Xilinx Artix-7 M.2 PCIE Accelerator Card Example Guide (V1.1)

May 13, 2024

Contact Details

Official Taobao store: GeekTang Electronics

Official Xianyu: ITANGTANGI

TEL: 13755322372 (same number on WeChat)

Document Revision History

2023.03.13 (V1.0)

Create files and directories, add basic tutorial, Riffa PCIE, XDMA PCIE and optical communication IBERT routines.

2023.03.13 (V1.01)

Change the Vivado2021.1 download link to Baidu Netdisk.

2024.05.13 (V1.1)

Modify and add Winbond Flash support method, change it to copy directly from txt file, add some IO baseboard and adapter board information.

Table of contents

Document Revision History.....I

Contents.....I

Chapter 1 Basic Introduction and Software Preparation.....1

1.1 Hardware Introduction.....1

1.2 Software Installation.....2

1.2.1 Vivado Software Installation.....2

1.2.2 XDMA supporting WDK, WSDK, VS2015 installation (compile driver and application) 7

1.2.3 Qt Software Installation (used by Riffa, can also be used by XDMA)...9 1.3 Program Solidification..... 11

1.3.1 Adding support for winbond W25Q128..... 11 1.3.2 Flashing the Vivado program to Flash..... 11

Chapter 2 PCIE XDMA Test.....15

2.1 XDMA Introduction.....15

2.2 XDMA Features.....15

2.3 XDMA test project creation.....16

2.4 XDMA IP Configuration.....18

2.4.1 Basic Items.....18

2.4.2 PCIE ID Configuration.....19

2.4.3 PCIE BAR Configuration.....20

2.4.4 PCIE Interrupt Configuration.....21

2.4.5 PCIE DMA Configuration.....22

2.4.6 Remaining Configuration.....24

2.5 Generate Bin File and Burn.....28

2.6 XDMA driver and application compilation (optional).....30

2.7 Preparation for the on-the-job test.....32

2.8 xdma_rw BRAM read and write test.....35 2.9 simple_dma simple speed test.....37 2.10 xdma_test basic test.....37 2.11 Postscript.....38

Chapter 3 Riffa Data Loopback.....39

3.1 Riffa Overview.....39

3.2 RIFFA Architecture.....40

3.3 RIFFA Driver Installation.....42

3.4 RIFFA Library Usage.....44
3.5 RIFFA Vivado Project Construction.....45
3.6 PCIE IP Configuration.....48
3.6.1 Basic 48
3.6.2 IDs.....49
3.6.3 BARs.....50
3.6.4 Core Capabilities.....50
3.6.5 Link Registers 51
3.6.6 Interrupts 52
3.6.7 Power Management.....53
3.6.8 Ext Capabilities.....54
3.6.9 Ext Capabilities-2.....55
3.6.10 TL Settings 55
3.6.11 DL&PL Settings.....56
3.6.12 Shared Logic 56
3.6.13 Core Interface Parameters.....57
3.6.14 Add.Debug Options.....57 3.6.15
Soft Reset Configuration.....59 3.6.16
PCIE Channel Modification.....59
3.7 Adding Riffa RTL source files and constraint files.....60
3.8 Computer Test.....65
3.8.1 Flash Burning.....65
3.8.2 Installing the Accelerator Card.....65
3.8.3 Qt Software Engineering Compilation and Runtime Tests.....66
3.9 Postscript.....67
Chapter 4 Optical Communication IBERT Test.....68
4.1 Hardware Preparation.....68
4.2 Creating a Vivado Test Project.....69
4.3 Ibert test.....75

Chapter 1 Basic Introduction and Software Preparation

1.1 Hardware Introduction

The M.2 M-Key FPGA accelerator card based on the Xilinx Artix-7 series FPGA chip design leads to 4 high-speed GTs of the Artix7-484-pin chip, supporting up to PCIE2.0*4 speed. The high-power 12A core power supply design can support

Artx7-XC7A35T, Artx7-XC7A50T, Artx7-XC7A75T, Artx7-XC7A100T and Artx7-XC7A200T

Chip. The front of the accelerator card is shown in Figure 1-1.

Figure 1-1 Front view of the M2 PCIE accelerator card The

onboard hardware resources of the accelerator card are shown in Figure 1-2.

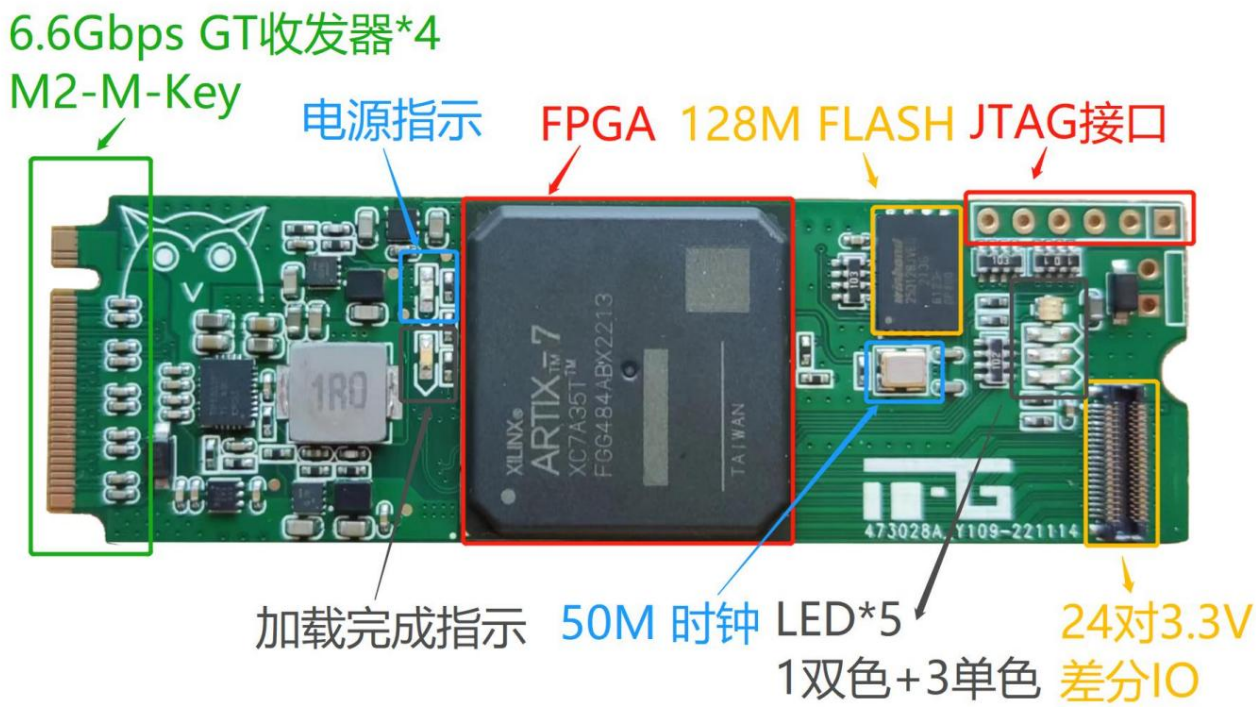

Figure 1-2 Accelerator onboard hardware resources

Since this PCIE accelerator card only leads out the GT transceiver in the form of an M2 interface, it can be connected through the M2 interface socket

The sub-adapter can be used to connect different types of application boards, not limited to PCIE applications, such as SFP, USB3.0 or HDMI.

Therefore, we designed and manufactured a 4-way SFP optical communication baseboard as shown in Figure 1-3, which can be used for optical communication testing or

Power supply for FPGA accelerator card.

Figure 1-3 Optical communication baseboard

1.2 Software Installation

The software versions used in this guide

are: FPGA Development: Vivado 2021.1

PC Windows software development (XDMA): WDK, WINSDK, VS2015 PC Windows

software development (Riffa): Qt5.9.9 Development

platform: Windows10

1.2.1 Vivado Software Installation

Vivado Design Suite is Xilinx's comprehensive FPGA development software that can complete the design input

This section will teach you how to install Vivado software.

Vivado 2021.1 download link:

Link: https://pan.baidu.com/s/15fkuISY8JP5FxbqwIgl8jg Extraction code: 4q2r

Download the

compressed package named Xilinx_Unified_2021.10610,2318.tar. The installation process is as follows: Unzip

the compressed package (note that the path name of the unzip directory can only contain letters, numbers, and underscores.

Double-click "xsetup.exe" in the unzipped folder, as shown in Figure 1-4:

Figure 1-4 Double-click "xsetup.exe" to install

Click "Next" on the welcome screen to enter the agreement license screen. Select all three check boxes and click Next to enter the version selection.

Select the top item Vitis here, which includes all development tools, as shown in Figure 1-5.

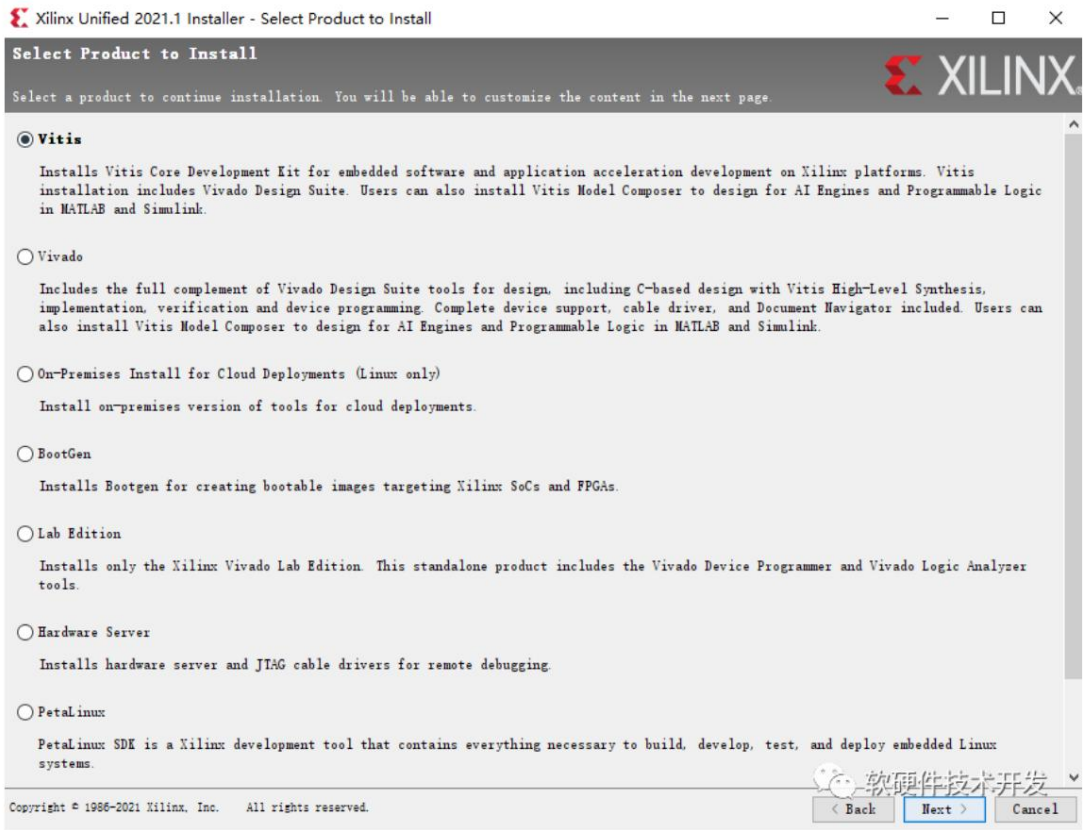

Figure 1-5 Layout selection

The next step is to select tool components and device libraries. This accelerator card only needs 7Series, but you can choose according to your own needs.

To save storage space, you can remove unused tool components and device libraries, as shown in Figure 1-6.

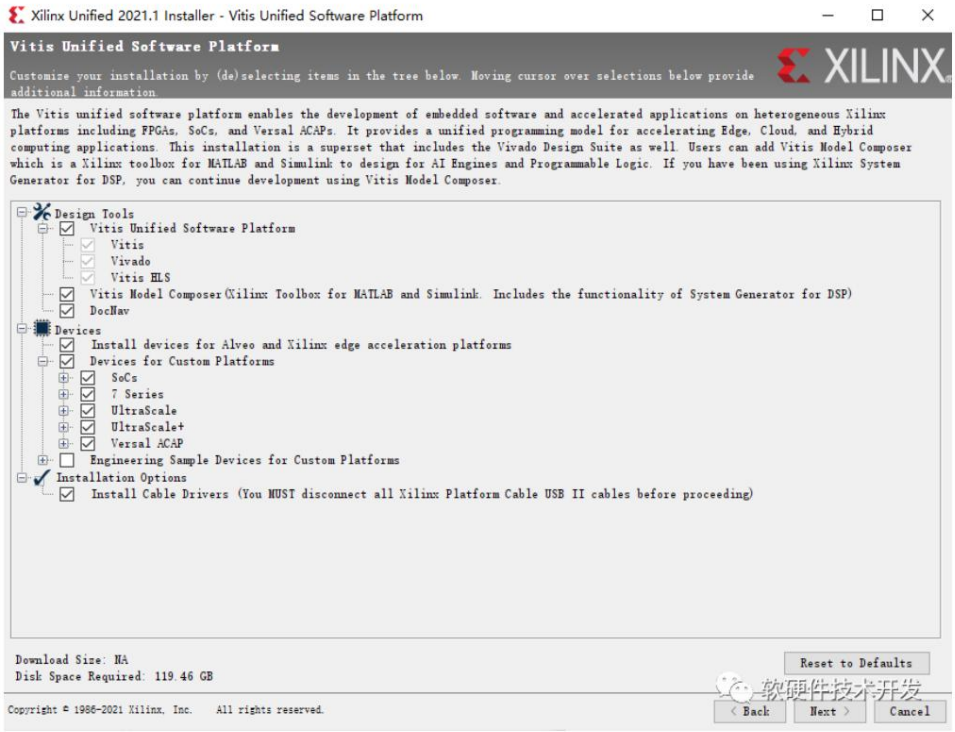

Figure 1-6 Device library and component selection

Click "Next" to enter the installation directory setting page, as shown in Figure 1-7.

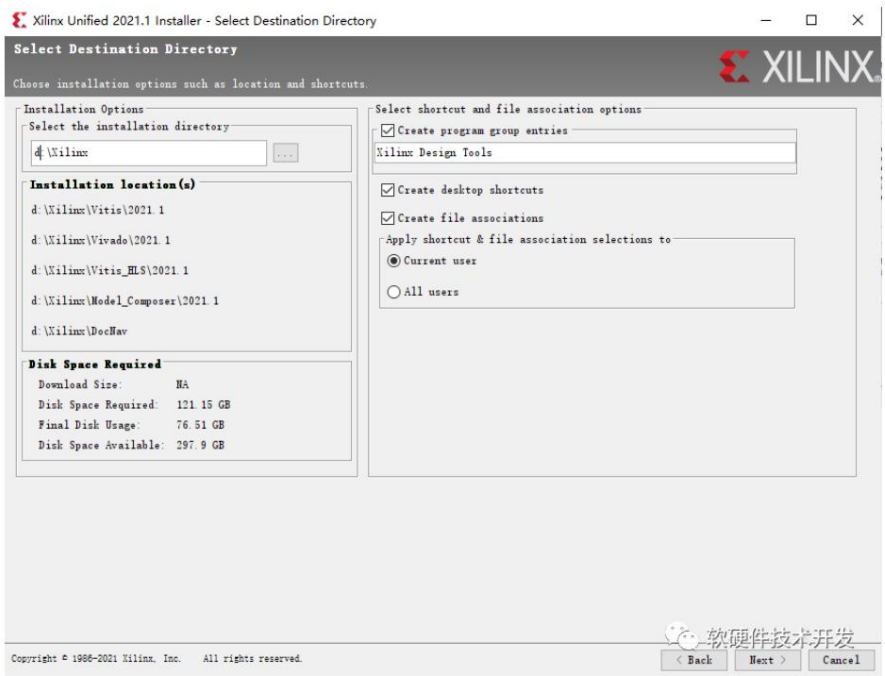

Figure 1-7 Select the installation directory and create a shortcut

Click "Next" and the Summary interface will appear, as shown in Figure 1-8. This interface summarizes all the previous installation configuration information.

After confirmation, click “Install” to start the installation of the Vivado Design Suite.

As shown in Figure 1-9: (During the installation, Vivado will occupy a large amount of computer CPU resources and memory resources.

It is recommended that you close other unnecessary application software on your computer before starting the installation.)

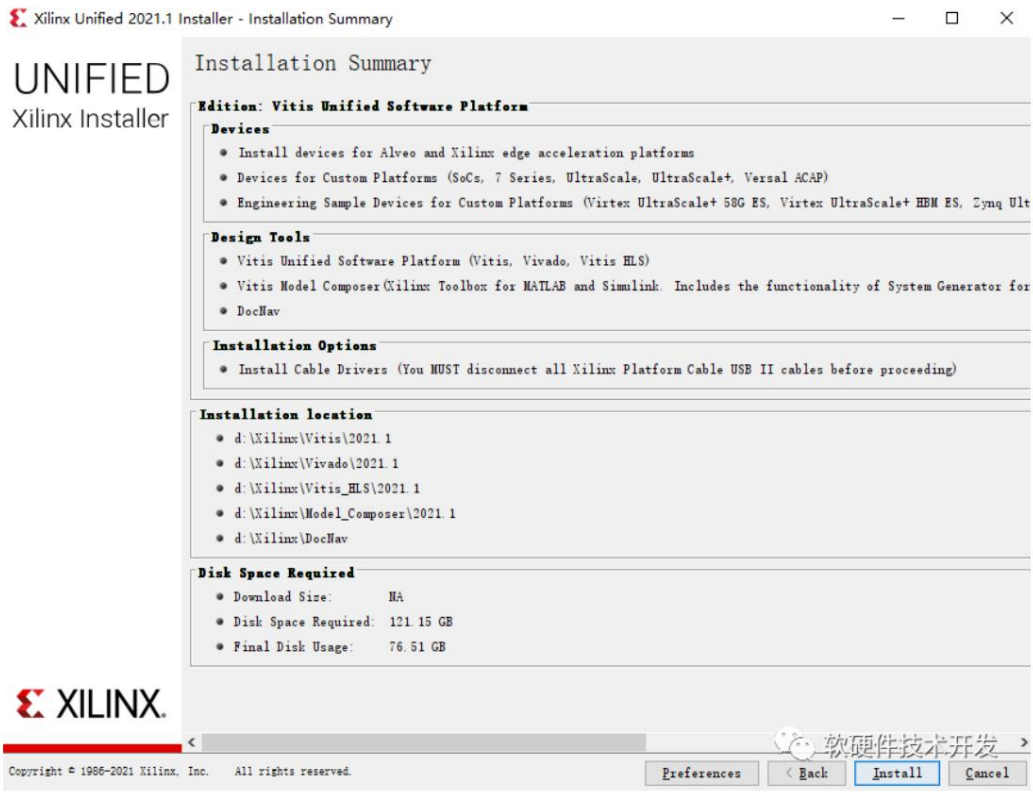

Figure 1-8 Installation information summary

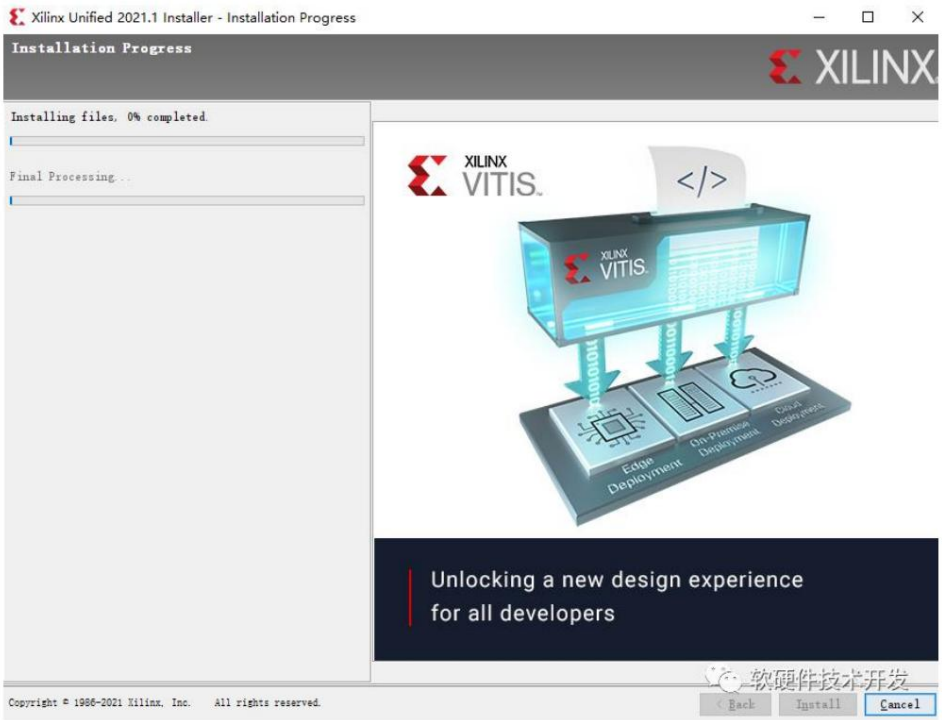

Figure 1-9 Installation process

During the installation process, the PinPcap installation interface will pop up. Just click "Next" to complete the installation, as shown in Figure 1-10.

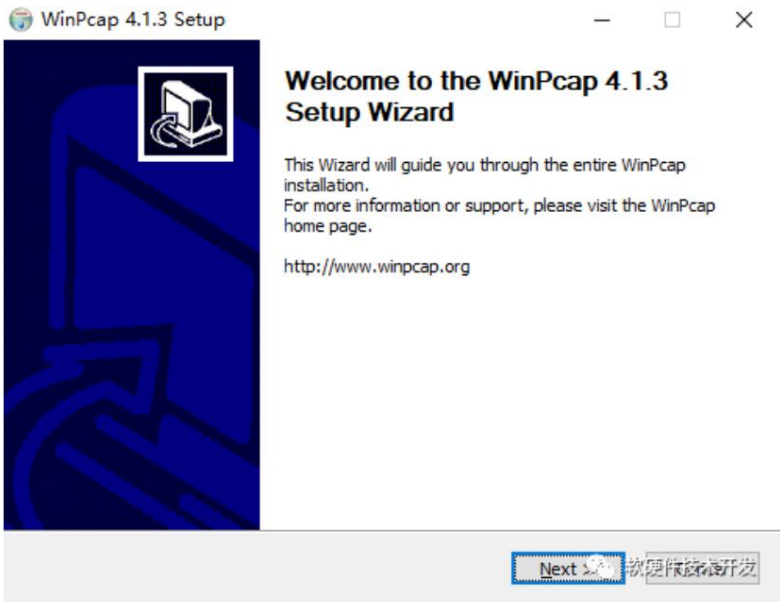

Figure 1-10 WinPcap installation

When the installation is complete, a prompt will appear, as shown in Figure 1-11. At the same time, the "Vivado License Manager" window will pop up. Click "Copy license" in the window. Select the provided license file in the resource manager and load it, as shown in Figure 1-12. The installation of the Vivado software is now complete.

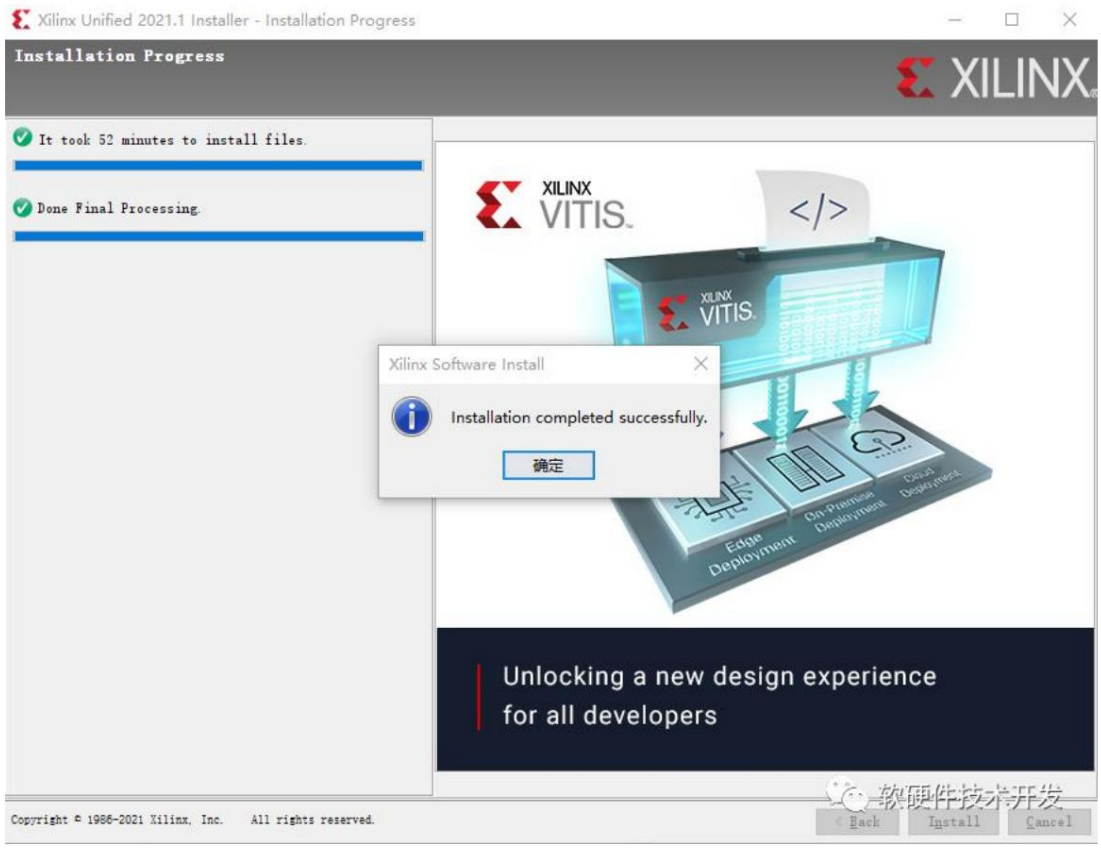

Figure 1-11 Vivado installation completed

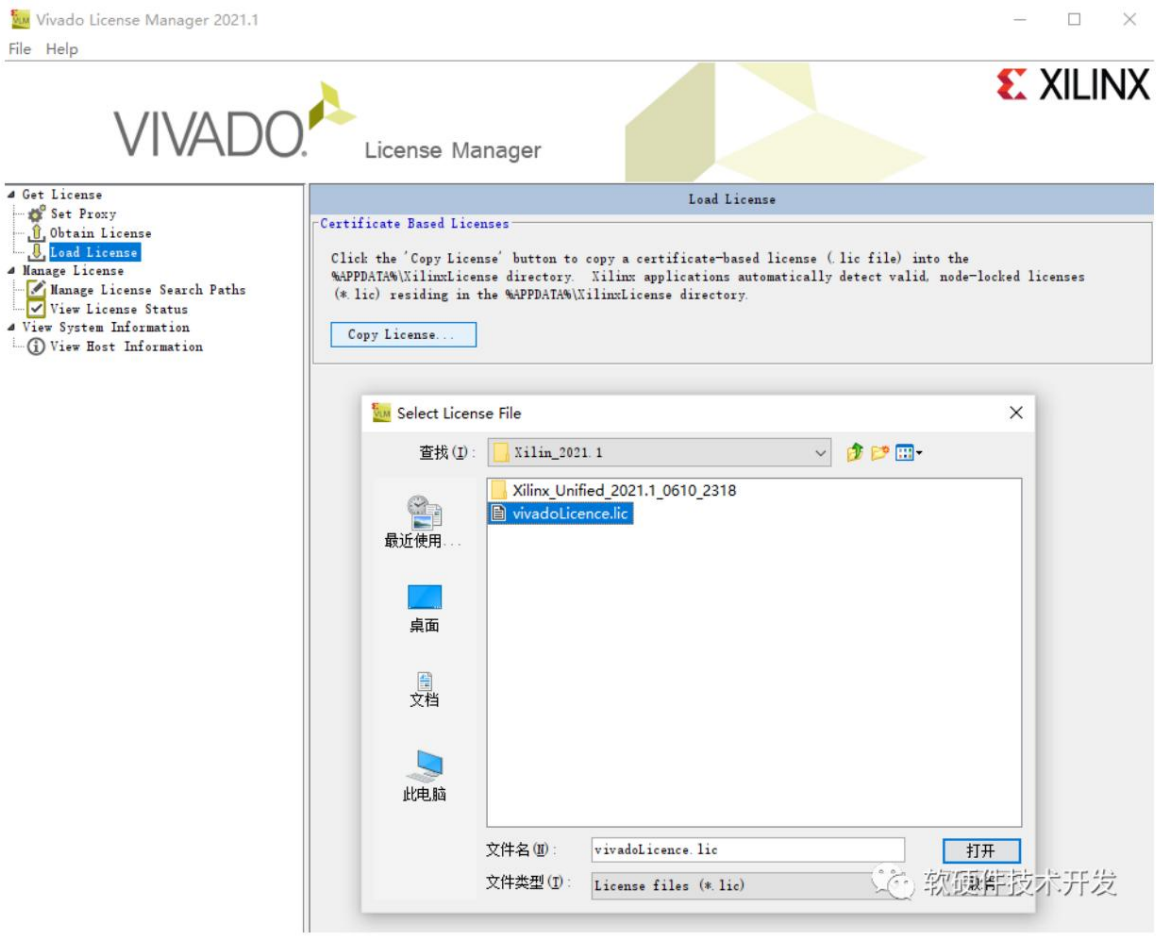

Figure 1-12 Loading the license file

1.2.2 XDMA supporting WDK, WSDK, VS2015 installation (compile driver and application)

WDK, WSDK, VS2015 download address:

Link: https://pan.baidu.com/s/1nIOSw68EwMelhI34WZWEDA Extraction code: 4f2p Please note that the

software installation

order must be to install vs2015 first, then install WINSDK, and finally install WDK.

If you accidentally get the order wrong, uninstall it cleanly and then reinstall it, otherwise it will definitely not succeed.

First, install VS2015. If you have already installed VS2010 or other versions, it is recommended to uninstall them first.

During the uninstallation process, do not use 360 or other software tools to uninstall, but use the uninstallation tool that comes with VS (Note: the operation process

Problems may occur in the system, which may prevent the system from compiling the program anyway. This tutorial is not responsible for this situation.

Double-click to open the VS2015 software installation package. As shown in Figure 1-13, select Custom Installation and then check Visual C++

option and Visual Studio 2015 Update 3 to avoid compilation errors. The error is basically due to version mismatch.

After that, just click Next and agree to the relevant agreements to complete the installation. After the installation is complete, fill in the registration

The serial number is: HM6NR-QXX7C-DFW2Y-8B82K-WTYJV. Complete the installation of VS2015.

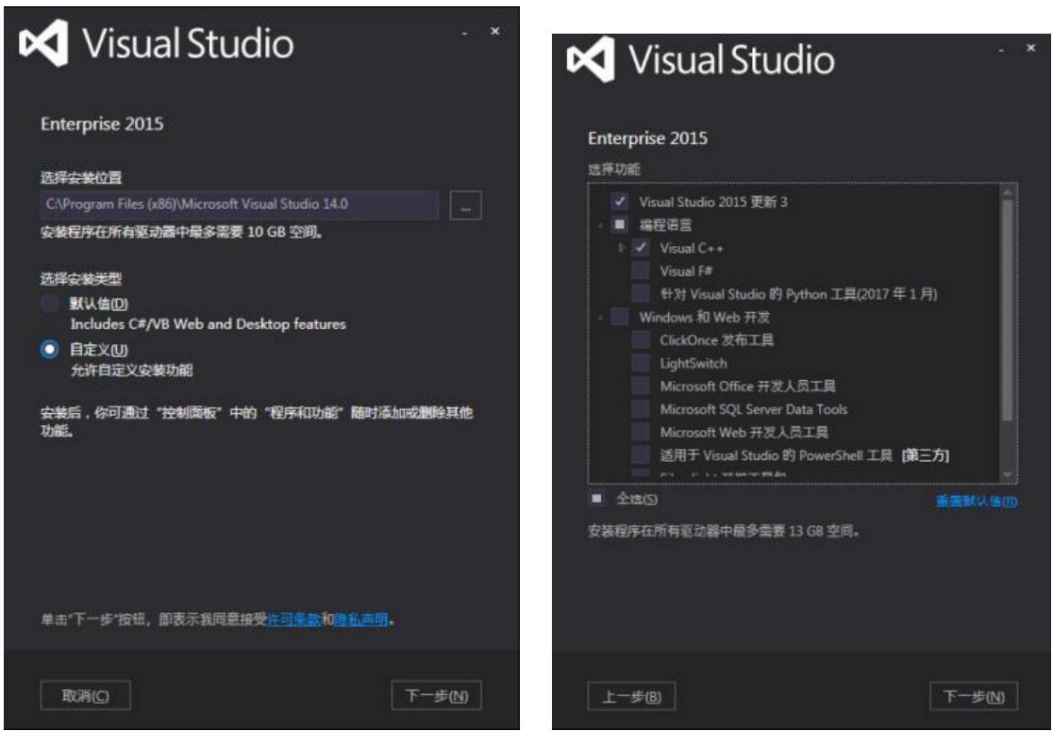

Figure 1-13 VS2015 installation configuration

The installation interfaces of WSDK and WDK are shown in Figure 1-14 and Figure 1-15 respectively. No configuration is required. Just follow the default settings.

Just click Next for configuration.

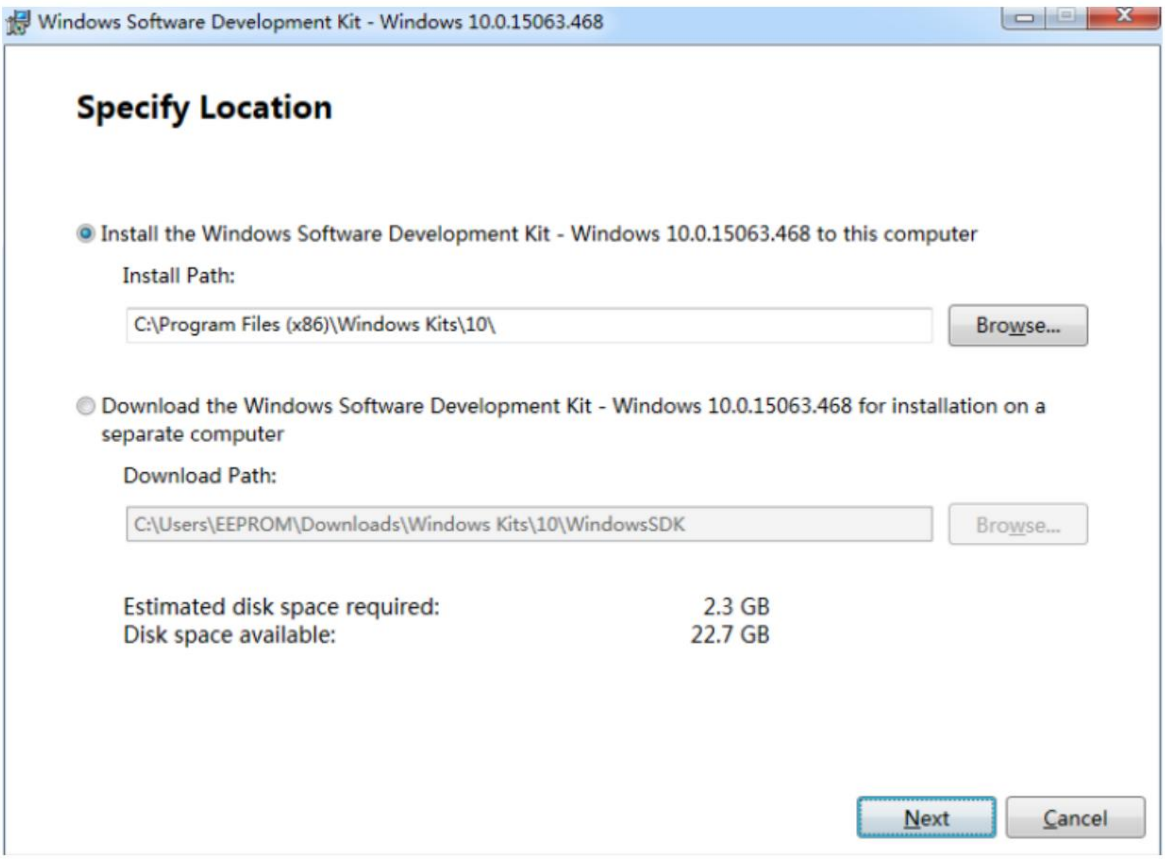

Figure 1-14 WSDK installation interface

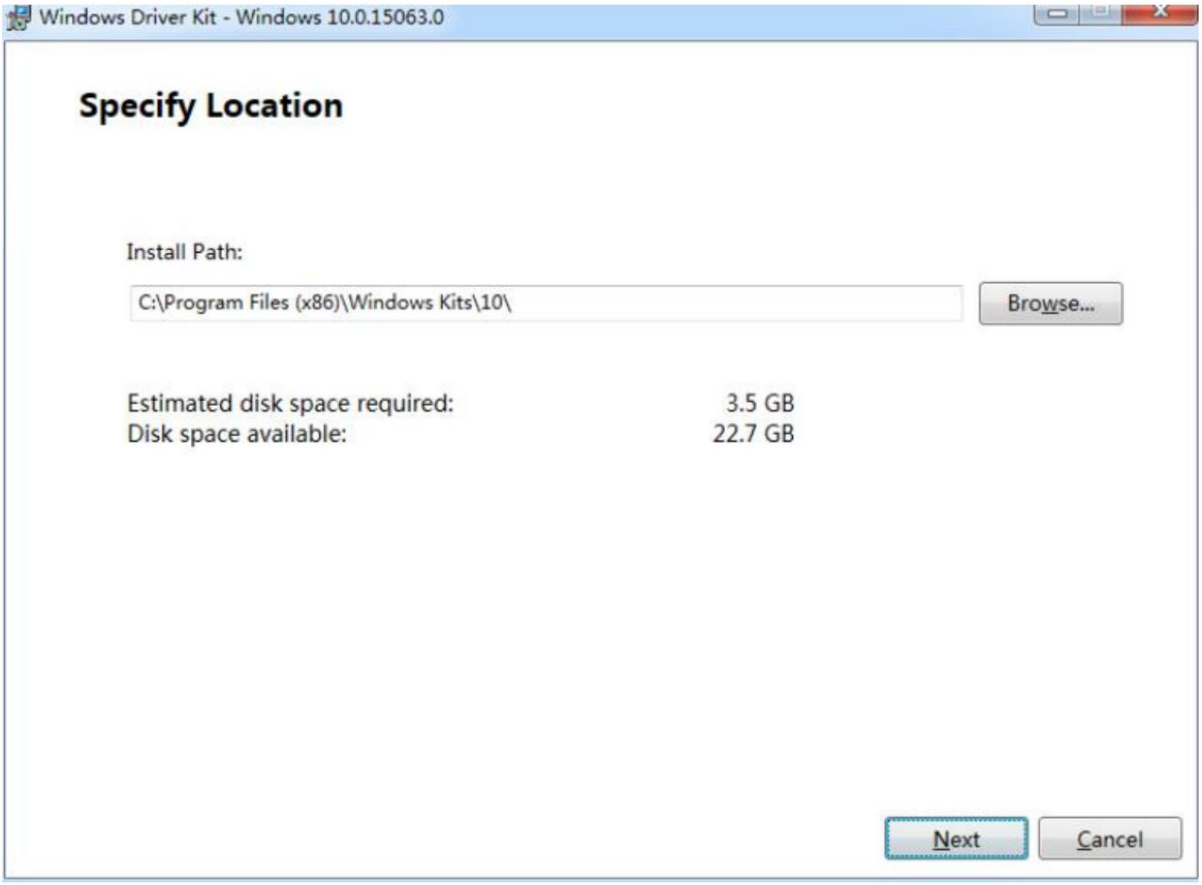

Figure 1-15 WDK installation interface

This completes the installation of the PC software required for XDMA. The environment configuration will be given in Chapter 2.

1.2.3 Qt software installation (used by Riffa, XDMA can also be used)

Qt software is used by Riffa to develop PC-side software engineering to interact with the accelerator card. Its download address is:

https://download.qt.io/archive/qt/5.9/5.9.9/ Select qt-opensource-windows-

x86-5.9.9.exe to download. After the download is complete, close the computer's network connection

first, otherwise Qt will ask you to log in to your account when installing. Double-click the software to install

The package opens, and click “Next” on the welcome screen to enter the configuration interface. Select the installation location as shown in Figure 1-16.

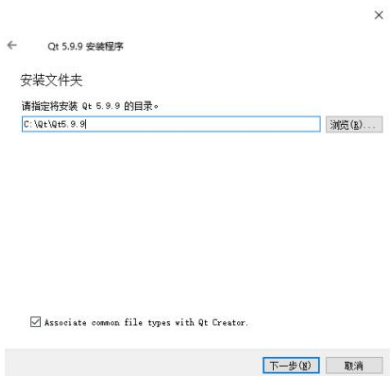

Figure 1-16 Qt selects the installation location

Check the MinGW 5.30 development components and Qt Creator components as shown in Figure 1-17, then click Next.

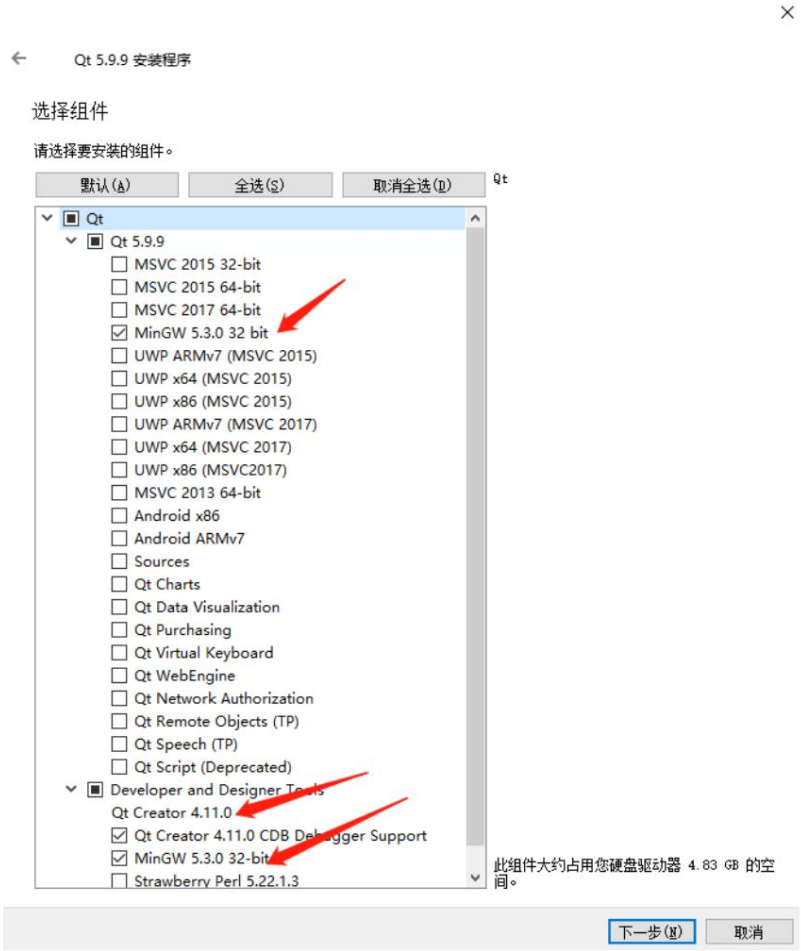

Figure 1-17 Installation component selection

As shown in Figure 1-18, select Agree to the license and confirm the subsequent installation information, and click Next to complete the installation.

Figure 1-18 Agree to the license agreement

1.3.1 Add support for winbond W25Q128

Reference link: https://xilinx.eetrend.com/blog/2021/100112739.html Since the FLASH model used on this accelerator card to store the FPGA boot program is Winbond's W25Q128, Vivado2021.1 software does not support it by default. You need to make changes in the Xilinx Vivado software database. The specific operation is as follows.

Use text editing software (such as Notepad++) (do not use **Excel** or WPS) to open **the installation directory\Xilinx\Vivado\2021.1\data\xicom\xicom_cfgmem_part_table.csv** file **and then open the data\EXAMPLES\ winbond FLASH add content.txt** file

As shown in Figure 1-19, copy the second line of the txt file to the last line of the csv file to obtain support.

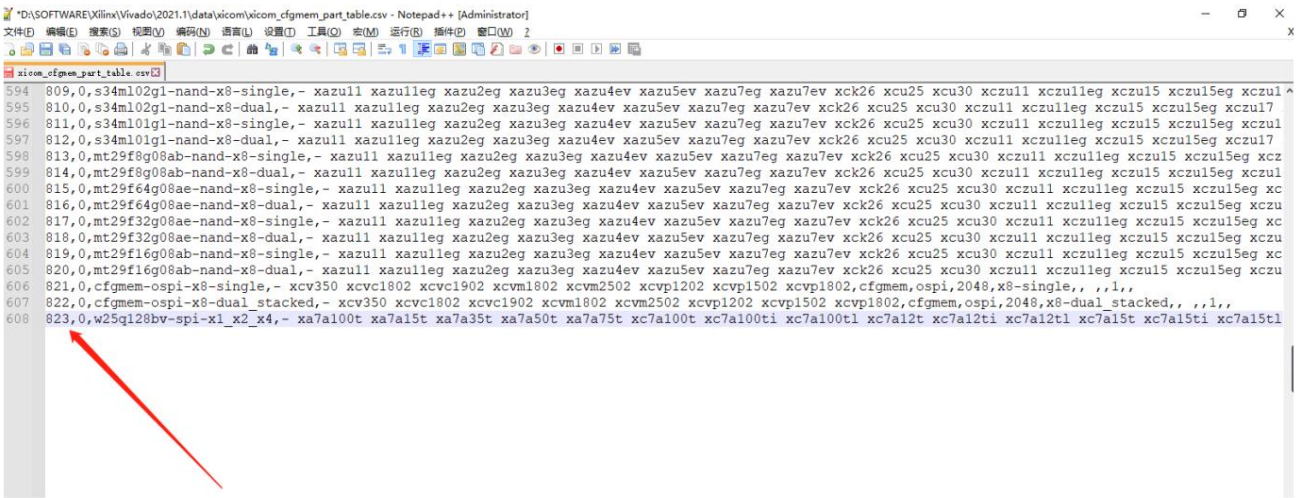

Figure 1-19 Add Winbond W25Q128 support statement

After adding, save the CSV file. If Vivado is running, you need to close and restart Vivado.

Note: If the Vivado version used is not Vivado2021.1, you need to select the txt file according to the Vivado version. Paste the corresponding line in the CSV file and modify the leftmost number to connect with the previous line number.

1.3.2 Vivado firmware to Flash

When the project is confirmed to be correct, right-click on the Generate Bitstream option and select Bitstream Setting, check the -bin_file option as shown in Figure 1-20, and click Generate Bitstream again to generate the bin file used to burn the Flash.

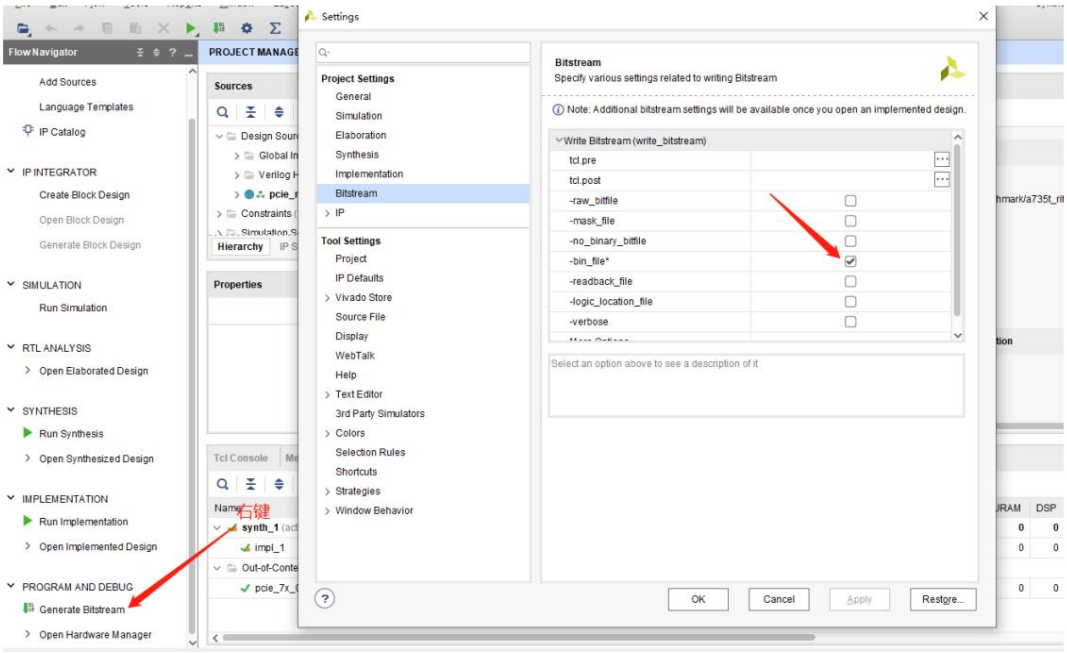

Figure 1-20 Generate bin file for burning Flash

After the development board is connected to JTAG, power it on, as shown in Figure 1-21. In Hardware Management, click Open Target. Select the Auto Connect option to connect to the FPGA.

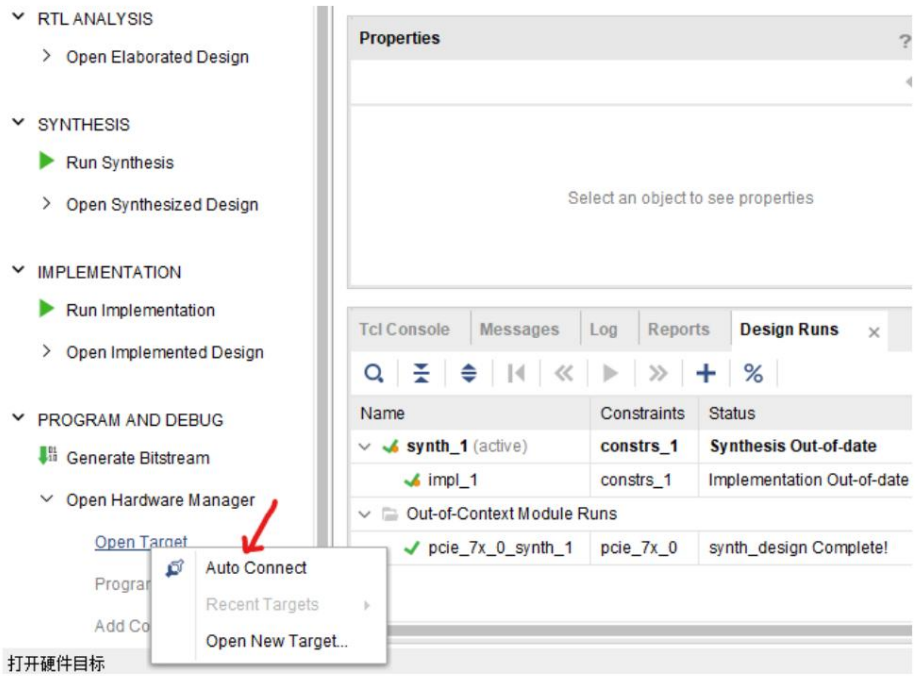

Figure 1-21 Connecting to FPGA

After connecting to the FPGA, right-click and select Add Configuration Memory Device, as shown in Figure 1-22.

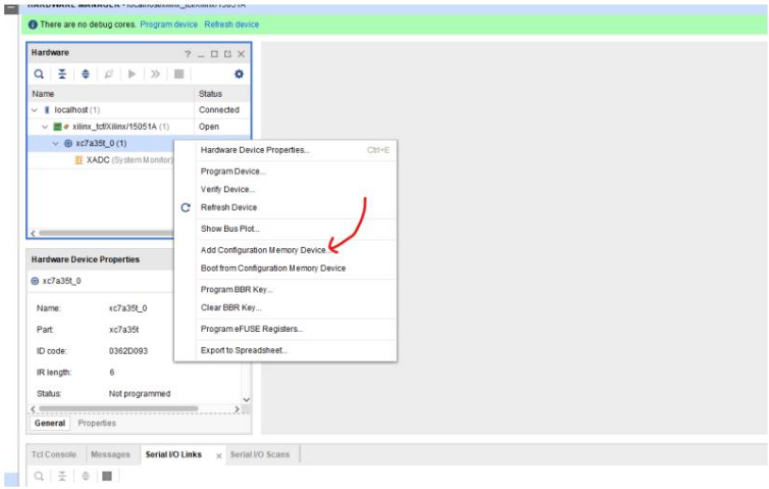

Figure 1-22 Adding a configuration storage device

In the pop-up interface, enter w25q in the search box, and the W25Q128 chip will appear as shown in the figure. Select it and click OK

Complete the selection, as shown in Figure 1-23. Right-click the selected memory chip and select Program memory device.

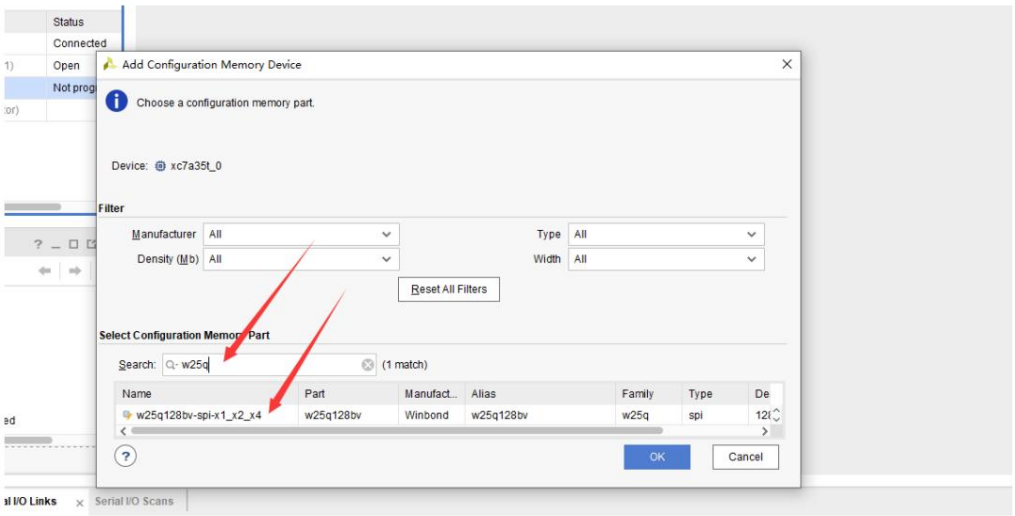

Figure 1-23 Select W25Q128 chip

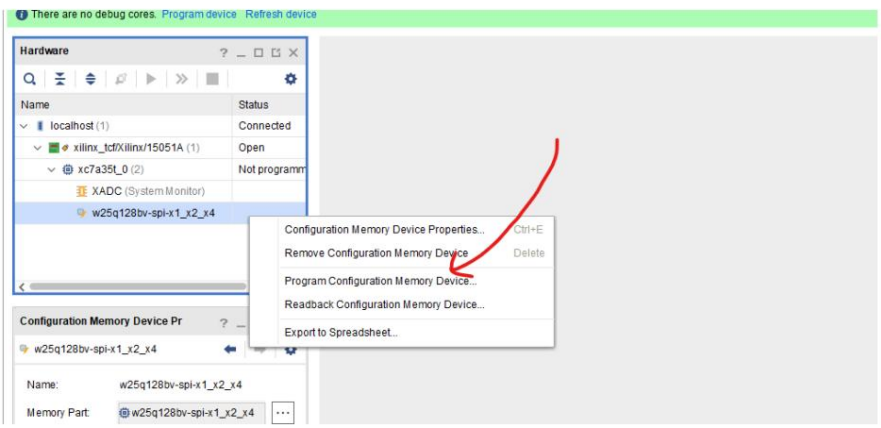

Figure 1-24 Select the memory to be burned

In the pop-up interface, select the project name.runs/impl_1/xx.bin file, and then click OK to start burning, as shown in Figure 1-25.

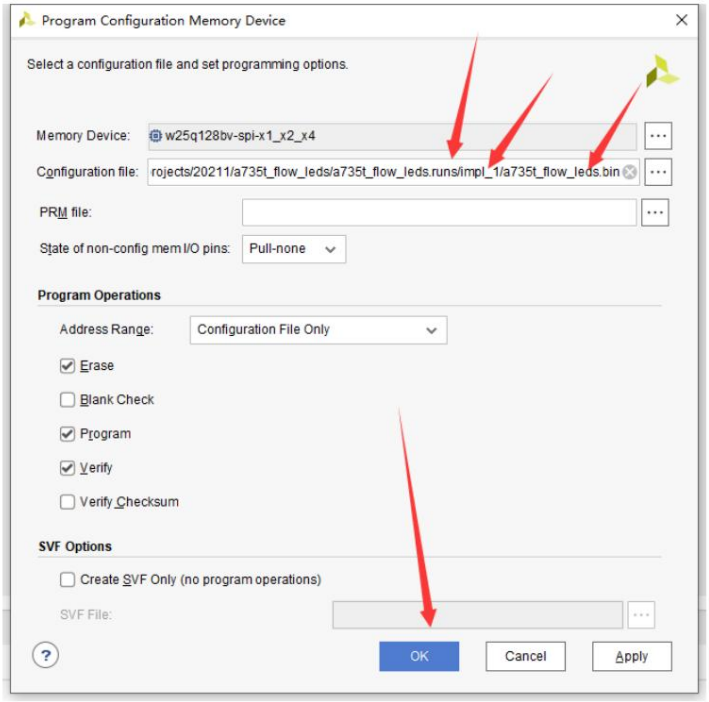

Figure 1-25 Select bin file to burn

Then the Flash will be burned until it is completed, as shown in Figure 1-26.

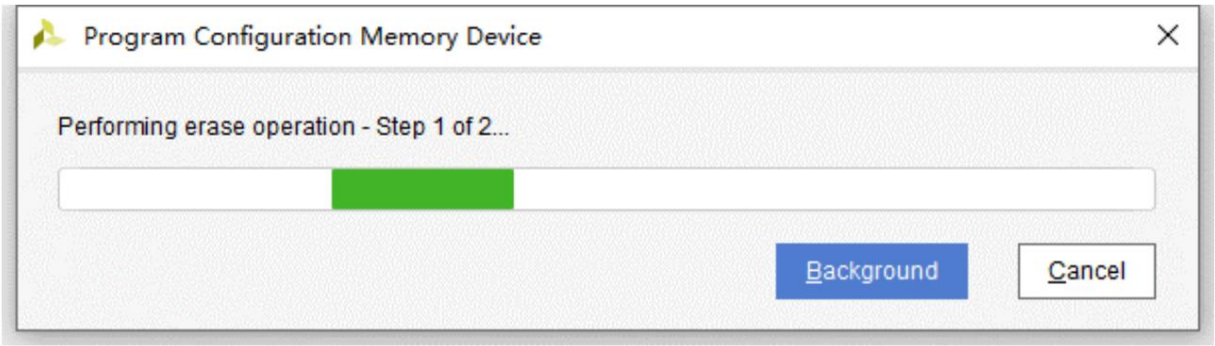

Figure 1-26 Burning process

After power failure and restart, FPGA will automatically load the program from Flash and run it.

Chapter 2 PCIE XDMA Test

2.1 Introduction to XDMA

The DMASubsystem for PCIExpressIP provided by Xilinx is a high-performance, configurable SG-mode DMA for PCIE2.0 and PCIE3.0, providing user-selectable AXI4 interface or AXI4-Stream interface.

Generally, it is configured as an AXI4 interface and can be added to the system bus interconnection, which is suitable for asynchronous transmission of large amounts of data. DDR is usually used, and the AXI4-Stream interface is suitable for low-latency data stream transmission.

XDMA is SGDMA, not Block DMA. In SG mode, the host will organize the data to be transmitted into a linked list. The first address of the linked list is sent to XDMA through BAR. XDMA will

The structure of XDMA is shown in Figure 2-1.

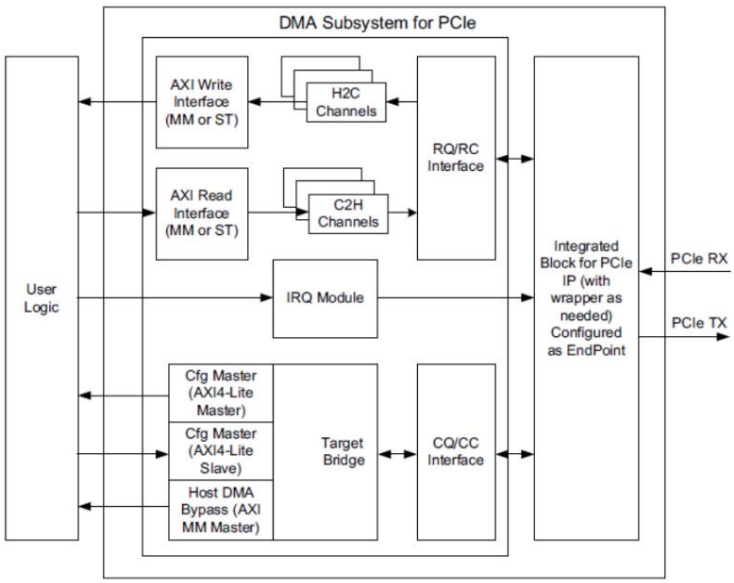

Figure 2-1 XDMA structure diagram

2.2 XDMA Features

XDMA provides the following interfaces:

You must select one of AXI and AXI4-Stream for data transmission.

AXI4-Lite Master, optional, used to implement the mapping of PCIE BAR address to AXI4-Lite register address, Can be used to read and write user logic registers.

AXI4-Lite Slave, optional, used to open the XDMA internal registers to the user logic. The XDMA internal registers are accessed through this interface and are not mapped to the BAR.

AXI4 Bypass interface, optional, used to implement PCIE direct user logic access, can be used for low-latency data transmission lose.

2.3 XDMA test project creation

Open the Vivado software, select Create Project and click Next, as shown in Figure 2-2.

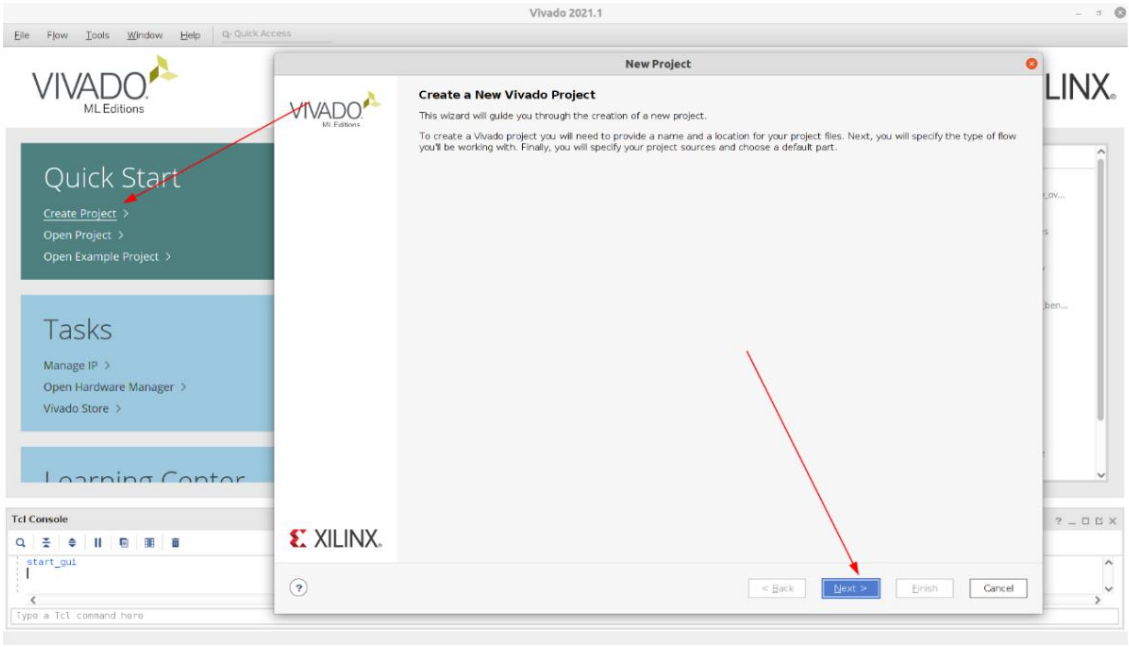

Figure 2-2 Creating a Vivado project

Enter the project name and project storage path, taking m2_artix7_xdma as an example, as shown in Figure 2-3.

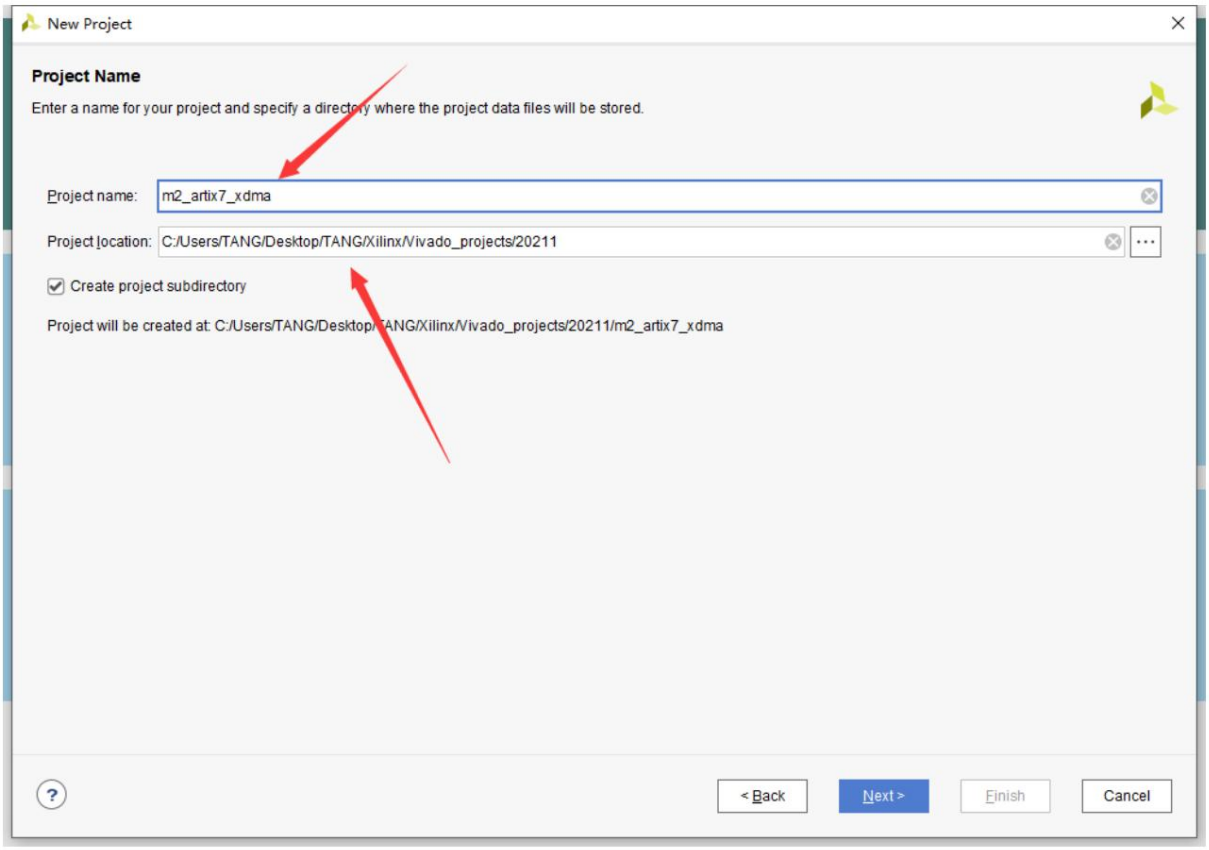

Figure 2-3 Enter the project name and path

Select RTL Project and check "No source file is currently specified", as shown in Figure 2-4.

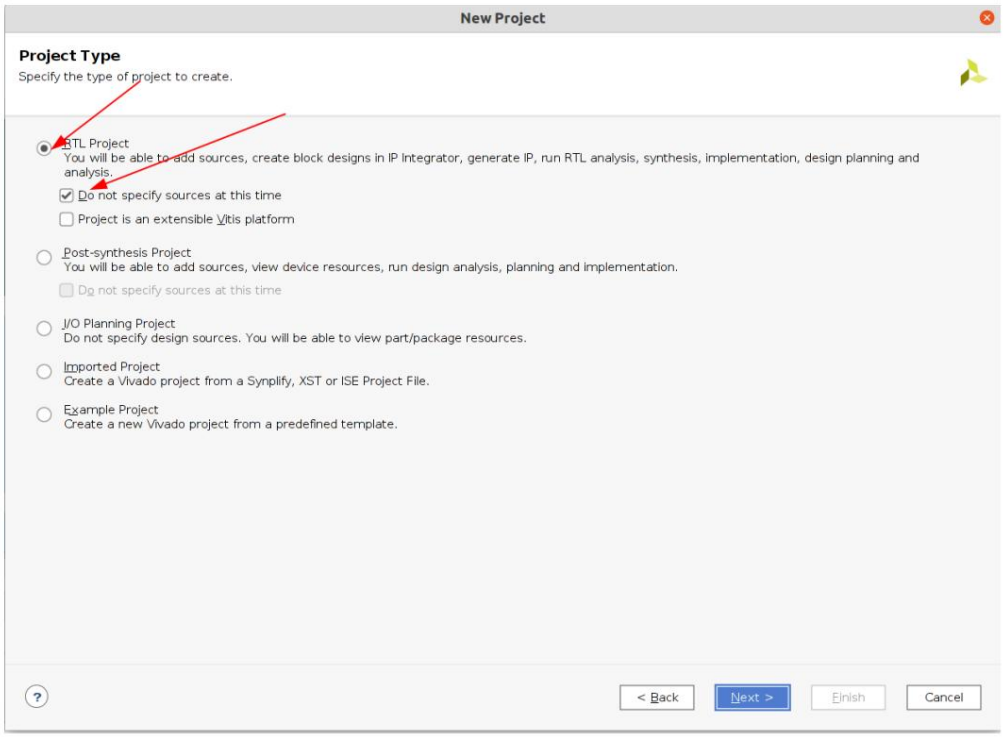

Figure 2-4 Select RTL Project

Select the chip according to the core board chip model and click Next. Take XC7A35T-2FGG484I as an example, as shown in Figure 2-5.

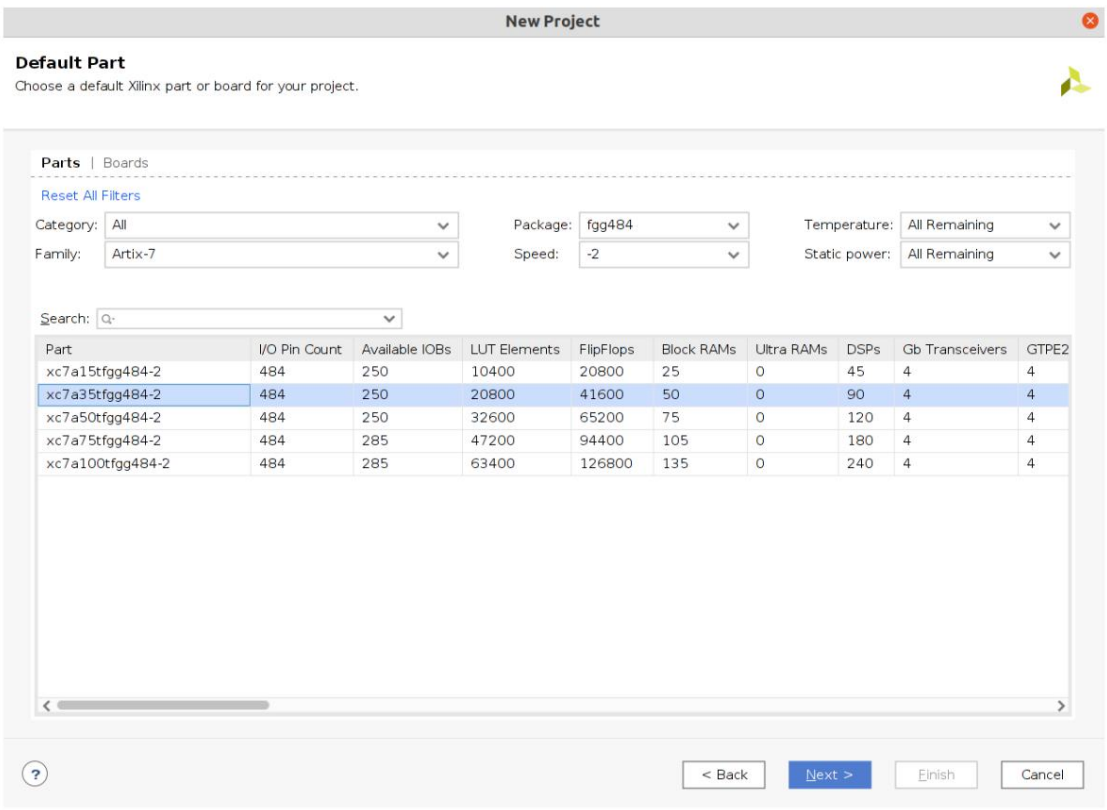

Figure 2-5 Select chip model

Click Create Block Design, enter the Block Design name and click OK, as shown in Figure 2-6.

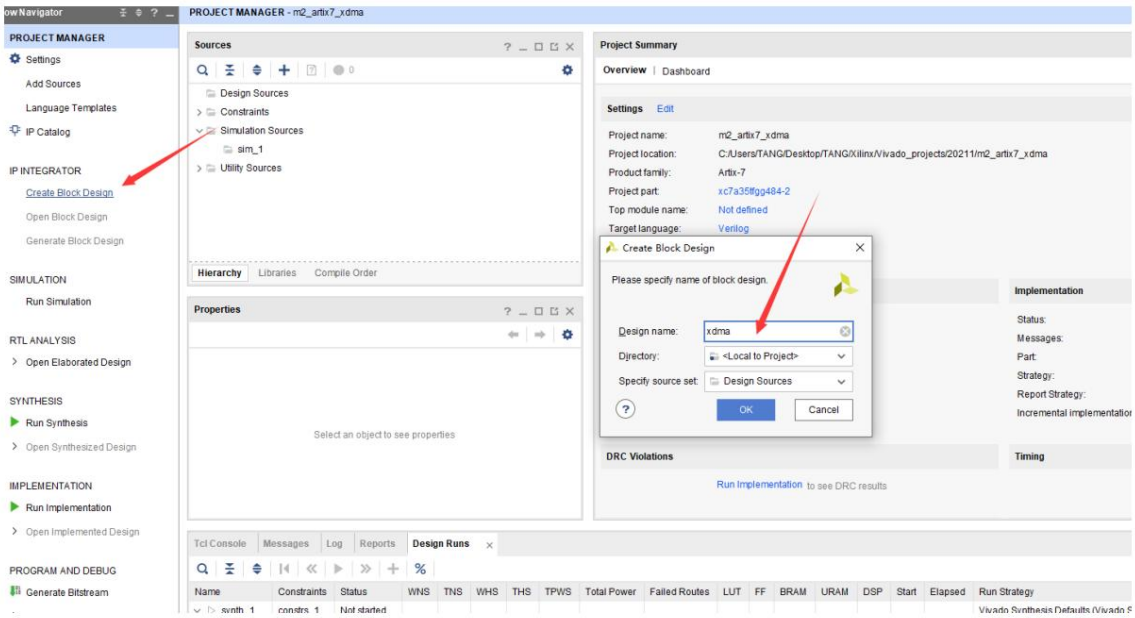

Figure 2-6 Create Block Design

Click the + sign to add IP, search for xdma, and click to add xdma IP, as shown in Figure 2-7.

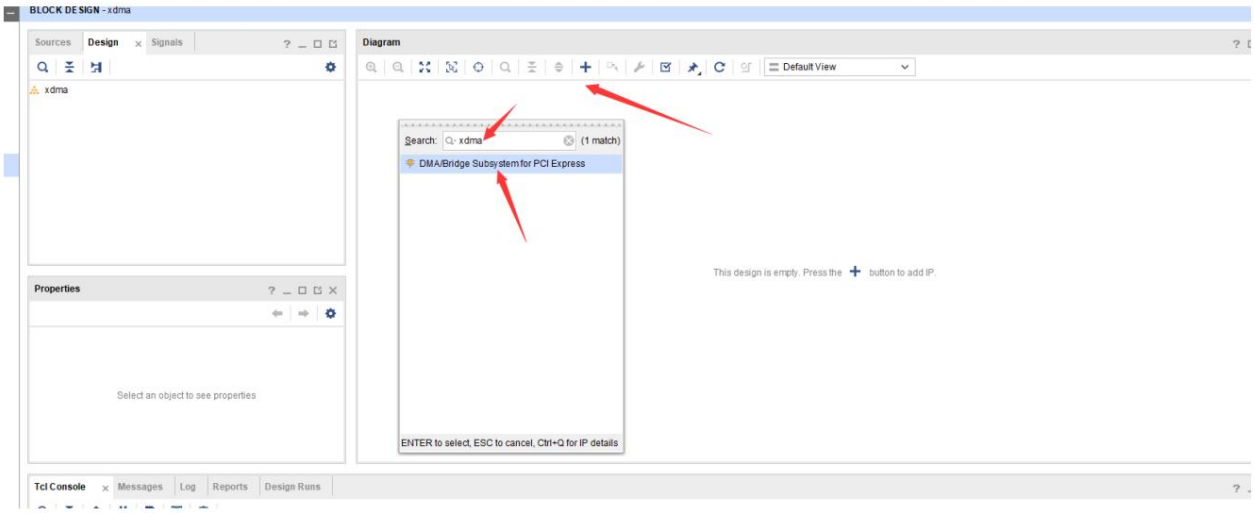

Figure 2-7 Adding XDMA IP

2.4 XDMA IP Configuration

2.4.1 Basic Item

Mode: Configuration mode, select BASE configuration

Lane Width: Select the number of PCIE lanes. The maximum number of lanes for an M2 accelerator card is 4. The more lanes, the faster the communication speed.

Users need to select the correct number of lanes based on the actual number of lanes that can be reached by the hardware. (35T can only be selected due to resource reasons.

*2, non-35T can be selected as *4), this project takes A7-35T as an example.

Max Link Speed: Select 5.0GT/s, which means PCIE2.0.

Reference Clock: 100MHZ, reference clock 100M

DMA Interface Option: Select AXI4 interface

AXI Data Width: 64bit, that is, the AXI4 data bus width is 64bit AXI Clock:

125M, that is, the AXI4 interface clock is 125MHZ DMA Interface

option is set to AXI Memory Mapped mode

As shown in Figure 2-8.

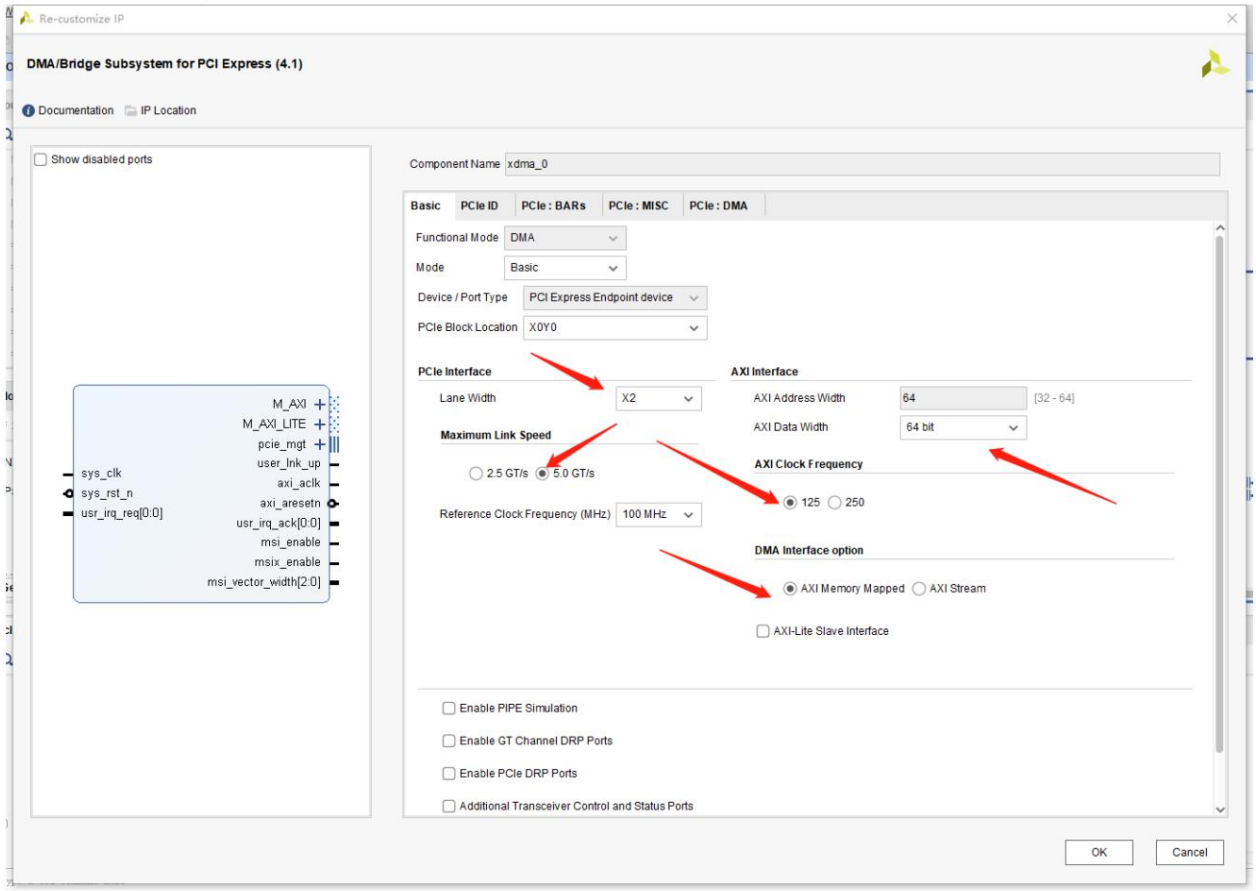

Figure 2-8 Basic Configuration

2.4.2 PCIE ID Configuration

PCIE ID configuration, here you can select the default configuration, the default device type is Simple communication controllers, as shown in Figure 2-9.

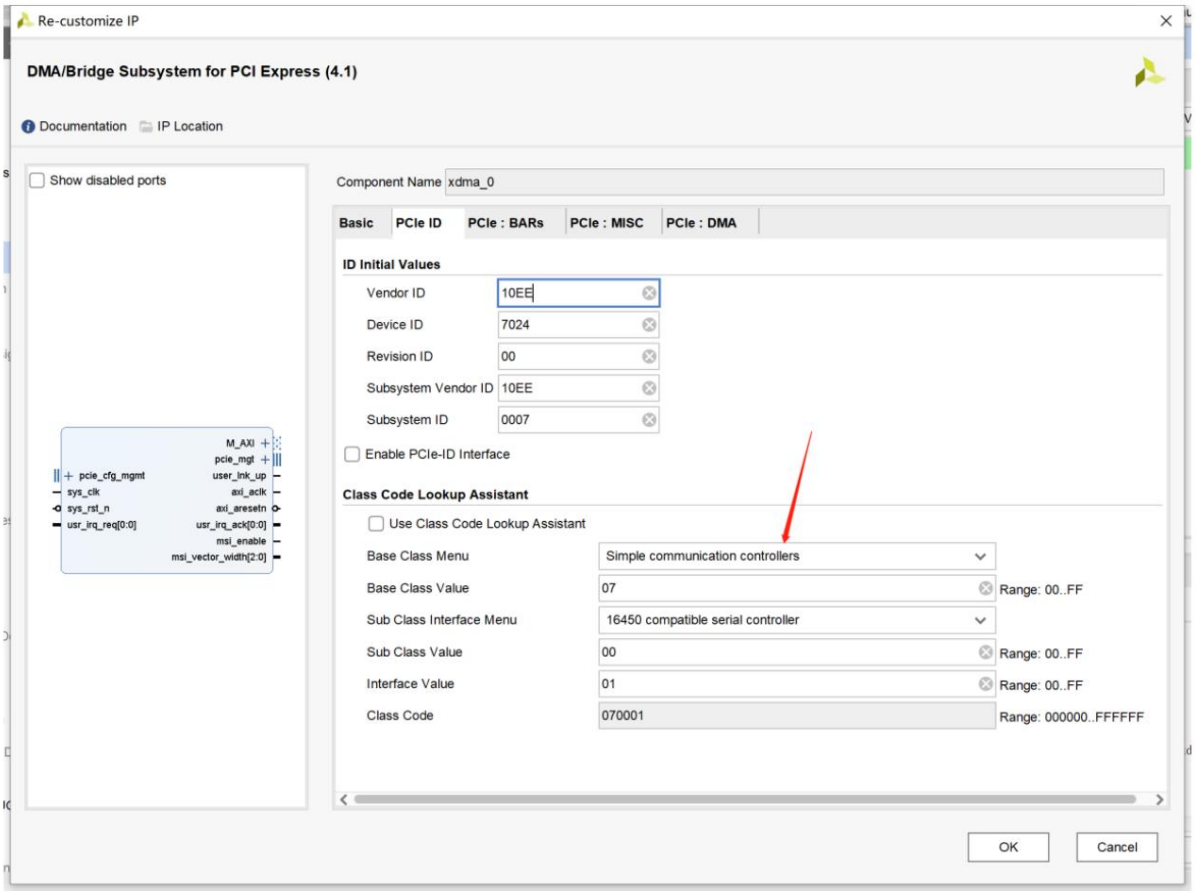

Figure 2-9 PCIE ID Configuration

2.4.3 PCIE BAR Configuration

PCIE BAR configuration, the configuration here is more important, first enable PCIE to AXI Lite Master Interface, so that the host side can access the user logic side registers or other AXI4-Lite bus device mapping space through PCIE. Select 1M, of course, users can also customize the size according to actual needs.

PCIE to AXI Translation: This setting is very important. Usually, the PCIE BAR address on the host side is different from the address on the user logic side. This setting is to convert the BAR address to the AXI address. For example, if the BAR address on the host side is 0 and the conversion setting in the IP is 0x40000000, then the host access BAR address 0 is converted to the AXI LIte bus address 0x40000000.

PCIE to DMA Interface: Select 64bit Enable

DMA Bypass is not used at the moment.

The PCIE BAR settings are shown in Figure 2-10.

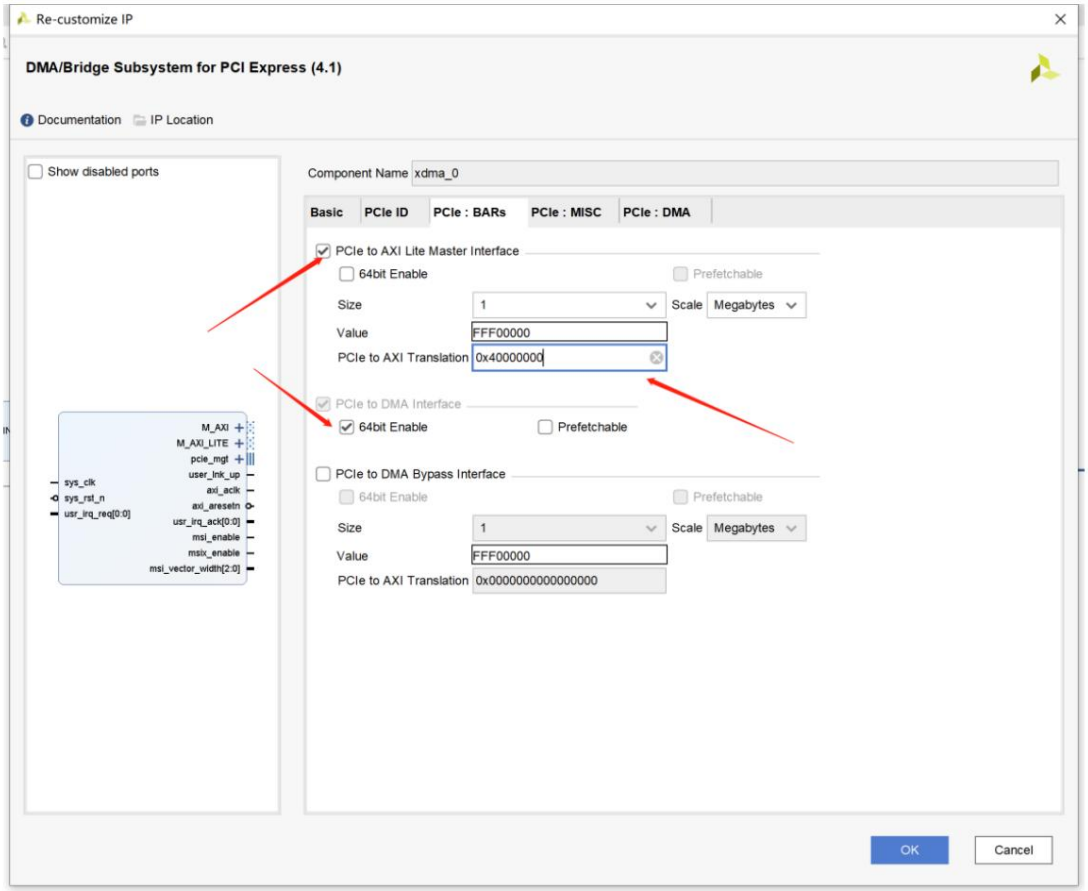

Figure 2-10 PCIE BAR configuration

2.4.4 PCIE interrupt configuration

Legacy interrupt Settings: Legacy interrupt settings, set to NONE, that is, do not use legacy interrupts; Enable MSI Capability Structure: Uncheck to disable the MSI interrupt function; Enable MSI-X Capability Structure: Uncheck to disable the MSI-X interrupt function; Configuration Management Interface: The PCIe configuration management interface is generally not used, so uncheck it. The configured PCIe: Misc tab is shown in Figure 2-11.

Some options on the PCIe: Misc tab are described as follows:

Number of User Interrupt Request: The number of user interrupt requests, up to 16 user interrupt requests can be selected; Legacy Interrupt Settings: You can select one of the legacy interrupts: INTA, INTB, INTC or INTD; MSI Capabilities: By default, the MSI function is enabled and 1 vector is enabled, and up to 32 vectors can be selected. Normally, Linux uses only 1 vector for MSI, so this option is turned on here; MSI-X Capabilities: Check to use the MSI-X feature; Finite Completion Credits (available in Advanced configuration mode): On systems that support limited completion credits, this option can be enabled for better performance;

Extended Tag Field: Extended tag field, by default, uses 6-bit completion tags. For UltraScale and Virtex-7 devices, the extended tag option provides 64 tags. For UltraScale+ devices, the extended tag option provides

256 tags. If the Extended Tags option is not selected, DMA uses 32 tags for all devices.

options.

Configuration Management Interface: Whether to use the PCIe configuration management interface. It is not used here. About MSI-

X interrupt: You can try to use MSI-X interrupt instead of MSI or traditional interrupt.

interrupts, the data rate is better than using MSI or traditional interrupt-based designs.

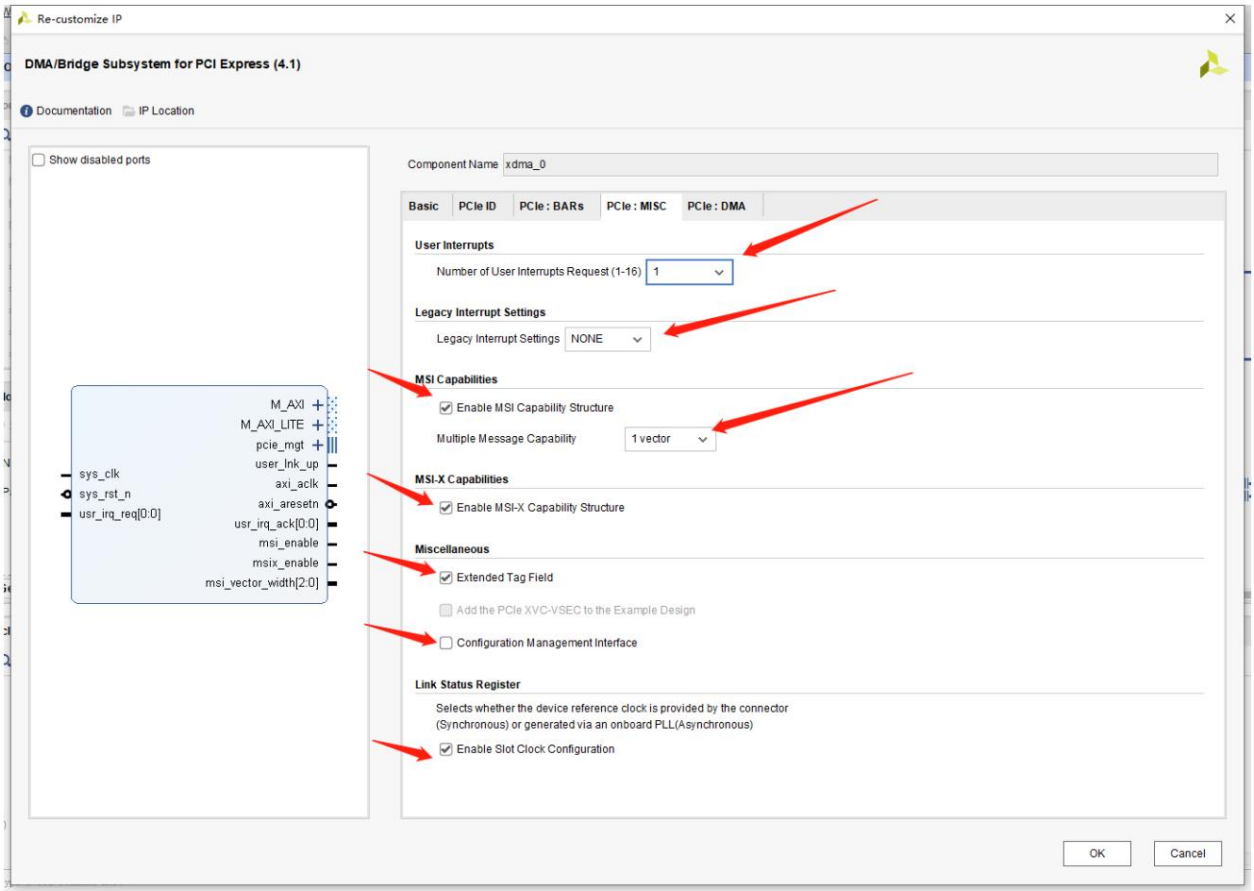

Figure 2-11 PCIE interrupt configuration

2.4.5 PCIE DMA Configuration

Number of DMA Read ChannelÿH2Cÿÿ Number of DMA Write ChannelÿC2Hÿÿÿ

For PCIE2.0, the maximum number of options is 2, which means XDMA can provide up to two independent write channels.

And two independent read channels, independent channels play a great role in practical applications. In advance, under the premise of

bandwidth permission, one PCIE can realize multiple different transmission functions without affecting each other. Here we choose 2

Number of Request IDs for Read (Write) channel: This is the maximum number of IDs allowed for each channel.

The default value is sufficient. The DMA configuration is shown in Figure 2-12.

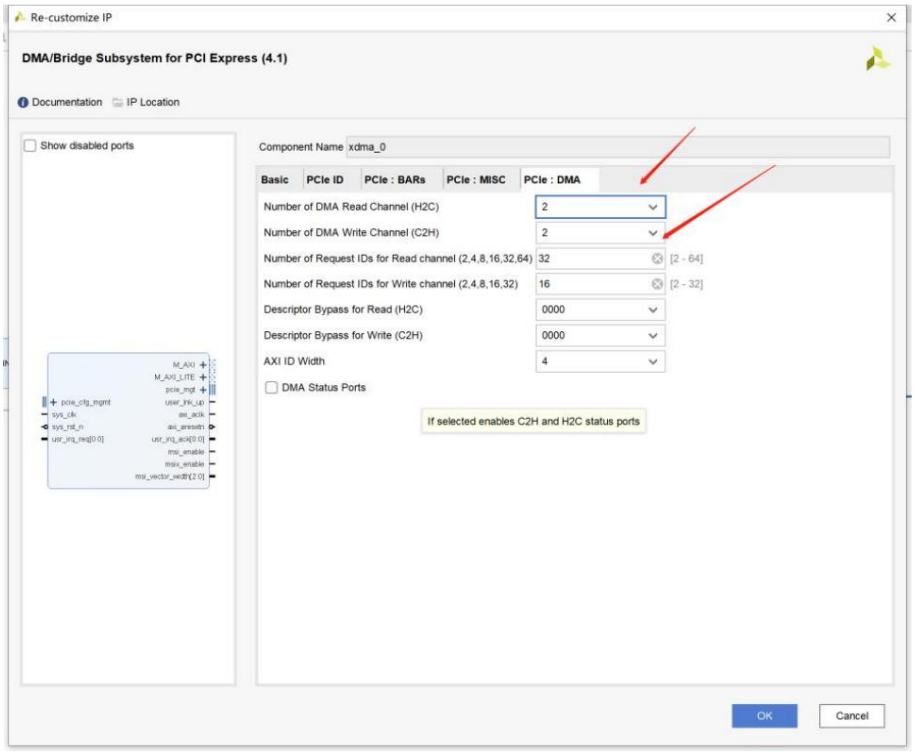

Figure 2-12 PCIE DMA configuration

Now the PCIE XDMA IP core configuration is completed, click OK.

Click Run Block Automation. In this interface, we need to check whether the configuration information in "Options" corresponds to the options we just configured when we configured XDMA. If not, we need to change it back (this can be said to be a part of Vivado).

The configuration items that require special attention are marked with red boxes in the figure above. After checking that the configuration is correct, click the "OK" button in the lower right corner. See Figure 2-13.

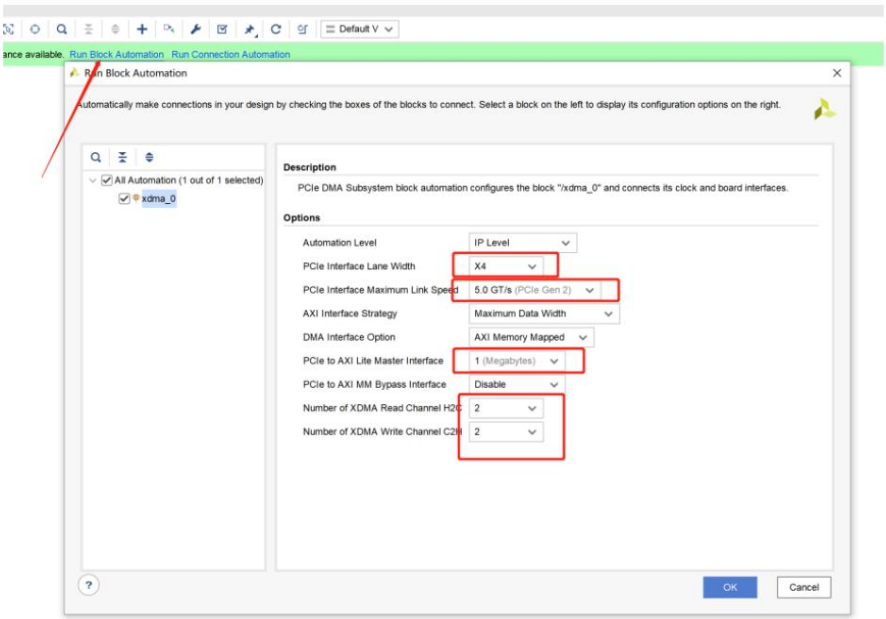

Figure 2-13 Automatic processing

After automatic processing is completed, the structure shown in Figure 2-14 is obtained.

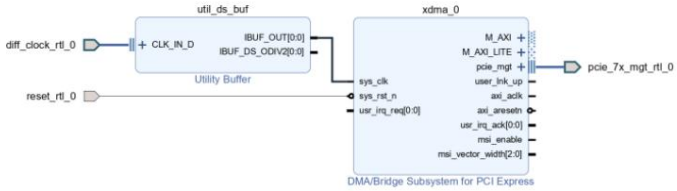

Figure 2-14 Automatic processing completed

2.4.6 Remaining Configuration

Then under the Add IP option, add an AXI BRAM Controller module, set the data width to 64, and configure

The configuration process is shown in Figures 2-14 and 2-15. Follow the arrows to configure and keep other items as default.

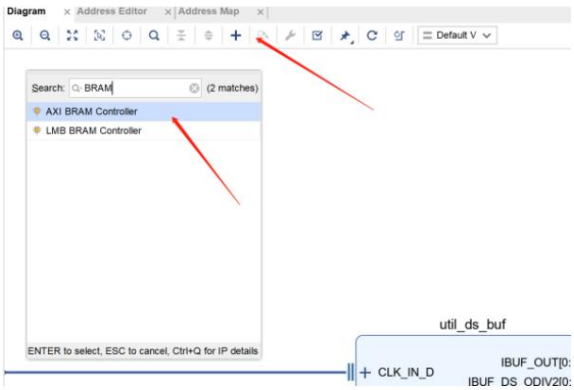

Figure 2-14 Select Add AXI BRAM Controller

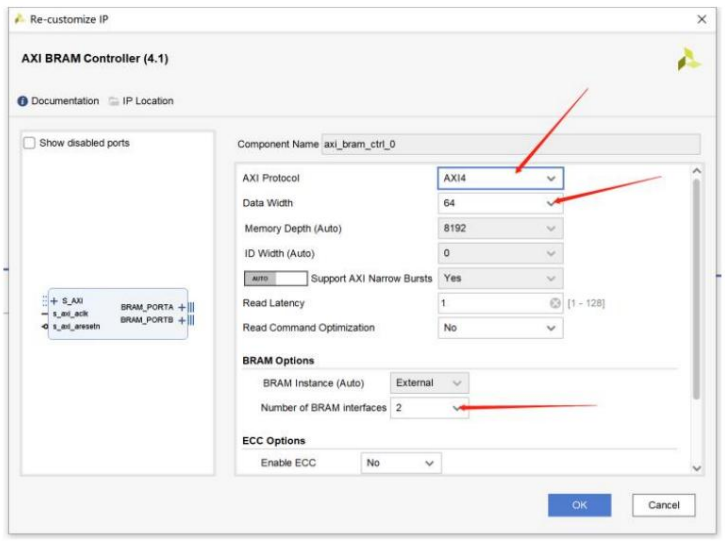

Figure 2-15 Select Configure AXI BRAM Controller

After adding, it is shown as Figure 2-16

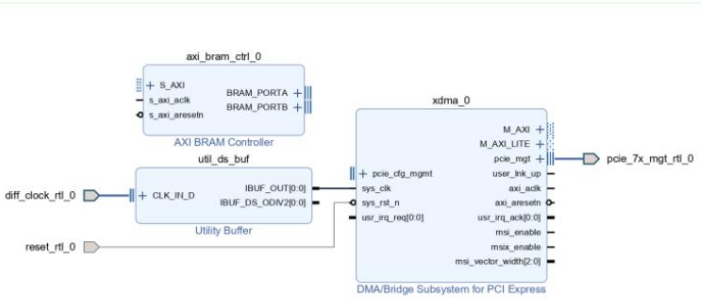

Figure 2-16 Adding BRAM Controller

Create an AXI-GPIO IP to control the three LED lights on the board. The IP configuration is shown in Figure 2-17.

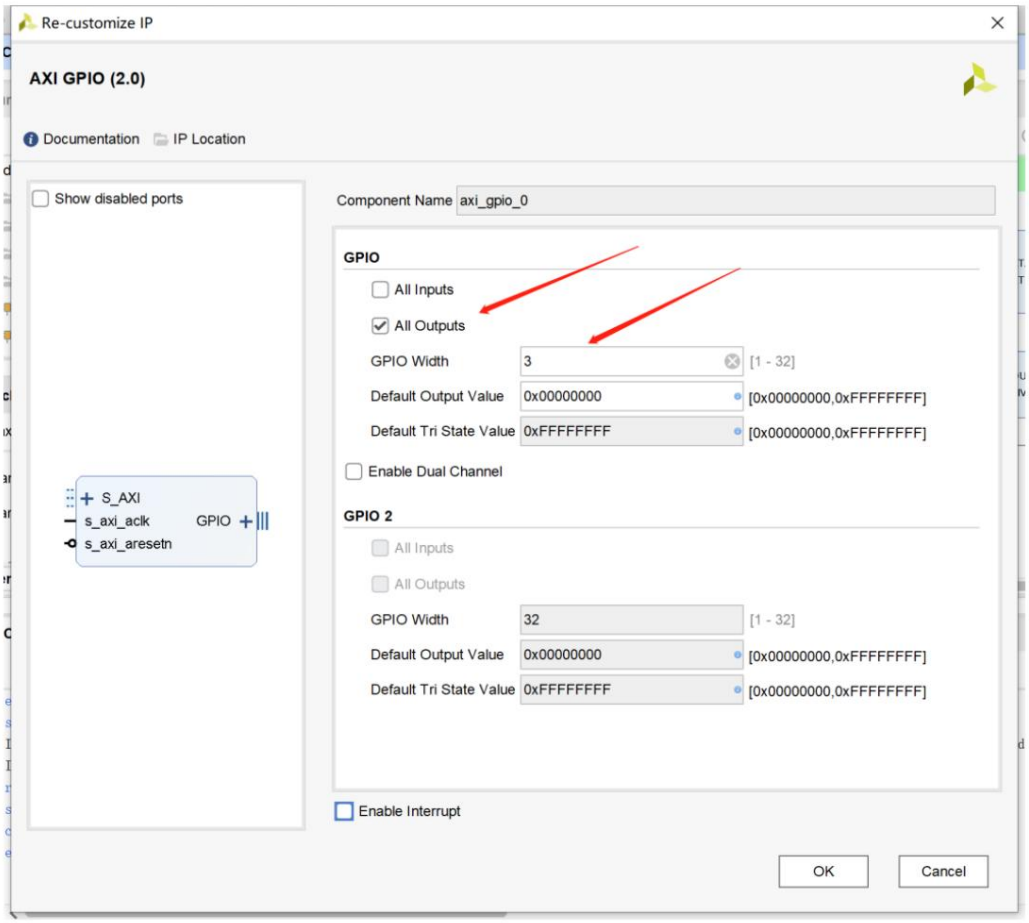

Figure 2-17 Creating AXI GPIO IP

Click "Run Connection Automation" and check "All Automation" in the pop-up window to let Vivado

The tool completes the rest of the interface connection work. As shown in Figure 2-18. As you can see, the Vivado tool automatically adds

Since the M_AXI interface of XDMA is connected to the memory and has high bandwidth requirements, the Vivado tool selects the AXI SmartConnect IP, which has better performance than the AXI Interconnect IP. The M_AXI_LITE interface is used to drive and control the user's own logic (such as controlling the LED's water effect, etc.), and generally does not require high bandwidth, so

The Vivado tool selects the generic AXI Interconnect IP.

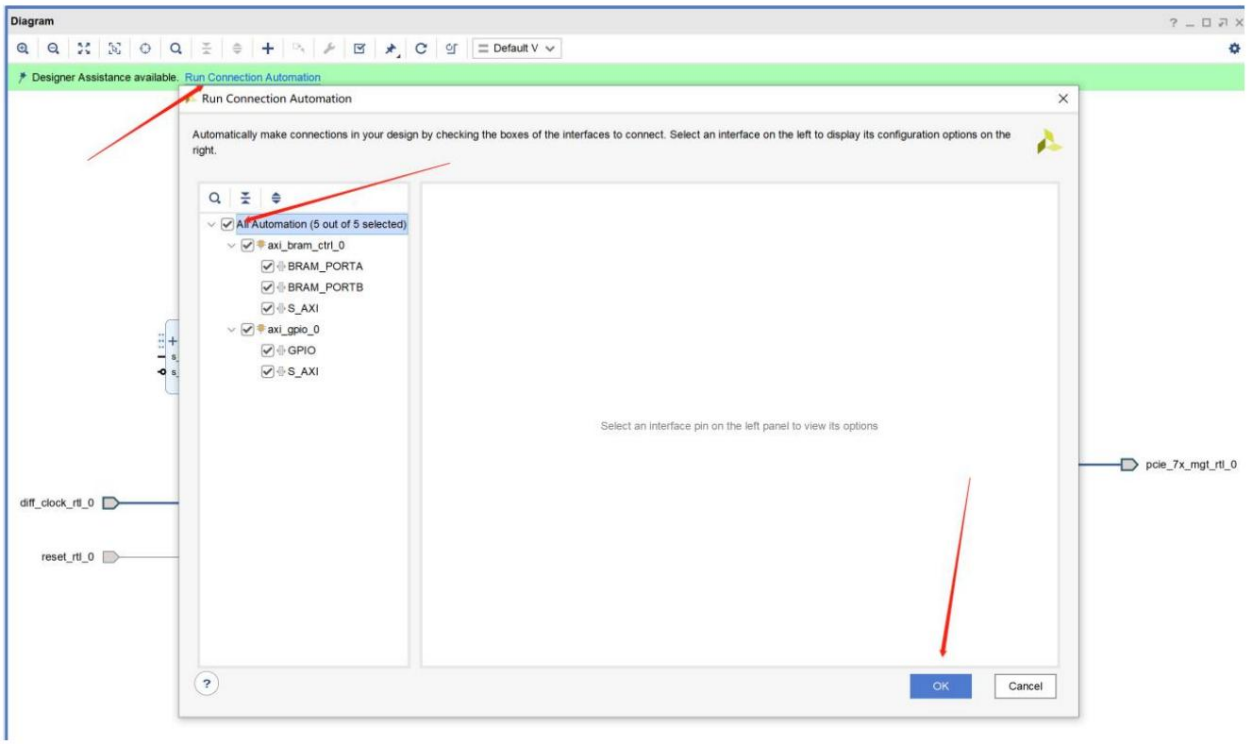

Figure 2-18 Automatic connection

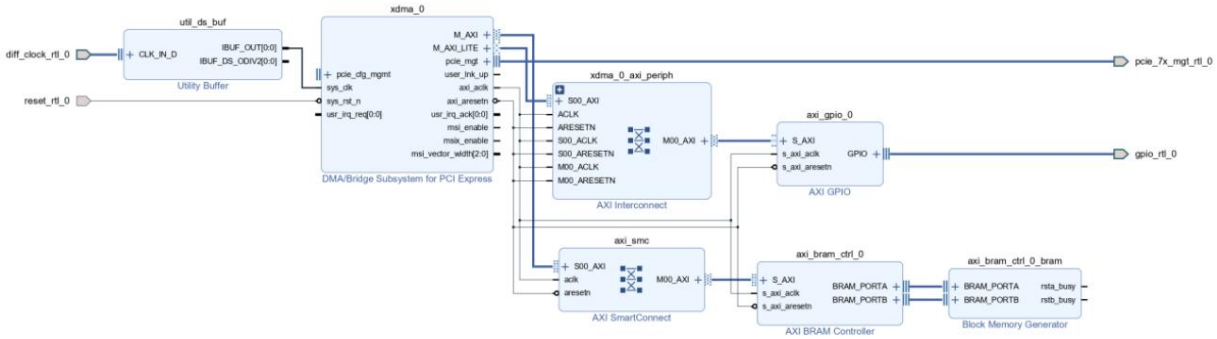

Figure 2-19 Automatic connection result

We don't need the rsta_busy and rstb_busy signals of the Block Memory Generator IP (used for the internal safety circuit of the Block Memory Generator, generally not needed). Keeping them will affect the appearance, so you can remove them. Block Memory Generator IP, open its configuration window, and uncheck the Enable Safety Circuit", and then click the "OK" button, as shown in Figure 2-20:

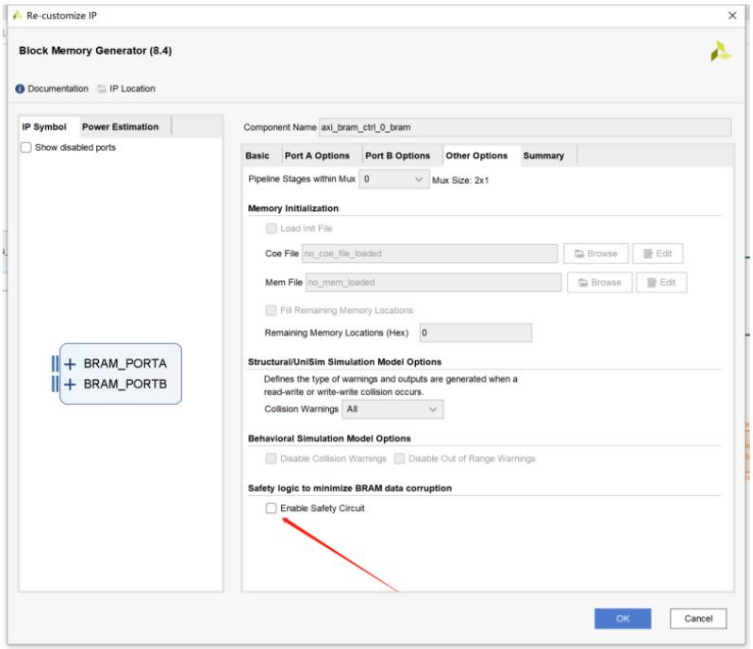

Figure 2-20 Removing the BRAM module safety circuit

Although the system has been built, there is still a key step left - modifying the address mapping.

The offset address of the BRAM connected to the M_AXI interface of XDMA is changed to 0x0000. Of course, it can be left unchanged. I have tried to use the default value and it works. The key point is the user-designed offset address connected to the M_AXI_LITE interface of XDMA. This offset address needs to be consistent with the offset address in the PCIe:BARs tab of XDMA configuration.

PCIe to AXI Lite Master Interface: "PCIe to AXI Translation:" under the configuration item PCIe to AXI Lite Master Interface:

Since the offset address of the M_AXI_LITE interface of XDMA is automatically assigned by the Vivado tool, there will generally be no problem. It is recommended to configure it according to the offset address of the M_AXI_LITE interface of XDMA "PCIe to AXI Translation:" in the XDMA Configuration tab PCIe:BARs.

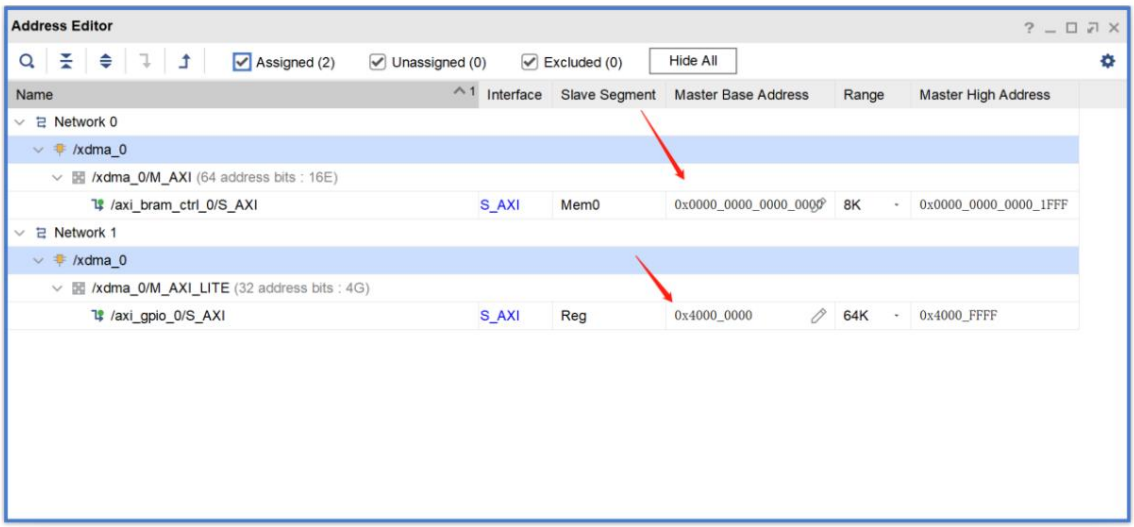

Figure 2-21 Address space

editing After editing, click Valid Design as shown in Figure 2-22 to ensure there are no Waring and Error, and complete the Block

Design configuration.

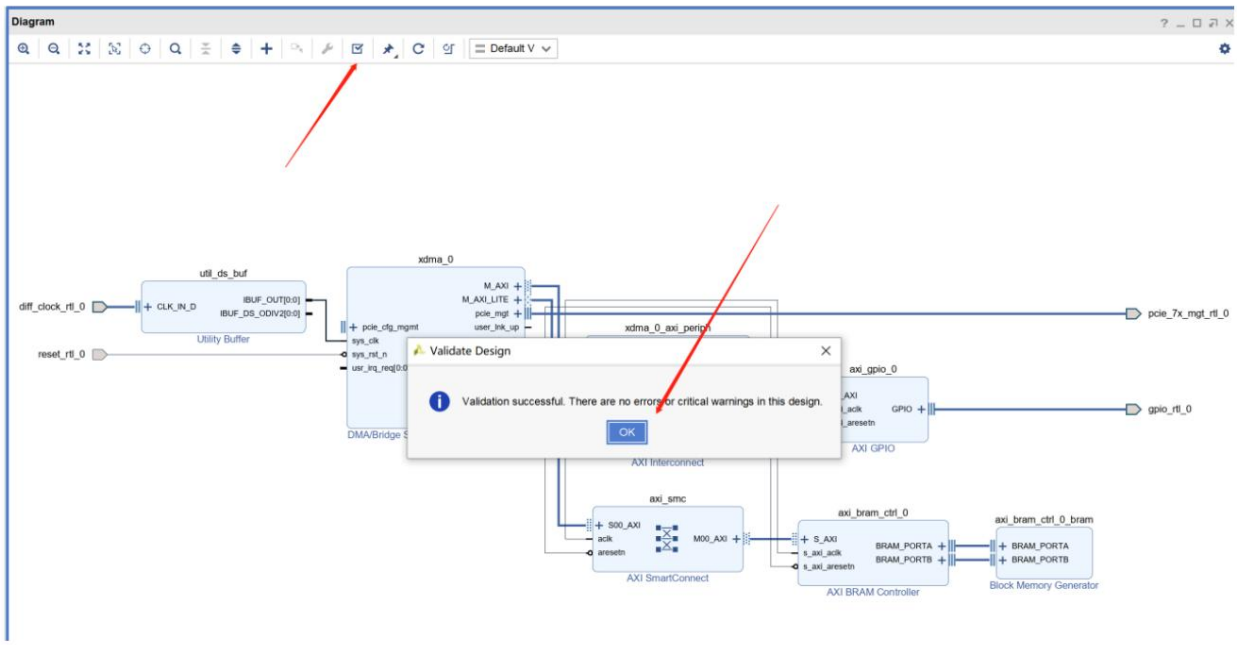

Figure 2-22 Block Design completed

2.5 Generate Bin file and burn

Right-click Block Design and select Generate as shown in Figure 2-23 to generate the output file.

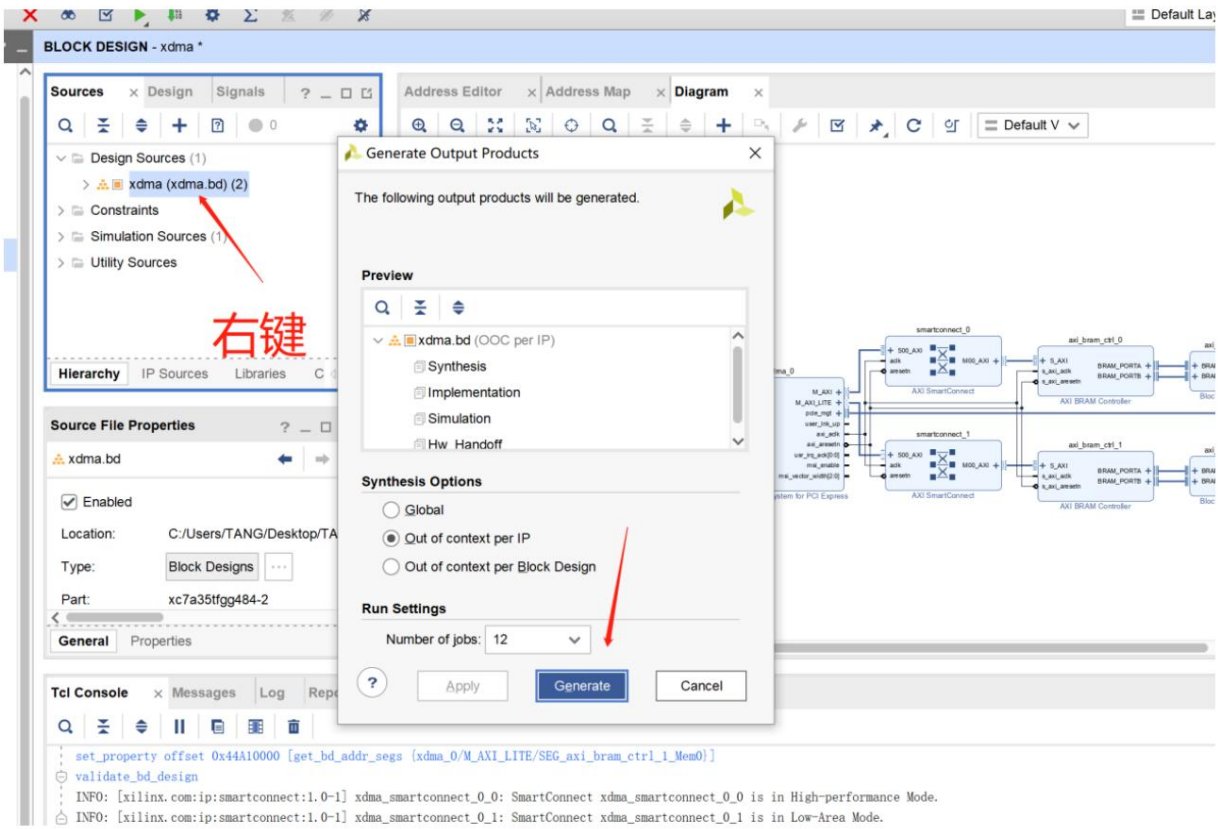

Figure 2-23 Generate output file

Right-click Block Design and select Generate Wapper according to the default configuration as shown in Figure 2-24.

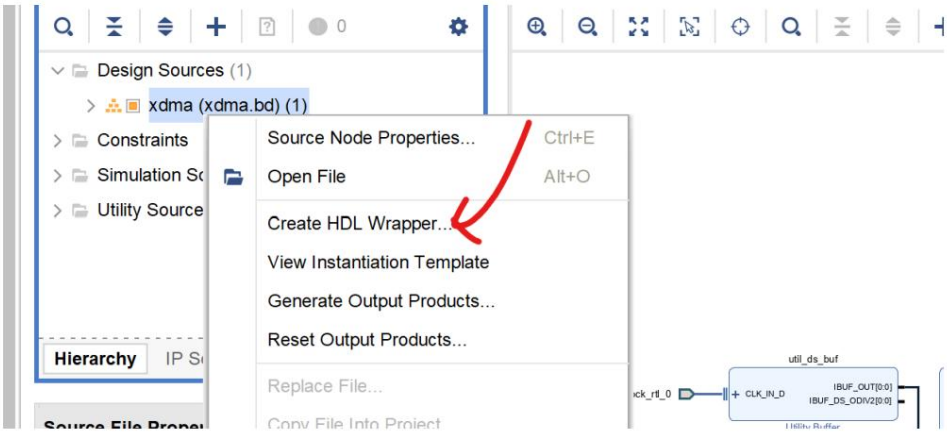

Figure 2-24 Generate HDL

Wapper creates a .xdc constraint file and uses the following constraint content. #bit compress spix4 speed up set_property CFGBVS VCCO [current_design] set_property CONFIG_VOLTAGE 3.3 [current_design] set_property BITSTREAM.GENERAL.COMPRESS true [current_design] set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design] set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design] set_property BITSTREAM.CONFIG.SPI_FALL_EDGE Yes [current_design]

```
#pcie reset_n input
set_property -dict {PACKAGE_PIN K22 IOSTANDARD LVCMOS33} [get_ports reset_rtl_0]

#led*3 outpuy
set_property -dict {PACKAGE_PIN AB7 IOSTANDARD LVCMOS33} [get_ports
gpio_rtl_0_tri_o[2]]
set_property -dict {PACKAGE_PIN AA6 IOSTANDARD LVCMOS33} [get_ports
gpio_rtl_0_tri_o[1]]
set_property -dict {PACKAGE_PIN AB6 IOSTANDARD LVCMOS33} [get_ports
gpio_rtl_0_tri_o[0]]

#pcie lanes
set_property PACKAGE_PIN F10 [get_ports {diff_clock_rtl_0_clk_p[0]}]

set_property PACKAGE_PIN D9 [get_ports {pcie_7x_mgt_rtl_0_rxp[3]}] set_property
PACKAGE_PIN B10 [get_ports {pcie_7x_mgt_rtl_0_rxp[2]}] set_property PACKAGE_PIN
D11 [get_ports {pcie_7x_mgt_rtl_0_rxp[1]}] set_property PACKAGE_PIN B8 [get_ports
{pcie_7x_mgt_rtl_0_rxp[0]}] set_property PACKAGE_PIN A8 [get_ports
{pcie_7x_mgt_rtl_0_rxn[0]}] set_property PACKAGE_PIN C11 [get_ports
{pcie_7x_mgt_rtl_0_rxn[1]}] set_property PACKAGE_PIN A10 [get_ports
{pcie_7x_mgt_rtl_0_rxn[2]}] set_property PACKAGE_PIN C9 [get_ports
{pcie_7x_mgt_rtl_0_rxn[3]}] set_property PACKAGE_PIN B4 [get_ports
{pcie_7x_mgt_rtl_0_txp[0]}] set_property PACKAGE_PIN D5 [get_ports
{pcie_7x_mgt_rtl_0_txp[1]}] set_property PACKAGE_PIN B6 [get_ports
{pcie_7x_mgt_rtl_0_txp[2]}] set_property PACKAGE_PIN D7 [get_ports
{pcie_7x_mgt_rtl_0_txp[3]}] set_property PACKAGE_PIN A4 [get_ports
{pcie_7x_mgt_rtl_0_txn[0]}] set_property PACKAGE_PIN C5 [get_ports
{pcie_7x_mgt_rtl_0_txn[1]}] set_property PACKAGE_PIN A6 [get_ports
{pcie_7x_mgt_rtl_0_txn[2]}] set_property PACKAGE_PIN C7 [get_ports
{pcie_7x_mgt_rtl_0_txn[3]}]
```

Since the PCIE Lane sequence of this accelerator card is inconsistent with the official recommendation of Xilinx, the generated PCIE IP Lane constraints conflict with the user constraints, so a Critical Warning will be generated in Implementation, which does not affect the use of functions. If necessary, you can follow the steps below to

resolve it. Open the following file:

\m2_artix7_xdma.gen\sources_1\bd\xdma\ip\xdma_xdma_0_0\ip_0\source\xdma_xdma_0_0_pcie2_ip-PCIE_X0Y0.xdc

Modify the position of the PCIE Lane according to Figure 2-25, and no Warning will be reported during subsequent layout and routing.

(35T changed to X0Y0-X0Y1, 50T-100T changed to **X0Y0-X0Y3, 200T** changed to **X0Y4-X0Y7)**

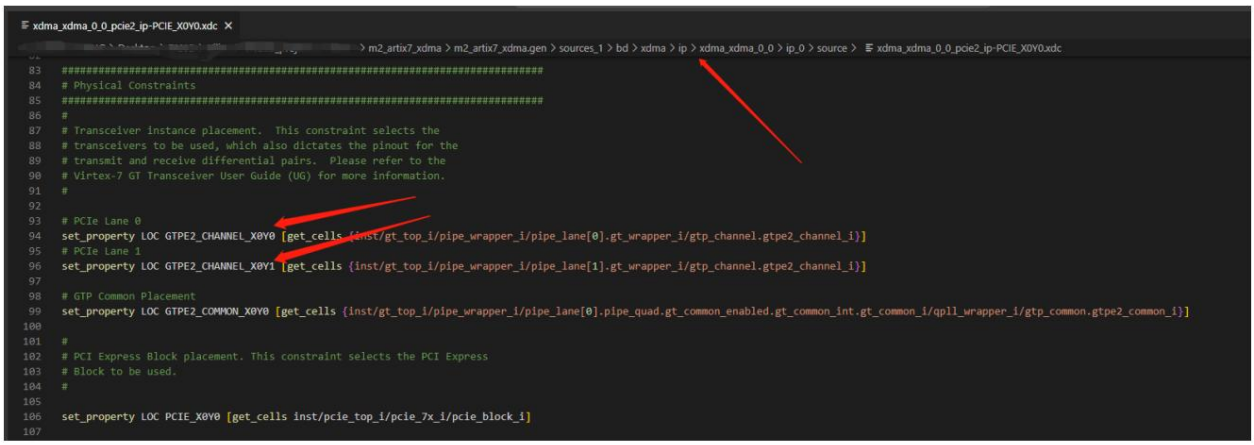

Figure 2-25 Modify PCIE Lane position

After the compilation is successful, generate the **Bin** file according to Section **1-3** and burn it into **the Flash ,** ready for testing.

The **Flash** curing process is described below .

2.6 XDMA driver and application compilation (optional)

Open the VS2015 software, as shown in Figure 2-26, and select the Open Project option.

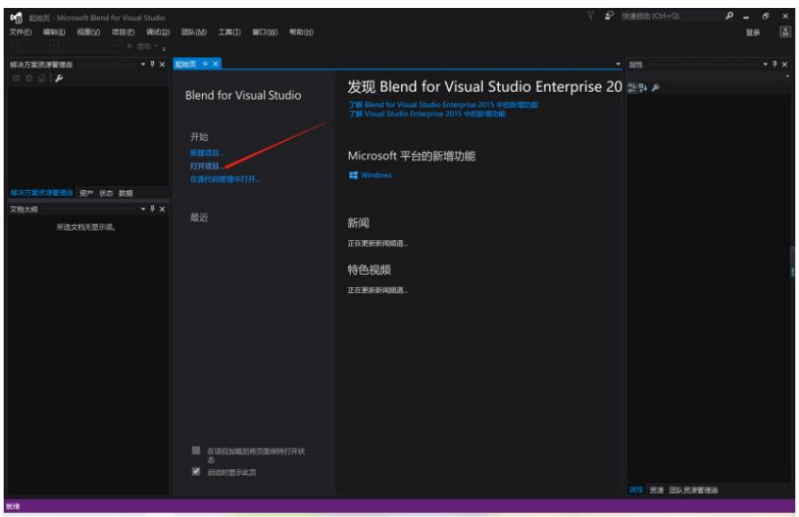

Figure 2-26 Open the project using VS2015

Unzip the provided driver and application compressed package, and open the projects in the provided driver and application folders, as shown in Figure 2-27 shown.

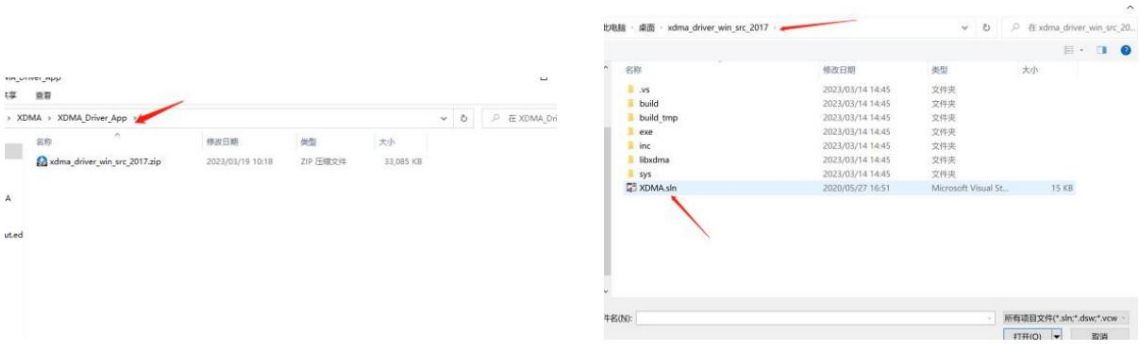

Figure 2-27 Open Project

Clean up the solution, as shown in Figure 2-28.

Figure 2-28 Cleaning solution

Regenerate the solution, as shown in Figure 2-29. If the software and environment in Section 1.2.2 are installed correctly, the

The dynamic compilation is successful, and the message "Generation Success" is displayed as shown in Figure 2-30.

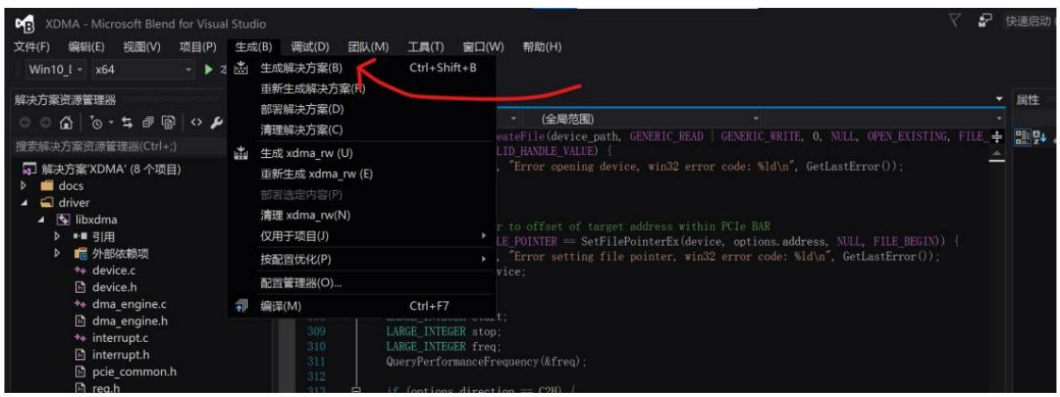

Figure 2-29 Regenerate solution

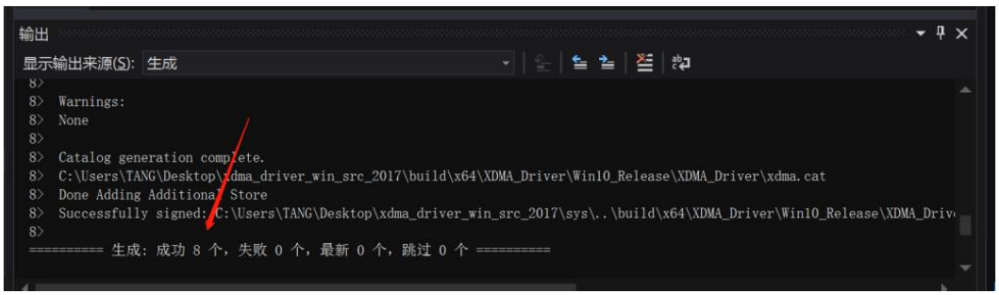

Figure 2-30 Driver and application compiled successfully

At this point, the driver and application are compiled.

2.7 Preparation for Computer Test

Next, there is an important setting. According to the official documentation, the XDMA driver does not provide a verification

Therefore, the system must enter test mode before installing the driver.

Use the following command to turn on or off the test mode.

bcdedit /set testsigning on Turn on the test mode bcdedit /set testsigning

off Turn off the test mode In WIN7/WIN10 system, as shown in Figure

2-31, open the terminal and enter the corresponding command and press Enter. Be sure to use the administrator

Permissions

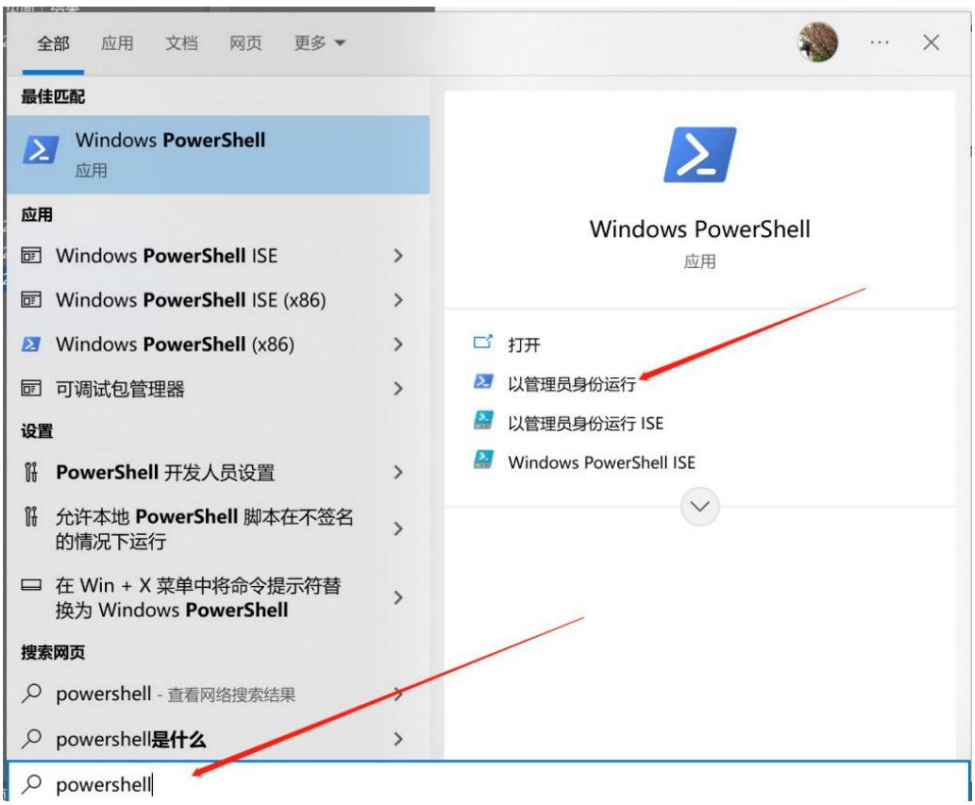

Figure 2-31 Search for PowerShell and open it as an administrator

Enter the command to turn on the test mode and press Enter, as shown in Figure 2-32.

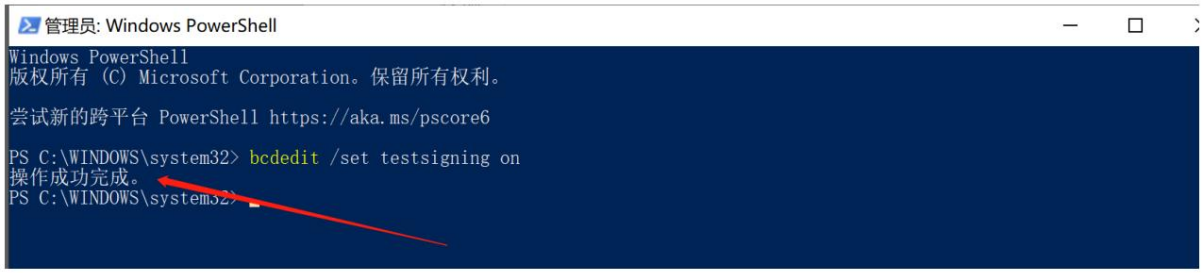

Figure 2-32 Successfully entering test mode

After the setup is complete, shut down the computer and install the M2 accelerator card with Flash burned into the M2 port of the computer.

The M2 interface NVME protocol is required. After the installation is complete, turn on the computer and the light on the accelerator card will light up, as shown in Figure 2-33.

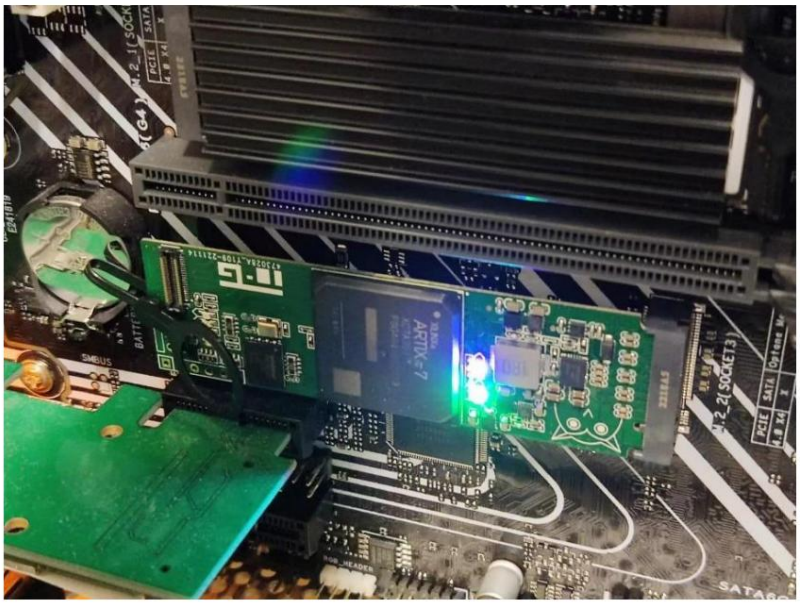

Figure 2-33 Installing the accelerator card and powering on the computer

Open the computer device manager and you will see "PCI Serial Port" in the Other Devices column, as shown in Figure 2-34.

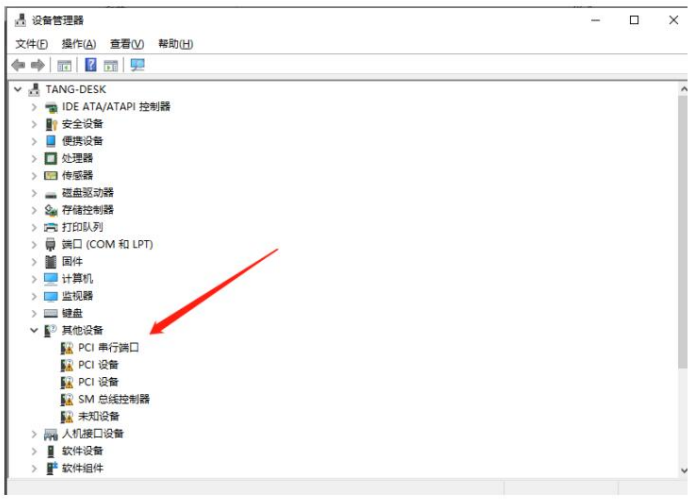

Figure 2-34 Unknown device identified by the device manager

Open the previously compiled driver folder Win10_Release, as shown in Figure 2-35.

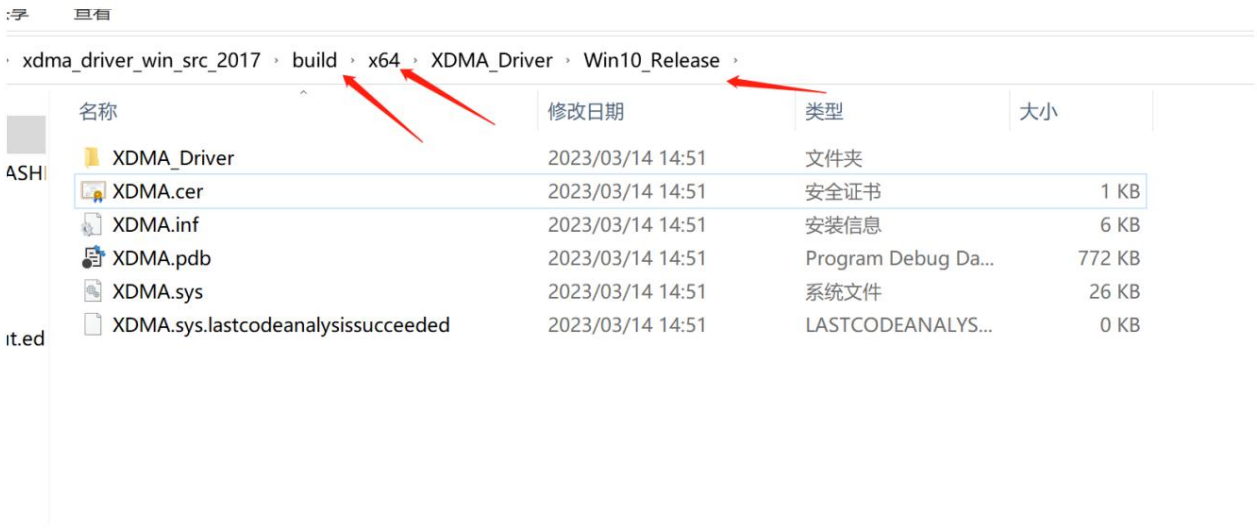

Figure 2-35 Right-click the compiled driver folder,

select the certificate file, and click "Install Certificate", as shown in Figure 2-36. In the pop-up wizard, click Next to complete the installation.

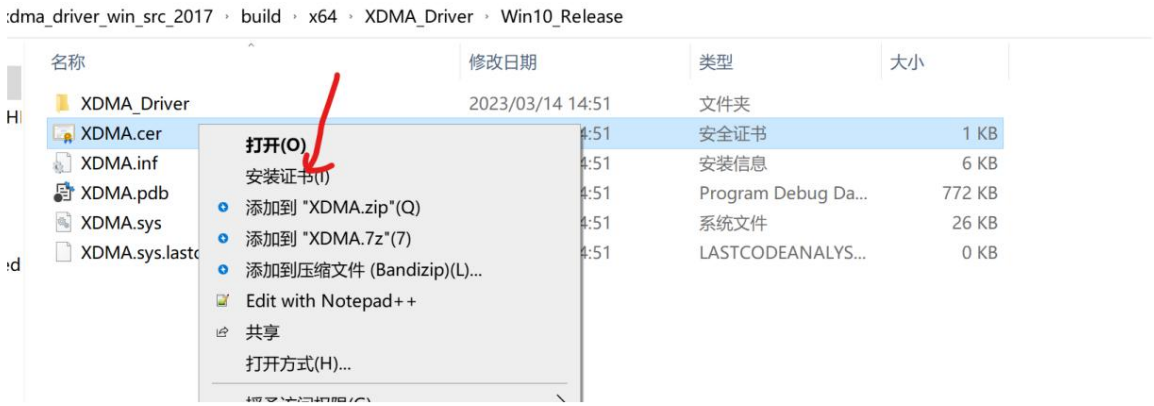

Figure 2-36 Installing the certificate

Enter the next level XDMA_Driver folder, right-click the XDMA.inf file to install the driver, such as

As shown in Figure 2-37.

Figure 2-37 Driver installation

During the driver installation process, a warning pops up that Windows cannot verify the driver publisher. Select Install Anyway and then complete the installation.

Complete the driver installation, as shown in Figure 2-38.

Figure 2-38 Driver installation completed

After the driver is installed, the XDMA device can be successfully identified in the device manager, as shown in Figure 2-39.

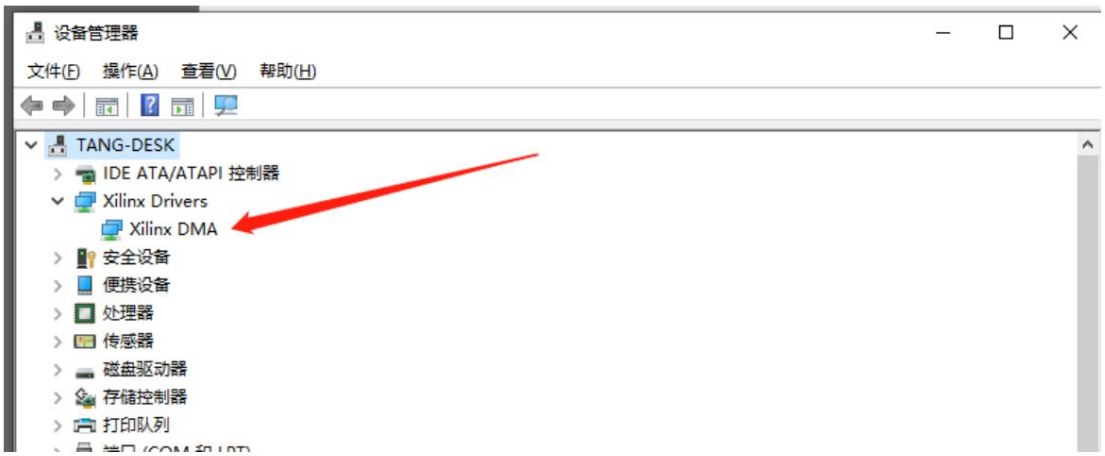

Figure 2-39 Device Manager recognizes XDMA device

2.8 xdma_rw BRAM read and write test

We open a terminal (if you double-click to run, it will exit quickly), cd into the compiled application directory

Find xdma_rw.exe in the directory. This application is used to operate all pcie devices. We only need to input .\xdma_rw.exe in the terminal, and we can see the prompt information, telling the user how to use this program. As shown in Figure 2-40.

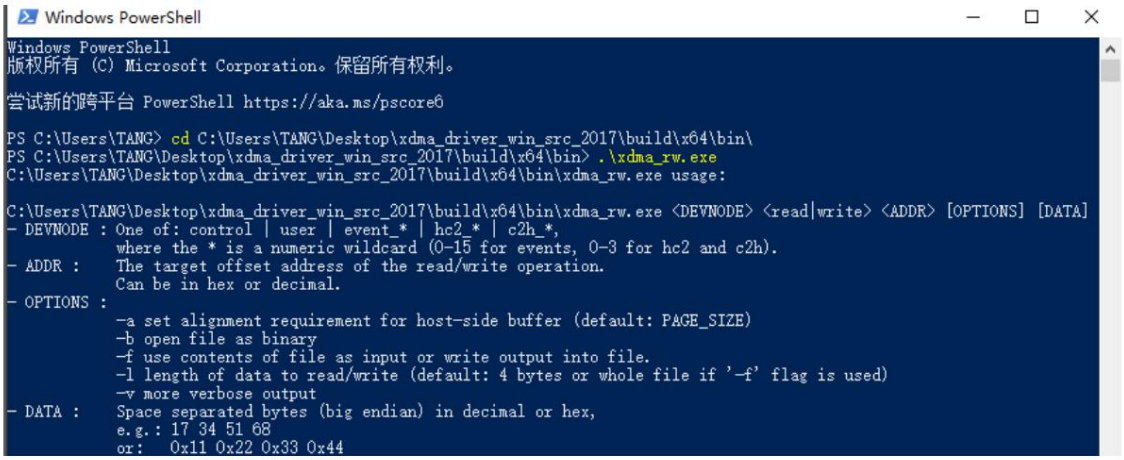

Figure 2-40 Enter the xdma_rw.exe folder and run it. You can see

that there is a datafile4k.bin file in the current directory. Then test to transfer this file to FPGA.

The BRAM is then read out.

First, enter the command in the terminal: .\xdma_rw.exe h2c_0 write 0x00000000 -b -f datafile4K.bin -l 4096 as shown in Figure 2-41.

This means that the file datafile4k.bin is read in binary form using the h2c_0 device and written to the BRAM memory address 0x0000000 with a length of 4096 bytes.

Figure 2-41 PCIE writes data to BRAM

Read it back next time.

Use the command .\xdma_rw.exe c2h_0 read 0x00000000 -b -f datafile4K_rd.bin -l 4096 as shown in Figure 2-42.

Figure 2-42 PCIE reads data back from BRAM

Use WinMerge software to open datafile4k.bin and datafile4K-recv.bin for comparison. The software prompts that the contents of the two files are completely equal, indicating that the PCIE read and write are successful, as shown in Figures 2-43 and 2-44.

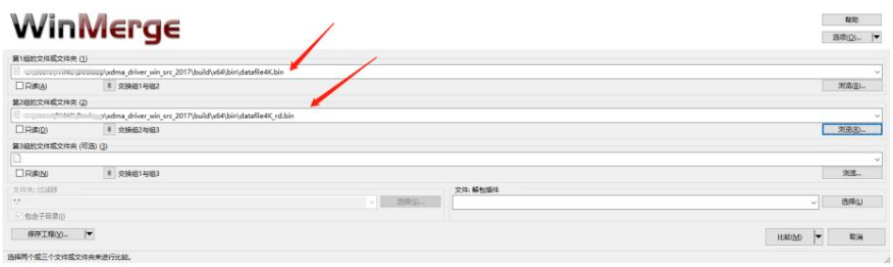

Figure 2-43 Select read and write files for comparison

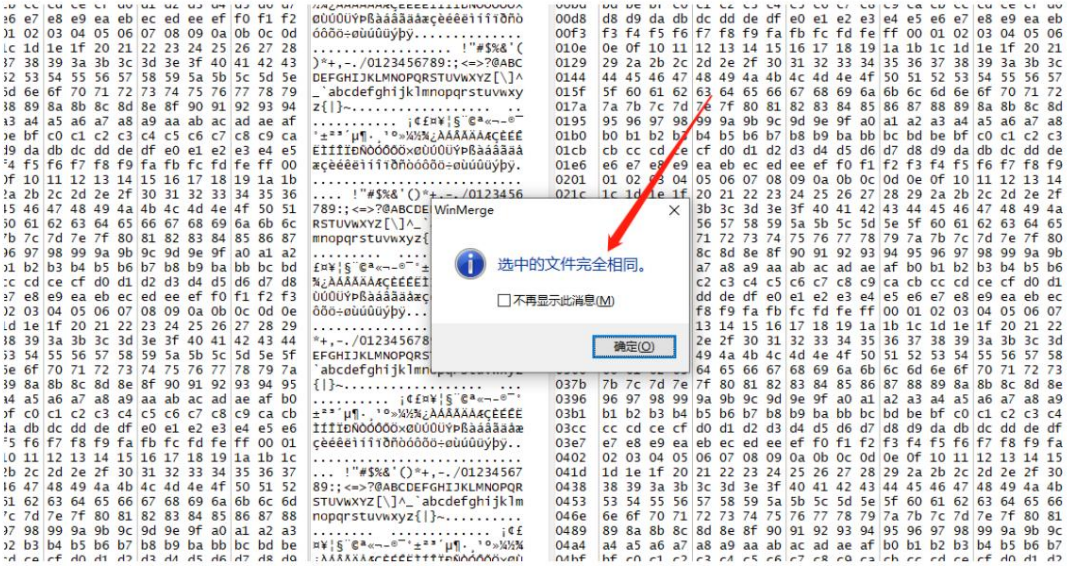

Figure 2-44 File comparison results

2.9 simple_dma Simple speed test

The simple_dma application is a simple speed test program. The way to use the simple_dma application is very simple. Just enter the following command in the open Powershell window.

The execution result of .

\simple_dma.exe is shown in Figure 2-45.

```
PS C:\Users\TANG\Desktop\xdma_driver_win_src_2017\build\x64\bin> .\simple_dma.exe
Found 1 Xdma device.
Starting single XDMA Card-to-Host transfer of 8 MB.
XDMA Card-to-Host transfer has finished after 11618 us.
Starting single XDMA Host-to-Card transfer for 8 MB.
XDMA Host-to-Card transfer has finished after 9443 us.
PS C:\Users\TANG\Desktop\xdma_driver_win_src_2017\build\x64\bin>
```

Figure 2-45 simple_dma test From the

execution results, the transfer rate can be calculated.

2.10 xdma_test Basic Test

The xdma_test application has the following features:

ÿ Detect how many h2c and c2h channels are enabled in the XDMA IP ÿ Detect if the

XDMA IP is configured in memory mapped (AXI-MM) or stream (AXI-ST) mode ÿ Perform data transfers on all available h2c

and c2h channels ÿ Verify that the data written to the device matches the data

read from the device ÿ Report a pass or fail completion status to the user Using the

xdma_test application is simple. Simply enter the following

command in the open Powershell window:

The execution result of .

\xdma_test.exe is shown in Figure 2-46.

```
PS C:\Users\TANG\Desktop\xdma_driver_win_src_2017\build\x64\bin> .\xdma_test.exe
Detected XDMA AXI-MM design.
Found h2c_0 and c2h_0:
    Initiating H2C_0 transfer of 4096 bytes...
    Initiating C2H_0 transfer of 4096 bytes...
    Transfers completed. Comparing data... OK!
Found h2c_1 and c2h_1:
    Initiating H2C_1 transfer of 4096 bytes...
    Initiating C2H_1 transfer of 4096 bytes...
    Transfers completed. Comparing data... OK!
Could not find h2c_2 and/or c2h_2
Could not find h2c_3 and/or c2h_3
Success!
PS C:\Users\TANG\Desktop\xdma_driver_win_src_2017\build\x64\bin>
```

Figure 2-46 xdma_test test results

From the execution results, we can see that XDMA uses the AXI-MM interface mode and uses a dual-channel h2c and c2h, transmit 4096 bytes of data on each h2c and c2h channel and verify that the data matches.

From the test results of xdma_test and the read and write tests, we can know that there is no problem with the XDMA Vivado project we designed.

2.11 Postscript

Due to the resource limitation of A7-35T, only simple BRAM read and write tests can be performed. The FPGA resources of this project are It has reached 83%, as shown in Figure 2-44. This is achieved by reducing the number of PCIE channels to *2.

*4 will exceed the available resources and fail to compile. In this case, choose a larger FPGA chip, such as A7-50T-200T model accelerator card. For more XDMA development information, please refer to the Milink XDMA and On-time Atomic XDMA Development Guide.

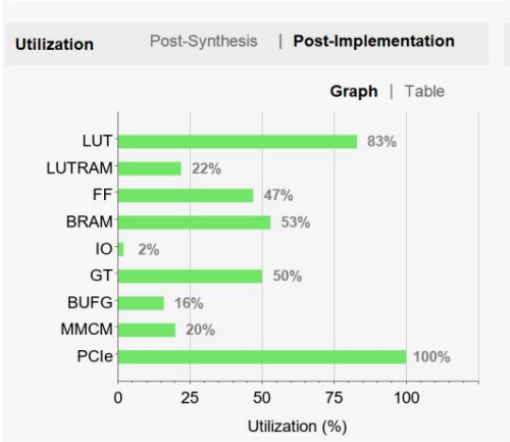

Figure 2-44 A7-35T XDMA resource usage

This guide recommends the Riffa PCIE DMA read and write method in Chapter 3, which is more flexible, takes up less resources, and has a better environment. Simple and open source.

Chapter 3 Riffa Data Loopback

3.1 Riffa Overview

RIFFA (Reusable Integration Framework for FPGA Accelerators) is a reusable integration framework for FPGA accelerators. It is a third-party open source PCIE framework. RIFFA is a simple framework that implements CPU and FPGA data communication through the PCI Express bus. The framework requires a workstation that supports PCIe and an FPGA board with a PCIe connector. RIFFA supports Windows, Linux, Altera and Xilinx.

Data can be sent and received through C/C++, Python, MATLAB or Java drivers. The driver can run on Linux or Windows, and each system supports up to 5 FPGA devices.

The user only needs to write a few lines of simple code to communicate with the FPGA IP core.

RIFFA does not rely on the PCIe Bridge and is therefore not limited by the bridge implementation. RIFFA uses direct memory access (DMA) transfers and interrupt signals to transfer data. This achieves high bandwidth on the PCIe link, running at speeds up to PCIe link saturation point.

Figure 3-1 shows the performance of the design using 32-bit, 64-bit, and 128-bit interfaces. The solid line is the theoretical maximum bandwidth. The dashed line is the maximum achievable bandwidth. PCIe Gen 1 and 2 use 8-bit/10-bit encoding, which limits the maximum achievable bandwidth to 80% of the theoretical value. Our experiments show that RIFFA can achieve 80% of the theoretical bandwidth in almost all cases. The 128-bit interface achieves 76% of the theoretical maximum.

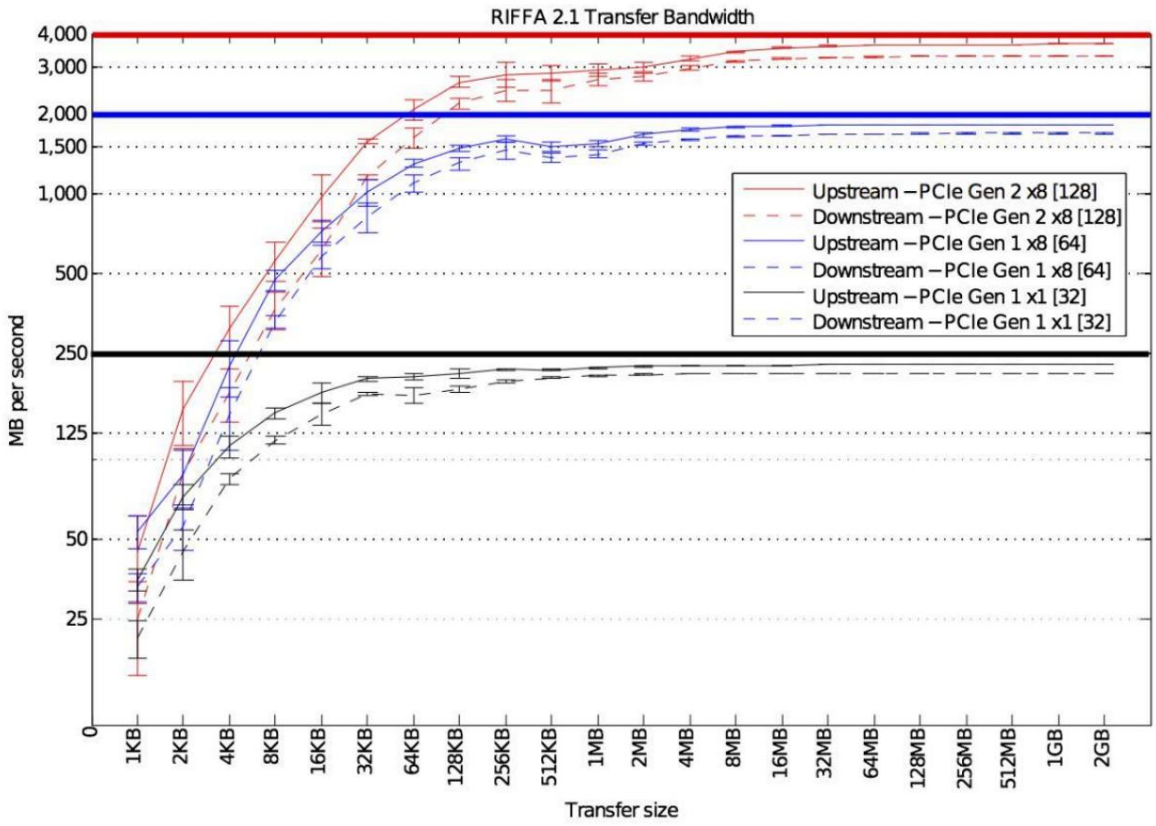

Figure 3-1 Riffa theoretical and actual bandwidth

RIFFA 2.2.2 supports: 1.

Supports VC707, ZC706 and Xilinx IP Core 7 series integrated blocks for PCI Express, providing
Example designs for VC707 and ZC706 power boards.

2. Support VC709 and similar boards with Xilinx IP Gen3 integrated blocks for PCI Express. Provide sample designs for VC709.

3. Supports Stratix V, Cyclone V and similar boards with DES-Net, hard core for PCI Express
(Avalon Streaming Interface). Provides a sample design for the DE5-Net board.

4. Supports Stratix IV, Cyclone IV, and Arria 11 devices of DE4. Provides design examples for DE4 board.
Pre-configuration supports all 64-bit and 128-bit Avalon Streaming interfaces.

3.2 RIFFA Architecture

The RIFFA architecture is shown in Figure 3-2.

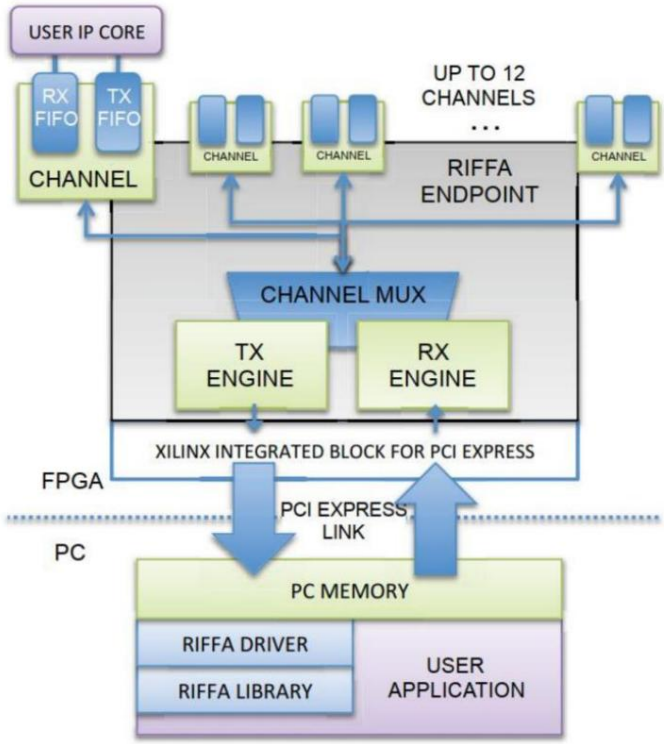

Figure 3-2 RIFFA architecture

In terms of hardware, the interface is simplified so that data can be easily retrieved and stored through FIFO.
RIFFA's RX and TX DMA Engine modules are implemented using a scatter-gather method. The RX Engin module collects valid data from the host computer and sends it to the Channel module after collection. The TX Engin collects data from the Channel module and packages it and sends it to the PCI Express endpoint. According to the PCIe link configuration, the RIFFA interface supports 32-bit, 64-bit, and 128-bit widths, and plans to provide support for the 256-bit interface of the PCIe Gen3 endpoint.

The PC receives the FPGA board data by calling the library function fpga_recv by the user application, and then the FPGA starts

The user application thread enters the kernel driver and then starts to receive read requests from the upstream FPGA, sends the data in packets, and waits for it to arrive if no request is received.

After starting the send function, the server will build a list of hash collection elements, put the data storage address and length information into it, and write it to the shared buffer. The user application sends the buffer address and data length information to the FPGA. The FPGA reads the scatter collection data and then issues a data write request for the corresponding address.

If there are multiple addresses for the element list, the FPGA will issue multiple requests through interrupts.

After all the data moved by TX is written into the buffer, the driver reads the number of bytes written by FPGA to confirm whether it is consistent with the length of the sent data. In this way, the transmission is completed. The process is shown in Figure 3-3.

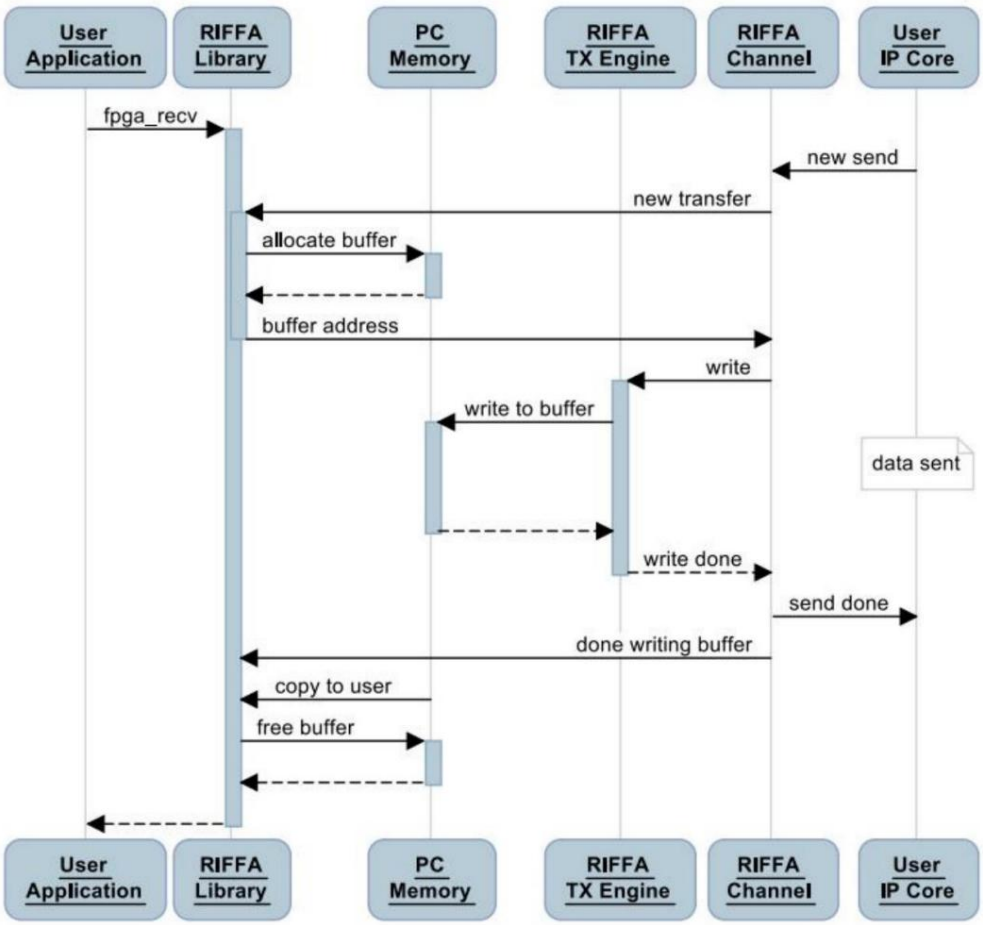

Figure 3-3 FPGA to PC transfer process

The process of sending data from PC to FPGA board is similar to the process of receiving data from PC to FPGA board, as shown in Figure 3.4. At the beginning, the user application calls the library function fpga_send, the transmission thread enters the kernel driver, and then the FPGA starts the transmission.

Start fpga_send, the server will request some space, write the data to be sent into it, then create a scatter-gather list, put the address and length of the stored data into it, and then write the address of the scatter-gather list and the data to be sent into it.

After receiving the list address, FPGA reads the information in the list and then sends the corresponding

The read request of address and length is then stored and sent to the FPGA board together.

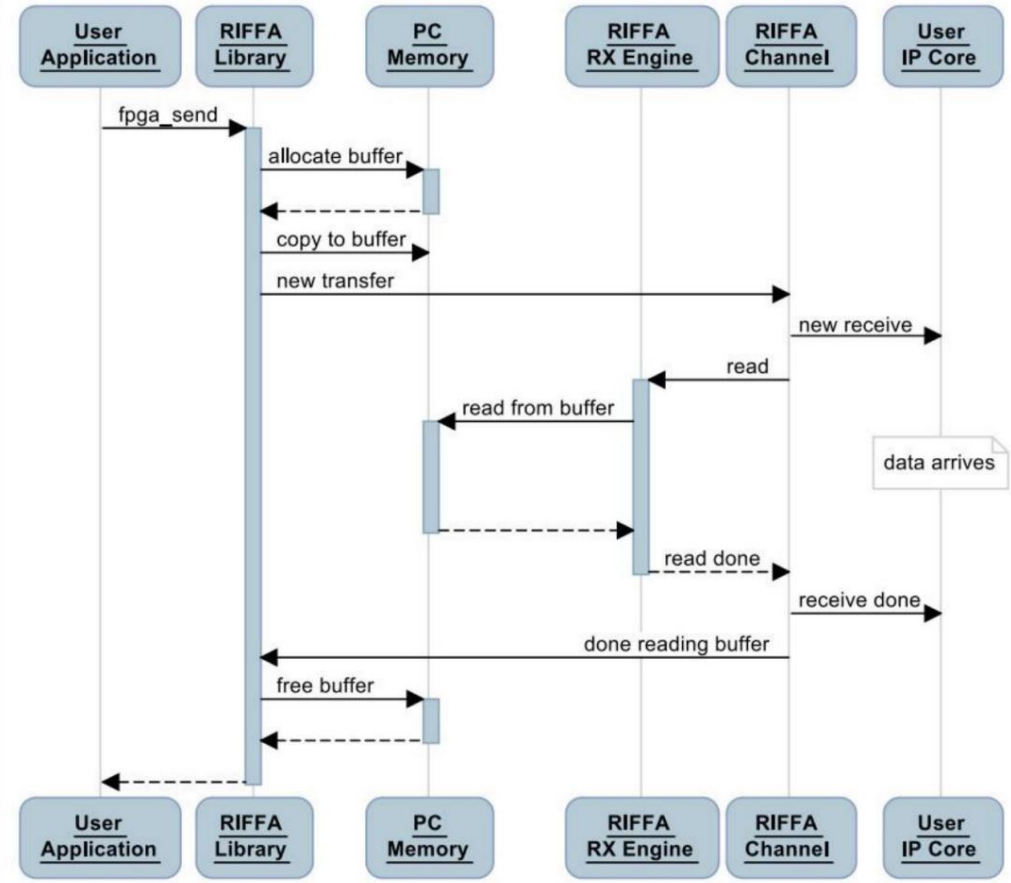

Figure 3-4 PC transmits data to FPGA

3.3 RIFFA driver installation

Run setup.exe or setup_dbg.exe in the provided riffa_2.2.2/nstall/windows/install file directory

Install the kernel driver and C/C++ library by using the installer, as shown in Figure 3-5.

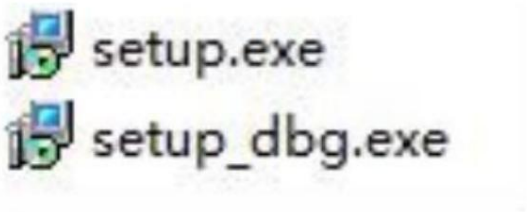

Figure 3-5 RIFFA Windows driver installation package

The RIFFA driver needs to be installed in WIndow7 environment and needs to be installed as an administrator program.

You need to disable the driver signature to run it, as shown in Figure 3-6.

Then follow the wizard and keep clicking Next until the installation is complete, as shown in Figure 3-7 and Figure 3-8.

Figure 3-6 Enable compatibility and run as administrator

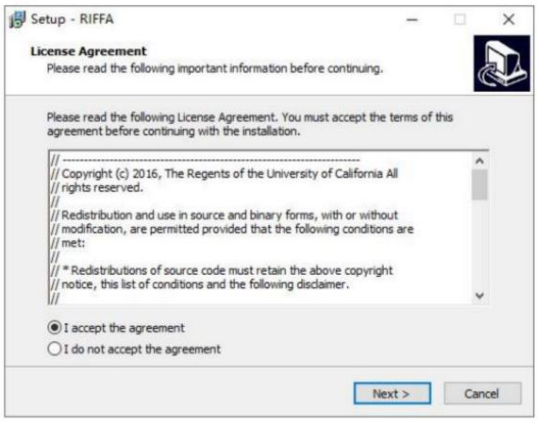

Figure 3-7 Agree to the agreement

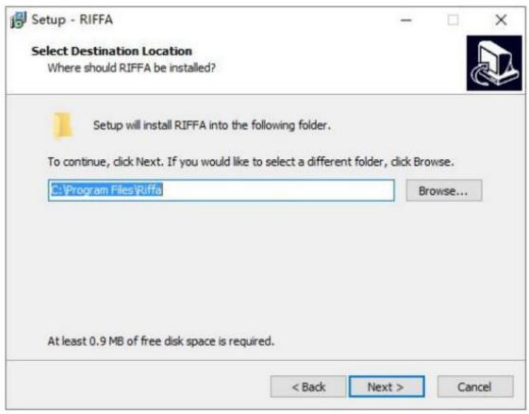

Figure 3-8 Select installation location

After the driver is installed, plug the FPGA accelerator card with the program burned into the PC and you can identify it in the device manager. Don't go to Riffa Device.

Note: For newer WIN10 or WIN11 computers, the Riffa driver signature is no longer maintained for a long time. If the Riffa device is recognized but the driver is not signed, you can solve it in the following way.

Method 1: Enable the test mode according to Section 2.7.

Method 2: Disable Secure_Boot in the computer BIOS. Method 3:

Select Disable Driver Enforced Signature in Windows Advanced Startup.

3.4 RIFFA library usage

RIFFA library supports Java, Python, C/C++, and Matlab. This guide uses C/C++ as an example, as C/C++ is suitable for writing The host computer is shown in Figure 3-9. The other three languages are similar to the above two. Please study them yourself if you need to know more.

The use of RIFFA library mainly consists of 4 functions, which are similar to regular file reading and writing. There are 4 functions: open, write, read, and close. Reading and writing are performed through these 4 functions. First, open the PCIE device through open, and then write or read data to the PC through write or read. If reading and writing are not required, close it with close.

The following uses Matlab and C language to briefly introduce the use of the riffa library. These library functions are used in the driver function. It is in the number. If you need to know more, you can check the source code yourself.

How to use C/C++ library:

```
#include <stdio.h>;
#include <stdlib.h>;
#include <riffa.h>;
#define BUF_SIZE (1*1024*1024)
unsigned int buf[BUF_SIZE];
int main(int argc, char* argv[])
{
    fpga_t * fpga;
    int fid = 0; // FPGA id
    int channel = 0; // FPGA channel
    fpga = fpga_open(fid);
    fpga_send(fpga, channel, (void *)buf, BUF_SIZE, 0, 1, 0);
    fpga_recv(fpga, channel, (void *)buf, BUF_SIZE, 0);
    fpga_close(fpga);
    return 0;
}</riffa.h></stdlib.h></stdio.h>

```

Figure 3-9 RIFFA C/C++ library usage example

The driver function corresponding to C language first uses fpga_open to open the pcie device, then uses fpga_send to send data, uses fpgarecv to receive data, and if no further acquisition is needed, uses fpga_close to close the pcie device. Finally, please pay attention to the above specifications when debugging

pcie, otherwise it will cause a blue screen restart at the least, or even cause system damage at the worst.

3.5 RIFFA Vivado Project Construction

This guide is based on the Riifa reference example on Github. Official Github:

URL: https://github.com/KastnerRG/riffa For the original

example, please refer to the example example in the riffa2.2.2 folder. The

following describes how to build a Riffa data loopback test example.

Open the Vivado software, select Create Project and click Next, as shown in Figure 3-10.

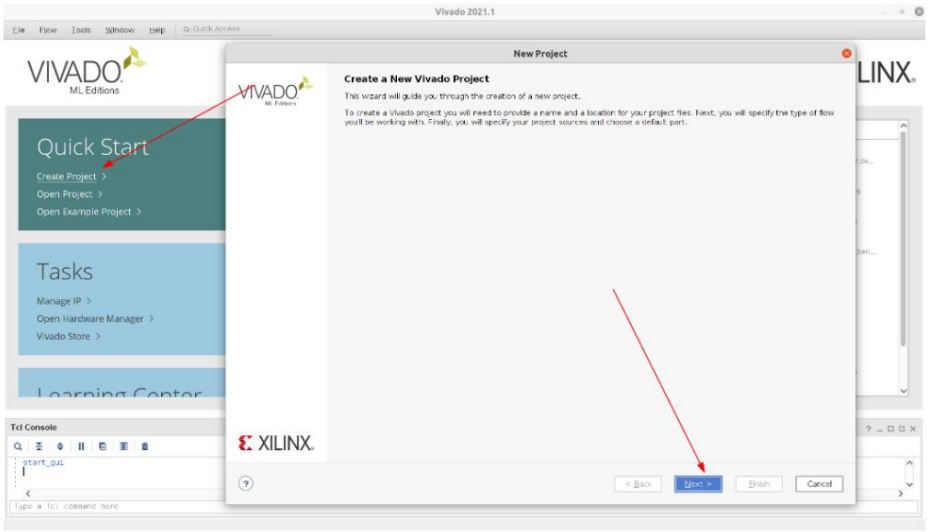

Figure 3-10 Creating a Vivado project

Enter the project name and project storage path, taking m2_artix7_riffa as an example, as shown in Figure 3-11. Click Next.

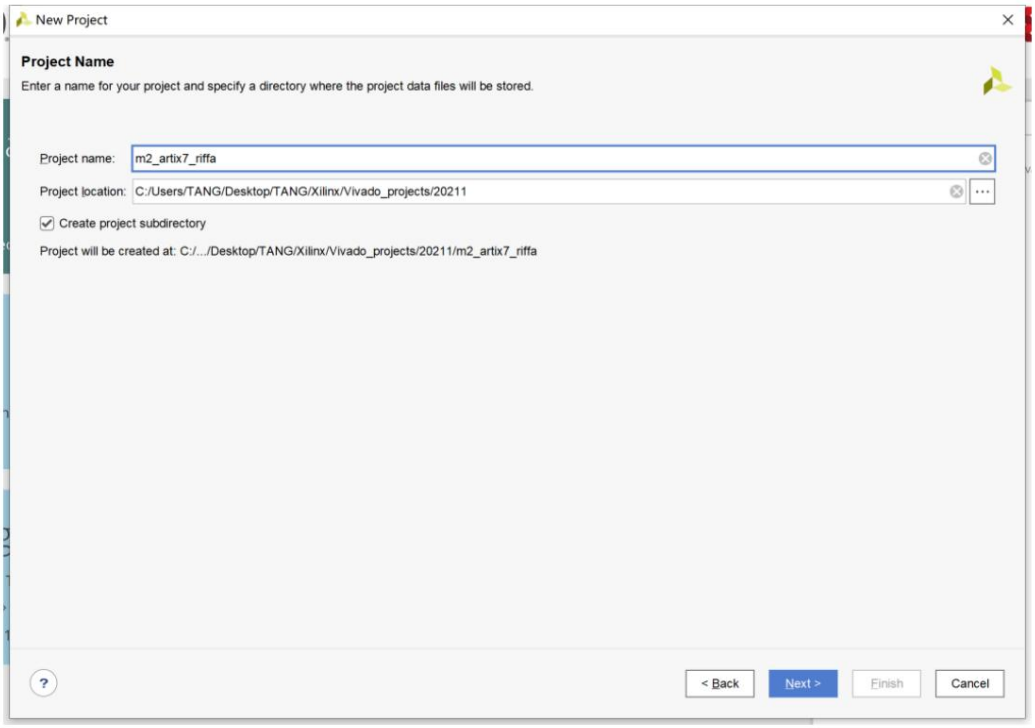

Figure 3-11 Enter the project name and path

Select RTL Project and check "Do not specify source files at this time", as shown in Figure 3-12.

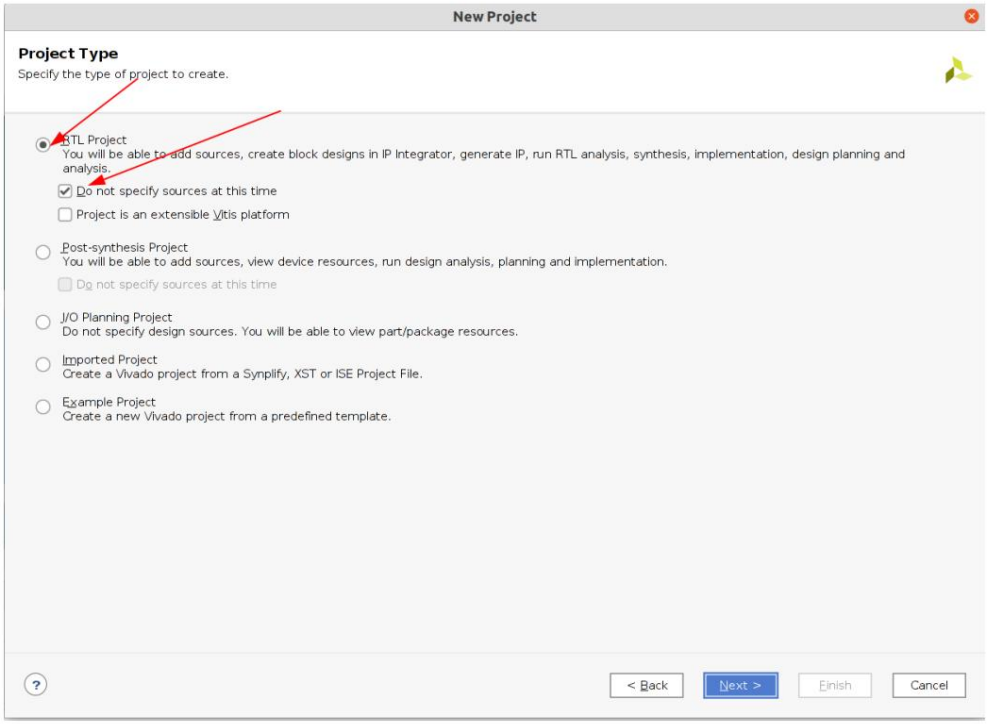

Figure 3-12 Select RTL Project Select

the chip according to the chip model and click Next. Take XC7A35T-2FGG484I as an example, as shown in Figure 3-13.

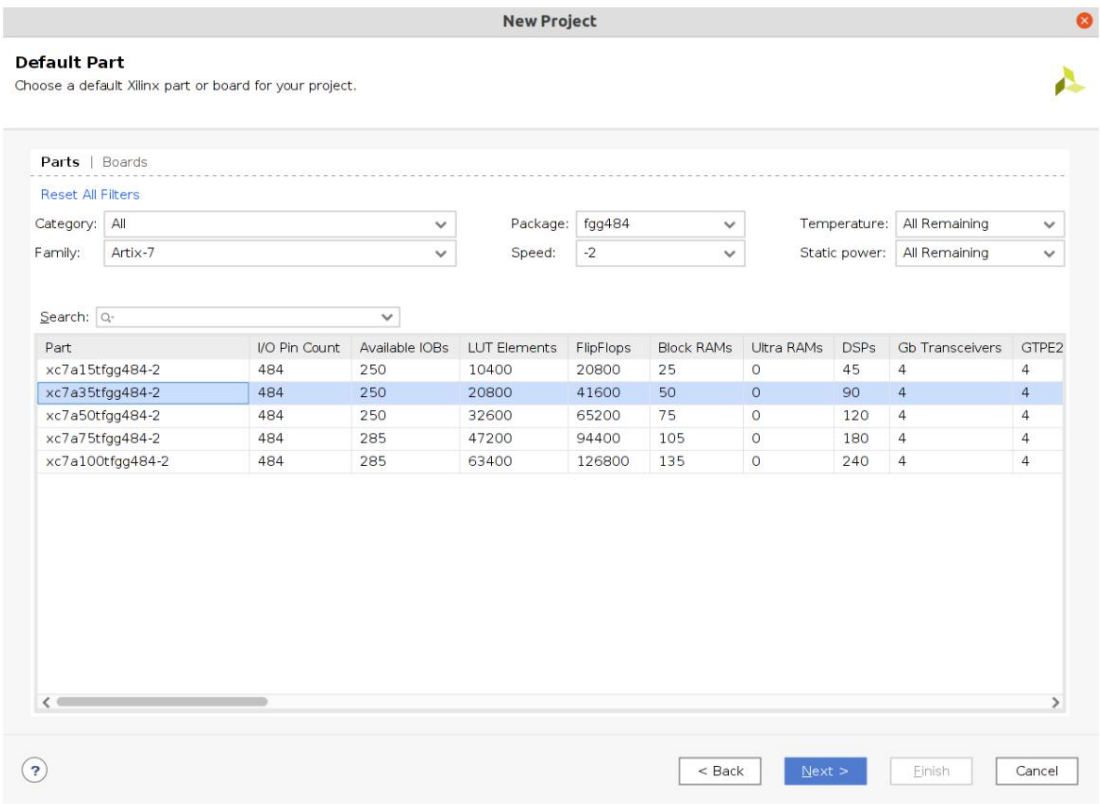

Figure 3-13 Select chip model

Click Finish in the new window to complete the project creation, enter the project main interface, and select IP catalog as shown in Figure 3-14.

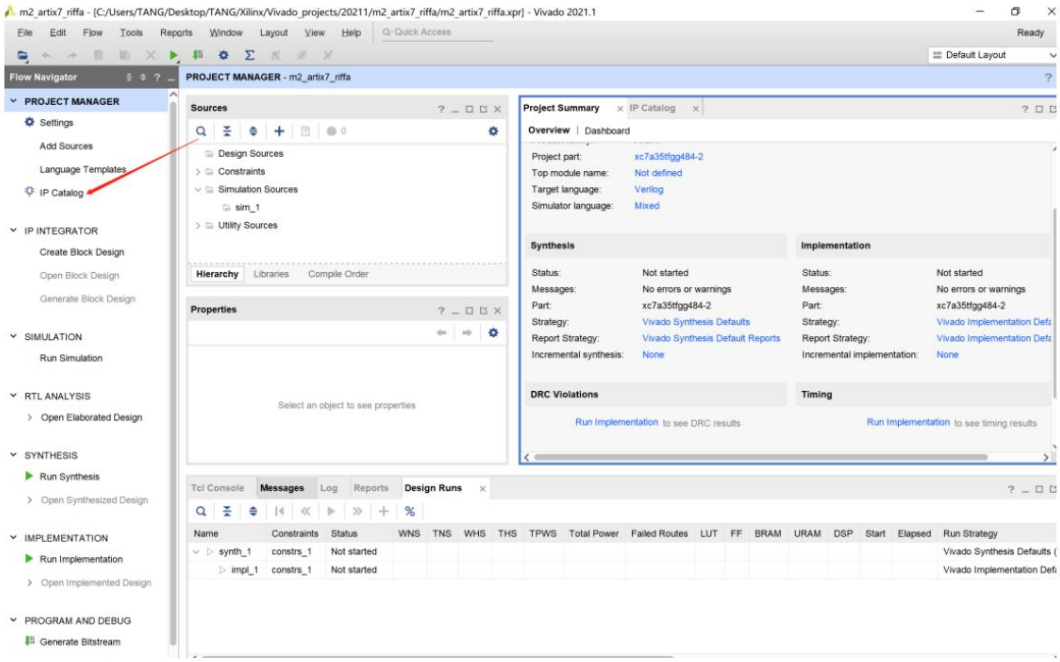

Figure 3-14 Click IP Catalog. In the

Search box of IP Catalog, enter pcie, find 7 Series Integrated Block for PCI Express and double-click it to enter the configuration interface, as shown in Figure 3-15.

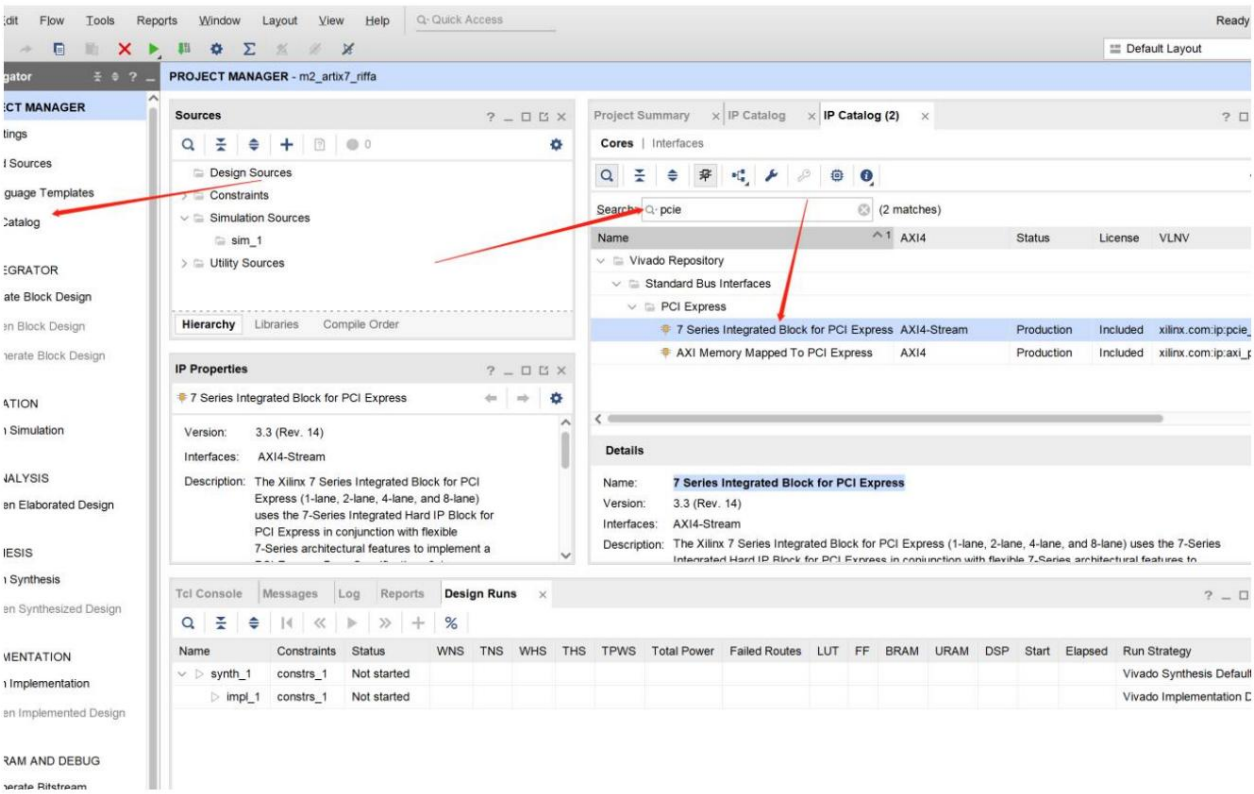

Figure 3-15 Find and add PCIE IP

3.6 PCIE IP Configuration

This section describes in detail the configuration contents of PCIE IP and the meaning and functions of the configuration items.

7 Series Integrated Block for PCI Express is a scalable, high-bandwidth and reliable serial interconnect building block on 7 series FPGAs from Xilinx. It is used to build PCIE applications and includes a complete transaction layer, data link components, and a 32-bit FPGA. The 7 Series PCI Express Integrated Block (PCIe) solution supports 1-lane, 2-lane, 4-lane and 8-lane endpoint and root port configurations at speeds up to 5gb/s (Gen2), and the interface uses AMBA's axi4 stream interface.

mouth.

3.6.1 Basic

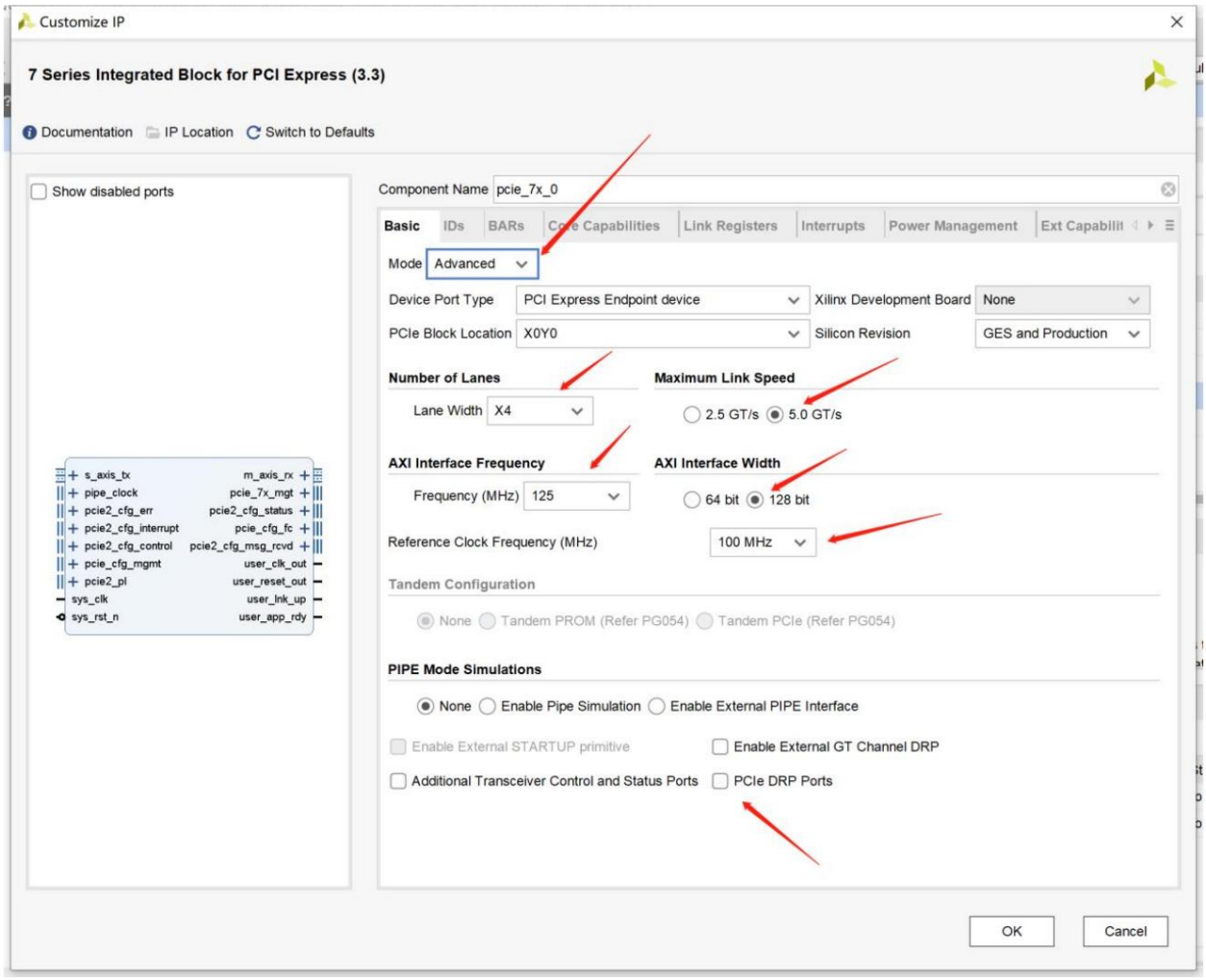

Figure 3-16 Basic Configuration

Items Figure 3-16 shows the Basic configuration items, which include the

following main contents: 1. Mode: You can choose Advance or base, advanced or basic mode, here select Advance.

2. Device Port Type: You can choose Endpoint device, LegacyEndpoint device, Root Complex, that is, endpoint device or root complex node.

3. Xilinx Development Board: If it is an official development board, you can choose this for one-click configuration.

4. PCIe Block Location: Select the PCIE location. For Artix chips, there is only one option.

5. Number of Lanes: Lane Width: Select the number of PCIE lanes, select X4.

6. Maximum Link Speed: Maximum link speed, PCIE1.0 is 2.5GT/s, PCIE2.0 is 5GT/s, select 5GT/s.

7. AXI Interface Frequency: The clock of the AXI Stream interface of the user-side interface. Here, select 125MHz.

8. AXI Interface Width: The data width of the AXI Stream interface of the user-side interface. Here, select 128 bits.

9. Reference Clock Frequency: The reference clock of the external PCIE. For endpoint devices, it is provided by the computer motherboard. Supply FPGA board, here choose 100MHz.

10. PCIE DRP Ports: Control and detect PCIE physical layer, can change speed, bit width, etc. Not used here, uncheck.

3.6.2 IDs

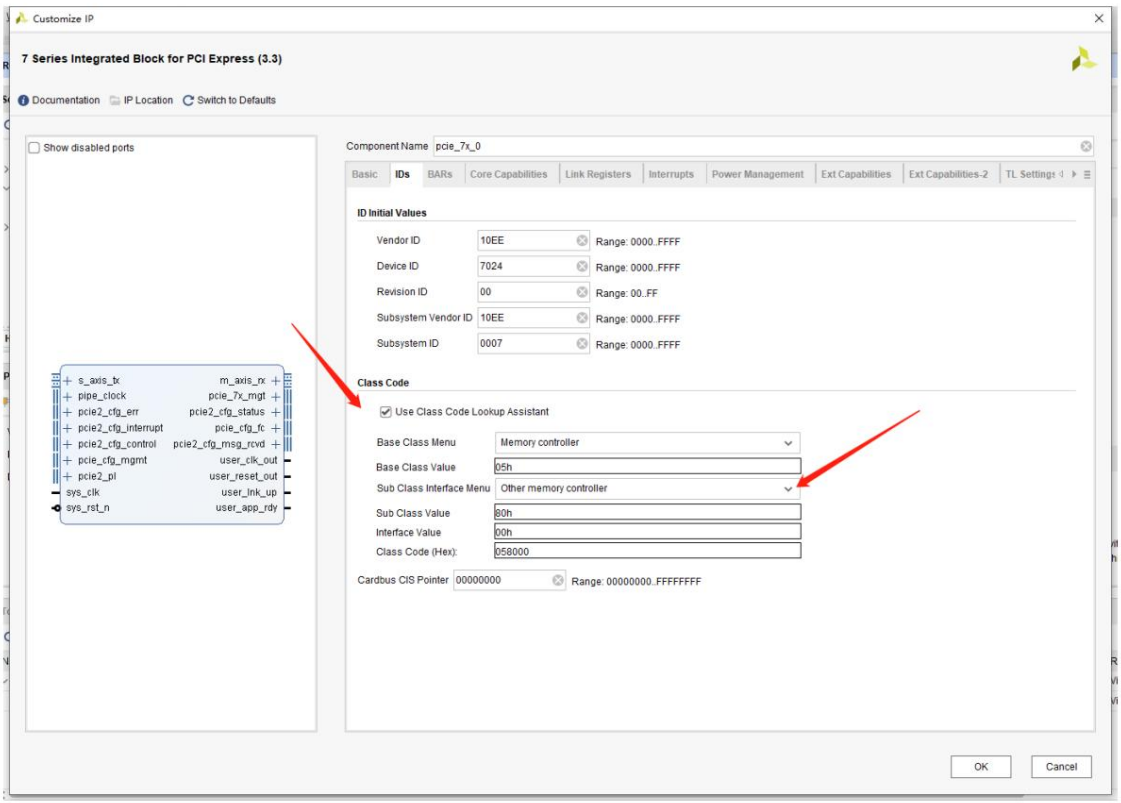

Figure 3-17 IDs Configuration

The IDs page is mainly about some data used to initialize the Bar space, including device ID, manufacturer ID, class code Etc., which is related to the driver program. Generally, it is not modified unless you customize the driver yourself. Otherwise, the driver program will not recognize PCIE after modification. Device. Check Use Class Code Lookup Assistant, and select Other memory controller in Sub Class Interface Menu, as shown in Figure 3-17.

1. Vendor ID: manufacturer ID.

2. Device ID: device ID.

3. Revision ID: Revision ID.

4. Subsystem Vendor ID: Similar to the vendor ID.

5. Subsystem ID: Similar to the device ID.

6. Class Code: PCIE class code, related to the driver.

3.6.3 BARs

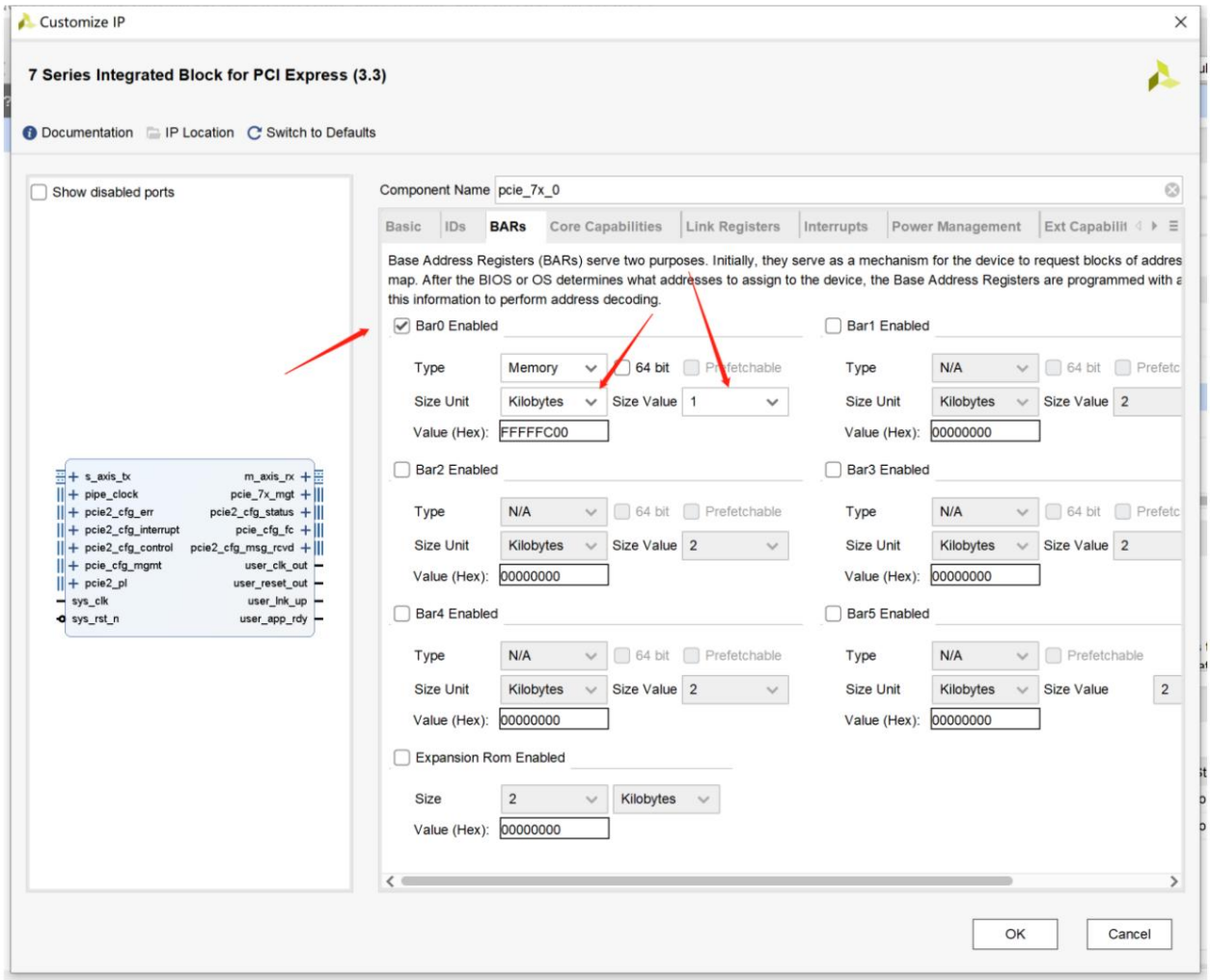

Figure 3-18 BARs configuration

The BAR page is used to configure the Bar space, which mainly plays the role of address mapping, mapping the computer address space with the PCIE address space. This page can open multiple BAR spaces. Here we set only BAR0 and select the size as 1Kbyte.

3.6.4 Core Capabilities

The Core Capability page is about the configuration of PCIE performance, including the maximum value of the effective load data of the transaction layer. and data buffering, etc. The configuration is shown in Figure 3-19.

1. Max Payload Size: The maximum value of the payload data can be set to 128, 256, 512, or 1024 bytes. This value can improve PCIE efficiency, but it also consumes more FPGA resources, mainly BRAM and other resources. Here we set The size is 256 bytes.

2. BRAM Configuration Options: Configure the PCIE buffer size, which can be Good or High. Good consumes less BRAM resources, while High consumes more BRAM resources. Here we choose High, and you can also check the buffer optimization option.

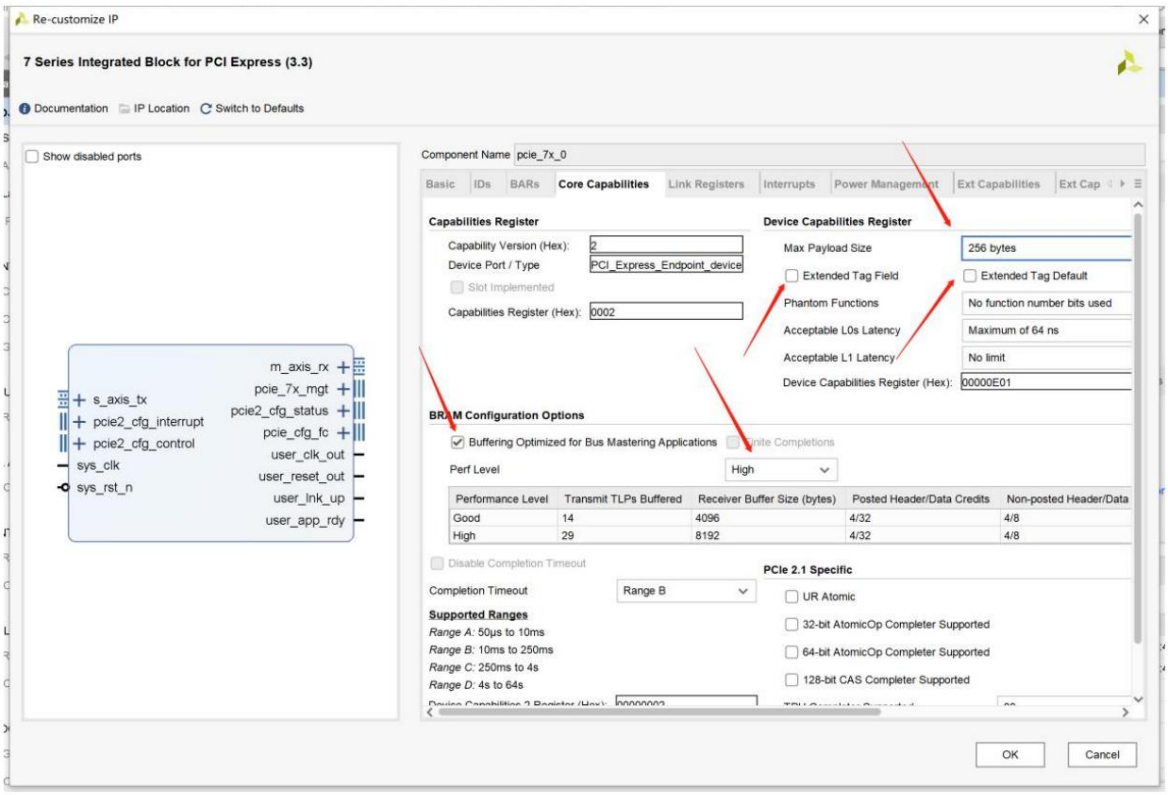

Figure 3-19 Core Capabilities settings

Keep other values as default.

3.6.5 Link Registers

We do not make any configuration on this page, just keep the default settings, as shown in Figure 3-20.

Link Register is a connection register used to inform the computer of the FPGA board speed and channel. The computer recognizes it as PCIE1.0, 2.0, 3.0, or X1, X2, X4, etc. The computer accesses this address to obtain the information. The address is accessed during the PCIE enumeration process. Ask for this address.

1. Support Link Speed: Link speed. If it is PCIE1.0, this value is 1, PCIE2.0 is 2, and PCIE3.0 is 3.

2. Maximum Link Width: Link channel, X1, X2, X4, X8 are 1, 2, 4, 8, etc.

3. Link Control Register: Link address control register.

4. Hardware Autonomous Speed Disable: Whether to disable the link self-matching function.

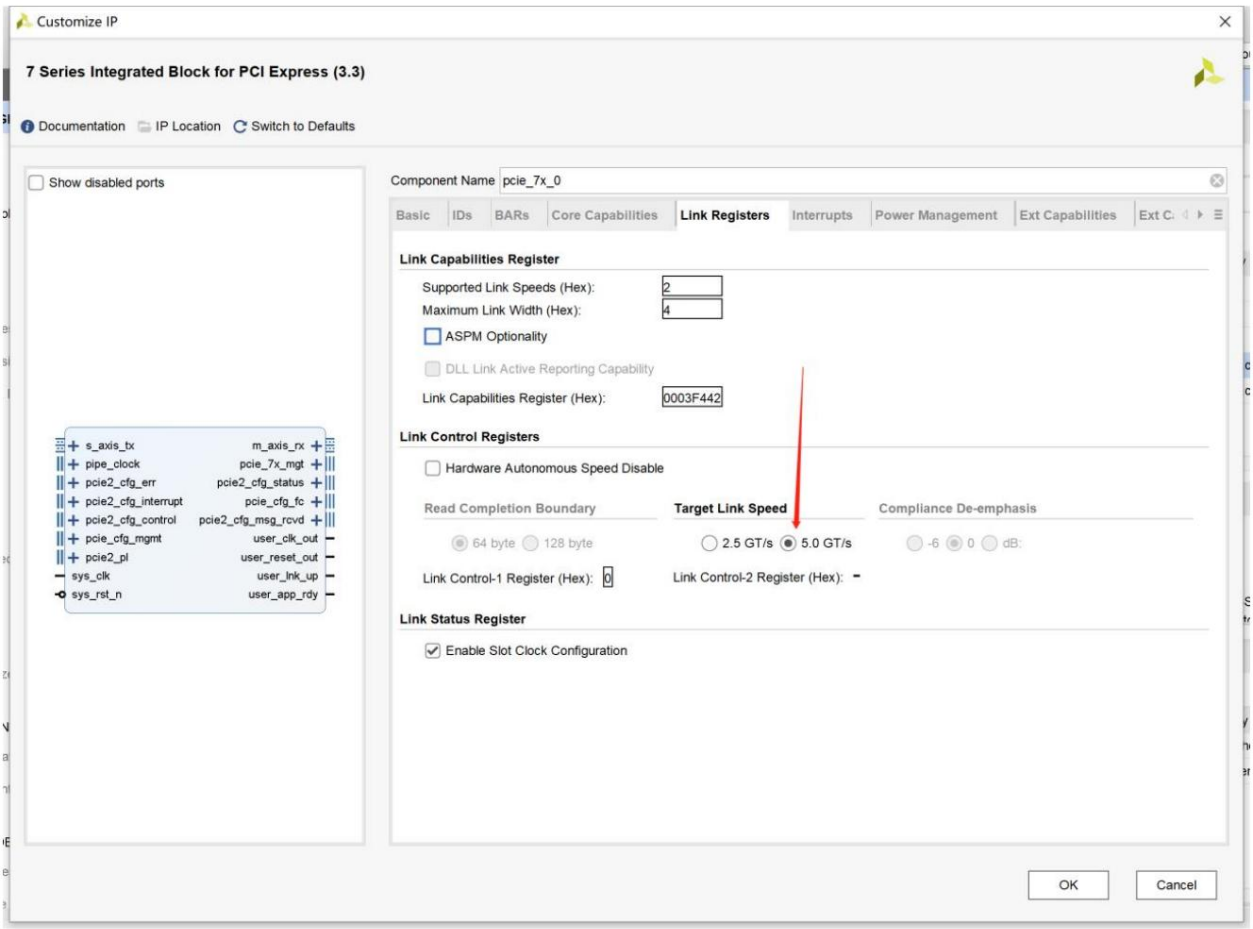

Figure 3-20 Link Registers Configuration

3.6.6 Interrupts

Interrupts is related to PCIE interrupts, and the configuration is shown in Figure 3-21.

1. Enable IntX: This is a pin interrupt, which is not commonly used now. There is no such function on the FPGA board, so I Select uncheck to turn off.

2. MSI Capability: Message interrupt, a mechanism for sending interrupts through data packets.

3. MSIx Capability: Another type of message interrupt. It cannot be enabled at the same time as MSI Capability. Only one of the two can be enabled.

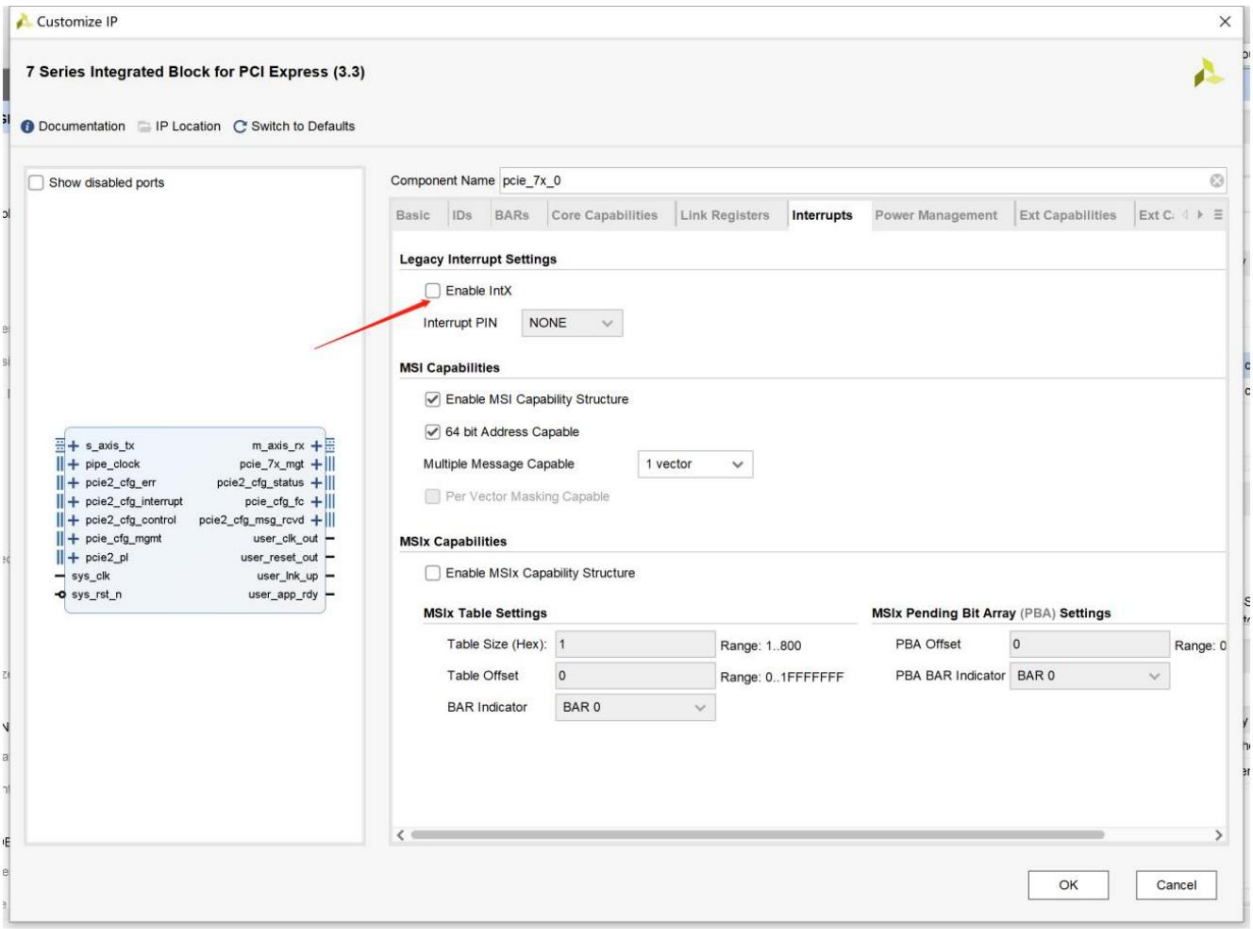

Figure 3-21 Interrupts Configuration

3.6.7 Power Management

The Power Management Register page is about the power management registers, including hot plug, soft reset, etc.

We will not make any changes here for the time being, as shown in Figure 3-22.

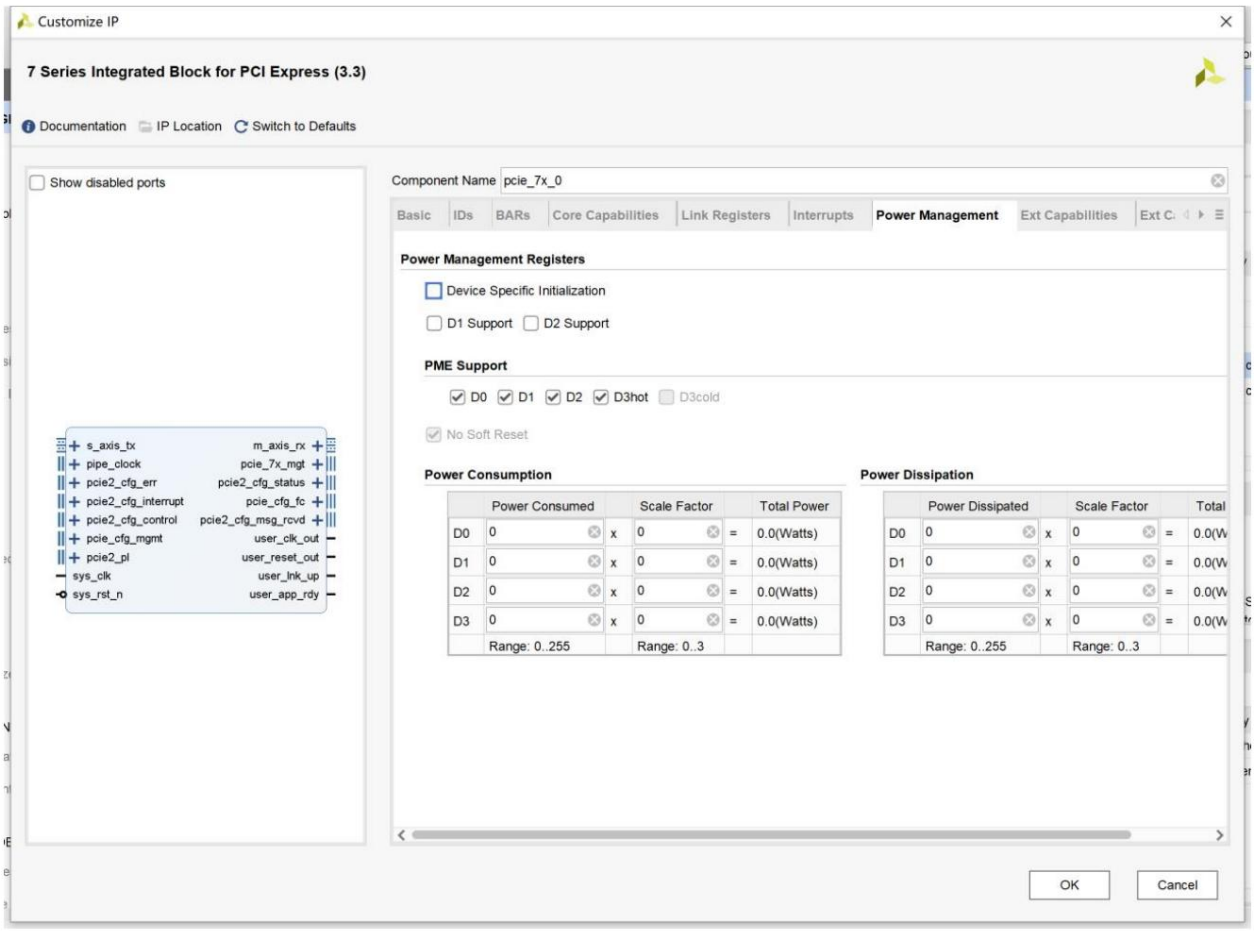

Figure 3-22 Power Management Configuration

3.6.8 Ext Capabilities

The specific content of this page is not introduced here, just keep the default, as shown in Figure 3-23.

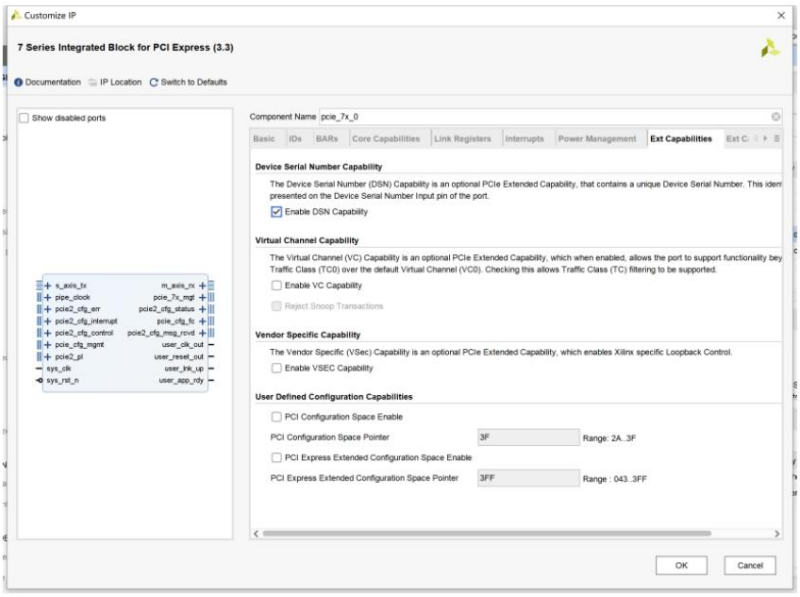

Figure 3-24 Ext Capabilities configuration remains the default

3.6.9 Ext Capabilities-2

The specific content of this page is not introduced here, just keep the default, as shown in Figure 3-24.

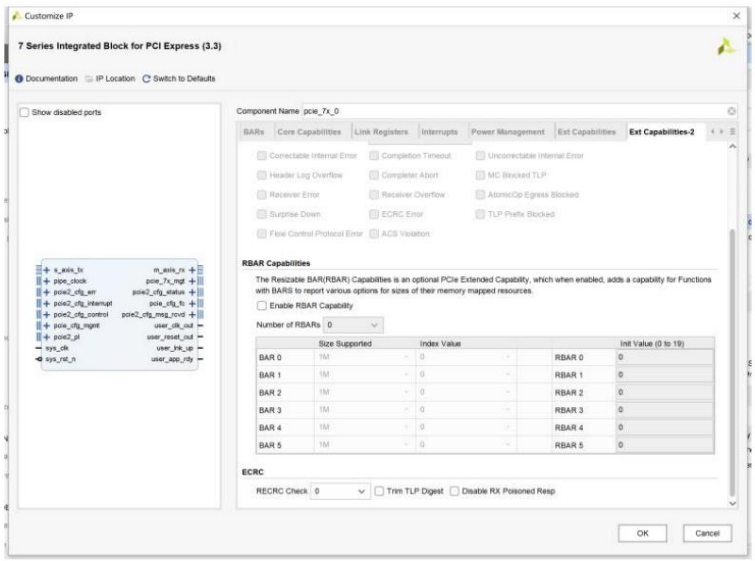

Figure 3-24 Ext Capabilities-2 configuration remains default

3.6.10 TL Settings

TL Settings are also some advanced settings of the transaction layer. Keep the default settings without modification, as shown in Figure 3-25.

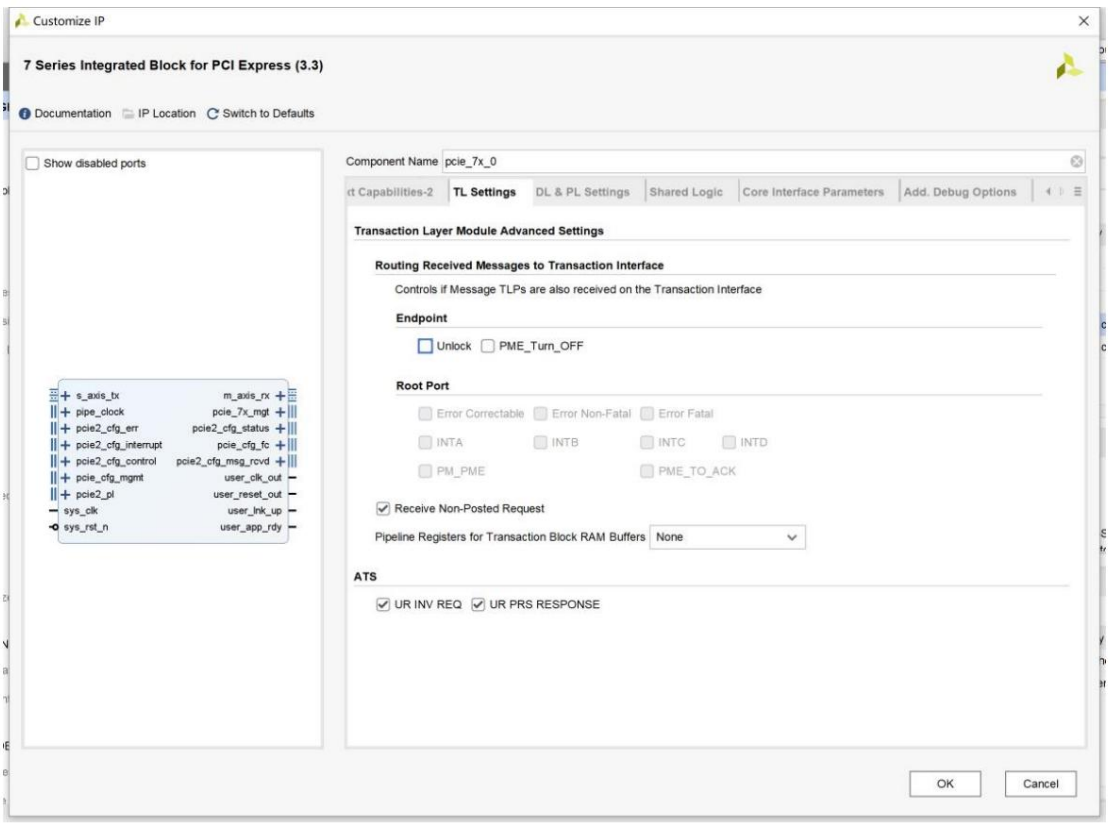

Figure 3-25 TL Settings Keep the default

3.6.11 DL&PL Settings

DL&PL Settings are also some advanced settings of the link layer. Keep the default settings without modification, as shown in Figure 3-26.

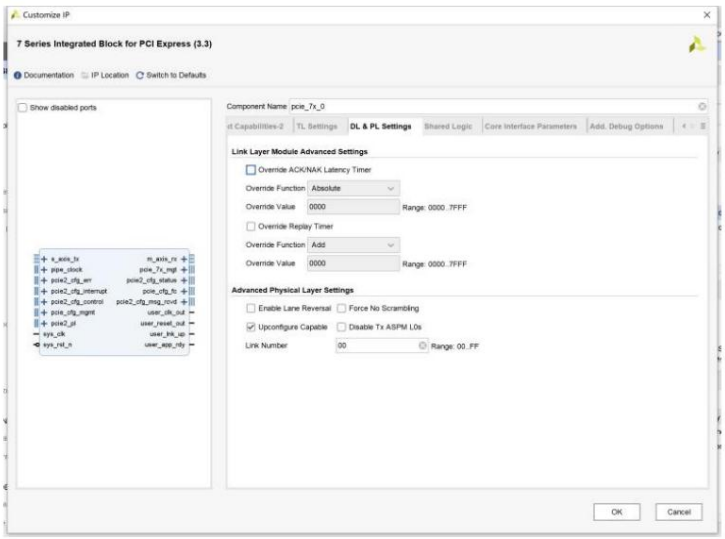

Figure 3-26 DL&PL Settings Keep the default

3.6.12 Shared Logic

Shared Logic is a modification of the IP core generation. It is a generated example design.

Frame, uncheck all options, as shown in Figure 3-27.

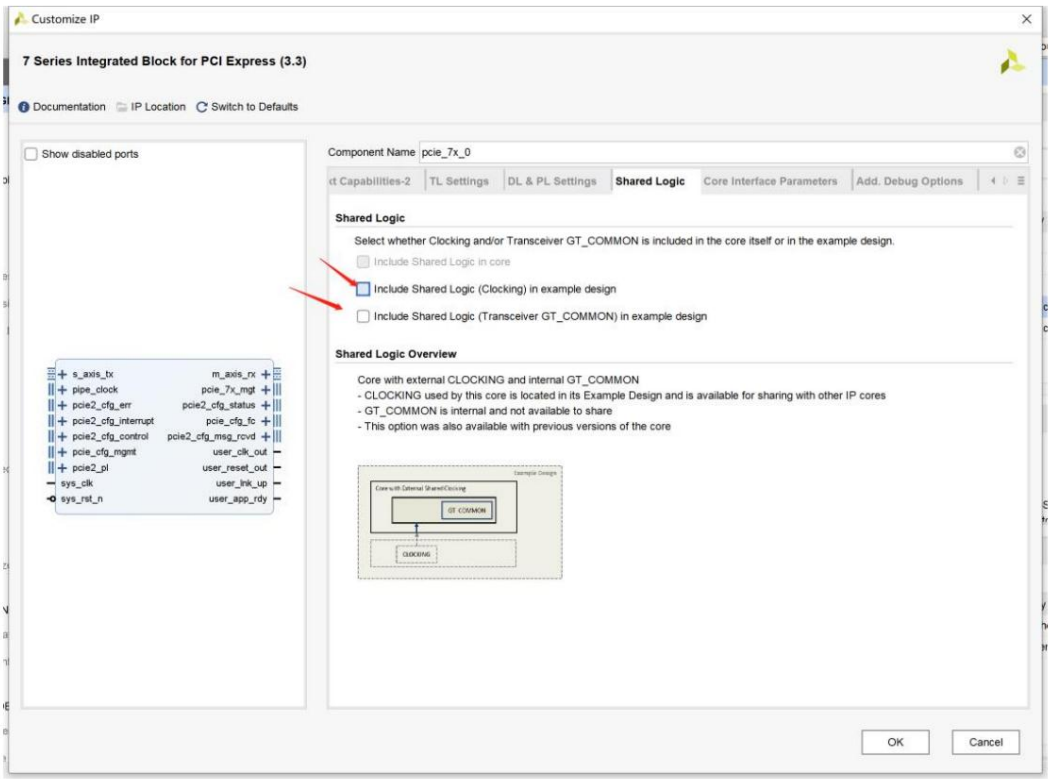

Figure 3-27 Cancel all Shared Logic

3.6.13 Core Interface Parameters

The Core Interface Parameters page is about opening some interfaces. The configuration is shown in Figure 3-28.

1. PL Interface: programming interface, which can be used for certain configurations and is not used.

2. Error RePorting: Error reporting interface, not used.

3. Config Management Interface: Power management interface, not used.

4. Config CTRL Interface: Configure the interface for configuring CTRL, required for riffa, check this option.

5. Config Status Interface: Configure the status interface, riffa requires, check this option.

6. Receive Msg Interface: message interrupt interface, not used.

7. Config FC Interface: Configure FC interface, required for riffa, check this option.

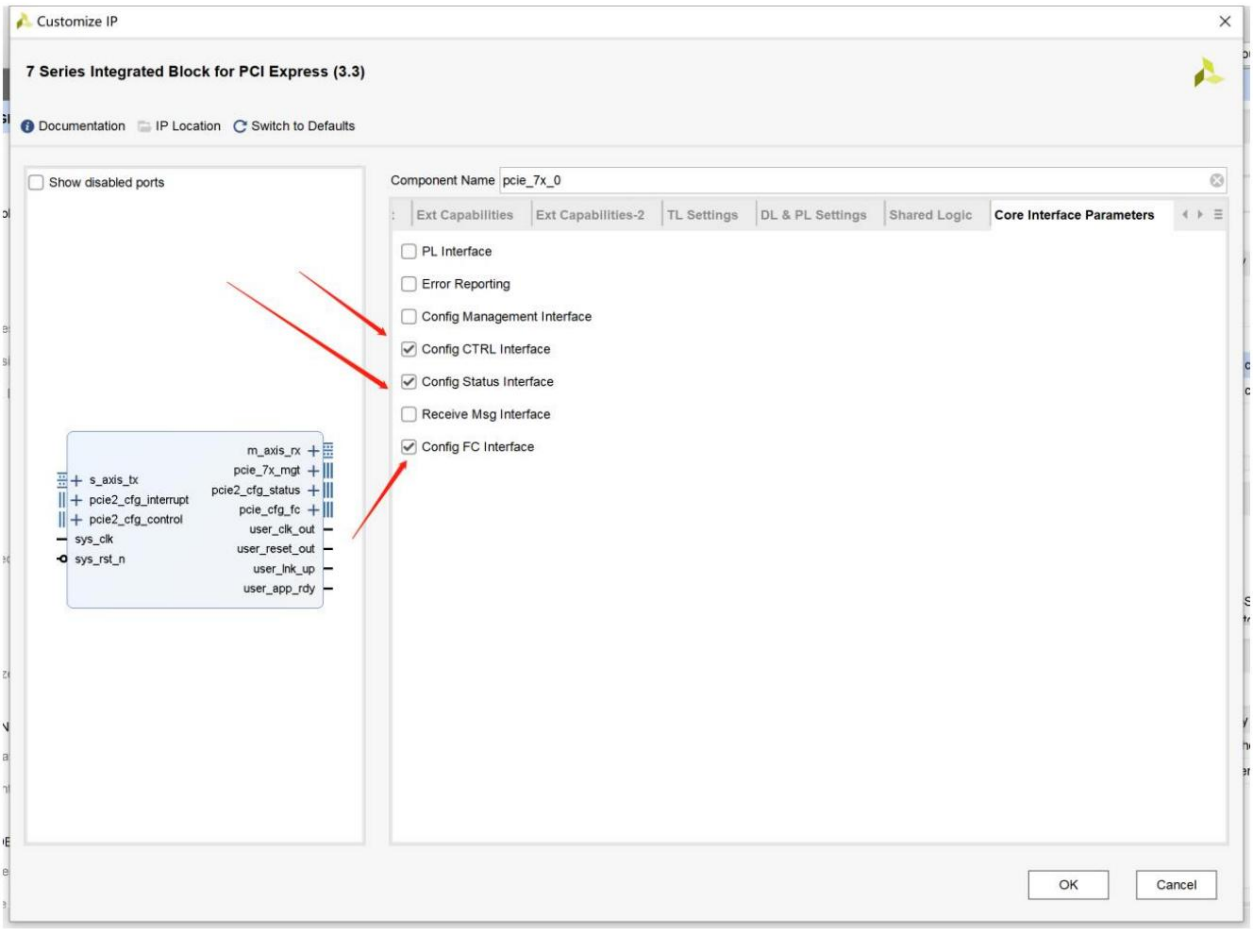

Figure 3-28 Core Interface Parameters Configuration

3.6.14 Add.Debug Options

Debug interface, PCIE has JTAG debug interface, this page is about whether to enable JTAG debug interface, here

We will not use it, so keep it unchecked, as shown in Figure 3-29.

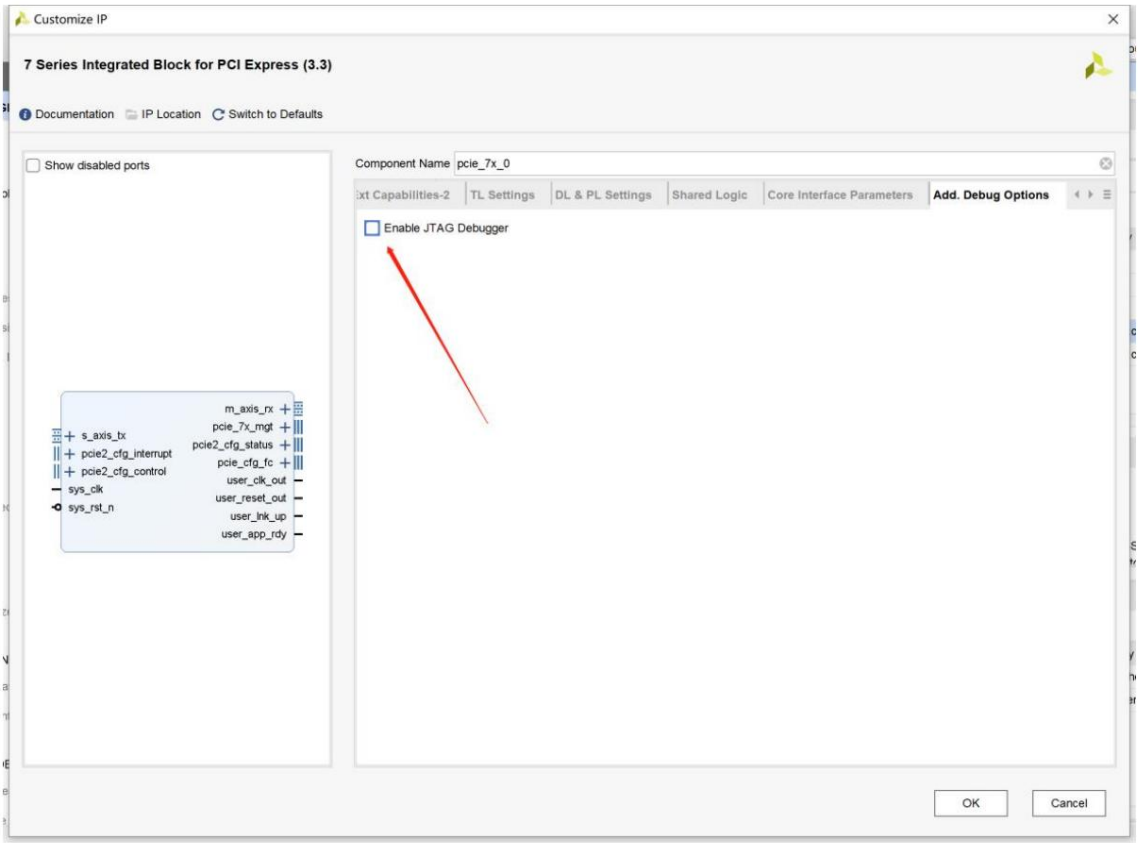

Figure 3-29 JTAG debugging options

At this point, the PCIE IP configuration is complete, click OK to complete. In the pop-up Generate Output File dialog box, select OK.

As shown in Figure 3-30.

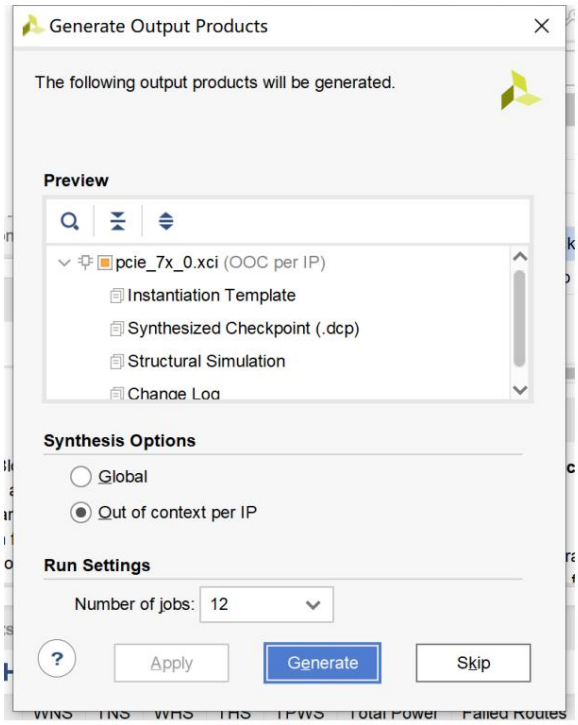

Figure 3-30 Generate output file

3.6.15 Soft Reset Configuration

After the PCIE IP is configured as above, the host restart will cause the card to not be recognized because the host restarts the PCIE in the accelerator card The IP did not restart. This is because the soft reset is not configured. Here we double-click the generated

Open the Power Management option and uncheck No soft Reset, as shown in Figure 3-31. This setting cannot be configured when the IP is first configured. I don't know why. After adding Soft Reset, regenerate the

Into IP.

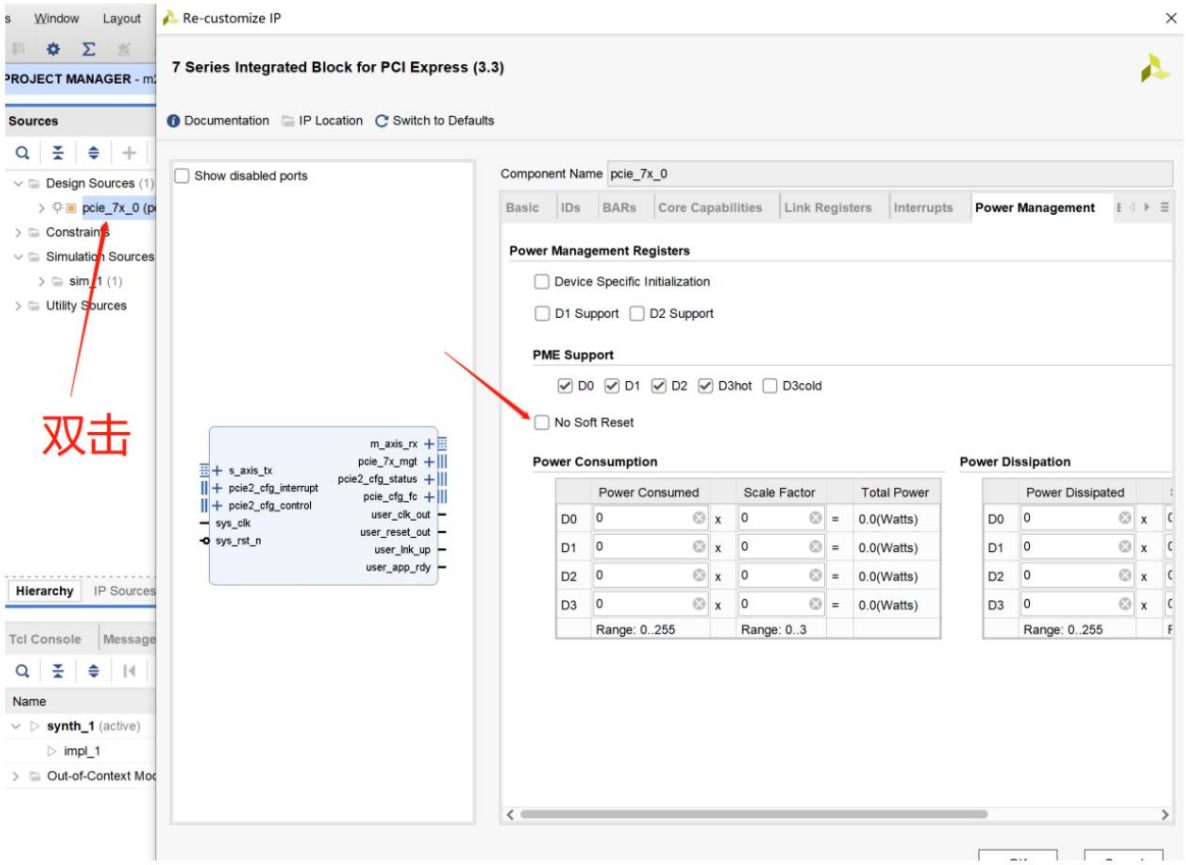

Figure 3-31 Adding soft reset

3.6.16 PCIE channel modification

After Xilinx's PCIE IP is generated, the officially recommended PCIE corresponding GT channel order will be used by default. Since the accelerator card hardware order is different from the officially recommended order, after reconstraining the channel, Vivado software will report a Critical Warning during layout and routing. Although it does not affect the use, here is how to eliminate it.

Open the constraint file generated after the IP configuration is completed as shown in Figure 3-32, and **modify the GT Lane numbers used by the 4 PCIE Lanes indicated by the arrows. (The GTP** channels of A7-35T-100T are **Lane0-Lane3** corresponding to **X0Y0** -X0Y3, and the **GTP** channels of **A7-200T** are **Lane0-Lane3** corresponding to X0Y4-X0Y7) After the modification is completed, save it.

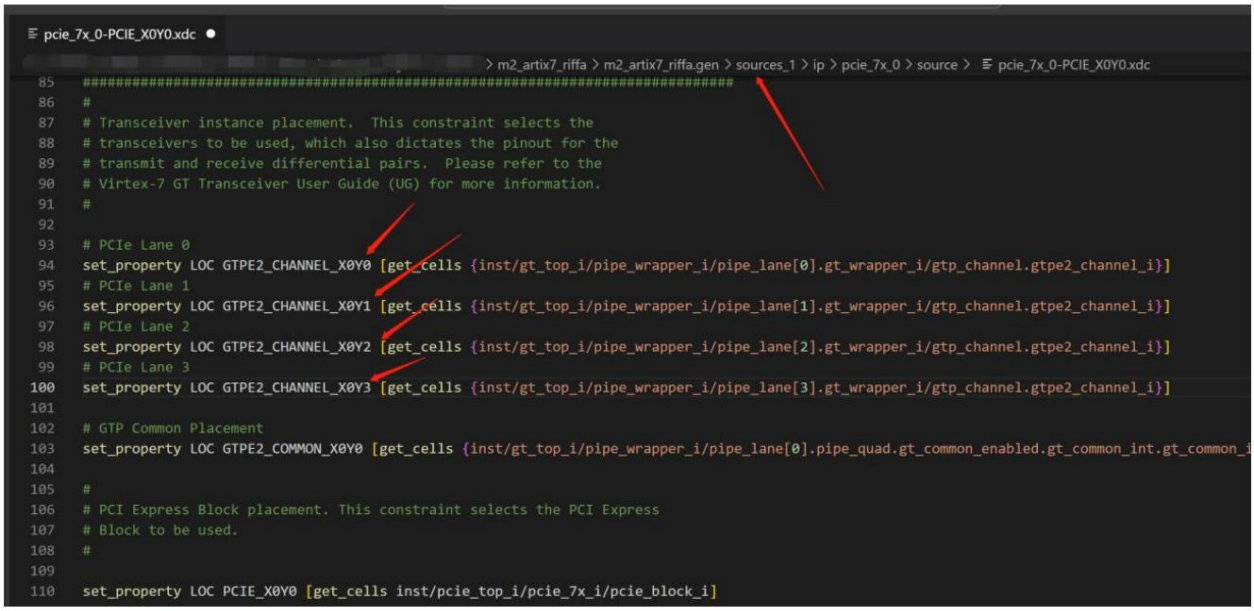

Figure 3-32 Modify PCIE Lane Order

At this point, the PCIE IP configuration is complete.

3.7 Add Riffa RTL source files and constraint files

Here we use the modified Riffa RTL code and constraint files provided in this tutorial.

Copy the .srcs folder in the project directory, as shown in Figure 3-34.

The modified code mainly adds the logic of LED lights and modifies some official content to adapt to IP settings.

The key point is that the following parameters need to be kept consistent with the IP configuration in the top-level source code. module pcie_riffa #(// Number of RIFFA Channels parameter C_NUM_CHNL = 1, // Number of PCIe Lanes parameter C_NUM_LANES = 4, // Settings from Vivado IP Generator parameter C_PCI_DATA_WIDTH = 128, parameter C_MAX_PAYLOAD_BYTES = 256, parameter C_LOG_NUM_TAGS = 5)

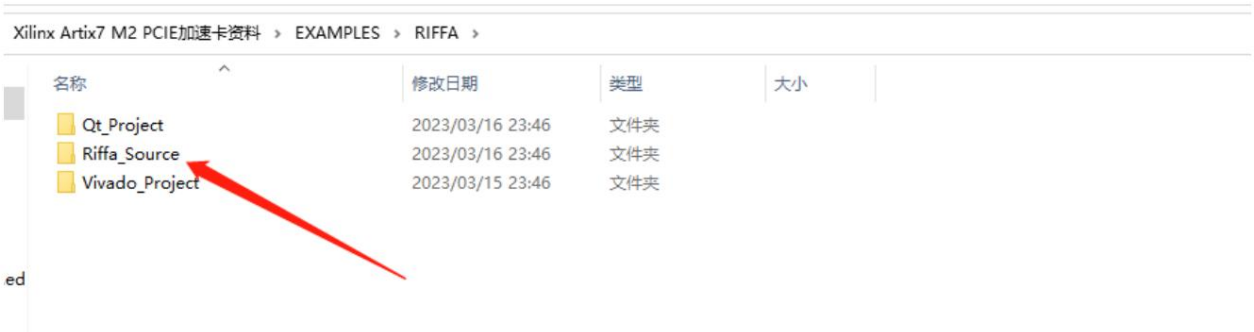

Figure 3-33 Riffa_Source folder

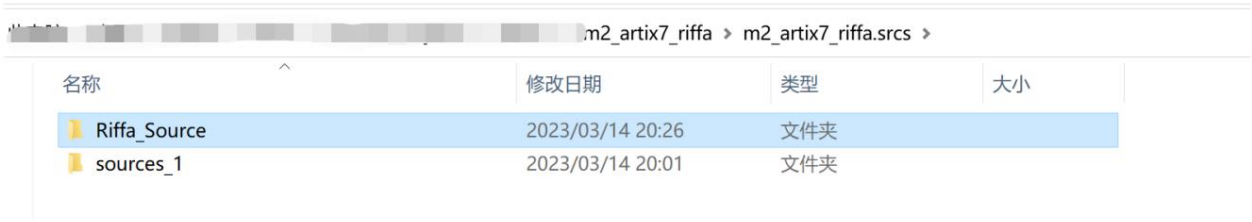

Figure 3-34 Copy the Riffa_Source folder to the project source file directory In

the project management window of the Vivado software, right-click the project Design Source option and select Add Source File, as shown in Figure 3-35.

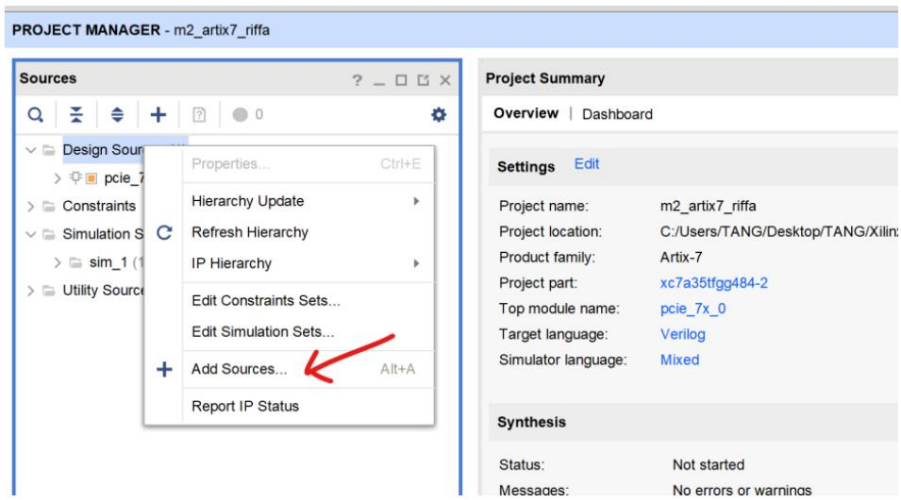

Figure 3-35 Add source file

Click Next.

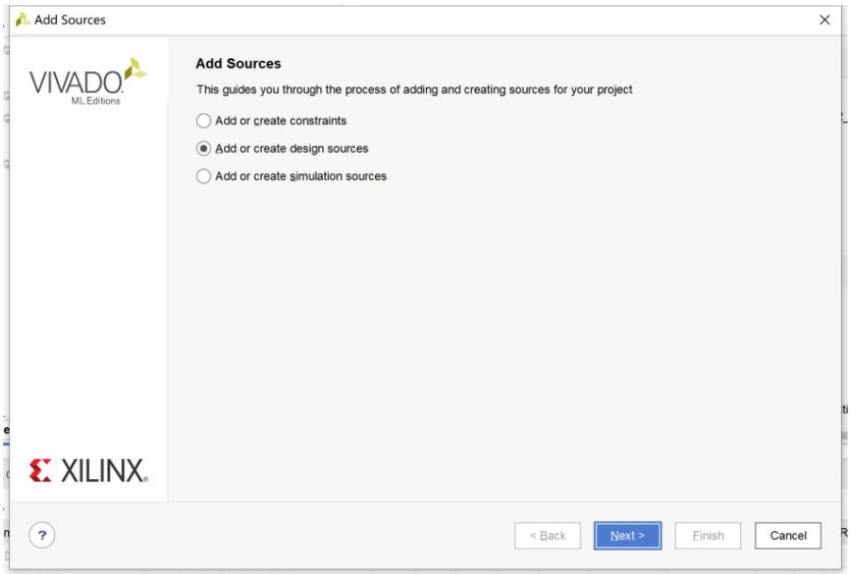

Figure 3-36 Add file

Select Add Path in the window, as shown in Figure 3-37.

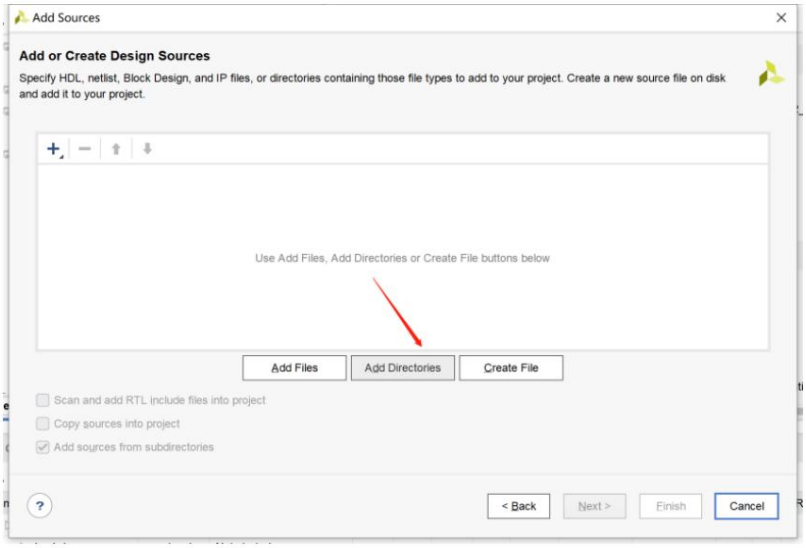

Figure 3-37 Select Add Path

Select the source_1 folder under the Riffa_Source folder and click Select, then click Finish, as shown in Figure 3-38.

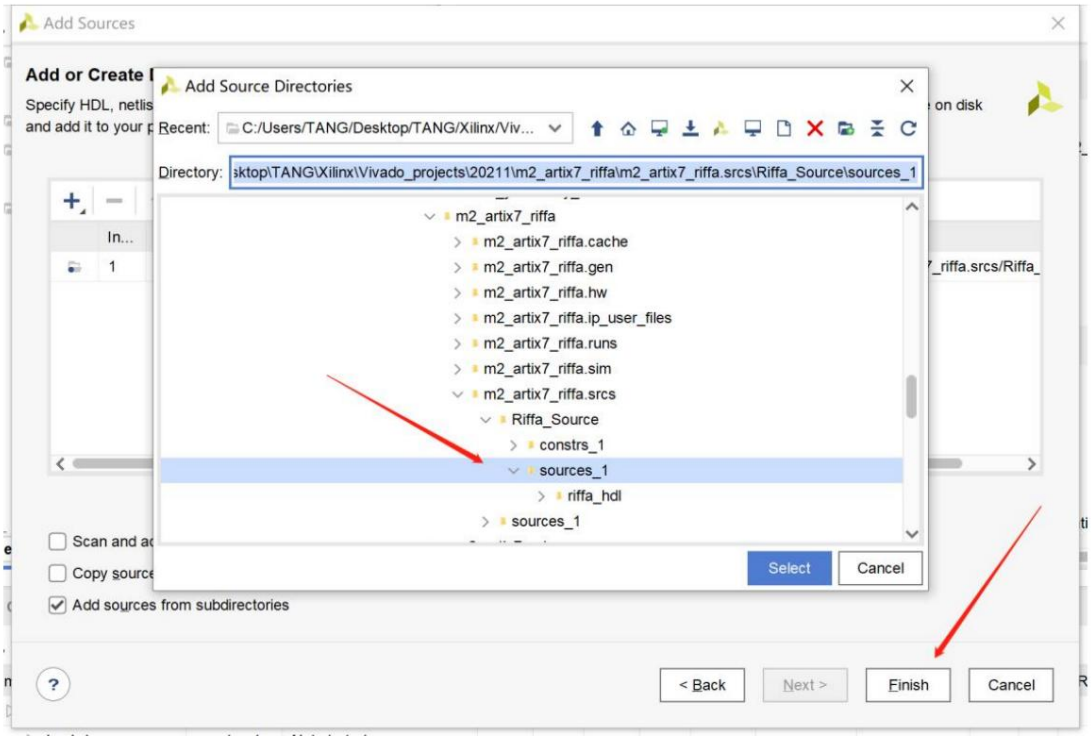

Figure 3-38 Add source_1 in the provided Riffa_Source folder

After adding the path, all files and structures will automatically appear in the project management directory, as shown in Figure 3-39.

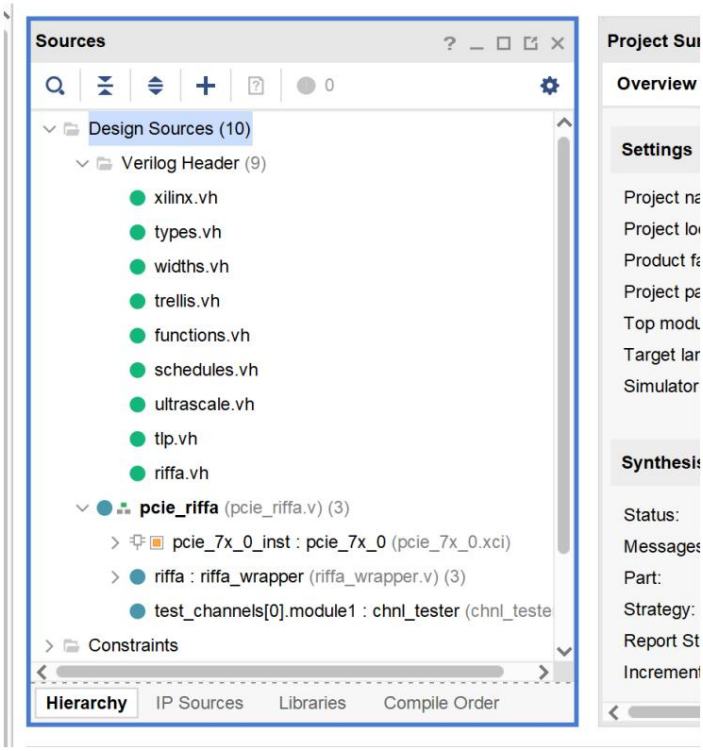

Figure 3-39 Riffa file structure

Right-click functions.vh and set it as Global include in the menu, as shown in Figure 3-40.

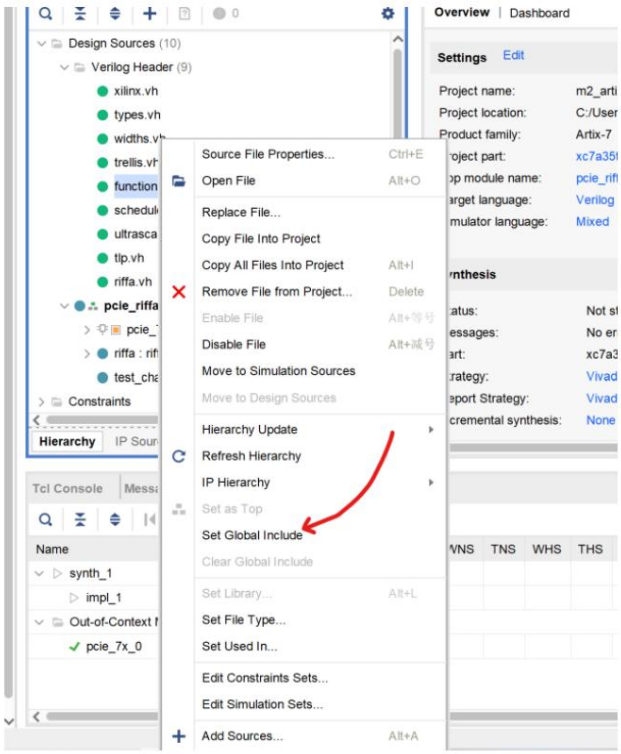

Figure 3-40 Setting functions.vh as Global include

Right click on the Constraints option, select Add Constraint File, and add constrs_1 in the Riffa_Source folder. pcie_riffa.xdc constraint file in the folder, and click Finish, as shown in Figure 3-41.

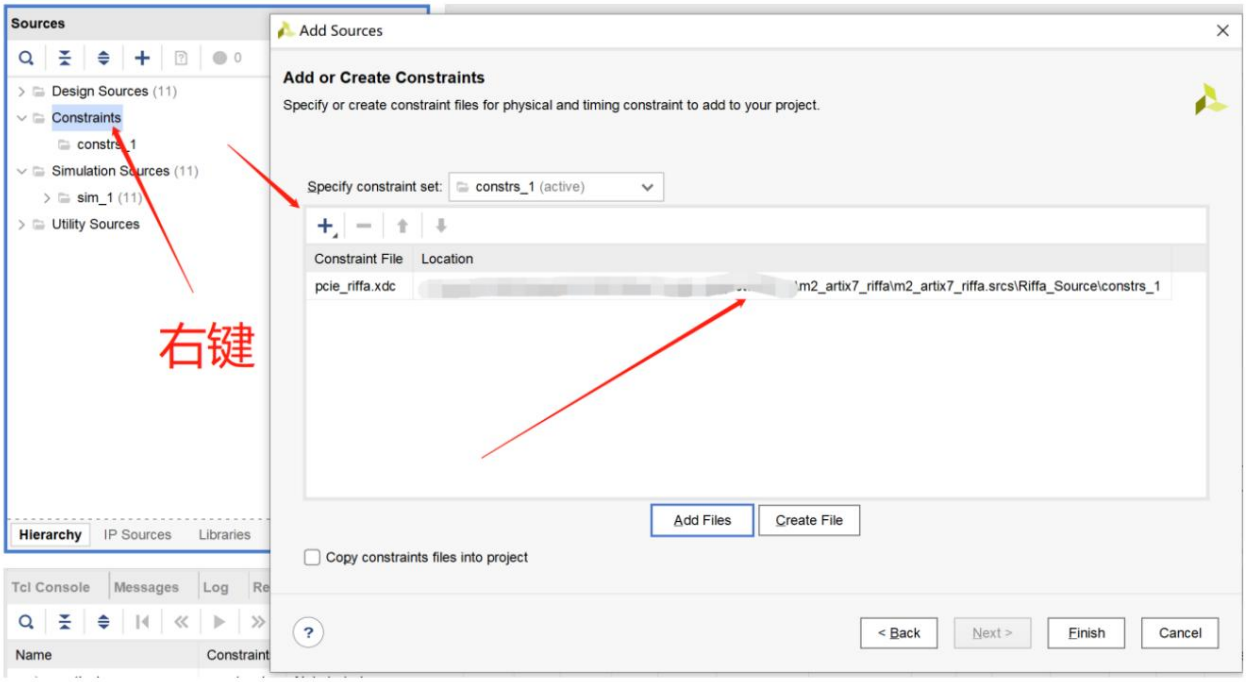

Figure 3-41 Adding Riffa constraint file After

adding, the project creation is completed. Click Generate Bitstream to generate the bitstream file.

It can be seen that the resource usage of Riffa is much less than that of XDMA, as shown in Figure 3-42.

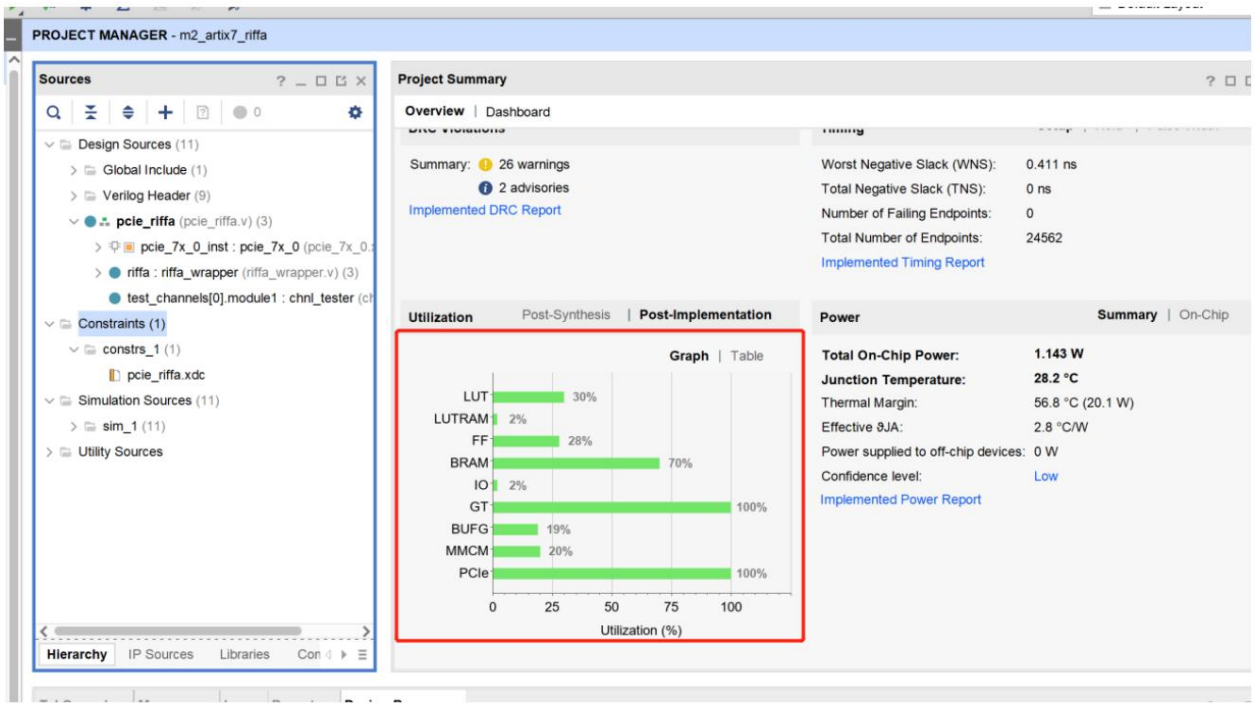

Figure 3-42 Riffa comprehensive results

3.8 Computer Test

3.8.1 Flash Burning

Burn the **Bin** file generated by the **Riffa** project into **the Flash .** The specific operation is to generate the **Bin** file according to Section **1-3 .**

The process of burning **the** data into **Flash** is not described here .

3.8.2 Installing the Accelerator Card

Install the Flash-burned M2 accelerator card to the M2 port of the computer. Note that the M2 port NVME protocol

After the installation is complete, the computer is turned on, the light on the accelerator card is on, and the green LED is on, as shown in Figure 3-43.

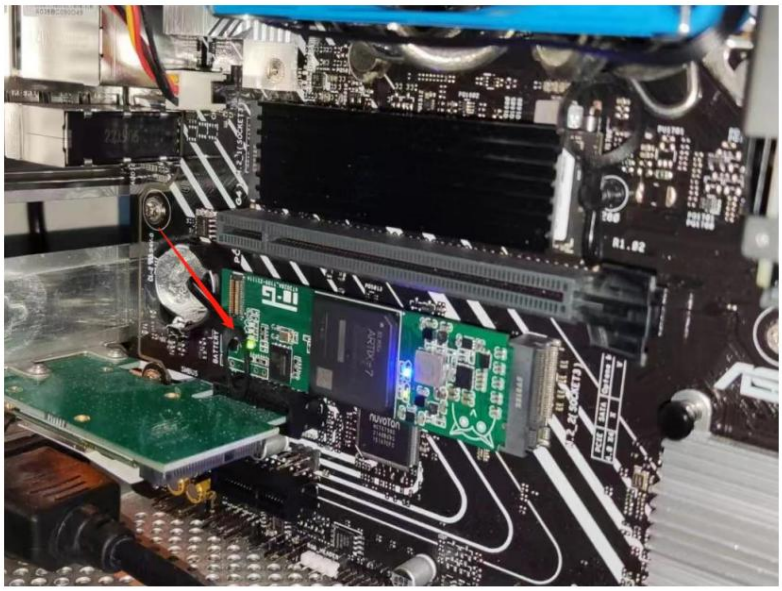

Figure 3-43 Installing the accelerator card and starting the computer

In the device manager of Windows system, the Riffa Device can also be identified, as shown in Figure 3-44.

Figure 3-44 Riffa device is identified in the device manager

3.8.3 Qt Software Engineering Compilation and Runtime Test

Open Qt Creator, select Open Project, and open the provided test_speed project file, as shown in Figure 3-45.

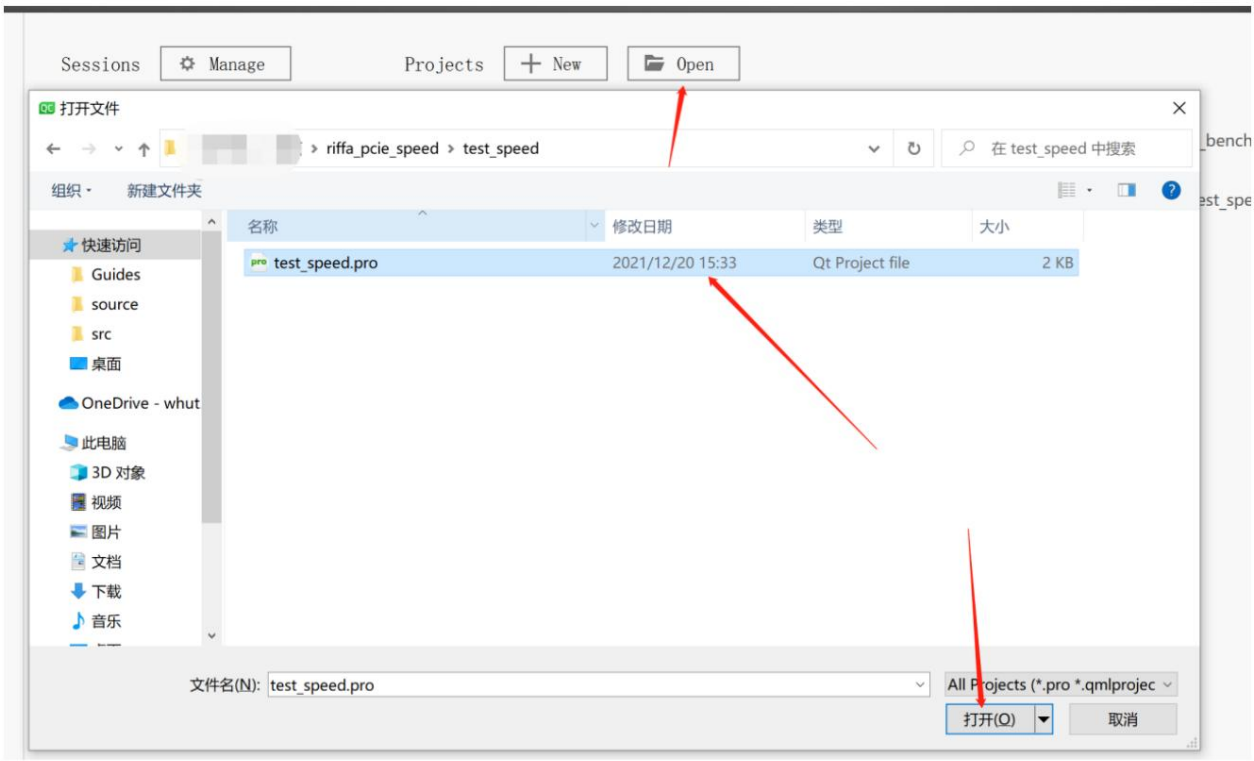

Figure 3-45 Open the Qt test_speed project.

Open the speed test project, where pcie_func is the main file. You can read the code yourself, as shown in Figure 3-46.

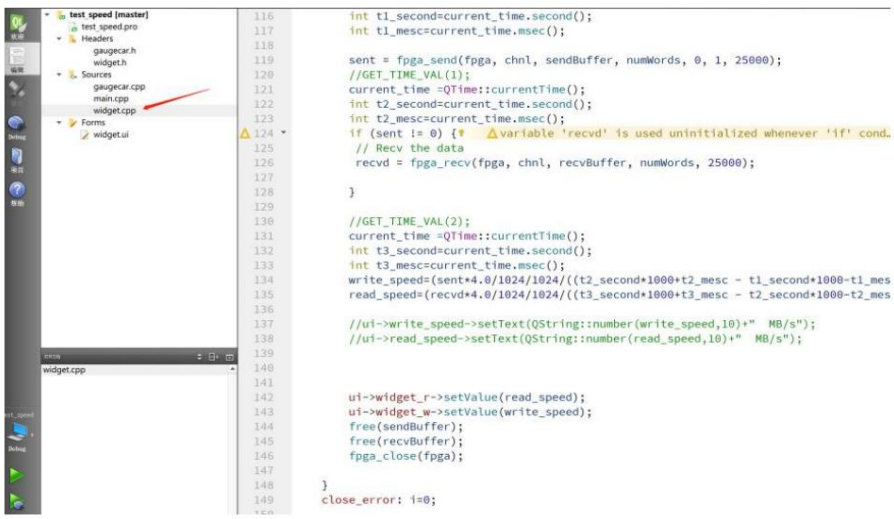

Figure 3-46 PCIE speed test project Click

the hammer icon in the lower left corner to compile the project. After the compilation is completed, some warnings will pop up, which can be ignored, as shown in Figure 3-47.

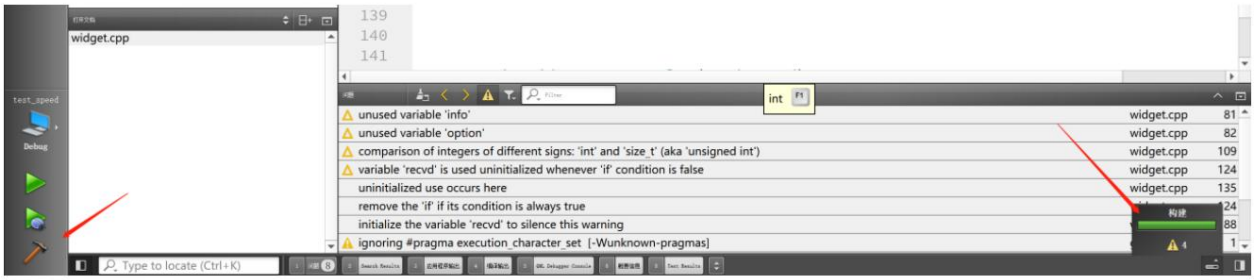

Figure 3-47 Compile speed test project

Click the run button in the lower left corner to start Riffa's loopback speed test, which will be displayed in the form of a speedometer GUI.

Click the test mode in the left drop-down menu of the GUI interface, and then click Start on the right to start the test.

The loopback test is performed and the speed is displayed, as shown in Figure 3-48.

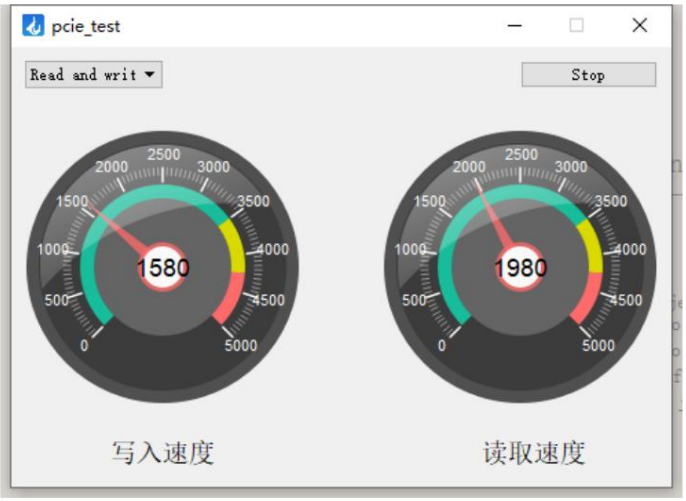

Figure 3-48 Riffa loopback test results

This concludes the Riffa-based loopback speed test.

The theoretical bandwidth of PCIE2.0*4 is (5GT/s*4)*(8/10)*(1/8)=2000MB/s, where 8/10 is 8b/10b encoding.

1/8 is the bits to bytes conversion.

It can be seen that the Riffa speed test results are 1580MB/s for writing and 1980MB/s for reading, which are very close to the theoretical bandwidth.

3.9 Postscript

This Riffa tutorial is modified from Wildfire's PCIE training tutorial. Wildfire's PCIE tutorial provides Riffa

For more examples, please refer to the Wildfire PCIE tutorial in the appendix. In actual use, Riffa's efficiency and

The speed is indeed much better than XDMA, and because it is open source and the driver is signed, it can have more operations.

Work space.

Chapter 4 Optical Communication Ibert Test

4.1 Hardware Preparation

Prepare the optical communication expansion baseboard, install the optical module into the slot, and make sure the clock chip output frequency configuration is turned on.

Select the corresponding position of 125MHz clock output, that is, 11010, as shown in Figure 4-1.

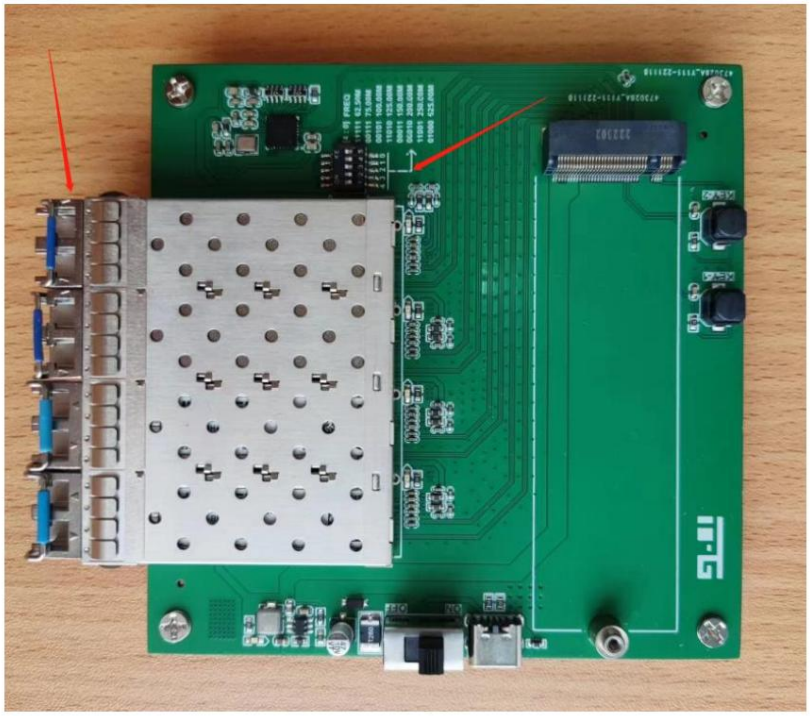

Figure 4-1 Optical module installation and clock chip configuration

Install the M2 FPGA core board into the M2 installation slot on the base plate and secure it with screws, as shown in Figure 4-2.

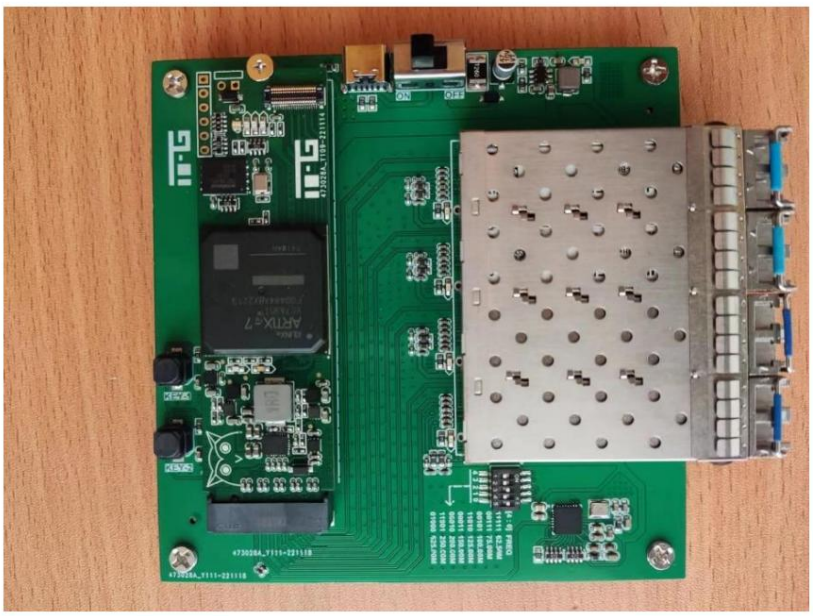

Figure 4-2 FPGA core board installation

4.2 Create a Vivado test project

Open the Vivado software, select Create Project and click Next, as shown in Figure 4-3.

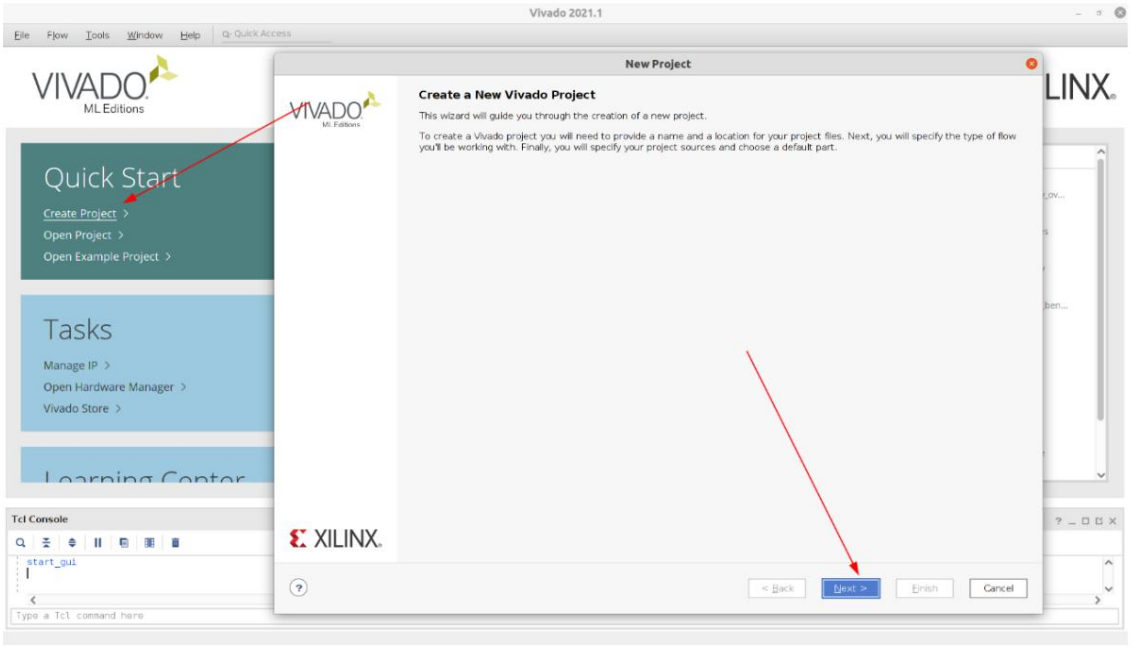

Figure 4-3 Creating a Vivado project

Enter the project name and project storage path, taking m2_artix7_ibert as an example, as shown in Figure 4-4.

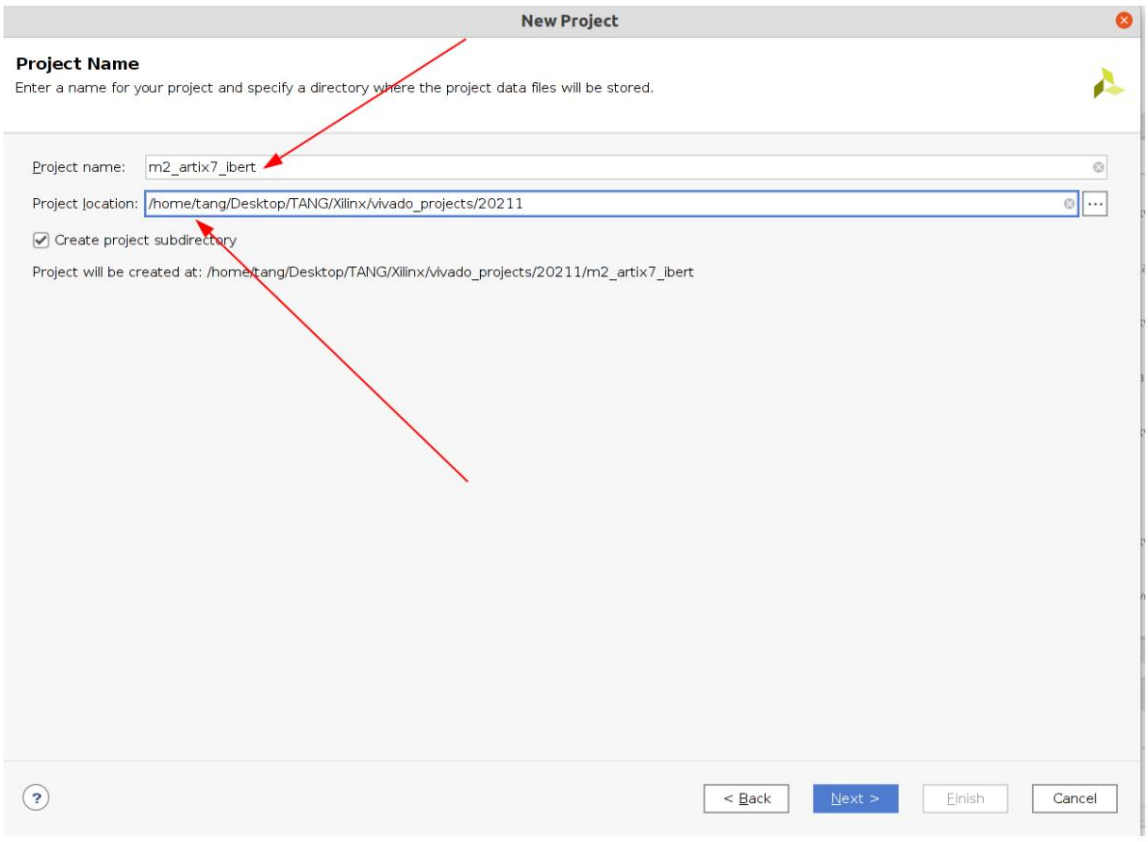

Figure 4-4 Enter the project name and path

Select RTL Project and check "No source file is currently specified", as shown in Figure 4-5.

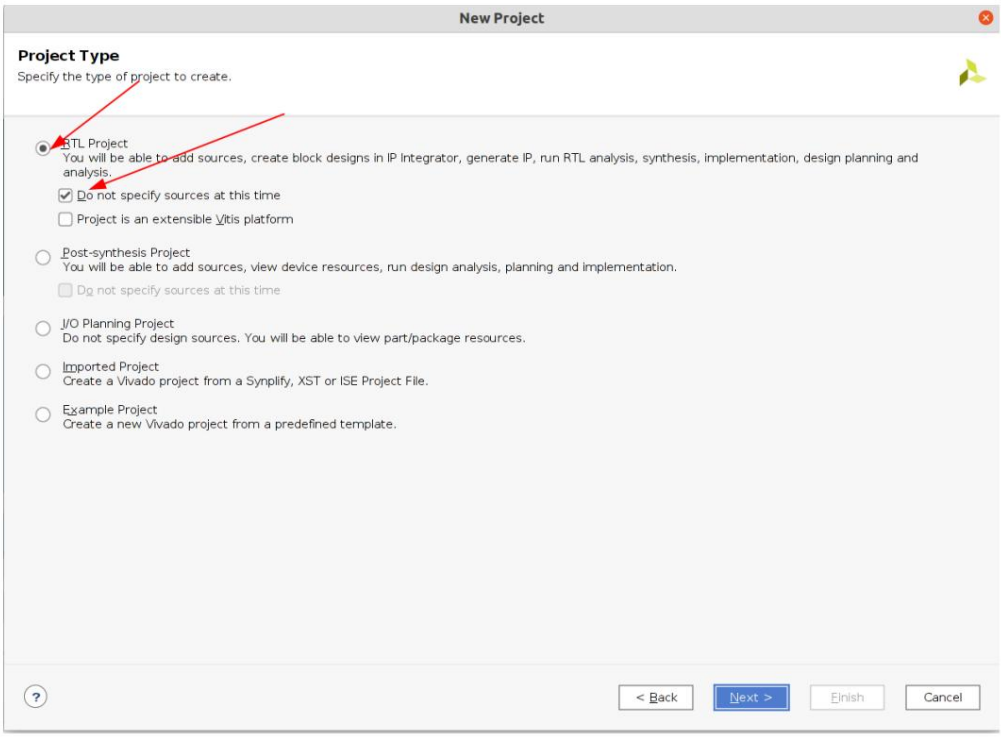

Figure 4-5 Select RTL Project

Select the chip according to the core board chip model and click Next. Take XC7A35T-2FGG484I as an example, as shown in Figure 4-6.

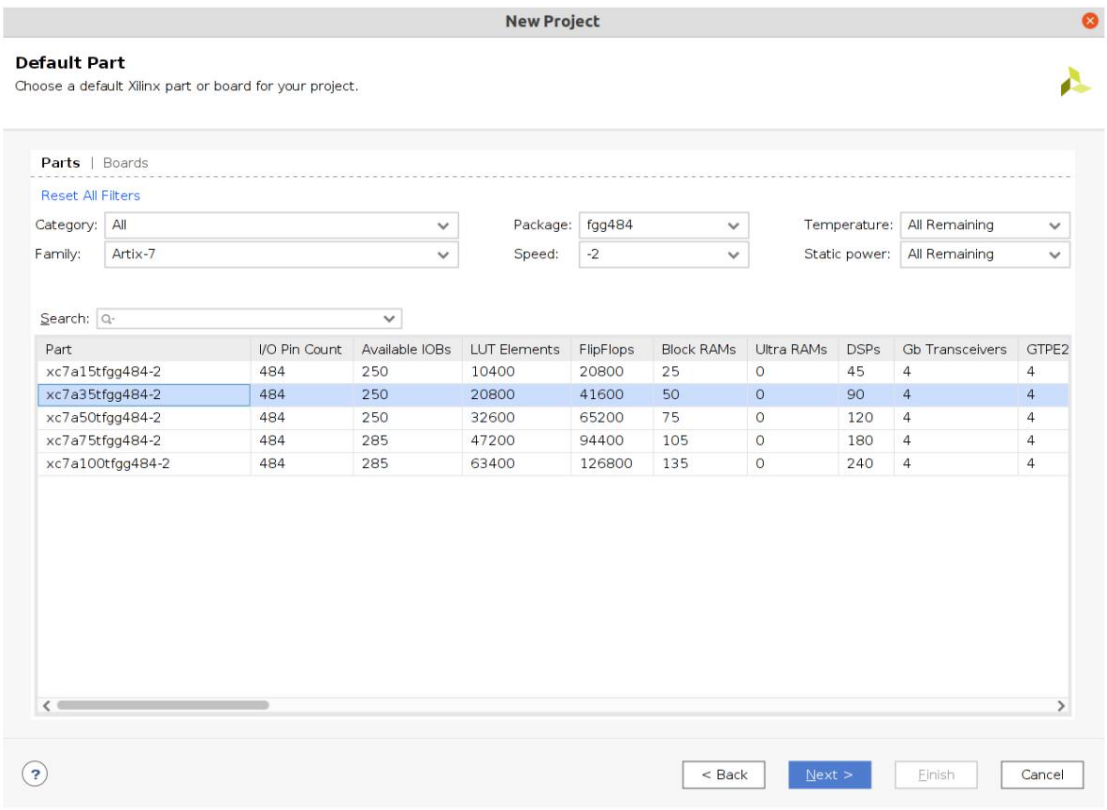

Figure 4-6 Select chip model

Click Finish in the new window to complete the project creation, enter the project main interface, and select IP catalog as shown in Figure 4-7.

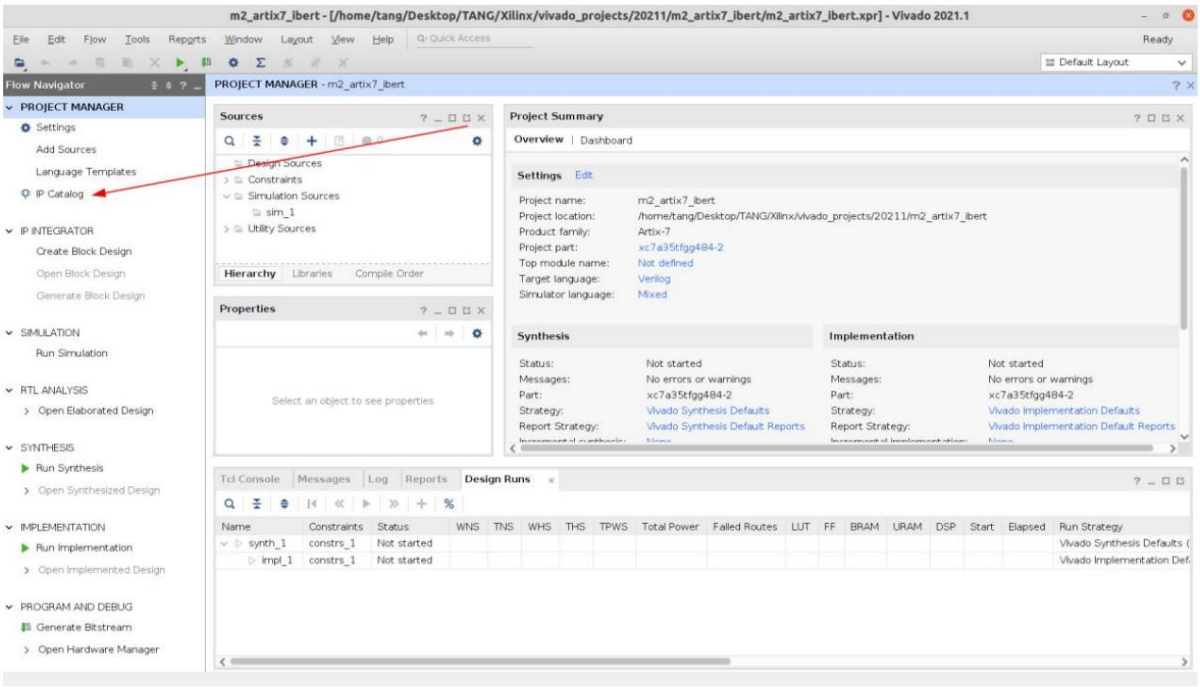

Figure 4-7 Select IP Catalog on the main project interface

In the Search box of IP Catalog, enter ibert, find IBERT 7 Series GTP and double-click it to enter the configuration interface.

As shown in Figure 4-8.

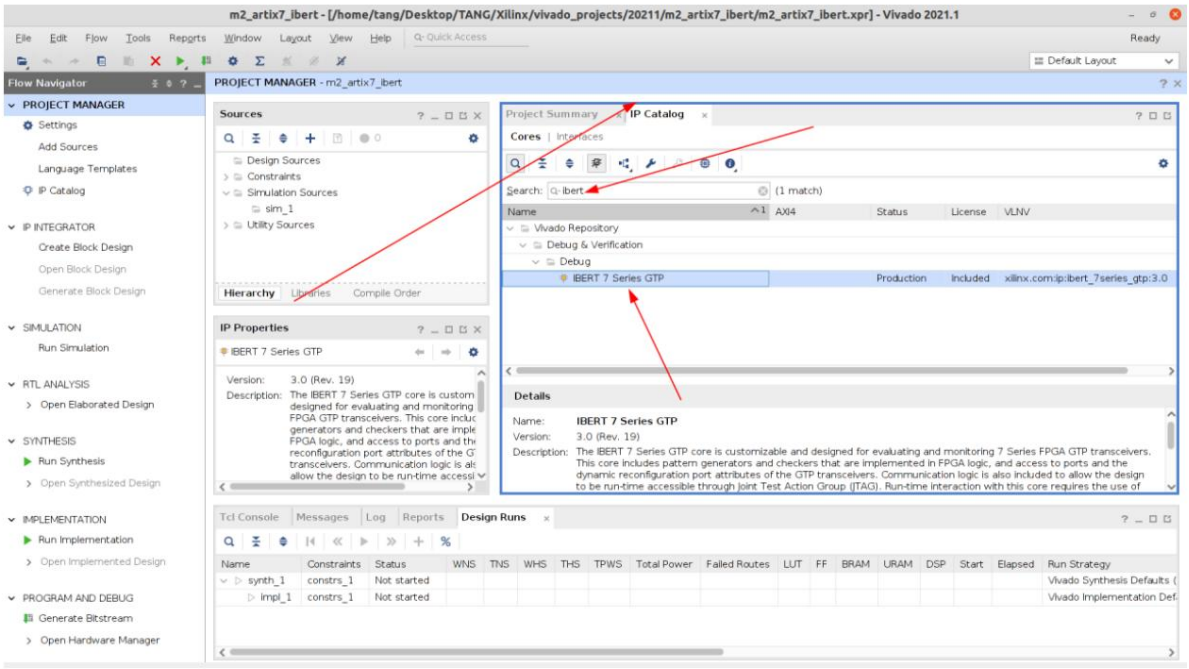

Figure 4-8 Find and double-click to enter ibert IP configuration

In the IP configuration interface, select Protocol Definition, enter LineRate as 6.25Gbps, and select Refclk

125MHz, as shown in Figure 4-9.

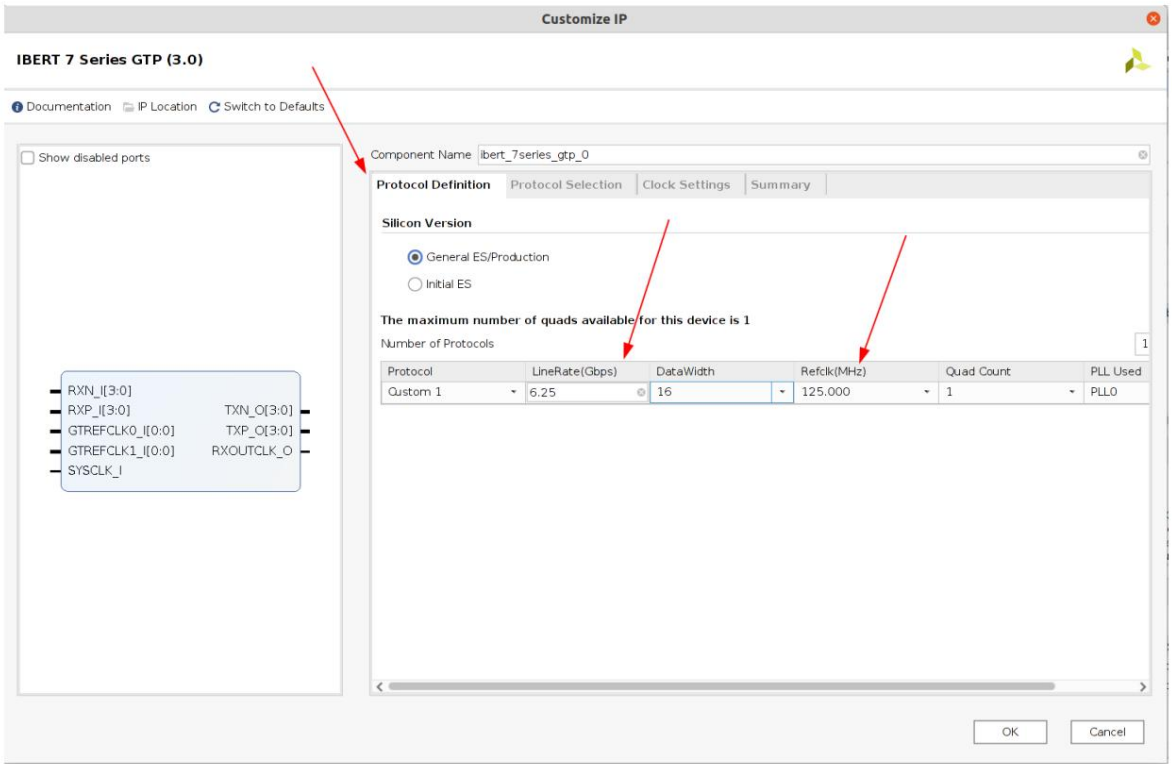

Figure 4-9 IP protocol specification

Select Protocol Selection, and select the 6.25Gbps protocol according to Figure 4-10 . Reference clock input selection MGTREFCLK1_216, Source selects Channel0.

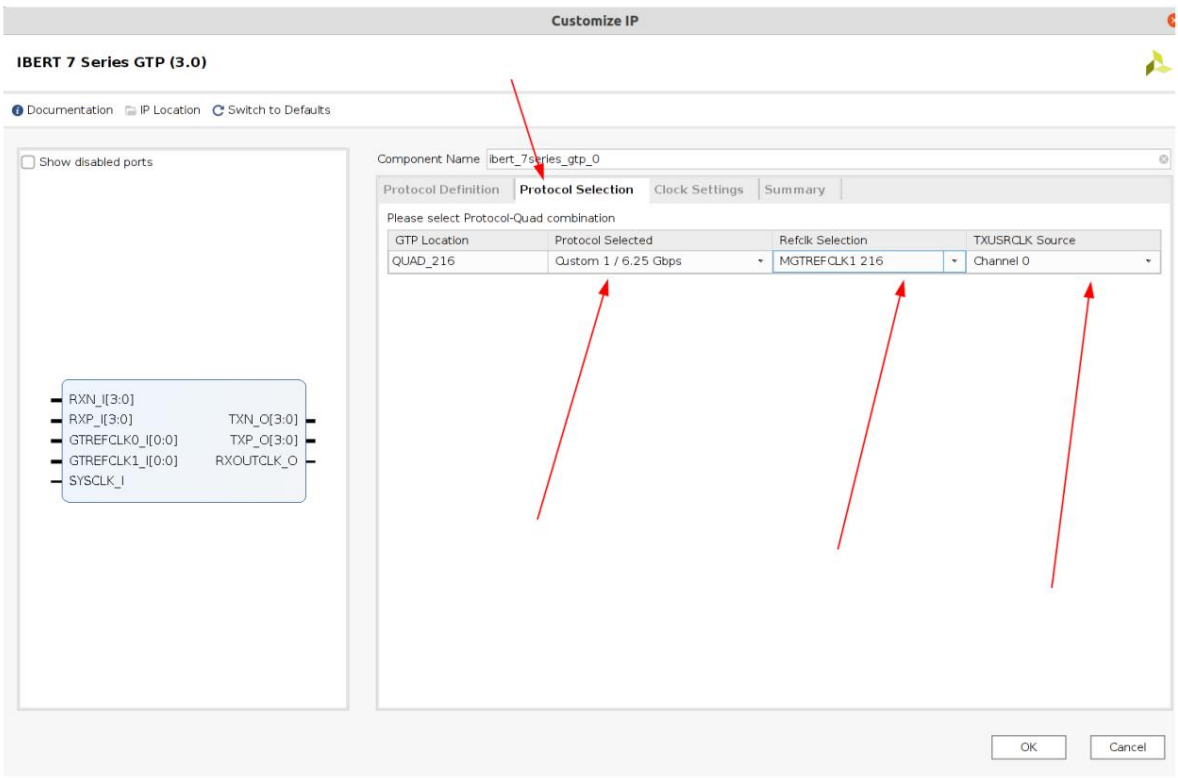

Figure 4-10 IP protocol selection

Select Clock Setting, select QUAD216_1 as Source, as shown in Figure 4-11.

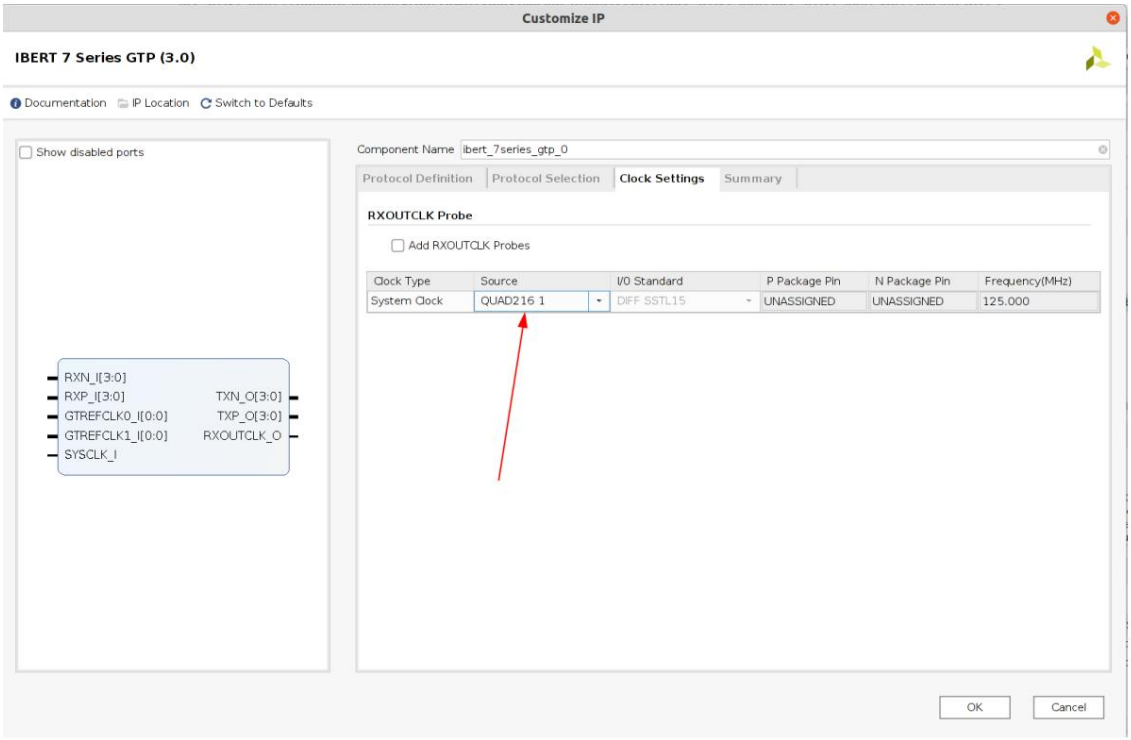

Figure 4-11 IP Clock Settings

After confirming that the configuration is correct in the Summary option, click OK to complete the IP setting, as shown in Figure 4-12.

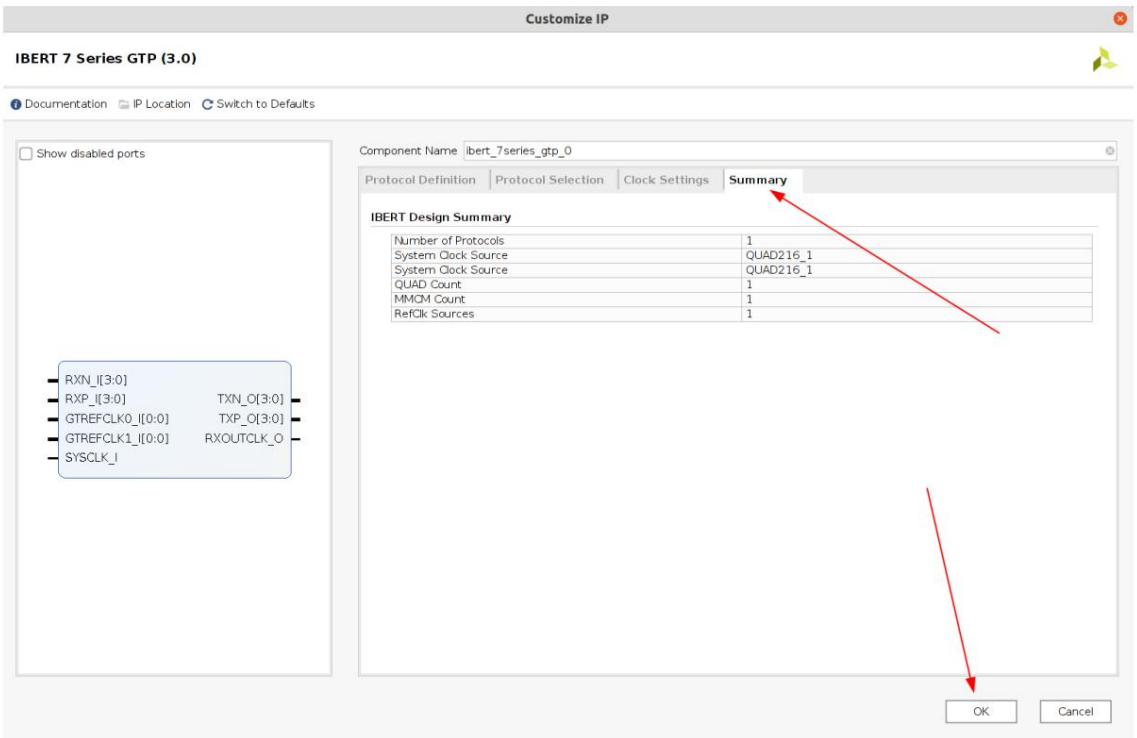

Figure 4-12 Complete IP settings.

Click Skip in the pop-up interface to temporarily not generate output files. See Figure 4-13.

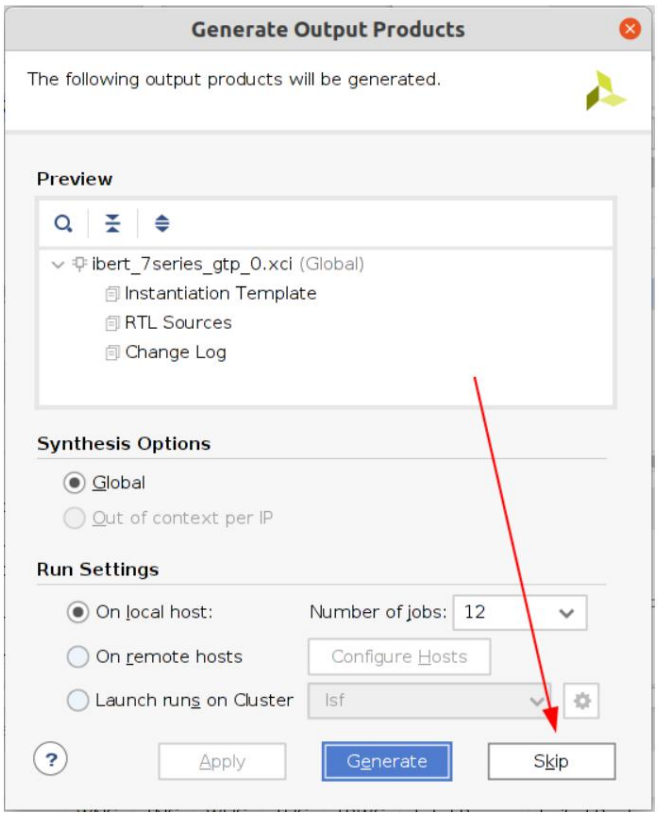

Figure 4-13 Skip output file generation

Under Design Source , right-click ibert IP and select Open IP Example Design, as shown in Figure 4-14.

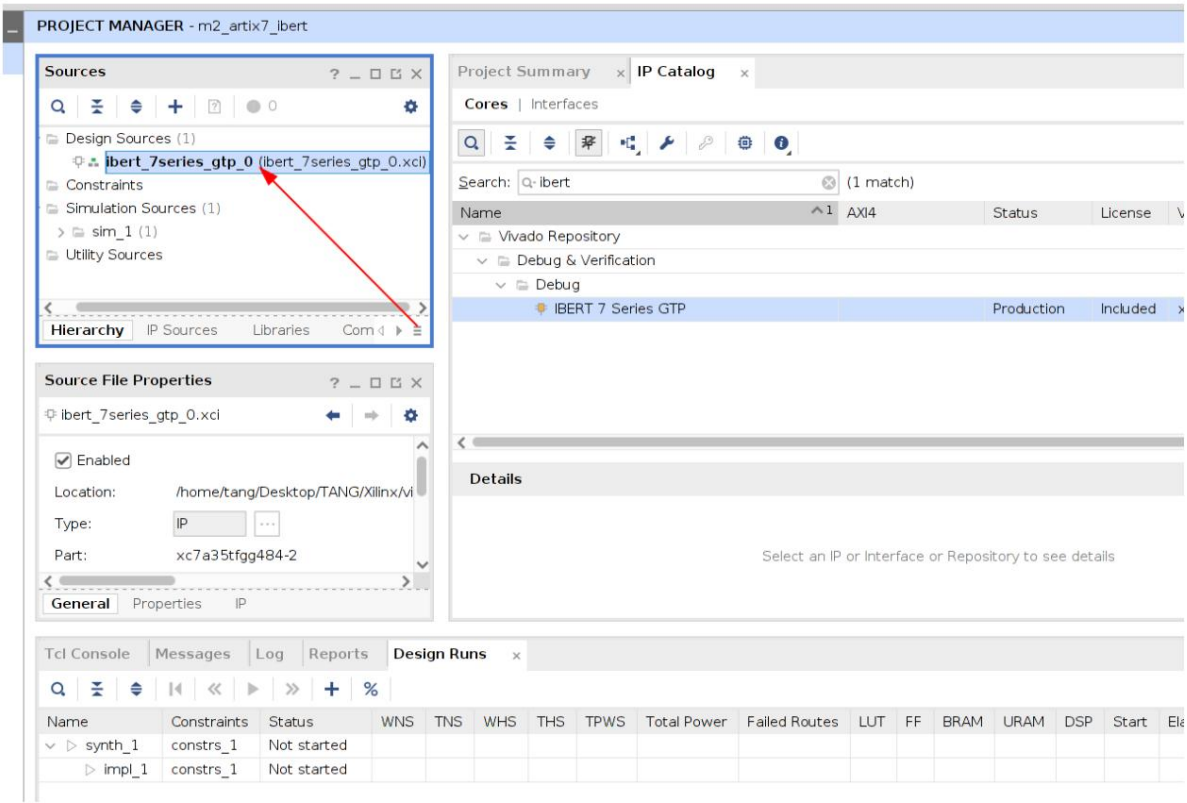

Figure 4-14 Selecting to output Example Design

Select the sample project output directory, as shown in Figure 4-15.

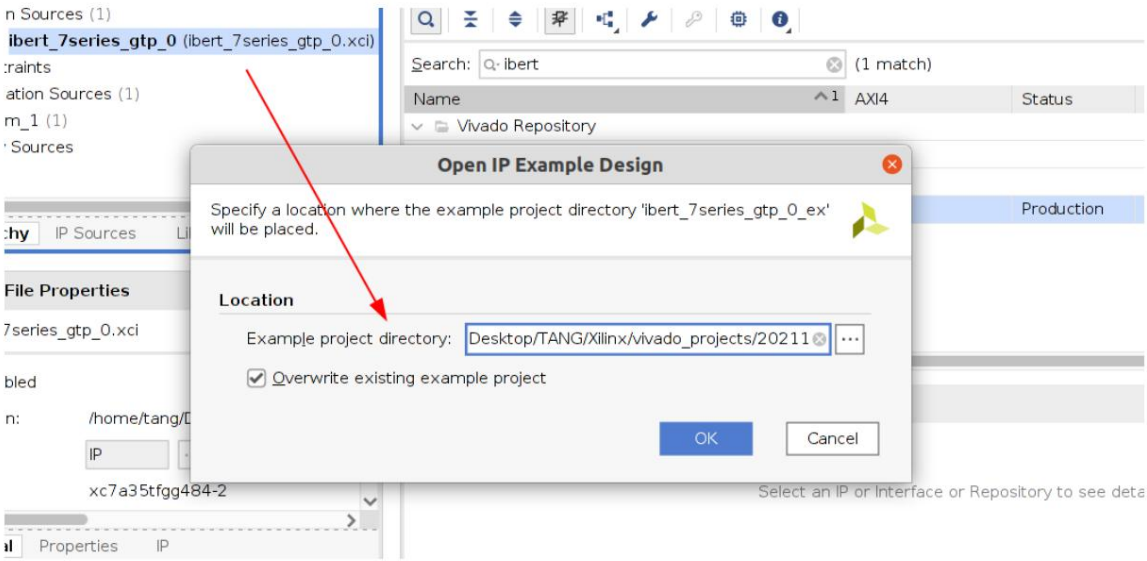

Figure 4-15 Select Example Design to generate the directory. After

clicking OK, Vivado will automatically generate a new Example project and open it automatically, as shown in Figure 4-16.

As it has been configured before, just click Generate Bitstream to generate the bitstream directly.

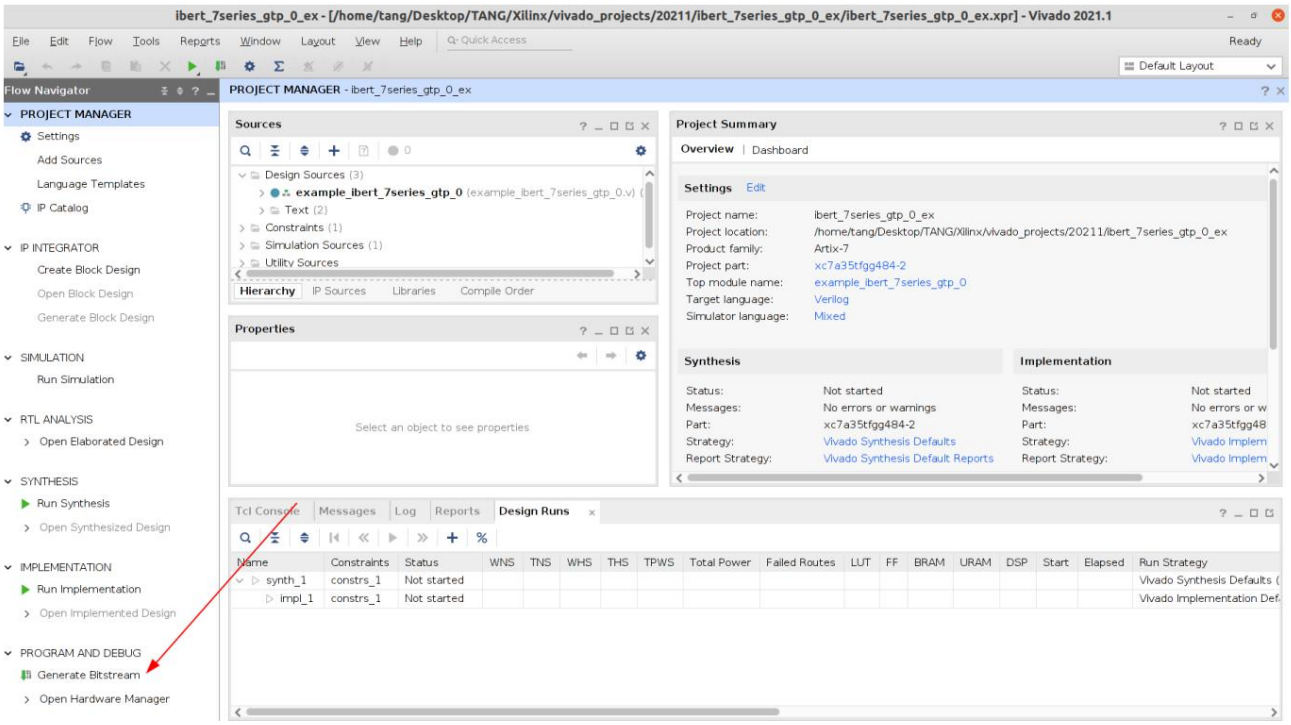

Figure 4-16 ibert sample project

4.3 Ibert test

After the Bitstream is generated, connect the JTAG and the backplane power supply, connect the optical module with optical fiber, and turn on the power, as shown in Figure 4-17.

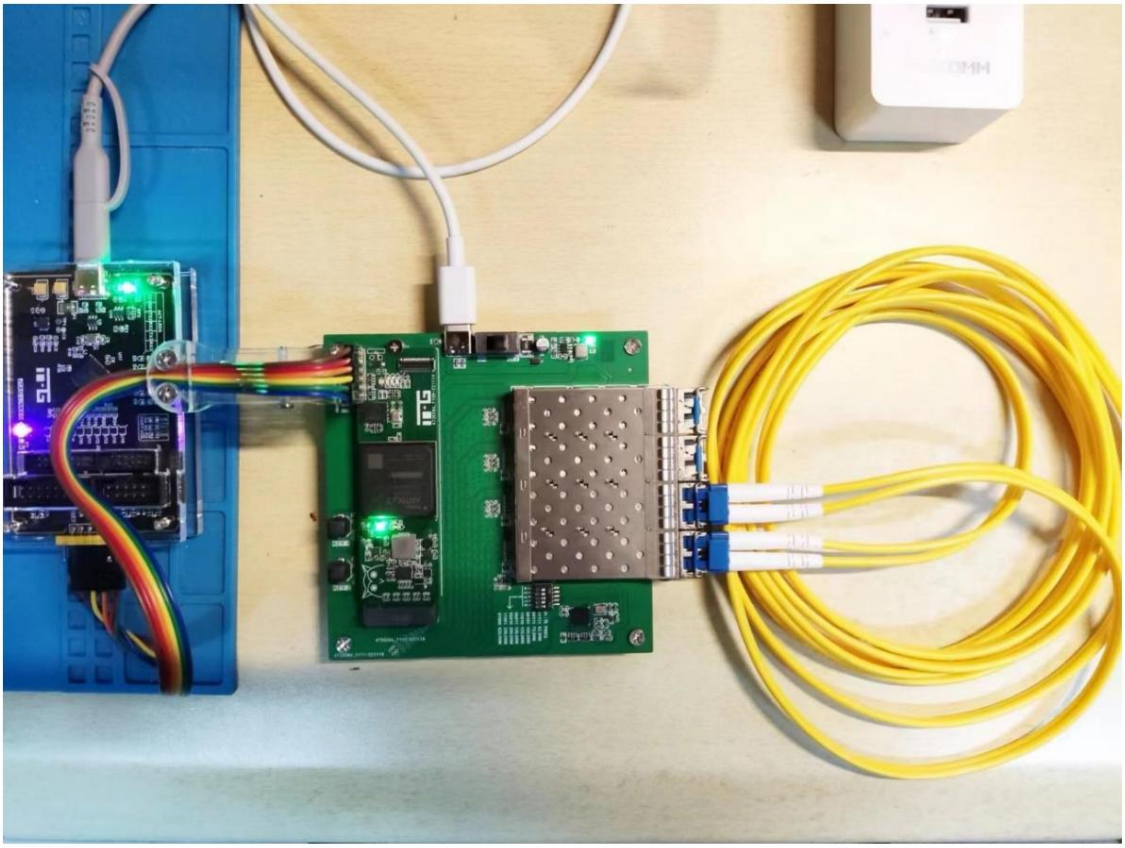

Figure 4-17 Connect the optical fiber and the downloader, and power on the development board

Select Open Hardware Manager in Vivado, click Open Target, select Auto Connect, connect to the development board, and identify the chip, as shown in Figure 4-18.

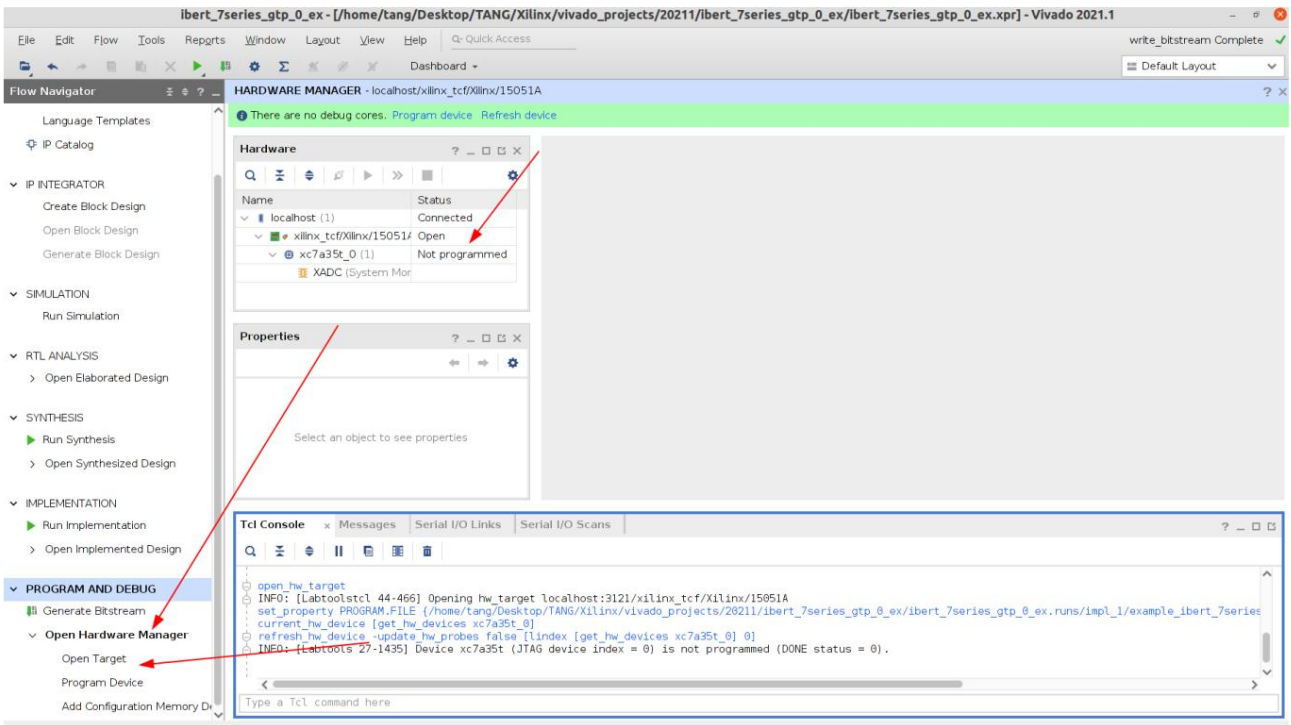

Figure 4-18 JTAG connection and chip identification

Burn the newly generated Bitstream, and ibert will start running. The PLL lock will appear automatically as shown in Figure 4-19.

In the Serial IO Link window, click Auto-detect links, and the two channels for loopback will be automatically scanned.

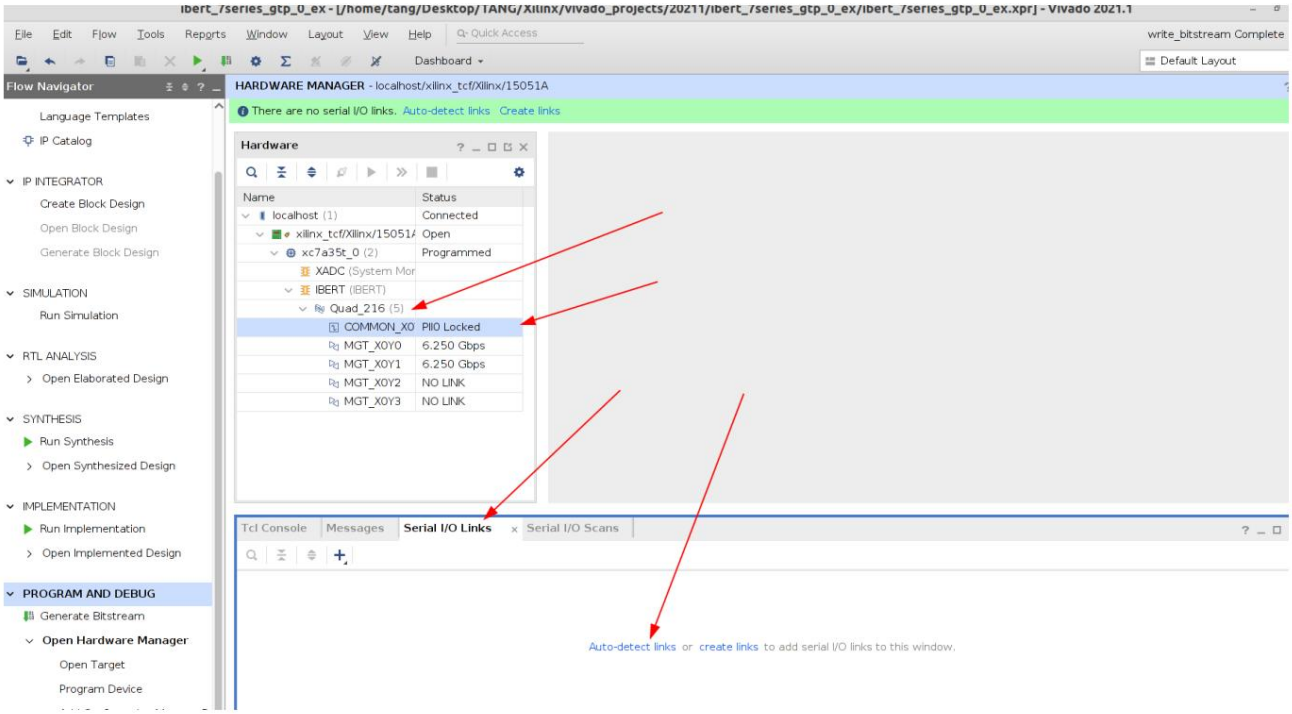

Figure 4-19 ibert runs.

Right-click on any link , select Create Scan, use the default parameters and click OK, as shown in Figure 4-20.

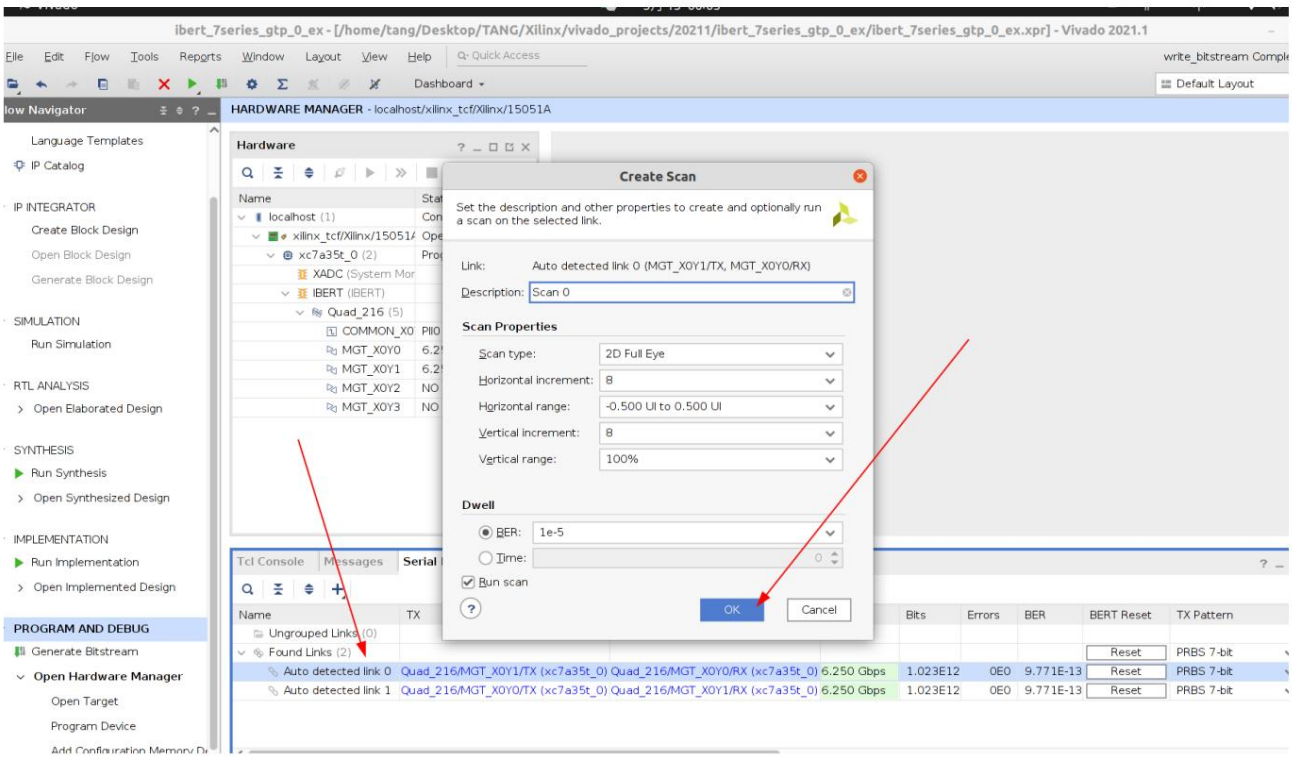

Figure 4-20 Create Scan

After completing a scan, the eye diagram information will be automatically generated, as shown in Figure 4-21.

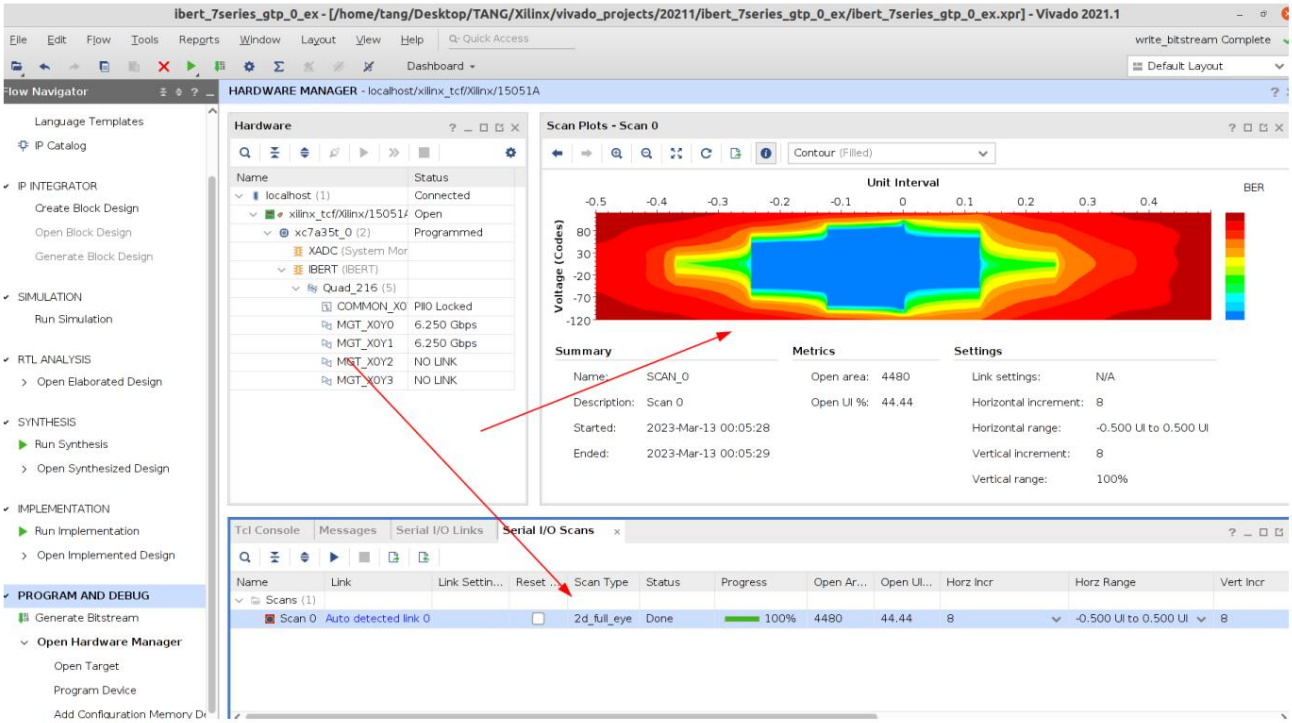

Figure 4-21 Ibert eye diagram information

Readers can refer to the Xilinx data sheet PG132 Integrated Bit Error Ratio Tester 7 in the reference material.

Series GTX Transceivers Explore more about these tests.