MYS-ZU5EV_FPGA Development Manual

File Status : [ ] Draft [√] Release	**File :**	MYS-ZU5EV_FPGA Development Manual
	Revision :	V2.0.3
	Author :	Rill yang
	Date :	2021-07-13
	Update :	2021-07-13

Revision history：

Date	Revision	Description
2021.05.31	v1.0.0	v1.0
2021.06.02	V1.0.1	Revise the directory structure, revise the chapter content arrangement
2021.06.04	V1.0.2	Revise the directory structure, revise the chapter content arrangement
2021.06.07	V1.0.3	Revise the directory structure, revise the chapter content arrangement
2021.06.08	V1.0.4	Revise the directory structure, revise the chapter content arrangement
2021.06.25	V2.0.0	Officially release revision
2021.07.03	V2.0.1	1. Modify the overall format and font format of the text. 2. Revise the controversial picture in Section 6.1.1 in the previous version, and revise the picture references and constraint statements in Section 2.3.4 regarding input constraints and output constraints. 3. The chapter table in section 1.4 is revised. 4. The software part in section 6.1.3 has been revised. 5. Amended the content cited in Section 7.3.2 and merged with Section 7.3.1. 6. Reference materials and contact information have been added at the end of the document.
2021.07.07	V2.0.2	1. Modified the name of the picture. 2. Explain the corresponding compressed file in the CD file for each sample project. 3. Simplified and revised the explanation about usage constraints in section 2.3.4. 4. Modified the basic knowledge of DMA in section 5.2.1 and

		quoted the official PG021 document content.
2021.07.09	V2.0.3	1. Added part of the VCU content in Section 6.2.

CONTENT

Chapter 1Abstract

1.1 About the Document

This document mainly introduces how to implement your own project development on the MYS-ZU5EV hardware platform. Through studying this document, I hope that customers can achieve several goals:

1. Understand the hardware platform of MYS-ZU5EV.
2. Be able to achieve your own target design on this hardware platform.
3. Understand the reference materials about the hardware platform of MYS-ZU5EV, customers can find them quickly, especially for some official reference documents and reference designs of xilinx.

Introduce the collection of materials and information that customers may use when using the MYS-ZU5EV hardware platform. When encountering key points, they can seek the direction of reference materials, and they can seek cooperation and development from MYIR.

Here is the origin of the heterogeneous platform ZYNQ/MPSOC.

This is a hardware platform developed on the basis of the SOC. When the required peripheral interfaces and the data to be processed are integrated by the ARM processor, it is the SOC chip, but there are some special tasks in the SOC that cannot be completed or reached. To the required standard, it is necessary to add additional resources to complete the dedicated function. At this time, there are two solutions, one is to plug a dedicated device outside the PCB, and the other is to add a dedicated area in the SOC chip to complete the dedicated function. Obviously now the ZYNQ platform is the second solution. The second solution is the development trend, the current trend AI function and the newly added various functional modules The primary solution is to embed the corresponding functional modules into the SOC chip to form a more complex SOC chip.

For detailed information about the heterogeneous platform of ZYNQZYNQ/MPSOC, please refer to the official xilinx document:

ds891-zynq-ultrascale-plus-overview.pdf

ug1137-zynq-ultrascale-mpsoc-swdev.pdf

ug1085-zynq-ultrascale-trm.pdf

1.2 About the Arrangement of the Later Chapters of this Document

The first chapter is an introduction, briefly introducing the purpose of wri ting this document and the chapter structure of the document. The second c hapter introduces some preparations for using this hardware platform or simil ar hardware platforms, including some preparations for the use of the hardw are platform. For the introduction of using xilinx's EDA tool vivado, from xilin x official website registration to download the required version Vivado softwa re, installation screenshots are introduced.

Secondly, it introduces the download, installation and use of FPGA simula tion tool modelsim. The corresponding vivado version can call the specific m odelsim version. Here, refer to the vivado installation document to find the c orresponding modelsim version. When using these EDA tools for the first tim e, license registration is required, and registration methods are introduced. In addition, some paid IPs from Xilinx have a trial period of 3 months. You ne ed to download a temporary license on the official website to register before you can use it. Otherwise, you need to design and write your own code. Fin ally, some verilog syntax, corresponding reference materials, the use of the d ocument finder DocNav, the syntax introduction of XDC constraint files, a bri ef introduction of TCL syntax and so on are introduced.

Chapter 3, Section 3.1 explains the detailed configuration of the detailed hardware platform. The general content includes a detailed configuration intr oduction of MPSOC according to the resources used by the hardware platfor m. For example, the hardware platform is connected to the DP interface on t he PS side. When establishing the vivado project, you need to check the DP interface option of the peripheral interface of the MPSOC IP core. , And selec t the corresponding link pin according to the schematic connection method. I n short, the interfaces and peripherals on the hardware platform will be intro duced in detail in the PS configuration in Chapter 3. The last step is to speci fically generate the hardware platform file xsa file, and the last step is to use vitis to build an app project to output hello world. Section 3.2 On the basis of Section 3.1, assuming that the PL side logic is added, we will introduce t he data interaction method between the PL and the PS side. It focuses on th

e AXI4 bus. This content is closely related to the content of the following ch apters.

The fourth chapter is divided into several sub-sections, and respectively i ntroduces a use case of using this hardware platform to achieve a goal. Thes e are all based on the AXI4-Lite bus PL end external expansion peripheral int erface module. The two general interfaces introduced are AXI UART interface and AXI-IIC interface.

The fifth chapter is an introduction to the use of high-performance AXI4 bus interface. 5.1 Introduce the use of PL-side BRAM to interact with the PS-side data of small modules. 5.2 introduces the demo using DMA.

Chapter 6 , Introduces the use of AXI-Stream in the transmission of video streams, focusing on the MIPI interface using IMX334 as a sensor, configurat ion, and the process of using AXI-stream to transmit video streams. The adde d things are customizable. How to add RTL module to AXI-Stream data strea m. Custom IP design, white balance module addition. The seventh chapter int roduces how to use some official reference materials of xilinx.

Chanpter 7. 7.1 Introduce how to quickly view the official user manual do cuments to quickly find the information we need. 7.2 Introduce how to quickl y find xilinx product manuals, and quickly find information about the function s, features, interfaces, timing and other aspects of the corresponding IP. 7.3 i ntroduces how to use the official reference design of xilinx. 7.4 How to quick ly seek help for problems encountered by customers, you can raise questions in the Chinese and English communities of xilinx's official website. There is d edicated xilinx technical support for maintenance. Working hours are online, where you can get quick feedback and solutions. Program.

Chapter 8, Conclusion.

1.3 MPSOC Series Chip Introduction

1 , Low-end CG series,as show ,

Zynq® UltraScale+™ MPSoCs: CG Devices

	Device Name[1]	ZU1CG	ZU2CG	ZU3CG	ZU4CG	ZU5CG	ZU6CG	ZU7CG	ZU9CG
Processing System (PS)	Application Processor Unit	Processor Core	**Dual-core** Arm® Cortex®-A53 MPCore™ up to 1.3GHz						
		Memory w/ECC	L1 Cache 32KB I / D per core, L2 Cache 1MB, on-chip Memory 256KB						
	Real-Time Processor Unit	Processor Core	**Dual-core** Arm Cortex-R5F MPCore up to 533MHz						
		Memory w/ECC	L1 Cache 32KB I / D per core, Tightly Coupled Memory 128KB per core						
	External Memory	Dynamic Memory Interface	x16: DDR4 w/o ECC; x32/x64: DDR4, LPDDR4, DDR3, DDR3L, LPDDR3 w/ ECC						
		Static Memory Interfaces	NAND, 2x Quad-SPI						
	Connectivity	High-Speed Connectivity	PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet						
		General Connectivity	2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO						
	Integrated Block Functionality	Power Management	Full / Low / PL / Battery Power Domains						
		Security	RSA, AES, and SHA						
		AMS - System Monitor	10-bit, 1MSPS – Temperature and Voltage Monitor						
PS to PL Interface			12 x 32/64/128b AXI Ports						
Programmable Logic (PL)	Programmable Functionality	System Logic Cells (K)	81	103	154	192	256	469	504	600
		CLB Flip-Flops (K)	74	94	141	176	234	429	461	548
		CLB LUTs (K)	37	47	71	88	117	215	230	274
	Memory	Max. Distributed RAM (Mb)	1.0	1.2	1.8	2.6	3.5	6.9	6.2	8.8
		Total Block RAM (Mb)	3.8	5.3	7.6	4.5	5.1	25.1	11.0	32.1
		UltraRAM (Mb)	-	-	-	13.5	18.0	-	27.0	-
	Clocking	Clock Management Tiles (CMTs)	3	3	3	4	4	4	8	4
	Integrated IP	DSP Slices	216	240	360	728	1,248	1,973	1,728	2,520
		PCI Express® Gen 3x16	-	-	-	2	2	-	2	-
		150G Interlaken	-	-	-	-	-	-	-	-
		100G Ethernet MAC/PCS w/RS-FEC	-	-	-	-	-	-	-	-
		AMS - System Monitor	1	1	1	1	1	1	1	1
	Transceivers	GTH 16.3Gb/s Transceivers	-	-	-	16	16	24	24	24
		GTY 32.75Gb/s Transceivers	-	-	-	-	-	-	-	-
	Speed Grades	Extended[2]	-1 -2 -2L							
		Industrial	-1 -1L -2							

Figure 1-1CG series features

The low-end series processor is dual-core A53, the main frequency can re ach 1.3GHz, the actual frequency is lower, only about 1GHz, the high-speed i nterface on the ps side has USB3.1, STAT3.0, DP, GEthernet, PCIe. The ability is low, and the logic unit can choose different devices as the demand fails.

2 , The mid-range EG series, as shown :

Zynq® UltraScale+™ MPSoCs: EG Devices

	Device Name[1]	ZU1EG	ZU2EG	ZU3EG	ZU4EG	ZU5EG	ZU6EG	ZU7EG	ZU9EG	ZU11EG	ZU15EG	ZU17EG	ZU19EG
Processing System (PS)	Application Processor Unit	Processor Core	**Quad-core** Arm® Cortex®-A53 MPCore™ up to 1.5GHz										
		Memory w/ECC	L1 Cache 32KB I / D per core, L2 Cache 1MB, on-chip Memory 256KB										
	Real-Time Processor Unit	Processor Core	**Dual-core** Arm Cortex-R5F MPCore™ up to 600MHz										
		Memory w/ECC	L1 Cache 32KB I / D per core, Tightly Coupled Memory 128KB per core										
	Graphic & Video Acceleration	Graphics Processing Unit	Mali™-400 MP2 up to 667MHz										
		Memory	L2 Cache 64KB										
	External Memory	Dynamic Memory Interface	x16: DDR4 w/o ECC; x32/x64: DDR4, LPDDR4, DDR3, DDR3L, LPDDR3 w/ ECC										
		Static Memory Interfaces	NAND, 2x Quad-SPI										
	Connectivity	High-Speed Connectivity	PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet										
		General Connectivity	2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO										
	Integrated Block Functionality	Power Management	Full / Low / PL / Battery Power Domains										
		Security	RSA, AES, and SHA										
		AMS - System Monitor	10-bit, 1MSPS – Temperature and Voltage Monitor										
PS to PL Interface			12 x 32/64/128b AXI Ports										
Programmable Logic (PL)	Programmable Functionality	System Logic Cells (K)	81	103	154	192	256	469	504	600	653	747	926	1,143
		CLB Flip-Flops (K)	74	94	141	176	234	429	461	548	597	682	847	1,045
		CLB LUTs (K)	37	47	71	88	117	215	230	274	299	341	423	523
	Memory	Max. Distributed RAM (Mb)	1.0	1.2	1.8	2.6	3.5	6.9	6.2	8.8	9.1	11.3	8.0	9.8
		Total Block RAM (Mb)	3.8	5.3	7.6	4.5	5.1	25.1	11.0	32.1	21.1	26.2	28.0	34.6
		UltraRAM (Mb)	-	-	-	13.5	18.0	-	27.0	-	22.5	31.5	28.7	36.0
	Clocking	Clock Management Tiles (CMTs)	3	3	3	4	4	4	8	4	8	4	11	11
	Integrated IP	DSP Slices	216	240	360	728	1,248	1,973	1,728	2,520	2,928	3,528	1,590	1,968
		PCI Express® Gen 3x16	-	-	-	2	2	-	2	-	4	-	4	5
		150G Interlaken	-	-	-	-	-	-	-	-	1	-	2	4
		100G Ethernet MAC/PCS w/RS-FEC	-	-	-	-	-	-	-	-	2	-	2	4
		AMS - System Monitor	1	1	1	1	1	1	1	1	1	1	1	1
	Transceivers	GTH 16.3Gb/s Transceivers	-	-	-	16	16	24	24	24	32	24	44	44
		GTY 32.75Gb/s Transceivers	-	-	-	-	-	-	-	-	16	-	28	28
	Speed Grades	Extended[2]	-1 -2 -2L			-1 -2 -2L -3				-1 -2 -2L -3				
		Industrial	-1 -1L -2											

Figure 1-2 EG series features

In the mid-range series, the processor is quad-core A53, the main frequency can reach 1.5GHz, the actual stable frequency is lower, only about 1.2GHz, and the high-speed interfaces on the ps side include USB3.1, STAT3.0, DP, GEthernet, and PCIe. The logic unit can choose different devices according to the demand. If the high-speed demand is relatively high, you can choose 4EG, 5EG upwards, if you need 100Gbps communication data, you need to choose 11EG and 17EG, 19EG

these three models. The biggest difference from the CG series is the addition of GPU processors on the PS side.

3，High-end EV series, as shown：

Zynq® UltraScale+™ MPSoCs: EV Devices

		Device Name[1]	ZU4EV	ZU5EV	ZU7EV
Processing System (PS)	Application Processor Unit	Processor Core	**Quad-core** Arm® Cortex®-A53 MPCore™ up to 1.5GHz		
		Memory w/ECC	L1 Cache 32KB I / D per core, L2 Cache 1MB, on-chip Memory 256KB		
	Real-Time Processor Unit	Processor Core	**Dual-core** Arm Cortex-R5F MPCore™ up to 600MHz		
		Memory w/ECC	L1 Cache 32KB I / D per core, Tightly Coupled Memory 128KB per core		
	Graphic & Video Acceleration	Graphics Processing Unit	Mali™-400 MP2 up to 667MHz		
		Memory	L2 Cache 64KB		
	External Memory	Dynamic Memory Interface	x16: DDR4 w/o ECC; x32/x64: DDR4, LPDDR4, DDR3, DDR3L, LPDDR3 w/ ECC		
		Static Memory Interfaces	NAND, 2x Quad-SPI		
	Connectivity	High-Speed Connectivity	PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet		
		General Connectivity	2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO		
	Integrated Block Functionality	Power Management	Full / Low / PL / Battery Power Domains		
		Security	RSA, AES, and SHA		
		AMS - System Monitor	10-bit, 1MSPS – Temperature and Voltage Monitor		
PS to PL Interface			12 x 32/64/128b AXI Ports		
Programmable Logic (PL)	Programmable Functionality	System Logic Cells (K)	192	256	504
		CLB Flip-Flops (K)	176	234	461
		CLB LUTs (K)	88	117	230
	Memory	Max. Distributed RAM (Mb)	2.6	3.5	6.2
		Total Block RAM (Mb)	4.5	5.1	11.0
		UltraRAM (Mb)	13.5	18.0	27.0
	Clocking	Clock Management Tiles (CMTs)	4	4	8
	Integrated IP	DSP Slices	728	1,248	1,728
		Video Codec Unit (VCU)	1	1	1
		PCI Express® Gen 3x16	2	2	2
		150G Interlaken	-	-	-
		100G Ethernet MAC/PCS w/RS-FEC	-	-	-
		AMS - System Monitor	1	1	1
	Transceivers	GTH 16.3Gb/s Transceivers	16	16	24
		GTY 32.75Gb/s Transceivers	-	-	-
	Speed Grades	Extended[2]	-1 -2 -2L -3		
		Industrial	-1 -1L -2		

Figure 1-3 EVseries features

The high-end EV series has a quad-core A53 processor, the main frequency can reach 1.5GHz, the actual stable frequency is lower, only about 1.2GHz, and the high-speed interfaces on the ps side include USB3.1, STAT3.0, DP, GEthernet, and PCIe. The logic unit can choose different devices as the demand is not available. If the high-speed demand is relatively high, you can choose 4EV, 5EV, and 7EV. If you need 100Gbps communication data, you still need to choose the mid-range 11EG and 17EG, 19EG. Three models. The biggest difference from the EG series is the addition of the VCU hard core module on the PL end, which can handle the video encoding and decoding of the H.264/H.265 protocol.

1.4 Vivado Project List

The projects introduced in this document can be seen in the following list, and the corresponding project documents can be downloaded and used in the information channels announced by the company. The specific development process is described in detail in Chapter 3。

Table 1 project list

Num	Name	Description
1	hello_world.rar	Introduced as a basic configuration project in Chapter 3
2	gpio_mio.rar	In the third chapter, the introduction of

		the use of ps-side LED lights is introduced
3	IIc_test.rar	Introduction to the use of the IIC peripheral bus on the PS side in Chapter 4
4	uart_cycle.rar	In the fourth chapter, we will introduce peripheral experiments based on the AXI-Lite bus
5	dma_loop.rar	In the fifth chapter, the introduction experiment using AXI4 bus
6	bram_test.rar	In the fifth chapter, use the introduction experiment of AXI4-Lite bus
7	MIPI_DP_4G	In the sixth chapter, the introduction experiment using AXI-Stream bus

1.5Sections of this Chapter

This chapter mainly introduces the summary of the entire document. The later chapters are basically mentioned in this chapter. The detailed content needs to be checked in the specific chapters, or the corresponding reference materials or projects can be found. For the MYIR MYS-ZU5EV hardware platform, the main application directions are edge computing, image processing and artificial intelligence.

Chapter 2 MYS-ZU5EV Preparation

2.1 Hardware Preparation

A little introduction to the basic characteristics of the MYS-ZU5EV hardware platform，as show in figure2-1

Figure 2-1 Hardware platform products

MYS-ZU5EV-32E4D-EDGE use Xilinx XCZU5EV-SFVC784 decice，The speed grade is -2. 32E means 32GEMMC, 4D means 4G DDR storage. XCZU5EV-2SFVC784I supports 1.5GHz (maximum -2) APU speed, 600MHz (maximum -2) RPU speed, 667MHz (maximum -2) GPU speed, and up to 2400Mbps DDR4 speed.

XCZU5EV-2SFVC784I device resources：

Processor System Unit (PS):

Processing core: Quad-core ARM Cortex-A53 multi-core processor up to 1.5GHz

The highest clock frequency: 1.5Ghz APU: L1 Cache 32KB I / D per core, L2 Cache 1MB.

RPU: L1 Cache 32KB I / D per core On-chip cache: 256KB

Off-chip interface:

support LPDDR4, DDR4, DDR3, DDR3L LPDDR3 with ECC External static storage: 2x Quad-SPI, NAND

DMA channels:

8 (4 of which are dedicated to PL)

Peripherals:

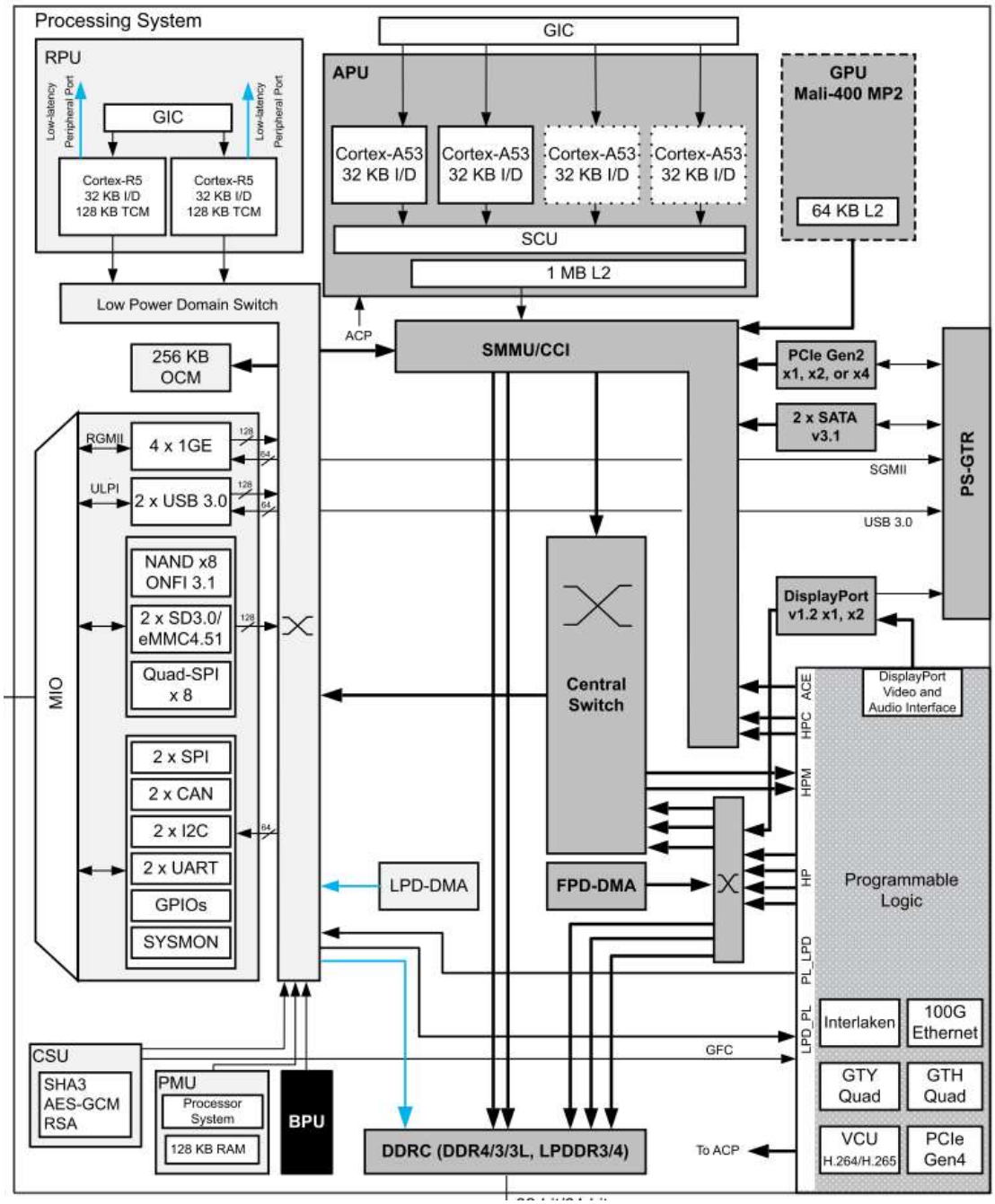


Figure 2-2 MPSOC interconnection block diagram

High performance interface：PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode

Gigabit Ethernet.

General interface：2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO.

Programble Logical Unit（PL）：

Table 2 PL side logic resources

resources	ZU5EV
Logical kernal	Xilinx Kintex Ultrascale+®FPGA
Programable logical Unit	256K
LUTs	117K
register	234K
Block RAM	Distributed RAM 5.1Mb / Block RAM 18.0Mb
DSP Slice	1248
AMS System Monitor	1

2.2 Software Preparation

This section mainly introduces the installation of Vivado2020.1. The installation of vivado2020.1 is applicable to all vivado versions. The difference between vivado versions is not very big. We recommend using the vivado software version of the same version of our engineering documents to facilitate the verification of the vivado software version we provide. Engineering documentation routines.

Why install vivado instead of ISE, here is the evolution process: As the FPGA chip enters the 28nm process, the performance of the ISE software can no longer follow the huge improvement in the hardware performance caused by the progress of the process, so there is the plan of vivado, although the follow-up ISE continues to update in 2008, it has also been added. For 28nm series devices and subsequent device libraries, it is necessary to accelerate design and development and improve software efficiency.

Vivado®DesignSuite aims to improve the overall work efficiency of designing, integrating and implementing systems using Xilinx®UltraScale™ and 7 series devices, Zynq®UltraScale+™ MPSoC devices and Zynq®-7000 SoC.

Then there are various problems with different versions of vivado. The first is the update of IP, and the second is that there will be unexpected problems in the process of generating bitstream in the comprehensive realization of vivado. It may be caused by problems such as spaces in the path. It may be caused by other

unknown reasons. For smooth engineering development, the same version is recommended. Secondly, there may be inexplicable reasons when using vivado. This reason is sometimes caused by errors or death of the PC processor process, or it may be caused by inconsistent use of vivado. Various complex reasons, if they occur, A recommended solution is to uninstall vivado and reinstall.

2.2.1 Vivado Download

To download the Vivado tool, you need to register an account on the xilinx official website. After completing the registration, you can find "Product——"Hardware Development——"Vivado Design Suit——"Find the corresponding version" to download. as the picture shows

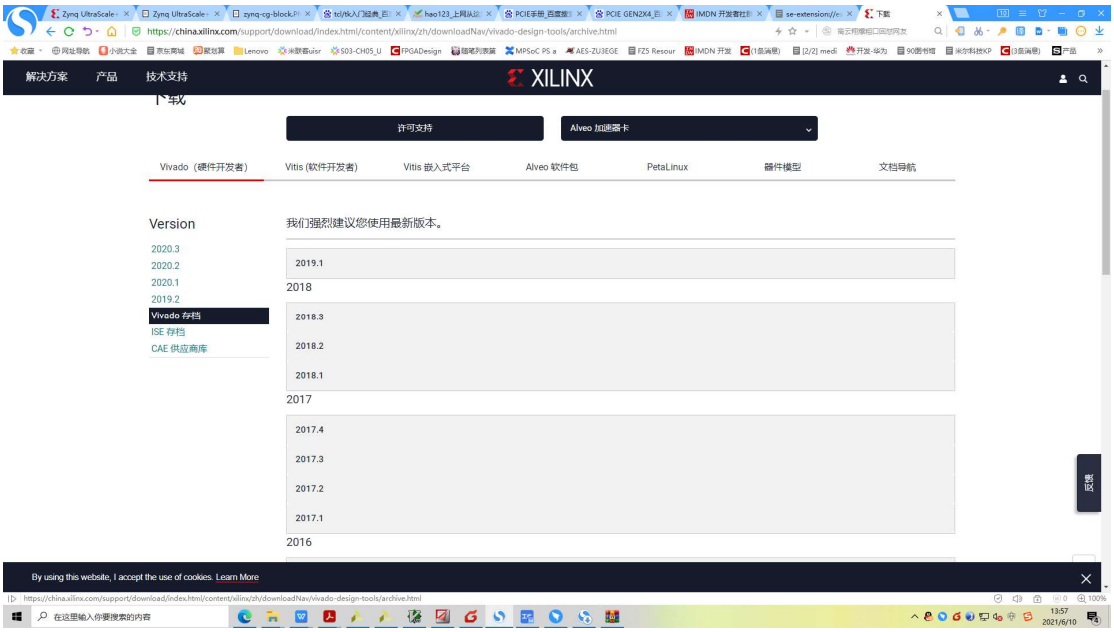


Figure 2-3vivado download page

You can see that the serial numbers of various versions are there. Download and select "ALL OS installer Signaler-file download". Although the installation package is relatively large, there are generally no extra problems afterdownloading and installing. And the download speed is relatively fast.

2.2.2 Vivado Installation

Before installation, please close the anti-virus software, especially the 360 anti-virus software.

Step 1: Unzip the downloaded installation document, double-click the installation program, as shown in the figure ,

ucrtbase.dll	2020/5/28 10:21	应用程序扩展	979 KB
vccorlib140.dll	2020/5/28 10:21	应用程序扩展	387 KB
vcruntime140.dll	2020/5/28 10:21	应用程序扩展	86 KB
xsetup	2020/5/28 10:21	文件	3 KB
xsetup	2020/5/28 10:50	应用程序	439 KB

Figure2-4 decompressed vivado installation file

Step 2: Click NEXT on the welcome page:

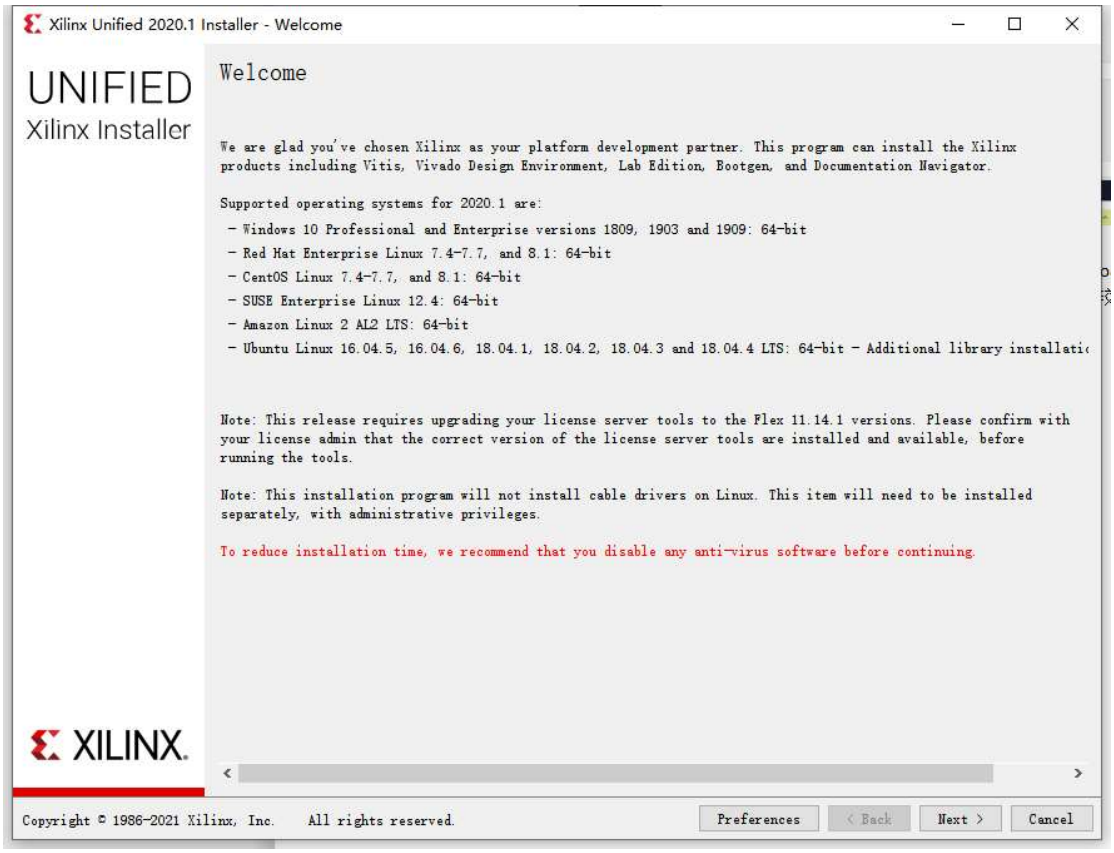


Figure 2-5 Installation Wizard

Step 3: Tick all of them, all agree, NEXT.

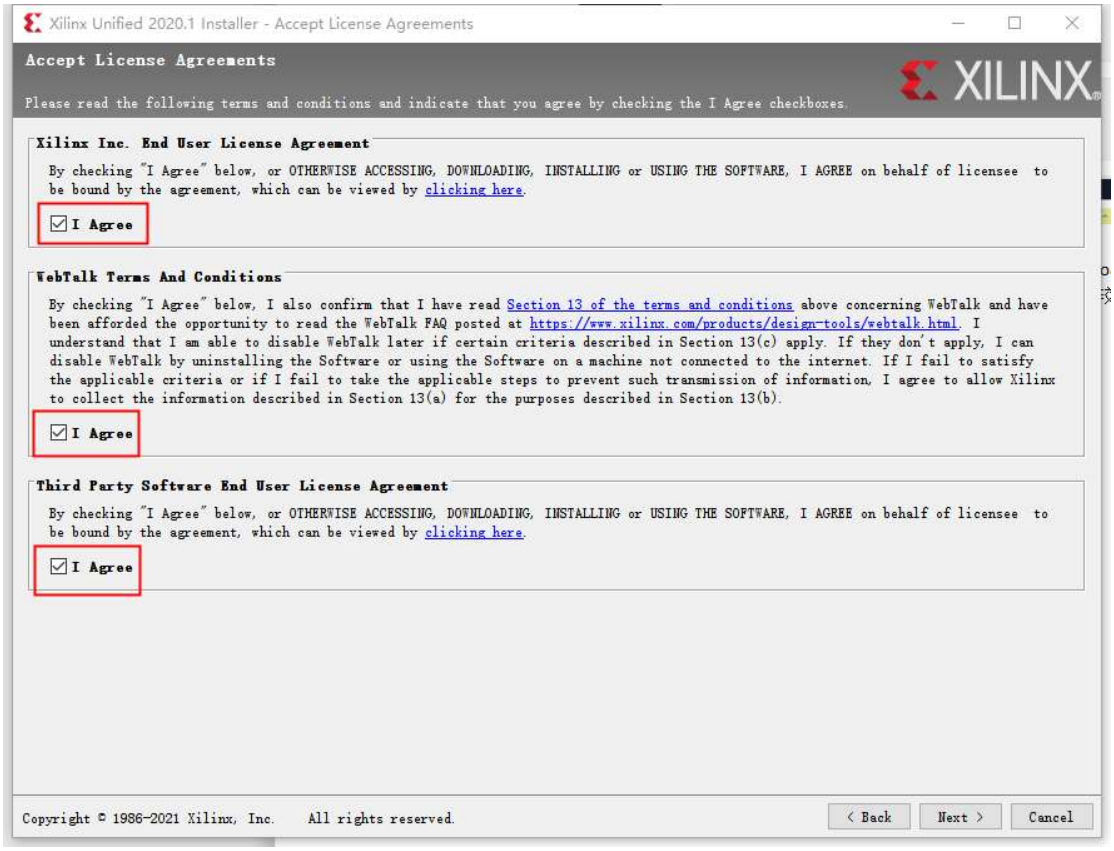


Figure 2-6 installation wiazrd

Step 4: Choose the product to install, because the 2020.1 version is updated from the SDK to the Vitis software tool, so choose Vitis

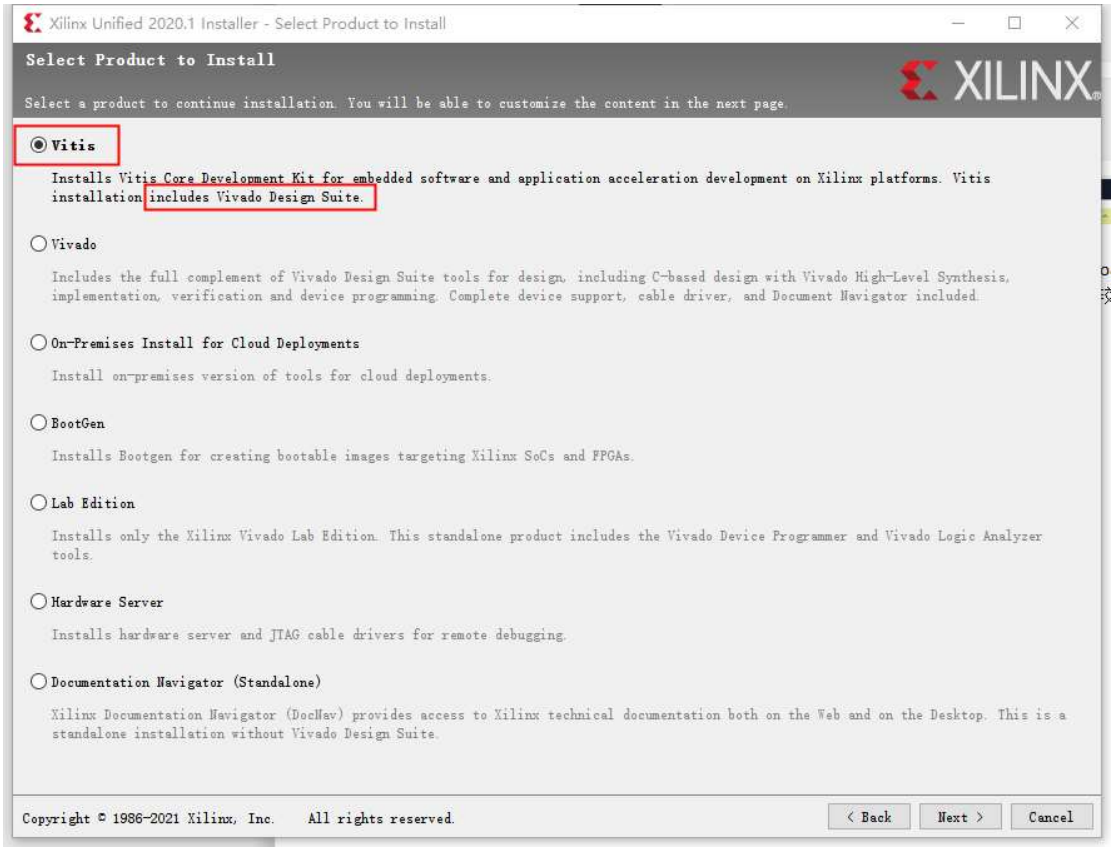


Figure 2-7 installation wizard

Step 5: Select the installation path and user, as shown in the figure

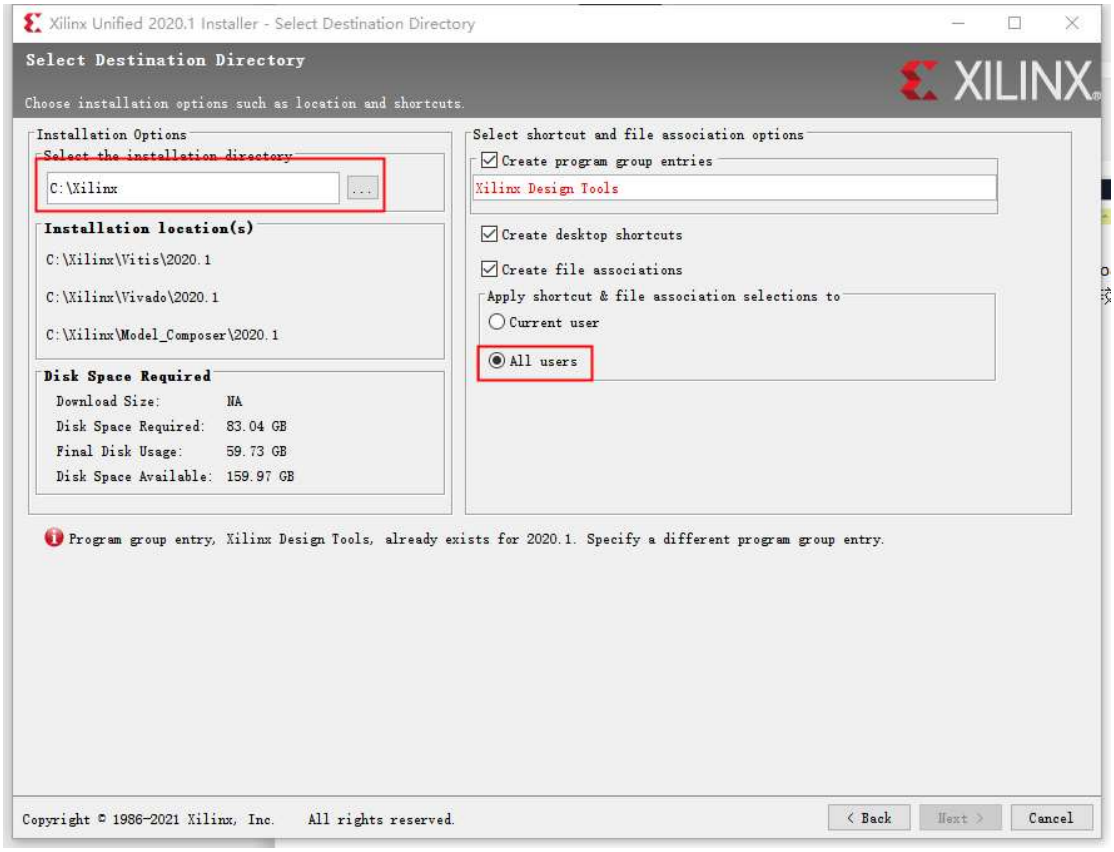


Figure 2-8 installation wizard-installation location

Step 6: Click YES

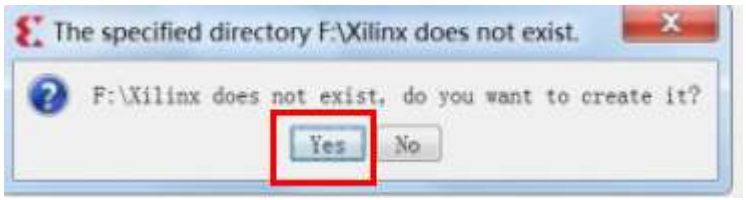


Figure 2-9 installation wizard

Step 7: Click Install, the installation will take about 45 minutes

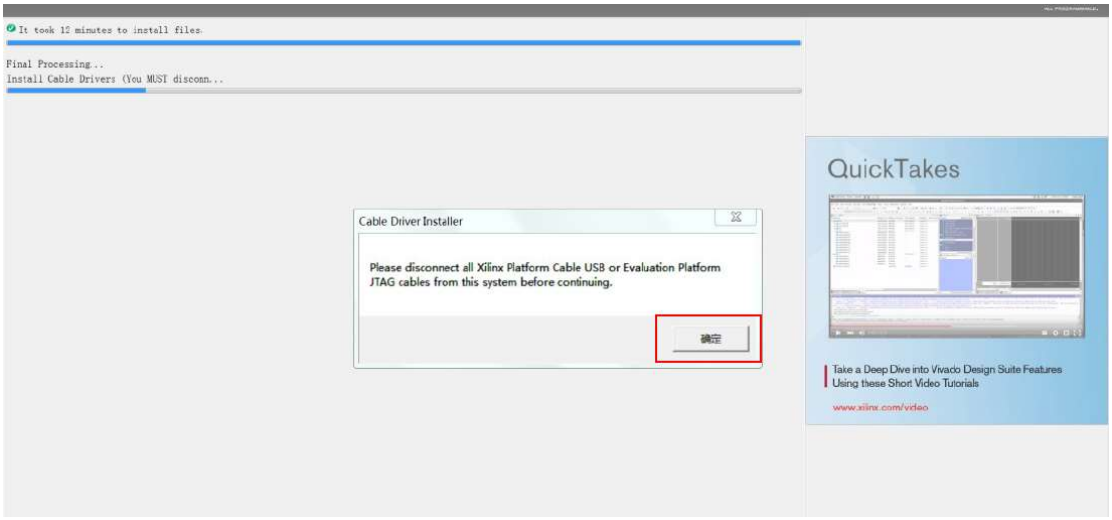


Figure 2-10 installation wizard

Step 8: Install the network adapter

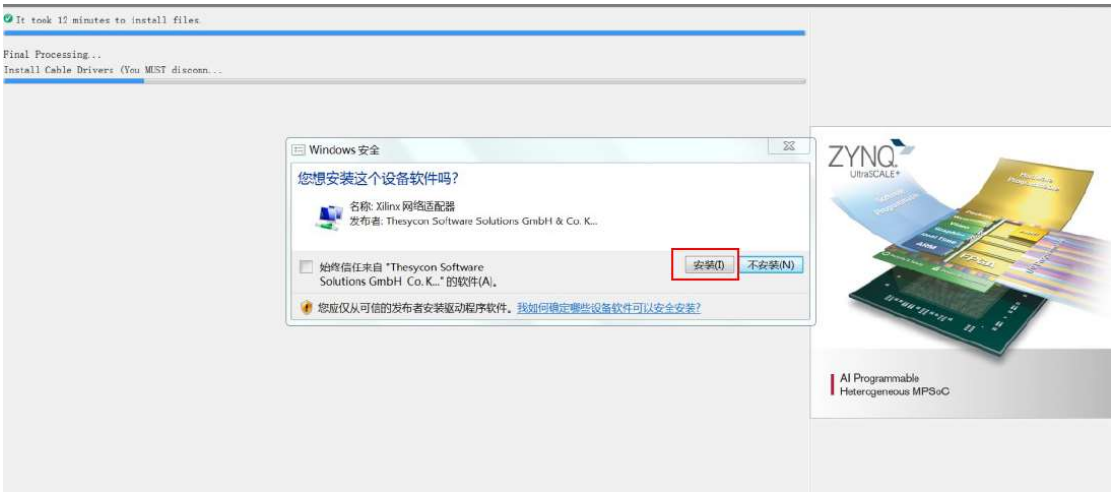


Figure 2-11 installation wizard

Step 9: Install the downloader driver

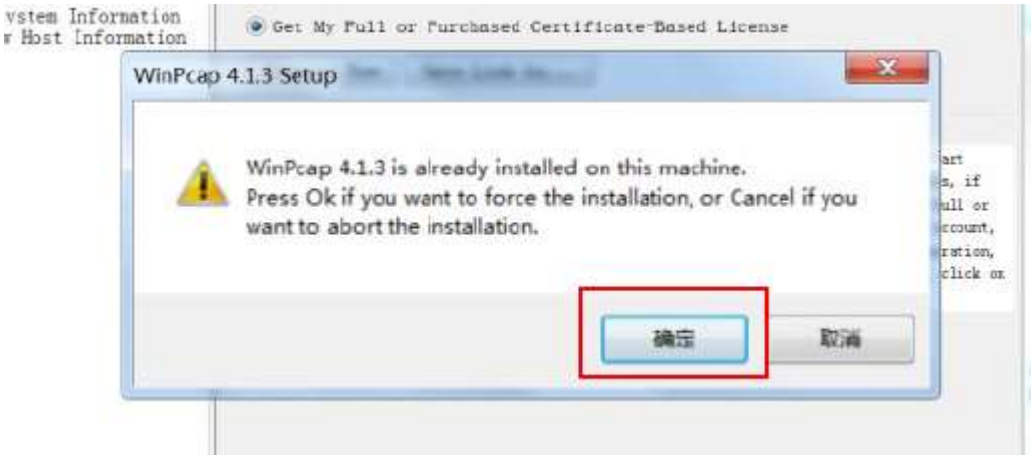


Figure 2-12 installation wizard

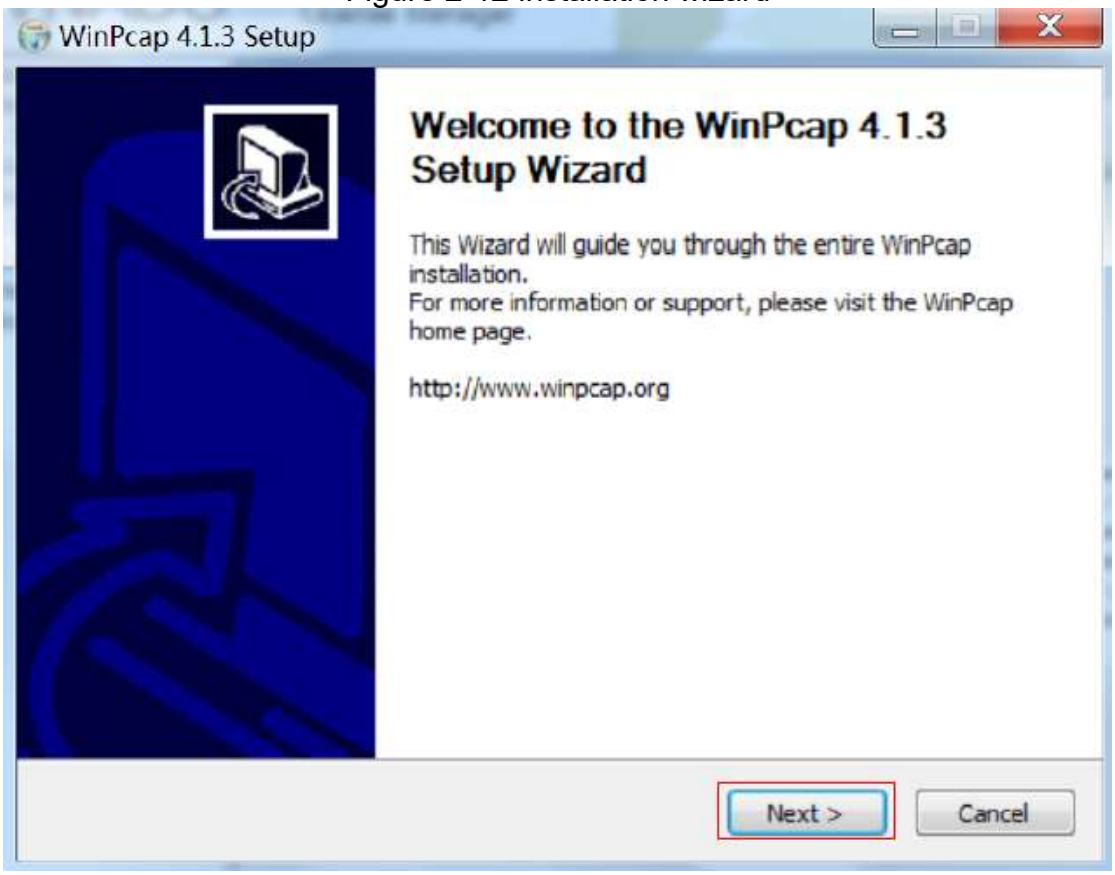


Figure 2-13 installation wizard

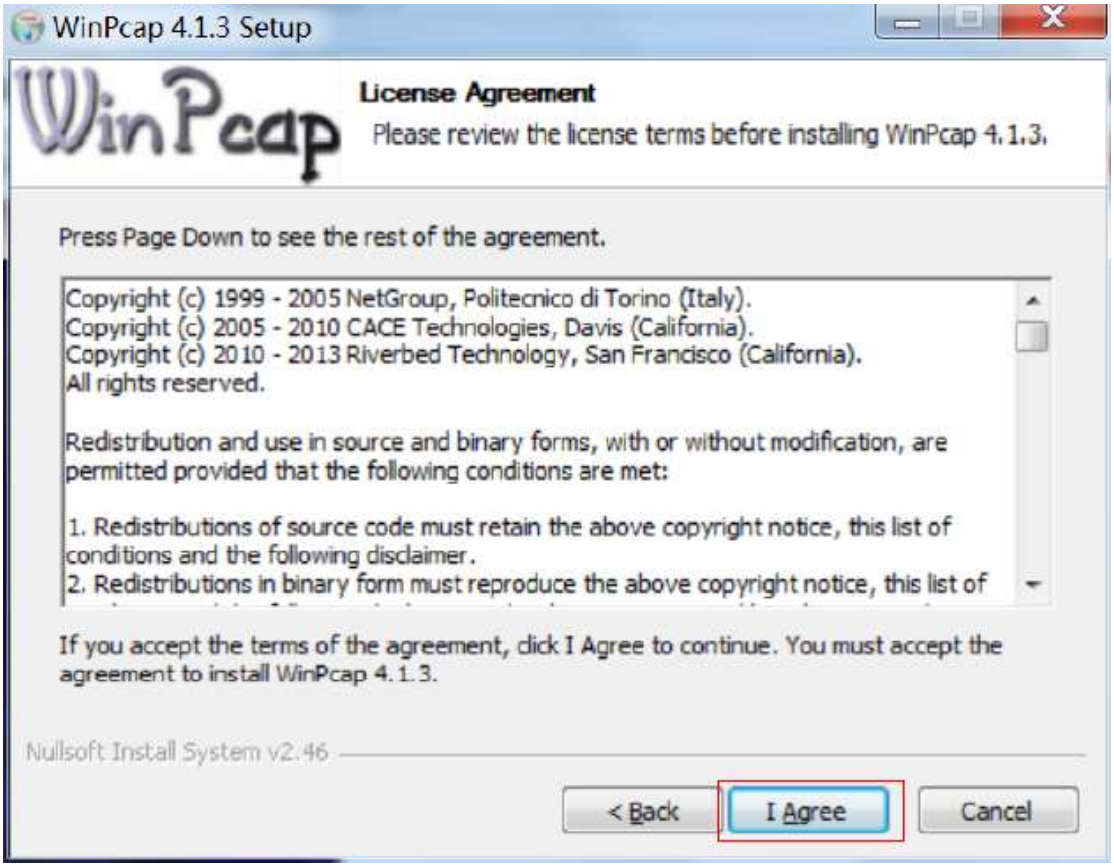


Figure 2-14 installation wizard-install wincap

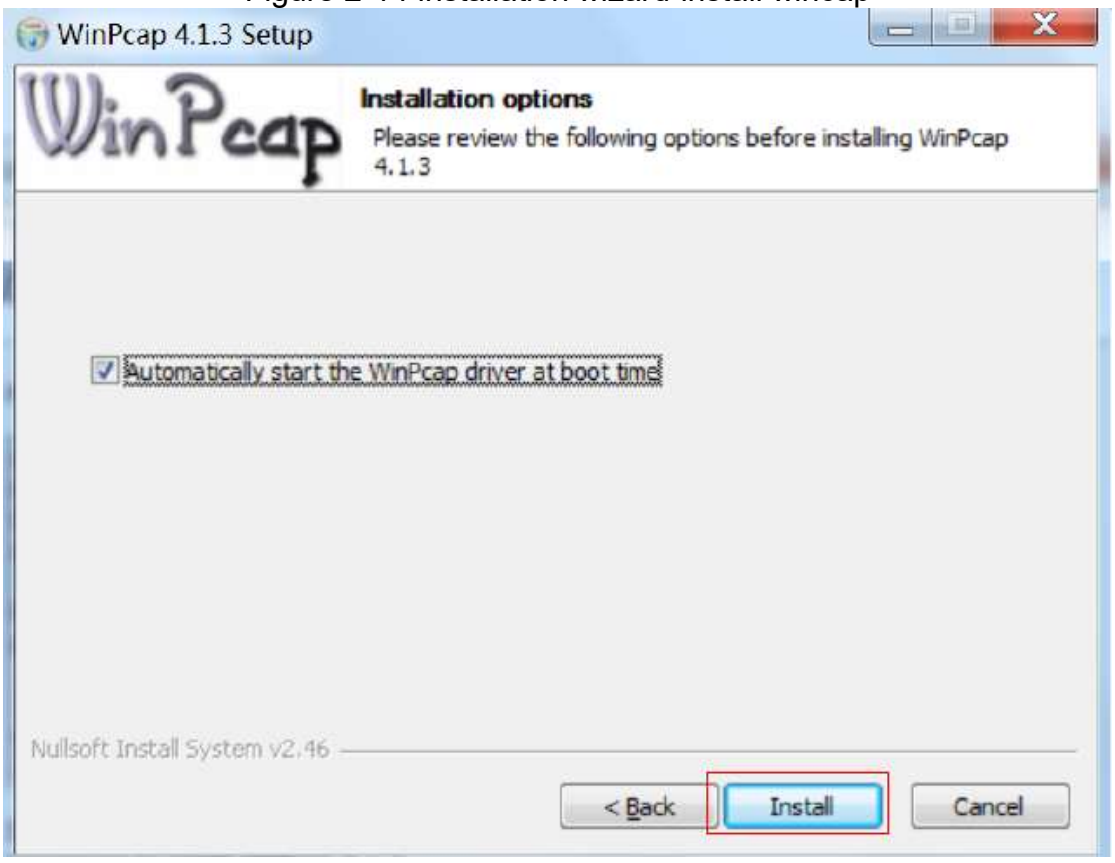


Figure 2-15 installation wizard-install wincap

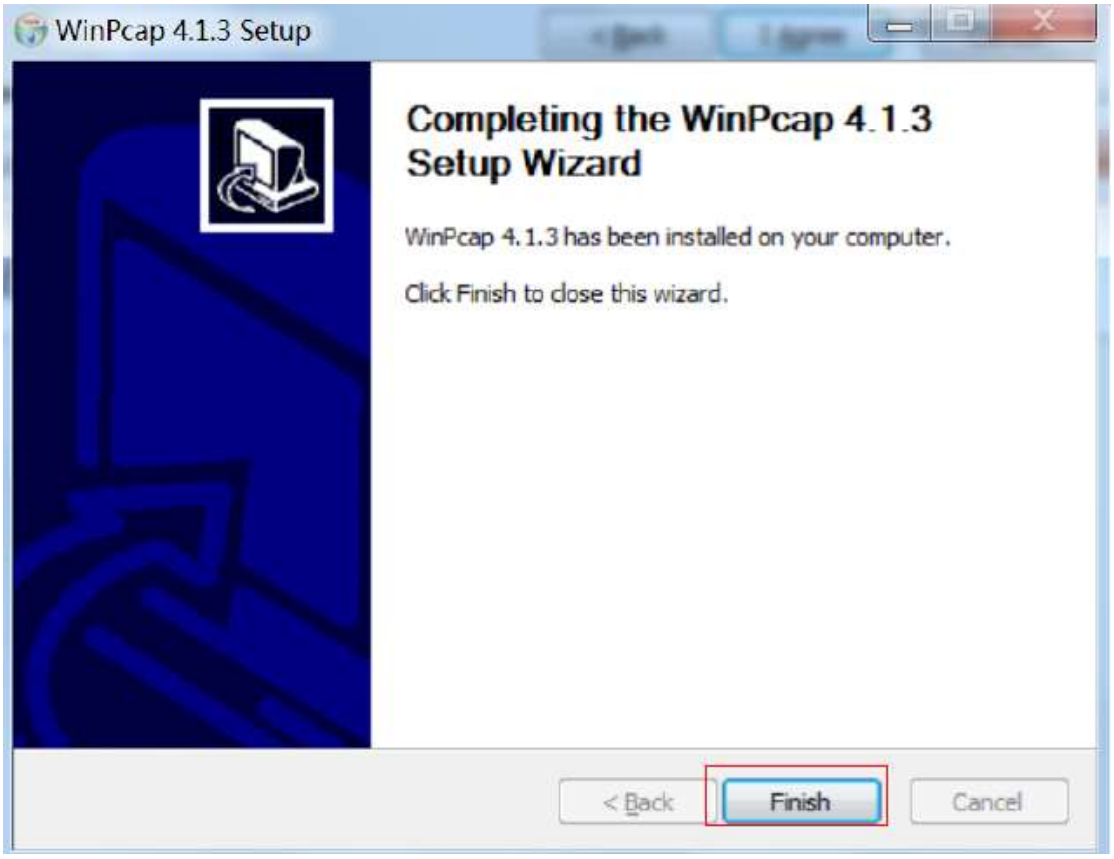


Figure 2-16 installation wizard

Step 10: Finally, a window to install MATLAB will pop up. Just fork it directly. If you need it, you can install it yourself.

Step11: the installation is successful, the window for installing the license pops up, and it is also directly crossed. Explained in the next section. Install wizard registered .

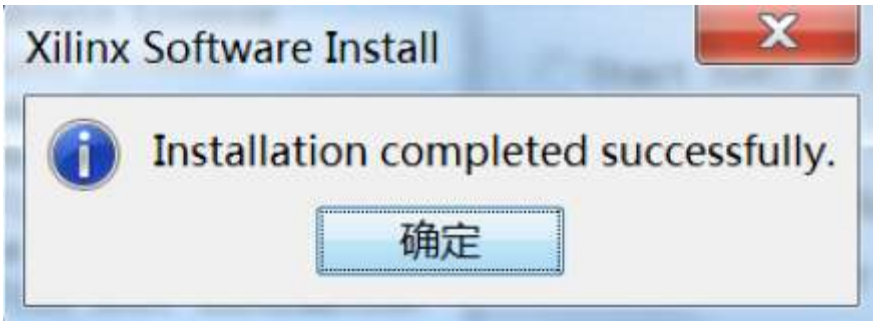


Figure 2-17installation wizard-finished

2.2.3 Vivado License Register

In the previous section, the license registration wascrossed directly during the installation. Now reopen the Vivado license Manager and find it in the start menu bar.

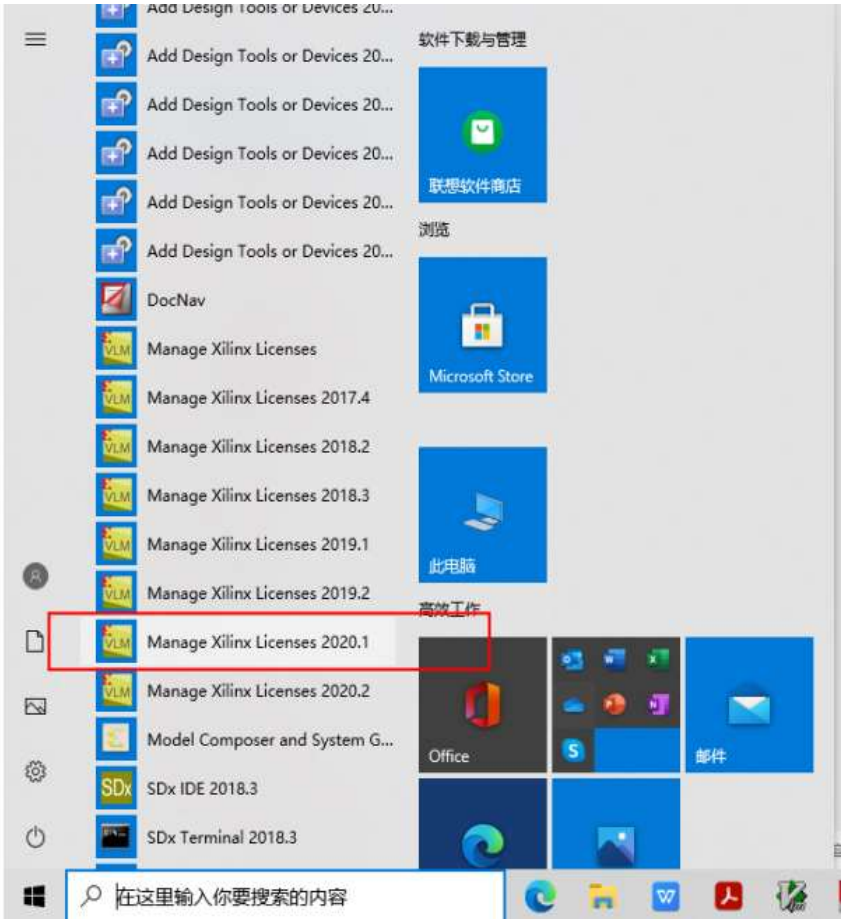


Figure 2-18 Manage Xilinx License

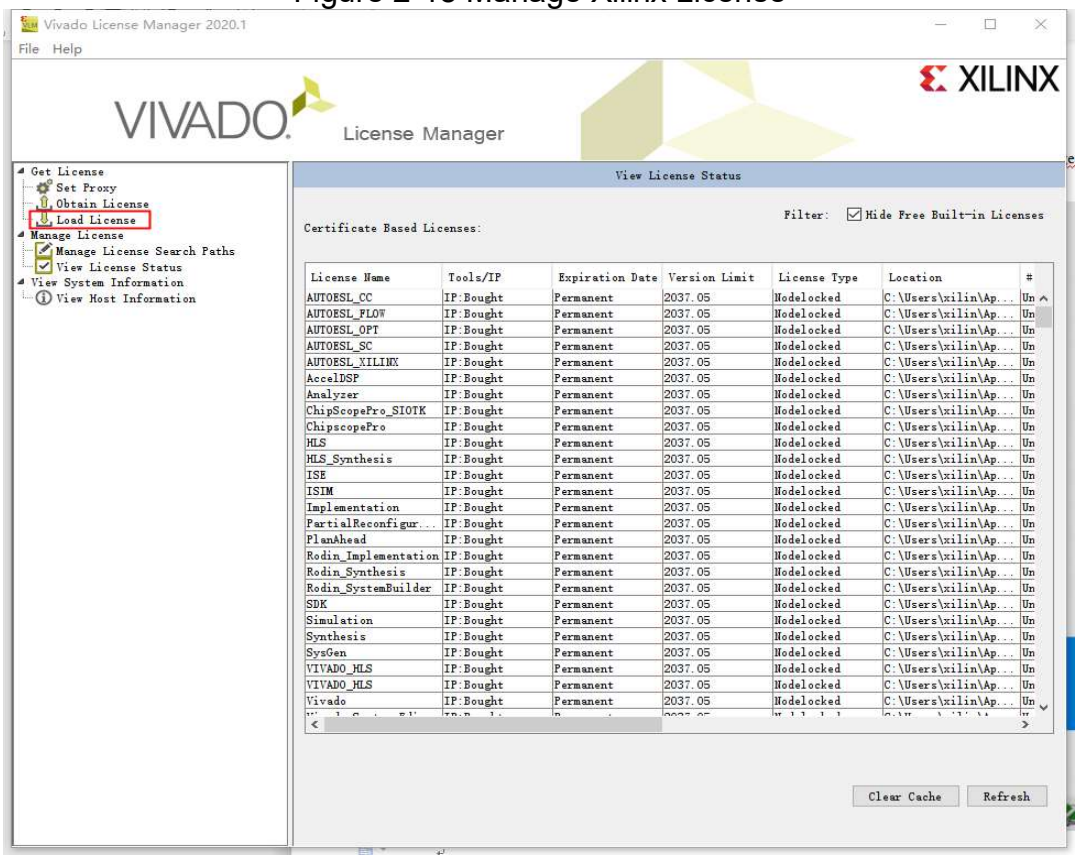


Figure 2-19 installation of license

Chose Load License，then select Copy License...

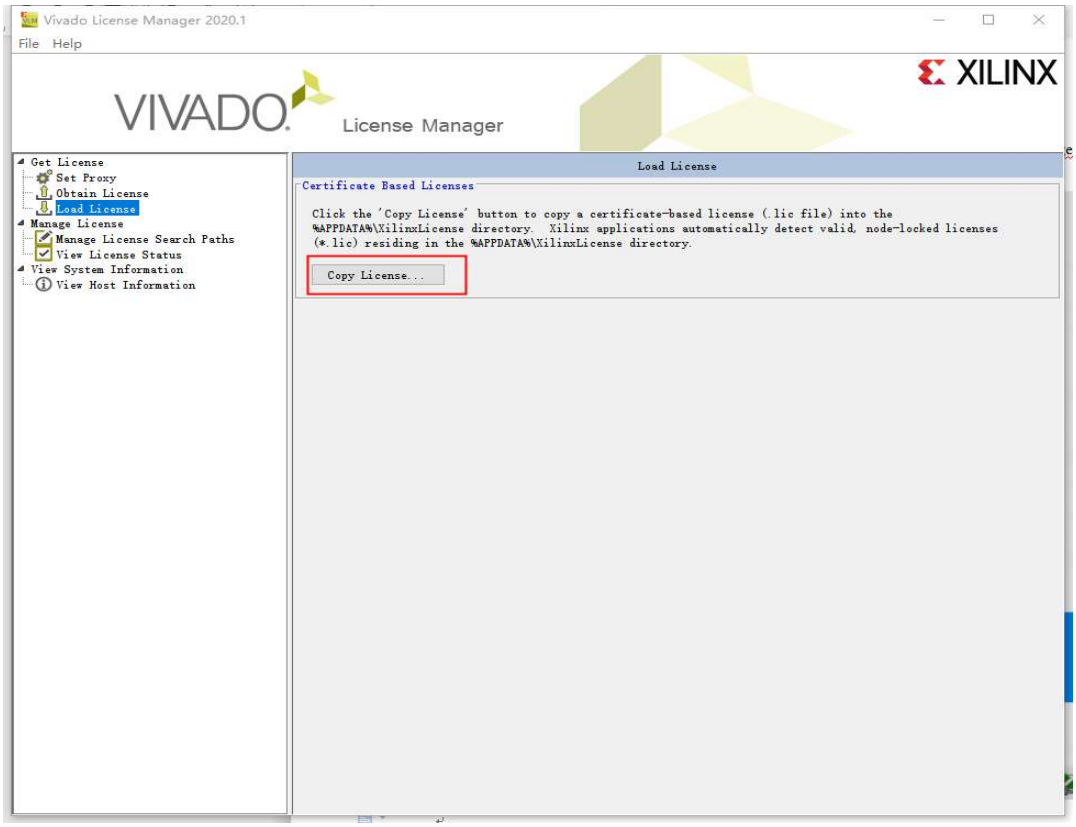


Figure 2-20installation of license

Find the vivadoregister file of Xilinx.lic

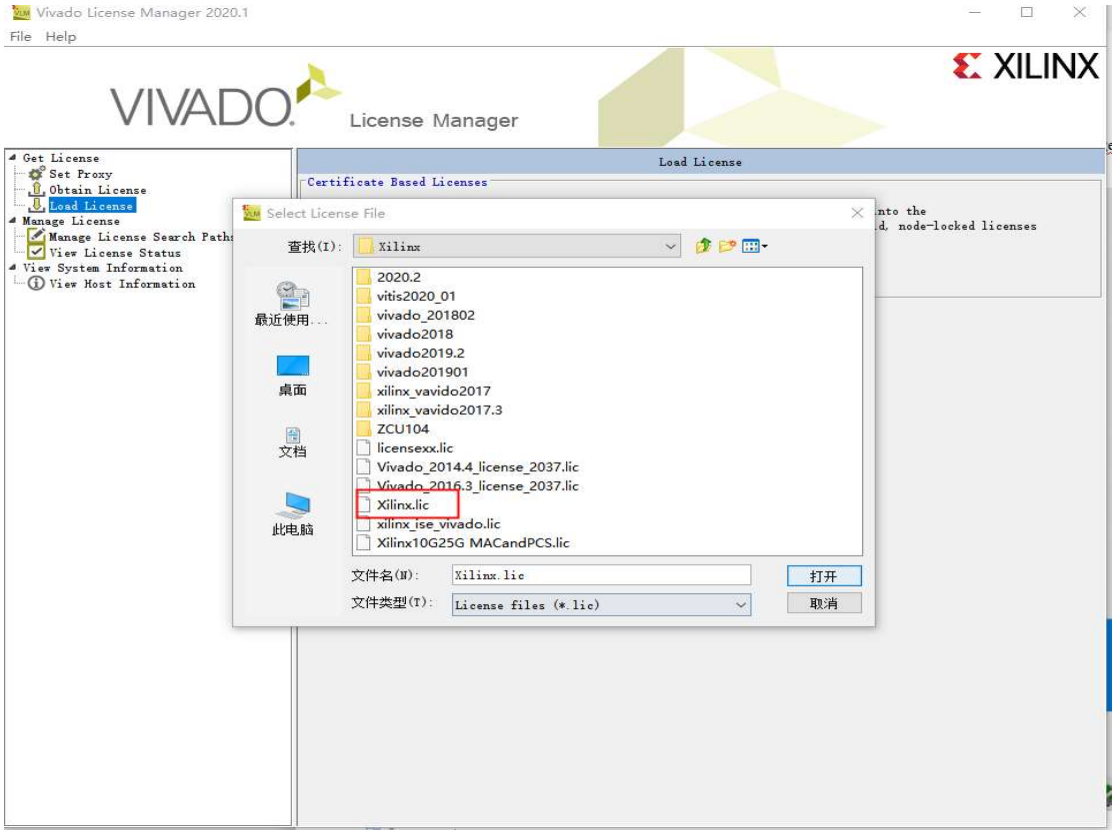


Figure 2-21installation license

At last click View License State，You can verify the license has been registered。

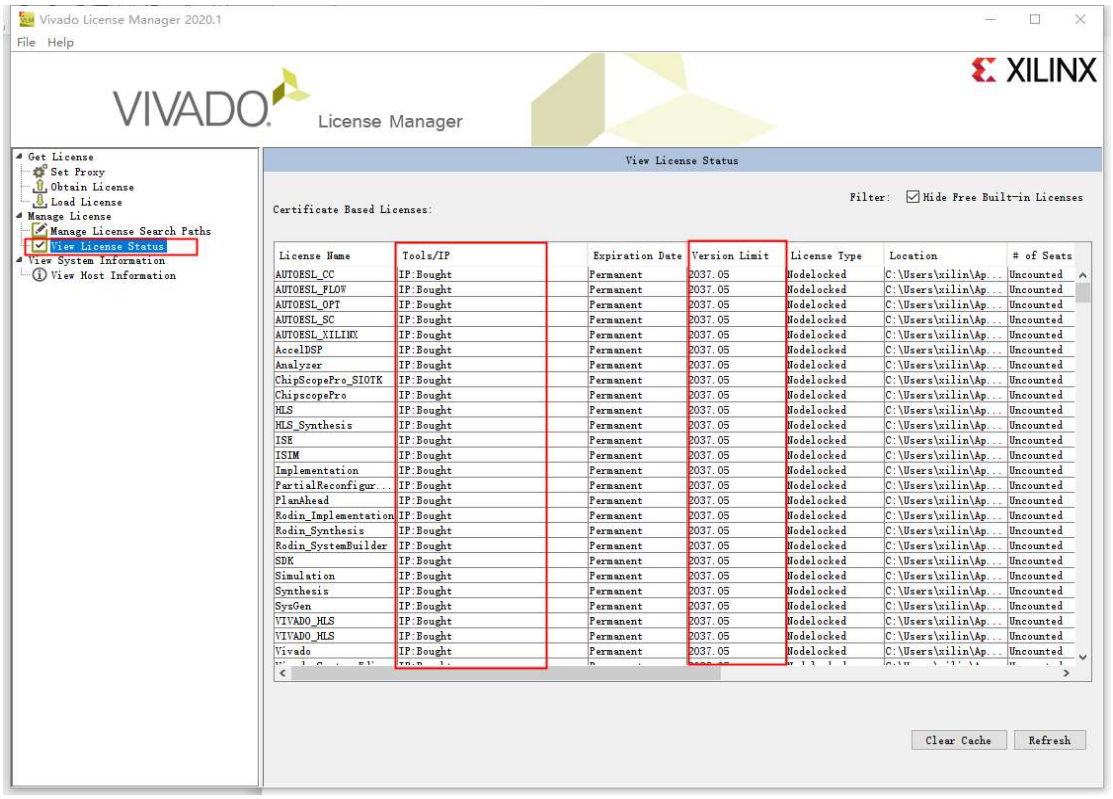


Figure 2-22installation license

2.2.4 Temporary License Registration of Paid IP in Vivado

Let's take an example of an IP that needs to be paid. If we want to use the IP of 10G/25G Ethernet Subsystem, when we configure, the status of the IP is IP: Design_Linking IP License available.

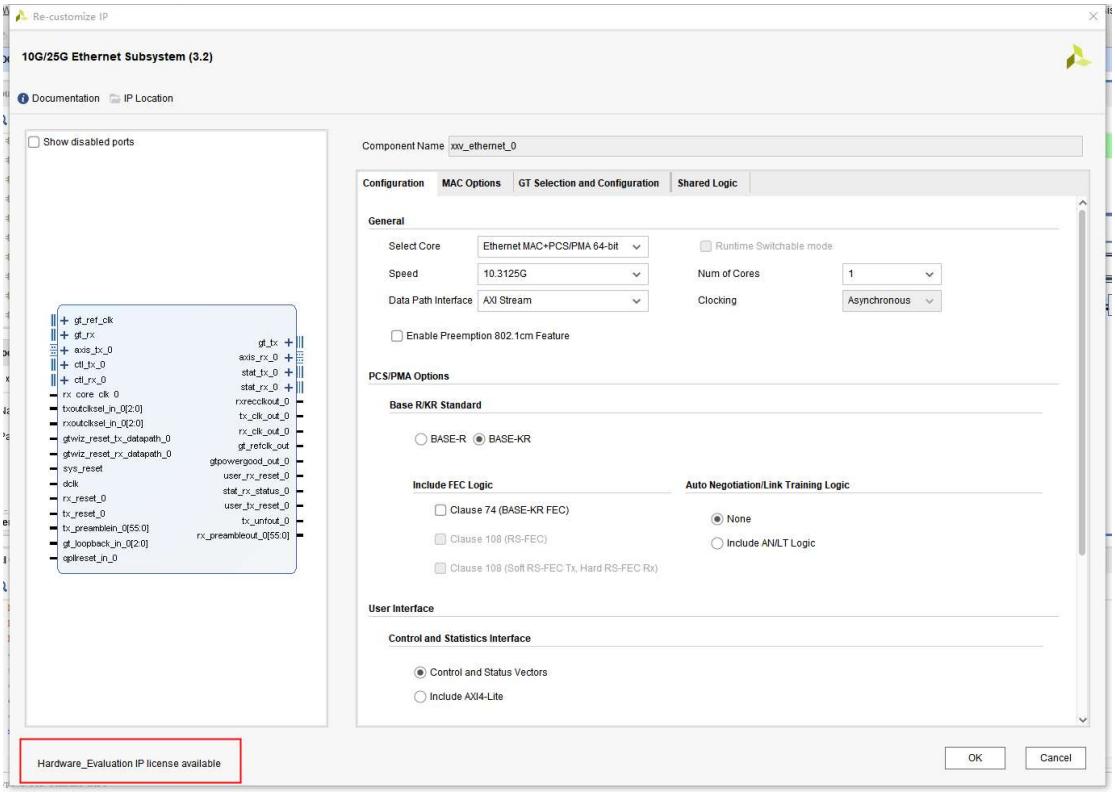


Figure 2-23installation of license state after a lic registered

Vivado License Manager find ,as show below:

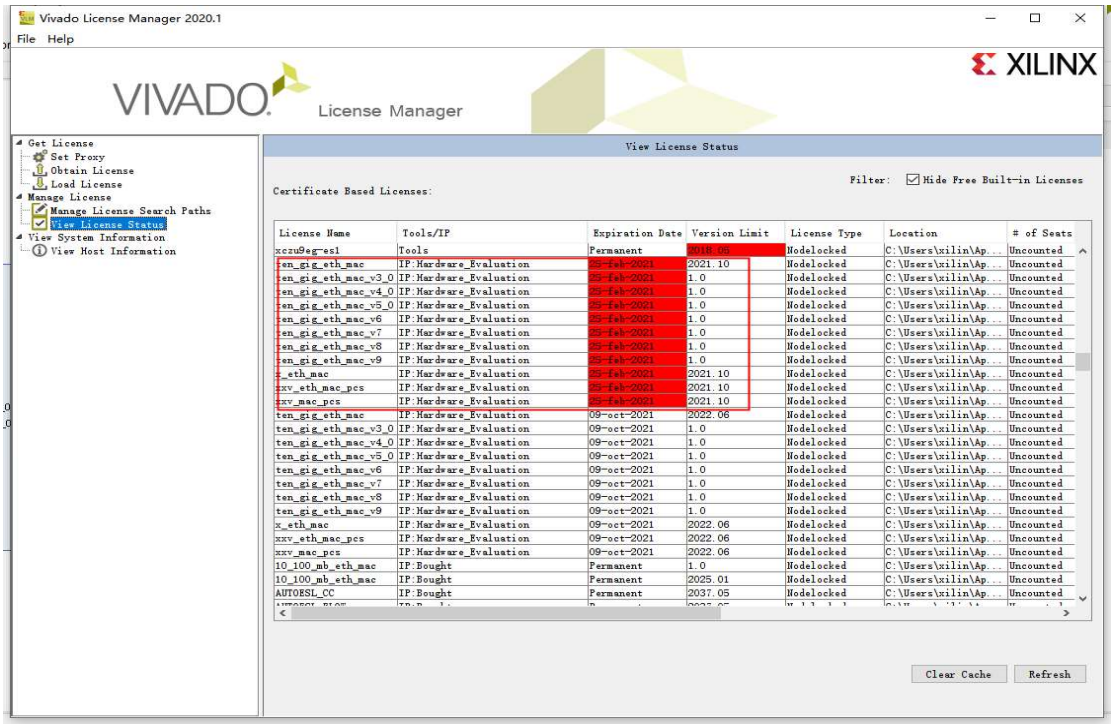


Figure 2-24Installation license

At this time, the license has expired, and it can be simulated but bitstream cannot be generated. Next, we will introduce how to apply for a specific license.

Just follow the steps shown in the figure.

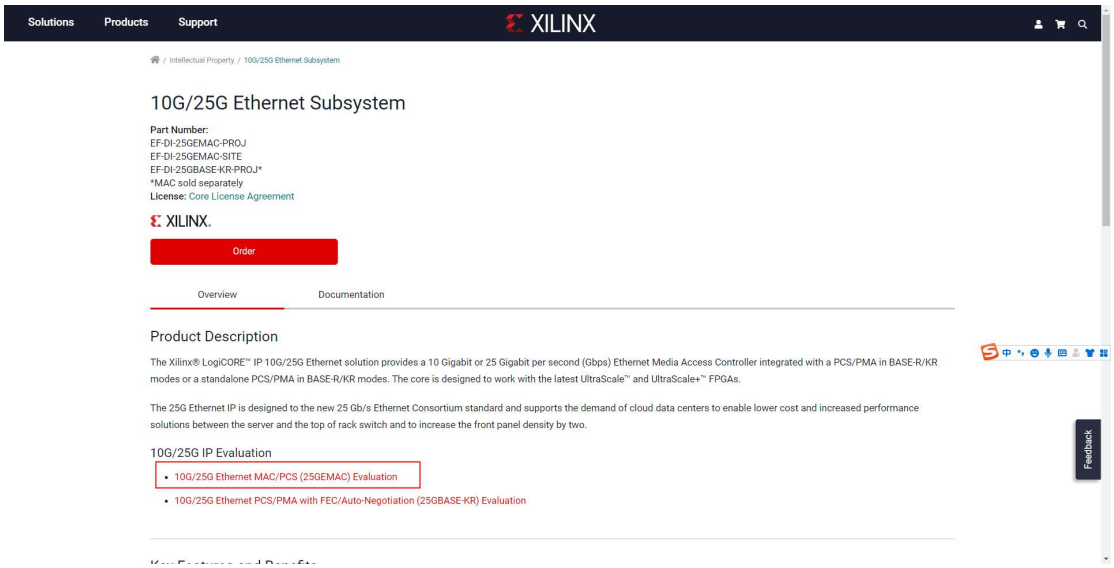


Figure 2-2510G/25G Ethernet Subsystem IP

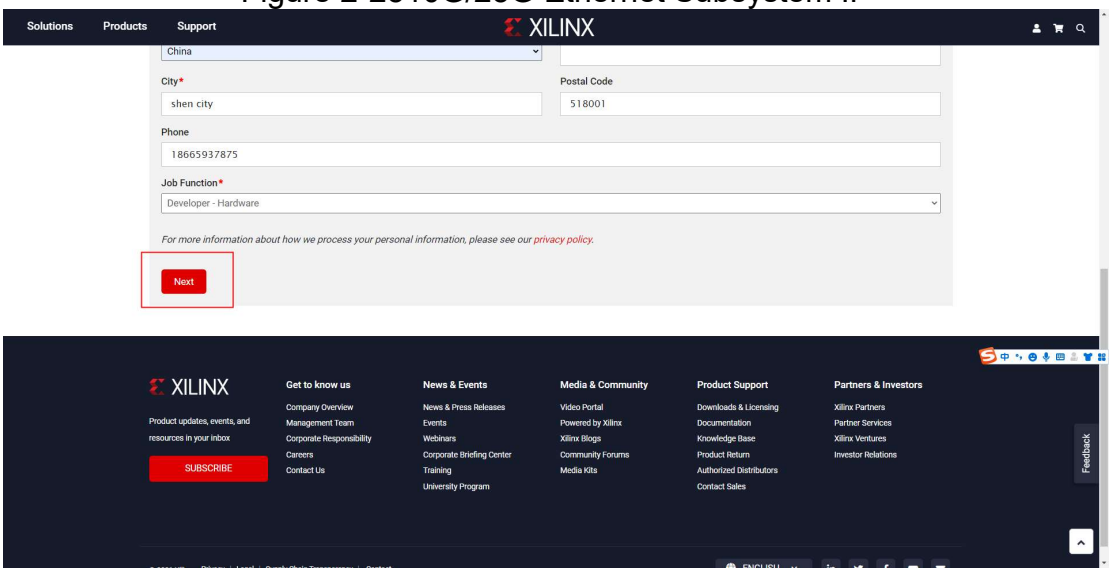


Figure 2-26 Go to license requested permission page

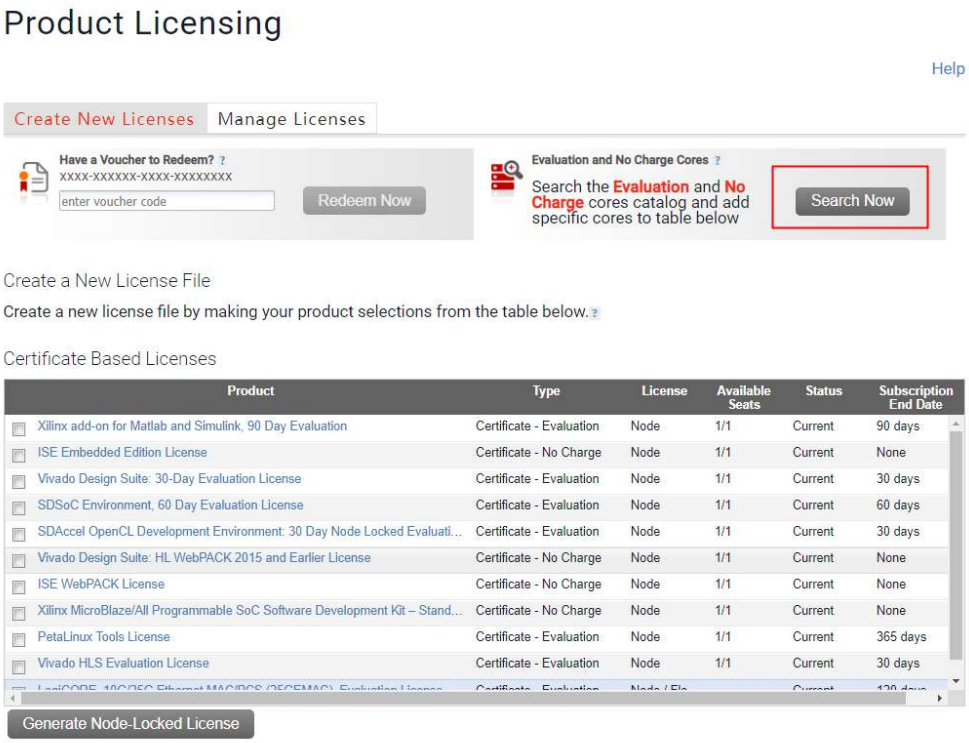


Figure 2-27Product Licensing

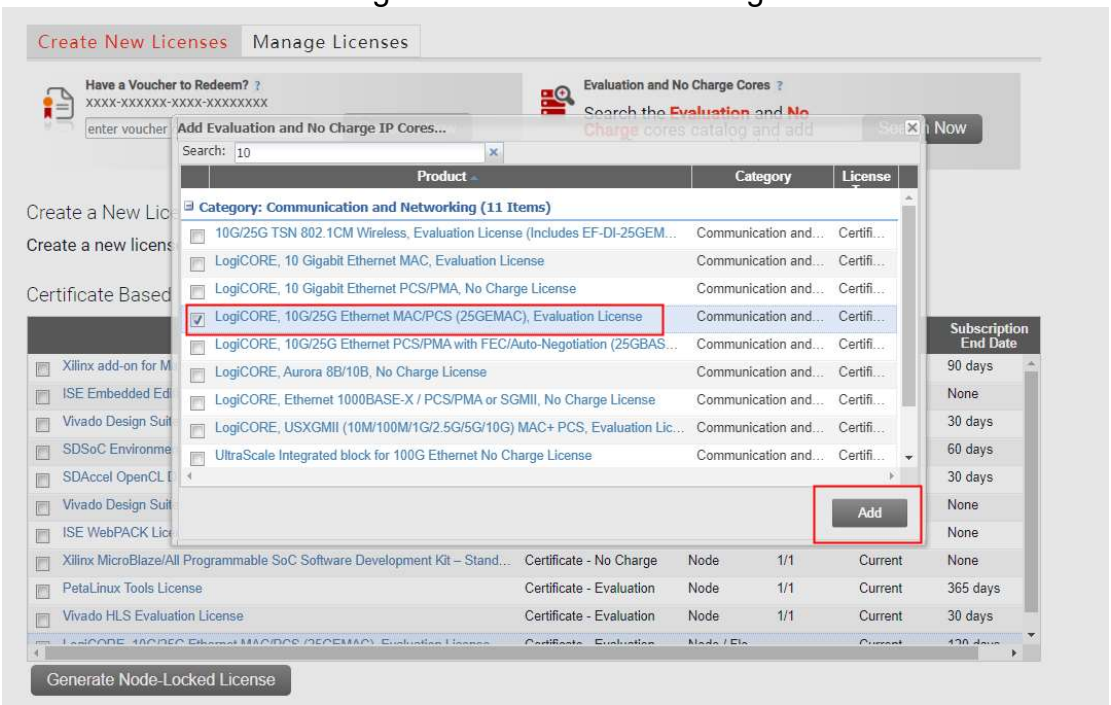


Figure 2-28 Search license

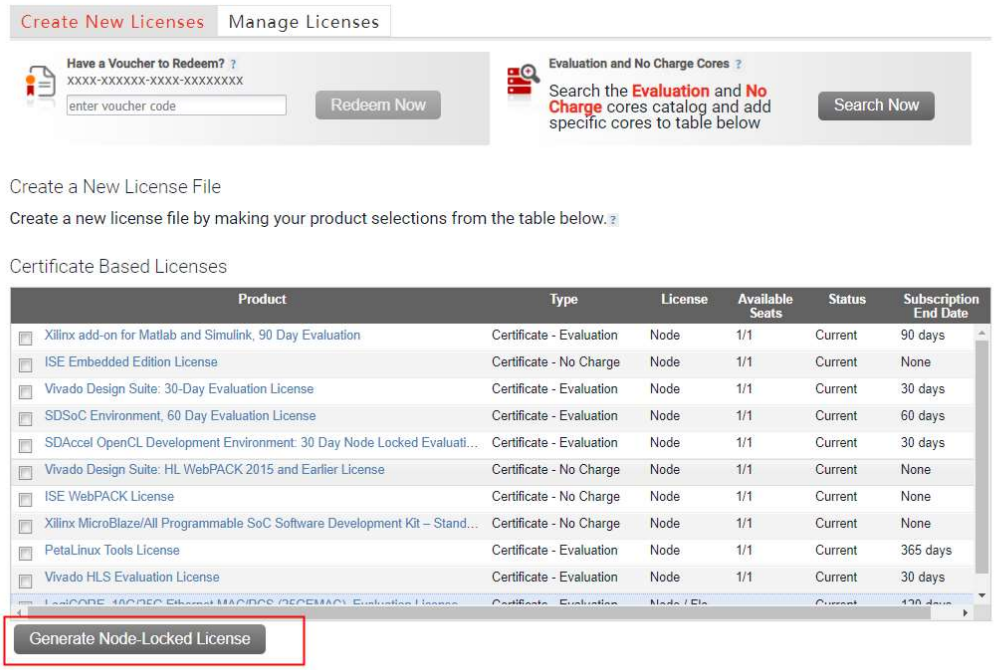


Figure 2-29 Generate license

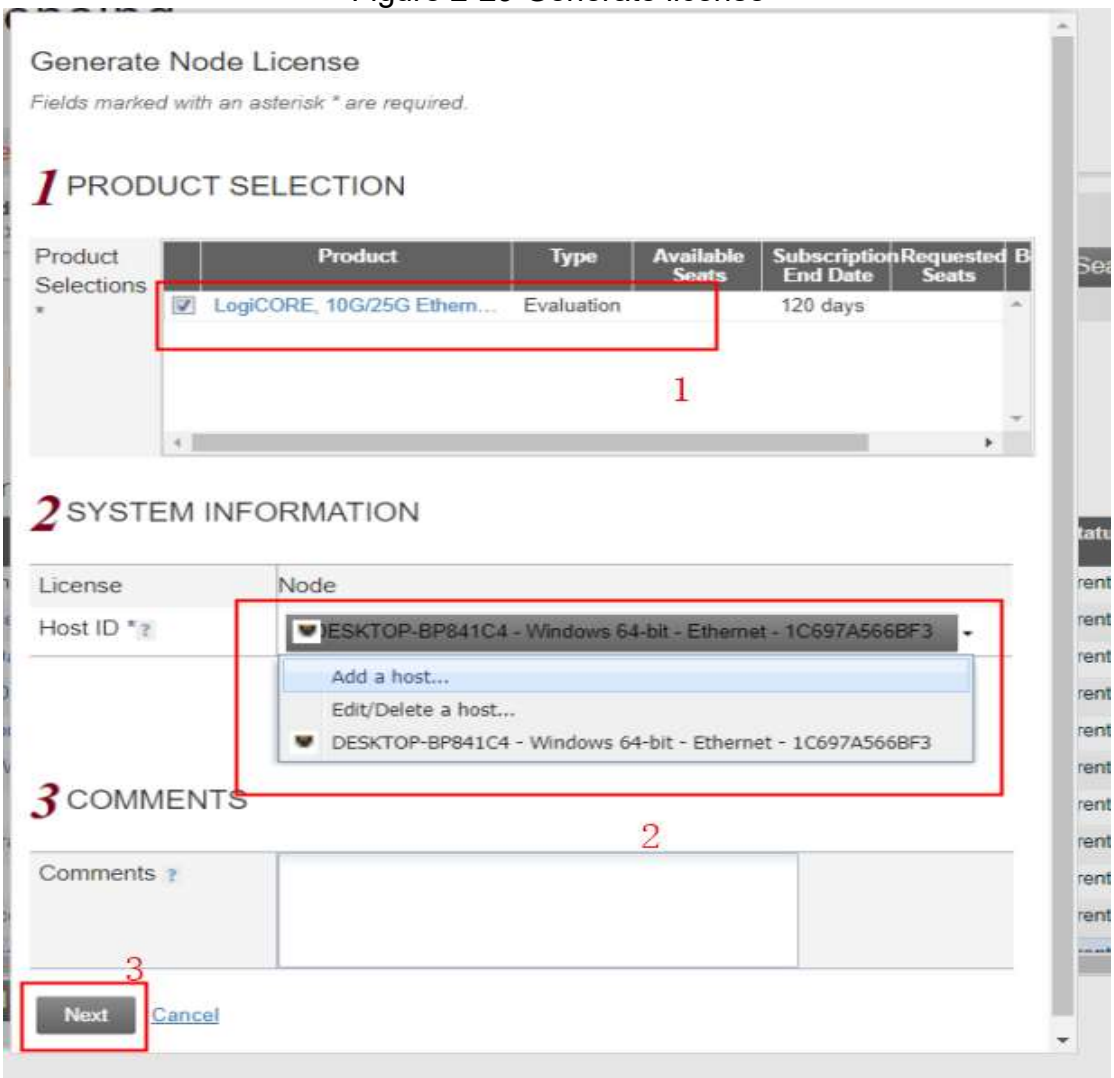


Figure 2-30 Fill ip with you PC MACaddress to generatelicense

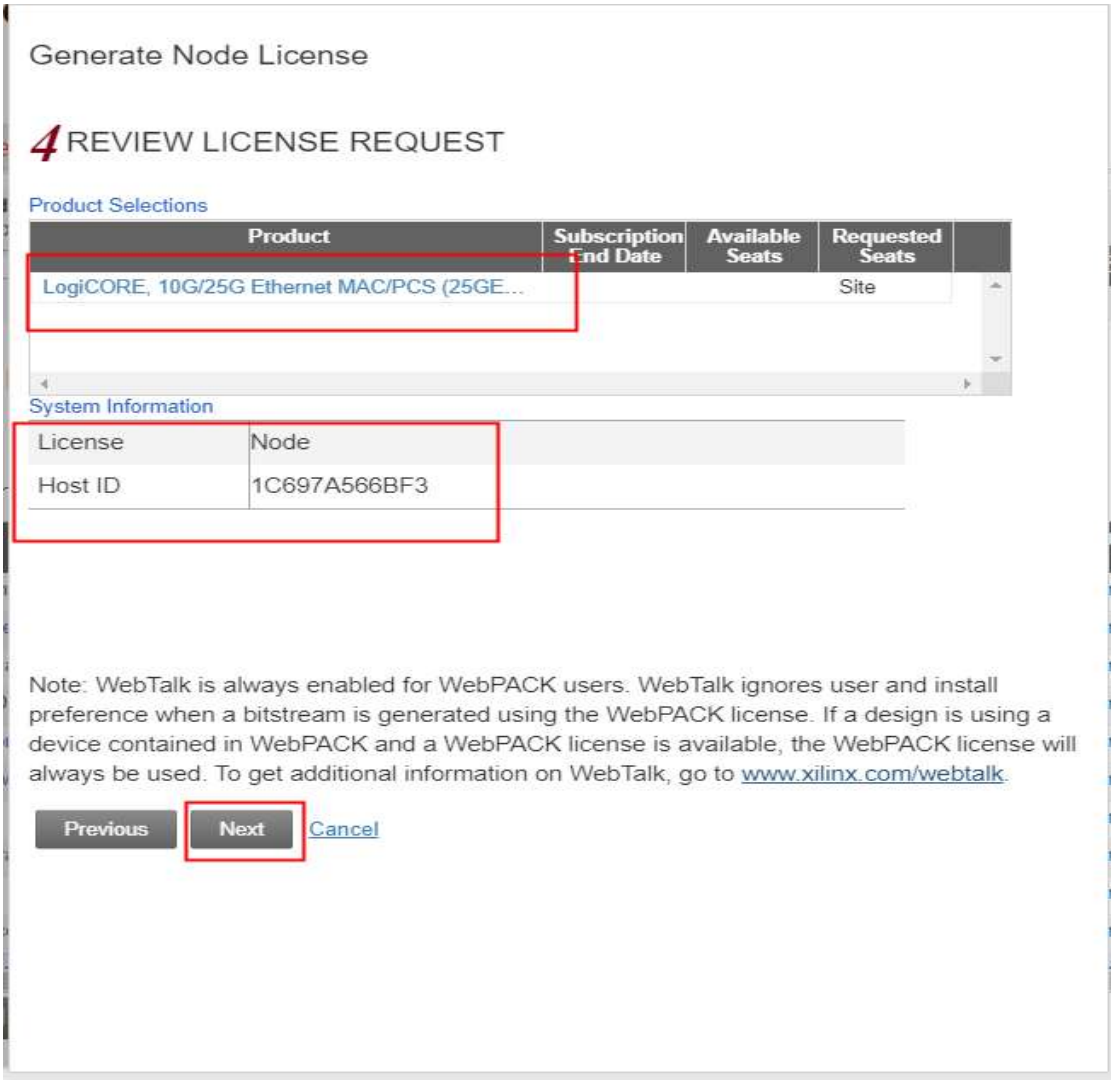


Figure 2-31 Confirm the license generation

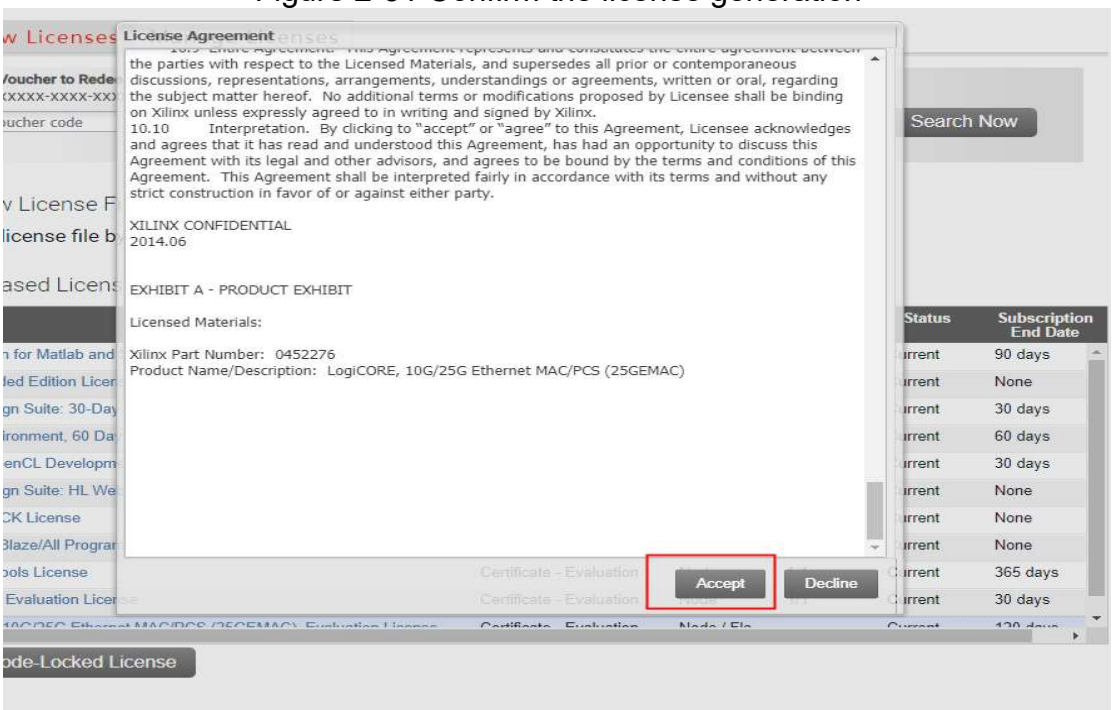


Figure 2-32 Agreement with xilinx license

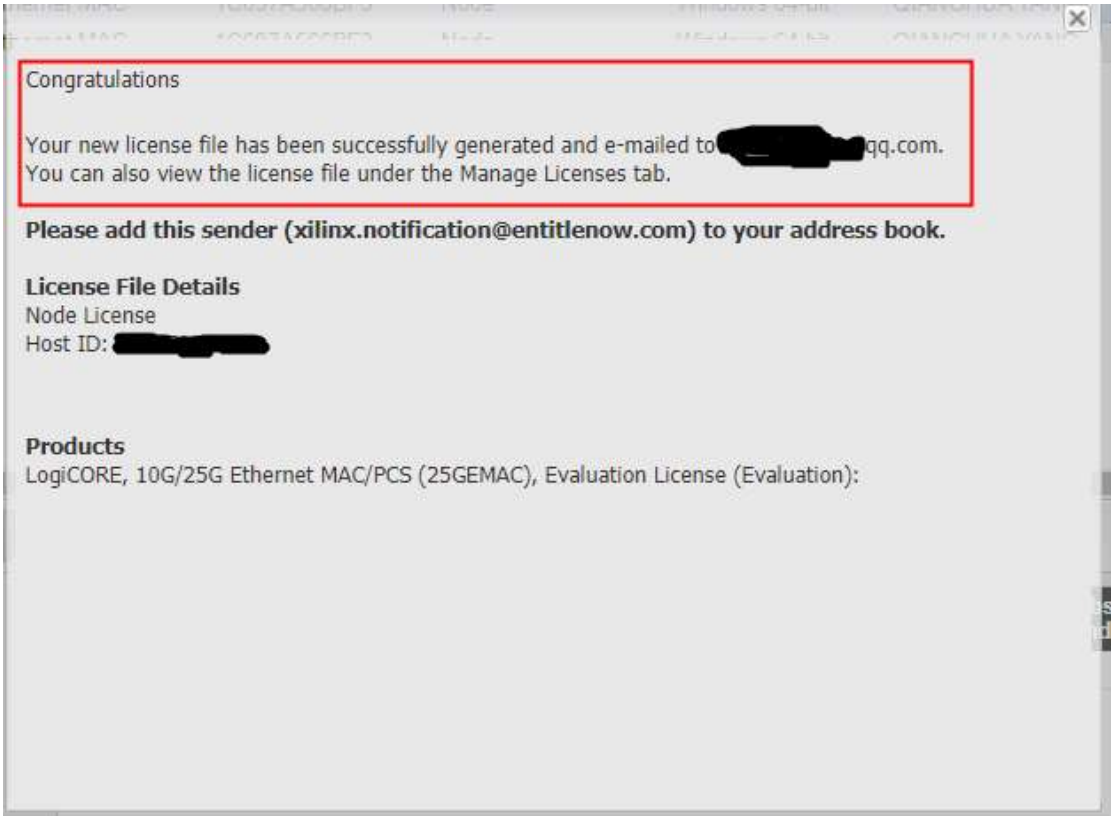


Figure 2-33 after send license to you mail

So far, you can receive the corresponding Xilinx.lic in the registered mailbox

Figure 2-34 Mail received license attachment

Download the received Xilinx.lic, and then re-register the license with the Vivado License manager (repeat the operation in section 2.2.3), as shown in the figure

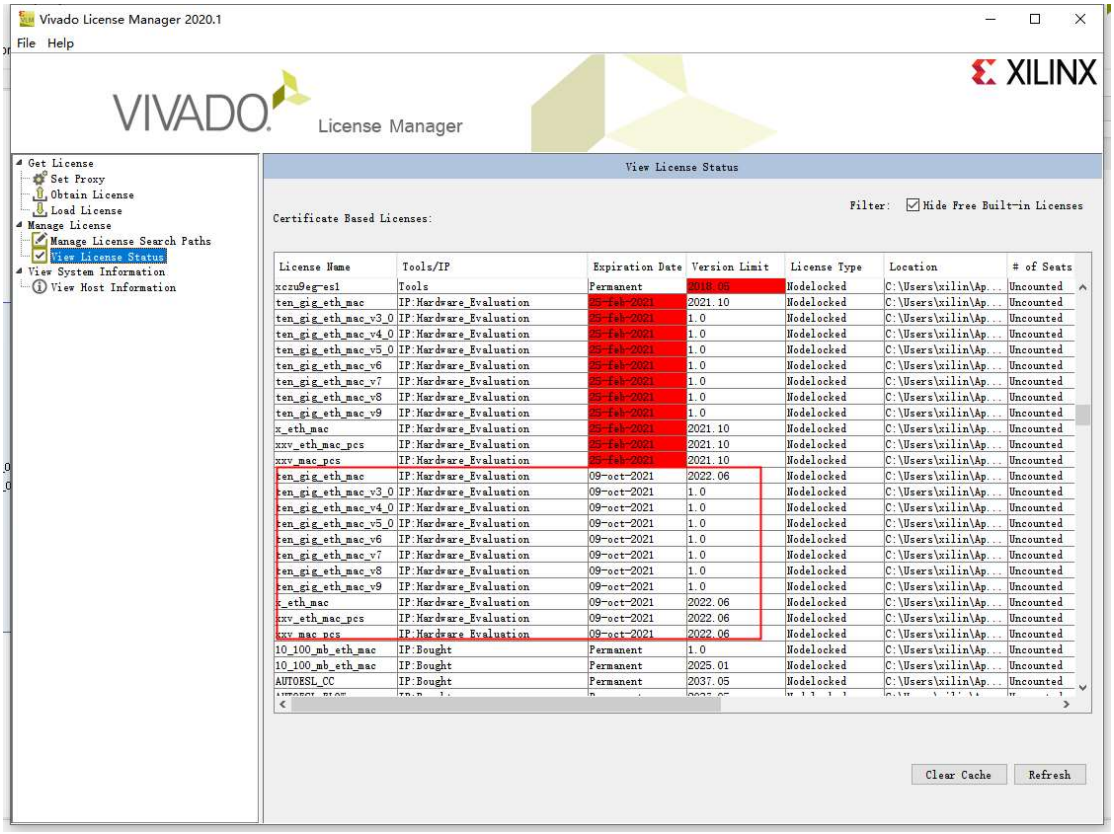


Figure 2-35 Installation of license

At this point, the corresponding IP status has become Hardware_Evaluation IP License available

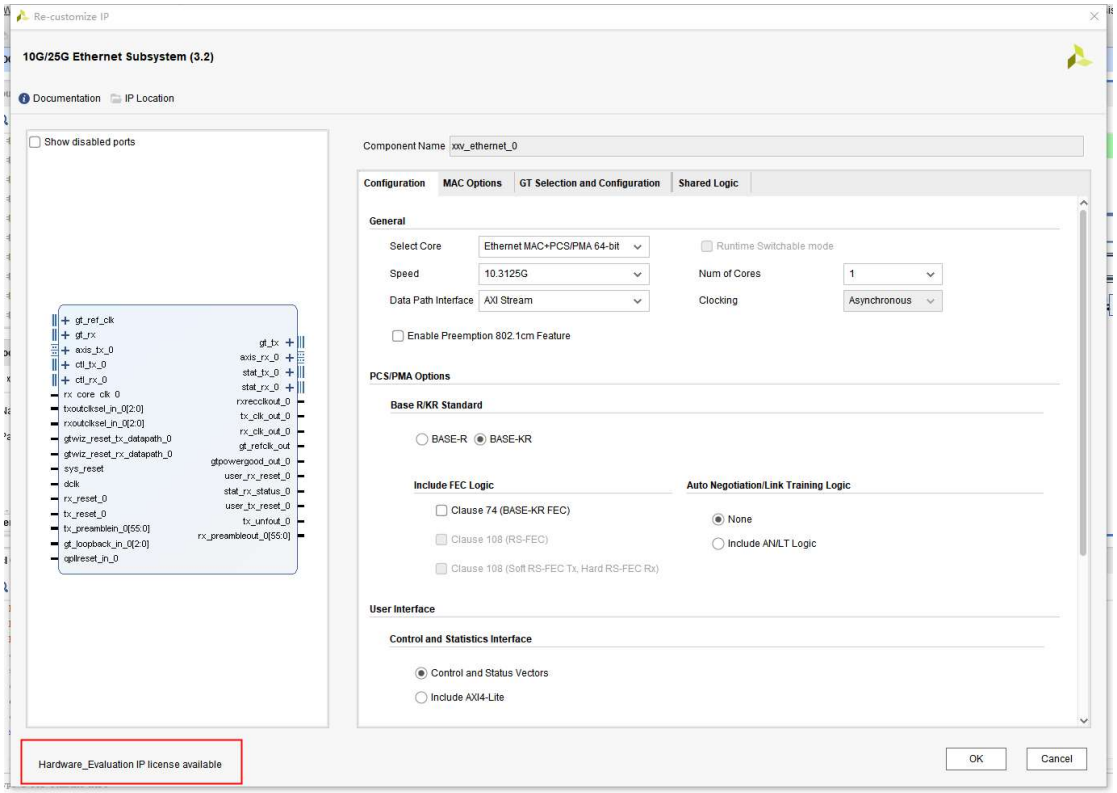


Figure 2-36 Installation license of state IP core

In the next 3 month,you can use the IP of 10G/25G Ethernet MAC/PCS (25GEMAC) Evaluation。

2.2.5 Modelsim Download and Installation

modelsim se 2019 is the latest version of HDL language simulation softwarethat has been improved and optimized on the basis of the original version of the software functions and performance, making its software functionality more complete. The 2019 new version provides comprehensive and high-performance verification functions, fully supporting the industry's wide range of standards; in addition, compared with the old version, the simulation speed is 10 times faster, and the graphical user interface is powerful, all windows will be automatically updated in any other windows activity. For example, selecting the design area in the Structure window will automatically update the Source, Signals, Process and Variables windows. You can edit, recompile and re-simulate without leaving the software environment. All user interface operations can be scripted, and the simulation can be run in batch or interactive mode. It is the preferred simulation software for FPGA/ASIC design.

Modelsim is a simulation tool of Mentor Graphics Corporation. The method of downloading here can be found on the eetop website to find the corresponding installation package, or download it from Mentor Graphics' official website.

2.2.6 Modelsim Installation

1,Unzip the compressed package of the software installation package to get the installation program。

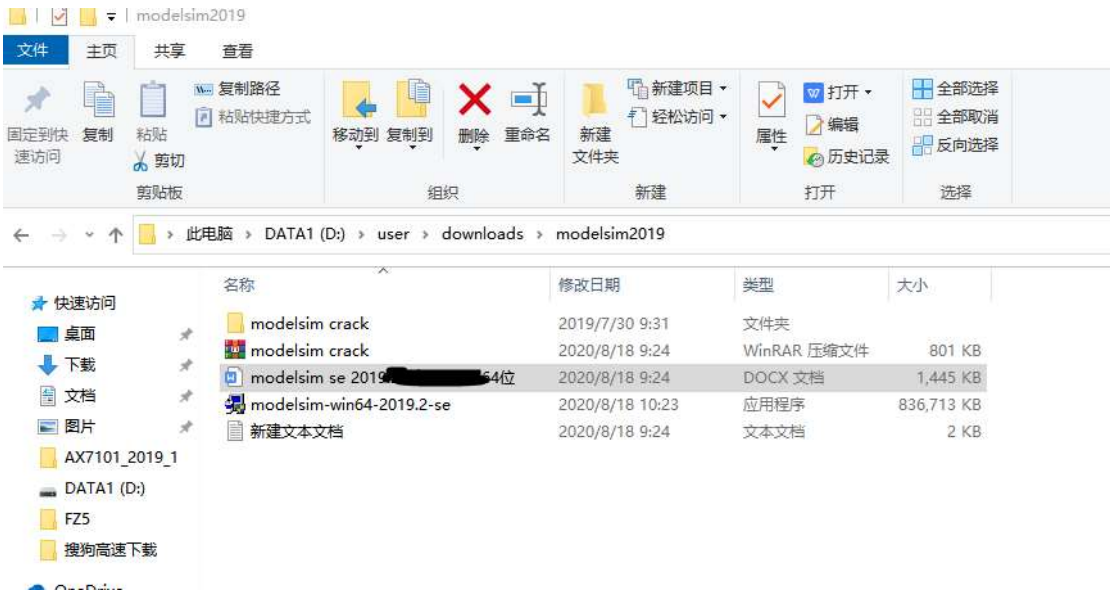


Figure 2-37 modelsim installation files

2,Double click the suffix .exe files to install，go to the installation wizard and click next。

Figure 2-38 modelsim installation wizard

3,Select the software installation path, click Browse to change the path, or set the installation path according to the default.

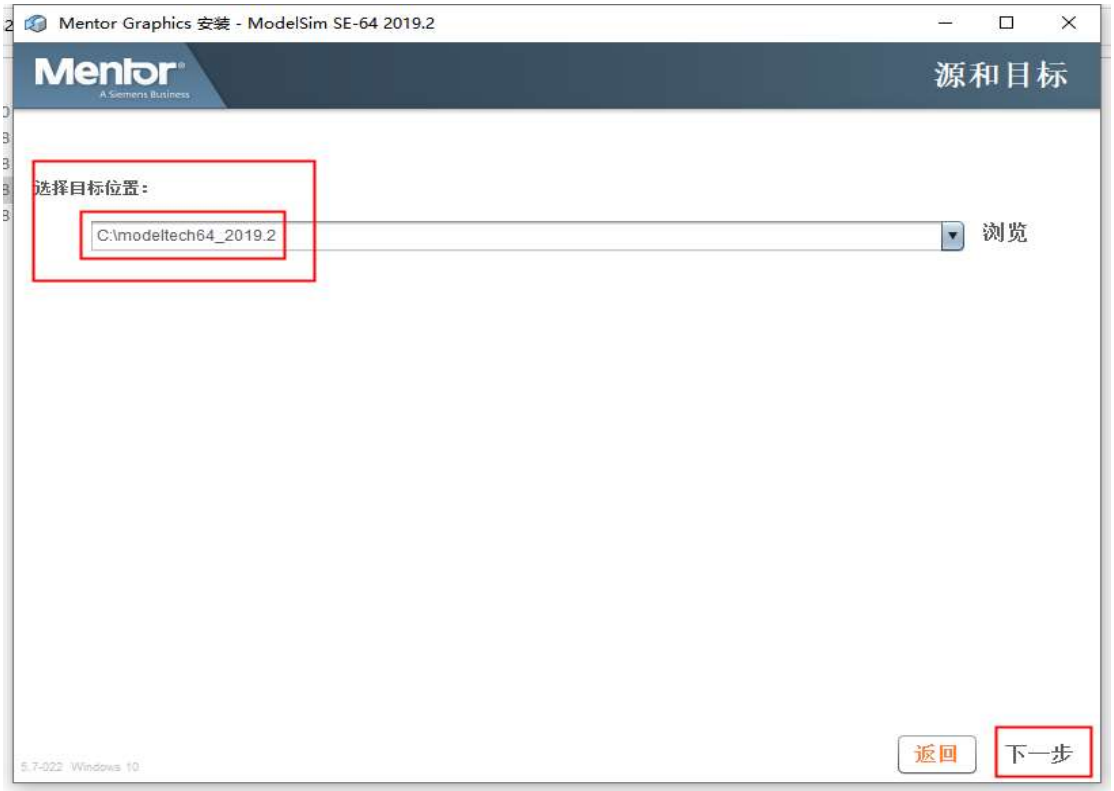


Figure 2-39 modelsim installation location

4,Agreement with the license

Figure 2-40 modelsim license agreement

5,The software enters the installation state and is being installed. The installation process will take some time, just wait

Figure 2-41 modelsim installation progress

6,click NO

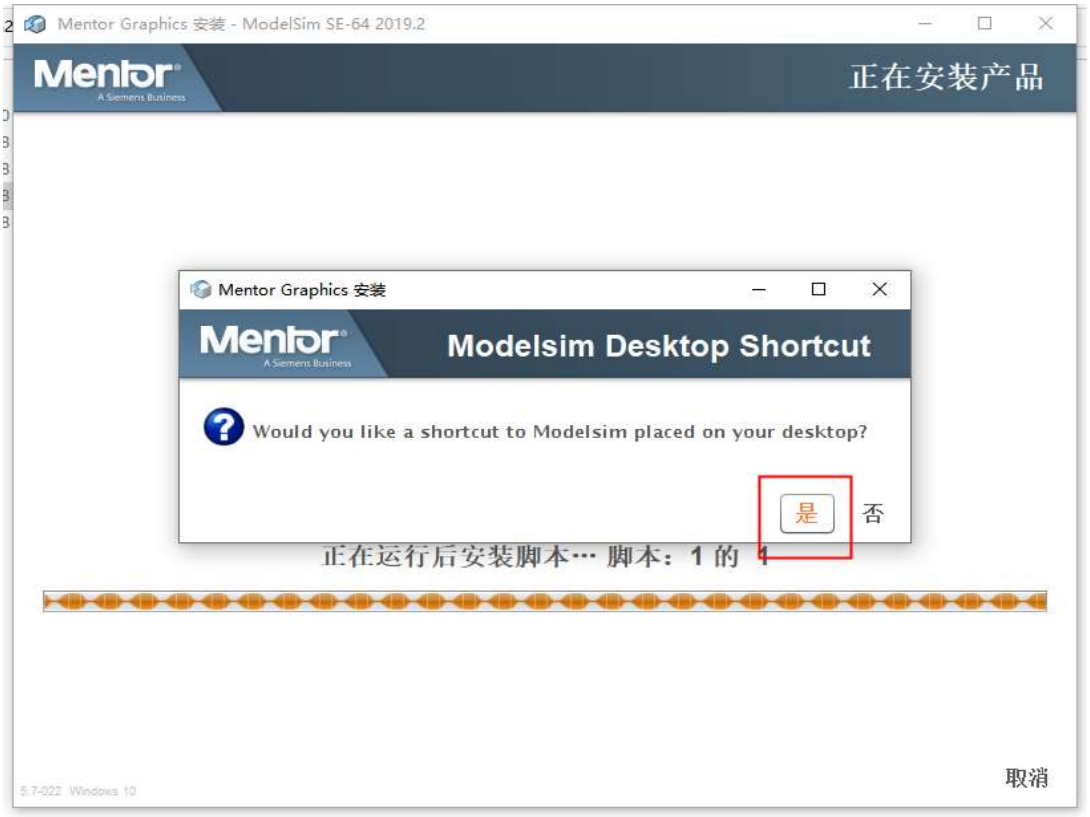


Figure 2-42 modelsim installation

7,In the last step to install the key driver of the hardware, select No, do not install. This completes the installation of modelsim。

2.2.7 Modelsim Project Establishment and Simulation

1，built new modelsim project，and make a location for the project。

2，open modelsim software，file->new->project

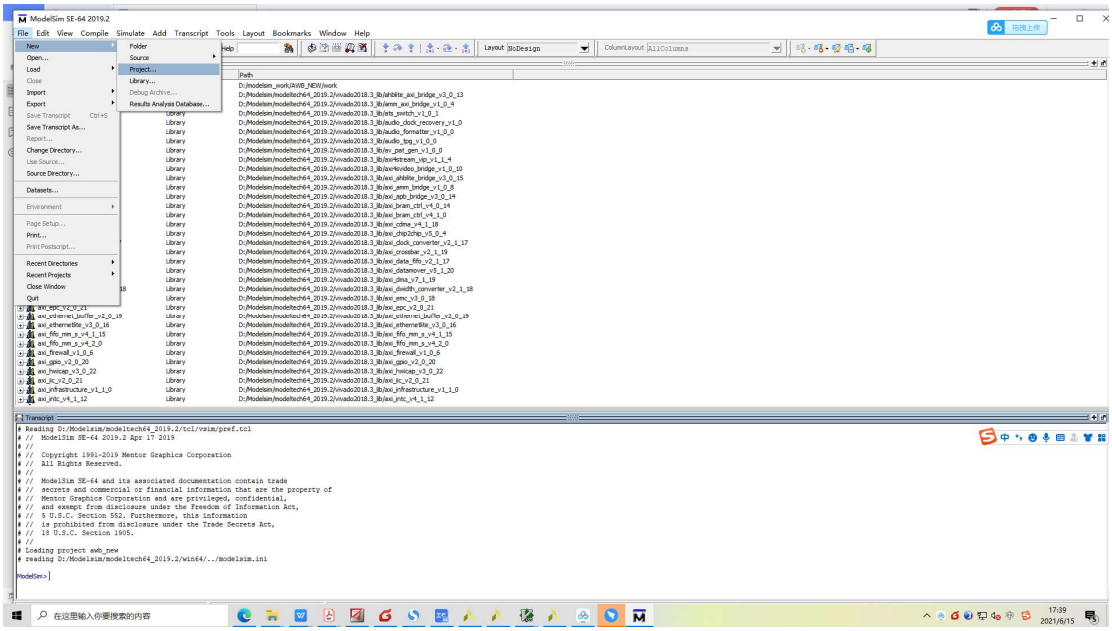


Figure 2-43 New modelsim project

3,fill the project name and location

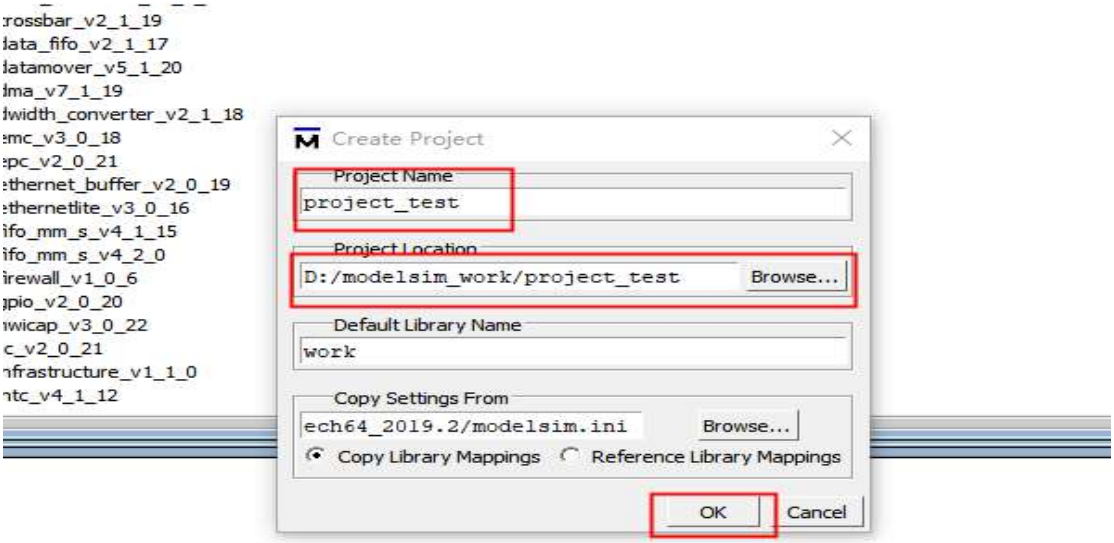


Figure 2-44 modelsim new project

4,Add existing files -> then chose RTL design files

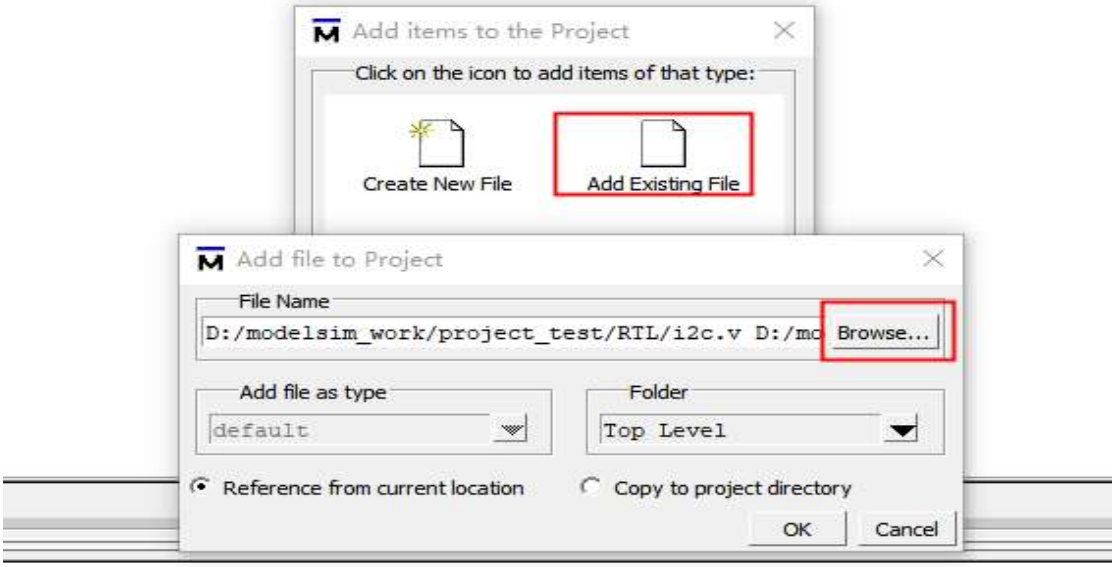


Figure 2-45 project to add existing files

5,compile-> compile ALL

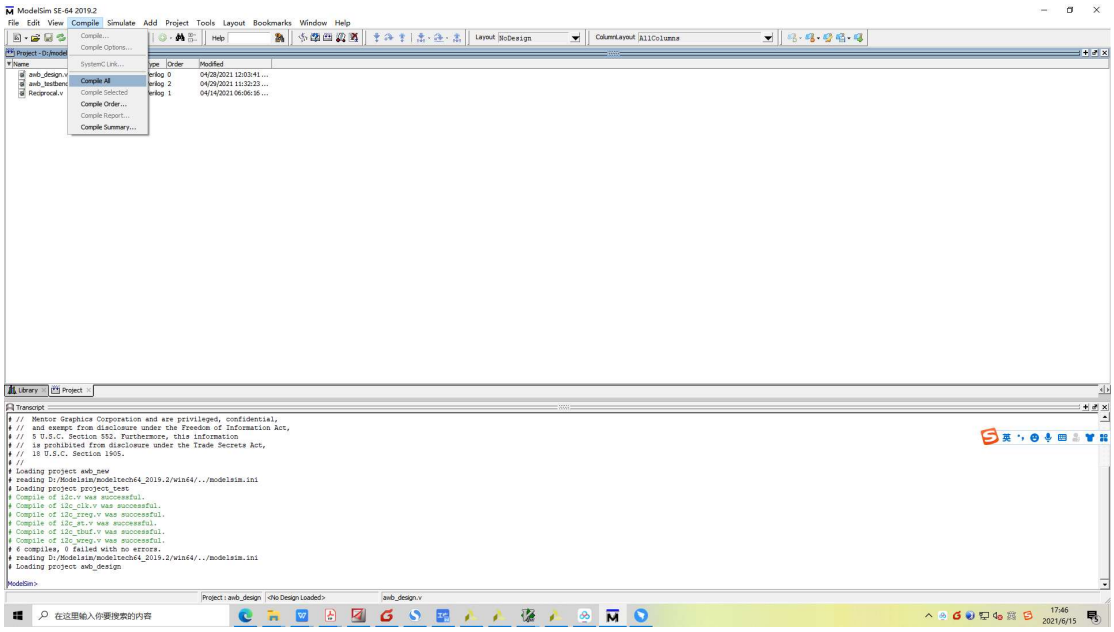


Figure 2-46 modelsim compile

6,Back to library window

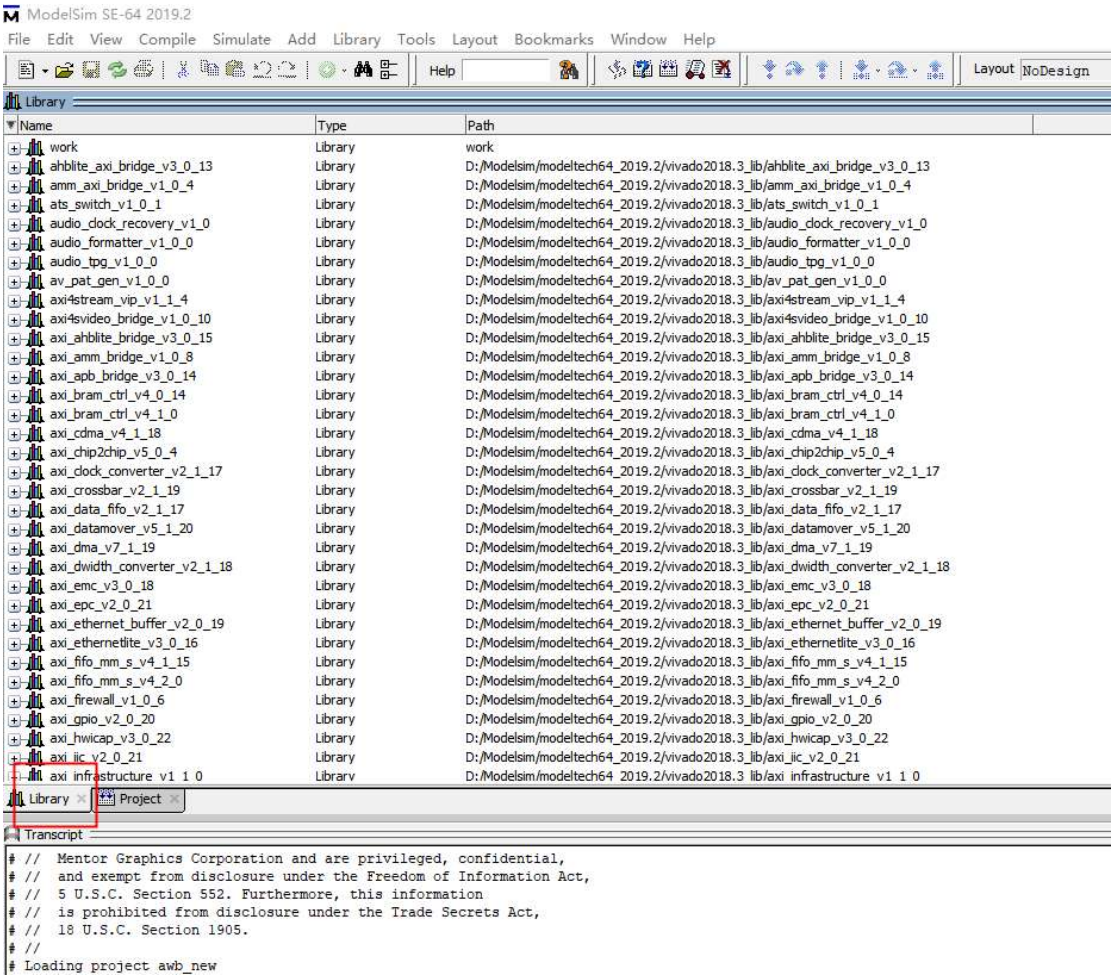


Figure 2-47modelsim library

7,Open work library-> “Testbench ” , and simulation :

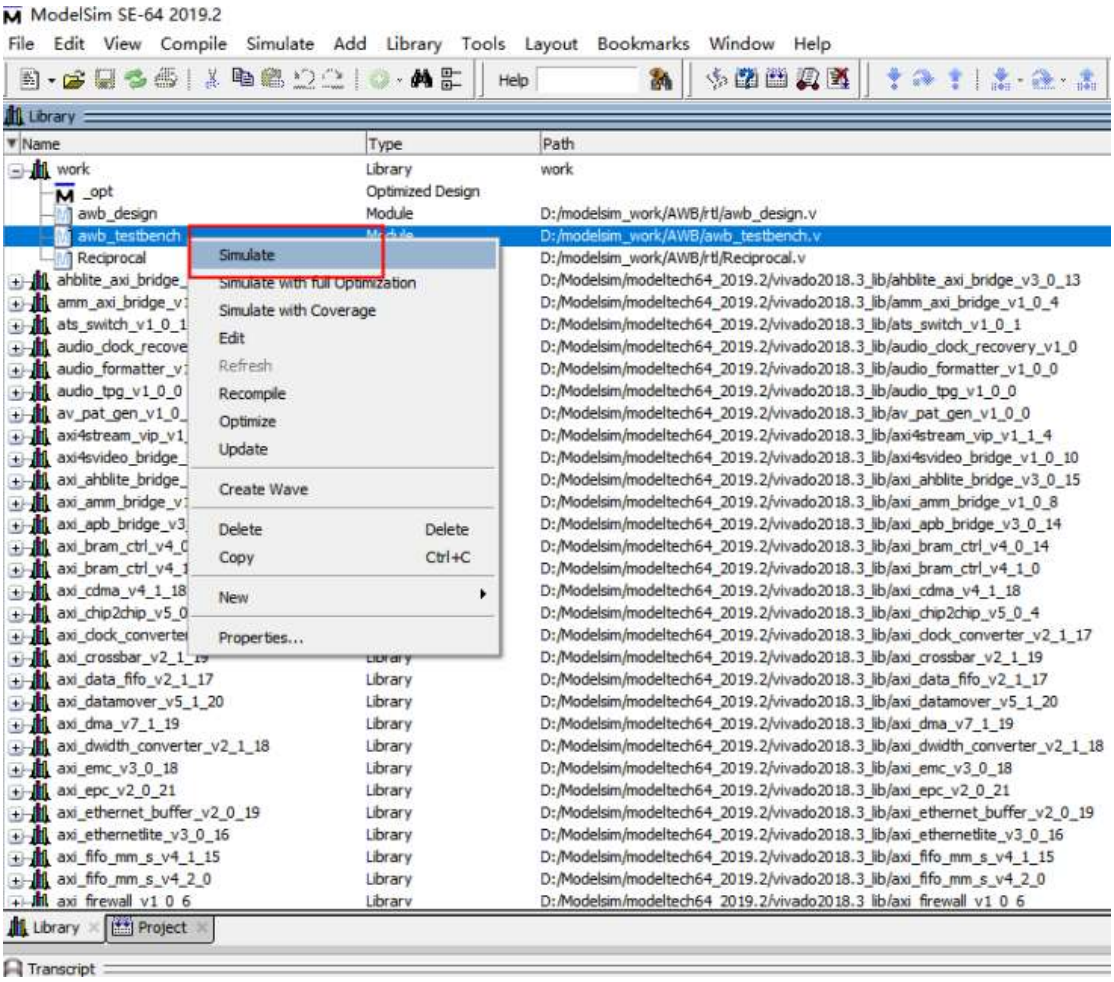


Figure 2-48 modelsim simulation begin

Add the signal to be observed in the sim window, and then you can see the simulated waveform window. If there is an error, you can modify it according to the prompt.

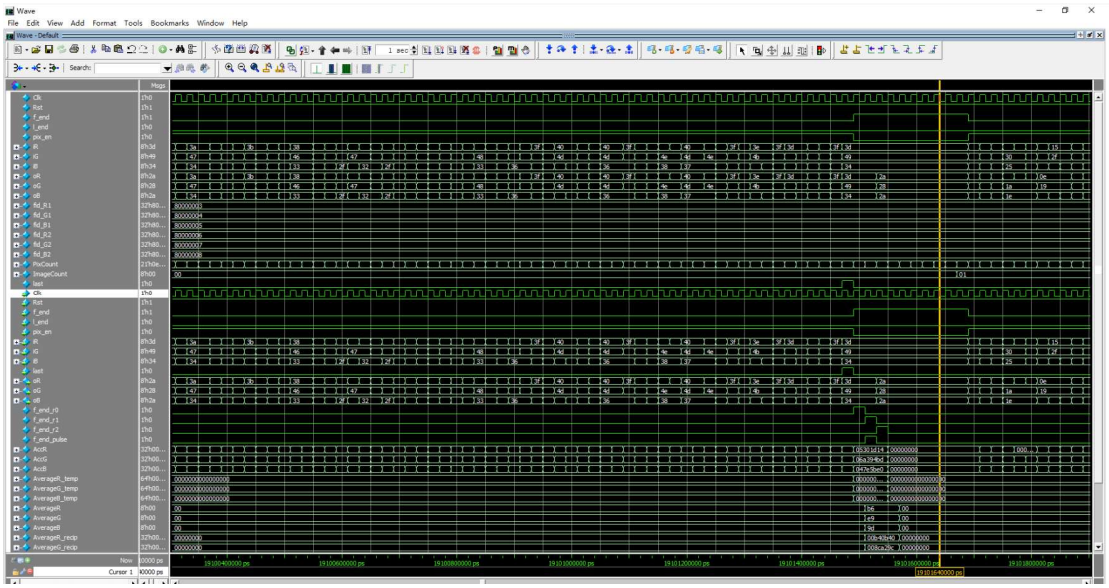


Figure 2-49 modelsim simulation waveform

2.3 Knowledge Preparation

2.3.1 Verilog Brief Introduction to Grammar

Verilog HDL is a hardware description language that describes the structure and behavior of digital system hardware in text form. It can be used to represent logic circuit diagrams, logic expressions, and logic functions completed by digital logic systems. Verilog HDL and VHDL are the two most popular hardware description languages in the world, both of which were developed in the mid-1980s. The former was developed by Gateway Design Automation (the company was acquired by Cadence in 1989). Both HDL are IEEE standards. Here are some synthesizable grammatical knowledge of Verilog:

1，Constant：

Constants can exist in Verilog documents, which will be converted to constant registers corresponding to logic circuits or ignored directly during the pretreatment stage. For example, integer, parameter DLY=1; here is to define a constant DLY, integer can be represented by binary b or B, octal o or O, decimal d or D, hexadecimal h or H, for example, 8'b00001111 means 8 bits A bit-wide binary integer, 4'ha represents a 4-bit wide hexadecimal integer. For example, underscore: assign a = 8’ b0000_1111; The underscore here is for easy reading. For example, X, or Z: X indicates that the variable is in an uncertain state, a = 8'bxxx0_11xx; here it means that the 0, 1, 5, 6, 7 bits of the variable a[7:0] are uncertain values, that is, In the actual result, there must be a result, but we cannot logically determine whether the exact value of this result is 0 or 1. On the whole, the value of a is in an uncertain state. If a = 8'bzzz0_11zz, here it means that the 0,1,5,6,7 bits of the variable a[7:0] are in the high-impedance state. As the name implies, the resistance is large enough to reach an open circuit, which is equivalent to floating. The value depends on the logic circuit that follows.

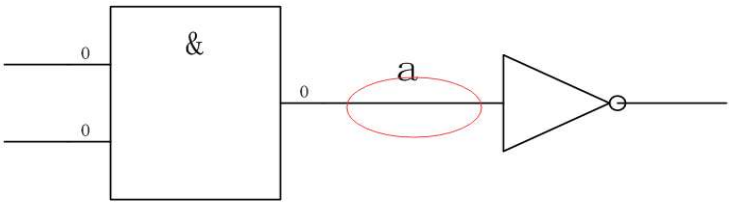
2，Variable

Refers to the amount of value that can be changed during operation. The amount of value that can be changed in Verilog is a variable of the register type, such as the reg type. The value of the line connecting a colleague and reg is also changed. This type is a wire type variable. There is also a whole block of data changes, which can be used to store data, such as RAM, ROM, etc., as well as

frequently used DDR, SDRAM, FLASH, and so on. These devices can all be instantiated in verilog's grammar, and can naturally store changed data. Here we introduce wire and reg.

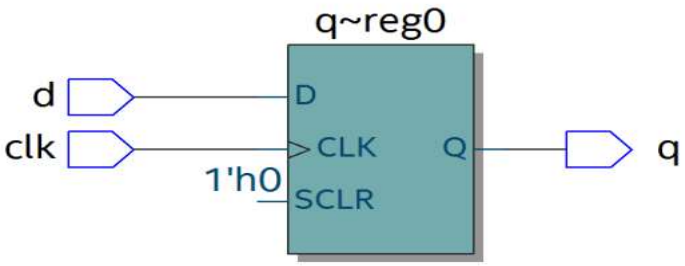
3，Wire：

Wire refers to a linear variable. Use wire to define a variable. Wire [7:0] a; In the process of synthesis, there is an 8-bit wide connection. As for what is connected at both ends of the connection, it needs to be specific It will be known in the design. For example, assign a = 8'b0000_1111; here is to link a to the output terminal of the 8bit constant register, so that the value on the entire line has changed, and it is always 8'b0000_1111. For example, link a to the output terminal of the register, Here the register needs to have a bit width of 8bit, and only the assign statement can be used in Verilog. Of course, the bit width assigned to a is mainly 8bit. as the picture shows:



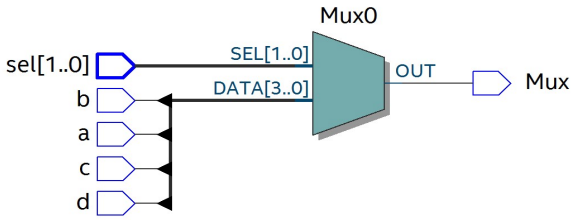
4，Reg：

This is called a register variable in verilog. In the circuit, a register is used to represent the corresponding variable. The bit width and number of variables are consistent with the number of registers implemented in the circuit. Obviously, reg type variables can store us The required data, its value can also be changed with the clock of the register, the value in the register can be retained, changed, etc. The change of the register value in the verilog syntax must be carried out in the always statement. There are two usages here, divided into Combinational logic circuit and sequential logic circuit; the register q is defined, the generated circuit is sequential logic, and the structure on the right is the D flip-flop.



```
module top(d, clk,
q) ;
input d ;
input clk ;
output reg q ;
always @(posedge clk)
begin
q <= d ;
end
endmodule
```

Combinational logic can also be generated, such as data selector, sensitive signals have no clock, reg Mux is defined, and the final generating circuit is combinational logic.




```
module top(a, b, c, d, sel,
Mux) ;
input a ;
input b ;
input c ;
input d ;
input [1:0] sel ;
output reg Mux ;
always @(sel or a or b or c or
d)
begin
case(sel)
2'b00 : Mux = a ;
2'b01 : Mux = b ;
2'b10 : Mux = c ;
2'b11 : Mux = d ;
endcase
end
endmodule
```

Memory type: We can use reg [15:0] a [1023:0]; to define a data block, there are 1024 data, and the bit width is 16bit. It can be instantiated in RAM, occupying RAM resources, and can be instantiated into ROM, FLASH, etc. memory in ASIC design。

Operator: Operators include: arithmetic operators, logical operators, bitwise operators, reduction operators, relational operators, shift operators, equality operators, conditional operators, and concatenation operators.

Arithmetic operators: Commonly used arithmetic operators include: + addition,-subtraction, * multiplication, / division,% modulo (remainder)

The above arithmetic operators are binocular operators.

Logical Operators: && Logical AND, || Logical OR,! Logical negation Bitwise operators: ~ Bitwise negation, & bitwise AND, | bitwise OR, ^ bitwise XOR, ^~, ~^ bitwise XOR Reduction operator: Abbreviated operators are unary operators, including the following: & And, ~& and or, | or, ~| NOR, ^ XOR, ^~, ~^ same or E.g:

```
reg [3:0] a;
b = & a; // equal to    b = ( ( ( a[0] & a[1] ) & a[2] ) & a[3] )
```

Relational operators:

<Less than, <= less than or equal to,> greater than, >= greater than or equal to Shift operator: <<, shift left. >>, move right Equality operator: == equals, != is not equal, === is equal, !== is not equal Equality operator (==):

The two operands involved in the comparison must be equal bit by bit, and the result of the equality comparison is 1. If some bits are in the indeterminate state or high impedance state, the result of the equality comparison is indefinite value;

Congruence operator (===): In the indeterminate or high-impedance state, the two operands must be exactly the same, and the result is 1.

Conditional operator ?:

Concatenation operator: {}

For example: {3{a,b}} is equivalent to {{a,b}, {a,b}, {a,b} }, and also equivalent to {a, b, a, b, a, b }.

The precedence levels of various operators are as follows:

{} {{}}	Concatenation, replication
unary + unary -	Unary operators
+ - * / **	Arithmetic
%	Modulus
> >= < <=	Relational
!	Logical negation
&&	Logical and
||	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or
^~ or ~^	Bitwise equivalence
&	Reduction and
~&	Reduction nand
|	Reduction or
~|	Reduction nor
^	Reduction xor
~^ or ^~	Reduction xnor
<<	Logical left shift
>>	Logical right shift
<<<	Arithmetic left shift
>>>	Arithmetic right shift
? :	Conditional

Figure 2-50 Operator Priority

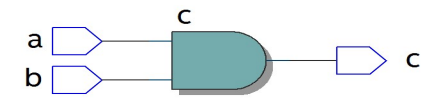
5，Combinational logical：

The characteristic of the combinational logic circuit is that the output at any

time depends only on the input signal, the input signal changes, the output changes immediately, and does not depend on the clock.

AND：

In verilog, "&" means bitwise AND, such as c=a&b, the truth table is as follows, when both a and b are equal to 1, the result is 1, RTL means as shown in the figure



输入		输出
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

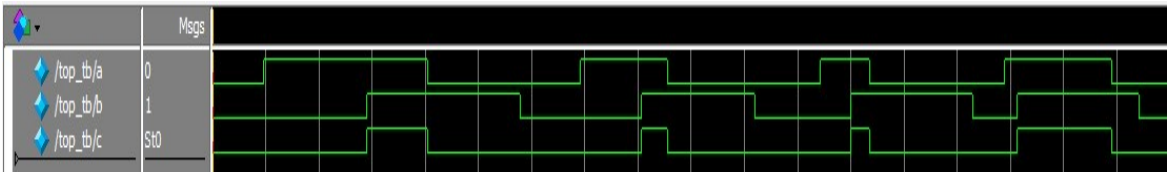
Code ：

```
module top(a, b,
c) ;
input a ;
input b ;
output c ;
assign c = a &b ;
endmodule
```

Test fot excitation：

```
`timescale 1 ns/1 ns
module top_tb() ;
reg a ;
reg b ;
wire c ;
initial
begin
a = 0 ;
b = 0 ;
forever
begin
#({$random}%100)
a = ~a ;
#({$random}%100)
b = ~b ;
end
end
top
t0(.a(a), .b(b),.c(c)) ;
endmodule
```

The simulation results are as follows ：

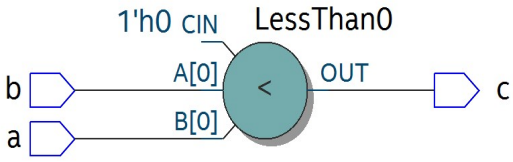


If the bit width of a and b is greater than 1, for example, define input [3:0] a, input [3:0]b, then a&b then The corresponding phases of a and b are AND. Such as a[0]&b[0], a[1]&b[1]. Other combinatorial logic similar processes can be learned.

Comparators：

In verilog, it is expressed as greater than ">", equal to "==", less than "<", greater than or equal to ">=", less than or equal to "<=", not equal to "!=". Take greater than for example, such as c= a > b; means that if a is greater than b, then the value of c is 1, otherwise it is 0. The truth table is as follows:

输入		输出
a	b	c
0	0	0
0	1	0
1	0	1
1	1	0



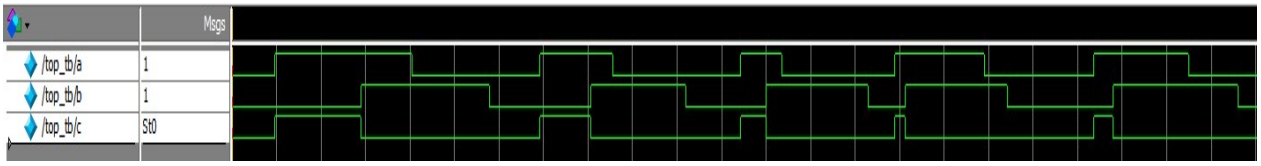
code：

```
module top(a, b, c) ;
input a ;
input b ;
output c ;
assign c = a >b ;
endmodule
```

Test fot excitation：

```
`timescale 1 ns/1
ns
module top_tb() ;
reg a ;
reg b ;
wire c ;
initial
begin
 a = 0 ;
 b = 0 ;
 forever
 begin
 #({$random}%100)
 a = ~a ;
 #({$random}%100)
 b = ~b ;
 end
 end
 top
 t0(.a(a), .b(b),.c
 endmodule
```

The simulation results are as follows ：



6，Timing logical：

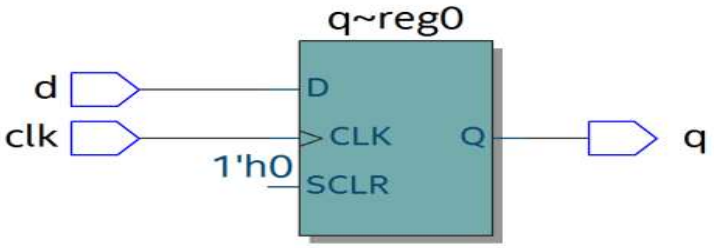
The characteristic of the logic function of the combinational logic circuit is that the output at any time depends only on the input at the current time, and has nothing to do with the original state of the circuit. The characteristic of sequential logic in logic function is that the output at any moment not only depends on the current input signal, but also depends on the original state of the circuit. The following is a typical sequential logic analysis.

D flip-flop：

D flip-flop stores data on the rising or falling edge of the clock, and the output is the same as the state of the input signal before the clock jumps. Verilog describes the D flip-flop ：

```
module top(d, clk,
q) ;
input d ;
input clk ;
output reg q ;
always @(posedge clk)
 begin
 q <= d ;
 end
 endmodule
```

RTL synthesis：

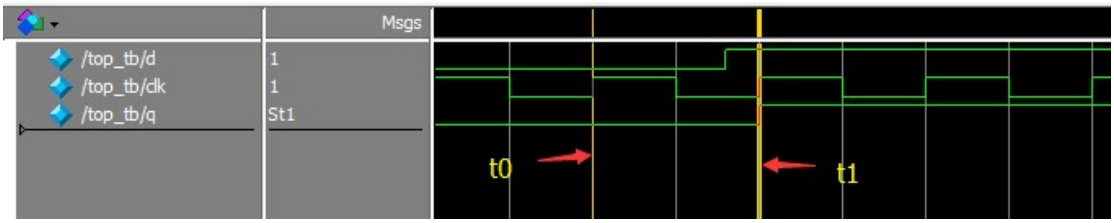


The simulation stimulus file is as follows：

```
`timescale 1 ns/1 ns
module top_tb() ;
reg d ;
reg clk ;
wire q ;
```

```
initial
begin
d = 0 ;
clk = 0 ;
forever
begin
#({$random}%100)
d = ~d ;
end
end
always #10 clk = ~clk ;
top
t0(.d(d),.clk(clk),.q(q)) ;
endmodule
```

The simulation results are as follows. It can be seen that at time t0, the value of d is 0, and the value of q is also 0; at time t1, when d changes and the value is 1, then q correspondingly changes and the value becomes 1. . It can be seen that in a clock cycle between t0-t1, no matter how the value of the input signal d changes, the value of q remains unchanged, that is, there is a storage function, and the saved value is on the transition edge of the clock. The value of d at time.



Dual port RAM：

Dual-port RAM is divided into true dual-port RAM and pseudo-dual-port RAM. The read-write address of pseudo-dual-port RAM is independent, and the write or read address can be randomly selected to perform read and write operations at the same time. True dual-port RAM has two sets of control lines and data lines, allowing two systems to read and write to it. Here is the introduction of true dual-port RAM

Verilog description of the dual port RAM：

```
module top
(
input [7:0] data_a, data_b,
input [5:0] addr_a, addr_b,
input wr_a, wr_b,
input rd_a, rd_b,
input clk,
output reg [7:0] q_a, q_b
);
reg [7:0] ram[63:0]; //declare ram
//Port A
always @ (posedge clk)
begin
if (wr_a) //write
begin
```

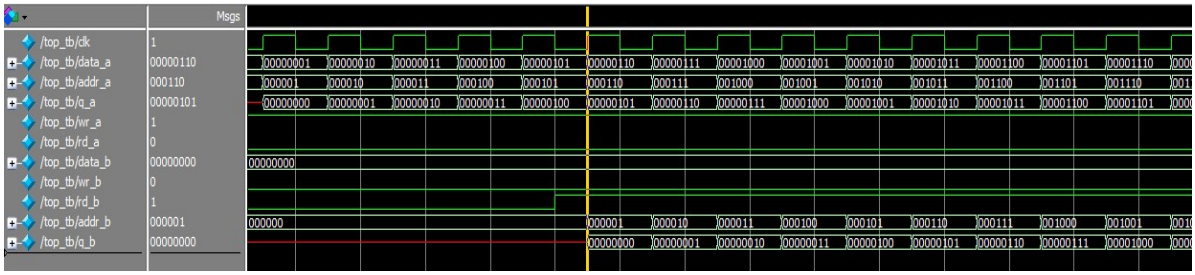
```
ram[addr_a] <= data_a;
q_a <= data_a ;
end
if (rd_a)
//read
q_a <= ram[addr_a];
end
//Port B
always @ (posedge clk)
begin
 if (wr_b) //write
 begin
 ram[addr_b] <= data_b;
 q_b <= data_b ;
 end
 if (rd_b)
 //read
 q_b <= ram[addr_b];
 end
 endmodule
```

RTL simulation stimulus file is as follows :

```
`timescale 1 ns/1 ns
module top_tb() ;
reg [7:0] data_a, data_b ;
reg [5:0] addr_a, addr_b ;
reg wr_a, wr_b ;
reg rd_a, rd_b ;
reg clk ;
wire [7:0] q_a, q_b ;
initial
begin
data_a = 0 ;
data_b = 0 ;
addr_a = 0 ;
addr_b = 0 ;
wr_a = 0 ;
wr_b = 0 ;
rd_a = 0 ;
rd_b = 0 ;
clk = 0 ;
#100 wr_a = 1 ;
#100 rd_b = 1 ;
end
always #10 clk = ~clk ;
always @(posedge clk)
begin
if (wr_a)
```

```
begin
data_a <= data_a + 1'b1 ;
addr_a <= addr_a + 1'b1 ;
end
else
begin
data_a <= 0 ;
addr_a <= 0 ;
end
end
always @(posedge clk)
begin
if (rd_b)
begin
addr_b <= addr_b + 1'b1 ;
end
else addr_b <= 0 ;
end
top
t0(.data_a(data_a), .data_b(data_b),
.addr_a(addr_a), .addr_b(addr_b
),
.wr_a(wr_a), .wr_b(wr_b),
.rd_a(rd_a), .rd_b(rd_b),
.clk(clk),
.q_a(q_a), .q_b(q_b)) ;
endmodule
```

Simulation results：



Finite State Machine：

The finite state machine (Finite-State Machine, FSM), has become a finite state automachine, abbreviated as a state machine, which is a mathematical model that represents a finite number of states and the behaviors such as transitions and actions between these states. I often apply FSM in application scenarios such as motor control and communication protocol analysis. What this article is talking about is the writing skills and specifications of the finite state machine based on the hardware description language Verilog HDL. As we all know, FPGA is known to the world for its parallelism and reconfigurability. In today’ s electronic world, basically all devices are serial, so FPGA as a control unit or a programmable unit needs to be converted from parallel to serial. It communicates and controls with the outside world, and the finite state machine plays this role with its simplicity, practicality, clear structure and appropriateness.

A finite state machine is a hardware sequential circuit composed of a register group and a combinational logic. Its state (that is, a finite number of states composed of the combined state of 1 and 0 of the register group) can only be changed from the same clock transition edge. When one state turns to another state, which state to turn to or stay in the original state not only depends on each input value, but also depends on the current state.

There are two types of state machines: Mealy and Moore. The output of the Moore-type state machine is only related to the current state, while the output of the Mealy-type state machine not only depends on the current state, but is also directly controlled by the input, and may have nothing to do with the state. The structure of the state machine is shown in the figure below.:

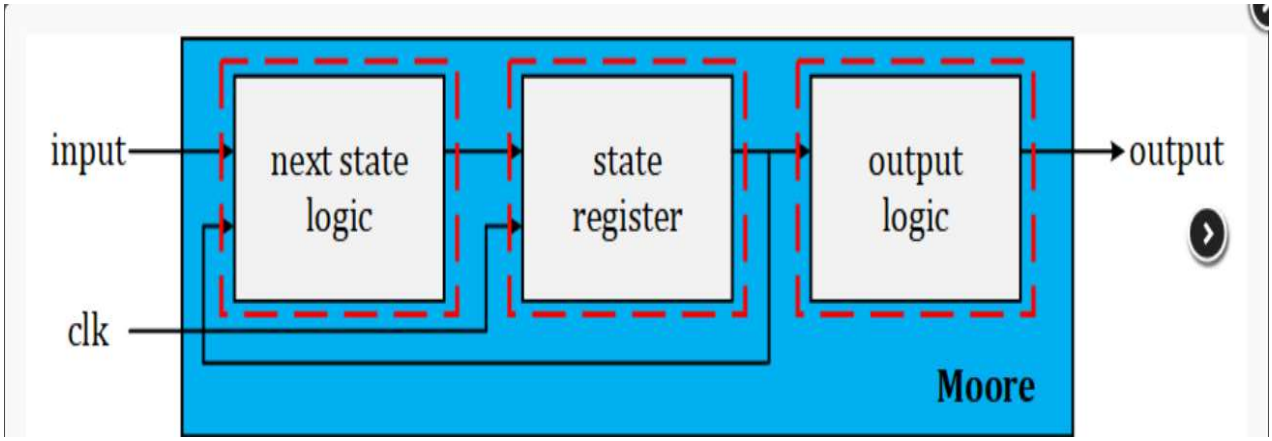


Figure 2-51 Moore finite state machine

The essence of the state machine is a method of describing events with logical sequence or timing law. The two most important words in this argument are "logical order" and "chronological logic". These two points are the core and strengths of the state machine to describe. In other words, all things with logical order and timing laws are suitable for description by the state machine.

Two application ideas of state machine. The first idea is to start with state variables. If a circuit has sequential logic or logical sequence, we can naturally plan out the states, start with these states, analyze the input, state transition and output of each state, and complete the circuit function; The second idea is to first clarify the output relationship of the circuit, these outputs are equivalent to the output of the state, backtracking plan each state, and state transition conditions and state inputs.

The two ways of thinking have the same goal in different ways. They must use the state machine to achieve the purpose of controlling a certain part of the

circuit, and complete a certain circuit design with logic sequence or timing law. The parameter definition of the state machine adopts the one-hot code.

Compared with the Gray code, although the one-hot code uses more triggers, the combined circuit used can save some, so the speed and reliability of the circuit are significantly improved. The number of units did not increase significantly. After adopting one-hot encoding, there are redundant states and some unreachable states. For this reason, the default branch direction needs to be added at the end of the case statement. This can be represented by a default item or a definite item to ensure that it returns to the initial state.

Generally, the synthesizer can handle the default items reasonably through the control of the synthesis instruction.

The function of the vending machine is described as follows :

The unit price of drinks is 2 yuan, and the vending machine can only accept 0.5 yuan and 1 yuan coins. Consider changing and shipping. The coin insertion and shipment processes are carried out one at a time, and there will be no such thing as a one-time investment of multiple coins or one-time shipment of multiple bottles of beverages. After each round of vending machines accept coin insertion, shipment, and change, it can enter the new automatic selling state. The working state transition diagram of the vending machine is shown below, including the input and output signal states. Among them, coin = 1 means 0.5 yuan coin is invested, and coin = 2 means 1 yuan coin is invested.

State machine design: 3-stage (recommended)

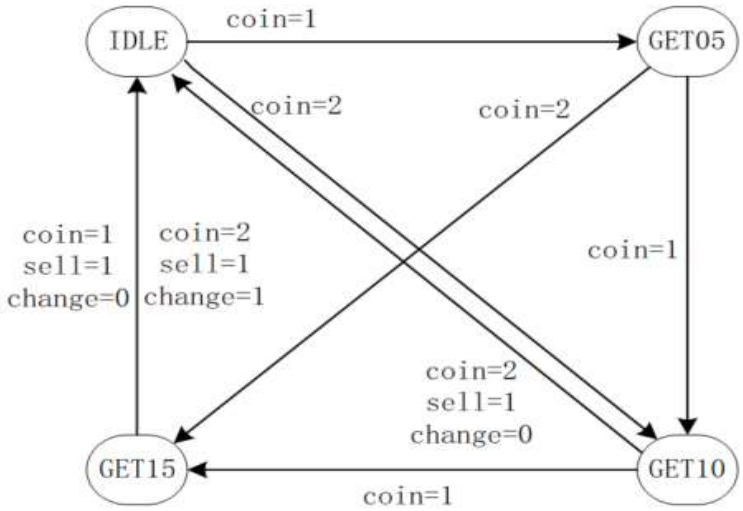
The state machine design is as follows:

First, determine the state machine code according to the number of state machines. Using codes to assign values to the status register makes the code more readable.

The first stage of the state machine, sequential logic, non-blocking assignment, transfer the state of the register.

The second stage of the state machine, combinatorial logic, block assignment, and determine the state of the next state machine based on the current state and current input.

The third generation of state machine, sequential logic, non-blocking assignment, because it is a Mealy type state machine, it determines the output signal according to the current state and current input.



Verilog RTL description：

```
module    vending_machine_p3
      (
            input                  clk ,
            input                  rstn ,
            input [1:0]          coin ,          //01 for 0.5 jiao, 10 for 1 yuan
            output [1:0]         change ,
            output                 sell        //output the drink
        );
    //machine state decode
    parameter                    IDLE     = 3'd0 ;
    parameter                    GET05    = 3'd1 ;
    parameter                    GET10    = 3'd2 ;
    parameter                    GET15    = 3'd3 ;
    //machine variable
    reg [2:0]                    st_next ;
    reg [2:0]                    st_cur ;
    //(1) state transfer
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            st_cur          <= 'b0 ;
        end
        else begin
            st_cur          <= st_next ;
        end
    end
    //(2) state switch, using block assignment for combination-logic
    always @(*) begin        //all case items need to be displayed completely
```

```
    case(st_cur)
        IDLE:
           case (coin)
              2'b01:        st_next = GET05 ;
              2'b10:        st_next = GET10 ;
              default:      st_next = IDLE ;
           endcase
        GET05:
           case (coin)
              2'b01:        st_next = GET10 ;
              2'b10:        st_next = GET15 ;
              default:      st_next = GET05 ;
           endcase
        GET10:
           case (coin)
              2'b01:        st_next = GET15 ;
              2'b10:        st_next = IDLE ;
              default:      st_next = GET10 ;
           endcase
        GET15:
           case (coin)
              2'b01,2'b10:
                            st_next = IDLE ;
              default:      st_next = GET15 ;
           endcase
       default:             st_next = IDLE ;
    endcase
end
//(3) output logic, using non-block assignment
reg   [1:0]     change_r ;
reg             sell_r ;
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        change_r          <= 2'b0 ;
        sell_r            <= 1'b0 ;
    end
    else if ((st_cur == GET15 && coin ==2'h1)
              || (st_cur == GET10 && coin ==2'd2)) begin
        change_r          <= 2'b0 ;
        sell_r            <= 1'b1 ;
    end
    else if (st_cur == GET15 && coin == 2'h2) begin
        change_r          <= 2'b1 ;
        sell_r            <= 1'b1 ;
    end
    else begin
        change_r          <= 2'b0 ;
        sell_r            <= 1'b0 ;
    end
end
```

```
    assign          sell      = sell_r ;
    assign          change    = change_r ;
endmodule
```

The test-bench design is as follows :

Four scenarios are simulated in the simulation, namely: Case1 corresponds to the consecutive input of 4 5 jiao coins; case2 corresponds to the coin insertion sequence of 1 yuan-5 jiao-1 yuan; case3 corresponds to the coin insertion sequence of 5 jiao-1 yuan-5 jiao; case4 corresponds to 3 consecutive 5 jiao and then a 1 The order of yuan coin insertion.

```
`timescale 1ns/1ps
module test ;
    reg             clk;
    reg             rstn ;
    reg [1:0]       coin ;
    wire [1:0]      change ;
    wire            sell ;
    //clock generating
    parameter       CYCLE_200MHz = 10 ; //
    always begin
        clk = 0 ; #(CYCLE_200MHz/2) ;
        clk = 1 ; #(CYCLE_200MHz/2) ;
    end
    //motivation generating
    reg [9:0]       buy_oper ; //store state of the buy operation
    initial begin
        buy_oper    = 'h0 ;
        coin        = 2'h0 ;
        rstn        = 1'b0 ;
        #8 rstn     = 1'b1 ;
        @(negedge clk) ;
        //case(1) 0.5 -> 0.5 -> 0.5 -> 0.5
        #16 ;
        buy_oper    = 10'b00_0101_0101 ;
        repeat(5) begin
            @(negedge clk) ;
            coin        = buy_oper[1:0] ;
            buy_oper    = buy_oper >> 2 ;
        end
        //case(2) 1 -> 0.5 -> 1, taking change
        #16 ;
        buy_oper    = 10'b00_0010_0110 ;
        repeat(5) begin
            @(negedge clk) ;
            coin        = buy_oper[1:0] ;
            buy_oper    = buy_oper >> 2 ;
        end
```

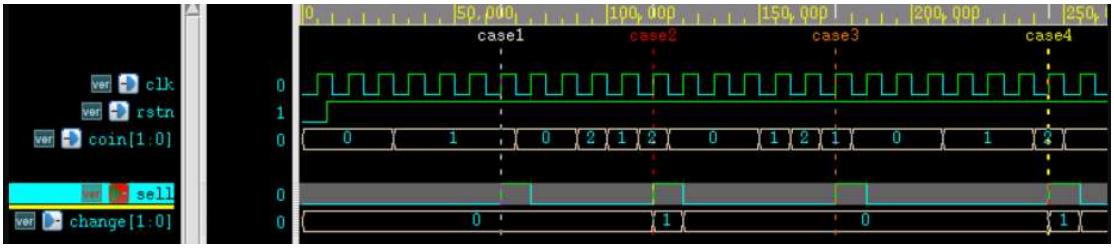
```
        //case(3) 0.5 -> 1 -> 0.5
        #16 ;
        buy_oper    = 10'b00_0001_1001 ;
        repeat(5) begin
            @(negedge clk) ;
            coin          = buy_oper[1:0] ;
            buy_oper    = buy_oper >> 2 ;
        end
        //case(4) 0.5 -> 0.5 -> 0.5 -> 1, taking change
        #16 ;
        buy_oper    = 10'b00_1001_0101 ;
        repeat(5) begin
            @(negedge clk) ;
            coin          = buy_oper[1:0] ;
            buy_oper    = buy_oper >> 2 ;
        end
    end
    //(1) mealy state with 3-stage
    vending_machine_p3       u_mealy_p3
       (
        .clk                    (clk),
        .rstn                   (rstn),
        .coin                   (coin),
        .change                 (change),
        .sell                   (sell)
        );
    //(2) mealy state with 2-stage
    wire [1:0]                  change_p2 ;
    wire                        sell_p2 ;
    vending_machine_p2       u_mealy_p2
       (
        .clk                    (clk),
        .rstn                   (rstn),
        .coin                   (coin),
        .change                 (change_p2),
        .sell                   (sell_p2)
        );

    //(3) mealy state with 1-stage
    wire [1:0]                  change_p1 ;
    wire                        sell_p1 ;
    vending_machine_p1       u_mealy_p1
       (
        .clk                    (clk),
        .rstn                   (rstn),
        .coin                   (coin),
        .change                 (change_p1),
        .sell                   (sell_p1));
    //(4) mealy state with 1-stage
    wire [1:0]                  change_moore ;
```

```
    wire                        sell_moore ;
    vending_machine_moore       u_moore_p3
      (
        .clk                    (clk),
        .rstn                   (rstn),
        .coin                   (coin),
        .change                 (change_moore),
        .sell                   (sell_moore));
    //simulation finish
    always begin
        #100;
        if ($time >= 10000)    $finish ;
    end
endmodule // test
```

Simulations results as follows:

It can be seen from the figure that the sell signal representing the shipment action can be pulled up normally after the coin is inserted, and the signal change representing the change action can also output the correct change signal according to the input coin scene



6 Conclusion

This document introduces the commonly used modules in combinatorial logic and sequential logic. Among them, finite state machines are more complicated, but they are often used. I hope everyone can understand them deeply, use them more in the code, and think more, which is conducive to rapid improvement.

2.3.2 VerilogGrammar Study References

There are many reference materials for learning verilog grammar, but before learning verilog grammar, you need to complete the study of digital logic circuits and the study of semiconductor principles in the analog circuit course.

For further competition and risk in combinational logic circuits, as well as delays in sequential circuits, input and output delays, about the integration of module instantiation, parameter transfer, etc., here is limited to the length of the introduction, you can refer to the verilog

The learning materials are as follows:

[1] Xiaomei Ge FPGA Tutorial

[2] Weisan Academy FPGA Tutorial

[3] Wu Houhang. Playing with FPGA in simple language[M]. Beijing University of Aeronautics and Astronautics Press, 2013.

[4] Xia Yuwen. Verilog digital system design tutorial. 3rd edition [M]. Beijing University of Aeronautics and Astronautics Press, 2013.

[5] Han Bin, Yu Xiaoyu, Zhang Leiming. Detailed explanation of FPGA design skills and case development[M]. Publishing House of Electronics Industry, 2014.

For information on timing, you can use xilinx official timing analysis and the use of constraints.

2.3.3 Usage of DocNav

Installation of DocNav:

-Can be installed separately

-The software is already included in the Vivado installation file, just check it

DocNav overview:

Open the software, Catalog View and Design Hub View will be displayed in the upper left corner.

Catalog View divides the data into categories according to chip series, development tools, IP, etc.

The Design Hub View classifies documents from the FPGA design perspective.

How to use this tool well? One of the purposes of this tool is to help us find Xilinx related documents. Furthermore, it means "use less time to find documents and save more time to read documents". If you are a beginner and have just contacted Xilinx, you may only know the basic flow of FPGA design or simple terminology, then you can start to find the information you need from Design Hub Veiw. If you open DocNav and the Design Hub View is not displayed, you can click the red box button in the figure below (in the upper right corner). Select System-Level Design Flow in the figure below, and a clear flow chart will appear. The wonderful thing about this flow chart is that the blue text in the figure is a hyperlink. Click it and you will find that related documents are all gathered together.

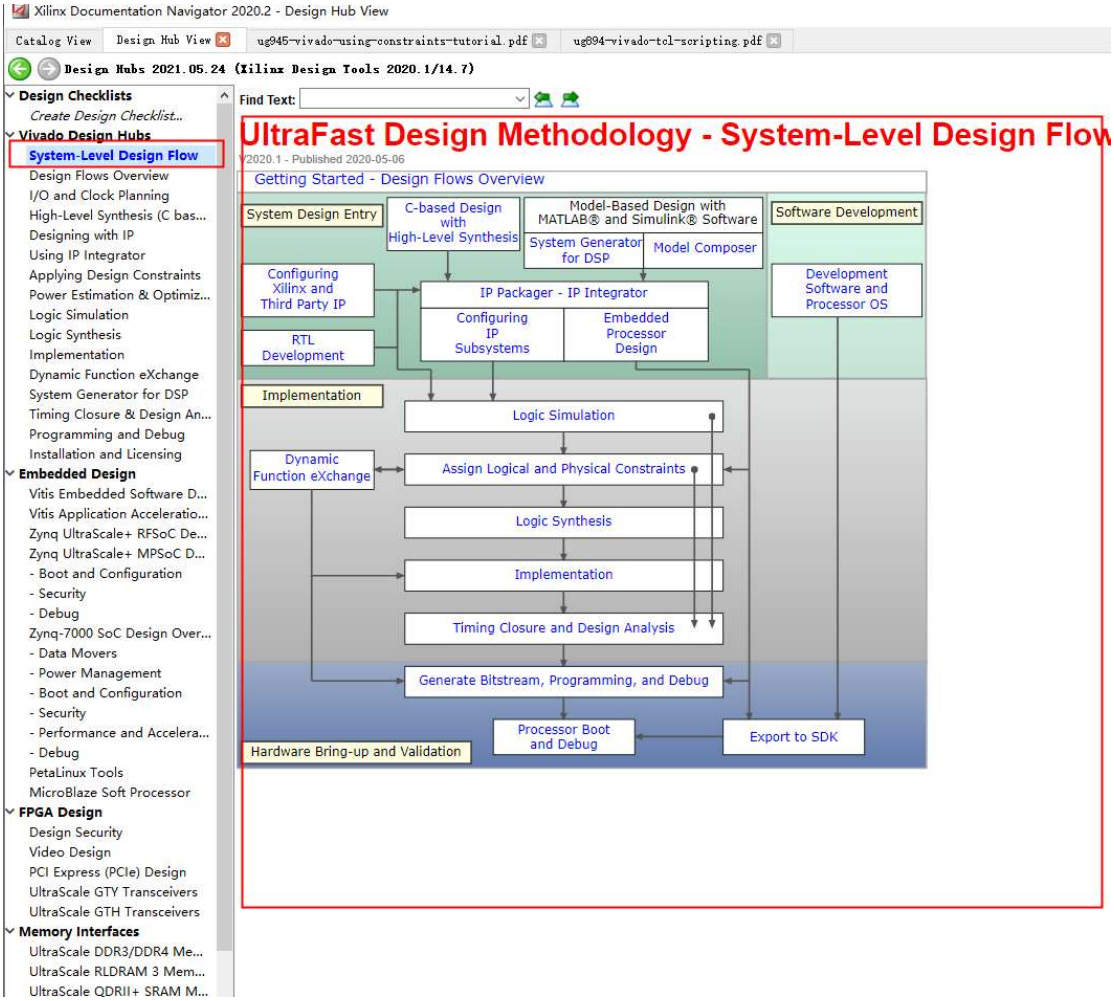


Figure 2-52 DocNav begin

For example, if you click Logic Simulation, the following documents will appear (only a part of them are shown here). Then you can view the document according to your reading habits. For beginners, GetTIng Started is a good part. There are videos, a Tutorial that teaches you step by step, and a User Guide for further in-depth learning. For the User Guide, I personally suggest that you can use it as a dictionary, and you can check it when you encounter problems, which will be more efficient. It's really unnecessary to read page by page.

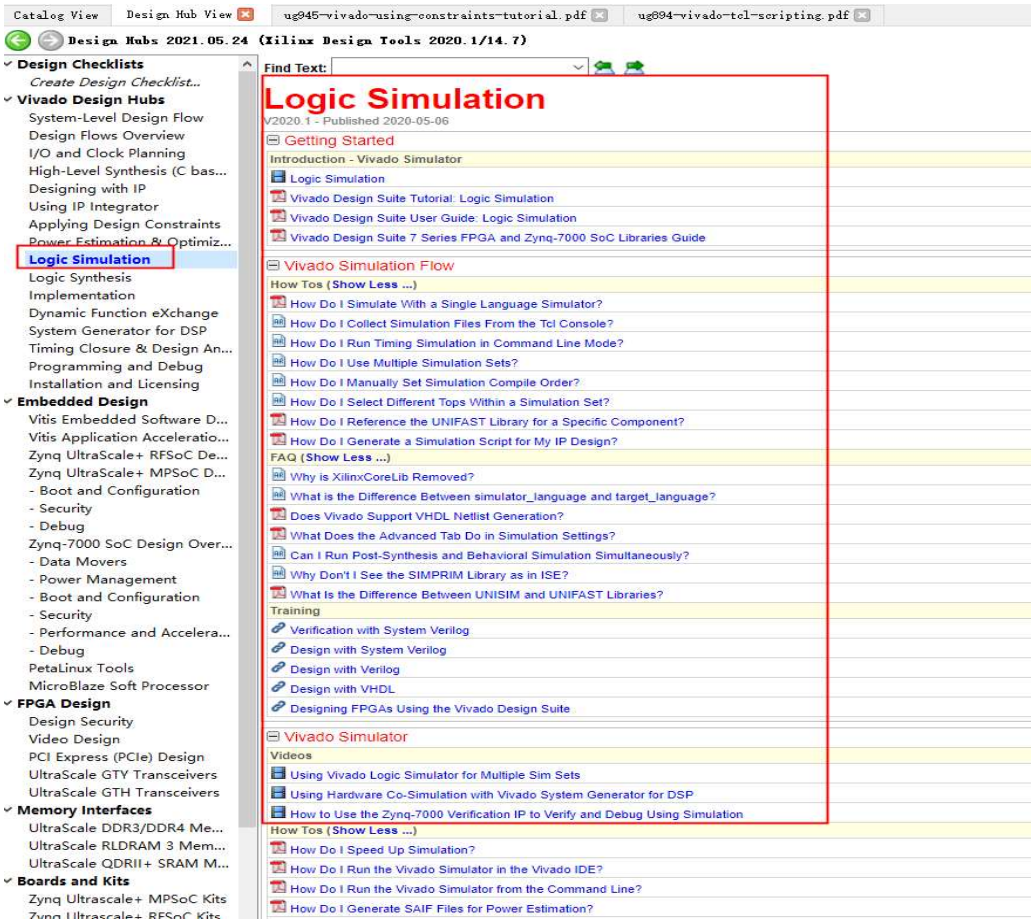


Figure 2-53 DocNavDesign Hub

Quickly view document chapter titles

In DocNav, you can quickly view the document chapter titles without downloading the document. As shown in the figure below, click the mark pointed by the red arrow to display the document chapter titles. To view the content of a chapter, you can directly click the title name to open the document and go to the corresponding chapter.

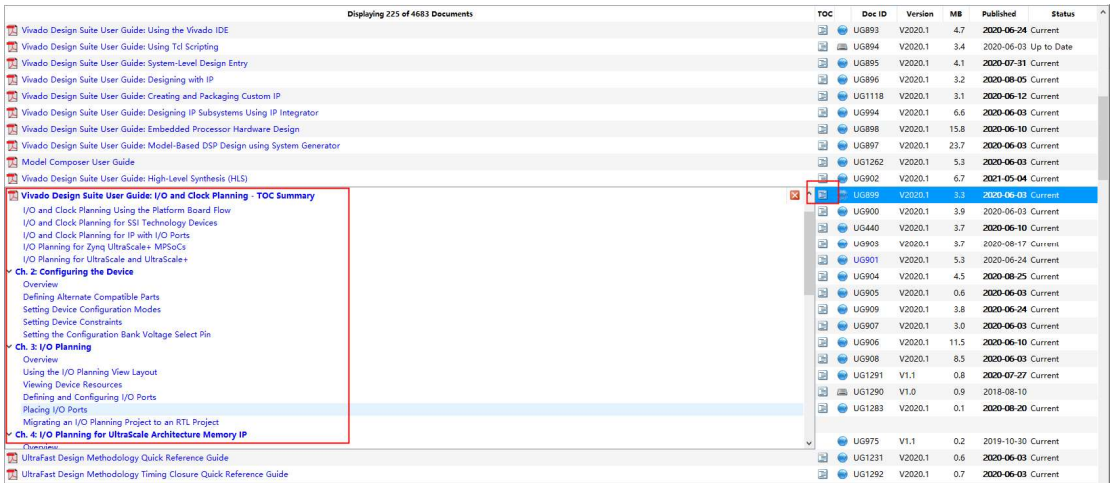


Figure 2-54 DocNav Directory navigation

Find files

If you know the document number (Doc ID) or the keywords in the document title, such as ug949, you can quickly find the document through the following three steps.

Figure 2-55DocNav search

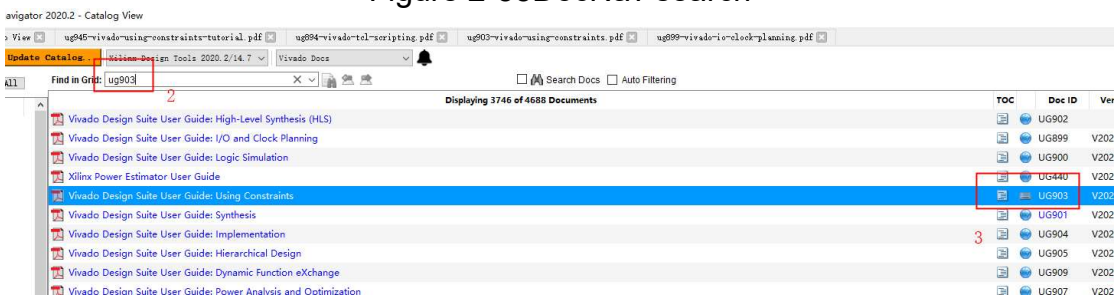


Figure 2-56DocNav search results

If you don’ t know the document number, you can search for documents based on keywords. For example, if you need to find documents related to timing constraints, you can quickly find the documents through the following three steps. For the searched documents, you can further filter, for example, you can filter according to the matching situation.

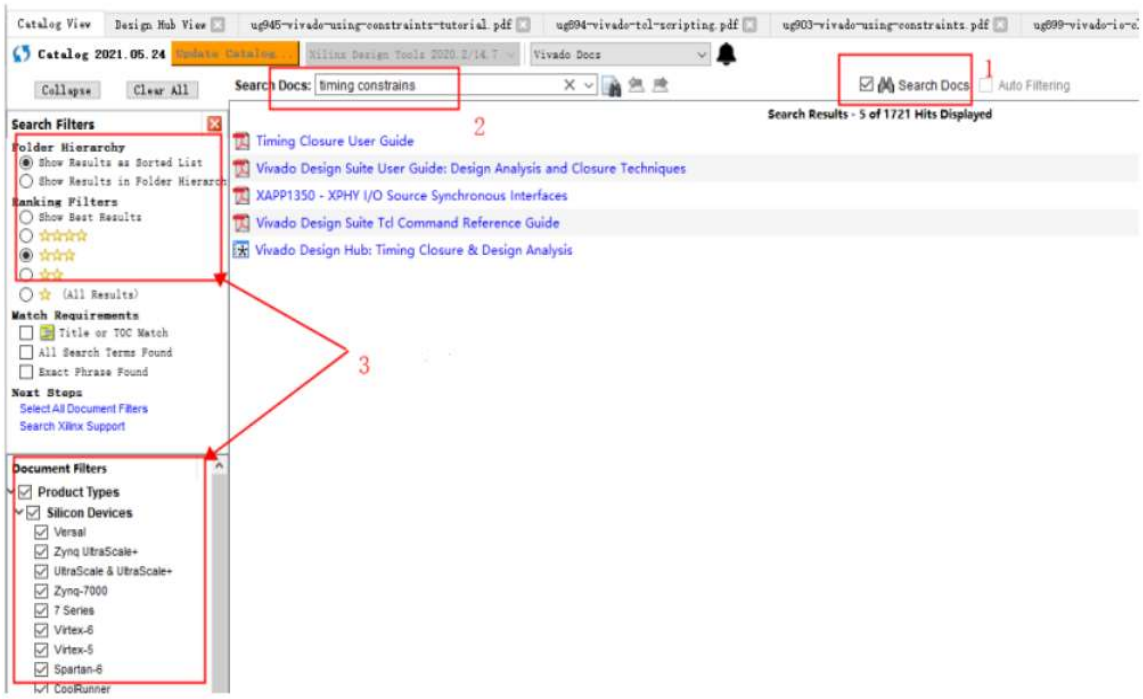


Figure 2-57 DocNav search results filters

Search filters:

Document Filters are on the far left of DocNav, as shown in the figure below. For example, to view the corresponding documents of Vivado, you can checkVivado and cancel the others (remove the "√" in the box). This helps narrow down the search 。

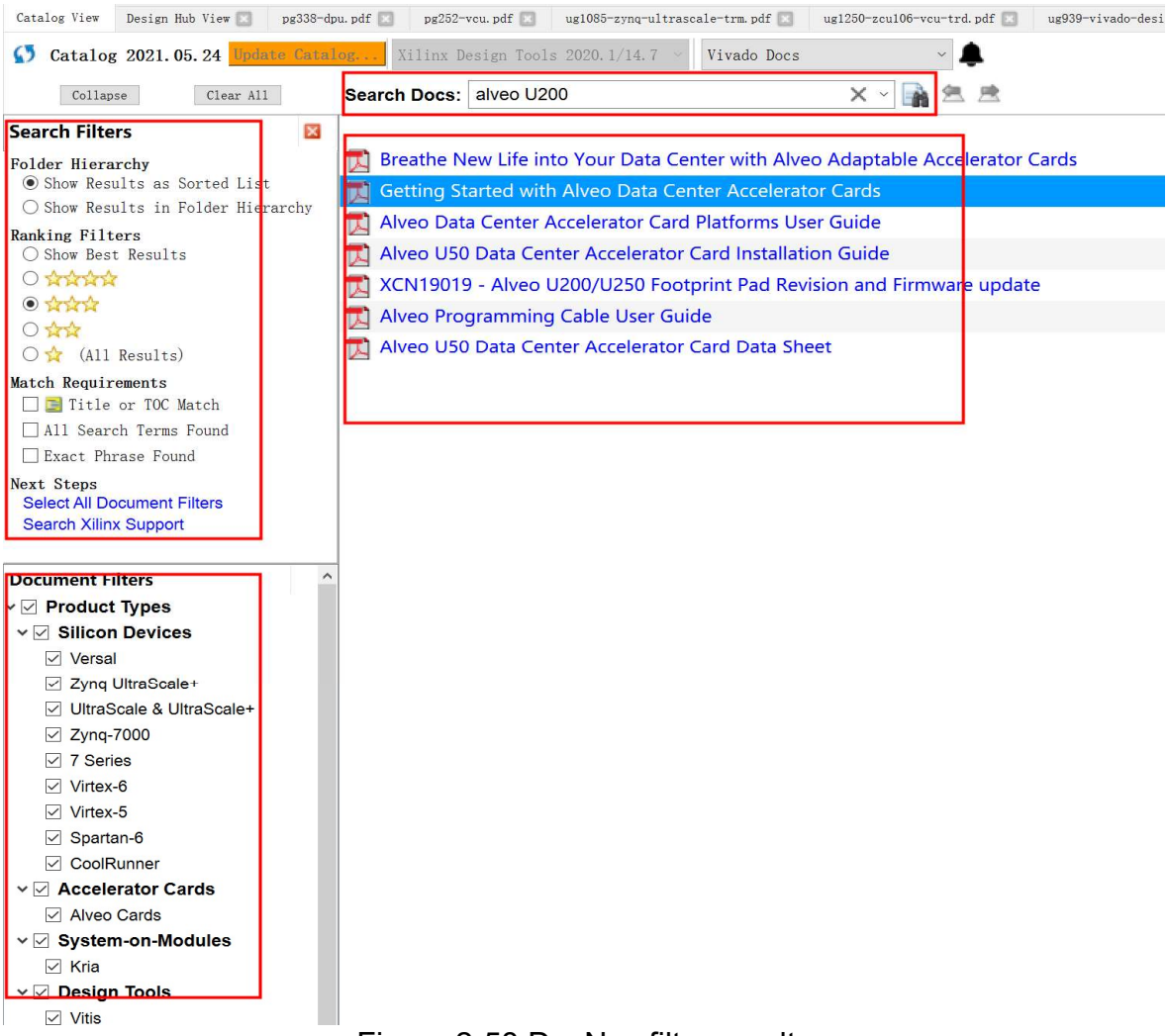


Figure 2-58 DocNav filter results

Document Tray

DocumentTray is located at the far right of DocNav, which can display recently opened documents. The advantage of this is that if you need to find the recently viewed documents, you can quickly find them here 。

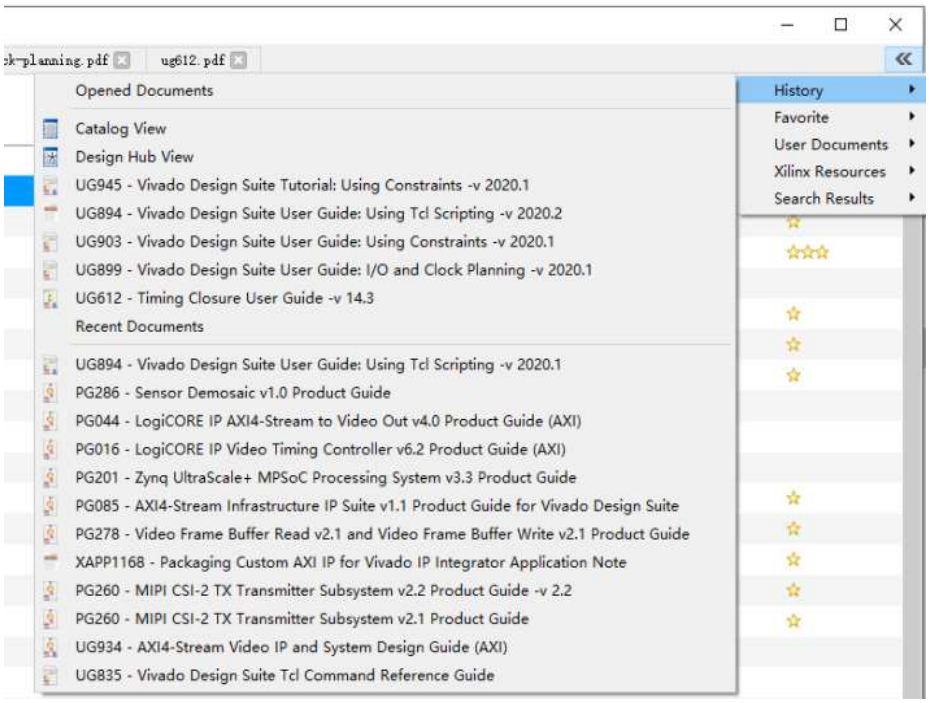


Figure 2-59 DocNav histotry documents

2.3.4 XDC Constraints File

XDC basis：

XDC constraint file refers to xilinx Design constrains, which is a constraint design document dedicated to vivado, including physical constraints and timing constraints.

The traditional ISE UCF constraint document format is not supported. The difference between them is that XDC is based on the standard Synopsys design constraint SDC format. SDC has been designed and used for more than 20 years, so it is the most popular and mature way to describe design constraints. XDC and UCF are fundamentally different, and these basic differences need to be understood. It can be determined that XDC is a combination of the standard Synopsys design constraints (SDC) of design constraints and xilinx's proprietary physical constraints.

The SDC here is (SDC1.9 version). The characteristics of the XDC document are as follows:

1.The commands written in XDC are all commands that follow the TCL syntax.

2. XDC is interpreted like any other command of Vivado Tcl interpreter.

3. The reading and parsing sequence of XDC is the same as other Tcl commands.

We can enter XDC commands at different points in different ways during the design process:

1. Design constraints can be divided into one or more XDC documents for input. In order to add XDC constraints to the memory, one of the following methods can be used, Use the read_xdc command Add it to one of the constraint sets of the project. The XDC file only accepts the built-in Tcl commands of set, list, and expr.

2. Use unmanaged Tcl scripts to generate constraints In order to execute the Tcl script, one of the following operations is sufficient Use the source command Use the read_xdc -unmanaged command Add Tcl commands to one of the constraint sets of the project.

Use specific role XDC constraints:

Compilation process design constraints defines the requirements that must be met in order to play a role in the design of the circuit board. Not all constraints are used to compile all the steps in the process. For example, physical constraints used only during implementation steps (i.e., layout and routers). Because Xilinx® Vivado® Integrated Design Environment (IDE) synthesis and implementation algorithms are timing-driven, so you have to create the appropriate timing constraints.

Over-constrained or under-constrained your design makes timing closure difficult. You must use your application requires corresponding reasonable constraints.

XDC written in a way:

Physical constraints written in a way: Xilinx pins dedicated physical constraints comprising: electrical pin location constraints and constraints.

Pin position constraints： set_property PAKAGE_PIN “Pin Number” [get_ports “Port name”]

Pin electrical constraints： set_property IOSTANDARD “Voltage” [get_ports “port name”]

For example：

```
set_property IOSTANDARD LVCMOS33 [ get_ports sys_clk ]
set_property IOSTANDARD LVCMOS33 [ get_ports {led [ 0 ] } ]
set_property IOSTANDARD LVCMOS33 [ get_ports {led [ 1 ] } ]
```

set_property PACKAGE_PIN U18 [get_ports sys_clk]
set_property PACKAGE_PIN M14 [get_ports {led [0] }]
set_property PACKAGE_PIN M15 [get_ports {led [1] }]

Note：

1) The above grammar is case sensitive;

2) When the port name is an array, it needs to be enclosed in {}, and the port name cannot be a keyword.

Differential signal constraints ：

1,Ordinary difference constraint

The differential signal constraint syntax is the same as in section 1. This is just an example.

1) HR I/O Bank，VCCO = 3.3V，HDMI port constraints

```
set_property PACKAGE_PIN N18 [ get_ports TMDS_clk_p ]
set_property PACKAGE_PIN V20 [ get_ports {TMDS_data_p [ 0 ] } ]
set_property IOSTANDARD TMDS_33 [ get_ports TMDS_clk_p ]
set_property IOSTANDARD TMDS_33 [ get_ports {TMDS_data_p [ 0 ] } ]
```

2) HP I/O Bank，VCCO = 1.8V，HDMI interface constraints

```
set_property PACKAGE_PIN N18 [ get_ports TMDS_clk_p ]
set_property PACKAGE_PIN V20 [ get_ports {TMDS_data_p [ 0 ] }]
set_property IOSTANDARD LVDS [ get_ports TMDS_clk_p ]
set_property IOSTANDARD LVDS [ get_ports {TMDS_data_p [ 0 ] } ]
```

Note：

Differential signal constraint, only P pin is required, the system automatically matches the N pin constraint, of course, there is no problem if both _P and _N pins are constrained;

The level of the differential signal should be restricted according to the VCCO Bank voltage.

Transceiver differential signal constraints：

Transceiver MGTREFCLK clock constraint pin location constraint:

set_property LOC “pin number” [get_ports “pin name”]

For example：

```
set_property LOC G7 [ get_ports Q2_CLK0_GTREFCLK_PAD_N_IN ]
```

set_property LOC G8 [get_ports Q2_CLK0_GTREFCLK_PAD_P_IN]

Transceiver MGT channel constraints :

For GTXE2_CHANNEL channel constraints: one method is to use the 7 series FPGAs transceiver wizard, after configuring the transceiver configuration parameters, automatically generate the XDC template, and then apply the template to your own design; the second method is to write your own XDC constraint file, its position constraint position should refer to the specific schematic signal pin to write the constraint file. Example: For the four-channel transceiver in Figure 1, the GTXE2_CHANNEL constraint.

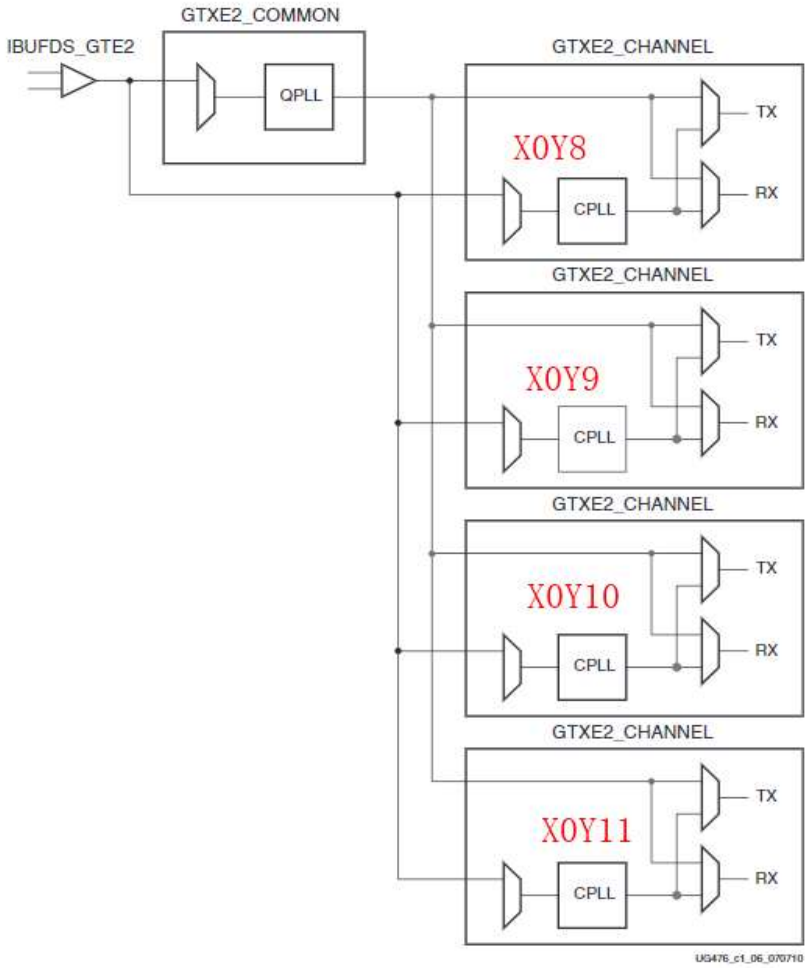


Figure 1-5: **Four Channel Configuration (Reference Clock from the QPLL of GTXE2_COMMON)**

Figure 2-60 GTXE2_ChANNEL

Transceiver MGT channel constraints location :

set_property LOC " GTXE2_CHANNEL_X* Y * " [get_cells "gtxe_2 inst/location"]

For example :

```
##---------- Set placement for gt0_gtx_wrapper_i/GTXE2_CHANNEL ------
set_property LOC GTXE2_CHANNEL_X0Y8 [get_cells gtx_support_i/gtx_init_i/inst/gtx_i/gt0_gtx_i/gtxe2_i]
##---------- Set placement for gt1_gtx_wrapper_i/GTXE2_CHANNEL ------
set_property LOC GTXE2_CHANNEL_X0Y9 [get_cells gtx_support_i/gtx_init_i/inst/gtx_i/gt1_gtx_i/gtxe2_i]
##---------- Set placement for gt2_gtx_wrapper_i/GTXE2_CHANNEL ------
set_property LOC GTXE2_CHANNEL_X0Y10 [get_cells gtx_support_i/gtx_init_i/inst/gtx_i/gt2_gtx_i/gtxe2_i]
```

Figure 2-61 Pin constraints of high-speed serial transceivers

Note：gtxe_2Refer to Figure 2-61 for the instantiated path, and the path name is modified according to the specific project implementation 。

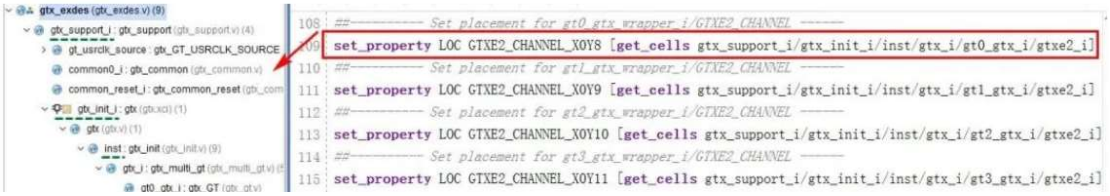


Figure 2-62 Pin constraints of high-speed serial transceivers

Timing Constraints：

Timing constrains include several types below：

Input Path.

Register-to-Register Path.

Output Path.

Path specific exceptions.

Input Constraint:

Figure: Example of an input delay constraint on a port

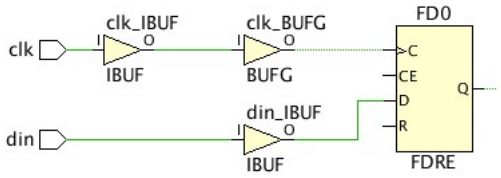


Figure 2-63 input delay example

The input signal din is captured by the register FD0 on the rising edge of the clock clk. The input delay constraint describes the delay between the start edge of the clock and the time when the signal din transitions at the input of the device. The maximum and minimum input delay must be specified to accurately perform setup/restore and hold/remove checks, respectively. After selecting the interface type (system or source synchronization, edge alignment, reference edge, and data

rate), you must define the various delay parameters required to calculate the minimum and maximum input delay values. E.g :

set_input_delay -clock clk 3.4 [get_ports din] -max

set_input_delay -clock clk 1.0 [get_ports din] -min -add

Register-to-Register Constraint:

Register-to-register constraints often refer to cycle constraints. The coverage of cycle constraints includes: Covers the timing requirements of the clock domain Covers the transmission of synchronous data between internal registers Analyze the path in a single clock domain Analyze all paths between related clock domains Consider all frequency, phase, and uncertainty differences between different clock domains

Output Constraint:

The output timing constraints constrain the data from the internal synchronization components or registers to the device pins. System Synchronous Output Constraint The simplified model of system synchronization output is shown in the figure. In the system synchronization output interface, data transmission and acquisition are based on the same clock.

Output delay constraint:

Figure: Example of an output delay constraint on a port

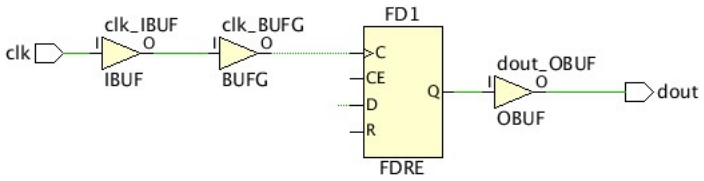


Figure 2-64 output delay example

The output signal dout is sent by the register FD1 on the rising edge of clk, and is captured outside the device by the board clock. In most cases, the board clock is also clk or a phase-shifted copy of clk. The output delay constraint describes the delay between the timedout transition at the device boundary and the capture edge of the board clock. The maximum and minimum output delay must be specified to accurately perform setup and hold checks. After selecting the

interface type (system or source synchronization, edge alignment, reference edge, and data rate), you must provide the various delay parameters required by the wizard to calculate the minimum and maximum output delay values. E.g:

```
set_output_delay -clock clk 5.555 [get_ports dout] -max
set_output_delay -clock clk 0.905 [get_ports dout] -min -add
```

2.3.5 TCL Brief Introduction to Grammar

When you have installed Tcl, the program you will then call to utilize it is tclsh. For instance, if you write some code to a file "hello.tcl", and you want to execute it, you would do it like so: tclsh hello.tcl. Depending on the version of Tcl installed, and the operating system distribution you use, the tclsh program may be a link to the real executable, which may be named tclsh8.6 or tclsh86.exe on Microsoft Windows.

The tclsh is a simple command-line interactive interpreter. You can either start it with a script on the command line, in which case it runs the script to completion and then exits, or you can start it without any arguments, in which case you will be presented with an interactive prompt, usually using a % symbol to prompt for input. In interactive mode, you can type in commands, which Tcl will then execute and display the result, or any error messages that result. To exit the interpreter, type exit and press Return. Playing around with the interactive interpreter is a great way to learn how to use Tcl. Most Tcl commands will produce a helpful error message explaining how they are used if you just type in the command with no arguments. You can get a list of all the commands that your interpreter knows about by typing info commands.

The tclsh executable is just one way of starting a Tcl interpreter. Another common executable, which may be installed on your system, is the wish, or WIndowing SHell. This is a version of Tcl that automatically loads the Tk extension for building graphical user interfaces (GUIs). This tutorial does not cover Tk, and so we will not use the wish interpreter here. Other options are also available, providing more functional environments for developing and debugging code than that provided by the standard tclsh.

One very popular choice isthe TkCon enhanced interpreter, written by Jeff Hobbs. The Eclipse IDE offers good Tcl support, in the form of the DLTK extension,

and the Tcl'ers Wiki offers a list of IDEs with Tcl support and a comprehensive catalogue of Tcl source code editors. Don't panic, though! If you don't know how to use a sophisticated development environment, it is still very easy to write Tcl code by hand in a simple text editor (such as Notepad).

Basis usage in Tcl：

1.output：tcl use " puts"keyword

Usage : puts -nonewline -channelId -string

The output command of Tcl is "puts", which outputs the string to the standard output channelId. The parameter between the two question marks in the syntax is optional.

puts hello

=> hello

puts –nonewline "hello hello"

=>hello hello

But if you output a piece of text with spaces, you must enclose it in double quotes or curly braces The -nonewline option tells puts not to output carriage returns and line feeds. Note: The function of double quotation marks and curly braces is to organize multiple words into one argument, but they are different! This difference is that when dealing with "replacement operations", the former allows replacement to occur, while the latter may prevent replacement. The usage and differences between the two will be discussed in the future. Both have the same effect here.

2. Assignment: tcl uses the "set" keyword to define parameters, without specifying the type of variable value, because there is only one type of variable value-string. When assigning a value to a variable, a section of memory space is opened for the variable to store the variable value.

set varName [value]

set a Hello ;#Define variable "a" and assign =>Hello

puts $a ;#output variable value =>Hello

set a "Hello world" ;#assignment again =>Hello world

set a "Hello world" => Test Tcl ;#output variables , don't add " $" this place

puts $a =>Hello world ;#output variables，need to add " $"

puts a => a ;#output character " a"

set b $a =>Hello world

puts $b =>Hello world ;#puts the value of "a" to b

3.replacement

(1):$

The "$" character implements reference replacement and is used to quote parameter values. Also used above Tcl only explains the replacement once, and ignores the nested "$".

set foo oo =>oo

set dollar foo =>foo

set x $$dollar =>$foo ;#Replace "$dollar" with the dollar value (foo). The ;# command is equivalent to set x {$foo}, and braces prevent substitution.

set x {$foo} =>$foo

set y $x =>$foo ; #First round of substitution

(2) :[]

The square brackets "[]" complete the command substitution. Enclose a command with "[]". After the command is executed, the result will be returned.

set b [set a 5] ;#set a 5 outputs assinged to b =>5

puts $b =>5

set c [expr 5 * 10] ;#Assign the result of the multiplication to c =>50

(3): ""and{}

Double quotation marks and curly braces organize multiple words into one parameter, which is also a substitution operation. "" and how to replace within {}? The general principle is that the replacement within "" will proceed normally, and the replacement within {} may be blocked.

set a 123

=>123

puts "$a" #replace =>123

puts {$a} #=>$a

In summary :

The order of execution of the TCL language is: first group, then replace, and finally execute No substitutions in curly braces Double quotation marks and curly braces are used for grouping, but the difference lies in whether to support the operation of the replacement statement is divided into three steps:

First group

Replace next

Last run The role of $ is the variable leader.

If you want to replace a variable in the string, you may also need to use {} to define the start and end of the variable. There are three ways to group: spaces, double quotes and curly braces. In addition, the escape character \ here is to improve or eliminate the ability of characters.

2.4 Sections of this Chapter

This chapter mainly introduces some basic knowledge, including the precautions for installation and use of basic tools (vivado and modelsim), and explains some common problems encountered in use, and the places that need attention are specially explained. A more in-depth knowledge is the constraint. If you are not very clear about digital logic circuits, you need to learn digital logic circuits first. There are some basic timing-related concepts such as setup time and hold time. The physical constraints are well understood, that is, the consistency of the output and input pin assignments and electrical characteristics with the chip data sheet, which facilitates the implementation process. The subsequent layout is a more in-depth problem, and interested friends can try to explore by themselves.

Chapter 3 Detailed Configuration of Hardware Platform

The main advantage of ZYNQ is the reasonable combination of FPGA and ARM, which places higher requirements on developers.

Starting from this chapter, we need FPGA engineers and software engineers to work together. FPGA engineers are responsible for setting up the Vivado project and providing good hardware to software developers, who can then develop applications on this basis. A good division of labor is also conducive to the advancement of the project. If it is a software developer who wants to do everything, it may take a lot of time and energy to learn FPGA knowledge. It is a painful process to change from software thinking to hardware thinking. If pure learning and time, then It's another matter. Professional people are a good choice to do professional things.

3.1 The First Project is Established

3.1.1 Vivado Project New

Open vivado，selcet Create Project

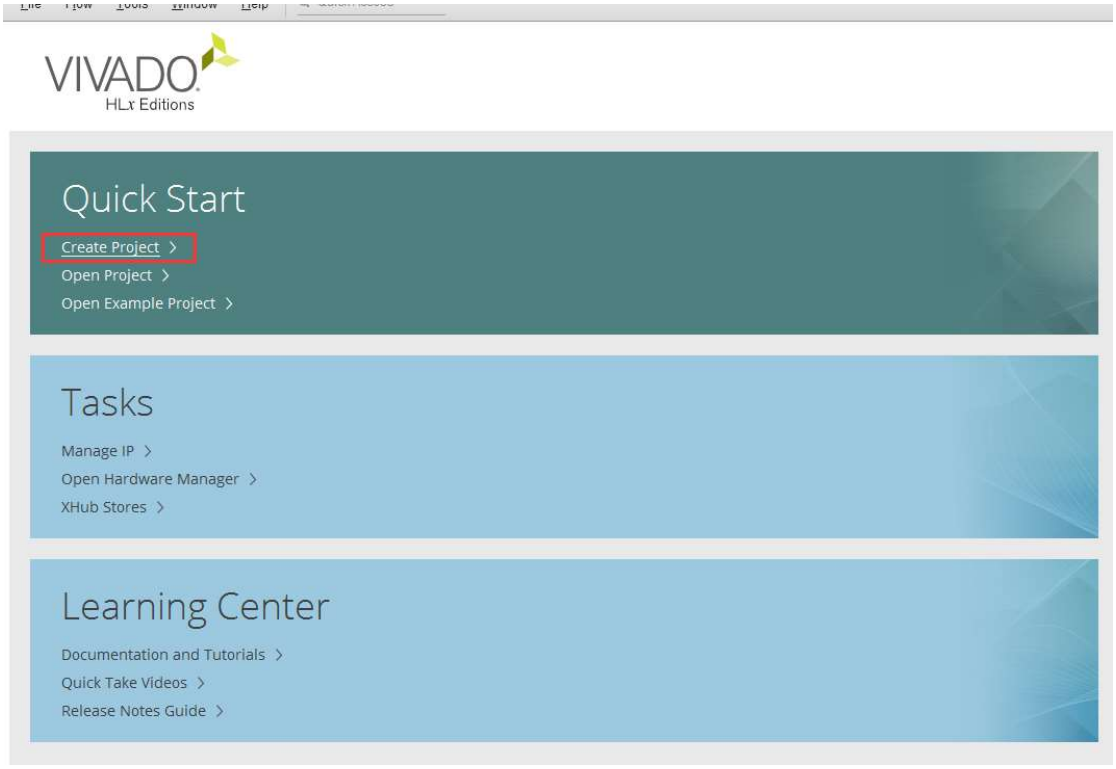


Figure 3-1 vivado page begin

Click Next

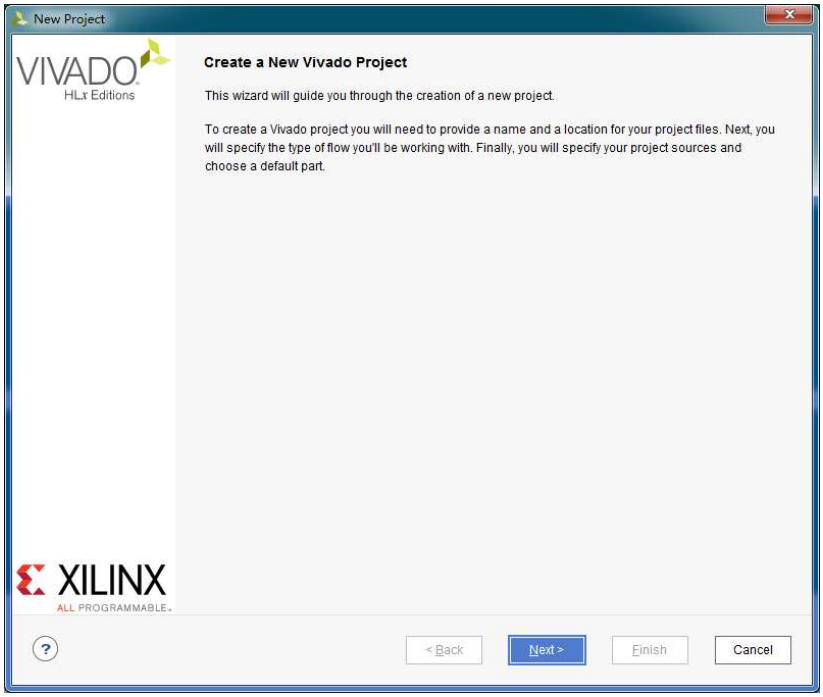


Figure 3-2 vivado page to buit project

Design project name and location

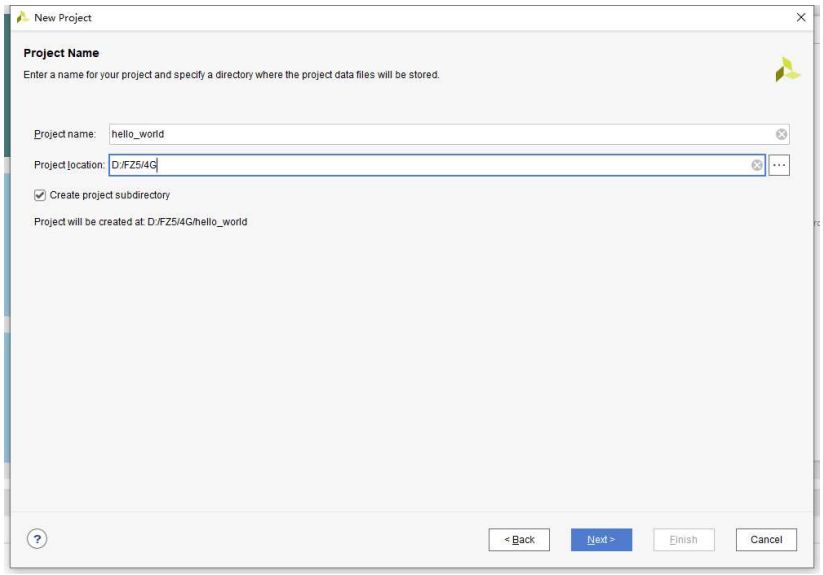


Figure 3-3 vivado page to built new

Click Next

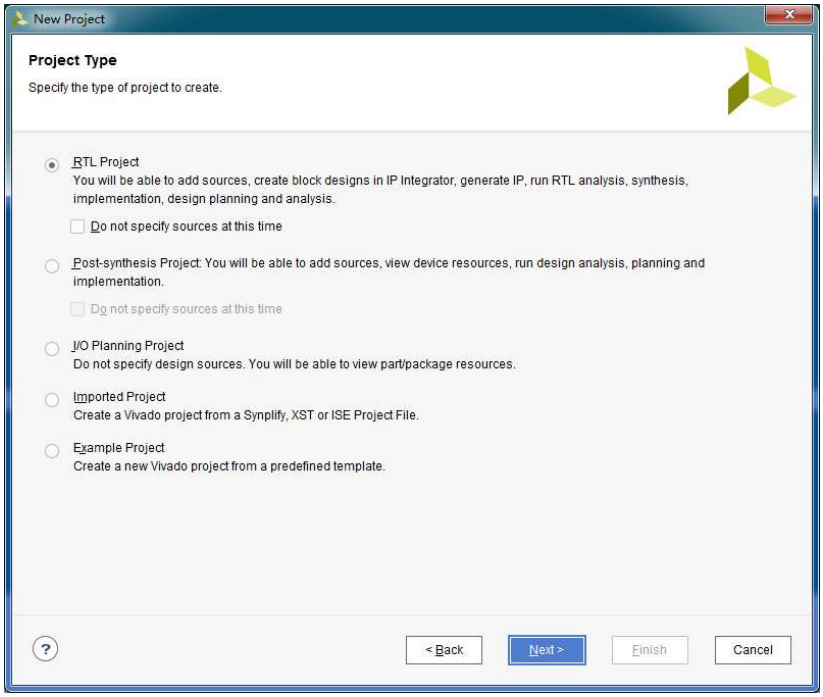


Figure 3-4 vivado built new project -usage RTL

In this place don't Add Sources or Add Constraints，click Next。

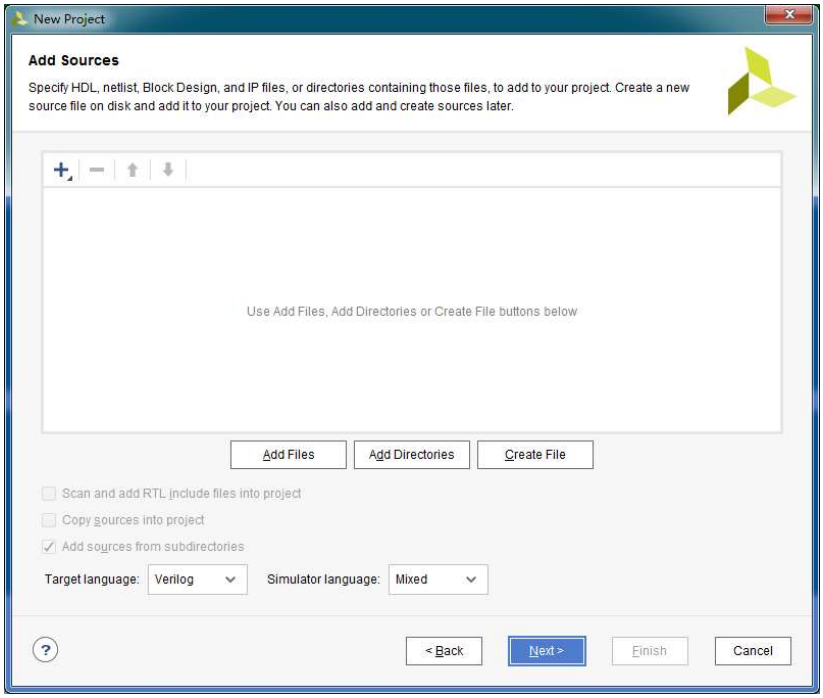


Figure 3-5 vivado-add existing files or constrains ,next

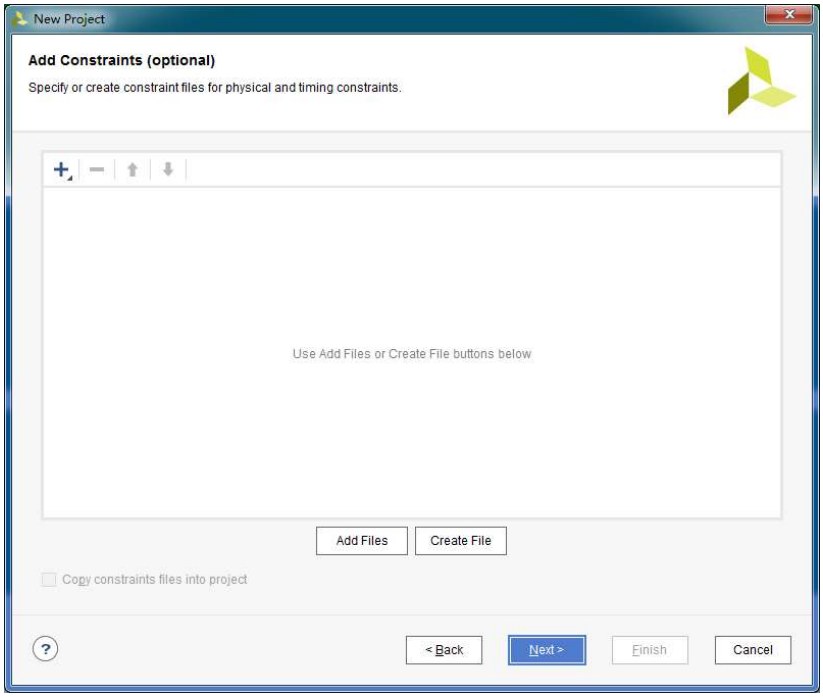


Figure 3-6 vivado built new project

Int Default Part，select xczu5ev-sfvc784-1-i ，click Next

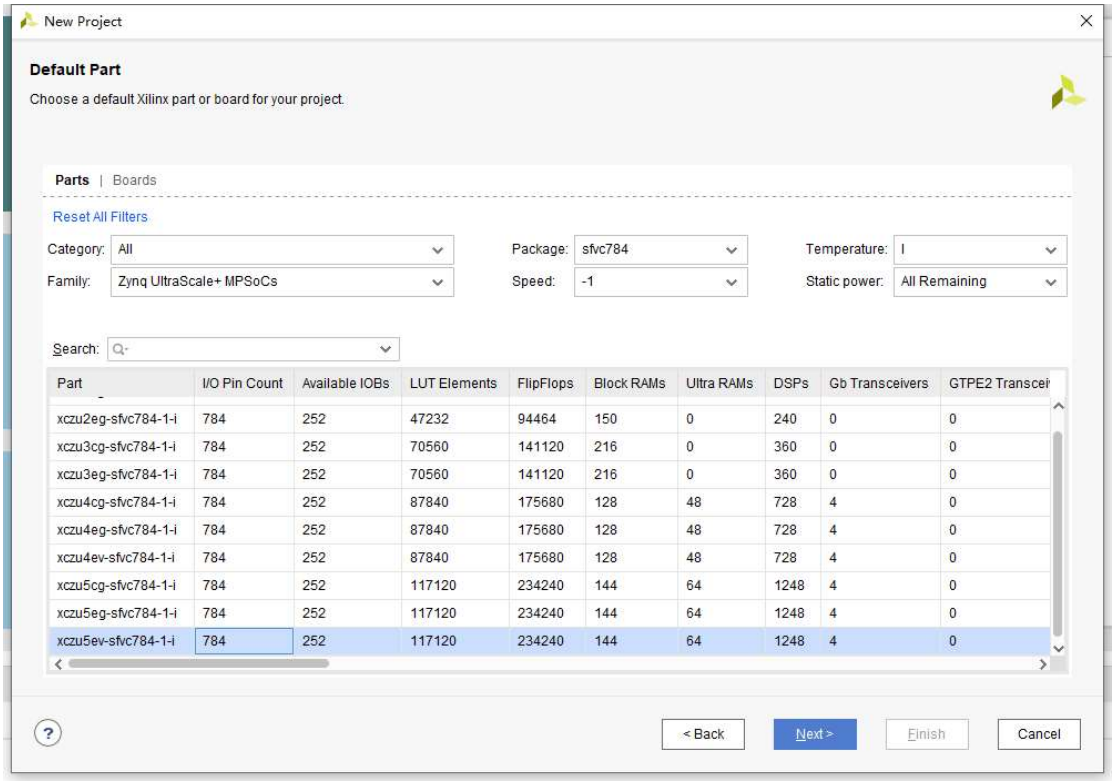


Figure 3-7 vivado built new project

Summary page，You can check the various setting options when creating the project. After confirming that they are correct, click Finish.

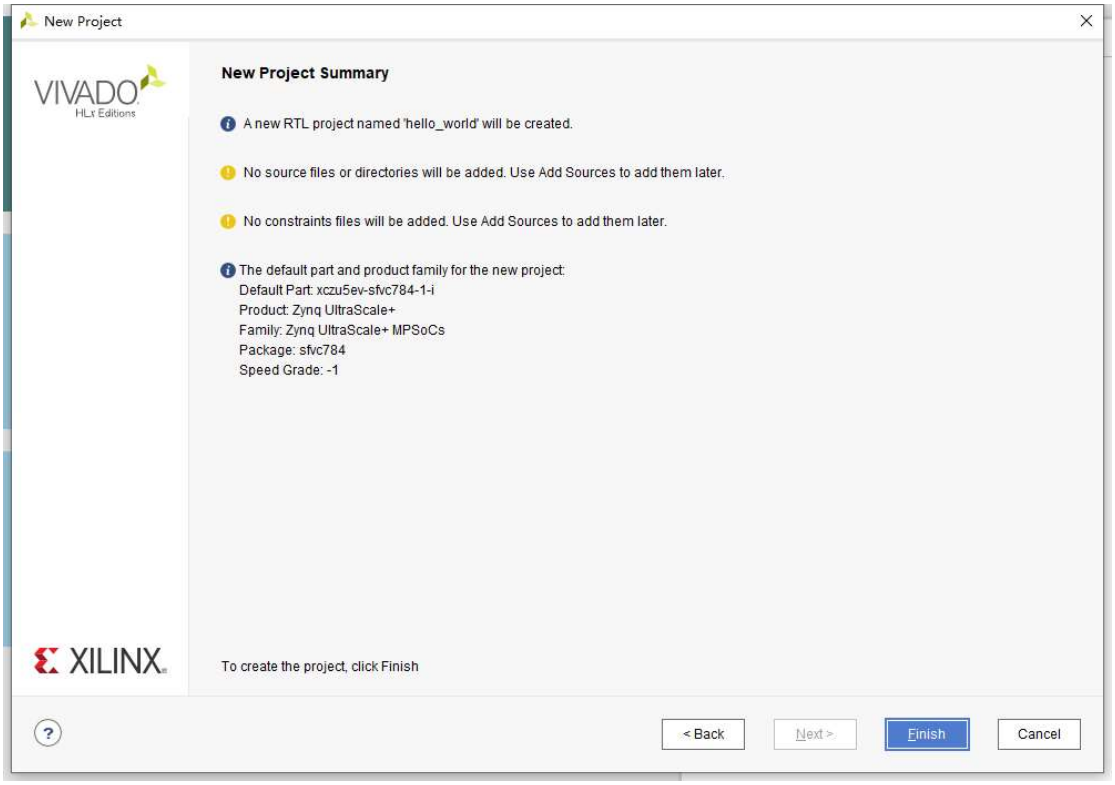


Figure 3-8 vivado summary page

The vivado page after built project。

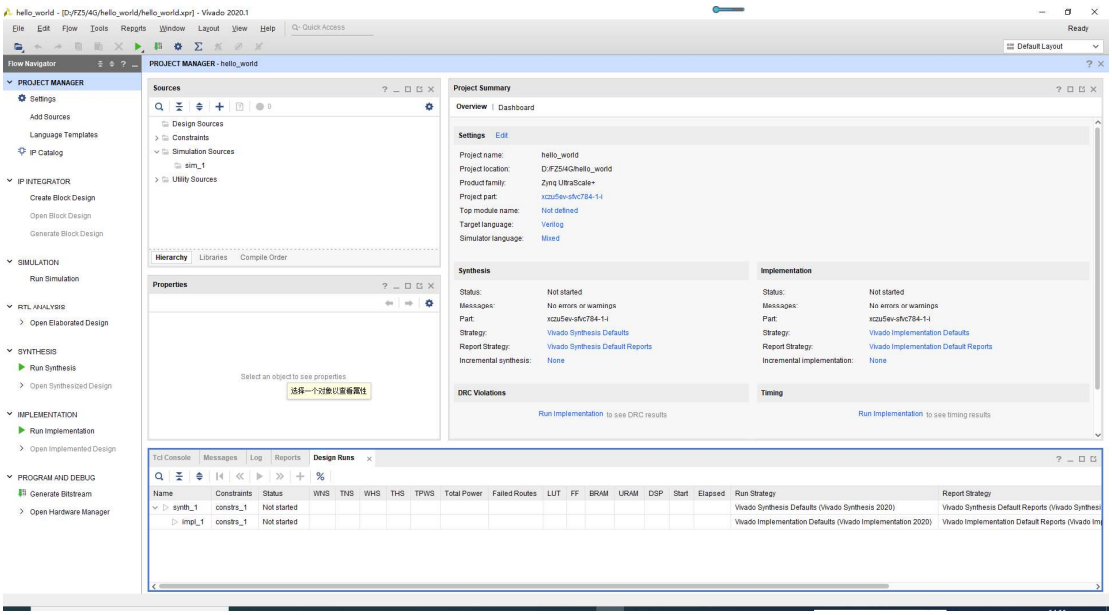


Figure 3-9 vivado begin page after built project

Below IP INTEGRATOR，click Create Block Design。

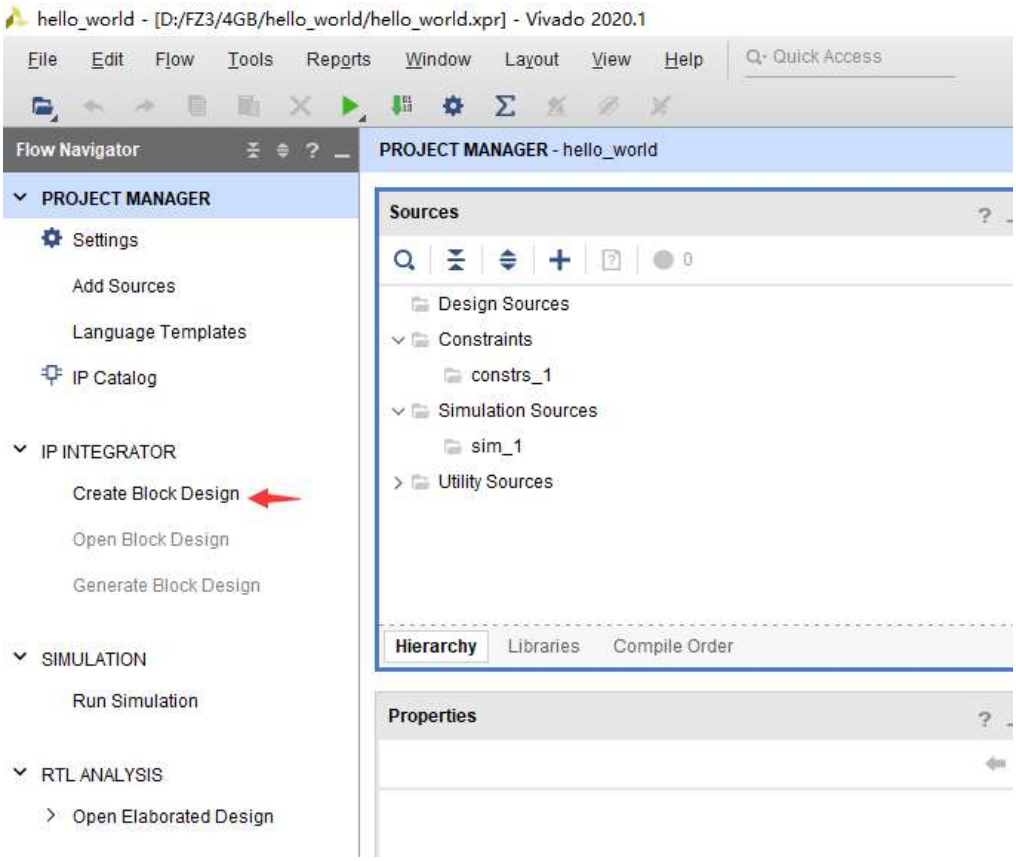


Figure 3-10

Design name remain design_1，click OK。

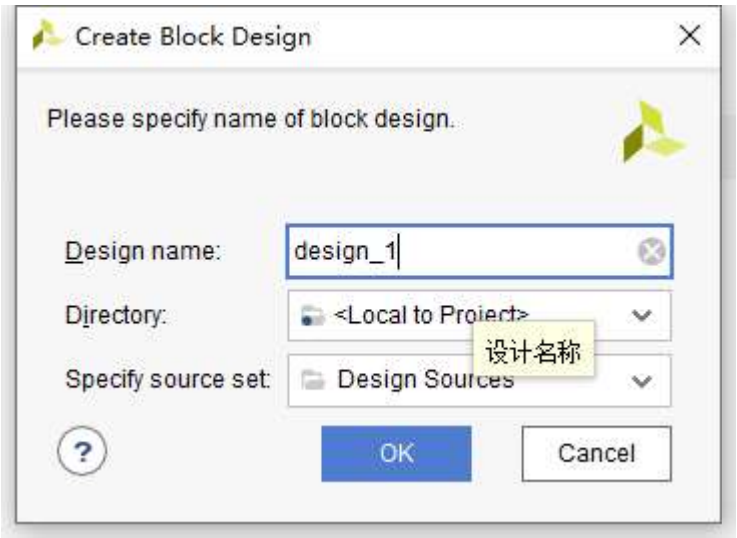


Figure 3-11 fill Block design name

3.1.2 PS Detail Configuration

Click Add IP to add IP

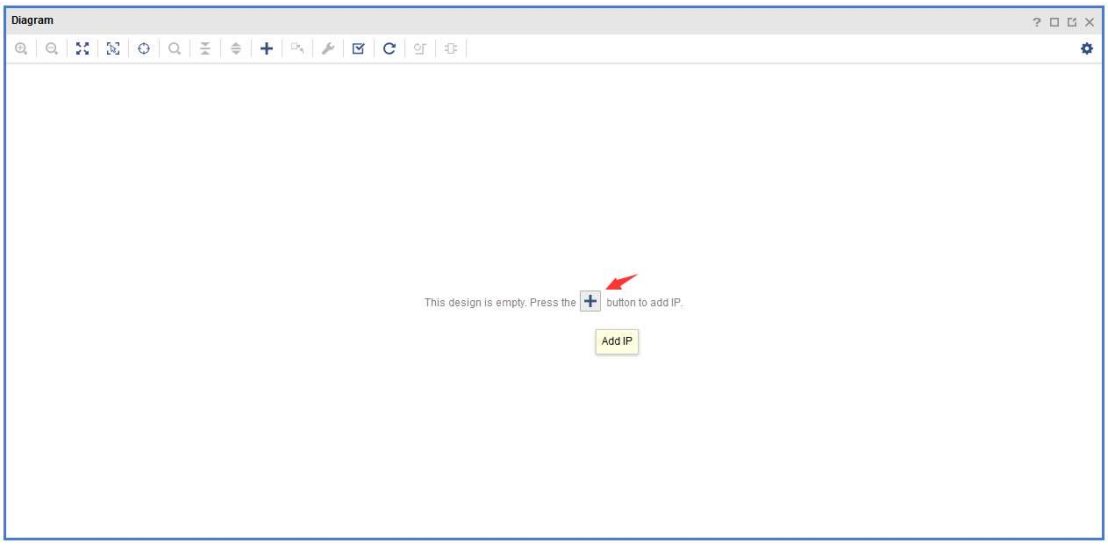


Figure 3-12 usage of Block design

Chose mpsoc，then double click Zynq UltraScale+MPSoC to add mpso IP

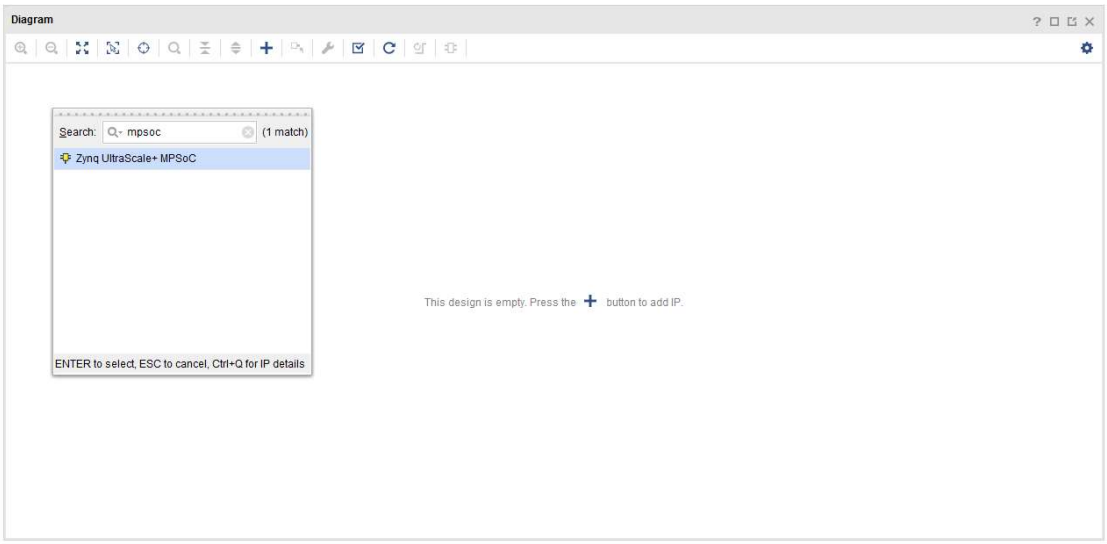


Figure 3-13 Add MPSOC IP

Zynq mpsoc Ip show as below

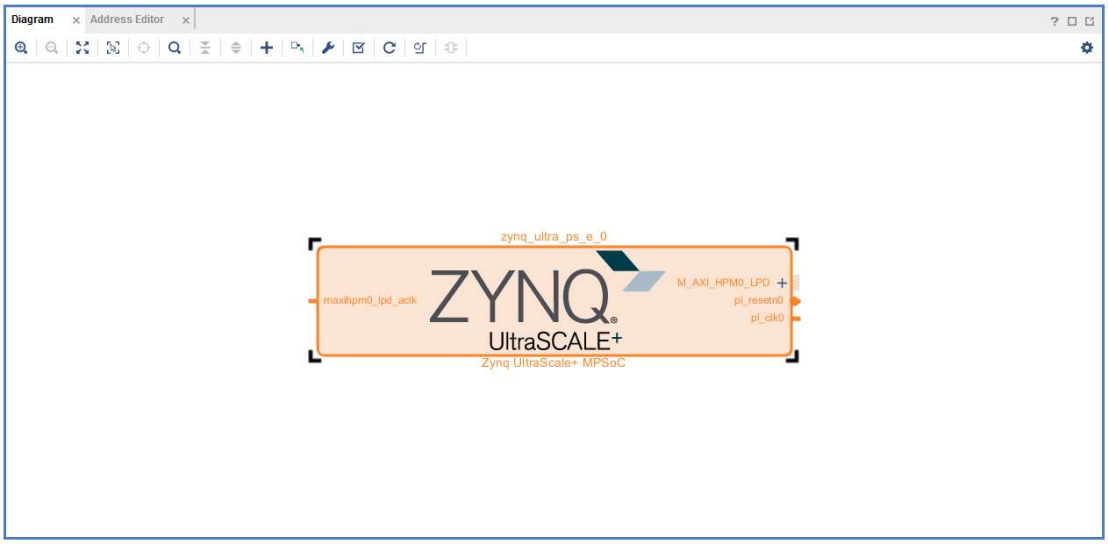


Figure 3-14 MPSOC IP

Double click zynq mpsoc IP to configuration。

The first interface that appears is the architecture diagram of the ZYNQ hard core. You can clearly see its structure. Refer to the ug1085 document, which contains a detailed introduction to ZYNQ. The green part in the picture is the configurable module, you can click to enter the corresponding editing interface, of course, you can also enter the editing in the left window. The functions of each window are introduced below.

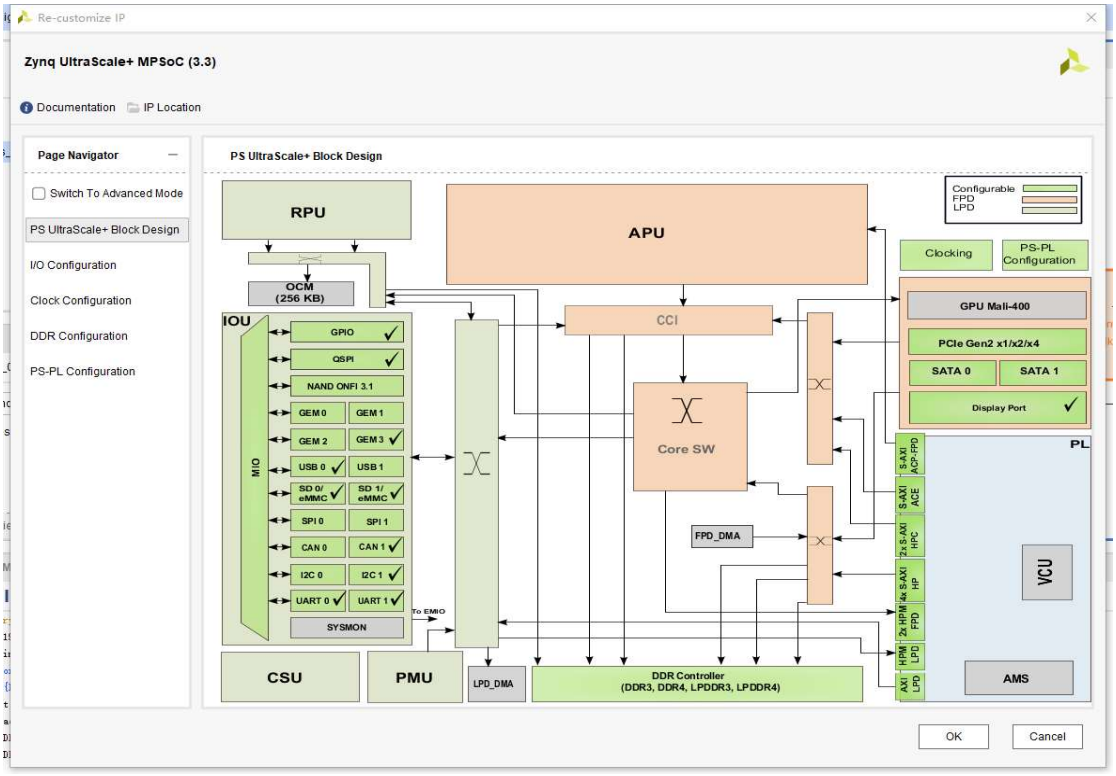


Figure 3-15 configuration MPSOC IP page

1、 Voltage configuration

2、 In the I/O Configuration window, configure the BANK0~BANK3 voltage LVCMOS33 according to the actual voltage of the hardware circuit.

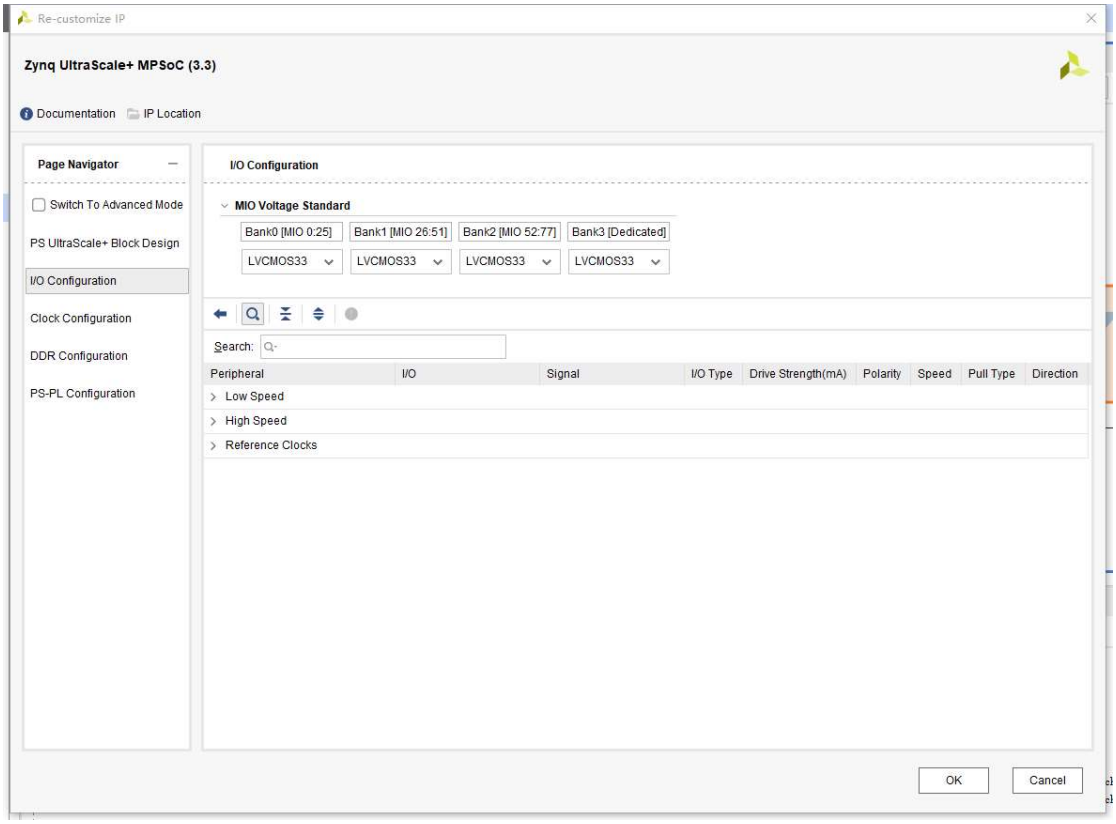


Figure 3-16 configuration IO

3、 Low Speed configuration

Select QSPI，" Single" mode， Data Mode is" x4 "

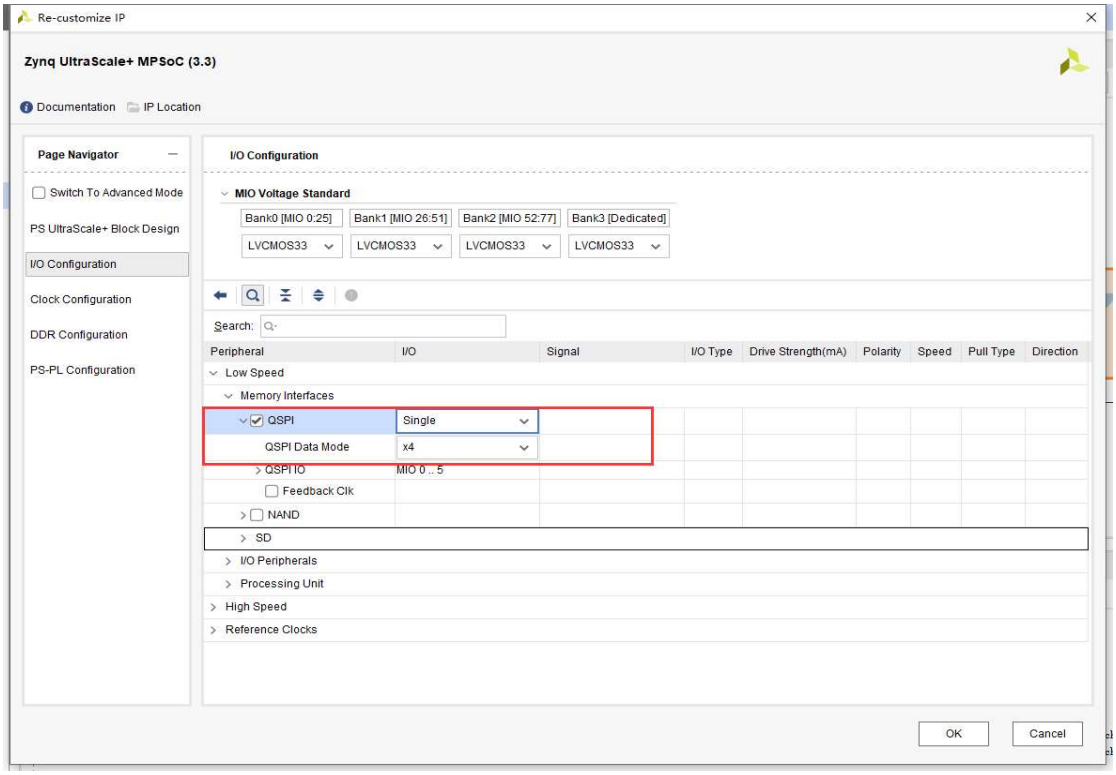


Figure 3-17 Configuration QSPI

Check SD 0 to configure eMMC. Select MIO13..22, Slot Type select eMMC, Data Transfer Mode is 8Bit, check Reset, and select MIO23.

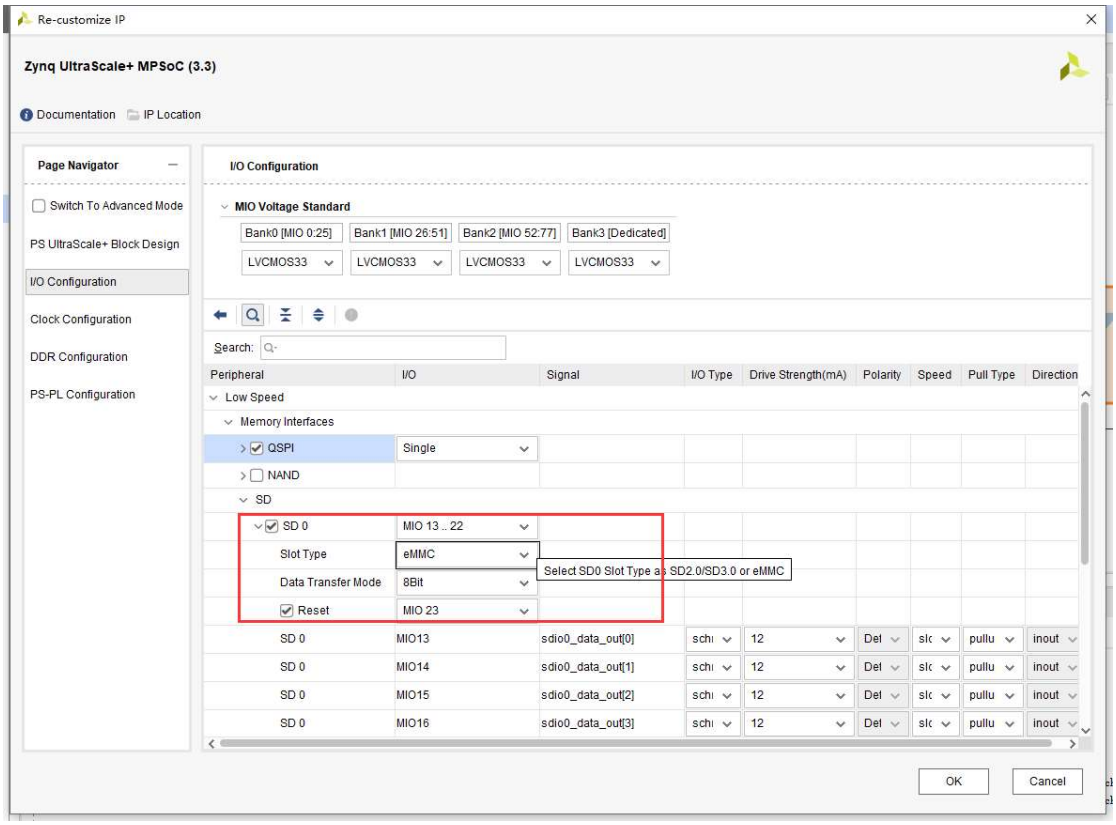


Figure 3-18 Configuration SD

Check SD 1 to configure SD card. Select MIO 46..51, Slot Type select SD 2.0, Data Transfer Mode select 4Bit, check CD, used to detect SD card insertion, select MIO45, check WP, select MIO44, used for SD card write protection 。

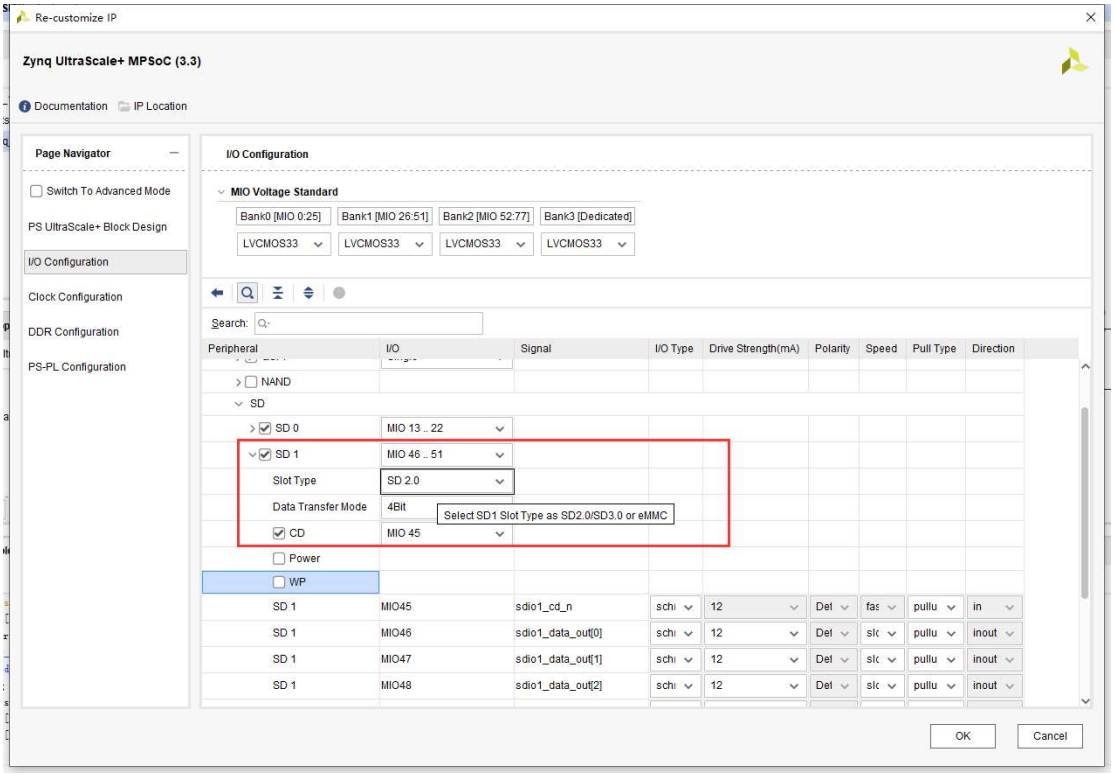


Figure 3-19 Configuration SD1

Check CAN 1, select MIO 28..29 Check I2C 1, select MIO24..25 Check the serial port UART 0, select MIO 26..27, check UART 1, select MIO 8..9 Check GPIO1 MIO, GPIO0 MIO

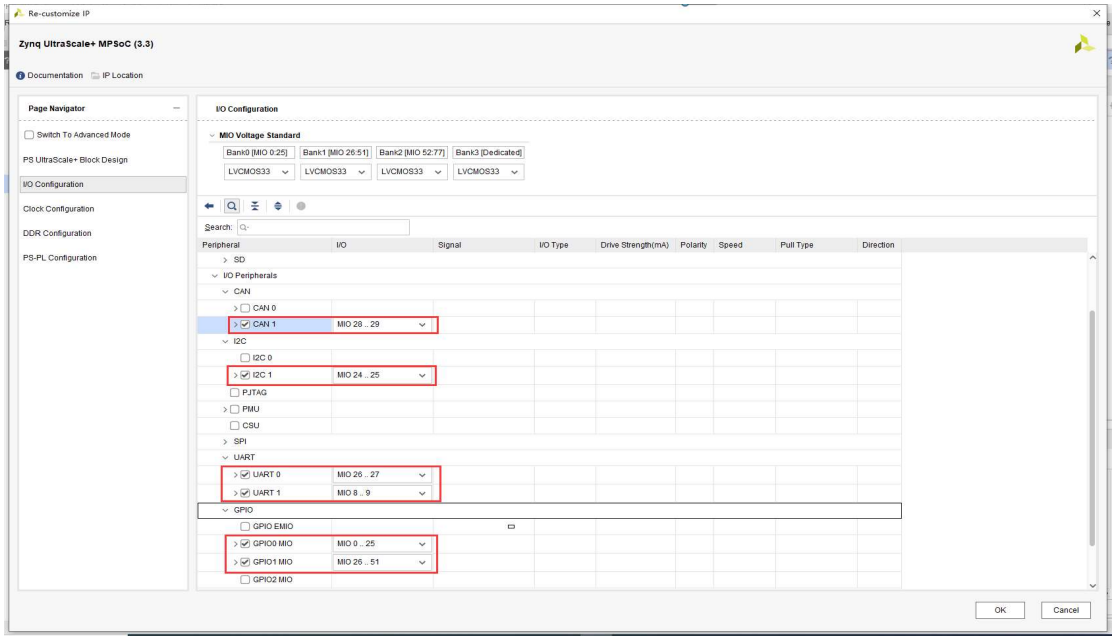


Figure 3-20 configuration CAN、I2C、UART、GPIO

Check SWDT 0、SWDT 1

Check TTC 0~TTC 3

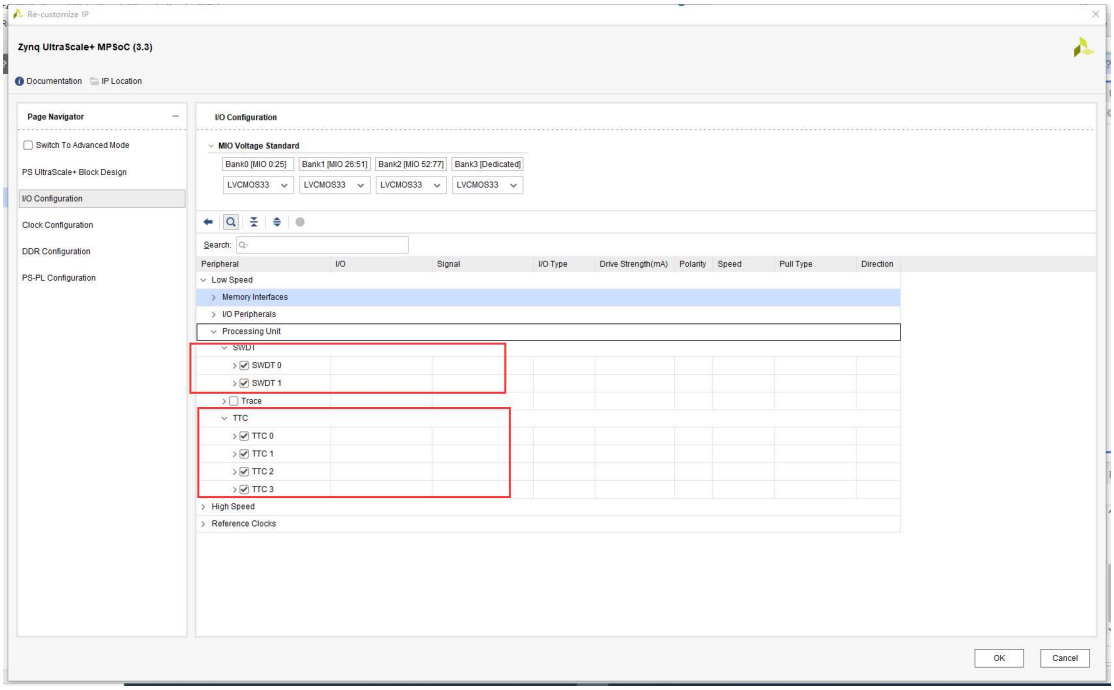


Figure 3-21 configuration SWDT、TTC

4、High Speed Configuration

In the High Speed section, first configure the PS side Ethernet, check GEM 3, select MIO 64..75, check MDIO3, select MIO 76..77

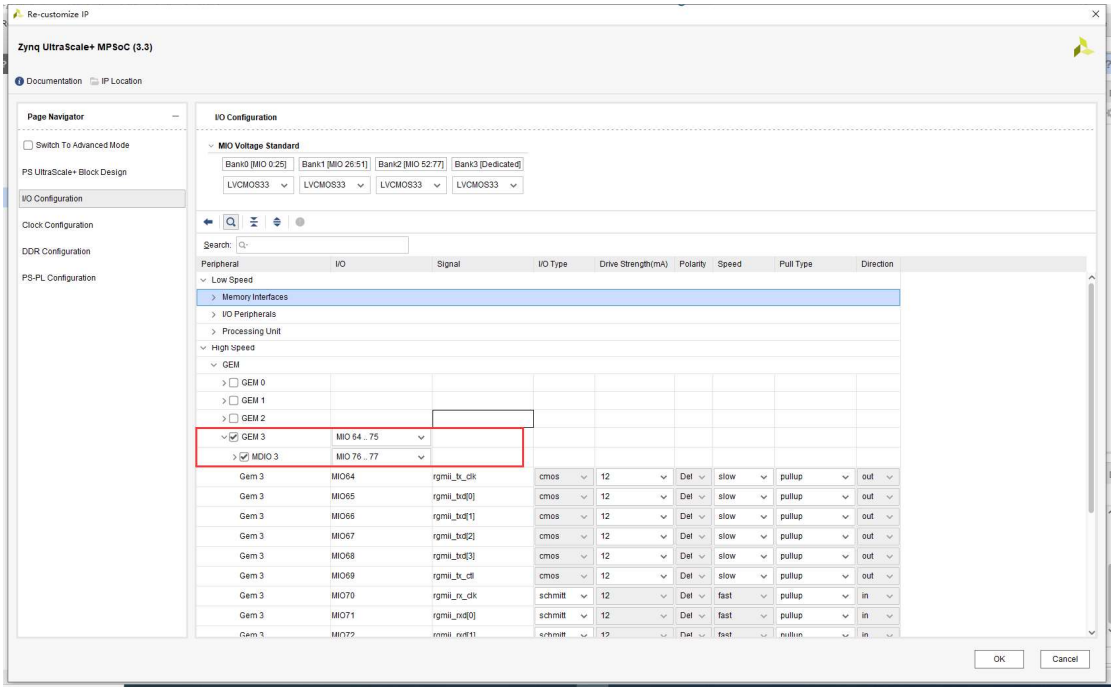


Figure 3-22 Configuration GEM

Check USB 0，select MIO 52..63，In USB 3.0，select GT Lane1

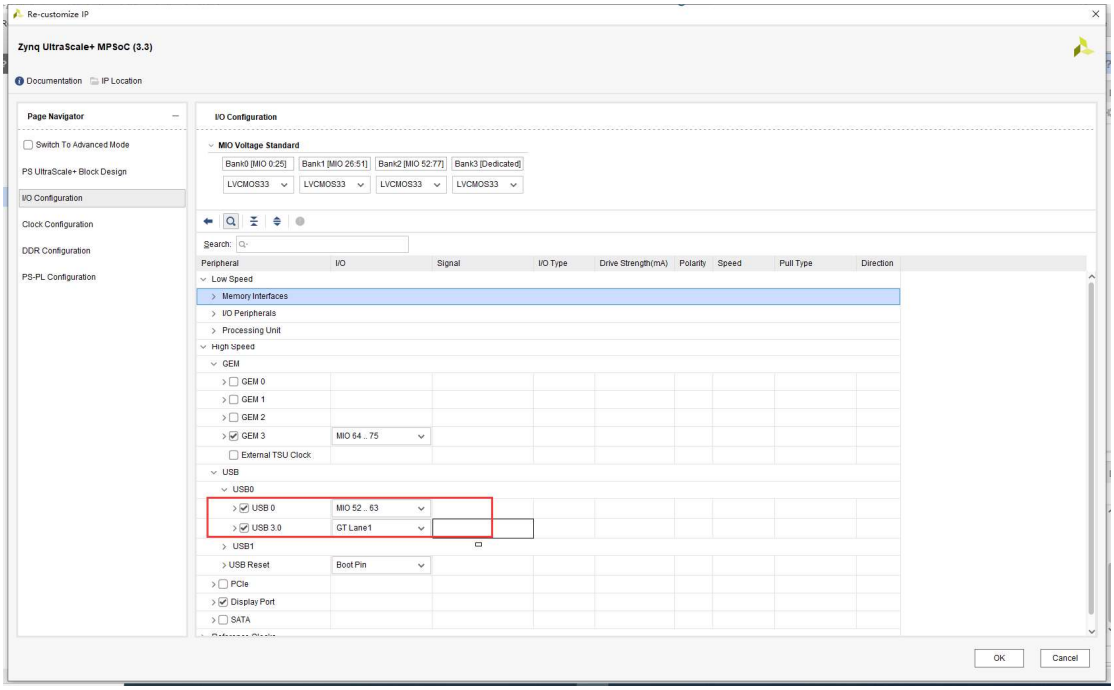


Figure 3-23 Configuration USB

Check Display Port，DPAUX select MIO 34..37， Lane Selection is Dual Higher

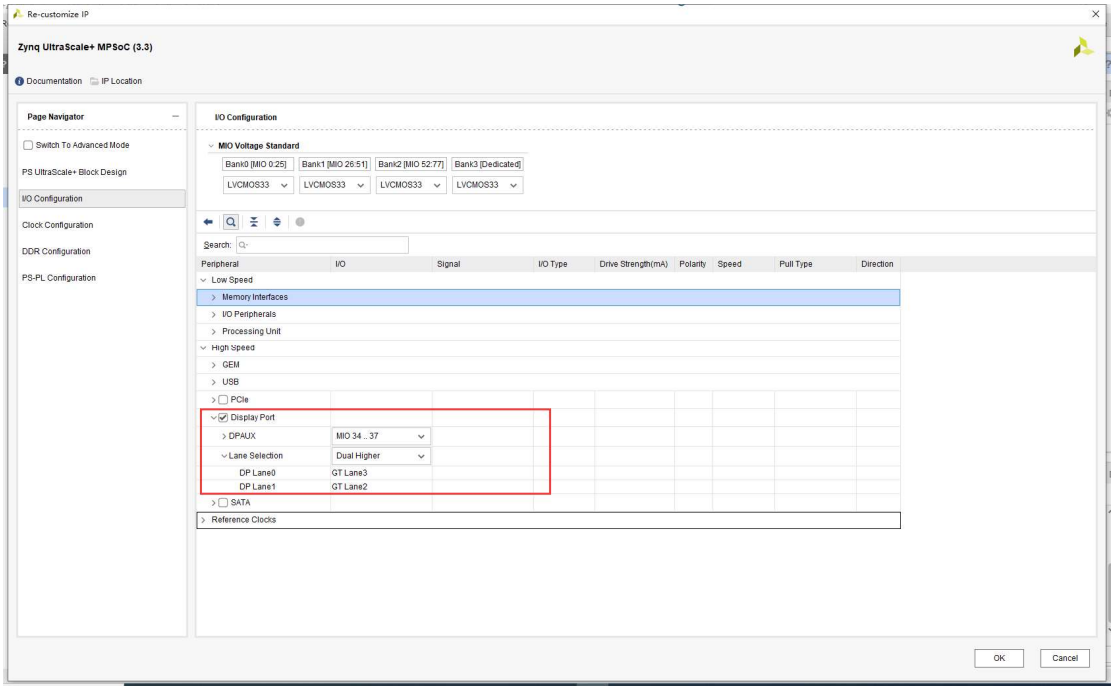


Figure 3-24 configuration Display Port

Then ，IO configuration finished。

4．Clock Configuration

In the Clock Configuration interface, the Input Clocks window configures the reference clock, where PSS_REF_CLOCK is the ARM reference clock, the default is 33.333MHz; Display Port selects Ref Clk2, 27MHz; USB0 selects Ref Clk1, 26MHz.

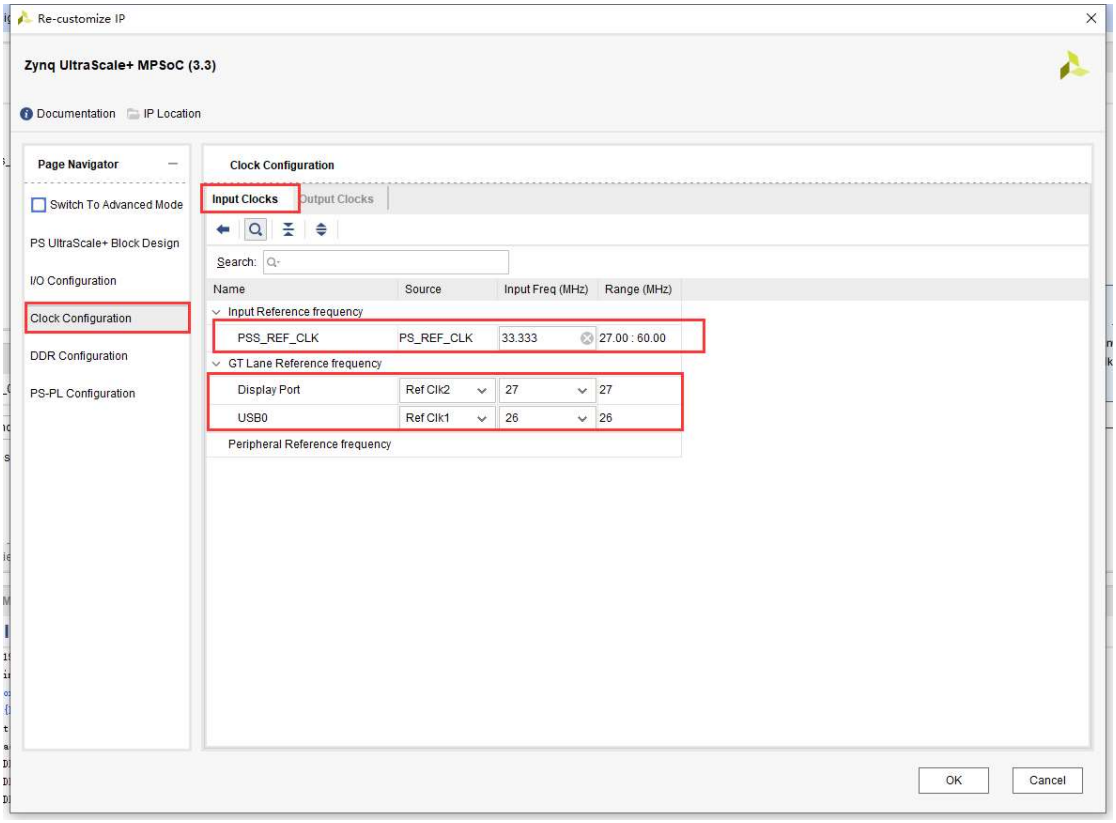


Figure 3-25 Configuration PS side input clock

The PL clock keeps the default, which is the clock provided to the PL side logic。

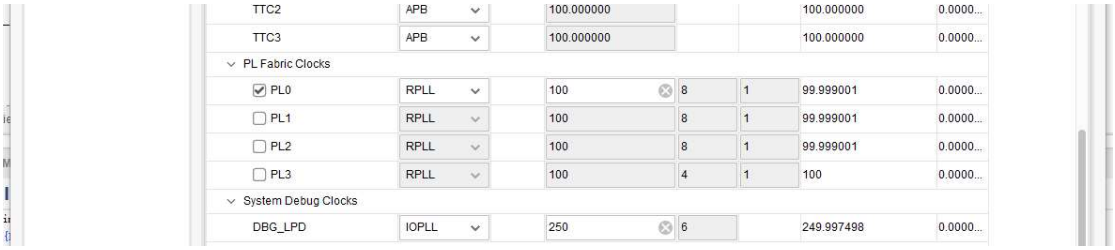


Figure 3-26 Configure to PL clock

Full Power domain，keep default except DP_VIDEO change to VPLL， DP_AUDIO and DP_STC change to RPLL。

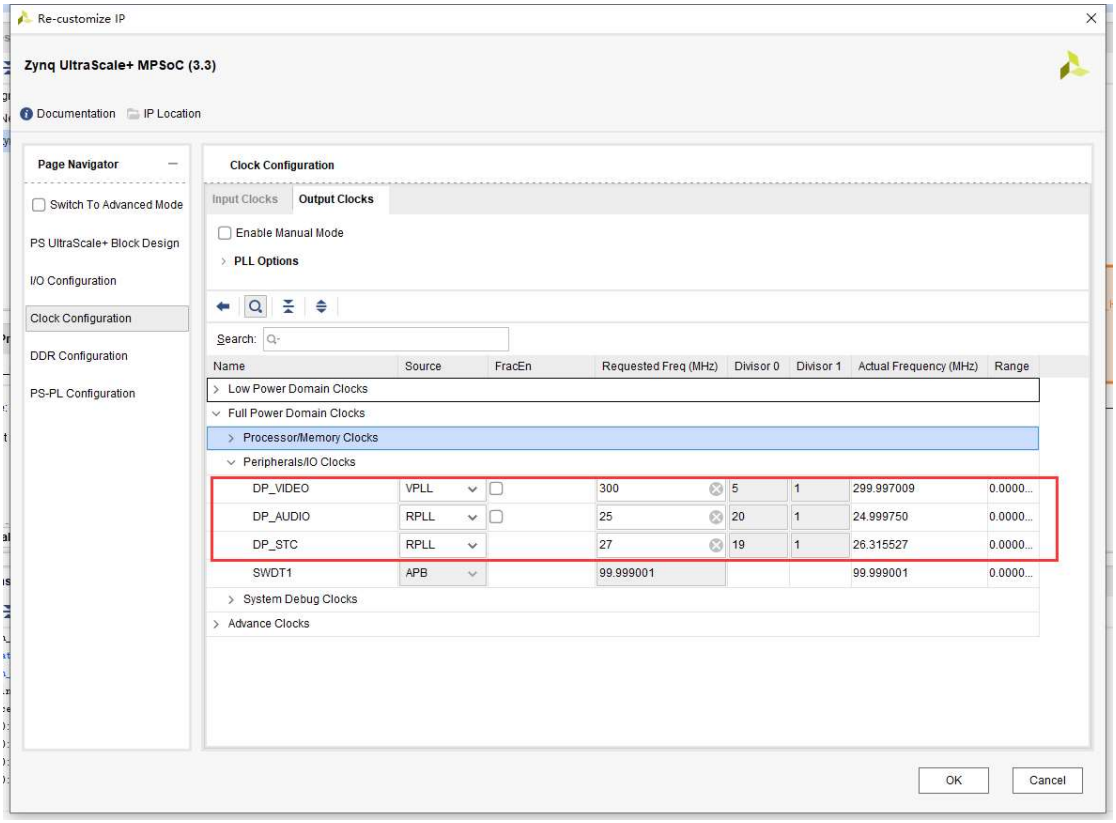


Figure 3-27 Configure Peripheral IO clock

Interconnect changed shows as below

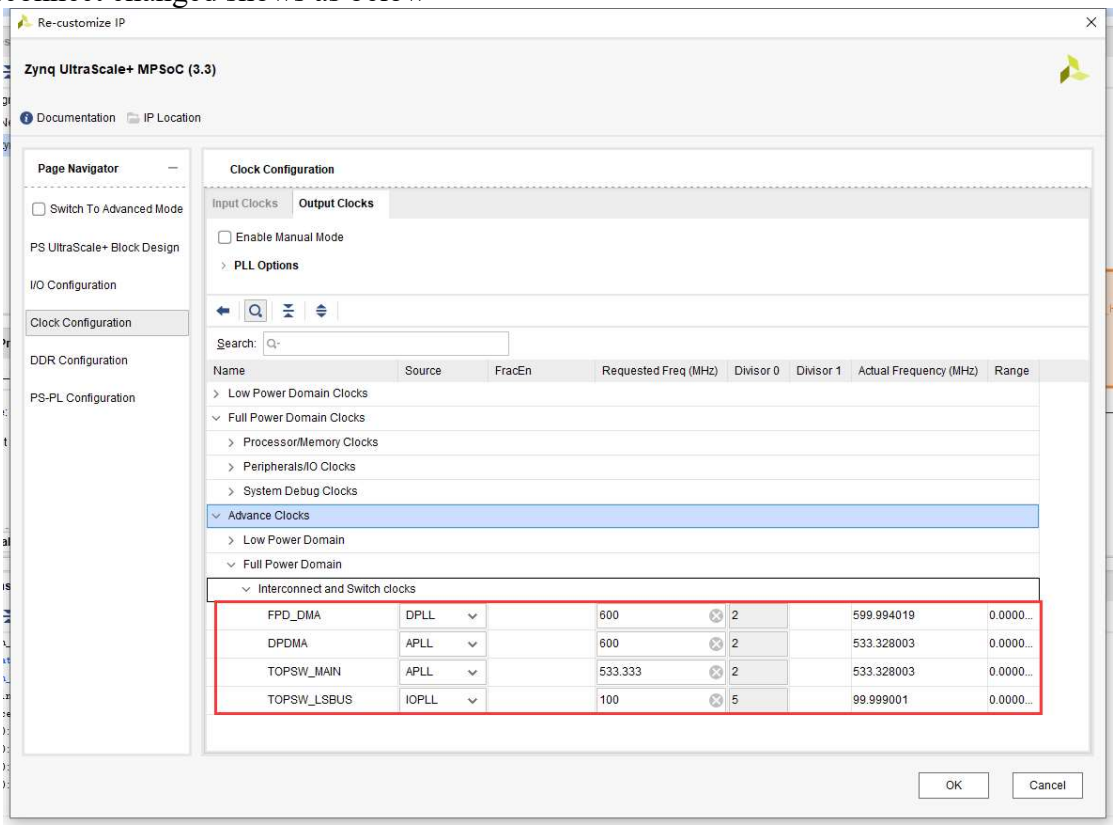


Figure 3-28 Configure the module clock that requires interconnection structure

Keep the other parts as default, so far, the clock part configuration is complete。

5、DDR configuration

In the window of DDR Configuration， Load DDR Presets select" DDR4_MICRON_MT40A256M16GE_083E"。

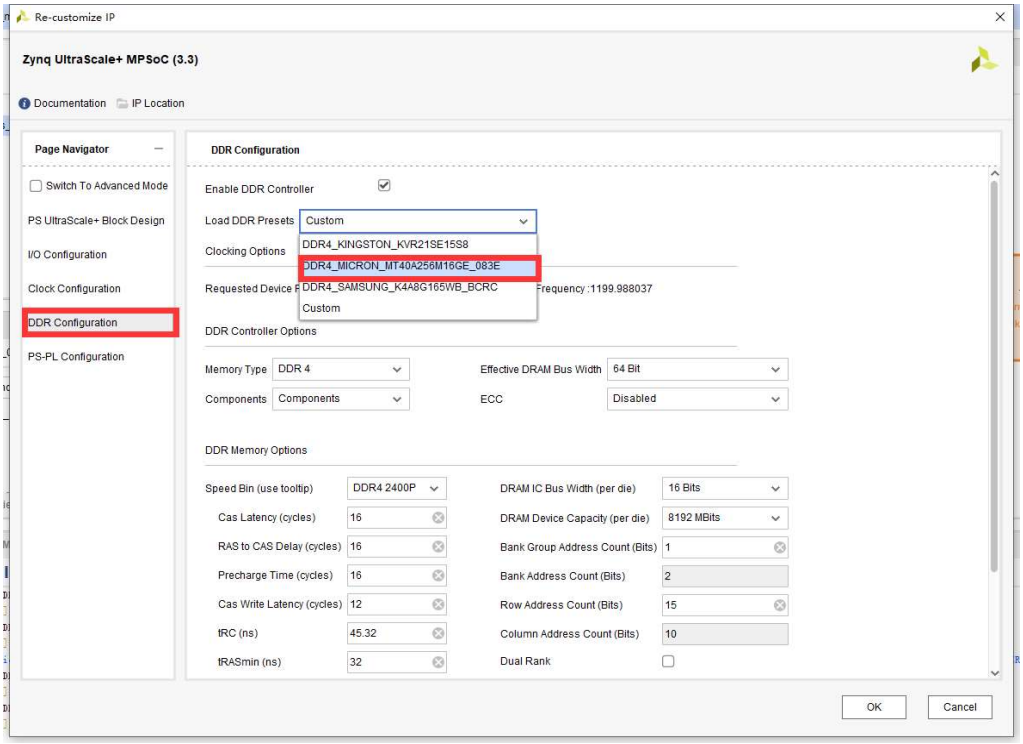


Figure 3-29 Configuration -DDR manufacturers

Some changes to the parameters are as follows:

4G memory version:

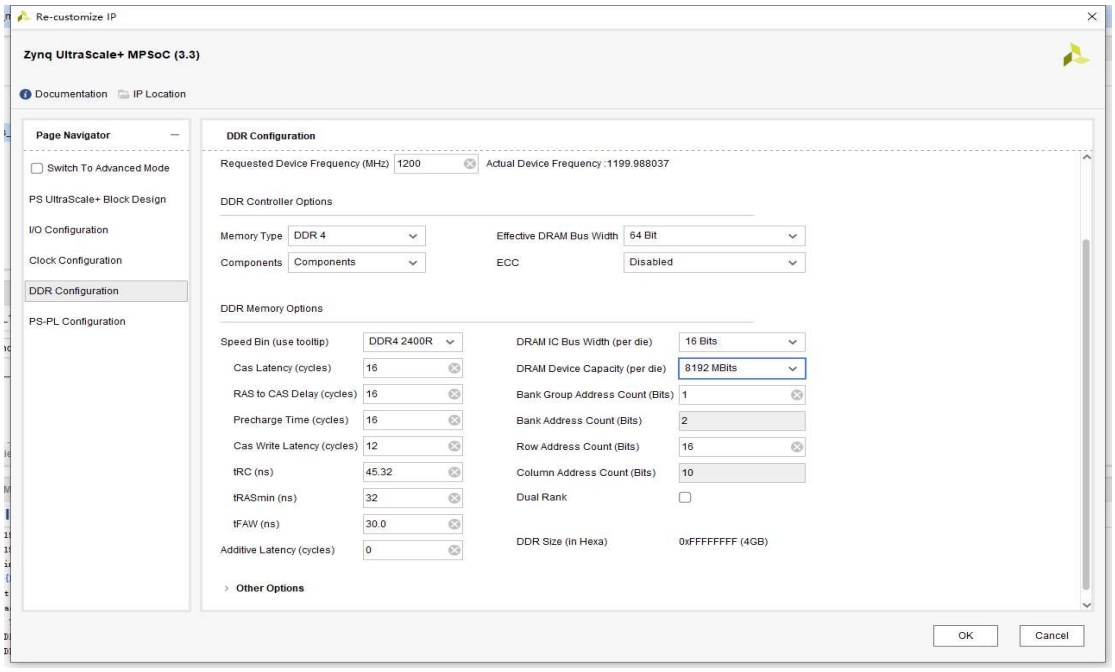


Figure 3-30 configuration of DDR width and capacity

8G memory version：

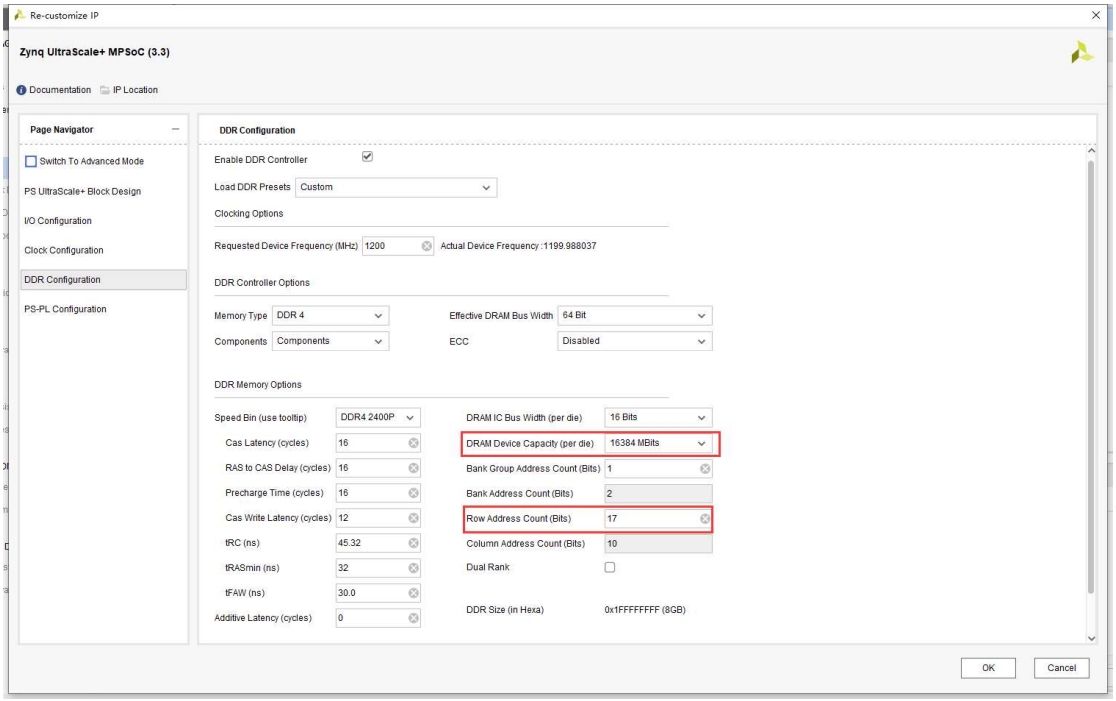


Figure 3-31 configuration DDR width and capacity

Keep other defaults, click OK, the configuration is complete, and connect the clock as follows：

Figure 3-32 After auto interconnection

3.1.3 Generate XSA File

Right click design_1.bd ->Generate Output Products->Generate

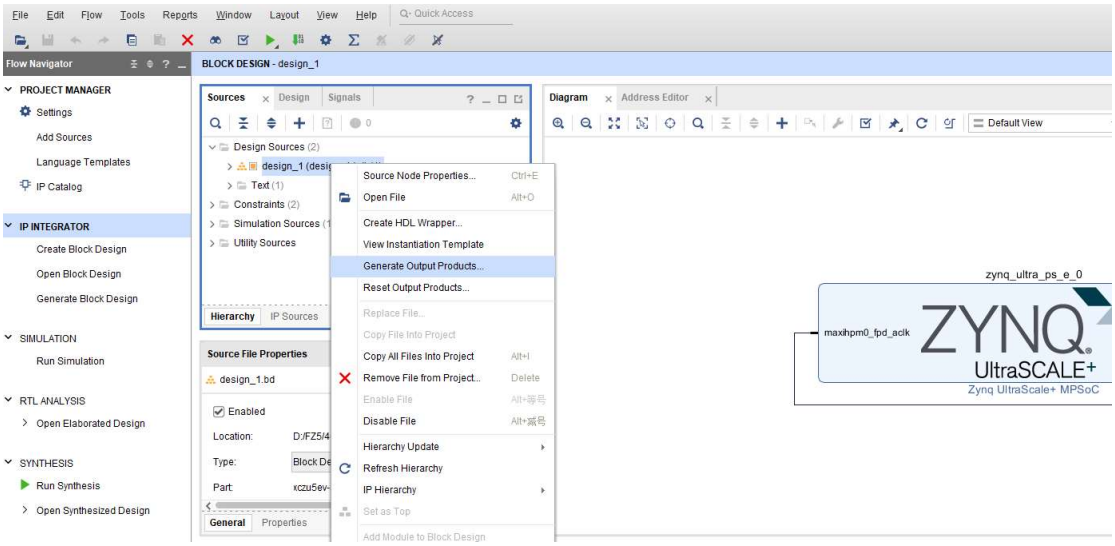


Figure 3-33 Generate output products

Click Generate

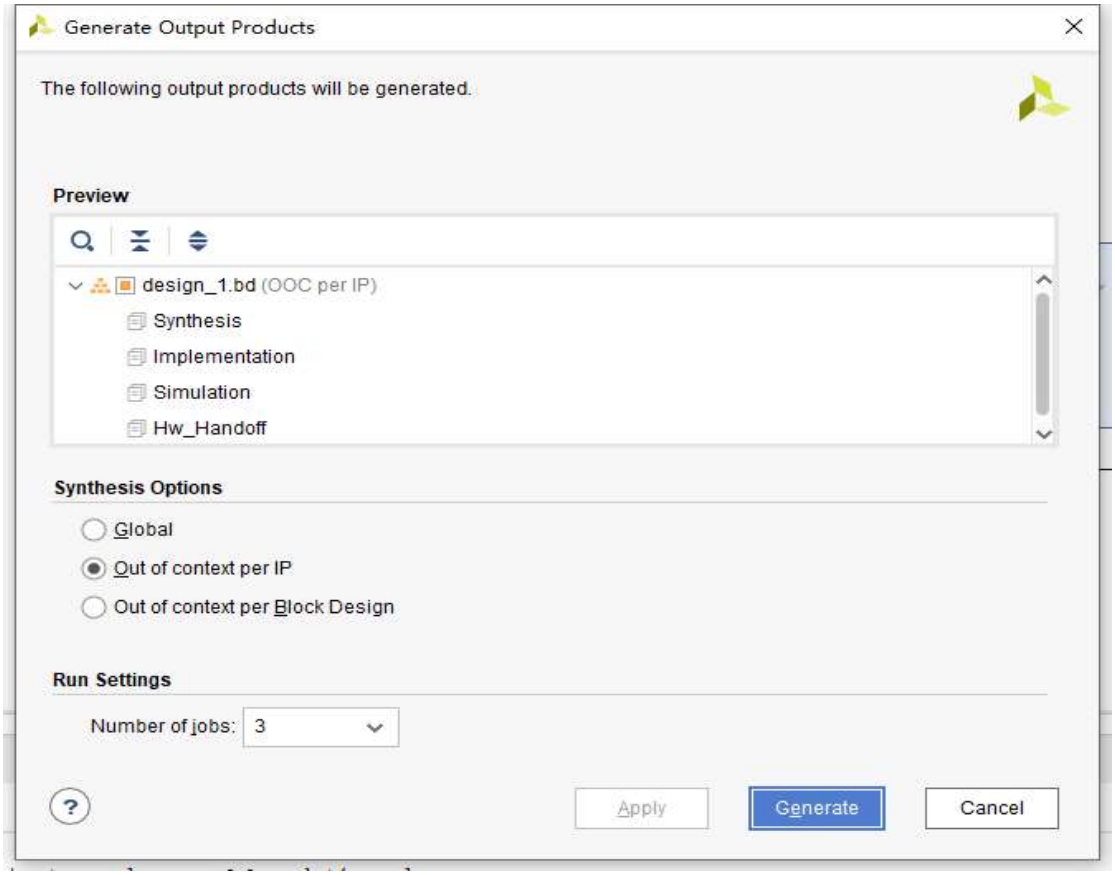


Figure 3-34 Synthesis Options Out of context per IP

Check design_1.bd,right click --CreateHDL Wrapper

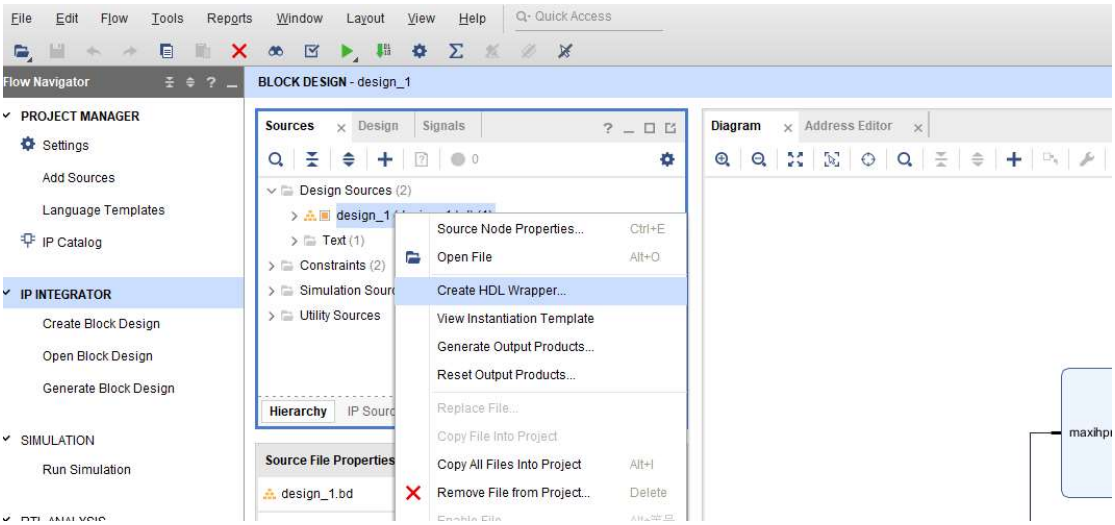


Figure 3-35 Create HDL Wrapper

Click OK

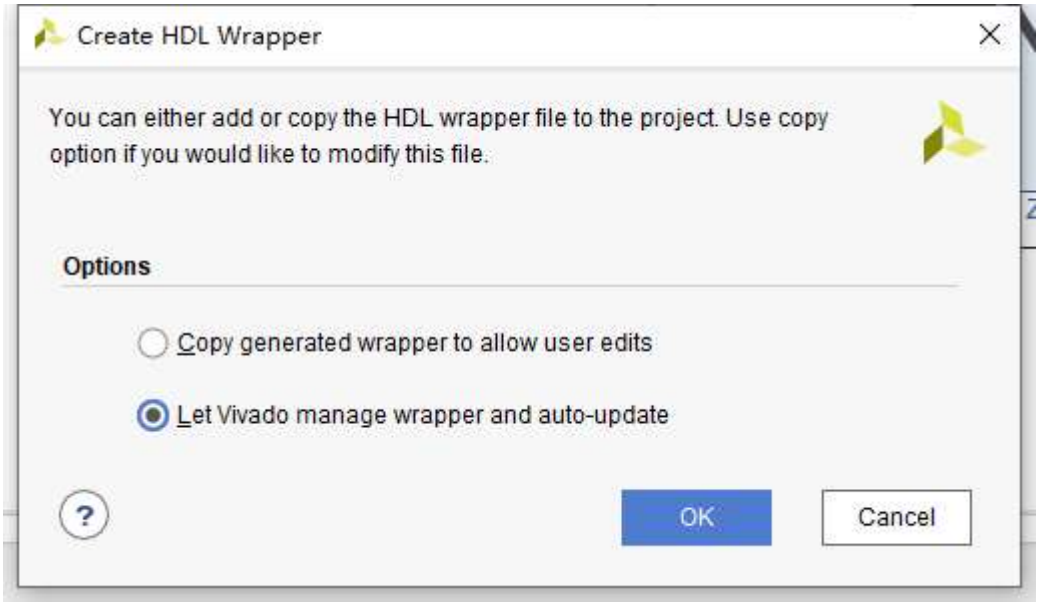


Figure 3-36 auto-update HDL Wrapper file

Click built button。

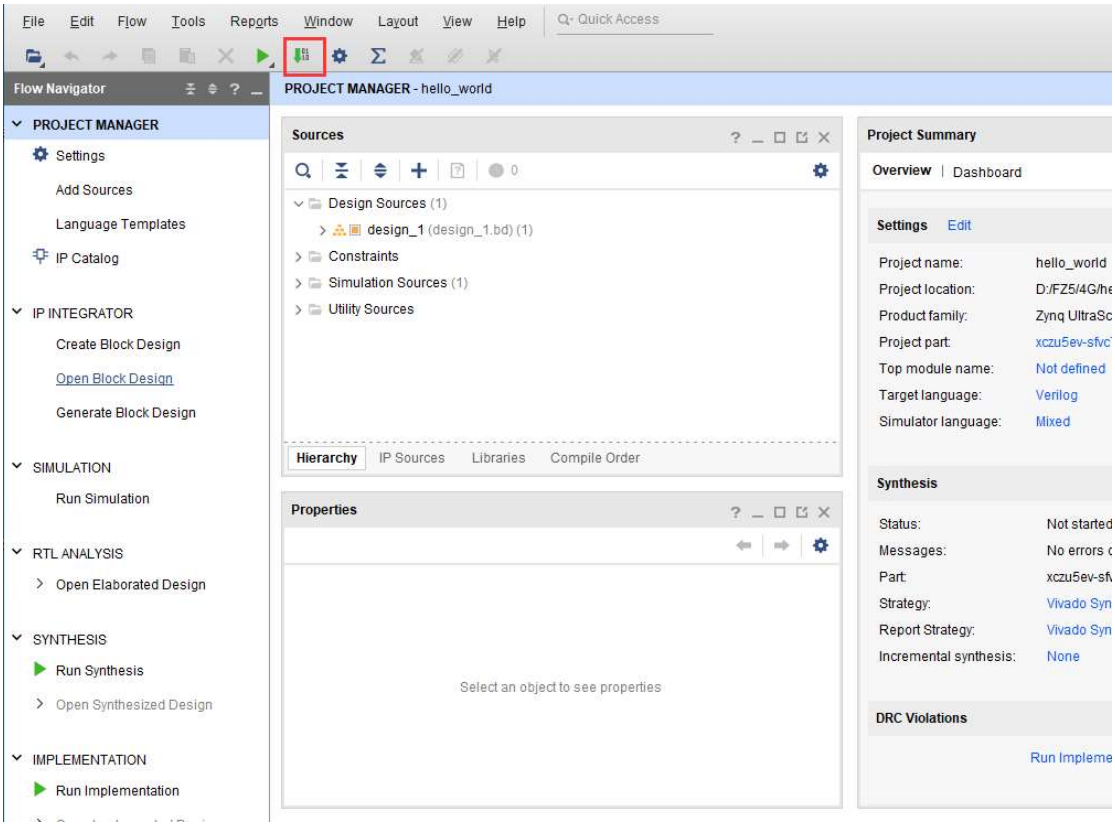


Figure 3-37 generate bitstream

Click Yes

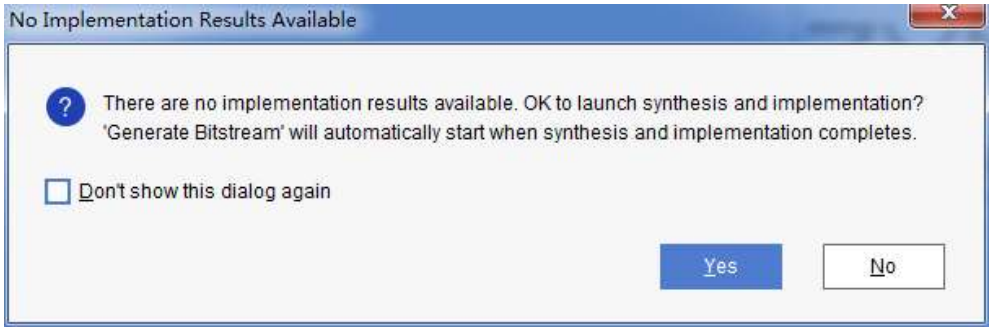


Figure 3-38 click YES

Click OK

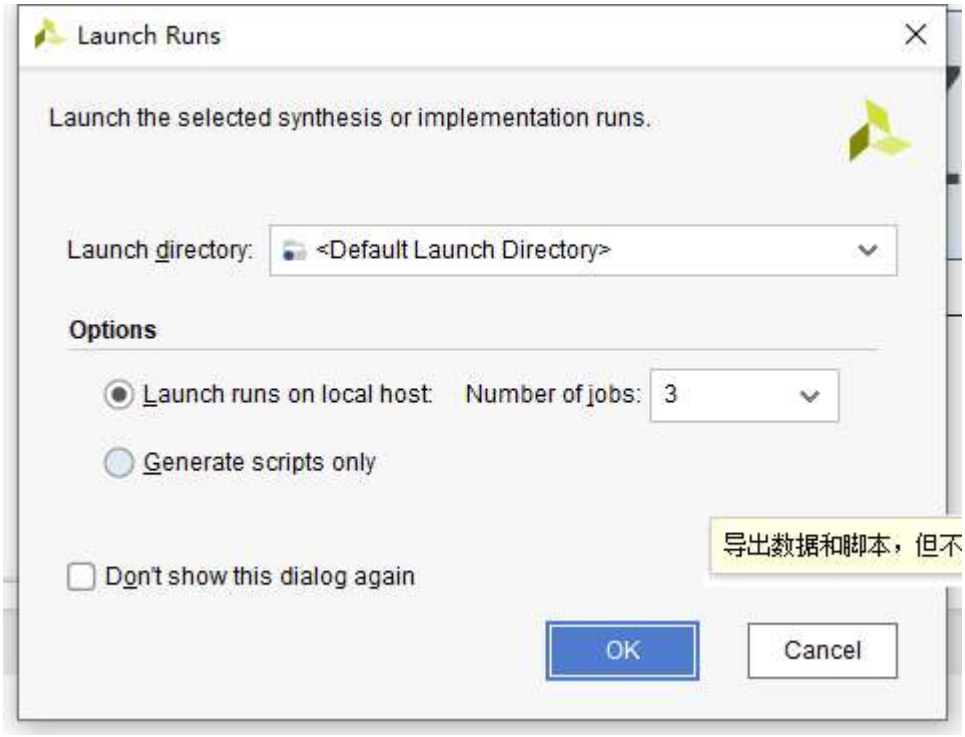


Figure 3-39 start synthesis、and implementation

Click Cancel

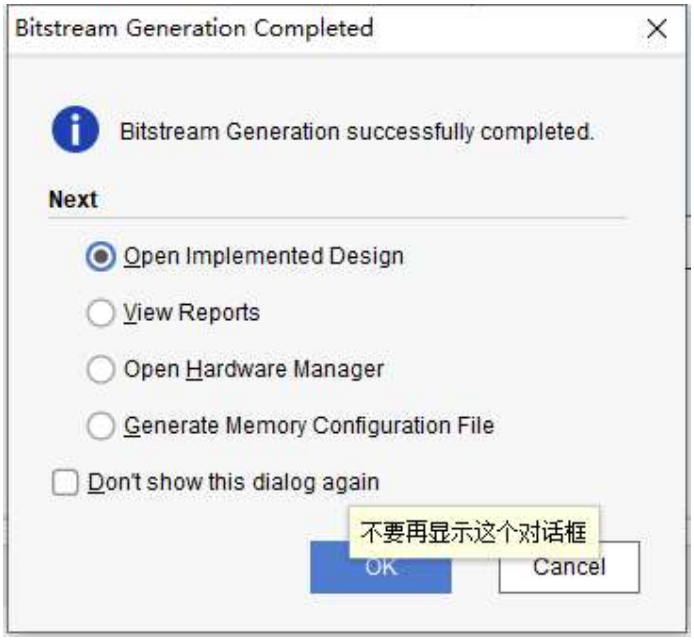


Figure 3-40 generate bitstream successfully

bitstream generate successfully.

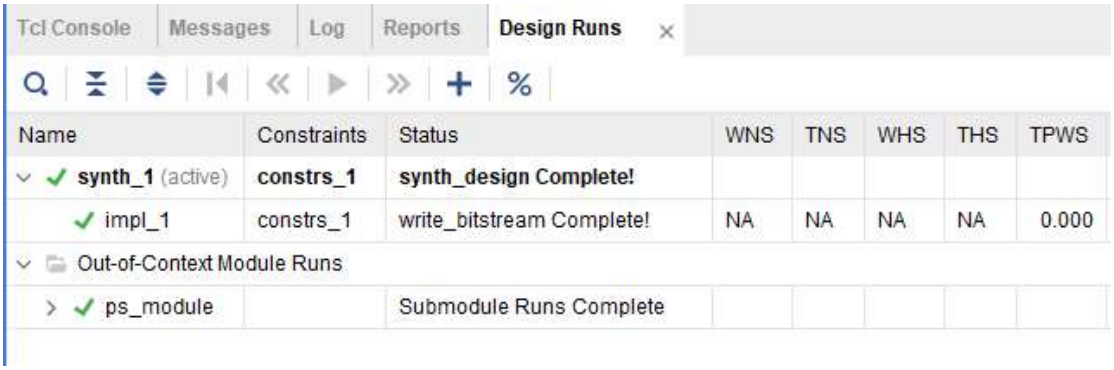


Figure 3-41 bitstream generate successfully

In the menu of File->Export->Export Hardware->OK export hardware configuration platform

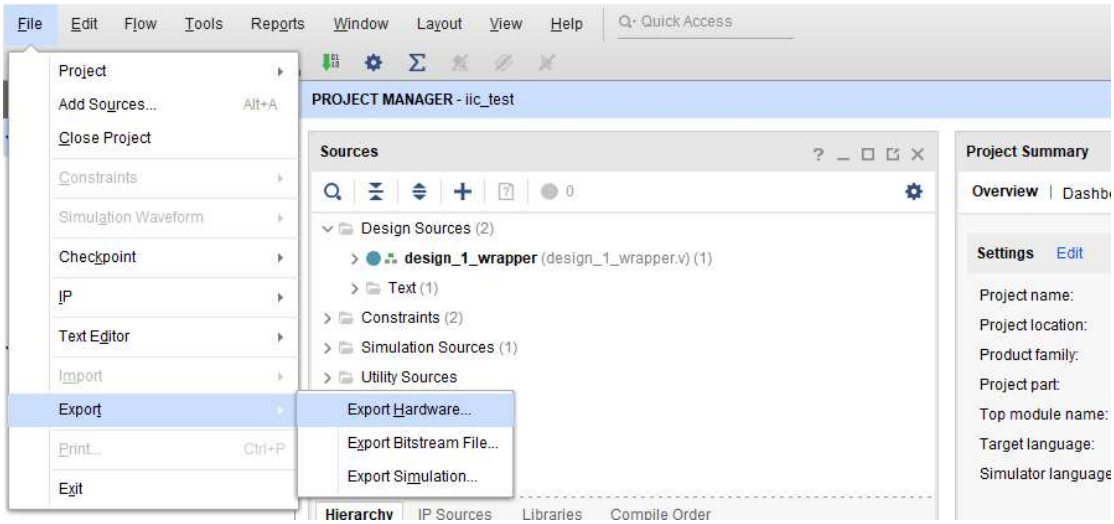


Figure 3-42 export the hardware design platform

Chose fixed，click Next

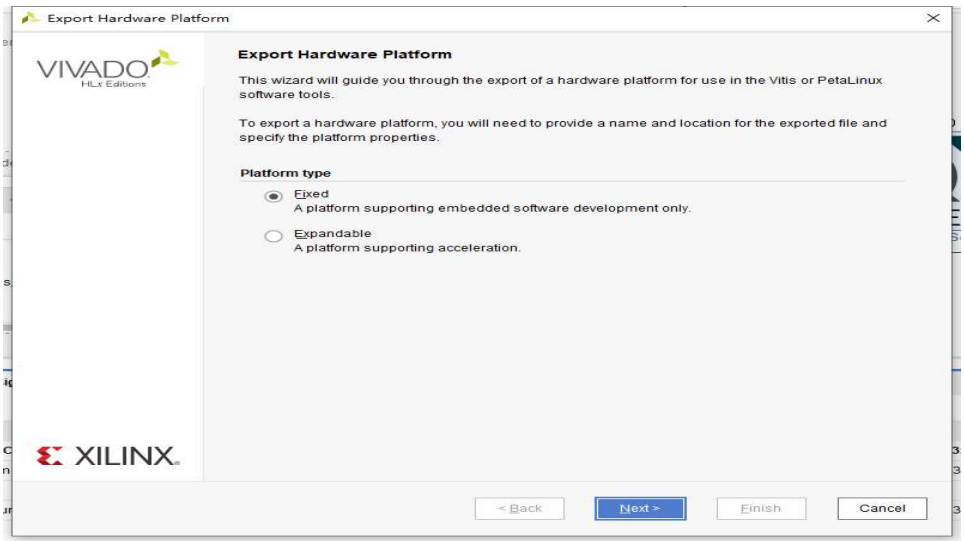


Figure 3-43 chose Fixed

Select include bitstream，click Next

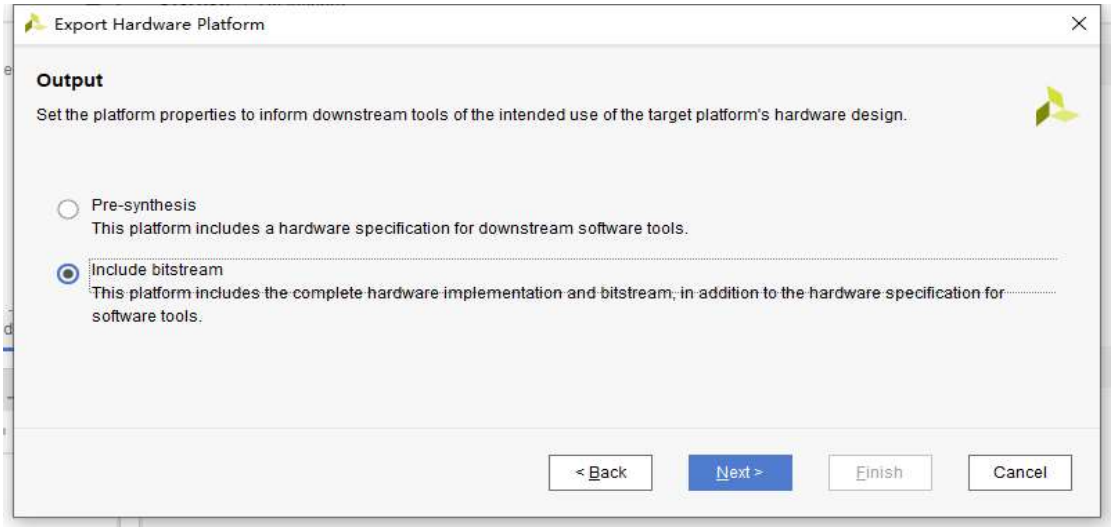


Figure 3-44 select the platform include bitstream

Select the export file name and export path. Here the file name is selected by default, and the path is the newly created vitis folder under the project file

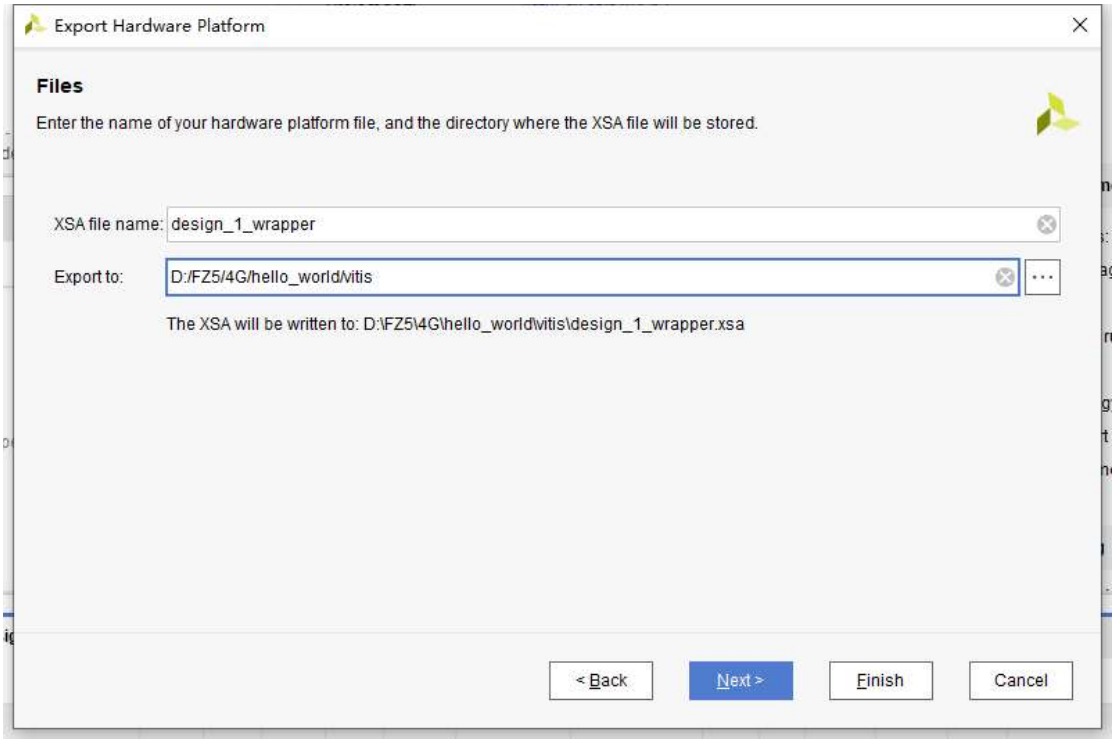


Figure 3-45 export file location

3.1.4 Built Vitis Application

Vitis is a stand-alone software, you can double-click the Vitis software to open it, or you can open the Vitis software by selecting Tools--Launch Vitis in the Vivado software

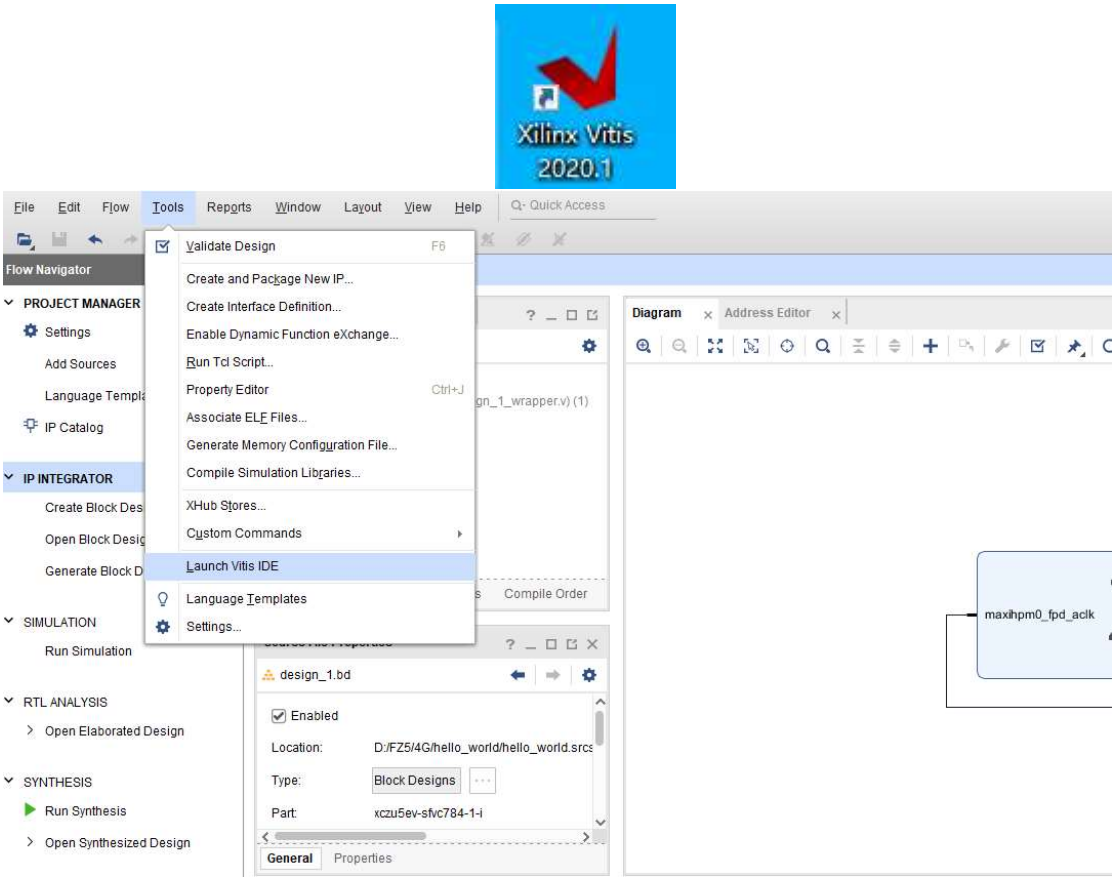


Figure 3-46 vivado to start vitis

Select the newly created vitis folder and click "Launch"

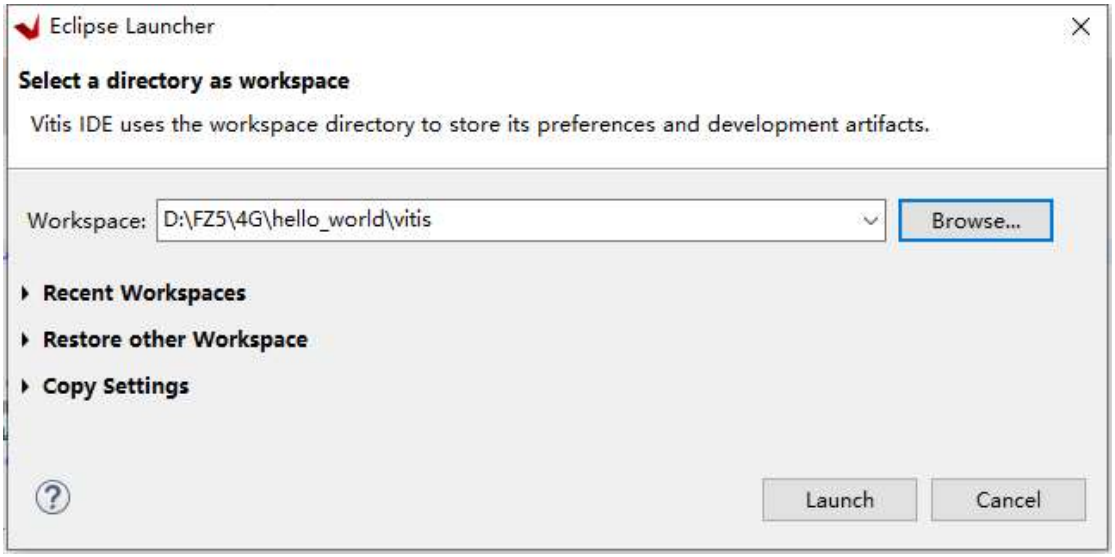


Figure 3-47 chose vitis work directory

After starting Vitis, the interface is as follows, click "Create Application Project", this option will generate APP project and Platfrom project, Platform project is similar to previous version of hardware platform, including hardware support related files and BSP。

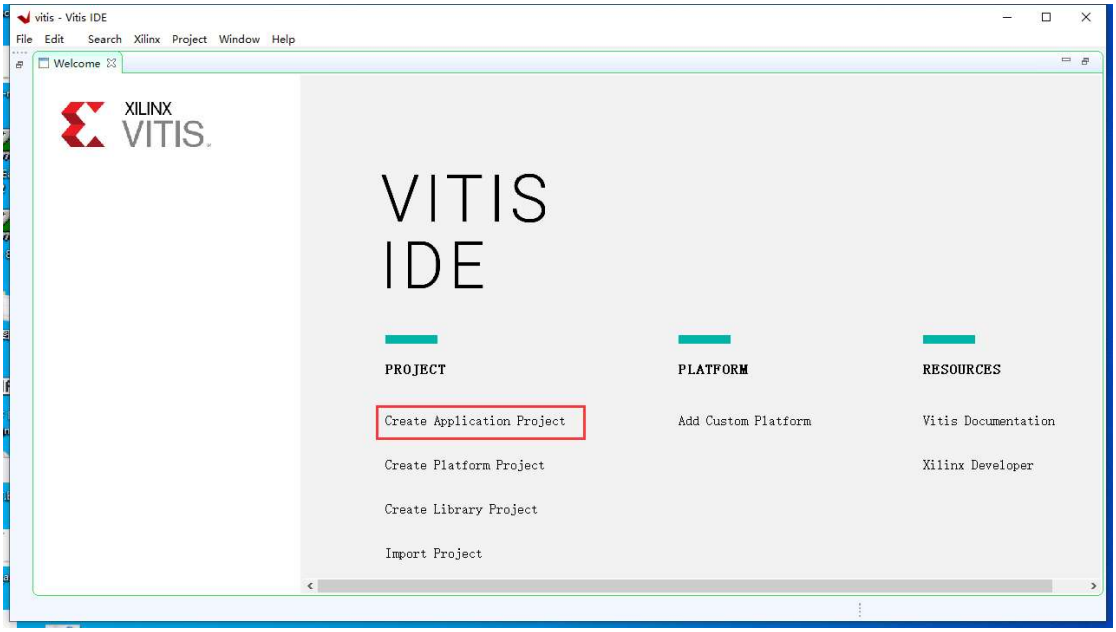


Figure 3-48 Vitis page begin

Click Next

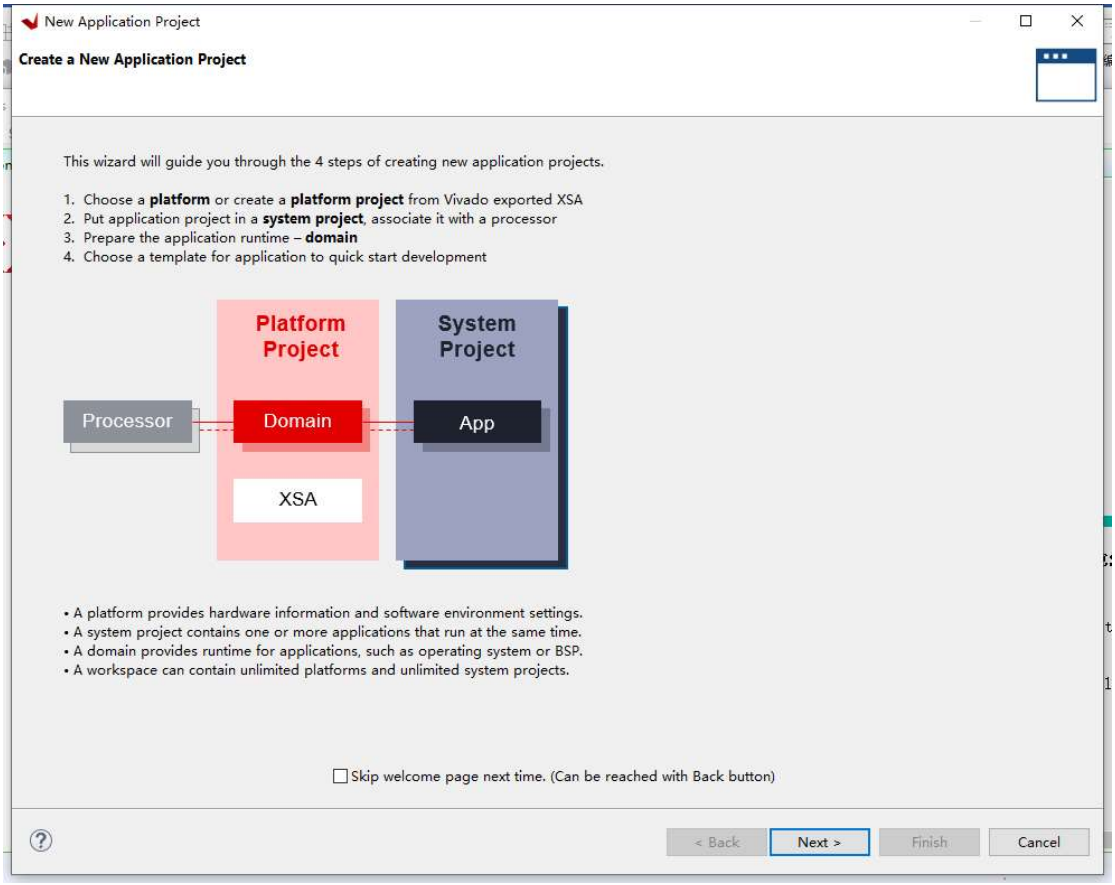


Figure 3-49 built new application

select“Create a new platform from hardware(XSA)”，click“Browse”，find the xsa file which was generated just now.

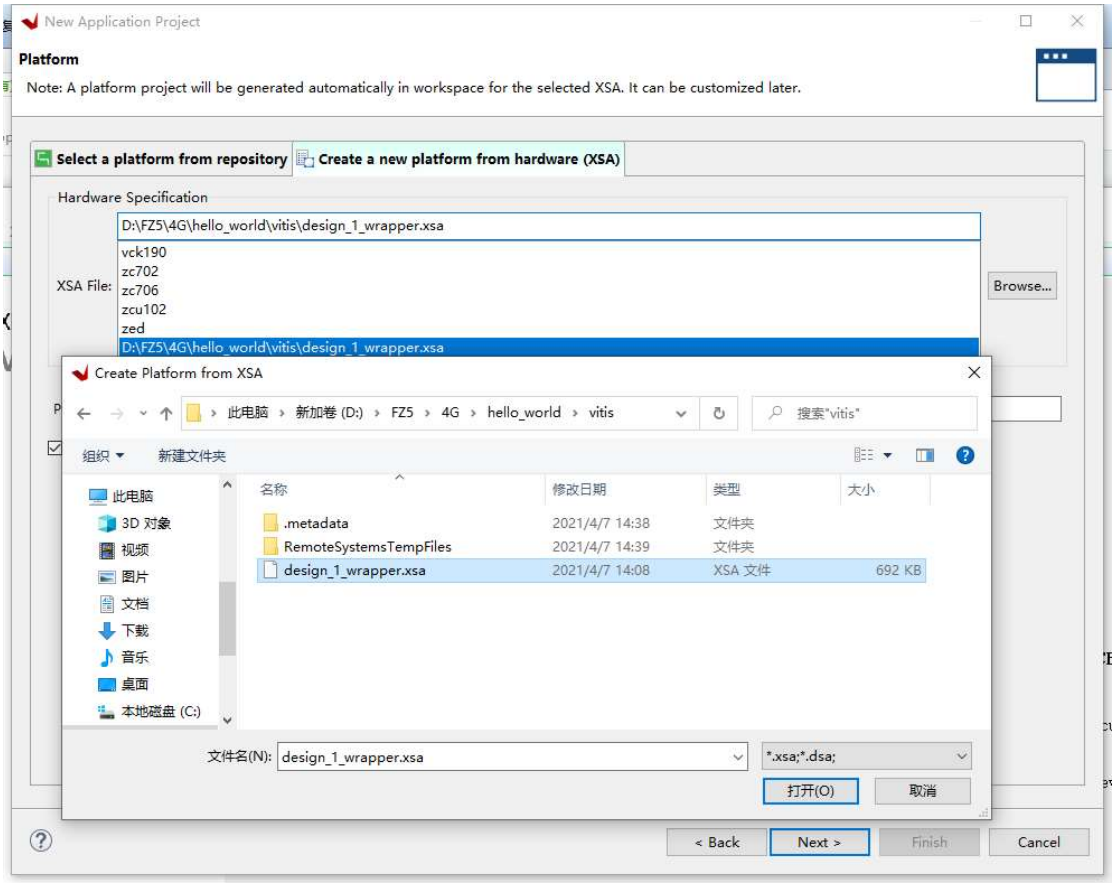


Figure 3-50 select vivado export xsa file as the platform

At the bottom of the Generate boot components option, if checked, the software will automatically generate the fsbl project. We generally choose to check it by default. Click next.

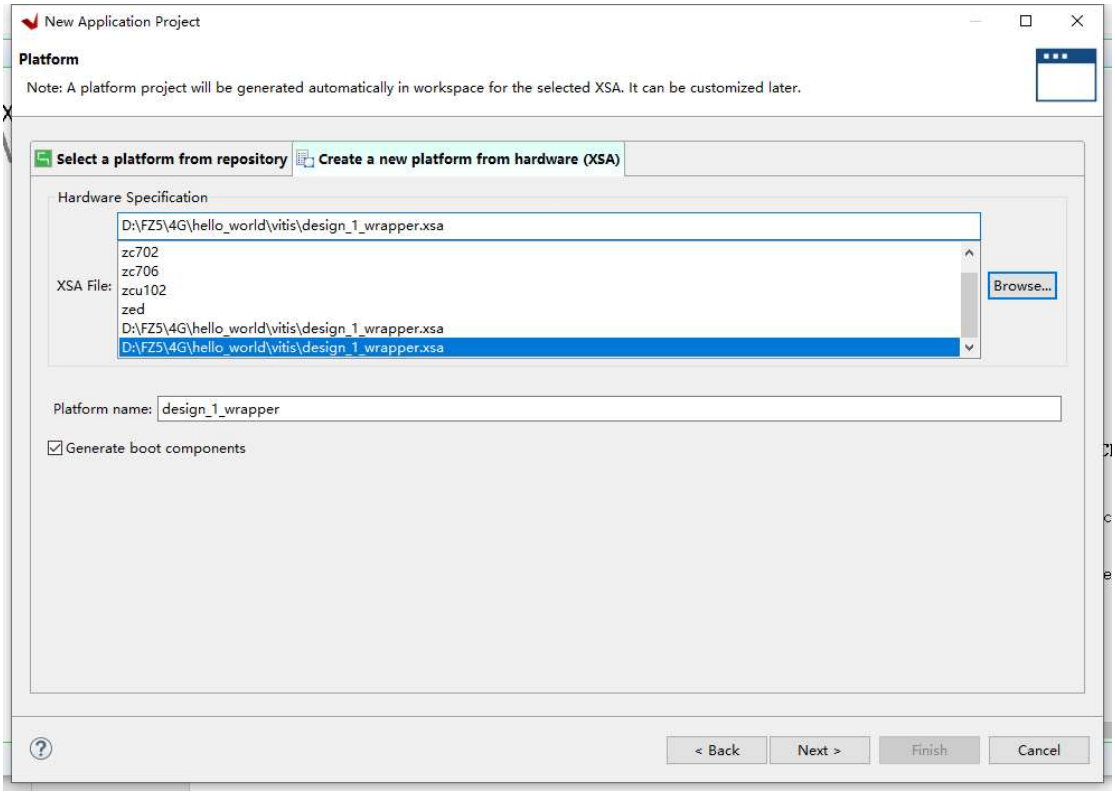


Figure 3-51 select XSA file

Fill in the APP project name hello_world, click on the box to select the corresponding processor, we keep the default here (note that the app project name and platform name cannot be the same, otherwise an error will occur)

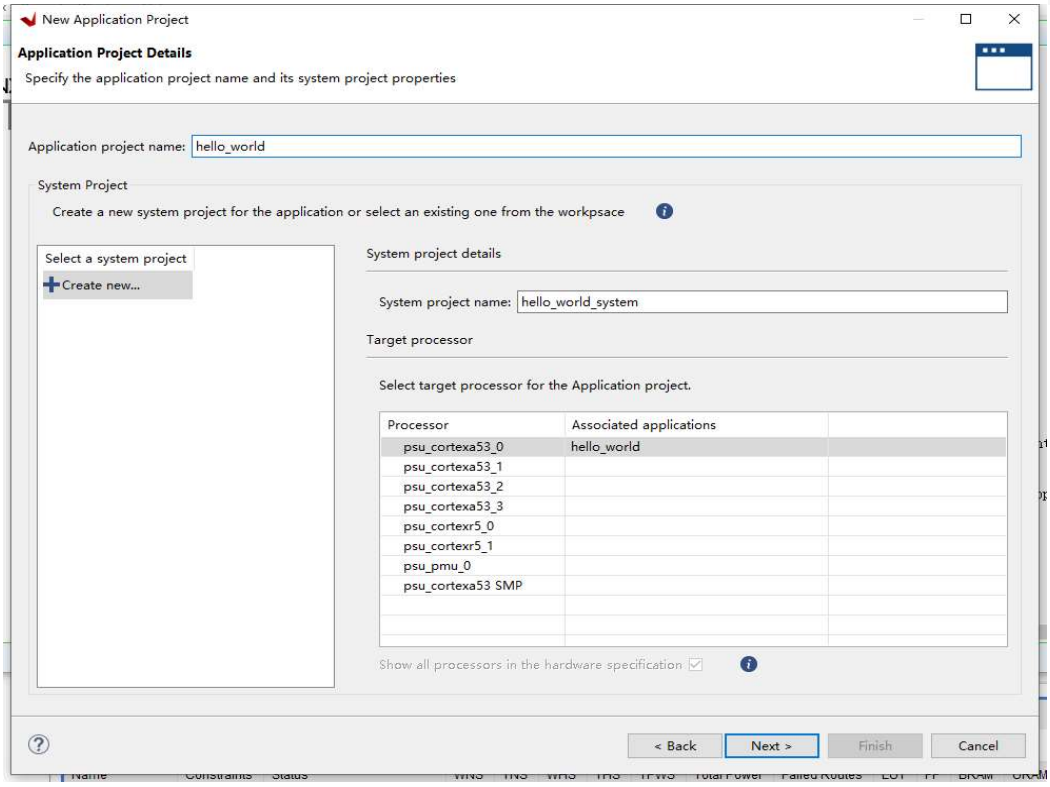


Figure 3-52 vitis application name

In this interface, you can modify the Domain name, select the operating system, ARM architecture, etc., keep the default here, and select standalone as the operating system, which is bare metal.

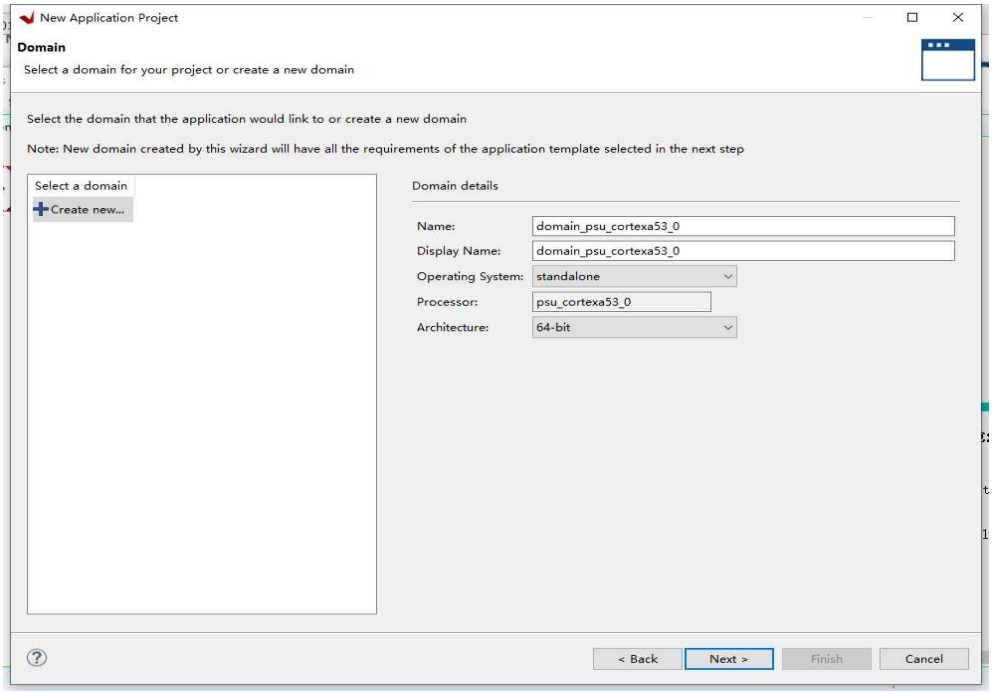


Figure 3-53 select OS for CPU

Select ″ Hellow World″ template，click ″Finish″

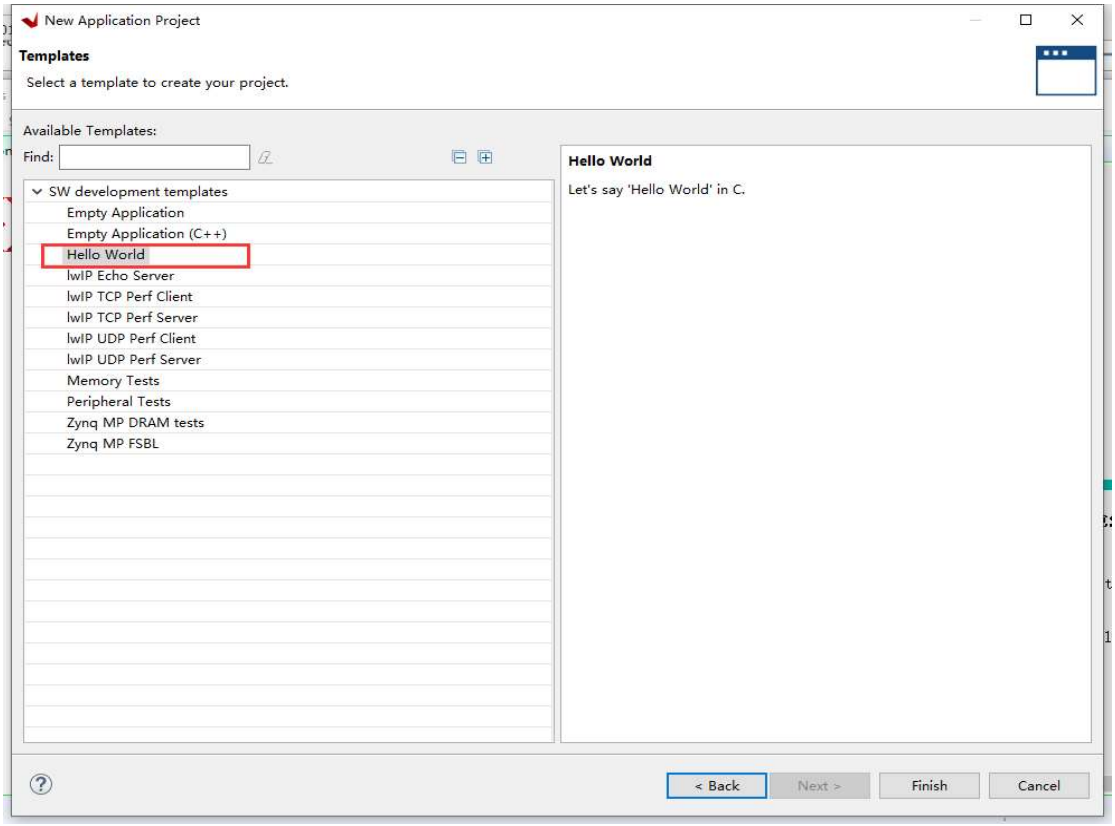


Figure 3-54 select application template

After the completion, you can see that two projects have been generated, one is the hardware platform project, which is the Platfrom project mentioned before, and the other is the APP project

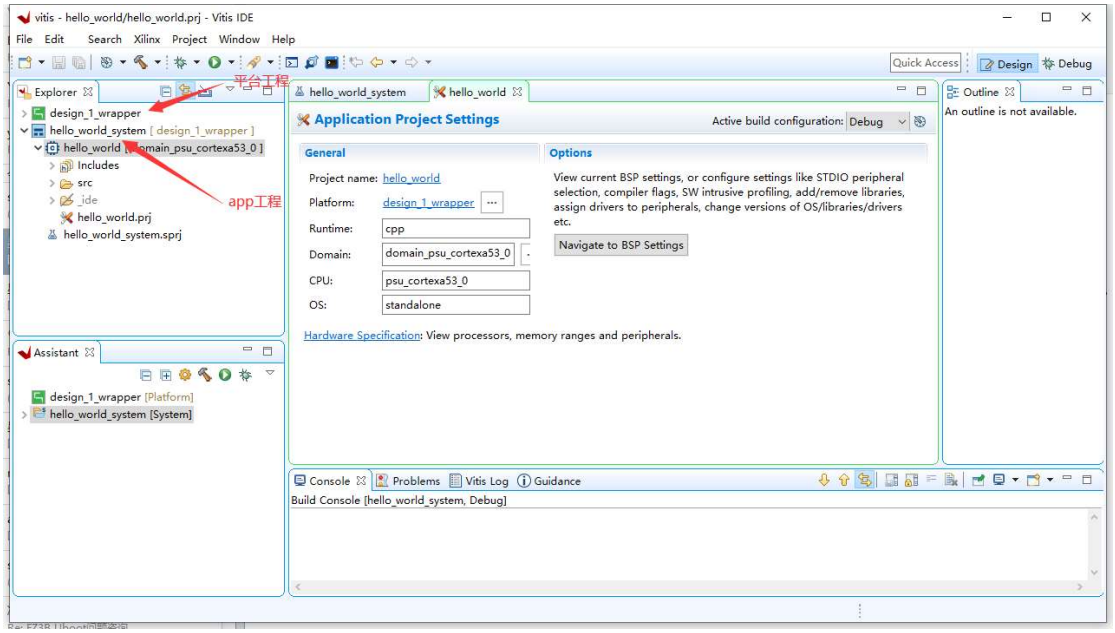


Figure 3-55 application

After unfolding the Platform project, you can see that it contains the BSP project and the zynq_fsbl project (this project is the result of selecting Generate boot components). Double-click platform.spr to see the corresponding BSP project generated by the Platform. You can perform the BSP project here. Configuration. BSP is also the meaning of Board Support Package, which contains the driver files needed for development and is used for application development. You can see that there are multiple BSPs under the Platform, which is different from the previous SDK software. Among them, zynqmp_fsbl is the BSP of fsbl, and domain_psu_cortexa53_0 is the BSP of the APP project.

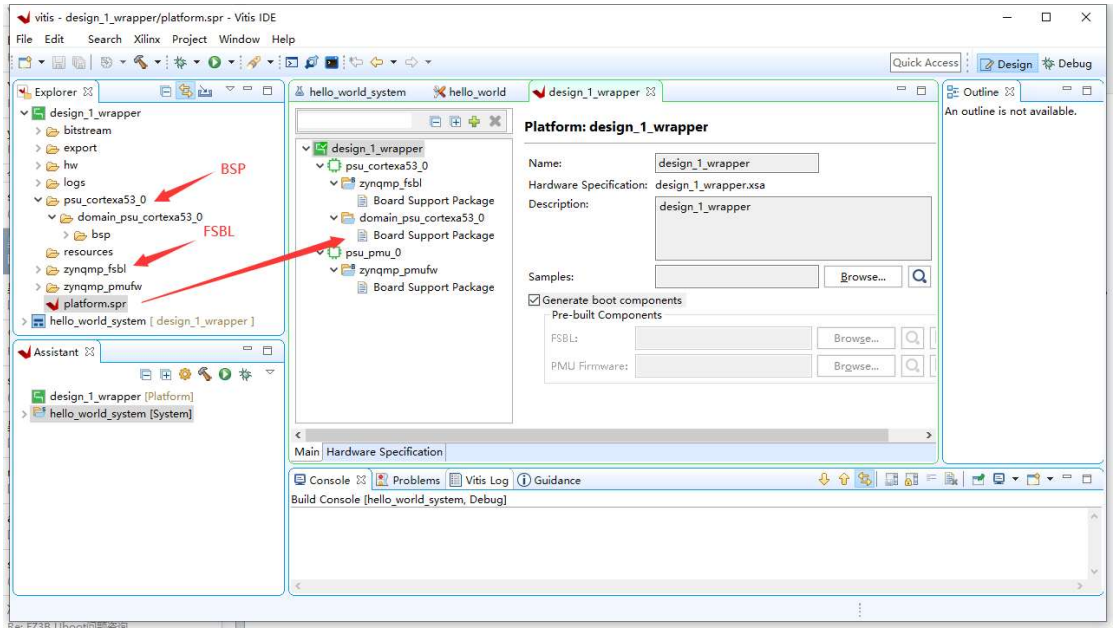


Figure 3-56 BSP、FSBL file folder

Click on the BSP, you can see the peripheral drivers of the project, where Documentation is the driver documentation provided by xilinx, and Import Examples is the example project provided by xilinx to speed up learning。

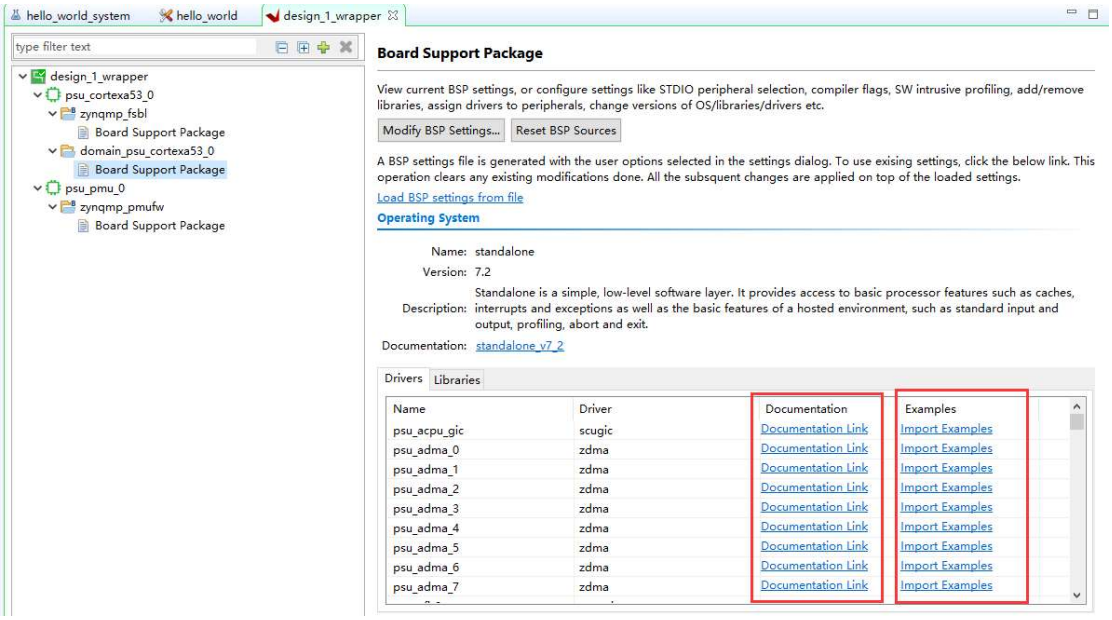


Figure 3-57 BSP example

In the APP project, right-click Build Project, or click the "hammer" button in the menu bar to compile the project

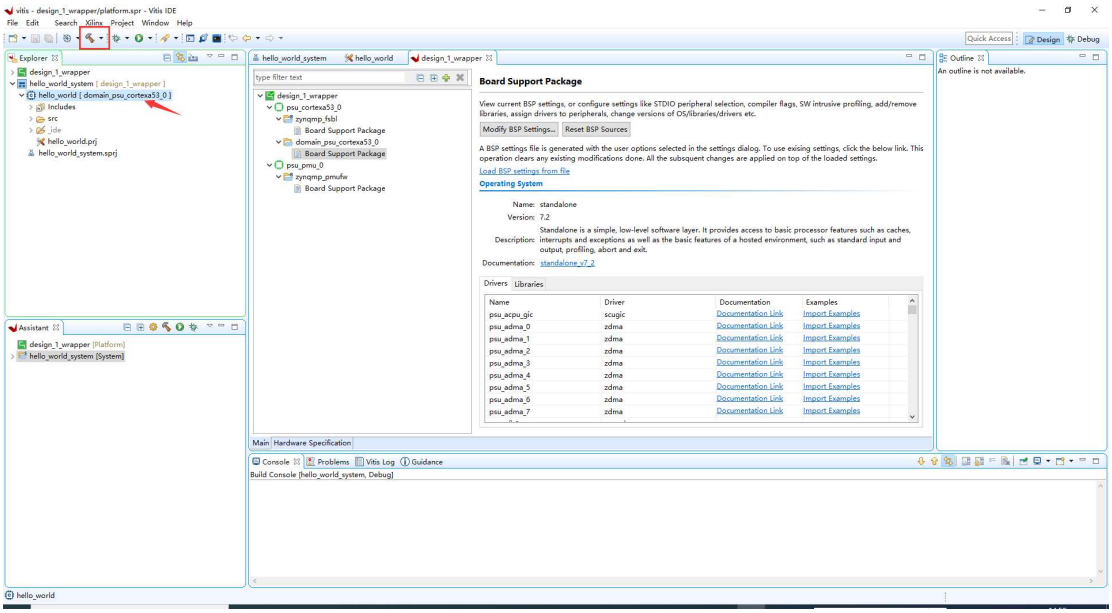


Figure 3-58 built

After built，generate elf folder

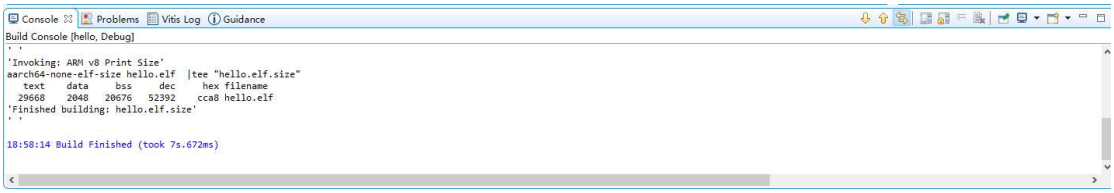


Figure 3-59 elf folder generate console window

3.1.5 Debug

Connect the JTAG line to the development board, and the UART USB line to the PC.

Use PuTTY software as a serial terminal debugging tool

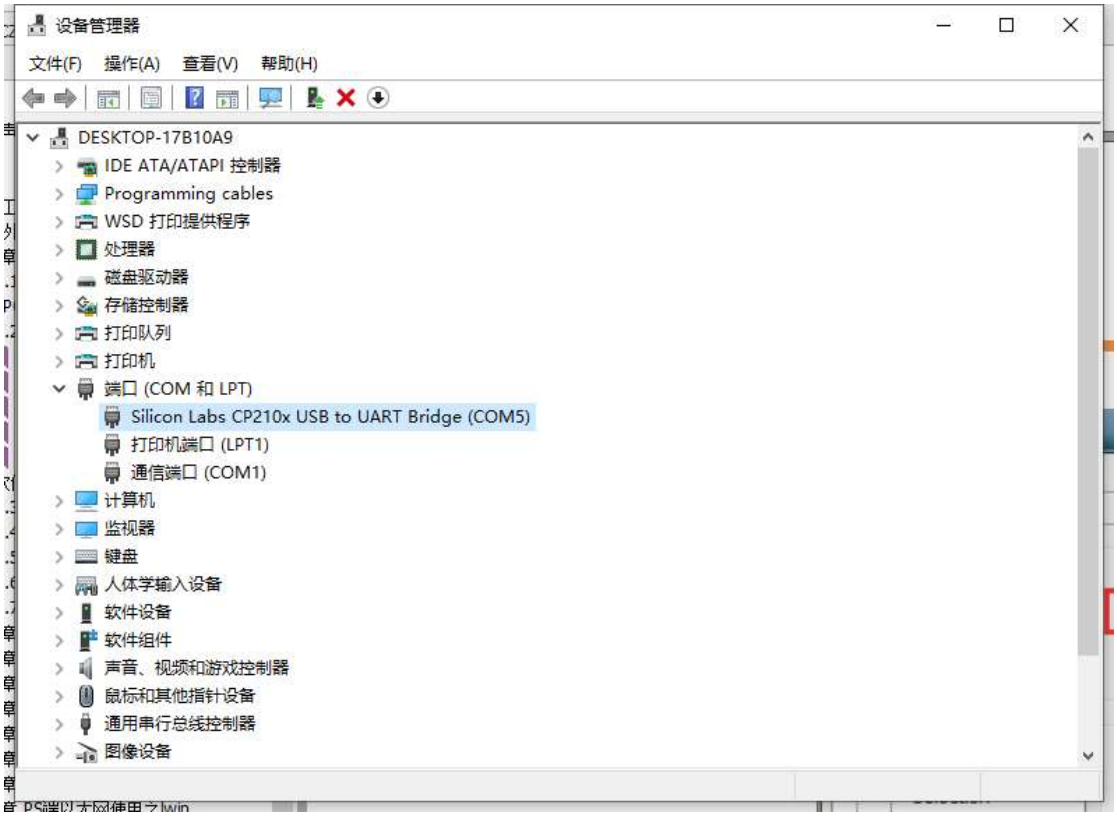


Figure 3-60 PC resource manager

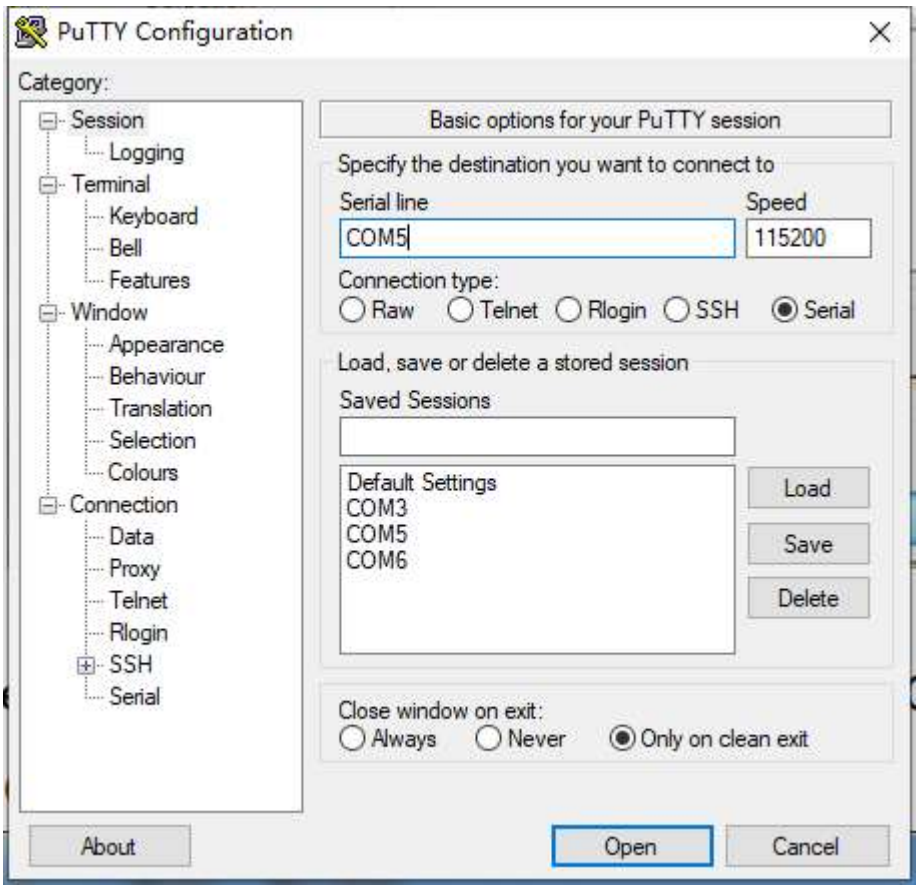


Figure 3-61 PuTTY configuration

Select "hello", right click, and you can see many options. The "Run as" that is used in this experiment is to run the program. There are many options in "Run as". Select the first "Launch onHardware(Single Application Debug)", use system debugging to run the program directly

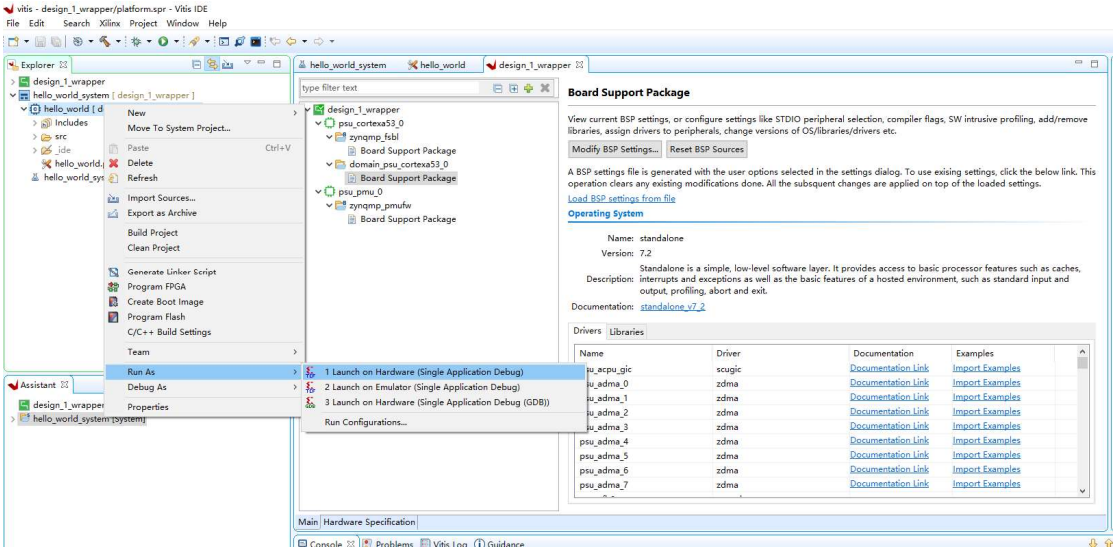


Figure 3-62 download and debug

In order to ensure the reliable debugging of the system, it is best to right-click "Run As -> Run Configuration..."

We can take a look at the configuration inside, where Reset entire system is selected by default, which is different from the previous SDK software. If there is still a PL design in the system, you must also select "Program FPGA"

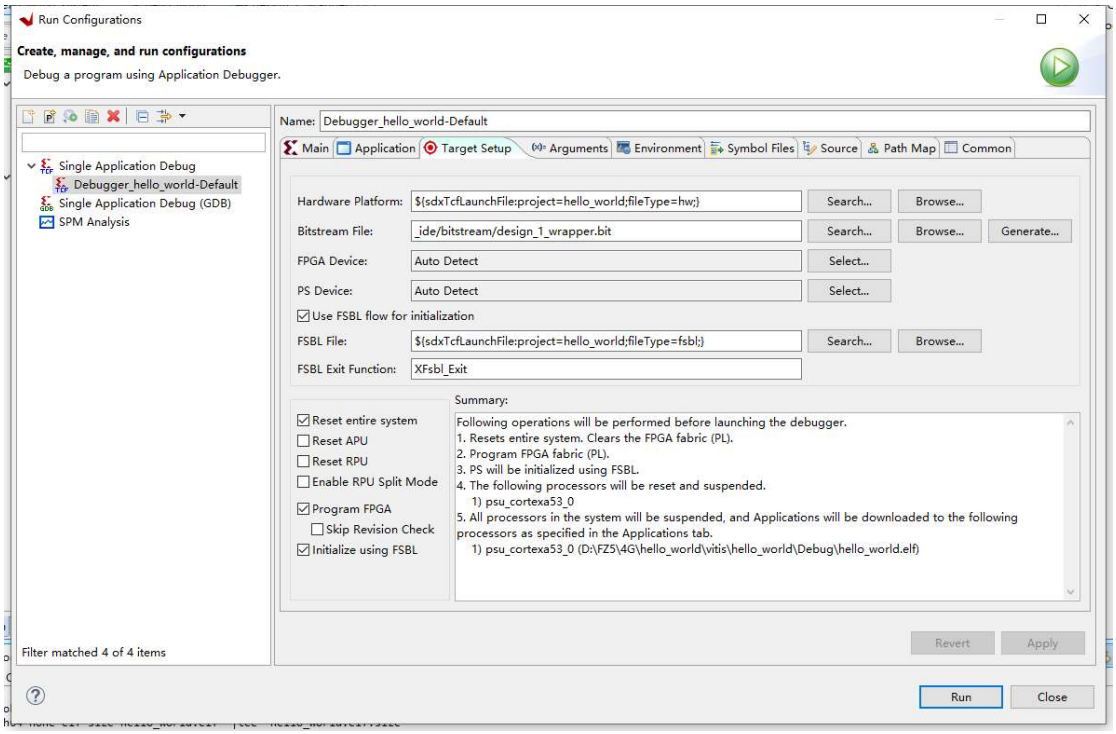


Figure 3-63 debug

3.1.6Serial Print Output“Hello World”

The serial output print result is as follows。

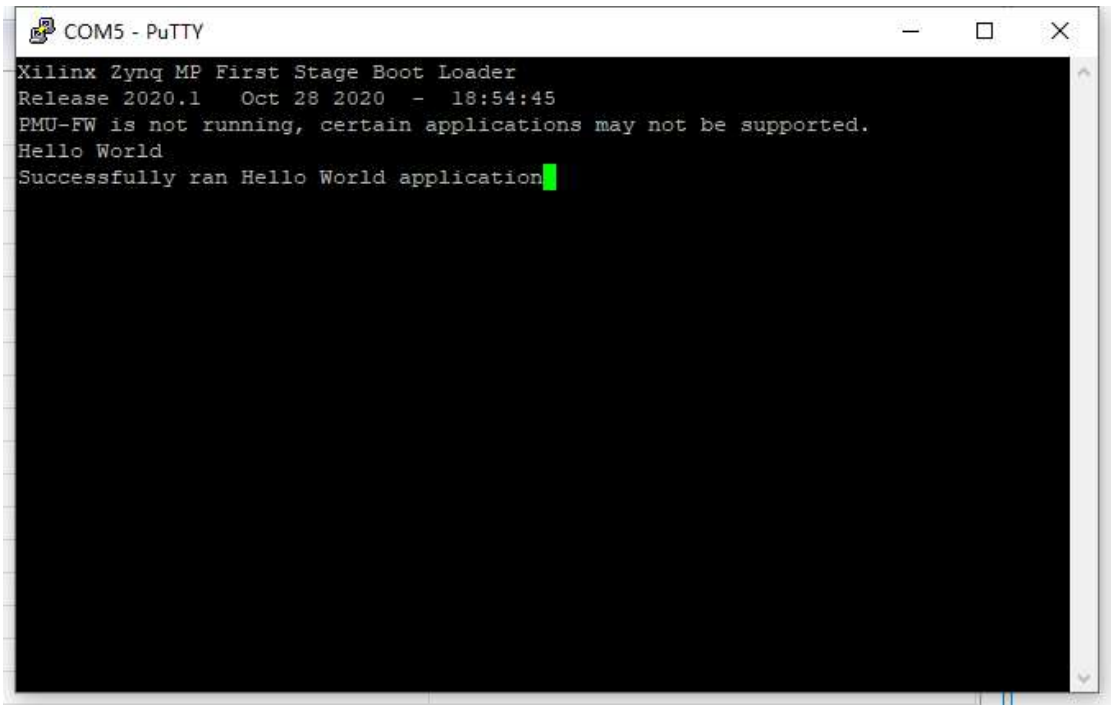


Figure 3-64 Serial output result

3.1.7Program Curing

Since the Generate boot components option was selected when creating a new one, Platform has imported the fsbl project and generated the corresponding elf file。

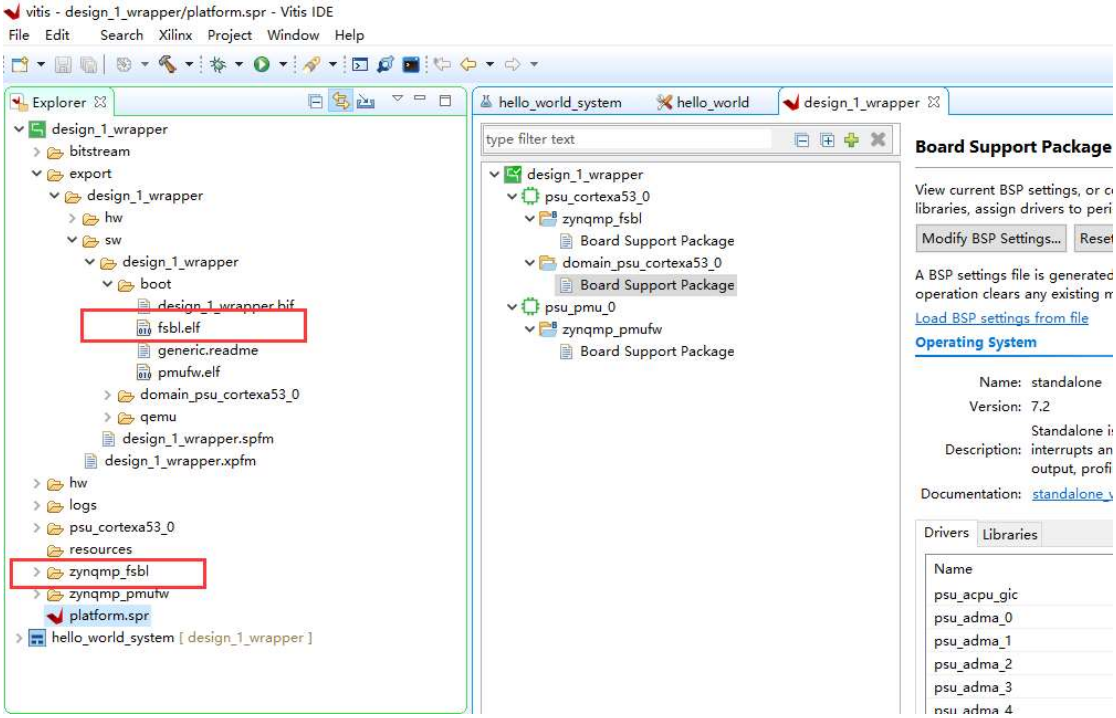


Figure 3-65 Generate fsbl.elf

Modify the debug macro definition FSBL_DEBUG_INFO_VAL to 1, which can output some status information of FSBL at startup, which is conducive to debugging, but it will cause the startup time to become longer.

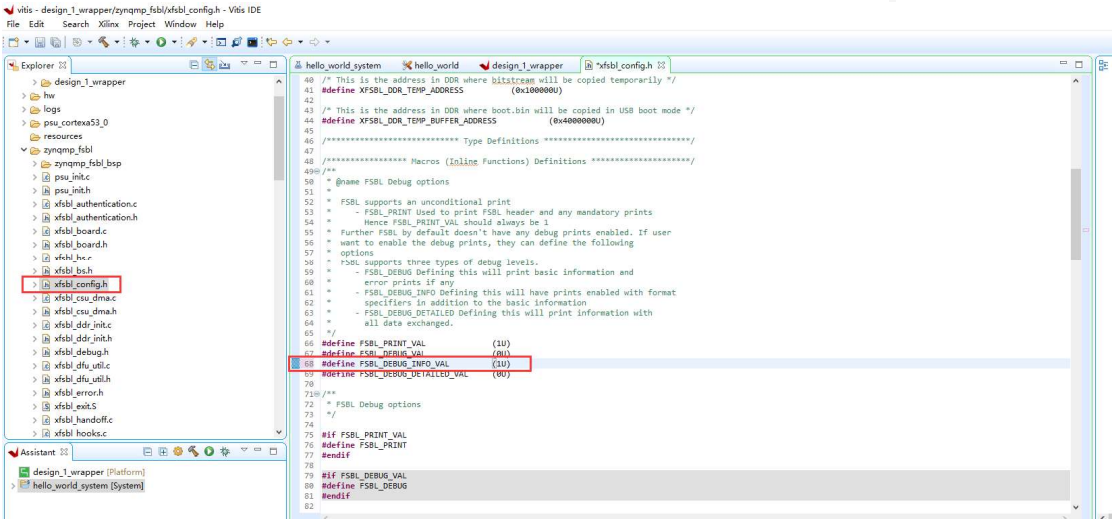


Figure 3-66 Modify macro definition

Right-click the hardware platform project hello_world and select Build Project

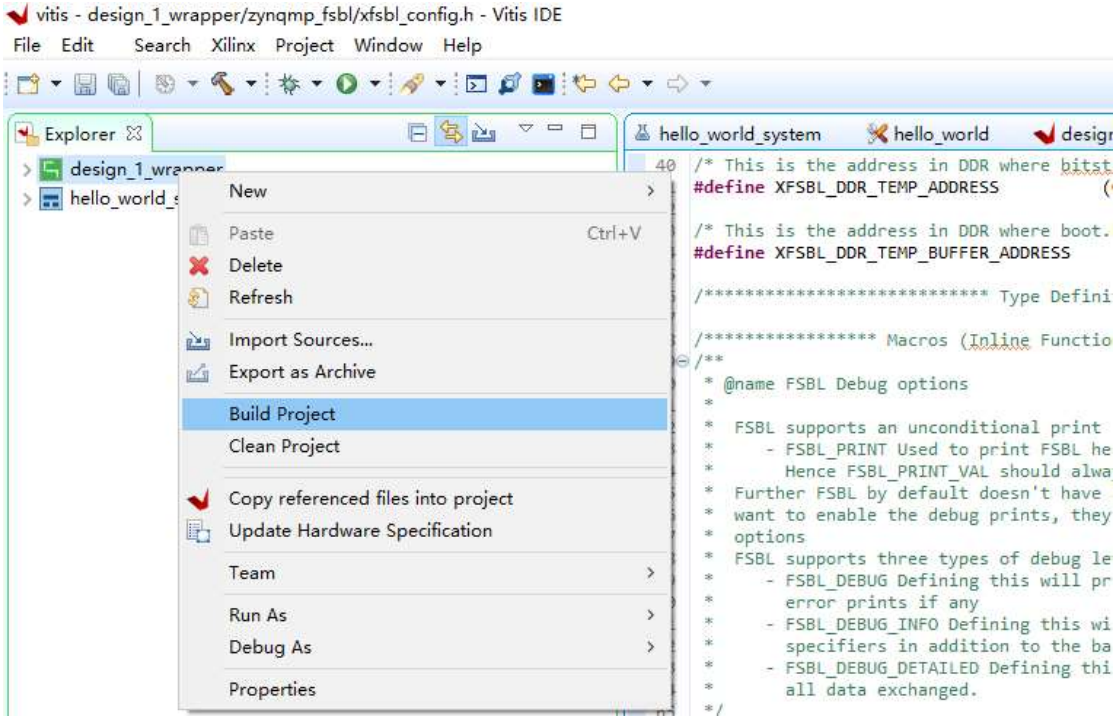


Figure 3-67 built

Click the system of the APP project, right-click and select Build project

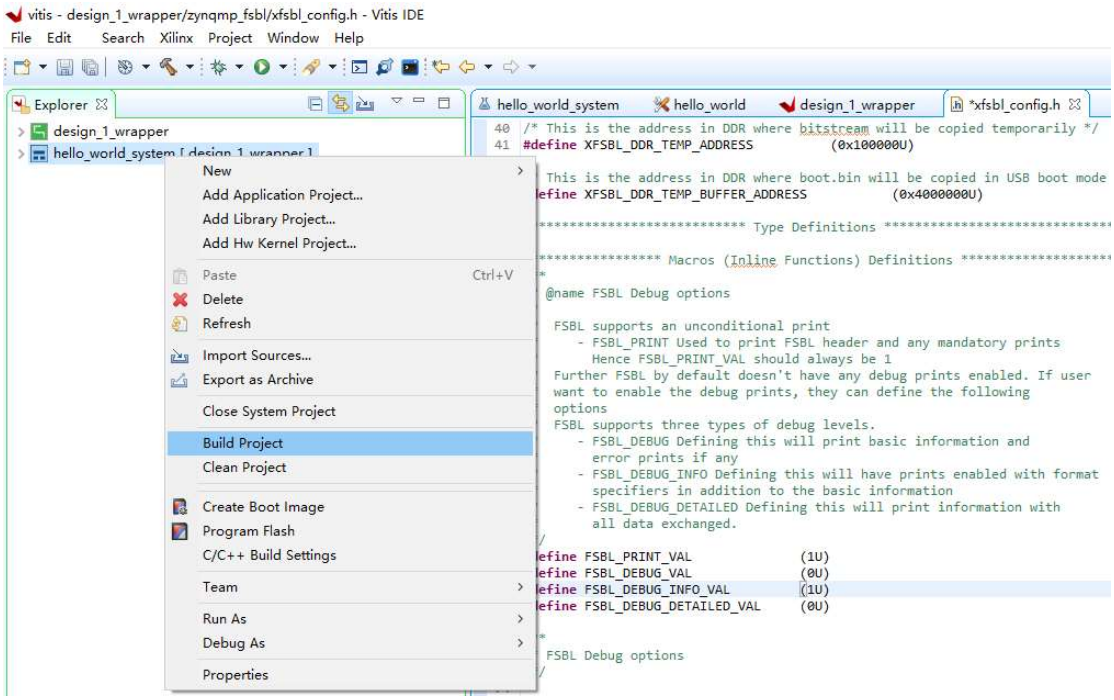


Figure 3-68 Re-built

At this time, there will be an extra Debug folder, and the corresponding BOOT.BIN will be generated

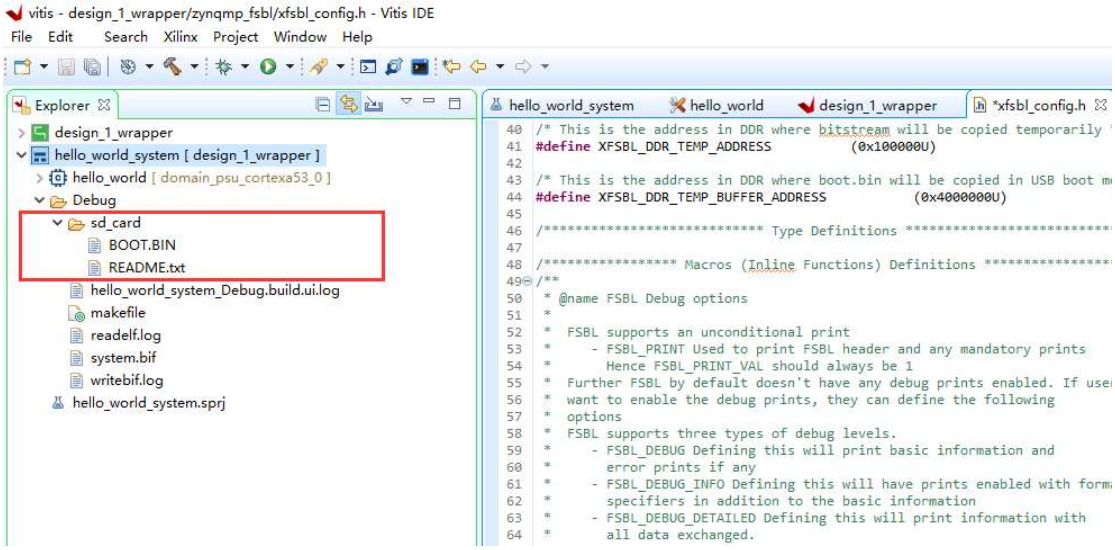


Figure 3-69 SD card start with the BOOT.BIN

Another way is to click the system right button of the APP project and select Create Boot Image. In the pop-up window, you can see the path of the generated BIF file. The BIF file is the configuration file for generating the BOOT file, as well as the path of the generated BOOT.bin file. The BOOT.bin file is the boot file we need. It can be placed on the SD card to boot, or it can be programmed to QSPI Flash.

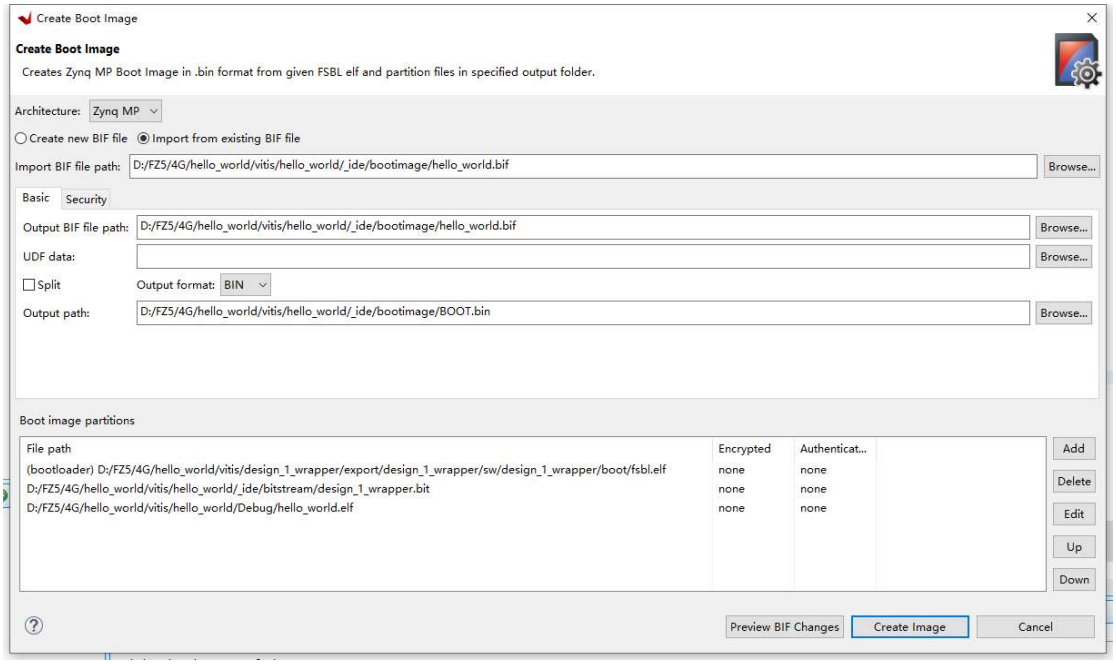


Figure 3-70 Generate BOOT.bin

There are files to be synthesized in the Boot image partitions list. The first file must be the bootloader file, which is the fsbl.elf file generated above, the second file is the FPGA configuration file bitstream, and the third is the application program. In this experiment The middle is hello_world.elf. Click Create Image to generate BOOT.bin.

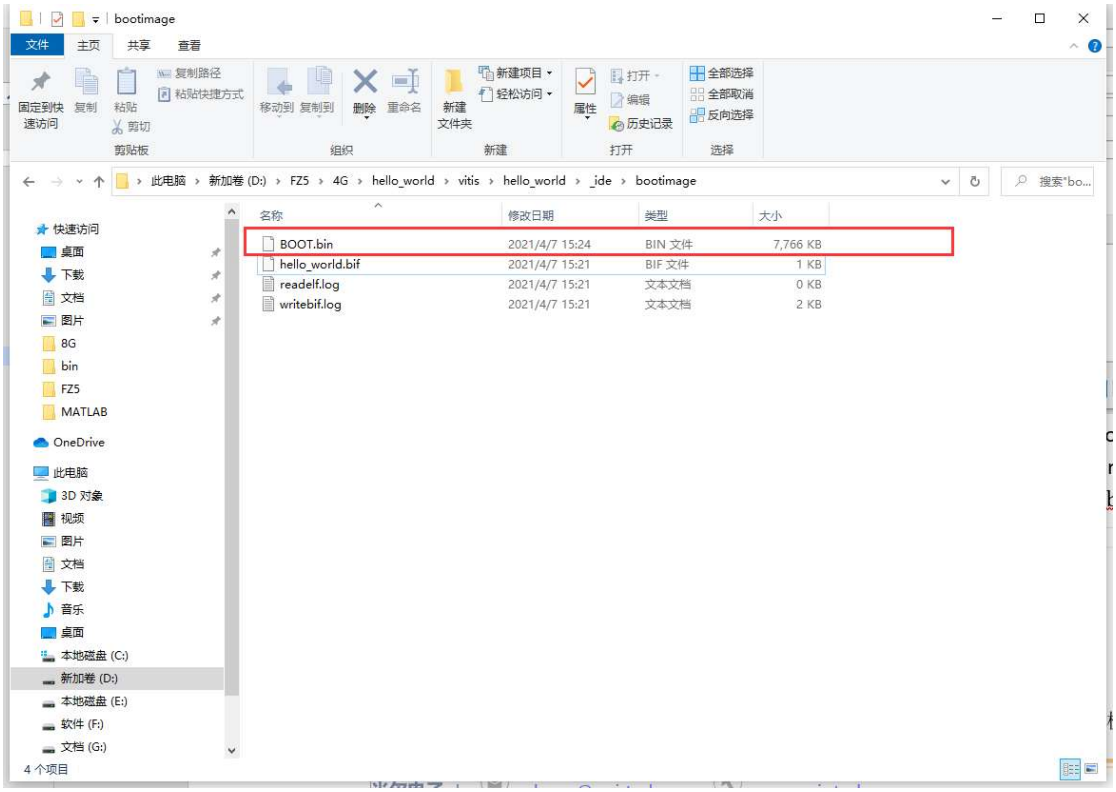


Figure 3-71 BOOT.bin

Switch the development board to SD card startup mode, and then copy the BOOT.bin file to the SD card and put it on the development board to run.

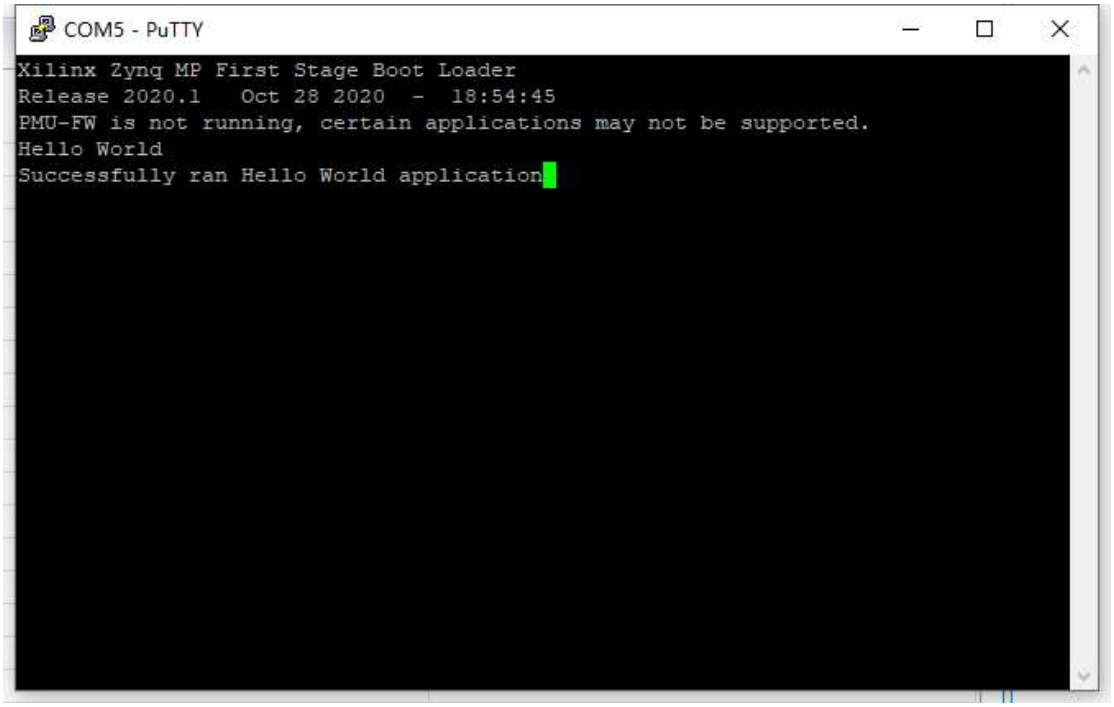


Figure 3-72 Output result

3.2 Data Interaction Between PL and PS

There are direct and indirect ways of data interaction between PS and PL. The direct way is through the design of EMIO. The PS terminal can directly access the registers of the PL terminal through a physical connection. Indirectly is to exchange data by sending interrupts and AXI4 bus interface, bus protocol, and bus.

3.2.1 Direct Data Exchange Between PL and PS

zynq's GPIO is divided into two types, MIO (multiuse I/O) and EMIO (extendable multiuse I/O)

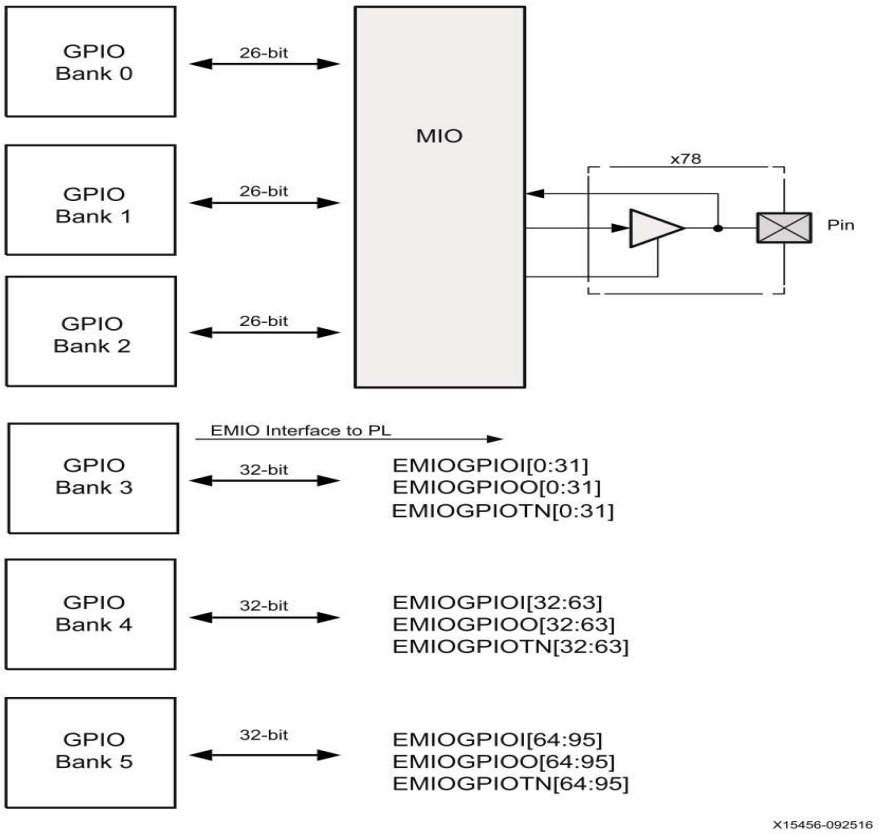


Figure 27-1: **GPIO Block Diagram**

Figure 3-73 MIO and EMIO architecture

There are 96 EMIOs of BANK3~BANK5, which can be directly accessed by the PS side. The advantage of using EMIO is that when MIO is not enough, PS can control the pins of PL part through EMIO. Of course, in addition to the direct GPIO interface, there are other interface methods that can directly access the resources of the PL end through the PS if the design meets the requirements.

3.2.2 Configurable Bus Between PS and PL

The AXI bus protocol is implemented with hardware inside the ZYNQ chip, including 12 physical interfaces, namely S_AXI_HP{0:3}_FPD, S_AXI_LPD, S_AXI_ACE_FPD, S_AXI_ACP_FPD, S_AXI_HPC{0,1}_FPD, M_AXI_HPM{0,1}_FPD, M_AXI_HPM0_LPD interface. S_AXI_HP{0:3}_FPD interface: It is a high-performance/bandwidth AXI4 standard interface, there are four in total, and the PL module is connected as the main device. Mainly used for PL to access memory on PS (DDR and FPD Main Switch) S_AXI_LPD interface: high-performance port, connect PL to LPD. Low-latency access to OCM and TCM, access to PS side DDR. S_AXI_HPC{0,1}_FPD interface: connect PL to FPD, connect to CCI, and access L1 and L2Cache. Due to CCI, access to DDR controller will be delayed.

M_AXI_HPM{0,1}_FPD interface: high-performance bus, PS is master, connects FPD to PL, and can be used for CPU, DMA, PCIe, etc. to push large amounts of data from PS to PL.

M_AXI_HPM0_LPD interface: low-latency interface bus, PS is the master, connects LPD to PL, can directly access BRAM, DDR, etc. on the PL side, and is often used to configure the registers on the PL side. Only M_AXI_HPM{0,1}_FPD and M_AXI_HPM0_LPD are Master Ports, namely host interfaces, and the rest are Slave Ports (slave interfaces).

The host interface has the authority to initiate reading and writing. ARM can use the two host interfaces to actively access the PL logic. In fact, it maps the PL to a certain address, and reading and writing the PL register is like reading and writing its own memory. The other slave interfaces are passive interfaces, accepting reads and writes from the PL, and accept them in reverse. In PS and PL interconnection applications, the most used interfaces are S_AXI_HP{0:3}_FPD, M_AXI_HPM{0,1}_FPD and M_AXI_HPM0_LPD.

The ARM on the PS side directly supports the AXI interface with hardware, while the PL needs to use logic to implement the corresponding AXI protocol. Xilinx provides ready-made IP such as AXI-DMA, AXI-GPIO, AXI-Dataover, AXI-Stream in the Vivado development environment to implement the corresponding interfaces. When in use, you can directly add them from the Vivado IP list to achieve the corresponding functions. The following picture shows various DMA IPs under Vivado:

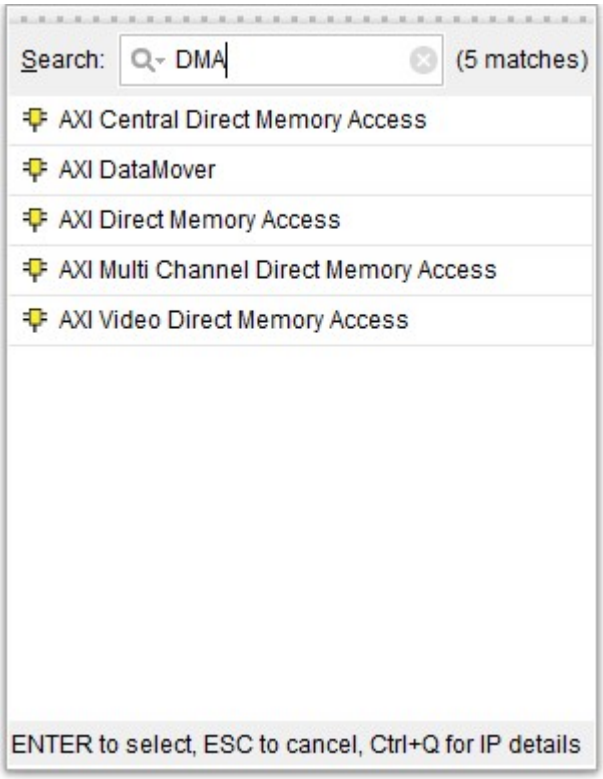


Figure 3-74 select DMA IP

The following is an introduction to the functions of several commonly used AXI interface IP: AXI-DMA: Realize the conversion from PS memory to PL high-speed transmission high-speed channel AXI-HP<---->AXI-Stream AXI-FIFO-MM2S: Realize the conversion from PS memory to PL general transmission channel AXI-HPM<----->AXI-Stream AXI-Datamover:

Realize the conversion from PS memory to PL high-speed transmission high-speed channel AXI-HP<---->AXI-Stream, but this time it is completely controlled by PL, and PS is completely passive. AXI-VDMA: Realize the conversion from PS memory to PL high-speed transmission high-speed channel AXI-HP<---->AXI-Stream, but it is specifically for video, image and other two-dimensional data. AXI-CDMA:

This is done by the PL to move the data from one location in the memory to another, without the CPU's intervention. , We will talk about it as an example in a later chapter. Sometimes, users need to develop their own defined IP to communicate with the PS, then you can use the wizard to generate the corresponding IP.

User-defined IP cores can have AXI4-Lite, AXI4, AXI-Stream, PLB and FSL interfaces. The latter two are not used because they are not supported by ARM. With the above official IP and the custom IP generated by the wizard, users

don’ t actually need to know much about AXI timing (unless they do encounter problems), because Xilinx has already encapsulated the details related to AXI timing, users only need Just focus on your own logic implementation. Strictly speaking, the AXI protocol is a point-to-point master-slave interface protocol. When multiple peripherals need to exchange data with each other, we need to add an AXI Interconnect module, which is the AXI interconnect matrix. The function is to provide one or more AXI master devices. A switching mechanism connected to one or more AXI slave devices (somewhat similar to the switching matrix in a switch). This AXI Interconnect IP core can support up to 16 master devices and 16 slave devices. If you need more interfaces, you can add more IP cores.

The basic connection modes of AXI Interconnect are as follows:

N to-1 Interconnect to-N Interconnect

N-to-M Interconnect (Crossbar Mode)

N-to-M Interconnect (Shared Access Mode)

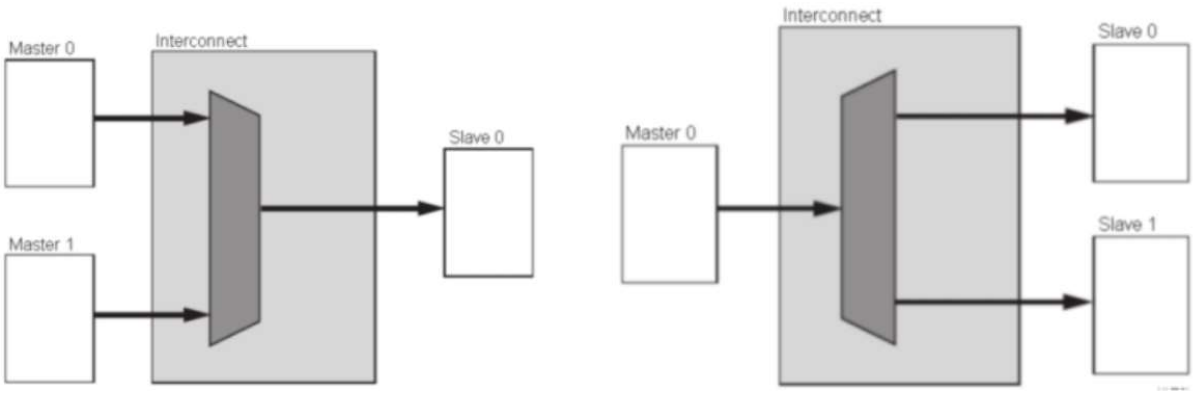


Figure 3-75 AXI interconnect

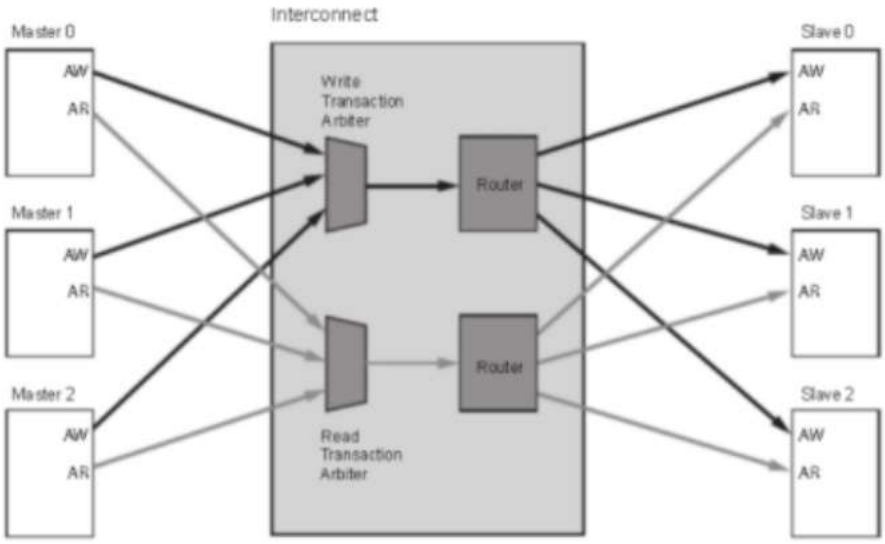


Figure 3-76 AXI interconnect structure

ZYNQ's internal AXI interface devices are interconnected by means of interconnection matrix, which not only ensures the efficiency of data transmission, but also ensures the flexibility of connection. In Xilinx Vivado, we provide the IP core axi_interconnect that realizes this interconnection matrix, and we only need to call it.

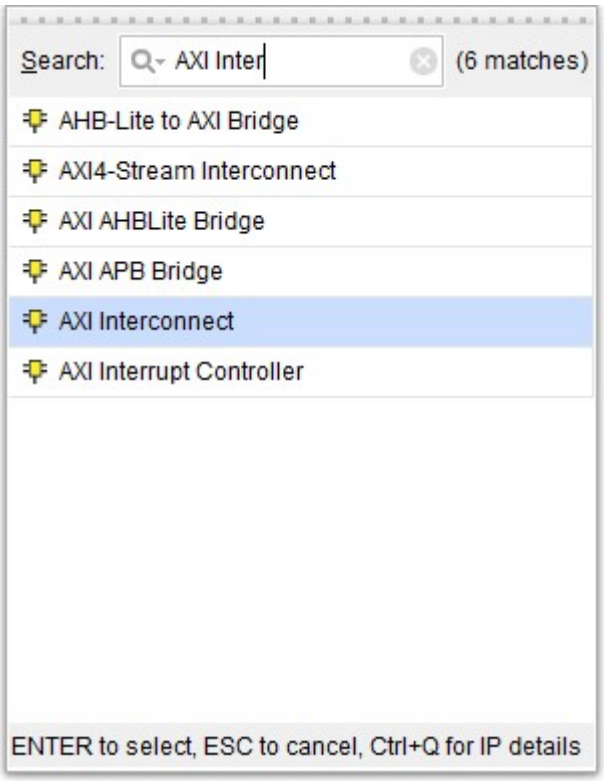


Figure 3-77 AXI interconnect select IP

3.3 Introduction to AXI4 Bus

3.3.1 AXI4 Protocol

Since XILINX launched the VIVADO development environment for 7 series FPGAs and SOCs, FPGA development tends to use existing IP cores for engineering construction and verification, reducing the workload of code writing, especially in the use of ZYNQ. .

Almost all IP cores in the Vivado development environment support the AXI bus, and the IP core interface is standardized. FPGA engineers only need to learn the AXI bus, and almost master the use of all IP core interfaces. Such standardization also makes it possible to build a system faster and more

convenient for verification, which is a huge improvement over ISE; For some customized functions, there is no well-defined IP core in the VIVADO development environment.

At this time, FPGA engineers may need to self-research, how to quickly integrate self-developed logic into the system, and use VIVADO' s design philosophy is to self-research The logic is standardized, using AXI bus as the interface, and VIVADO also provides users with corresponding tools. AXI (Advanced eXtensible Interface) was originally a bus protocol proposed by ARM. Xilinx started to support the AXI bus from the 6 series of FPGAs, and currently uses the AXI4 version.

(1) AXI4: (For high-performance memory-mapped requirements.) Mainly oriented to the needs of high-performance address mapping communication, it is an address mapping-oriented interface that allows a maximum of 256 rounds of data burst transmission;

(2) AXI4-Lite: (For simple, low-throughput memory-mapped communication) is a lightweight address-mapped single transmission interface that occupies very few logical units.

(3) AXI4-Stream: (For high-speed streaming data.)

For high-speed streaming data transmission; remove the address item, allowing unlimited data burst transmission mode. The AXI4 bus and the AXI4-Lite bus have the same components:

(1) Read address channel, including ARVALID, ARADDR, ARREADY signals;

(2) Read data channel, including RVALID, RDATA, RREADY, RRESP signals;

(3) Write address channel, including AWVALID, AWADDR, AWREADY signals;

(4) Write data channel, including WVALID, WDATA, WSTRB, WREADY signals;

(5) Write response channel, including BVALID, BRESP, BREADY signals;

(6) System channels, including: ACLK, ARESETN signals.

The composition of the AXI4-Stream bus is:

(1) ACLK signal: bus clock, the rising edge is valid;

(2) ARESETN signal: bus reset, active low;

(3) TREADY signal: The slave tells the host to prepare for transmission;

(4) TDATA signal: data, optional width 32, 64, 128, 256bit

(5) TSTRB signal: each bit corresponds to a valid byte of TDATA, the width is TDATA/8;

(6) TLAST signal: the master tells the slave that this transmission is the end of the burst transmission;

(7) TVALID signal: the master tells the slave that the data is valid for this transmission;

(8) TUSER signal: user-defined signal with a width of 128/8bit.

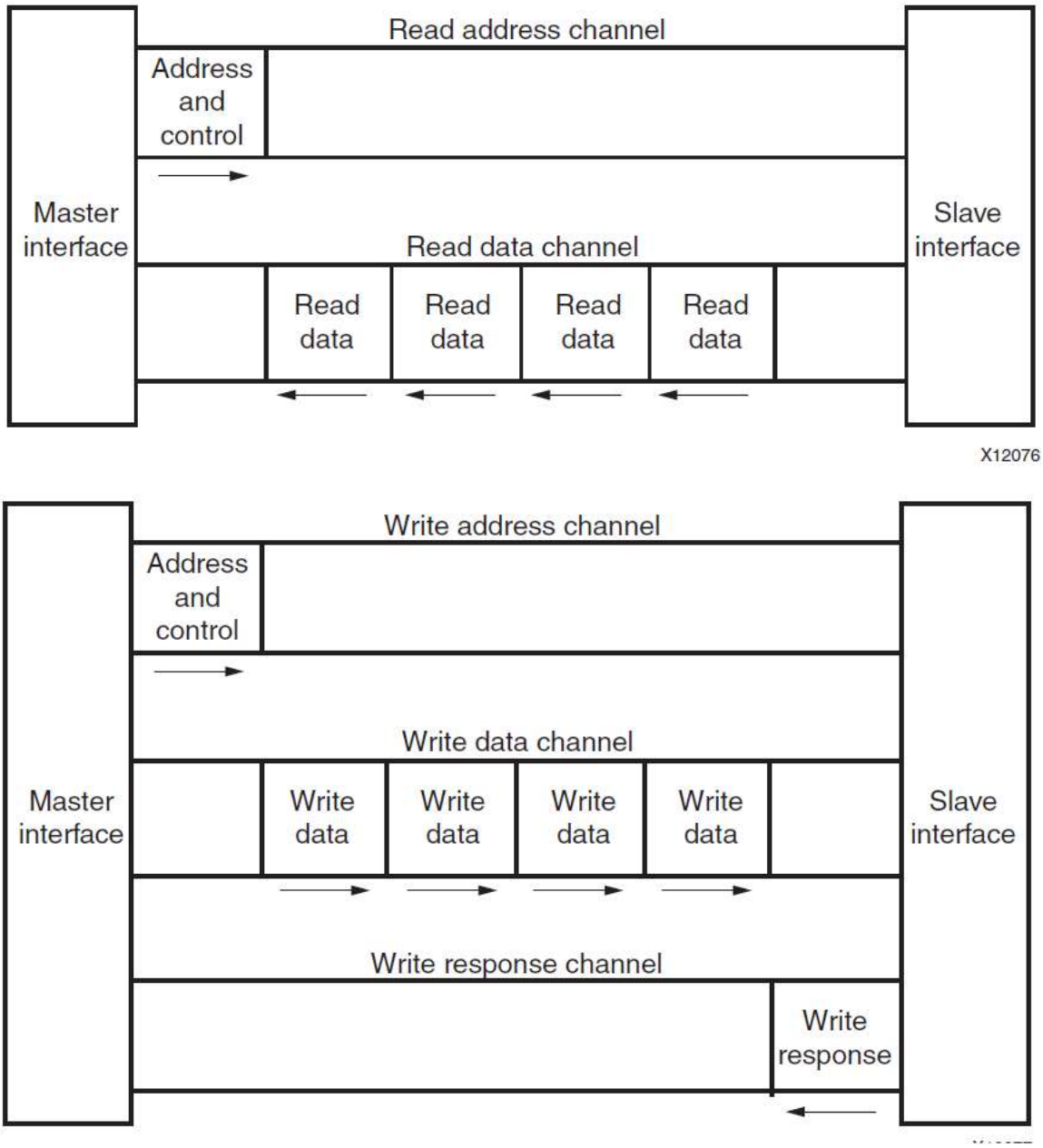


Figure 3-78 AXI bus read and write channel analysis

The formulation of the protocol is based on the bus composition. Therefore, AXI4, AXI4-Lite, and AXI4-Stream are all AXI4 protocols. The two ends of the AXI bus protocol can be divided into master (master) and slave (slave). Generally, they need to be connected through an AXI Interconnect. The function is to provide one or more AXI master devices to connect to one or An exchange mechanism for multiple AXI slave devices.

The main function of AXI Interconnect is that when there are multiple masters and slaves, AXI Interconnect is responsible for connecting and managing them. Because AXI supports out-of-order sending, out-of-order sending requires the ID signal support of the host, and the ID sent by different hosts may be the same, and AXI Interconnect solves this problem. It will process the ID signals of different hosts to make the ID. only.

The AXI protocol separates the read address channel, read data channel, write address channel, write data channel, and write response channel. Each channel has its own handshake protocol. Each channel does not interfere with each other but is dependent on each other. This is one of the reasons why AXI is efficient.

3.3.2 AXI Handshake Protocol

AXI4 uses a READY, VALID handshake communication mechanism. Simply put, there is a handshake process before the master and the slave perform data communication. The transmission source generates the VLAID signal to indicate when the data or control information is valid. The destination source generates a READY signal to indicate that it is ready to receive data or control information. The transfer occurs when the VALID and READY signals are high at the same time. The example in the following figure:

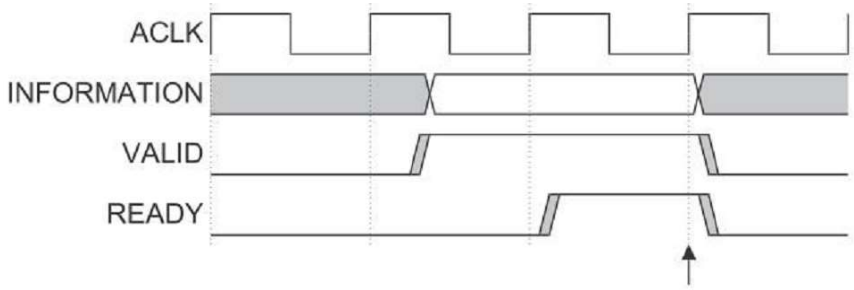


Figure 3-79 Handshake with Valid before READY

When the address appears on the address bus, the transmitted data will appear on the read data channel. The device keeps VALID low until the read data

is valid. In order to indicate the completion of a burst read and write, the device uses the RLAST signal to indicate the last data to be transmitted.

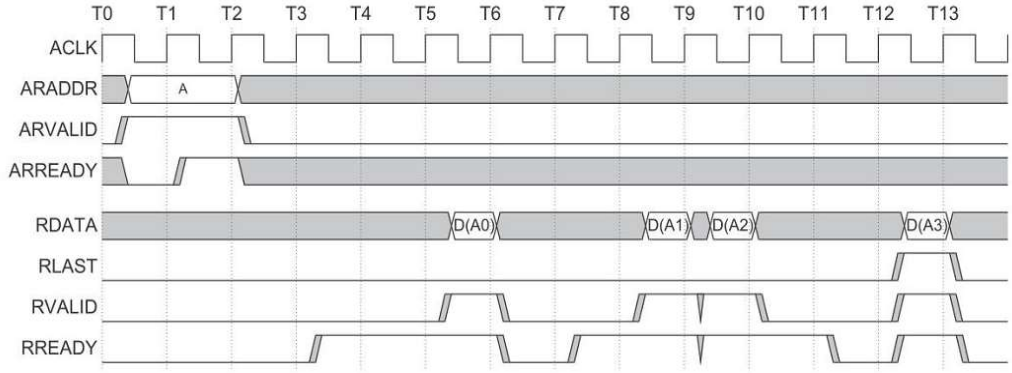


Figure 3-80 Handshake and transmission of READY earlier than VALID

The host sends the address and control information to the write address channel, and then the host sends each write data to the write data channel. When the host sends the last data, the WLAST signal goes high. When the device has received all the data, it sends a write response back to the host to indicate the completion of the write transaction.

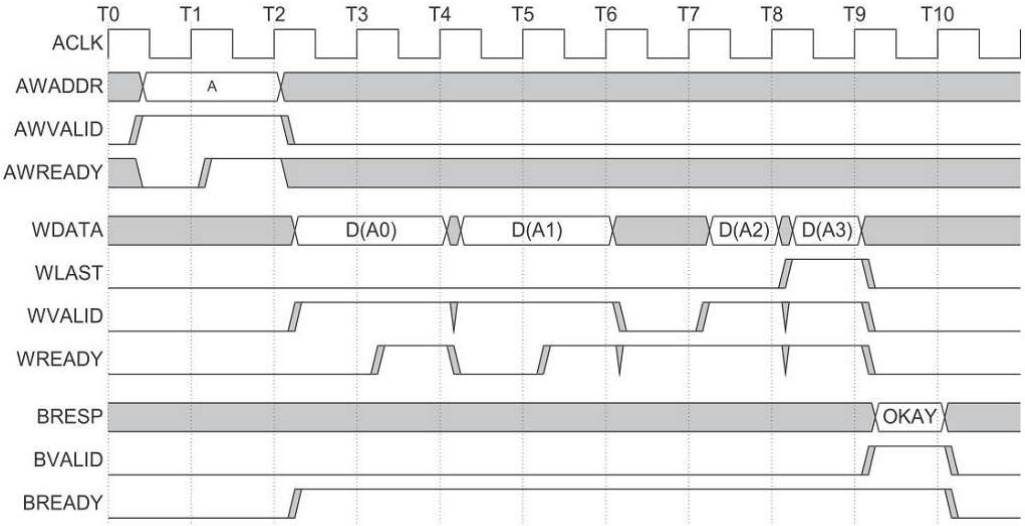


Figure 3-80 Example of data transmission

The data read and write timing of AXI_LITE is the same as the AXI burst timing, except that only one data is transmitted at a time; Timing of AXI4-Stream: Data stream-oriented transmission mode, the omitted address channel, the remaining timing is the same as the AXI burst timing;

3.4 Sections of this Chapter

This chapter introduces the basic configuration of the hardware platform establishment project. The configuration is based on the software and hardware resources of the hardware platform. Secondly, it briefly introduces the interconnection technology between PS and PL. For further study, you need to refer to ARM's AMBA protocol spec, which describes the bus design method protocol principle in SOC design in detail. Secondly, the interconnection module AXI Interconnecter between PS and PL greatly facilitates the use of ARM+FPGA heterogeneous platform.

Chapter 4Interface Device Module Based on AXI4-Lite Bus

This chapter is mainly to understand the AXI4-Lite bus interface types reserved by the PS side and connected to the PL side, and their specific use, focusing on the introduction of GPIO, AXI UART, AXI IIC and other control buses and general IO control interfaces.

4.1 AXI UART

4.1.1 AXI UART Basis Knowledge

The Xilinx LogiCORE IP AXI general-purpose input/output (GPIO) core provides general-purpose input/output interfaces for the AXI interface. This 32-bit soft intellectual property (IP) core is designed to interface with the AXI4-Lite interface. The specific characteristics are as follows: Support AXI4-Lite interface specification Support configurable single or dual GPIO channels Supports configurable channel widths from 1 to 32-bit GPIO pins Supports dynamic programming of each GPIO bit as input or output Supports individual configuration of each channel Support each independent reset value of all registers Support optional interrupt request generation

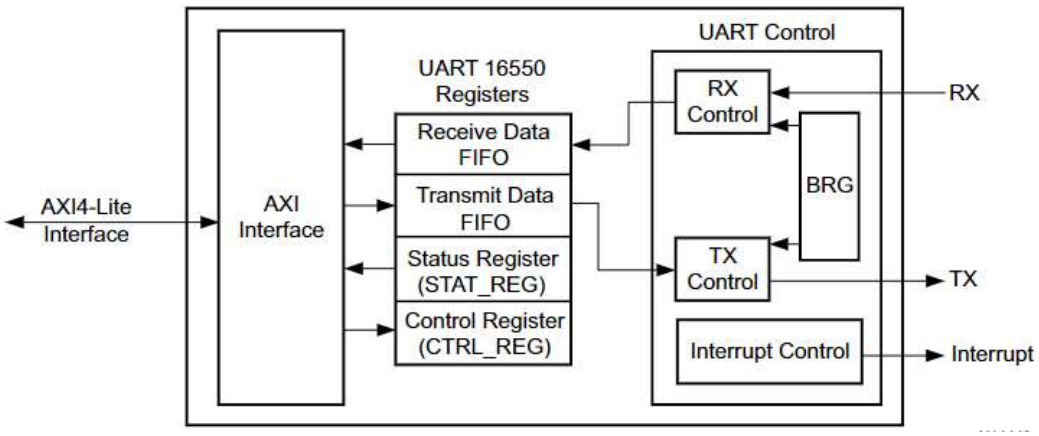


Figure 4-1 Block diagram of the structure

Design of AXI UART function: AXI GPIO is designed to provide a general-purpose input/output interface for the AXI4-Lite interface. AXI GPIO can be configured as a single-channel or dual-channel device. The width of each channel can be configured independently. By enabling or disabling the tri-state buffer, the port can be dynamically configured as input or output. The channel can be

configured to generate an interrupt when any of its inputs are converted. The general-purpose input/output (GPIO) core is an interface that allows easy access to the internal properties of the device. The same kernel can be used to control the behavior of external devices. For further information, please refer to pg144-LogiCORE IP AXI GPIO V2.0 Product Guide(AXI).pdf

4.1.2 Experimental Logical

We want to use this AXI UART to realize ps control the uart IP core on the pl side through the axi bus, output the serial port signal, perform level conversion through the chip on the hardware, and then output to the port, the port is connected to the computer with the corresponding serial line, the schematic diagram Shown:

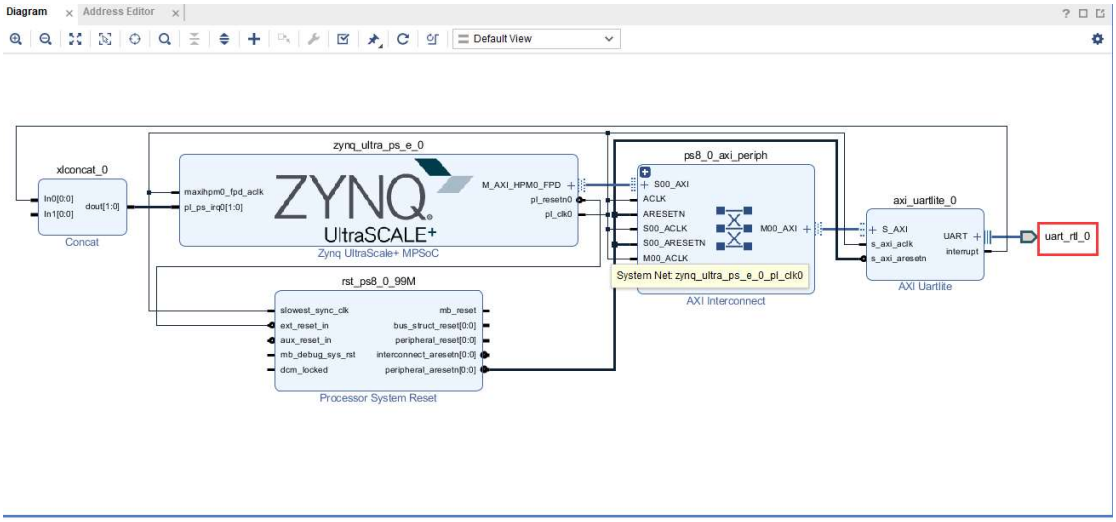


Figure 4-2 Block diagram of engineering design

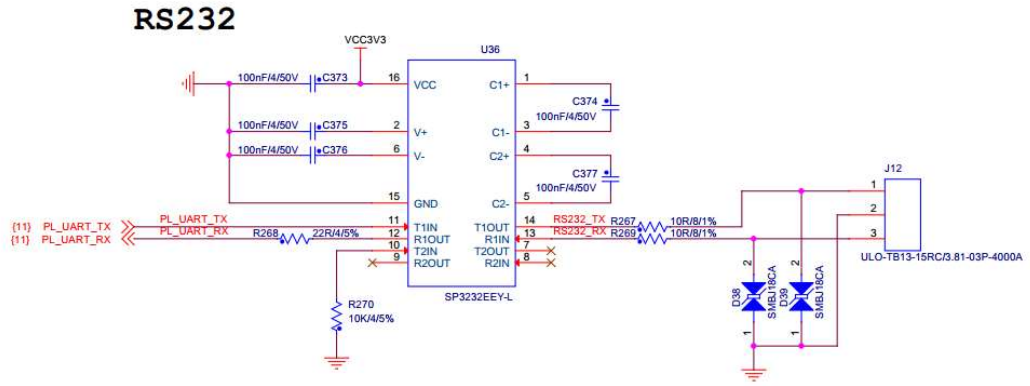


Figure 4-3 Schematic design

4.1.3Experimental Steps

Vivado project：

Create a new vivado project and name it axi_uart. The basic configuration of the PS end is the same as the configuration parameters in the "hello_world" project. The corresponding CD-ROM project document is axi_uartlite.rar.

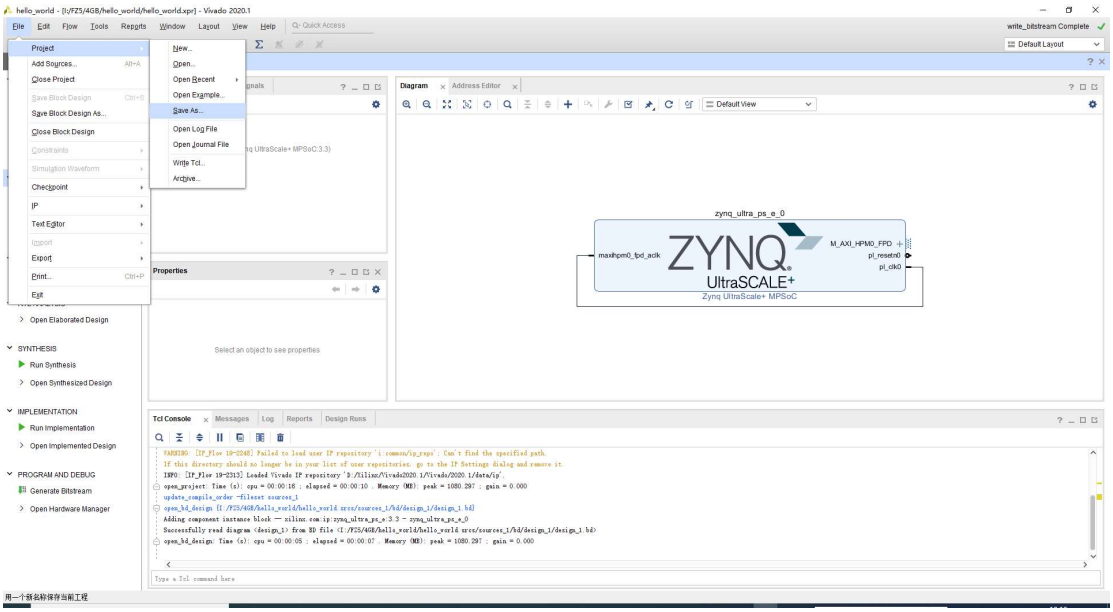


Figure 4-4 Vivado design

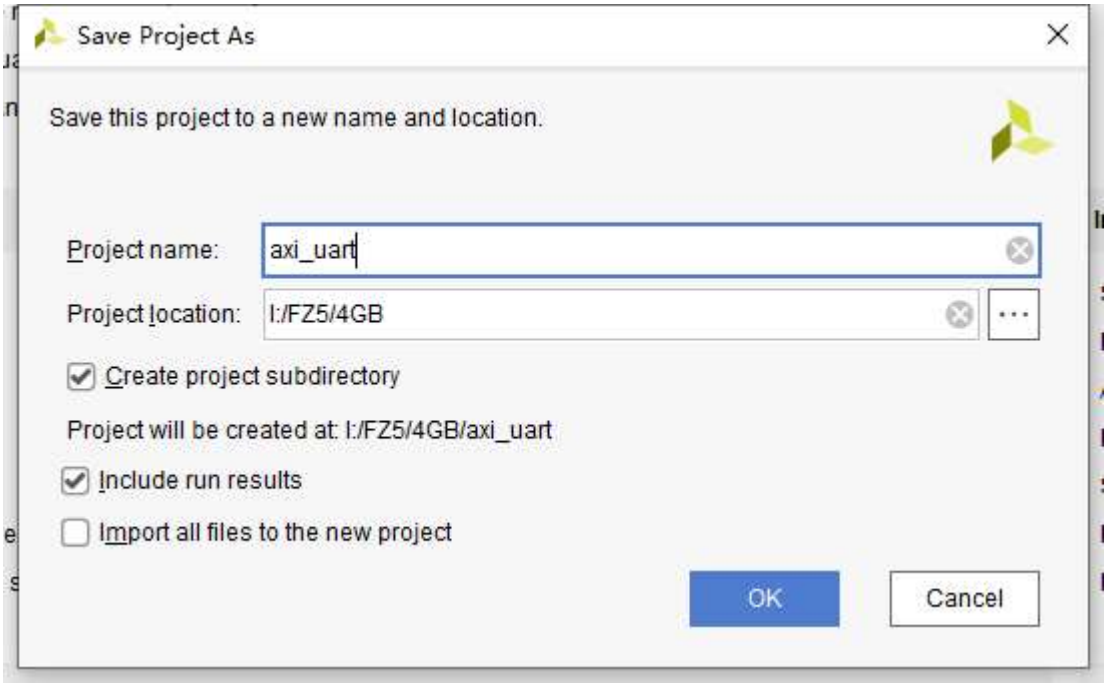


Figure 4-5 Name of Save As Project

Configure PL interrupt.

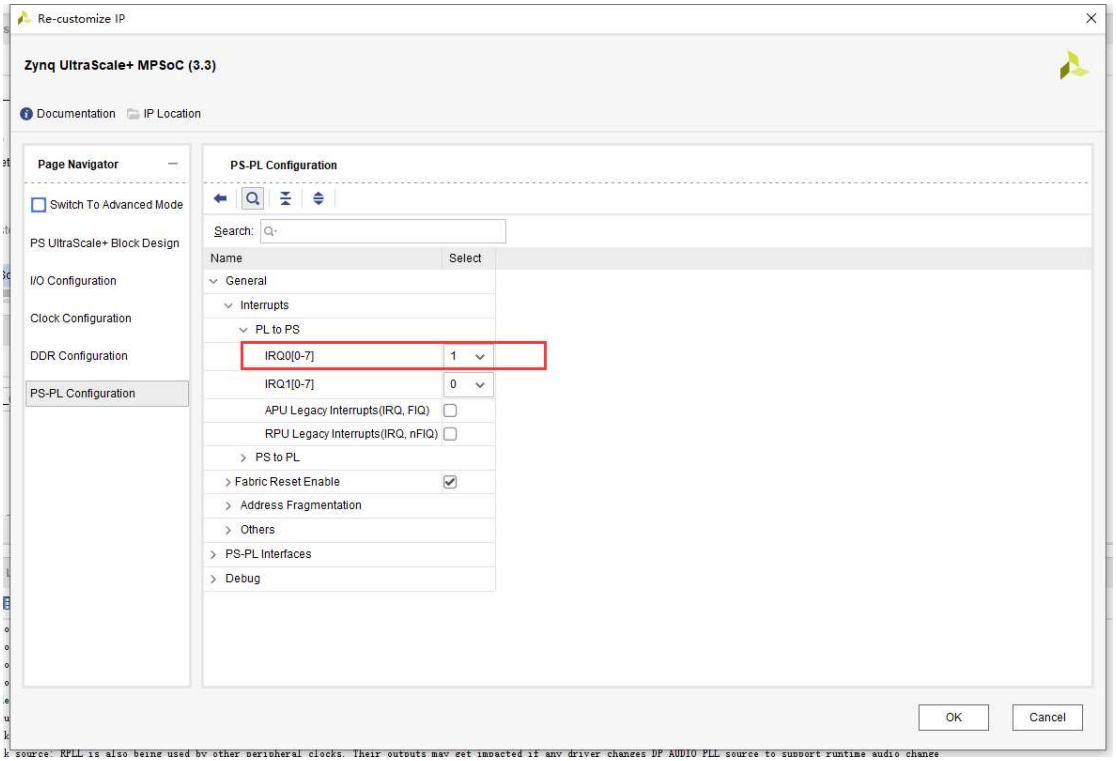


Figure 4-6 Add configuration

Add an AXI Uart IP core, here configuration directly select the default.

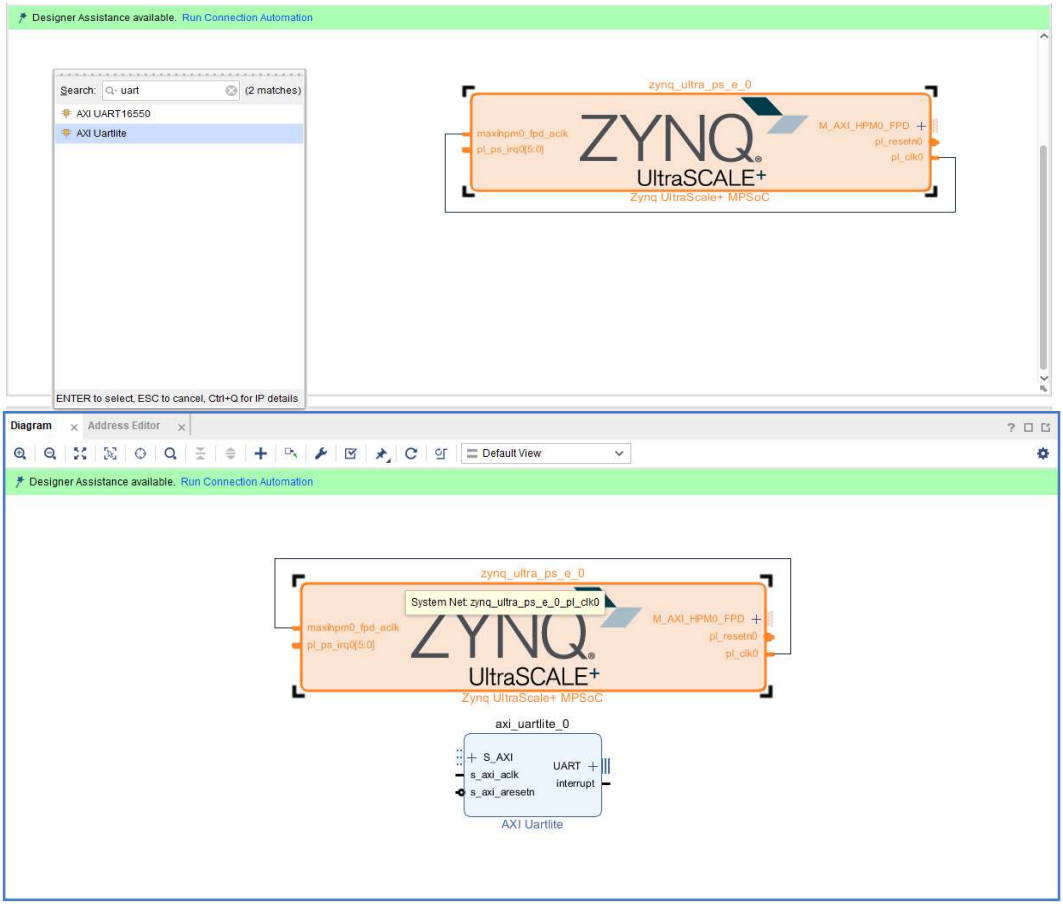


Figure 4-7 Add serial port IP

The concat IP core was added, and the connection was interrupted.

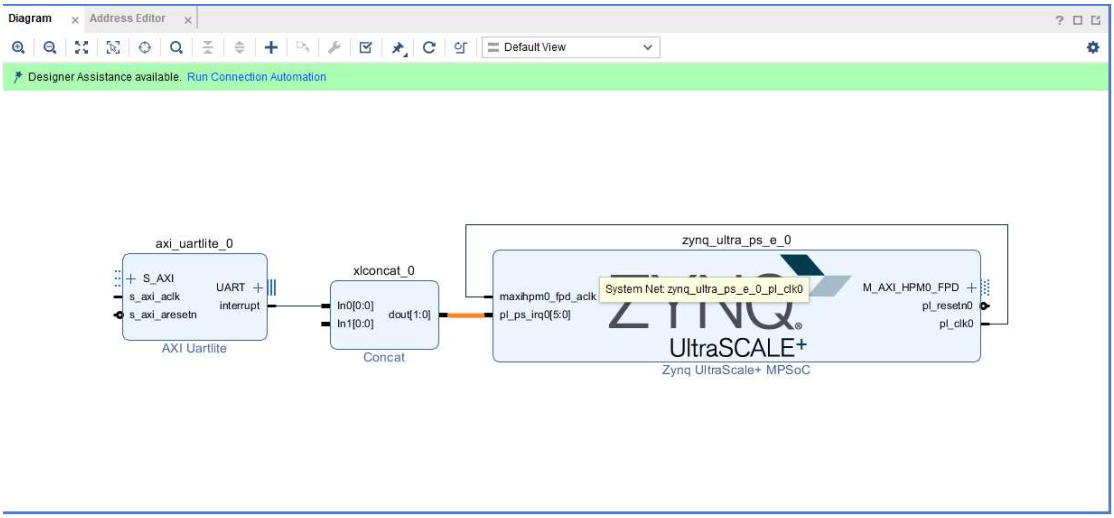


Figure 4-8 Add concat cable

Click Run Connection Automation-->OK at the top to automatically connect

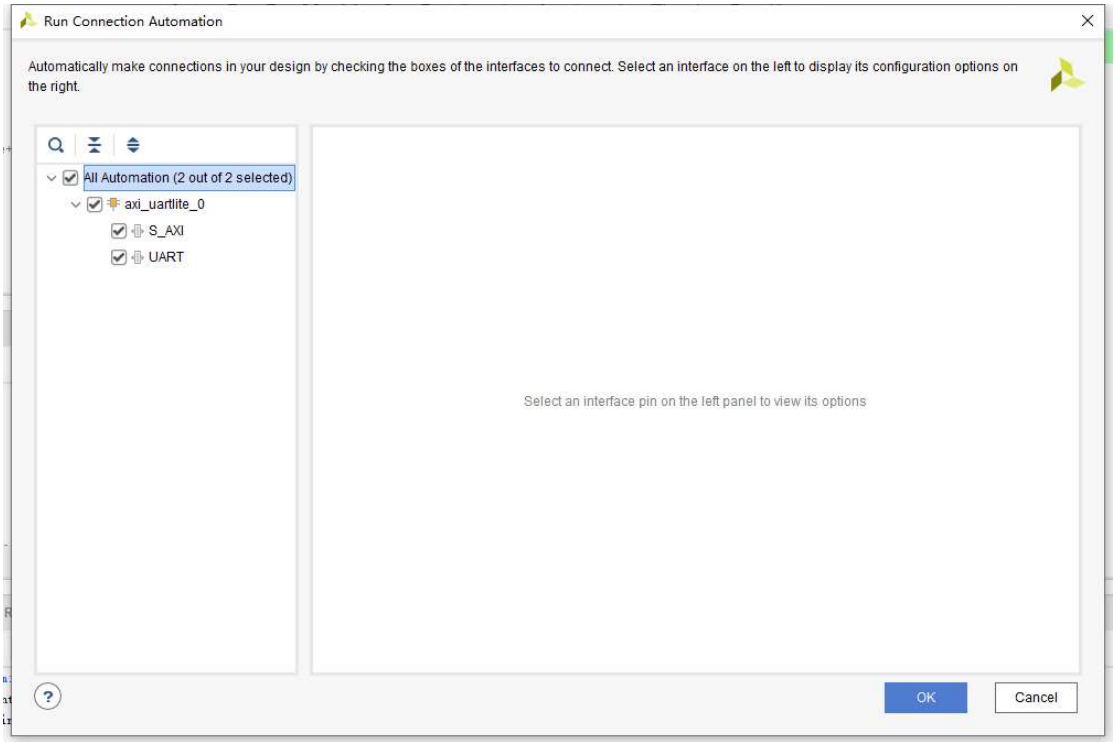


Figure 4-9 Automatic connection

After the automatic connection is completed, as shown in the figure below

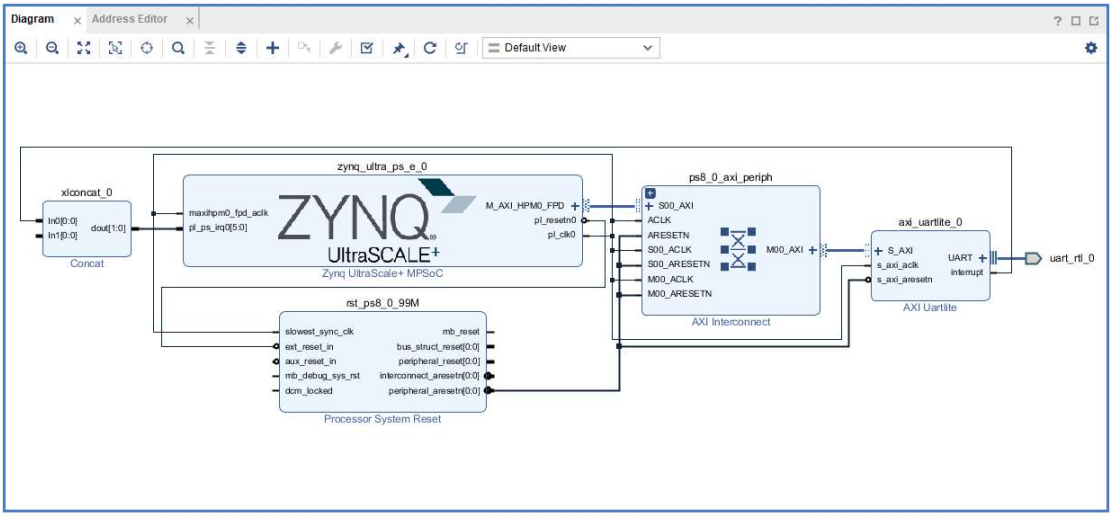


Figure 4-10 BD project after connection

After the top-level file is generated, the constraint file is added.

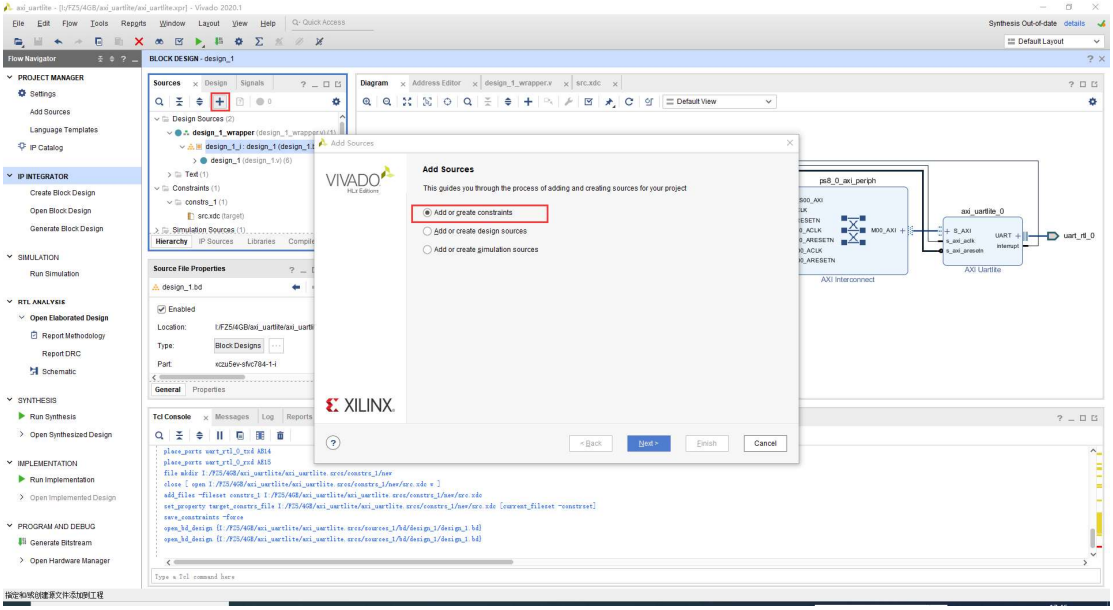


Figure 4-11 Adding a constraint document

Create a src constraint file.

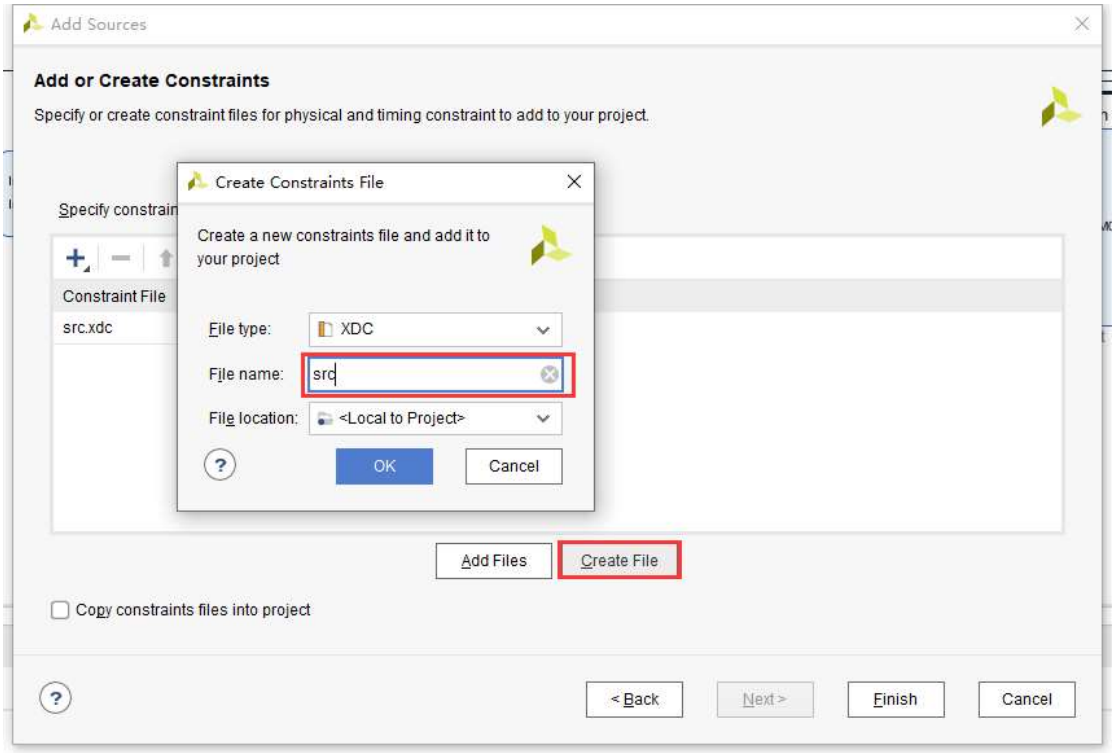


Figure 4-12 Constraining the document name

Add the following content to led.xdc, the port name should be consistent with the top file port.

```
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_txd]
set_property PACKAGE_PIN AB14 [get_ports uart_rtl_0_txd]
set_property PACKAGE_PIN AE15 [get_ports uart_rtl_0_rxd]
```

Figure 4-12 Add pin assignment

The following steps to generate bitstream and export xsa files are the same as those in the first project, so I won’ t repeat them here. Vitis project: Enter the Vitis software, create a new project named axi_uartlite, select the empty template on the Templates template selection page, and click platform.xprBoard Support Package(axi_uartlite) Import ExamplesExample Directory.

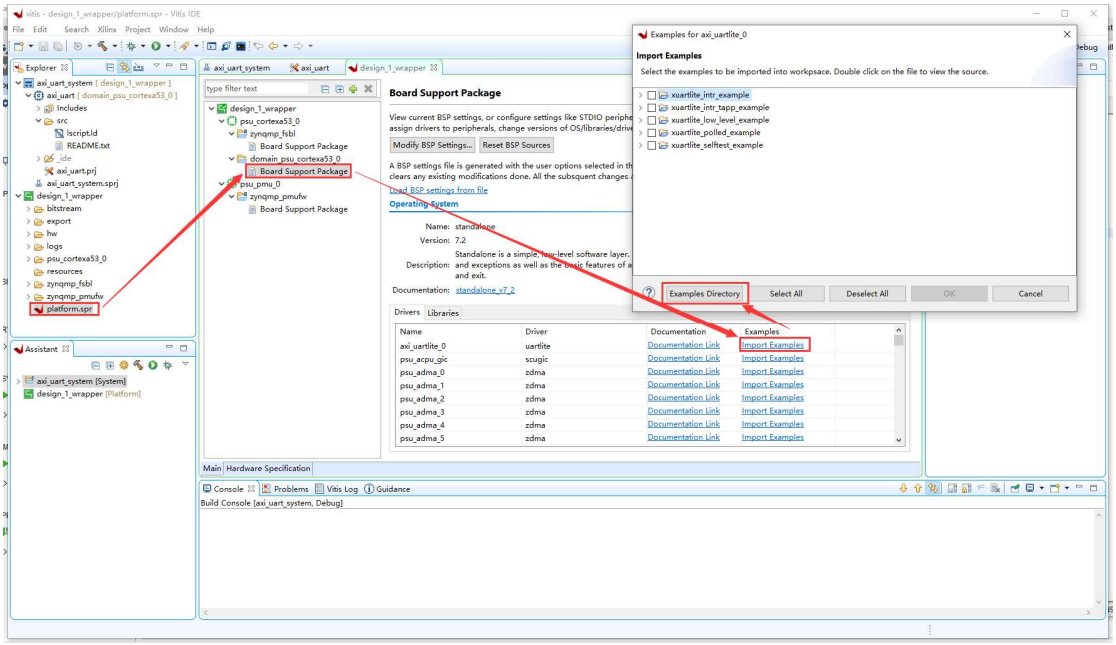


Figure 4-13 vitis project

In the opened directory, copy xuartlite_low_level_example.c to our project src directory.

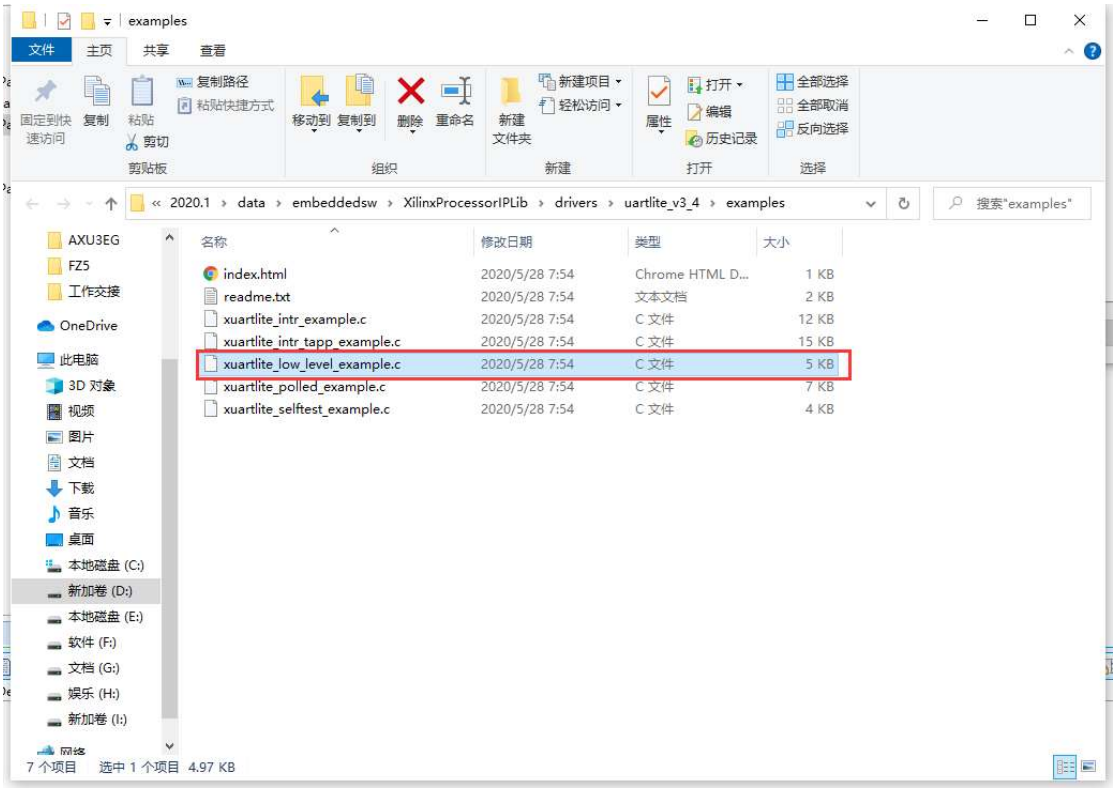


Figure 4-14 Official driver porting

After compiling, connect the JTAG line to the development board and the UART USB line to the PC Use PuTTY software as a serial terminal debugging tool. The Debug mode is the same as the first project. After the program is

downloaded, then we can find the debug serial port output results with the software serial port PuTTY.

4.2 IIC

4.2.1 IIC Basis Knowledge

The I2C bus is a serial data bus with only two signal lines, one is the bidirectional data line SDA, and the other is the clock line SCL. The two lines can connect multiple devices. There is a fixed address in the IIC device, and it will respond only when the value transmitted on the two lines is equal to the fixed address of the IIC device. Usually we divide the IIC device into master device and slave device for convenience, whoever controls the clock line is the master device.

4.2.2 Experiment Logical

The work of this experiment is to read and write the EEPROM through the I2C bus on the PS side after configuring the I2C on the PS side.

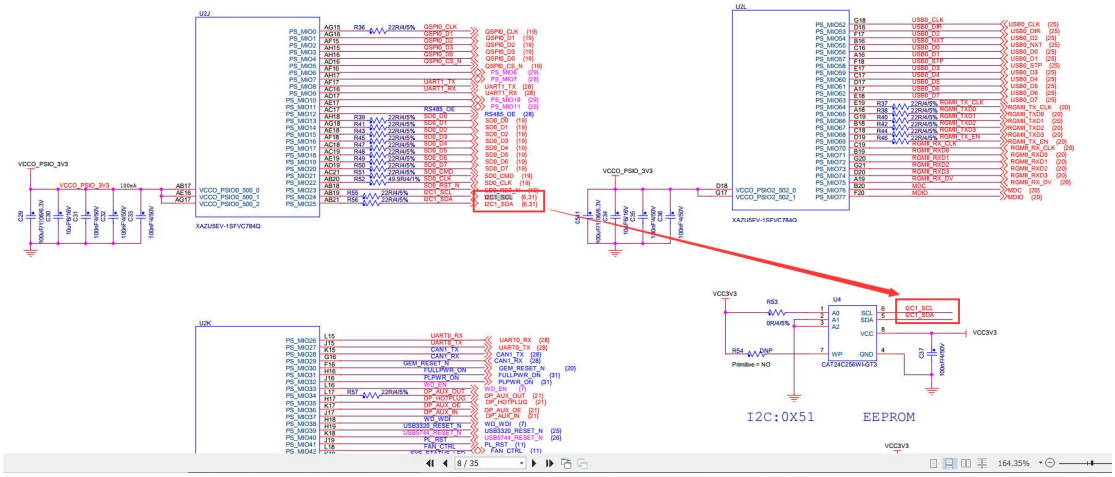


Figure 4-15 Schematic diagram of schematic connection

4.2.3Experiment Steps

Vivado project：

Based on the "hello_world" project, save it as an iic_test project. The name of the CD is iic_test.rar. Here iic is the IIC derived from the MIO on the PS side. It has been configured in the first project, so no changes are needed. Generate bitstream directly and export xsa file. Vitis project: The steps of creating a new project can refer to the first project. After creating an application project named iic_test, use the same method as axi uart, click platform.xprBoard Support

Package(psu_i2c_1)Import ExamplesExample Directory, In the opened directory, copy xiipcs_eeprom_polled_example.c to our project src directory.

Since the address of our iic device is 0x51, modify the first address of Eeproto 0x51 here, just click compile.

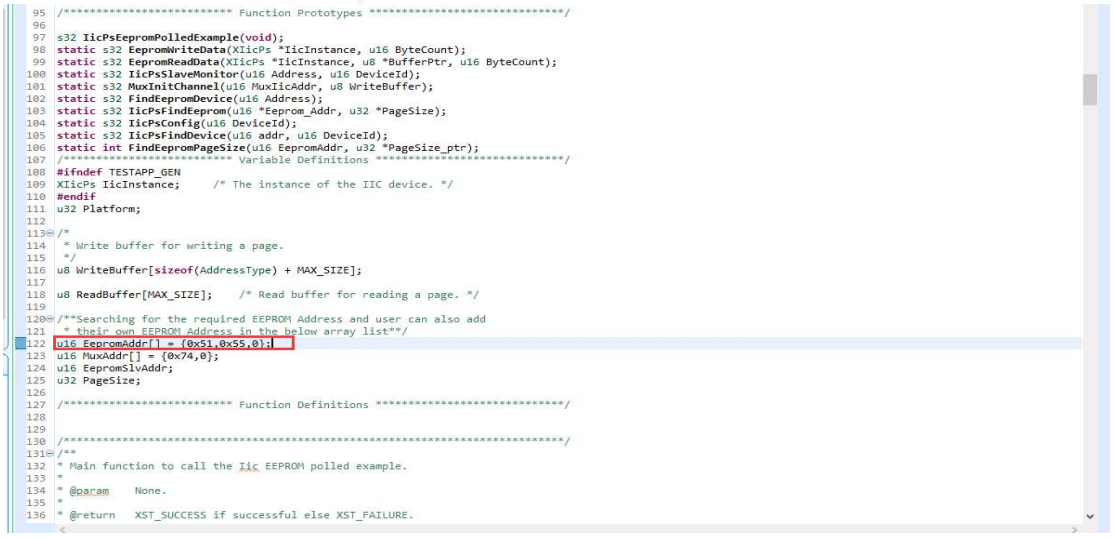


Figure 4-16 vitis program design

The Jtag debugging and SD card startup methods are the same as the first project. After the program is downloaded, the debugging serial port output results are as follows.

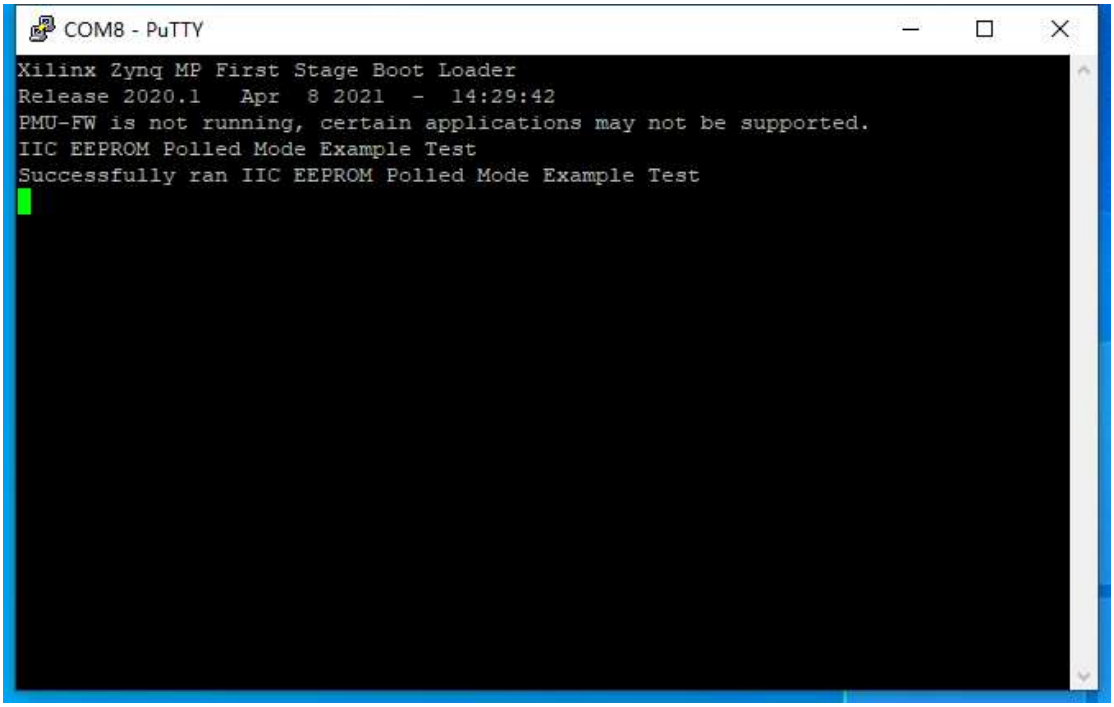


Figure 4-17 Results display

4.3 Sections of this Chapter

This chapter has two sub-sections. The first section describes how PS controls uart serial port IP printing data through axi bus. The second section is about some knowledge about IIC, and using the IIC peripherals of PS to read and write the EEPROM on the board. The knowledge point is the instantiation of the axi bus IP core on the PL side, which is connected to the PS through the axi bus and controlled by software.

Chapter 5Device ModuleBased on AXI4 High-Speed Data Interface

5.1 BRAM

5.1.1AXI BRAM Contorller Basis Knowledge

BRAM is a RAM resource customized by FPGA, which has a large storage space and is frequently used in daily projects. BRAM is arranged inside the FPGA in an array. It is the main part of the FPGA to realize various storage functions. It is a real synchronous RAM with dual read/write ports. 。

5.1.2Experiment Logical

The purpose of this article is to use Block Memory for data interaction or data sharing between PS and PL, through the M_AXI_HPM0_FPD port on the zynq PS side to control two axi bram controllers, write data to bram through controller 0, and then use controller 1 to The data in bram is read, and the read result is printed out to the serial port terminal for display.

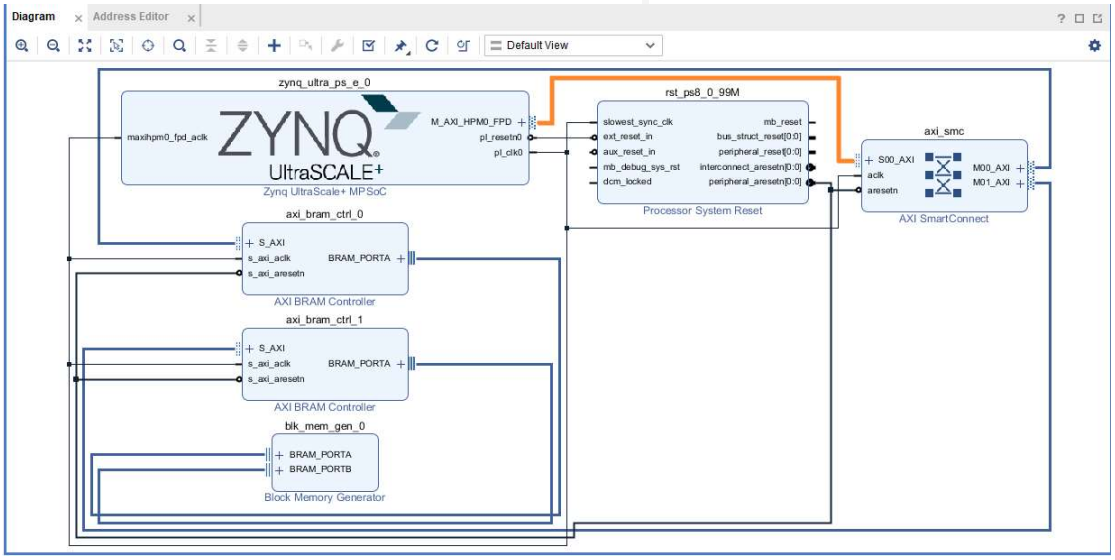


Figure 5-1 BRAM experimental logic structure block diagram

5.1.3Experiment Steps

Based on the "hello_world" project, save it as a bram_test project. Corresponding to the CD file bram_test.rar. Enter block in the search bar and double-click the Block Memory Generator to add the bram core. Double-click the

bram core, the configuration is as follows. Set Memory Type to True Dual Port RAM.

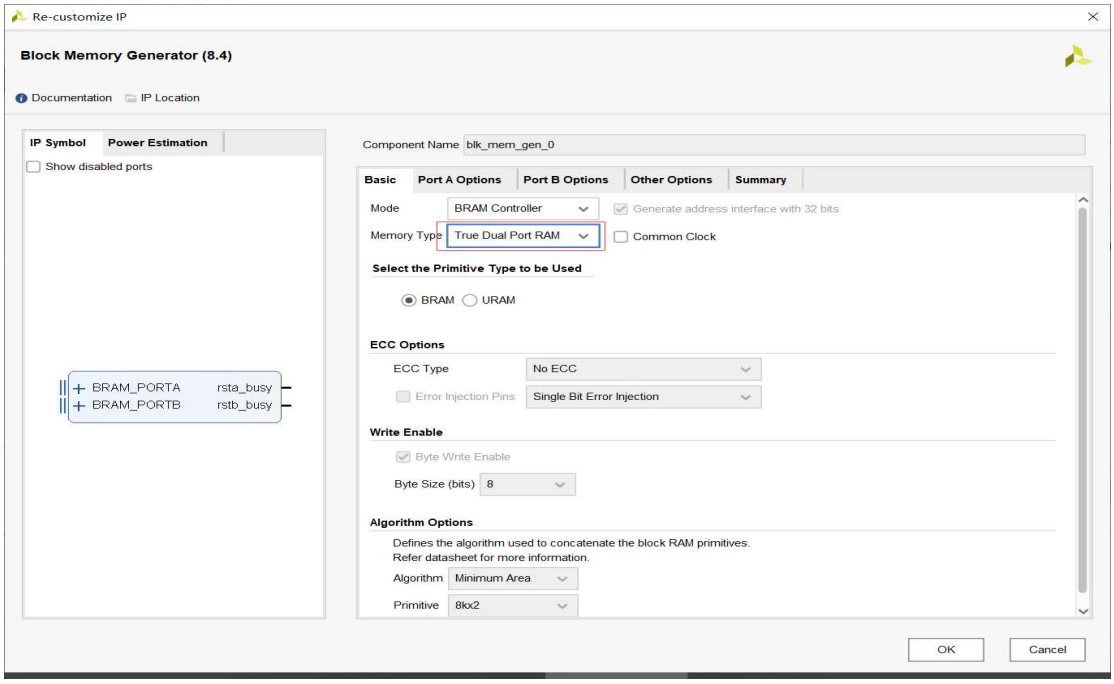


Figure 5-2 Configure axi BRAM

Uncheck Enable Safety Circuit and click OK.

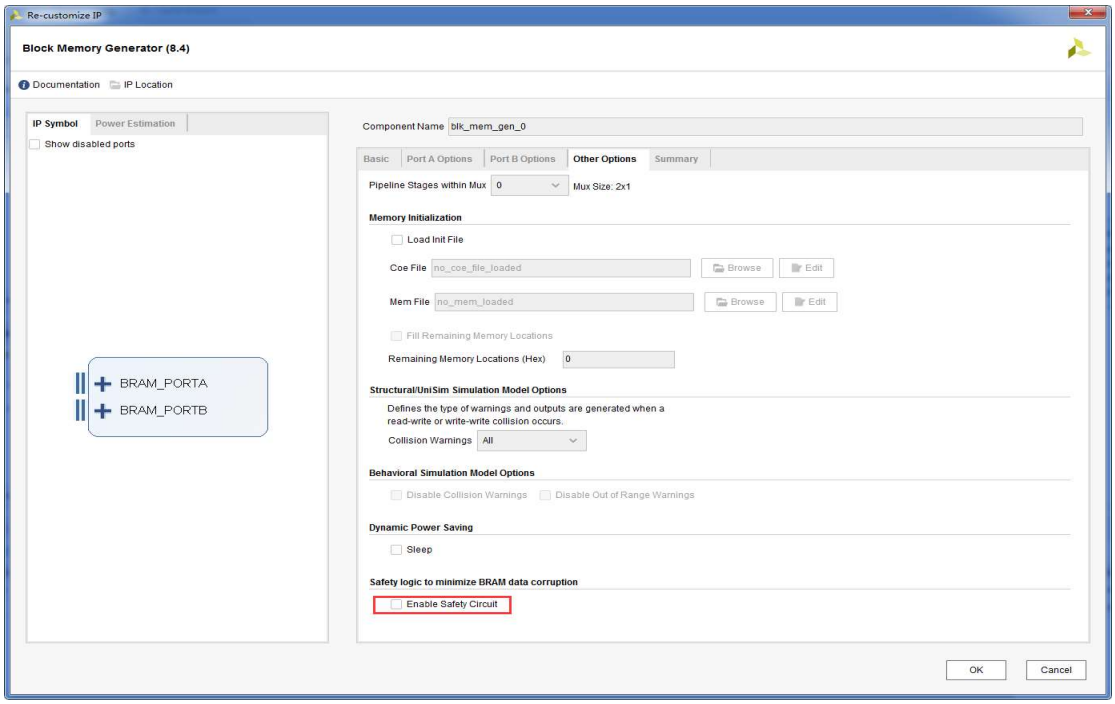


Figure 5-3 Configure AXI BRAM

(2) Keep other options as default, click ok to complete the configuration. Enter axi_bram in the search bar, double-click AXI BRAM Controller to add the axi_bram_controller core. Double-click axi_bram_controller to set the parameters,

set the Number of BRAM interfaces to 1, the other axi_bram_controller is also set like this

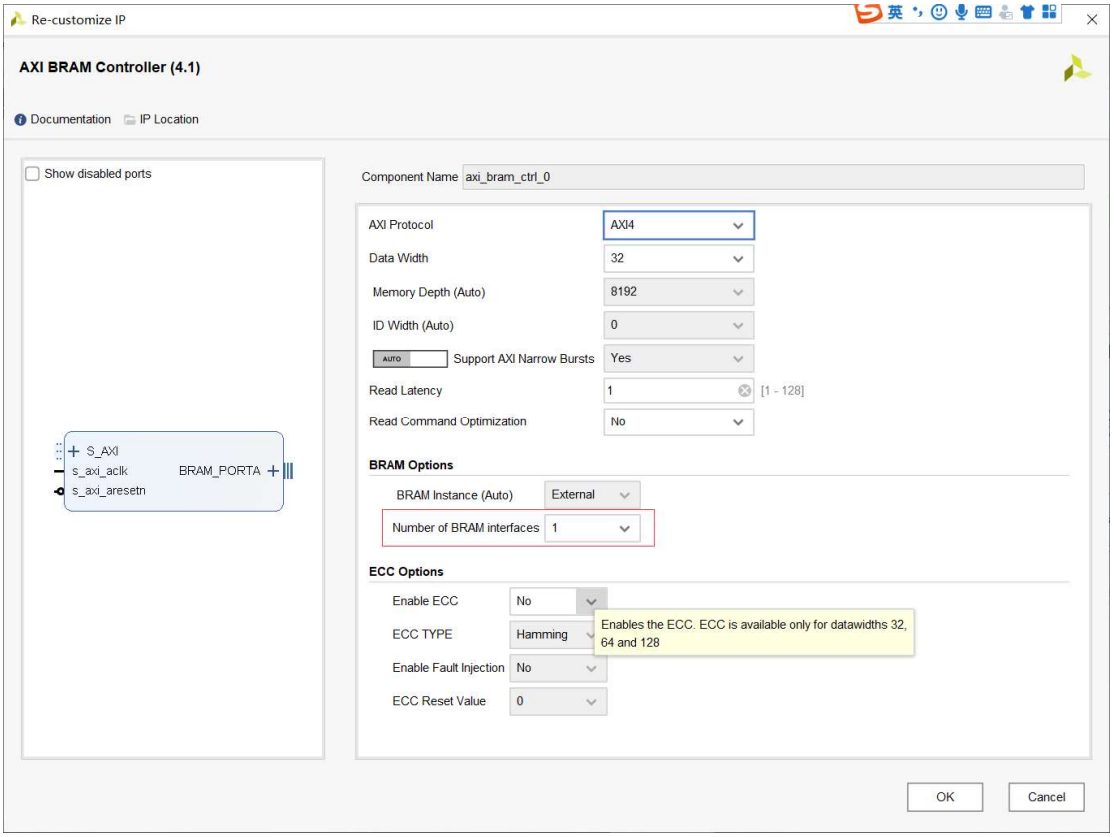


Figure 5-4 Configure BRAM controller

Click Run Block Automation->OK to automatically connect, the way is the same as in uart, check all the options, and click OK. Set address

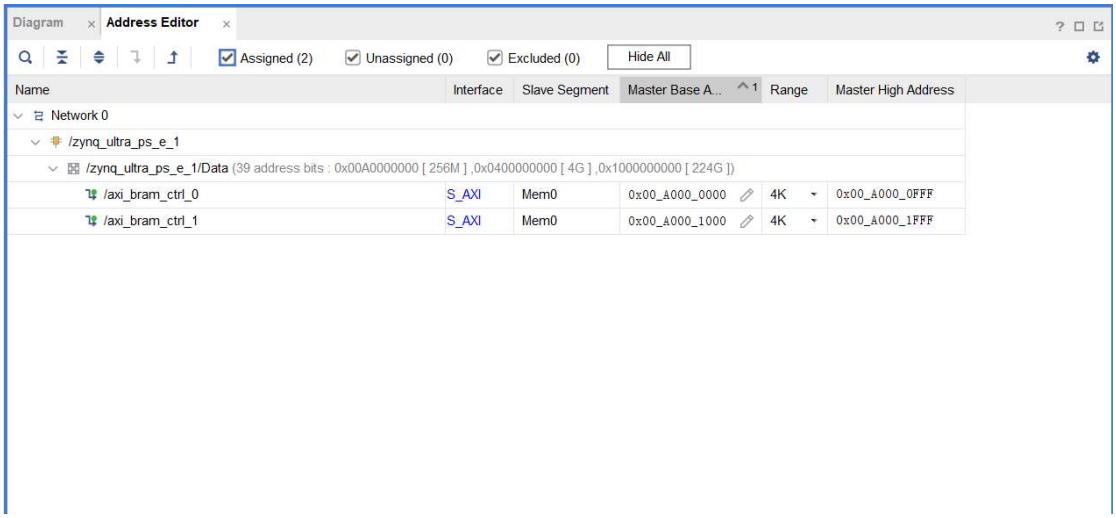


Figure 5-5 Allocating address space

The following process is consistent with the uart project, but since bram test does not use the pl-side io, there is no need to perform io constraints. To Vitis project: Enter the Vitis software, create a new project named bram_test,

select the empty template on the Templates template selection page, and then copy the program files in the routine in the generated project. The code is as follows, which is to write the numbers 0-15 into the bram register, then read them out, and then print out the read data.

```c
#include <stdio.h>
#include "xil_io.h"
#include "xparameters.h"

int main()
{
    int num;
    int rev;

    xil_printf("------The test is start...------\n\r");

    for (num = 0; num < 16; num++)
    {
        Xil_Out32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + num * 4, 0x10000000 + num); //
    }

    for (num = 0; num < 16; num++)
    {
        rev = Xil_In32(XPAR_AXI_BRAM_CTRL_1_S_AXI_BASEADDR + num * 4);
        xil_printf("The data at %x is %x \n\r", XPAR_AXI_BRAM_CTRL_1_S_AXI_BASEADDR + num * 4, rev);
    }

    xil_printf("------The test is end!------\n\r");

    return 0;
}
```

Figure 5-6 vitis code design

The Jtag debugging and SD card startup methods are the same as the first project. After the program is downloaded, the debugging serial port output results are as follows.

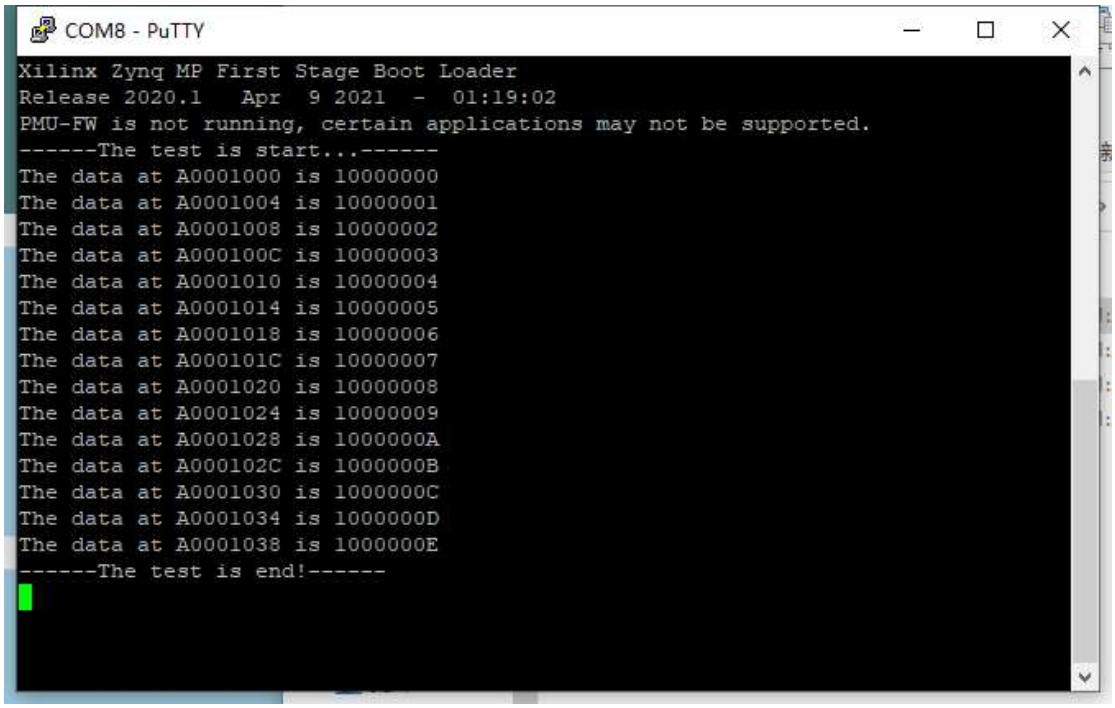


Figure 5-7 result display

5.2 AXI DMA

5.2.1AXI DMA Basis Knowledge

The background of DMA: DMA (Direct Memory Access) refers to an interface technology in which external devices directly exchange data with system memory without passing through the CPU. To read the peripheral data into the memory or transfer the memory to the peripheral, generally it must be controlled by the CPU, such as query or interrupt mode. Although the interrupt mode can improve the utilization of the CPU, there are also efficiency problems. For the case of batch data transfer, the DMA method can solve the efficiency and speed problems.

The CPU only needs to provide the address and length to the DMA, and the DMA can take over the bus. , Access the memory, and after the DMA finishes its work, inform the CPU to hand over bus control. DMA workflow: First, the CPU must receive the DMA request interrupt from the peripheral, and then the CPU interrupt, set the DMA transmission address, length, interrupt and other information, and start the DMA transmission. The next step is for the CPU to do other things. Peripherals use DMA for data transfer. Finally, the peripheral data transfer is completed, and the interrupt is sent to the CPU for the completion of the transfer. After the CPU has processed the interrupt, it will do other things. The following mainly introduces the basic situation of AXI DMA IP, here is mainly an excerpt from the content in PG021. 1.

The AXI DMA module uses three types of buses. AXI4-Lite is used to configure registers, and AXI4 Memory Map is used to interact with memory. In this module, two interfaces, AXI4 Memory Map Read and AXI4 Memory Map Write, are separated. , And they are called M_AXI_MM2S and M_AXI_S2MM respectively.

One is to read and the other is to write. This should be clear and not confused.

The AXI4 Stream interface is used to read and write to peripherals, where AXI4 Stream Master (MM2S) is used to write to the peripherals, and AXI4-Stream Slave (S2MM) is used to read to the peripherals.

It also supports Scatter/Gather functions. (MM2S stands for Memory Map to Stream, S2MM stands for Stream to Memory Map).

AXI Memory Map data width supports 32, 64, 128, 256, 512, 1024bits

AXI Stream data width supports 8, 16, 32, 64, 128, 256, 512, 1024bits

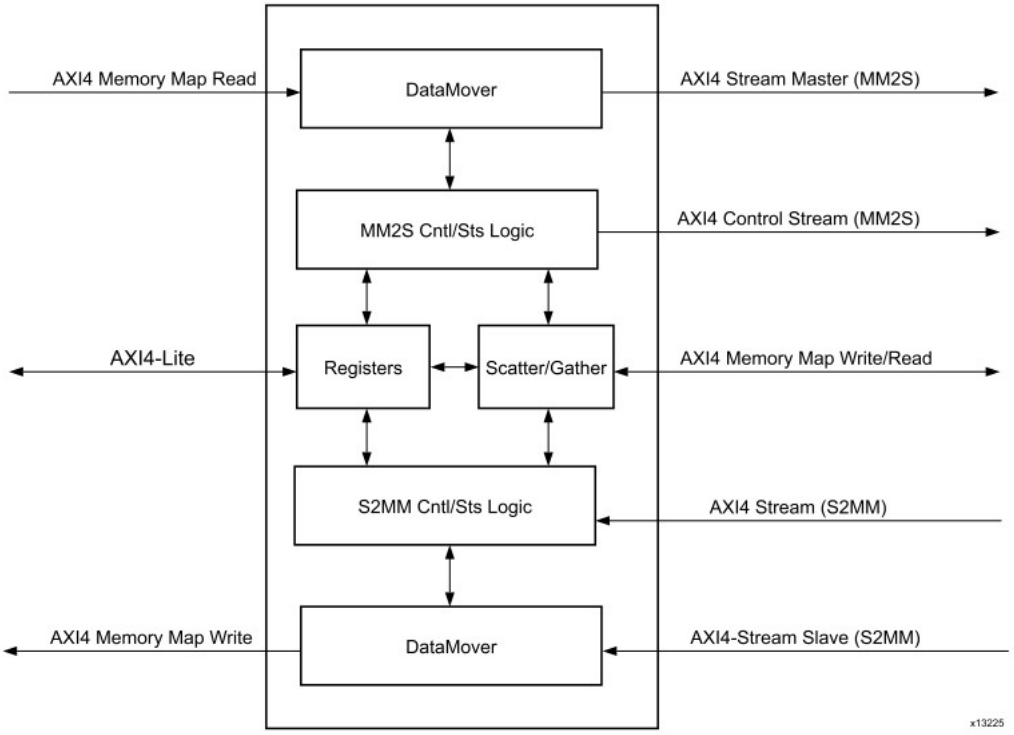


Figure 1-1: **AXI DMA Block Diagram**

Figure 5-8 AXI DMA Block Diagram

2，The direct register mode is used in this experiment. The following figure shows the register description. It is mainly divided into two parts. One is MM2S, which includes Control Register, Status Register, Source Address, and Transfer Length. The other is S2MM, which also includes Control Register and Status Register.，Destination Address, Buffer Length four parts, note that the Source Address and Destination Address here refer to the memory address.

Table 2-6: **Direct Register Mode Register Address Map**

Address Space Offset[1]	Name	Description
00h	MM2S_DMACR	MM2S DMA Control register
04h	MM2S_DMASR	MM2S DMA Status register
08h – 14h	Reserved	N/A
18h	MM2S_SA	MM2S Source Address. Lower 32 bits of address.
1Ch	MM2S_SA_MSB	MM2S Source Address. Upper 32 bits of address.
28h	MM2S_LENGTH	MM2S Transfer Length (Bytes)
30h	S2MM_DMACR	S2MM DMA Control register

Figure 5-9 Direct Register Mode Register Address Map

Address Space Offset[1]	Name	Description
34h	S2MM_DMASR	S2MM DMA Status register
38h – 44h	Reserved	N/A
48h	S2MM_DA	S2MM Destination Address. Lower 32 bit address.
4Ch	S2MM_DA_MSB	S2MM Destination Address. Upper 32 bit address.
58h	S2MM_LENGTH	S2MM Buffer Length (Bytes)

Figure 5-10 Direct Register Mode Register Address Map

3，The following is the description of the MM2S_DMACR control register. The more important ones are Bit0, Run/Stop, which means to start or stop DMA. The other content will not be described.

MM2S_DMACR (MM2S DMA Control Register – Offset 00h)

This register provides control for the Memory Map to Stream DMA Channel.

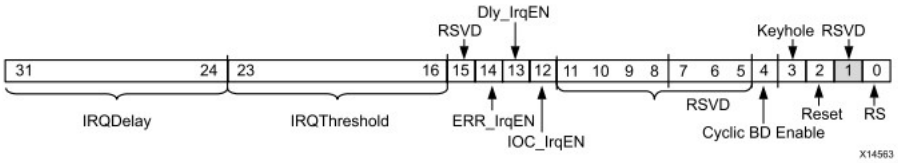


Figure 2-2: **MM2S DMACR Register**

Figure 5-11 MM2S_DMACR register

Bits	Field Name	Default Value	Access Type	Description
0	RS	0	R/W	Run / Stop control for controlling running and stopping of the DMA channel. • 0 = Stop – DMA stops when current (if any) DMA operations are complete. For Scatter / Gather Mode pending commands/ transfers are flushed or completed. AXI4-Stream outs are potentially terminated early. Descriptors in the update queue are allowed to finish updating to remote memory before engine halt. For Direct Register mode pending commands/transfers are flushed or completed. AXI4-Stream outs are potentially terminated. The halted bit in the DMA Status register asserts to 1 when the DMA engine is halted. This bit is cleared by AXI DMA hardware when an error occurs. The CPU can also choose to clear this bit to stop DMA operations. • 1 = Run – Start DMA operations. The halted bit in the DMA Status register deasserts to 0 when the DMA engine begins operations.

Figure 5-12 MM2S_DMACR register details

There are several interrupts that can be set here, IOC_IrqEn, enable completion interrupt, Dly_IrqEn enable delayed interrupt, and Err_IrqEn enable error interrupt.

12	IOC_IrqEn	0	R/W	Interrupt on Complete (IOC) Interrupt Enable. When set to 1, allows DMASR.IOC_Irq to generate an interrupt out for descriptors with the IOC bit set. • 0 = IOC Interrupt disabled • 1 = IOC Interrupt enabled
13	Dly_IrqEn	0	R/W	Interrupt on Delay Timer Interrupt Enable. When set to 1, allows DMASR.Dly_Irq to generate an interrupt out. • 0 = Delay Interrupt disabled • 1 = Delay Interrupt enabled ***Note:*** This bit is ignored when AXI DMA is configured for Direct Register Mode.
14	Err_IrqEn	0	R/W	Interrupt on Error Interrupt Enable. • 0 = Error Interrupt disabled • 1 = Error Interrupt enabled

Figure 5-13 MM2S_DMACR reg details

4，MM2S_DMASR is the description of the status register. Bits 12, 13, and 14 are the interrupt status. Write 1 to clear the interrupt.

MM2S_DMASR (MM2S DMA Status Register – Offset 04h)

This register provides the status for the Memory Map to Stream DMA Channel.

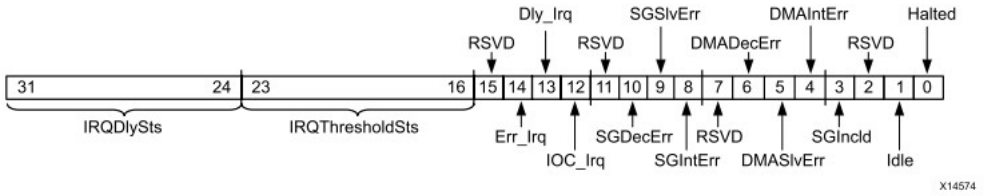


Figure 2-3: **MM2S DMASR Register**

Figure 5-14 MM2S_DMASR register

12	IOC_Irq	0	R/WC	Interrupt on Complete. When set to 1 for Scatter/Gather Mode, indicates an interrupt event was generated on completion of a descriptor. This occurs for descriptors with the End of Frame (EOF) bit set. When set to 1 for Direct Register Mode, indicates an interrupt event was generated on completion of a transfer. If the corresponding bit is enabled in the MM2S_DMACR (IOC_IrqEn = 1) and if the interrupt threshold has been met, causes an interrupt out to be generated from the AXI DMA. • 0 = No IOC Interrupt. • 1 = IOC Interrupt detected. Writing a 1 to this bit clears it.
13	Dly_Irq	0	R/WC	Interrupt on Delay. When set to 1, indicates an interrupt event was generated on delay timer timeout. If the corresponding bit is enabled in the MM2S_DMACR (Dly_IrqEn = 1), an interrupt out is generated from the AXI DMA. • 0 = No Delay Interrupt. • 1 = Delay Interrupt detected.1 = IOC Interrupt detected. Writing a 1 to this bit clears it. ***Note:*** This bit is not used and is fixed at 0 when AXI DMA is configured for Direct Register Mode.
14	Err_Irq	0	R/WC	• Interrupt on Error. When set to 1, indicates an interrupt event was generated on error. If the corresponding bit is enabled in the MM2S_DMACR (Err_IrqEn = 1), an interrupt out is generated from the AXI DMA. Writing a 1 to this bit clears it. • 0 = No error Interrupt. • 1 = Error interrupt detected.

Figure 5-14 MM2S_DMASR register details

5.2.2 Experiment Logical

This experiment is to instantiate the dma IP core on the pl side, and test the data transmission of the dma IP core through the axi bus on the PS side. The function of this experiment is that MM2S reads data from DDR3, writes to AXI Stream Data FIFO, and then from FIFO. To read and write to DDR3, to realize the loop-through test, you need to turn on the IOC_Irq of S2MM_DMACR, that is, write memory end interrupt. The functional block diagram is as follows:

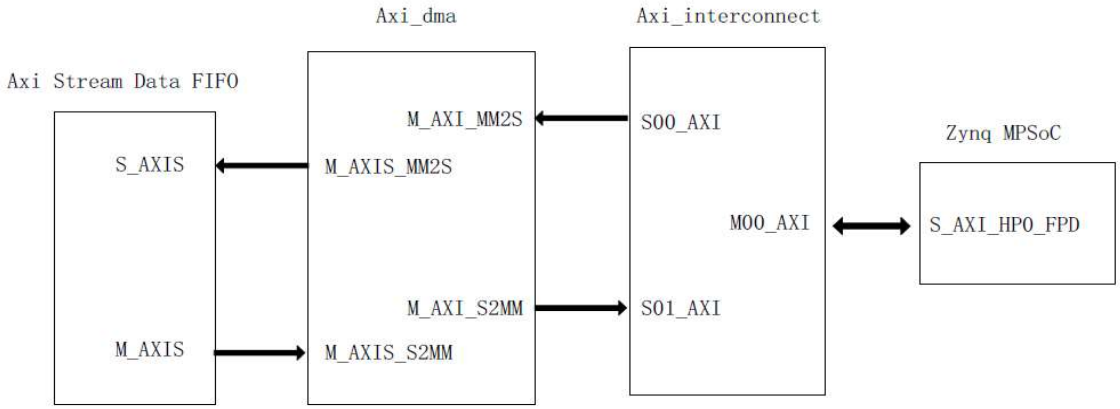


Figure 5-15 design structure

5.2.3 Experiment Steps

Vivado engineering:

Based on the "hello_world" project, save it as a dma_loop project. The corresponding design document on the CD is dma_loop.rar. Open reset, HPM0, HP0, PL to PS interrupt IRQ0. Enter dma in the search bar to call the dma IP core Double-click the DMA IP core to set the parameters, and the configuration is as follows.

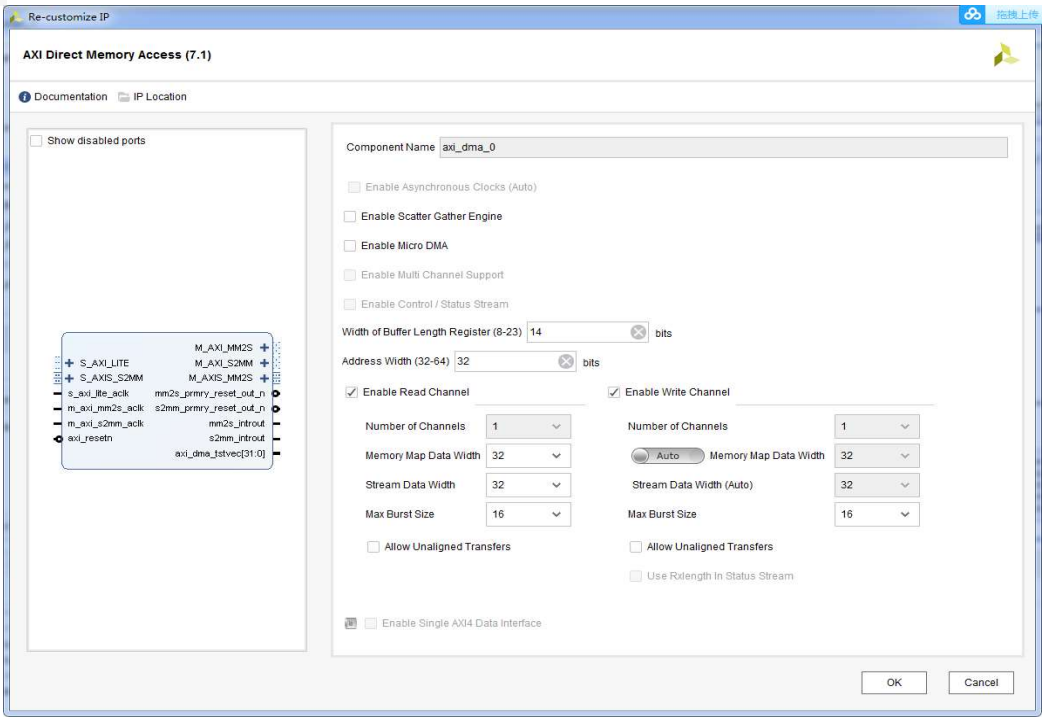


Figure 5-16 DMA configuration block diagram

Add the AXI Stream Data FIFO module and set it as follows, set the depth to 1024, TDATA Width to 4 bytes, and turn on TKEEP and TLAST.

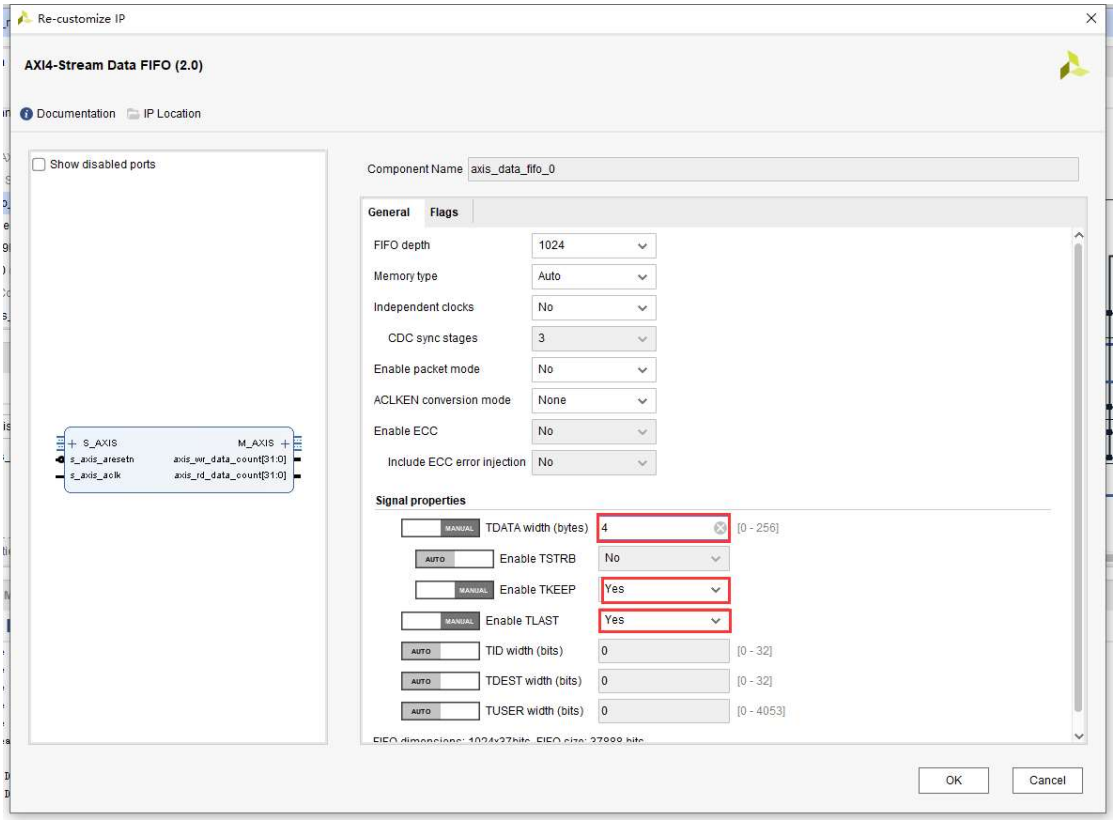


Figure 5-17 Configure FIFO

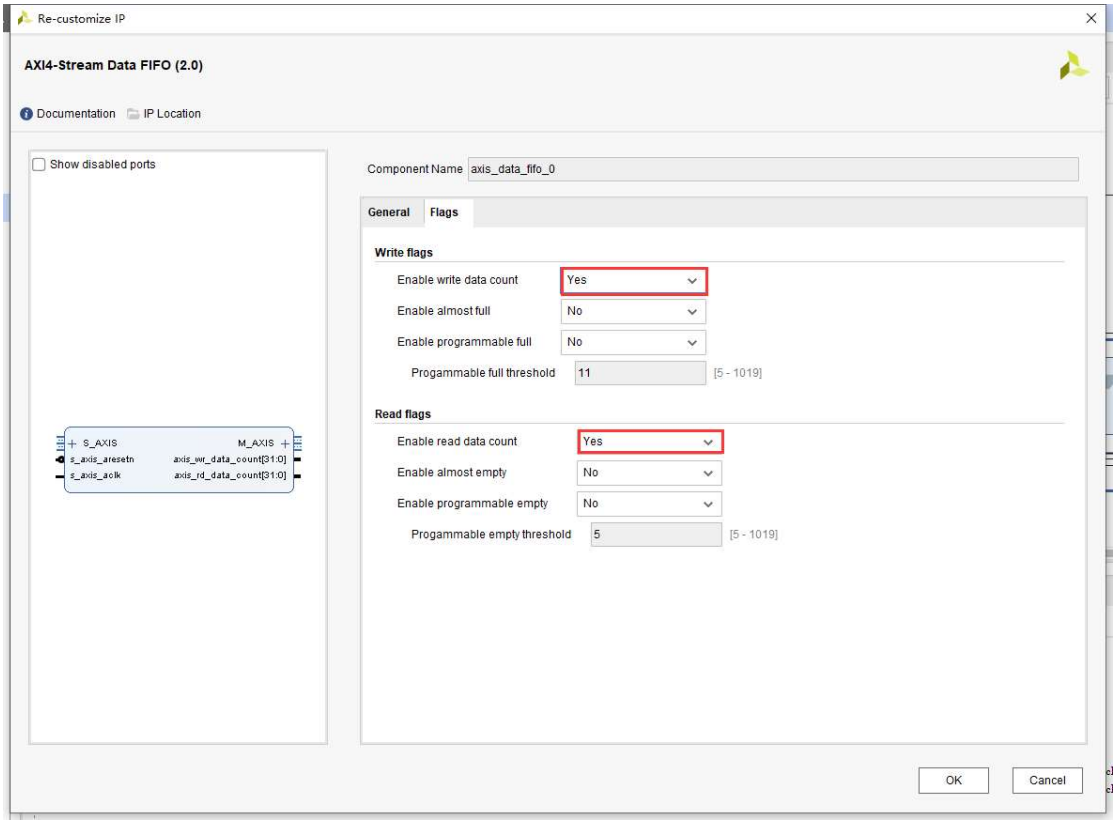


Figure 5-18 FIFO configuration

Automatically connect, and connect S_AXIS and M_AXIS of FIFO to dma, add Concat, connect MM2S and S2MM interrupt to pl_ps_irq, the final result is as shown in the figure below.

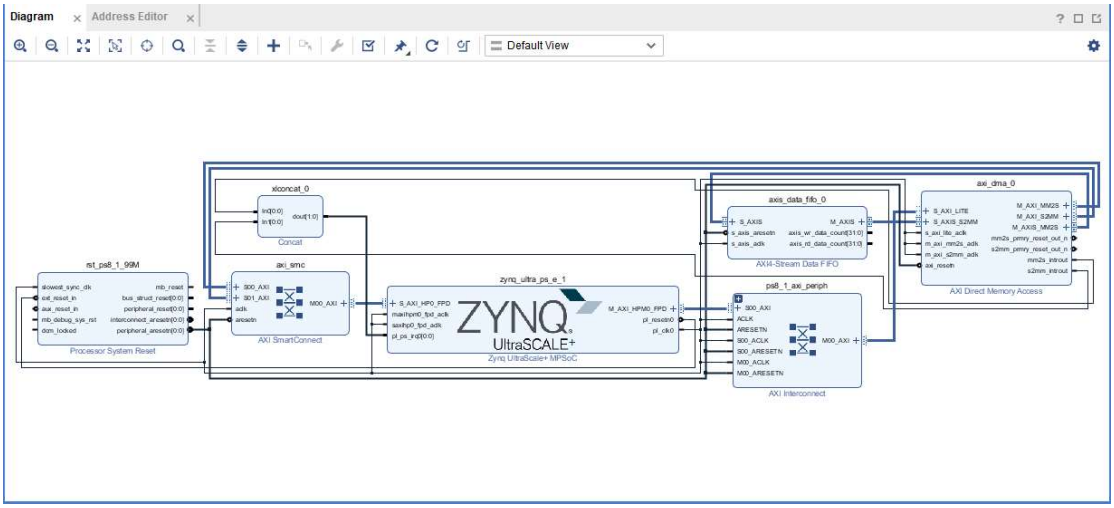


Figure 5-19 BD design block diagram

The following process is consistent with the uart project, and the same as bram test, it also does not require pin constraints. Vitis project: For the steps of creating a new project, please refer to the first project. After creating a new application project named dma_loop, open the Example Directory folder corresponding to dma in the same way as axi uart. In the opened directory, copy xaxidma_example_simple_poll.c to our project. src directory. Add a Tries Count print statement in the position below, and click Compile.

```
234
235     /* Disable interrupts, we use polling mode
236      */
237     XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK,
238                         XAXIDMA_DEVICE_TO_DMA);
239     XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK,
240                         XAXIDMA_DMA_TO_DEVICE);
241
242     Value = TEST_START_VALUE;
243
244     for(Index = 0; Index < MAX_PKT_LEN; Index ++) {
245             TxBufferPtr[Index] = Value;
246
247             Value = (Value + 1) & 0xFF;
248     }
249     /* Flush the buffers before the DMA transfer, in case the Data Cache
250      * is enabled
251      */
252     Xil_DCacheFlushRange((UINTPTR)TxBufferPtr, MAX_PKT_LEN);
253     Xil_DCacheFlushRange((UINTPTR)RxBufferPtr, MAX_PKT_LEN);
254
255     for(Index = 0; Index < Tries; Index ++) {
256
257         xil_printf("Tries Count : %d\r\n", Index);
258
259         Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR) RxBufferPtr,
260                     MAX_PKT_LEN, XAXIDMA_DEVICE_TO_DMA);
261
262         if (Status != XST_SUCCESS) {
263             return XST_FAILURE;
264         }
265
266         Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR) TxBufferPtr,
267                     MAX_PKT_LEN, XAXIDMA_DMA_TO_DEVICE);
268
269         if (Status != XST_SUCCESS) {
270             return XST_FAILURE;
271         }
272
273         while ((XAxiDma_Busy(&AxiDma,XAXIDMA_DEVICE_TO_DMA)) ||
274             (XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE))) {
275                 /* Wait */
276         }
```

Figure 5-20 Vitis source code design

The Jtag debugging and SD card startup methods are the same as the first project. After the program is downloaded, the debugging serial port output results are as follows.

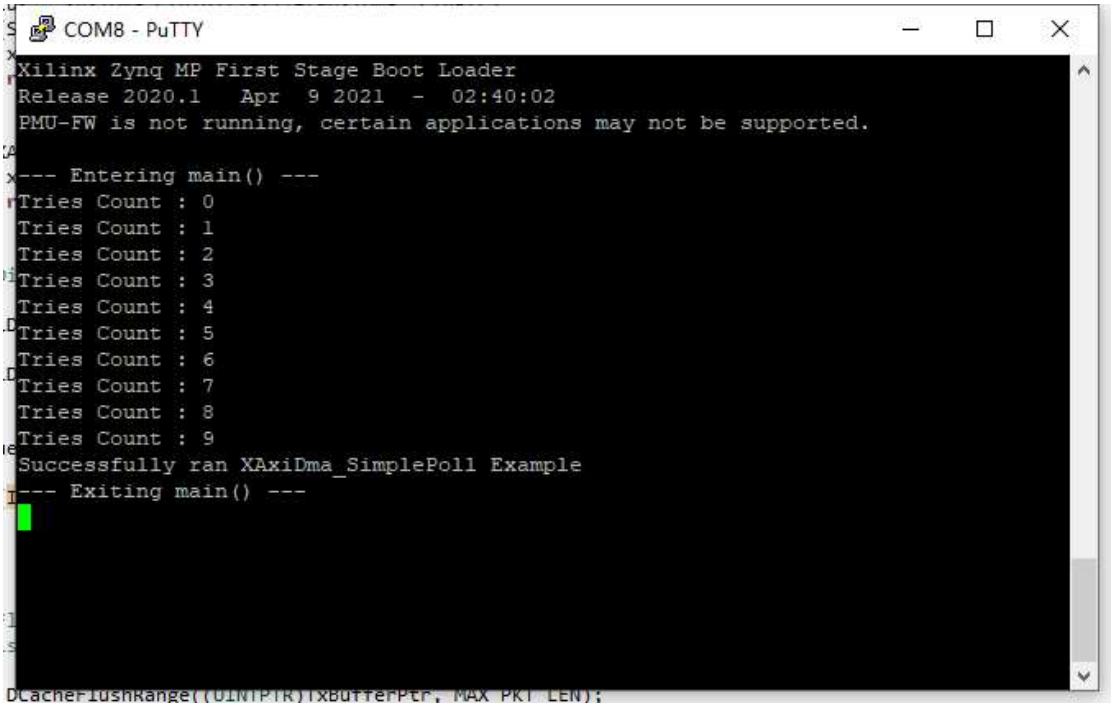


Figure 5-21 Result display

5.3Sections of this Chapter

In the first section, PS and PL realize the experiment of low-bandwidth data interaction through BRAM. The two are interconnected through the GP port, which can realize small-batch data interaction.

The second section uses DMA to access memory. DMA is a way of data interaction between PS and PL. The control of DMA is mainly on the PS side, and the PS configures the read and write of DMA.

Chapter 6 Device Module Based on AXI-Stream Interface

6.1 MIPI

This project is a sample project of MIPI, here is a brief introduction to the MIPI physical layer protocol. MIPI (Mobile Industry Processor Interface) is the abbreviation of Mobile Industry Processor Interface. MIPI (Mobile Industry Processor Interface) is an open standard for mobile application processors initiated by the MIPI Alliance. MIPI Alliance' s MIPI DSI Specification

1. Noun explanation

• DCS (DisplayCommandSet): DCS is a standardized command set for display modules in command mode.

• DSI, CSI (DisplaySerialInterface, CameraSerialInterface

• DSI defines a high-speed serial interface between the processor and the display module.

• CSI defines a high-speed serial interface between the processor and the camera module.

• D-PHY: Provides the physical layer definition of DSI and CSI 2. DSI layered structure DSI is divided into four layers, corresponding to D-PHY, DSI, DCS specifications, and the layered structure diagram is as follows:

• PHY defines the transmission medium, input/output circuits and clock and signal mechanisms.

• Lane Management layer: Send and collect data streams to each lane.

• Low Level Protocol layer: defines how to frame and parse, and error detection.

• Application layer: describe the high-level coding and parsing data flow.

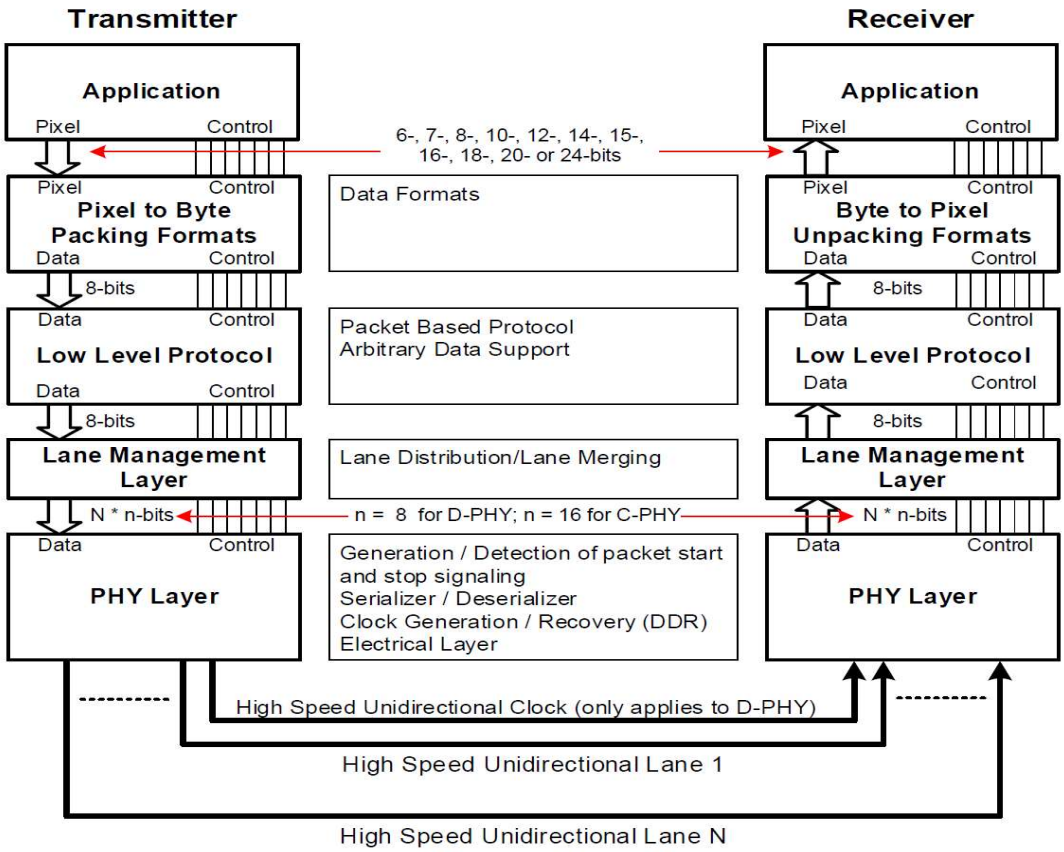


Figure 6-1 MIPI CSI-2 layer definitions

Command and Video mode

• DSI compatible peripherals support Command or Video operation mode, which mode is determined by the architecture of the peripheral

• Command mode refers to the use of sending commands and data to the controller with display buffer. The host indirectly controls peripherals through commands. Command mode uses bidirectional interface

• Video mode refers to the stream of real pixels when transmitted from the host to the peripherals. This mode can only transmit at high speed. In order to reduce complexity and save costs, a system that only uses Video mode may have only one unidirectional data path.

D-PHY introduction

1. D-PHY describes a synchronous, high-speed, low-power, low-cost PHY.

• A PHY configuration includes

• A clock lane

• One or more data lanes

• The PHY configuration of two Lanes is shown below

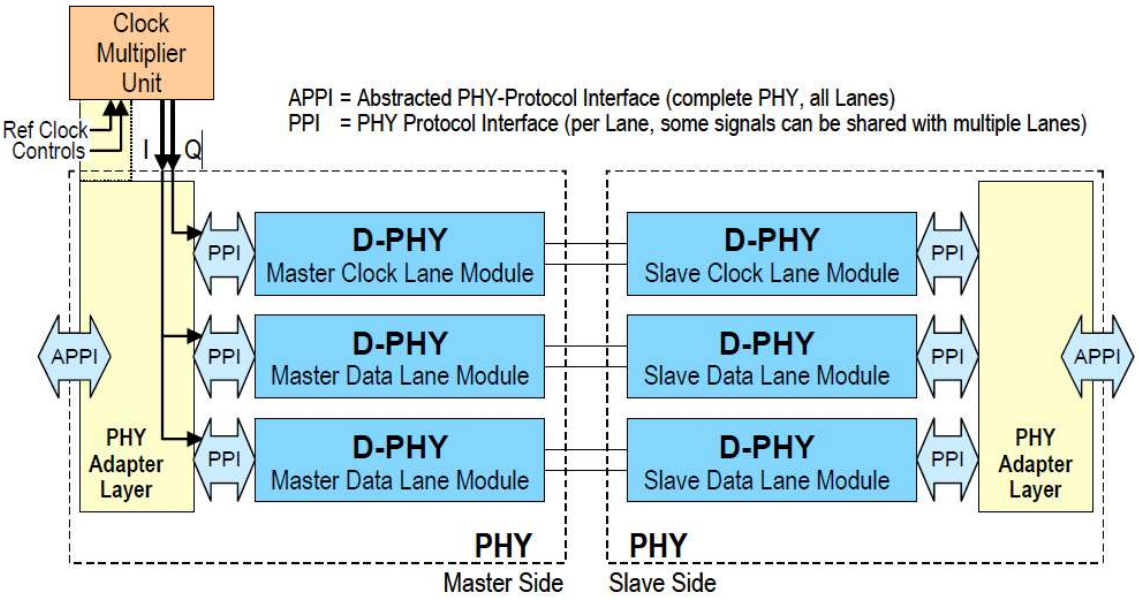


Figure 2 Two Data Lane PHY Configuration

Figure 6-1 TWO lane PHY Configuration

- Three main types of lanes
- One-way clock Lane
- One-way data Lane
- Two-way data Lane
- D-PHY transmission mode
- Low-Power signal mode (for control): 10MHz (max)
- High-Speed signal mode (for high-speed data transmission): 80Mbps ~ 1Gbps/Lane
- The D-PHY low-level protocol stipulates that the minimum data unit is one byte
- When sending data, the low bit must be in front and the high bit in the back.
- D-PHY is suitable for mobile applications
- DSI: display serial interface
- One clock lane, one or more data lanes
- CSI: Camera serial interface

2. Lane module

- PHY consists of D-PHY (Lane module)
- D-PHY may include:
- Low-power transmitter (LP-TX)
- Low power receiver (LP-RX)
- High-speed transmitter (HS-TX)

• High-speed receiver (HS-RX)
• Low-power contention detector (LP-CD)
• Three main lane types
• One-way clock Lane
• Master: HS-TX, LP-TX
• Slave: HS-RX, LP-RX
• One-way data Lane
• Master: HS-TX, LP-TX
• Slave: HS-RX, LP-RX
• Two-way data Lane
• Master, Slave: HS-TX, LP-TX, HS-RX, LP-RX, LP-CD
3. Lane status and voltage
• Lane status
• LP-00, LP-01, LP-10, LP-11 (single-ended)
• HS-0, HS-1 (differential)
• Lane voltage (typical)
• LP: 0-1.2V
• HS: 100-300mV (200mV)
4. Operation mode
• Three operating modes of Data Lane
• Escape mode, High-Speed(Burst) mode, Control mode
• The possible events starting from the stop state of the control mode are:
• Escape mode request (LP-11→LP-10→LP-00→LP-01→LP-00)
• High-Speed mode request (LP-11→LP-01→LP-00)
• Turnaround request (LP-11→LP-10→LP-00→LP-10→LP-00)
• Escape mode is a special operation of Data Lane in LP state
• In this mode, you can enter some additional functions: LPDT, ULPS, Trigger
• Data Lane enters Escape mode through LP-11→LP-10→LP-00→LP-01→LP-00
• Once entering Escape mode, the sender must send an 8-bit command to respond to the requested action
• Escape mode uses Spaced-One-Hot Encoding
•Ultra-Low Power State
• In this state, lines are in an empty state (LP-00)

• Ultra-low power consumption state of clock Lane • Clock Lane enters ULPS state through LP-11→LP-10→LP-00 • Exit this state through LP-10 → TWAKEUP → LP-11, the minimum TWAKEUP time is 1ms • High-speed data transmission

• The act of sending high-speed serial data is called high-speed data transmission or burst

• All Lanes gates start synchronously, and the end time may be different.

• The clock should be in high-speed mode

• Transmission process under each mode operation mode

•The process of entering Escape mode: LP-11→LP-10→LP-00→LP-01→LP-00→Entry Code → LPD (10MHz)

•The process of exiting Escape mode: LP-10→LP-11

•The process of entering high-speed mode: LP-11→LP-01→LP-00→SoT(00011101) → HSD (80Mbps ~ 1Gbps)

•The process of exiting high-speed mode: EoT→LP-11

•Control mode-BTA transmission process: LP-11→LP-10→LP-00→LP-10→LP-00

•Control mode-BTA receiving process: LP-00→LP-10→LP-11

• State transition diagram

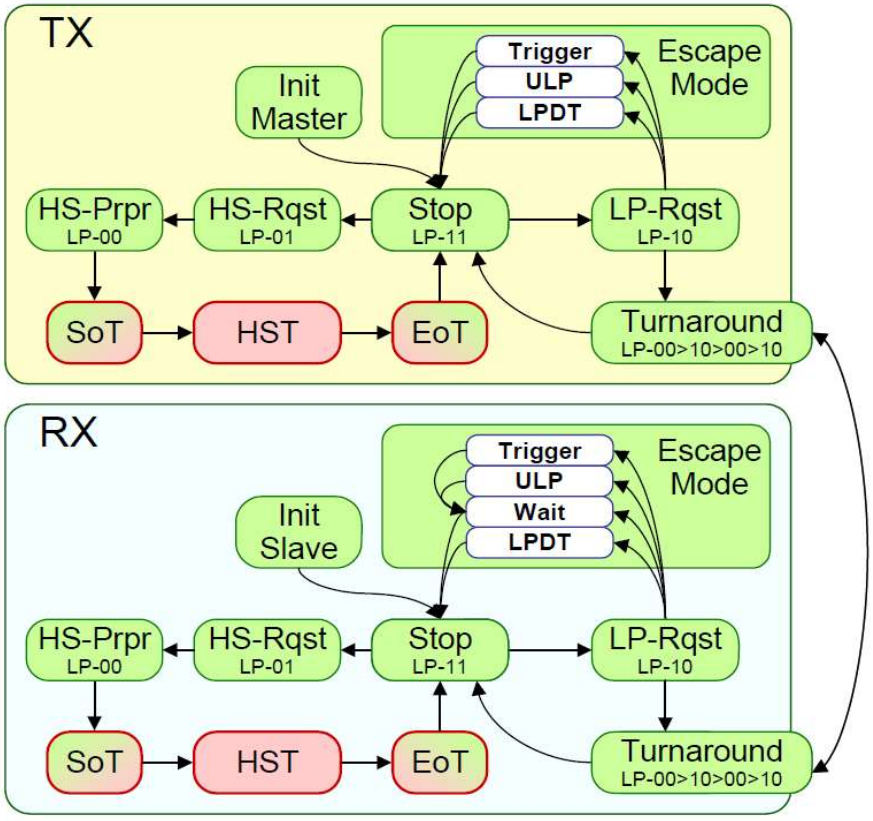


Figure 24 Data Lane Module State Diagram

Figure 6-3 Data lane Module state diagram

DSI introduction

1. DSI is a Lane expandable interface, 1 clock Lane/1-4 data Lanes

• DSI compatible peripherals support 1 or 2 basic operation modes:

• Command Mode (similar to MPU interface)

• Video Mode (similar to RGB interface)-must use high-speed mode to transmit data, support 3 formats of data transmission

• Non-Burst sync pulse mode

• Non-Burst synchronous event mode

• Burst mode

• Transmission mode:

• High-Speed signaling mode

• Low-Power signaling mode-Only data lane 0 is used (the clock is derived from the exclusive OR of DP and DN).

• Frame type

• Short frame: 4 bytes (fixed)

• Long frame: 6~65541 bytes (variable)

• Two data Lane high-speed transmission examples

2. Short frame structure

• Frame header (4 bytes)

• Data identification (DI) 1 byte

• Frame data-2 bytes (the length is fixed at 2 bytes)

• Error detection (ECC) 1 byte

• Frame size

• The length is fixed at 4 bytes

3. Long frame structure

• Frame header (4 bytes)

• Data identification (DI) 1 byte

• Data count-2 bytes (number of data filling)

• Error detection (ECC) 1 byte

• Data filling (0~65535 bytes)

• Length=WC*Byte

• End of frame: checksum (2 bytes)

• Frame size:

• 4 + (0~65535) + 2 = 6 ~ 65541 bytes

4. Frame data type

Table 16 Data Types for Processor-sourced Packets

Data Type, hex	Data Type, binary	Description	Packet Size
0x01	00 0001	Sync Event, V Sync Start	Short
0x11	01 0001	Sync Event, V Sync End	Short
0x21	10 0001	Sync Event, H Sync Start	Short
0x31	11 0001	Sync Event, H Sync End	Short
0x08	00 1000	End of Transmission packet (EoTp)	Short
0x02	00 0010	Color Mode (CM) Off Command	Short
0x12	01 0010	Color Mode (CM) On Command	Short
0x22	10 0010	Shut Down Peripheral Command	Short
0x32	11 0010	Turn On Peripheral Command	Short
0x03	00 0011	Generic Short WRITE, no parameters	Short
0x13	01 0011	Generic Short WRITE, 1 parameter	Short
0x23	10 0011	Generic Short WRITE, 2 parameters	Short
0x04	00 0100	Generic READ, no parameters	Short
0x14	01 0100	Generic READ, 1 parameter	Short
0x24	10 0100	Generic READ, 2 parameters	Short
0x05	00 0101	DCS Short WRITE, no parameters	Short
0x15	01 0101	DCS Short WRITE, 1 parameter	Short
0x06	00 0110	DCS READ, no parameters	Short
0x37	11 0111	Set Maximum Return Packet Size	Short
0x09	00 1001	Null Packet, no data	Long
0x19	01 1001	Blanking Packet, no data	Long
0x29	10 1001	Generic Long Write	Long
0x39	11 1001	DCS Long Write/write_LUT Command Packet	Long
0x0C	00 1100	Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format	Long
0x1C	01 1100	Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format	Long
0x2C	10 1100	Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format	Long
0x0D	00 1101	Packed Pixel Stream, 30-bit RGB, 10-10-10 Format	Long
0x1D	01 1101	Packed Pixel Stream, 36-bit RGB, 12-12-12 Format	Long

Data Type, hex	Data Type, binary	Description	Packet Size
0x3D	11 1101	Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format	Long
0x0E	00 1110	Packed Pixel Stream, 16-bit RGB, 5-6-5 Format	Long
0x1E	01 1110	Packed Pixel Stream, 18-bit RGB, 6-6-6 Format	Long
0x2E	10 1110	Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format	Long
0x3E	11 1110	Packed Pixel Stream, 24-bit RGB, 8-8-8 Format	Long
0xX0 and 0xXF, unspecified	XX 0000 XX 1111	DO NOT USE All unspecified codes are reserved	

6.1.1 MIPI_CSI2_Rx_Subsystem Basis Knowledge

The Xilinx MIPI CSI-2 RX controller implements a camera serial interface between the camera sensor and the programmable device that performs baseband processing. Due to the development of higher resolution cameras, the bandwidth requirements of camera sensor interfaces have increased.

Traditional parallel interfaces require more and more signal lines, resulting in higher power consumption. New high-speed serial interfaces, such as the MIPI CSI specification, can meet these ever-expanding bandwidth requirements without sacrificing power.

MIPI is a set of protocols defined by mobile industry organizations to standardize all interfaces in mobile platforms such as mobile phones and tablets. However, the large capacity and economies of scale in the mobile industry are forcing other applications to adopt these standards.

Therefore, MIPI-based camera sensors are increasingly used in emerging applications such as driver assistance technology, video security surveillance cameras, video conferencing, and virtual and augmented reality in automotive applications. Block diagram of delay calculation about IP

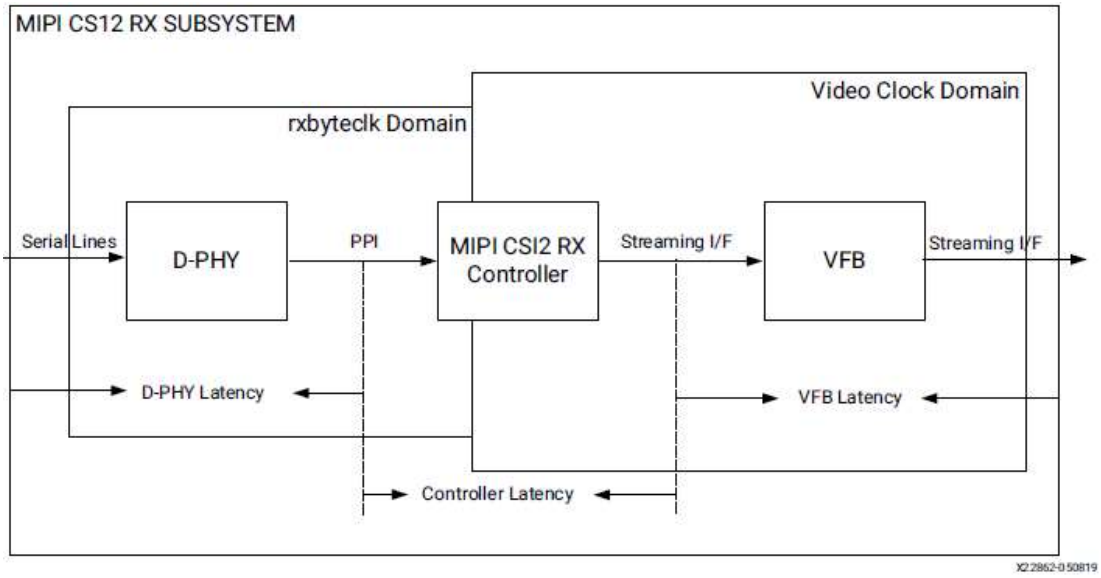


Figure 2-1: **MIPI CSI-2 RX Subsystem Latency Calculation**

Figure 6-4 MIPI CSI-2 Rx Subsystem latency calculation

Interface：

Port Descriptions

The MIPI CSI-2 RX Subsystem I/O signals are described in Table 2-5.

Table 2-5: **Port Descriptions**

Signal Name	Direction	Description
lite_aclk	Input	AXI clock
lite_aresetn	Input	AXI reset. Active-Low
S00_AXI*		AXI4-Lite interface, defined in the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 7]
dphy_clk_200M	Input	Clock for D-PHY core. Must be 200 MHz.
video_aclk	Input	Subsystem clock
video_aresetn(1)	Input	Subsystem reset. Active-Low.
AXI4-Stream Video Interface when Video Format Bridge is Present		
video_out_tvalid	Output	Data valid

Figure 6-5 Port description

Table 2-5: **Port Descriptions *(Cont'd)***

Signal Name	Direction	Description
video_out_tready	Input	Slave ready to accept the data
video_out_tuser[n-1:0]	Output	n is based on TUSER width selected in the Vivado IDE 95-80 CRC(3) 79-72 ECC 71-70 Reserved 69-64 Data Type 63-48 Word Count 47-32 Line Number 31-16 Frame Number 15-2 Reserved 1 Packet Error 0 Start of Frame(2)
video_out_tlast	Output	End of line
video_out_tdata[n-1:0]	Output	Data n is based on Data type and number of pixels selected in the Vivado IDE (see video_out Port Width).
video_out_tdest[9:0]	Output	9-4 Data Type 3-0 Virtual Channel Identifier (VC) NOTE: The TDEST value stays constant throughout the line.
AXI4-Stream Interface when Embedded Non-image Interface is Selected		
emb_nonimg_tdata[n-1:0]	Output	Data n is based on Data type selected in the Vivado IDE (see Table 1-1).
emb_nonimg_tdest[3:0]	Output	Specifies the Virtual Channel Identifier (VC) value of the embedded non-image packet
emb_nonimg_tkeep[n/8-1:0]	Output	Specifies valid bytes
emb_nonimg_tlast	Output	End of line
emb_nonimg_tready	Input	Slave ready to accept data

Figure 6-6 Port description

Table 3-1: **Subsystem Clocks**

Clock Name	Description
lite_aclk(1)	AXI4-Lite clock used by the register interface of all IP cores in the subsystem.
video_aclk(2)	Clock used as core clock for all IP cores in the subsystem.
dphy_clk_200M	See the *MIPI D-PHY LogiCORE IP Product Guide* (PG202) [Ref 3] for information on this clock.
clkoutphy_out	The clkoutphy_out signal is generated within the PLL with 2500 Mb/s line rate when the **Include Shared logic in core** option is selected. ***Note:*** When Deskew detection is enabled, the clockoutphy_out signal is generated with the same line rate same as the subsystem line rate.
clkoutphy_in	The clkoutphy_in signal should be connected to the clkoutphy_out signal generated when the **Include Shared logic in core** option is selected.
rxbyteclkhs_cnts_out	The rxbyteclkhs_cnts_out is the continuous clock signal generated within the PLL with the same frequency as rxbyteclkhs when the **Include Shared logic in core** option is selected and line rates are greater than 1500 Mb/s.
rxbyteclkhs_cnts_in	The rxbyteclkhs_cnts_in signal should be connected to the rxbyteclkhs_cnts_out signal generated when the **Include Shared logic in core** option is selected and line rates are greater than 1500 Mb/s.

Notes:

1. The lite_aclk clock should be less than or equal to video_aclk.
2. The maximum recommended video clock to meet timing is 250 MHz for UltraScale+ devices and 175 MHz for 7 series devices. If required, a higher throughput can be achieved by increasing the **Pixels per clock** option from Single to Dual or Quad.

Figure 6-7 Clock description

Resets

The subsystem has two reset ports:

- `lite_aresetn`: Active-Low reset for the AXI4-Lite register interface.
- `video_aresetn`: Active-Low reset for the subsystem blocks.

The duration of `video_aresetn` should be a minimum of 40 `dphy_clk_200M cycles` to propagate the reset throughout the system. See Figure 3-8.

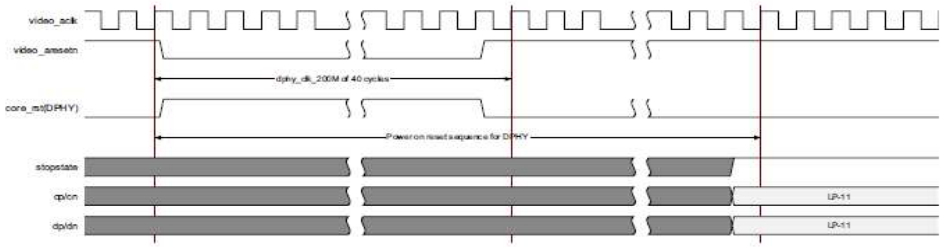


Figure 3-8: **MIPI CSI-2 RX Reset**

Table 3-3 summarizes all resets available to the MIPI CSI-2 RX Subsystem and the components affected by them.

Table 3-3: **Resets**

Sub-core	Lite_aresetn	Video_aresetn
MIPI CSI-2 RX Controller	Connected to s_axi_aresetn core port	Connected to m_axis_aresetn core port
MIPI D-PHY	Connected to s_axi_aresetn core port	Inverted signal connected to core_rst core port
Video Format Bridge	N/A	Connected to s_axis_aresetn core port
AXI Crossbar	Connected to aresetn core port	N/A

Note: The effect of each reset (`lite_resetn`, `video_aresetn`) is determined by the ports of the sub-cores to which they are connected. See the individual sub-core product guides for the effect of each reset signal.

Figure 6-8 Reset description

6.1.2 Experiment Logical

This example uses the camera of IMX334, the MIPI physical interface of the board, as shown in the figure

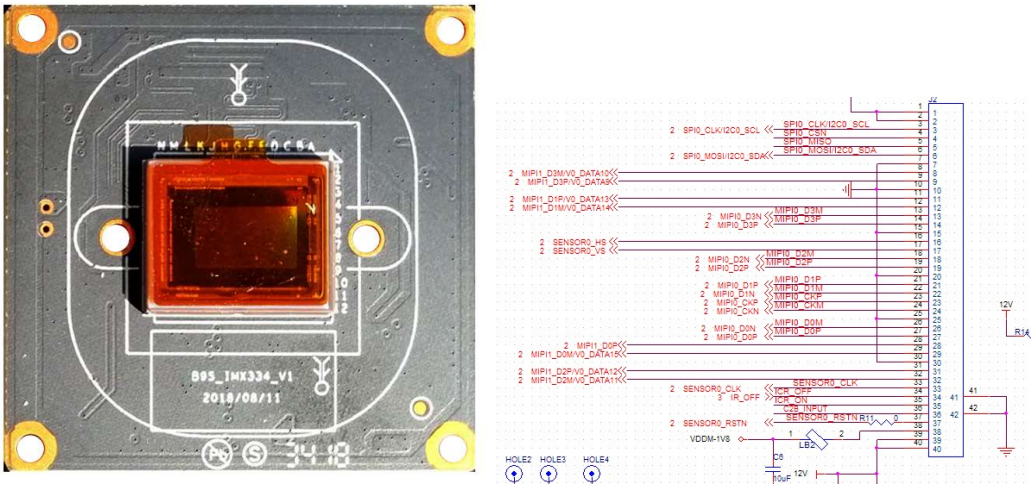


Figure 6-9 The physical and MIPI interface of the MIPI camera daughter board used

MIPI physical interface on the hardware platform used ,

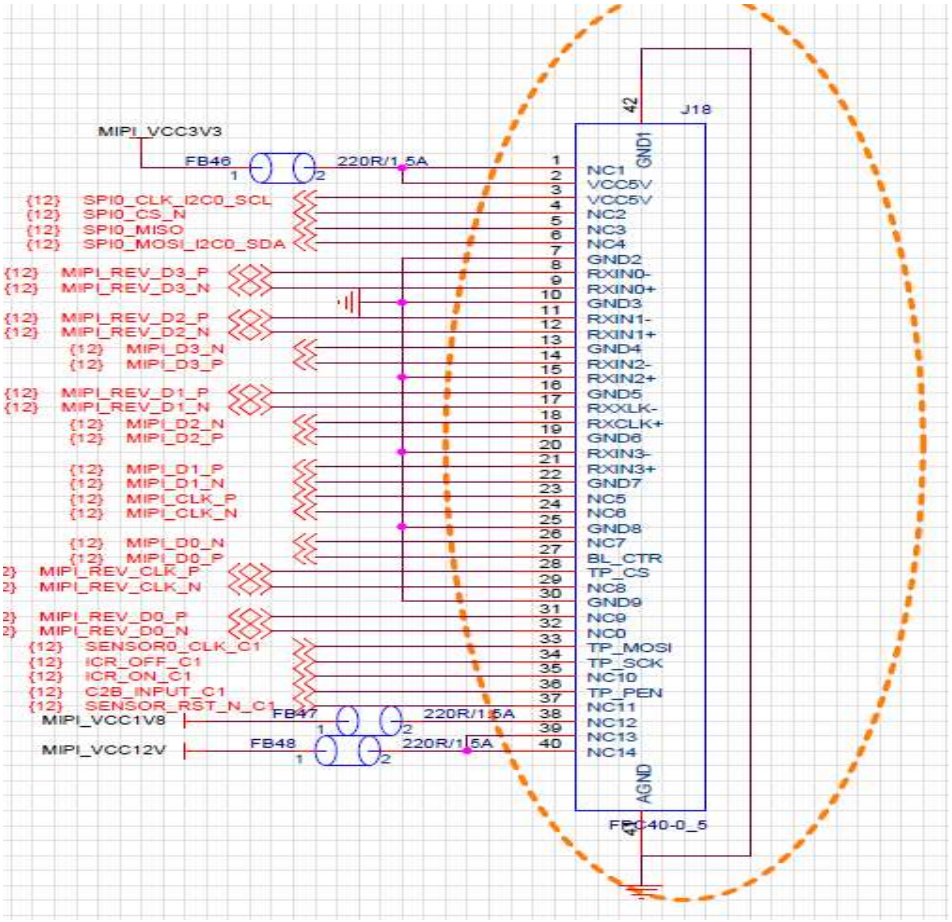


Figure 6-10 MIPI physical interface on schematic

ZU5EV pins for MIPI ：

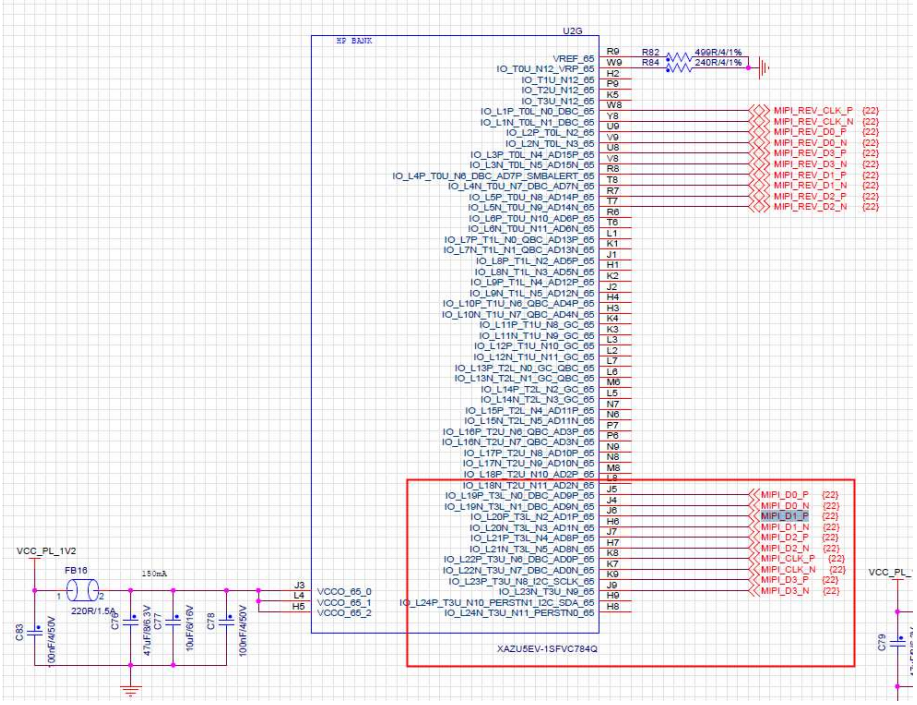


Figure 6-11 Schematic diagram of XAZU5EV MIPI pin connection

The pin instantiated by Xilinx MIPI CSI-2 RX is the MIPI access pin in the figure aboveTherefore, MIPI is linked to the IP of the receiving end of the FPGA chip on the hardware, and the subsequent processing is some processing on the MIPI access data format, and the purpose is to output the video taken by the MIPI camera to the display. Here is a solution: MIPI's RAW10 format data-axis_subset_converter-Sensor Demosaic-Gamma LUT-Video processing system (RAW2RGB)-embedded white balance module-Video processing system (Scale)-Video Frame Buffer Write- -Axi data fifo-AXI Inteconnection-PS DDR-DP interface-display.

6.1.3Experiment Steps

1. According to the "Hello world" basic configuration project in Chapter 3, save a file and rename it MIPI_DP_4G, which corresponds to MIPI_DP_4G.rar in the CD-ROM file.

2. Establish a data processing link on the PL side according to the above solution. The configuration of each module is not described in detail here. Please configure it according to the actual situation in the project. As shown below:

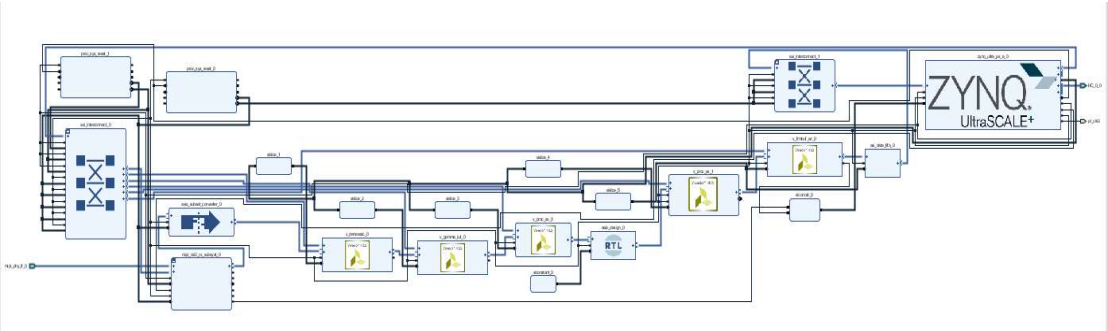


Figure 6-12 MIPI video capture design block diagram

Pay attention to the clock connection and reset in the project. After completing the module configuration and wiring, you can perform physical constraints and timing constraints. Because the total project resource consumption is not very large, you can automatically select layout and routing. After passing, there is no timing violation Can generate bitstream. Then export the hardware platform document XSA document. The follow-up is the driver development of the linux system. The modules using the AXI-lite bus need to be driven and developed, and then start the production of the boot.bin file. The production process is described in detail in the linux document, how to use the xsa document to make BOOT.bin. This piece is software-related content, you can refer to our software-related documents. The update process will not be described in detail, and the final result will be an output of successful update. Secondly, power off the hardware platform and set it to QSPI startup. After power on again, enter /mnt/mmcblk1p1 to do the necessary work for program operation. After correctly connecting the dp cable of the hardware and the monitor, you can see the image output of the monitor.

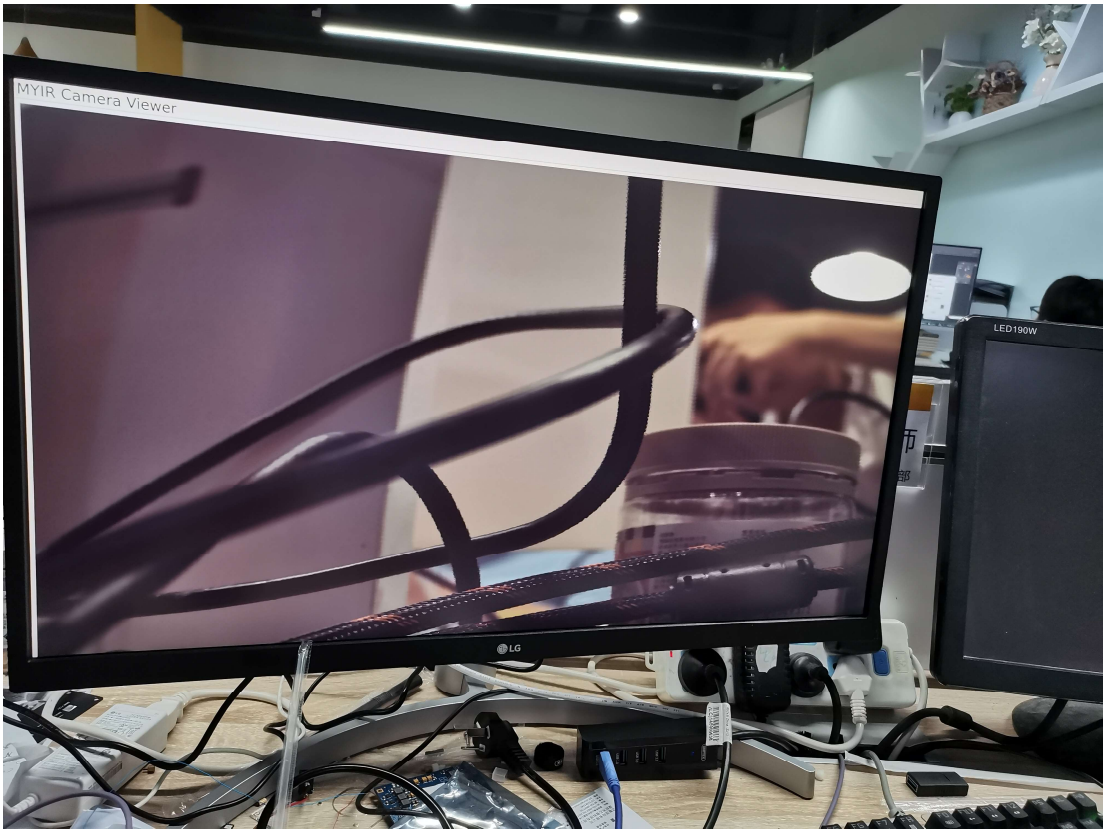


Figure 6-13 DP display link display result

6.1.4How to Add White Balance Module

In the vivado project, a design file is added, and then right-click in the block design block diagram to choose to add a module, as shown in the figure

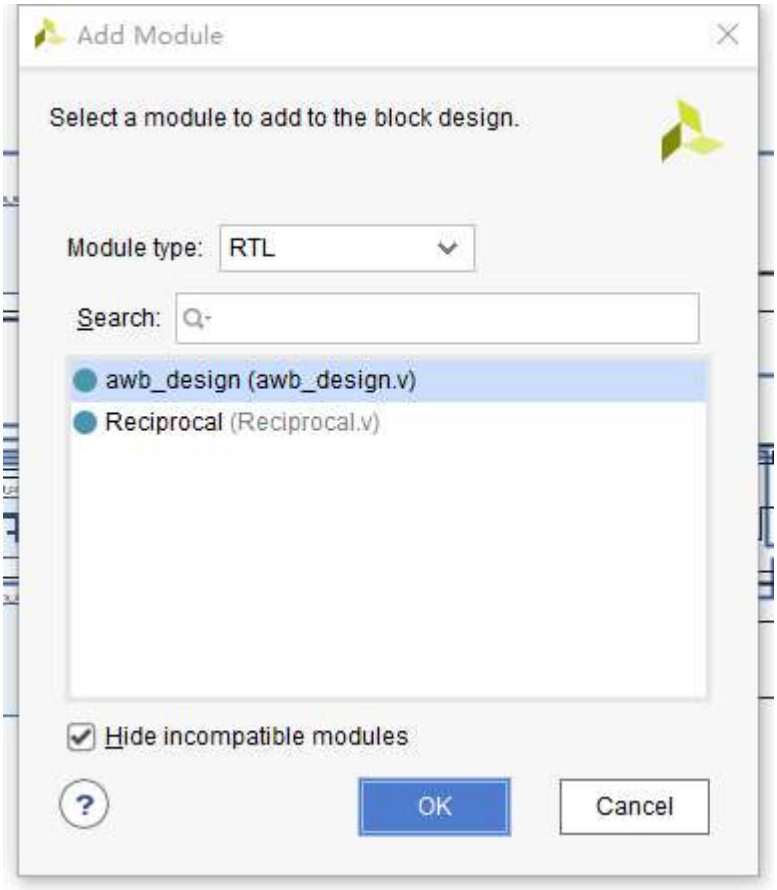


Figure 6-14 Block diagram of adding

RTL module The way of adding here is added in block design. The added position is after conversion to RGB format data. You can also choose to create a new project, custom IP, and there are two custom IPs. The first chapter is the IP defined by user logic, and the second is to define an AXI peripheral.

6.2 VCU

6.2.1VCU Basis Knowledge

The LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) core supports multi-standard videoencoding and decoding, including support for the high-efficiency Video Coding (HEVC) andAdvanced Video Coding (AVC) H.264 standards. The unit contains both encode (compress) anddecode (decompress) functions, and is capable of simultaneous encode and decode.

The VCU is an integrated block in the programmable logic (PL) of selected Zynq UltraScale+MPSoCs with no direct connections to the processing system (PS), and contains encoder anddecoder interfaces. The VCU also contains additional functions that facilitate the interfacebetween the VCU and the PL. VCU

operation requires the application processing unit (APU) toservice interrupts to coordinate data transfer. The encoder is controlled by the APU through atask list prepared in advance, and the APU response time is not in the execution critical path. TheVCU has no audio support. Audio encoding and decoding can be done in software using the PSor through soft IP in the PL. The following figure shows the top-level block diagram with the VCUcore.

Figure 1: **Top-level Block Diagram**

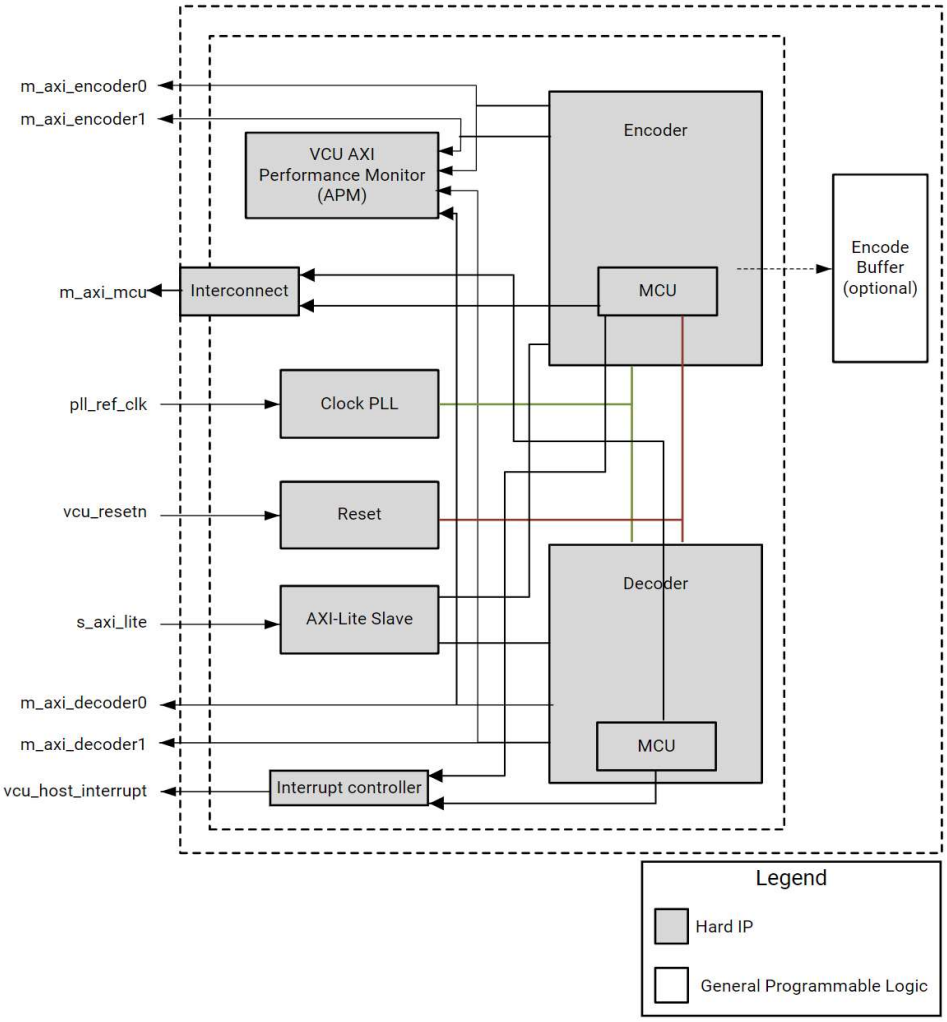


Figure 6-15 Top-level block diagram

The encoder engine is designed to process video streams using the HEVC (ISO/IEC 23008-2high-efficiency Video Coding) and AVC (ISO/IEC 14496-10 Advanced Video Coding) standards. Itprovides complete support for these standards, including support for 8-bit and 10-bit color, Y-only(monochrome), 4:2:0 and 4:2:2 Chroma formats, up to 4K UHD at 60 Hz performance. Theencoder contains global registers, an interrupt controller, and a timer. The encoder is controlledby a microcontroller (MCU) subsystem. VCU applications running on the

APU use the Xilinx®VCU Control Software library API to interact with the encoder microcontroller. Themicrocontroller firmware (MCU Firmware) is not user modifiable.

A 32-bit AXI4-Lite interface is used by the APU to control the MCU (to configure encodingparameters). Two 128-bit AXI4 master interfaces are used to move video data and metadata toand from the system memory. A 32-bit AXI4 master interface is used to fetch the MCU software(instruction cache interface) and load/store additional MCU data (data cache interface).

The Decoder block is capable of processing video streams using the HEVC (ISO/IEC 23008-2High Efficiency Video Coding) and AVC (ISO/IEC 14496-10 Advanced Video Coding) standards.It provides a complete support for these standards, including support for 8-bit and 10-bit colordepth, Y-only (monochrome), 4:2:0 and 4:2:2 Chroma formats, up to 4K UHD at 60 Hzperformance. It also contains global registers, an interrupt controller, and a timer.

The VCU decoder is controlled by a microcontroller (MCU) subsystem. A 32-bit AXI4-Lite slaveinterface is used by the APU to control the MCU. Two 128-bit AXI4 master interfaces are used tomove video data and metadata to and from the system memory. A 32-bit AXI4 master interfaceis used to fetch the MCU software (instruction cache interface) and load/store additional MCUdata (data cache interface). VCU applications running on the APU use the Xilinx VCU Control

Software library API to interact with the decoder microcontroller. The microcontroller firmware is not user modifiable.The decoder includes control registers, a bridge unit and a set of internal memories. The bridgeunit manages the request arbitration, burst addresses, and burst lengths for all external memoryaccesses required by the decoder.

The encoder and decoder blocks each implement a 32-bit microcontroller unit (MCU) to handleinteraction with the hardware blocks. The MCU receives commands from the APU, parses thecommand into multiple slice- or tile-level commands, and executes them on the encoder anddecoder blocks. After the command is executed, the MCU communicates the status to the APUand the process is repeated.

The VCU core is a dedicated circuitry located in the PL to enable maximum flexibility for a wideselection of use cases, memory bandwidth being a key driver. Whether the application requiressimultaneous 4K UHD at 60 Hz encoding and decoding or a single SD stream to be processed, asystem design and memory topology can be implemented that balances performance,optimization, and integration for the specific use case. The following figure shows the use caseexample where the VCU core works with the PS and the PL DDR external memory.

Figure 2: **VCU Application**

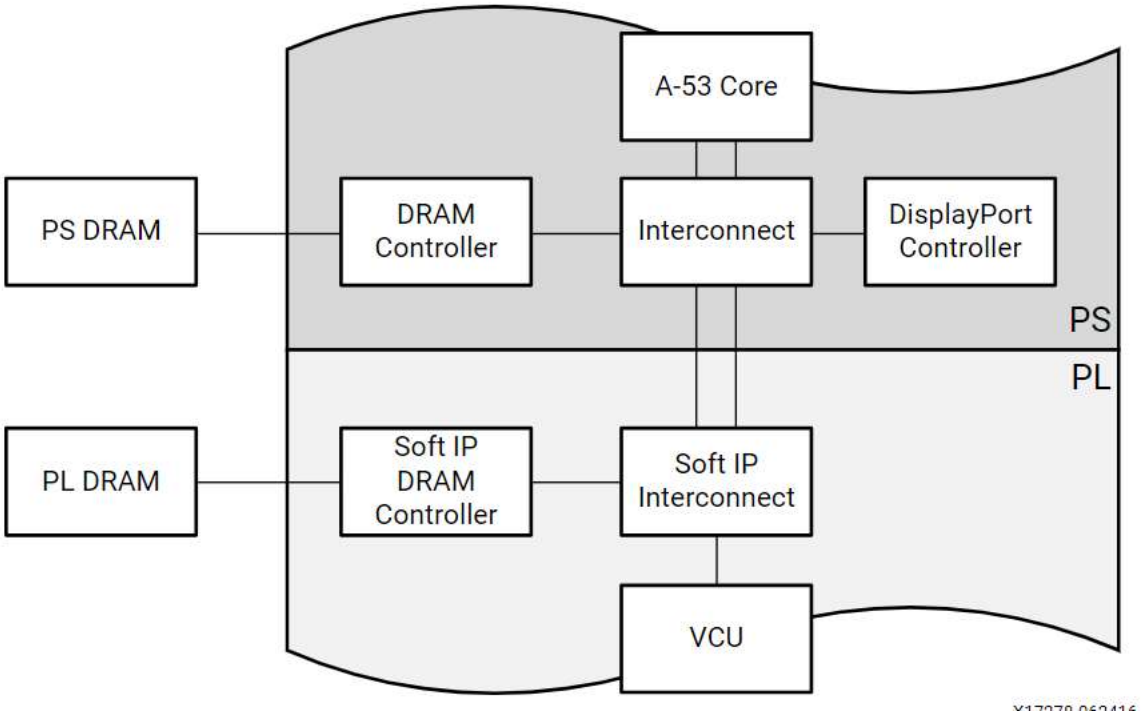


Figure 6-16 VCU Application

The typical clock frequencies for the target devices are described in the Zynq UltraScale+ MPSoCData Sheet: DC and AC Switching Characteristics (DS925). The maximum achievable clockfrequency of the system can vary. The maximum achievable clock frequency and all resourcecounts can be affected by other tool options, additional logic in the device, using a differentversion of Xilinx tools and other factors.

The VCU supports simultaneous encoding and decoding up to 4K UHD resolution at 60 Hz. Thisthroughput can be a single stream at 4K UHD or can be divided into up to 32 smaller streams ofup to 480p at 30 Hz. Several combinations

of one to 32 streams can be supported with differentresolutions provided the cumulative throughput does not exceed 4K UHD at 60 Hz.

Streams of 4K UHD at 60 Hz consume significant amounts of the bandwidth of the externalmemory interfaces and significant amounts of the Arm® AMBA® AXI4 bus bandwidth betweenthe Processing System and the Programmable Logic.For simultaneous encoder and decoder operation (including transcode use cases), consider usingboth a Xilinx PS Memory Controller and dedicated a Xilinx VCU Memory Controller.

The VCU core top-level signaling interface is shown in the following figure.

Figure 3: **VCU Core Top-Level Signaling Interface**

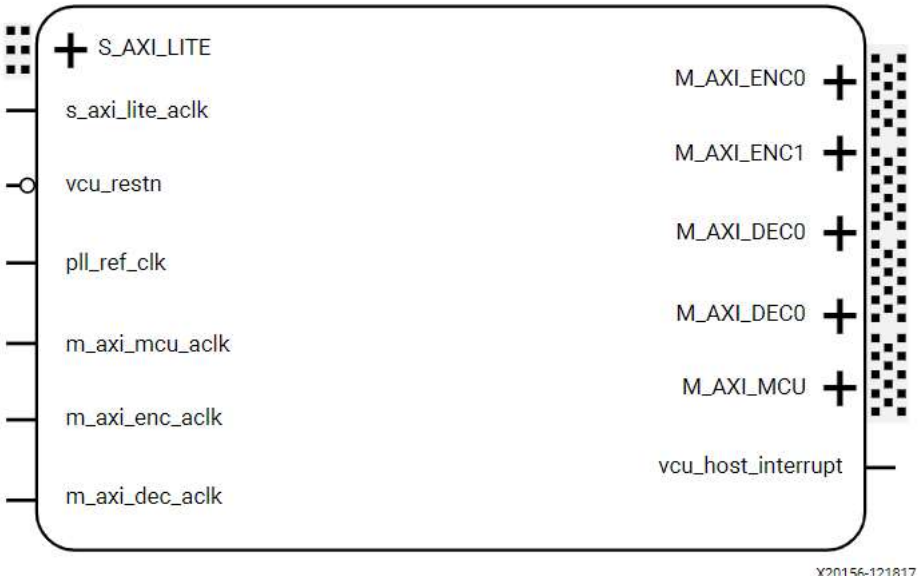


Figure 6-17 VCU Core Top-Level Signaling Interface

The following table summarizes the core interfaces：

Interface Name	Interface Type	Description
M_AXI_ENC0	Memory mapped AXI4 master interface	128-bit memory mapped interface for Encoder block.
M_AXI_ENC1	Memory mapped AXI4 master interface	128-bit memory mapped interface for Encoder block.
M_AXI_DEC0	Memory mapped AXI4 master interface	128-bit memory mapped interface for Decoder block.
M_AXI_DEC1	Memory mapped AXI4 master interface	128-bit memory mapped interface for Decoder block.
M_AXI_MCU	Memory mapped AXI4 master interface	32-bit memory mapped interface for MCU.

Interface Name	Interface Type	Description
S_AXI__LITE	Memory mapped AXI4-Lite slave interface	AXI4-Lite memory mapped interface for external master access.

Port Name	Direction	Description
m_axi_enc_aclk	Input	AXI clock input for M_AXI_VCU_ENCODER0 and M_AXI_VCU_ENCODER1
s_axi_lite_aclk	Input	AXI clock input for S_AXI_PL_VCU_LITE
pll_ref_clk	Input	PLL reference clock input
vcu_resetn	Input	Active-Low reset input from PL
vcu_host_interrupt	Output	Active-High interrupt output from VCU. Can be mapped to PL-PS interrupt pin.
m_axi_dec_aclk	Input	AXI input clock for M_AXI_VCU_DECODER0 and M_AXI_VCU_DECODER1
m_axi_mcu_aclk	Input	Input clock for M_AXI_MCU interface

Figure 6-18,19 VCU Interfaces

6.2.2 Experiment Logical

According to the basic knowledge of the VCU hard core, it is understood that to use the codec function of the VCU hard core, the first need to have video data input to the VCU, the video input path can be externally input through the HDMI interface on the PL end, and then stored to the DDR on the PL end , And then read the encoder interface input to the VCU from the DDR through the AXI4 bus, and then write it into the DDR after the encoding is completed. The PS terminal is only used as a simple control and communication with the MCU. The process of using decoding is similar. Another way is to use the DDR on the PS side as storage, because the AXI bus of the vcu can be in read and write mode. The PS side is the control center for video encoding and decoding, AXI bus interaction, control and data transmission, but it occupies the DDR bandwidth resources and AXI bus resources on the PS side. Either way, the data source of the VCU and the data completed by encoding and decoding are stored in the DDR. This experiment uses the PS terminal as a data access and control center for operation. The specific project implementation plan is as follows:

The data read by the first VCU is from the DDR on the PS side, and the data after the encoding is completed is stored in the DDR on the PS side.

The data decoded by the second VCU is the DDR from the PS side, and the data after the decoding is completed is stored in the DDR on the PS side.

The data completed from the VCU encoding and decoding in the middle will pass through an axi register slice module.

The function of this module is to connect an AXI memory-mapped master device to an AXI memory-mapped slave device through a set of pipeline registers,

usually used to interrupt critical timing paths. Similar to the role of AXI Interconnect.

As for the internal structure of VCU, I will not introduce it in detail. Customers can refer to the product documentation of VCU.

6.2.3 Experiment Steps

1，According to the "Hello world" basic configuration project in Chapter 3, save a file and rename it project_fz5_vcu_707, which corresponds to project_fz5_vcu_707.rar in the CD file.

2，Establish the data processing link at the PL end according to the above solution. The configuration of each module is not described in detail here, please configure it according to the actual situation in the project. As shown below.

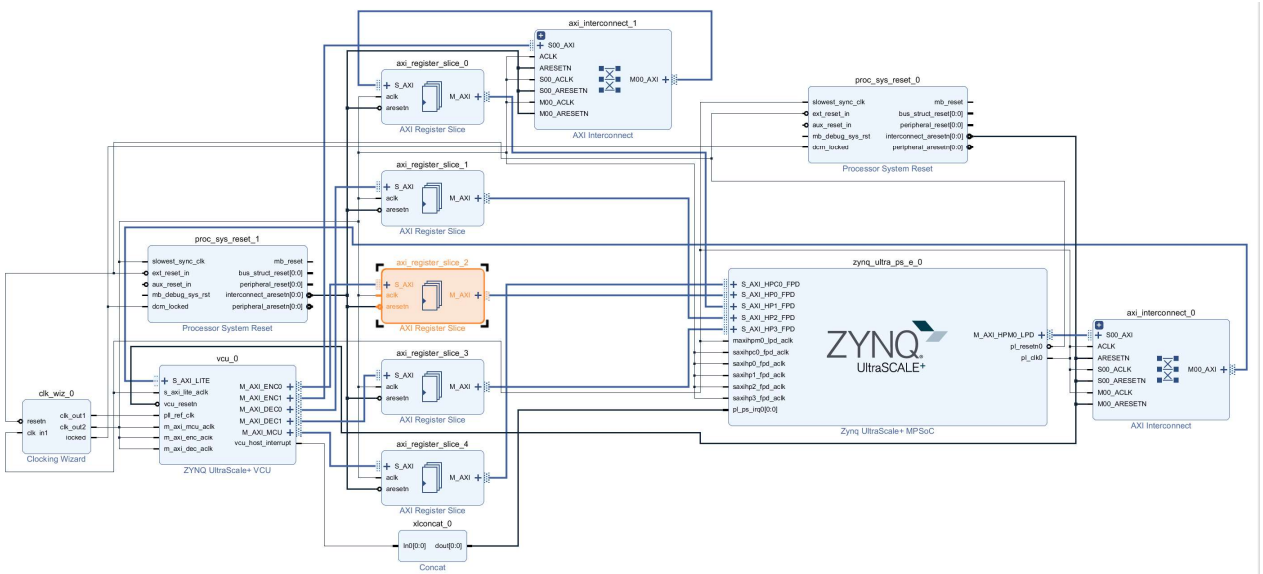


Figure 6-20 VCU BD design block diagram

1，Here is a detailed introduction to the clock system: the clock of the control route is the 100MHz clock output from the PS side to the PL side, and the clock required for VCU work varies according to the use. Here, the output clk_out2 from the PLL is the 332MHz clock used by the VCU , Which is the input of the three clocks of m_axi_mcu, m_axis_enc, and m_axi_dec. The reference clock of the PLL is 33MHz, which is used as the reference clock input of the entire VCU, that is, pll_ref_clk.

2，Regarding the data path: As can be seen from the block diagram of the project, the VCU does not have a separate data input interface. It reads data directly from the memory DDR4. After the encoding or decoding is completed,

the data is directly written to the DDR4, so that it can be It is confirmed that there is a DDR4 SDRAM controller inside the VCU.

According to the official PG252, this controller is customized and cannot be used for other purposes. After the project is completed, you can refer to the content of the VCU part in the software usage document. There is a detailed content introduction and operation details on the software PS side.

6.3 Sections of this Chapter

This chapter mainly introduces the use of MIPI and some introductory physical layer protocol knowledge. For further study, you need to view the specialized physical layer protocol documents. The newly introduced knowledge is to add a custom RTL module, or create a new project to define an IP. Then add the IP generation path to the MIPI project, so that we can embed our logic design into the entire solution design process。

Chapter 7 How to Use Xilinx Official Information

7.1 User Guide

7.1.1How to Use “User Guide”

In the design process, we usually encounter some problems, sometimes these problems are because we don’ t know the specific IP timing or some specific configurations, so we need to check the user manual, how to quickly find us in the user manual The required information is the key. For example, we need to find out the interface of PS-GTR and how to verify it.

Open a user manual as follows:

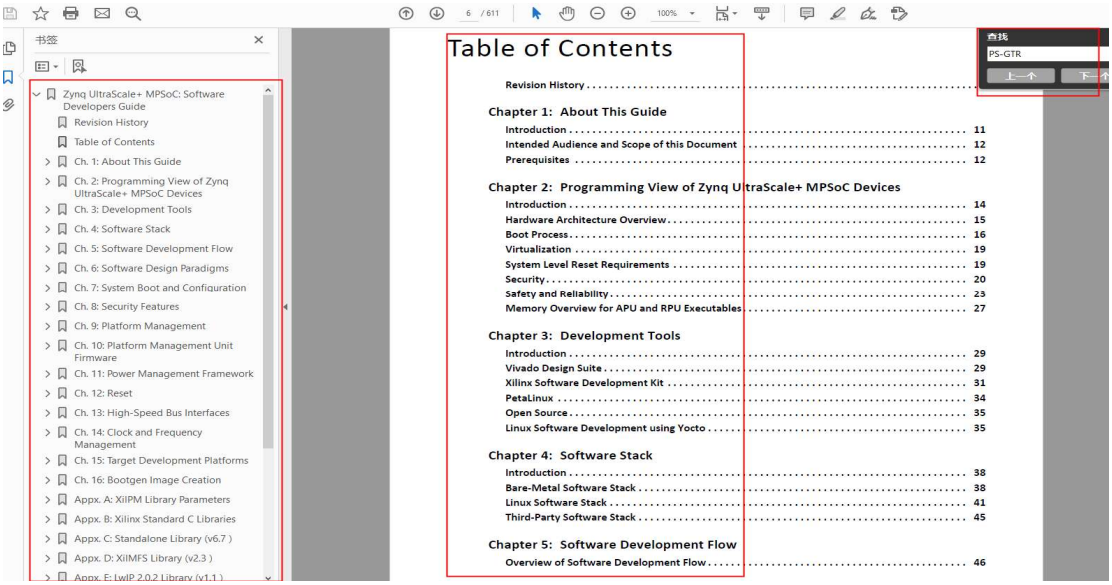


Figure 7-1 Directory structure of xilinx user manual

After opening the user manual, you can see the contents in the catalog. Some user manuals have less content, and some have more. If we need to find the corresponding content quickly, we need the customer to have a certain English reading ability, and then we will use the corresponding search tool. Click next:

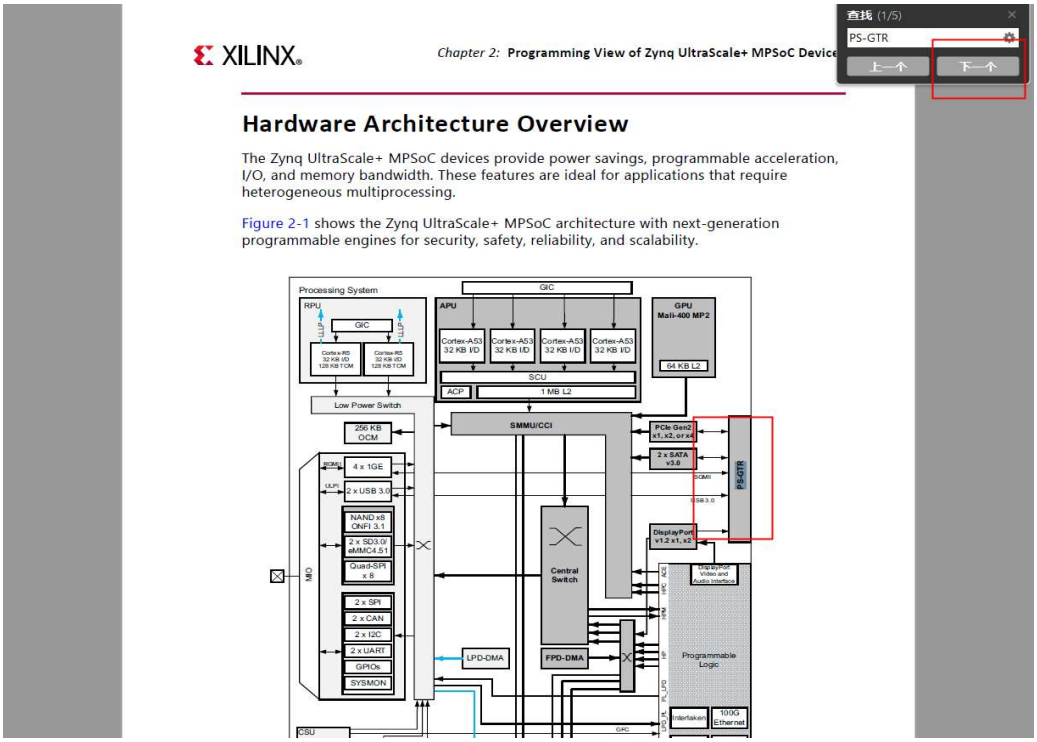


Figure 7-2 Jump to the specified chapter

You can find the location of the PS-GTR that appears in the block diagram, the interface of the link, and so on. Click Next again:

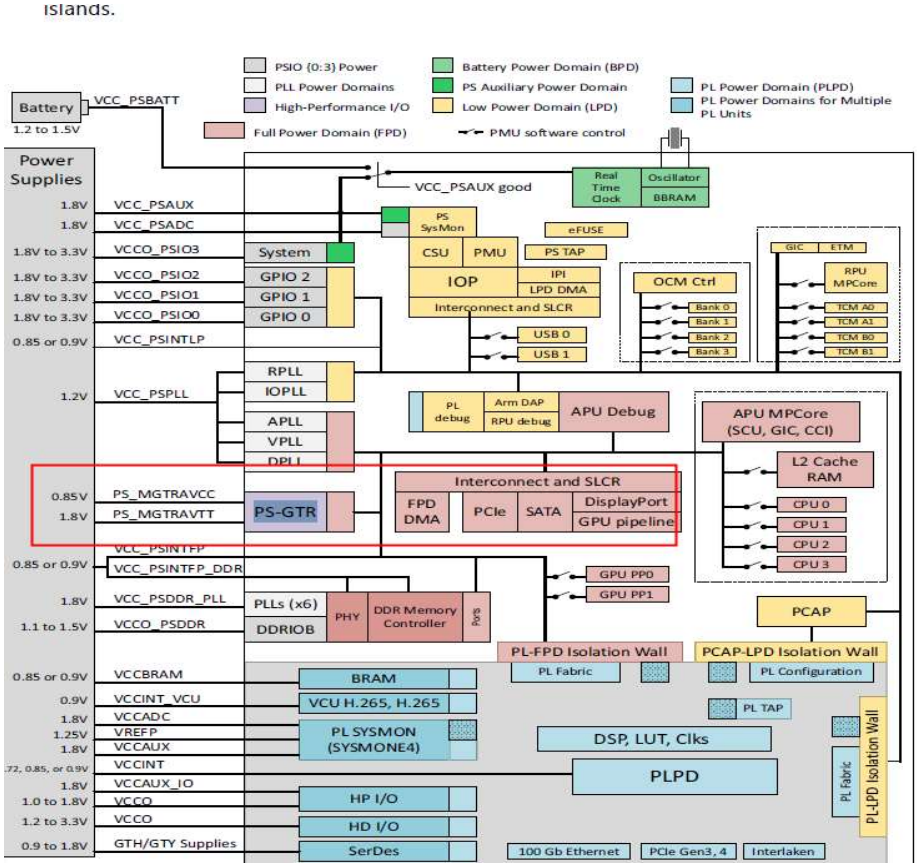


Figure 7-3 Find the required interface jump

See the corresponding level standard and the next place where PS-GTR appears:

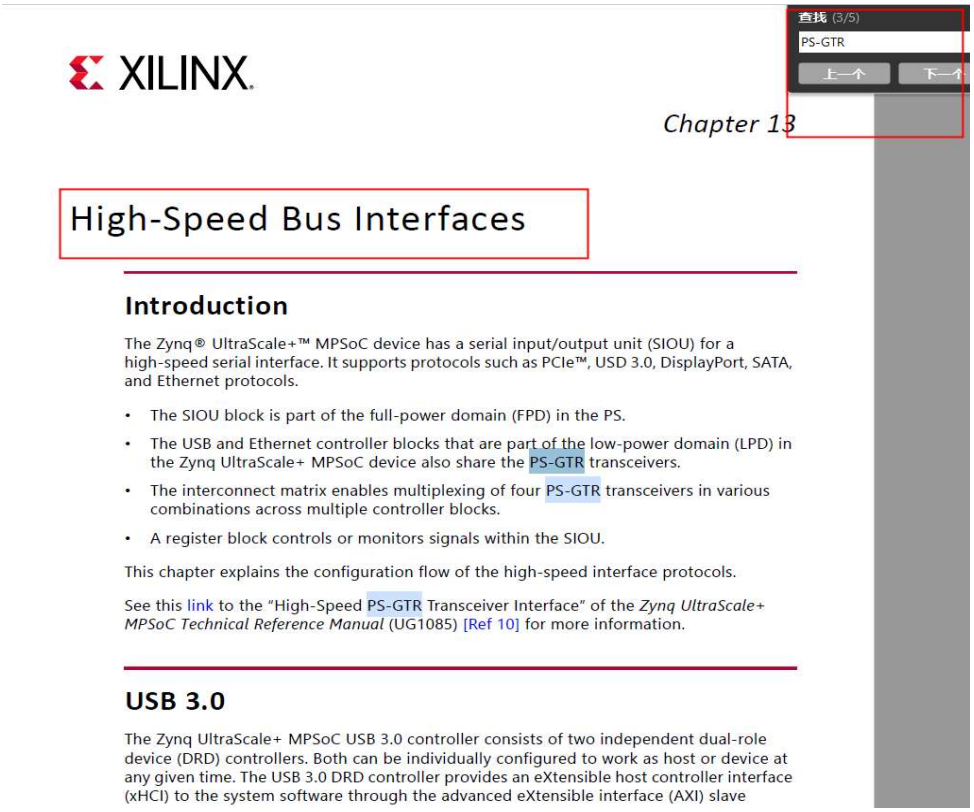


Figure 7-4 Find the chapter of high-speed bus interface

In Chapter 13 of the document, click on the table of contents in Chapter 13 to see what physical interfaces are using the PS-GTR high-speed serial interface, so that you can find the corresponding introduction, and then use the document finder to find Corresponding verification and debugging documents:

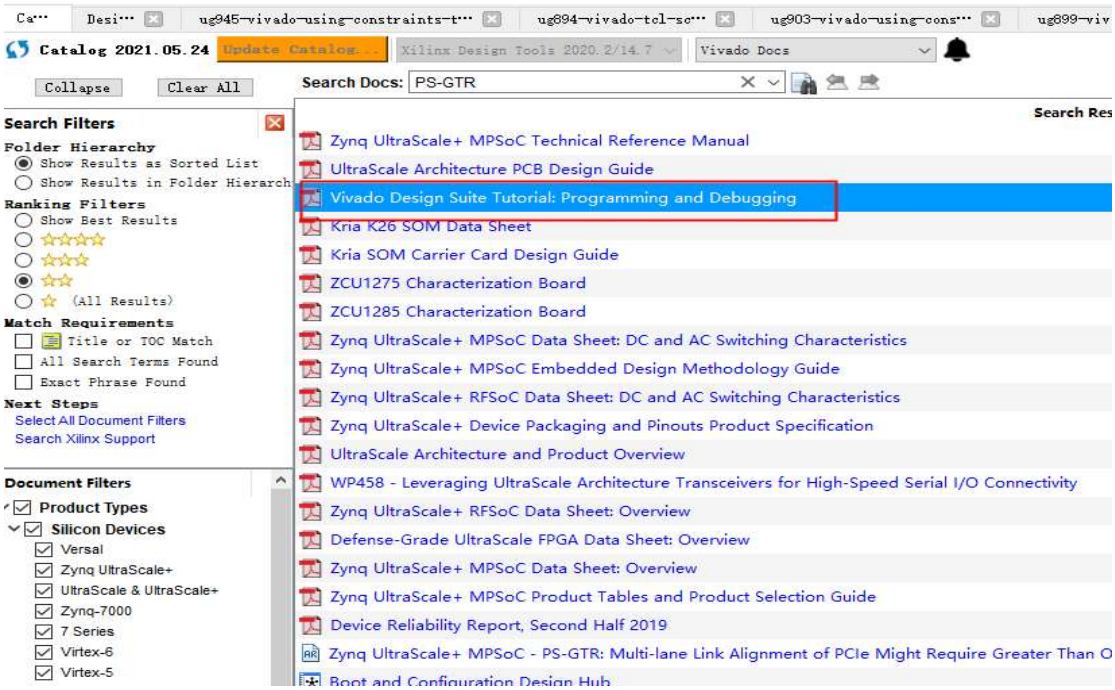


Figure 7-5 DocNav search for PS-GTR filtering results

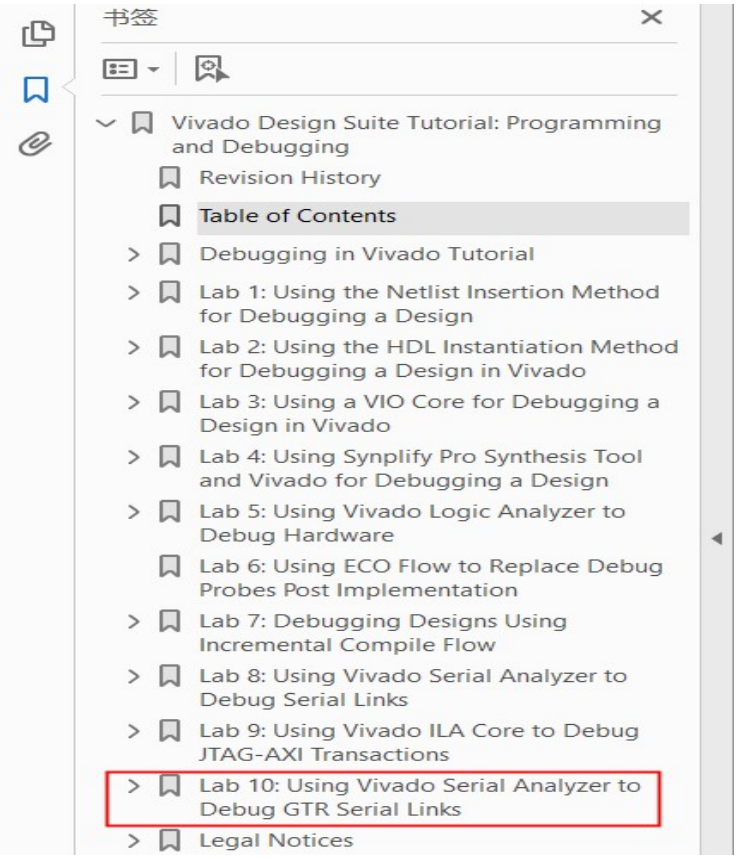


Figure 7-6 Find the corresponding document experiment-analyze the PS-GTR link experiment

7.2 Product Guide

7.2.1 How to Use the “Product Guide”

In the design process, we usually encounter some problems. Sometimes these problems are because we are not clear about the specific IP timing or some specific configurations. At this time, we need to read the corresponding IP product manual to find information, such as looking up MIPI -CSI2-Subsystem's product manual, look at how this IP is configured to 4lane, physical layer protocol configuration, and so on.

First find the product manual, as follows :

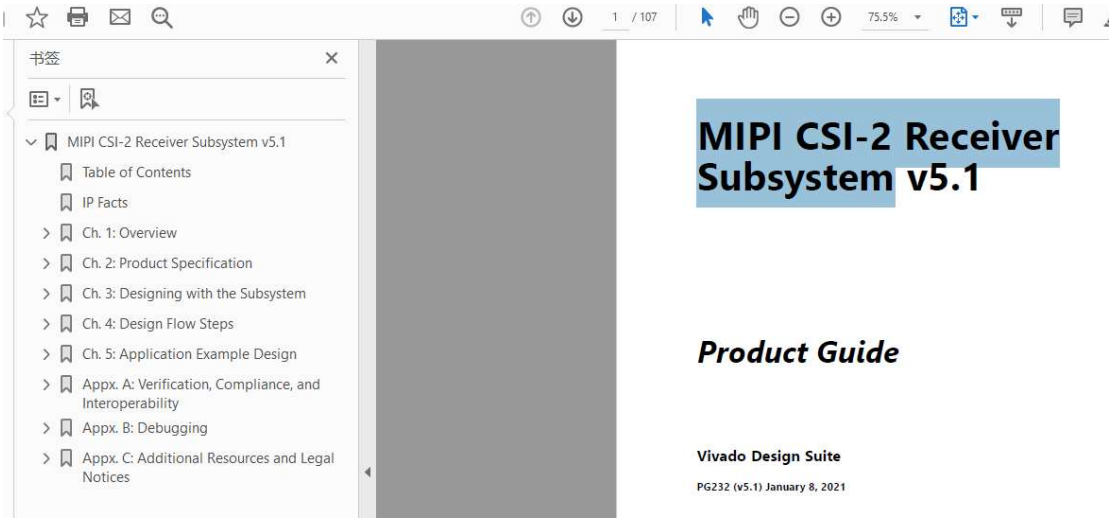


Figure 7-7 IP product manual

You can see the corresponding introduction to this IP from the catalog, including summary, physical layer structure block diagram, physical layer protocol used, interface and timing, and IP configuration process, etc. You can find the information we need.

7.3 Reference Design

7.3.1Reference Design for Vivado Project Usage

There are some development kits on the official website of xilinx, and the corresponding development kits have also made some reference designs. These reference designs are usually demos for a certain interface or protocol, and some hardware modules are used. If you need to reproduce this demo If you need the corresponding hardware environment, we generally just refer to it. For example, some reference designs of the UltraScale+ MPSOC series are placed on Wikipedia

at https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages /444006775/Zynq+UltraScale+MPSoC;

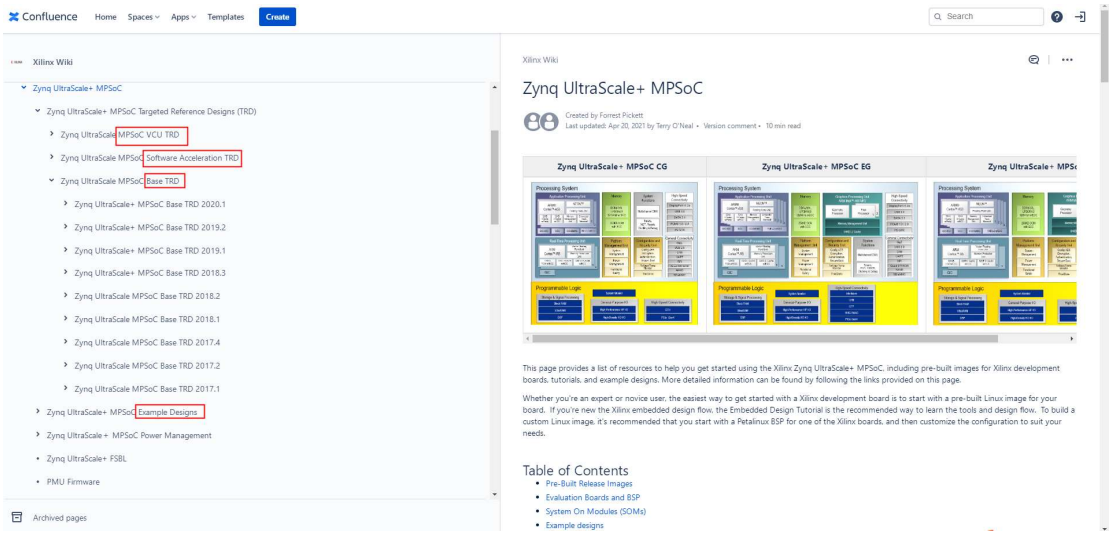


Figure 7-8 download reference design

Figure 7-9 Find the reference design

After finding the corresponding reference design, download the corresponding document, then open vivado, follow the introduction in the readme of the downloaded reference design material, find the corresponding engineering document Tcl document, and use the corresponding command to reconstruct the vivado project under vivado:

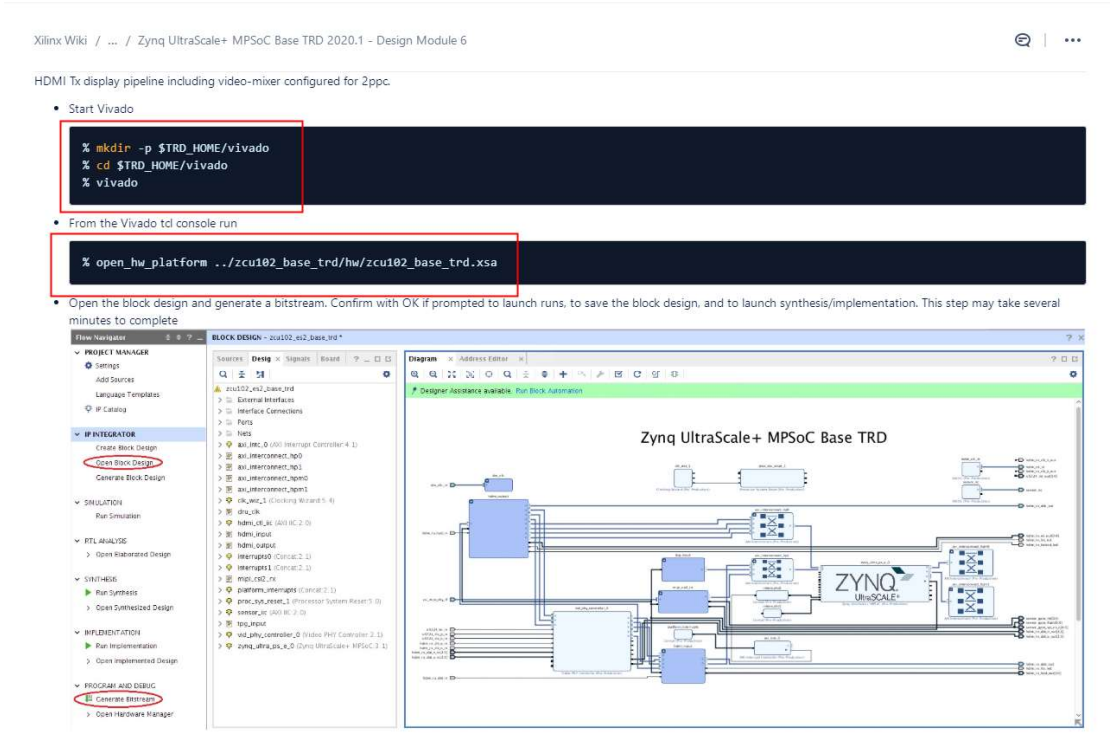


Figure 7-10 Find the project generated by the practical tcl document of the reference design

Then you can view some of the corresponding design ideas and actual configurations in the project. The reference design can help us solve some references to the IP configuration, IP module sequence, and clock in the design.

7.3Xilinx Community Forum

7.3.1 How to Ask Questions in the Community Forum

Regarding the use of the design forum, you can ask questions like normal forums. The online time of xilinx technical support engineers is working hours, and they usually don't work overtime to solve them. Of course, ask questions on it, and sometimes enthusiastic netizens will give good answers. Or if you search for the same question, you can see how the predecessors solved similar problems.

As follows

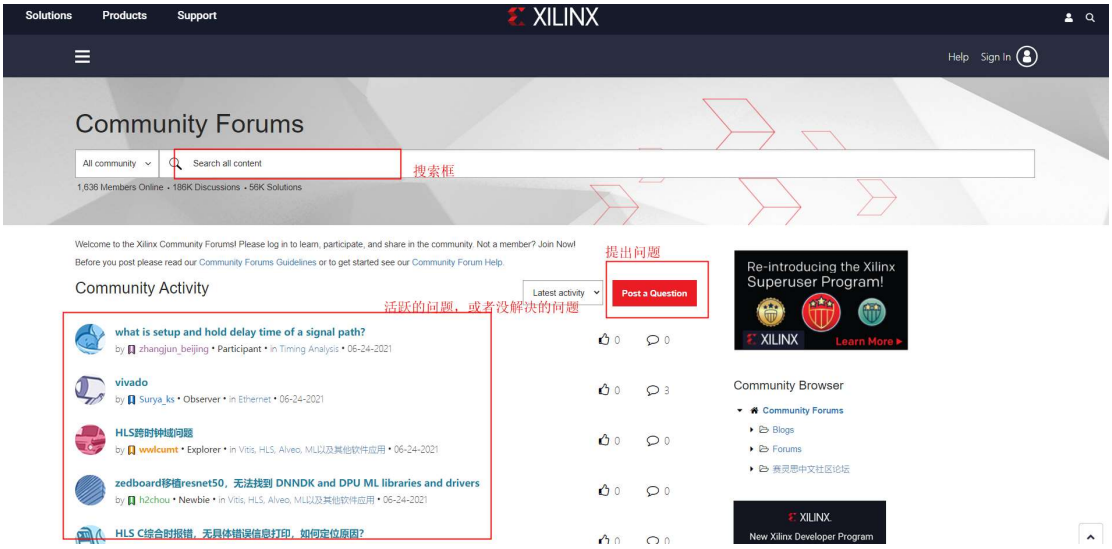


Figure 7-11 Community Forum Home Page

For example, we look for questions about 10GEthernet subsystem

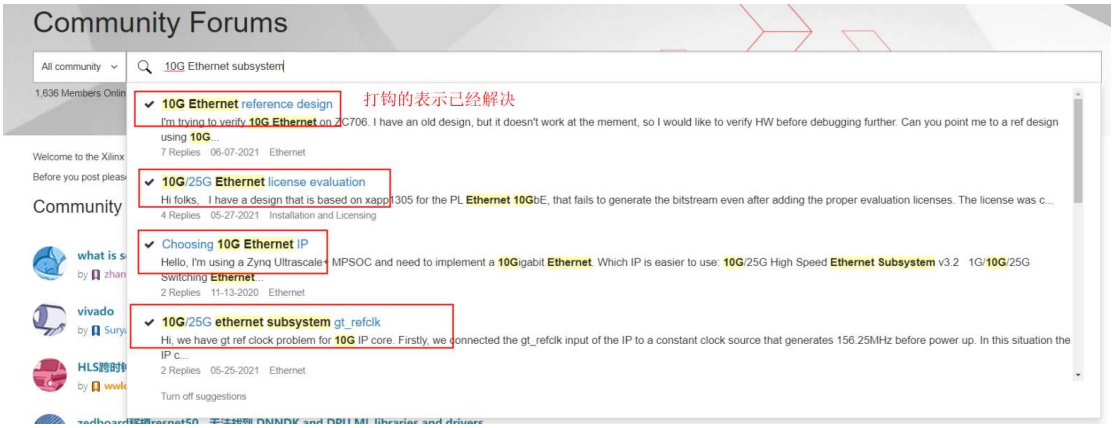


Figure 7-12 Search for similar questions

We can see many questions and resolved questions about 10GEthernet subsystem

chapter 8 Conclusion

8.1 Conclusion

This document gives some introductions to customers who use this hardware platform, and briefly talks about some knowledge and problems about learning FPGA. Of course, there is more to learn, mainly in the following aspects:

1. Regarding the syntax of verilog, further in-depth study is needed, and further, system verilog can be studied systematically.

2. Regarding Tcl, for those engaged in large and complex designs, mastering a script tool is very helpful to improve work efficiency.

3. Regarding the knowledge of constraints, no specific engineering examples are provided here to give an introduction. The space is limited. However, when you refer to xilinx official documents on constraints and documents and sample projects on timing closure, there should be a more in-depth To understanding.

4. For the knowledge of AXI bus, interface and protocol, it is recommended to refer to the official ARM protocol document.

5. Regarding the sample projects we have given, some are relatively simple and some are more complicated. It is recommended that customers study carefully. Finally, I hope that customers can gain something after studying this document.

Reference

- Official Xilinx WiKi: https://xilinx-wiki.atlassian.net/wiki/home
- DocNav Finder
- ds891-zynq-ultrascale-plus-overview.pdf
- ug1137-zynq-ultrascale-mpsoc-swdev.pdf
- ug1085-zynq-ultrascale-trm.pdf
- Verilog User Manual
- MYS-ZU5EV schematic diagram
- MYS-ZU5EV hardware manual
- Little Mei Ge FPGA Tutorial
- Wei San Academy FPGA Tutorial
- Wu Houhang. Learning FPGA in simple language[M]. Beijing University of Aeronautics and Astronautics Press, 2013.
- Xia Yuwen. Verilog Digital System Design Tutorial. 3rd Edition [M]. Beijing University of Aeronautics and Astronautics Press, 2013.
- Han Bin, Yu Xiaoyu, Zhang Leiming. FPGA design skills and case development detailed explanation[M]. Publishing House of Electronics Industry, 2014.
- TclTk Introduction Classic_Second Edition_Chinese.pdf
- Vivado Design Suite User GuideDesign Analysis and Closure Techniques.pdf
- Vivado Design Suite UserGuideUsing Tcl Scripting.pdf
- Vivado Design Suite User GuideUsing Constraints
- Vivado Design Suite UserGuideRelease Notes, Installation, andLicensing

- Vivado Design Suite TutorialUsing Constraints
- Timing Closure User Guide
- AXI4-Stream Video IP and System Design Guide

Appendix A

Warranty & Technical Support Services

MYIR Electronics Limited is a global provider of ARM hardware and software tools, design solutions for embedded applications. We support our customers in a wide range of services to accelerate your time to market.

MYIR is an ARM Connected Community Member and work closely with ARM and many semiconductor vendors. We sell products ranging from board level products such as development boards, single board computers and CPU modules to help with your evaluation, prototype, and system integration or creating your own applications. Our products are used widely in industrial control, medical devices, consumer electronic, telecommunication systems, Human Machine Interface (HMI) and more other embedded applications. MYIR has an experienced team and provides custom design services based on ARM processors to help customers make your idea a reality.

The contents below introduce to customers the warranty and technical support services provided by MYIR as well as the matters needing attention in using MYIR’ s products.

Service Guarantee

MYIR regards the product quality as the life of an enterprise. We strictly check and control the core board design, the procurement of components, production control, product testing, packaging, shipping and other aspects and strive to provide products with best quality to customers. We believe that only quality products and excellent services can ensure the long-term cooperation and mutual benefit.

Price

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish

long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

Delivery Time

MYIR will always keep a certain stock for its regular products. If your order quantity is less than the amount of inventory, the delivery time would be within three days; if your order quantity is greater than the number of inventory, the delivery time would be always four to six weeks. If for any urgent delivery, we can negotiate with customer and try to supply the goods in advance.

Technical Support

MYIR has a professional technical support team. Customer can contact us by email (support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

After-sale Service

MYIR offers one year free technical support and after-sales maintenance service from the purchase date. The service covers:

Technical support service

MYIR offers technical support for the hardware and software materials which have provided to customers:

- To help customers compile and run the source code we offer;
- To help customers solve problems occurred during operations if users follow the user manual documents;
- To judge whether the failure exists;
- To provide free software upgrading service.

However, the following situations are not included in the scope of our free technical support service:

- Hardware or software problems occurred during customers' own development;
- Problems occurred when customers compile or run the OS which is tailored by themselves;
- Problems occurred during customers' own applications development;
- Problems occurred during the modification of MYIR's software source code.

After-sales maintenance service

The products except LCD, which are not used properly, will take the twelve months free maintenance service since the purchase date. But following situations are not included in the scope of our free maintenance service:

- The warranty period is expired;
- The customer cannot provide proof-of-purchase or the product has no serial number;
- The customer has not followed the instruction of the manual which has caused the damage the product;
- Due to the natural disasters (unexpected matters), or natural attrition of the components, or unexpected matters leads the defects of appearance/function;
- Due to the power supply, bump, leaking of the roof, pets, moist, impurities into the boards, all those reasons which have caused the damage of the products or defects of appearance;
- Due to unauthorized weld or dismantle parts or repair the products which has caused the damage of the products or defects of appearance;
- Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

Warm tips

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.

2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.

3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.

4. Do not clean the surface of the screen with chemicals.

5. Please read through the product user manual before you using MYIR’ s products.

6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR’ s support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

Maintenance period and charges

- MYIR will test the products within three days after receipt of the returned goods and inform customer the testing result. Then we will arrange shipment within one week for the repaired goods to the customer. For any special failure, we will negotiate with customers to confirm the maintenance period.
- For products within warranty period and caused by quality problem, MYIR offers free maintenance service; for products within warranty period but out of free maintenance service scope, MYIR provides maintenance service but shall charge some basic material cost; for products out of warranty period, MYIR provides maintenance service but shall charge some basic material cost and handling fee.

Shipping cost

During the warranty period, the shipping cost which delivered to MYIR should be responsible by user; MYIR will pay for the return shipping cost to users when the product is repaired. If the warranty period is expired, all the shipping cost will be responsible by users.

Products Life Cycle

MYIR will always select mainstream chips for our design, thus to ensure at least ten years continuous supply; if meeting some main chip stopping production, we will inform customers in time and assist customers with products updating and upgrading.

Value-added Services

1. MYIR provides services of driver development base on MYIR’ s products, like serial port, USB, Ethernet, LCD, etc.

2. MYIR provides the services of OS porting, BSP drivers’ development, API software development, etc.

3. MYIR provides other products supporting services like power adapter, LCD panel, etc.

4. ODM/OEM services.

MYIR Electronics Limited

Room 04, 6th Floor, Building No.2, Fada Road,

Yunli Inteiligent Park, Bantian, Longgang District.

Support Email: support@myirtech.com

Sales Email: sales@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com