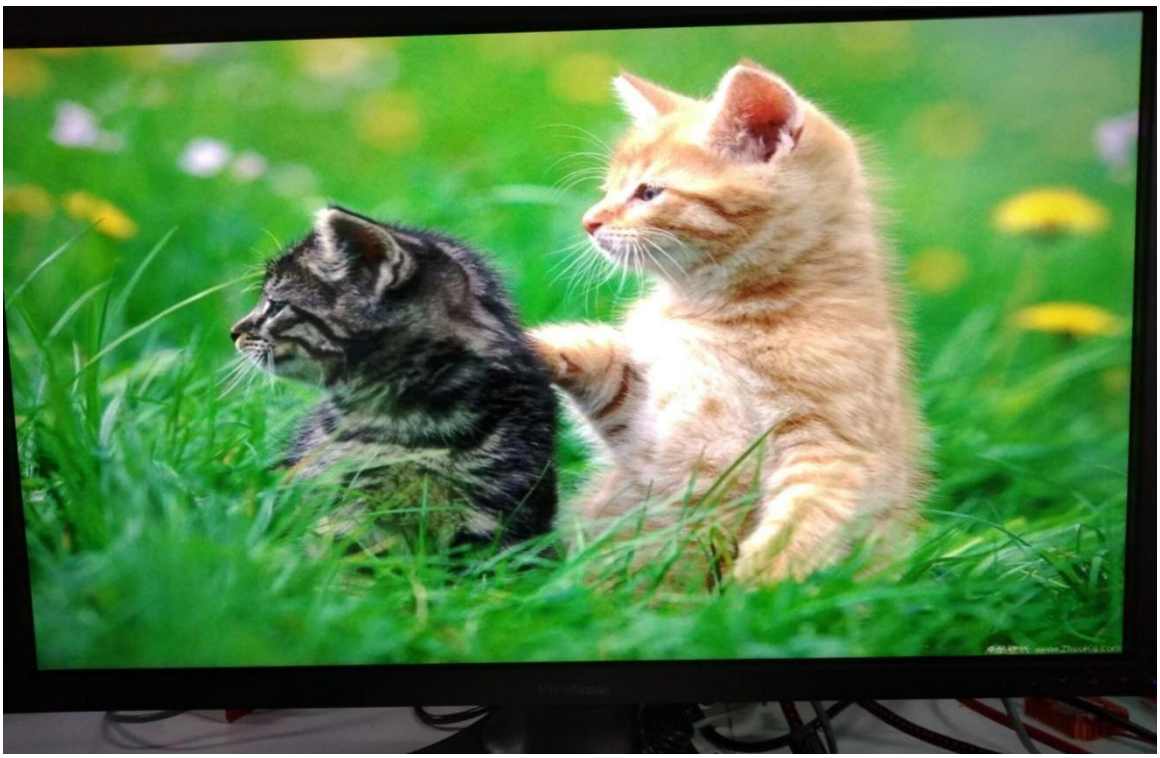


Part 9.2.3: Debugging

33)Connect the HDMI output port to the monitor, compile and run



34)HDMI display shows a picture



Part 9.3: Experiment Summary

The resolution output used in this experiment is an image of 800*600, and users can use tools (such as Image2Lcd) to generate other resolution images for display. In addition, the FPGA development board is ax7035 or ax7050, and there is no HDMI chip (Si9134) on the board, so you need to add an RGB to DVI IP (rgb2dvi_v1_3) in the vivado project, and delete the I2C and GPIO of the HDMI configuration.