MYS-ZU5EV FPGA 指导手册

文件状态： [] 草稿 [√] 正式发布	**文件标识：**	MYS-ZU5EV FPGA 指导手册
	当前版本：	V2.0.3
	作　　者：	Rill.Yang
	创建日期：	2021-05-31
	最近更新：	2021-07-09

版本历史：

时间	版本	修改历史
2021.05.31	v1.0.0	v1.0 创建
2021.06.02	V1.0.1	修正目录结构，修正章节内容安排。
2021.06.04	V1.0.2	修正目录结构，修正章节内容安排
2021.06.07	V1.0.3	修正目录结构，修正章节内容安排
2021.06.08	V1.0.4	修正目录结构，修正章节内容安排
2021.06.25	V2.0.0	正式发布版本
2021.07.03	V2.0.1	1，修改文本整体格式，字体格式,。2，修改前版本中 6.1.1 节的有争议的图片,以及修正了 2.3.4 节中关于输入约束以及输出约束的图片引用和约束语句。3，修正了 1.4 节的章节表格。4，修正了 6.1.3 节中的软件部分内容。5，修正了 7.3.2 节中引用的内容，与 7.3.1 节进行了方合并处理。6，文档结尾增加了参考资料和联系方式。
2021.07.07	V2.0.2	1，修改了图片的名称。2，说明了每一个示例工程在光盘文档中对应的压缩文档。3，对 2.3.4 节中关于使用约束的讲解进行了简化修改。4，对 5.2.1 节 DMA 基础知识进行了修改，引用了官方的 PG021 文档内容。
2021.07.09	V2.0.3	1，在第 6.2 节增加了 VCU 的部分内容。

目录

第一章 概要

1.1 文档介绍

本文档主要是介绍怎么在 MYS-ZU5EV 硬件平台上实现自己的工程开发。通过学习本文档，希望客户可以达到几个目标：

1、了解 MYS-ZU5EV 这个硬件平台。

2、能够在这个硬件平台上实现自己的目标设计。

3、了解有关 MYS-ZU5EV 这个硬件平台的参考资料，客户可以快速的找到，特别是对于 xilinx 官方的一些参考文档和参考设计。

介绍客户在使用 MYS-ZU5EV 硬件平台的时候，可能用到的资料信息搜集，遇到关键点可以寻求参考资料方向，以及可以向米尔寻求合作开发。

这里说一下 ZYNQ/MPSOC 这个异构平台的起源。

这是在 SOC 的基础上发展起来的硬件平台，当所需要的各种外设接口以及需要处理的数据经过 ARM 处理器整合到一起之后就是 SOC 芯片，但是 SOC 有一些专用的工作完成不了或者达不到需要的标准，所以就需要添加额外的资源来完成专用的功能，这个时候就有两个方案，一是在 PCB 外挂专用器件，二是在 SOC 芯片里面添加专用的区域去完成专用的功能，显然现在 ZYNQ 平台就是第二种方案。第二种方案是发展的趋势，现有流行趋势 AI 功能以及新添加的各种功能模块首要的方案是将对应的功能模块嵌入进 SOC 芯片，形成一个更复杂的 SOC 芯片。

关于 ZYNQZYNQ/MPSOC 这个异构平台详细信息介绍可以参考 xilinx 官方的文档：

ds891-zynq-ultrascale-plus-overview.pdf
ug1137-zynq-ultrascale-mpsoc-swdev.pdf
ug1085-zynq-ultrascale-trm.pdf

1.2 关于本文档后面章节安排

第一章概论，简要介绍了本文档撰写目的以及文档的章节结构。

第二章介绍了使用本硬件平台或者相似的硬件平台的一些准备工作，包括大概的硬件平台使用的一些准备工作，对于使用 xilinx 公司的 EDA 工具 vivado 的介绍，从 xilinx 官网注册账号到下载需要版本的 vivado 软件，安装截图介绍。其次是介绍了 FPGA 仿真工具 modelsim 的下载安装和使用，相对应的 vivado 版本可以调用具体的 modelsim 版本，这里参考 vivado 安装文档，可以找到对应的 modelsim 版本。对于第一次使用这些 EDA 工具的时候，都需要 license 注册，注册方式方法都有介绍。除此之外，使用 Xilinx 的一些付费的

IP，试用期有 3 个月，需要在官方网站上下载临时的 license 进行注册才可以使用，否则就需要自己设计，写代码。最后介绍了一些 verilog 的语法，相应的参考资料，文档查找器 DocNav 的使用，以及 XDC 约束文件的语法介绍，TCL 语法的简要介绍等等。

第三章，3.1 节讲解详细的硬件平台的详细配置。大概的内容有根据硬件平台使用的资源对 MPSOC 进行详细的配置介绍，比如硬件平台在 PS 端连接了 DP 接口，在建立 vivado 工程的时候就需要将 MPSOC 这个 IP 核的外设接口 DP 接口选项打钩，并且根据原理图连接方式选择对应的链接引脚之类的。总之，就是硬件平台上有的接口和外设都会在第三章的 PS 配置中详细介绍。最后就是具体的生成硬件平台文件 xsa 文件步骤，最后就是使用 vitis 建立 app 工程输出 hello world。

3.2 小节在 3.1 小节的基础上，假设添加了 PL 端逻辑的情况下，介绍一下 PL 与 PS 端的数据交互方式。着重介绍了 AXI4 总线。这个内容跟后面的章节内容是息息相关的。

第四章，分几个小节内容，分别介绍一个使用本硬件平台来达到一个目标的使用案例，这些都是基于 AXI4-Lite 总线 PL 端外扩外设接口模块。介绍的两个通用的接口 AXI UART 接口以及 AXI-IIC 接口。

第五章，高性能 AXI4 总线接口使用的案例介绍。5.1 介绍使用 PL 端 BRAM 来交互小模块的 PS 端数据。5.2 介绍的是使用 DMA 的 demo。

第六章，介绍了在视频流的传输中 AXI-Stream 的使用，着重介绍了使用 IMX334 作为 sensor 的 MIPI 接口，配置，以及使用 AXI-stream 传输视频流的过程，添加了的东西是自定义的 RTL 模块怎么添加到 AXI-Stream 数据流中。自定义 IP 的设计，白平衡模块的添加。

第七章，介绍如何使用 xilinx 官方的一些参考资料。7.1 介绍怎么迅速的查看官方的用户手册文档，达到迅速找到我们需要的信息。7.2 介绍怎么迅速查找 xilinx 的产品手册，迅速找到相对应 IP 的功能、特性、接口、时序等方面的信息。7.3 介绍的是怎么使用 xilinx 的官方参考设计。7.4 对于客户遇到的问题怎么迅速的寻求帮助，在 xilinx 的官方网站的中英文社区可以发起提问，这里有专门的 xilinx 的技术支持进行维护，工作时间在线，在这里可以得到迅速的反馈和解决方案。

第八章，总结。

1.3MPSOC 系列芯片介绍

1，低端的 CG 系列，如图所示，

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							

图 1-1CG 系列器件特性

低端的系列在处理器是双核 A53，主频率表示可以达到 1.3GHz，实际的频率更低，只有 1GHz 左右，ps 端的高速接口有 USB3.1，STAT3.0，DP，GEthernet，PCIe。方面能力偏低，逻辑单元随着需求的不通可以选择不同的器件。

2，中端的 EG 系列，如图所示：

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											

图 1-2 EG 系列器件特性

中端的系列在处理器是四核 A53，主频率可以达到 1.5GHz，实际稳定的频率更低，只有 1.2GHz 左右，ps 端的高速接口有 USB3.1，STAT3.0，DP，GEthernet，PCIe。逻辑单元随着需求的不通可以选择不同的器件，高速需求比较高的话可以选择 4EG，5EG 往上的，如果需要 100Gbps 的通信数据的话那就需要选择 11EG 以及 17EG，19EG 这三个型号。跟 CG 系列最大的区别就是在 PS 端增加了 GPU 处理器。

3，高端的 EV 系列，如图所示：

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		

图 1-3 EV 系列器件

高端的 EV 系列在处理器是四核 A53，主频率可以达到 1.5GHz，实际稳定的频率更低，只有 1.2GHz 左右，ps 端的高速接口有 USB3.1，STAT3.0，DP，GEthernet，PCIe。逻辑单元随着需求的不通可以选择不同的器件，高速需求比较高的话可以选择 4EV，5EV，7EV 都是可以的，如果需要 100Gbps 的通信数据的话那就还是需要选择中端的 11EG 以及 17EG，19EG 这三个型号。跟 EG 系列最大的区别就是在 PL 端增加了 VCU 硬核模块，可以处理 H.264/H.265 协议的视频编码译码。

1.4Vivado 工程列表

本文档介绍的工程中在如下列表中可以看到，对应的工程文档可以在公司公布的资料渠道下载使用。具体的开发流程在第三章有详细介绍。

表 1 工程列表

序号	文档名称	说明
1	hello_world.rar	在第三章中作为基础配置工程介绍
2	gpio_mio.rar	在第三章中介绍 ps 端 LED 灯的使用介绍
3	Iic_test.rar	在第四章 PS 端的 IIC 外设总线的使用介绍
4	uart_cycle.rar	在第四章中介绍基于 AXI-Lite 总线外设实验
5	dma_loop.rar	在第五章中以 AXI4 总线使用的介绍实验
6	bram_test.rar	在第五章中以 AXI4-Lite 总线使用的介

		绍实验
7	MIPI_DP_4G	在第六章中以 AXI-Stream 总线使用的介绍实验

1.5 本章小节

本章节主要介绍了整个文档的概要，后学的章节基本都在本章提到，详细的内容需要到具体的章节去查看，或者找到对应的参考资料或者工程。对于 MYIR MYS-ZU5EV 硬件平台，主要的应用方向在于边缘计算，图像处理以及人工智能。

第二章 MYS-ZU5EV 使用准备

2.1 硬件准备

稍微介绍一下 MYS-ZU5EV 硬件平台的基本特性，产品预览图 2-1

图 2-1 硬件平台产品图示

MYS-ZU5EV-32E4D-EDGE 采用 Xilinx XCZU5EV-SFVC784 器件，速度等级为-2。32E 表示 32GEMMC，4D 表示 4G DDR 存储。XCZU5EV-2SFVC784I 支持 1.5GHz（最大-2）的 APU 速度，600MHz（最大-2）的 RPU 速度，667MHz（最大-2）的 GPU 速度，以及高达 2400Mbps 的 DDR4 速度。

XCZU5EV-2SFVC784I 器件具有以下资源：

处理器系统单元（PS）：

处理核心：四核 ARM Cortex-A53 多核处理器 高达 1.5GHz

最高时钟频率：1.5Ghz

APU：L1 Cache 32KB I / D 每个核心, L2 Cache 1MB.

RPU：L1 Cache 32KB I / D 每个核心

片内缓存：256KB

片外接口：支持 LPDDR4，DDR4，DDR3, DDR3L LPDDR3 with ECC

外部静态存储：2x Quad-SPI，NAND

DMA 通道：8（其中 4 个 PL 专用）

外设：

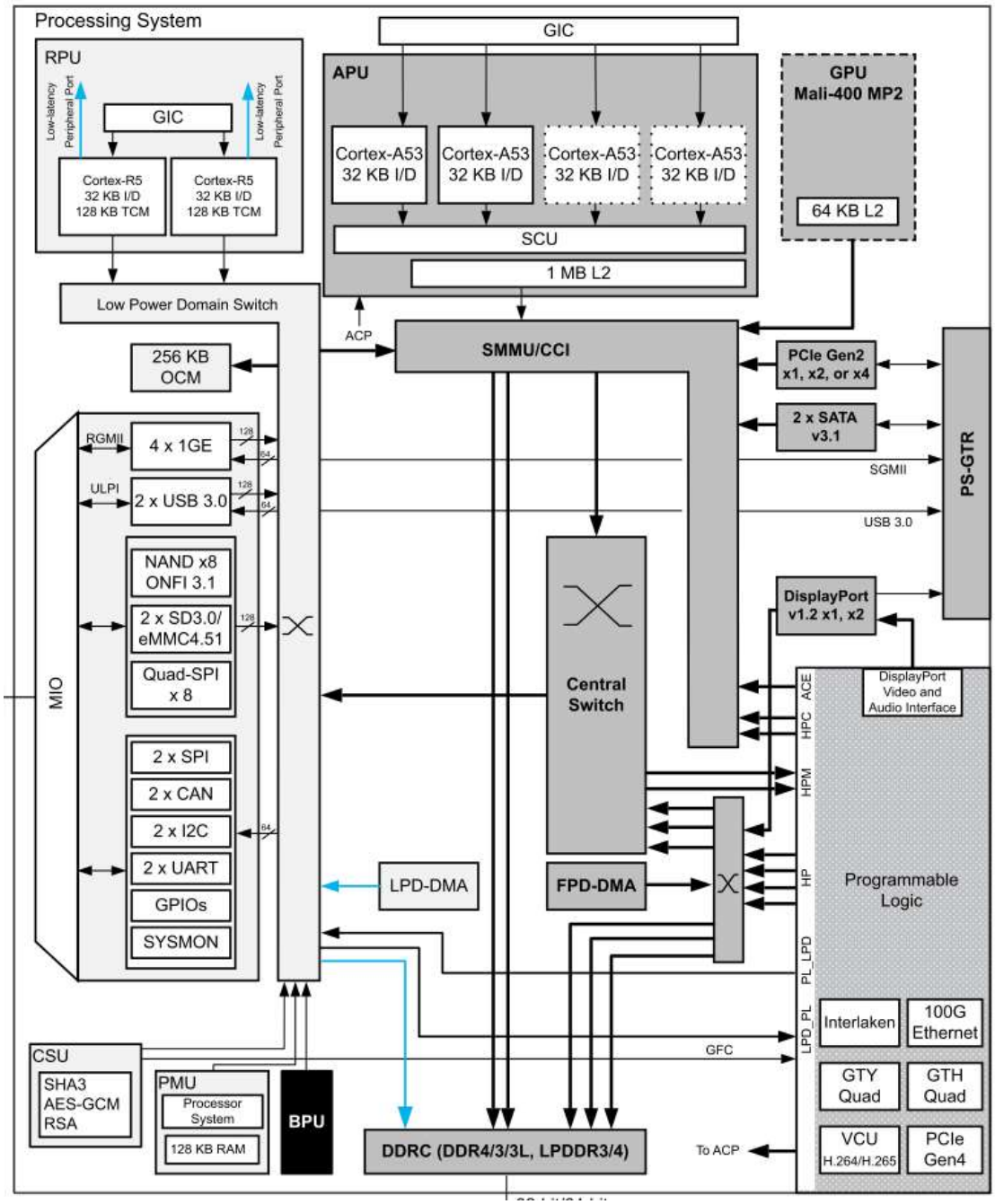


图 2-2 MPSOC 互联框图

高速接口：PCIe® Gen2 x4, 2x USB3.0, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet。

通用接口：2xUSB 2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32bGPIO。

可编程逻辑单元（PL）：

表 2 PL 端逻辑资源

资源	ZU5EV

逻辑核心	Xilinx Kintex Ultrascale+®FPGA
可编程逻辑单元	256K
LUTs	117K
触发器	234K
Block RAM	Distributed RAM 5.1Mb / Block RAM 18.0Mb
DSP Slice	1248
AMS System Monitor	1

2.2 软件准备

本节主要介绍 Vivado2020.1 的安装，vivado2020.1 的安装适用于所有的 vivado 版本，vivado 各个版本的区别不是很大，我们推荐使用我们工程文档同样版本的 vivado 软件版本，以方便验证我们提供的工程文档例程。

为什么是安装 vivado 而不是 ISE，这里说一下演进过程：

随着 FPGA 芯片进入 28nm 的工艺，ISE 软件的性能已经不能够跟进工艺的进步对硬件性能产生的巨大提升，所以就有了 vivado 的计划，虽然 2008 年后续的 ISE 持续更新，也添加进入了 28nm 系列的器件以及后续的器件库，但是加速设计开发，提升软件效率是必要的。

Vivado®DesignSuite 旨在提高使用 Xilinx®UltraScale™和 7 系列器件，Zynq®UltraScale+™MPSoC 器件和 Zynq®-7000 SoC 设计，集成和实施系统的整体工作效率。

再有就是 vivado 版本不同会出现各种问题，首先就是 IP 的更新，其次是 vivado 在综合实现生成 bitstream 过程中会出现一下意想不到的问题，可能是因为路径有空格汉字之类的问题导致，也可能是其他的不知名原因导致。为了顺利的进行工程开发，推荐同样的版本。

其次就是使用 vivado 时可能出现的莫名其妙的原因，这个原因有时候是因为 PC 处理器进程出错或者死亡导致的，也可能是不符合 vivado 的使用习惯导致的，各种复杂的原因，如果出现，我们推荐的一种解决办法就是卸载 vivado，重新安装。

2.2.1 Vivado 的下载

下载 Vivado 工具需要在 xilinx 官方网站上注册账号，完成注册之后，可以找到“产品——》硬件开发——》Vivado Design Suit——》找到对应的版本”进行下载。如图所示

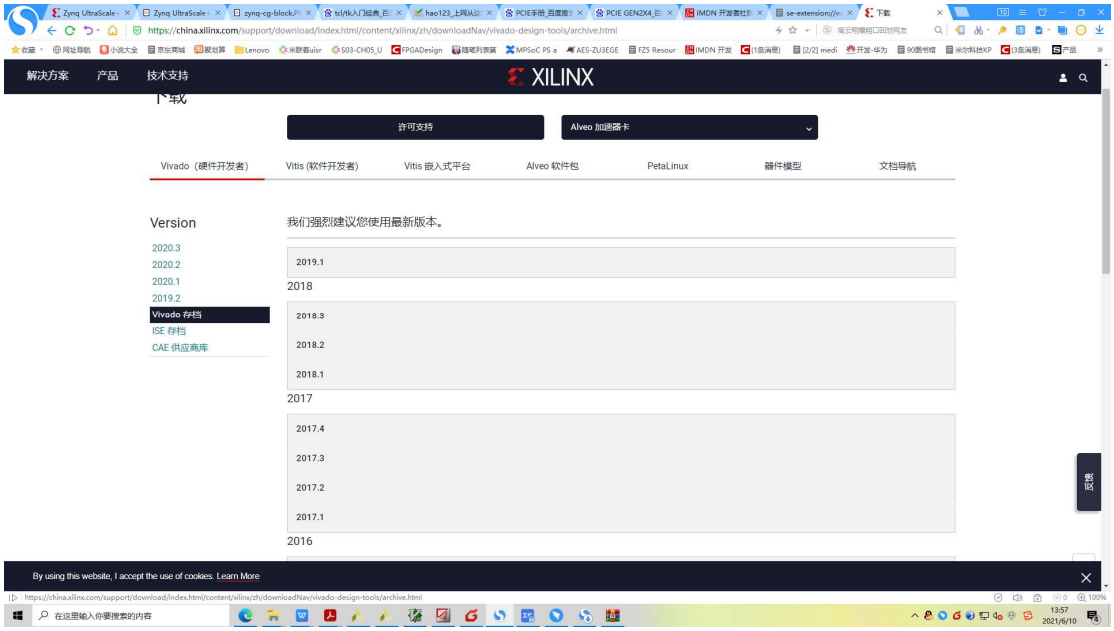


图 2-3vivado 下载导航网页

可以看到各种版本的序号都在，下载选择”ALL OS installer Signaler-file download”，虽然安装包比较大，但是下载安装之后，一般没有多余的问题。而且下载的速度比较快。

2.2.2 vivado 的安装

安装之前，请关闭杀毒软件，特别是 360 防护杀毒软件。

第一步：解压下载的安装文档，双击安装程序，如图，

图 2-4 vivado 安装文件解压后的安装程序图

第二步：在欢迎界面点击 NEXT：

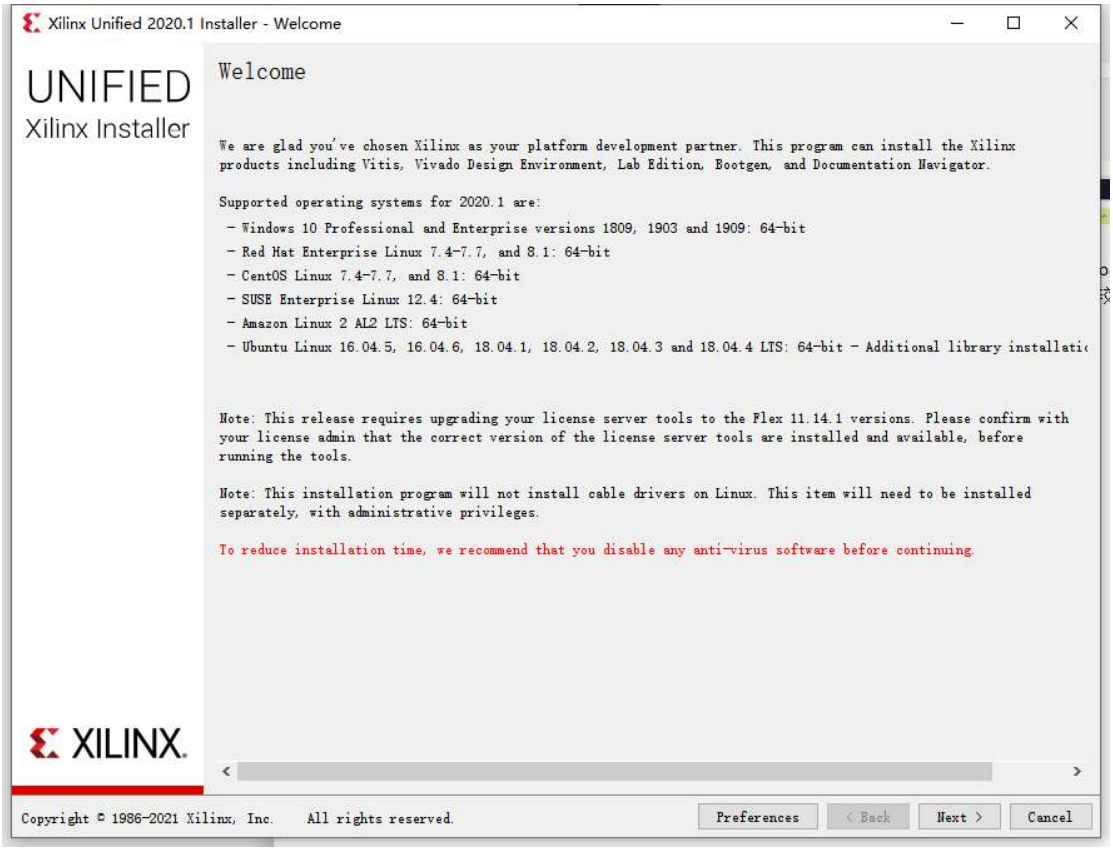


图 2-5 安装导航图

第三步：全部勾选上，都同意，NEXT。

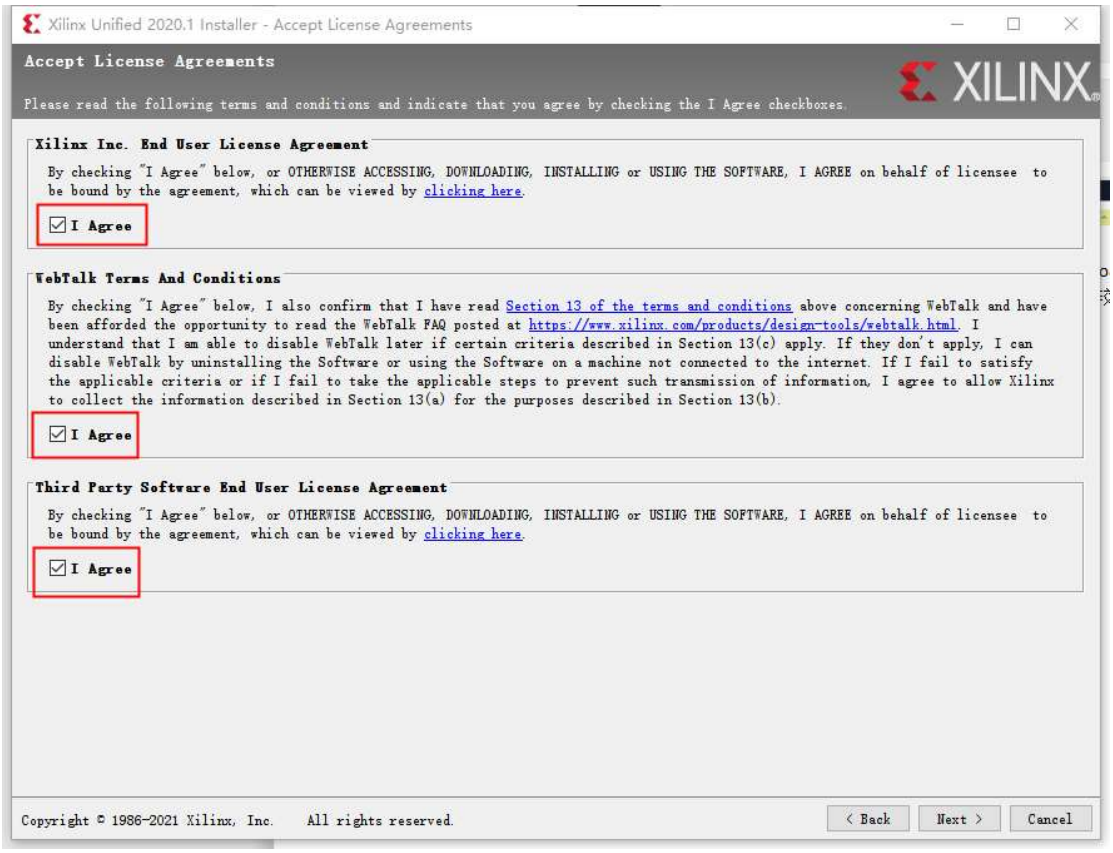


图 2-6 安装导航图

第四步：选择要安装的产品，因为 2020.1 版本是从 SDK 更新到 Vitis 软件工具，所以选择 Vitis

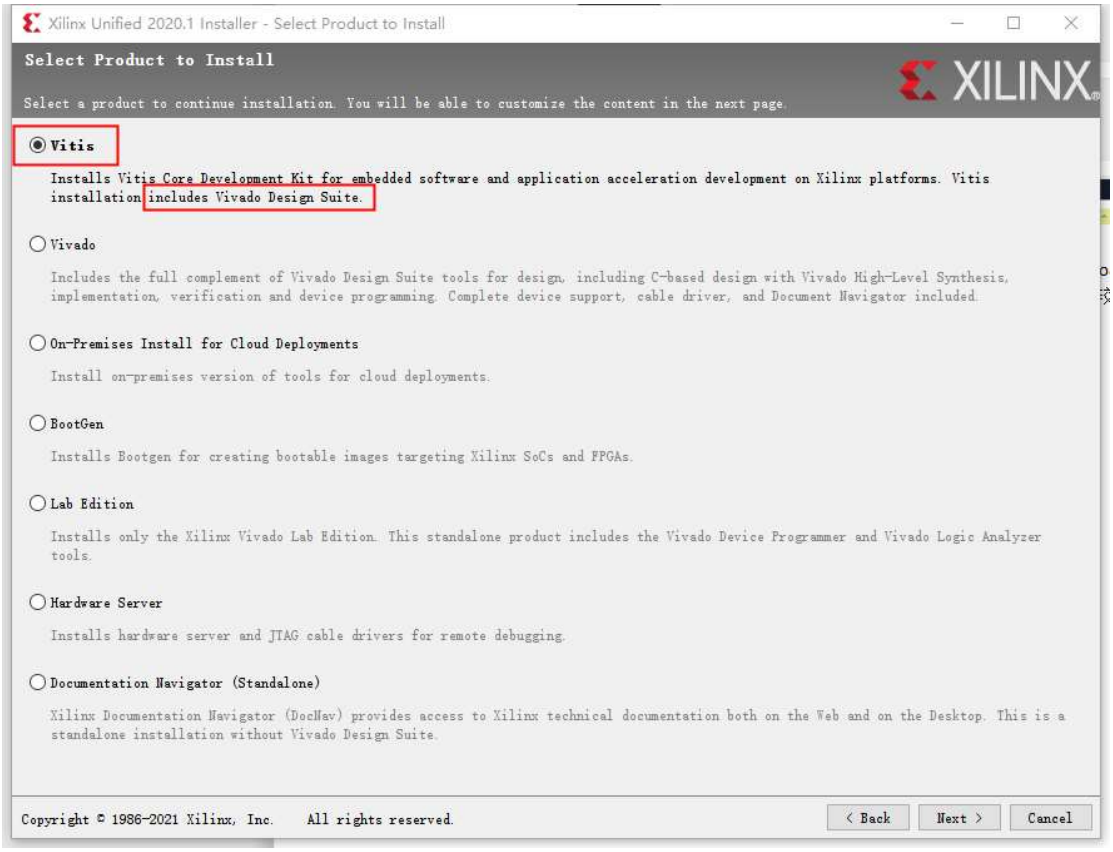


图 2-7 安装导航图

第五步：选择安装路径以及使用用户，如图所示

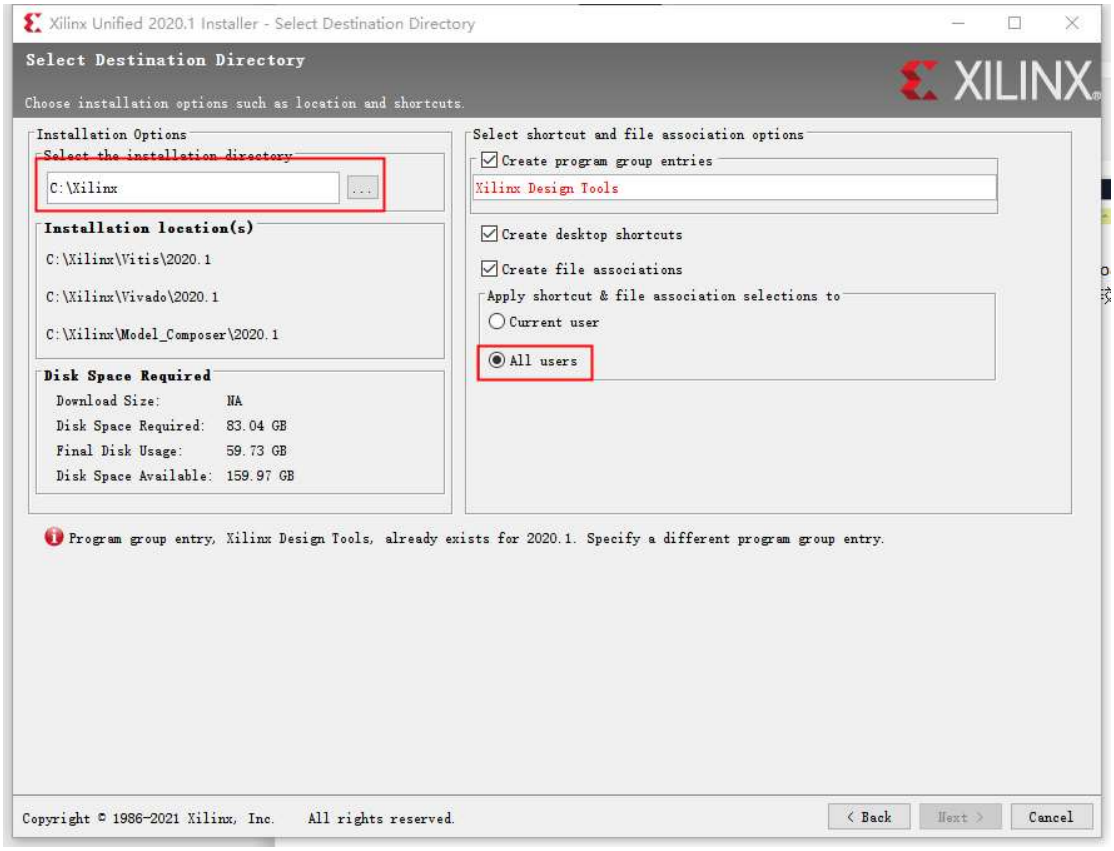


图 2-8 安装导航图-安装路径

第六步：打击 YES

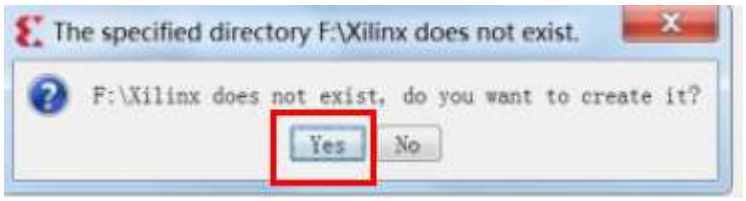


图 2-9 安装导航图-建立安装文件目录

第七步：单击 Install，安装需要 45 分钟左右。

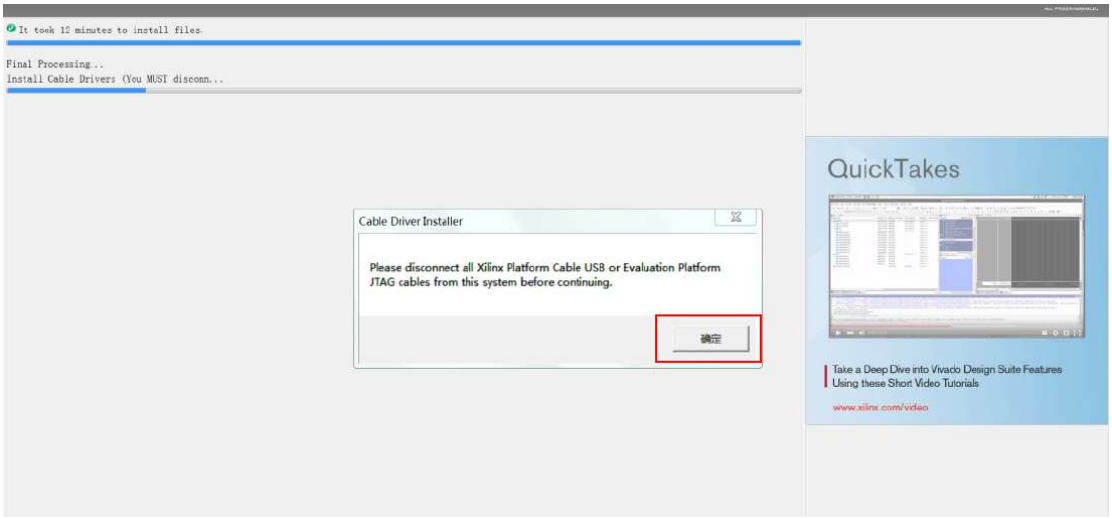


图 2-10 安装导航图

第八步：安装网络适配器

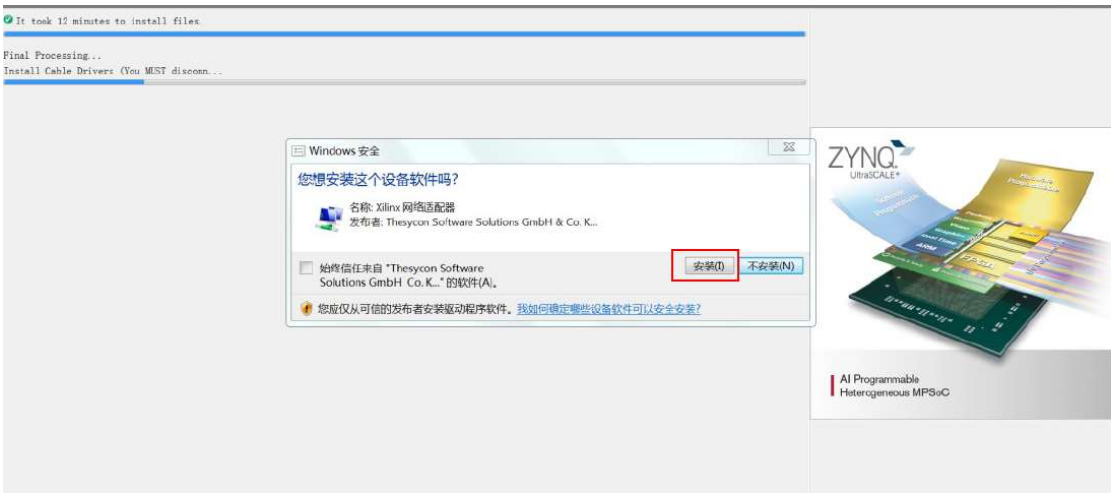


图 2-11 安装导航图

第九步：安装下载器驱动

图 2-12 安装导航图

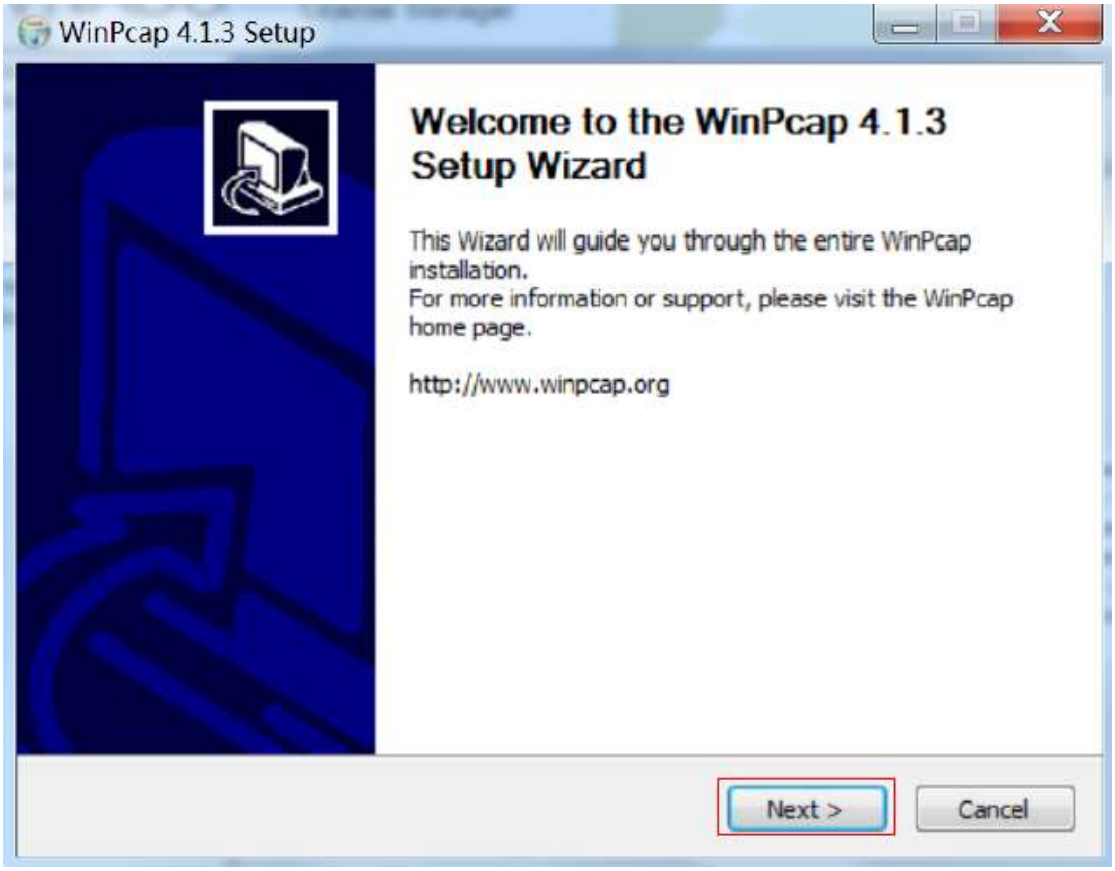


图 2-13 安装导航图

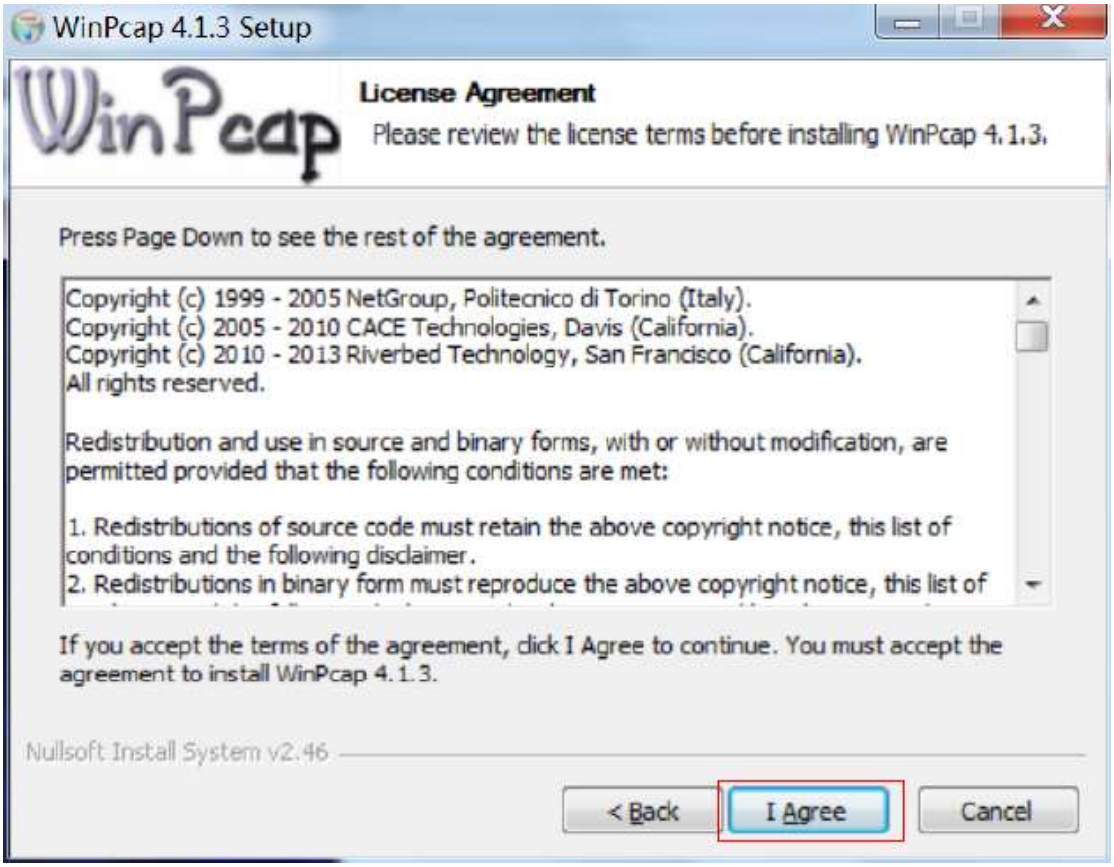


图 2-14 安装导航图-安装 wincap

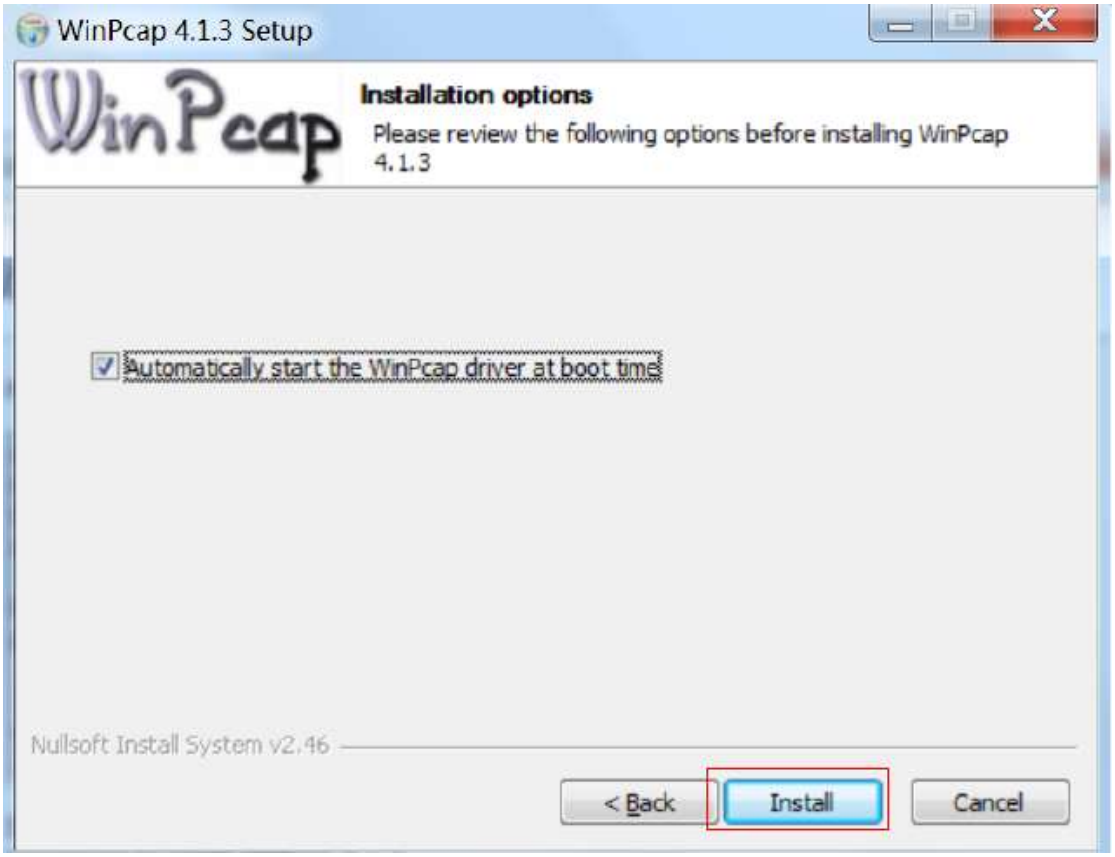


图 2-15 安装导航图-安装 wincap

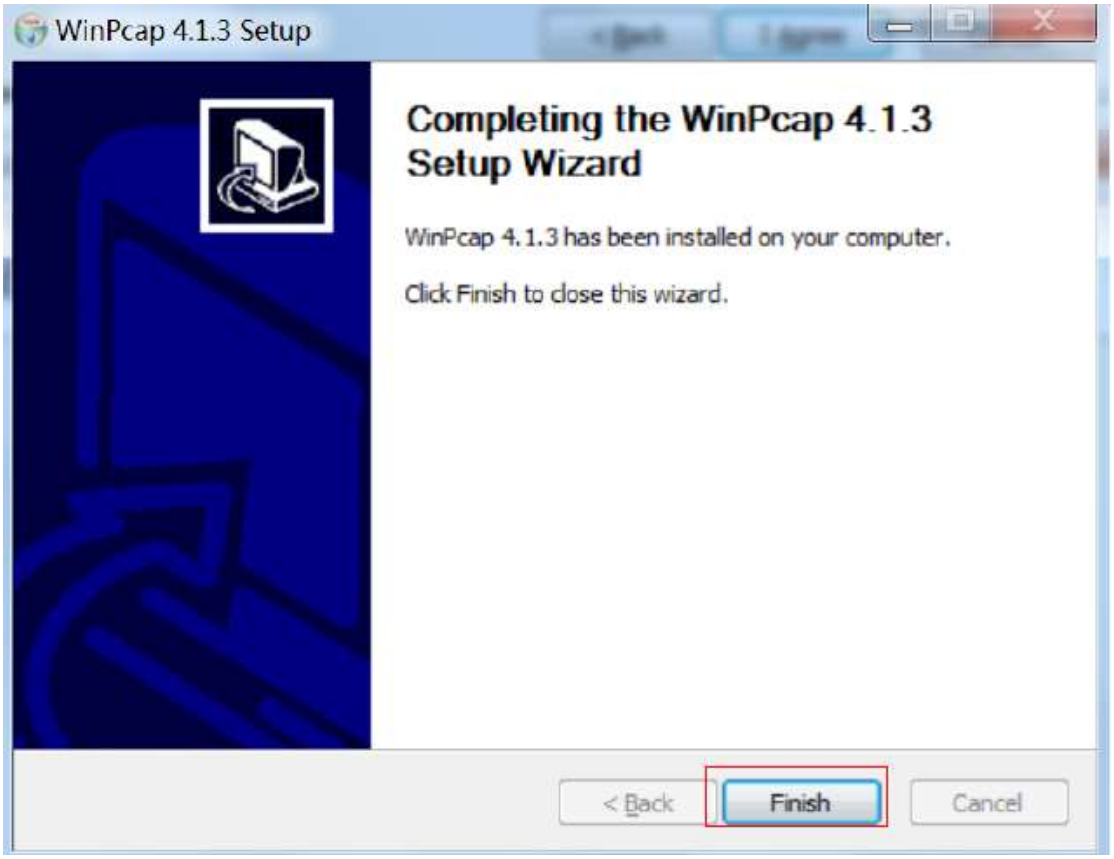


图 2-16 安装导航图

第十步：最后会弹出一个安装 MATLAB 的窗口，直接叉掉就可以，若果需要的话，可以自行安装。

第十一步：安装成功，弹出安装 license 的窗口，也直接叉掉。在下一节讲解。

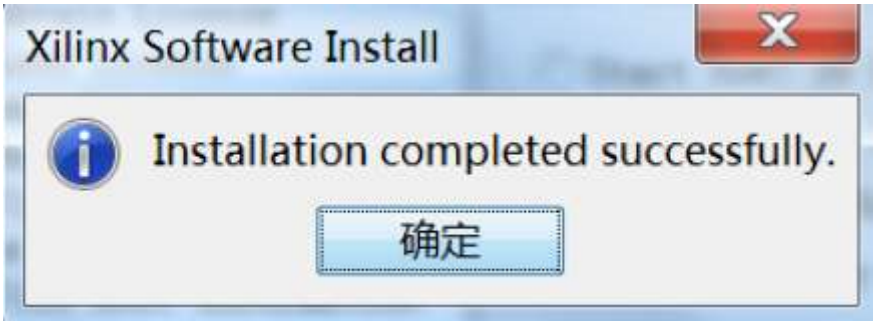


图 2-17 安装导航图-安装完成

2.2.3 vivado license 注册

上一小节直接在安装的时候就叉掉了 license 注册，现在重新打开 VivadolicenseManage，在开始菜单栏找到，

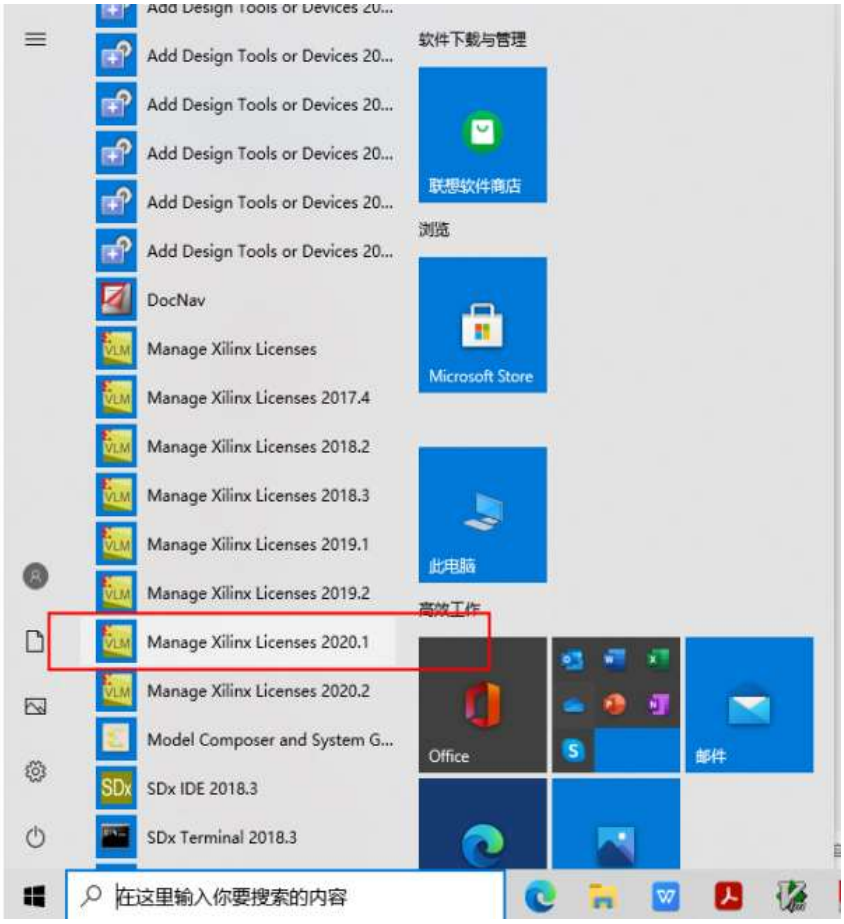


图 2-18 Manage Xilinx License

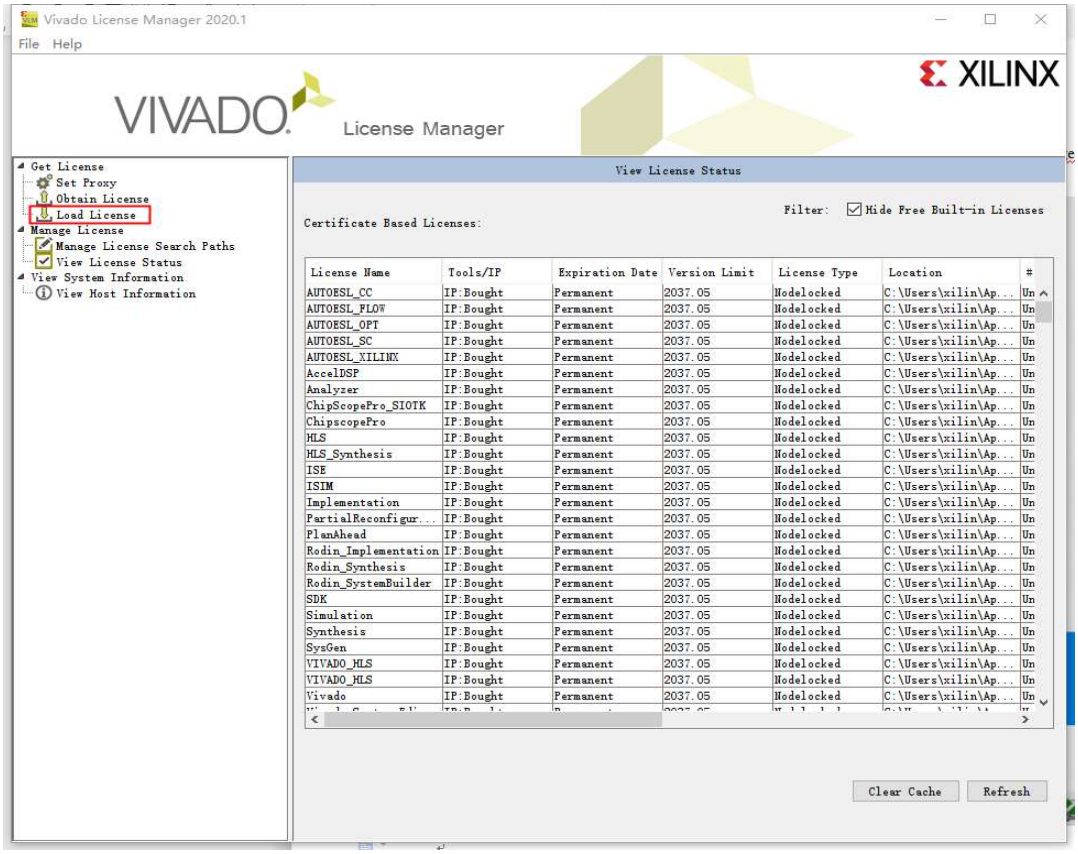


图 2-19 安装 license 界面 1

选中 Load License，然后就是选择 Copy License...

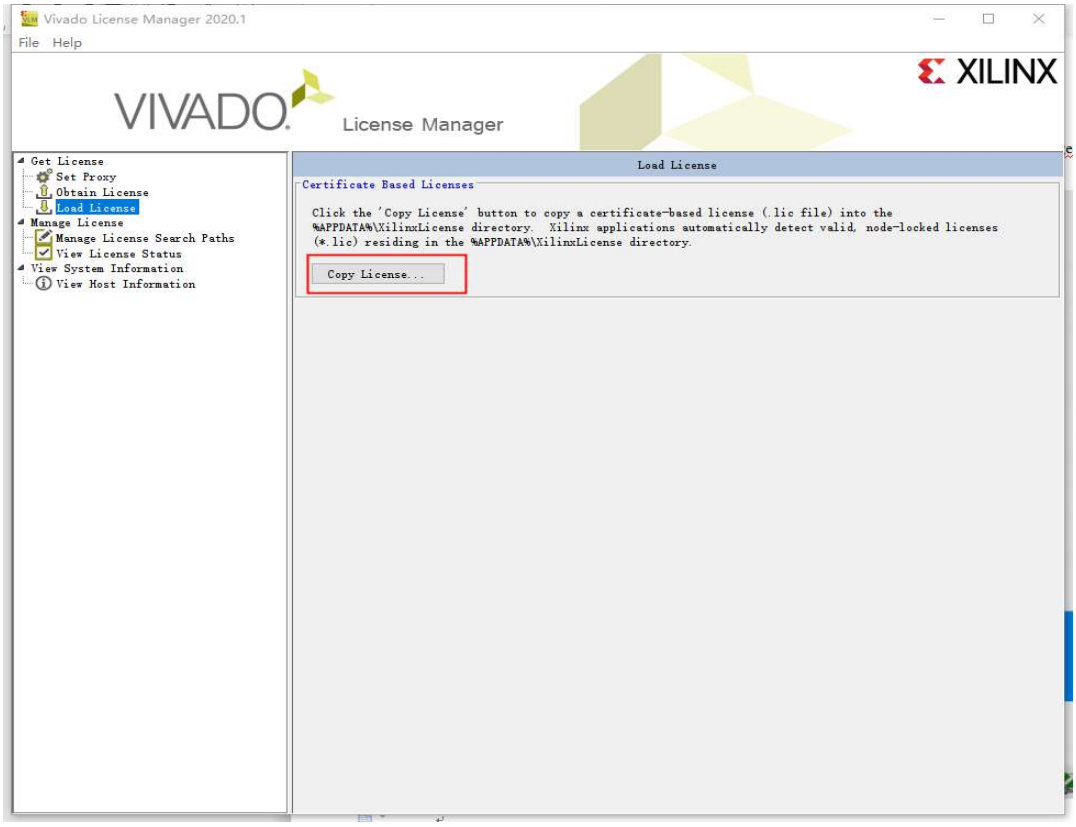


图 2-20 安装 license 界面 2

选择准备的 vivado 注册文档 Xilinx.lic

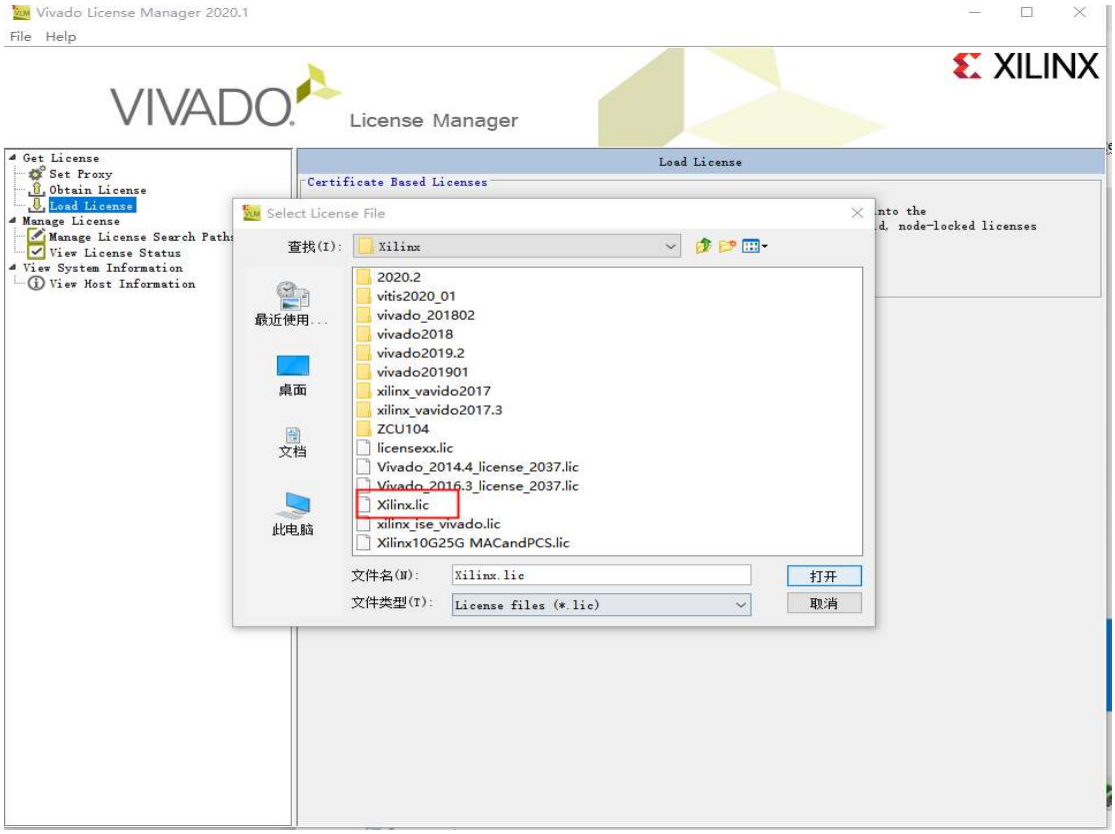


图 2-21 安装 license 界面 3

最后可以点击 View License State，可以看到已经注册好了 license。

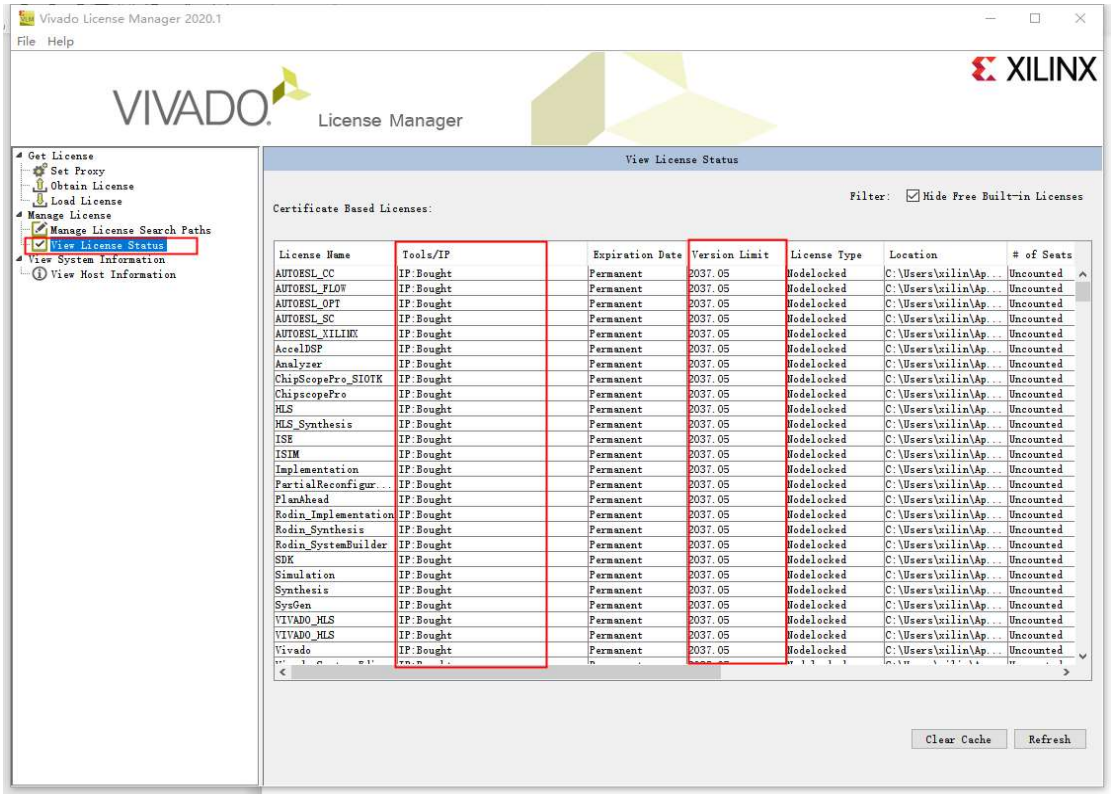


图 2-22 安装 license 界面 4

2.2.4 vivado 中需要付费 IP 的临时 license 注册

我们以一个需要付费的 IP 进行举例说明，如果我们要用到 10G/25G Ethernet Subsystem 这个 IP，我们在配置的时候，IP 的状态是 IP :Design_Linking IP License available

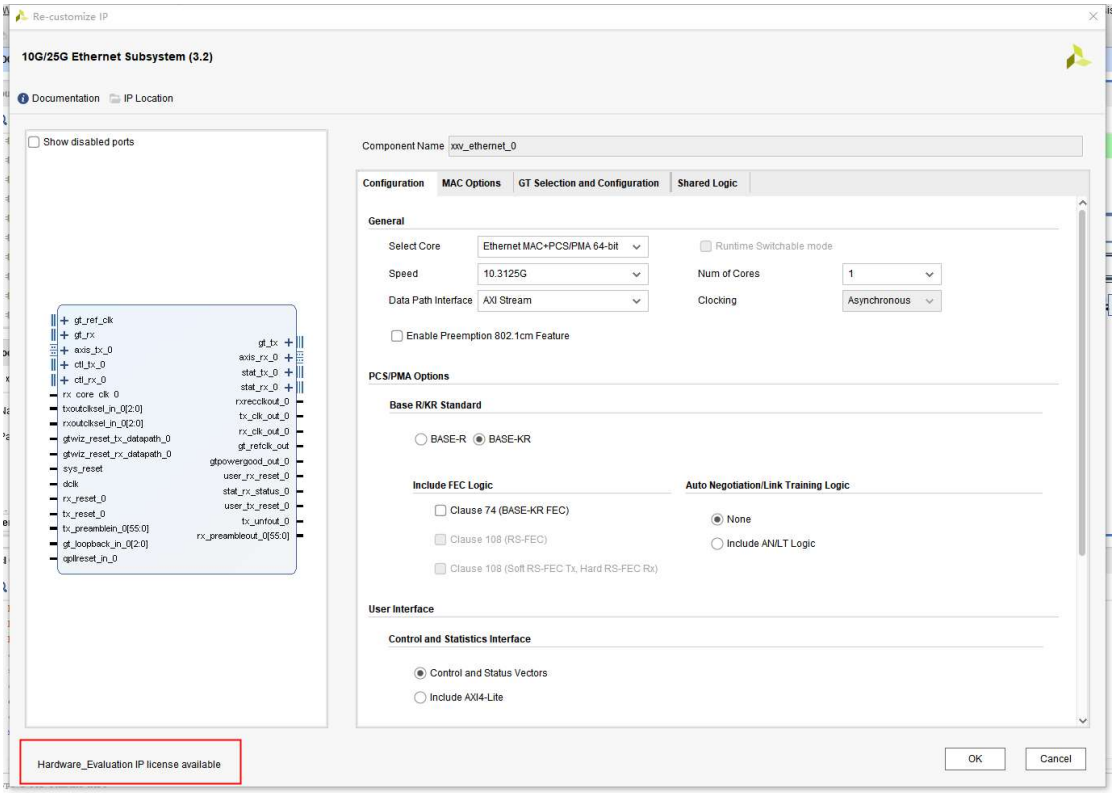


图 2-23 安装 license 之后的 IP 状态界面

在 Vivado License Manager 可以看到

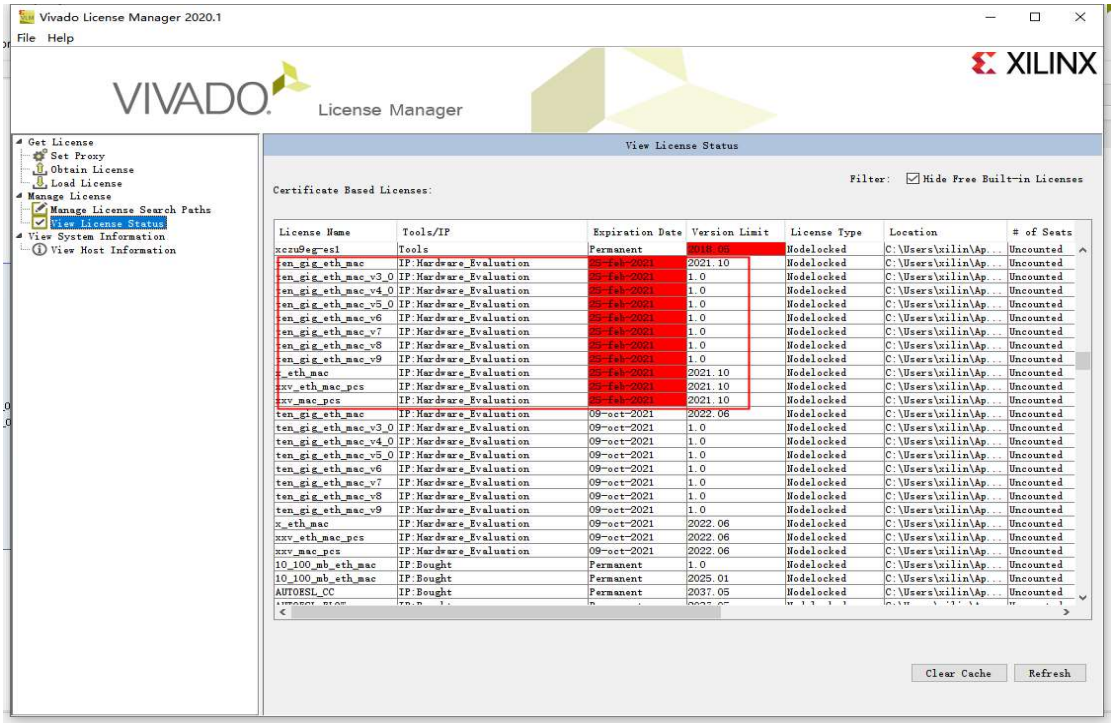


图 2-24 安装 license 界面 5

这个时候是 License 过期了，可以仿真但是不能生成 bitstream。接下来我们介绍一下怎么申请特定的 license.按照图示的步骤操作即可。

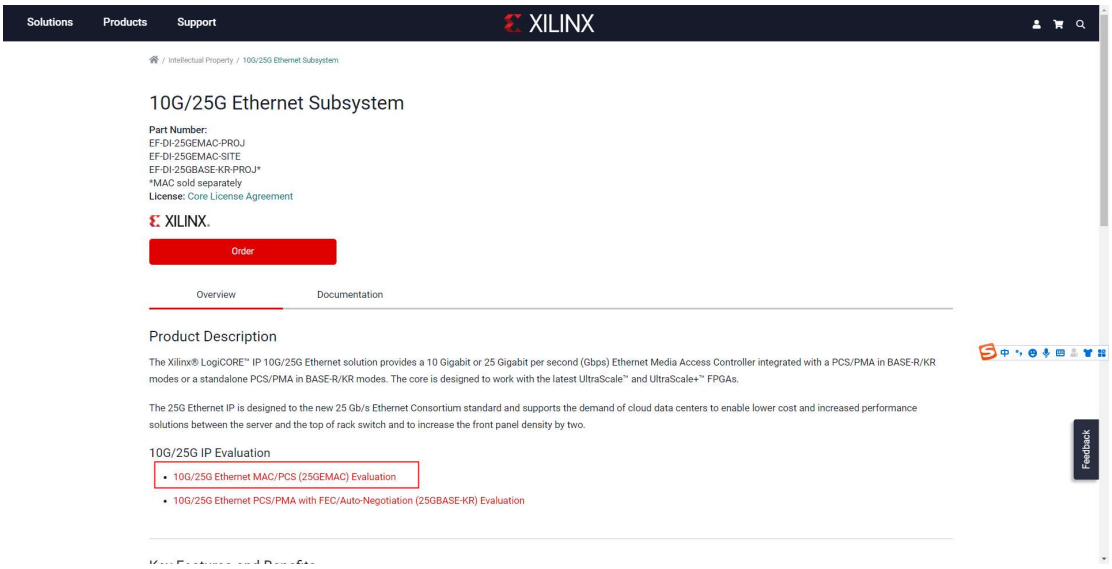


图 2-2510G/25G Ethernet Subsystem IP 的介绍页面

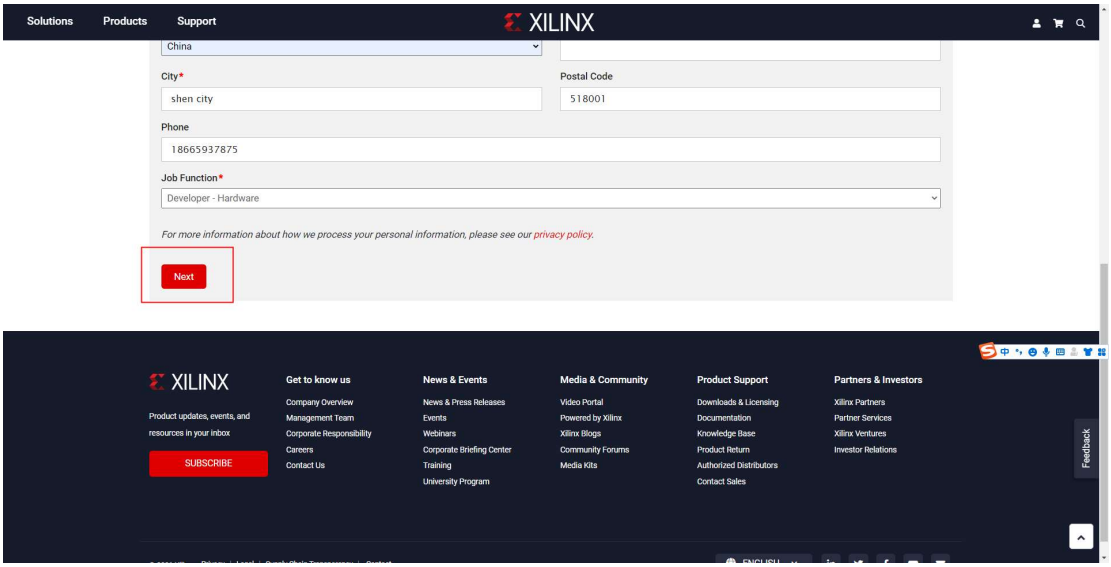


图 2-26 进入 license 申请的许可页面

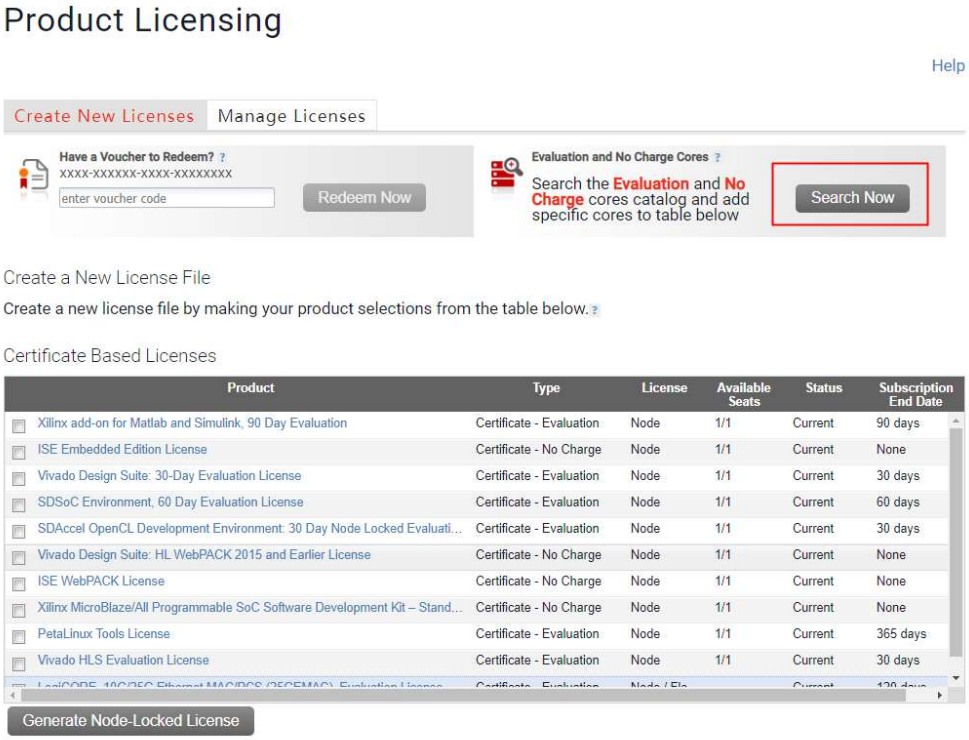


图 2-27Product Licensing 页面

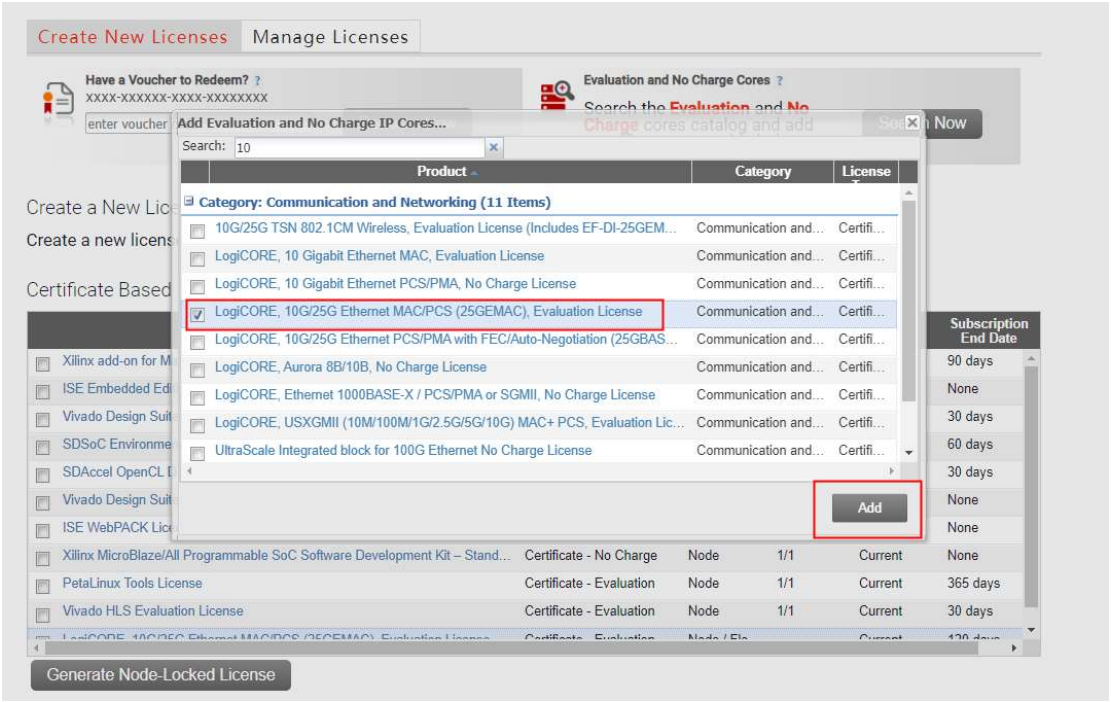


图 2-28 查找 license

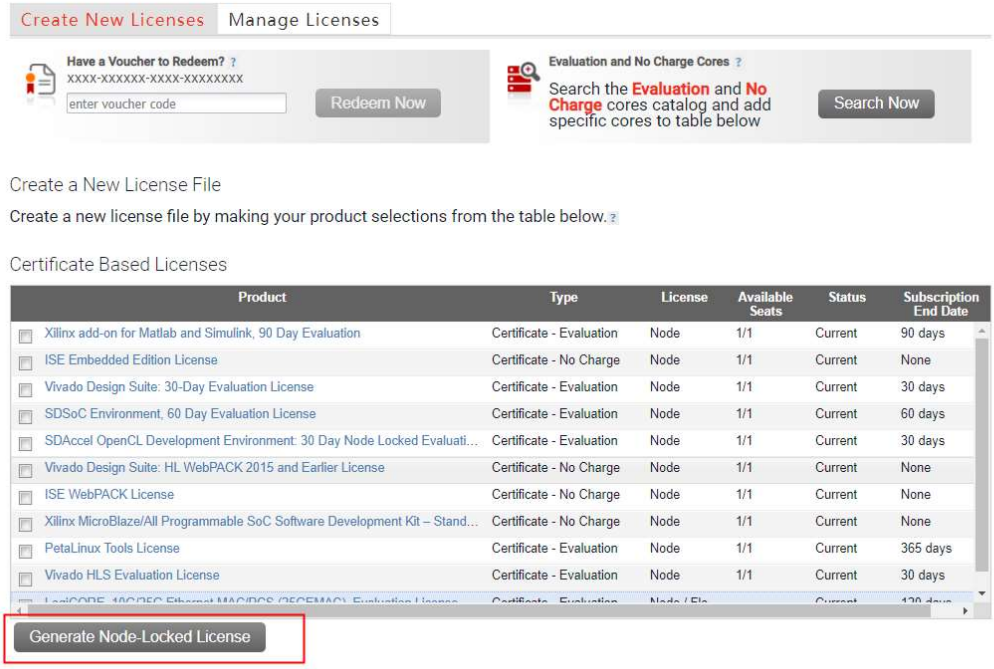


图 2-29 生成 license

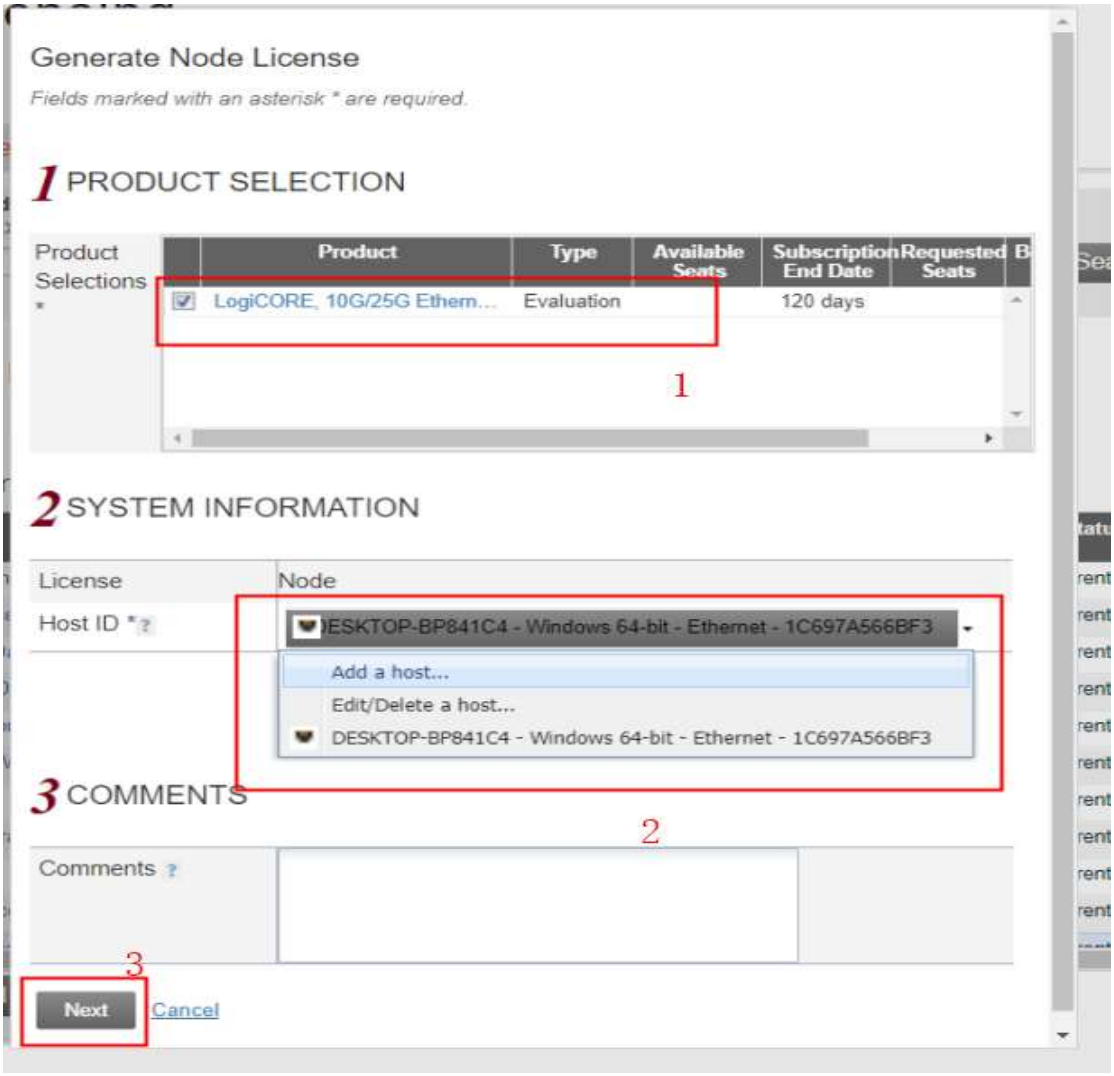


图 2-30 填写电脑 MAC 地址生成 license

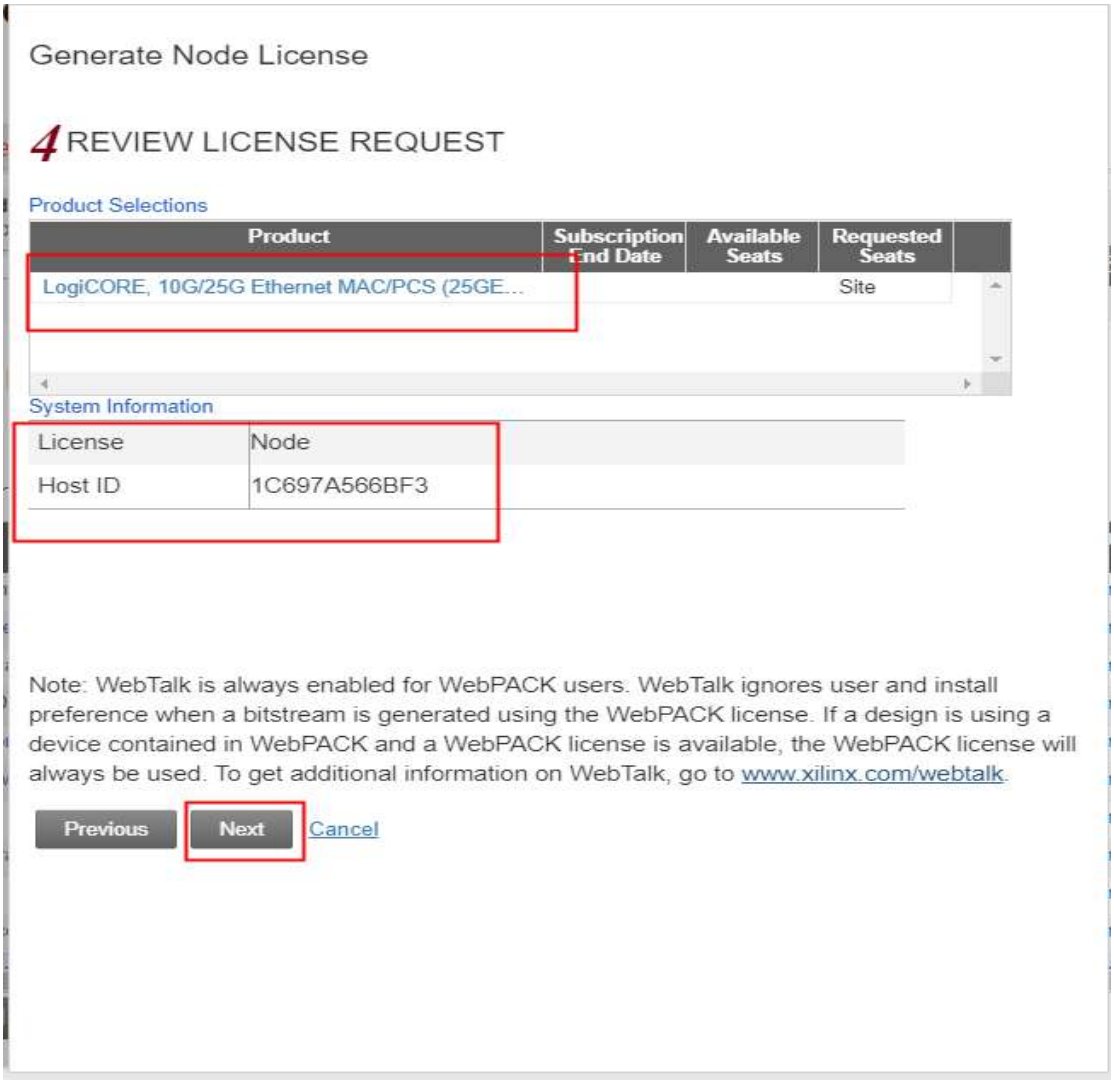


图 2-31 生成 license 确认界面

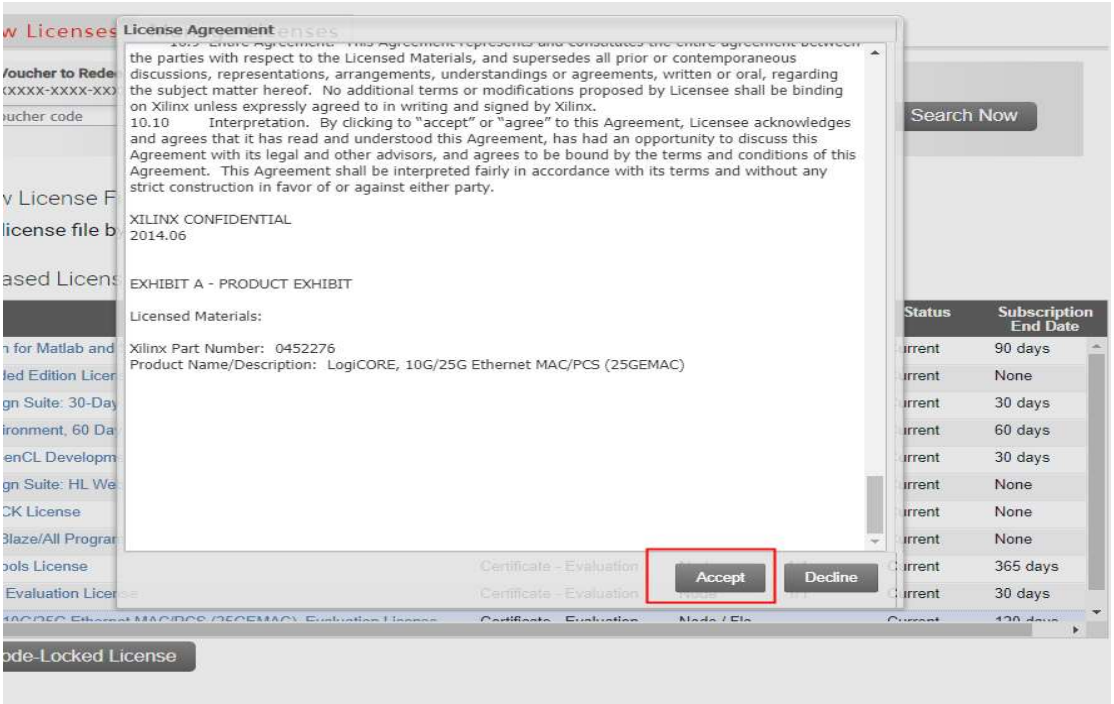


图 2-32 同意 xilinx 使用 license 的协议

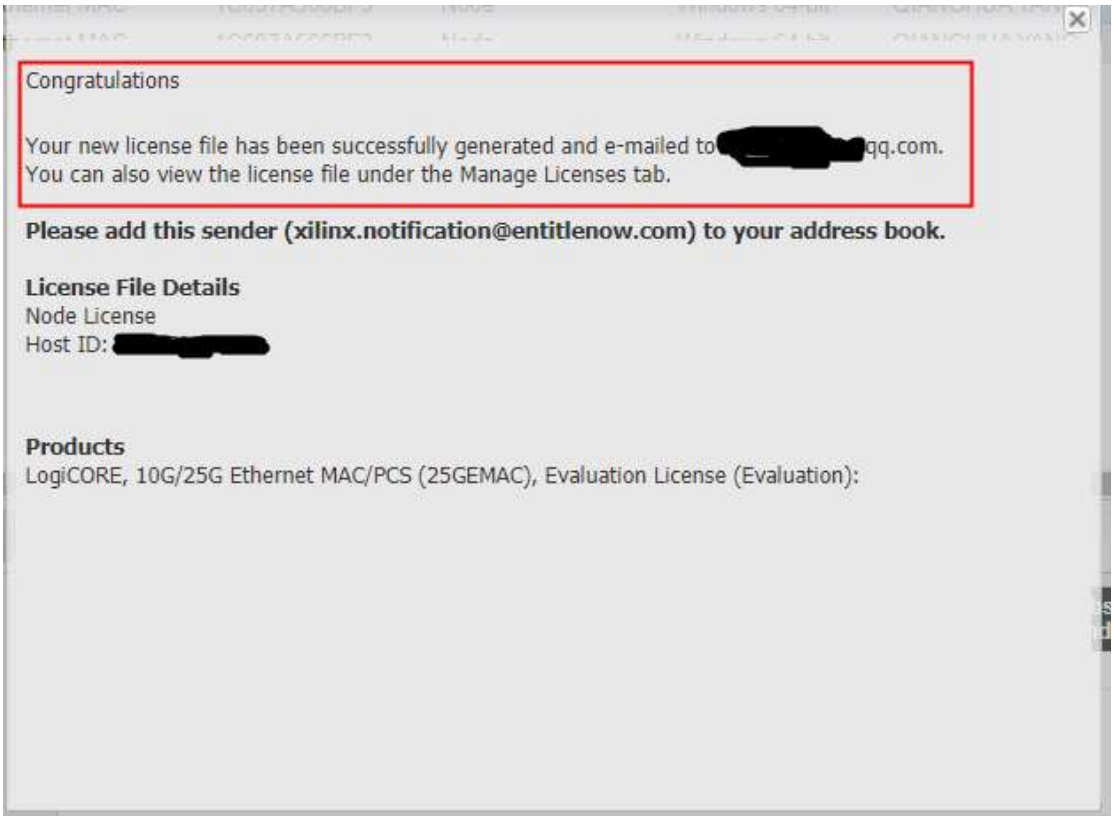


图 2-33 发送 license 之后的结果页面

到此为止，可以在注册的邮箱收到对应的 Xilinx.lic

图 2-34 邮箱收到 license 的附件

将收到的 Xilinx.lic 下载下来，然后重新通过 Vivado License manager 注册 License（重复 2.2.3 节的操作），如图

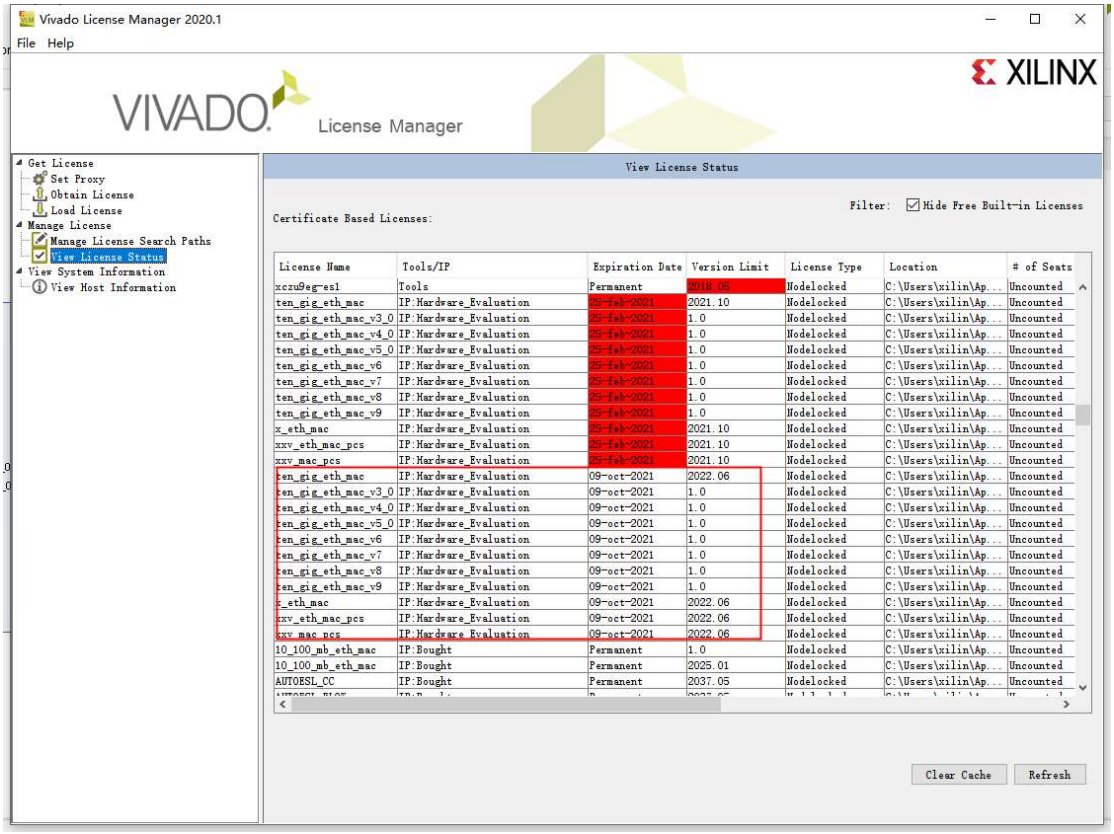


图 2-35 安装 license 之后的界面

至此，对应的 IP 状态变成了 Hardware_Evaluation IP License available

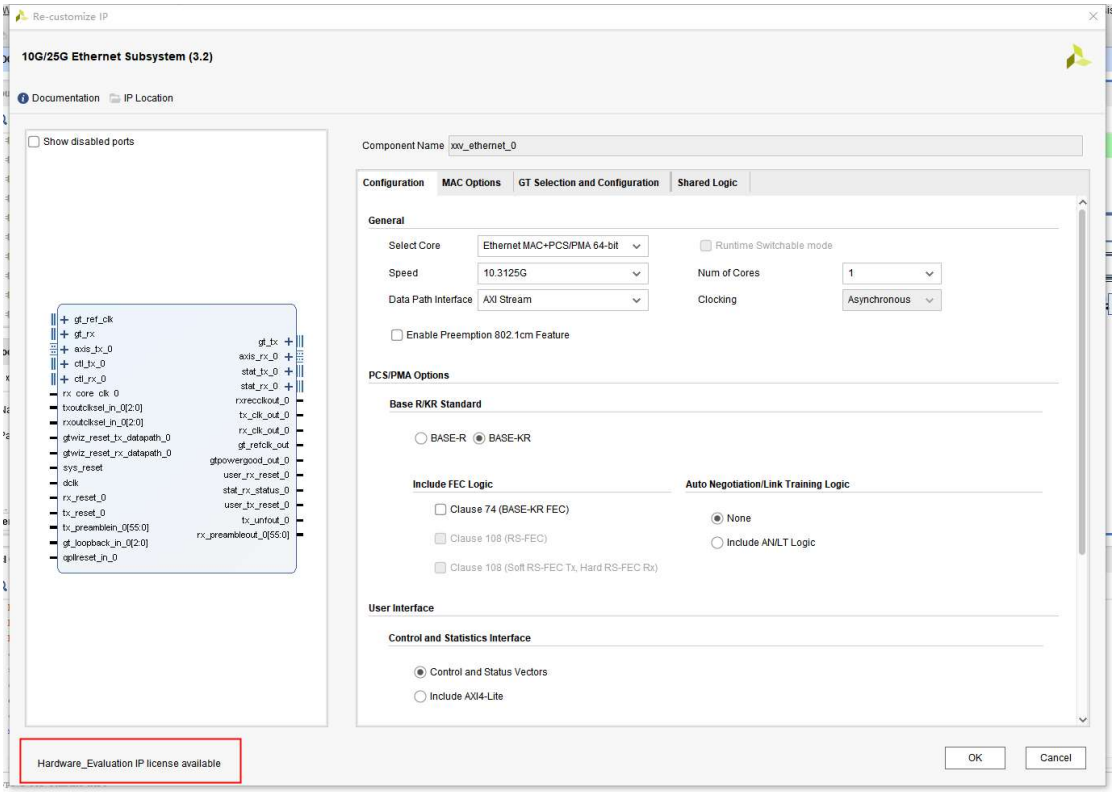


图 2-36 安装 license 之后的 IP 状态界面

- 到这里就可以使用 10G/25G Ethernet MAC/PCS (25GEMAC) Evaluation。

2.2.5 modelsim 下载安装

modelsim se 2019 是一款在原版本软件功能和性能基础上得到改进以及优化的最新版本 HDL 语言仿真软件，使其软件功能性更加完善。2019 新版本提供全面完善以及高性能的验证功能，全面支持业界广泛的标准；另外相比老版本，仿真速度要快 10 倍，并且图形用户界面功能强大，所有窗口都会在任何其他窗口中自动更新活动。比如在 Structure 窗口中选择设计区域会自动更新 Source，Signals，Process 和 Variables 窗口。您可以在不离开软件环境的情况下编辑，重新编译和重新模拟，所有用户界面操作都可以编写脚本，模拟可以批量或交互模式运行，是 FPGA/ASIC 设计的首选仿真软件。

Modelsim 是 Mentor Graphics Corporation 公司的仿真工具，这里下载的方法可以找 eetop 网站找到相应的安装包，或者到 Mentor Graphics 公司官网下载。

2.2.6modelsim 安装

1、解压软件安装包压缩包，得到安装程序（补丁文件）。

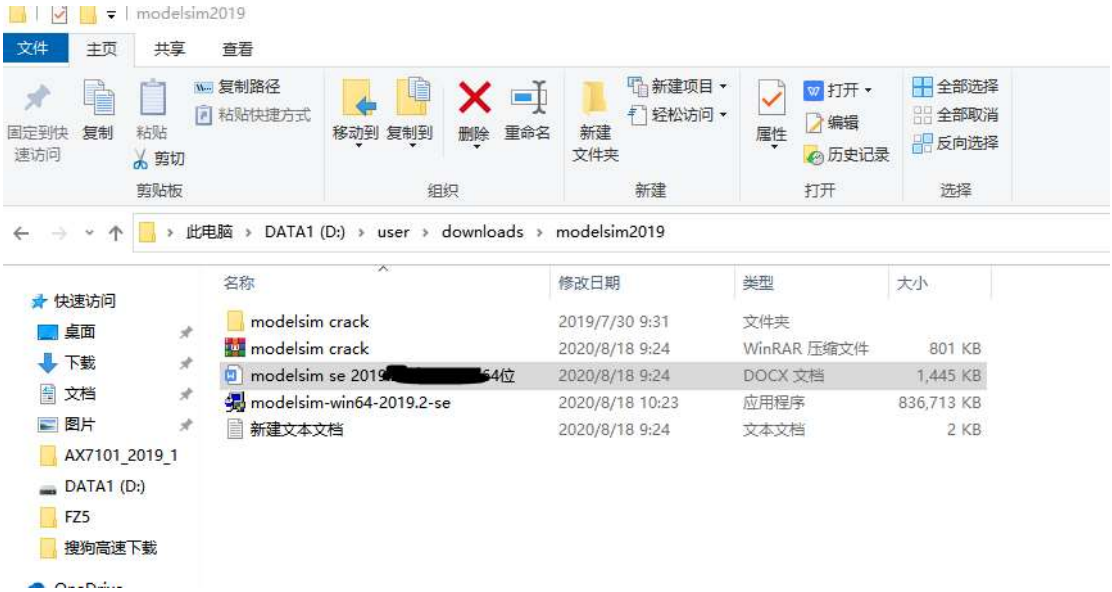


图 2-37 modelsim 安装文档

2 然后双击运行 exe 文件夹程序进行软件安装，弹出界面，进入安装向导界面，点击下一步继续安装。

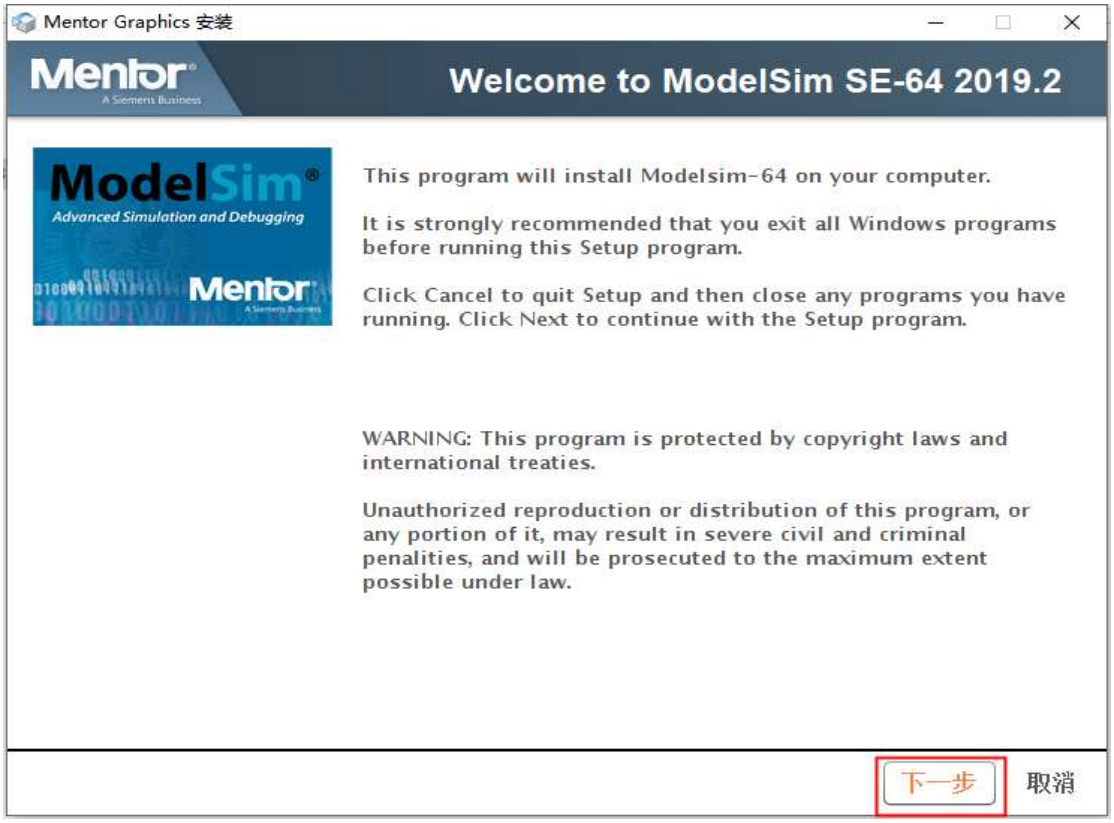


图 2-38 modelsim 安装

3，选择软件安装路径，点击浏览可更改路径，也可按照默认设置安装路径即可。

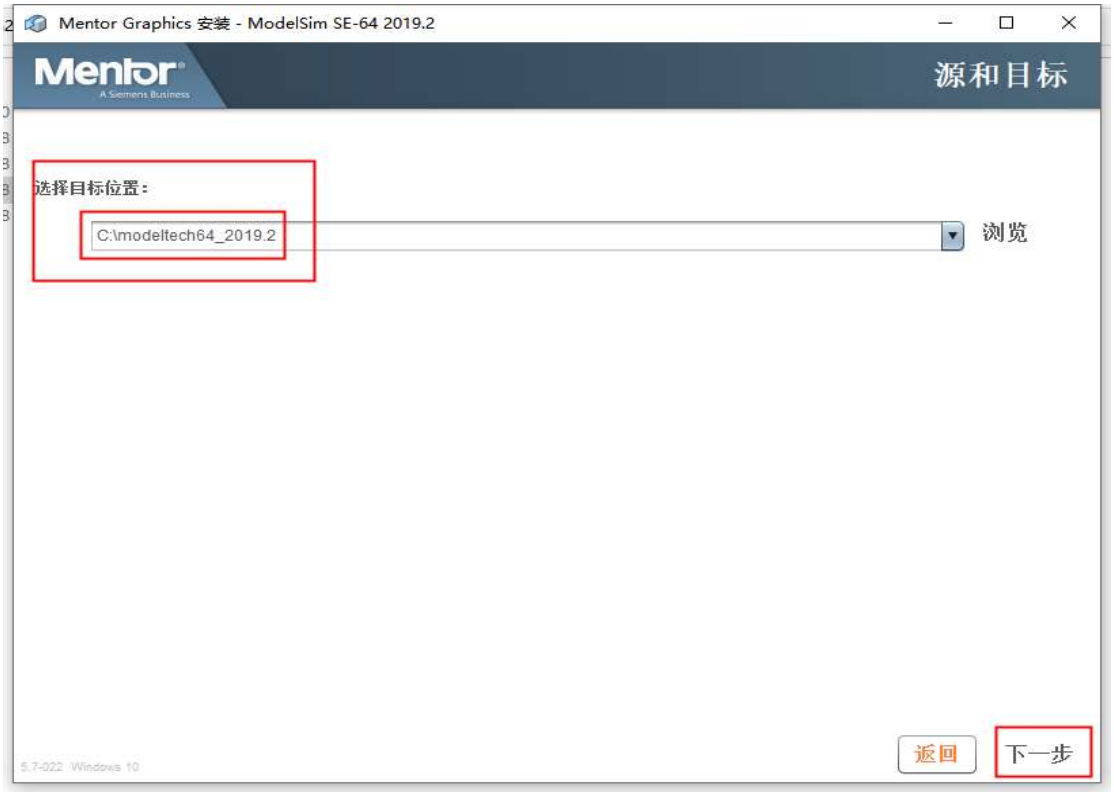


图 2-39 modelsim 安装路径

4，同意相关许可声明

图 2-40 modelsim 同意许可声明

5，软件进入安装状态，正在安装，安装过程需要一些时间，等待即可

图 2-41 modelsim 安装过程

6，在安装过程中弹出如下窗口，点击否

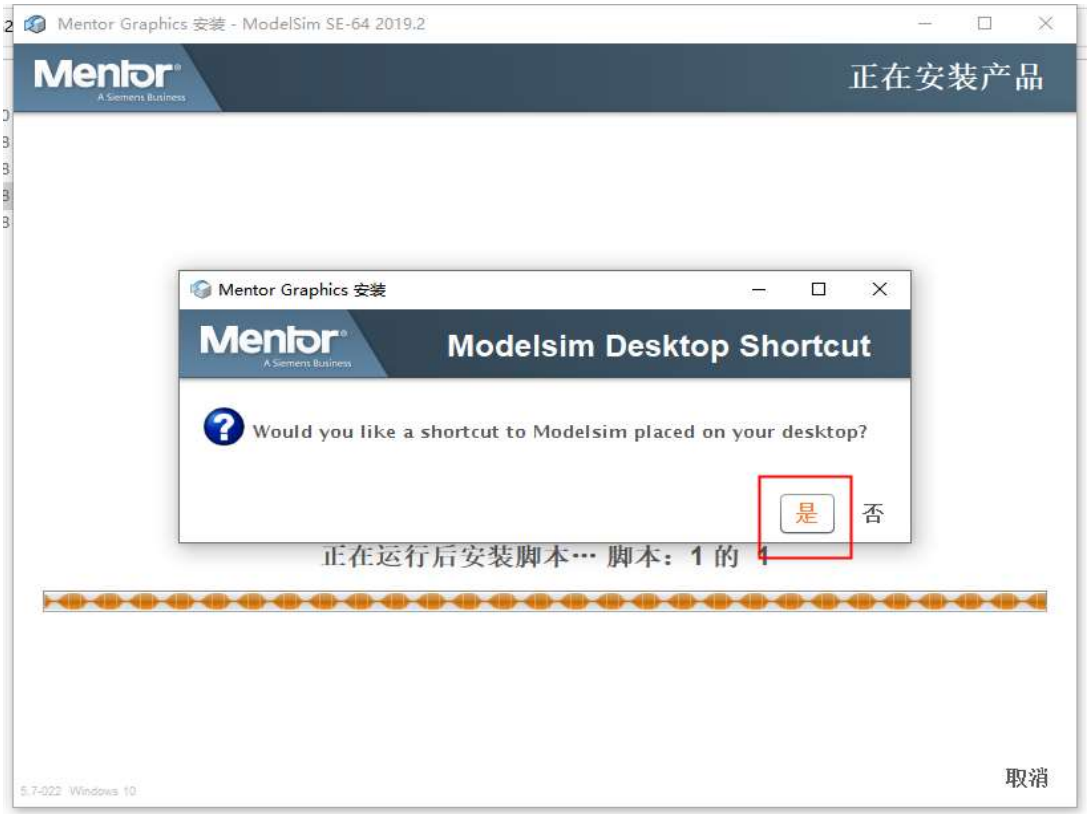


图 2-42 modelsim 安装桌面快捷方式

7，在最后一步安装硬件关键驱动的选择上，选择否，不要安装。至此完成了 modelsim 的安装。

2.2.7modelsim 的第一次运行方法

第一次运行方法这里涉及到 modelsim 的使用版权问题，我们会将对应的补丁文档，以及环境变量的更改都写进另一个文旦放到网上，可以供客户下载使用。文档的名称是“modelsim se 2019.2 最新破解版 64 位.docx”。下载的链接给出了百度网盘的永久下载链接：

链接：https://pan.baidu.com/s/1iEAiTedHffJunc14N-nXaA

提取码：hbcm

或者客户可以联系本公司的技术支持，获取相应的文档。

2.2.8 modelsim 工程建立以及仿真

1，新建 modelsim 工作文档，后面新建工程和源码文档都会放到相应的文件夹。

2，打开 modelsim 软件，选择 file->new->project

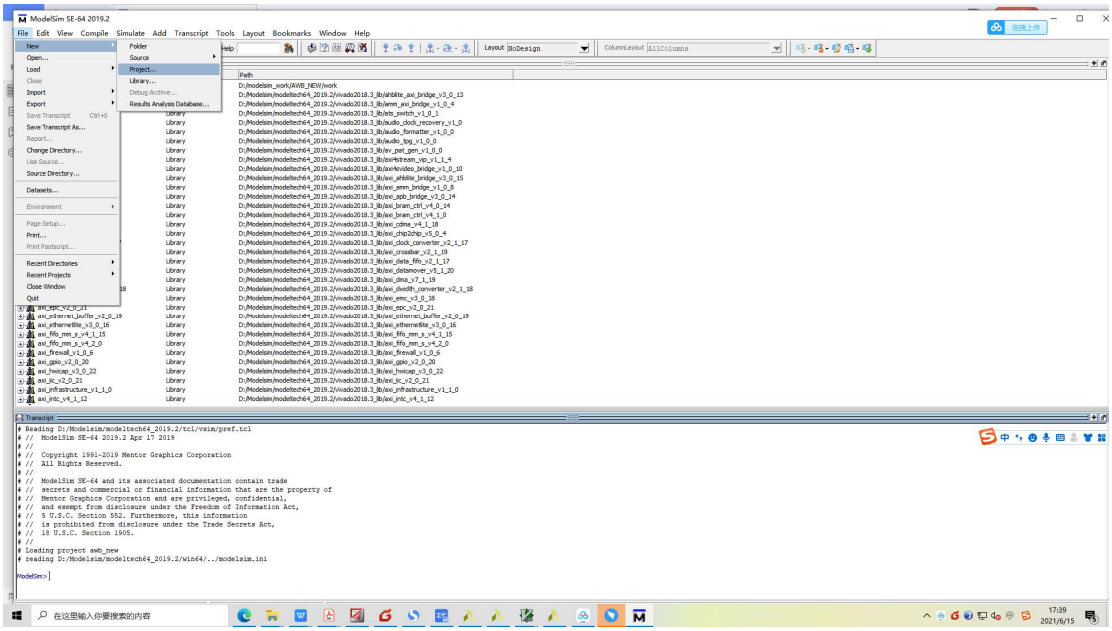


图 2-43modelsim 新建 project

3,设置工程名称，以及选择刚刚新建的工作文件夹

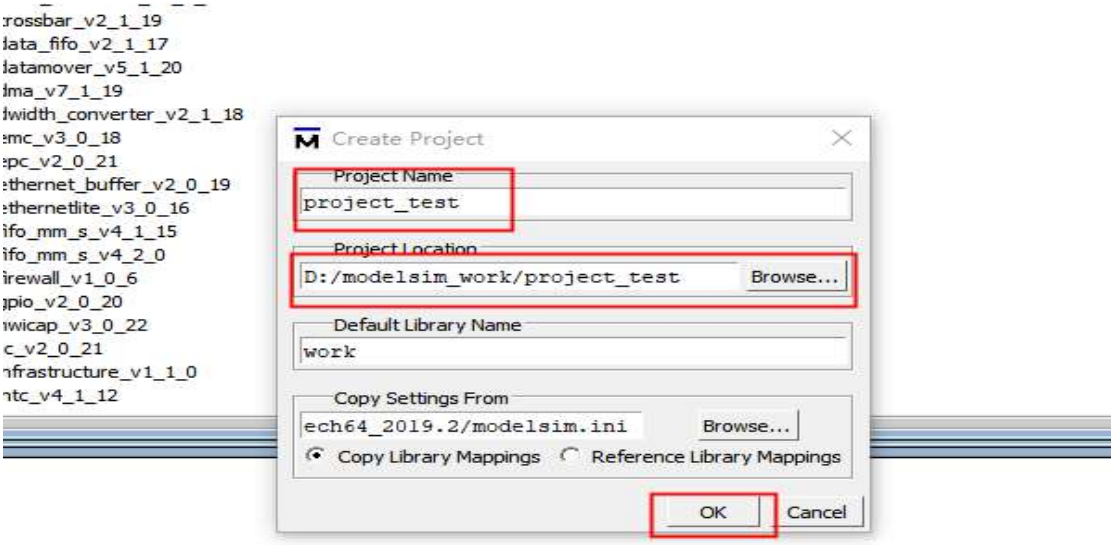


图 2-44 modelsim 新建 project 名称

4，选择 Add existing files ->然后选择 RTL 文件夹中的设计文档

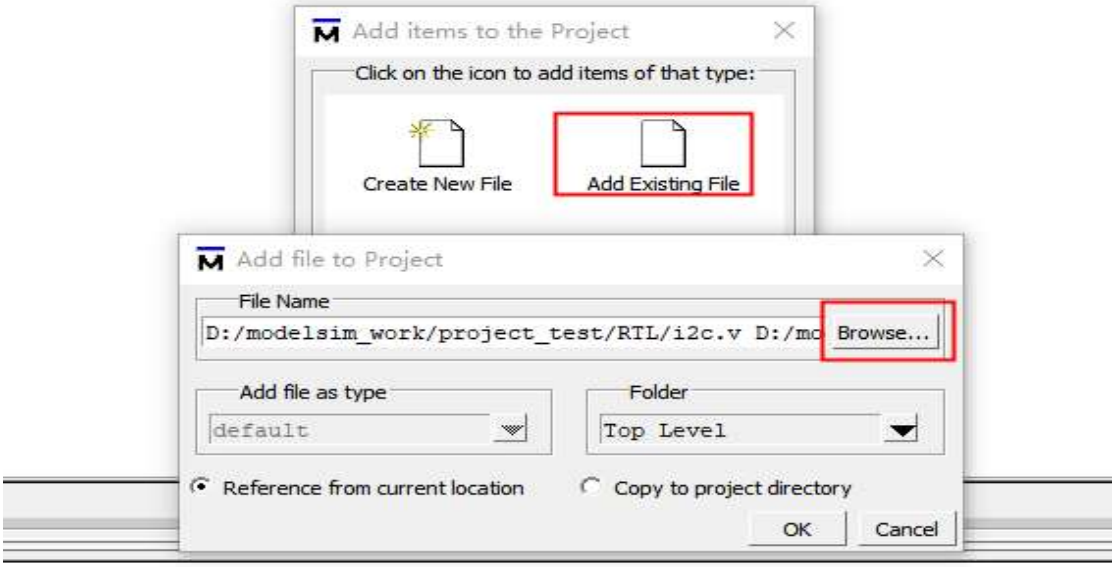


图 2-45 project 添加设计文档

5，compile-> compile ALL

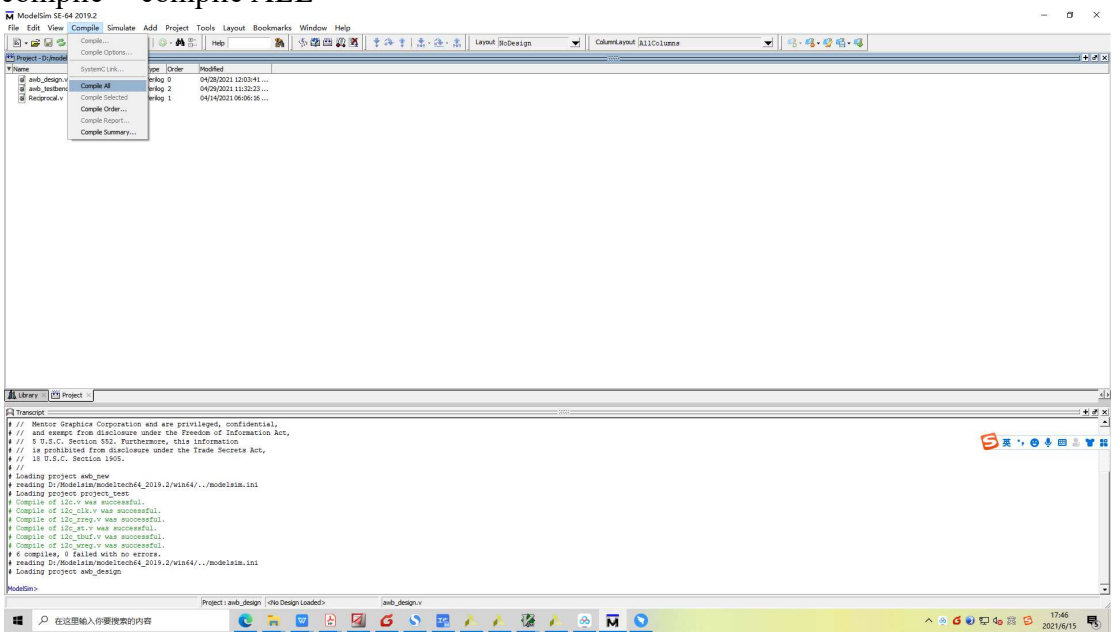


图 2-46 modelsim 编译

7,回到 library 窗口

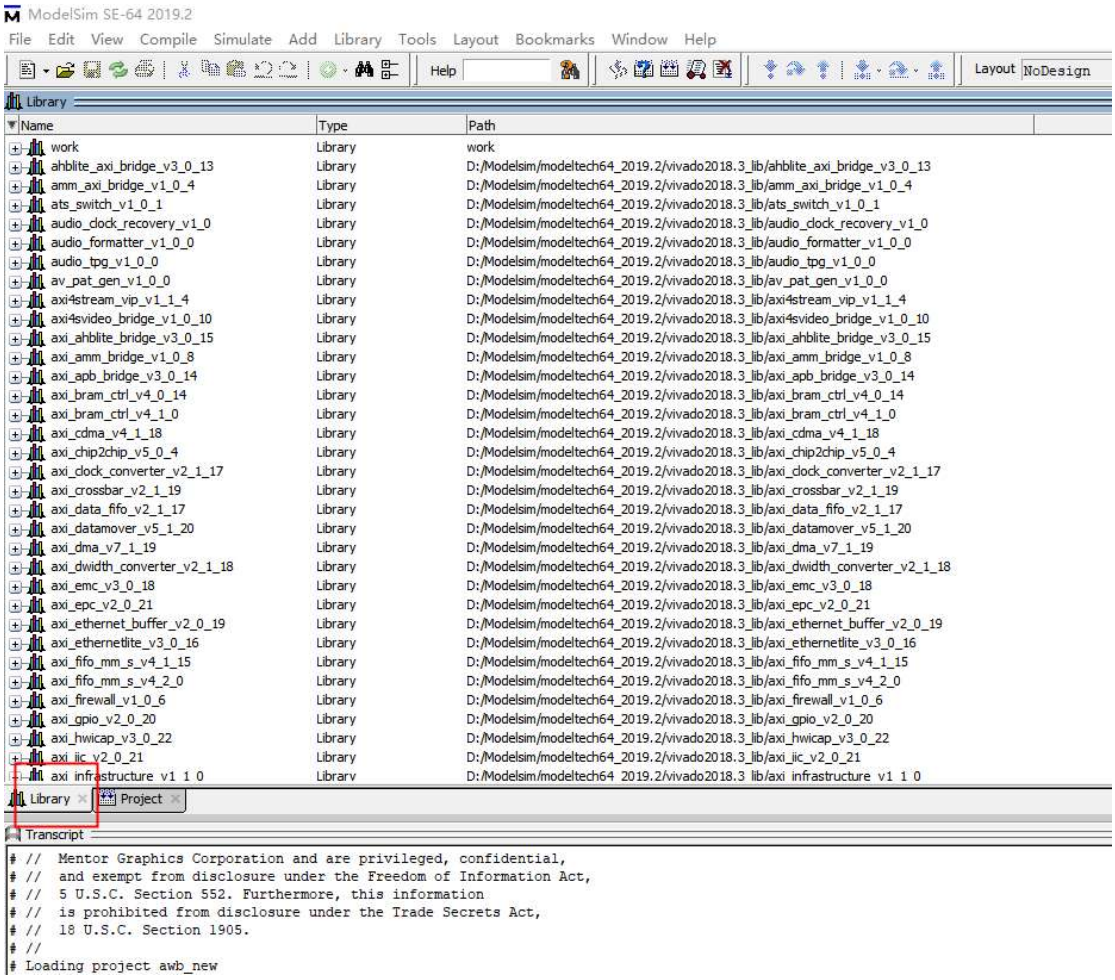


图 2-47modelsim 的 library

8，打开 work library，选择测试平台，右键选择 simulation：

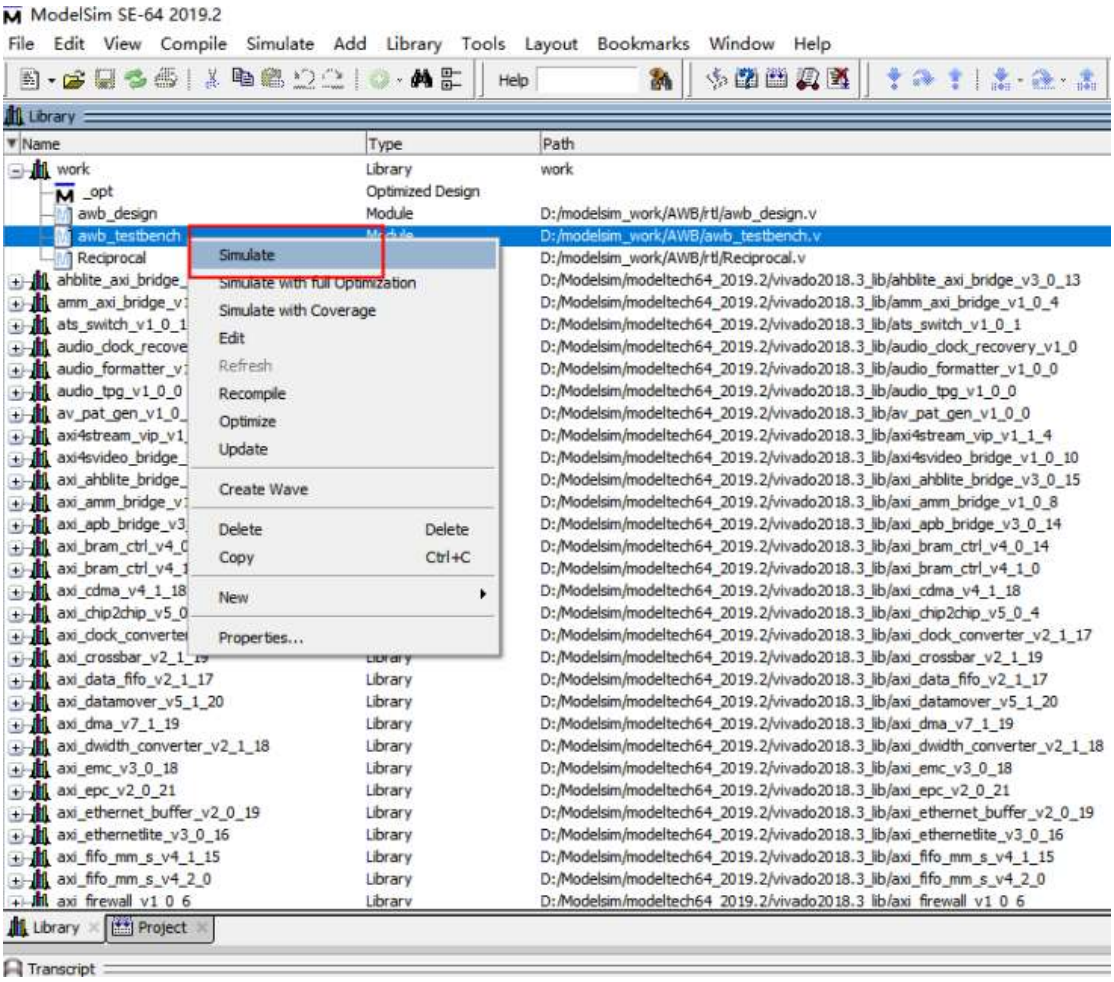


图 2-48 modelsim 仿真启动

至此就可以看到仿真的波形窗口，如果出现错误，就根据提示进行修改即可。

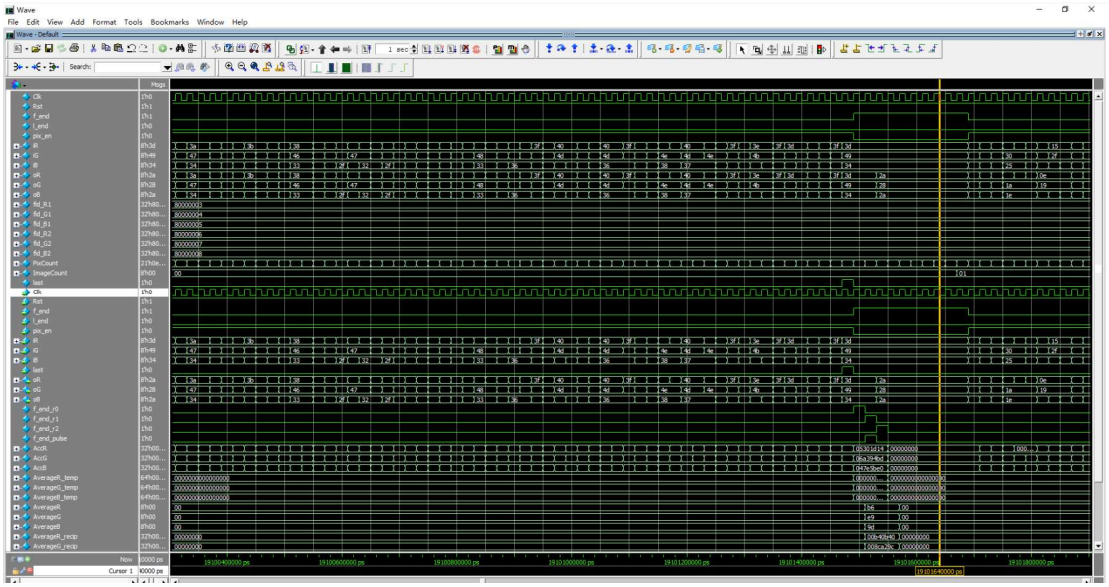


图 2-49 modelsim 仿真结果页面

2.3 知识准备

2.3.1 verilog 语法简要介绍

Verilog HDL 是一种硬件描述语言，以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。 Verilog HDL 和 VHDL 是世界上最流行的两种硬件描述语言，都是在 20 世纪 80 年代中期开发出来的。前者由 Gateway Design Automation 公司（该公司于 1989 年被 Cadence 公司收购）开发。两种 HDL 均为 IEEE 标准。下面介绍 verilog 的一些可综合的语法知识：

1，常量

Verilog 的文档中可以存在常量，在预处理阶段会转变为逻辑电路对应的常量寄存器或者直接忽略。

比如整数，parameter DLY= 1；这里就是定义一个常量 DLY，整数可以用二进制 b 或 B，八进制 o 或 O，十进制 d 或 D，十六进制 h 或 H 表示，例如， 8’b00001111 表示 8 位位宽的二进制整数，4’ha 表示 4 位位宽的十六进制整数。

比如下划线：assign a = 8’b0000_1111; 这里的下划线是为了方便阅读。

比如 X,或 Z：X 表示变量处于不确定的状态，a = 8’bxxx0_11xx;这里表示变量 a[7:0]的 0,1,5,6,7bit 位是不确定值，也就是说，在实际的结果中肯定是有一个结果，但是我们不能从逻辑上确定这个结果的准确值是 0 还是 1.从整体上来说整个 a 的值就是处于不确定状态。如果 a = 8’bzzz0_11zz,这里表示变量 a[7:0]的 0,1,5,6,7bit 位是高阻态，顾名思义就是阻力足够大以至于达到开路到程度，相当于悬空，他的值根据后面接的逻辑电路有关。

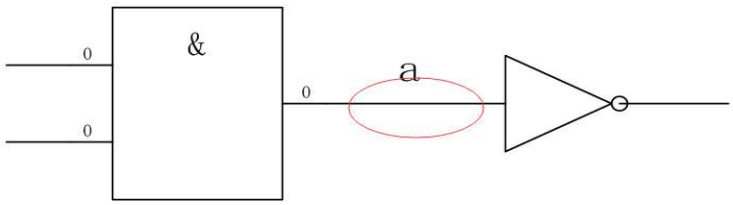
2，变量

指的是运行过程中可以改变值的量，verilog 中可以改变值的量是寄存器类的变量，比如 reg 类型，同事与 reg 相连接的线的值也同样改变，这一个类型是 wire 类型变量。还有一个就整块的数据变化的量，可以用来存储数据的，RAM，ROM 等以及经常使用到的 DDR，SDRAM，FLASH 等等。这些器件都可以在 verilog 的语法中出现例化，自然可以存储改变数据。这里介绍 wire 以及 reg。

Wire 类型：

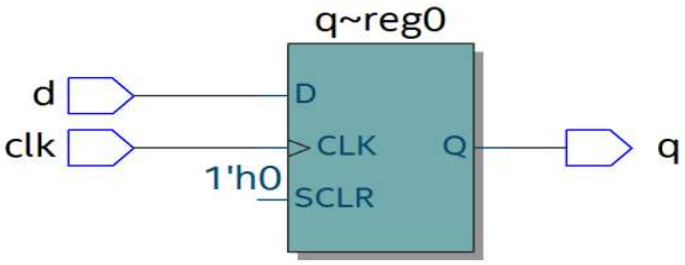
wire 指的是线型变量，用 wire 定义一个变量，wire [7:0] a;在综合的过程中是有一条 8bit 位宽的连线，至于连线的两端链接什么，需要到具体的设计中才会知道。比如 assign a = 8’b0000_1111;这里就是将 a 链接到 8bit 常量寄存器的输出端，从而整条线上的值发生了变化，一直都是 8’b0000_1111.在比如将 a 链接到寄存器的输出端，这里寄存器需要有 8bit 的位宽，而且在 verilog 中只能用 assign 语句。当然赋值给 a 的位宽主要是 8bit 的就可以。

如图所示：



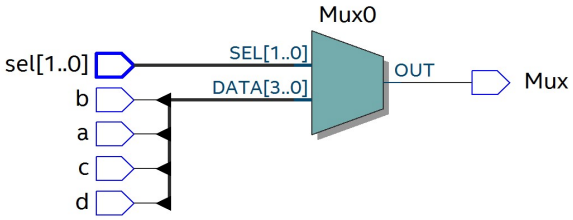
Reg 类型：

这个在 verilog 中称为寄存器变量，在电路中是使用寄存器代表相对应的变量，变量的位宽和数量与在电路中实现的寄存器数量位宽是一致的，显而易见，reg 类型的变量可以存储我们需要的数据，他的值也可以随着寄存器的时钟进行改变，寄存器中的值可以保留，改变等，verilog 语法中的寄存器值的变化必须在 always 语句中进行，这里有两种用法，分为组合逻辑电路和时序逻辑电路；定义了寄存器 q，生成的电路为时序逻辑，右图为其结构，为 D 触发器。




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

也可以生成组合逻辑，如数据选择器，敏感信号没有时钟，定义了 reg Mux，最终生成电路为组合逻辑。



```
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
```

存储器 memory 类型：

我们可以使用 reg [15:0] a [1023:0];来定义一个数据块，数据有 1024 个，位宽是 16bit。他可以被例化在 RAM 中，占用 RAM 资源，在 ASIC 设计中可以被例化成 ROM，FLASH 等之类的存储器。

3，运算符：

运算符包括：算术运算符、逻辑运算符、按位运算符、归约运算符、关系运算符、移位运算符、等式运算符、条件运算符以及拼接运算符。

算数运算符：

常用的算术运算符包括：+ 加 、- 减、* 乘、/ 除、% 求模（求余）

以上算术运算符嗾使双目运算符。

逻辑运算符：

&& 逻辑与、|| 逻辑或、 ！ 逻辑非

按位运算符：

~ 按位取反、& 按位与、| 按位或、^ 按位异或、^~,~^ 按位同或

归约运算符：

缩位运算符是单目运算符，包括以下几种：

& 与、~& 与或、| 或、~| 或非、^ 异或、^~,~^ 同或

例如：

reg [3:0] a;

b = & a; // 等效于 b = (((a[0] & a[1]) & a[2]) & a[3])

关系运算符：

< 小于、<= 小于或等于、> 大于、>= 大于或等于

移位运算符：<<,左移。>>，右移
等式运算符：== 等于，!= 不等于，=== 全等，!== 不全等

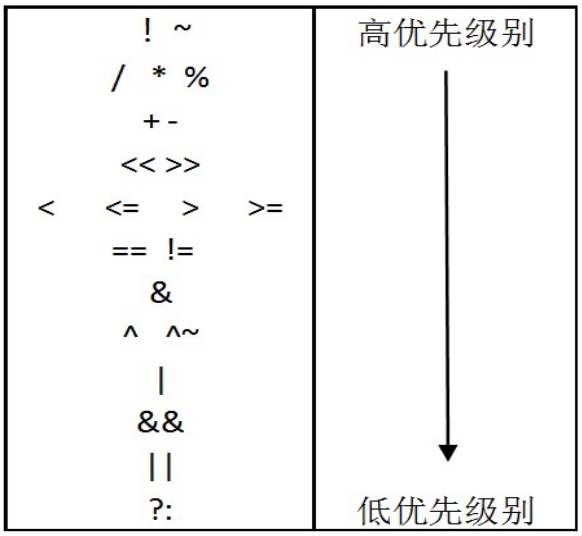
相等运算符（==）：参与比较的两个操作数必须逐位相等，其相等比较的结果才为 1，如果某些位是不定态或高阻态，其相等比较得到的结果是不定值；

全等运算符（===）：在对不定态或高阻态的位也进行比较，两个操作数必须完全一致，其结果才是 1。

条件运算符 ?:
拼接运算符：{ }

例如：{ 3{a,b} } 等同于{ {a,b} , {a,b} , {a,b} }，也等同于{ a, b, a, b, a, b }.

各种运算符的优先级别如下：

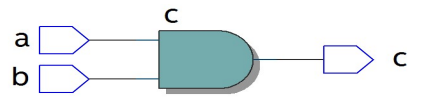


4，组合逻辑

组合逻辑电路的特点是任意时刻的输出仅仅取决于输入信号，输入信号变化，输出立即变化，不依赖于时钟。

与门：

在 verilog 中以“&”表示按位与，如 c=a&b，真值表如下，在 a 和 b 都等于 1 时结果才为 1，RTL 表示如图



输入		输出
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

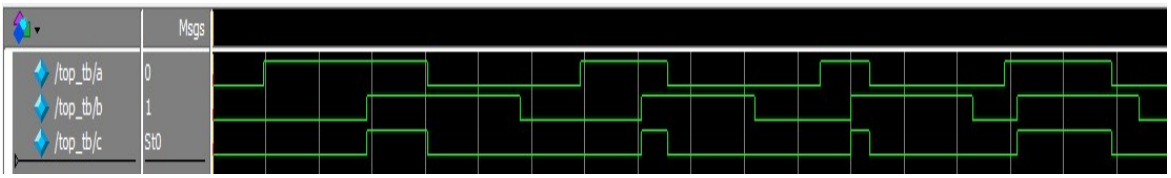
代码如下：

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

激励如下：

```
`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
```

仿真结果如下：



如果 a 和 b 的位宽大于 1，例如定义 input [3:0] a, input [3:0]b，那么 a&b 则 a 与 b 的对应位相与。如 a[0]&b[0],a[1]&b[1]。

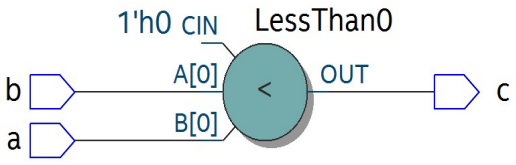
其他的组合逻辑类似流程可以学习。

比较器：

在 verilog 中以大于“>”，等于”==”，小于”<”，大于等于”>=”，小于等于”<=”，不等于”!=”表示，以大于举例，如 c= a > b ;表示如果 a 大于 b，那么 c 的值就为 1，否则为 0。

真值表如下：

输入		输出
a	b	c
0	0	0
0	1	0
1	0	1
1	1	0



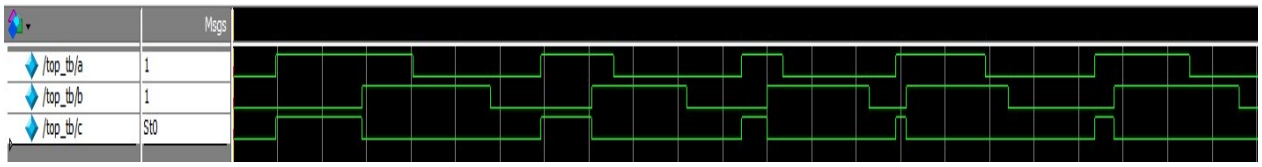
代码如下：

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

激励文件如下：

```
`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
```

仿真结果如下：



5 时序逻辑

组合逻辑电路在逻辑功能上特点是任意时刻的输出仅仅取决于当前时刻的输入，与电路原来的状态无关。而时序逻辑在逻辑功能上的特点是任意时刻的输出不仅仅取决于当前的输入信号，而且还取决于电路原来的状态。下面以典型的时序逻辑分析。

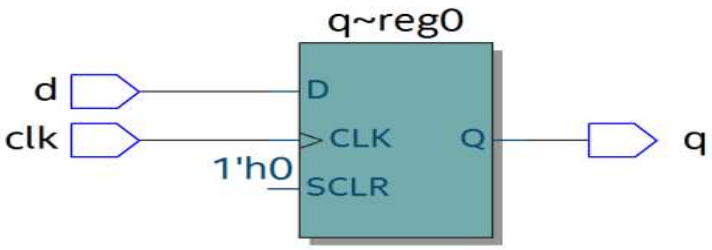
D 触发器：

D 触发器在时钟的上升沿或下降沿存储数据，输出与时钟跳变之前输入信号的状态相同。

Verilog 描述 D 触发器：

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

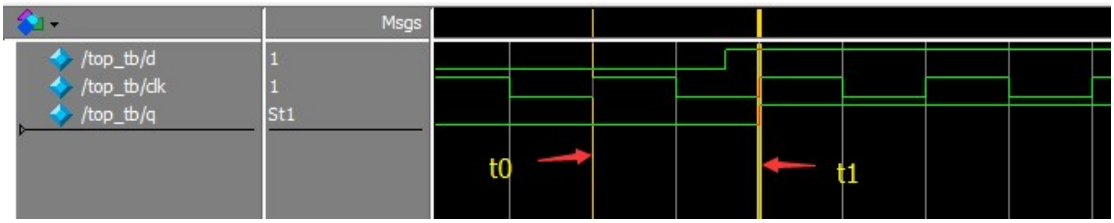
RTL 综合结果图示：



仿真激励文件如下：

```
`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
```

仿真结果如下，可以看到在 t0 时刻时，d 的值为 0，则 q 的值也为 0；在 t1 时刻 d 发生了变化，值为 1，那么 q 相应也发生了变化，值变为 1。可以看到在 t0-t1 之间的一个时钟周期内，无论输入信号 d 的值如何变化，q 的值是保持不变的，也就是有存储的功能，保存的值为在时钟的跳变沿时 d 的值。



真双口 RAM：

双口 RAM 分为真双口 RAM 和伪双口 RAM，伪双口 RAM 指的读写地址是独立的，可以随机选择写或读地址，同时进行读写操作。真双口 RAM 有两套控制线，数据线，允许两个系统对其进行读写操作。这里介绍真双口 RAM

Verilog 描述真双口 RAM：

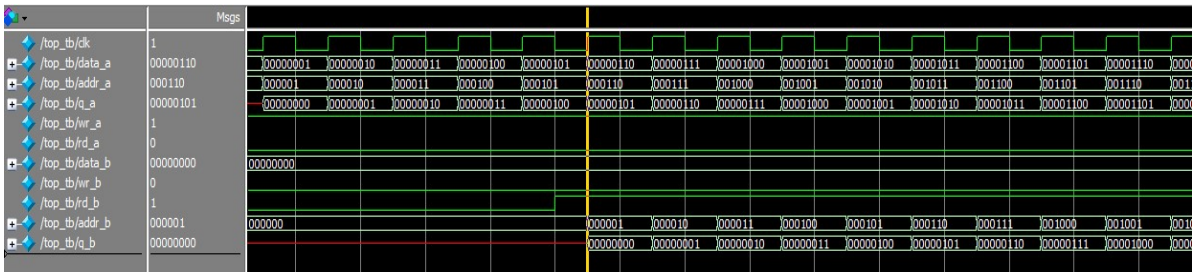
```
module top
(
input [7:0] data_a, data_b,
input [5:0] addr_a, addr_b,
input wr_a, wr_b,
input rd_a, rd_b,
input clk,
output reg [7:0] q_a, q_b
);
reg [7:0] ram[63:0]; //declare ram
//Port A
always @ (posedge clk)
begin
if (wr_a) //write
begin
ram[addr_a] <= data_a;
q_a <= data_a ;
end
if (rd_a)
//read
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 描述添加激励文档进行仿真：

```
`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
```

仿真结果：



有限状态机：

有限状态机（Finite-State Machine，FSM），又成为有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。笔者常在电机控制、通信协议解析等应用场景下应用 FSM。 本文所讲的是基于硬件描述语言 Verilog HDL 的有限状态机的编写技巧及规范。众所周知 FPGA 以其并行性和可重构性为世人所知，而在当今的电子世界，基本所有的器件都是串行的，所以作为控制单元或者是可编程单元的 FPGA 需要进行并行转串行与外界进行通信、控制等，而有限状态机以其简单实用、结构清晰而恰如其分的充当着这个角色。

有限状态机是由寄存器组和组合逻辑构成的硬件时序电路，其状态（即由寄存器组的 1 和 0 的组合状态所构成的有限个状态）只可能在同一时钟跳变沿的情况下才能从一个状态转向另一个状态，究竟转向哪一状态还是留在原状态不但取决于各个输入值，还取决于当前所在状态。

状态机有两大类：Mealy 型和 Moore 型。Moore 型状态机的输出只与当前状态有关，而 Mealy 型状态机的输出不仅取决于当前状态，还受到输入的直接控制，并且可能与状态无关，其状态机结构请看下图：

图 2-50 Mealy 状态机图示

状态机的本质是对具有逻辑顺序或时序规律事件的一种描述方法。这个论断的最重要的两个词就是“逻辑顺序”和“时序逻辑”，这两点就是状态机所要描述的核心和强项，换言之，所有具有逻辑顺序和时序规律的事情都适合用状态机描述。

状态机的两种应用思路。

第一种思路，从状态变量入手。如果一个电路具有时序逻辑或者逻辑顺序，我们就可以自然而然地规划出状态，从这些状态入手，分析每个状态的输入，状态转移和输出，从而完成电路功能；

第二种思路，需要首先明确电路的输出关系，这些输出相当于状态的输出，回溯规划每个状态，和状态转移条件与状态输入。两种思路殊途同归，都要通过使用状态机的目的来达到控制某部分电路的目的，完成某种具有逻辑顺序或时序规律的电路设计。

状态机的参数定义采用的都是独热码，和格雷码相比，虽然独热码多用了触发器，但所用组合电路可以省一些，因而使电路的速度和可靠性有显著提高，而总的单元数并无显著增加。采用独热编码后有了多余的状态，就有一些不可达到的状态。为此在 case 语句的最后需要增加 default 分支向。这可以用默认项表示该项，也可以用确定项表示，以确保回到初始状态。一般综合器都可以通过综合指令的控制来合理地处理默认项。

自动售卖机的功能描述如下：

饮料单价 2 元，该售卖机只能接受 0.5 元、1 元的硬币。考虑找零和出货。投币和出货过程都是一次一次的进行，不会出现一次性投入多币或一次性出货多瓶饮料的现象。每一轮售卖机接受投币、出货、找零完成后，才能进入到新的自动售卖状态。

该售卖机的工作状态转移图如下所示，包含了输入、输出信号状态。

其中，coin = 1 代表投入了 0.5 元硬币，coin = 2 代表投入了 1 元硬币。

状态机设计：3 段式（推荐）

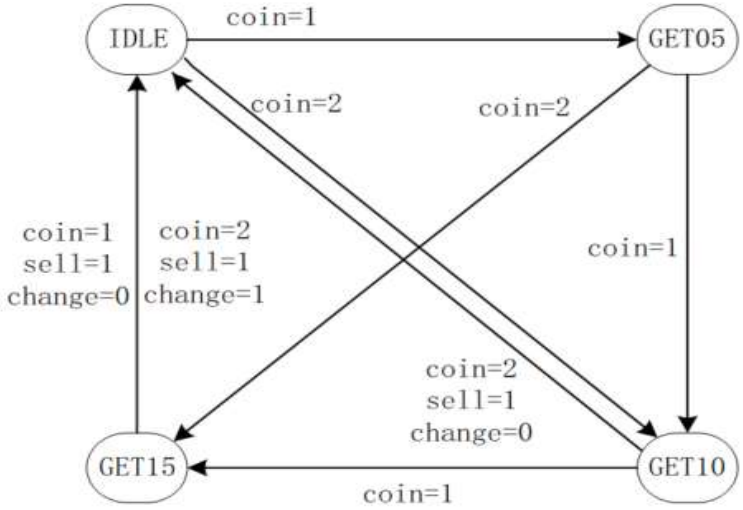
状态机设计如下：

(0) 首先，根据状态机的个数确定状态机编码。利用编码给状态寄存器赋值，代码可读性更好。

(1) 状态机第一段，时序逻辑，非阻塞赋值，传递寄存器的状态。

(2) 状态机第二段，组合逻辑，阻塞赋值，根据当前状态和当前输入，确定下一个状态机的状态。

(3) 状态机第三代，时序逻辑，非阻塞赋值，因为是 Mealy 型状态机，根据当前状态和当前输入，确定输出信号。



Verilog 的 RTL 描述如下：

```
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
```

Testbench 设计如下。仿真中模拟了 4 种情景，分别是：

case1 对应连续输入 4 个 5 角硬币；case2 对应 1 元 - 5 角 - 1 元的投币顺序；case3 对应 5 角 - 1 元 - 5 角的投币顺序；case4 对应连续 3 个 5 角然后一个 1 元的投币顺序。

```
`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 ;
```

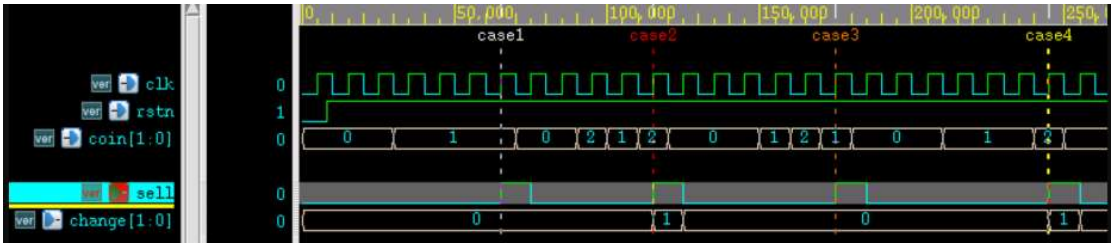
```
        end
        //case(3) 0.5 -> 1 -> 0.5
        #16 ;
        buy_oper    = 10'b00_0001_1001 ;
        repeat(5) begin
            @(negedge clk) ;
            coin          = buy_oper[1:0] ;
            buy_oper    = buy_oper >> 2 ;
        end
        //case(4) 0.5 -> 0.5 -> 0.5 -> 1, taking 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
```

仿真结果如下:

由图可知，代表出货动作的信号 sell 都能在投币完毕后正常的拉高，而代表找零动作的信号 change 也都能根据输入的硬币场景输出正确的是否找零信号。



6 总结

本文档介绍了组合逻辑以及时序逻辑中常用的模块，其中有限状态机较为复杂，但经常用到，希望大家能够深入理解，在代码中多运用，多思考，有利于快速提升水平。

2.3.2 Verilog 语法学习参考文献

学习 verilog 的语法有许多可以参考的资料，但是在学习 verilog 语法之前需要完成对数字逻辑电路的学习，以及对模拟电路课程关于半导体原理的学习。

关于进一步的组合逻辑电路中的竞争与冒险，以及时序电路中的延时，输入输出延时，关于模块例化的整合，参数的传递，等等，这里限于篇幅不在介绍，可以参考的 verilog 的学习资料如下：

[1]小梅哥 FPGA 教程

[2]威三学院 FPGA 教程

[3]吴厚航. 深入浅出玩转 FPGA[M]. 北京航空航天大学出版社, 2013.

[4]夏宇闻. Verilog 数字系统设计教程.第 3 版[M]. 北京航空航天大学出版社, 2013.

[5]韩彬, 于潇宇, 张雷鸣. FPGA 设计技巧与案例开发详解[M]. 电子工业出版社, 2014.

关于时序方面的资料可以使用 xilinx 的官方关于时序分析以及约束的使用。

2.3.3 文档查找器 DocNav 的使用

Xilinx 的文档查找器 DocNav 能够找到需要的参考资料，包括数据手册、使用指导、产品手册、设计流程、参考设计以及使用中一些典型的问题的解决方法总结文档，介绍视频等等。

DocNav 的安装：

-可单独安装

-Vivado 安装文件中已包含该软件，只需勾选即可

DocNav 概览：

打开该软件，在左上角会显示 Catalog View 和 Design Hub View。Catalog View 按芯片系列、开发工具、IP 等将资料分门别类。Design Hub View 则是从 FPGA 设计视角将文档分类的。

怎么用好这个工具的？这个工具的目的之一就是帮助我们查找 Xilinx 的相关文档。进一步说就是“用更少的时间找文档，省出更多的时间读文档”。

如果你是初学者，刚接触 Xilinx 不久，可能只知道 FPGA 设计的基本流程或者简单术语，那你可以从 Design Hub Veiw 开始找你需要的资料。如果打开 DocNav，没有显示 Design Hub View，可以点击下图红色方框按钮（在右上角的位置）。

选择下图中的 System-Level Design Flow，就会出现一个清晰的流程图，这个流程图的奇妙之处在于图中的蓝色文字是个超链接，点击它就会发现相关的文档都汇聚在一起了。

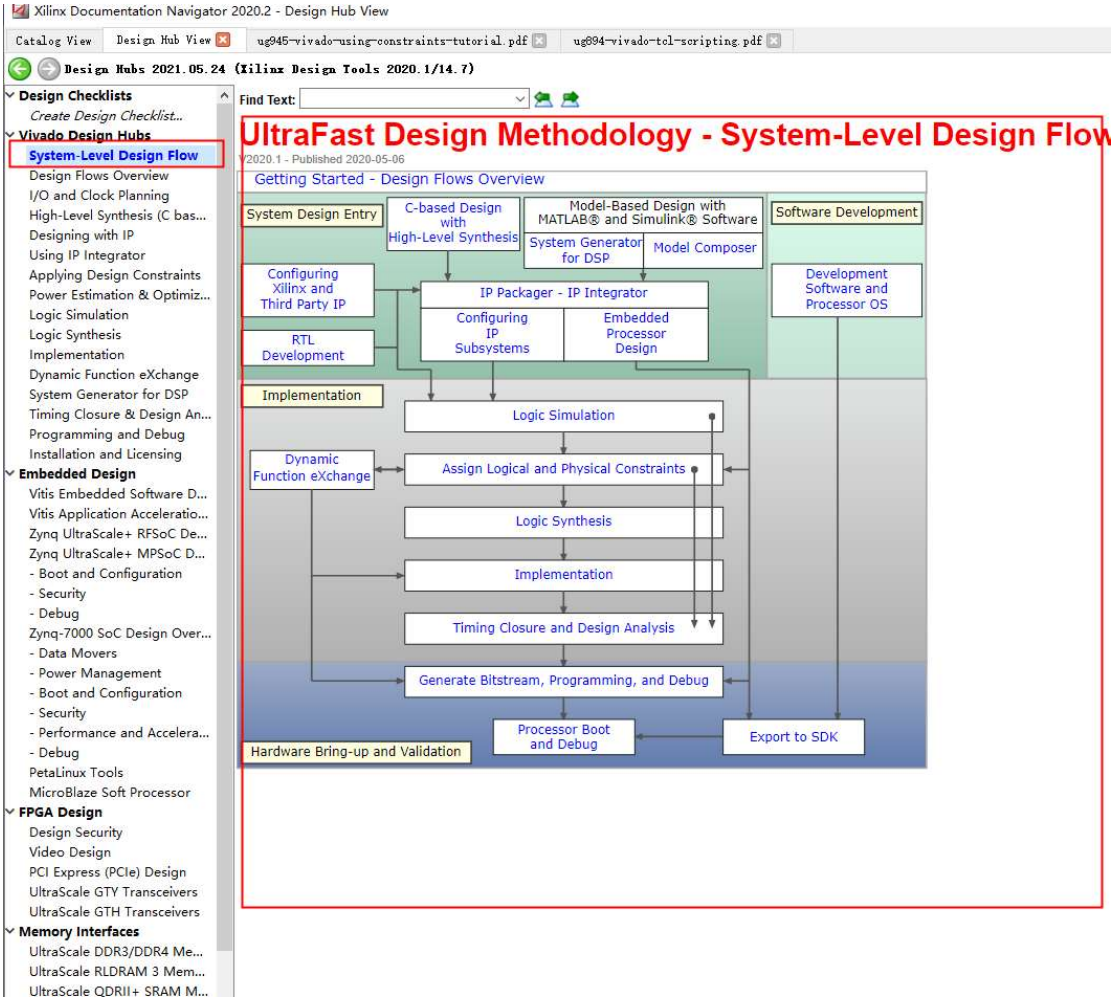


图 2-51 DocNav 启动

比如点击 Logic Simulation，就会出现下面这些文档（这里只显示了一部分）。接下来你就可以按照自己的阅读习惯查看文档了。对于初学者而言，GetTIng Started 是个不错的部分，有视频，有一步一步教你学习的 Tutorial，还有进一步深入学习的 User Guide。对于 User Guide，个人建议可以把它当作字典，遇到问题时可以去查看，这样效率会更高。一页一页去读实在没太大必要。

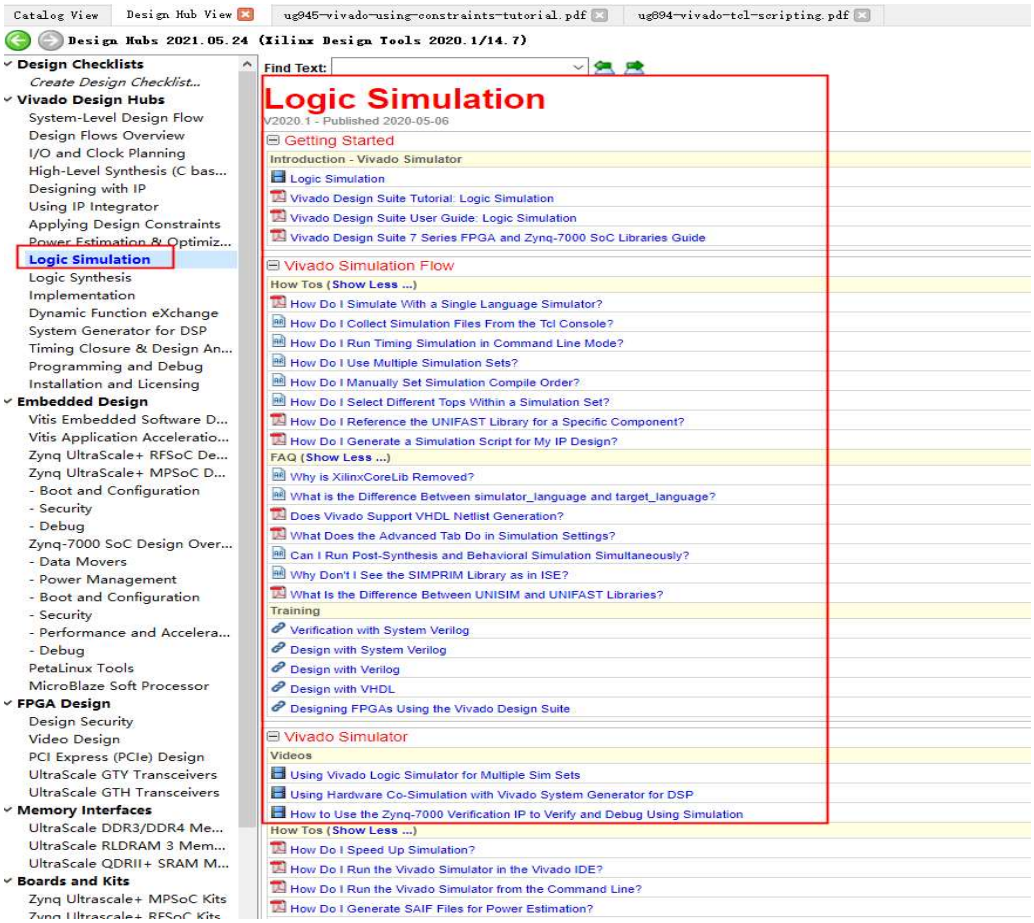


图 2-52DocNav 的 Design Hub

快速查看文档章节标题

在 DocNav 中可以快速查看文档章节标题而无需下载该文档。如下图所示，点击红色箭头所指的标记，即可显示文档章节标题。若要查看某章节内容，可直接点击标题名称，从而打开该文档并到相应章节处。

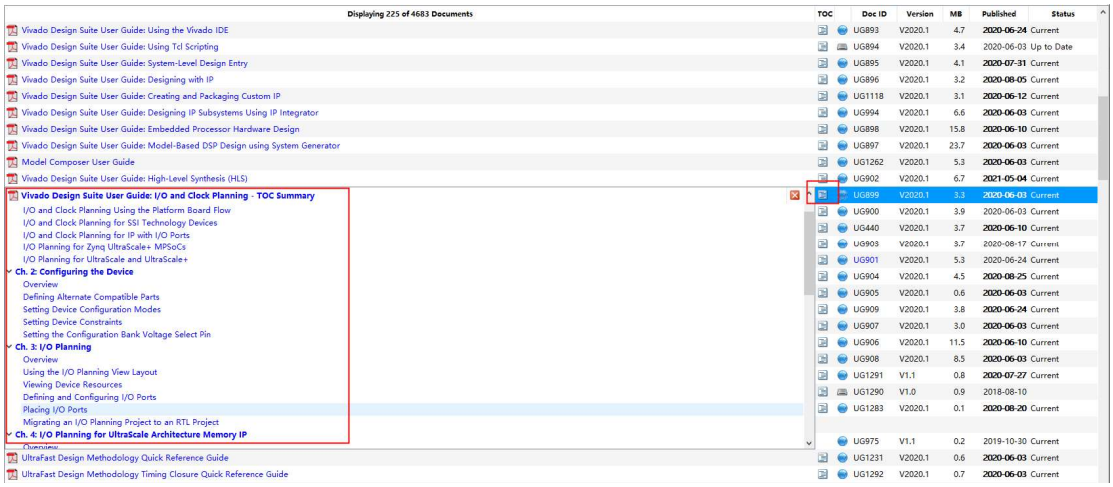


图 2-53DocNav 的文档目录导航

查找文档

如果知道文档的编号(Doc ID)或文档标题中的关键字，例如 ug949，可通过如下三步快

速查找到该文档。

图 2-54DocNav 的查找

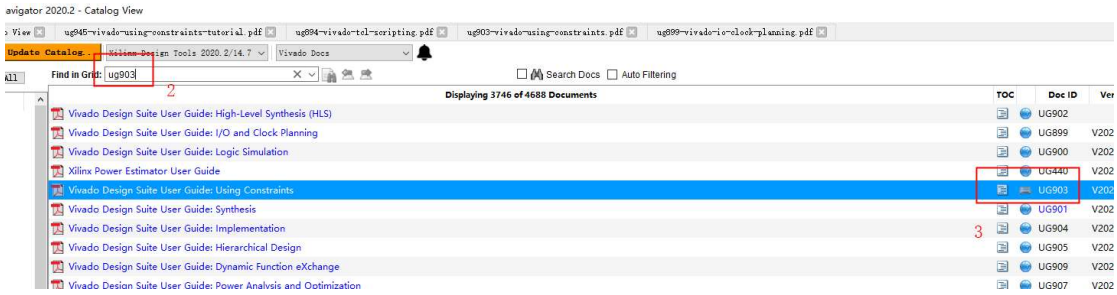


图 2-55DocNav 的查找结果

如果不知道文档编号，可根据关键字查找文档，例如需要查找时序约束相关的文档，可通过如下三步快速找到该文档。对于搜索到的文档，可进一步过滤，例如，可根据匹配情形过滤。

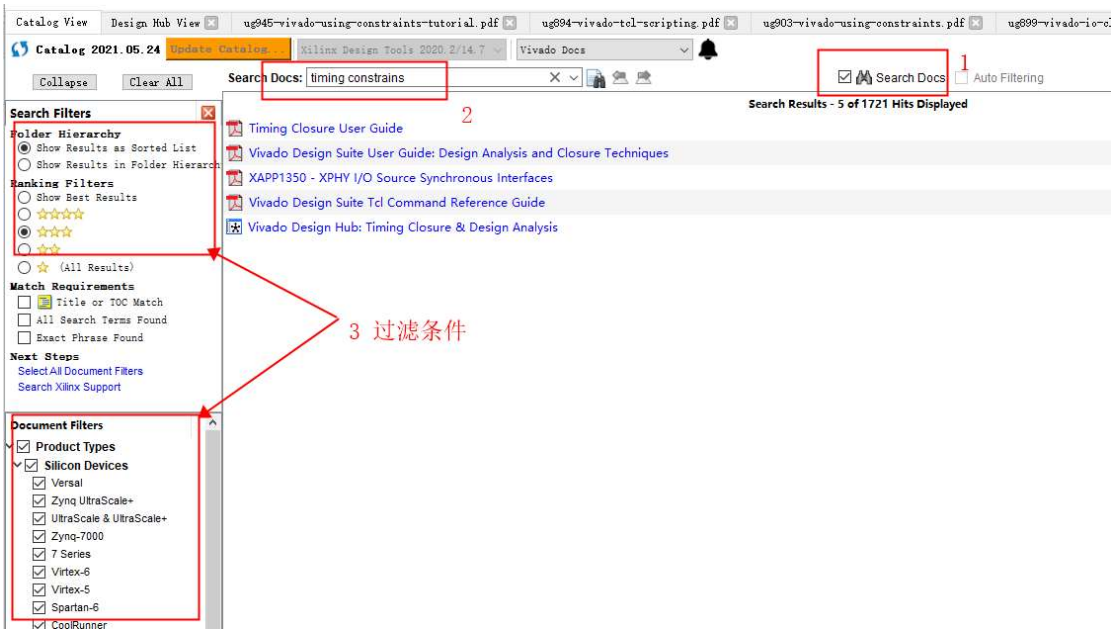


图 2-56DocNav 的查找过滤

文档过滤器

Document Filters 在 DocNav 的最左侧，如下图所示。例如，要查看 Vivado 相应的文档，就可以勾选 Vivado，而撤销其他（去掉方框中的“√”即可）。这有利于缩小查找范围。

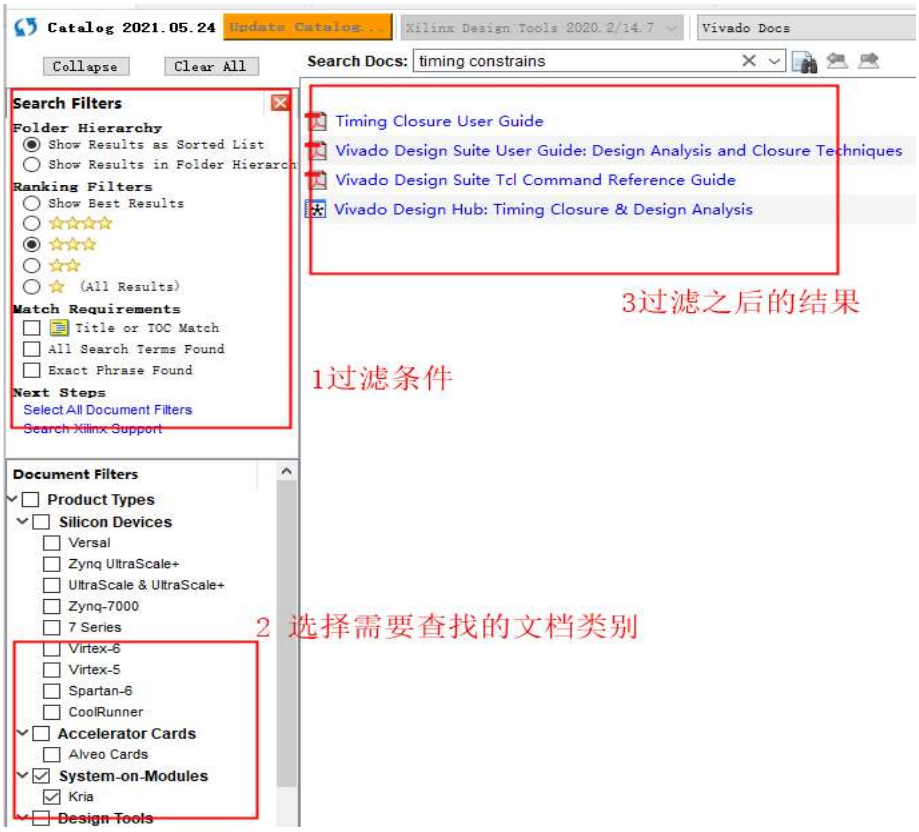


图 2-57DocNav 的查找过滤结果

Document Tray

DocumentTray 位于 DocNav 的最右侧，可显示最近打开过的文档。这样的好处是如需查找最近看过的文档，可在这里快速找到。

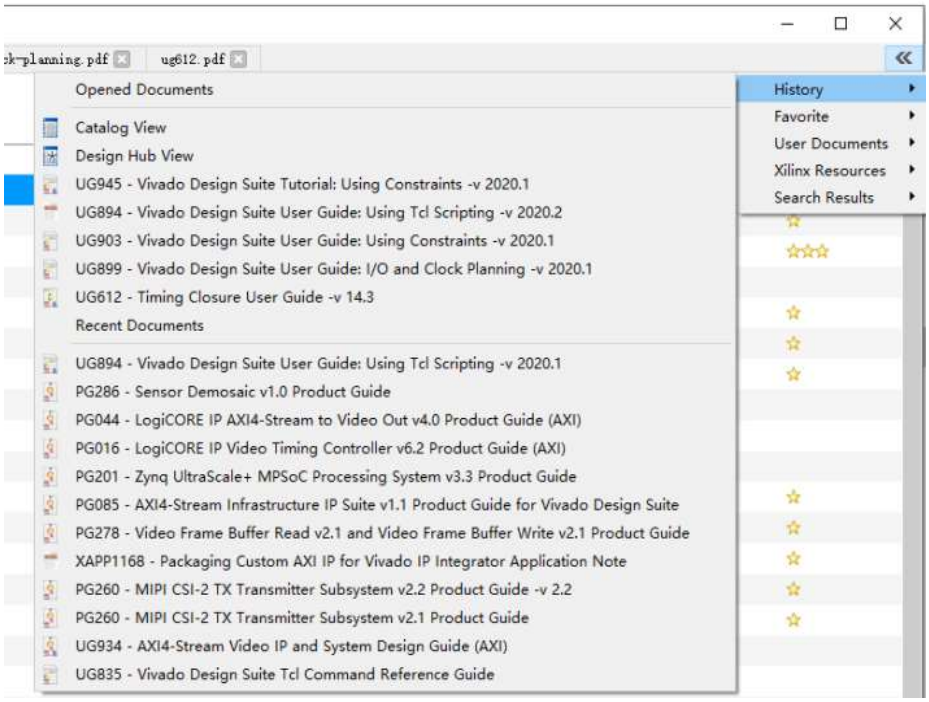


图 2-58DocNav 的历史查找文档

2.3.4 XDC 约束文件语法

XDC 的基础：

XDC 约束文件指的是 xilinx Design constrains，这是 vivado 专用的约束设计文档，包括物理约束和时序约束。不支持传统的 ISE 的 UCF 约束文档格式。它们的区别在于 XDC 是基于标准 Synopsys 设计约束 SDC 格式，SDC 已经设计和使用超过 20 年时间，从而其描述设计约束是最流行和最成熟的方式。XDC 与 UCF 是有根本性质的区别的，而这些基本的区别需要去了解。

从而可以确定，XDC 就是设计约束的标准 Synopsys design constrains (SDC)和 xilinx 专有物理约束的组合。这里的 SDC 是（SDC1.9 版本）。

对于 XDC 文档的特性具有如下：

1，XDC 中写的都是遵循 TCL 语法的命令。

2，XDC 是像 Vivado Tcl 解释器的任何其他命令一样被解释。

3，XDC 的读入和解析顺序和其他的 Tcl 命令一样。

我们可以在设计过程中通过不通的方式不同的点输入 XDC 命令：

1，设计约束可以被分成一个或者多个 XDC 文档进行输入。

为了在内存中添加 XDC 约束，用以下一种方式就可以，

使用 read_xdc 命令

将其添加到项目的约束集之一，XDC 文件只接受 set、list、expr 内置 Tcl 命令。

2，使用 unmanaged Tcl 脚本生成约束

而为了执行 Tcl 脚本，以下操作其一即可

使用 source 命令

使用 read_xdc -unmanaged 命令

添加 Tcl 命令到项目的约束集之一。

使用 XDC 约束的具体作用：

设计约束定义了编译流程必须满足的要求，以便设计在电路板上发挥作用。并非所有约束都被编译流程中的所有步骤使用。例如，物理约束仅在实现步骤（即布局器和路由器）期间使用。 由于 Xilinx® Vivado® 集成设计环境 (IDE) 综合和实现算法是时序驱动的，因此您必须创建适当的时序约束。过度约束或不足约束您的设计会使时序收敛变得困难。您必须使用与您的应用程序要求相对应的合理约束。

XDC 编写方式：

物理约束编写方式：

Xilinx 专用的管脚物理约束包括：管脚位置约束和电气约束。

管脚位置约束： set_property PAKAGE_PIN “管脚编号” [get_ports “端口名称”]

管脚电平约束： set_property IOSTANDARD “电压” [get_ports “端口名称”]

举例：

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] }]

注意：

1）以上语法对大小写敏感；

2）端口名称为数组时，需要用{}括起来，端口名不能为关键字。

差分信号约束

普通差分约束

差分信号约束语法和 1 节中相同。此处仅举例。

1）HR I/O Bank，VCCO = 3.3V，HDMI 接口约束

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 接口约束

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] }]

注意：

1）差分信号约束，只约束 P 管脚即可，系统自动匹配 N 管脚约束，当然_P 和_N 管脚都约束也没有问题；

2）差分信号电平要根据 VCCO Bank 电压进行相应的约束。

收发器差分信号约束

1）收发器 MGTREFCLK 时钟约束管脚位置约束：

set_property LOC “管脚编号” [get_ports “端口名称”]

举例：

set_property LOC G7 [get_ports Q2_CLK0_GTREFCLK_PAD_N_IN]

set_property LOC G8 [get_ports Q2_CLK0_GTREFCLK_PAD_P_IN]

2）收发器 MGT 通道约束

对于 GTXE2_CHANNEL 通道约束：一种方法是可以利用 7 系列 FPGAs 收发器向导，在配置好收发器配置参数后，自动生成 XDC 模板，然后将该模板应用到自己的设计中；第二种方法是自己编写 XDC 约束文件，其位约束位置要参照具体原理图信号管脚来进行编写约束文件。举例：对于图 1 中四通道收发器对 GTXE2_CHANNEL 约束。

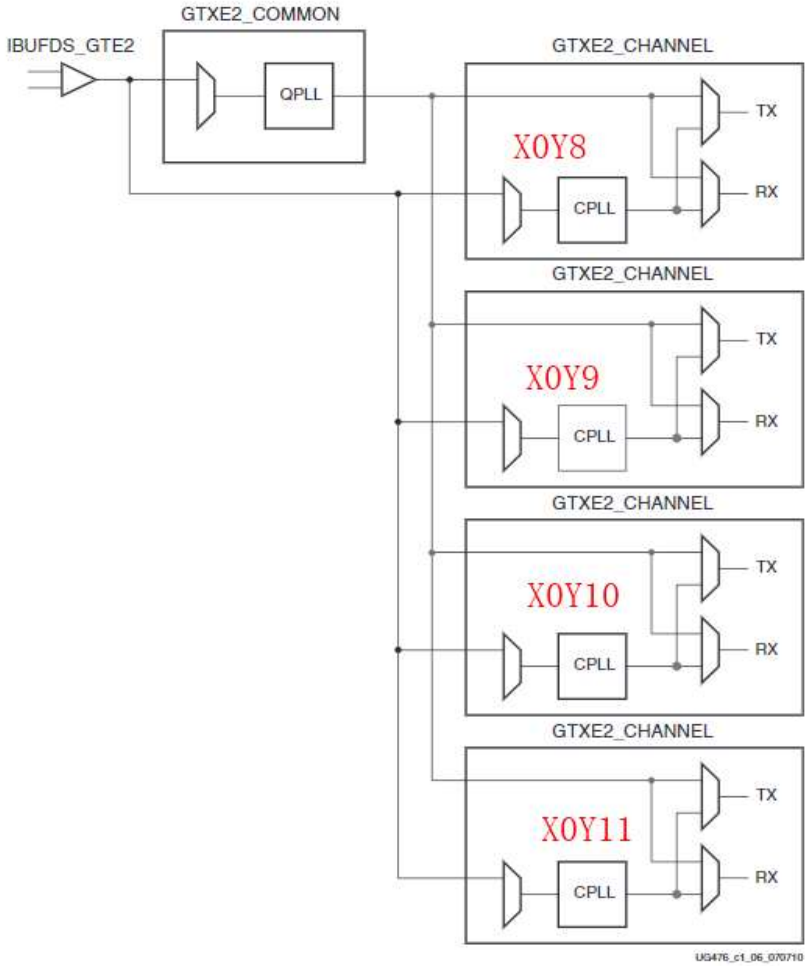


Figure 1-5: **Four Channel Configuration (Reference Clock from the QPLL of GTXE2_COMMON)**

图 2-59 GTXE2_ChANNEL

收发器通道位置约束：

set_property LOC “ GTXE2_CHANNEL_X* Y * ” [get_cells “gtxe_2 例化路径”]

举例：

```
##---------- 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]
```

图 2-60 高速串行收发器的引脚约束

注意：gtxe_2 例化路径参照图 3 所示，路径名称依据具体工程实现进行修改。

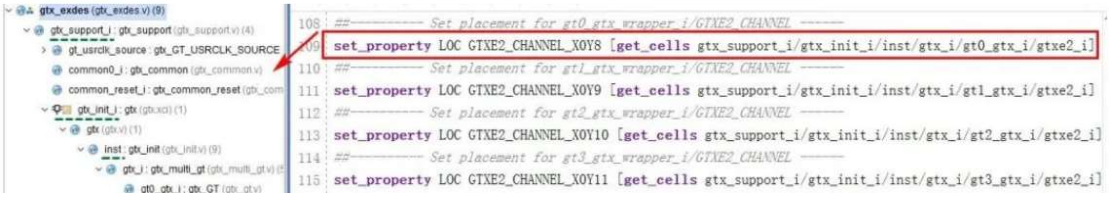


图 2-61 高速串行收发器的引脚约束

时序约束：

时序约束包括一下几个分类：

- 输入路径（Input Path），使用输入约束
- 寄存器到寄存器路径（Register-to-Register Path），使用周期约束
- 输出路径（Output Path），使用输出约束
- 具体的异常路径（Path specific exceptions），使用虚假路径、多周期路径约束

输入延时约束 Input Constraint

Figure: Example of an input delay constraint on a port

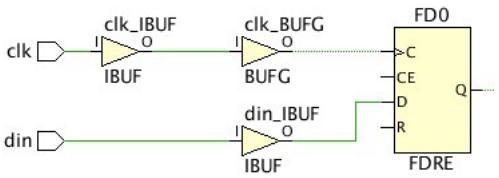


图 2-62 输入延时示例

输入信号 din 在时钟 clk 的上升沿被寄存器 FD0 捕获。输入延迟约束描述了时钟启动沿和信号 din 在设备输入端转换的时间之间的延迟。必须指定输入延迟最大值和最小值以分别准确地执行设置/恢复和保持/移除检查。选择接口类型（系统或源同步、边沿对齐、参考边沿和数据速率）后，您必须定义所需的各种延迟参数，以计算最小和最大输入延迟值。例如：

```
set_input_delay -clock clk 3.4 [get_ports din] -max
set_input_delay -clock clk 1.0 [get_ports din] -min -add
```

寄存器到寄存器约束 Register-to-Register Constraint

寄存器到寄存器约束往往指的是周期约束，周期约束的覆盖范围包括：

- 覆盖了时钟域的时序要求
- 覆盖了同步数据在内部寄存器之间的传输
- 分析一个单独的时钟域内的路径
- 分析相关时钟域间的所有路径
- 考虑不同时钟域间的所有频率、相位、不确定性差异

输出约束 Output Constraint

输出时序约束约束的是从内部同步元件或寄存器到器件管脚的数据。

系统同步输出约束 System Synchronous Output Constraint

系统同步输出的简化模型如图所示，在系统同步输出接口中，传输和获取数据是基于同一个时钟的。

输出延时约束

Figure: Example of an output delay constraint on a port

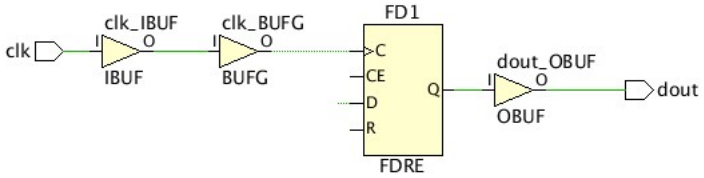


图 2-63 输出延时示例

输出信号 dout 由寄存器 FD1 在 clk 的上升沿发出，并由板时钟在器件外部捕获。在大多数情况下，电路板时钟也是 clk 或者是 clk 的相移副本。输出延迟约束描述了器件边界处的 timedout 转换与板时钟的捕获沿之间的延迟。必须指定输出延迟最大值和最小值以准确执行建立和保持检查。选择接口类型（系统或源同步、边沿对齐、参考边沿和数据速率）后，您必须提供向导所需的各种延迟参数来计算最小和最大输出延迟值。例如：

```
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 语法简要介绍

Tcl 全称是 Tool command Language。它是一个基于字符串的命令语言，基础结构和语法非常简单，易于学习和掌握。

Tcl 语言是一个解释性语言，所谓解释性是指不象其他高级语言需要通过编译和联结，它象其他 shell 语言一样，直接对每条语句顺次解释执行。

Tcl 数据类型简单。对 Tcl 来说，它要处理的数据只有一种——字符串。Tcl 将变量值以字符串的形式进行存储，不关心它的实际使用类型。

基本语法：

1.输出：tcl 使用”puts"关键字来作为输出语句

[语法] : puts -nonewline -channelId -string

Tcl 的输出命令是“puts”，将字符串输出到标准输出 channelId。语法中两个问号之间的参数为可选参数。

puts hello

结果=> hello

puts –nonewline "hello hello"

结果=>hello hello

但如果输出一段有空格的文本，则要用双引号或者花括号括起来

-nonewline 选项告诉 puts 不输出回车换行。

注意：双引号和花括号的作用是将多个词组织成一个变元，但他们是有差别的！这种差别就是在处理“替换操作”时，前者允许替换发生，而后者则可能会阻止替换。关于两者用法与差别以后会陆续讲到。在这里两者作用相同。

2.赋值： tcl 使用“set”关键字来定义参数，不必指定变量值的类型，因为变量值的类型仅一种——字符串。为变量赋值时，会为变量开辟一段内存空间来存储变量值。

[语法] set varName [value]

set a Hello ;#定义变量 a 并赋值 =>Hello

puts $a ;#输出变量值 =>Hello

set a “Hello world” ;#重新赋值 =>Hello world

set a “Hello world” => Test Tcl ;#输出变量值，此时不加”$”

puts $a =>Hello world ;#输出变量值，此时要加”$”

puts a => a ;#输出字符”a”

set b $a =>Hello world

puts $b =>Hello world ;#将 a 的值赋给 b

3.替换

(1):$

“$”符实现引用替换，用以引用参数值。上面也用到过

Tcl 对替换只进行一遍解释，对嵌套的”$”不于理睬。

set foo oo =>oo

set dollar foo =>foo

set x $$dollar =>$foo ;#只解释一次，将“$dollar”用 dollar 的值（foo）代替， ;#命令等效为 set x {$foo}，大括阻止替换。

```
set x {$foo}      =>$foo
set y $x =>$foo                    ; #一轮替换
```

(2) :[]

方括号“[]”完成命令替换。用“[]”将一条命令括起来，命令执行完成后，返回结果。

```
set b [set a      5]                          ;#set a 5  命令输出的结果赋给  b =>5
puts $b =>5
set c [expr 5 * 10]                           ;#将乘式结果赋给  c =>50
```

(3): "" 和{}

双引号和花括号将多个单词组织成一个参数，也是一种替换操作。””和{}内的替换如何进行

呢？一般的原则是在""内的替换正常进行，而在{}内的替换有可能会被阻止。

```
set a 123
=>123
puts "$a"              #会替换=>123
puts {$a}               #不会替换=>$a
```

总之：

TCL 语言的执行顺序是：先分组，再替换，最后执行

花括号中不准替换

双引号和花括号的作用都是分组，但是不同之处在于是否支持替换语句的运行分 3 步走：

1、首先分组

2、其次替换

3、最后运行

$的作用是变量引导符，在字符串中如果要替换某个变量，可能还需要用{}来界定变量的起始和终点。

分组有三种方法：空格，双引号和花括号。

另外转义符\在这里的作用是提升或者消除字符的能力。

2.4 本章小节

本章节主要介绍了一些基本的知识，包括基本的工具（vivado 和 modelsim）安装和使用的注意事项，讲明白了使用中会遇到的一些常见的问题，需要注意的地方都特别进行了说明。深入一点的知识就是约束那一块，如果对于数字逻辑电路不是很清楚的，需要先学习数字逻辑电路，里面有一些基本的建立时间、保持时间等之类的时序相关的概念。物理

约束很好理解，就是输出输入引脚的分配和电气特性与芯片数据手册的一致性，方便实现过程，后面的布局布线就是一个更加深入的问题，感兴趣的朋友可以自己尝试着探索。

第三章 硬件平台详细配置

ZYNQ 的主要优势就是 FPGA 和 ARM 的合理结合，这对开发人员提出了更高的要求。从本章开始，我们需要 FPGA 工程师和软件工程师分工协同合作。

FPGA 工程师负责把 Vivado 工程搭建好，提供好硬件给软件开发人员，软件开发人员便能在这个基础上开发应用程序。做好分工，也有利于项目的推进。如果是软件开发人员想把所有的事情都做了，可能需要花费很多时间和精力去学习 FPGA 的知识，由软件思维转成硬件思维是个比较痛苦的过程，如果纯粹的学习，又有时间，就另当别论了。专业的人做专业的事，是个很好的选择。

3.1 第一个工程建立

3.1.1 vivado 工程建立

打开 vivado，点击 Create Project 新建工程

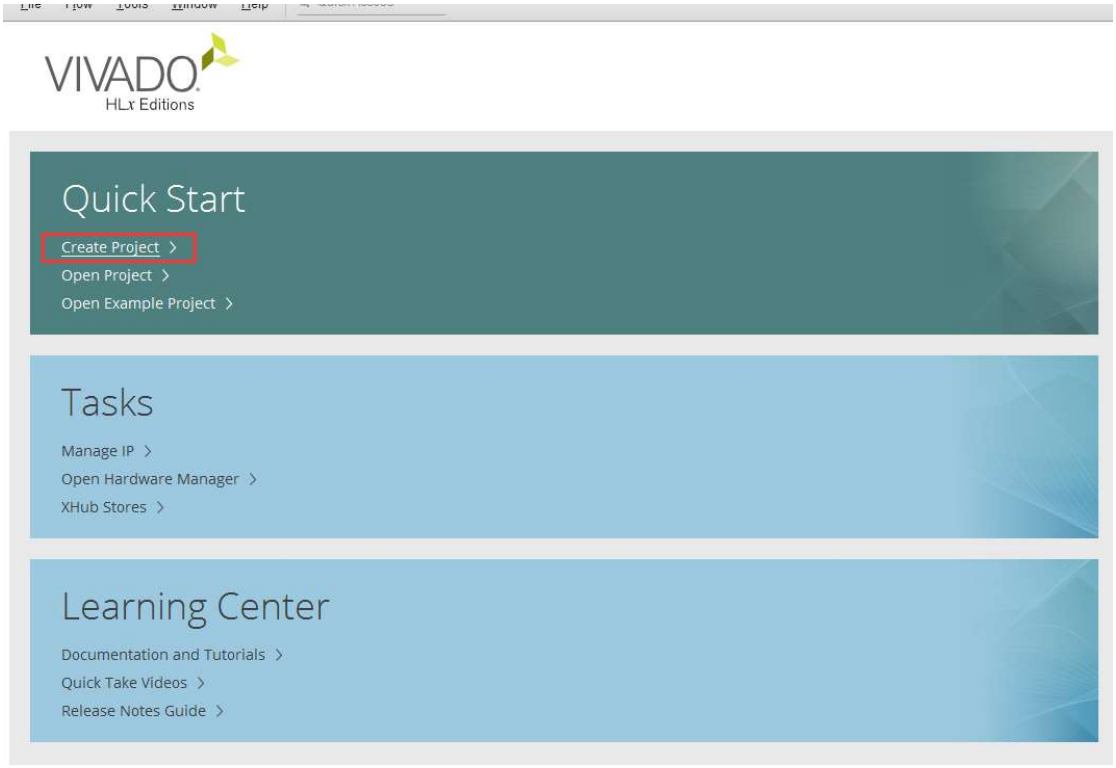


图 3-1 vivado 启动界面

点击 Next

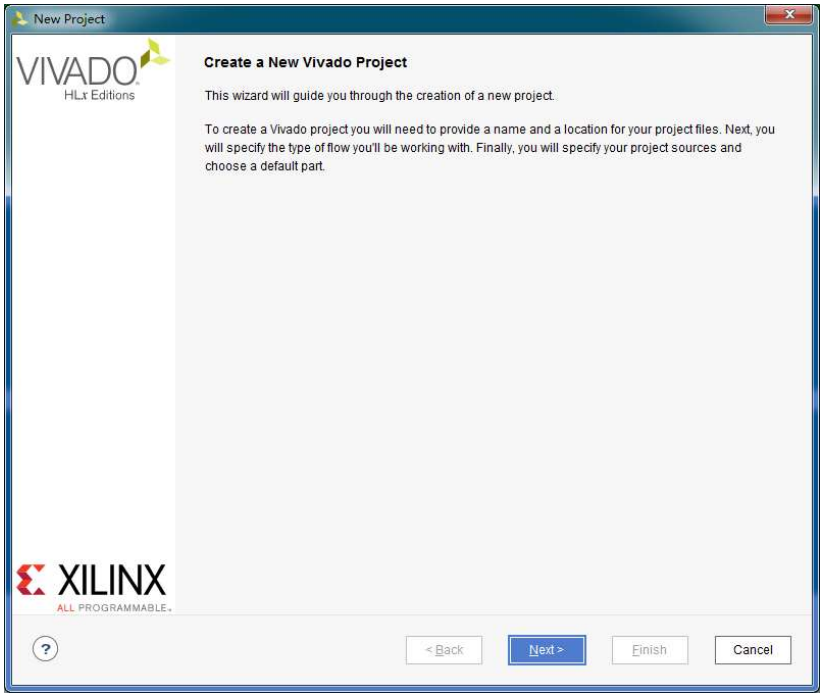


图 3-2 vivado 创建工程界面

输入工程名和保存路径

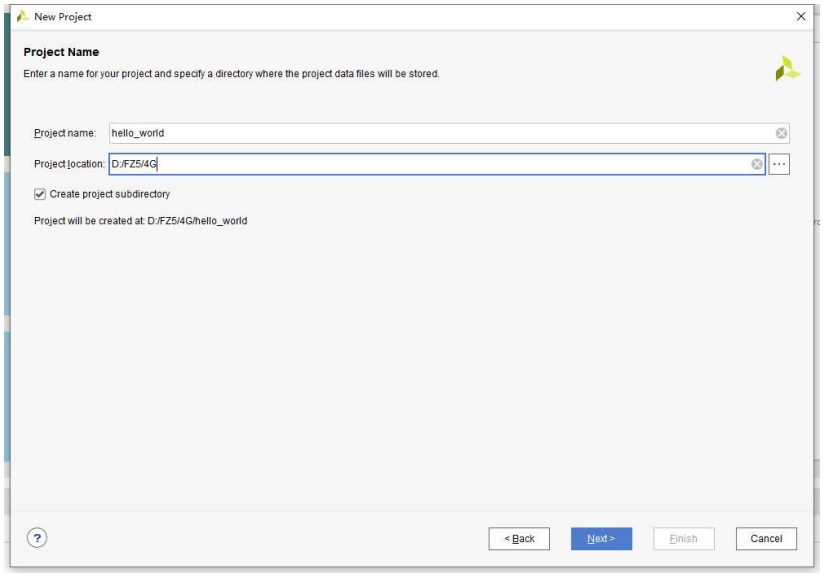


图 3-3 vivado 创建工程-添加工程名称和路径

点击 Next

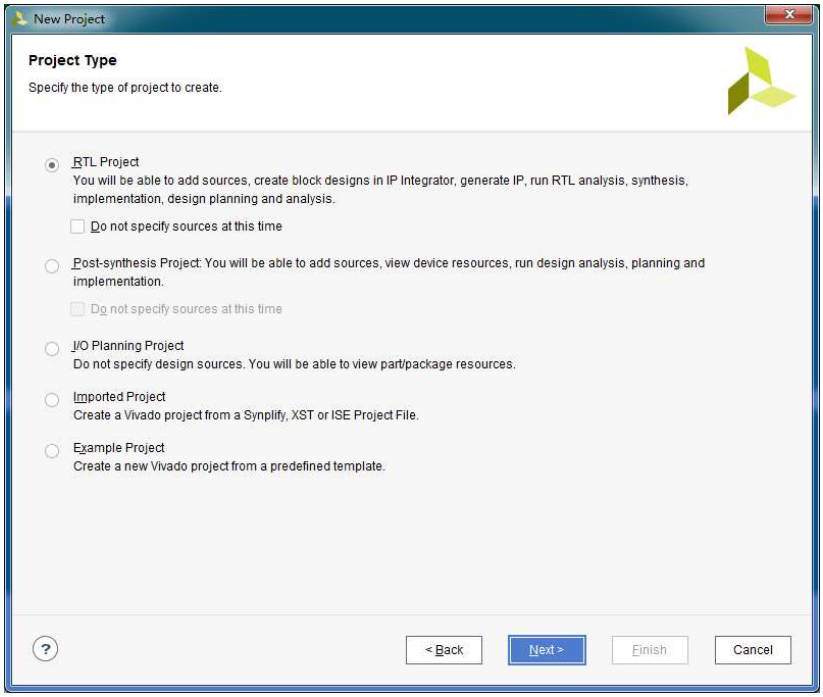


图 3-4 vivado 创建工程-使用 RTL 开发方式

这里 Add Sources 跟 Add Constraints 都不需要添加文件，直接点击 Next 跳过。

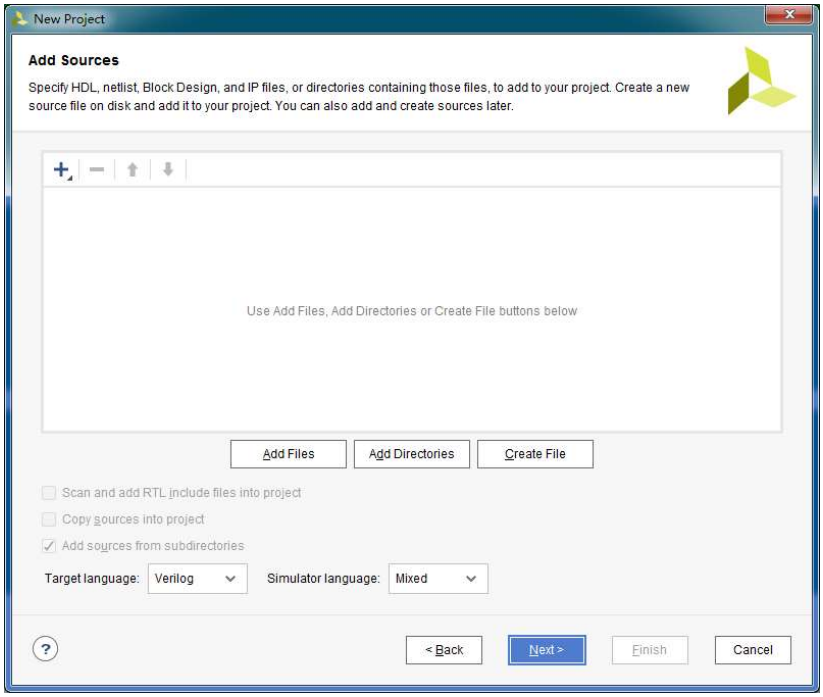


图 3-5 vivado 创建工程-添加已经存在的文档或者跳过

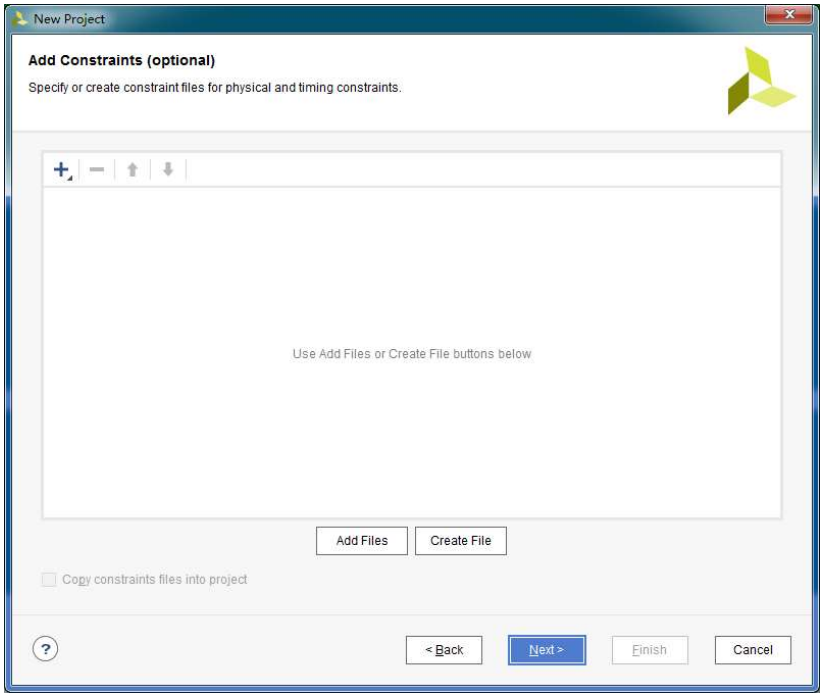


图 3-6 vivado 创建工程-添加约束文件

在 Default Part 界面，选择芯片型号为 xczu5ev-sfvc784-1-i ，然后点击 Next

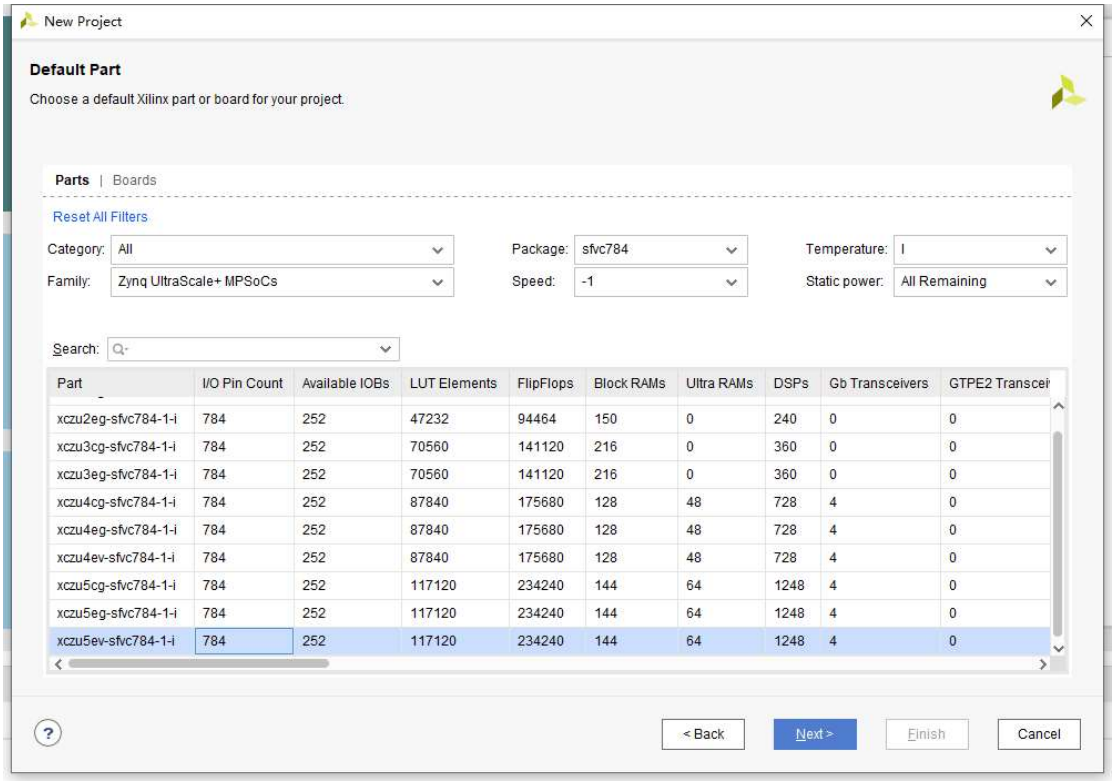


图 3-7 vivado 创建工程-选择器件

在这个 Summary 界面，可以检查创建工程时的各个设置选项，确认无误后，点击 Finish。

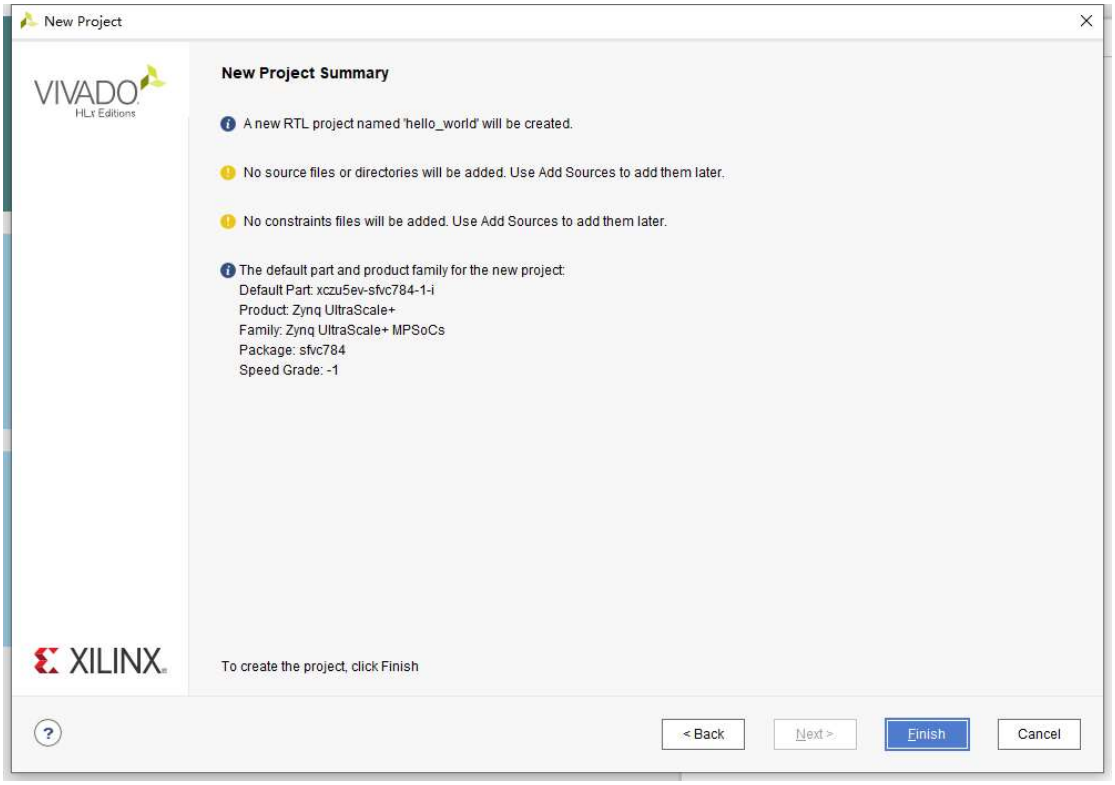


图 3-8 vivado 创建工程-综合界面

生成 vivado 工程，进入如下界面。

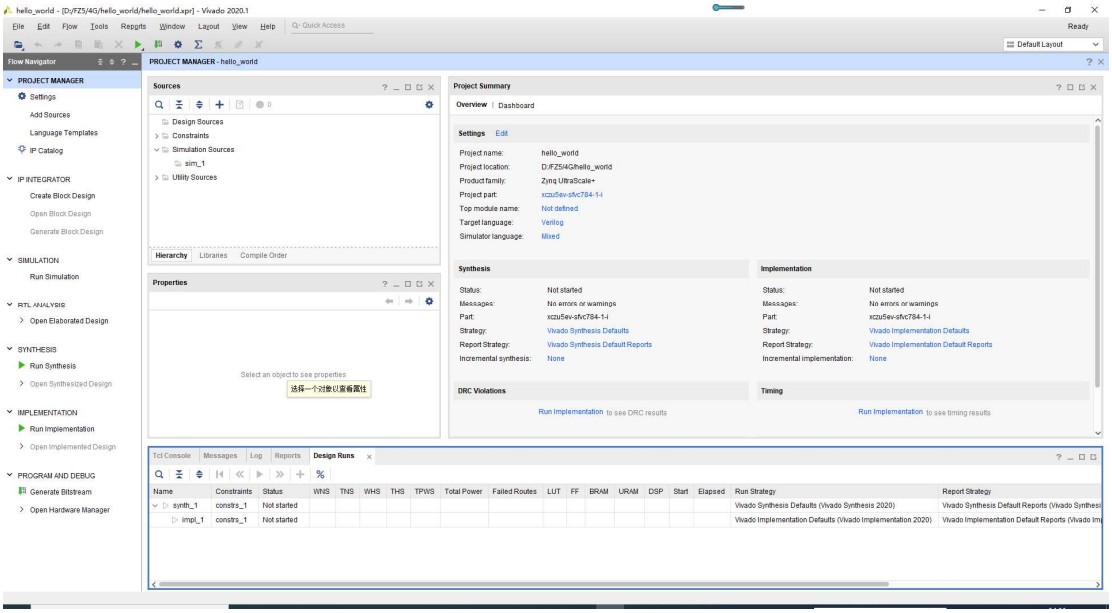


图 3-9 vivado 创建工程-工程开始视窗界面

在 IP INTEGRATOR 下面，点击 Create Block Design。

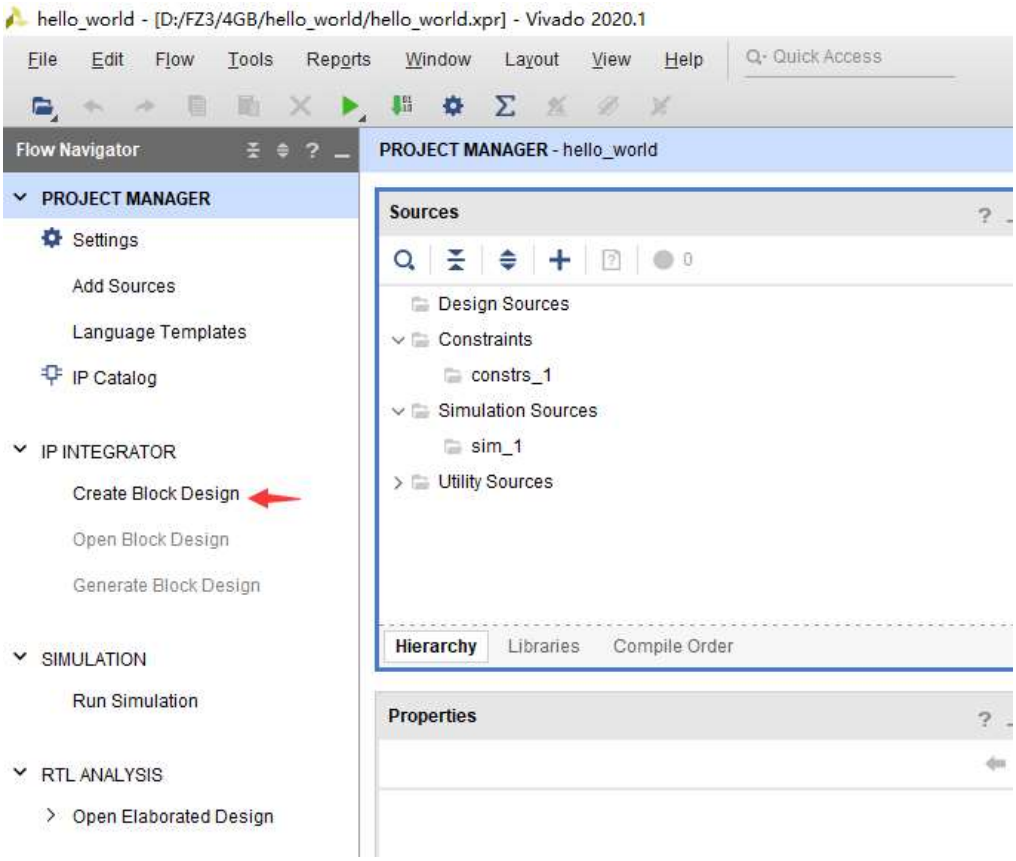


图 3-10

Design name 保持 design_1 不变即可，点击 OK。

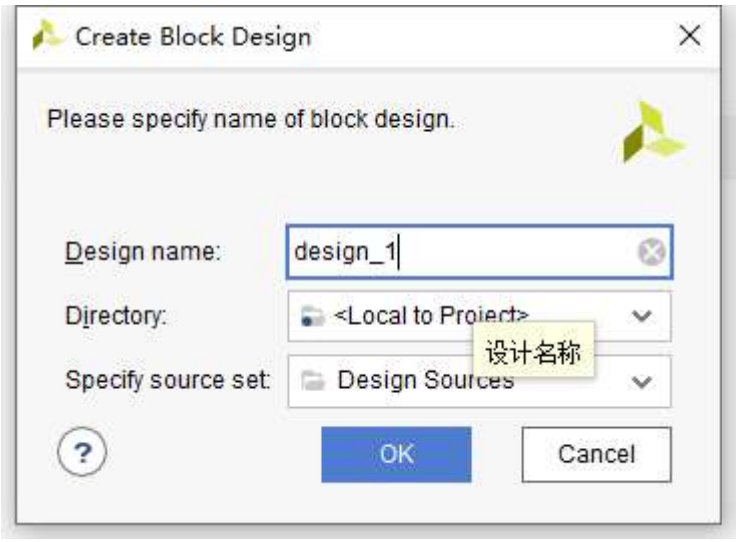


图 3-11 填写 Block design 名称

3.1.2 PS 的详细配置

点击 Add IP 添加 IP 核

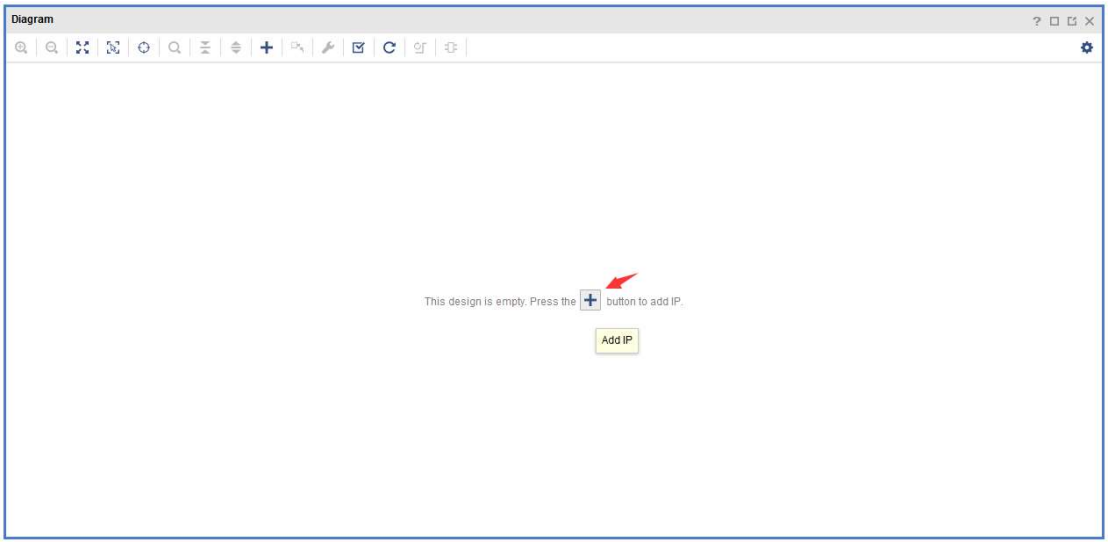


图 3-12 使用 Block design

输入 mpsoc，然后双击 Zynq UltraScale+MPSoC 添加 mpsoc 核

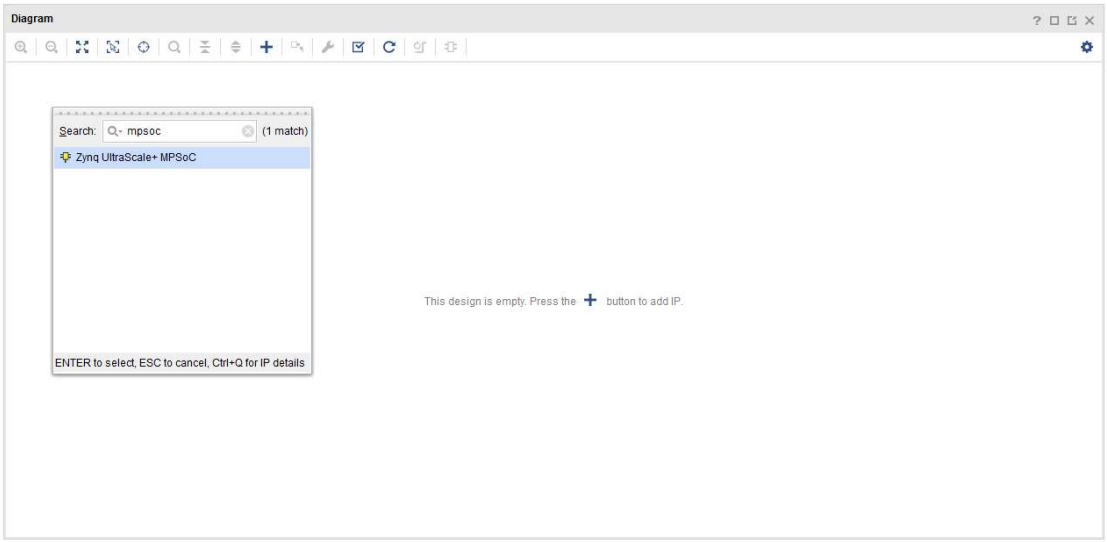


图 3-13 添加 MPSOC 核

Zynq mpsoc 核如下图所示

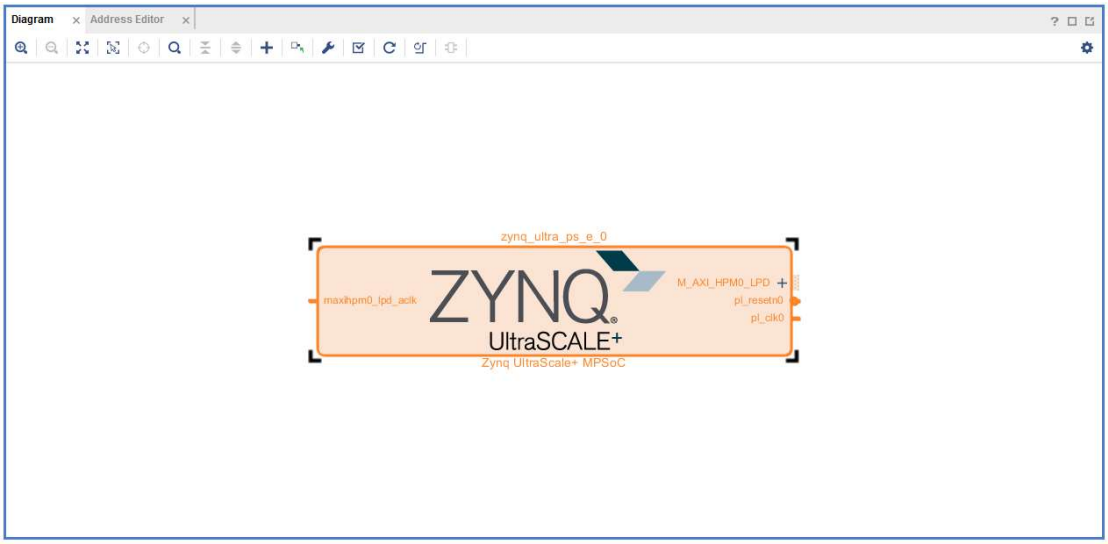


图 3-14 MPSOC 核

双击 zynq mpsoc 核配置相关参数。

首先出现的界面是 ZYNQ 硬核的架构图，可以很清楚看到它的结构，参考 ug1085 文档，里面有对 ZYNQ 的详细介绍。图中绿色部分是可配置模块，可以点击进入相应的编辑界面，当然也可以在左侧的窗口进入编辑。 下面对各个窗口的功能一一介绍。

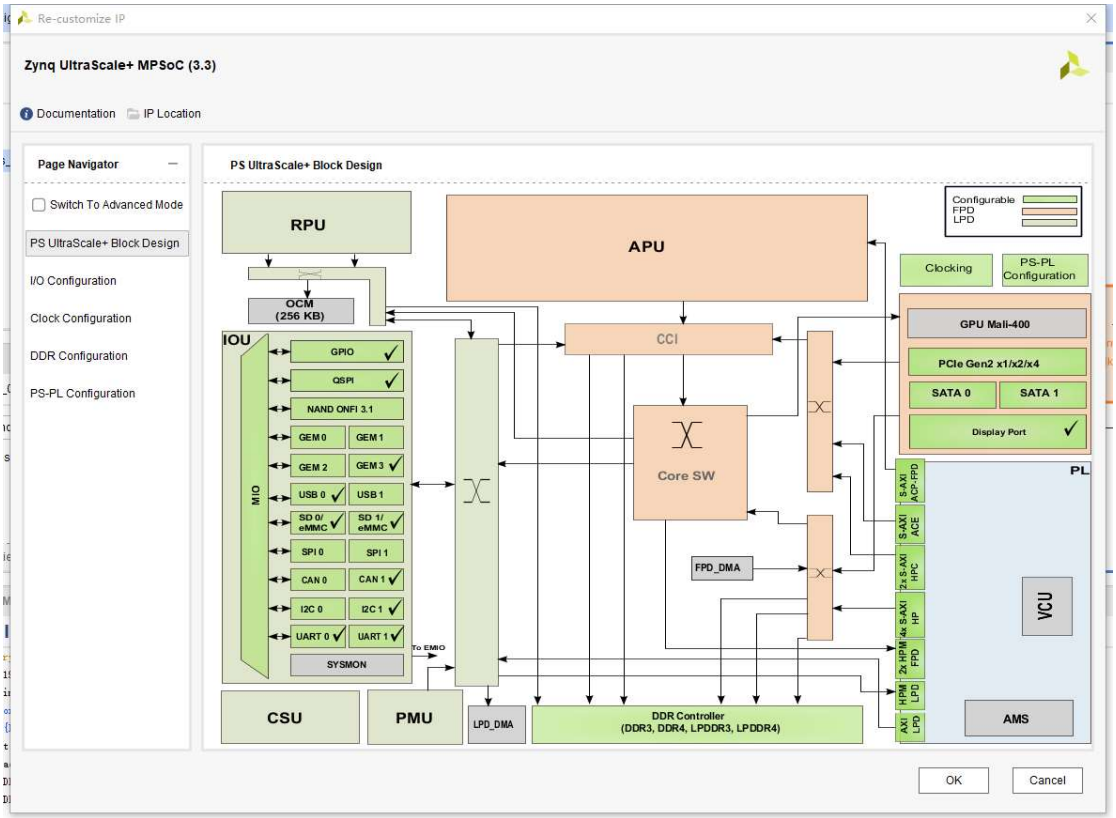


图 3-15 配置 MPSOC 界面

1、 电压配置

在 I/O Configuration 窗口，根据硬件电路的实际电压，配置 BANK0~BANK3 电压 LVCMOS33。

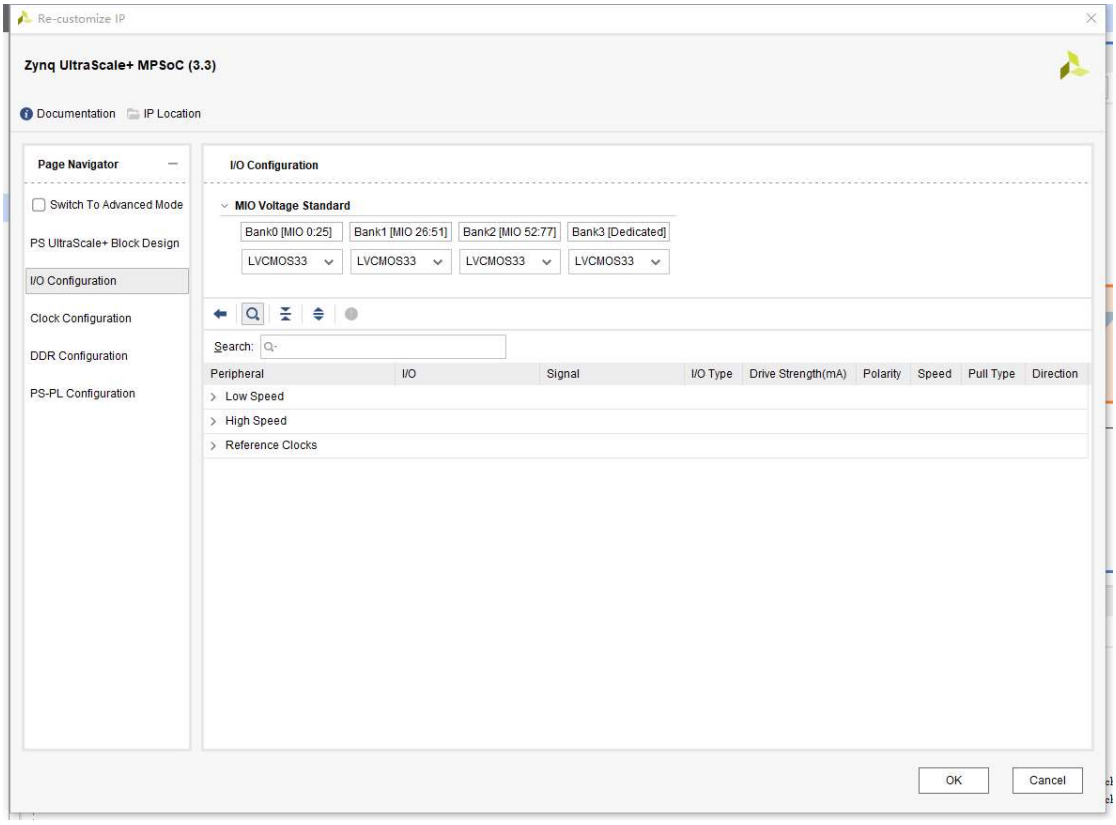


图 3-16 配置 IO

2、 Low Speed 配置

勾选 QSPI，并设置为”Single”模式， Data Mode 为”x4 “

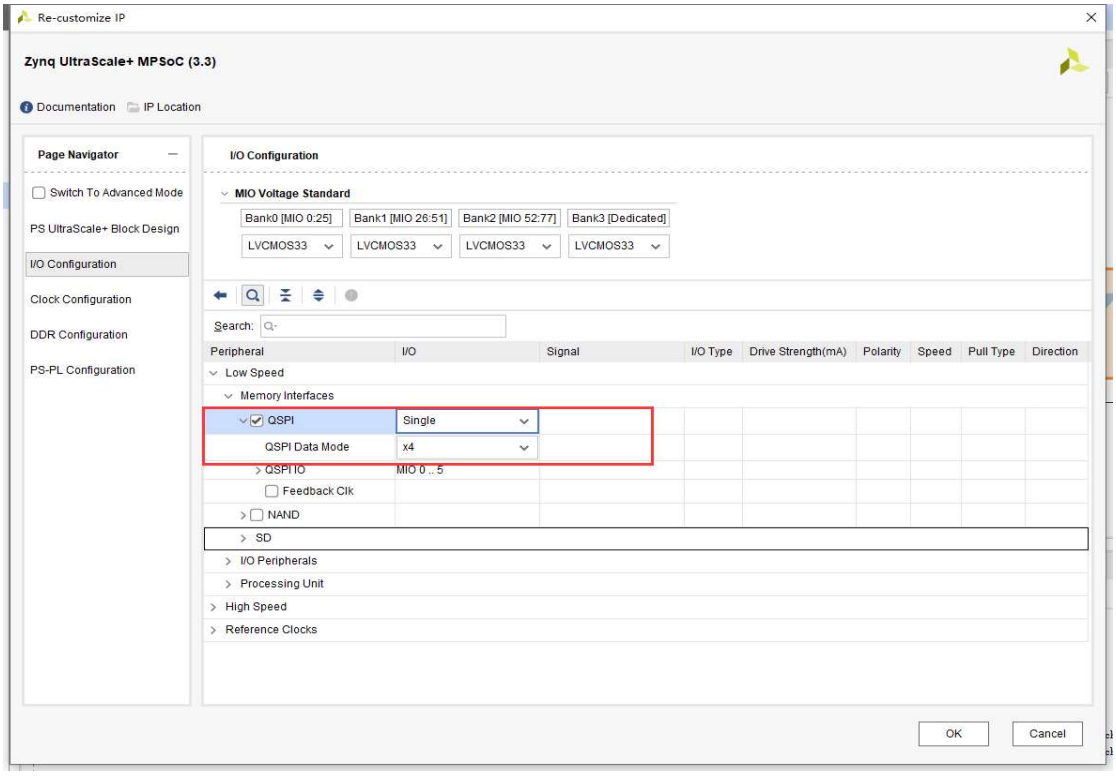


图 3-17 配置 QSPI

勾选 SD 0，配置 eMMC。 选择 MIO13..22， Slot Type 选择 eMMC， Data Transfer Mode 为 8Bit，勾选 Reset，并选择 MIO23。

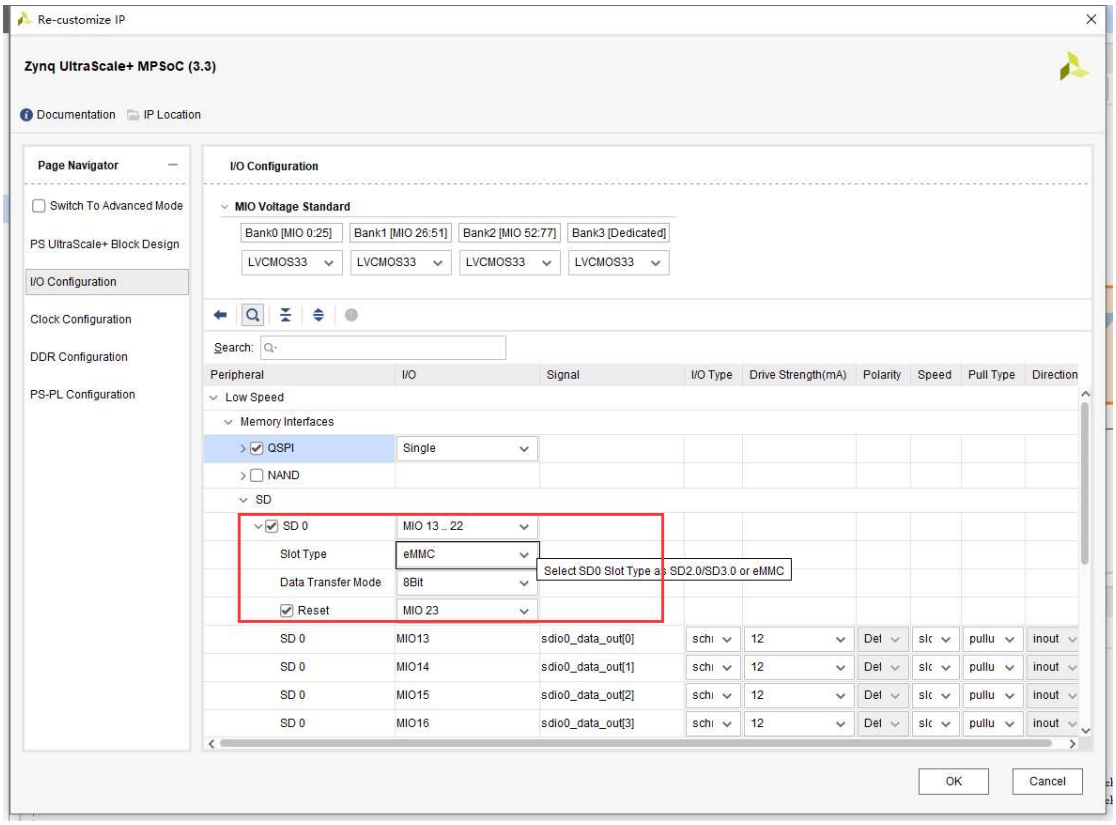


图 3-18 配置 SD

勾选 SD 1，配置 SD 卡。选择 MIO 46..51， Slot Type 选择 SD 2.0， Data Transfer Mode 选择 4Bit，勾选 CD，用于检测 SD 卡插入，选择 MIO45，勾选 WP，选择 MIO44，用于 SD 卡写保护。

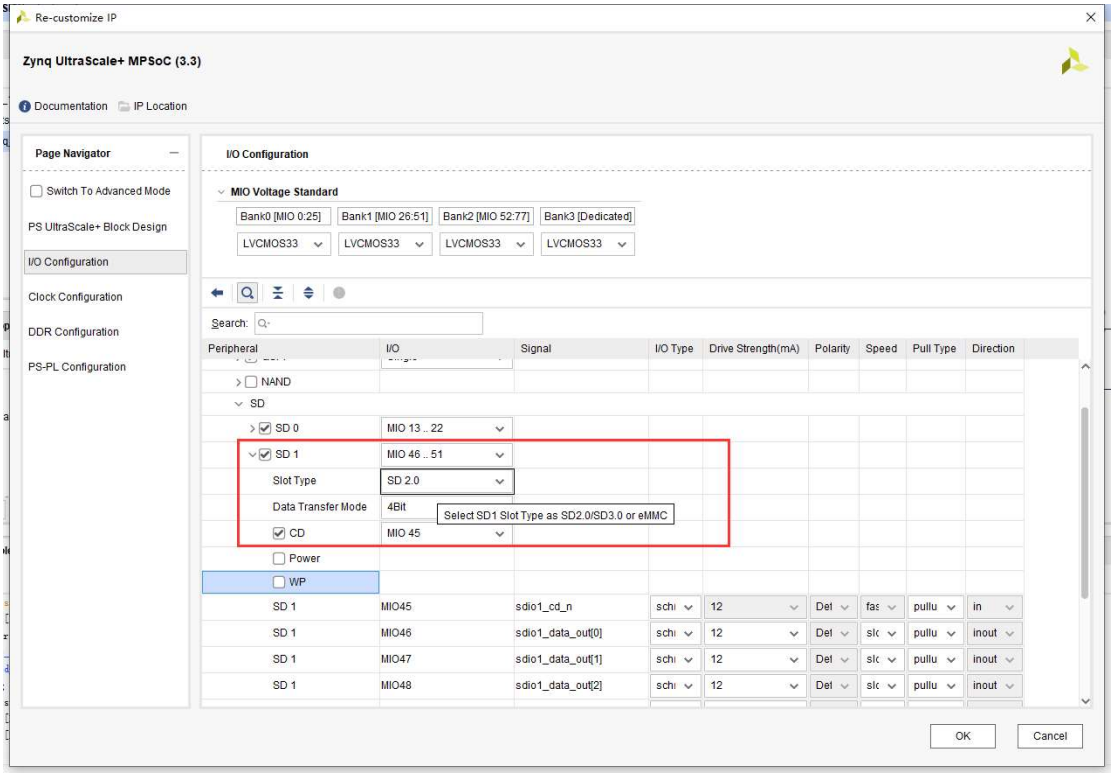


图 3-19 配置 SD1

勾选 CAN 1，选择 MIO 28..29

勾选 I2C 1，选择 MIO24..25

勾选串口 UART 0，选择 MIO 26..27，勾选 UART 1，选择 MIO 8..9

勾选 GPIO1 MIO、GPIO0 MIO

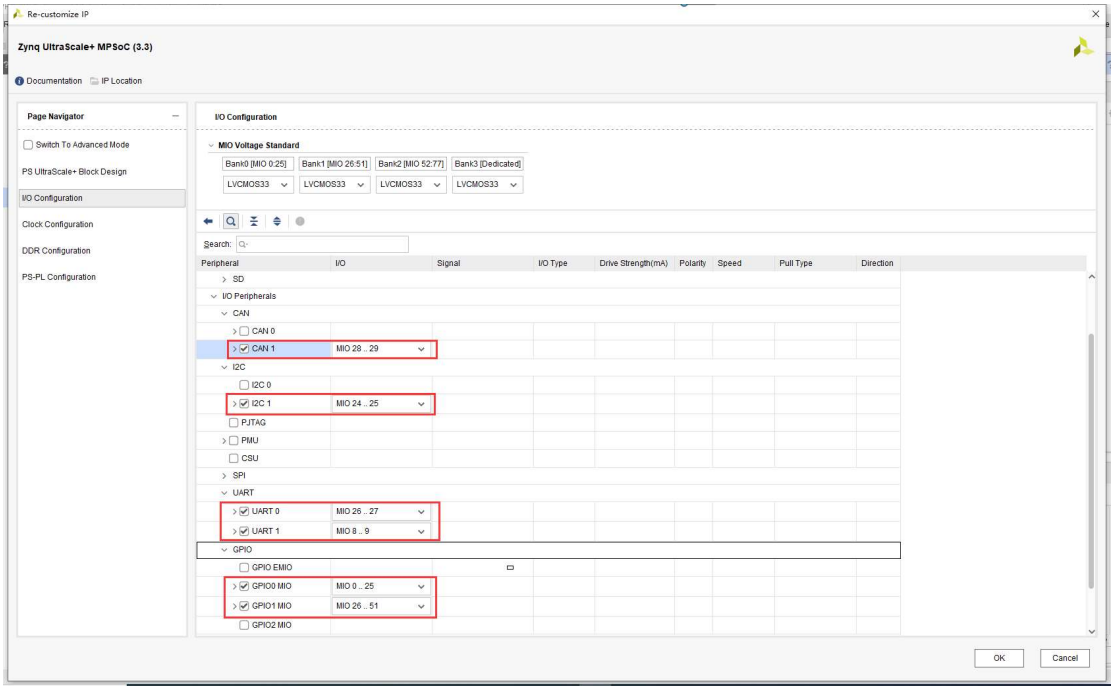


图 3-20 配置 CAN、I2C、UART、GPIO

勾选 SWDT 0、SWDT 1

勾选 TTC 0~TTC 3

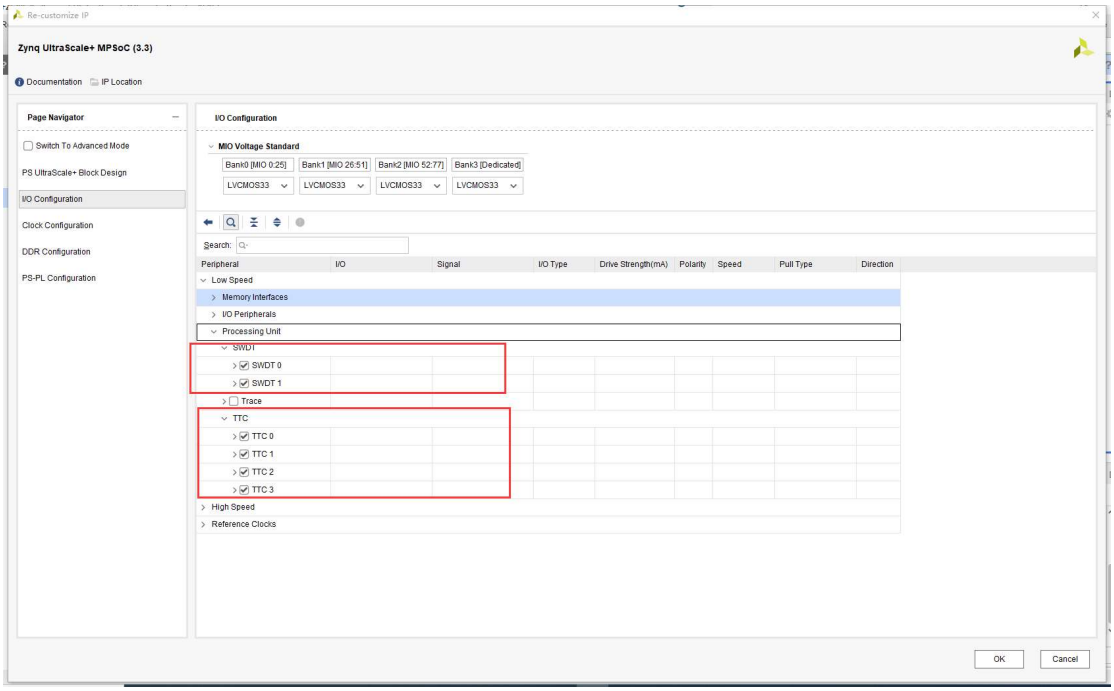


图 3-21 配置 SWDT、TTC

3、High Speed 配置

High Speed 部分首先配置 PS 端以太网，勾选 GEM 3，选择 MIO 64..75，勾选 MDIO3，选择 MIO 76..77

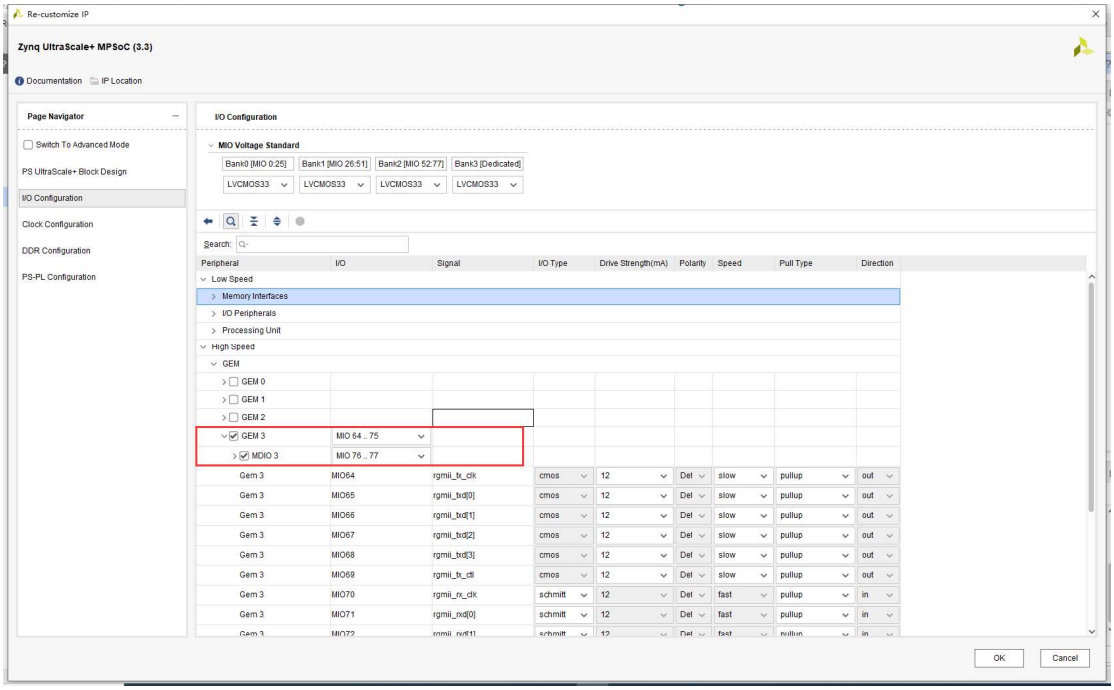


图 3-22 配置 GEM

勾选 USB 0，选择 MIO 52..63，勾选 USB 3.0，选择 GT Lane1

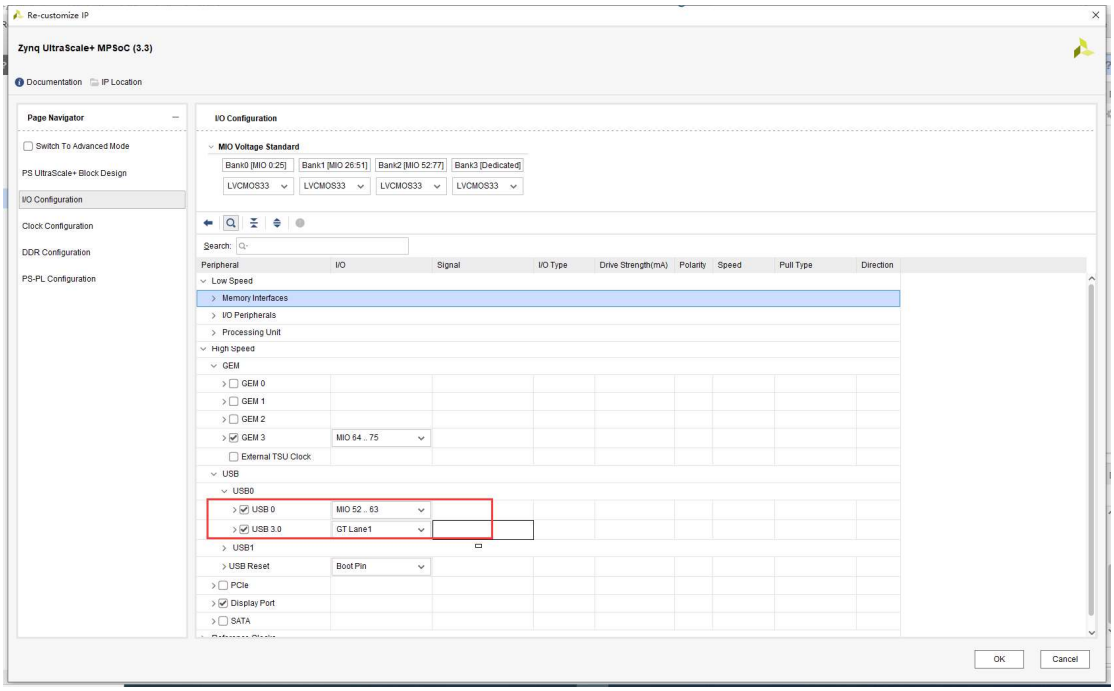


图 3-23 配置 USB

勾选 Display Port，DPAUX 选择 MIO 34..37， Lane Selection 选择 Dual Higher

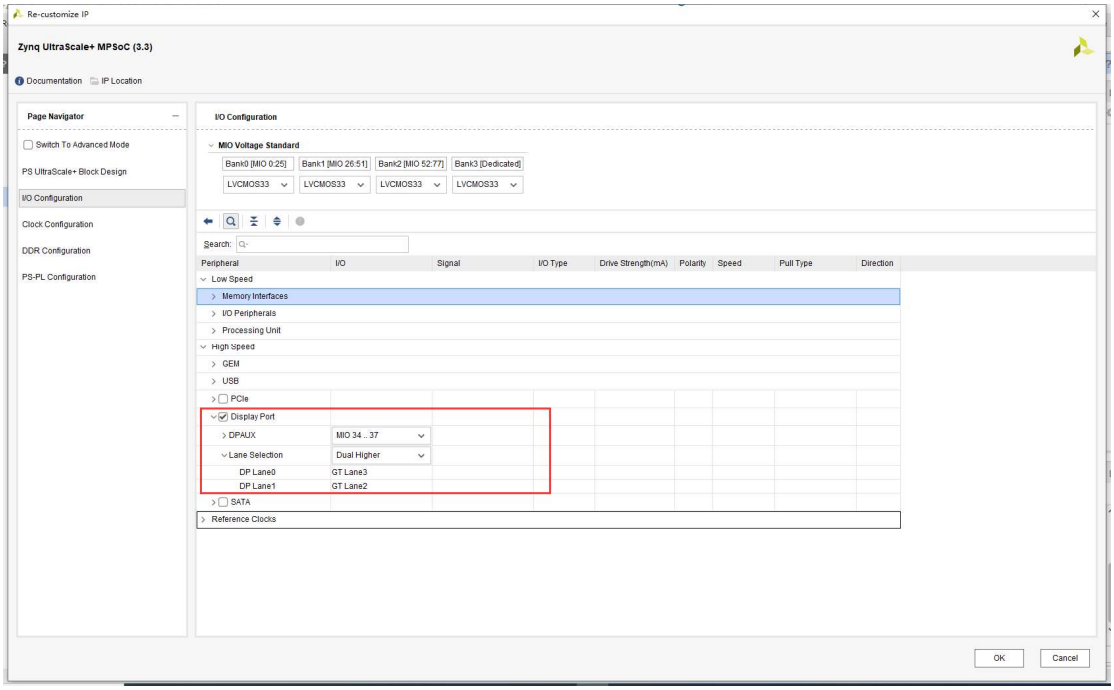


图 3-24 配置 Display Port

至此，IO 部分配置完成。

4．时钟配置

在 Clock Configuration 界面， Input Clocks 窗口配置参考时钟，其中 PSS_REF_CLOCK 为 ARM 的参考时钟，默认为 33.333MHz；Display Port 选择 Ref Clk2，27MHz； USB0 选择 Ref Clk1， 26MHz。

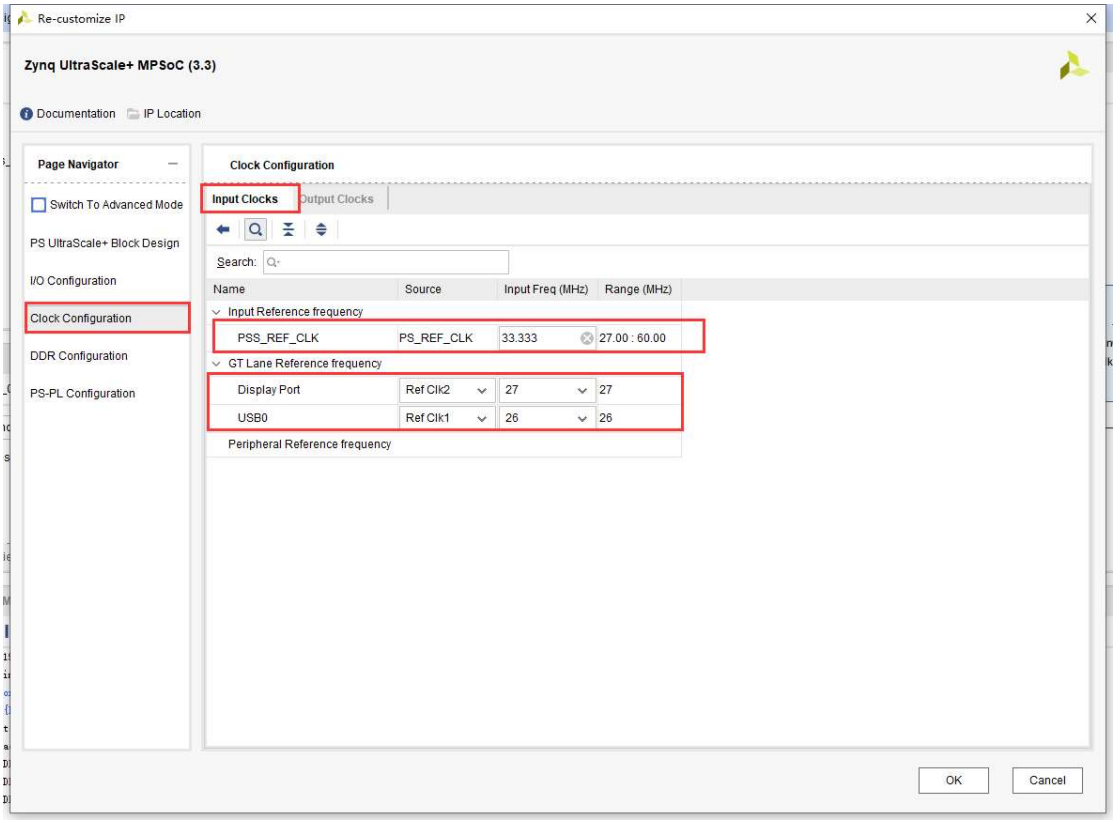


图 3-25 配置 PS 段输入时钟

PL 的时钟保持默认，这是给 PL 端逻辑提供的时钟。

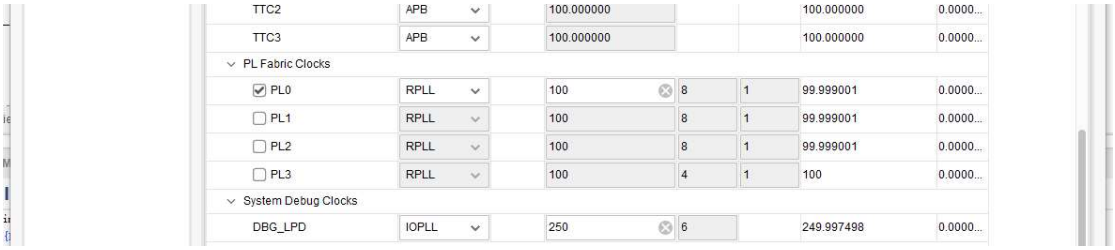


图 3-26 配置给 PL 的时钟

Full Power 部分，其他保持默认，将 DP_VIDEO 改为 VPLL， DP_AUDIO 和 DP_STC 改为 RPLL。

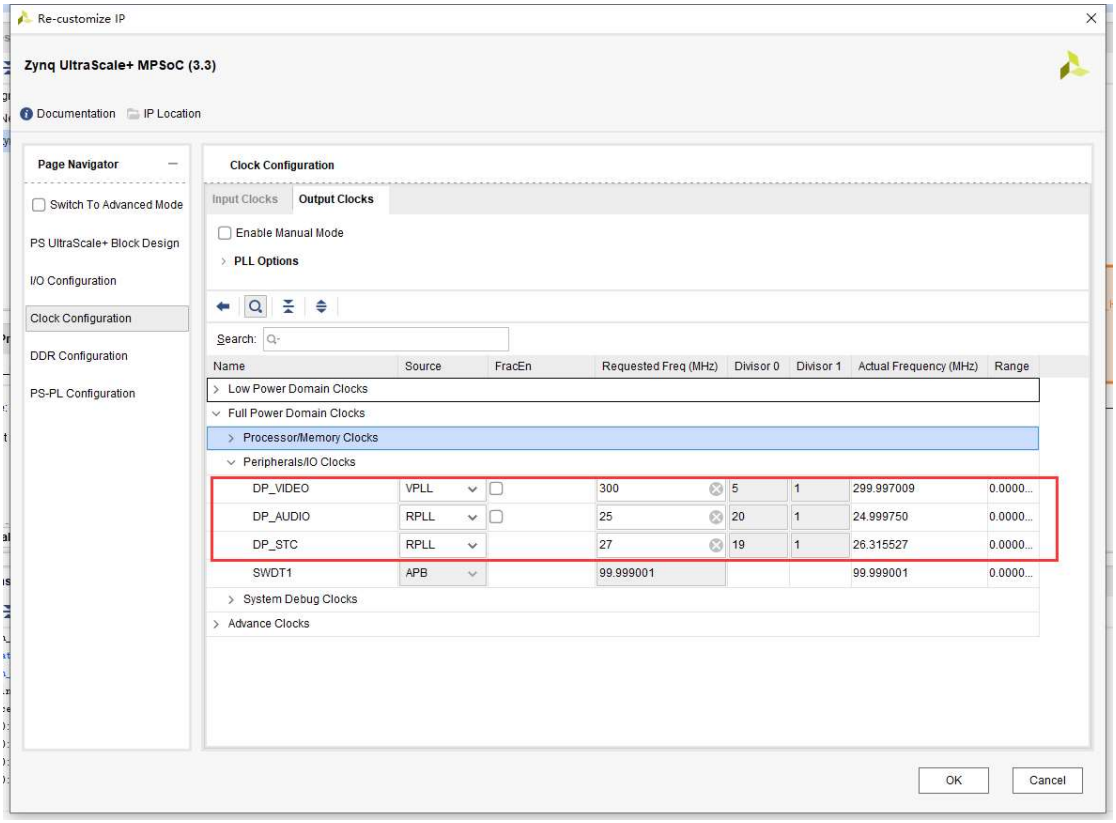


图 3-27 配置外设 IO 时钟

最下面的 Interconnect 修改如下

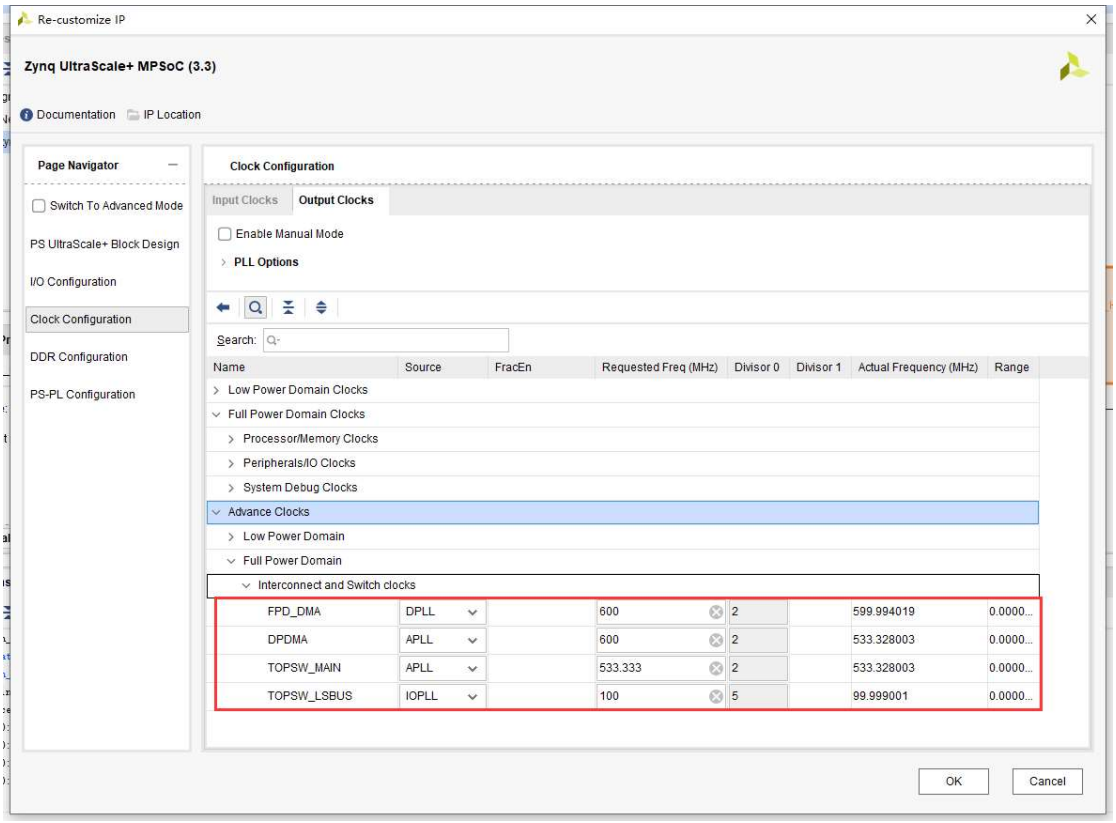


图 3-28 配置需要互联结构的模块时钟

其他部分保持默认，至此，时钟部分配置完成。

5、DDR 配置

在 DDR Configuration 窗口中， Load DDR Presets 选择”DDR4_MICRON_MT40A256M16GE_083E”。

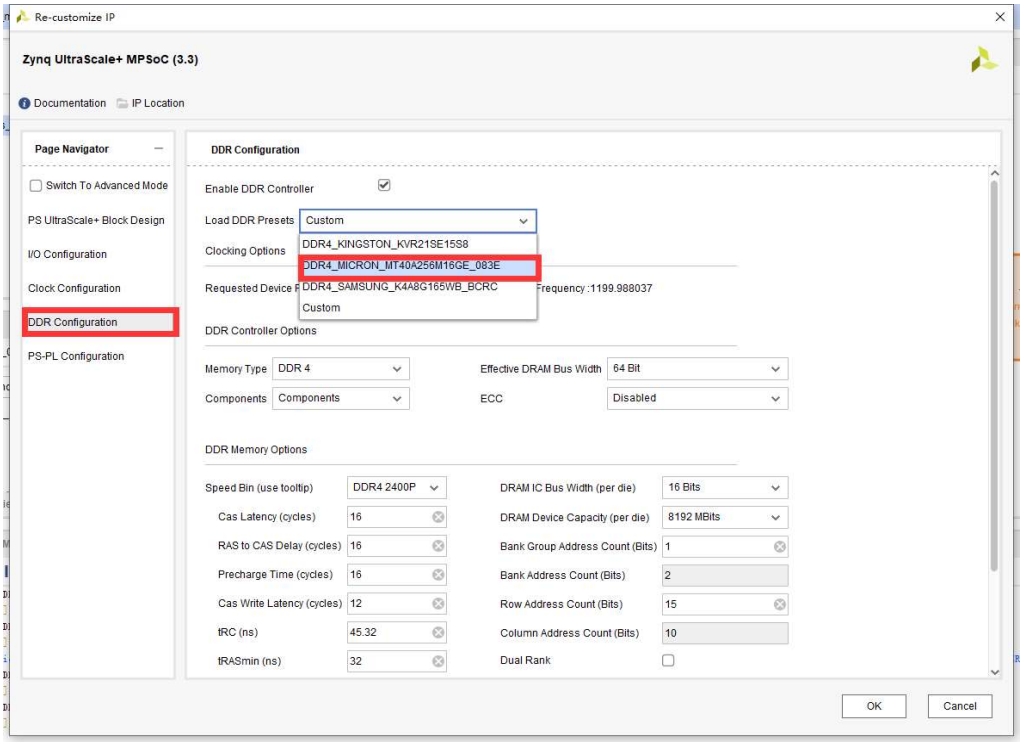


图 3-29 配置-DDR 厂家

参数做些修改如下：

4G 内存版本

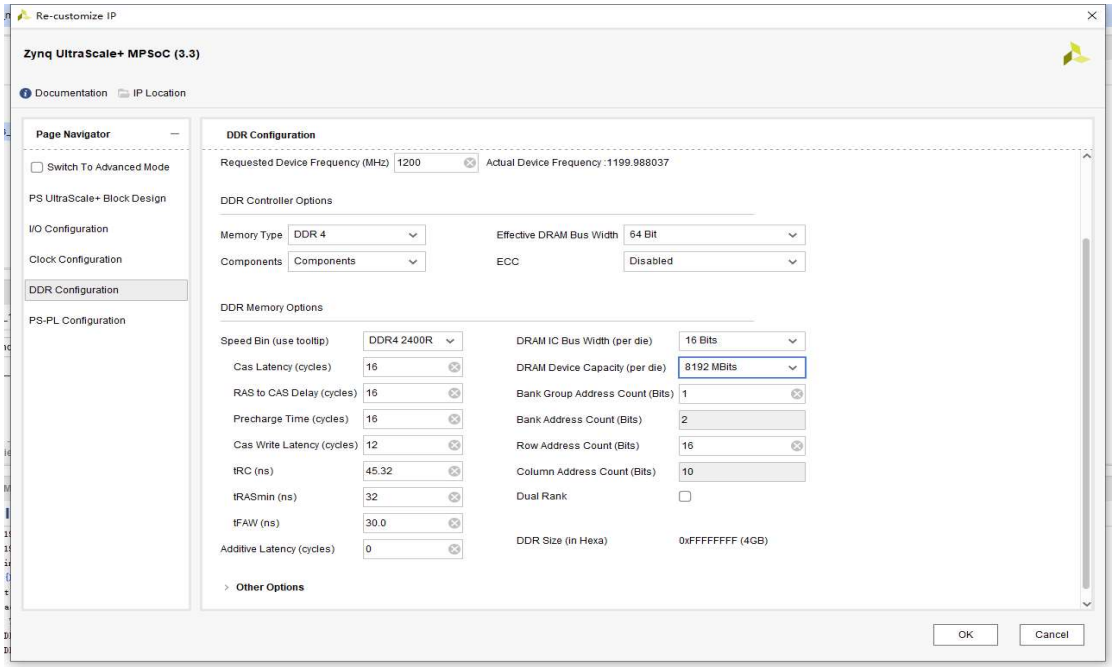


图 3-30 配置 DDR 位宽容量

8G 内存版本：

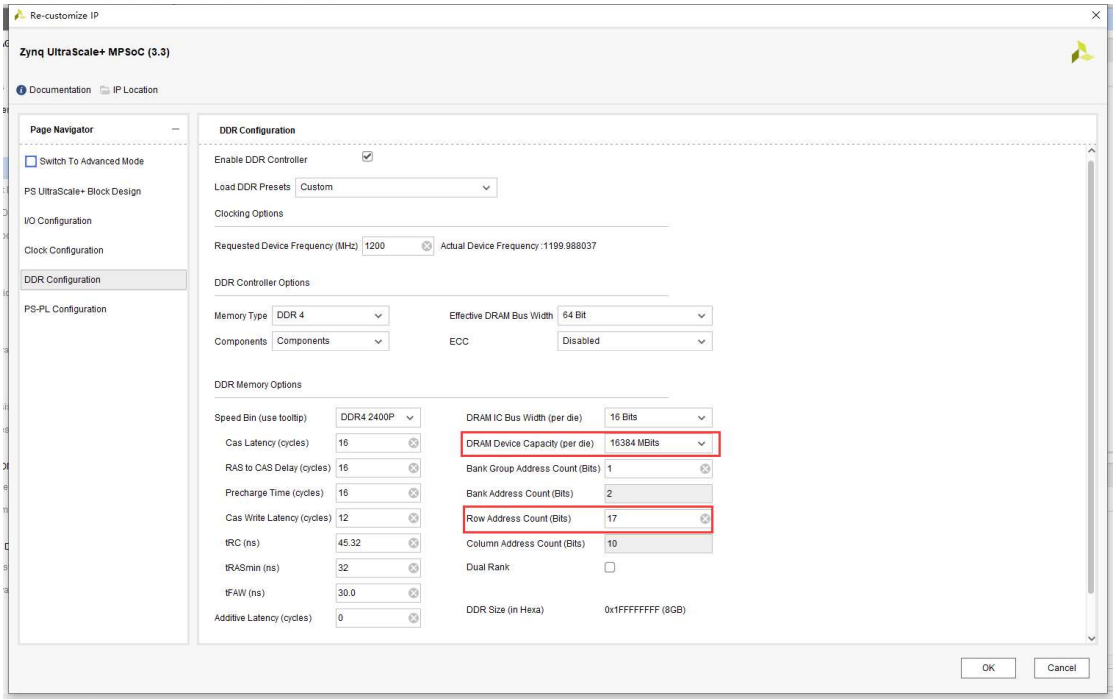


图 3-31 配置 DDR 位宽容量

其它保持默认，点击 OK， 配置完成，并连接时钟如下：

图 3-32 自动连线

3.1.3 生成 xsa 文件

右击 design_1.bd->Generate Output Products->Generate

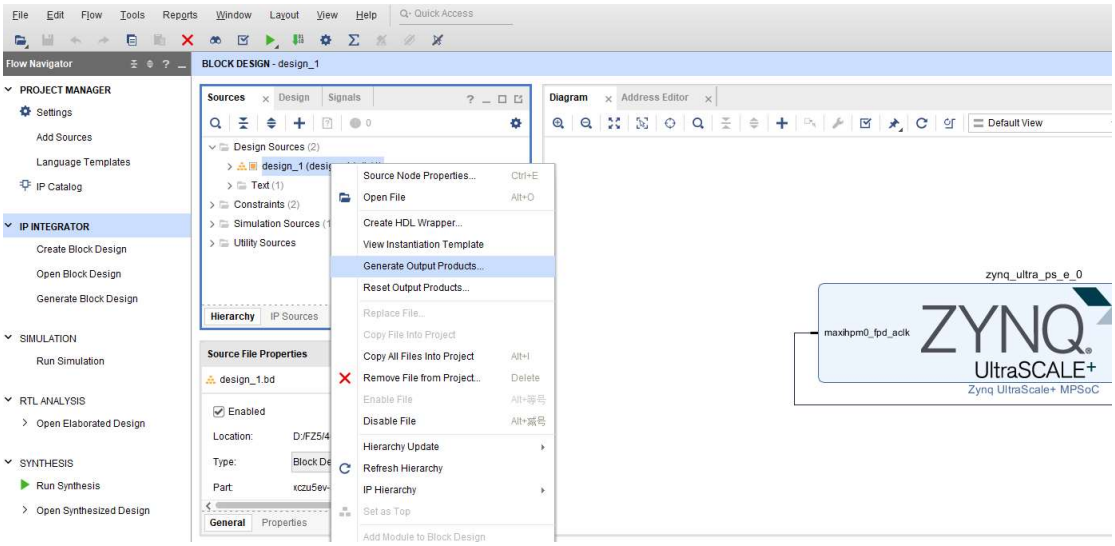


图 3-33 Generate output products

点击 Generate

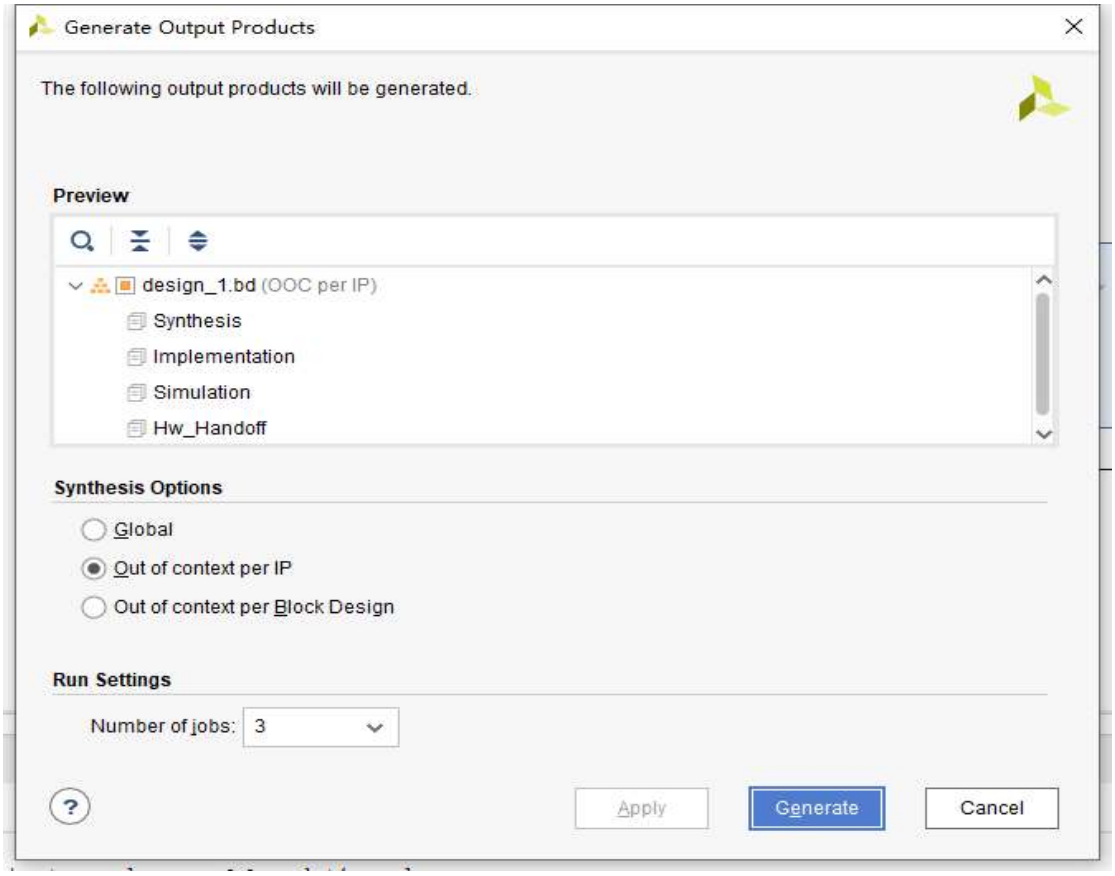


图 3-34 综合选项 Out ofcontext per IP

选择 design_1.bd,鼠标右键--CreateHDL Wrapper

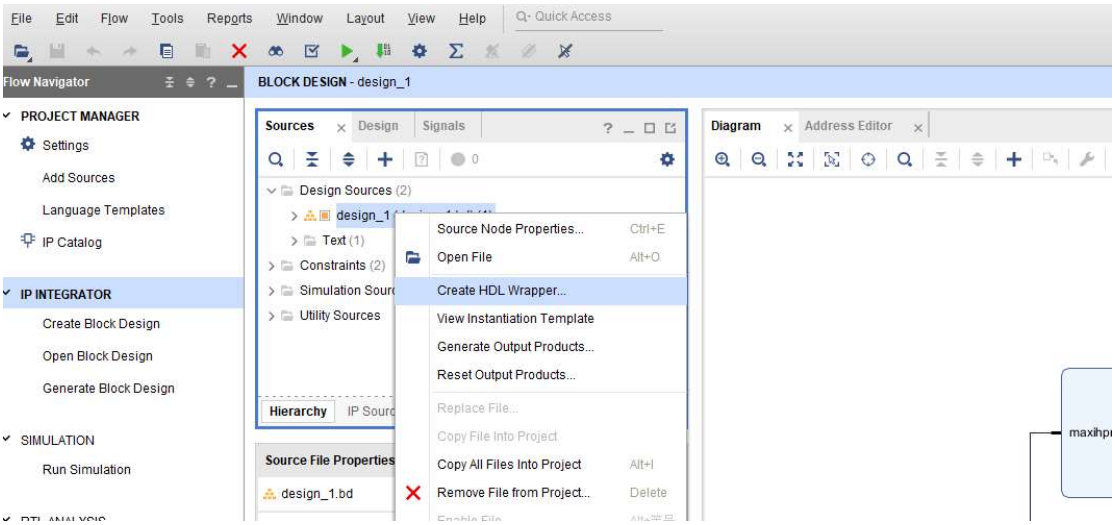


图 3-35 生成顶层文件

点击 OK

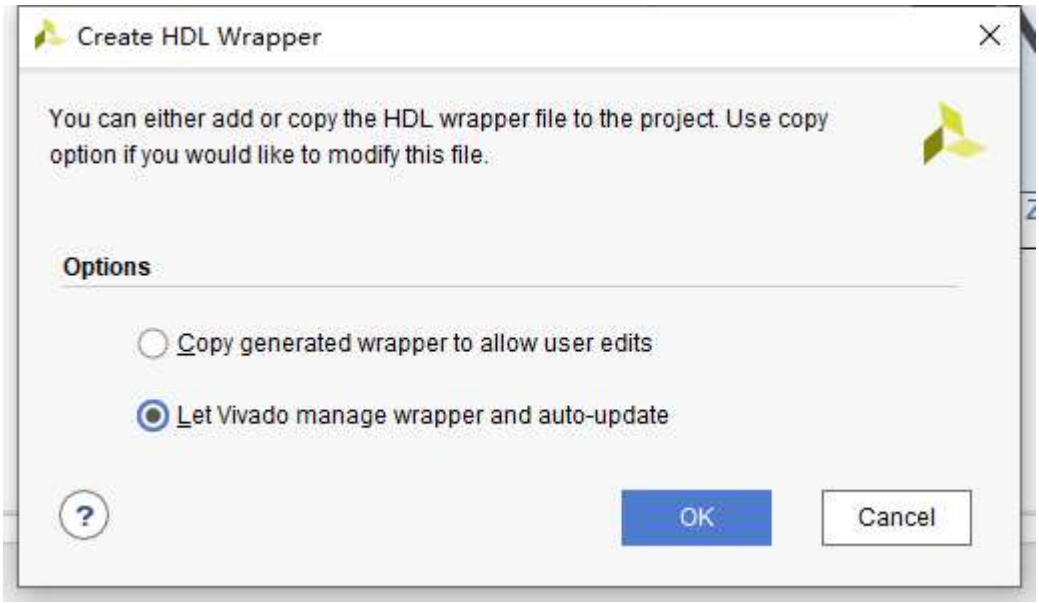


图 3-36 自动更新顶层文件

点击编译按钮。

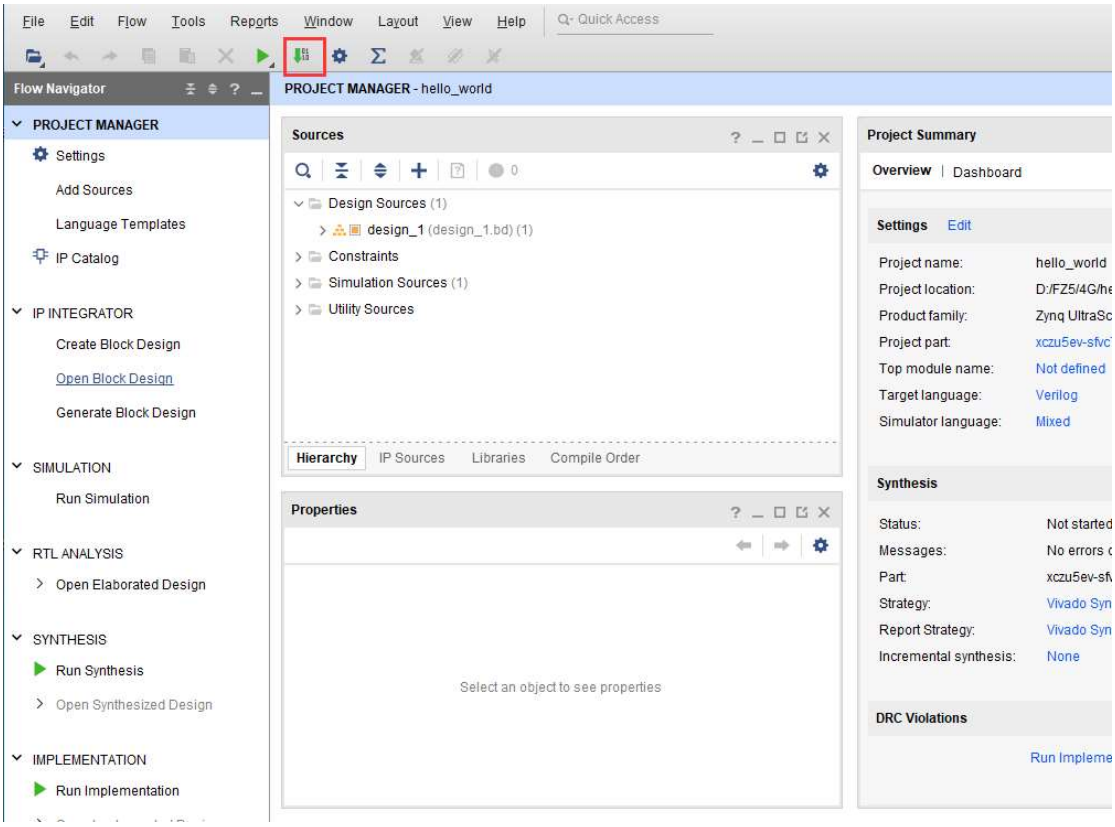


图 3-37 生成 bitstream

点击 Yes

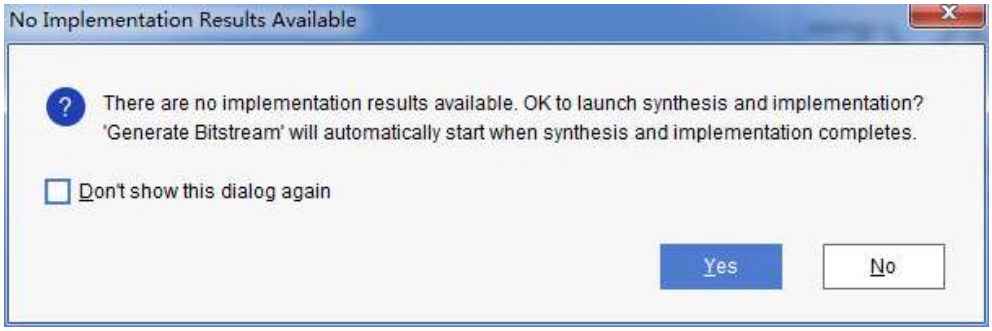


图 3-38 确认

点击 OK

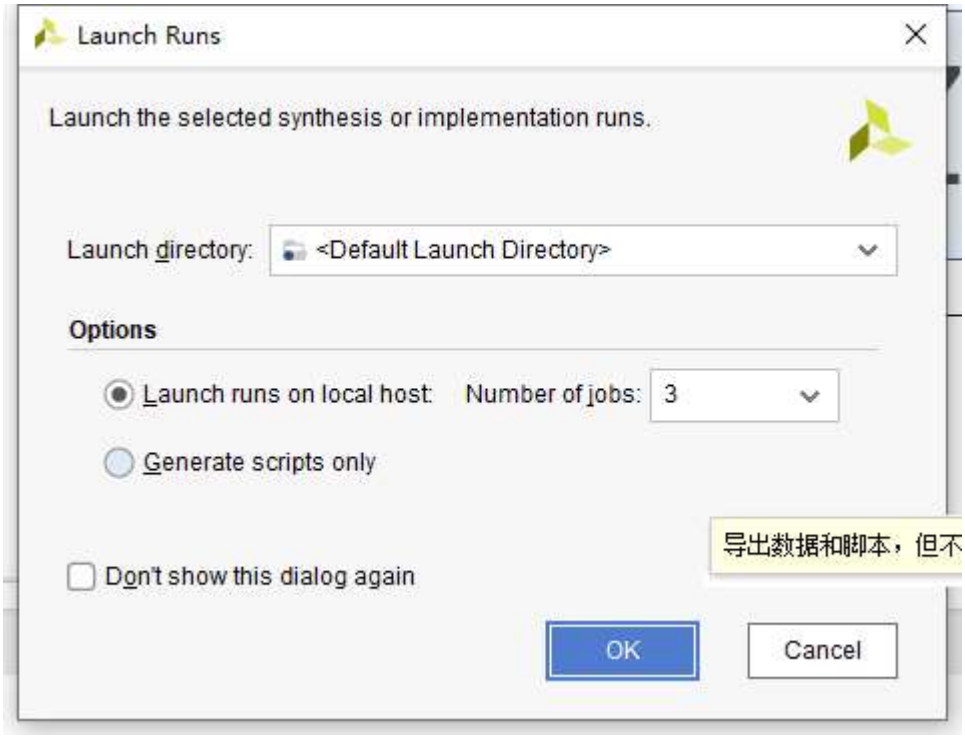


图 3-39 启动综合、以及实现

点击 Cancel

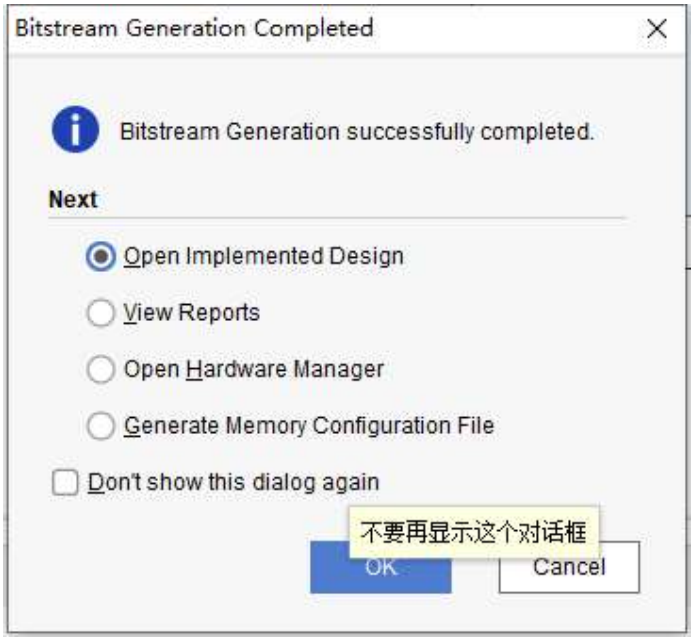


图 3-40 生成 bitstream 成功

可以看到 bitstream 编译成功

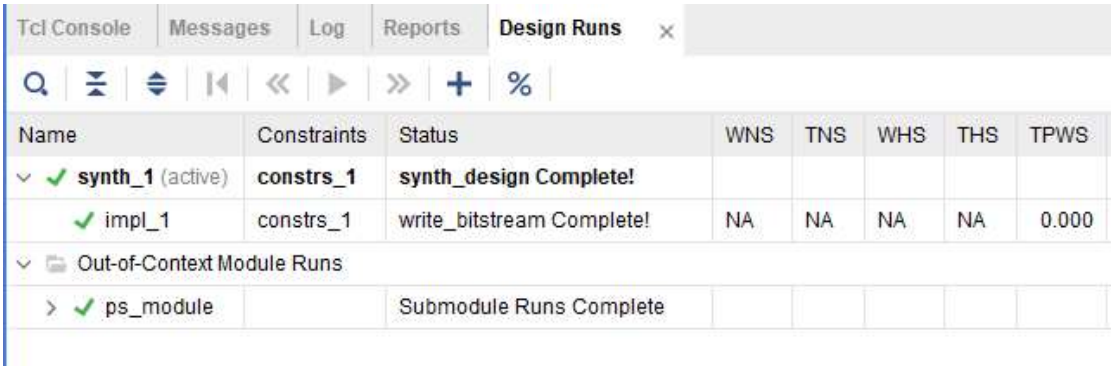


图 3-41 bitstream 生成成功

点击菜单栏上的 File->Export->Export Hardware->OK 导出硬件配置文件

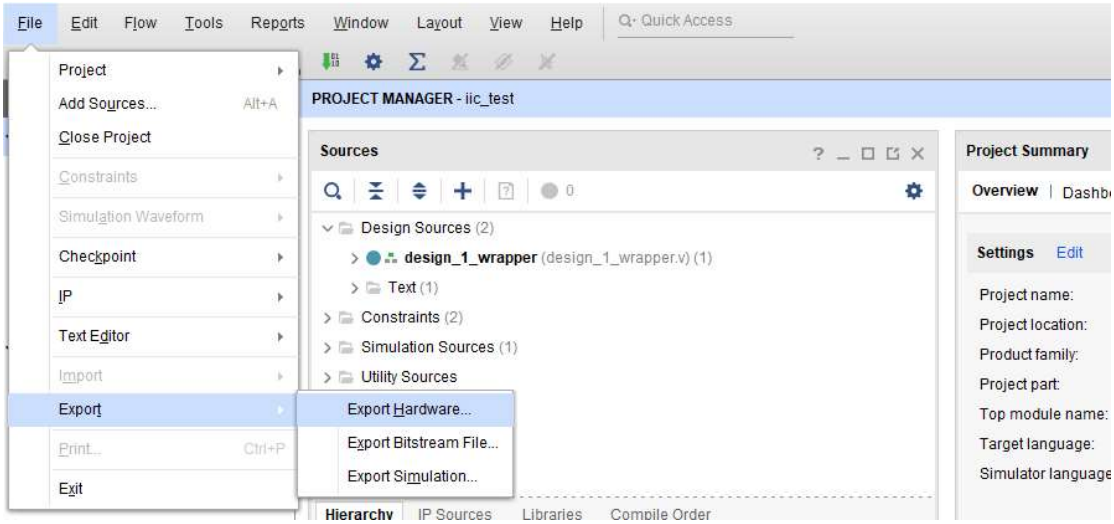


图 3-42 导出硬件设计平台

选择 fixed，点击 Next

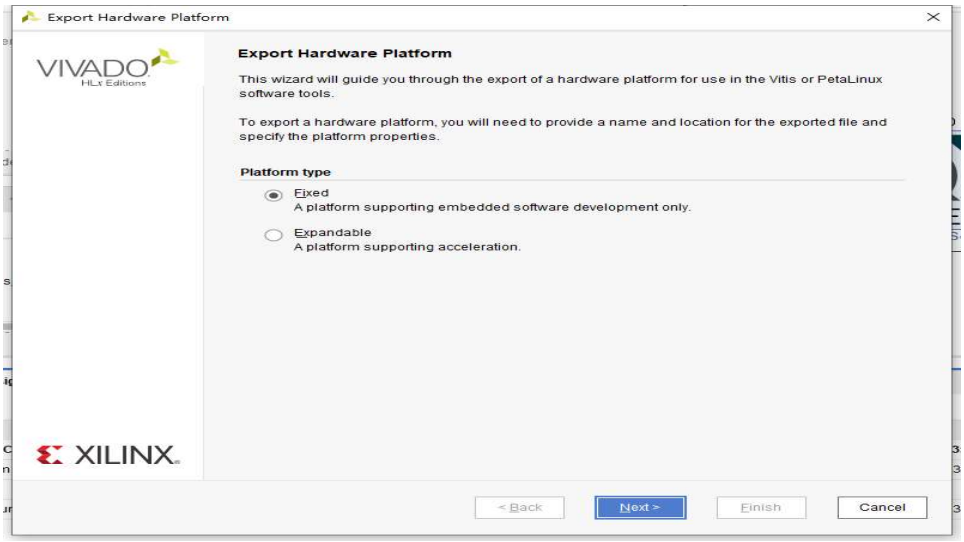


图 3-43 选择 Fixed

选择 include bitstream，点击 Next

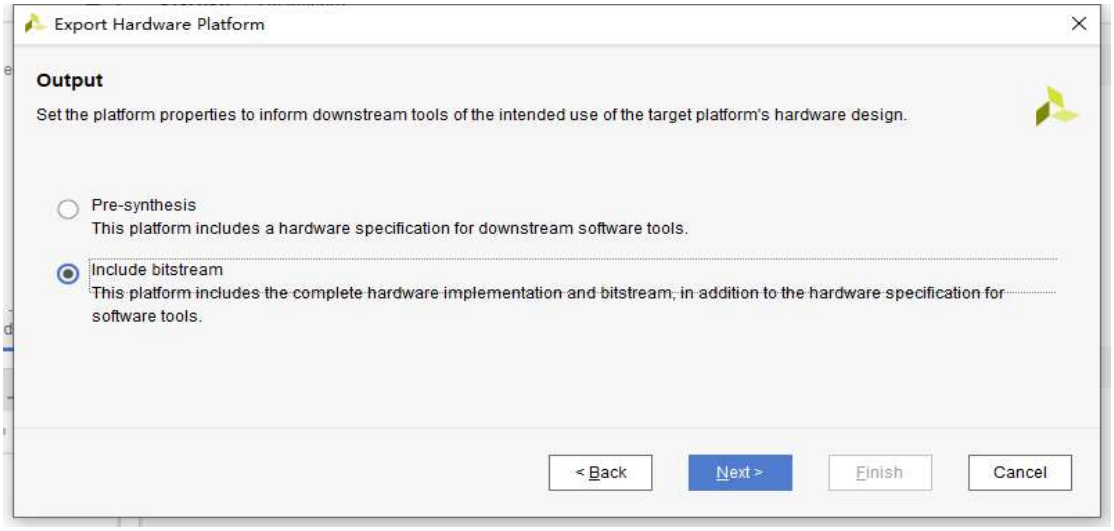


图 3-44 选择导出的包含 bitstream

选择导出文件名和导出路径。这里文件名选择默认，路径为工程文件下新建的 vitis 文件夹。

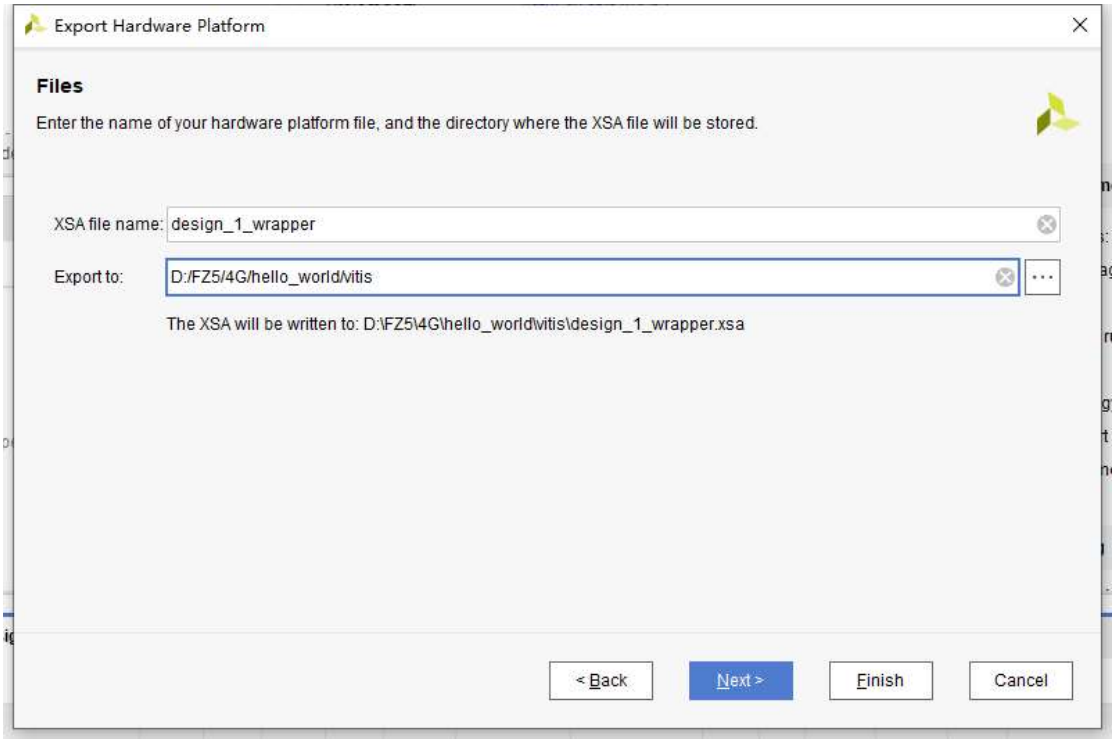


图 3-45 导出位置

3.1.4 建立 vitis 的 app 工程

Vitis 是独立的软件，可以双击 Vitis 软件打开，也可以通过在 Vivado 软件中选择 Tools--Launch Vitis 打开 Vitis 软件

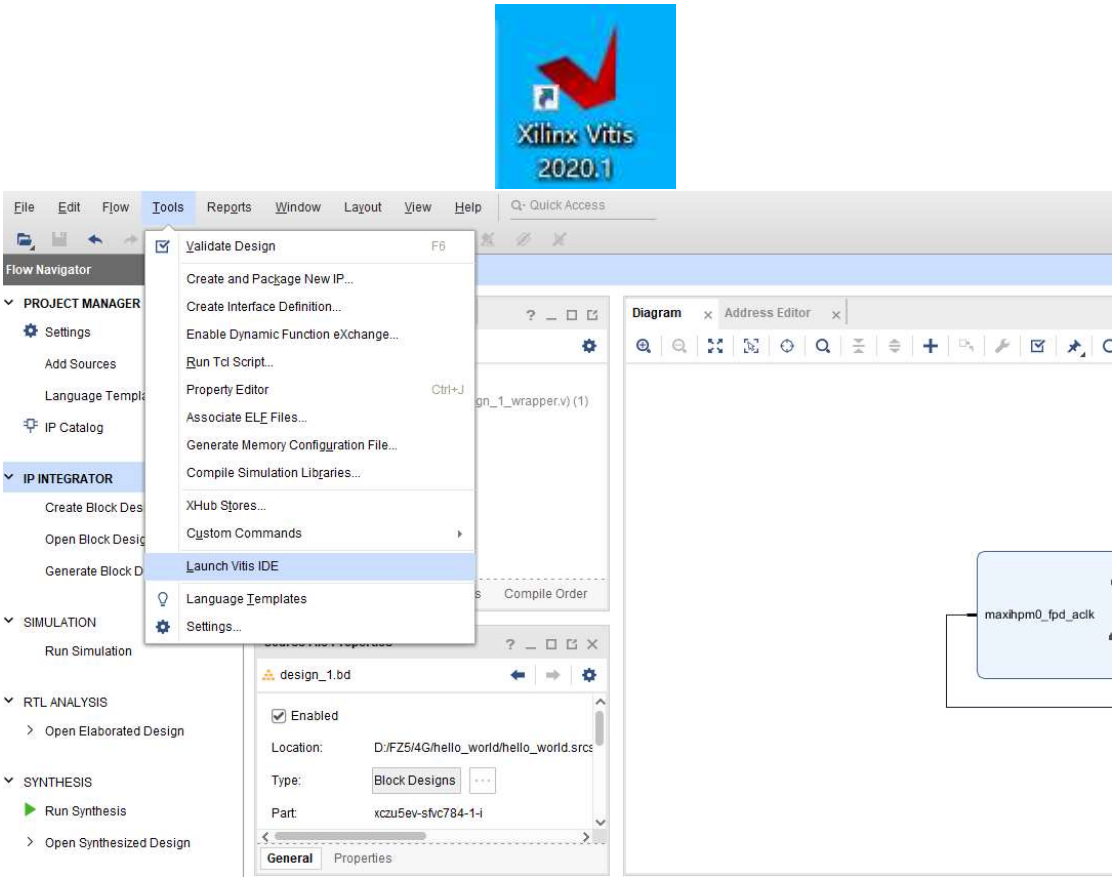


图 3-46 vivado 上启动 vitis

选择之前新建的 vitis 文件夹，点击” Launch”

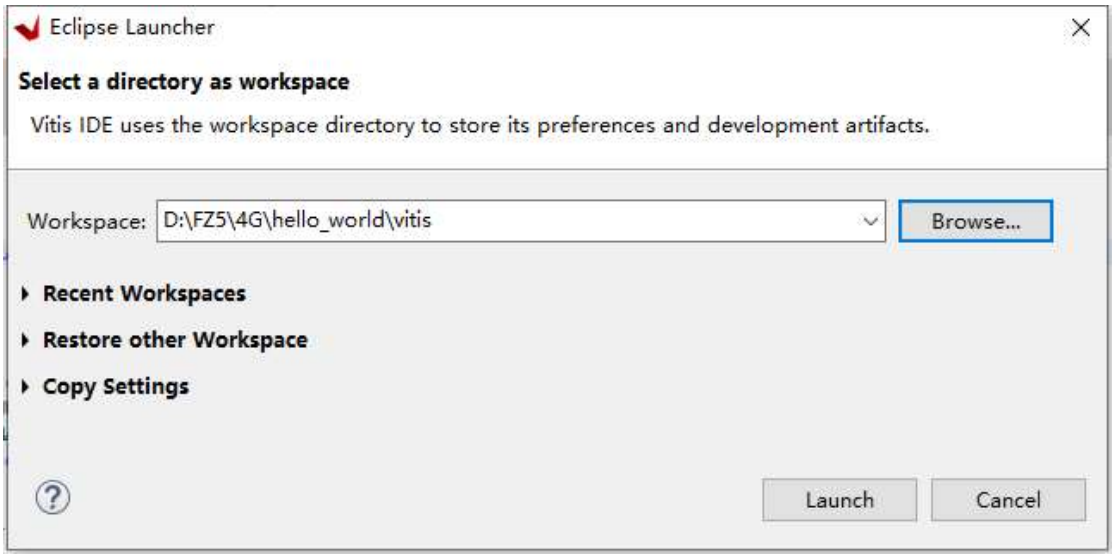


图 3-47 选择 vitis 工作文件夹

启动 Vitis 之后界面如下，点击“Create Application Project” ，这个选项会生成 APP 工程以及 Platfrom 工程， Platform 工程类似于以前版本的 hardware platform，包含了硬件支持的相关文件以及 BSP。

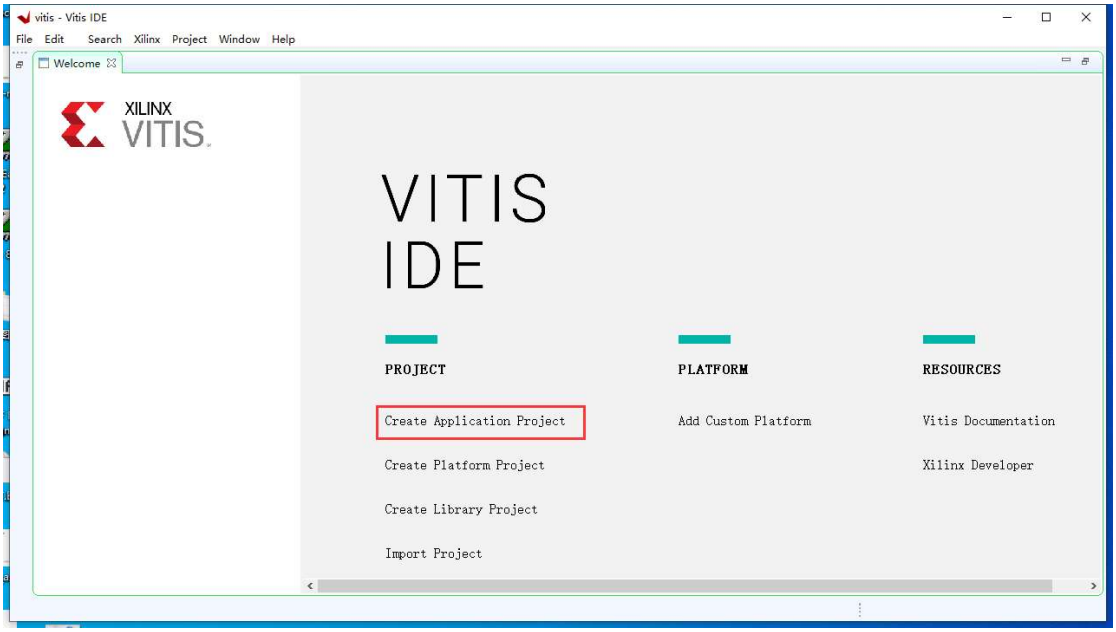


图 3-48 初始界面

第一页为介绍页，直接跳过，点击 Next

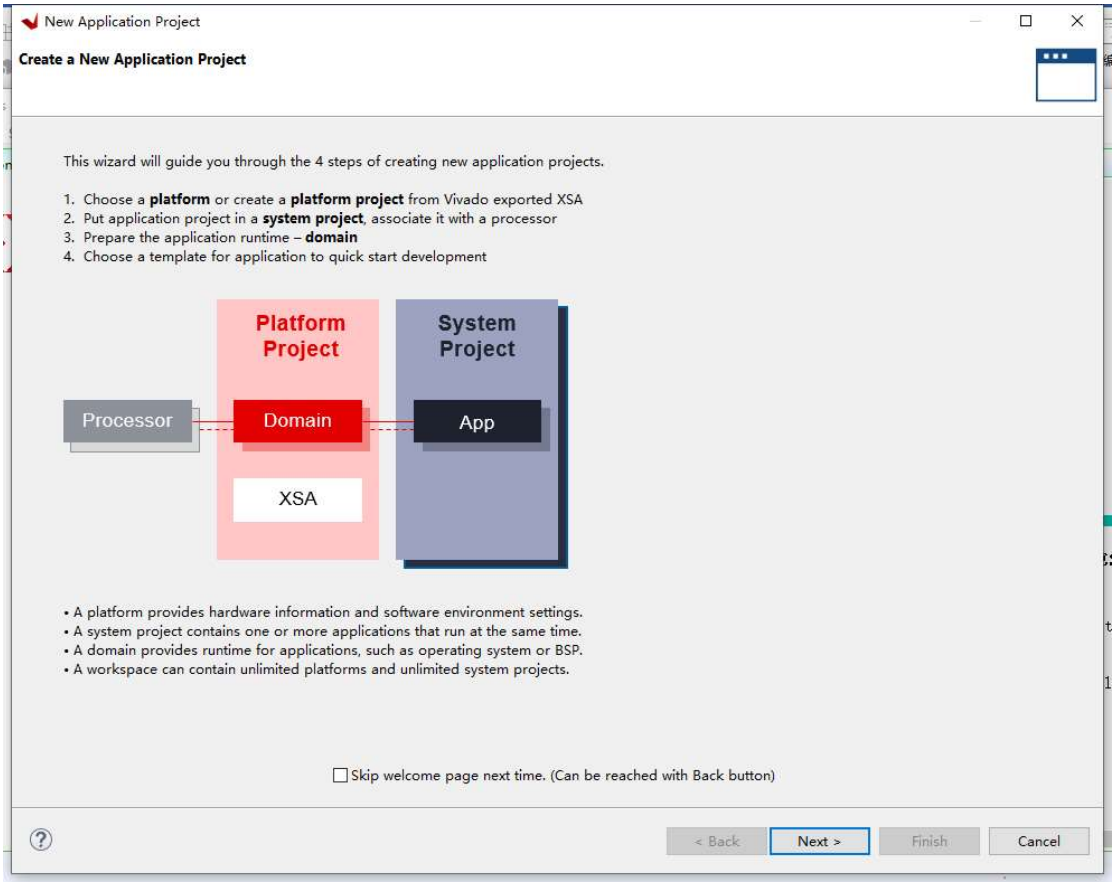


图 3-49 新建 application

选择“Create a new platform from hardware(XSA)”，打开“Browse”，选择刚才导出的 xsa 文件

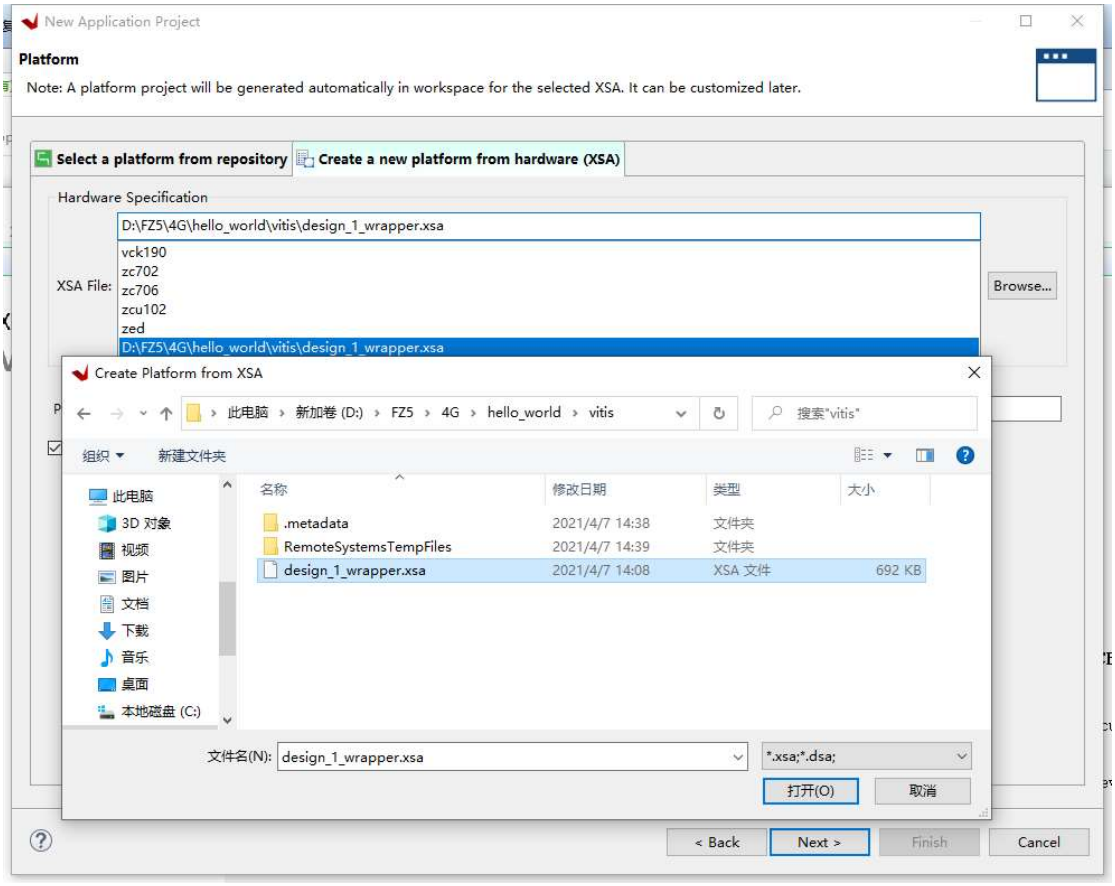


图 3-50 选择 vivado 导出的 xsa 文件作为硬件平台

最下面的 Generate boot components 选项，如果勾选上，软件会自动生成 fsbl 工程，我们一般选择默认勾选上。

点击 next。

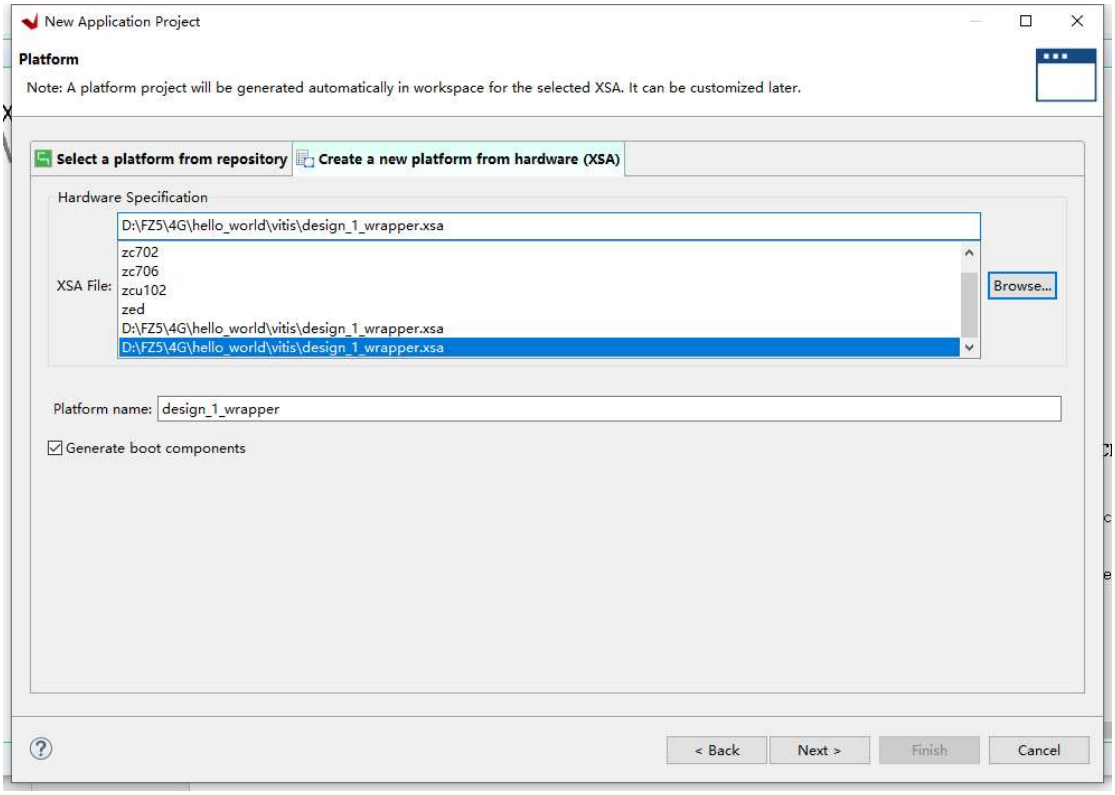


图 3-51 选择 XSA 文件

填入 APP 工程名称 hello_world，在方框处点击可以选择对应的处理器，我们这里保持默认（注意，app 工程名与 platform name 不能相同，否则会出错）

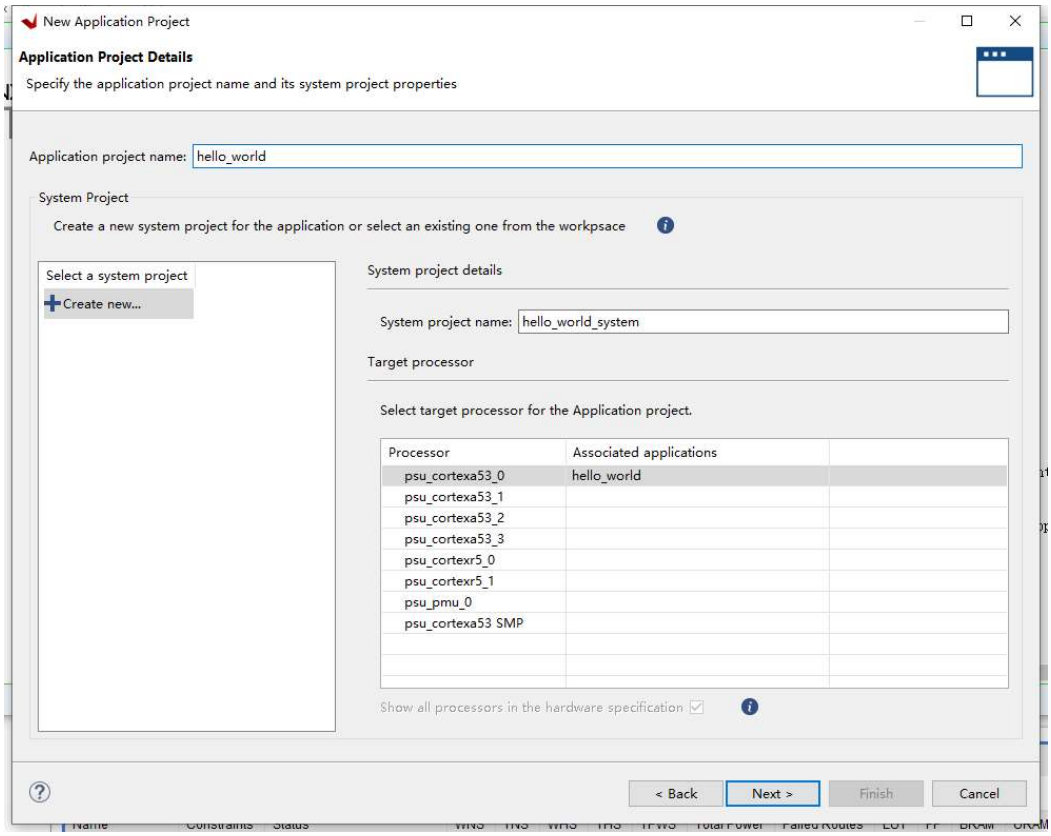


图 3-52 vitis 的 application 名称

在这个界面可以修改 Domain 名称，选择操作系统， ARM 架构等，这里保持默认，操作系统选择 standalone，也就是裸机。

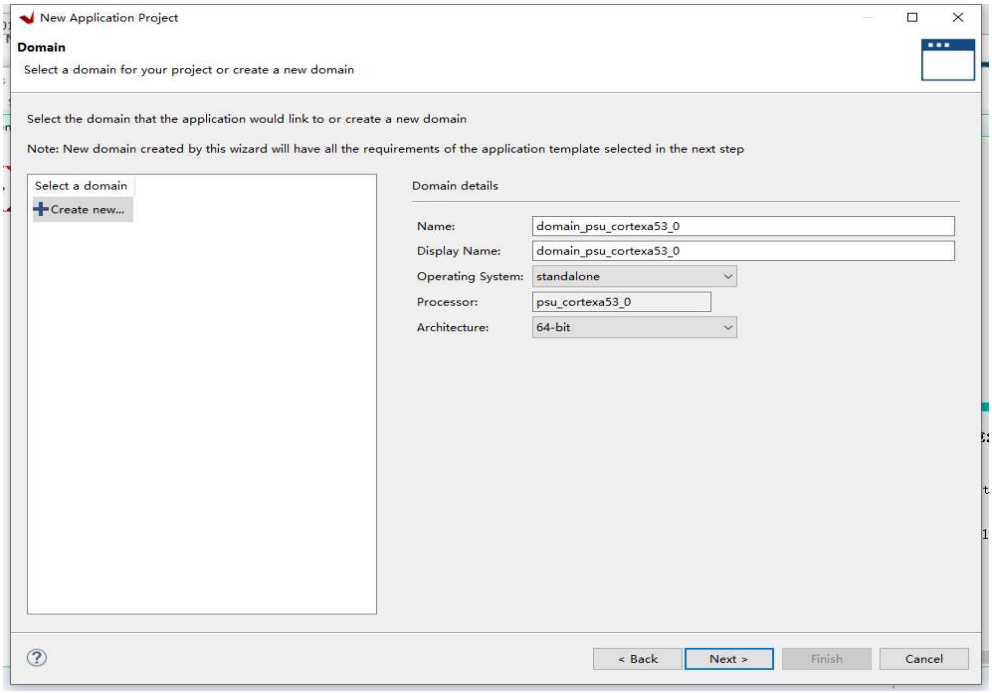


图 3-53 选择对应的操作系统处理器

选择”Hellow World”模板，点击“Finish”完成

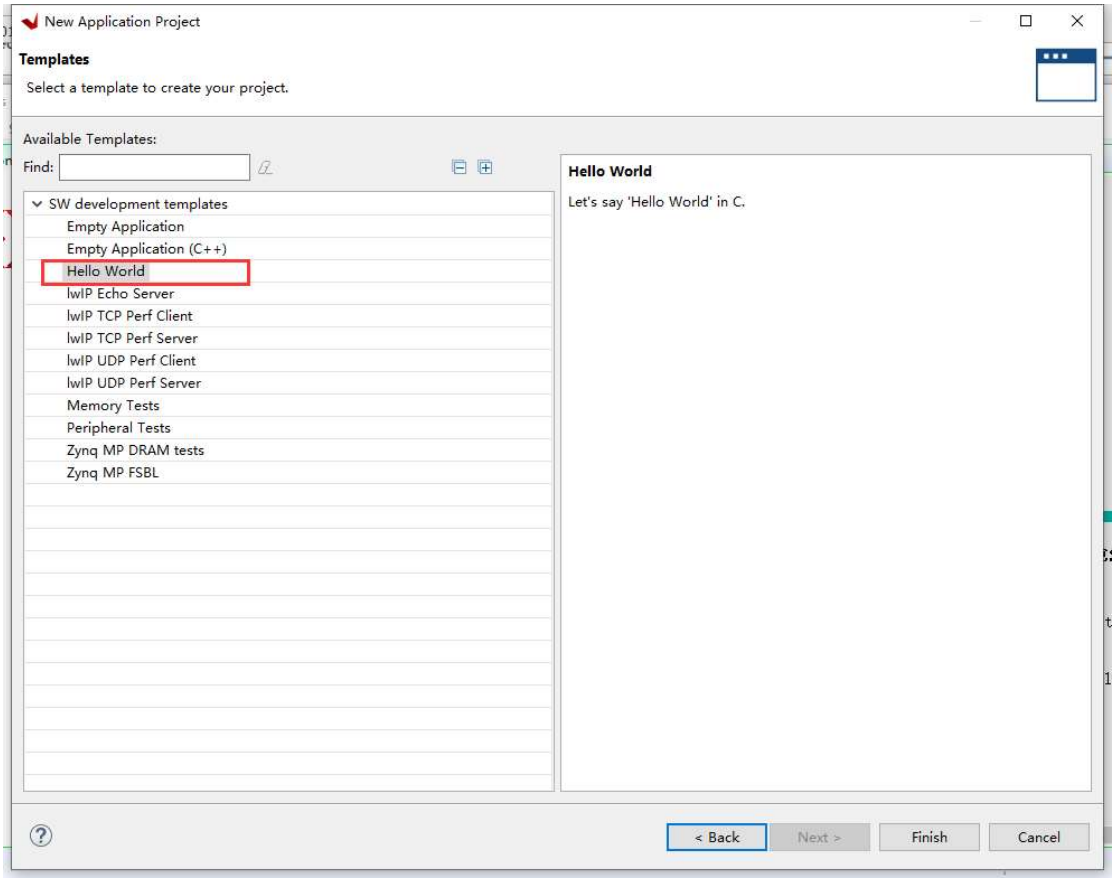


图 3-54 选择 application 模板工程

完成之后可以看到生成了两个工程，一个是硬件平台工程，即之前所说的 Platfrom 工程，一个是 APP 工程

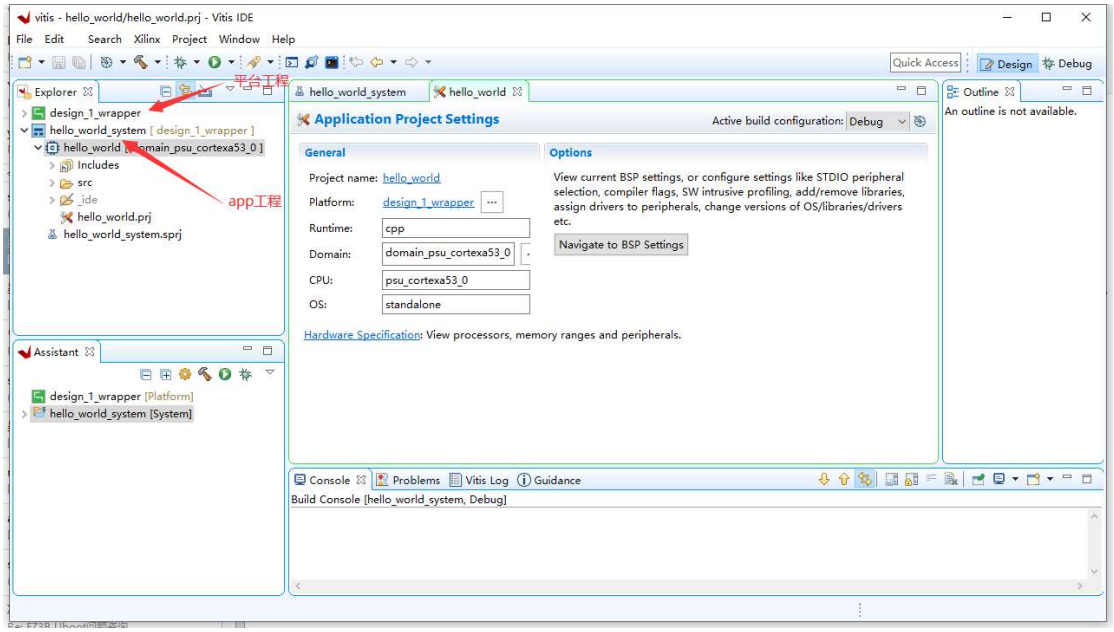


图 3-55 application 工程界面

展开 Platform 工程后可以看到里面包含有 BSP 工程，以及 zynq_fsbl 工程（此工程即选择 Generate boot components 之后的结果），双击 platform.spr 即可看到 Platform 对应生成的 BSP 工程，可以在这里对 BSP 进行配置。 BSP 也就是 Board Support Package 板级支持包的意思，里面包含了开发所需要的驱动文件，用于应用程序开发。 可以看到 Platform 下有多个 BSP，这是跟以往的 SDK 软件不一样的，其中 zynqmp_fsbl 即是 fsbl 的 BSP， domain_psu_cortexa53_0 即是 APP 工程的 BSP。

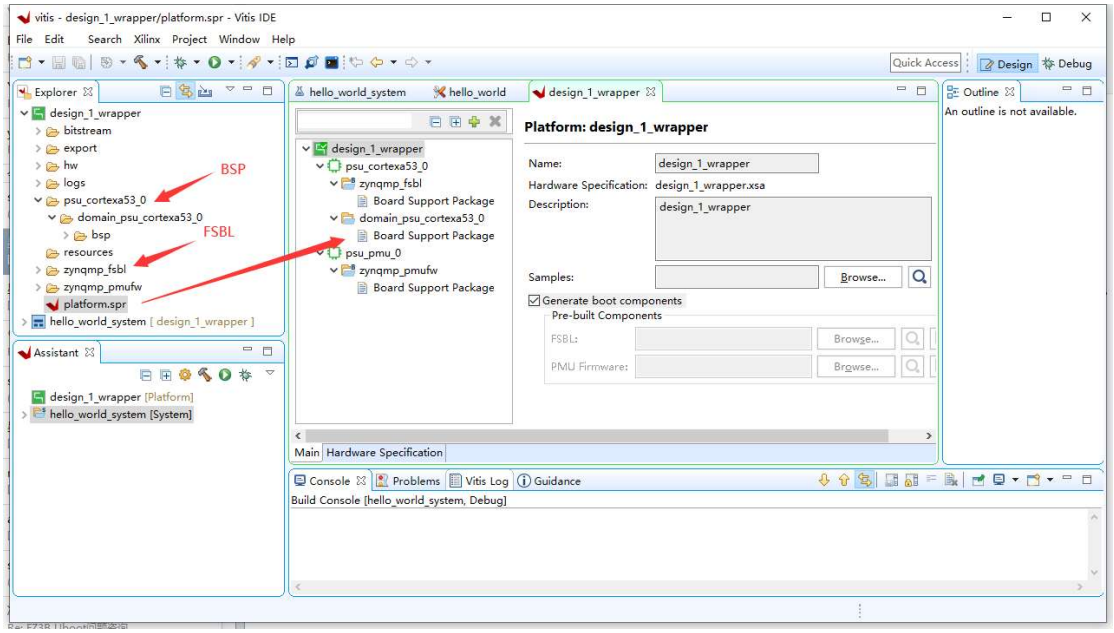


图 3-56 BSP、FSBL 等文件夹

点开 BSP，即可看到工程带有的外设驱动，其中 Documentation 是 xilinx 提供的驱动的说明文档， Import Examples 是 xilinx 提供的 example 工程，加快学习。

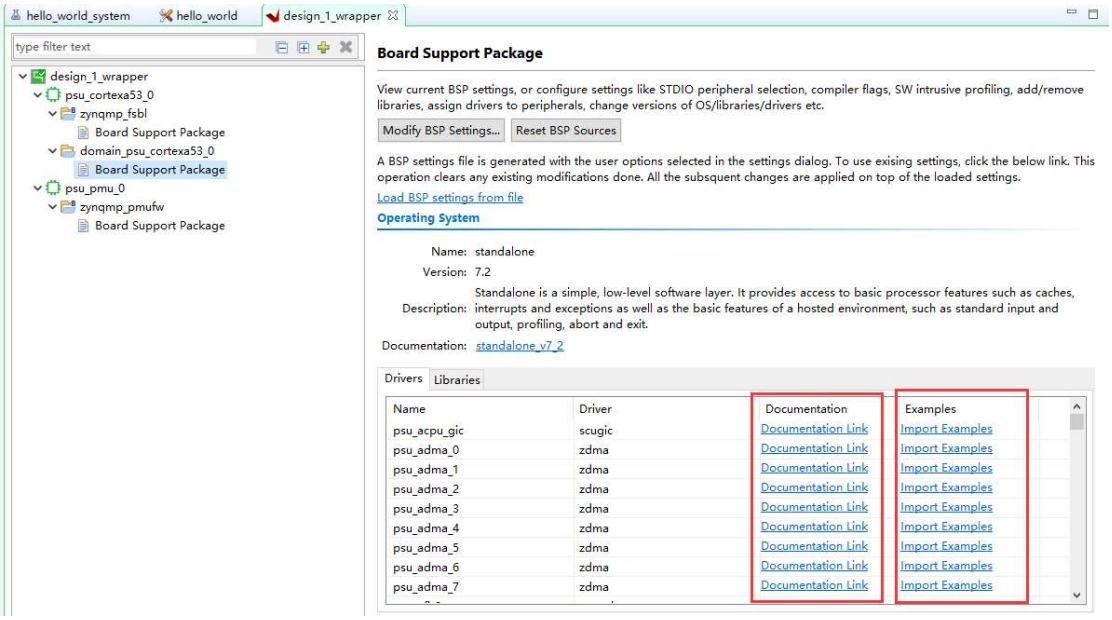


图 3-57 BSP example 示例

选中 APP 工程，右键 Build Project，或者点击菜单栏的“锤子”按键，进行工程编译

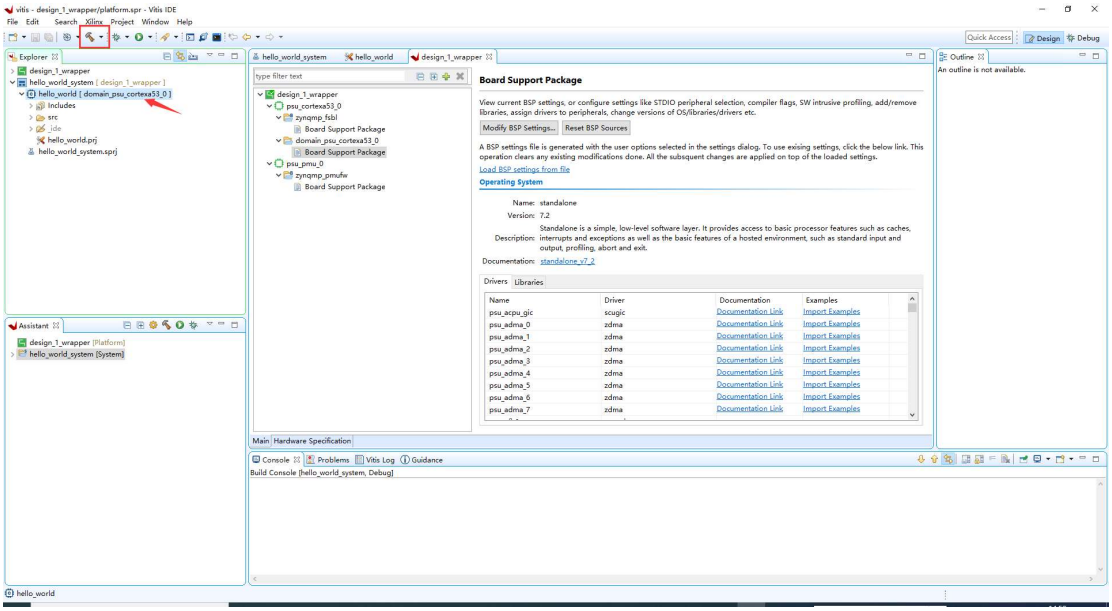


图 3-58 编译

编译结束，生成 elf 文件

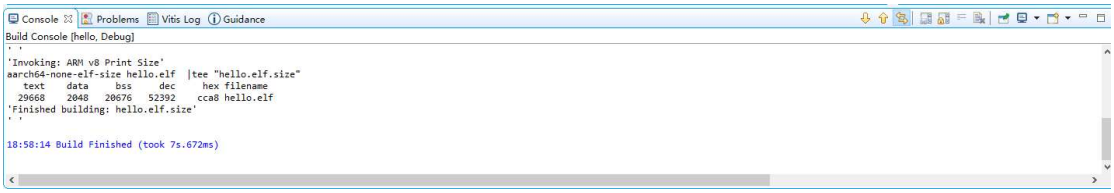


图 3-59 生成 elf 文件 console 窗口

3.1.5 debug

连接 JTAG 线到开发板、 UART 的 USB 线到 PC

使用 PuTTY 软件做为串口终端调试工具

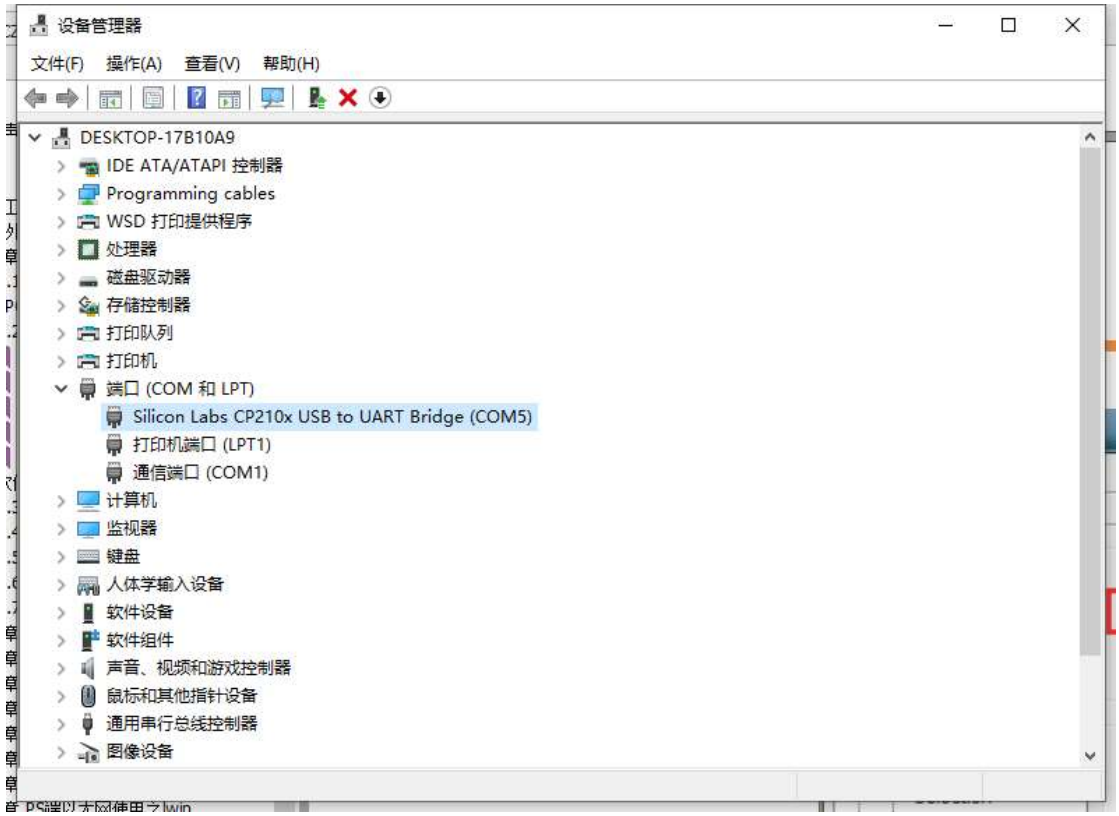


图 3-60 PC 的资源管理器

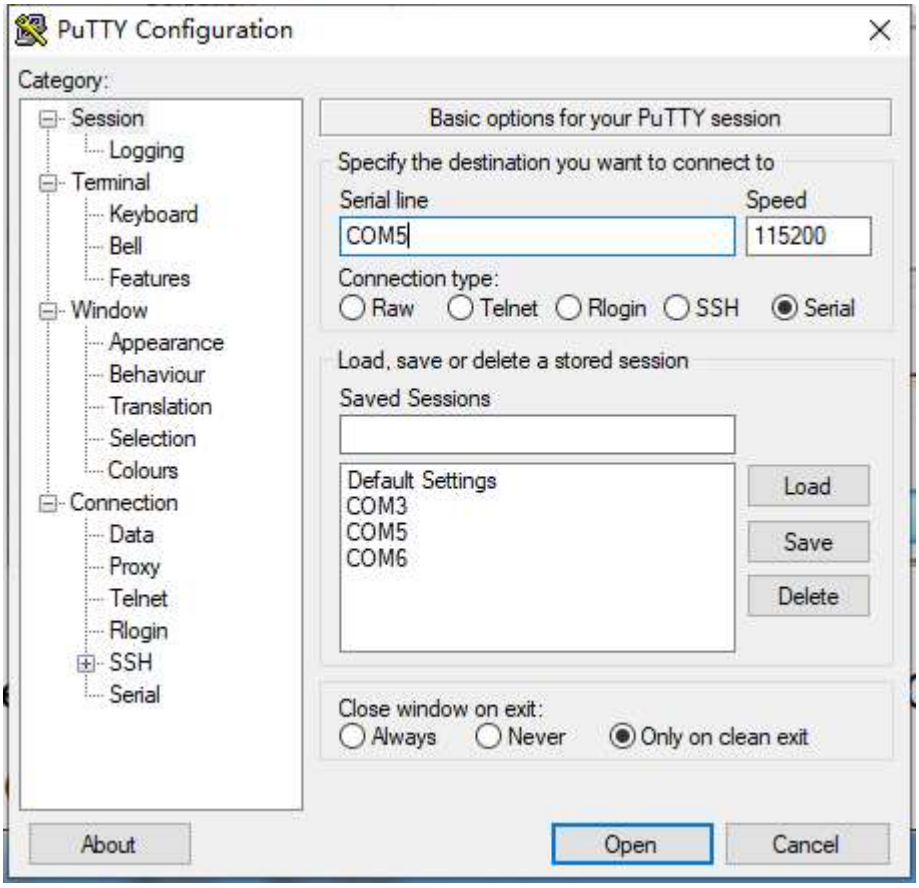


图 3-61 PuTTY 配置

选择“hello”，右键，可以看到很多选项，本实验要用到这里的“Run as”，就是把程序运行起来，“Run as”里又有很多选项，选择第一个“Launch onHardware(Single Application Debug)”，使用系统调试，直接运行程序。

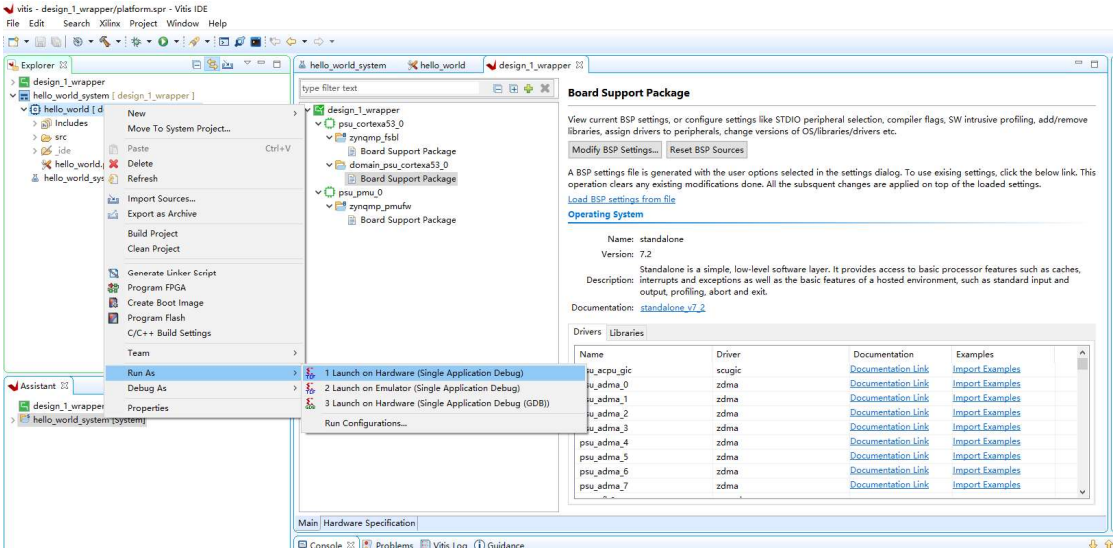


图 3-62 下载调试

为了保证系统的可靠调试， 最好是右键“Run As -> Run Configuration...”

我们可以看一下里面的配置，其中 Reset entire system 是默认选中的，这是跟以前的 SDK 软件不同的。 如果系统中还有 PL 设计，还必须选择“Program FPGA”。

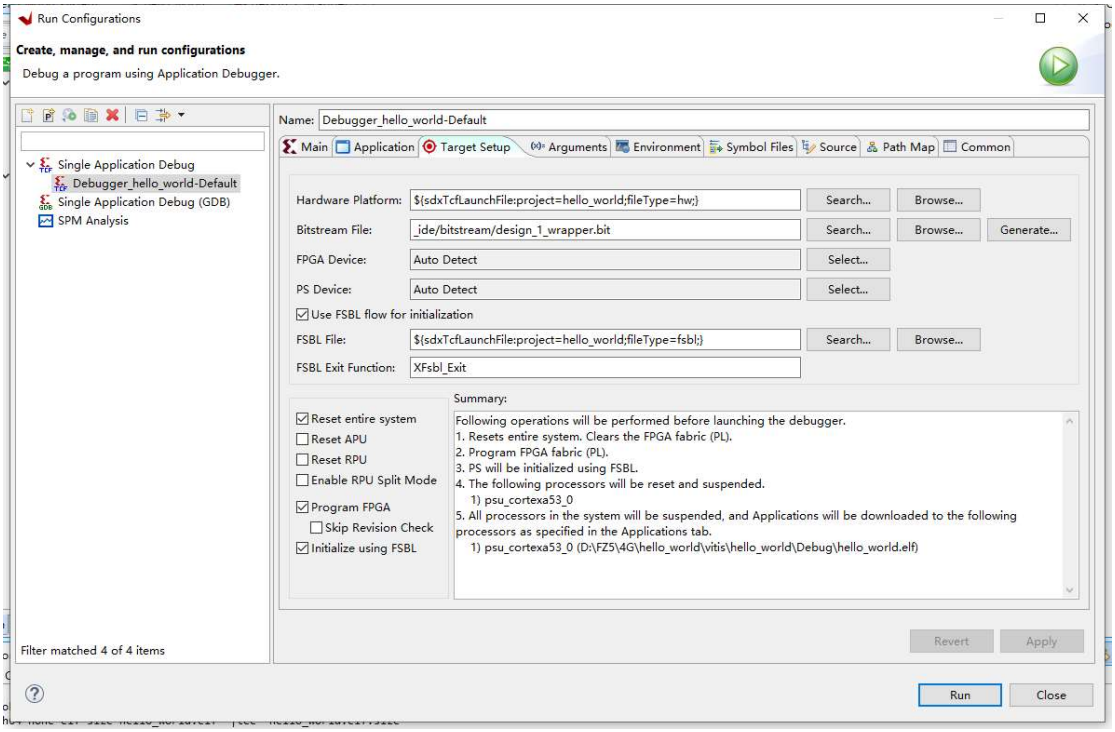


图 3-63 debug 设置

3.1.6 串口打印输出 helloworld

串口输出打印结果如下所示。

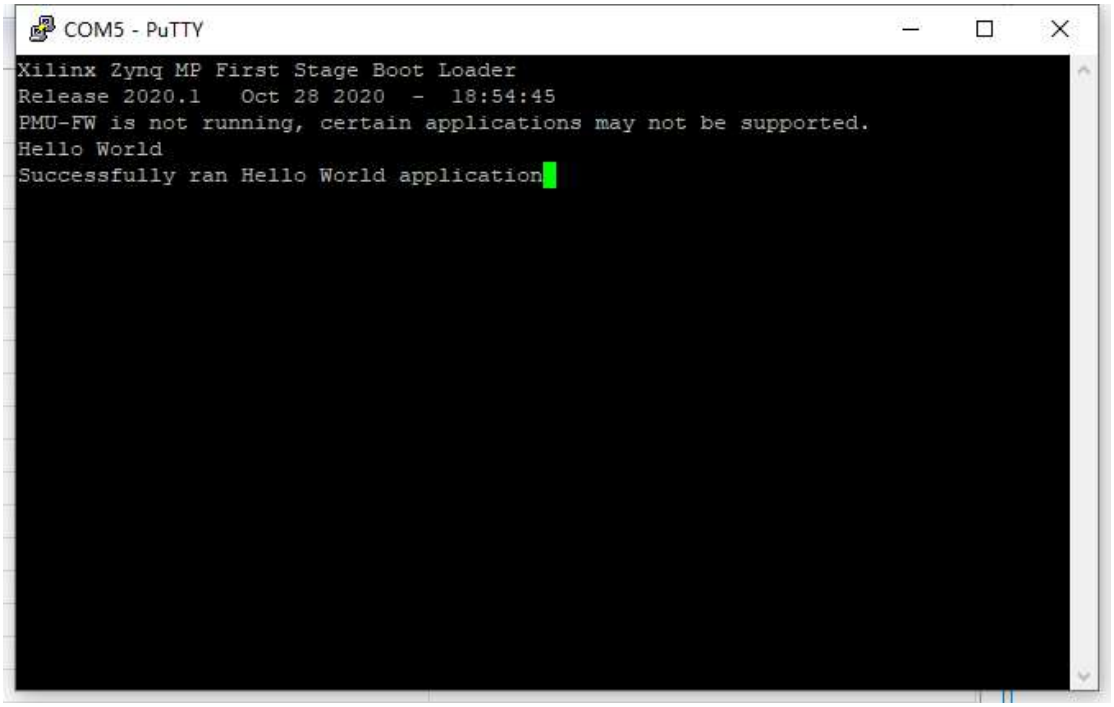


图 3-64 结果显示

3.1.7 程序固化

由于在新建时选择了 Generate boot components 选项，所以 Platform 已经导入了 fsbl 的工程，并生成了相应的 elf 文件。

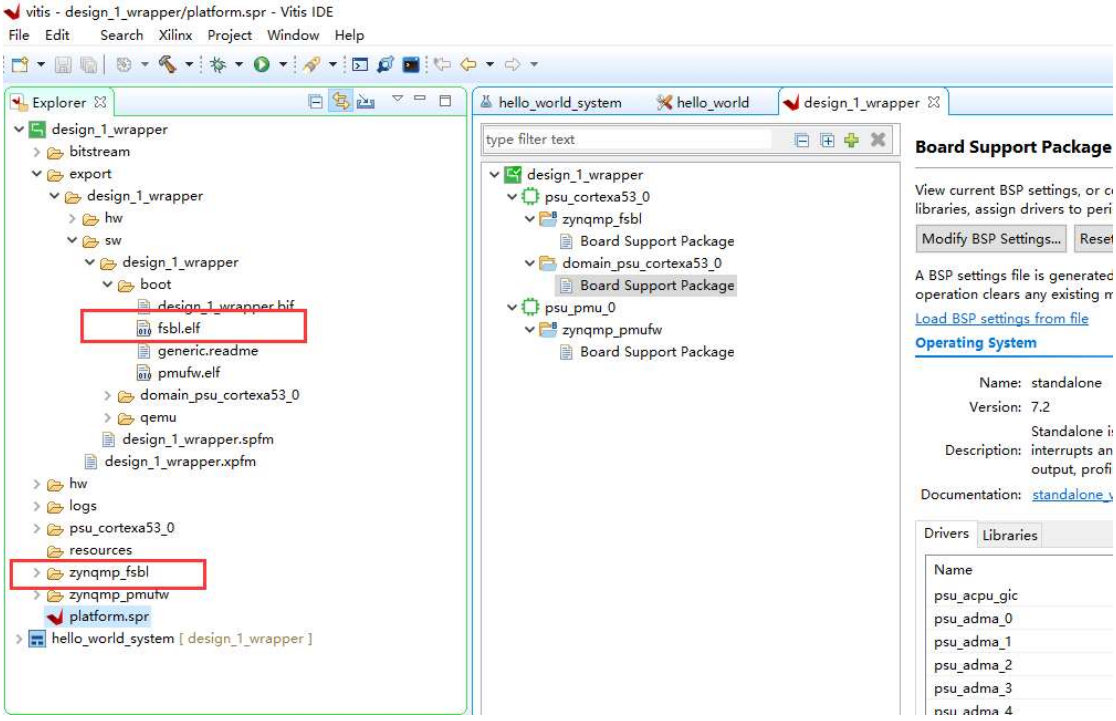


图 3-65 生成 fsbl.elf

修改调试宏定义 FSBL_DEBUG_INFO_VAL 为 1，可以在启动输出 FSBL 的一些状态信息，有利于调试，但是会导致启动时间变长。

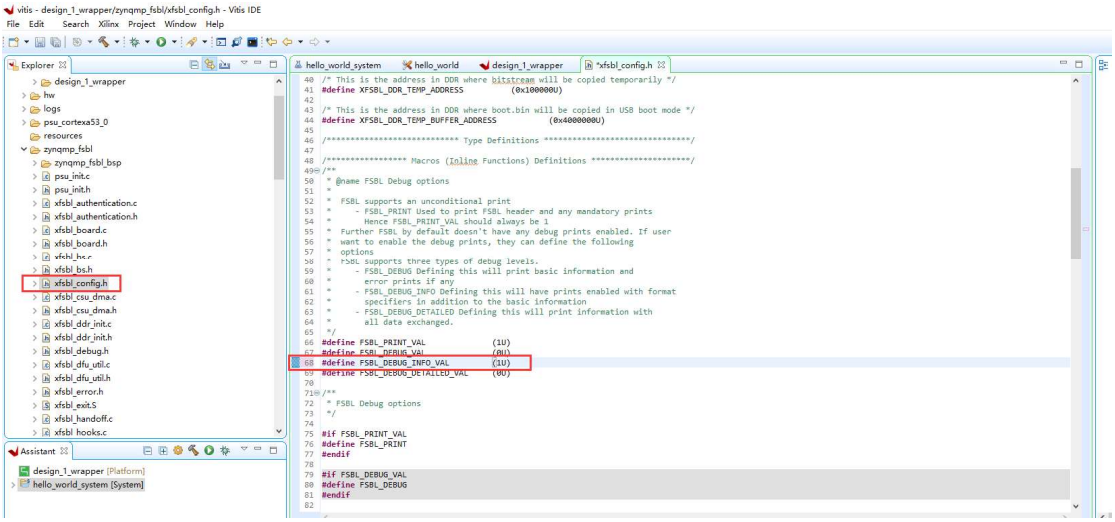


图 3-66 修改参数

右键硬件平台工程 hello_world，选择 Build Project

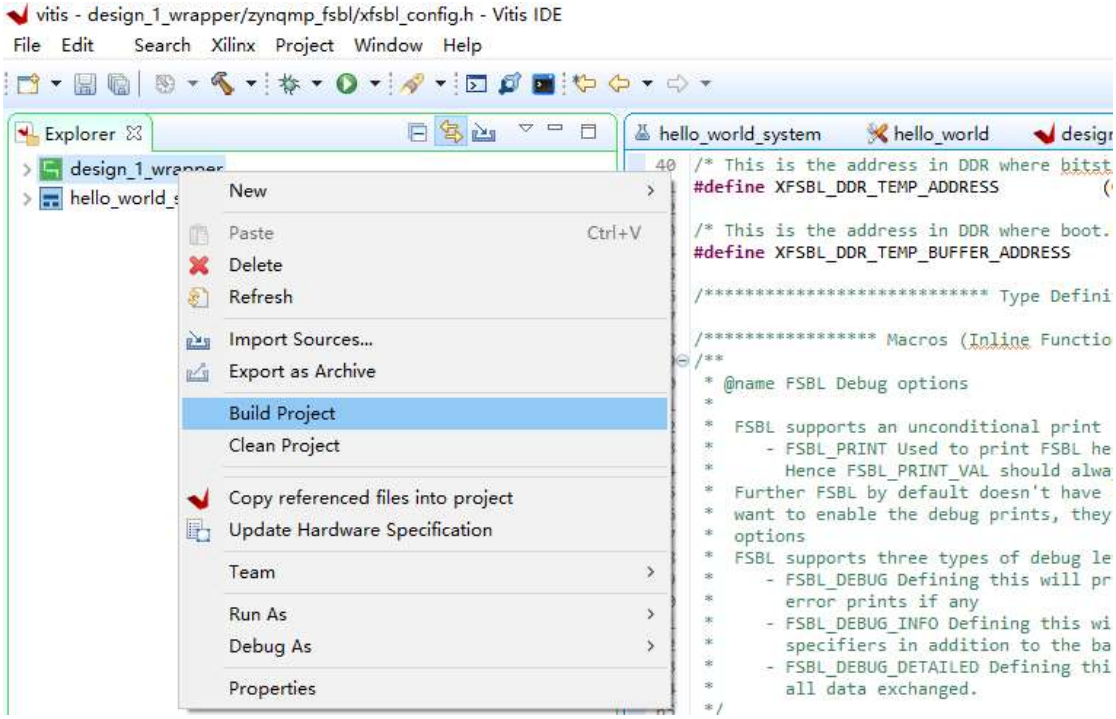


图 3-67 编译

点击 APP 工程的 system，右键选择 Build project

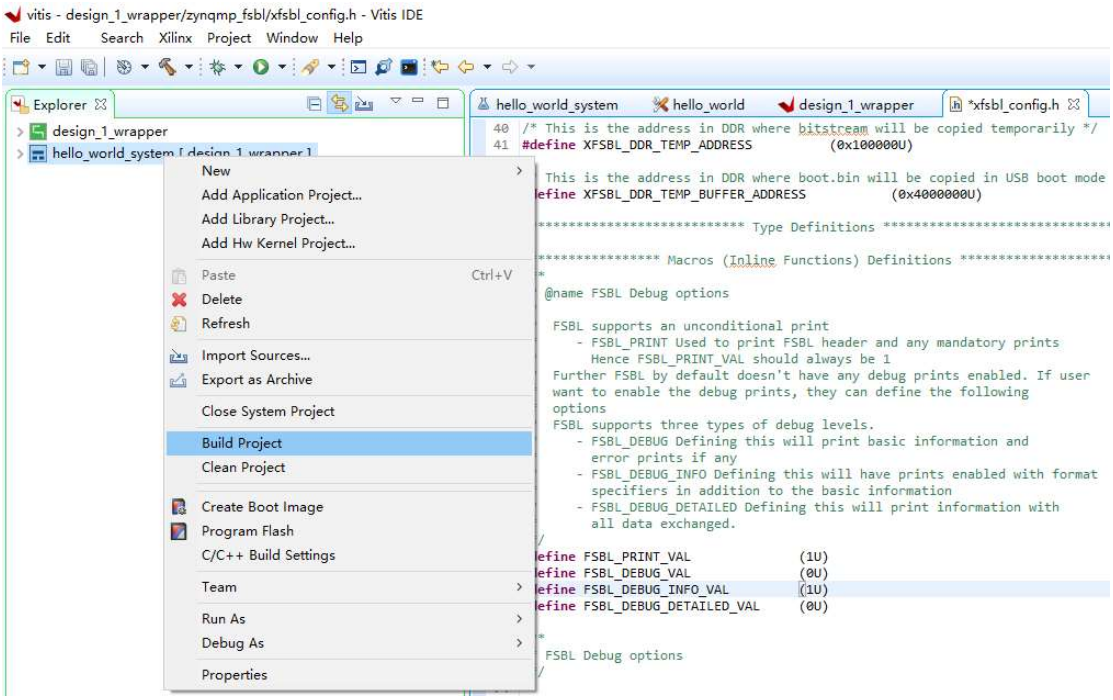


图 3-68 重新编译

这个时候就会多出一个 Debug 文件夹，生成了对应的 BOOT.BIN

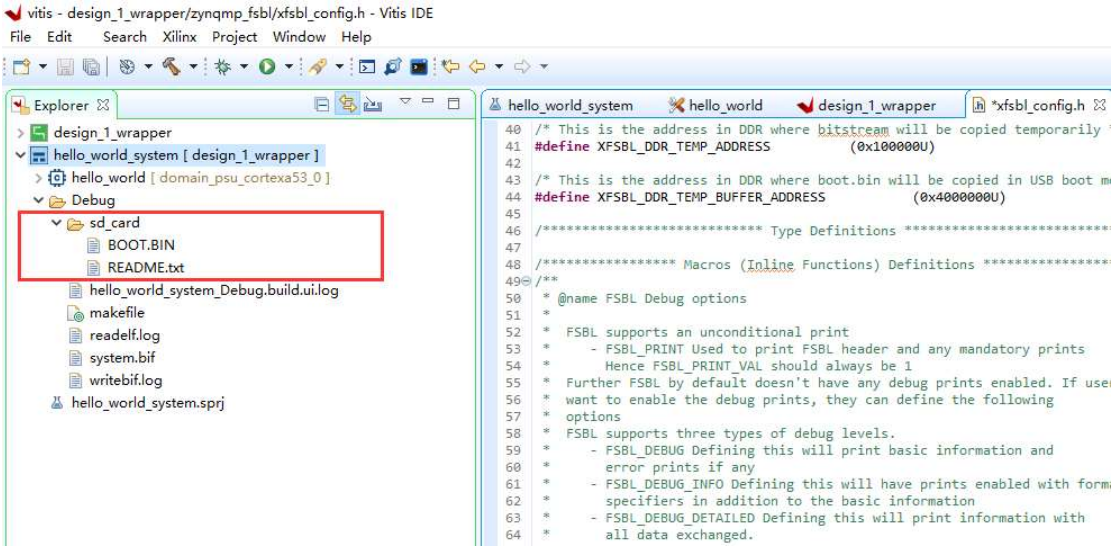


图 3-69 SD 卡启动的 BOOT.BIN

还有一种方法就是，点击 APP 工程的 system 右键选择 Create Boot Image， 弹出的窗口中可以看到生成的 BIF 文件路径， BIF 文件是生成 BOOT 文件的配置文件，还有生成的 BOOT.bin 文件路径， BOOT.bin 文件是我们需要的启动文件，可以放到 SD 卡启动，也可以烧写到 QSPI Flash。

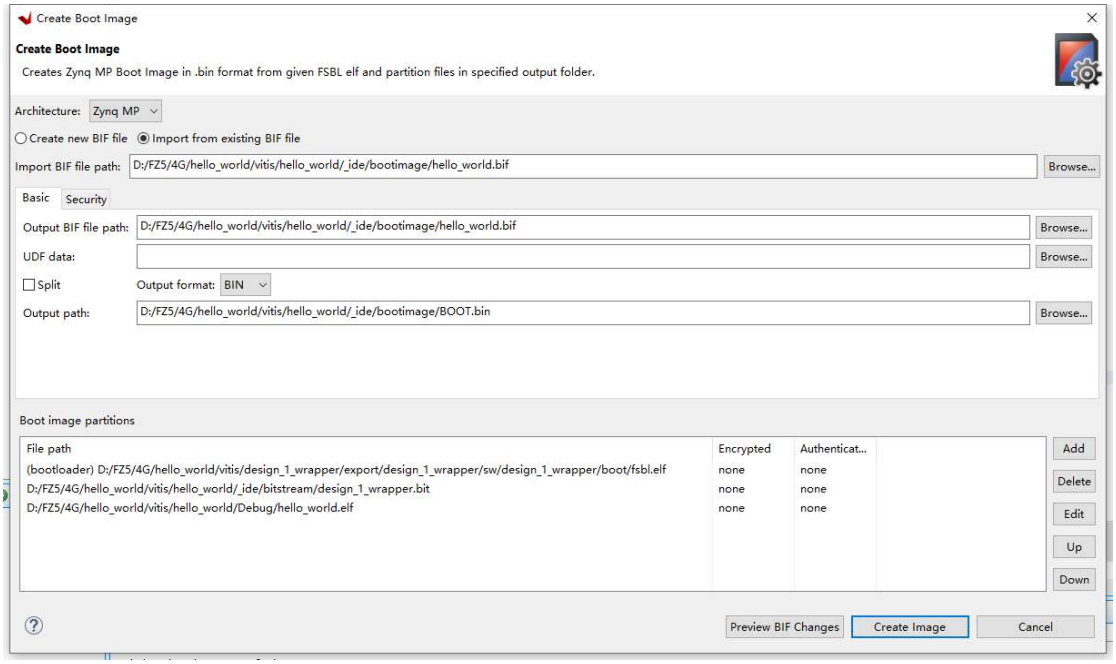


图 3-70 生成 BOOT.bin

在 Boot image partitions 列表中有要合成的文件，第一个文件一定是 bootloader 文件，就是上面生成的 fsbl.elf 文件，第二个文件是 FPGA 配置文件 bitstream，第三个是应用程序，在本实验中为 hello_world.elf。点击 Create Image 生成 BOOT.bin。

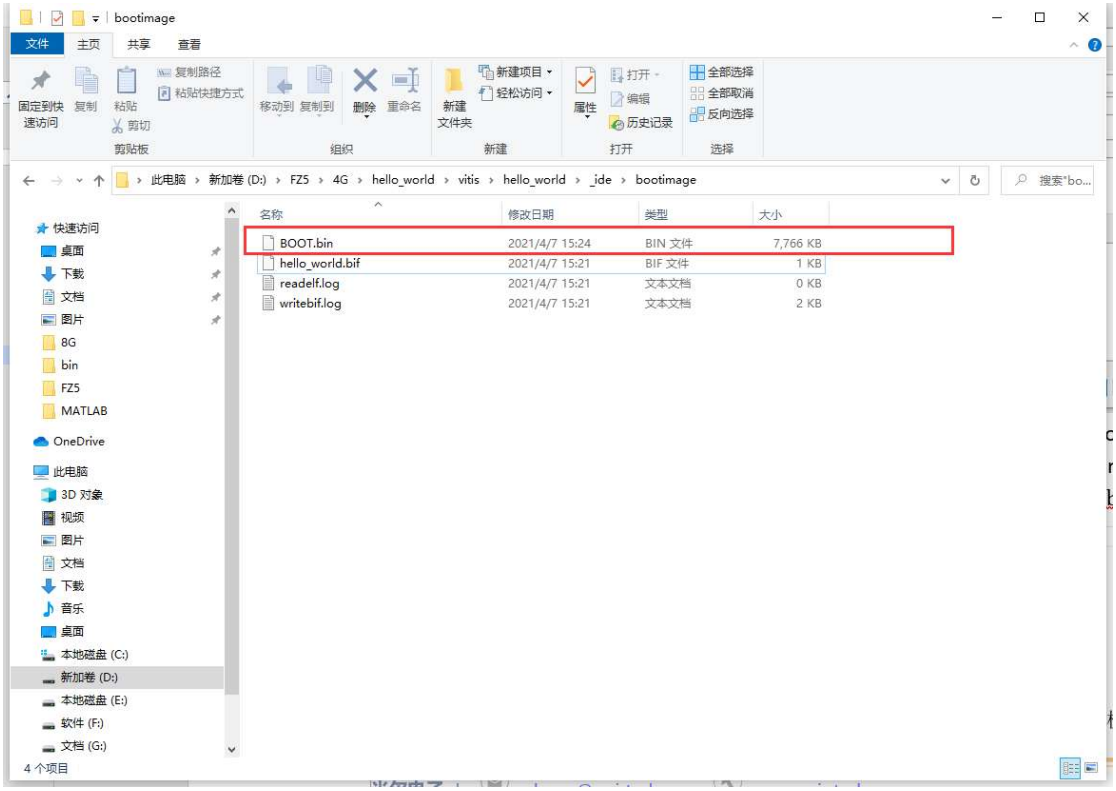


图 3-71 BOOT.bin

将开发板打到 SD 卡启动模式，然后将这个 BOOT.bin 文件拷贝到 SD 卡放到开发板上运行。

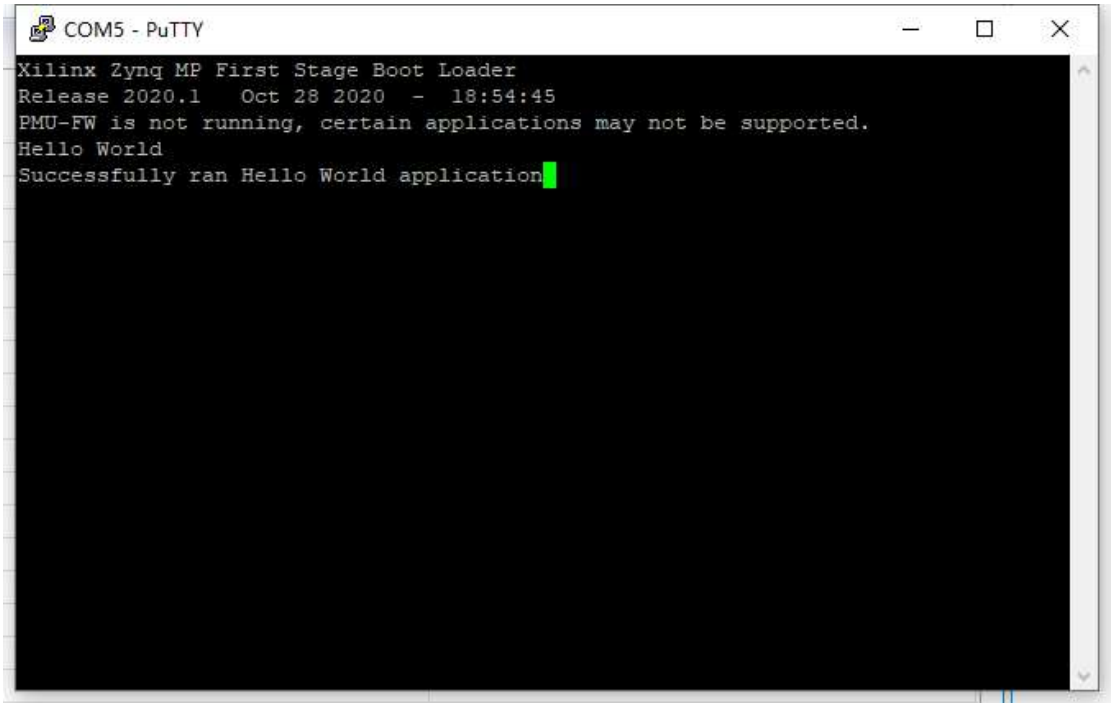


图 3-72 输出结果

3.2 PL 端与 PS 端数据交互方式

PS 与 PL 之间的数据交互方式有直接和间接，直接的方式就是通过 EMIO 的设计，PS 端可以直接通过物理的连线访问 PL 端的寄存器。间接就是通过发送中断和 AXI4 总线接口、总线协议、总线的方式来交互数据。

3.2.1 PL 与 PS 之间直接交互数据

zynq 的 GPIO，分为 2 种，MIO(multiuse I/O)和 EMIO(extendable multiuse I/O)

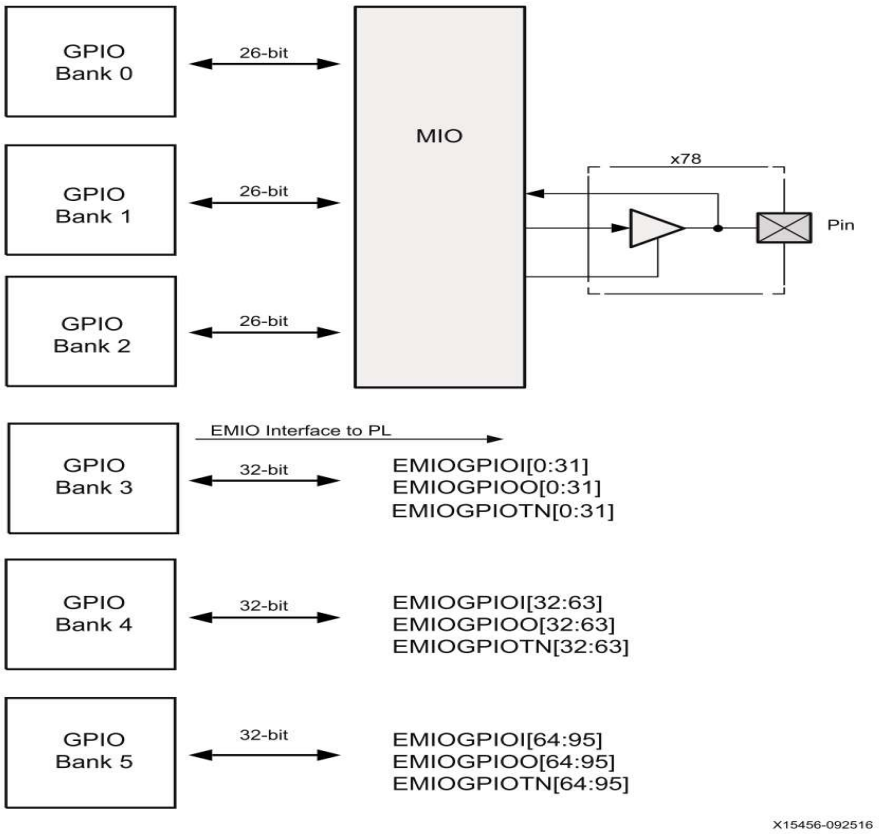


Figure 27-1: **GPIO Block Diagram**

图 3-73 MIO 与 EMIO 结构

BANK3~BANK5 的 EMIO 有 96 个，这个是 PS 端可以直接访问。使用 EMIO 的好处就是，当 MIO 不够用 PS 可以通过 EMIO 控制 PL 部分的引脚。

当然除了直接的 GPIO 接口，在设计满足要求的条件下还有其他的接口方式可以直接通过 PS 访问 PL 端的资源。

3.2.2 PS 与 PL 端之间的可配置总线

在 ZYNQ 芯片内部用硬件实现了 AXI 总线协议，包括 12 个物理接口，分别为 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 接口。

S_AXI_HP{0:3}_FPD 接口：是高性能/带宽的 AXI4 标准的接口，总共有四个，PL 模块作为主设备连接。主要用于 PL 访问 PS 上的存储器（DDR 和 FPDMain Switch）

S_AXI_LPD 接口：高性能端口，连接 PL 到 LPD。低延迟访问 OCM 和 TCM，访问 PS 端 DDR。

S_AXI_HPC{0,1}_FPD 接口：连接 PL 到 FPD，可连接到 CCI，访问 L1 和 L2Cache，由于通过 CCI，访问 DDR 控制器会有较大延迟。

M_AXI_HPM{0,1}_FPD 接口：高性能总线，PS 为 master，连接 FPD 到 PL，可用于 CPU, DMA, PCIe 等从 PS 推送大量数据到 PL。

M_AXI_HPM0_LPD 接口：低延迟接口总线，PS 为 master，连接 LPD 到 PL，可直接访问 PL 端的 BRAM，DDR 等，也经常用于配置 PL 端的寄存器。

只有 M_AXI_HPM{0,1}_FPD 和 M_AXI_HPM0_LPD 是 Master Port，即主机接口，其余都是 Slave Port（从机接口）。主机接口具有发起读写的权限，ARM 可以利用两个主机接口主动访问 PL 逻辑，其实就是把 PL 映射到某个地址，读写 PL 寄存器如同在读写自己的存储器。其余从机接口就属于被动接口，接受来自 PL 的读写，逆来顺受。在 PS 与 PL 互联应用，用的最多的接口为 S_AXI_HP{0:3}_FPD、M_AXI_HPM{0,1}_FPD 和 M_AXI_HPM0_LPD。

位于 PS 端的 ARM 直接有硬件支持 AXI 接口，而 PL 则需要使用逻辑实现相应的 AXI 协议。Xilinx 在 Vivado 开发环境里提供现成 IP 如 AXI-DMA，AXI-GPIO，AXI-Dataover, AXI-Stream 都实现了相应的接口，使用时直接从 Vivado 的 IP 列表中添加即可实现相应的功能。下图为 Vivado 下的各种 DMA IP：

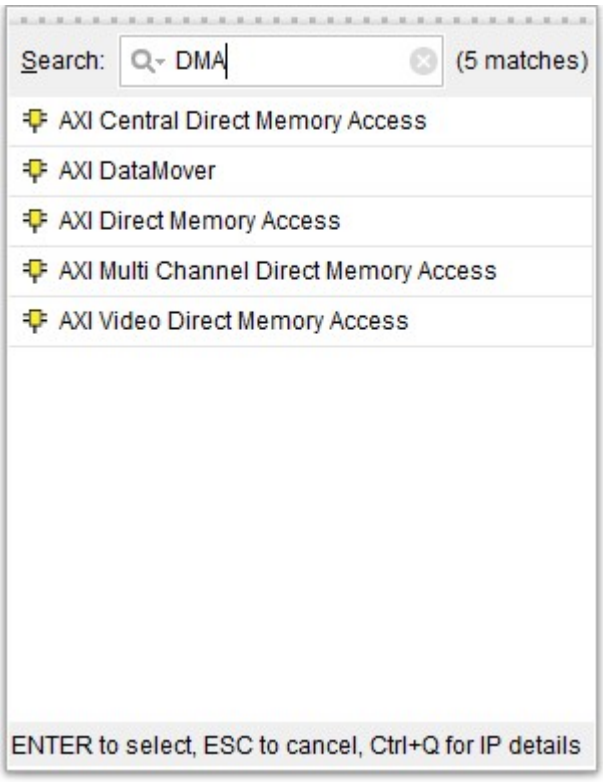


图 3-74 选择 DMA IP

下面为几个常用的 AXI 接口 IP 的功能介绍：

AXI-DMA：实现从 PS 内存到 PL 高速传输高速通道 AXI-HP<---->AXI-Stream 的转换

AXI-FIFO-MM2S：实现从 PS 内存到 PL 通用传输通道 AXI-HPM<----->AXI-Stream 的转换

AXI-Datamover：实现从 PS 内存到 PL 高速传输高速通道 AXI-HP<---->AXI-Stream 的转换，只不过这次是完全由 PL 控制的，PS 是完全被动的。

AXI-VDMA：实现从 PS 内存到 PL 高速传输高速通道 AXI-HP<---->AXI-Stream 的转换，只不过是专门针对视频、图像等二维数据的。

AXI-CDMA：这个是由 PL 完成的将数据从内存的一个位置搬移到另一个位置，无需 CPU 来插手。

，我们会在后面的章节中举例讲到。有时，用户需要开发自己定义的 IP 同 PS 进行通信，这时可以利用向导生成对应的 IP。用户自定义 IP 核可以拥有 AXI4-Lite，AXI4，AXI-Stream，PLB 和 FSL 这些接口。后两种由于 ARM 这一端不支持，所以不用。

有了上面的这些官方 IP 和向导生成的自定义 IP，用户其实不需要对 AXI 时序了解太多（除非确实遇到问题），因为 Xilinx 已经将和 AXI 时序有关的细节都封装起来，用户只需要关注自己的逻辑实现即可。

AXI 协议严格的讲是一个点对点的主从接口协议，当多个外设需要互相交互数据时，我们需要加入一个 AXI Interconnect 模块，也就是 AXI 互联矩阵，作用是提供将一个或多个 AXI 主设备连接到一个或多个 AXI 从设备的一种交换机制（有点类似于交换机里面的交换矩阵）。这个 AXI Interconnect IP 核最多可以支持 16 个主设备、 16 个从设备，如果需要更多的接口，可以多加入几个 IP 核。

AXI Interconnect 基本连接模式有以下几种：
N-to-1 Interconnect
to-N Interconnect
N-to-M Interconnect (Crossbar Mode)
N-to-M Interconnect (Shared Access Mode)

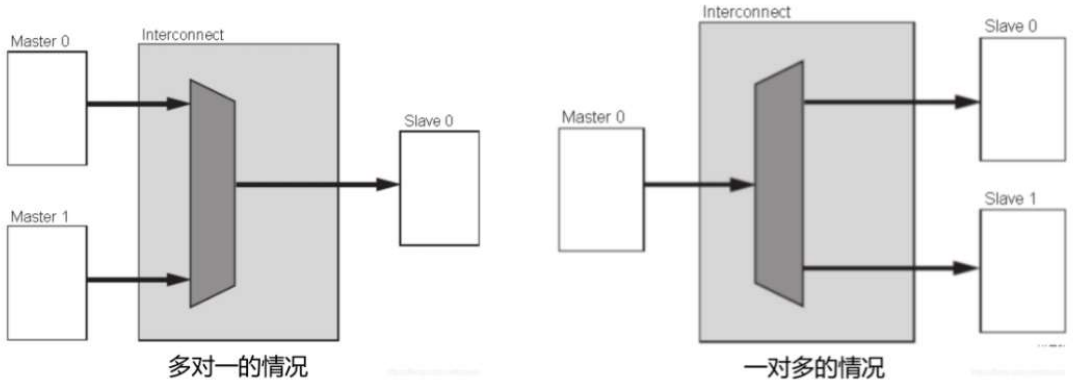


图 3-75 AXI interconnect

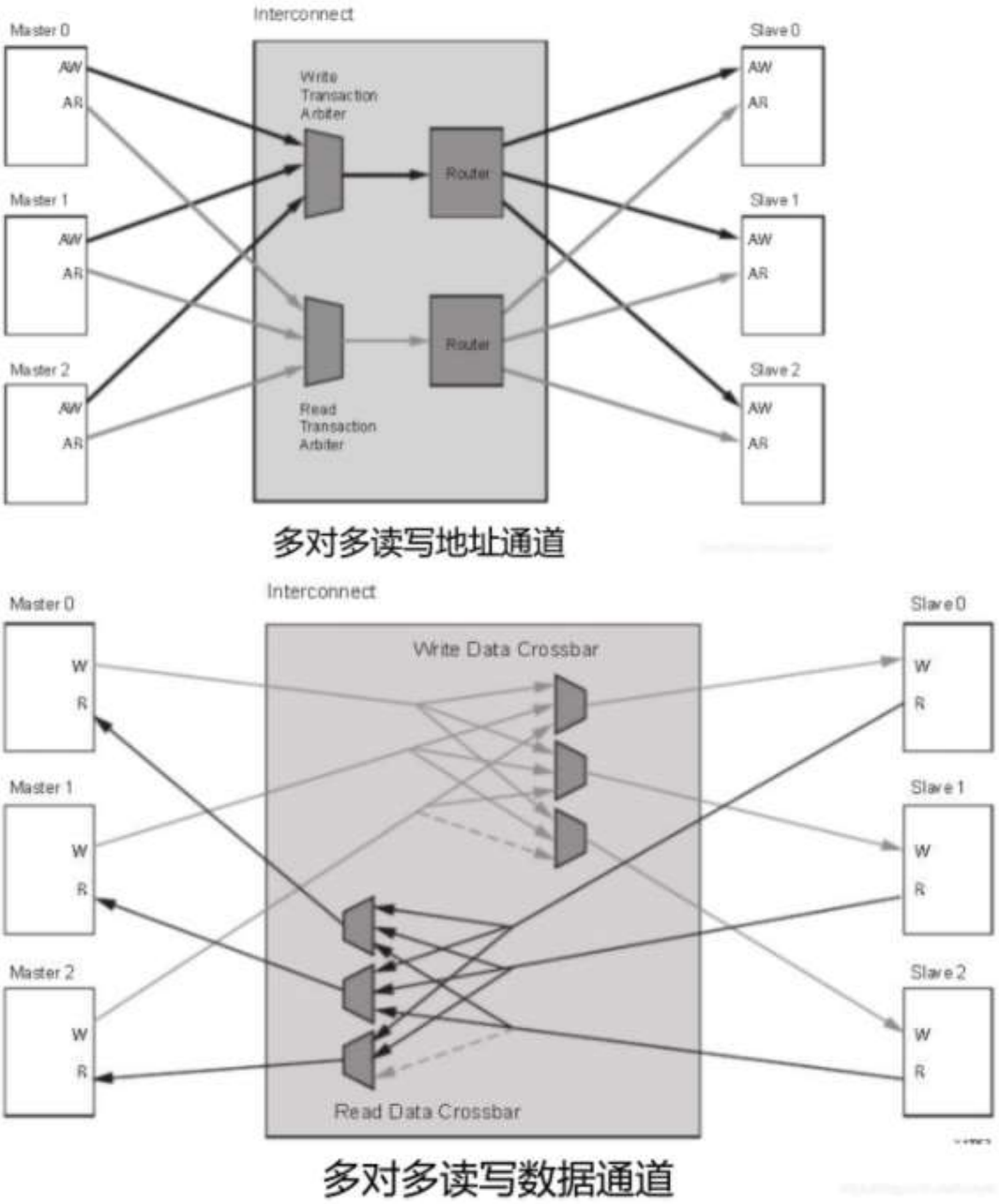


图 3-76 AXI interconnect 结构

ZYNQ 内部的 AXI 接口设备就是通过互联矩阵的的方式互联起来的，既保证了传输数据的高效性，又保证了连接的灵活性。 Xilinx 在 Vivado 里我们提供了实现这种互联矩阵的 IP 核 axi_interconnect，我们只要调用就可以。

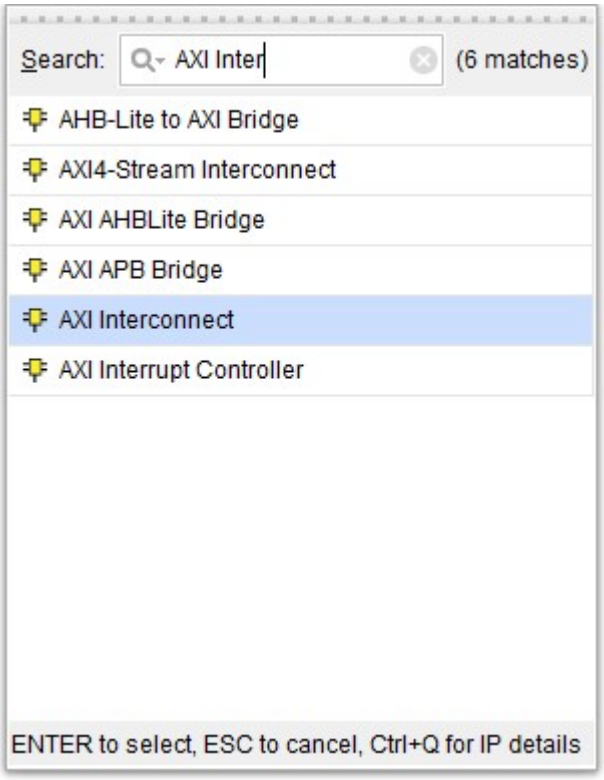


图 3-77 AXI interconnect 选择 IP

3.3 AXI4 总线介绍

3.3.1 AXI4 协议

自 XILINX 针对 7 系列 FPGA、SOC 推出 VIVADO 开发环境后，使得 FPGA 开发更加趋于使用现有 IP 核进行工程搭建和验证，减少代码编写的工作量，尤其是在 ZYNQ 的使用中体现的更为明显。

VIVADO 开发环境中几乎所有的 IP 核都支持 AXI 总线，IP 核接口得以标准化。FPGA 工程师只需要学习好 AXI 总线，几乎就掌握了所有 IP 核接口的使用方法。这样的标准化也使得可以更快和更方便的搭建系统进行验证，这一点相对于 ISE 是巨大的进步；

某些偏向定制的功能，VIVADO 开发环境中并没有定义好的 IP 核，这时可能需要 FPGA 工程师自研，如何将自研逻辑快速的融入到系统中，使用 VIVADO 的设计理念，就是对自研逻辑进行标准化，使用 AXI 总线作为接口，同时 VIVADO 也为用户提供了相应的工具。

AXI （Advanced eXtensible Interface） 本是由 ARM 公司提出的一种总线协议， Xilinx 从 6 系列的 FPGA 开始对 AXI 总线提供支持，目前使用 AXI4 版本。

（1）AXI4：（For high-performance memory-mapped requirements.）主要面向高性能地址映射通信的需求，是面向地址映射的接口，允许最大 256 轮的数据突发传输；

（2）AXI4-Lite：（For simple, low-throughput memory-mapped communication ）是一个轻量级的地址映射单次传输接口，占用很少的逻辑单元。

（3）AXI4-Stream：（For high-speed streaming data.）面向高速流数据传输；去掉了地址项，允许无限制的数据突发传输模式。

AXI4 总线和 AXI4-Lite 总线具有相同的组成部分：

（1）读地址通道，包含 ARVALID, ARADDR, ARREADY 信号；

（2）读数据通道，包含 RVALID, RDATA, RREADY, RRESP 信号；

（3）写地址通道，包含 AWVALID，AWADDR, AWREADY 信号；

（4）写数据通道，包含 WVALID, WDATA，WSTRB, WREADY 信号；

（5）写应答通道，包含 BVALID, BRESP, BREADY 信号；

（6）系统通道，包含：ACLK，ARESETN 信号。

而 AXI4-Stream 总线的组成有：

（1）ACLK 信号：总线时钟，上升沿有效；

（2）ARESETN 信号：总线复位，低电平有效；

（3）TREADY 信号：从机告诉主机做好传输准备；

（4）TDATA 信号：数据，可选宽度 32,64,128,256bit

（5）TSTRB 信号：每一 bit 对应 TDATA 的一个有效字节，宽度为 TDATA/8；

（6）TLAST 信号：主机告诉从机该次传输为突发传输的结尾；

（7）TVALID 信号：主机告诉从机数据本次传输有效；

（8）TUSER 信号 ：用户定义信号，宽度为 128/8bit。

	地址通道		数据通道	
读通道	ARVALID	读地址有效。此信号表明该信道此时能有效读出地址和控制信息	RVALID	读数据有效。此信号表明该信道此时能有效读出数据
	ARADDR	读地址	RDATA	读数据
	ARREADY	读地址准备好了。该信号指示从器件准备好接受一个地址和相关联的控制信号	RREADY	读数据准备好了。该信号指示从器件准备好接收数据
	ARPROT	保护类型。这个信号表示该事务的特权和安全级别，并确定是否该事务是一个数据存取或指令的访问	RRESP	读取响应。这个信号表明读事务处理的状态。

写通道	AWVALID	写地址有效。这个信号表示该主信令有效的写地址和控制信息。	WVALID	写有效。这个信号表示有效的写数据和选通信号都可用。	BVALID	写响应有效。此信号表明写命令的有效写入响应。
	AWADDR	写地址	WDATA	写数据	BREADY	响应准备。该信号指示在主主机可以接受一个响应信号
	AWREADY	写地址准备好了。该信号指示从器件准备好接受一个地址和相关联的控制信号	WSTRB	写选通。这个信号表明该字节通道持有效数据。每一bit对应WDATA一个字节	BRESP	写响应。这个信号表示写事务处理的状态。
	AWPROT	写通道保护类型。这个信号表示该事务的特权和安全级别，并确定是否该事务是一个数据存取或指令的访问	WREADY	写准备好了。该信号指示从器件可以接受写数据。		

图 3-78 AXI 总线读写通道分析

协议的制定是要建立在总线构成之上的。因此说 AXI4，AXI4-Lite，AXI4-Stream 都是 AXI4 协议。AXI 总线协议的两端可以分为分为主（master）、从（slave）两端，他们之间一般需要通过一个 AXI Interconnect 相连接，作用是提供将一个或多个 AXI 主设备连接到一个或多个 AXI 从设备的一种交换机制。

AXI Interconnect 的主要作用是，当存在多个主机以及从机器时，AXI Interconnect 负责将它们联系并管理起来。由于 AXI 支持乱序发送，乱序发送需要主机的 ID 信号支撑，而不同的主机发送的 ID 可能相同，而 AXI Interconnect 解决了这一问题，他会对不同主机的 ID 信号进行处理让 ID 变得唯一。

AXI 协议将读地址通道，读数据通道，写地址通道，写数据通道，写响应通道分开，各自通道都有自己的握手协议。每个通道互不干扰却又彼此依赖。这是 AXI 高效的原因之一。

3.3.2 AXI 握手协议

AXI4 所采用的是一种 READY，VALID 握手通信机制，简单来说主从双方进行数据通信前，有一个握手的过程。传输源产生 VLAID 信号来指明何时数据或控制信息有效。

而目地源产生 READY 信号来指明已经准备好接受数据或控制信息。 传输发生在 VALID 和 READY 信号同时为高的时候。如下图中的实例：

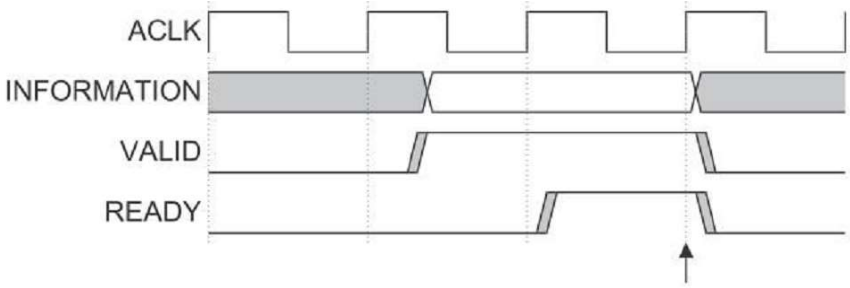


图 3-79 Valid 早于 READY 的握手

当地址出现在地址总线后，传输的数据将出现在读数据通道上。设备保持 VALID 为低直到读数据有效。为了表明一次突发式读写的完成，设备用 RLAST 信号来表示最后一个被传输的数据。

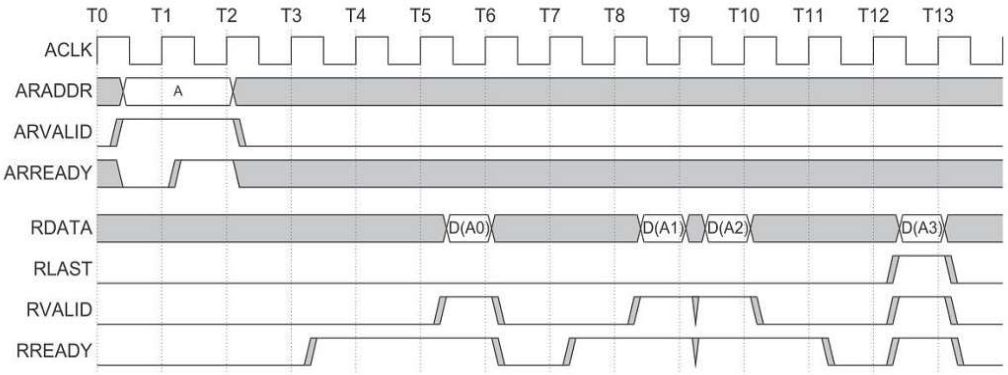


图 3-80 READY 早于 VALID 的握手以及传输

主机发送地址和控制信息到写地址通道中，然后主机发送每一个写数据到写数据通道中。当主机发送最后一个数据时，WLAST 信号就变为高。当设备接收完所有数据之后他将一个写响应发送回主机来表明写事务完成。

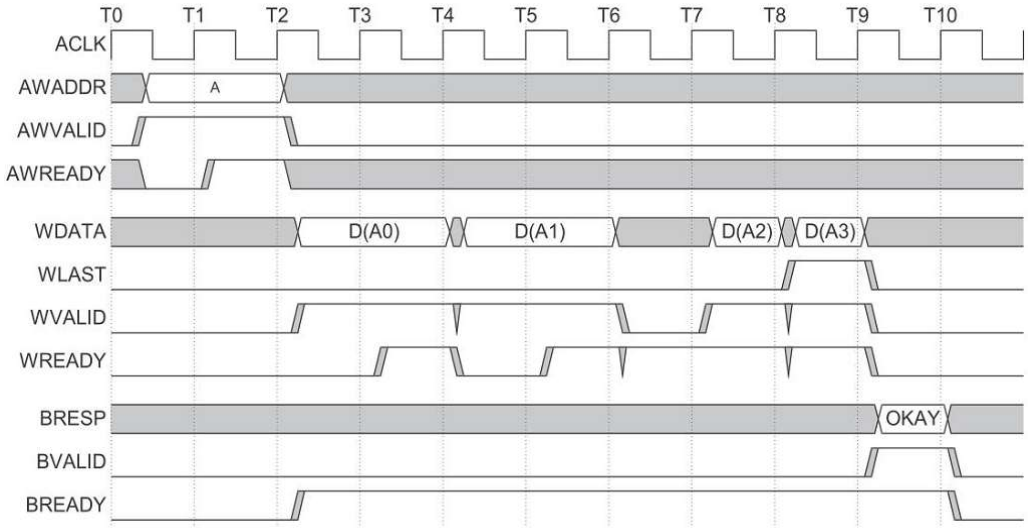


图 3-80 传输数据示例

AXI_LITE 的数据读写时序与 AXI 突发时序相同，只是每次只传输一个数据而已；

AXI4-Stream 的时序：

面向数据流的传输方式，省略的地址通道，其余时序与 AXI 突发时序相同；

3.4 本章小节

本章介绍了硬件平台建立工程的基础配置情况，配置的根据是硬件平台所拥有的软硬件资源。其次稍微介绍了 PS 与 PL 之间的互联技术。更进一步的学习需要参考 ARM 公司的 AMBA 协议 spec，里面详细的描述了 SOC 设计中总线的设计方法协议原理。其次 PS 与 PL 之间的互联模块 AXI Interconnecter 极大的方便了用户对 ARM+FPGA 这个异构平台的使用。

第四章基于 AXI4-Lite 总线的接口设备模块

本章主要是为了理解 PS 端预留出来的与 PL 端连接的 AXI4-Lite 总线接口类型，以及它们的具体使用，注重介绍了 GPIO，AXI UART，AXI IIC 等控制总线和通用 IO 控制接口。

4.1 AXI UART

4.1.1 AXI UART 的基础知识

Xilinx LogiCORE IP AXI 通用输入/输出 (GPIO) 内核为 AXI 接口提供通用输入/输出接口。该 32 位软知识产权 (IP) 内核旨在与 AXI4-Lite 接口连接。具体的特性如下：

支持 AXI4-Lite 接口规范

支持可配置的单或双 GPIO 通道

支持 1 到 32 位 GPIO 引脚的可配置通道宽度

支持将每个 GPIO 位动态编程为输入或输出

支持每个通道的单独配置

支持所有寄存器的每一位独立的复位值

支持可选的中断请求生成

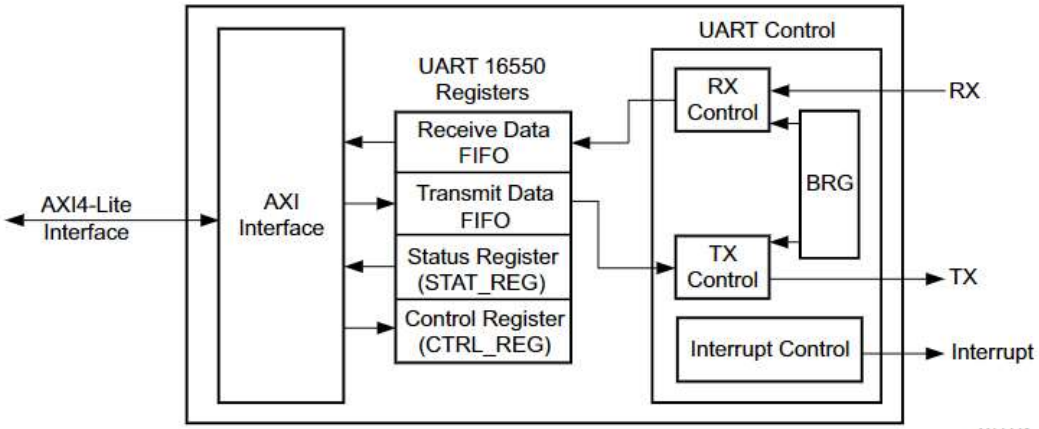


图 4-1 AXI UART 的结构设计框图

功能作用：

AXI GPIO 设计为 AXI4-Lite 接口提供通用输入/输出接口。 AXI GPIO 可以配置为单通道或双通道设备。每个通道的宽度都可以独立配置。通过启用或禁用三态缓冲器，端口可以动态配置为输入或输出。通道可配置为在其任何输入发生转换时生成中断。

通用输入/输出 (GPIO) 内核是一个接口，可以轻松访问设备的内部属性。同样这个内核可以用来控制外部设备的行为。

更进一步的信息可以参考 pg144-LogiCORE IP AXI GPIO V2.0 Product Guide(AXI).pdf

4.1.2 实验逻辑

我们想用这个 AXI UART 实现 ps 通过 axi 总线控制 pl 端的 uartIP 核，输出串口信号，通过硬件上的芯片进行电平转换，然后输出到端口，端口用对应的串口线连接到电脑上，原理图所示：

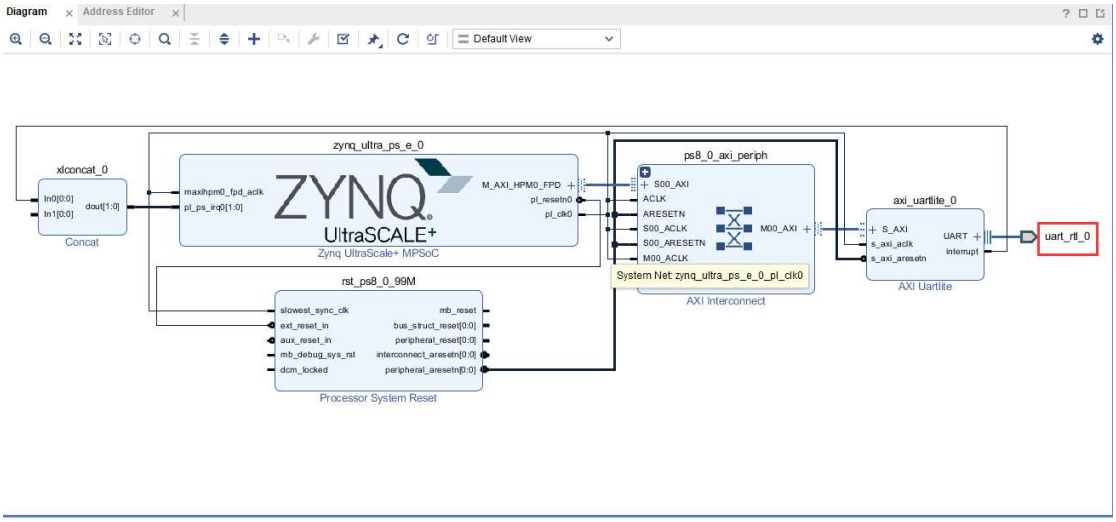


图 4-2 工程设计框图

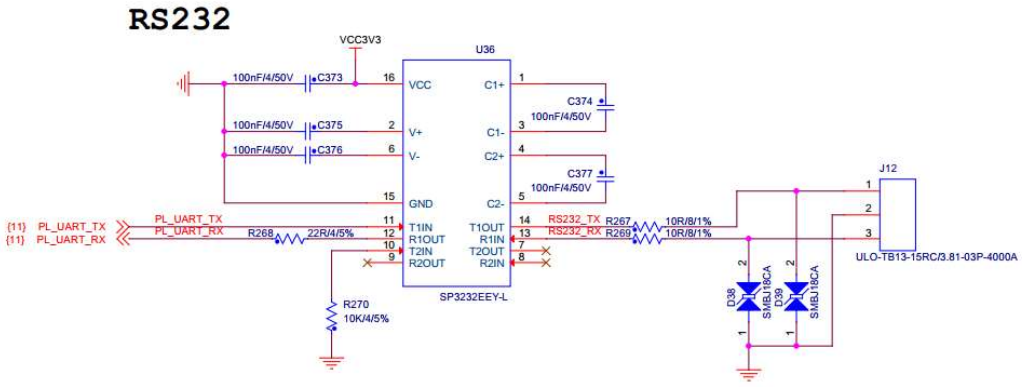


图 4-3 原理图设计

4.1.3 实验步骤

Vivado 工程：

新建立 vivado 工程，命名为 axi_uart,PS 端的基本配置按照”hello_world”工程中的配置参数一样。对应光盘工程文档为 axi_uartlite.rar。

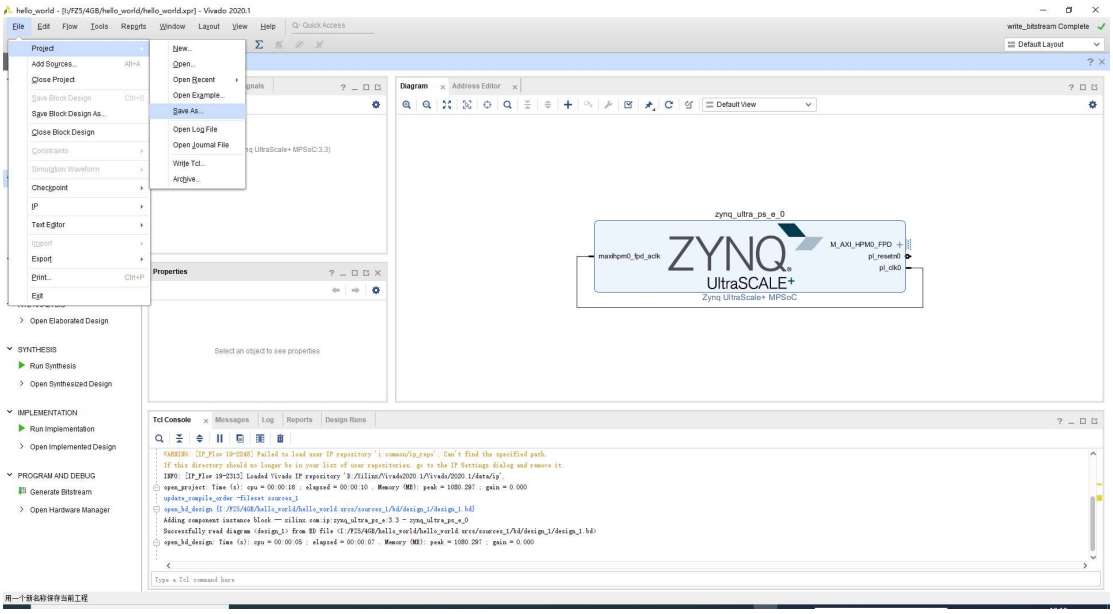


图 4-4 Vivado 设计

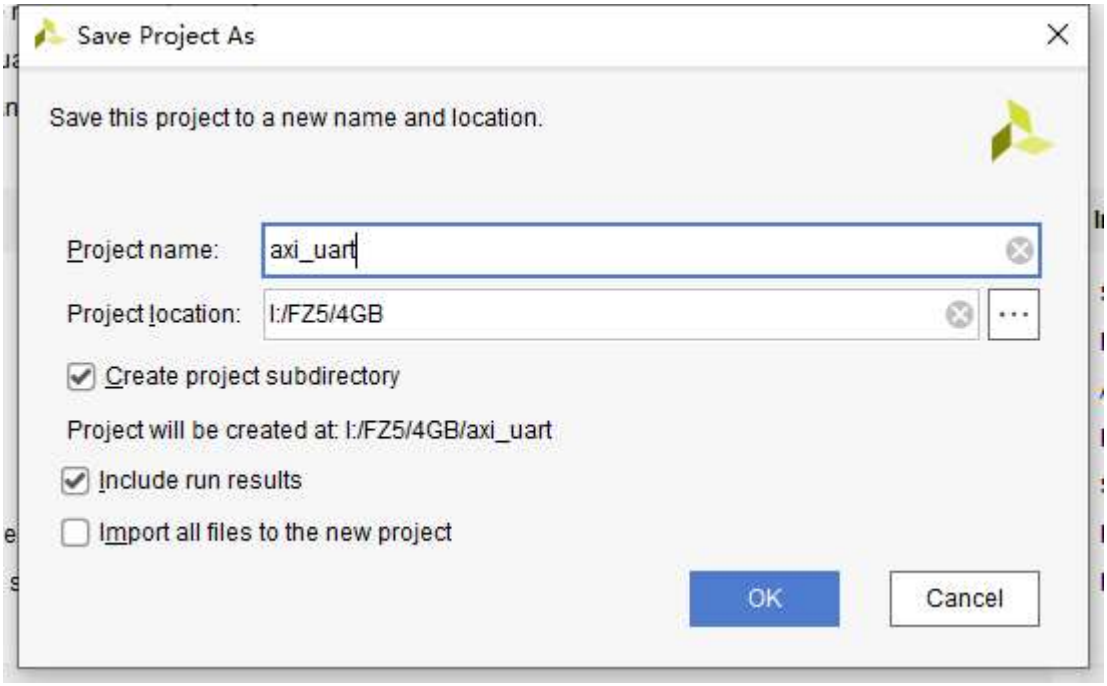


图 4-5 另存工程的名称

配置 PL 中断。

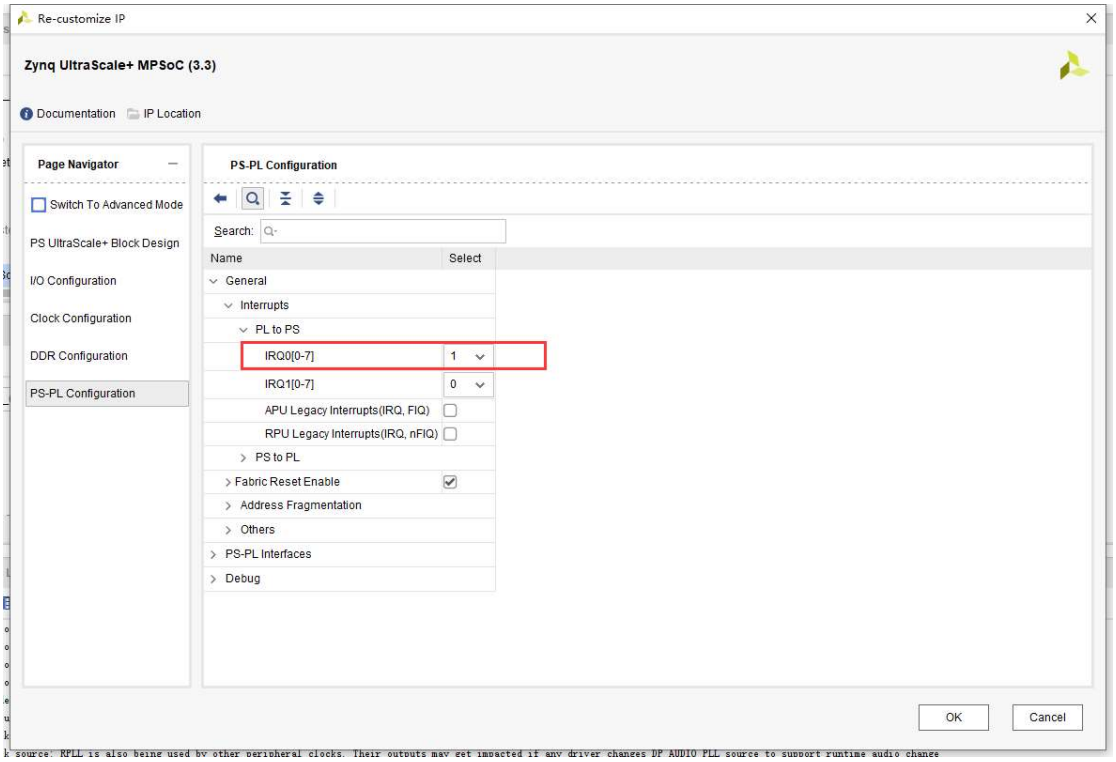


图 4-6 添加配置

添加一个 AXI Uart IP 核，这里配置直接选择默认的。

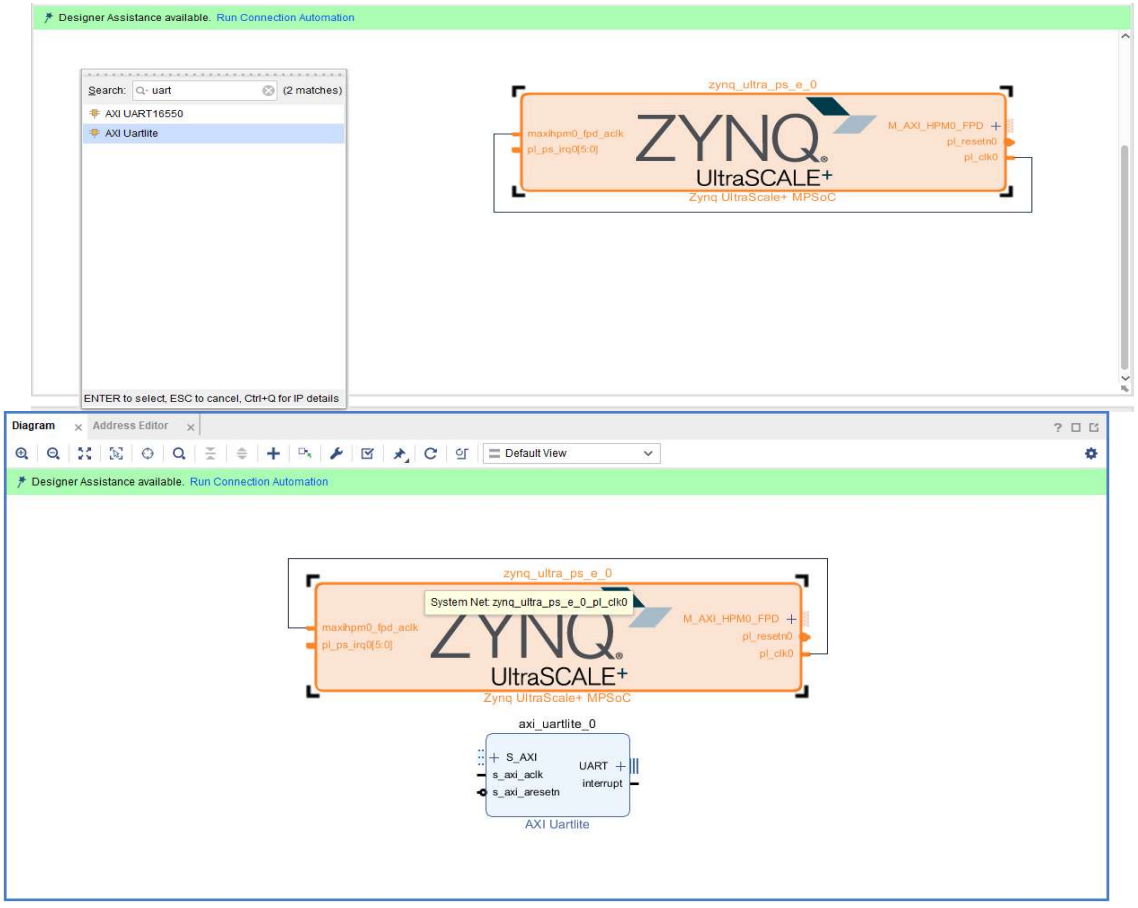


图 4-7 添加串口 IP

添加 concat IP 核，连接中断。

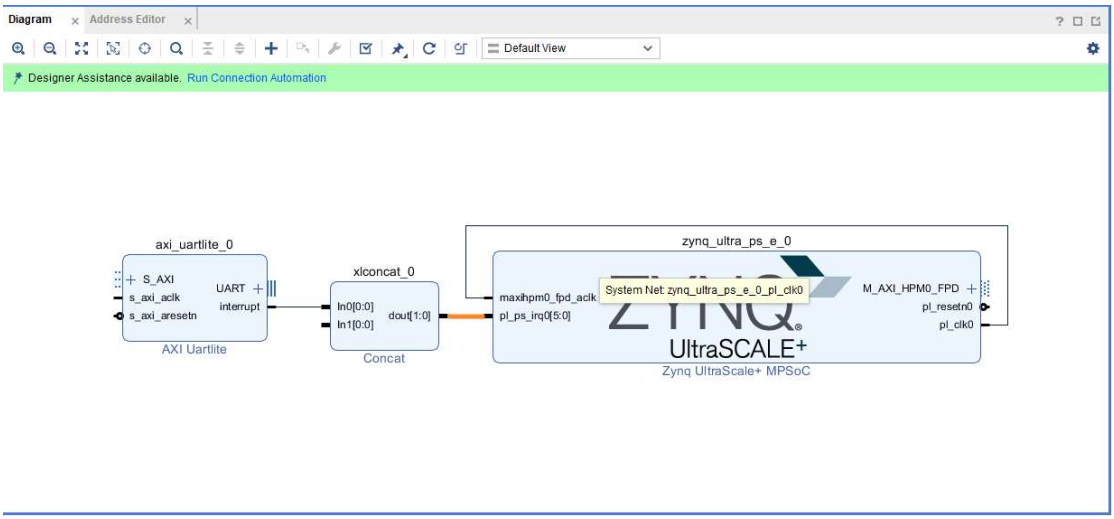


图 4-8 添加 concat 连接线

点击顶部的 Run Connection Automation-->OK 进行自动连线

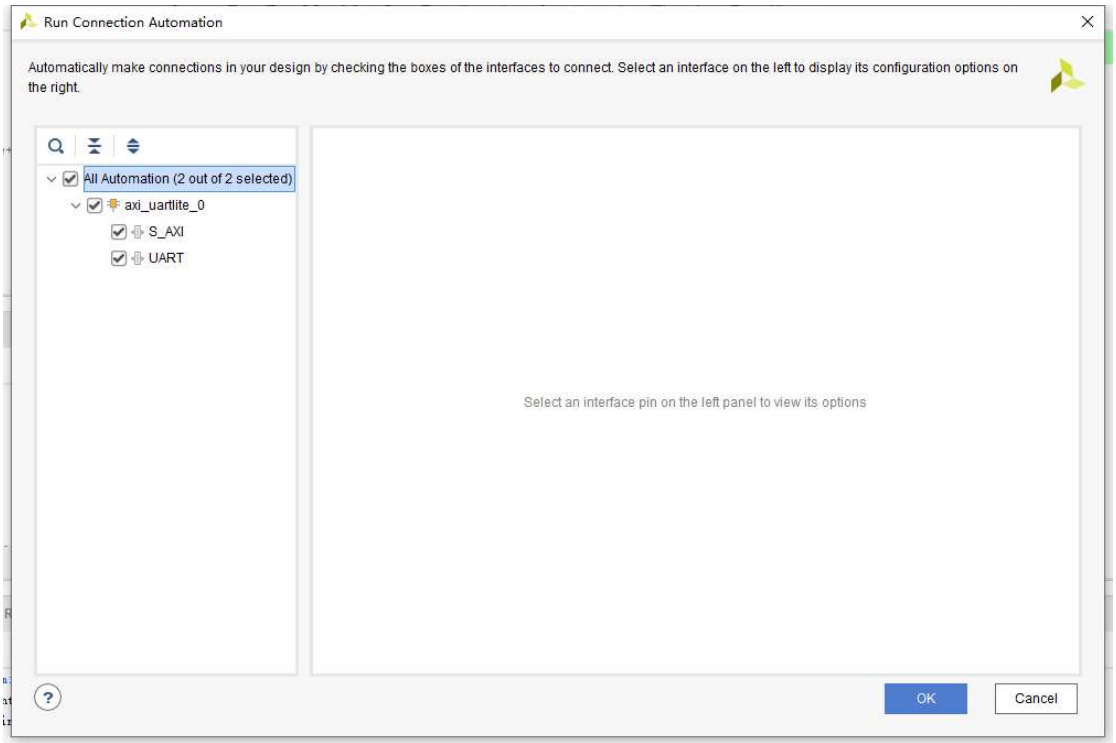


图 4-9 自动连线

自动连线完成后，如下图所示

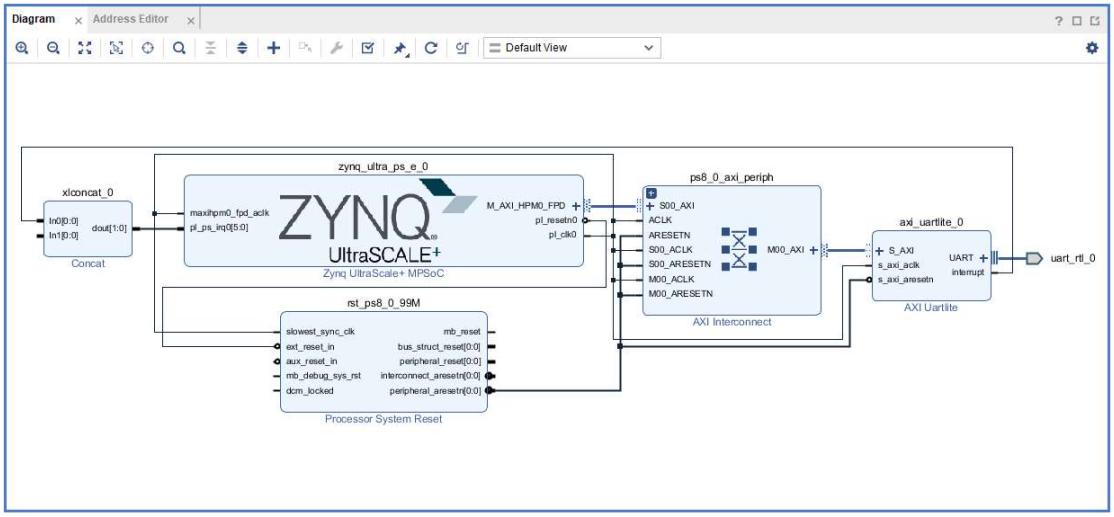


图 4-10 连线完成的 BD 工程

生成顶层文件后，加入约束文件。

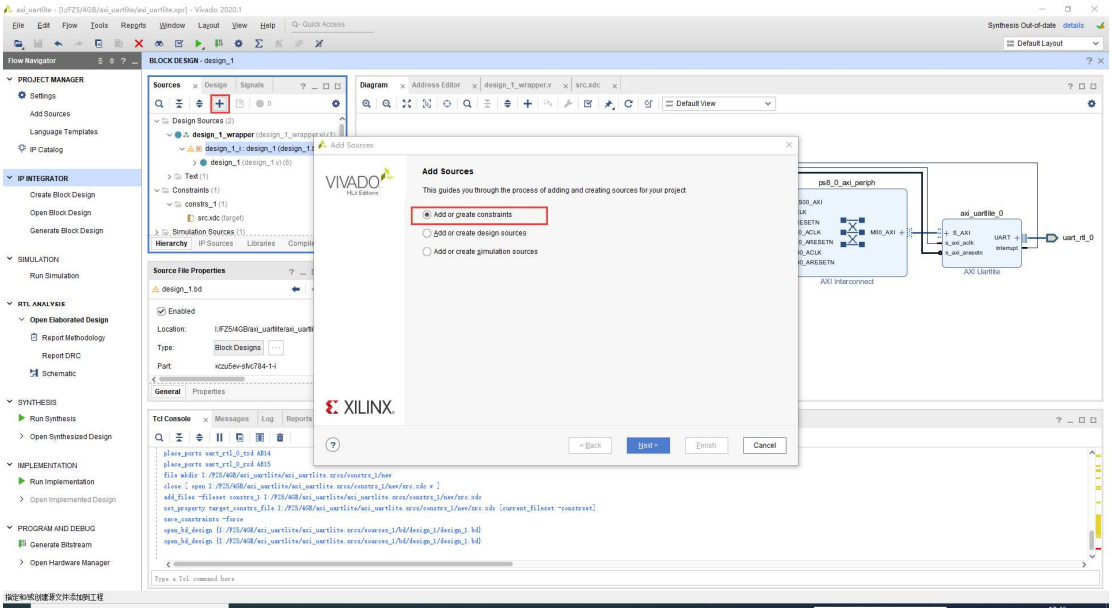


图 4-11 添加约束文档

创建 src 约束文件。

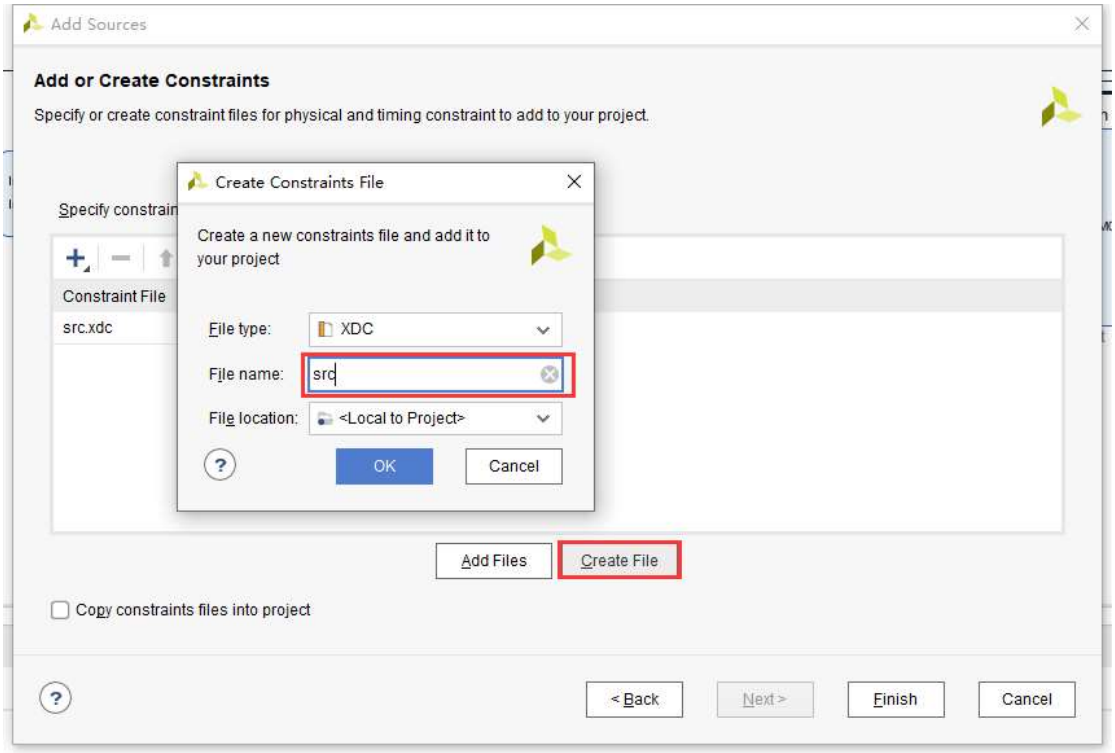


图 4-12 约束文档名称

led.xdc 添加以下内容，端口名称要和顶层文件端口一致。

```
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]

```

图 4-12 添加引脚分配

下面生成 bitstream 跟导出 xsa 文件的步骤与第一个工程一致，就不再赘述。

Vitis 工程：

进入 Vitis 软件，新建名为 axi_uartlite 的工程，在 Templates 模板选择页面，选择 empty 模板，点击 platform.xpr→Board Support Package→(axi_uartlite)Import Examples→Example Directory。

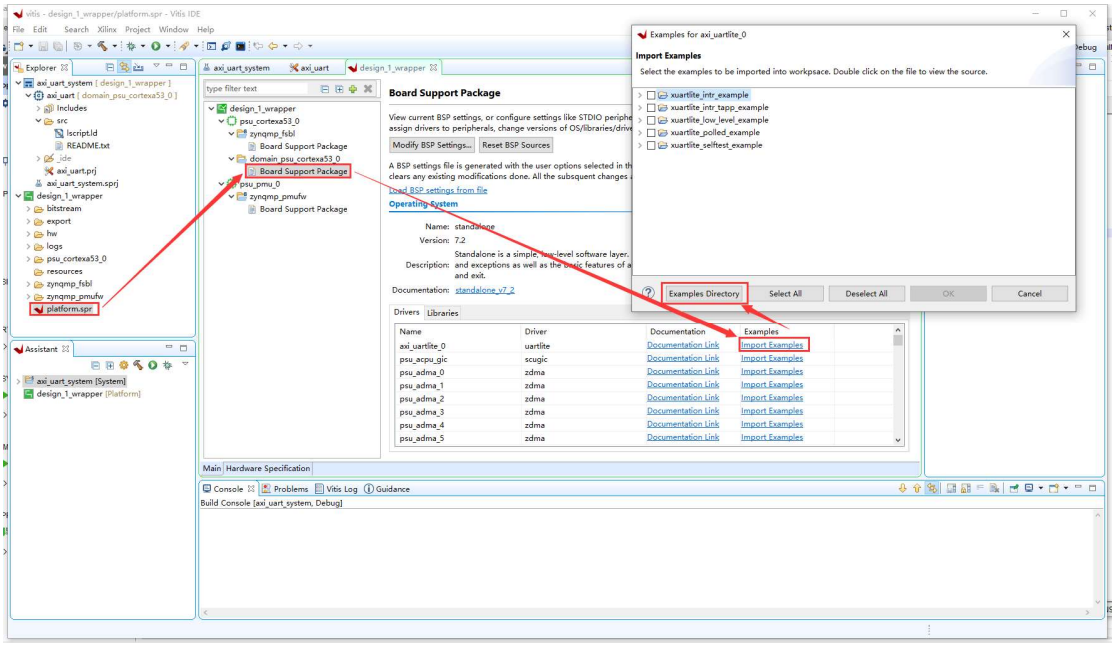


图 4-13 vitis 工程

在打开的目录中，复制 xuartlite_low_level_example.c 到我们的工程 src 目录下。

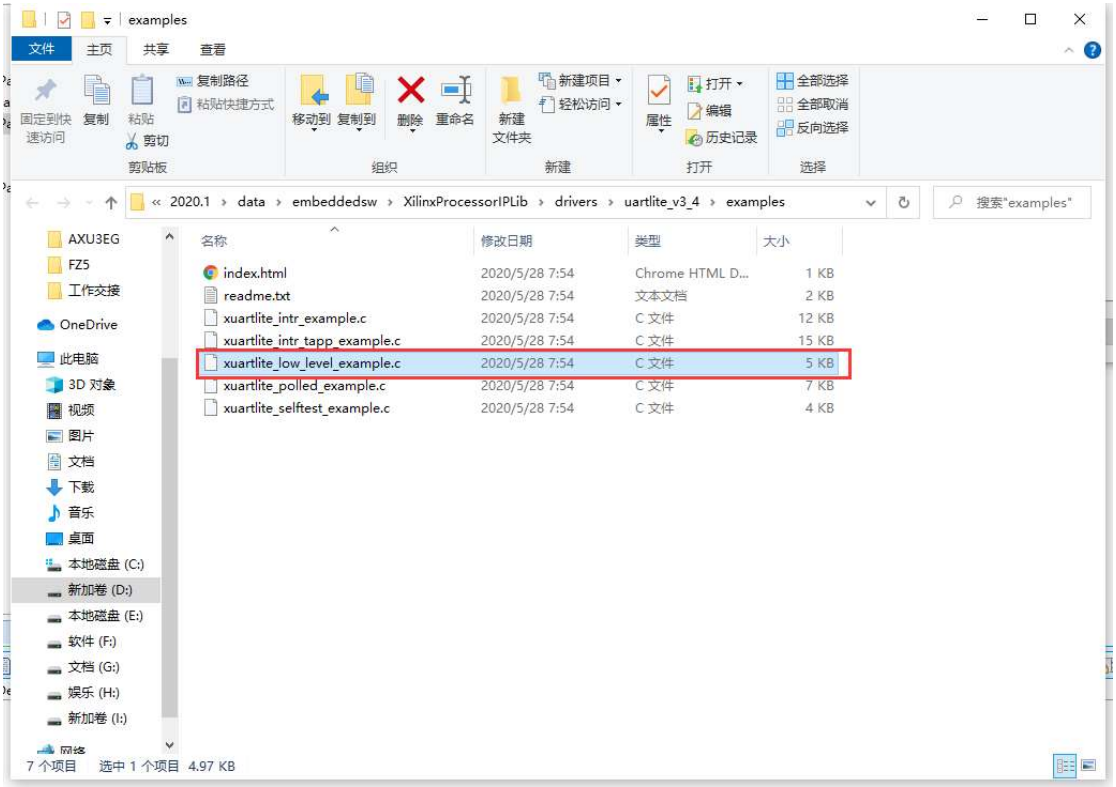


图 4-14 官方驱动移植

编译完成后，连接 JTAG 线到开发板、 UART 的 USB 线到 PC 使用 PuTTY 软件做为串口终端调试工具。

Debug 方式与第一个工程一致，程序下载后，调试串口输出结果如下。

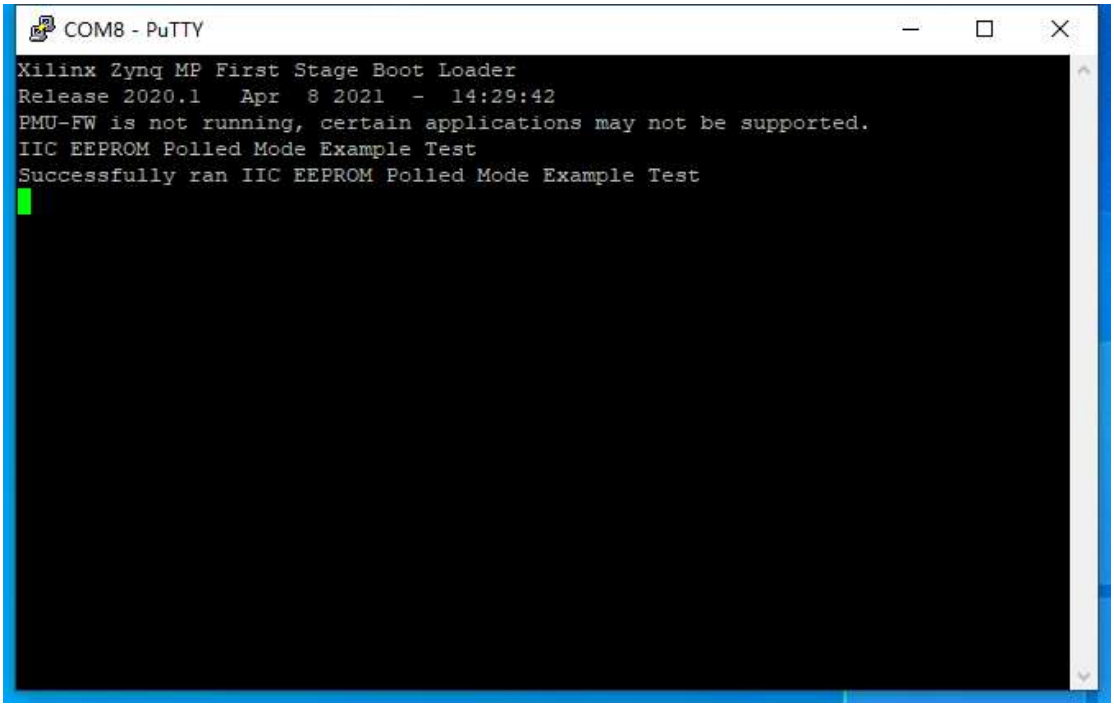


图 4-15 结果显示

4.2 IIC

4.2.1 IIC 基础知识

I2C 总线是一种串行数据总线，只有二根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL，两条线可以挂多个设备。 IIC 设备里有个固化的地址，只有在两条线上传输的值等于 IIC 设备的固化地址时，其才会作出响应。通常我们为了方便把 IIC 设备分为主设备和从设备，谁控制时钟线谁就是主设备。

4.2.2 实验逻辑

本实验的工作是配置好 PS 端的 I2C 以后，通过 PS 端的 I2C 总线对 EEPROM 进行读写测试。

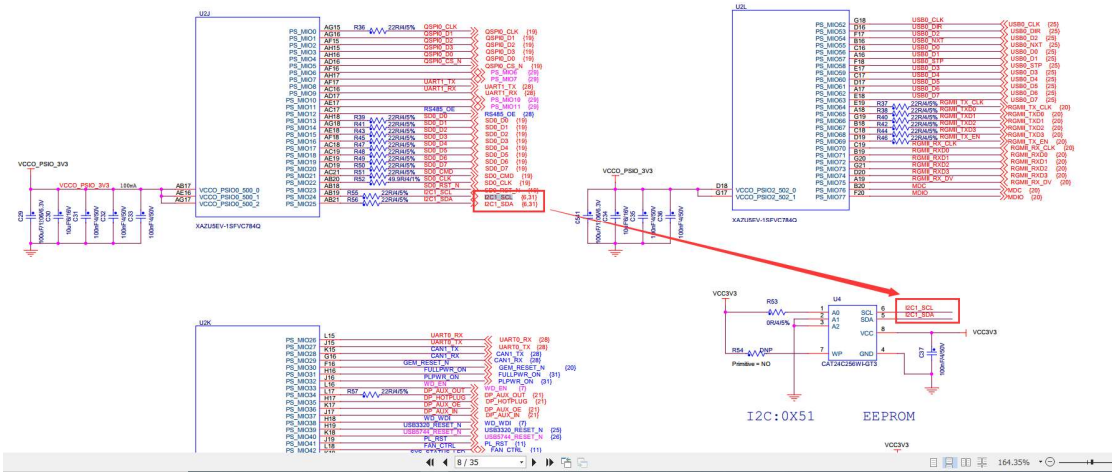


图 4-16 原理图连接示意图

4.2.3 实验步骤

Vivado 工程：

基于”hello_world”工程，另存为一份 iic_test 工程。光盘名称为 iic_test.rar。

这里 iic 是使用 PS 端的 MIO 引出的 IIC，已经在第一个工程中配置过了，所以不需要再做改动。直接生成 bitstream，导出 xsa 文件即可。

Vitis 工程：

新建工程的步骤可以参考第一个工程，新建名为 iic_test 的 application project 后，与 axi uart 相同的方法，点击 platform.xpr→Board Support Package→(psu_i2c_1)Import Examples→Example Directory，

在打开的目录中，复制 xiicps_eeprom_polled_example.c 到我们的工程 src 目录下。

由于我们的 iic 设备地址为 0x51，在这里修改 Eeprom 的第一个地址为 0x51，点击编译即可。

```
/************************** Function Prototypes ******************************/

s32 IicPsEepromPolledExample(void);
static s32 EepromWriteData(XIicPs *IicInstance, u16 ByteCount);
static s32 EepromReadData(XIicPs *IicInstance, u8 *BufferPtr, u16 ByteCount);
static s32 IicPsSlaveMonitor(u16 Address, u16 DeviceId);
static s32 MuxInitChannel(u16 MuxIicAddr, u8 WriteBuffer);
static s32 FindEepromDevice(u16 Address);
static s32 IicPsFindEeprom(u16 *Eeprom_Addr, u32 *PageSize);
static s32 IicPsConfig(u16 DeviceId);
static s32 IicPsFindDevice(u16 addr, u16 DeviceId);
static int FindEepromPageSize(u16 EepromAddr, u32 *PageSize_ptr);
/************************** Variable Definitions *****************************/
#ifndef TESTAPP_GEN
XIicPs IicInstance;      /* The instance of the IIC device. */
#endif
u32 Platform;

/*
 * Write buffer for writing a page.
 */
u8 WriteBuffer[sizeof(AddressType) + MAX_SIZE];

u8 ReadBuffer[MAX_SIZE];    /* Read buffer for reading a page. */

/**Searching for the required EEPROM Address and user can also add
 * their own EEPROM Address in the below array list**/
u16 EepromAddr[] = {0x51,0x55,0};
u16 MuxAddr[] = {0x74,0};
u16 EepromSlvAddr;
u32 PageSize;

/************************** Function Definitions *****************************/


/*****************************************************************************/
/**
* Main function to call the Iic EEPROM polled example.
*
* @param    None.
*
* @return   XST_SUCCESS if successful else XST_FAILURE.
```

图 4-17 vitis 程序设计

Jtag 调试与 SD 卡启动方式与第一个工程一致，程序下载后，调试串口输出结果如下。

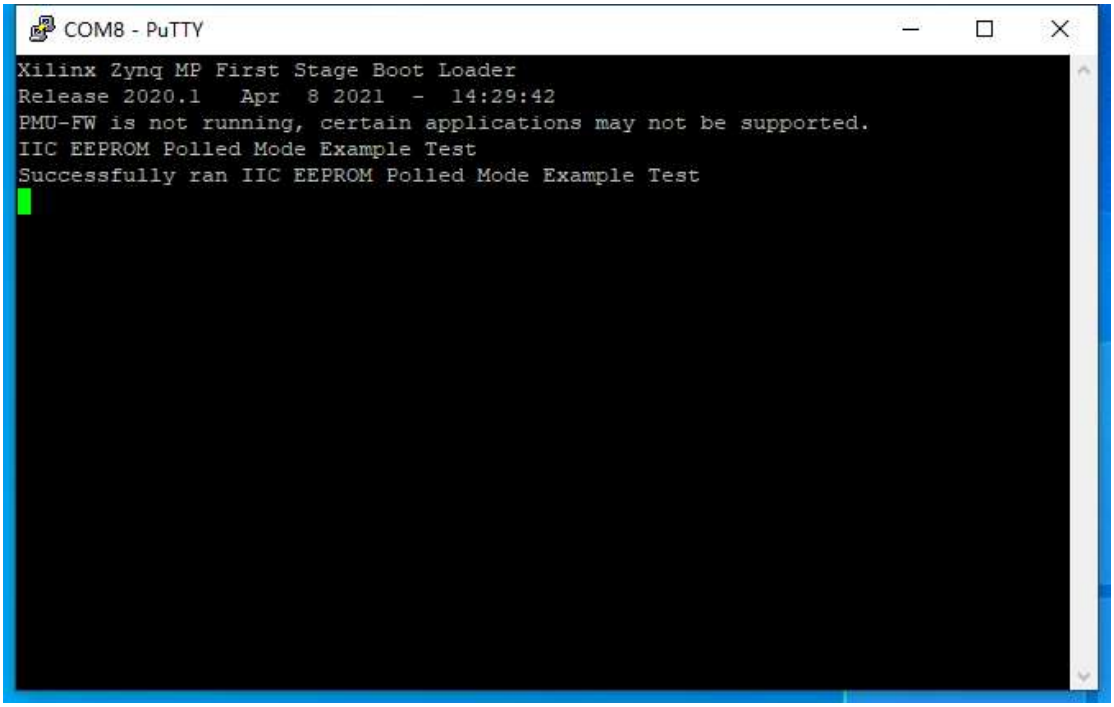


图 4-18 结果显示

4.3 本章小节

本章有两个小节，第一节讲述了 PS 通过 axi 总线控制 uart 串口 IP 打印数据，第二节是关于 IIC 的一些知识，并使用 PS 的 IIC 外设，对板子上的 EEPROM 进行读写。

知识点为 PL 端 axi 总线 IP 核的例化，通过 axi 总线连接到 PS 上，并通过软件控制。

第五章 基于 AXI4 高速数据接口设备模块

5.1 BRAM

5.1.1AXI BRAM Contorller 基础知识

BRAM 是 FPGA 定制的 RAM 资源，有着较大的存储空间，且在日常的工程中使用较为频繁。BRAM 以阵列的方式排布于 FPGA 的内部，是 FPGA 实现各种存储功能的主要部分，是真正的双读/写端口的同步的 RAM。

5.1.2 实验逻辑

本篇文章目的是使用 Block Memory 进行 PS 和 PL 的数据交互或者数据共享， 通过 zynq PS 端的 M_AXI_HPM0_FPD 端口向 控制两个 axi bram 控制器， 通过控制器 0 向 bram 中写数据，然后用控制器 1 把 bram 中的数据读出来， 将读出的结果打印输出到串口终端显示。

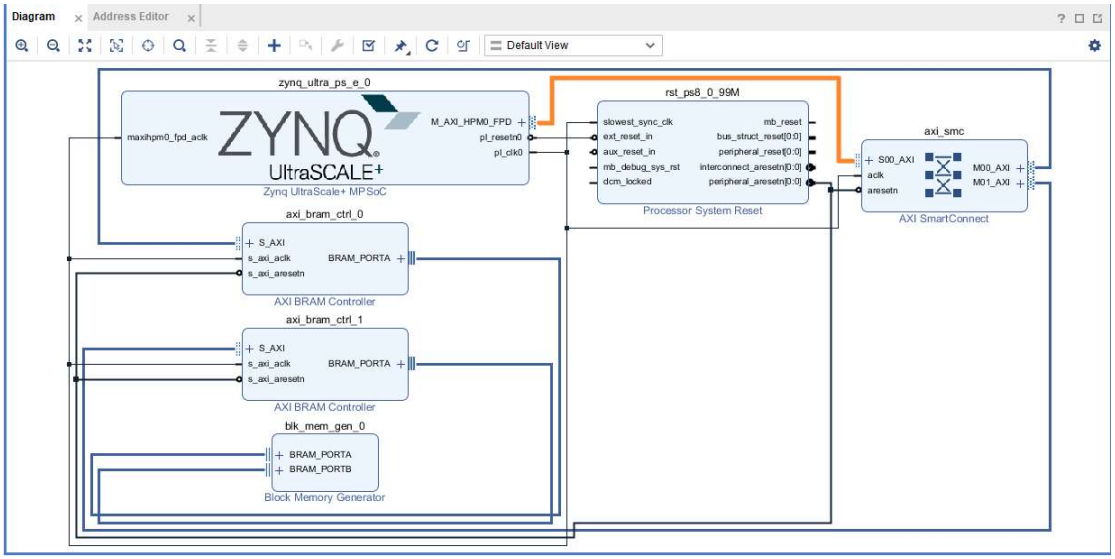


图 5-1 BRAM 实验逻辑结构框图

5.1.3 实验步骤

基于”hello_world”工程，另存为一份 bram_test 工程。对应光盘文档 bram_test.rar。

在搜索栏输入 block，双击 Block Memory Generator 添加 bram 核。

双击 bram 核，配置如下。

将 Memory Type 设置为 True Dual Port RAM。

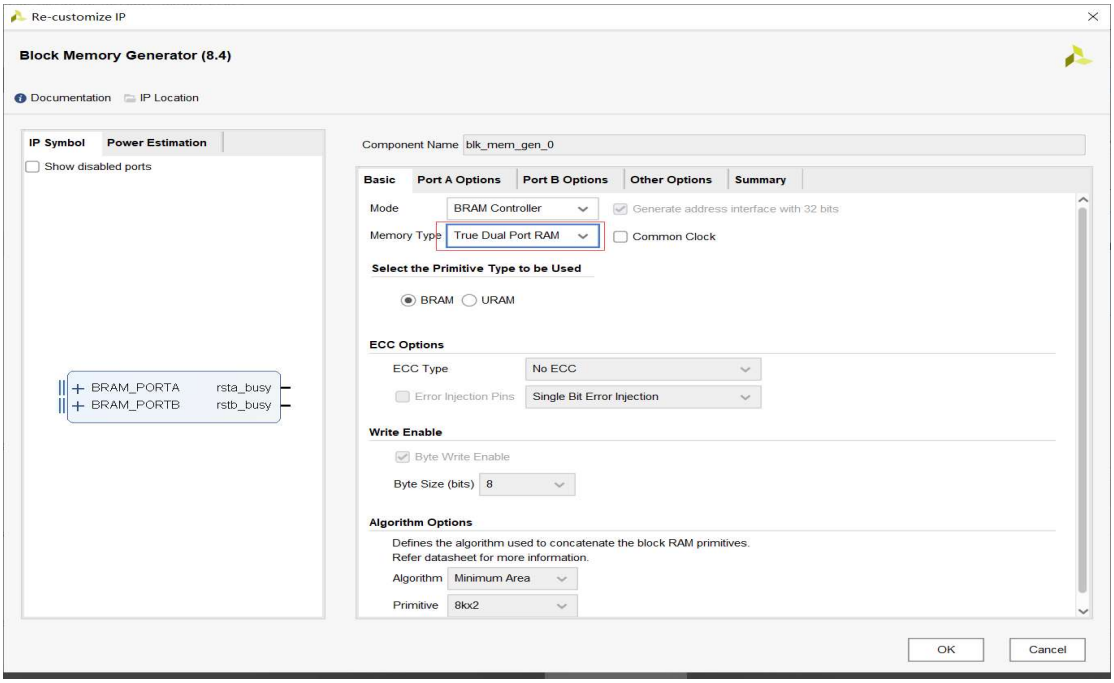


图 5-2 配置 axi BRAM

取消 Enable Safety Circuit 的勾选,点击 OK。

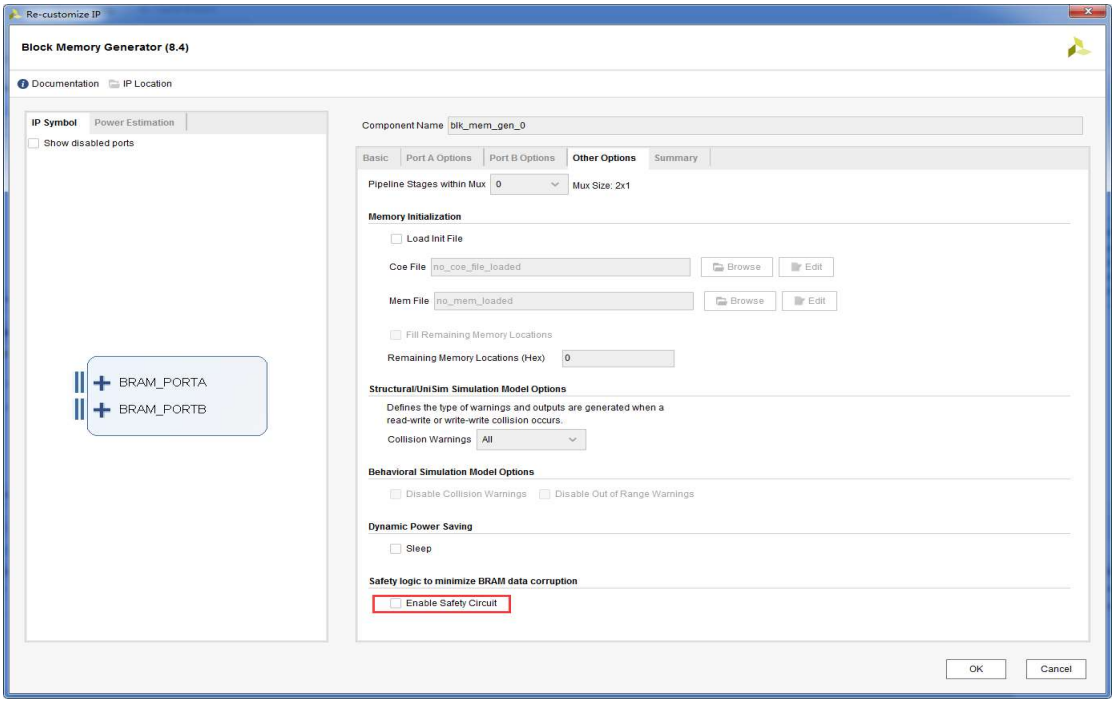


图 5-3 配置 AXI BRAM（2）

其他选项保持默认值，点击 ok 完成配置。

在搜索栏输入 axi_bram，双击 AXI BRAM Controller 添加 axi_bram_controller 核。

双击 axi_bram_controller 设置参数，将 Number of BRAM interfaces 设置为 1，另外一个 axi_bram_controller 也这样设置

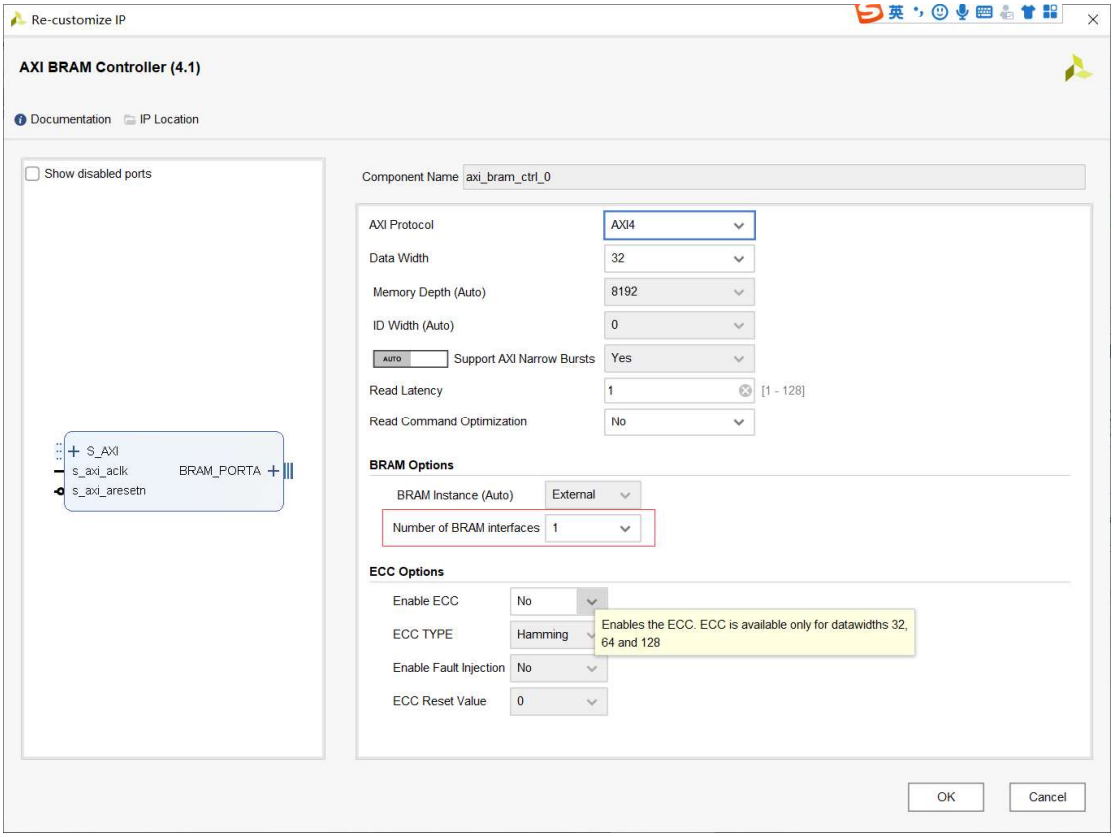


图 5-4 配置 BRAM 控制器

点击 Run Block Automation->OK 进行自动连线，方式与 uart 中一样，勾选所有选项，点击 OK。

设置地址

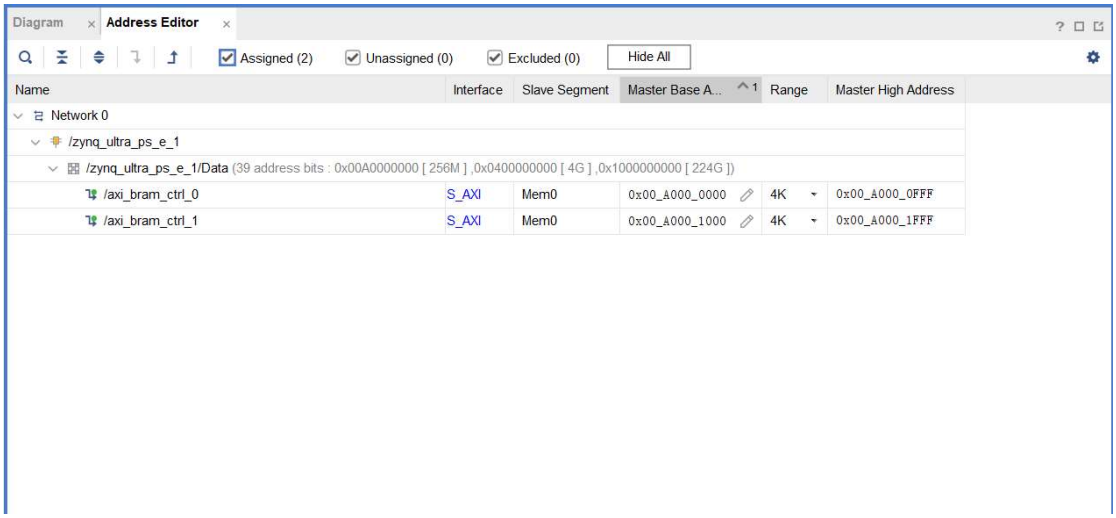


图 5-5 分配地址空间

后面的流程跟 uart 工程一致，不过由于 bramtest 没有用到 pl 端 io，所以不需要进行 io 约束。

Vitis 工程：

进入 Vitis 软件，新建名为 bram_test 的工程，在 Templates 模板选择页面，选择 empty 模板，然后在生成的工程中将例程中的程序文件复制进去。

代码如下，就是向 bram 的寄存器中写入 0-15 这几个数字，然后读出来，再将读出的数据打印出来。

```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 < 15; num++)
    {
        Xil_Out32(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + num * 4, 0x10000000 + num); //
    }

    for (num = 0; num < 15; 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;
}
```

图 5-6 vitis 代码设计

Jtag 调试与 SD 卡启动方式与第一个工程一致，程序下载后，调试串口输出结果如下。

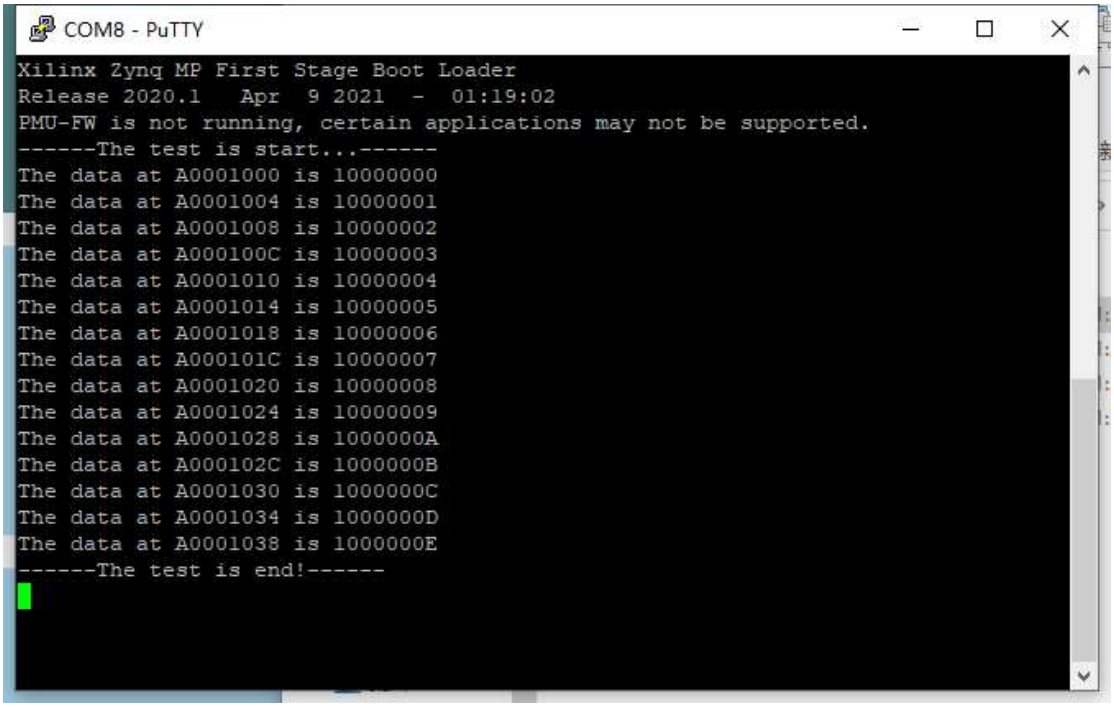


图 5-7 结果显示

5.2 AXI DMA

5.2.1AXI DMA 基础知识

DMA 的产生背景：DMA（Direct Memory Access，直接内存存取），是指外部设备不通过 CPU 直接与系统内存交换数据的接口技术。要将外设数据读入内存或将内存传送到外设，一般都要通过 CPU 控制完成，如采用查询或中断方式。虽然中断方式可以提高 CPU 的利用率，但是也会有效率问题，对于批量传送数据的情况，采用 DMA 方式，可解决效率与速度问题，CPU 只需要提供地址和长度给 DMA， DMA 即可接管总线，访问内存，等 DMA 完成工作后，告知 CPU，交出总线控制权。

DMA 工作流程：首先就要 CPU 收到外设的 DMA 请求中断，然后就是 CPU 中断，设置好 DMA 的传输地址、长度、中断等信息，启动 DMA 传输。再接下来就是 CPU 去干其他的事情，外设使用 DMA 方式进行数据传输，最后就是外设数据传输完成，发送给 CPU 传输完成的中断，CPU 处理完中断之后再去干其他的事情。

下面主要介绍一下 AXI DMA IP 的基本情况，这里主要摘录 PG021 中的内容。

1，AXI DMA 模块用到了三种总线，AXI4-Lite 用于对寄存器进行配置，AXI4 Memory Map 用于与内存交互，在此模块中又分立出了 AXI4 Memory Map Read 和 AXI4 Memory Map Write 两个接口，又分别叫做 M_AXI_MM2S 和 M_AXI_S2MM，一个是读一个是写，这里要搞清楚，不能混淆。AXI4 Stream 接口用于对外设的读写，其中 AXI4 Stream Master（MM2S）用于对外设写，AXI4-Stream Slave(S2MM)用于对外设读。另外还支持 Scatter/Gather 功能。（MM2S 表示 Memory Map to Stream，S2MM 表示 Stream to Memory Map）。

AXI Memory Map 数据宽度支持 32，64，128，256，512，1024bits

AXI Stream 数据宽度支持 8，16，32，64，128，256，512，1024bits

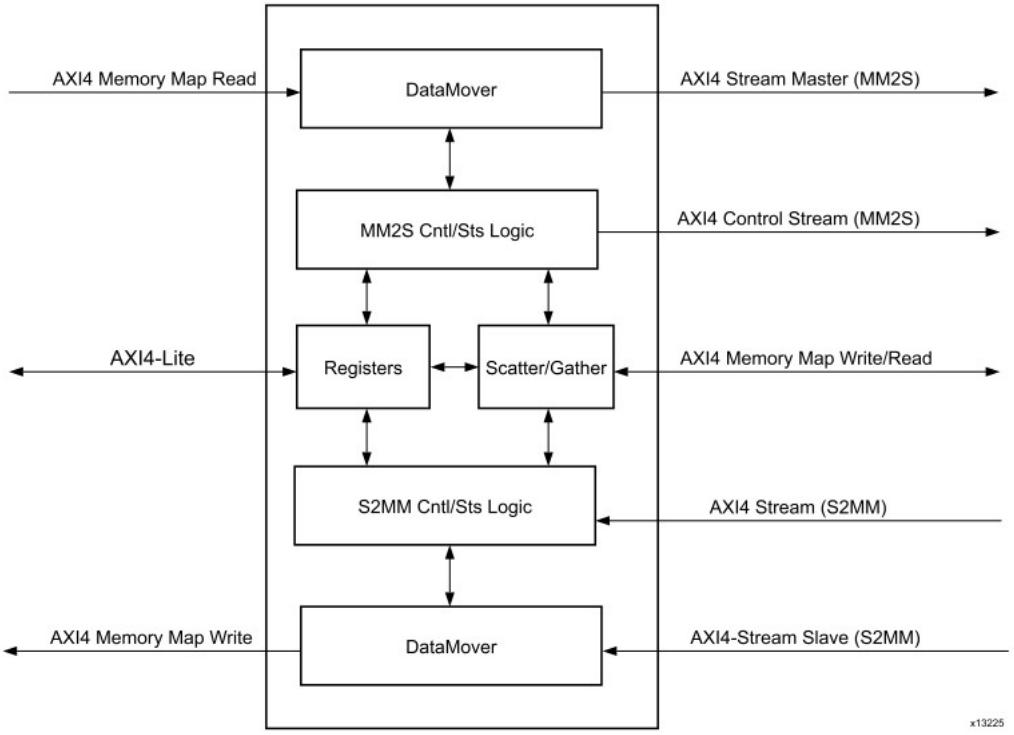


Figure 1-1: **AXI DMA Block Diagram**

图 5-8 DMA 内部框图

2，本实验中采用直接寄存器模式，如下图为寄存器说明，主要分为两部分，一是 MM2S，包括 Control Register，Status Register，Source Address，Transfer Length 四部分，二是 S2MM，同样包括 Control Register，Status Register，Destination Address，Buffer Length 四部分，注意这里的 Source Address 和 Destination Address 指的是内存地址。

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

图 5-9 直接寄存器存取地址空间

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)

图 5-10 地址空间

3，以下为 MM2S_DMACR 控制寄存器说明，比较重要的是 Bit0，Run/Stop，表示开启或停止 DMA。其他内容不再讲述。

MM2S_DMACR (MM2S DMA Control Register – Offset 00h)

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

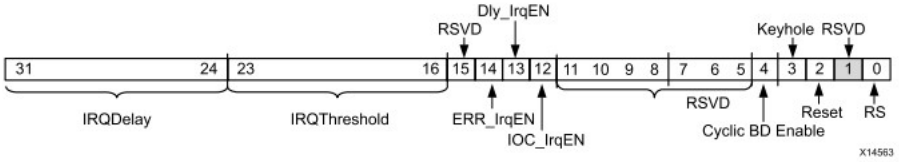


Figure 2-2: **MM2S DMACR Register**

图 5-11 MM2S_DMACR 控制器寄存器

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.

图 5-12 MM2S_DMACR 控制器寄存器 bit0 描述

在这里有几个中断可以设置，IOC_IrqEn，使能完成中断，Dly_IrqEn 使能延迟中断，Err_IrqEn 使能错误中断。

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

图 5-13 MM2S_DMACR 控制器寄存器 bit12、13、14 描述

4，MM2S_DMASR 为状态寄存器说明，bit12,13,14 为中断状态，写 1 可清除中断。

MM2S_DMASR (MM2S DMA Status Register – Offset 04h)

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

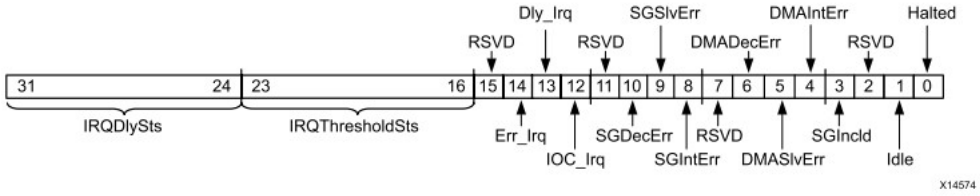


Figure 2-3: **MM2S DMASR Register**

图 5-14 MM2S_DMASR 控制器寄存器

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.

图 5-14 MM2S_DMASR 为状态寄存器 bit12、13、14 说明

5，MM2S_DA，MM2S_LENGTH 代表地址和长度设置，S2MM 端的寄存器与 MM2S 类似，不再讲述。

5.2.2 实验逻辑

本实验是通过在 pl 端例化 dma IP 核，PS 端通过 axi 总线，对 dma IP 核进行数据传输测试，本实验功能是 MM2S 从 DDR3 中读取数据，写到 AXI Stream Data FIFO，再从 FIFO 读出写到 DDR3，实现环通测试，需要打开 S2MM_DMACR 的 IOC_Irq，即写内存结束中断，功能框图如下所示：

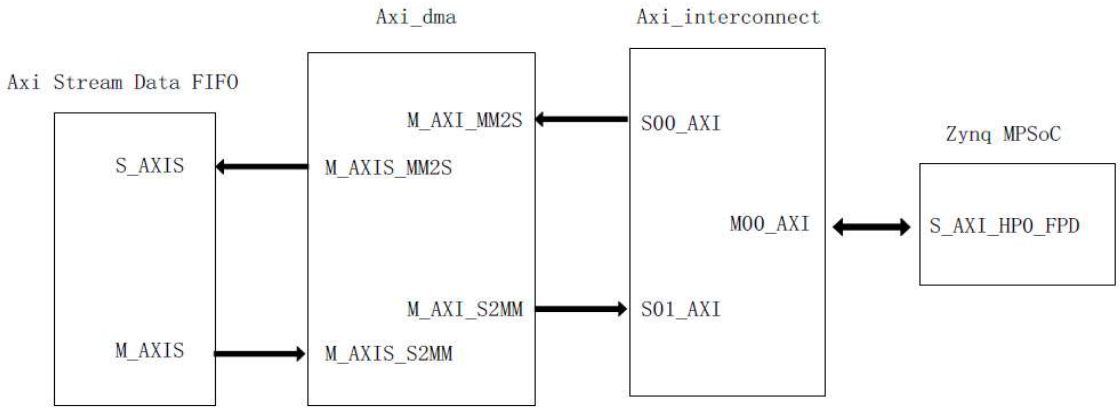


图 5-15 原理框图

5.2.3 实验步骤

Vivado 工程:

基于”hello_world”工程，另存为一份 dma_loop 工程，对应在光盘的设计文档是 dma_loop.rar。打开复位，HPM0,HP0，PL to PS 中断 IRQ0。

在搜索栏输入 dma，调用 dma IP 核

双击 DMA IP 核设置参数，配置如下。

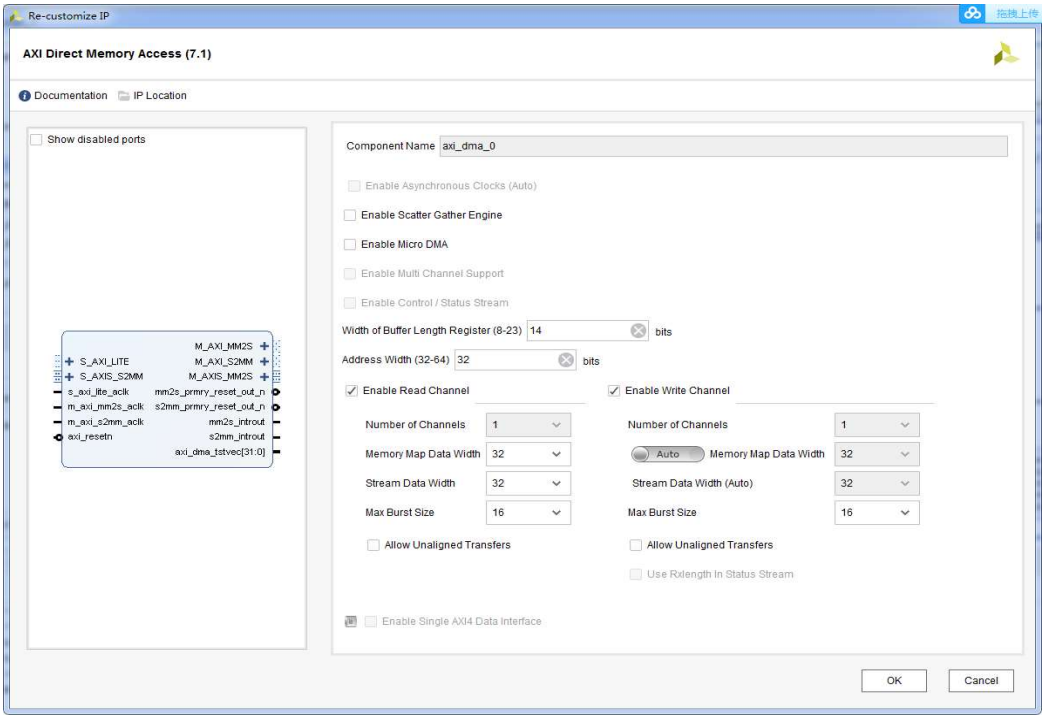


图 5-16 DMA 配置框图

添加 AXI Stream Data FIFO 模块，并设置如下，设置深度为 1024， TDATA Width 为 4 字节，打开 TKEEP， TLAST。

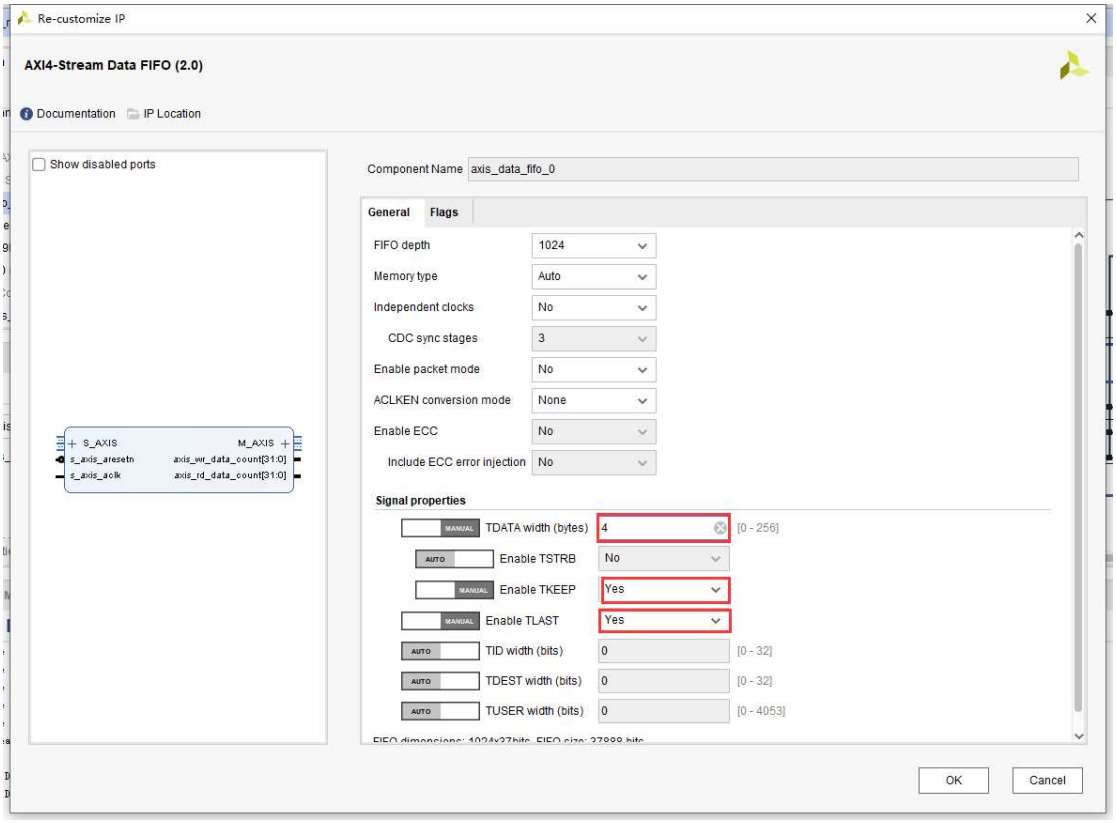


图 5-17 配置 FIFO

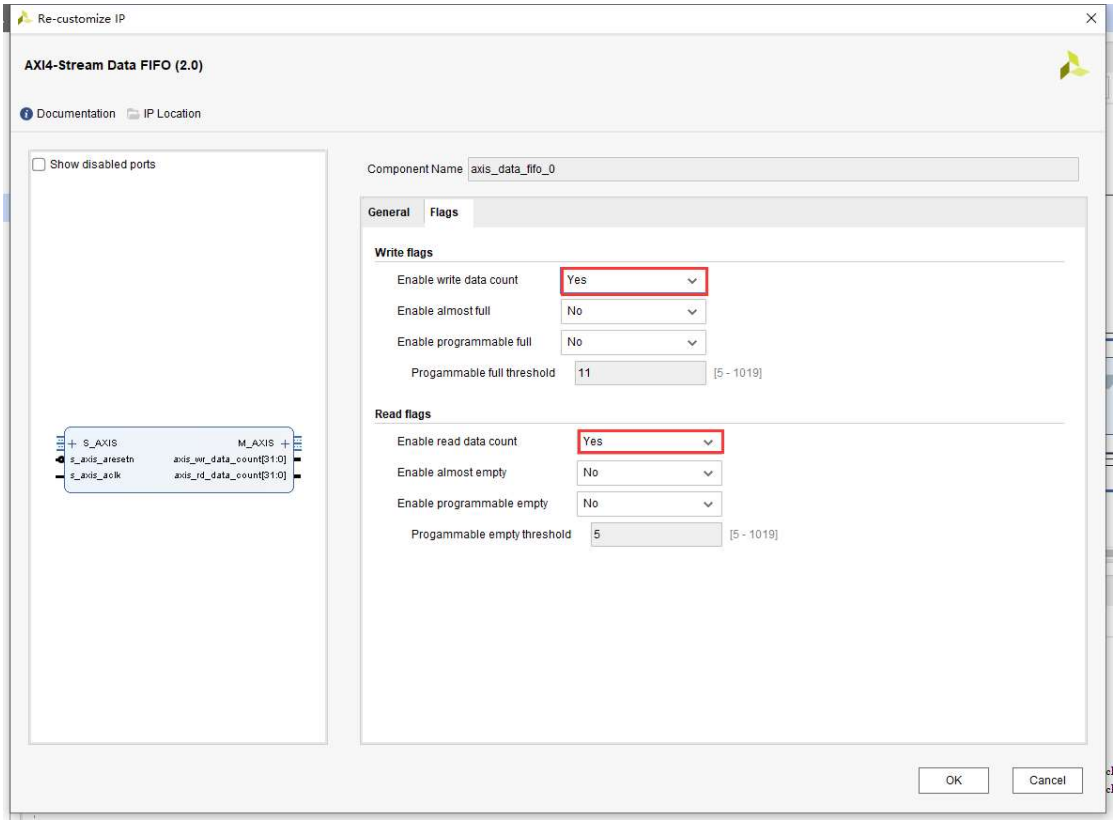


图 5-18 FIFO 配置

自动连接,并连接 FIFO 的 S_AXIS 和 M_AXIS 到 dma，添加 Concat，连接 MM2S 和 S2MM 中断到 pl_ps_irq，最终结果如下图。

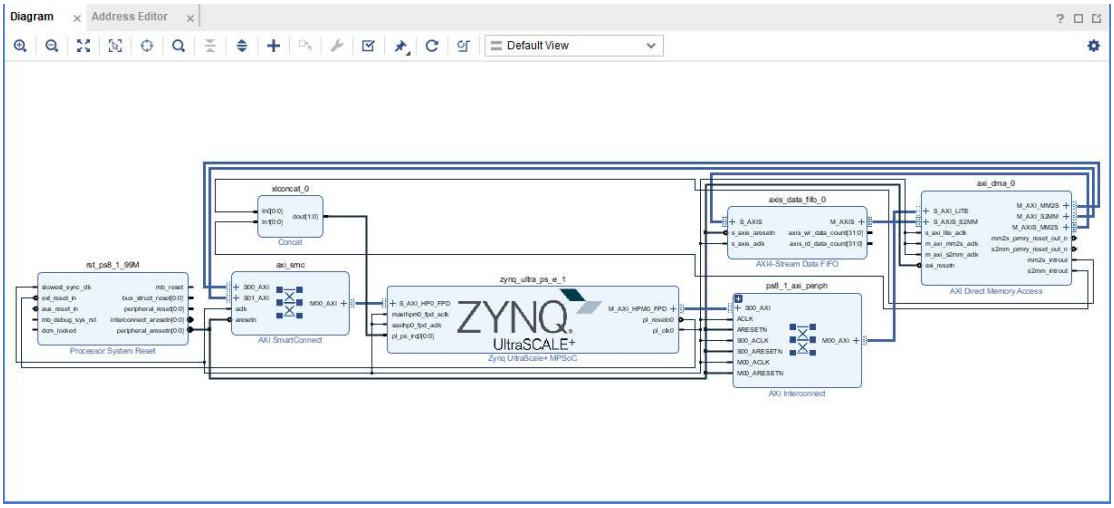


图 5-19 BD 设计框图

后面的流程跟 uart 工程一致，跟 bram test 一样的是，同样不需要引脚约束。

Vitis 工程：

新建工程的步骤可以参考第一个工程，新建名为 dma_loop 的 application project 后，与 axi uart 相同的方法，打开 dma 对应的 Example Directory 文件夹，在打开的目录中，复制 xaxidma_example_simple_poll.c 到我们的工程 src 目录下。

在下图位置增加一个 Tries Count 打印语句，点击编译。

```
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         }
```

图 5-20 vitis 源码设计

Jtag 调试与 SD 卡启动方式与第一个工程一致，程序下载后，调试串口输出结果如下。

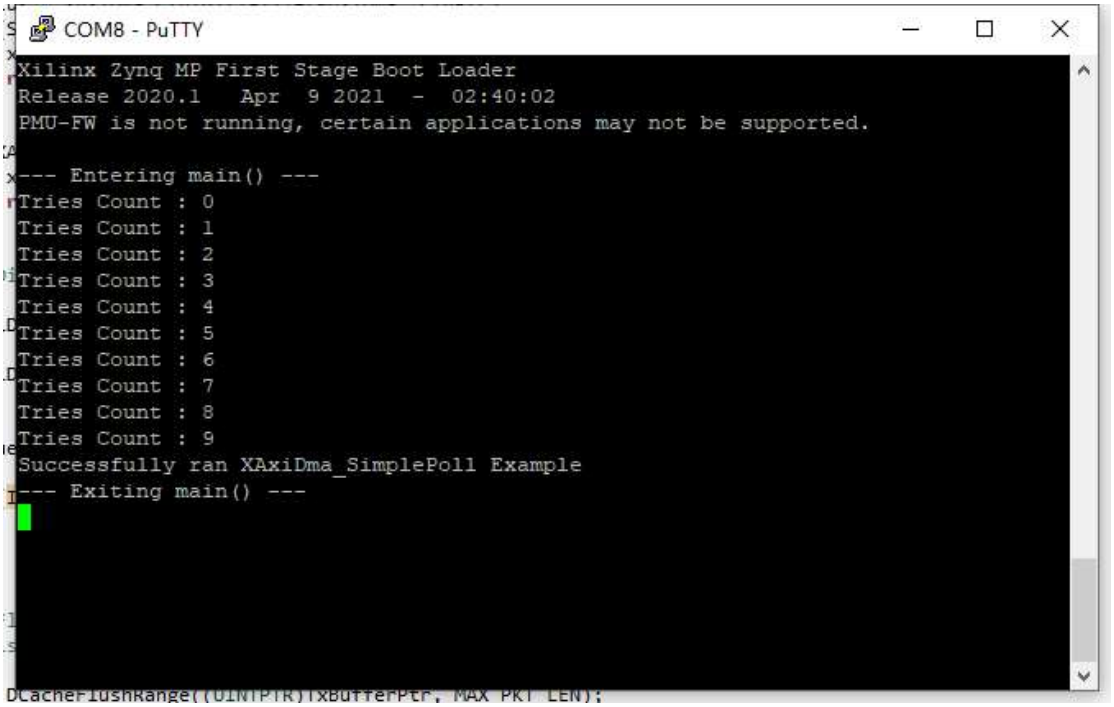


图 5-21 结果显示

5.3 本章小节

第一节 PS 与 PL 通过 BRAM 实现低带宽数据交互的实验，两者通过 GP 口进行数据互连，可以实现小批量的数据交互。

第二节运用了 DMA 进行内存的访问，DMA 是一种 PS 与 PL 数据交互方式，DMA 的方式控制权主要在 PS 端，由 PS 配置 DMA 的读写。

第六章 基于 AXI-Stream 接口设备模块

6.1 MIPI

本工程给出的是 MIPI 的示例工程，这里简要介绍一下 MIPI 物理层协议。

MIPI（移动行业处理器接口）是 Mobile Industry Processor Interface 的缩写。MIPI（移动行业处理器接口）是 MIPI 联盟发起的为移动应用处理器制定的开放标准。

MIPI 联盟的 MIPI DSI 规范

1、名词解释

• DCS (DisplayCommandSet)：DCS 是一个标准化的命令集，用于命令模式的显示模组。
• DSI, CSI (DisplaySerialInterface, CameraSerialInterface
• DSI 定义了一个位于处理器和显示模组之间的高速串行接口。
• CSI 定义了一个位于处理器和摄像模组之间的高速串行接口。
• D-PHY：提供 DSI 和 CSI 的物理层定义

2、DSI 分层结构

DSI 分四层，对应 D-PHY、DSI、DCS 规范、分层结构图如下：

• PHY 定义了传输媒介，输入/输出电路和和时钟和信号机制。
• Lane Management 层：发送和收集数据流到每条 lane。
• Low Level Protocol 层：定义了如何组帧和解析以及错误检测等。
• Application 层：描述高层编码和解析数据流。

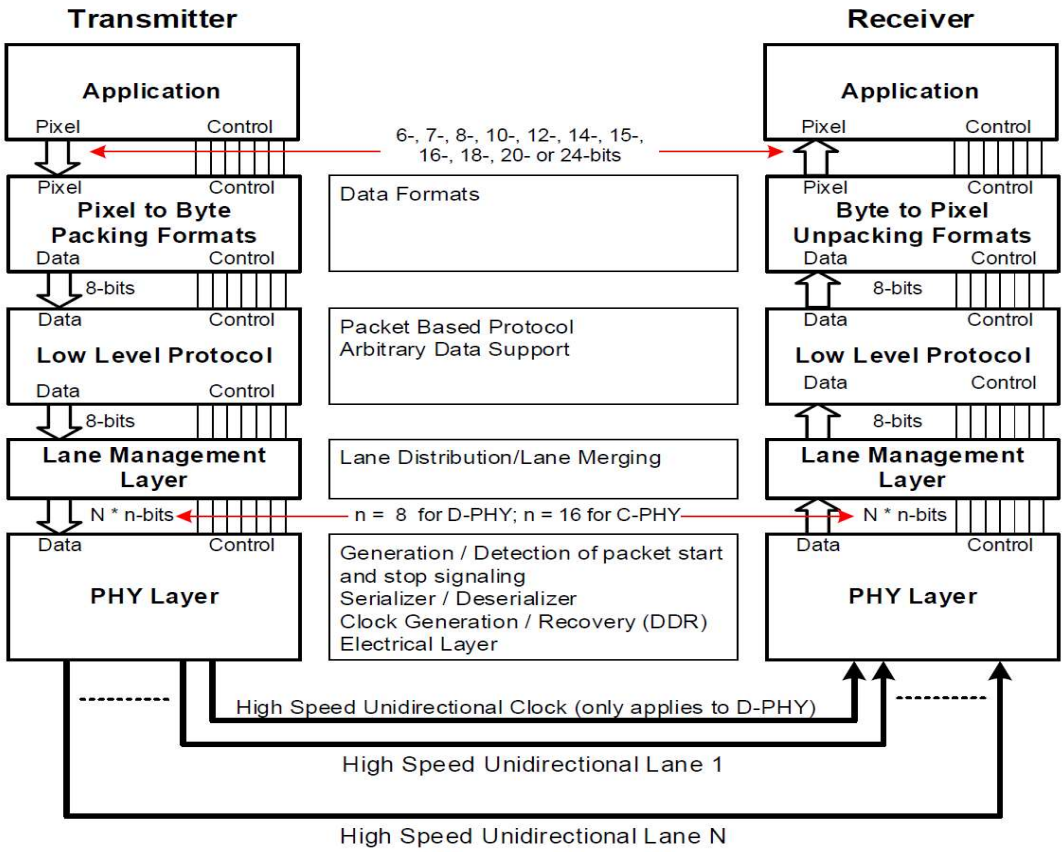


Figure 3 CSI-2 Layer Definitions

图 6-1 MIPI CSI-2 子层定义

Command 和 Video 模式

• DSI 兼容的外设支持 Command 或 Video 操作模式，用哪个模式由外设的构架决定

• Command 模式是指采用发送命令和数据到具有显示缓存的控制器。主机通过命令间接的控制外设。Command 模式采用双向接口

• Video 模式是指从主机传输到外设采用时实象素流。这种模式只能以高速传输。为减少复杂性和节约成本，只采用 Video 模式的系统可能只有一个单向数据路径。

D-PHY 介绍

1、 D-PHY 描述了一同步、高速、低功耗、低代价的 PHY。

• 一个 PHY 配置包括

• 一个时钟 lane

• 一个或多个数据 lane

• 两个 Lane 的 PHY 配置如下图

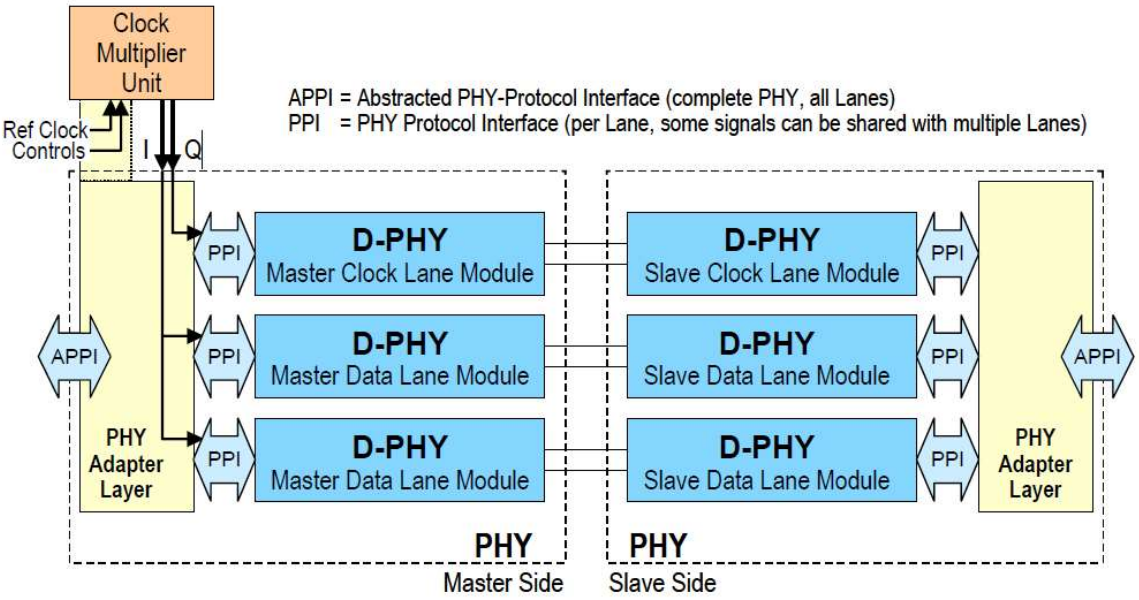


Figure 2 Two Data Lane PHY Configuration

图 6-1 两条 lane 的物理配置

- 三个主要的 lane 的类型
- 单向时钟 Lane
- 单向数据 Lane
- 双向数据 Lane
- D-PHY 的传输模式
- 低功耗（Low-Power）信号模式（用于控制）：10MHz (max)
- 高速（High-Speed）信号模式（用于高速数据传输）：80Mbps ~ 1Gbps/Lane
- D-PHY 低层协议规定最小数据单位是一个字节
- 发送数据时必须低位在前，高位在后.
- D-PHY 适用于移动应用
- DSI：显示串行接口
- 一个时钟 lane，一个或多个数据 lane
- CSI：摄像串行接口

2、Lane 模块

- PHY 由 D-PHY(Lane 模块)组成
- D-PHY 可能包含：
- 低功耗发送器（LP-TX）
- 低功耗接收器（LP-RX）
- 高速发送器（HS-TX）
- 高速接收器（HS-RX）

• 低功耗竞争检测器（LP-CD）
• 三个主要 lane 类型
• 单向时钟 Lane
• Master：HS-TX, LP-TX
• Slave：HS-RX, LP-RX
• 单向数据 Lane
• Master：HS-TX, LP-TX
• Slave：HS-RX, LP-RX
• 双向数据 Lane
• Master, Slave：HS-TX, LP-TX, HS-RX, LP-RX, LP-CD

3、Lane 状态和电压

• Lane 状态
• LP-00, LP-01, LP-10, LP-11 (单端)
• HS-0, HS-1 (差分)
• Lane 电压（典型）
• LP：0-1.2V
• HS：100-300mV (200mV)

4、操作模式

• 数据 Lane 的三种操作模式
• Escape mode, High-Speed(Burst) mode, Control mode
•从控制模式的停止状态开始的可能事件有：
• 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 是数据 Lane 在 LP 状态下的一种特殊操作
•在这种模式下，可以进入一些额外的功能：LPDT, ULPS, Trigger
•数据 Lane 进入 Escape mode 模式通过 LP-11→LP-10→LP-00→LP-01→LP-00
•一旦进入 Escape mode 模式，发送端必须发送 1 个 8-bit 的命令来响应请求的动作
• Escape mode 使用 Spaced-One-Hot Encoding
•超低功耗状态（Ultra-Low Power State）
•这个状态下，lines 处于空状态 (LP-00)
• 时钟 Lane 的超低功耗状态
•时钟 Lane 通过 LP-11→LP-10→LP-00 进入 ULPS 状态
•通过 LP-10 → TWAKEUP →LP-11 退出这种状态，最小 TWAKEUP 时间为 1ms
• 高速数据传输

•发送高速串行数据的行为称为高速数据传输或触发（burst）

•全部 Lanes 门同步开始，结束的时间可能不同。

•时钟应该处于高速模式

• 各模操作式下的传输过程

•进入 Escape 模式的过程 ：LP-11→LP-10→LP-00→LP-01→LP-00→Entry Code → LPD (10MHz)

•退出 Escape 模式的过程：LP-10→LP-11

•进入高速模式的过程：LP-11→LP-01→LP-00→SoT(00011101) → HSD (80Mbps ~ 1Gbps)

•退出高速模式的过程：EoT→LP-11

•控制模式 - BTA 传输过程：LP-11→LP-10→LP-00→LP-10→LP-00

•控制模式 - BTA 接收过程：LP-00→LP-10→LP-11

• 状态转换关系图

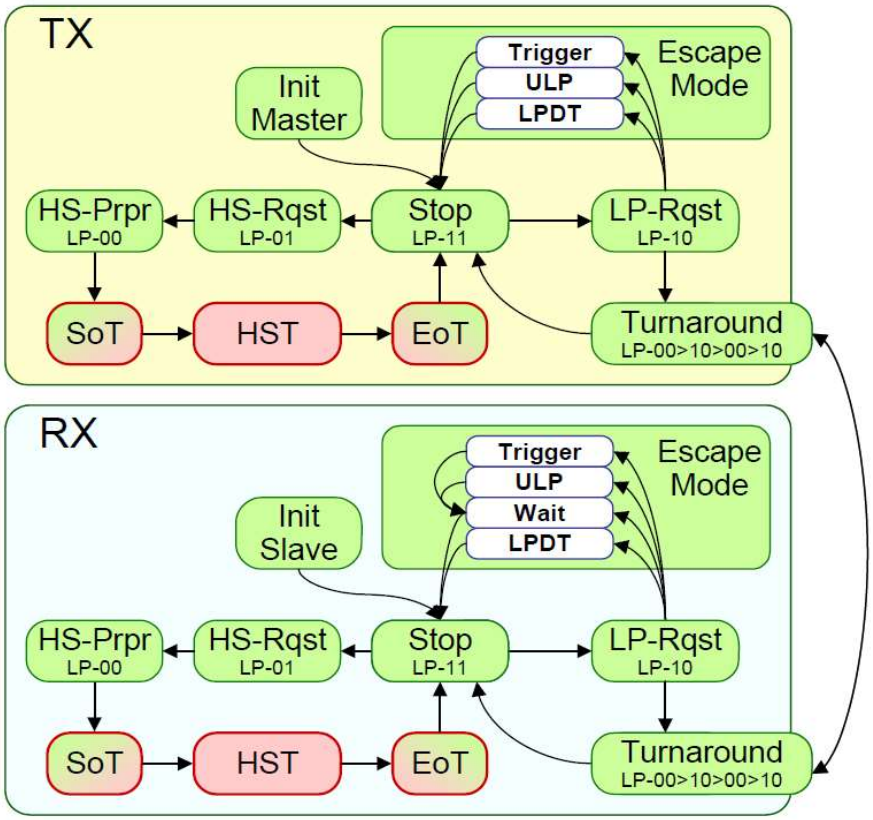


Figure 24 Data Lane Module State Diagram

图 6-3 数据 lane 的模块状态转移图

DSI 介绍

1、DSI 是一种 Lane 可扩展的接口，1 个时钟 Lane/1-4 个数据 Lane

• DSI 兼容的外设支持 1 个或 2 个基本的操作模式：

• Command Mode（类似于 MPU 接口）

• Video Mode（类似于 RGB 接口）- 必须用高速模式传输数据，支持 3 种格式的数据传输
  • Non-Burst 同步脉冲模式
  • Non-Burst 同步事件模式
  • Burst 模式
• 传输模式：
• 高速信号模式（High-Speed signaling mode）
• 低功耗信号模式（Low-Power signaling mode） - 只使用数据 lane 0（时钟是由 DP，DN 异或而来）。
• 帧类型
• 短帧：4 bytes (固定)
• 长帧：6~65541 bytes (可变)
• 两个数据 Lane 高速传输示例

2、短帧结构

• 帧头部（4 个字节）
• 数据标识(DI) 1 个字节
• 帧数据- 2 个字节 （长度固定为 2 个字节）
• 错误检测(ECC) 1 个字节
• 帧大小
• 长度固定为 4 个字节

3、长帧结构

• 帧头部（4 个字节）
• 数据标识(DI) 1 个字节
• 数据计数- 2 个字节 （数据填充的个数）
• 错误检测(ECC) 1 个字节
•数据填充(0~65535 字节)
• 长度=WC*字节
• 帧尾：校验和（2 个字节）
• 帧大小：
• 4 + (0~65535) + 2 = 6 ~ 65541 字节

4、帧数据类型

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 基础知识

Xilinx MIPI CSI-2 RX 控制器在相机传感器和执行基带处理的可编程器件之间实现了相机串行接口。由于更高分辨率相机的发展，相机传感器接口的带宽需求已经上升。

传统的并行接口需要越来越多的信号线，从而导致更高的功耗。新的高速串行接口，例如 MIPI CSI 规范，可以在不牺牲功率的情况下满足这些不断扩展的带宽要求。

MIPI 是移动行业组织定义的一组协议，用于标准化手机和平板电脑等移动平台内的所有接口。然而，移动行业的大容量和规模经济正迫使其他应用程序也采用这些标准。

因此，基于 MIPI 的摄像头传感器越来越多地用于汽车应用中的驾驶员辅助技术、视频安全监控摄像头、视频会议以及虚拟和增强现实等新兴应用中。

关于 IP 的延时计算框图

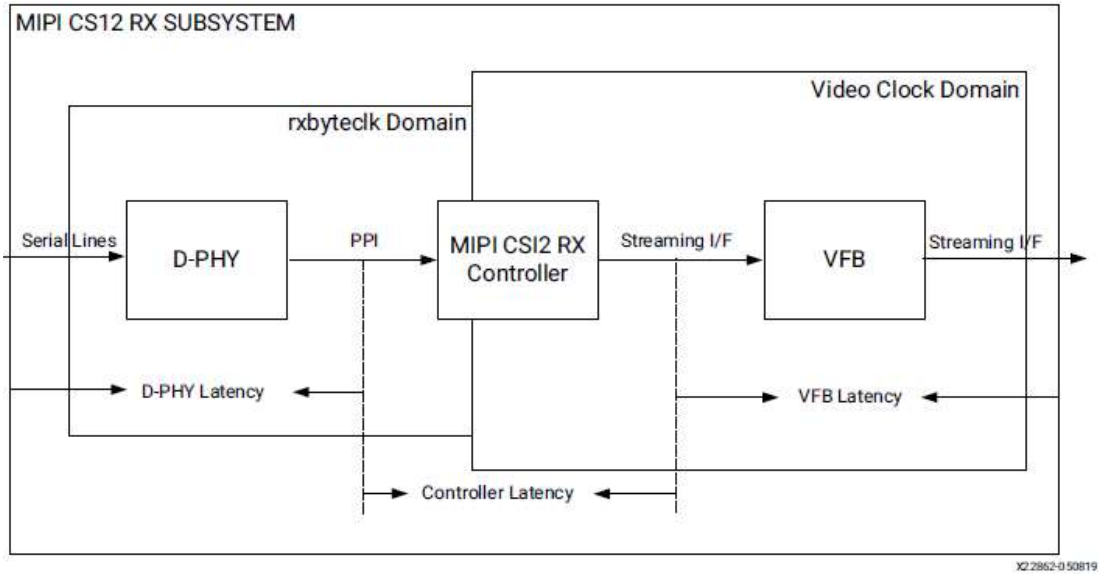


Figure 2-1: **MIPI CSI-2 RX Subsystem Latency Calculation**

图 6-4 MIPI CSI-2 Rx Subsystem 延时计算框图

关于接口：

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

图 6-5 接口描述

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

图 6-6 接口描述

寄存器空间参考产品手册。

关于时钟：

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.

图 6-7 时钟描述

关于复位：

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.

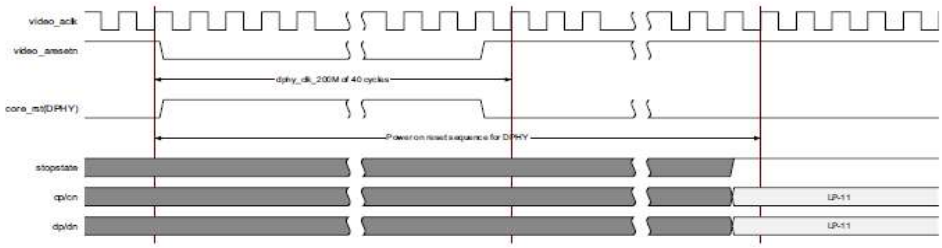


Figure 3-8: **MIPI CSI-2 RX Reset**

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.

图 6-8 复位描述

6.1.2 实验逻辑

本示例使用的是 IMX334 的摄像头，板卡的 MIPI 物理接口，如图所示

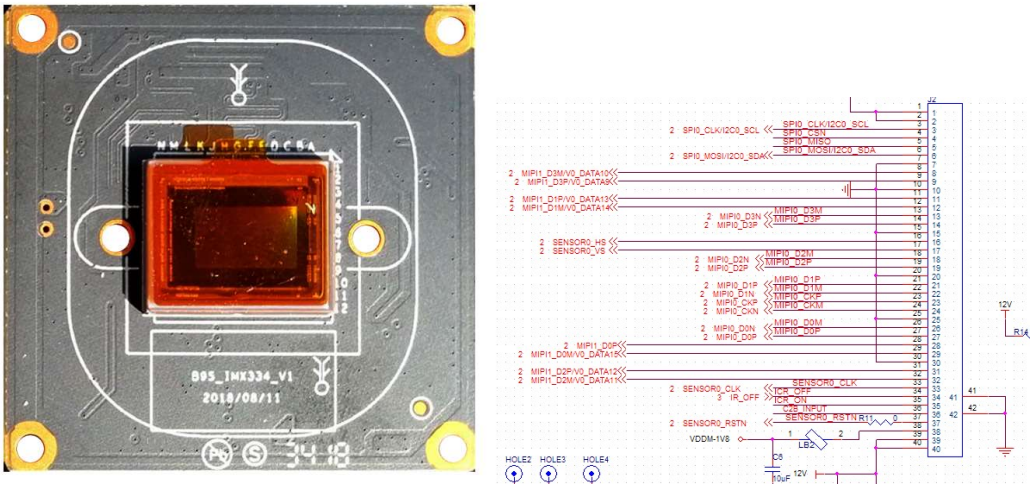


图 6-9 使用的 MIPI 摄像头子板实物与 MIPI 接口

使用的硬件平台上的 MIPI 物理接口，

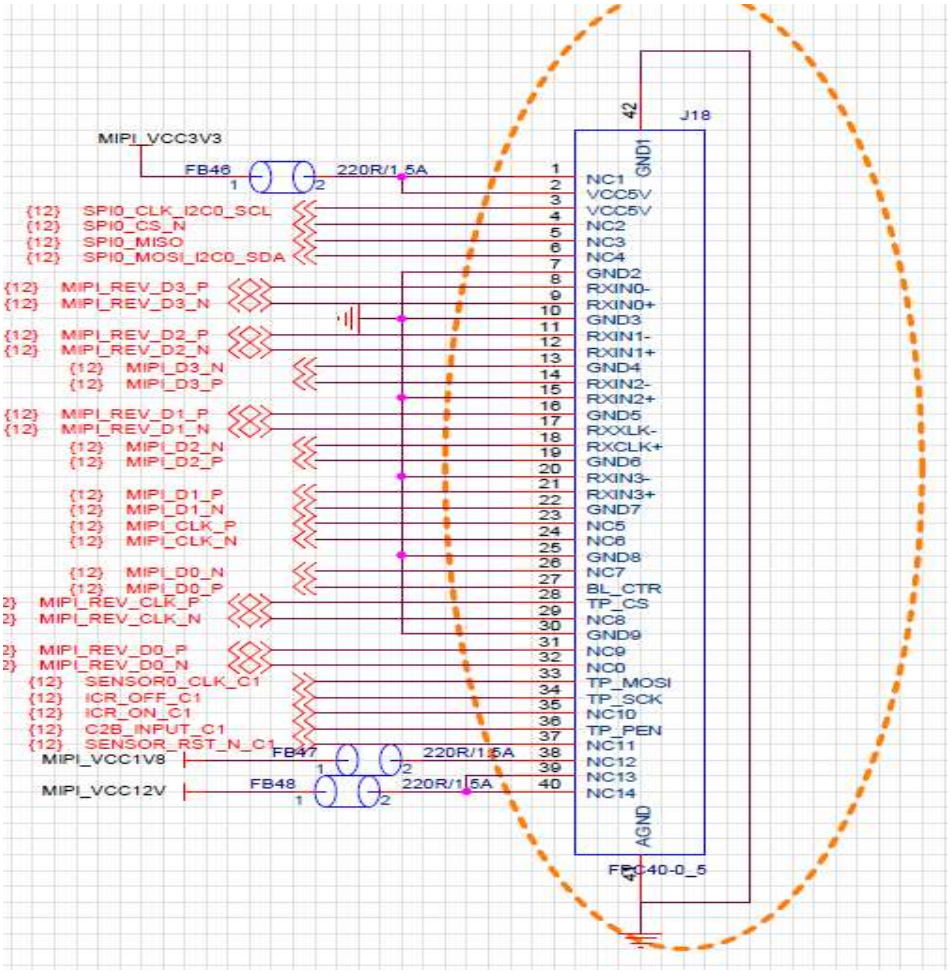


图 6-10 硬件平台原理图上的 MIPI 物理接口

ZU5EV 的接入引脚：

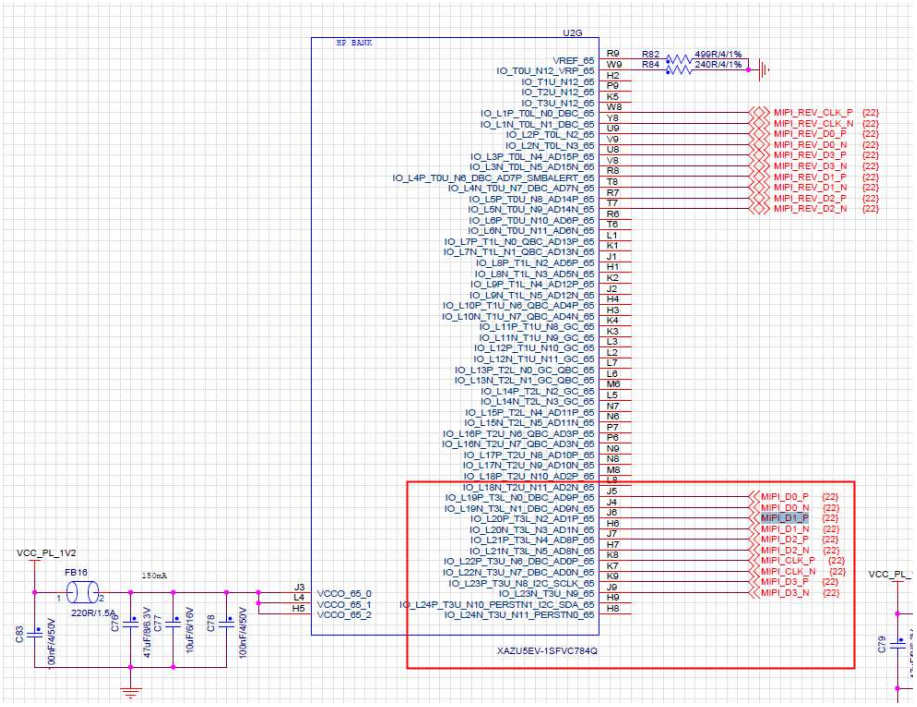


图 6-11 XAZU5EV 的原理图 MIPI 引脚连线

Xilinx MIPI CSI-2 RX 例化的引脚就是上图中的 MIPI 接入引脚。

从而在硬件上链接了 MIPI 到 FPGA 芯片内部接收端的 IP，后续的处理就是关于 MIPI 接入数据格式的一些处理，目的是输出 MIPI 摄像头拍摄的视频到显示器。这里有一个解决方案：MIPI 的 RAW10 格式数据——axis_subset_converter——Sensor Demosaic——Gamma LUT——Video processing system（RAW2RGB）—嵌入的白平衡模块—Video processing system（Scale）——Video Frame Buffer Write —— axi data fifo——AXI Inteconnection——PS DDR——DP 接口——显示器。

6.1.3 实验步骤

1，根据第三章的“Hello world”基础配置工程，另存一份文档，重新命名 MIPI_DP_4G，对应光盘文档中的 MIPI_DP_4G.rar。

2，根据上述的解决方案建立 PL 端的数据处理链路，各个模块的配置这里不在详细说明，请根据工程中的实际情况进行配置就可。如下图所示：

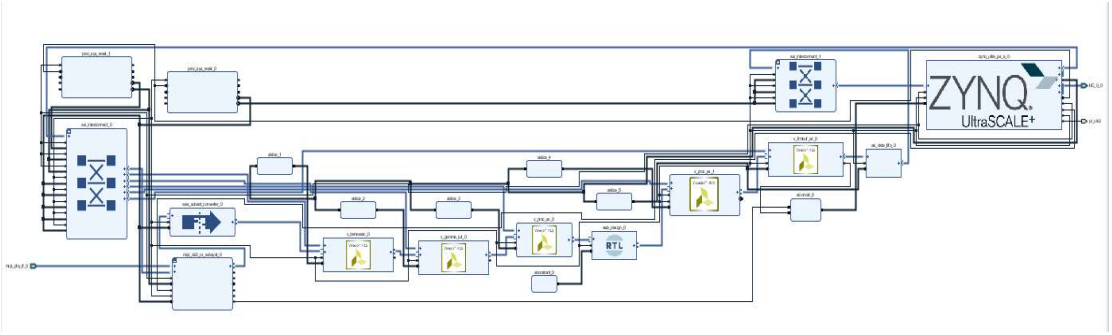


图 6-12 MIPI 视频采集设计框图

注意在工程中的时钟连线和复位，完成模块配置和连线之后，就可以进行物理约束和时序约束，因为总的工程资源消耗不是很大，可以自动选择布局布线，通过之后，没有时序违规即可生成 bitstream。然后导出硬件平台文档 XSA 文档。

后续就是 linux 系统的驱动开发，使用 AXI-lite 总线的模块都需要驱动开发，然后就是启动 boot.bin 文件的制作，制作流程在 linux 文档中有详细的说明，怎么使用 xsa 文档制作 BOOT.bin。这一块是软件相关的内容，大家可以参考我们的软件相关的文档。

更新过程不再详细介绍，最后的结果会出现更新成功的输出。

其次将硬件平台断电，设置成 QSPI 启动，重新上电之后，进入/mnt/mmcblk1p1,做好程序运行的必要工作。

在正确链接好硬件的 dp 线和显示器之后可以看到显示器的图像输出。

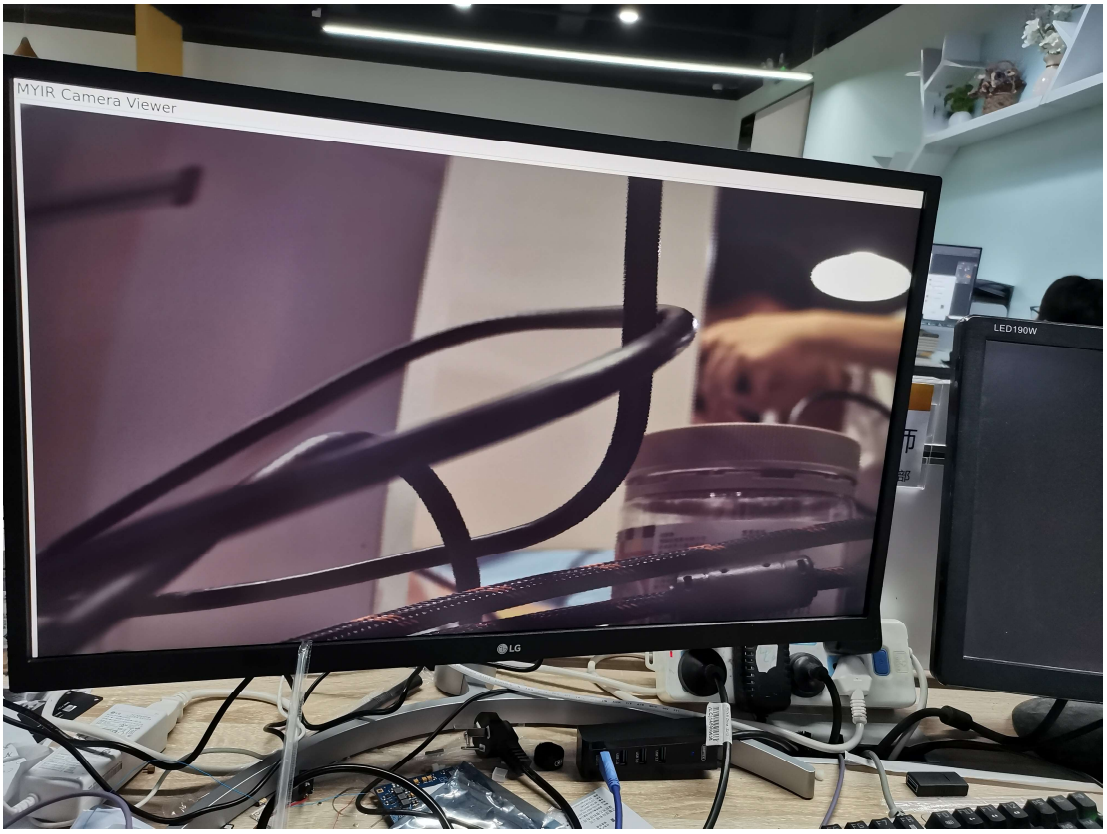


图 6-13 DP 显示器链接显示结果

6.1.4 白平衡模块的添加方式

在 vivado 工程中，添加了一个设计文件，然后再 block design 框图里面右键点击可以选择添加模块，如图

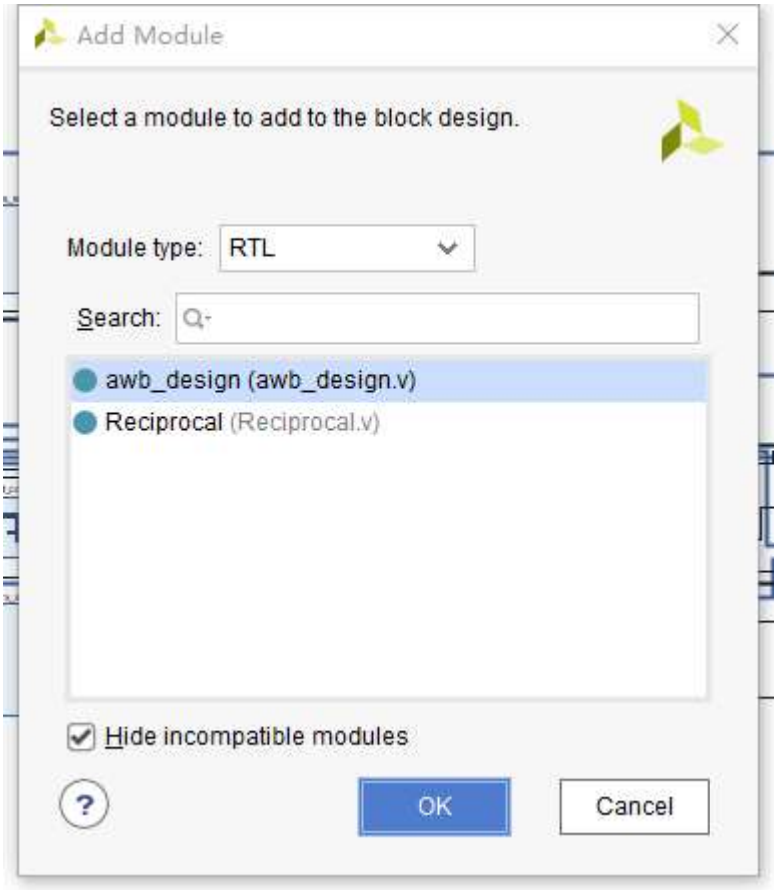


图 6-14 添加 RTL 模块的框图

这里添加的方式在 block design 中添加。添加的位置在转换成 RGB 格式的数据之后。也可以选择新建工程，自定义 IP，自定义 IP 有两种，第一章就是用户逻辑定义的 IP，第二种就是定义一个 AXI 外设。

6.2 VCU

6.2.1VCU 基础知识

IP H.264/H.265 视频编解码器单元 (VCU) 内核支持多标准视频编码和解码，包括支持高效视频编码 (HEVC) 和高级视频编码 (AVC) H.264 标准。该单元包含编码（压缩）和解码（解压缩）功能，并且能够同时进行编码和解码。

VCU 是所选 Zynq UltraScale+MPSoC 可编程逻辑 (PL) 中的集成块，与处理系统 (PS) 没有直接连接，并且包含编码器和解码器接口。 VCU 还包含促进 VCU 和 PL 之间接口的附加功能。 VCU 操作需要应用处理单元 (APU) 来服务中断以协调数据传输。编码器由 APU 通过预先准备好的任务列表进行控制，APU 响应时间不在执行关键路径中。VCU 没有音频支持。音频编码和解码可以通过 PL 中的软 IP 或者使用 PS 在软件中完成。下图显示了 VCU core 的顶层框图。

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

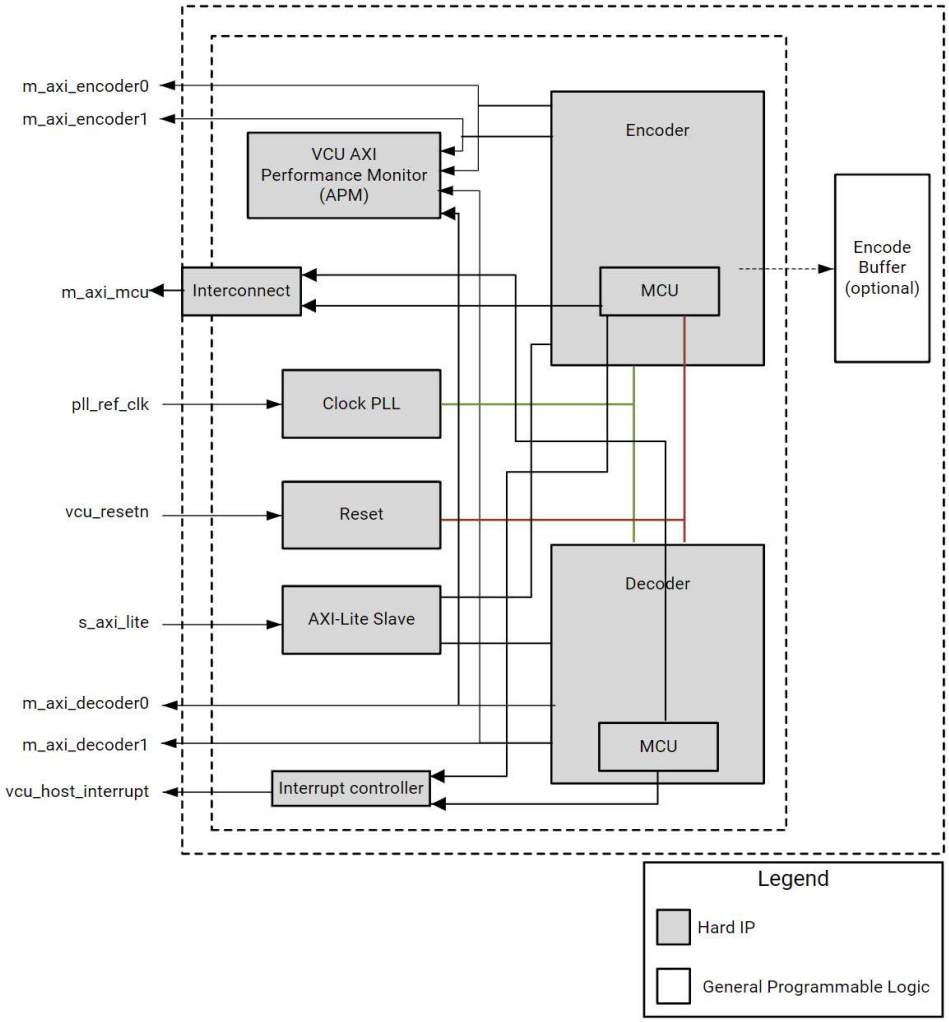


图 6-15 VCU IP core 的顶层框图

编码器引擎旨在使用 HEVC（ISO/IEC 23008-2 高效视频编码）和 AVC（ISO/IEC 14496-10 高级视频编码）标准处理视频流。它提供对这些标准的完整支持，包括支持 8 位和 10 位颜色、仅 Y 位（单色）、4:2:0 和 4:2:2 色度格式，以及高达 60 Hz 性能的 4K UHD。编码器包含全局寄存器、中断控制器和定时器。编码器由微控制器 (MCU) 子系统控制。在 APU 上运行的 VCU 应用程序使用 Xilinx VCU 控制软件库 API 与编码器微控制器进行交互。微控制器固件（MCU 固件）用户不可修改。APU 使用 32 位 AXI4-Lite 接口来控制 MCU（配置编码参数）。两个 128 位 AXI4 主接口用于将视频数据和元数据移入和移出系统存储器。一个 2 位的 AXI4 主接口用于获取 MCU 软件（指令缓存接口）和加载/存储额外的 MCU 数据（数据缓存接口）。

解码器模块能够使用 HEVC（ISO/IEC 23008-2 高效视频编码）和 AVC（ISO/IEC 14496-10 高级视频编码）标准处理视频流。它提供对这些标准的完整支持，包括支持 8

位和 10 位色深、Y-only（单色）、4:2:0 和 4:2:2 色度格式，在 60 Hz 性能下高达 4K UHD。它还包含全局寄存器、中断控制器和定时器。

VCU 解码器由微控制器 (MCU) 子系统控制。APU 使用 32 位 AXI4-Lite 从接口来控制 MCU。两个 128 位 AXI4 主接口用于将视频数据和元数据移入和移出系统存储器。一个 32 位 AXI4 主接口用于获取 MCU 软件（指令缓存接口）和加载/存储额外的 MCU 数据（数据缓存接口）。在 APU 上运行的 VCU 应用程序使用 Xilinx VCU 控制软件库 API 与解码器微控制器进行交互。微控制器固件不可由用户修改。

解码器包括控制寄存器、桥接单元和一组内部存储器。桥接单元管理解码器所需的所有外部存储器访问的请求仲裁、突发地址和突发长度。

编码器和解码器块均实现了一个 32 位微控制器单元 (MCU) 来处理与硬件块的交互。 MCU 接收来自 APU 的命令，将命令解析为多个切片或瓦片级命令，并在编码器和解码器块上执行它们。 命令执行后，MCU 将状态传达给 APU 并重复该过程。

VCU 内核是位于 PL 中的专用电路，可为广泛的用例选择提供最大的灵活性，内存带宽是关键驱动因素。 无论应用需要同时进行 60 Hz 编码和解码的 4K UHD 还是要处理单个 SD 流，都可以实施系统设计和存储器拓扑，以针对特定用例平衡性能、优化和集成。 下图显示了 VCU 内核与 PS 和 PL DDR 外部存储器配合使用的用例示例。

Figure 2: **VCU Application**

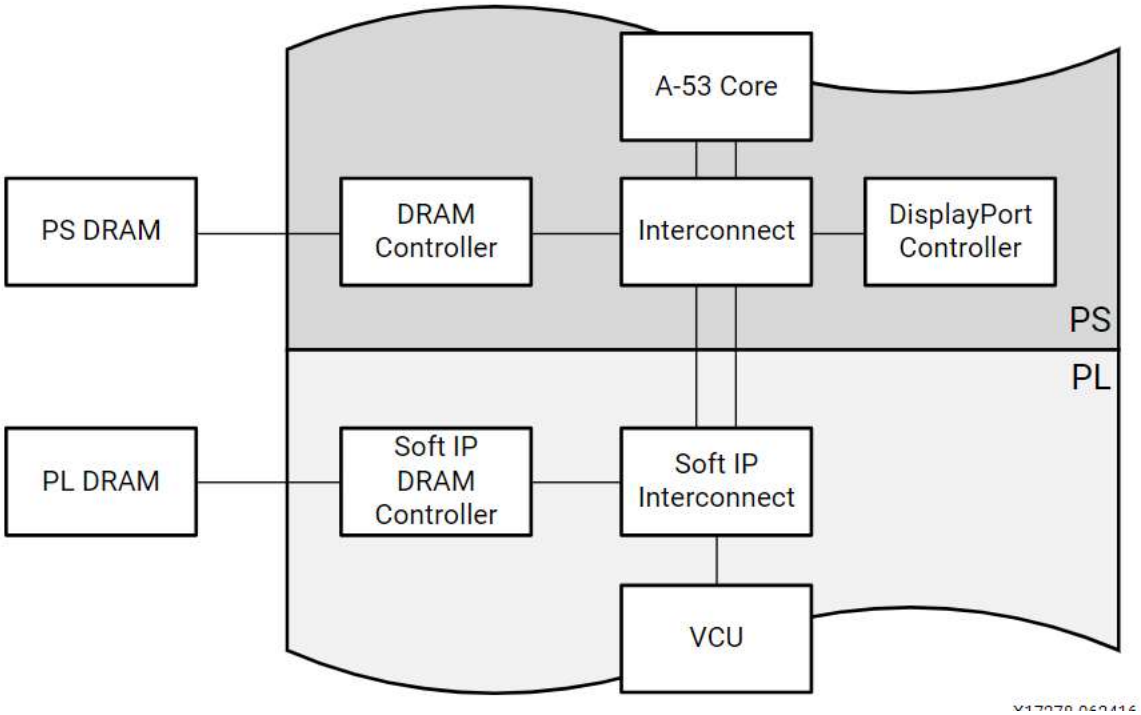


图 6-16 VCU 应用框图

Zynq UltraScale+ MPSoC 数据表：直流和交流开关特性 (DS925) 中描述了目标器件的典型时钟频率。系统可达到的最大时钟频率可能会有所不同。可实现的最大时钟频率和

所有资源计数可能受其他工具选项、器件中的其他逻辑、使用不同版本的 Xilinx 工具和其他因素的影响。

VCU 支持在 60 Hz 下同时编码和解码高达 4K UHD 分辨率。此吞吐量可以是 4K UHD 的单个流，也可以分成多达 32 个较小的流，最高 480p，频率为 30 Hz。如果累积吞吐量不超过 60 Hz 时的 4K UHD，则可以支持不同分辨率的 1 到 32 个流的几种组合。

60 Hz 的 4K UHD 流消耗大量外部存储器接口的带宽和处理系统和可编程逻辑之间的大量 Arm® AMBA® AXI4 总线带宽。对于同时编码器和解码器操作（包括转码用例），考虑使用 Xilinx PS 存储器控制器和专用 Xilinx VCU 存储器控制器。

下面介绍一下接口，下面的图给出的是 VCU 的 IP 框图，

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

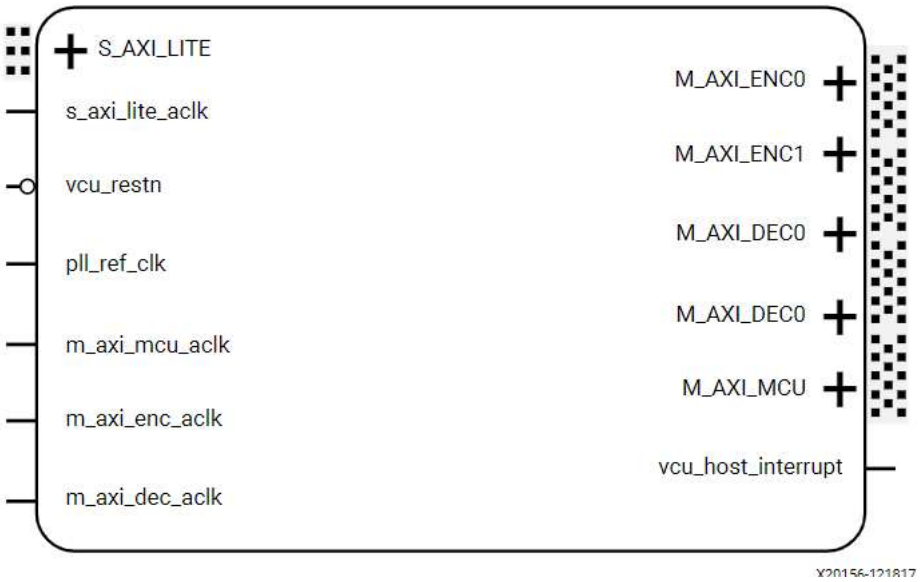


图 6-17 VCU IP 信号接口框图

其中的各个接口和端口信号如下所示：

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.

图 6-18 VCU 的接口以及详细描述

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

图 6-19 VCU 的端口以及方向详细描述

6.2.2 实验逻辑

根据 VCU 硬核的基础知识了解到：要使用 VCU 硬核的编解码功能，第一需要有视频数据输入到 VCU，视频的输入途径可以是外部通过 PL 端的 HDMI 接口输入，然后存储到 PL 端的 DDR，再从 DDR 通过 AXI4 总线读出输入到 VCU 的编码器接口，编码完成之后再写入到 DDR 中，PS 端只是作为一个简便的控制和 MCU 通信。使用解码的过程类似。另一个方式是使用 PS 端的 DDR 作为存储，因为 vcu 的 AXI 总线可以是读写模式。PS 端进行视频编码解码的控制中心，AXI 总线交互，控制和数据传输，不过占用了 PS 端的 DDR 带宽资源以及 AXI 总线资源。无论哪一种方式，VCU 的数据来源和编解码完成的数据都是存入 DDR。本实验使用的就是 PS 端作为数据存取以及控制中心的方式进行操作，具体的工程实施方案如下：

第一 VCU 读取的数据是从 PS 端 DDR，编码完成之后的数据存入 PS 端的 DDR。

第二 VCU 解码的数据是从 PS 端 DDR，解码完成之后的数据存入 PS 端的 DDR。

中间从 VCU 编解码完成的数据会经过一个 axi register slice 模块，这个模块的功能：通过一组流水线寄存器将一个 AXI 内存映射主设备连接到一个 AXI 内存映射从设备，通常用于中断关键时序路径。类似于 AXI Interconnect 的作用。

至于 VCU 的内部结构就不详细介绍了，客户可以参考 VCU 的产品文档。

6.2.3 实验步骤

1，根据第三章的“Hello world”基础配置工程，另存一份文档，重新命名 project_fz5_vcu_707，对应光盘文档中的 project_fz5_vcu_707.rar。

2，根据上述的解决方案建立 PL 端的数据处理链路，各个模块的配置这里不在详细说明，请根据工程中的实际情况进行配置就可。如下图所示

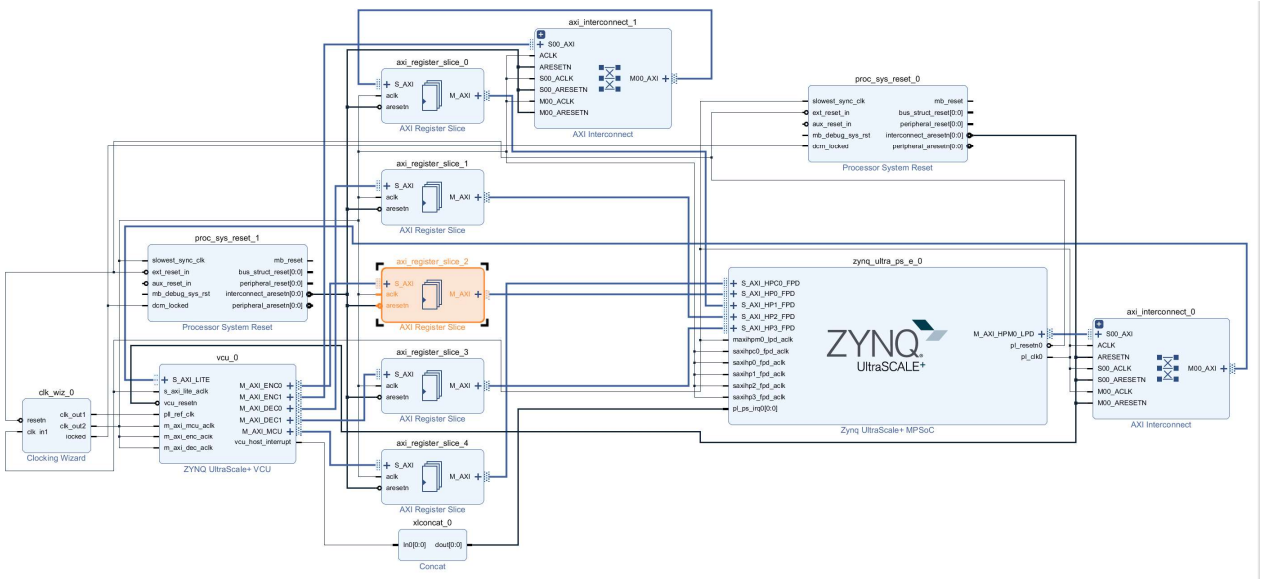


图 6-20 VCU 的 BD 工程链接框图

3，这里详细介绍一下时钟系统：控制路线的时钟是 PS 端输出到 PL 端的 100MHz 时钟，而 VCU 工作需要的时钟根据使用的不同而不同，这里我们从 PLL 中的输出 clk_out2 就是给 VCU 使用的 332MHz 的时钟，也就是 m_axi_mcu、m_axis_enc、m_axi_dec 三个时钟的输入。PLL 的参考时钟是 33MHz，这是作为整个 VCU 的参考时钟输入，也就是 pll_ref_clk。

4，关于数据路径：从工程的框图可以看到，VCU 没有单独的数据输入的接口，它都是直接从内存 DDR4 读取数据，编码或者译码完成之后就直接将数据写入 DDR4，从而可以确定，VCU 内部是有一个 DDR4 SDRAM 控制器，根据官方 PG252 的说法这个控制器是定制的，不可以用于其他的方面。

当工程家里完成之后就可以参考软件使用文档中关于 VCU 部分的内容，在软件 PS 端有详细的内容介绍以及操作细节。

6.3 本章小节

本章节主要介绍了 MIPI 的使用和一些入门的物理层协议知识，更进一步的学习需要查看专门的物理层协议文档。新引入的知识就是添加自定义的 RTL 模块，或者另外新建工程定义一个 IP。然后添加 IP 的生成路径到 MIPI 工程中，这样都可以将我们的逻辑设计嵌入进整个方案设计流程中。

第七章 如何使用 xilinx 官方资料

7.1 用户手册

7.1.1 用户手册的阅读方式

在设计的流程中，我们通常会遇到一些问题，有时候这些问题是因为我们不清楚具体的 IP 的时序或者一些特定的配置，所以我们就需要去查看用户手册，如何在用户手册快速找到我们需要的信息是关键。比如我们需要查找一下 PS-GTR 的接口情况，以及如何进行验证。

打开一个用户手册，如下：

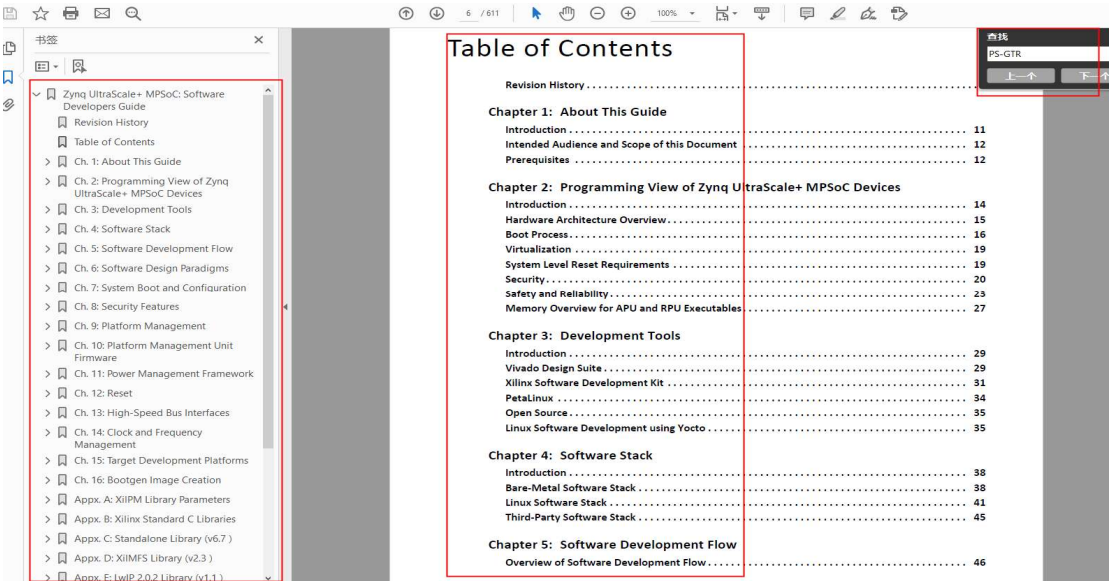


图 7-1 xilinx 用户手册的目录结构

在打开用户手册之后可以看到目录中的内容，有的用户手册内容少，有的比较多，如果我们需要迅速找到对应的内容，需要客户有一定的英文阅读能力，其次是会使用对应的查找工具。点击下一个：

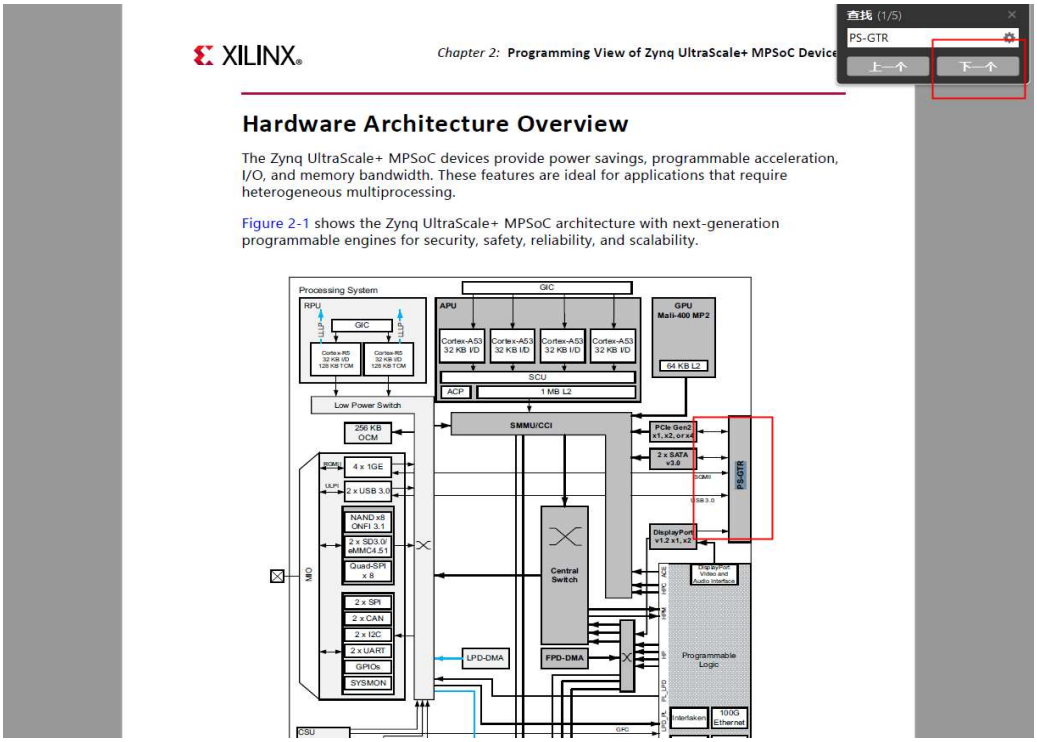


图 7-2 跳转到制定的章节

可以找到对应在框图中出现的 PS-GTR 所在位置，链接的接口之类的，再次点击下一个：

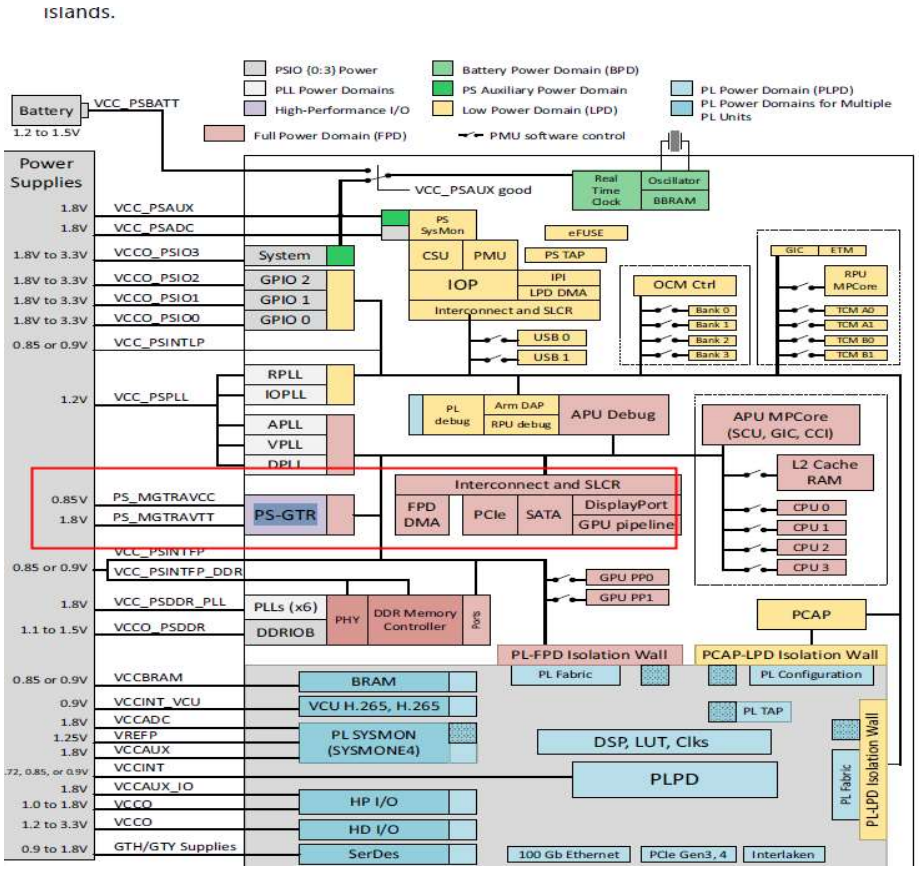


图 7-3 查找需要的接口跳转

看到对应的电平标准，以及 PS-GTR 下一个出现的地方：

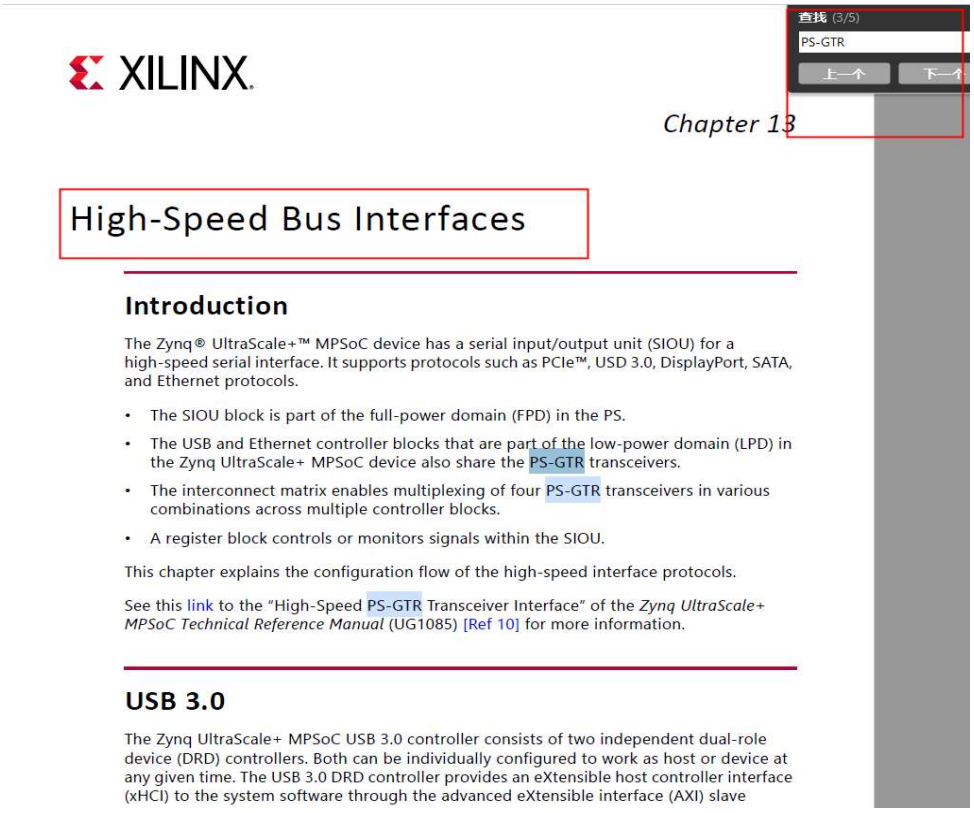


图 7-4 查找高速总线接口章节

在文档中的第 13 章，点开第 13 章的目录可以看到，使用了 PS-GTR 这个高速串行接口的物理接口有哪些，从而可以找到对应的介绍，其次就是使用文档查找器，找到对应验证调试文档：

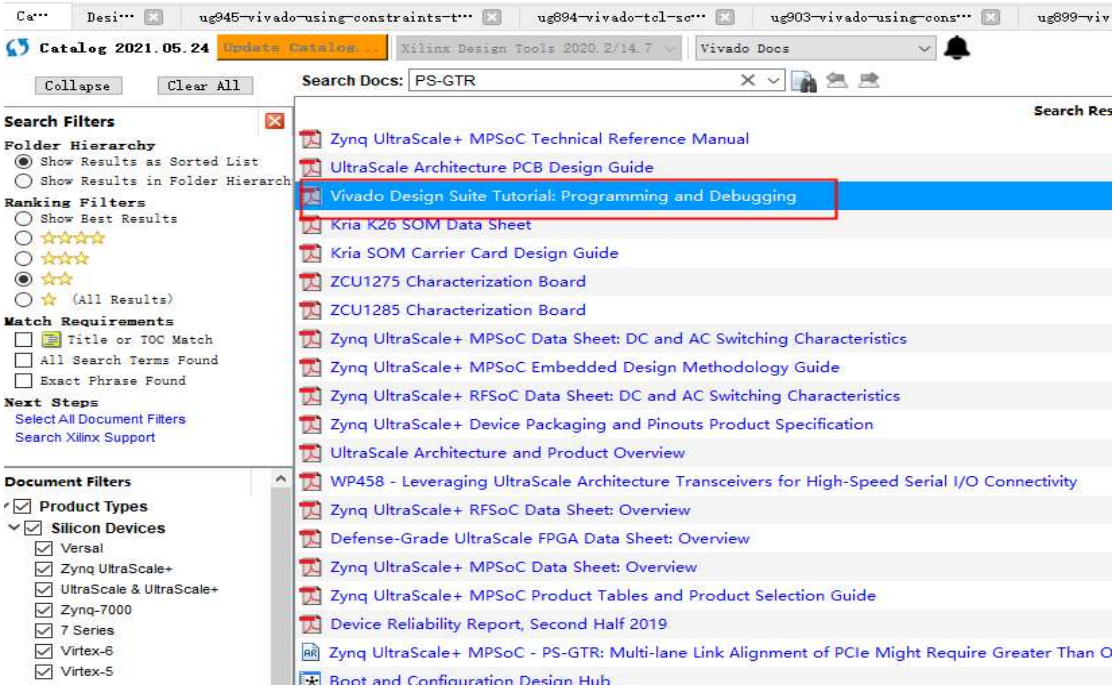


图 7-5 DocNav 查找 PS-GTR 过滤结果

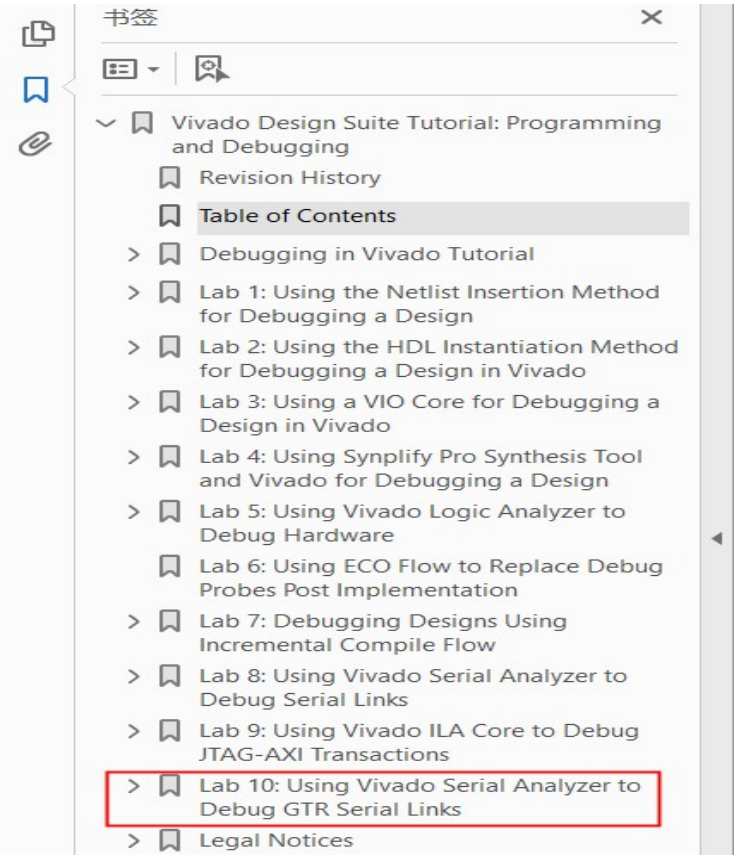


图 7-6 找到对应文档实验-分析 PS-GTR 链接实验

7.2 产品手册

7.2.1 产品手册的阅读方式

在设计的流程中，我们通常会遇到一些问题，有时候这些问题是因为我们不清楚具体的 IP 的时序或者一些特定的配置，这个时候需要阅读对应 IP 的产品手册来查找信息，比如查找 MIPI-CSI2-Subsystem 的产品手册，看看这个 IP 如何配置成 4lane，物理层协议配置等等之类的详细信息。

首先找到产品手册，如下：

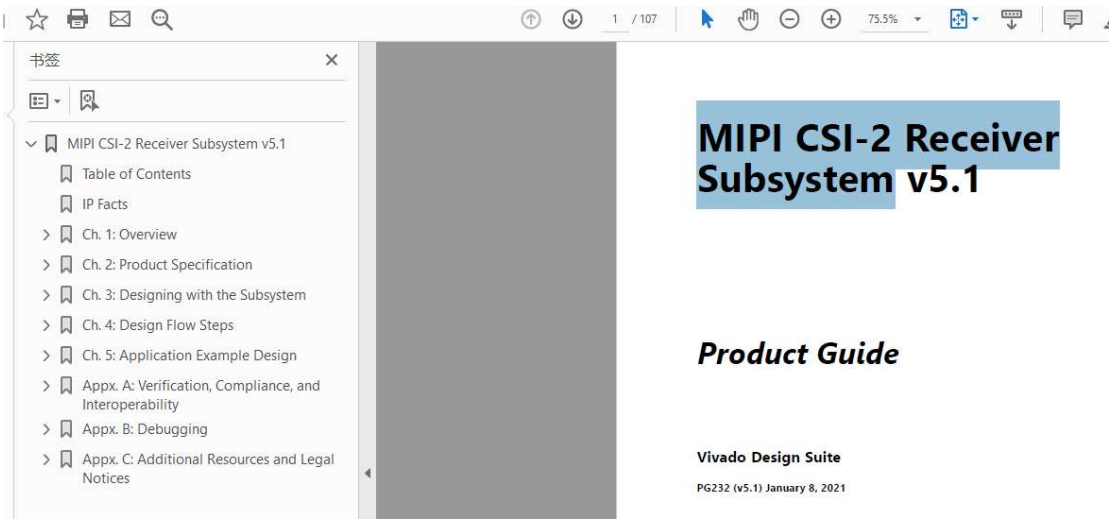


图 7-7 IP 的产品手册

从目录中可以看到相应的介绍这个 IP 的内容，包括概要、物理层结构框图、使用的物理层协议、接口以及时序、以及 IP 配置流程等等。可以找到我们需要的信息。

7.3 参考设计

7.3.1 参考设计的 vivado 工程使用

在 xilinx 的官网上有一些开发套件，而且对应的开发套件也做了一些参考设计，这些参考设计通常都是针对某一个接口或者协议做的 demo，使用了一些硬件模块，如果需要复现这个 demo 的话，需要对应的硬件环境，所以我们一般只是进行参考，比如关于 UlteraScale+ MPSOC 系列的一些参考设计放在了维基百科，地址 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/444006775/Zynq+UltraScale+MPSoC；

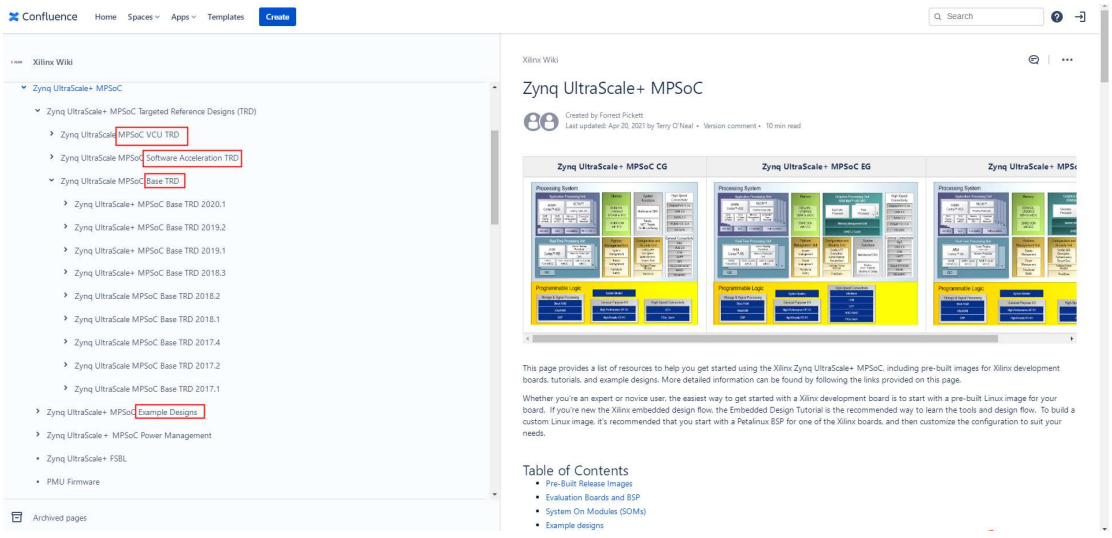


图 7-8 参考设计位置

图 7-9 参考设计分类

找到对应的参考设计之后，下载对应的文档，然后打开 vivado，按照下载的参考设计资料中 readme 里面的介绍，找到对应的工程文档 Tcl 文档，在 vivado 下使用对应的命令进行 vivado 工程重建：

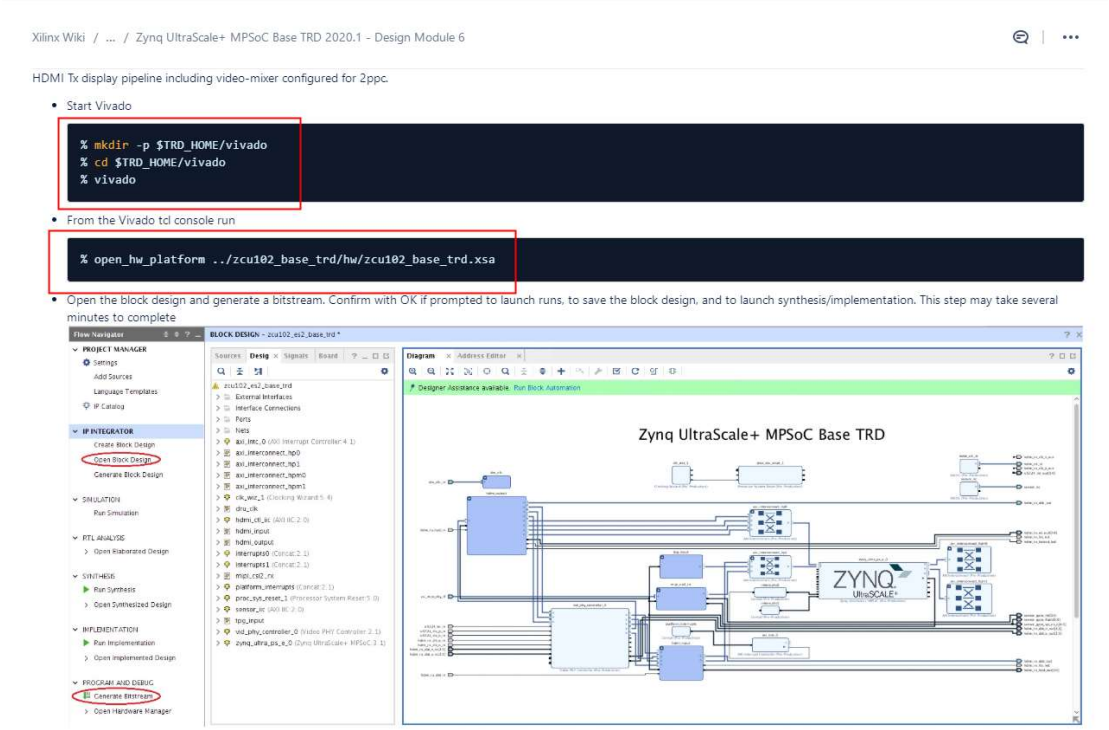


图 7-10 找到参考设计实用 tcl 文档生成的工程

然后就可以查看工程中对应的一些设计思想和实际配置。

参考设计可以帮我们解决一些对于设计中 IP 的配置、IP 模块顺序、以及时钟方面的参考。

7.3Xilinx 中文/英文社区论坛

7.3.1 在社区提问

关于设计论坛的使用，可以像正常的论坛那样提问，xilinx 的技术支持工程师在线时间是工作时间，一般不会加班来解决，当然在上面发问，有时热心的网友也会给出很好的解答。

或者搜索到同样的提问，就可以看到前人怎么解决类似的问题。

如下所示

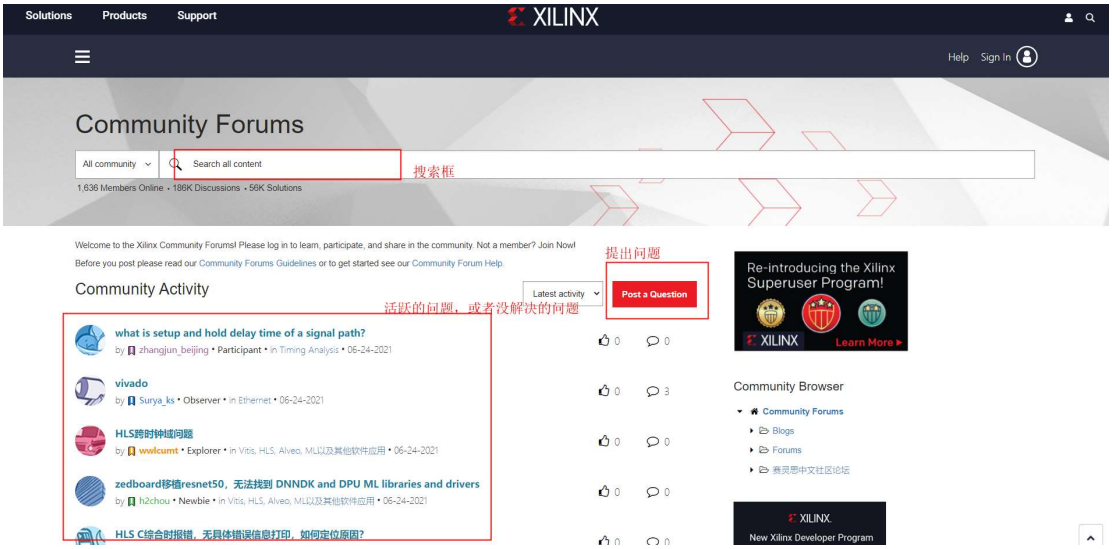


图 7-11 社区论坛首页

比如我们查找关于 10GEthernet subsystem 方面的问题

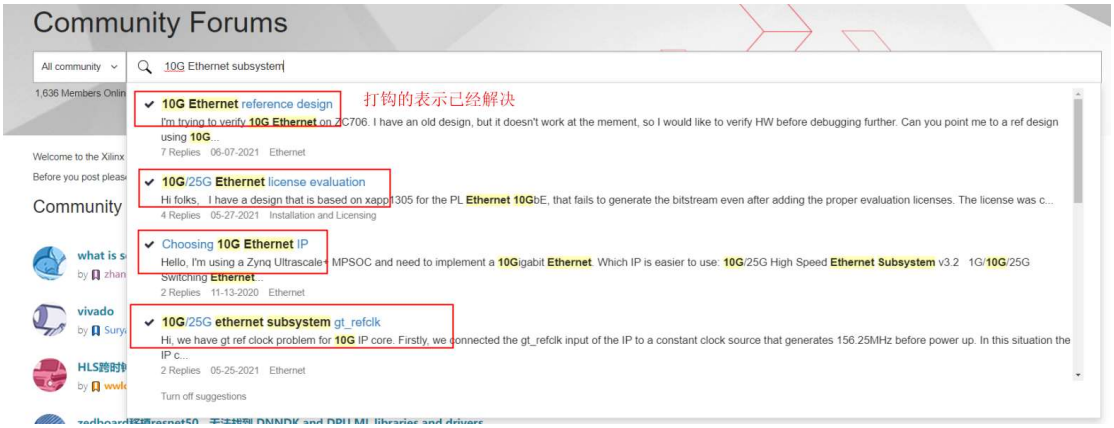


图 7-12 搜索类似的问题

我们可以看到许多关于 10GEthernet subsystem 方面的提问和被解决的提问。

第八章 总结

8.1 总结

本文档给使用本硬件平台的客户做了一些介绍，浅显的讲了一些关于学习 FPGA 方面的知识和问题，当然还有更多的需要学习，主要在一下几个方面：

1，关于 verilog 的语法，需要进一步的深入学习，跟进一步的可以将 system verilog 进行一个系统的学习。

2，关于 Tcl 这一块，对于从事大型复杂设计的人员来说，掌握一个脚本工具非常有利于提升工作效率。

3，关于约束方面的知识，这里没有提供具体的工程例子给出介绍，限于篇幅，不过当参考 xilinx 官方关于 constrains 的一些文档以及关于时序收敛方面的文档和示例工程，应该就有一个比较深入的了解。

4，关于 AXI 总线、接口、协议方面的知识建议参考 ARM 官方的协议文档。

5，关于我们给出的示例工程，有的比较浅显有的比较复杂，建议客户认真学习。

最后，希望客户学习本文档之后能有所收获。

参考资料

- Xilinx 官方 WiKi：https://xilinx-wiki.atlassian.net/wiki/home
- DocNav 查找器
- ds891-zynq-ultrascale-plus-overview.pdf
- ug1137-zynq-ultrascale-mpsoc-swdev.pdf
- ug1085-zynq-ultrascale-trm.pdf
- Verilog 使用手册
- MYS-ZU5EV 原理图
- MYS-ZU5EV 硬件手册
- 小梅哥 FPGA 教程
- 威三学院 FPGA 教程
- 吴厚航. 深入浅出玩转 FPGA[M]. 北京航空航天大学出版社, 2013.
- 夏宇闻. Verilog 数字系统设计教程.第 3 版[M]. 北京航空航天大学出版社, 2013.
- 韩彬, 于潇宇, 张雷鸣. FPGA 设计技巧与案例开发详解[M]. 电子工业出版社, 2014.
- TclTk 入门经典_第 2 版_中文.pdf
- Vivado Design Suite User GuideDesign Analysis andClosure 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

附录一 联系我们

深圳总部

负责区域：广东 / 四川 / 重庆 / 湖南 / 广西 / 云南 / 贵州 / 海南 / 香港 / 澳门
电话：0755-25622735 18924653967
地址：深圳市龙岗区坂田街道发达路云里智能园 2 栋 6 楼 604 室

上海办事处

负责区域：上海 / 湖北 / 江苏 / 浙江 / 安徽 / 福建 / 江西
电话：021-62087019 18924632515
地址：上海市普陀区中江路 106 号北岸长风I 座 302

北京办事处

负责区域：北京/天津/陕西/辽宁/山东/河南/河北/黑龙江/吉林/山西/甘肃/内蒙古/宁夏
电话：010-84675491 13316862895
地址：北京市昌平区东小口镇中滩村润枫欣尚 1 号楼 505

销售联系方式

网址：www.myir-tech.com
邮箱：sales.cn@myirtech.com

技术支持联系方式

电话：0755-22316235（深圳），027-59621647/027-59621648（武汉）
邮箱：support.cn@myirtech.com

如果您通过邮件获取帮助时，请使用以下格式书写邮件标题：

[公司名称/个人--开发板型号]问题概述

这样可以使我们更快速跟进您的问题，以便相应开发组可以处理您的问题。

附录二 售后服务与技术支持

凡是通过米尔电子直接购买或经米尔电子授权的正规代理商处购买的米尔电子全系列产品，均可享受以下权益：

1、6 个月免费保修服务周期
2、终身免费技术支持服务
3、终身维修服务
4、免费享有所购买产品配套的软件升级服务
5、免费享有所购买产品配套的软件源代码，以及米尔科技开发的部分软件源代码
6、可直接从米尔科技购买主要芯片样品，简单、方便、快速；免去从代理商处购买时，漫长的等待周期
7、自购买之日起，即成为米尔科技永久客户，享有再次购买米尔科技任何一款软硬件产品的优惠政策
8、OEM/ODM 服务

如有以下情况之一，则不享有免费保修服务：
1、超过免费保修服务周期
2、无产品序列号或无产品有效购买单据
3、进液、受潮、发霉或腐蚀
4、受撞击、挤压、摔落、刮伤等非产品本身质量问题引起的故障和损坏
5、擅自改造硬件、错误上电、错误操作造成的故障和损坏
6、由不可抗拒自然因素引起的故障和损坏

产品返修：

用户在使用过程中由于产品故障、损坏或其他异常现象，在寄回维修之前，请先致电米尔科技客服部，与工程师进行沟通以确认问题，避免故障判断错误造成不必要的运费损失及周期的耽误。

维修周期：

收到返修产品后，我们将即日安排工程师进行检测，我们将在最短的时间内维修或更换并寄回。一般的故障维修周期为 3 个工作日（自我司收到物品之日起，不计运输过程时间），由于特殊故障导致无法短期内维修的产品，我们会与用户另行沟通并确认维修周期。

维修费用：

在免费保修期内的产品，由于产品质量问题引起的故障，不收任何维修费用；不属于免费保修范围内的故障或损坏，在检测确认问题后，我们将与客户沟通并确认维修费用，我们仅收取元器件材料费，不收取维修服务费；超过保修期限的产品，根据实际损坏的程度来确定收取的元器件材料费和维修服务费。

运输费用：

产品正常保修时，用户寄回的运费由用户承担，维修后寄回给用户的费用由我司承担。非正常保修产品来回运费均由用户承担。