HLT-XCZU67DR-B 开 发 套 件 资 料

硬件参考手册

测试快速使用指南

windows的VIVADO及驱动安装

VIVADO新建工程流程和软件介绍

最简单的LED流水灯设计

VIVADO仿真的使用之呼吸灯

在线调试进阶之MARK DEBUG

08petalinux开发简介

光口开发之Ibert眼图及光通信测试

PS处理器配置及之helloword打印

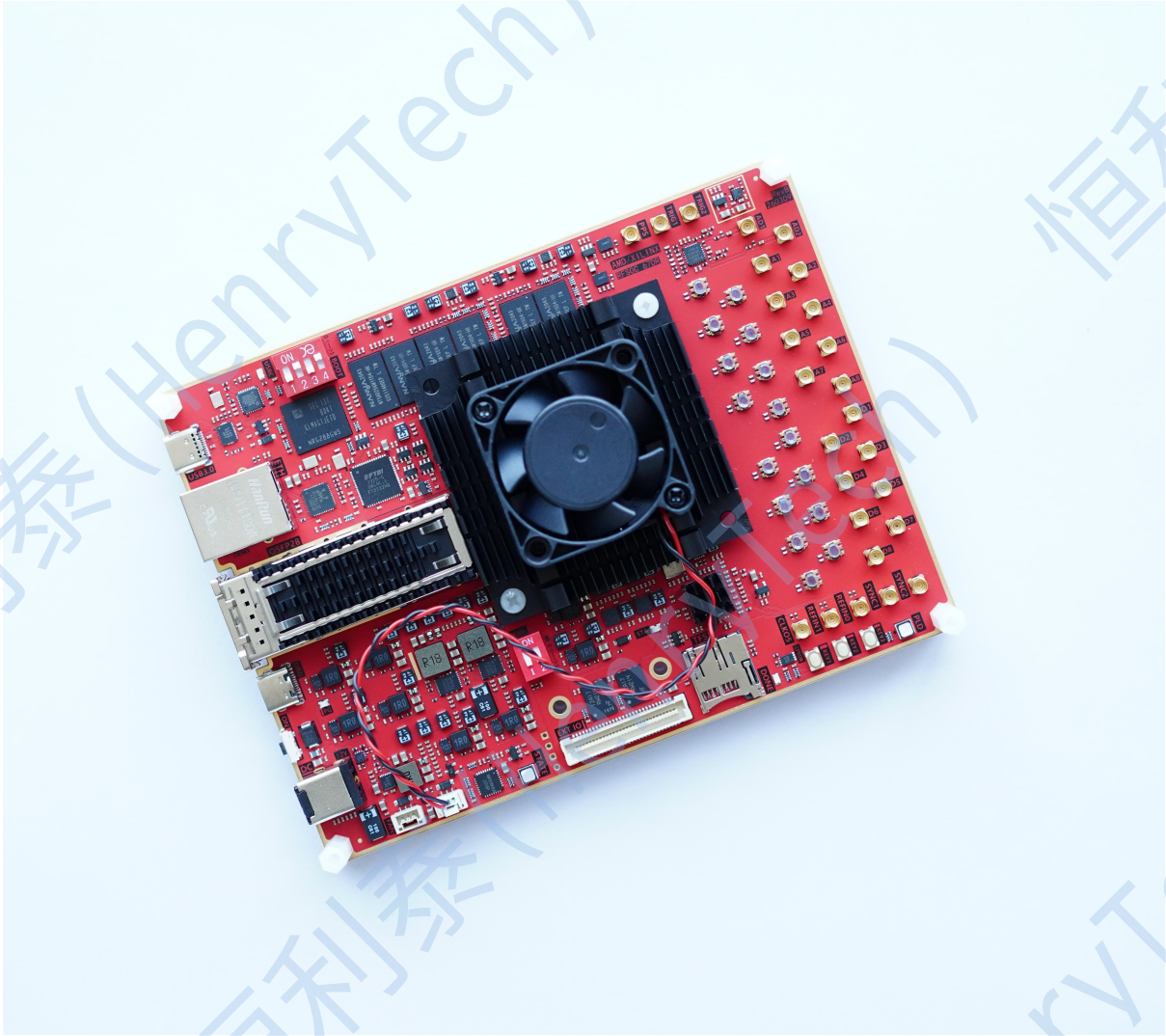
使用我们的PS处理器运行RTOS系统

LMK04828时钟配置

RF_Data_Converter射频回环实验

VIVADO对设计工程的芯片功耗分析和报告

HLT-XCZU67DR-B 硬件参考手册



------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

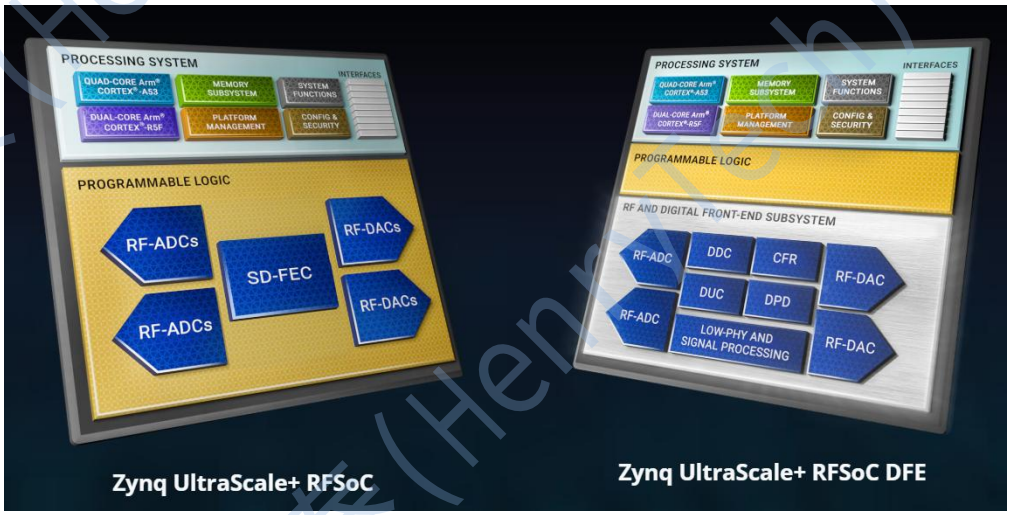
目录

一、 赛灵思RFSOC简介 3
二、 文档概述 4
三、 硬件接口功能 4
四、 电源 5
五、 电源管理MCU 6
六、 DEBUG调试部分 6
七、 PS DDR4部分 7
八、 PL DDR4部分 8
九、 EMMC / SD / EEPROM/FLASH存储 10
十、 100GE QSFP28光口 11
十一、 开发板时钟部分 12
十二、 开发板USB3.0部分 13
十三、 LED和按钮 14
十四、 PS的千兆网口 15
十五、 扩展接口 15
十六、 同步/触发/PPS接口 16
十七、 射频ADC部分 16
十八、 射频DAC部分 17
十九、 尺寸标注 18
二十、 其他重要说明 18

恒利泰(HenryTech)

一、赛灵思RFSOC简介

RFSOC是原XILINX公司（现已被AMD收购）的ZYNQ Ultrascale+ RFSOC系列针对射频5G等应用的一款SOC芯片，其集成了高速ADC和DAC（14位），其中芯片自身的采样率最大ADC速度为5.9GSPS，DAC最大为10GSPS。其最大特点就是可以进行前端射频信号直接采样，省去变频等复杂操作，可以最大可能保留其高速特性。其处理器提供了四核cortex A53以及双核cortex R处理器，提供强悍的软件处理性能。并且集成的ultrascale+架构的FPGA，几百K资源，满足常规数据预处理能力。由于RFSOC分为好几代，其中以2系列为例，典型的有27DR,29DR等，3系列有39DR等，4系列有47DR，49DR等，然后是6系列，典型的有63DR和67DR。6系列官方属于RFSOC DFE器件，增加了一些射频片上功能以及处理单元，除此之外，处理器部分都是一样的：



本文档配套的使用67DR器件，我们开发板RF67DR-Lite完整型号HLT-XCZU67DR-B，属于第4代DFE产品，我们从选型手册可以看到如下表格，文档名字《zynq-usp-rfsoc-product-selection-guide.pdf》：

Device Name		ZU63DR		ZU64DR		ZU65DR	ZU67DR	
							DFE	
		Quad-core Arm® Cortex®-A53 MPCore™ up to 1.3GHz, **Dual-core** Arm Cortex-R5F MPCore up to 533MHz						
14-bit RF-ADC w/DDC	# of ADCs	4	2	8	2	6	8	2
	Max Rate (GSPS)	2.95	5.9	2.95	5.9	5.9	2.95	5.9
14-bit RF-DAC w/DUC	# of DACs	4		8		6	8	
	Max Rate (GSPS)	10.0[3]		10.0[3]		10.0[3]	10.0[3]	
SD-FEC		0		0		0	0	
Digital Front-End Hard IP (DFE IP)		Channel Filter, DUC/DDC, Mixer, CFR, Complex Equalizer, PQ Resampler, DPD						
Low PHY Hard IP		FFT/iFFT, PRACH		None		FFT/iFFT, PRACH	FFT/iFFT, PRACH	
Number of DDCs per RF-ADC[1]		1		1		1	1	
RF input Freq max. GHz		7.125						
Decimation / Interpolation		1x, 2x, 3x, 4x, 5x, 6x, 8x, 10x, 12x, 16x, 20x, 24x, 40x						
System Logic Cells (K)		393		328		489	489	
CLB LUTs (K)		180		150		224	224	
Max. Dist. RAM (Mb)		5.47		4.56		6.8	6.8	
Total Block RAM (Mb)		17.6		15.8		22.8	22.8	
UltraRAM (Mb)		36.6		22.5		45.0	45.0	
DSP Slices		1,200		1,872		1,872	1,872	
GTY Transceivers		4		8		8	8	
PCIe® Gen3 x16/Gen4 x8 / CCIX[2]		0		0		0	0	
150G Interlaken		0		0		0	0	
100G Ethernet MAC/PCS w/RS-FEC		1		1		1	1	
System Monitor		2		2		2	2	
Speed Grades		-1I, -1LI, -2I, -2LI		-1I, -1LI, -2I, -2LI		-1I, -1LI, -2I, -2LI	-1I, -1LI, -2I, -2LI	
Package Footprint	Package Dimensions	PSIO, HDIO, HPIO GTR, GTY RF-ADC, RF-DAC		PSIO, HDIO, HPIO GTR, GTY RF-ADC, RF-DAC		PSIO, HDIO, HPIO GTR, GTY RF-ADC, RF-DAC	PSIO, HDIO, HPIO GTR, GTY RF-ADC, RF-DAC	
E1156	35x35	214, 24, 130 4, 8 6, 4		214, 24, 130 4, 8 10, 8		214, 24, 130 4, 8 6, 6	214, 24, 130 4, 8 10, 8	

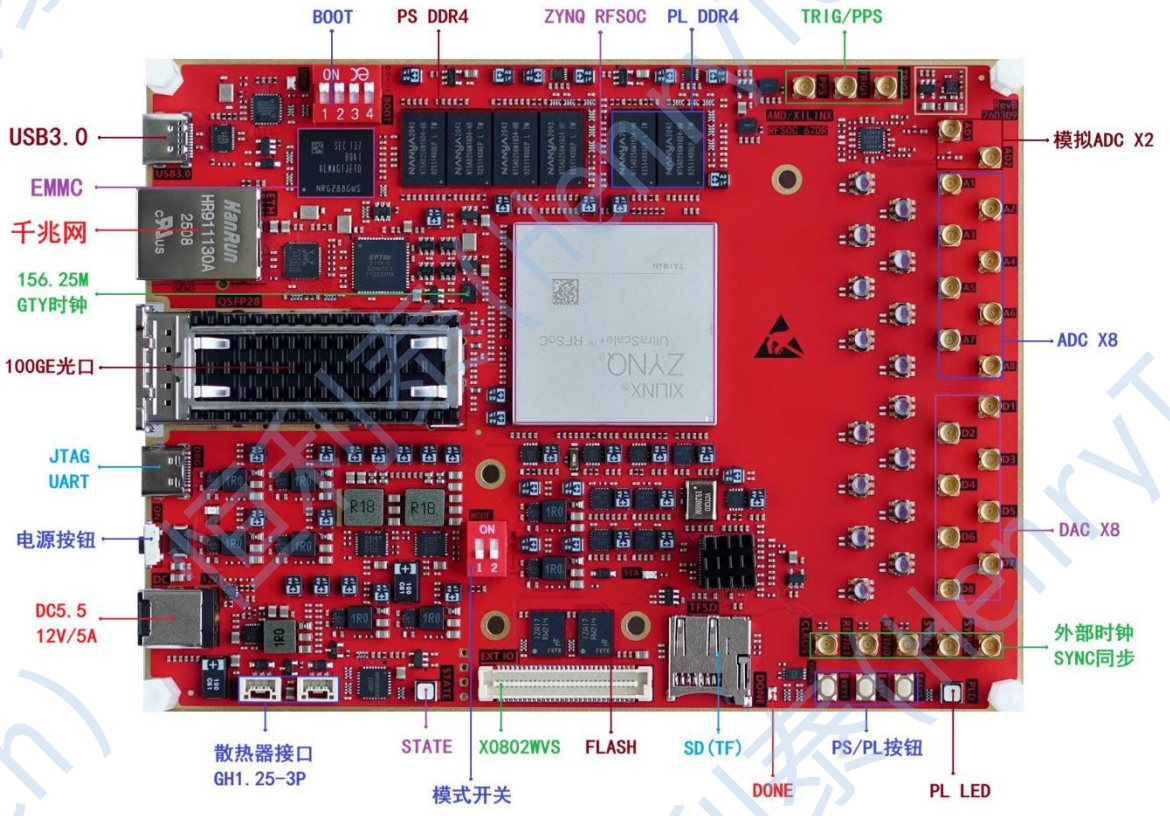
从上方表格可以看到，67DR包含8通道2.95G的ADC和2通道的5.9G采样率ADC，以及8通道最大10GSPS采样率的DAC，资源489K的FPGA。

二、文档概述

本文档对本开发板规格设计进行概述性讲解。本文档对开发板的主要部分硬件参数和接口等定义描述，通过本文档，针对用户自定义开发起到快速参考的目的。如本手册未提及的细节，大家可以直接参考原理图PDF文件即可。更多关于开发板的出厂开箱测试可以参考文档《02RF67DR-Lite快速测试使用指南》。

三、硬件接口功能

硬件各部分功能接口总体如下，通过这个图可以对本产品每个功能接口有个大致的了解，可以快速知道每个功能和接口在哪里：



开发板主要包含电源及按钮，指示灯，数据接口，扩展口，存储，射频ADC DAC及时钟接口等等。对于总体功能参考此图。

四、电源

开发板电源各路型号及参数如下，电源位于原理图第一部分和第二部分：

	电源名	型号	电压	最大电流
主电源	VCC5.0V	MP8774H	5.0V	12A
	VCC_INT	MPQ8645	0.85V	60A
ARM/FPGA	MGT_AVCC	MP2145	0.9V	6A
	MGT_AVTT	MP2145	1.2V	6A
	DDR_1V2	MP2145	1.2V	6A
	VCC_AUX	MP2145	1.8V	6A
	VCC_1V8	MP2145	1.8V	6A
	VCC_3V3	MP2145	3.3V	6A
	PS_AVCC	TPS74901	0.866V	3A
	PS_AVTT	TPS74901	1.8V	3A
	MGTVCCAUX	TPS74901	1.8V	3A
	PS_DDR4_VPP	LP5907	2.5V	0.3A
	PL_DDR4_VPP	LP5907	2.5V	0.3A
	VCC_1V1	MP2145	1.1V	6A
	VCC_2V0	MP2145	2.0V	6A
	DAC_AVTT	TPS74901	2.5V	3A
AD DA部分	ADC_AVCC	TPS74901	0.925V	3A
	DAC_AVCC	TPS74901	0.925V	3A
	ADC_AVCCAUX	TPS74901	1.8V	3A
	DAC_AVCCAUX	TPS74901	1.8V	3A

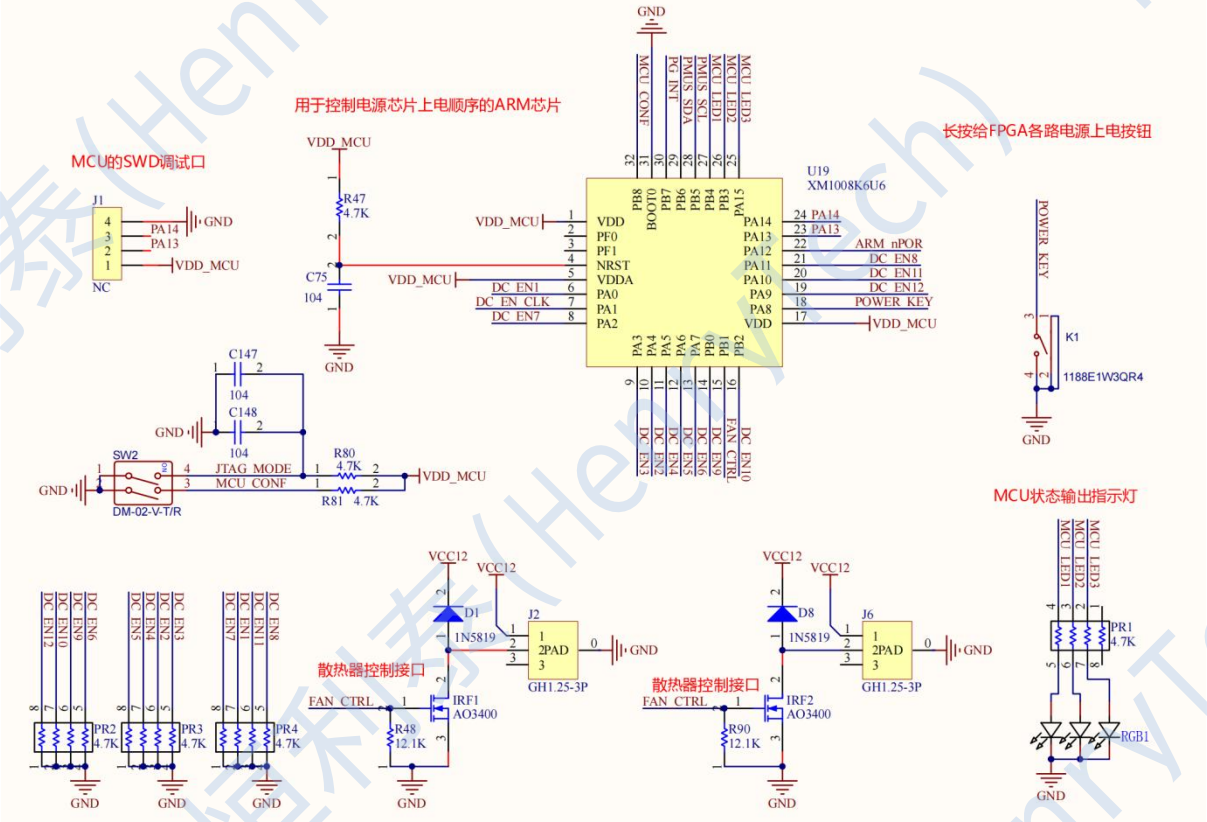
表格中，仅列出主电源。其中还有一些比如时钟，调试JTAG供电，以及电源管理MCU控制器电源没有列出。上述表格除了5V电源通电即工作，其余需要MCU进行上电顺序控制。电源管理MCU上电后直接工作。各路电源上电顺序为：

VCC_INT->MGT_AVCC->MGT_AVTT->VCC_AUX->DDR_1V2->

VCC_1V8/VCC_3V3/VCC_CLK/PS_DDR4_VPP/PL_DDR4_VPP/VCC_AVCC_AUX/PS_AVTT/PS_AVCC.DAC相关电源无上电顺序要求：ADC的ADC_AVCCAUX->ADC_AVCC。

五、电源管理MCU

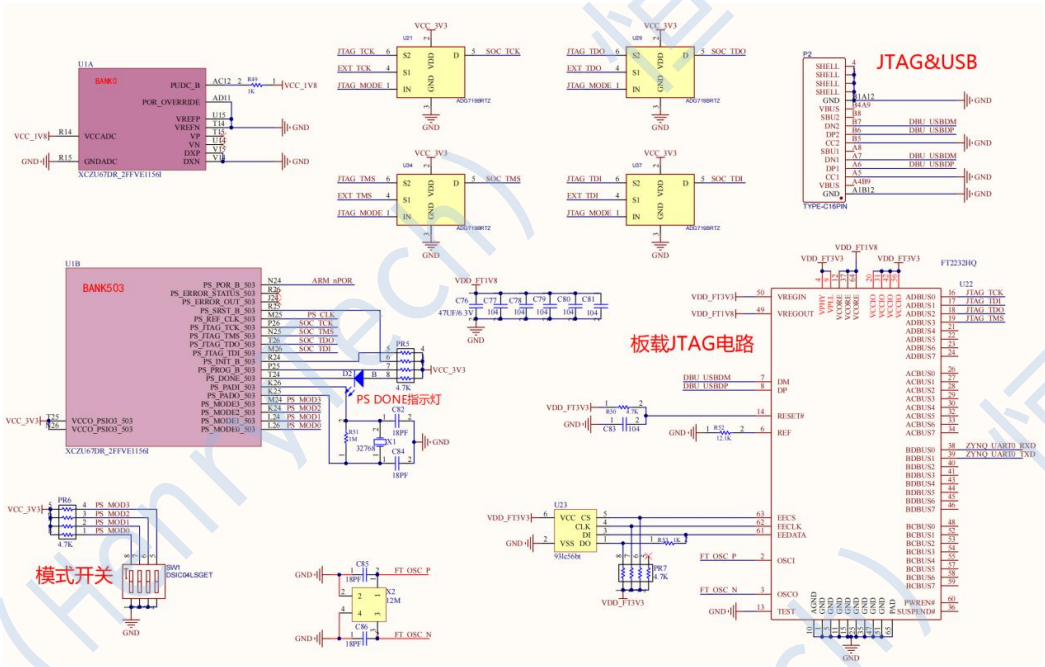
电源管理MCU位于原理图第五部分，我们使用一颗兼容STM32F031的国产替代型号的器件XM1008K6U6，进行电源上电控制以及核心电源配置。除了电源控制，还做按钮电子开关长按开关机以及电源状态指示灯和散热器风扇控制，以及拨码开关SW2可做开机模式切换，支持长按开机和通电自动启动设置，并且拨码开关还可以作为RFSOC 67DR的JTAG调试接口切换，用户可以通过JTAG_MODE设置JTAG通过板载FT2232HQ还是通过BTB 60P扩展IO的外接JTAG进行调试。这部分电路如下：



K1按钮主要用作长按开机/关机 （SW2一段设置为长按开机），短按复位主芯片的功能。

六、DEBUG调试部分

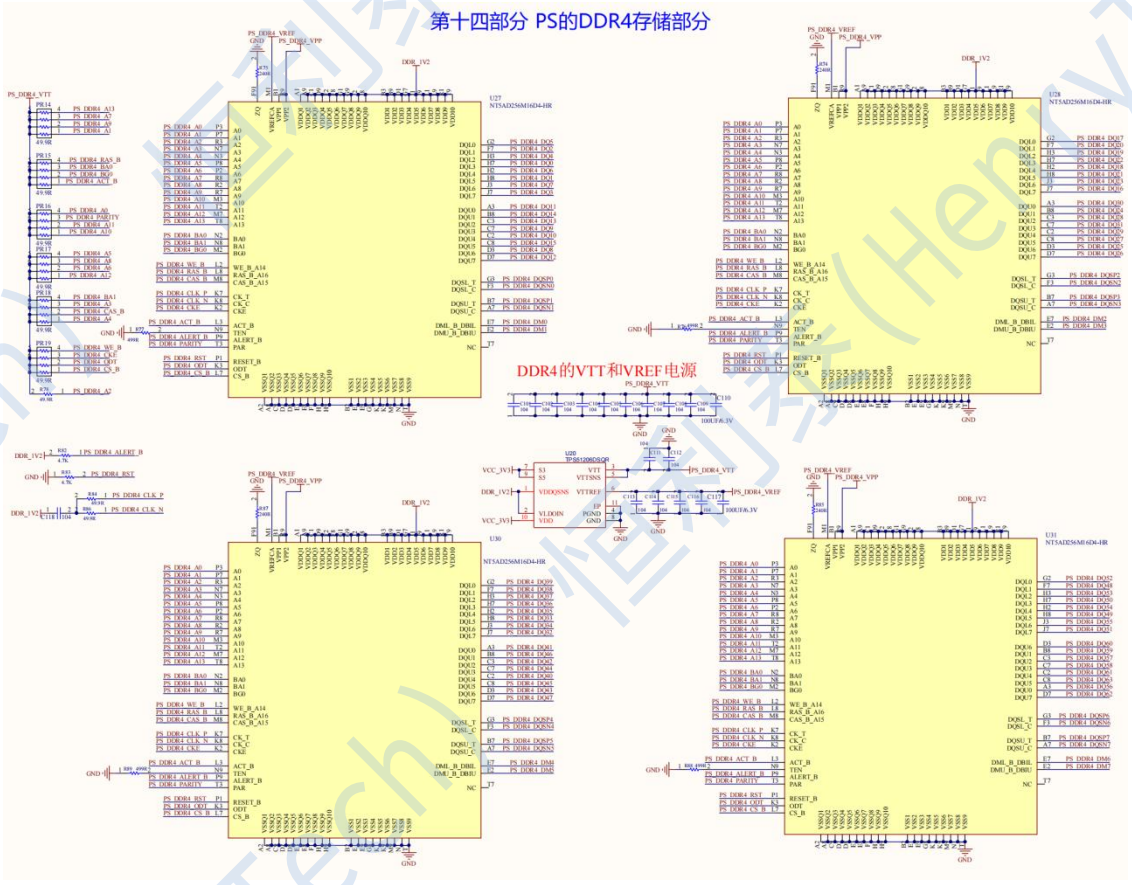
开发板使用一个FT2232HQ方案，进行JTAG以及串口的调试，原理图的第六部分上。JTAG直接连接到ZYNQ的BANK503上，作为整个芯片的ARM/FPGA开发，串口连接到ZYNQ的BANK501的MIO42 43上。这部分还包含BANK503的启动拨码开关BOOT：



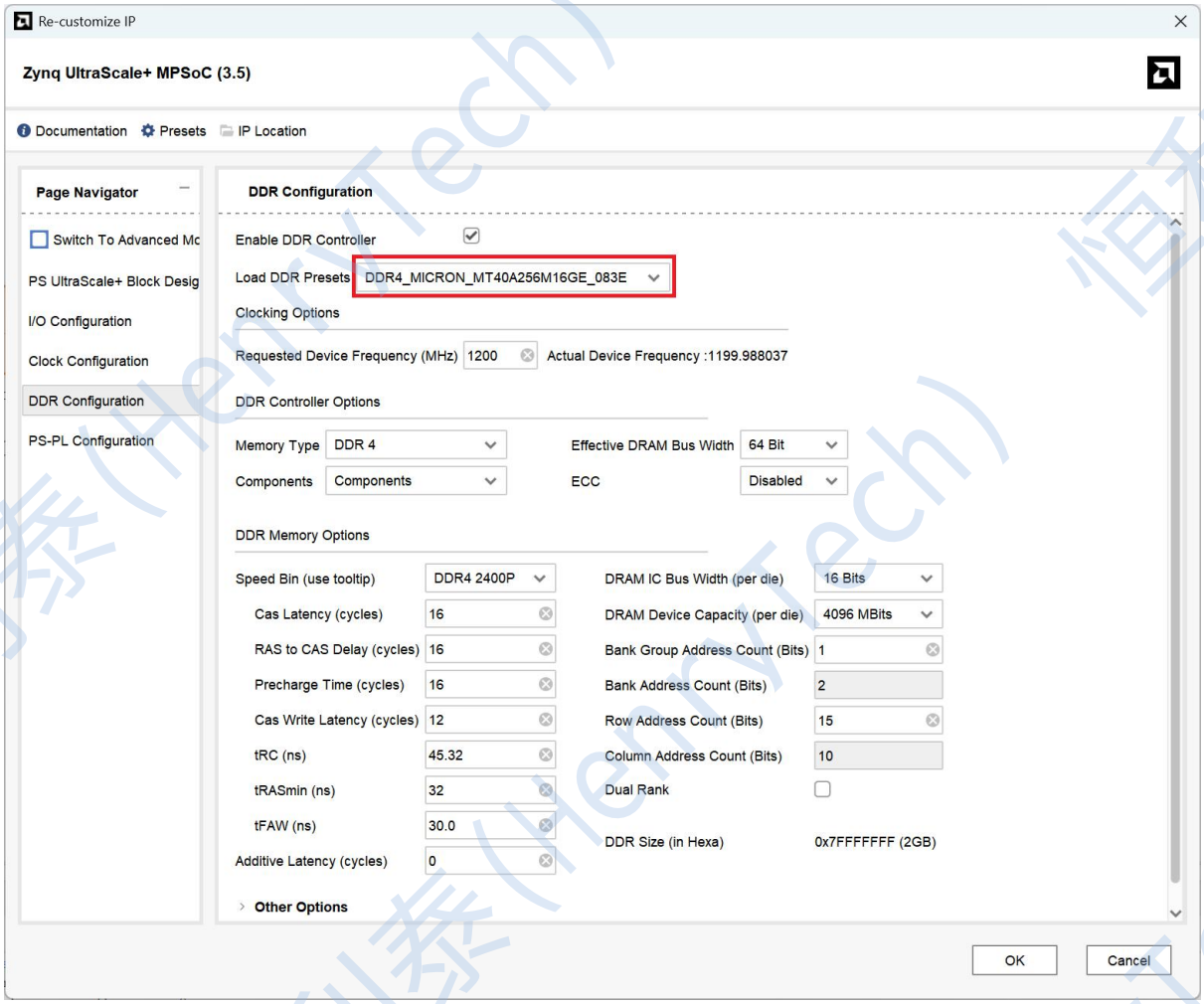
我们DEBUG接口使用TYPE-C，仅需一根TYPE-C USB3.0的线即可完成所有调试。JTAG最大支持30MBPS速率。如需外接JTAG，需将第五部分的拨码开关JTAG MODE设置为0以支持外部扩展接口通过BTB连接器自行设计转接模块外接JTAG，JTAG模式设置在快速测试手册详细说明。

七、PS DDR4部分

开发板ZYNQ处理器采用四颗DDR4，总容量2GB/64位，单颗512M的NT5AD256M16D4-HR这颗型号，电路如下，原理图第十四部分（看不清请直接查看原理图）：



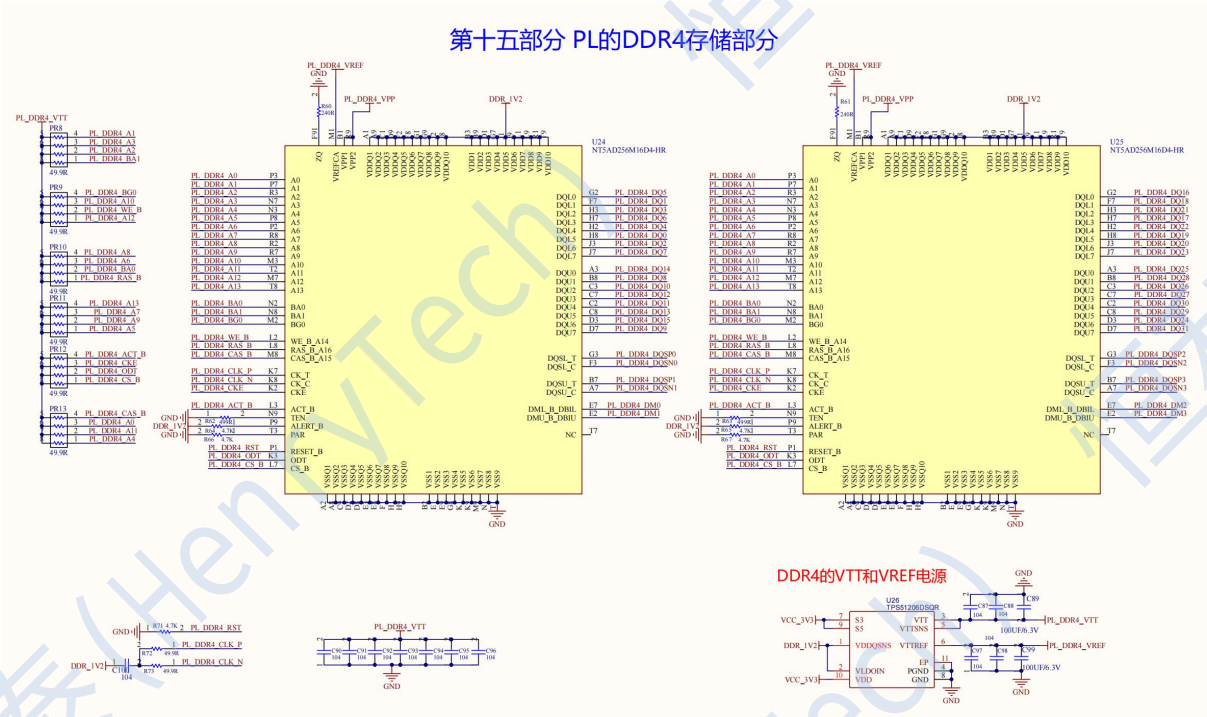
我们对应的ZYNQ的DDR配置如下，具体配置参考《11PS处理器配置及helloword打印》的第六节进行配置。DDR4在PS上最大能够运行2400M速率，这个是由于PS处理器的控制器参数决定，实际上这颗内存最大能跑2666M，兼容镁光预设型号：



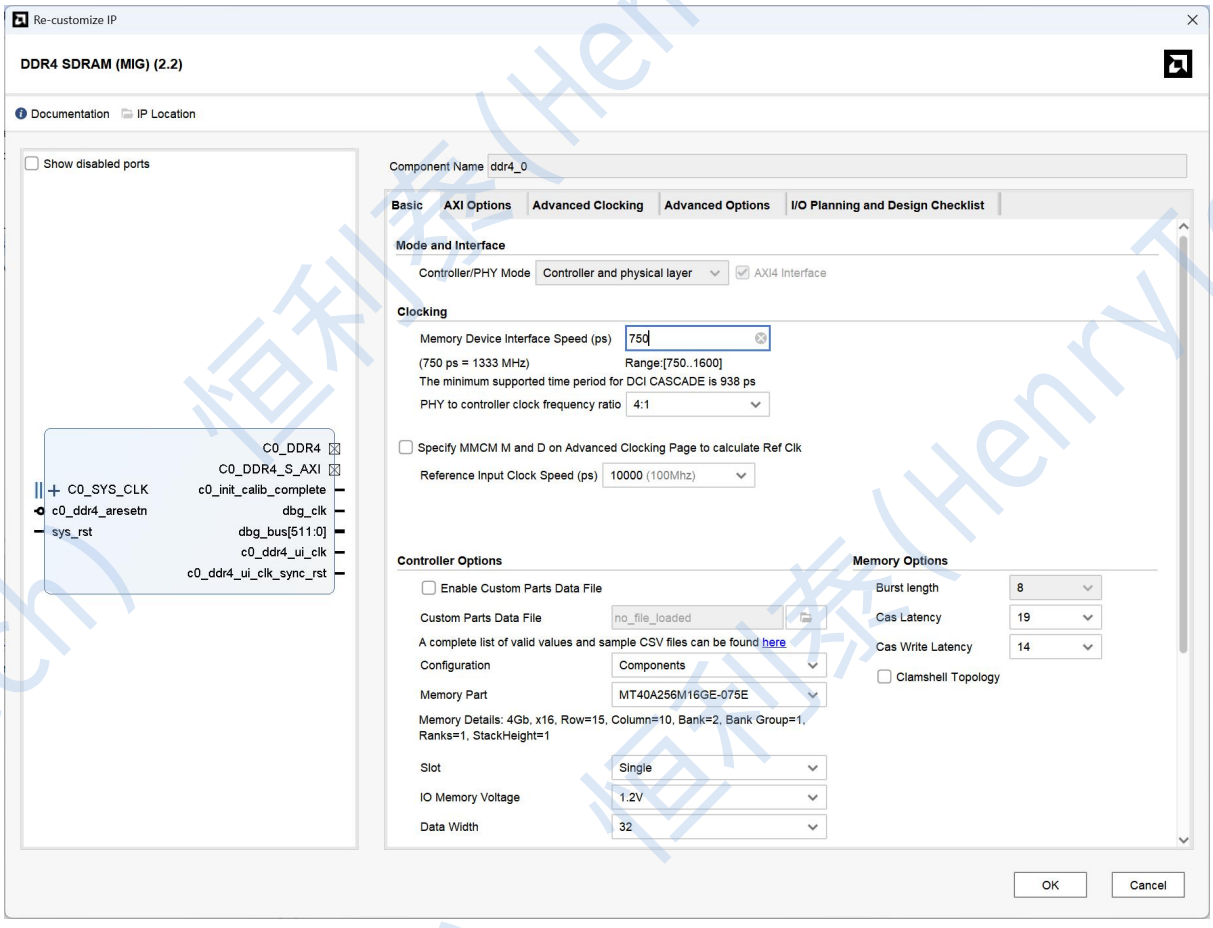
实际上颗粒决定的是内存本身最大性能，受制于PS控制器。目前ZYNQ MPSOC最大只支持到2400M的速度。

八、PL DDR4部分

与ZYNQ的PS上的DDR4方案一样，型号NT5AD256M16D4-HR，仅仅是PL部分的DDR采两颗，总共32位宽的器件。但是注意，PL与PS不同，PL部分，主芯片最大支持2666M速率，比PS部分的2400M要快一些，这是由于FPGA的MIG DDR4 IP核决定。由于射频算法有的应用需要大容量存储，片上的RAM资源只有几十M空间，远不能满足一些复杂应用需求，所以我们PL才有两颗DDR4。同时由于PL BANK引脚不足，所以只能设计32位的DDR存储，如图原理图第十五部分（看不清可以直接参考原理图）：



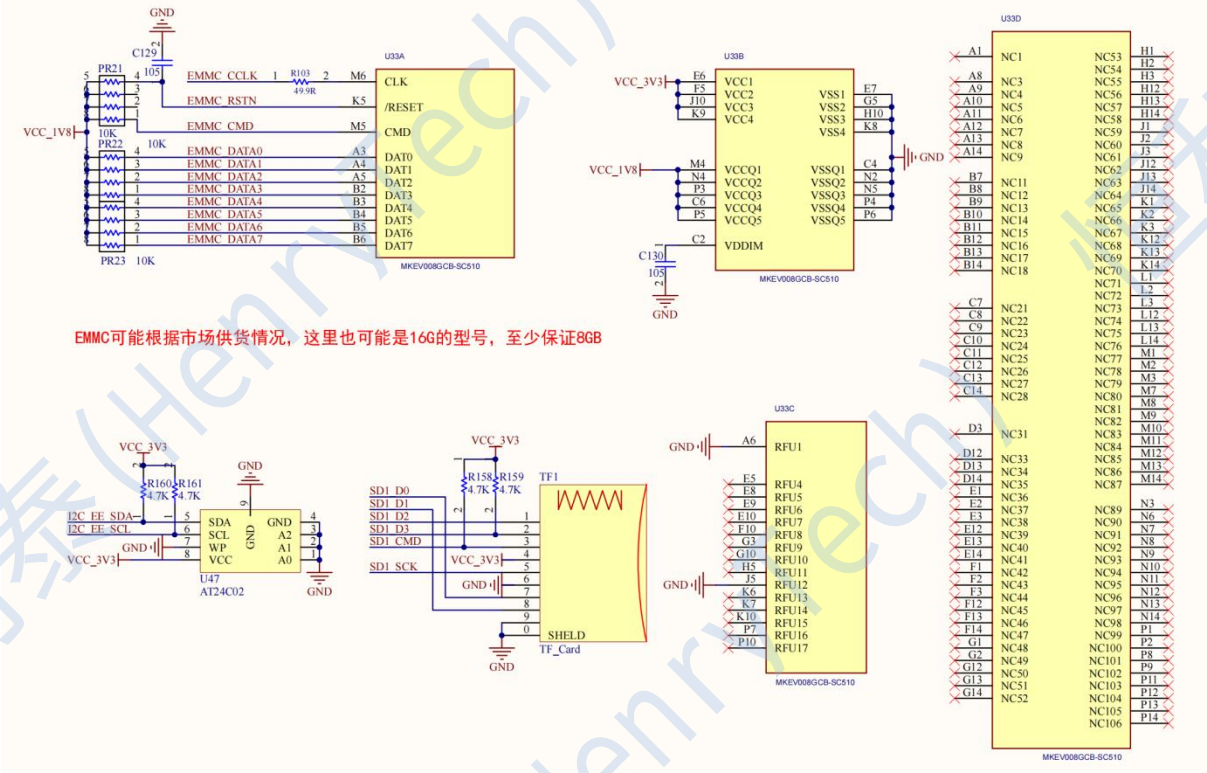
关于PL的DDR4引脚，连接到PL的BANK65 BANK66上，这两个BANK属于HP BANK，电压为1.2V，具体约束绑定引脚可以直接参考《06常用引脚约束文件》中zu67dr_pl_ddr4.xdc文件。这里不单独列出每个IO。关于PL DDR4配置 IP核如下：



引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

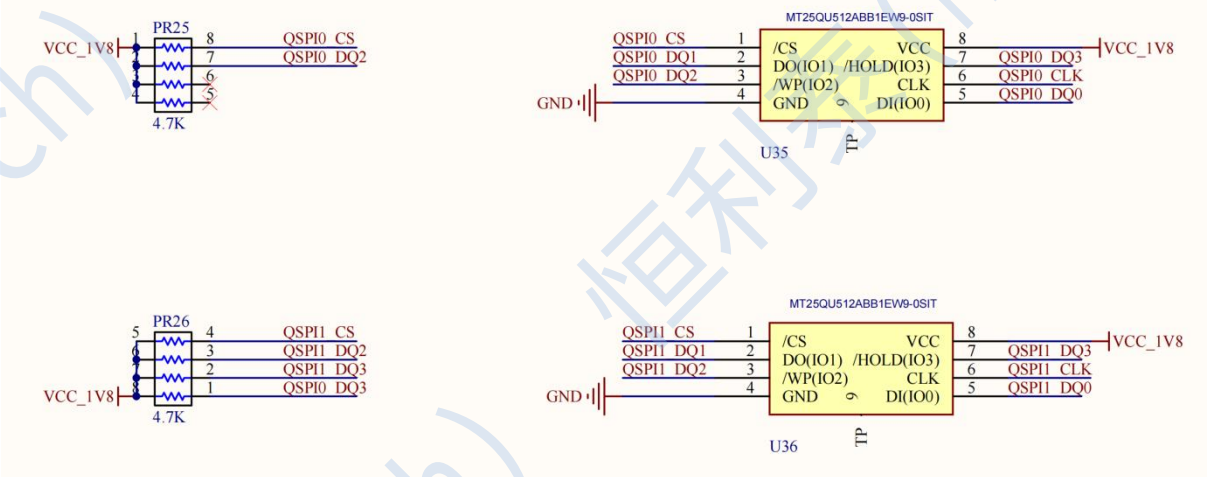
九、EMMC / SD / EEPROM/FLASH存储

开发板具有PS部分的EMMC存储，SD接口，以及EEPROM。原理图的第十六部分：



我们EMMC芯片采用5.1版本协议的芯片，型号是三星KLMAG1JETD-B041或者国产替代型号MKEV008GCB-SC510，容量至少8GB空间；SD接口采用TF卡自弹式，最大支持32G（SD2.0协议最大支持32G），注意选择内存卡不要超过32GB容量，否则可能文件系统不支持，当然熟悉协议的大牛另说。另外24C02的EEPROM，这个器件有很多厂家，一般采用上海贝岭的BL24C02，最常用的是AT24C02，时序，参数，完全一致，可相互直接替代。

原理图的第十八部分，开发板有两片QSPI FLASH，型号是MT25QU512ABB1EW9-0SIT，组成X8位宽，使用dual parallel模式：



关于存储部分配置，具体参考《11PS处理器配置及helloword打印》中关于QSPI,SD接口，以及I2C接口配置。

十、100GE QSFP28光口

开发板有一个100G以太网口。由于67DR的GTY最大支持32Gbps，100G=4X25Gbps，我们采用一个QSFP28（QSFP表示四线光口，28表示接口最大频率28Gbps），67DR自带两个100G网口控制器硬核，在选型表可以看到：

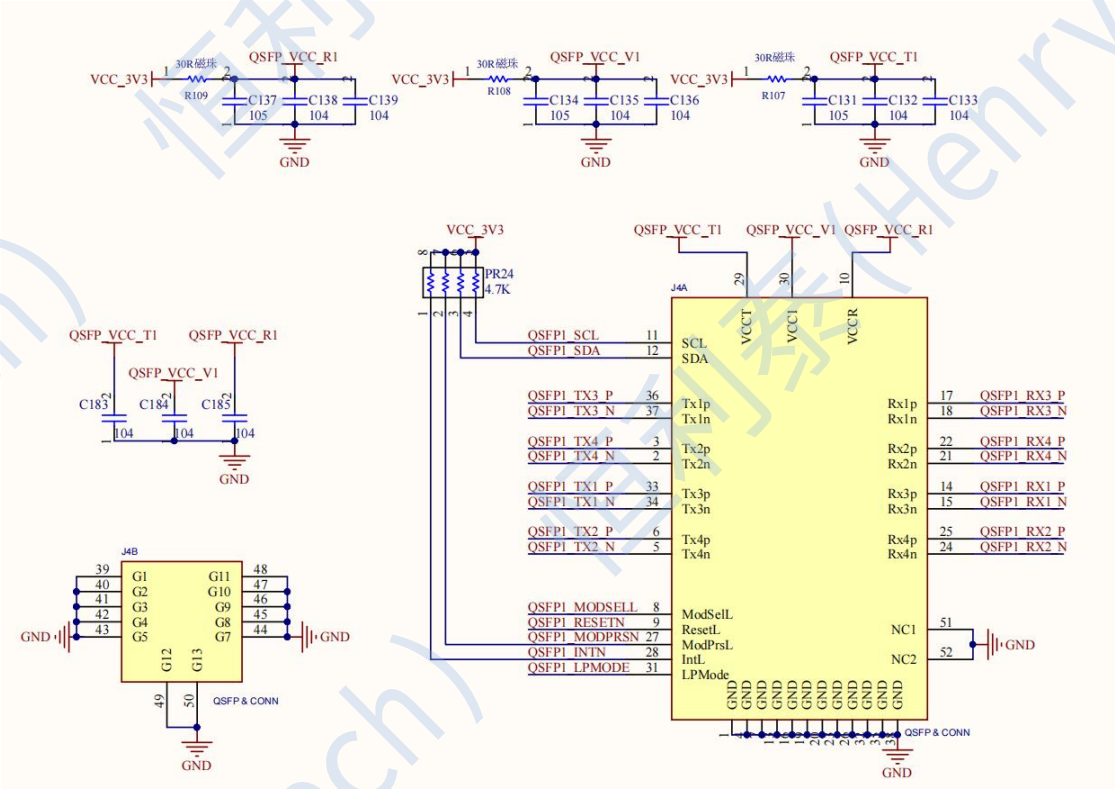
		ZU63DR		ZU64DR		ZU65DR		ZU67DR	
最大 RF 输入频率 (GHz)		7.125							
抽取/内插		1x、2x、3x、4x、5x、6x、8x、10x、12x、16x、20x、24x、40x							
14 位 RF-ADC	ADC 数量	4	2	8	2	6		8	2
	最大速率 (GSPS)	2.95	5.9	2.95	5.9	5.9		2.95	5.9
14 位 RF-DAC	DAC 数量	4		8		6		8	
	最大速率 (GSPS)	10.0*		10.0*		10.0*		10.0*	
数字前端 (DFE) 硬 IP		通道滤波器、DUC/DDC、Mixer、CFR、Complex Equalizer、PQ、Resampler、DPD							
低 PHY 硬 IP		FFT/iFFT、PRACH		-		FFT/iFFT、PRACH		FFT/iFFT、PRACH	
SD-FEC		0		0		0		0	

*请联系您当地的 AMD 销售代表获取 10GSPS 器件支持

可编程逻辑

	ZU63DR	ZU64DR	ZU65DR	ZU67DR
系统逻辑单元 (K)	393	328	489	489
DSP Slice	1,200	1,872	1,872	1,872
内存 (Mb)	54.2	38.3	67.8	67.8
GTY 收发器	4	8	8	8
PCIe Gen3x16	0	0	0	0
PCIe Gen3x16 / Gen4x8 / CCIX	0	0	0	0
支持 RS-FEC 的 100G 以太网 MAC/PCS	1	1	1	1
最大 I/O 引脚数	154	154	154	154

100G口原理图的第十七部分，电路如下：



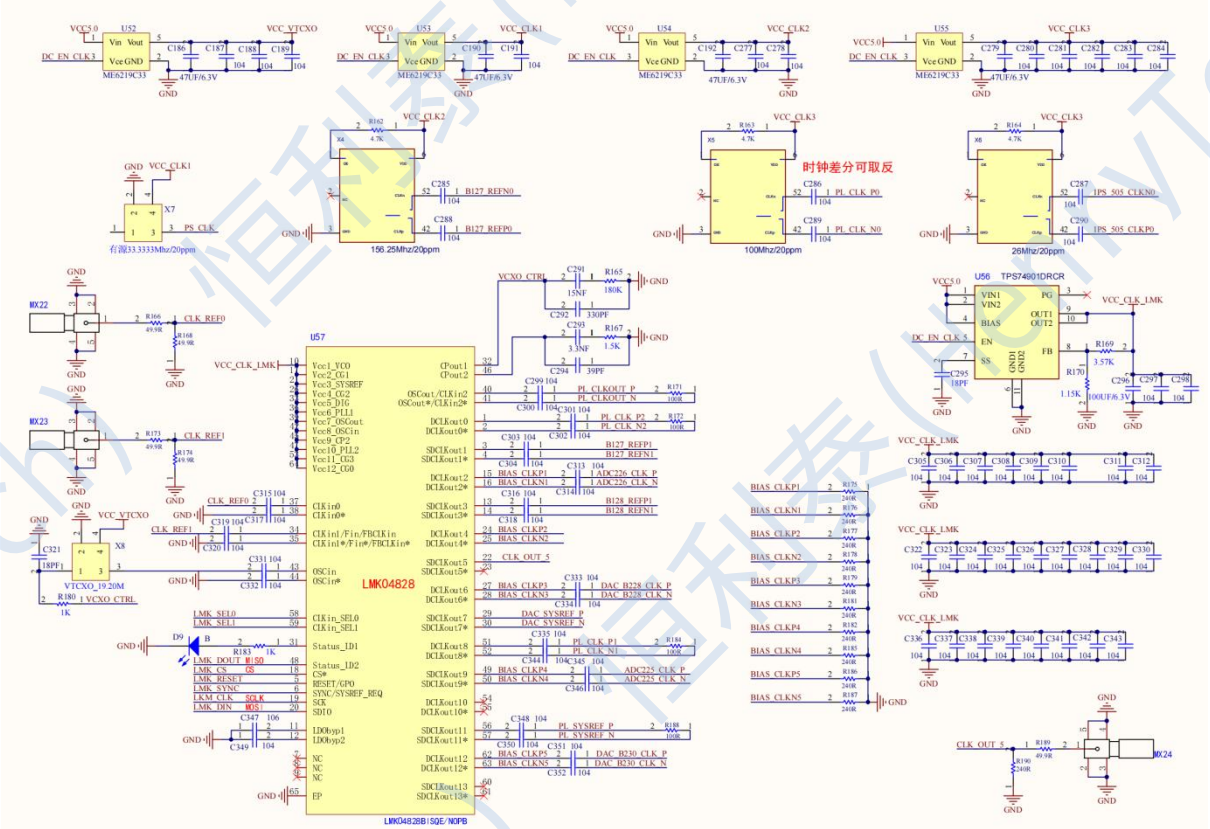
接口连接使用到67DR的BANK128上，时钟使用BANK128的REF0输入，REF0连接一个156.25M的差分晶振提供时钟。控制的引脚位于BANK88（HD BANK）。默认的，一般100G光口仅需控制RESETN和LPMODE引脚。引脚绑定可以直接参考《06常用引脚约束文件》中qsfp28_gty.xdc,引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

十一、开发板时钟部分

开发板的时钟，位于原理图第十九部分。其中包括几个独立晶振。关于几个独立的晶振均为普通有源晶振，无非就是单端和差分区别。

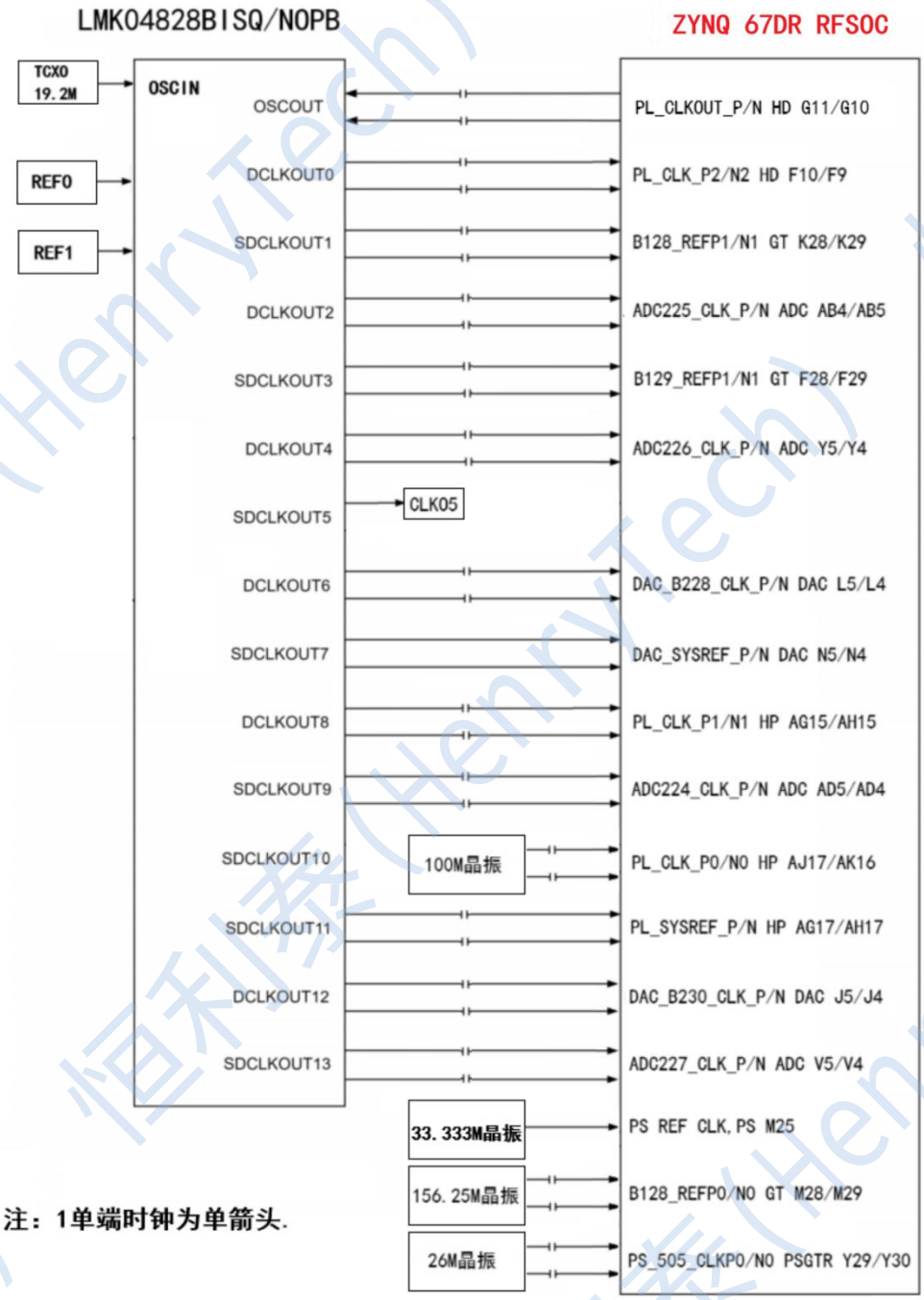
具体我们说一下开发板上的专用于多路时钟的LMK04828BISQE芯片，主要给ADC DAC部分和PL同步提供工作时钟。这部分电路如下，其中部分晶振和LMK04828都是独立供电，由于LMK04828供电要求高，我们采用TPS74901进行供电，首先是由于线性电源，纹波小，其次是这颗电源芯片最大输出能够超过3A电流，LMK04828芯片在工作的时候，夏天室温条件下可以达到80+摄氏度以上，所以对电源供电要求很高。

LMK04828其主要的参考输入的时钟使用一颗TCXO的温度补偿晶振进行输入，并且给它预留了CLK_REF0和CLK_REF1作为单端时钟参考输入，可以外接，接口采用MMCX母头端子。除了输入参考，还有一路CLK_OUT_5输出，可以外接给其他板子使用，也是采用单端时钟，MMCX母座接口：



上图也可以直接参考原理图PDF。

我们上述电路整个时钟架构如下：

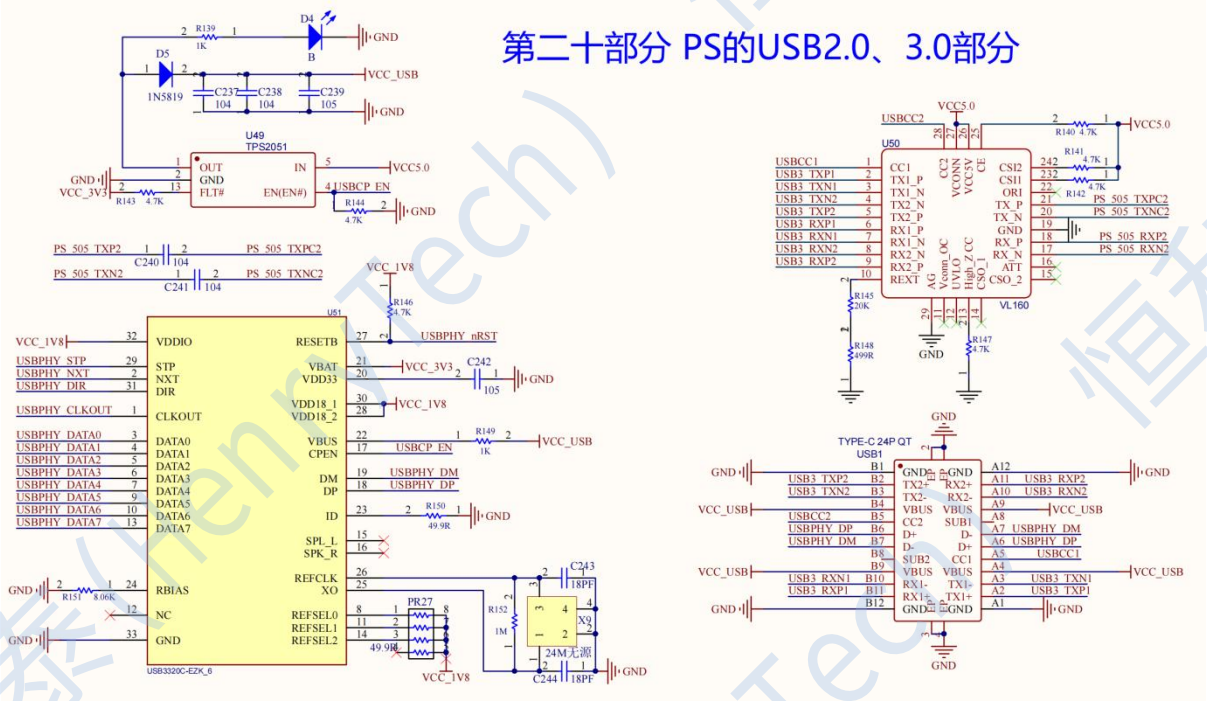


引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

十二、开发板USB3.0部分

开发板具有一个TYPE-C的USB3.0接口。原理图的第二十部分。可以作为主HOST设备，也可以作为从SLAVE接口。USB2.0部分采用PS的USB0,USB3.0部分采用PS GTR。注意USB接口按照SLAVE设计，支持正反插，由于电路使用VL160切换开关。但是对于HOST，切换开关只支持单面USB3.0。所以推荐使用带灯的TYPE-C转A的3,0转接头。关于USB使用可以参考快速测试使用

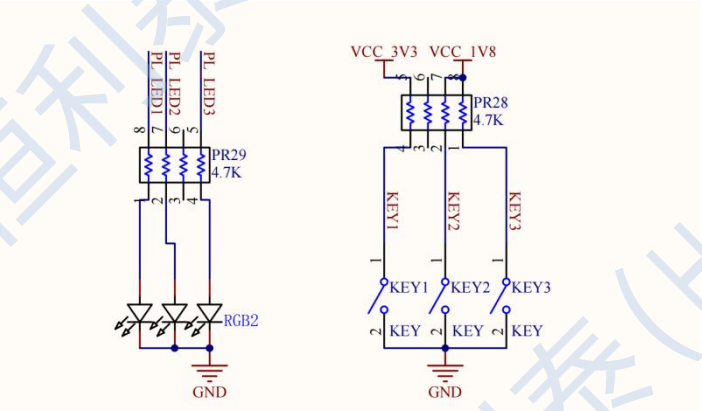
手册。这部分电路如下：



USB具体配置参考《11PS处理器配置及之helloword打印》中关于USB配置详细信息，包括USB的MIO引脚，复位，以及USB3.0的GTR配置，和USB3.0参考时钟。

十三、LED和按钮

原理图第二十一部分，开发板有3个用户按钮和一个RGB三色LED灯，原理图如下：



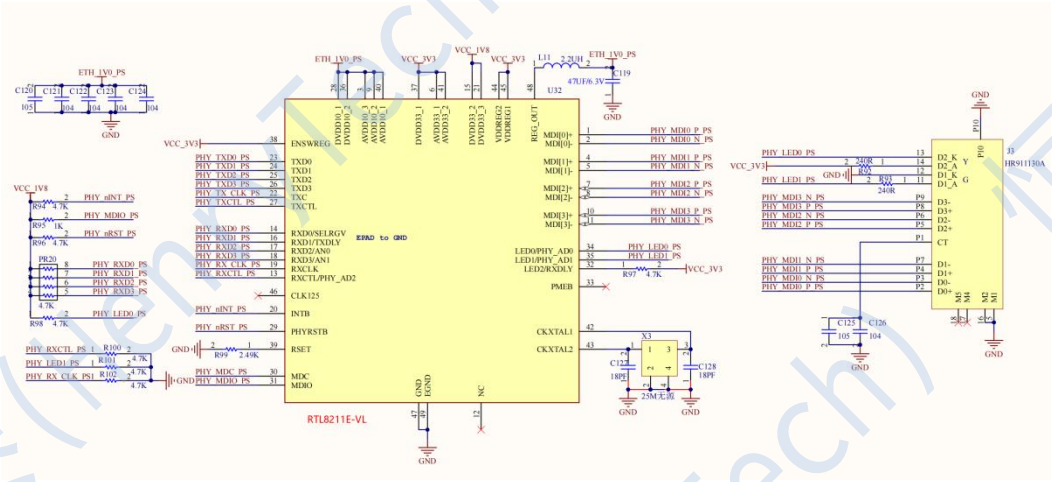
引脚对应关系：

- PL_LED1->A9, 3.3V, PL
- PL_LED2->A10, 3.3V, PL
- PL_LED3->D11. 3.3V, PL
- KEY1->MIO45, 3.3V， PS
- KEY2->H11, 3.3V, PL
- KEY3->J11, 3.3V, PL

引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

十四、PS的千兆网口

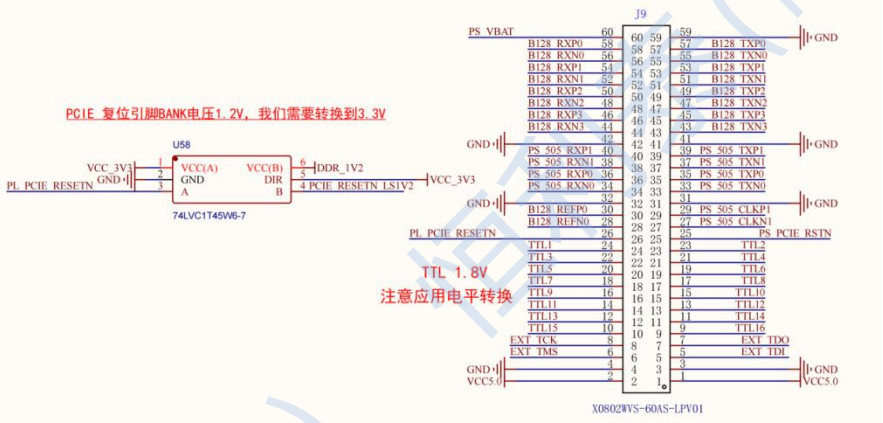
开发板有一个千兆网口，位于PS上。采用RTL8211E-VL芯片，支持10/100/1000M自协商。电路位于原理图第二十二部分：



关于以太网口配置，请参考《11PS处理器配置及之helloword打印》中的GEM3的配置。

十五、扩展接口

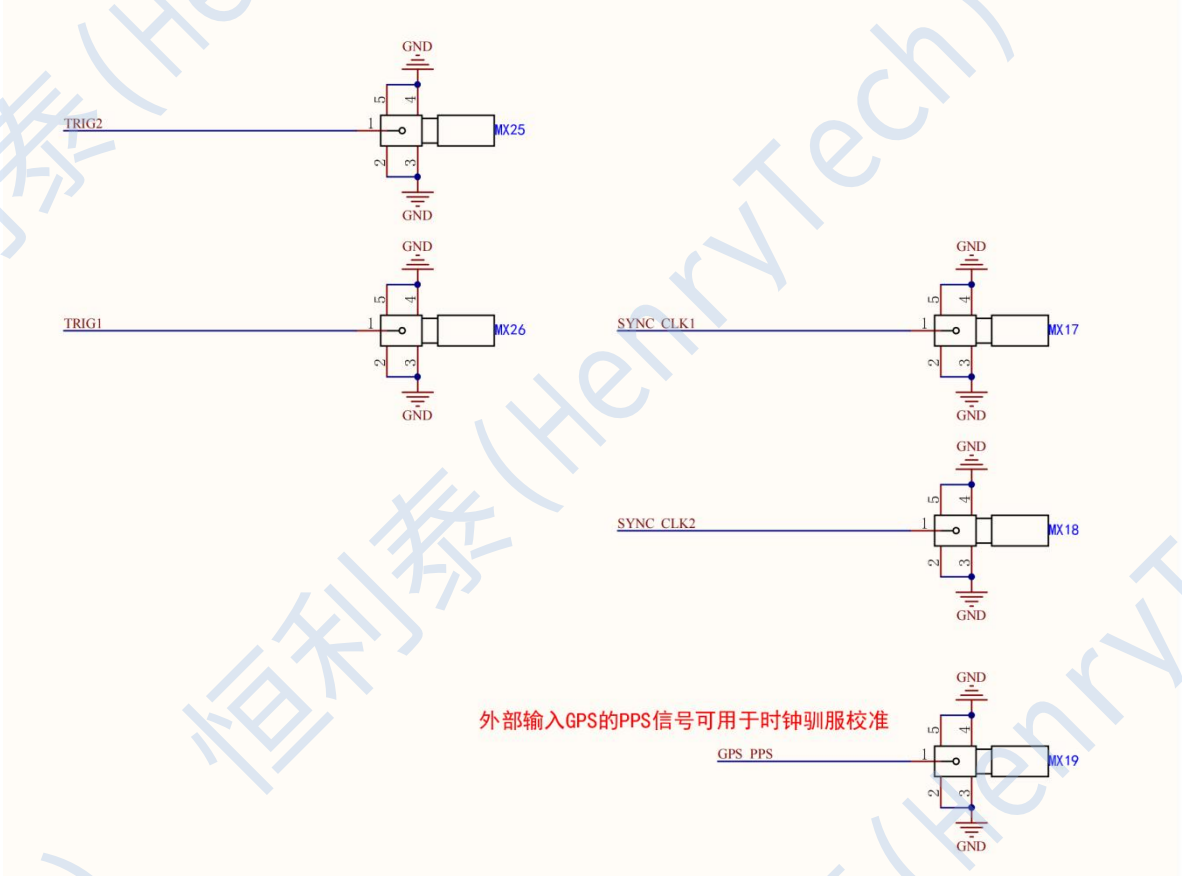
开发板带有一个型号为X0802WVS-60AS-LPV01的BTB连接器，可以配套X0802FVS-60AS-LPV01连接器成对使用。扩展口包含2对PS的GTR收发器，可用于PS的PCIE接口等，扩展比如M.2的PCIE接口协议固态硬盘。还包含4对GTY收发器，受制连接器性能，保守支持PCIE2.0的最大5Gbps速率。剩下的引脚还有PS GTR的参考时钟输入，GTY的参考时钟输入，以及还有16个1.8V的HP BANK的GPIO引脚，连接BANK67。另外，针对PS连接M.2固态硬盘的NVME接口，还有一个PS_PCIE_RSTN复位引脚，3.3V电平，连接到MIO44，以及一个PL PCIE复位专用的硬核指定PL_PCIE_RESETN引脚，位于HP BANK65的AE18。注意由于PL_PCIE_RESETN是1.2V电平，需要一个74LVC1T45转换芯片转换成3.3V。电路如下，原理图第二十三部分：



扩展接口为用户自定义发挥设计扩展模块使用。引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

十六、同步/触发/PPS接口

由于需要一些外部引脚作为触发/SYNC同步信号，另外，如果需要做时钟驯服，外部需要一个标准秒脉冲pps信号。一般来自GPS模块提供，PPS信号取两个上升沿，对时钟进行计数，判断两次PPS上升沿的脉冲数即可判断时钟误差，然后结合LMK04828的外部晶振控制DAC电压功能进行微调晶振频率达到驯服目的。这部分硬件很简单，仅仅保留MMCX连接器引脚。注意，这部分引脚为了保证信号质量，仅对layout控制50欧姆阻抗，未加电压保护，使用的时候不能短路，接地或者接电源输入，并且要注意引脚电平标准均为3.3V注意匹配，这部分电路如下：



引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

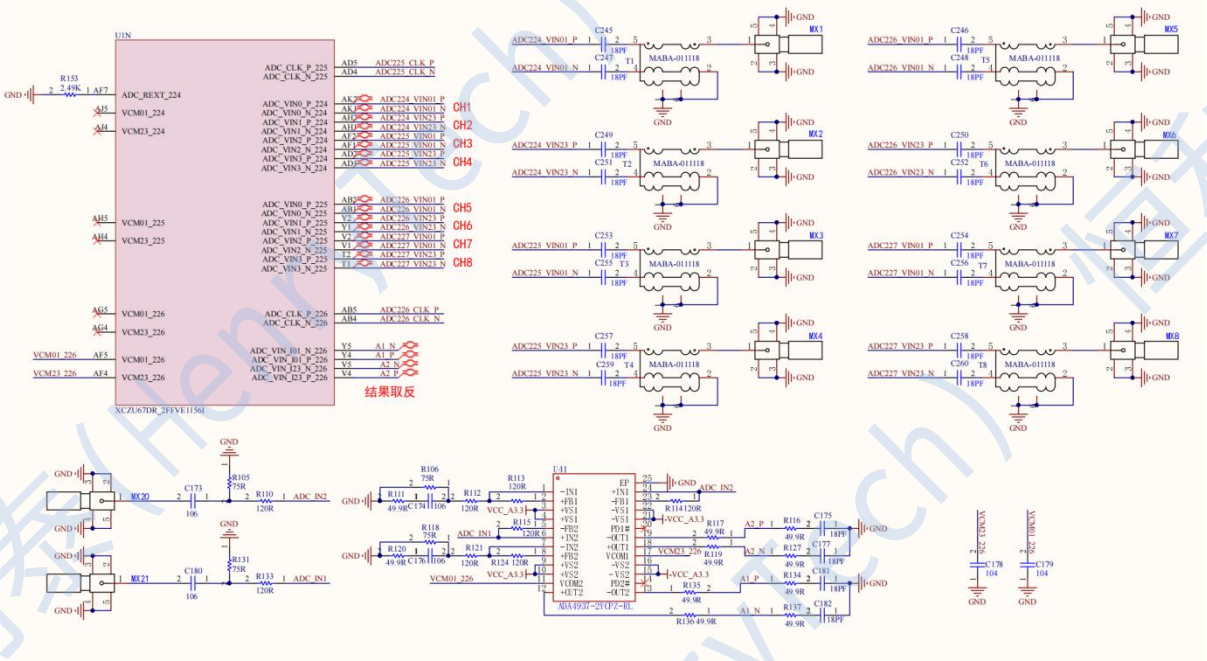
十七、射频ADC部分

原理图第八部分，开发板的RF ADC，其中BANK224 BANK225一共8通道输入采用型号为H3-MABA-011118(HenryTech（恒利泰）)的2:1巴伦，由于射频ADC接口是差分输入，我们需要将单端转换为差分再输入到ADC接口，通过AC电容耦合，RFIP核这两个BANK（Title）设置为AC耦合，未对接口射频信号做任何处理。如果需要前端滤波，需要单独设计滤波后再输入MMCX。

针对BANK226，由于这个BANK采样率高达5.9GSPS，我们采用运算放大器输入，针对极致要求性能的低频输入信号采样（非射频，做高速AD采样使用）设计的，运放前端采用交流耦

合，但是运放输入到67DR采用直流耦合，注意RF IP核BANK226（Title）配置需要配置成DC耦合，这点需要注意。

这部分原理图如下：

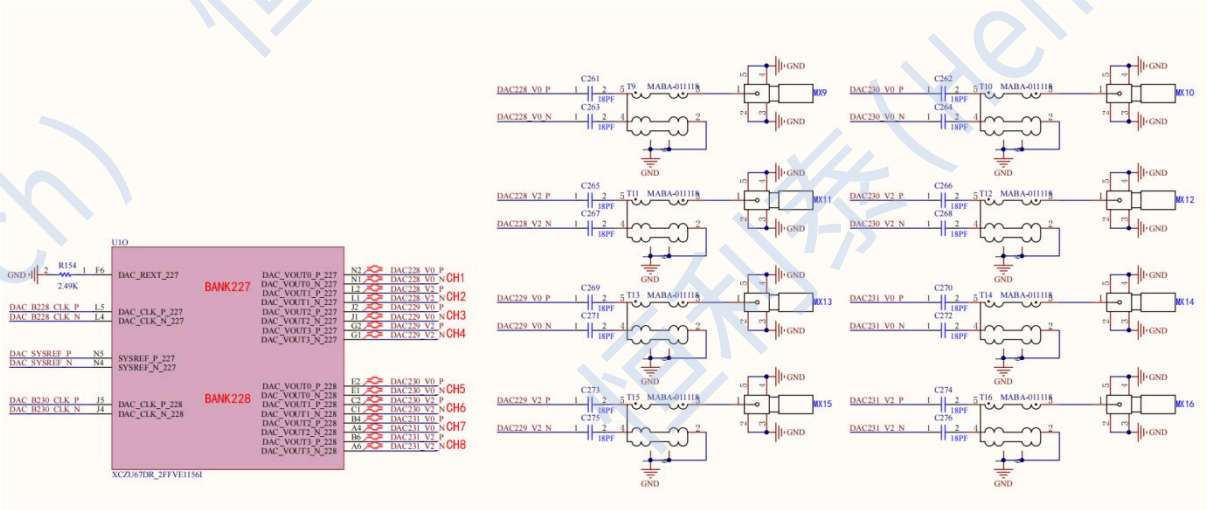


ADC针对67DR有3个BANK，BANK 224 225各有四个ADC输入。BANK226仅2个ADC输入。并且每个ADC都有一个独立时钟输入，每个ADC BANK可独立时钟。ADC DAC的一个BANK又叫Title。

引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》。ADC DAC引脚可不用绑定。

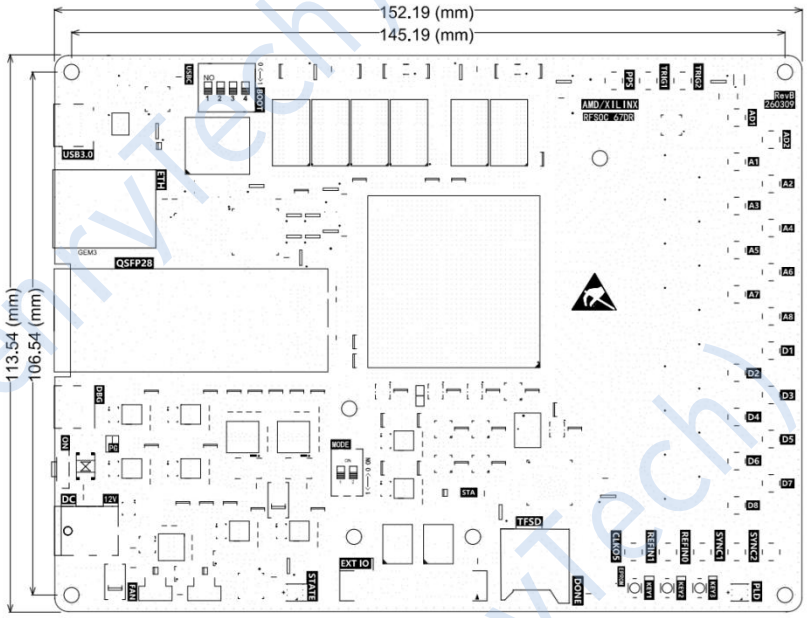
十八、射频DAC部分

同ADC部分，DAC仅仅与ADC相反，DAC是将差分信号转成单端然后输出。原理图的第九部分中，方案同样使用H3-MABA-011118巴伦，电路如下：



我们DAC前端不对射频输入输出做任何滤波处理，由于是开发板，前端需要用户结合实际频率需求来做单独的滤波再输入/输出ADC/DAC。引脚表格可参阅《01硬件资料/RF67DR_PIN_ASSIGN.xlsx》

十九、尺寸标注



二十、其他重要说明

1、其他未提及的细节请参考《01硬件资料/原理图PDF》中的对应版本原理图文件。

2、后续如果有更新优化，相应更新此文档。

HLT-XCZU67DR-B 测试快速使用指南

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 开发板供电 3
三、 开发板模式设置 3
四、 开发板启动设置 4
五、 开发板上电开机/复位 4
六、 开发板关机 4
七、 DEBUG口连接板载调试器 5
八、 SD卡模式启动测试 5
九、 QSPI FLASH启动 9
十、 其他 9

恒利泰(HenryTech)

一、文档实现功能介绍

本文档旨在大家拿到开发板后，进行开箱快速测试，和上手入门的资料说明，主要目的是让大家拿到板子第一时间对开发板进行一个简单的了解和使用指导。大家拿到开发板后，如需进行简单的测试，可根据本文档来做个快速测试和使用入门。如需FPGA开发运行相关例子，需要按照如下文档进行：

软件安装：《03Windows的VIVADO及驱动安装》

软件介绍：《04VIVADO新建工程流程和软件介绍》

本文档作为开发板出厂简单的测试使用，RF47DR-PCIE开发板属于特殊领域无线电开发平台，不是针对FPGA入门，ARM嵌入式开发的学习参考。关于开发板功能描述硬件规格和接口标注说明具体参考《01RF67DR_Lite硬件参考手册》。

二、开发板供电

开发板左边有一个DC5.5圆孔，出厂配置了12V/5A电源。除了使用电源适配器供电，板卡如果安装在机箱内，可以使用ATX转DC5.5的电源线进行供电。供电电源要求：

- 默认要求电压：9—12V;
- 电流要求：3A以上，建议5A.

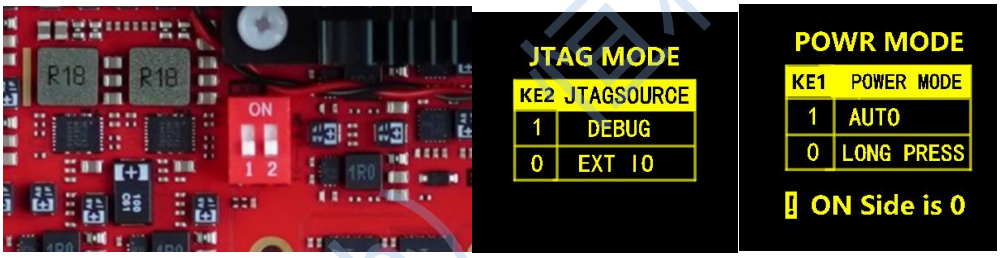
建议使用我们出厂自带的电源进行供电。运行ADDA所有通道，功耗会超过20W。

三、开发板模式设置

开发板模式设置开关对应两种设置。2段拨码开关的1段决定开发板上电启动模式，1段位于ON一侧，开发板通电自动开机启动；位于数字1侧，则需要长按PW/RST按钮开机，并且支持长按关机。

拨码开关2段决定JTAG接口使用板载DEBUG口还是外置JTAG口。2段置于ON一侧，JTAG需要自己设计一个BTB转接板连接开发板扩展接口转接外部JTAG插针。比如smartlynq等。正常调试，2段置于数字2一侧即可。正常情况下，板载DEBUG接口下载调试开发完全足够。外置JTAG需自制BTB转接模块！

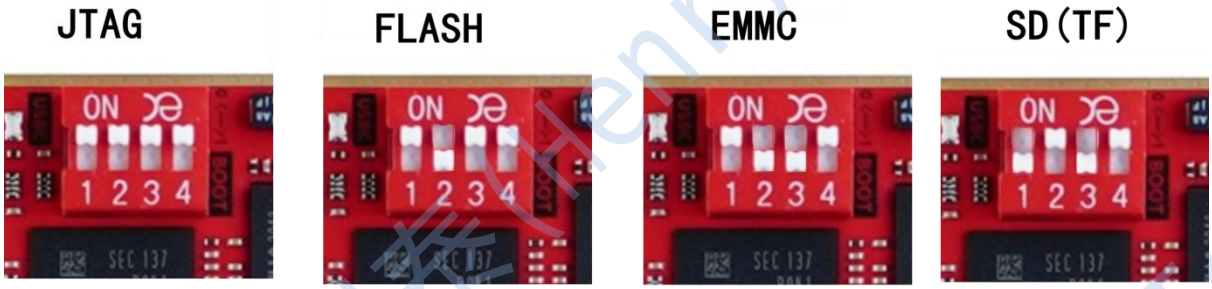
对应的拨码开关实物图与英文标注表格如下所示：



四、开发板启动设置

我们的RF47DR-PCIE支持多种启动模式，支持JTAG，QSPI,SDCARD,EMMC四种，具体启动模式在开发板左侧丝印可以看到具体信息表格，拨码开关每一段都有数字对应，1,2,3,4段。对应段开关放在ON KE印字一侧（板子外侧）是把这一段置0，印数字1234一侧（板子外侧）为1：

对应实际拨码开关位置如下实物图所示：



默认出厂由于平时需要进行调试FPGA和ARM，模式必须使用JTAG；除此之外，SDCARD模式和QSPI FLASH模式比较常见。EMMC由于出厂未做分区，烧写也比较麻烦，暂不做讨论，留给实际有EMMC启动需求的客户使用。

BOOT开关设置必须在断电模式下操作！！否则设置不会生效，开机上电的时刻就会读取BOOT设置从对应模式启动芯片，开机后再设置无效。开机后BOOT生效另一种方法就是短按开机按钮POWER复位重启。

五、开发板上电开机/复位

开发板连接出厂电源到DC-IN之后，参考第三节，如果2段拨码开关设置1段位于ON一侧，通电自动开机；ON按钮短按可以复位，驱动ZYNQ的POR引脚拉低。如果1段位于数字1侧，则长按ON按钮可开机启动板卡。

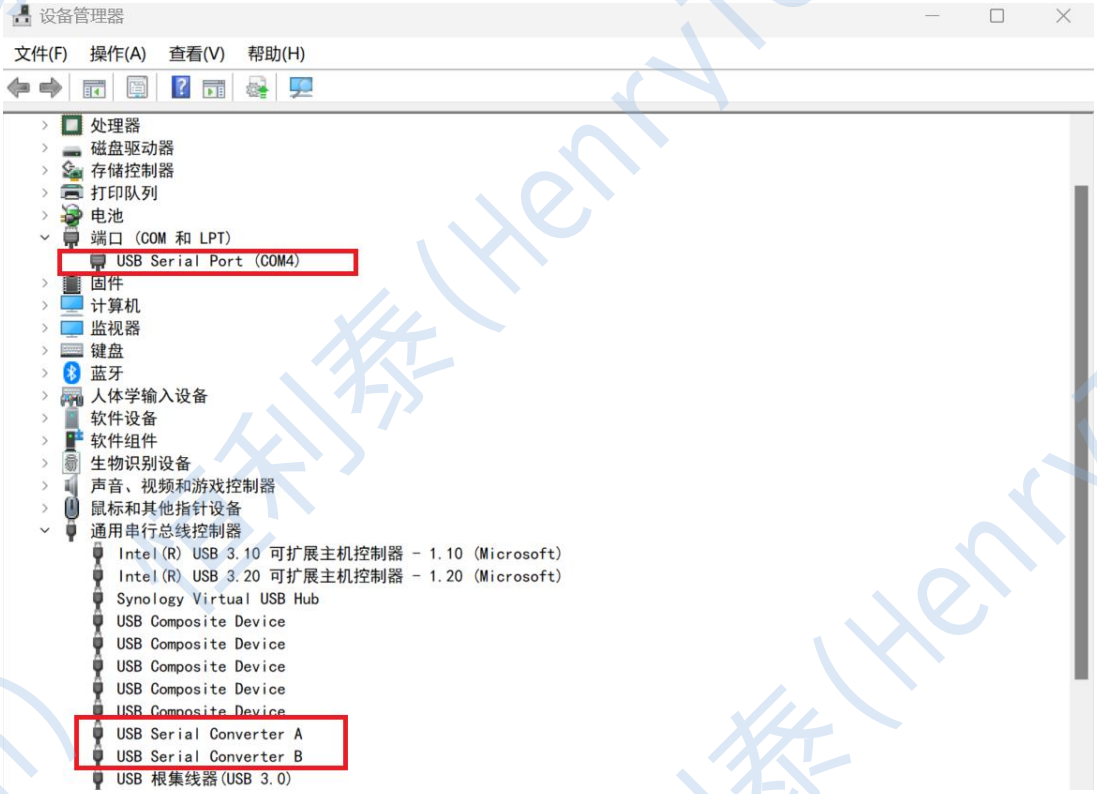
六、开发板关机

第三节中的2段拨码开关1处于数字1侧，支持长按关机。

七、DEBUG口连接板载调试器

开发板的DEBUG口可用于串口终端，以及做裸机实验或者相关有开发经验的客户做自定义开发的时候，连接JTAG调试ZYNQ用。DEBUG口使用的FT2232HQ方案，两通道，一个通道包含UART串口设备，连接到我们ZYNQ的PS部分的UART1上，MIO42,MIO43引脚。另一个通道则是我们的JTAG，最高支持30MSPS速率进行调试。

使用DEBUG口对于用户来说，如果您是赛灵思开发者，安装好了VIVADO会自动安装下载器驱动，所以FT2232HQ驱动一般不用自己安装。对于win10,11，可以说是免驱动。大家如果使用的时候发现没有驱动，可以尝试自己安装，驱动位于《02软件和相关linux包/FT2232HQ下载器驱动》。我们将开发板通电上电开机之后，DEBUG连接电脑USB，打开windows硬件管理器，可以找到如下几个新的设备，其中串口设备可能对应大家电脑，com号不一样，以实际为准，换个USB口可能也会跟着变，只要确保拔掉没有，插上就有串口新增出来即可，另外usb serail converter A和B即对应我们的JTAG调试器：



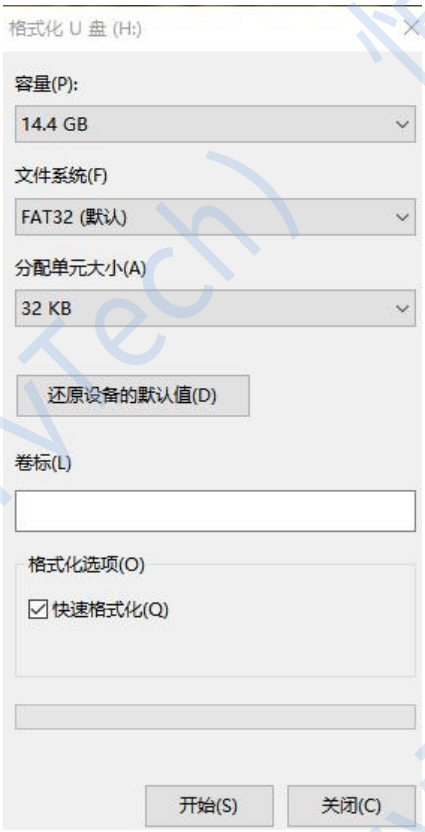
如果驱动有问题需要重新安装FT2232HQ驱动，安装完成后再重新插上。注意如果是自己安装驱动，拔掉DEBUG线再安装，否则可能会安装失败。如上识别到设备，我们可以使用VIVADO软件打开hardware manager硬件管理即可连接我们ZYNQ芯片。注意如需调试自己的FPGA或者vitis软件代码需BOOT设置为JTAG模式开机上电。

八、SD卡模式启动测试

出厂随货配送一张SD卡，可用于SDCARD启动petalinux做个简单测试。我们需要用到

SecureCRT5.5，这个是一个访问服务器或者串口的工具，主要用作串口终端。由于petalinux需要串口进行交互，其支持键盘输入比较方便。位于《02软件和相关linux包/SecureCRT5.5.zip》我们将其下载解压待用。在开始之前，如果大家自己买的内存卡要注意，推荐使用金士顿的16G卡或者32G，建议与下图保持一样，由于ZYNQ的SD接口是使用SD2.0标准，2.0协议最大支持默认不超过32G。开始之前，先将开发板BOOT开关设置为SDCARD模式，然后插上DEBUG线和电源，先不开机。BOOT设置参考本文档第三节的开发板启动设置部分。

我们将内存卡格式化如下，注意，一定要是FAT32格式：

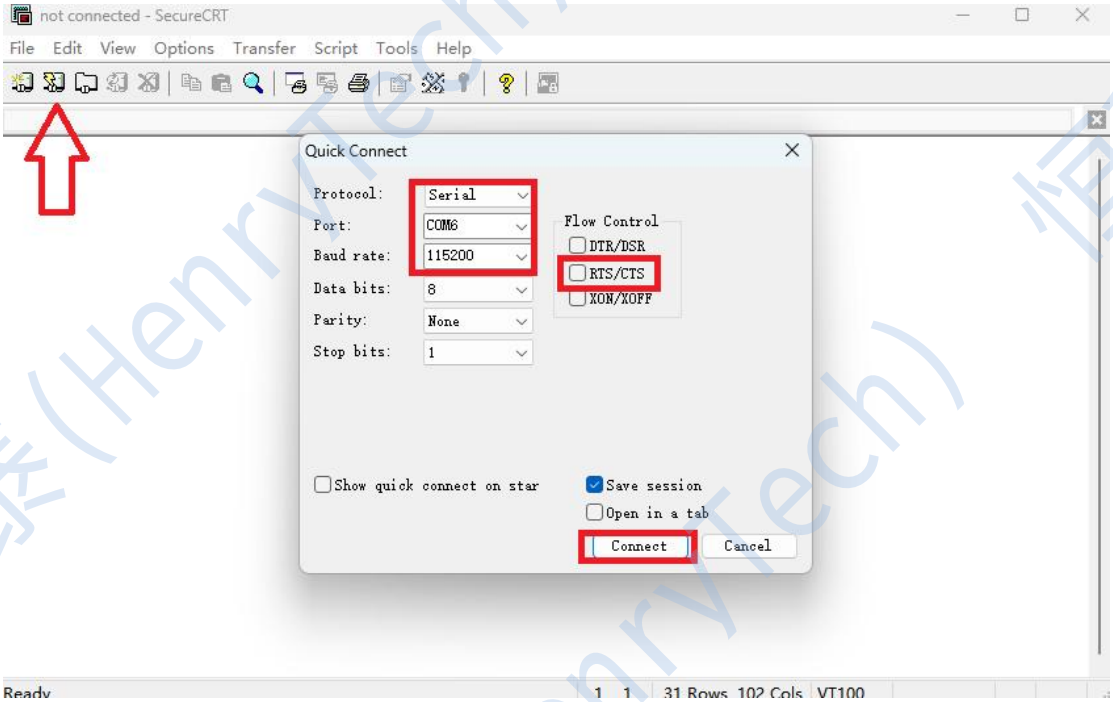


接着可以自己将目录《03例程demo》的Petalinux中的《01petalinux/编译好的启动文件》中的三个文件，BOOT.BIN，boot.scr，image.ub复制到内存卡上，然后插上开发板的SD（TF）卡槽中：

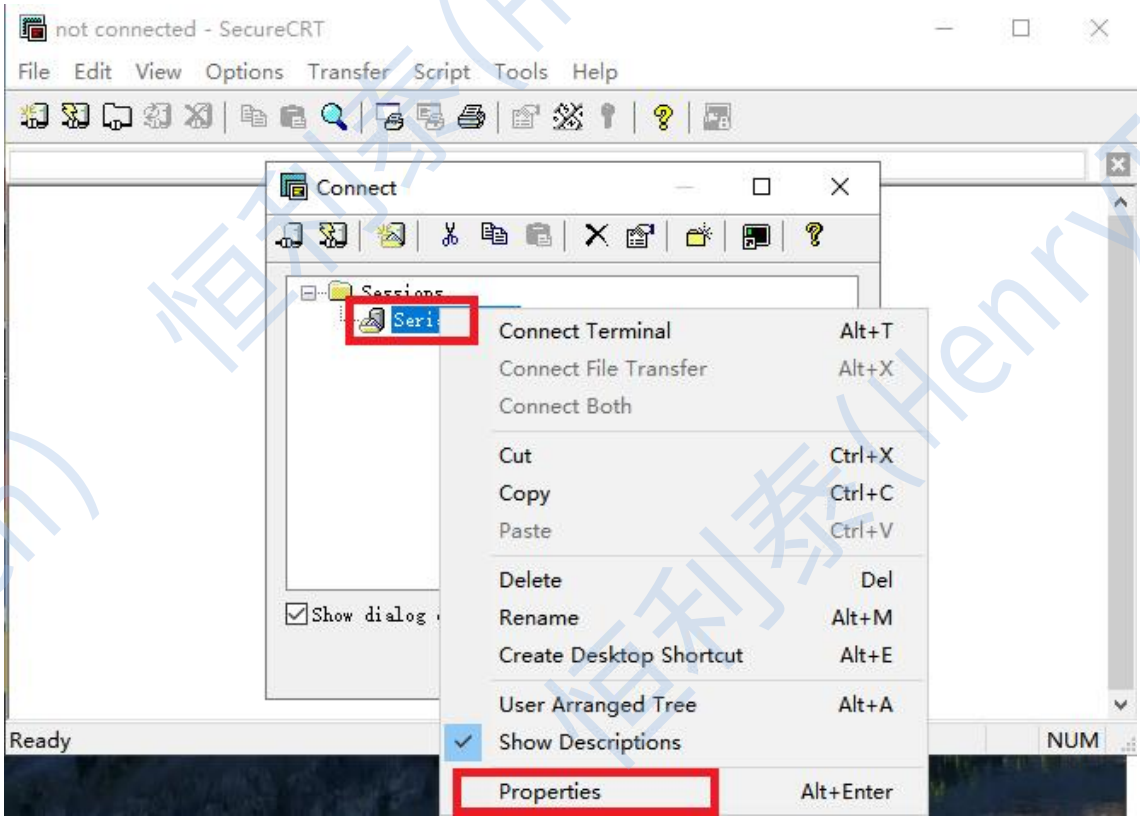
接着我们就可以准备开机启动测试。我们找到解压的SecureCRT5.5目录找到这个可执行程序SecureCRT.EXE双击启动。这个工具不需要安装。

第一次使用这个工具的启动会打开一个Quick connect，打开的时候配置界面如下，第一

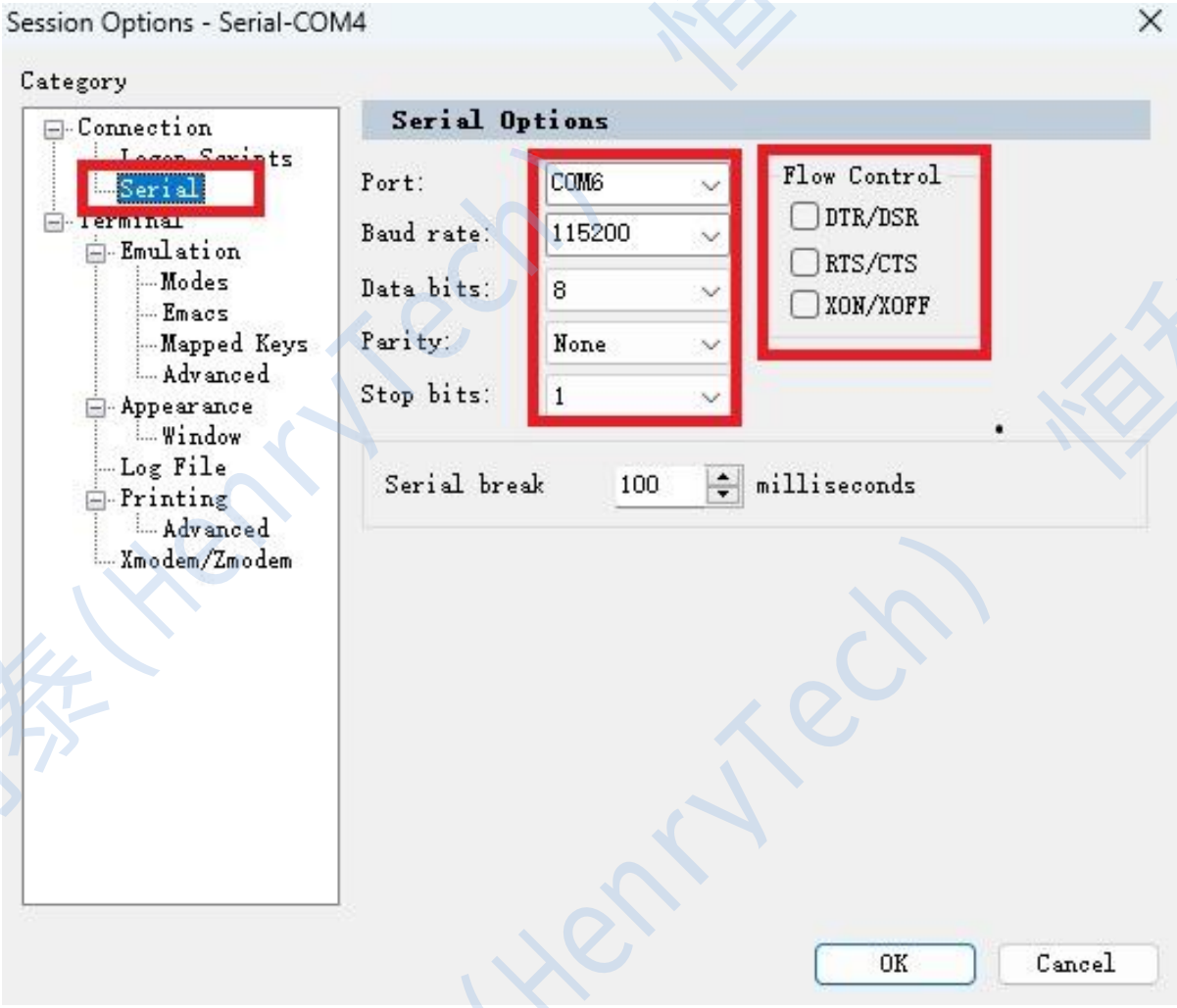
项必须选择Serial，第三项波特率115200，第二项串口号是刚才设备管理器看的串口号（这里的配置是用的之前文档复制的截图，串口号可能不一样，大家自己根据自己的电脑查看结果选择串口号），然后右侧的几项一定都不能勾选，有勾要去掉。有的朋友右边勾选了，发现不能在开发板连接启动后使用键盘输入，所以这里务必要注意。选择好之后点击Connect：



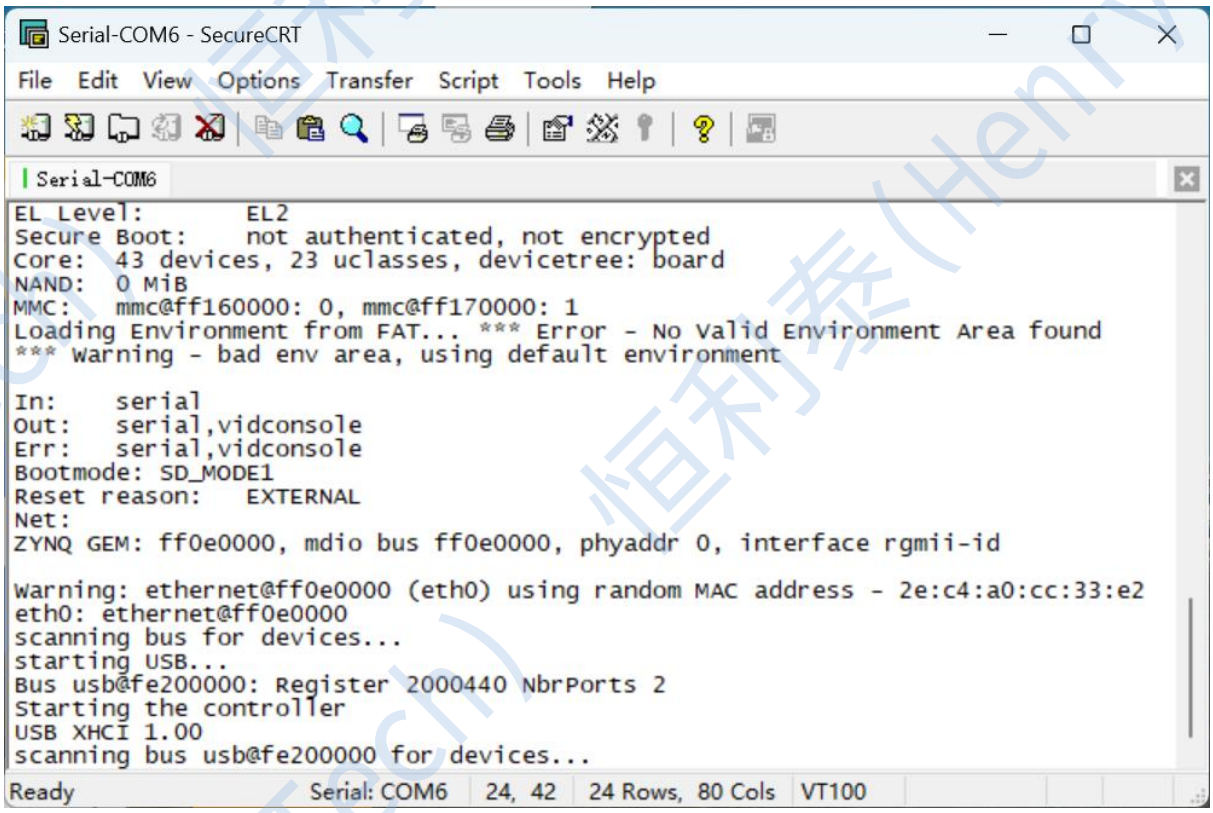
如果不是第一次连接，打开串口工具弹出的连接选择框右键最后一项：



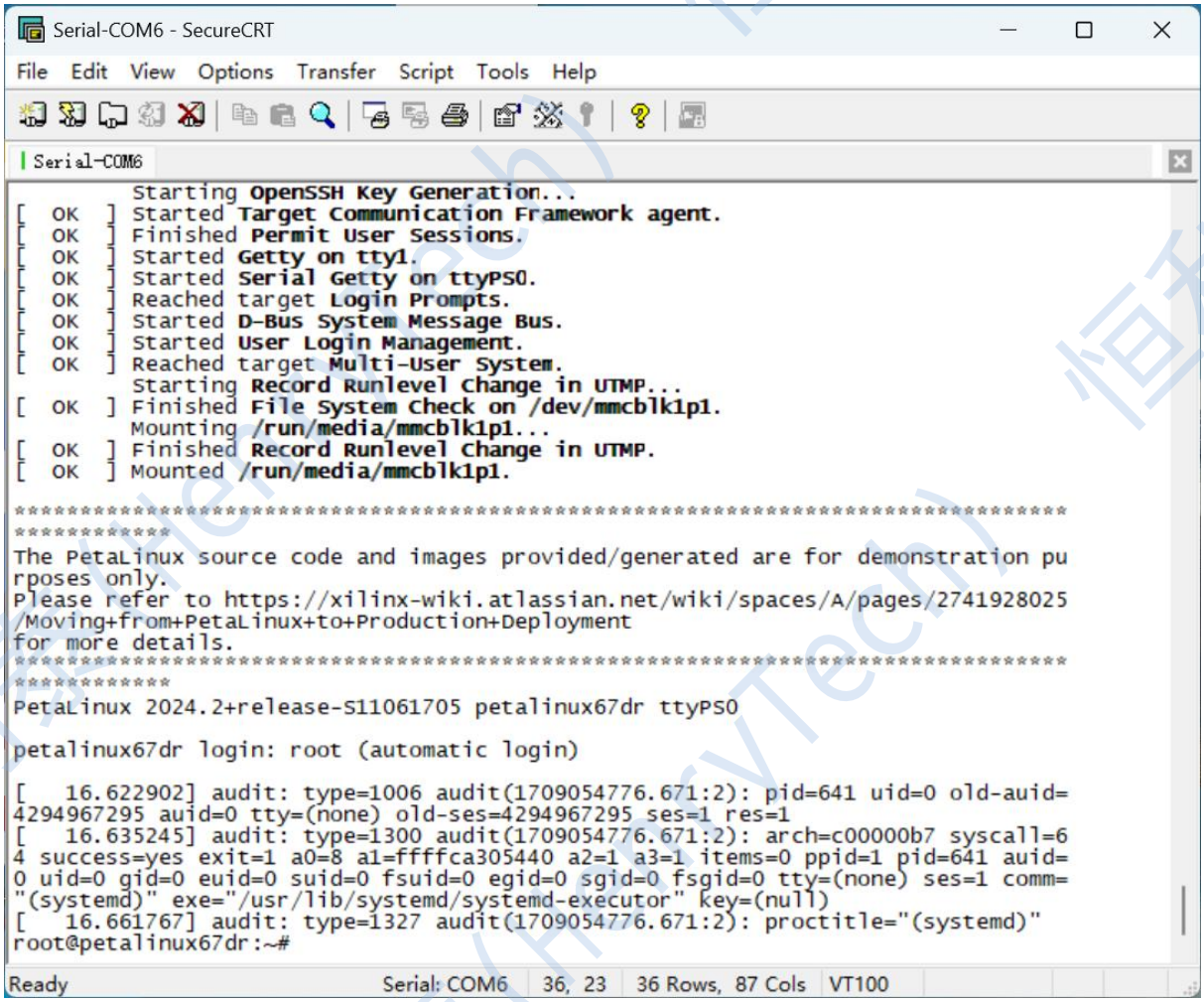
我们连接配置如下：



配置好后，点OK即可连接串口，如果配置错误或者串口号在设备管理器中找不到这个串口号，则会连接出错。连接好后，我们就可以进行启动测试了。我们这个时候按下POWER按钮，长按2—3秒开机，然后串口终端开始打印启动信息：



我们等待大约半分钟启动完成，可以看到终端自动登录：



开发板USB3.0接口附近的USBCP黄灯亮起，表明USB完成初始化。

九、QSPI FLASH启动

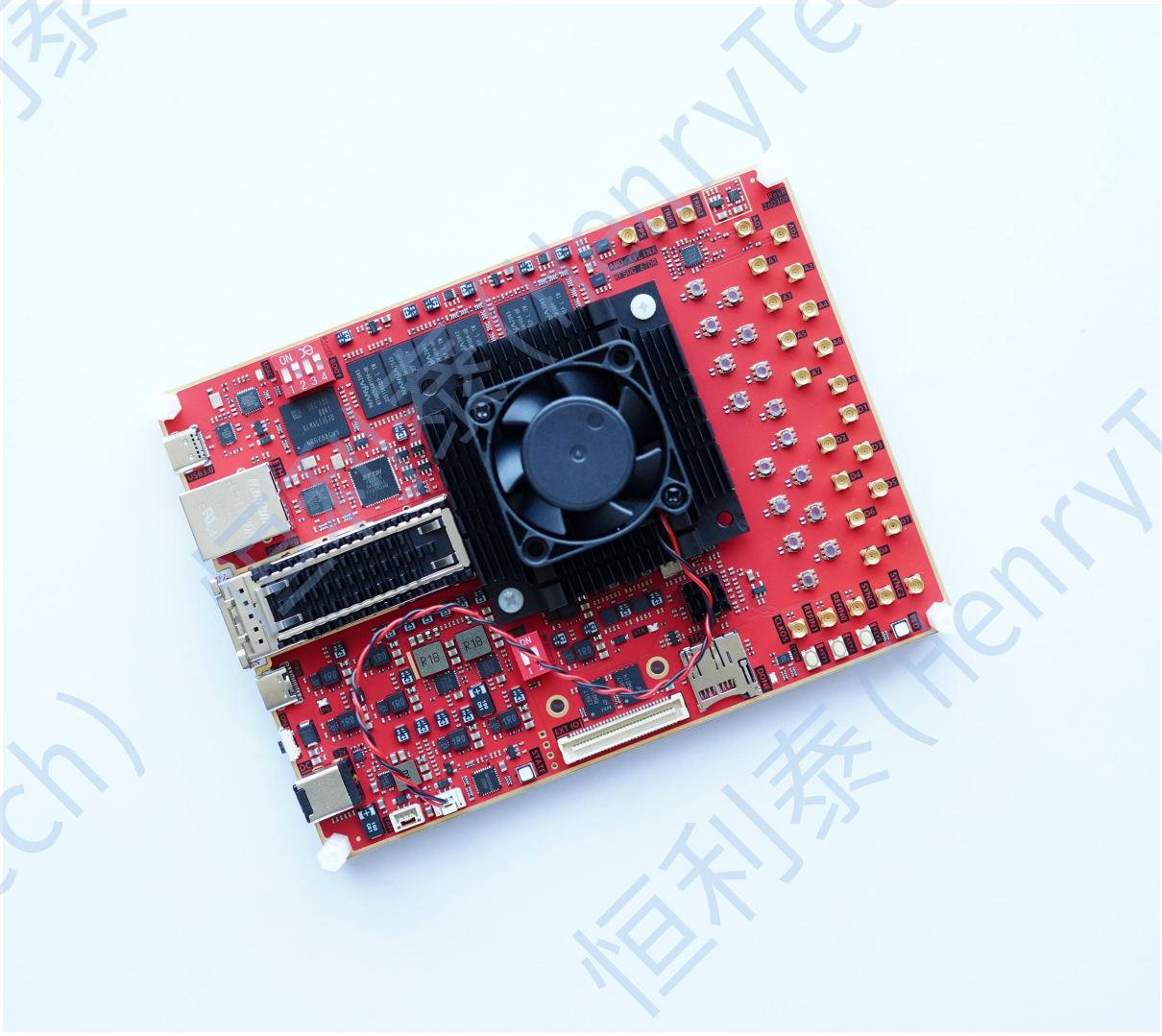
我们将BOOT设置为QSPI模式，然后上电开机，可以看到PLD的RGB LED三色交替闪烁。FLASH固化烧写了一个ZYNQ通过PS控制AXI接口，使用AXI GPIO控制LED闪烁的程序。出厂FLASH的VITIS工程位于《03例程demo/ARM/出厂FLASH的vitis工程/factory_vitis_flash》中。

十、其他

注意，开发板散热器正常使用请不要拆掉，由于主芯片RFSOC发热较大，工作的时候需要散热好，否则会出现不稳定情况或者损坏。另外防静电亚克力也尽量不要拆掉，对设备起到保护作用，如果大家自己重新设计金属外壳另说。

windows的VIVADO及驱动安装

------ 基于RF67DR-Lite开发板套件



恒利泰(HenryTech)

目录

一、 教程简介 3
二、 注意事项 3
三、 注册账户 3
四、 获取安装包 4
五、 安装VIVADO&VITIS 6
六、 安装License 12
七、 安装驱动 15

恒利泰(HenryTech)

一、教程简介

本教程学习如何下载vivado及VITIS安装包，然后安装到自己的windows系统。注意，本教程使用的是VIVADO2024.2版本。学习的时候，如果是初学者，一定要用我们配套的这个版本，不要用其他版本，因为例程打不开，另外赛灵思VIVADO每个版本都有一些BUG，大家新年一定不要随便使用其他版本。实际测试目前新版本比较稳定，而且编译时间较短的版本为2024.2，也就是本开发板配套的版本。

二、注意事项

1）教程使用的操作系统平台：WIN11专业版，win7及WIN10类似，也可参考

2）支持的系统：WINDOWS,ubuntu，教程仅提供WINDOWS平台参考。

3）VIVADO推荐电脑配置：8G内存，I5/I7双核以上，2.0主频以上，硬盘512G以上固态硬盘。如果电脑配置不佳，会影响运行性能，发挥不出软件的强大功能。如果有条件，可以采用实体机ubuntu的LINUX系统安装VIVADO，软件效率略高于Windows，如果是虚拟机，效率很慢。Ubuntu建议使用20.4，高版本亲测安装有问题，依赖包的关系比较复杂。需要自己研究linux下面的开发环境搭建。

4）安装包一定要解压到英文路径！路径不能过长！安装路径也必须是英文！最好是默认路径在C盘！License存放路径也不能为英文！

5）安装前请先卸载或者退出安全卫士或者杀毒一类的软件！

6）安装前确认我们的安装盘有至少300G以上的空闲空间，建议直接安装到C盘 默认路径。如果大家电脑太老旧，建议更换配置更好的。由于安装包已经接近100G，安装下来大概150G空间占用，软件对电脑要求配置极高。

三、注册账户

由于VIVADO软件下载需要在XILINX官方网站注册账户，才能下载软件，申请免费的LICENSE，所以先简单介绍一下如何注册。目前赛灵思已被AMD收购。

XILINX 国内网站：https://www.amd.com/zh-cn.html

点进去，可以看到如下注册页面，将每一行的信息填写。注意，注册有一些要求，大家反复尝试即可：

注册了账号就可以下载安装包或者一些资料，后续学习使用比较方便。关于注册详细过程不做讲解，大家按照官网要求注册即可。另外邮箱建议使用163邮箱，大家可以用手机号在网易邮箱官网注册一个。

四、获取安装包

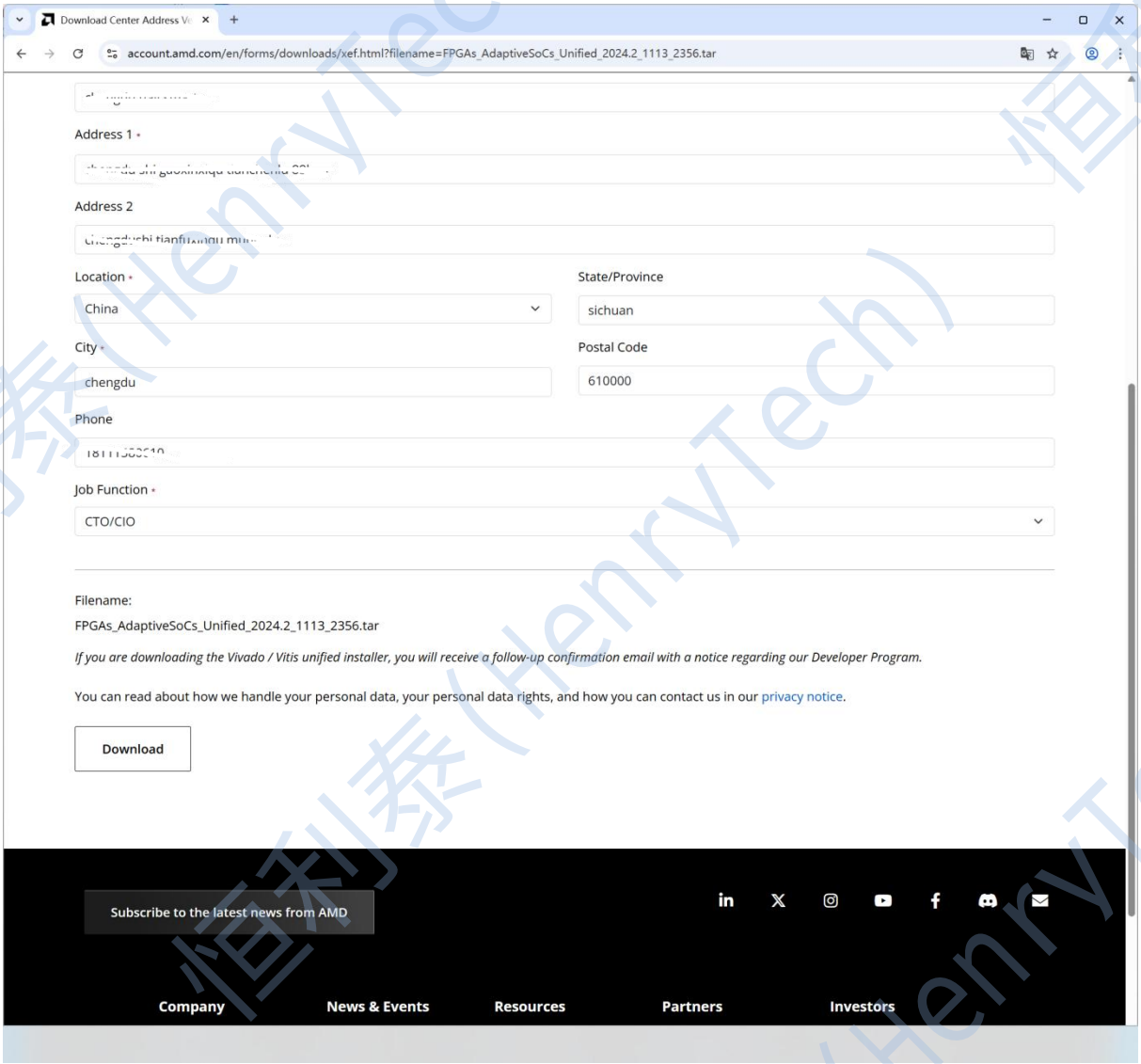
赛灵思官方网站下载。我们在官网可以找到下载与支持下拉目录，FPGA和自适应SOC下面的Vivado ML开发者工具：

找到2024. 2版本：

然后我们找到单个所有系统的安装文件，大概89G的那个文件下载，这个安装包适用于windows和linux所有平台：

点击文件进去下载，他会让你填写一系列的姓名和地址验证，需要美国政府出口批

准。信息不一定正确，但是要符合规定。大家填完之后点击最下方的下载即可。注意，如果大家下载填写过一次，点击这个填写单子的页面会自动帮你填写，只需要等一等，因为网站反应慢。填完之后点击下载即可，浏览器会开始下载文件。如果大家觉得浏览器下载不稳定或者太慢，建议大家复制浏览器正在下载的链接，然后使用迅雷下载。

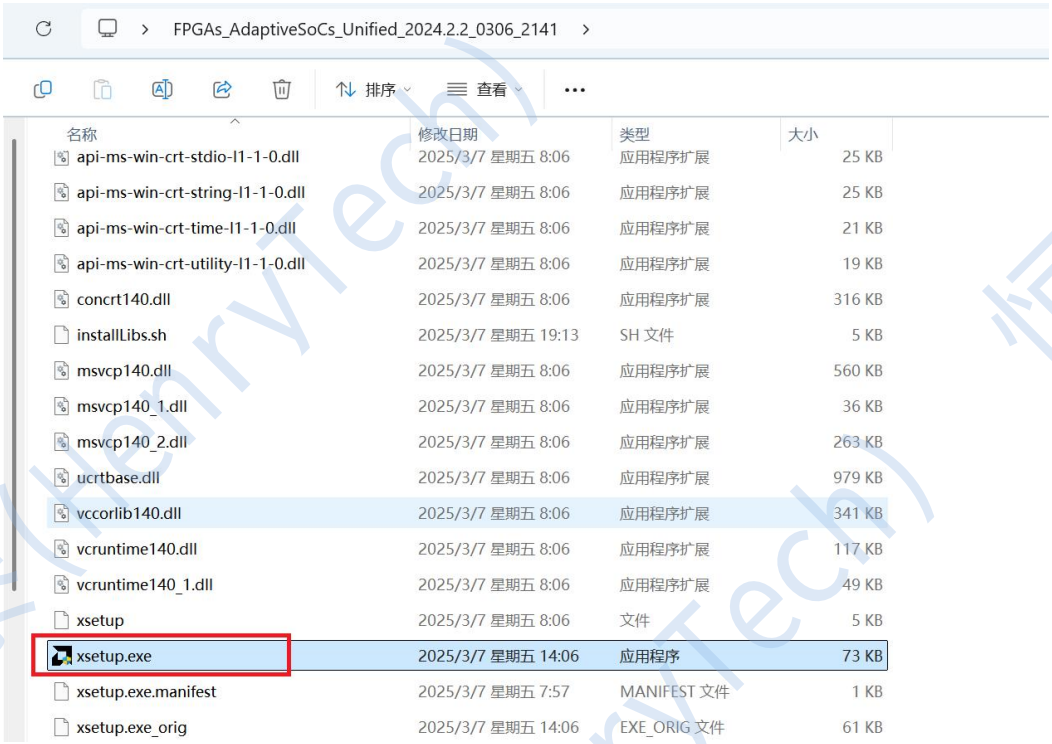


下载完成，安装包就可以解压进行安装。

五、安装VIVADO&VITIS

根据上一节我们获取到的安装包，大家解压到一个英文路径，不能有任何中文路径的文件夹。我们在硬盘根目录新建一个VIVADO2024.2的文件夹，然后解压进去之后，我们开始安装。解压需要解压两次，由于安装包打包的时候文件过大，我们通过文件夹存放之后分卷压缩，第一次解压出来是一个文件夹包含安装压缩包，所以还需要再次解压安装包。解压过程非常慢，笔者使用I9 13900处理器一次解压大约5～6分钟时间，两次大约12分钟。安装之前大家再次检查有没有中文路径，杀毒软件和安全卫士一类是否退出或者卸载。建议也关掉windows

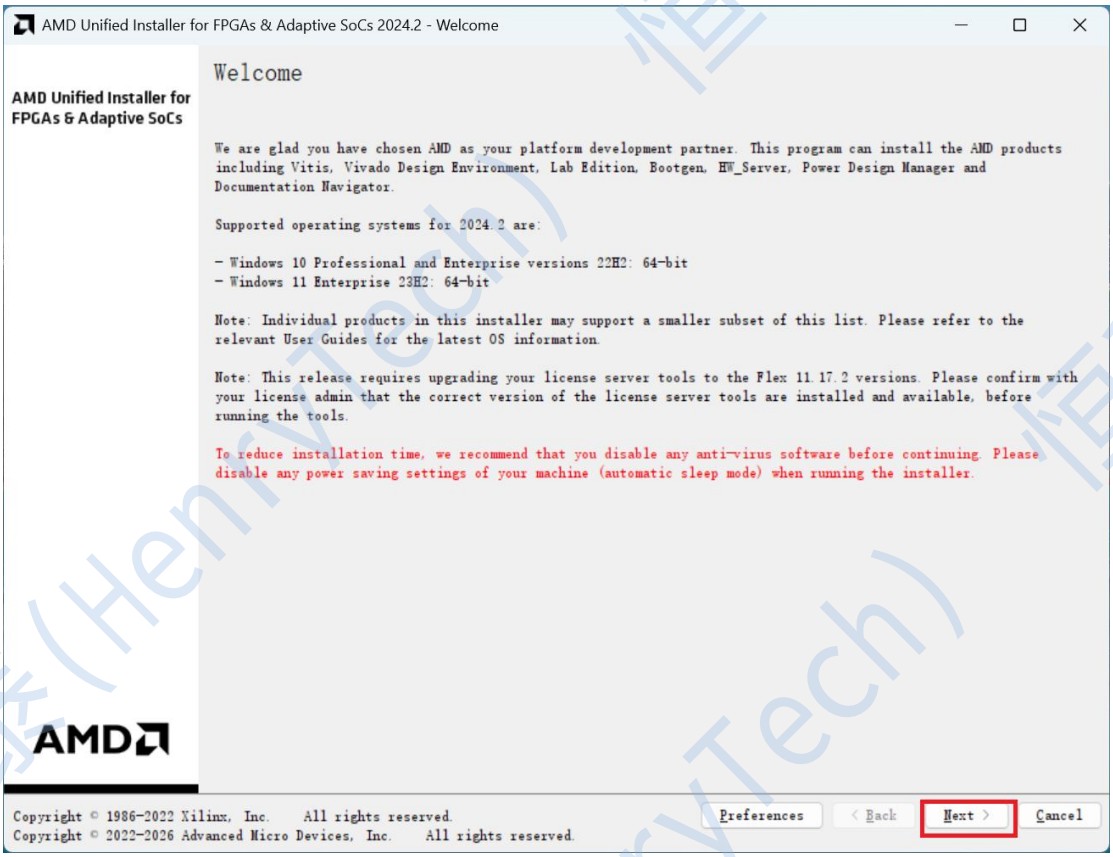
自带的安全功能和防火墙一类。否则安装过程会失败或者安装后出现一些奇怪的问题。接着我们开始安装。我们找到解压的安装包里面的xsetup.exe双击运行：



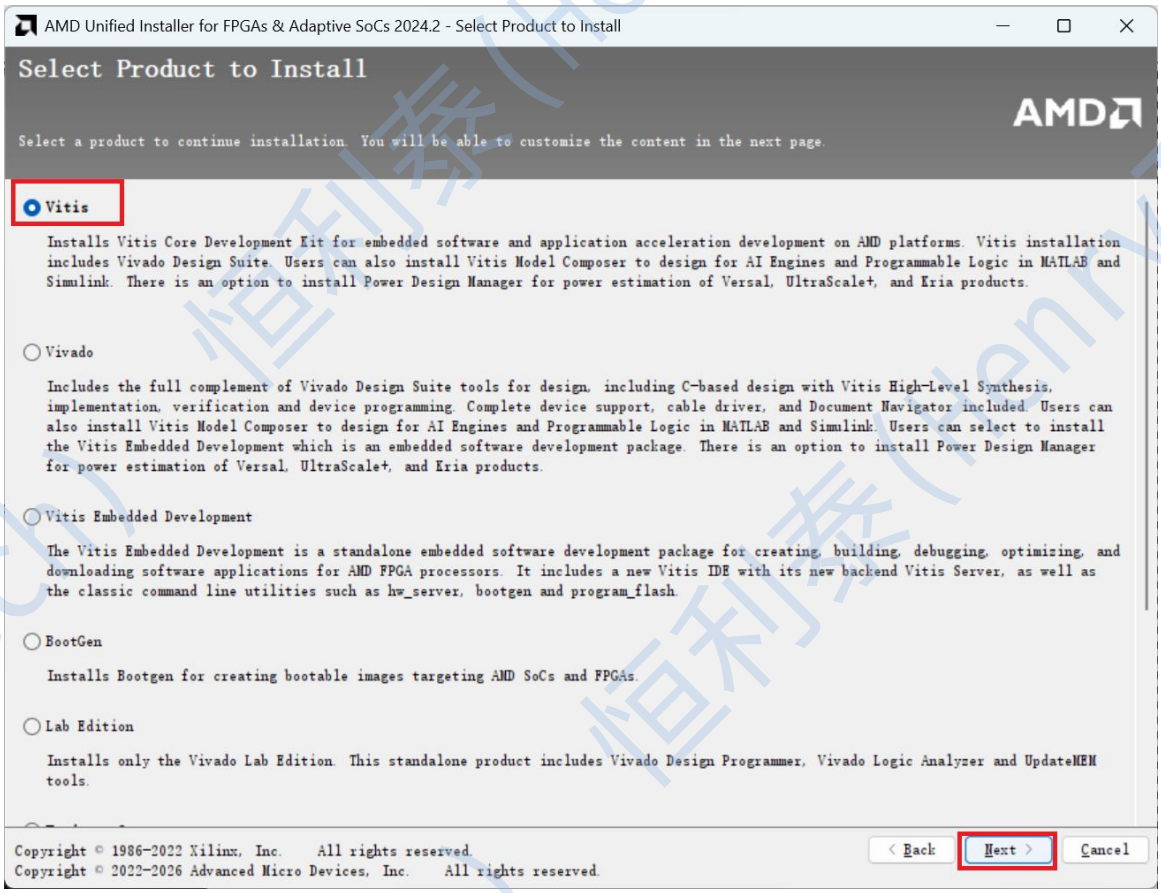
然后等待安装启动，可能弹出用户账户控制，是否允许对设备更改对话框直接点是即可。启动过程有点慢，大家耐心等待其启动。

启动后可能会弹出防火墙网络访问，我们点击允许访问。注意，后续软件使用弹出防火墙网络访问，务必全部允许，否则可能会出现软件识别不到下载器的情况。

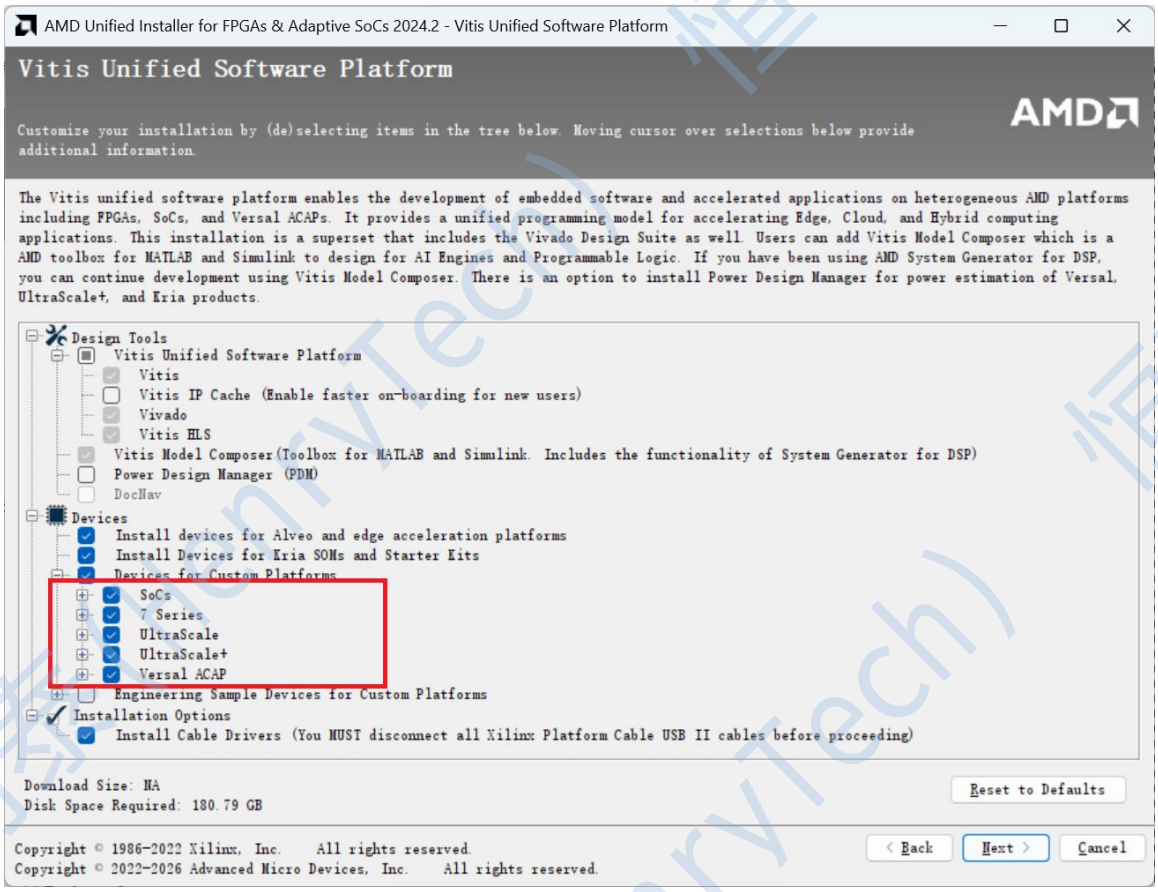
安装界面进入之后可能弹出对话框提示有更新的版本可以使用。如果弹出点击Continue关掉。这一页大家可以看到我们安装包支持的操作系统版本等等信息，然后点击Next进入下一页：



接下来界面我们选择Vitis。注意，如果大家后续需要Vitis必须选择这项，他会安装VIVADO和Vitis；但是如果选择VIVADO，只会安装VIVADO不会安装Vitis。点击NEXT：



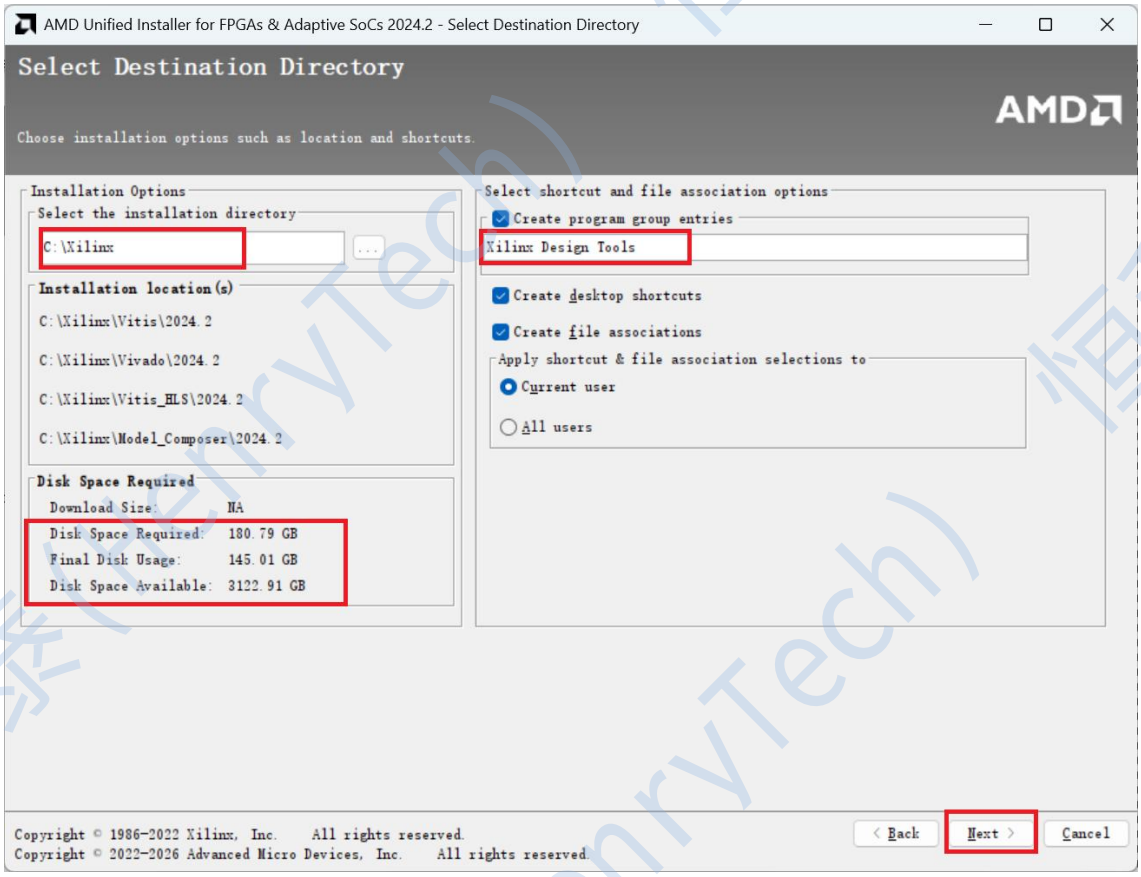
这一页我们器件库支持，最重要的是SOCS包含ZYNQ系列，点击NEXT：



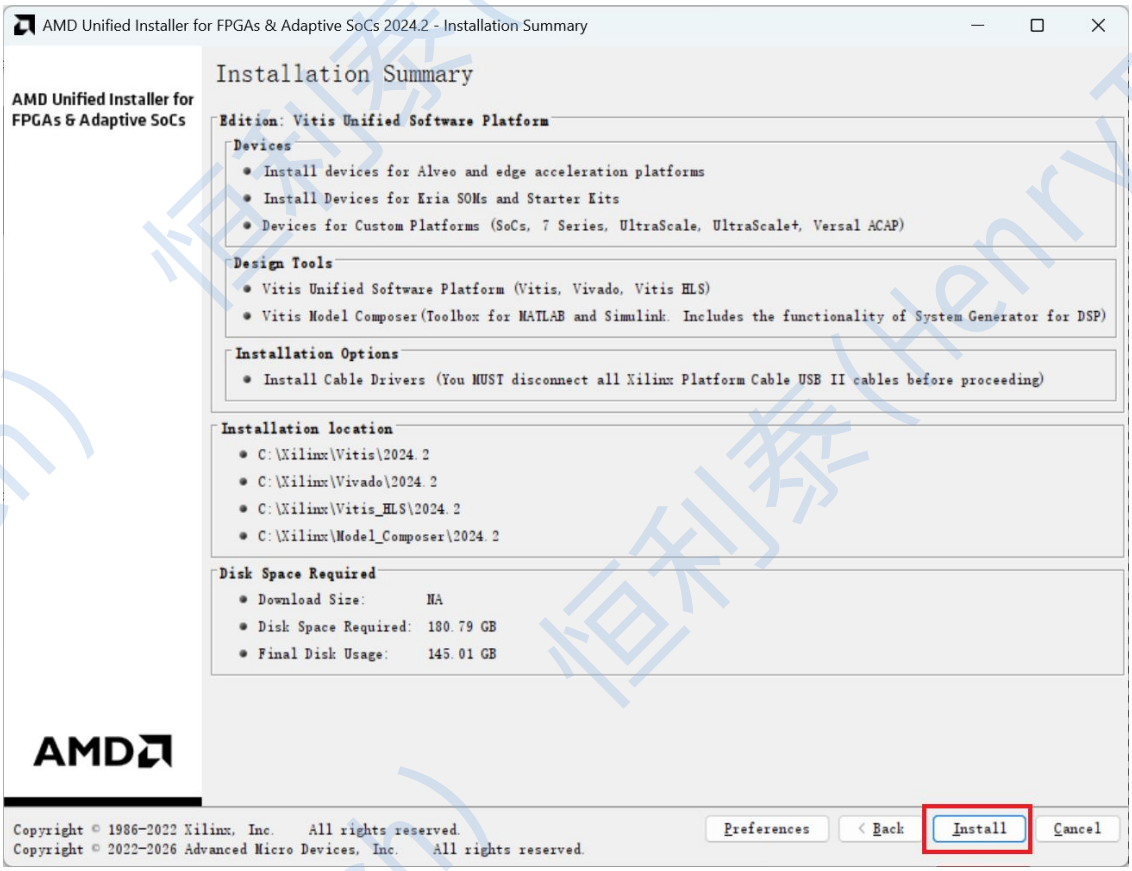
这里我们有很多条款我们勾选允许即可，注意很多，右边可以下拉，全部勾选，点击Next进入下一页：

这一页是安装磁盘目录，默认C盘。建议条件允许直接安装到C盘，其他盘可能会有一

些问题，大家C盘不够可以自己尝试。点击NEXT：

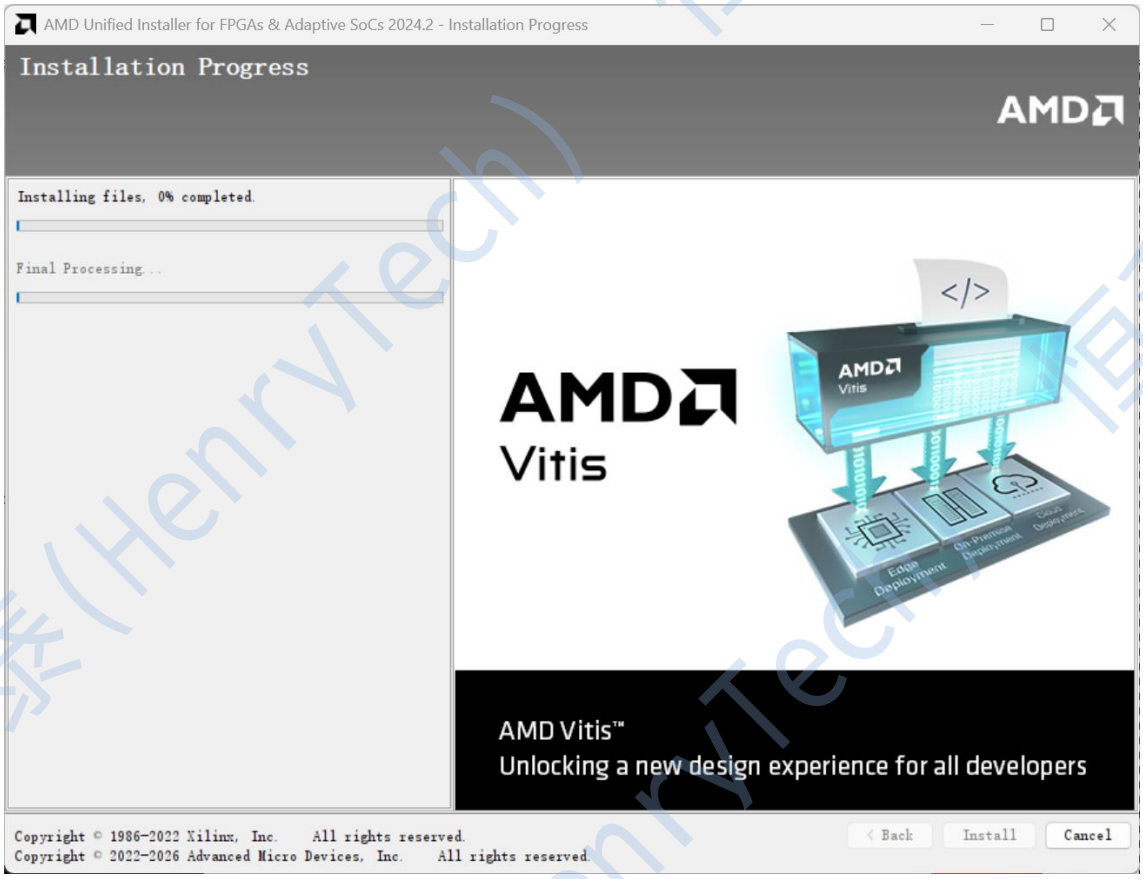


这一页我们是安装的一些选项信息确认页面，大家根据自己的安装选项核对，比如安装选项，目录是否正确，接着我们就可以点击install安装了：

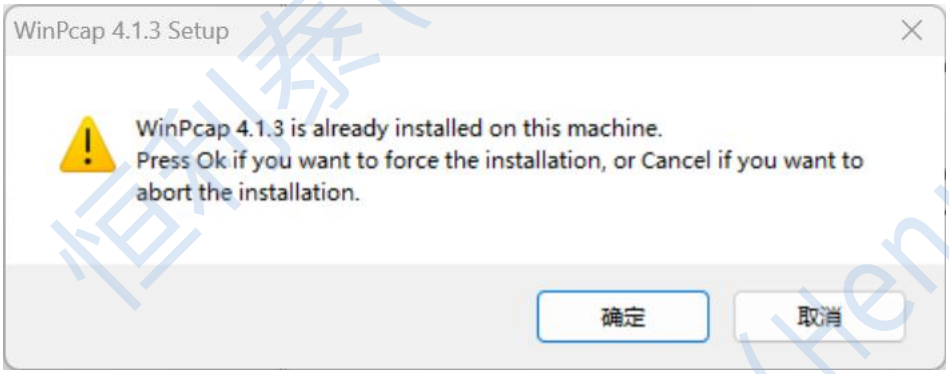


开始安装，时间很久，至少在15分钟，根据大家电脑配置，如果是机械硬盘，甚至可

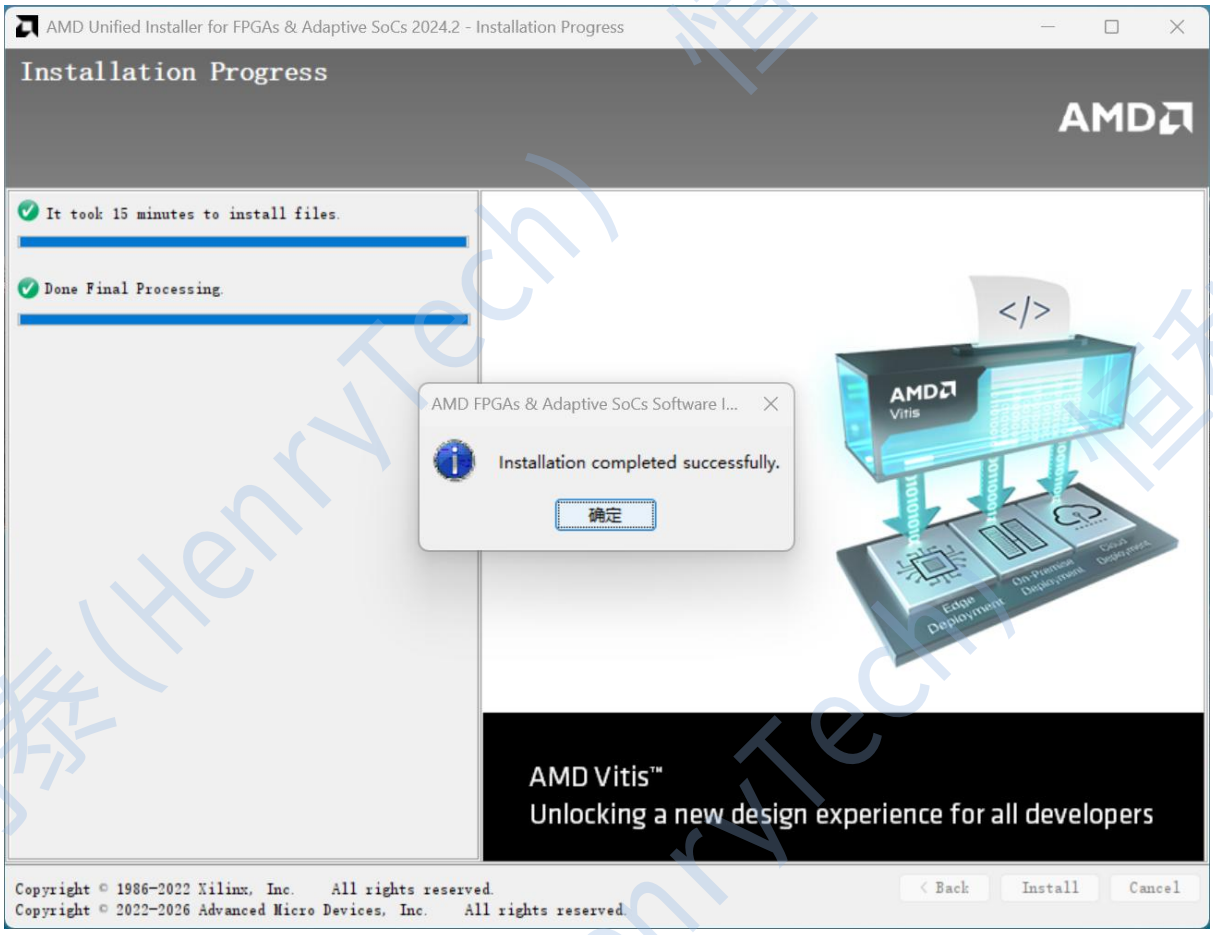
能会超过一个小时。我们耐心等待安装完成：



中途可能会弹出对话框提示已经安装过wincap，我们点击取消，如果没有安装过，就直接安装一遍即可，笔者之前安装过其他版本VIVADO所以提示已经安装过：

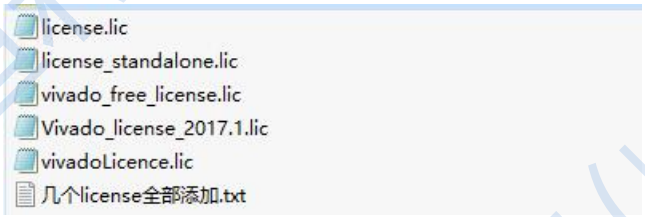


我们安装完成，弹出对话框提示，我们点击确定就完成了安装：

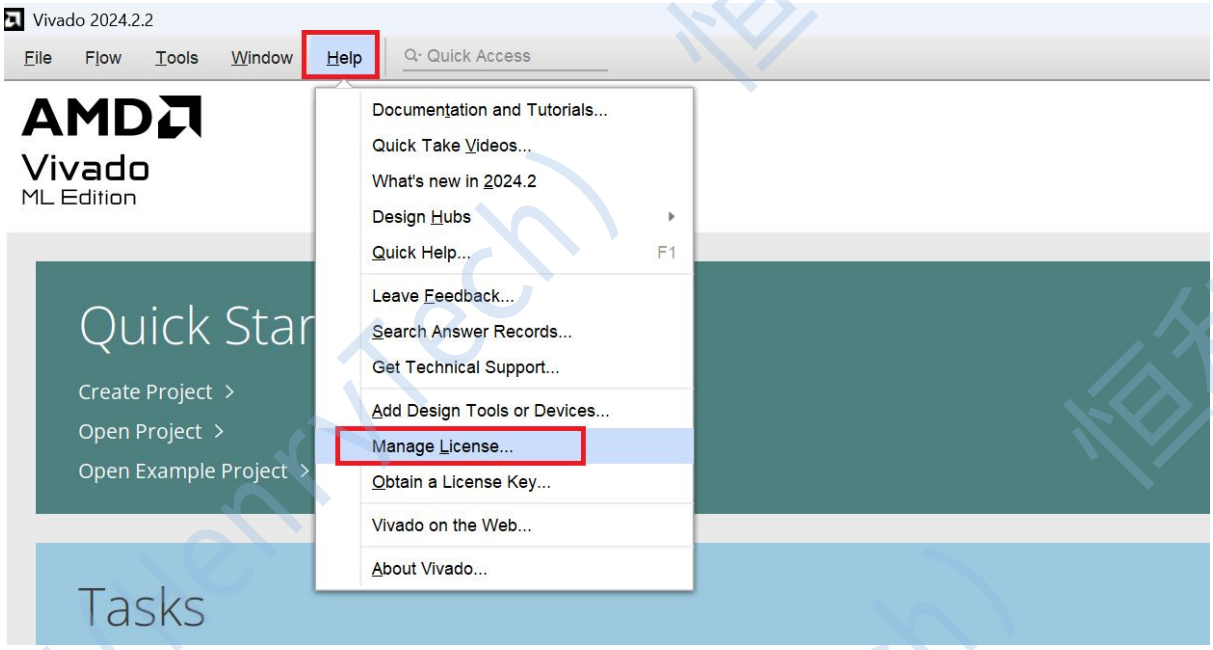


六、安装License

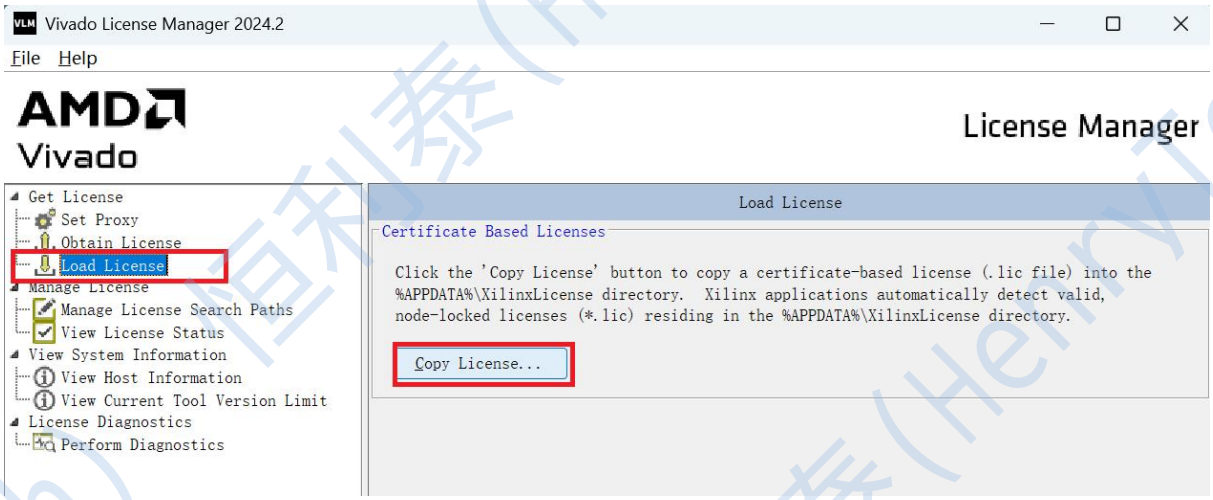
网盘我们提供了一个叫作vivadolicense的压缩包，里面包含了很多个license。其中一些是个人账户官方申请，一些是网上搜集整理的。使用的时候建议大家全部添加一遍：



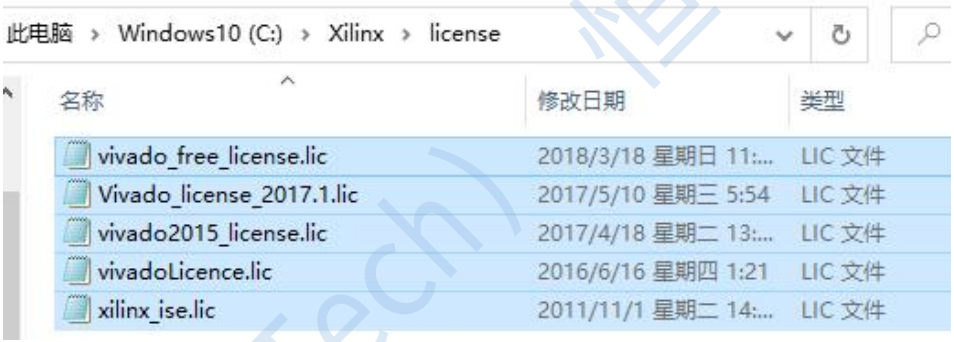
我们license添加有两种方式，第一种，在我们安装完成之后可以在开始菜单栏找到我们的赛灵思的工具软件目录，可以找到我们license管理器。第二种是启动我们vivado，然后在help菜单下面可以找到我们的license管理器。



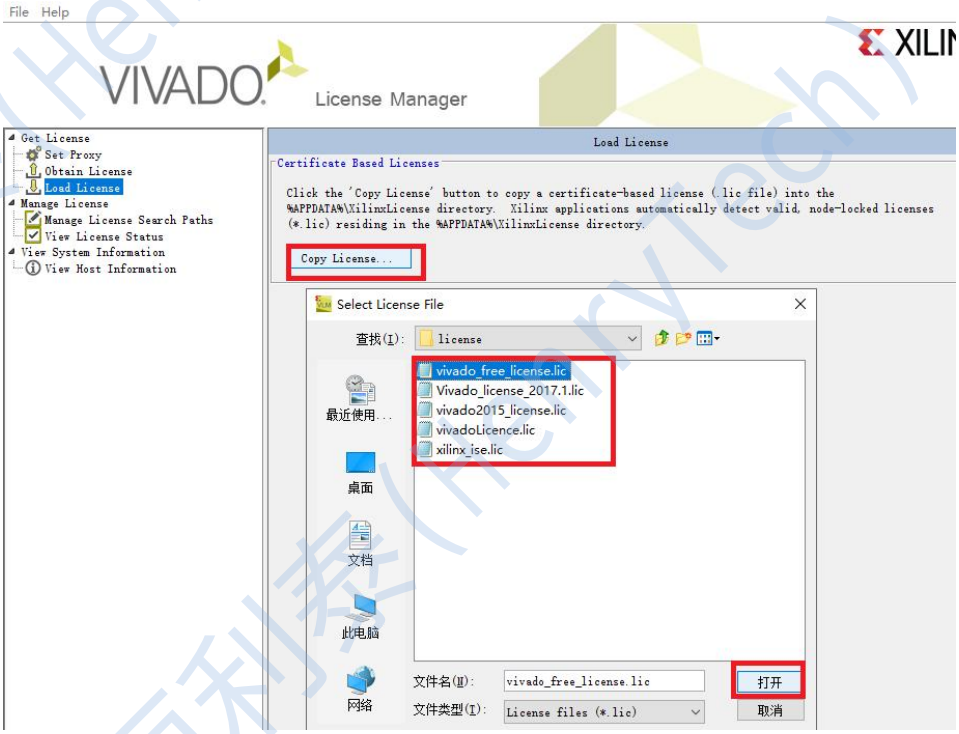
打开license管理器之后，我们点击选择左边load license，然后右边点击copy lisence选择我们的license文件添加。大家注意，license文件一定要存放到英文路径，否则加载不成功：



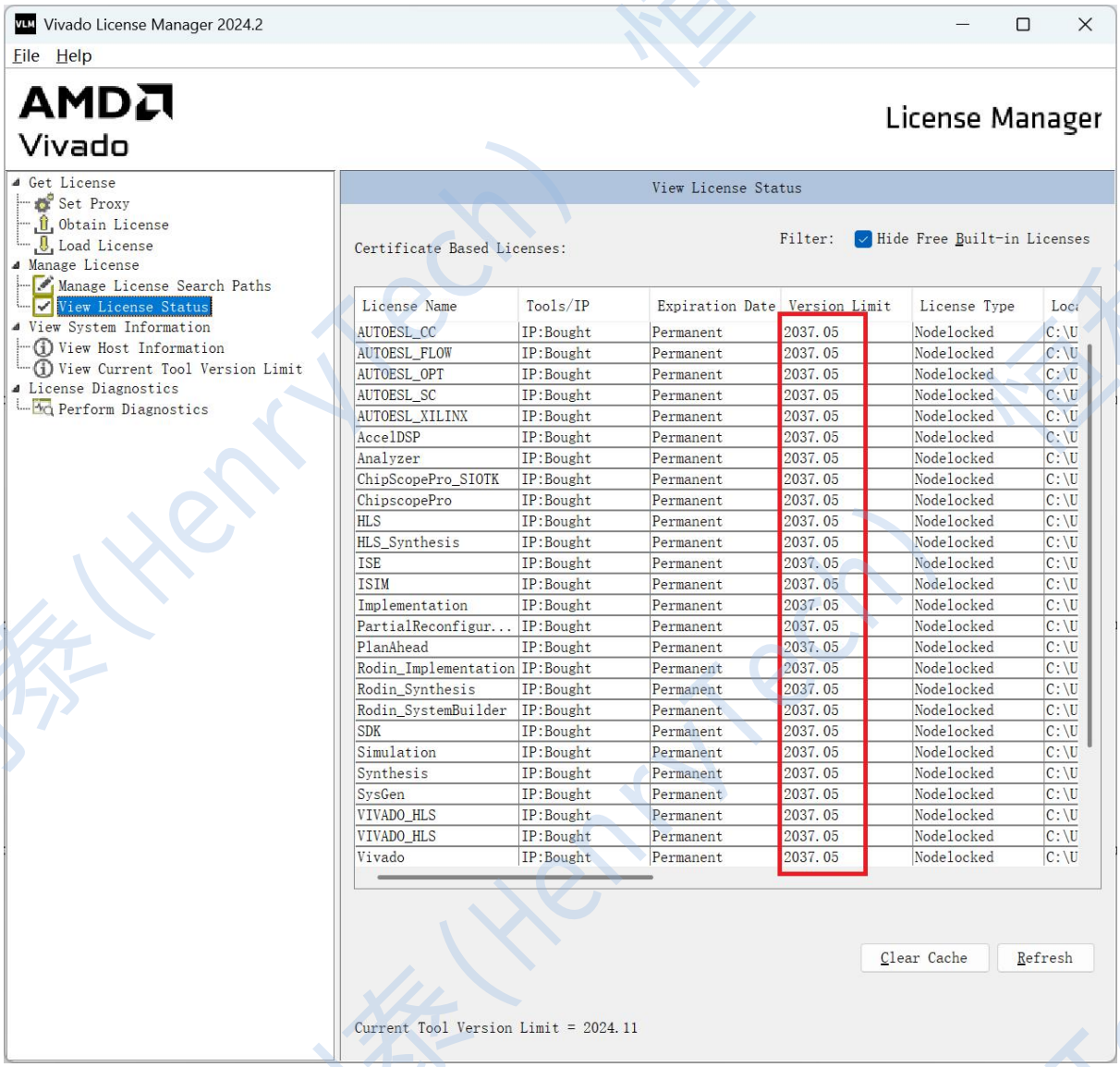
我们建议大家把license文件放到安装完成的目标目录下，新建一个叫作license的文件夹将我们提供的license放进去：



然后进行添加。注意，添加一次只能添加一个！建议把我们提供的几个license文件全部都添加一遍。选中license文件，点击打开即可。



点击之后弹出任何对话框选择是就可以了。全部添加之后我们就可以在license状态查看一下，点击左边View License Status，右边窗口可以看到license大多有效期在2037年：



我们完成license添加，就可以正常使用VIVADO和VITIS了。

七、安装驱动

驱动包含我们下载器驱动和串口驱动，FT2232HQ是二合一方案。其中下载器驱动一般情况下win10 win11是不需要单独安装驱动的。正常情况下安装完vivado会自动安装驱动。也可以自行安装。下载器JTAG和串口使用FT2232HQ，驱动位于《02软件和相关linux包->FT2232HQ下载器驱动》目录中。

安装过程这里不讲解，直接下载安装即可，注意安装驱动建议拔掉DEBUG线再安装。

VIVADO新建工程流程和软件介绍

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 新建工程步骤 3
三、 界面功能介绍 8

恒利泰(HenryTech)

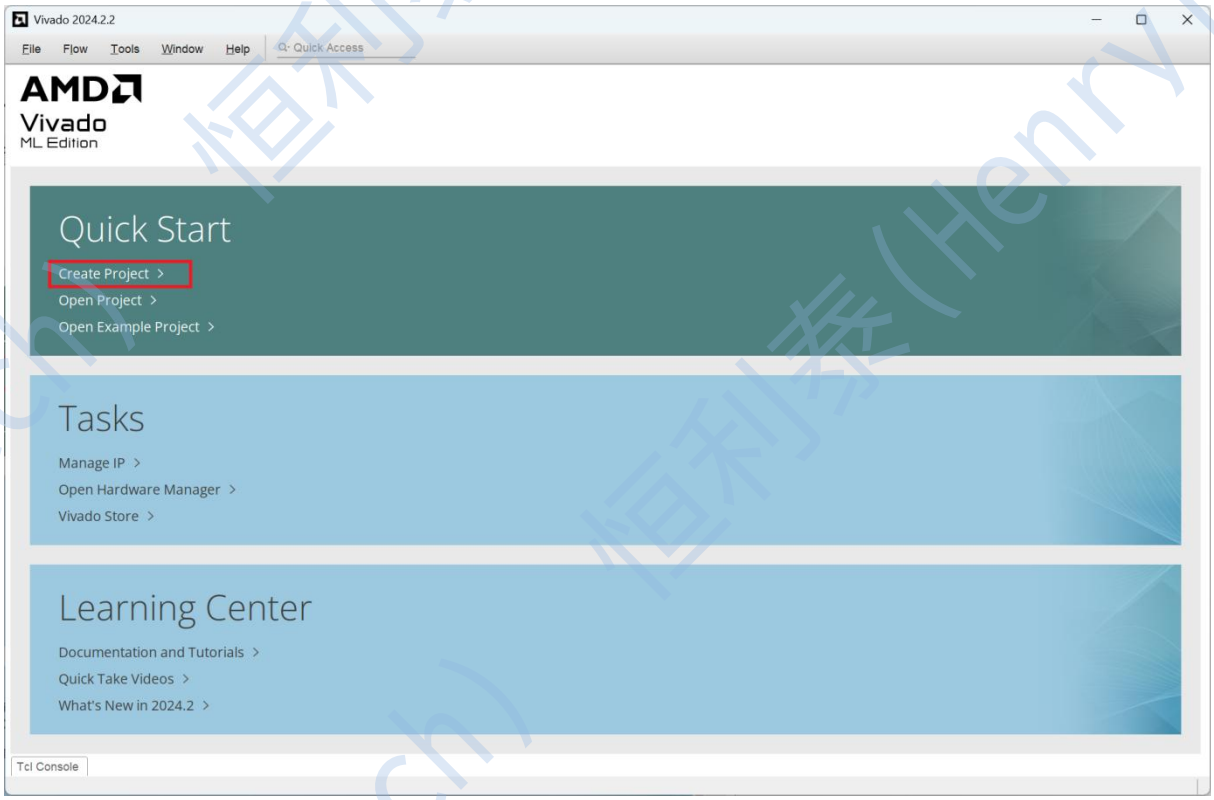
一、文档实现功能介绍

本文档详细介绍如何使用VIVADO新建工程，以及VIVADO软件的一些功能和特性。其中包括软件在我们开发FPGA时的整套完整设计流程，等等。VIVADO软件集成了FPGA开发的全套工具，包含仿真，综合，IP调用，布局布线，绑定管脚，生成bit文件，烧写BIT等等。不同于其他软件，很多功能比如综合，调用的第三方的综合工具，VIVADO本身集成了赛灵思的自己的综合工具，非第三方。另外，引脚绑定功能和烧写BIT文件功能也是集成了的，无需打开其他工具进行操作，软件功能本身很强大。但同时对电脑系统配置要求也很高，建议大家使用VIVADO的时候，电脑配置高一些，运行起来流畅，并且综合，布局布线等消耗的时间会短一些。并且，VIVADO支持多线程技术，尤其是综合阶段，很多IP代码的设计工程中，我们运行综合的时候速度会快很多，电脑配置也推荐越高越好。

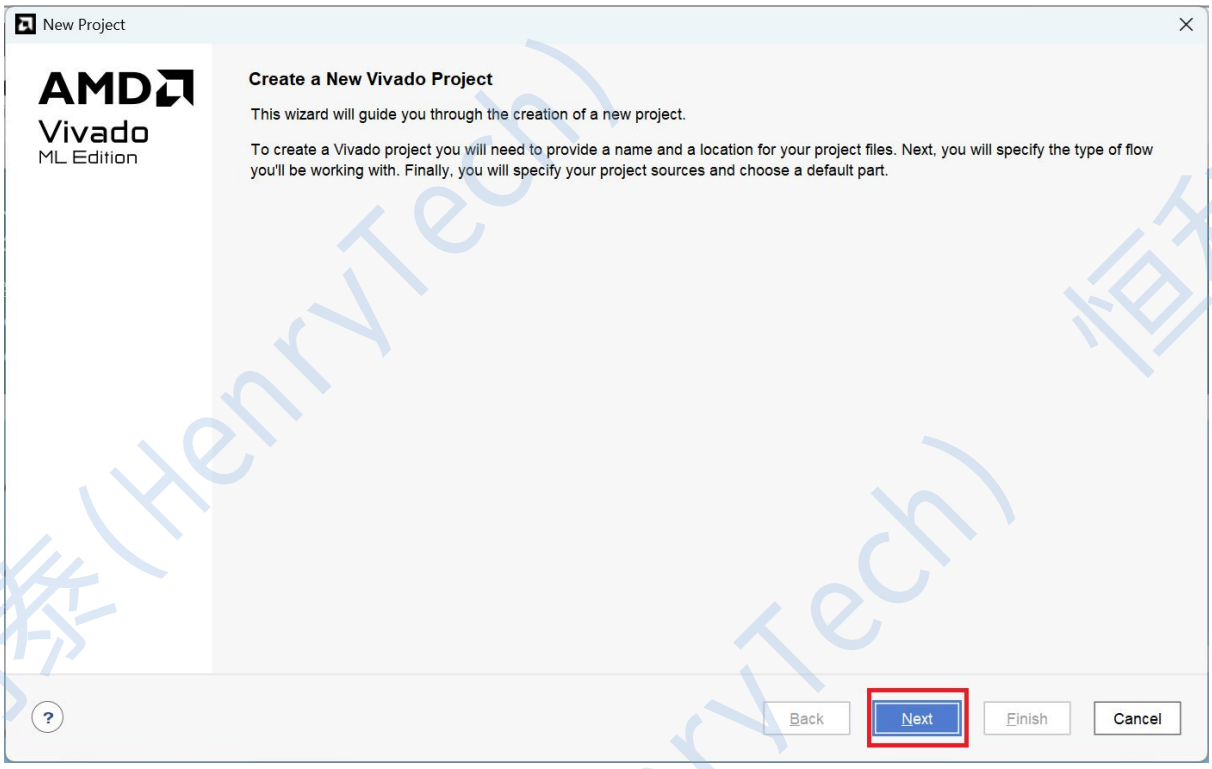
二、新建工程步骤

我们找到赛灵思的VIVADO图标双击，或者开始菜单栏找到vivado2022.2启动它：

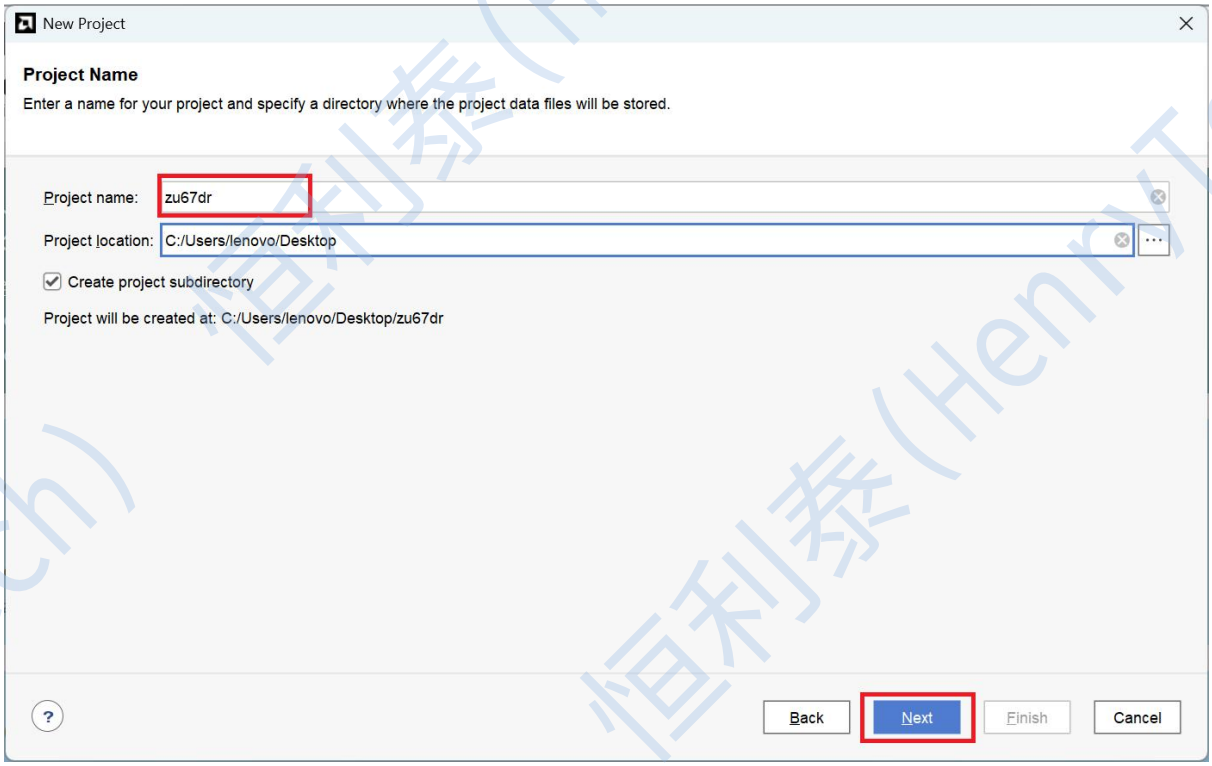
打开之后我们进入主界面：



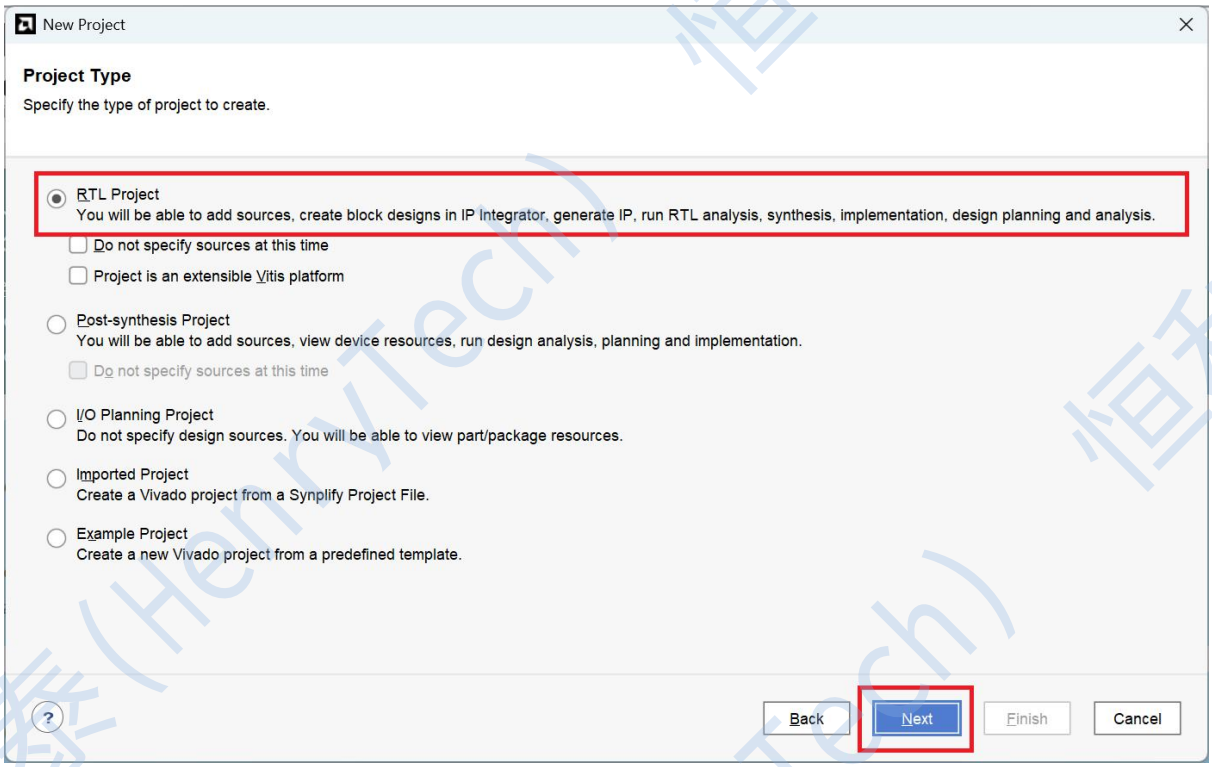
我们点击Quick Start或者左上角的菜单栏的File->Project->New来新建一个VIVADO的工程。弹出的第一个窗口我们点击Next->



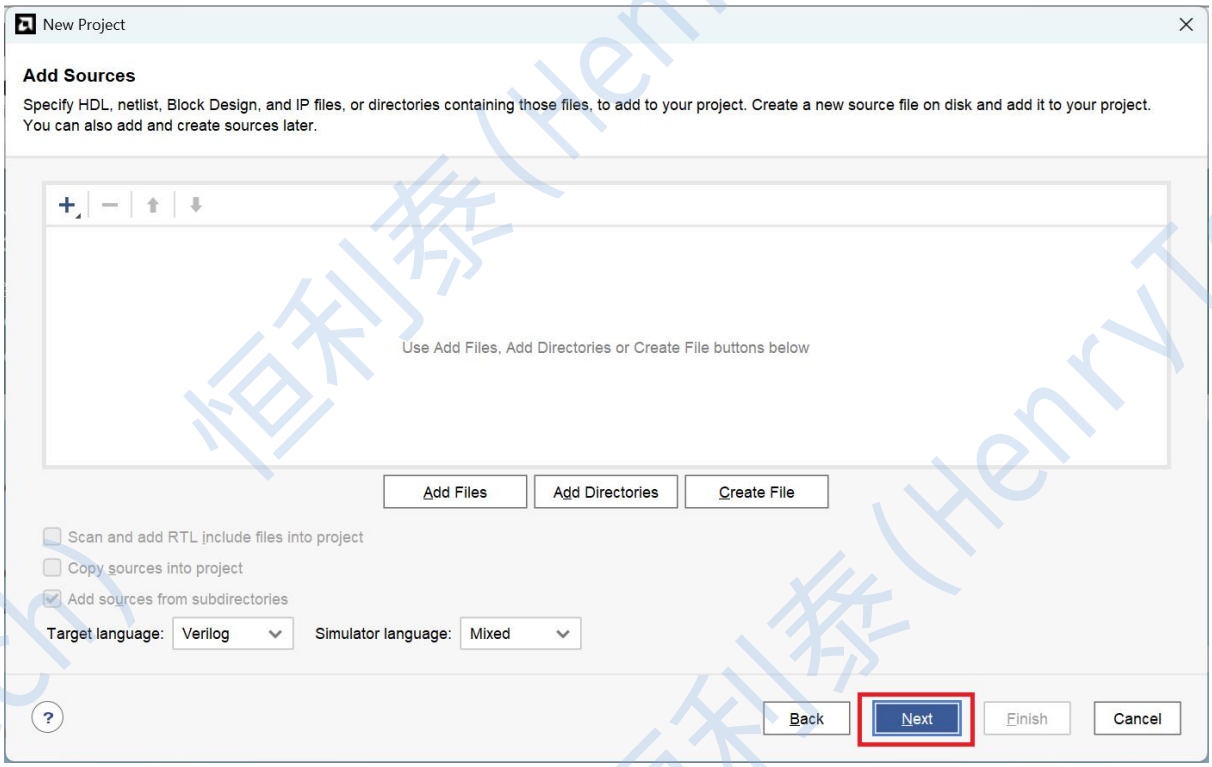
接着我们在这一页填入工程名字和路径，工程位置可放到桌面，然后点击Next>



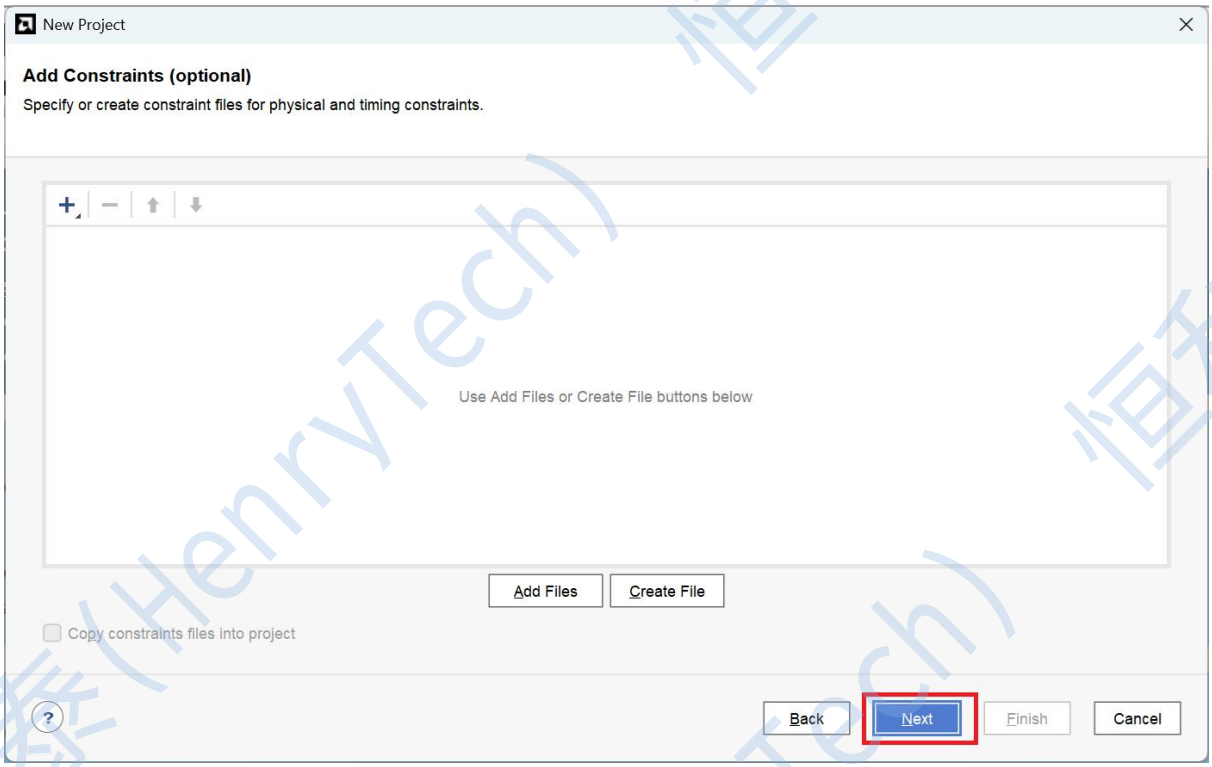
接着这一页 默认选择第一项，RTL Project，表示工程是一个标准的全套FPGA开发流程的设计工程，然后我们点击Next>



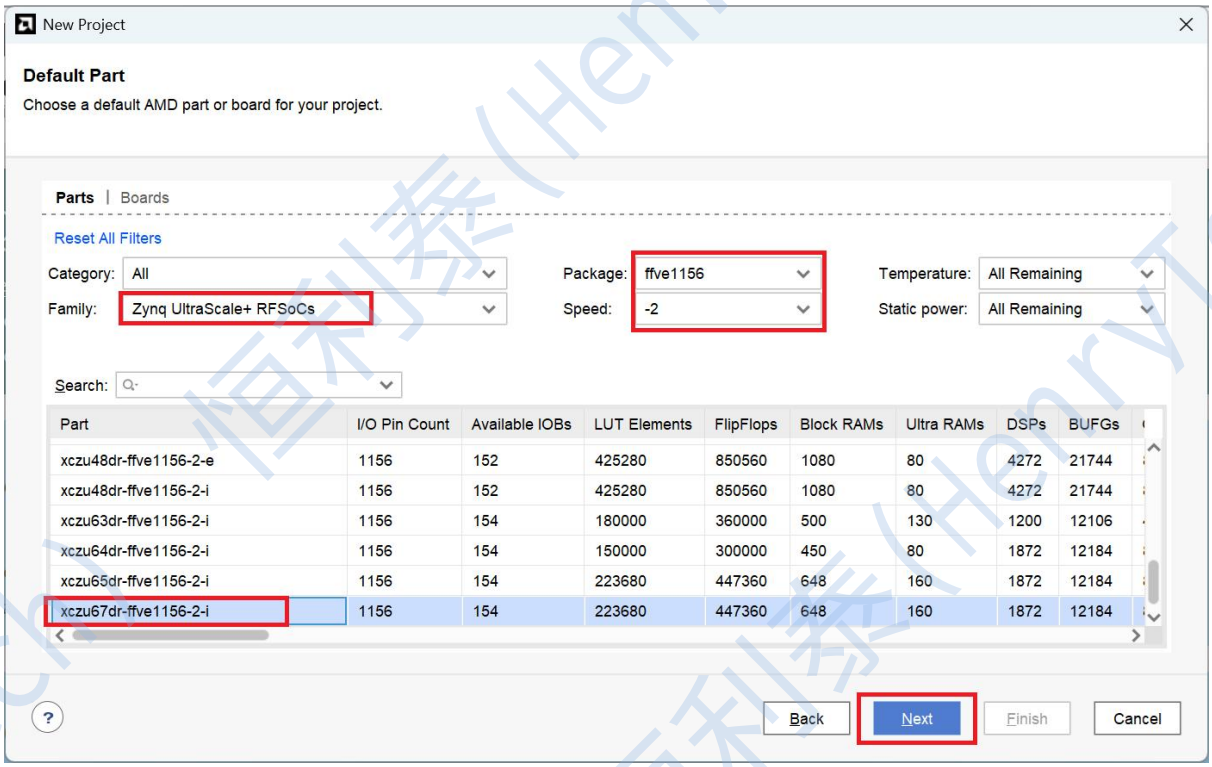
接着这一页让添加我们的FPGA源代码文件，我们不需要添加，我们点击Next>



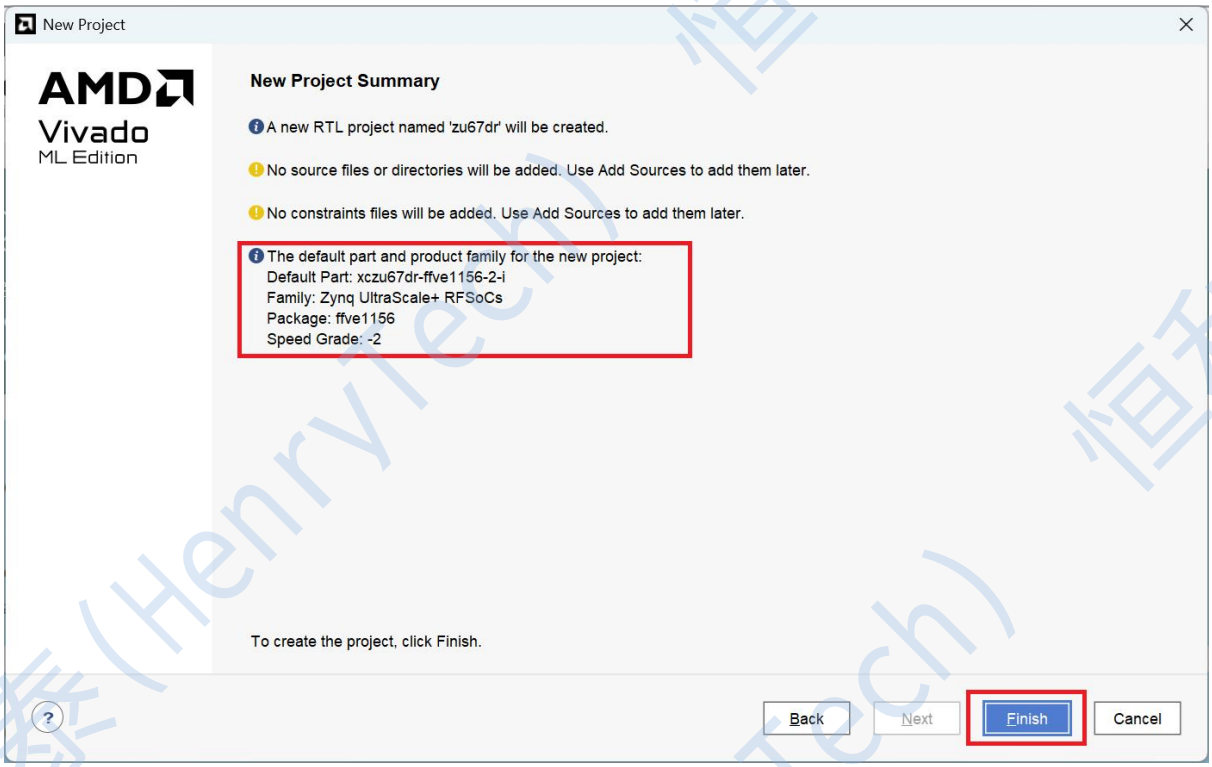
接着的这一页是让添加我们的约束文件。所谓约束文件就是后缀为。xdc文件，这个文件包含我们的引脚约束，电平标准，时钟约束等等。后面我们其他教程讲到如何绑定引脚，最后绑定好的这些信息都会被存储到一个后缀为。xdc的文件里面：



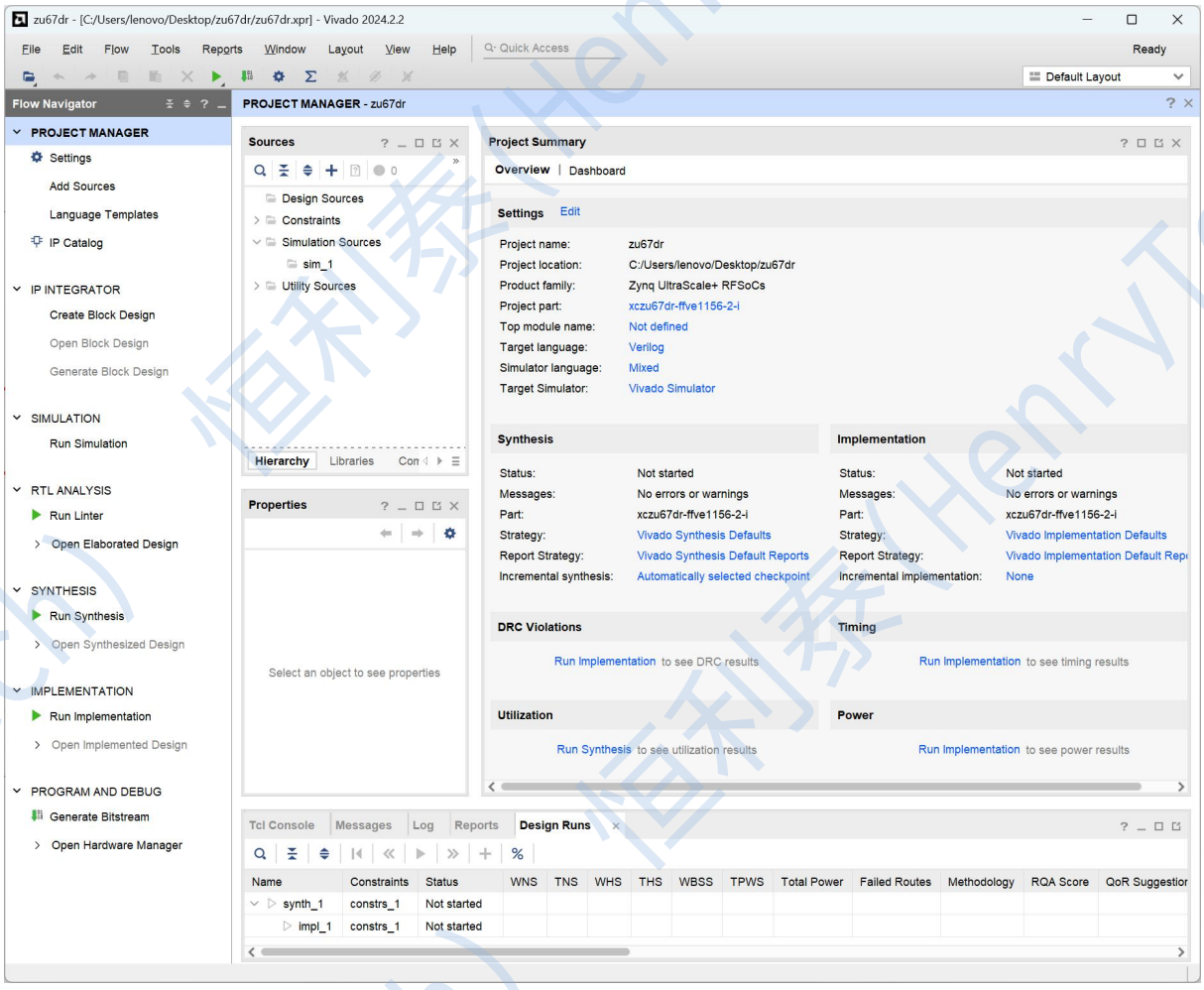
接着这一页我们进行芯片的选型。我们芯片型号xczu47dr-ffve1156-2-i，点击NEXT>



最后一页，工程概况。大家认真核对一下配置，比如芯片型号是否正确，然后点击Finish完成我们的工程新建。



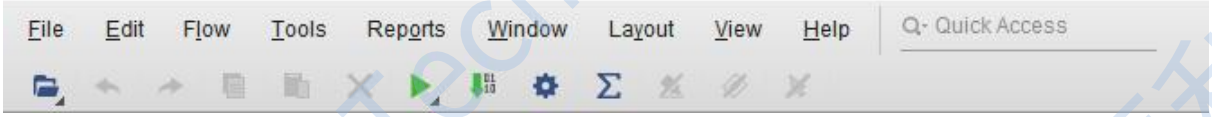
我们等待工程新建之后打开。到这里我们新建工程就结束了，大家可以看到我们工程主界面：



三、界面功能介绍

1、菜单栏

菜单栏位于软件最上方。菜单下拉包含各种功能和工具：



2、开发流程导航栏。

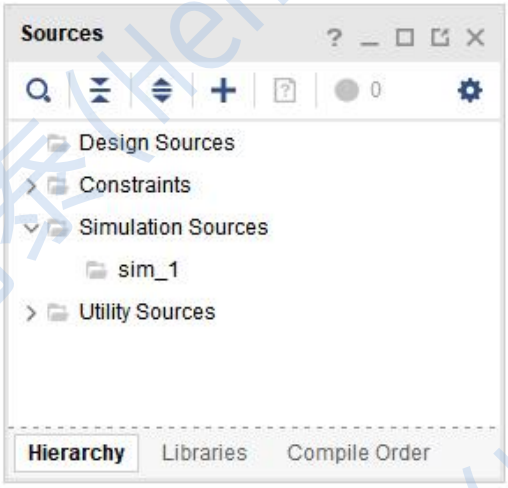
开发流程导航栏位于软件左边，我们可以通过下图了解一下软件功能和开发流程：

- 工程设置：可以设置整个工程配置，常见的有芯片型号，IP库路径等等。。里面几乎包含整个工程所有设置；
- 添加文件：可以向工程添加设计源代码，仿真源代码，约束文件，IP核等等；
- 原语模板：包含芯片有关的底层逻辑单元的代码模块调用的范例；
- IP库：包含VIVADO所有自带IP以及添加到工程的IP库的调用；
- 新建bd文件：VIVADO不仅支持源代码开发，也支持图表模块连接可视化的设计方法，bd是

viavdo的图形设计文档格式文件；

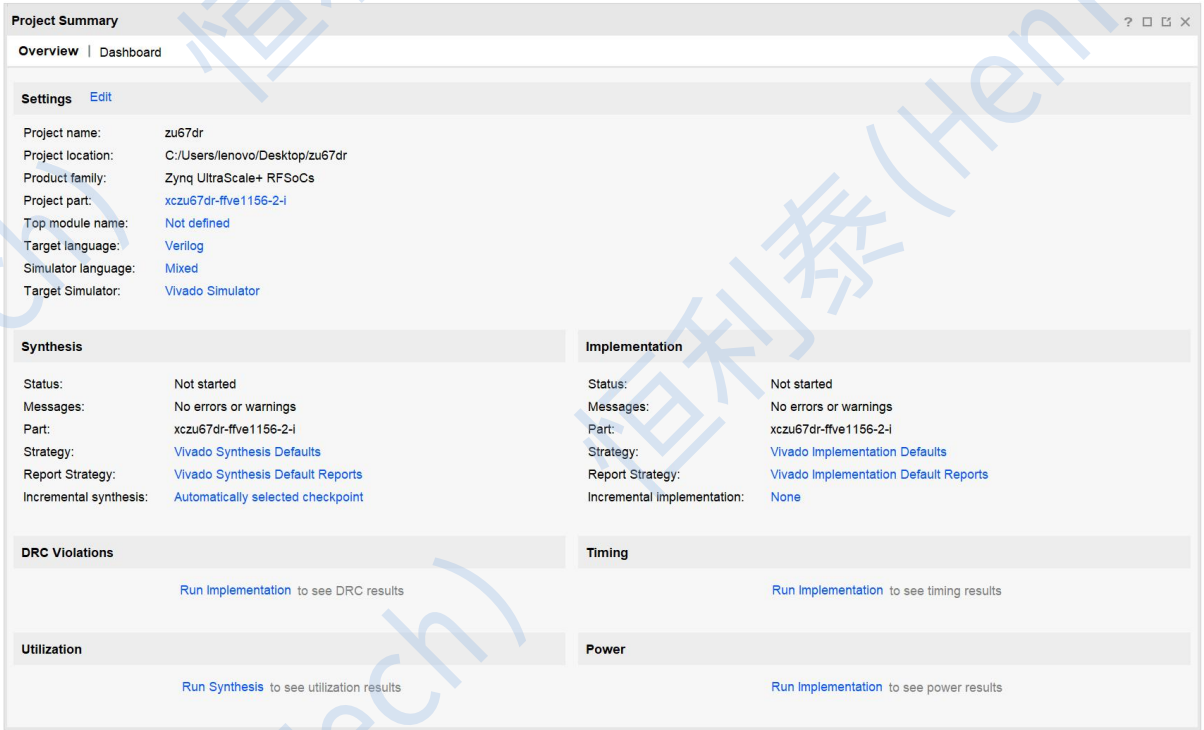
- 运行仿真：vivado自带的赛灵思的仿真工具，提供代码开发的仿真功能，无需调用modelsim等第三方工具。
- 工程详尽设计：包含工程的代码的逻辑原理图，以及管脚约束绑定等等；
- 综合：将源代码编译成原理图网表，获得整个工程的代码的逻辑关系；
- 芯片的实现：包含布局布线。根据芯片选型，在指定的芯片上完成布局布线功能；
- 生成BIT文件：将布局布线结果，生成BIT文件，可下载到芯片完成逻辑的部署；
- 硬件管理：可以打开连接我们的开发板的芯片，通过下载器。并且支持烧写FLASH。

3、source资源窗口：



source窗口可以看到我们整个工程所有的源代码和仿真文件（verilog,VHDL等），约束文件（xdc），以及我们IP核，rom初始化文件（mif），仿真的tb文件等等。假如工程使用bd图表文档方式，还可以看到我们的bd文档。

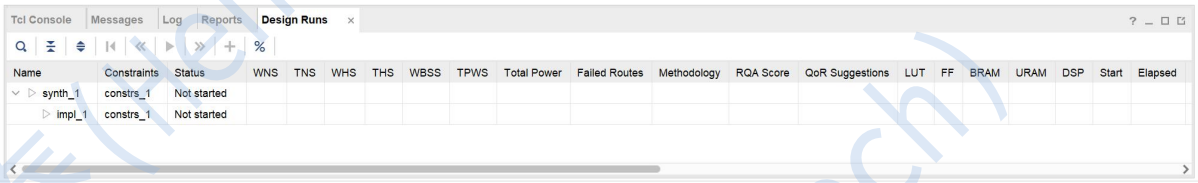
4、工程概况



我们适当调整一下窗口，如上图，可以看到工程概况更多信息展示。工程概况包含我们基本配置信息，比如芯片型号，工程使用的逻辑语言（verilog还是VHDL），仿真语言设置，综合的信息（比如综合的策略和报告策略），以及布局布线的策略及配置等。

5、状态和信息，终端，运行状态栏

这部分界面包含很多个工具窗口，包含我们的TCL脚本交互TCL Console，以及打印信息Message，日志LOG，报告Report，运行状态Design runs等子窗口。这些窗口在我们使用脚本、运行我们的综合、布局、以及布线或者生成BIT文件的时候，打印相关过程信息或者错误信息以及日志，报告信息等等。Design runs窗口可以查看我们比如综合和布局布线等运行状态以及持续时间，方便我们知道软件当前正在做什么。



6、状态显示

软件运行的时候，会显示状态，在软件的右上角。如果没有综合、布局布线、生成bit等等，就会显示ready：

以上我们关于VIVADO主界面的东西就简单介绍完了。我们后续通过我们的实际例子，给大家再详细介绍软件更详细的使用。

最简单的LED流水灯设计

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 VIVADO 工程建立 3
三、 代码添加编写 3
四、 绑定引脚约束 11
五、 约束 15
六、 生成BIT上板验证 15
七、 常见问题 20

恒利泰(HenryTech)

一、文档实现功能介绍

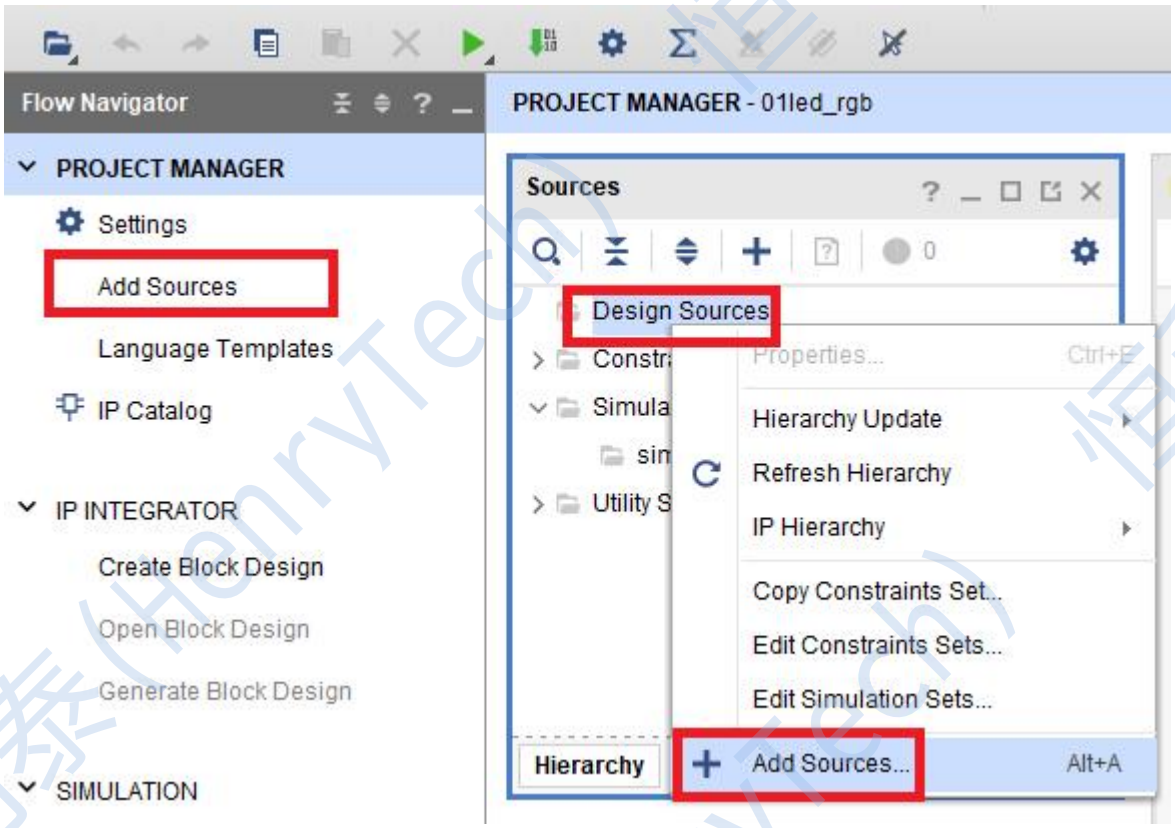
本文档实现在 VIVADO 下面新建一个 FPGA 开发工程，然后采用时序和计数实现 LED 的流水显示。开发板进行最简单的测试和开发流程就是进行最简单的点灯操作，以此来完整的跑一边简单的流程，工程新建方法请参考教程文档《04VIVADO新建工程流程和软件介绍》。注意，如果是第一次使用VIVADO请务必按照此教程完整的学习一遍流程，这样后续使用开发就比较方便，针对其他平台切换过来的朋友。

二、VIVADO 工程建立

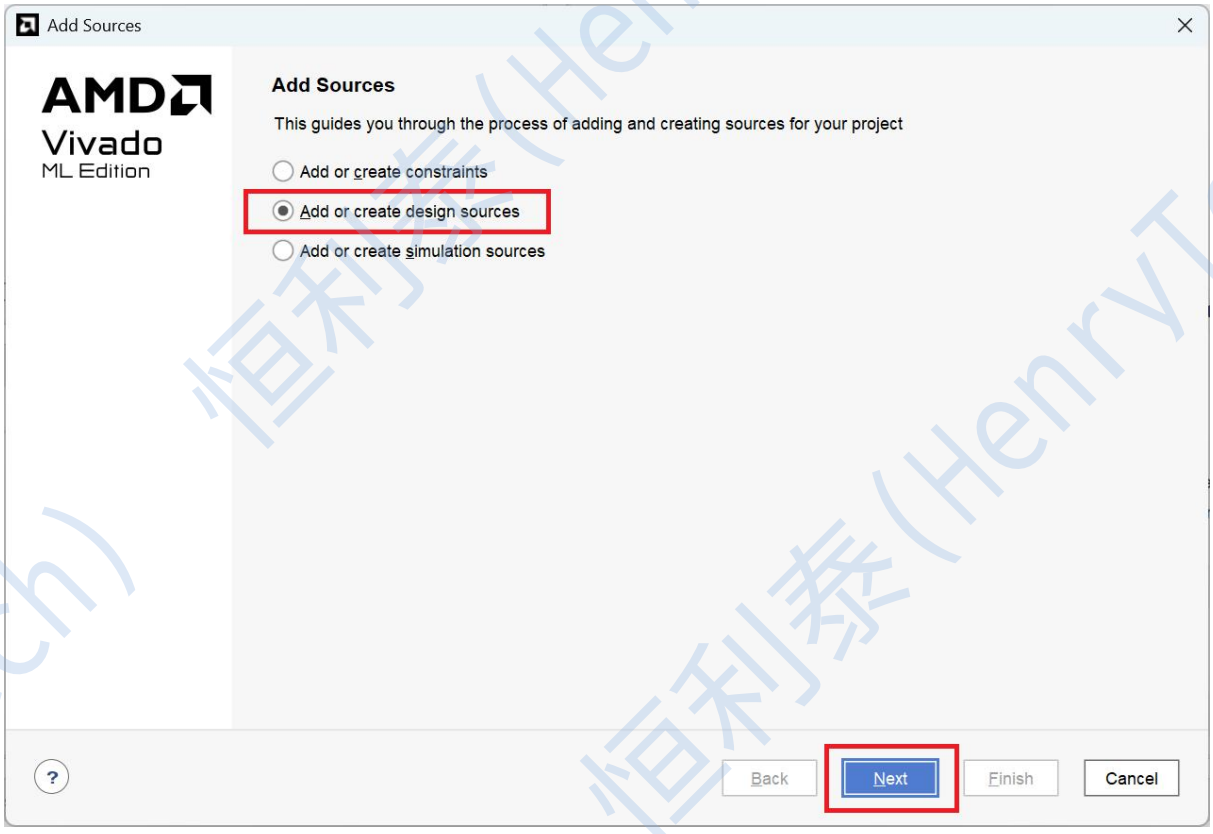
起始页（或 file->Project->New）	创建新工程(Create New Project)
向导起始页面	点击 Next->
Project Name(工程名)	工程名：01led_rgb 工程路径：(自己选择，不要有中文路径) 勾选 Create Project Subdirectory,点击 Next->
Add Source(添加设计源文件)	点击 Next->
Add Exsixting IP(添加已有的 IP)	点击 Next->
Add Constraints(添加已有约束文件)	点击 Next->
Default Part(默认配置，芯片选型)	Family->Zynq UltraScale_RFSoCs Package->FFVE1156 Speed->-2 按具体型号选择 XCZU67DR-FFVE1156-2-i
New Project Summary(新建工程概况)	确认工程信息，选型等，点击 Finish 完成

三、代码添加编写

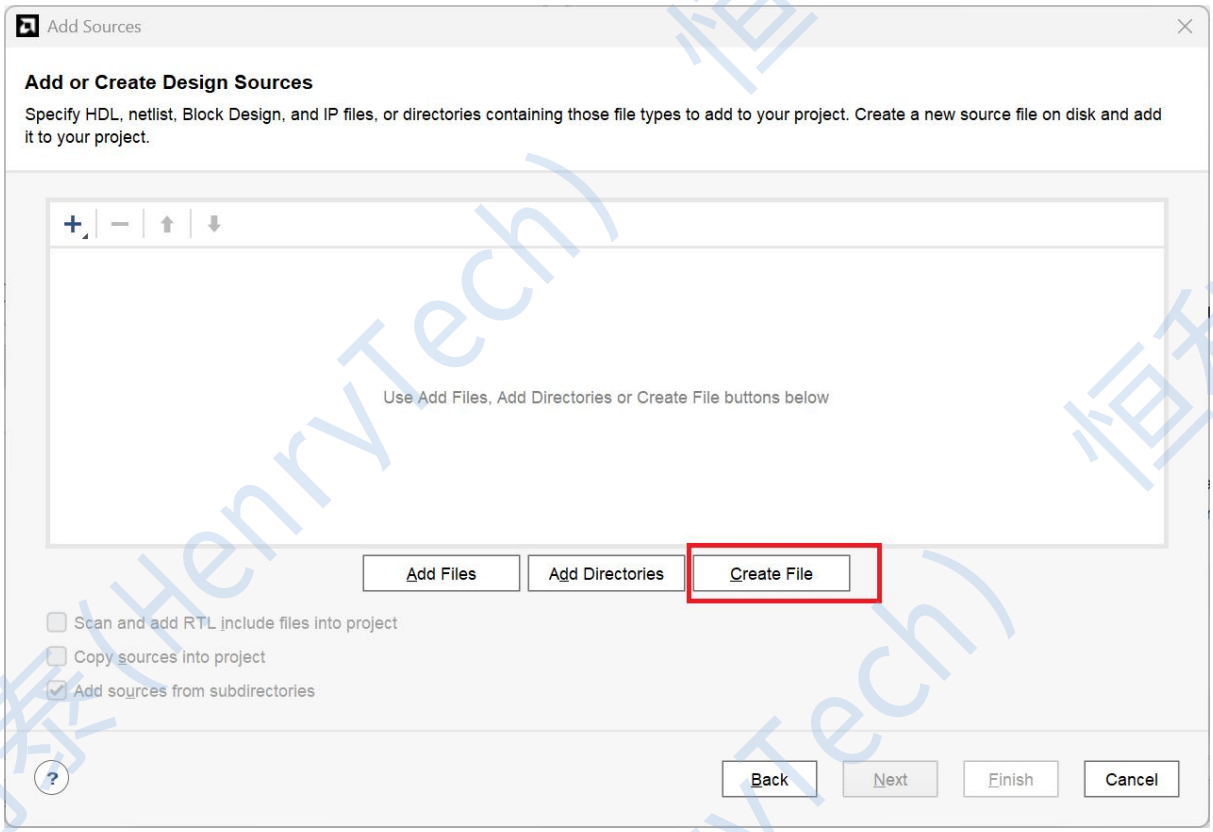
工程管理栏点击 Add Sources 添加我们需要编写设计的流水灯代码文件，或者在sources窗口的Design Sources文件夹右键选择Add Sources创建我们的代码文件。



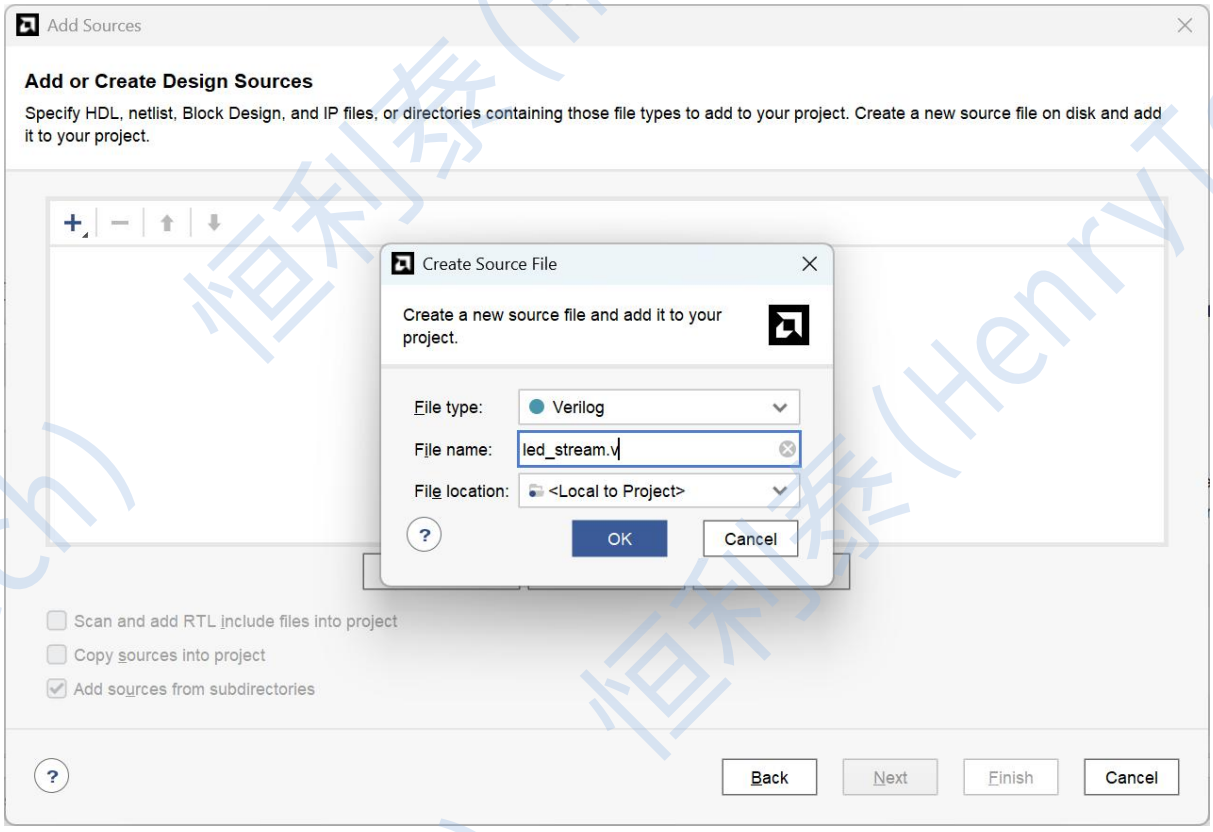
第一页选择中间这项，是创建design source，点击next:



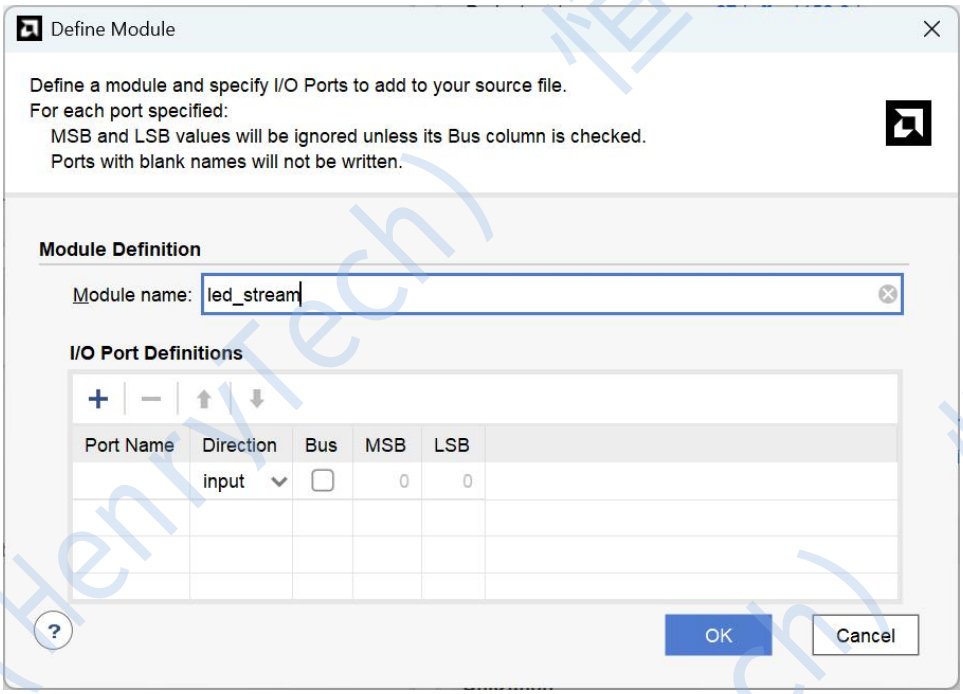
弹出的界面选择Create File。当然如果你有写好的代码，可以点击Add Files添加进来。



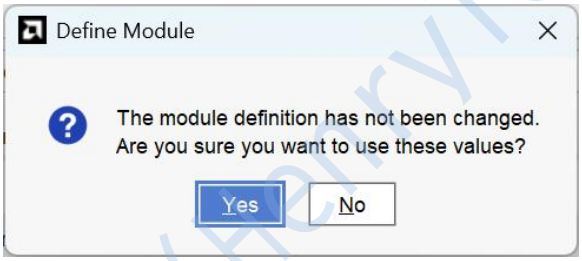
然后点击 Create File，弹出的子对话框中，File Type 默认是 verilog，如果采用 VHDL 编写，则选择为 VHDL。我们默认采用 verilog。File Name 填入 verilog 文件名，我们这里填入的是 led_stream.v。然后点击 OK完成文件创建。



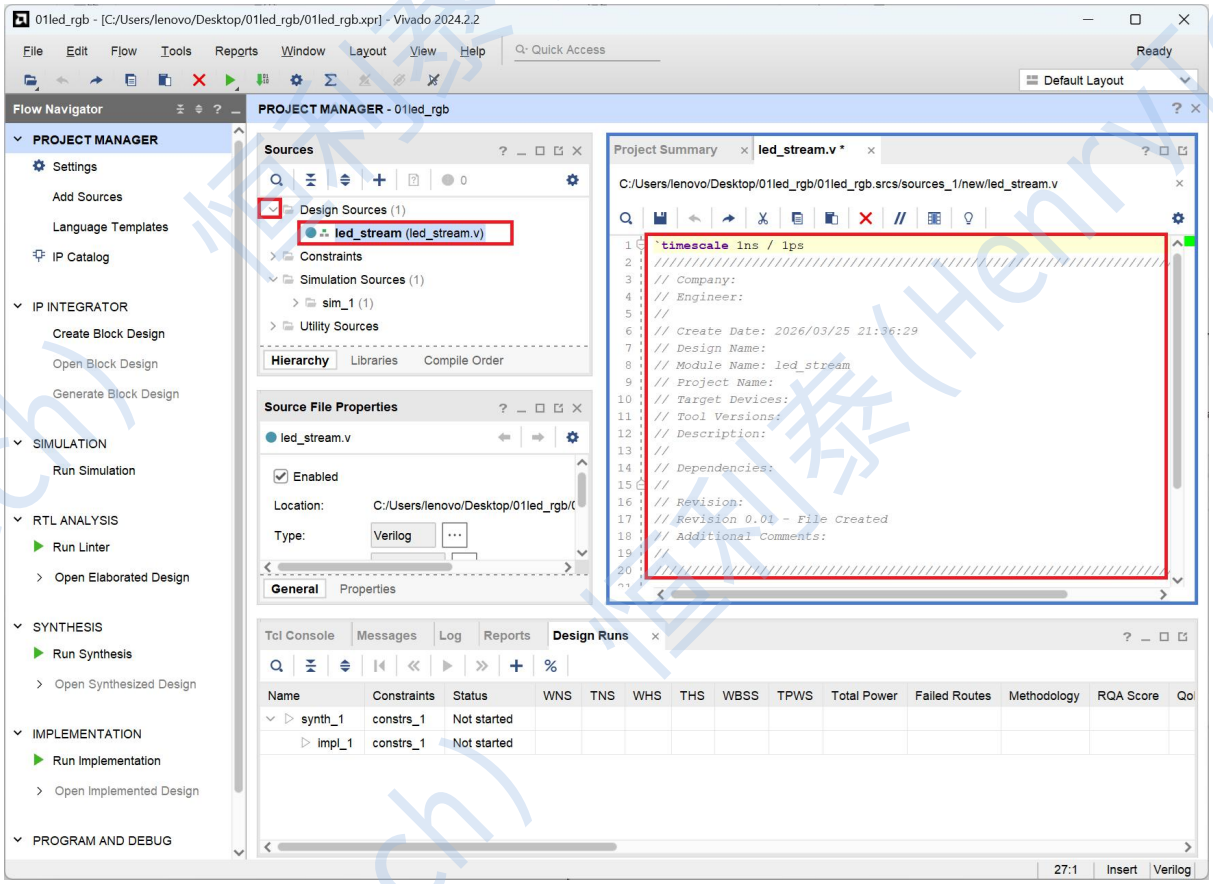
点击 Finish 完成文档创建。点击Finish 之后，弹出对应文档的模块名称。这里我们默认led_stream，然后点击OK。



然后弹出的对话框提示直接点击YES：



我们可以看到，led_stream 模块和文档已经在设计文件层次窗口中。



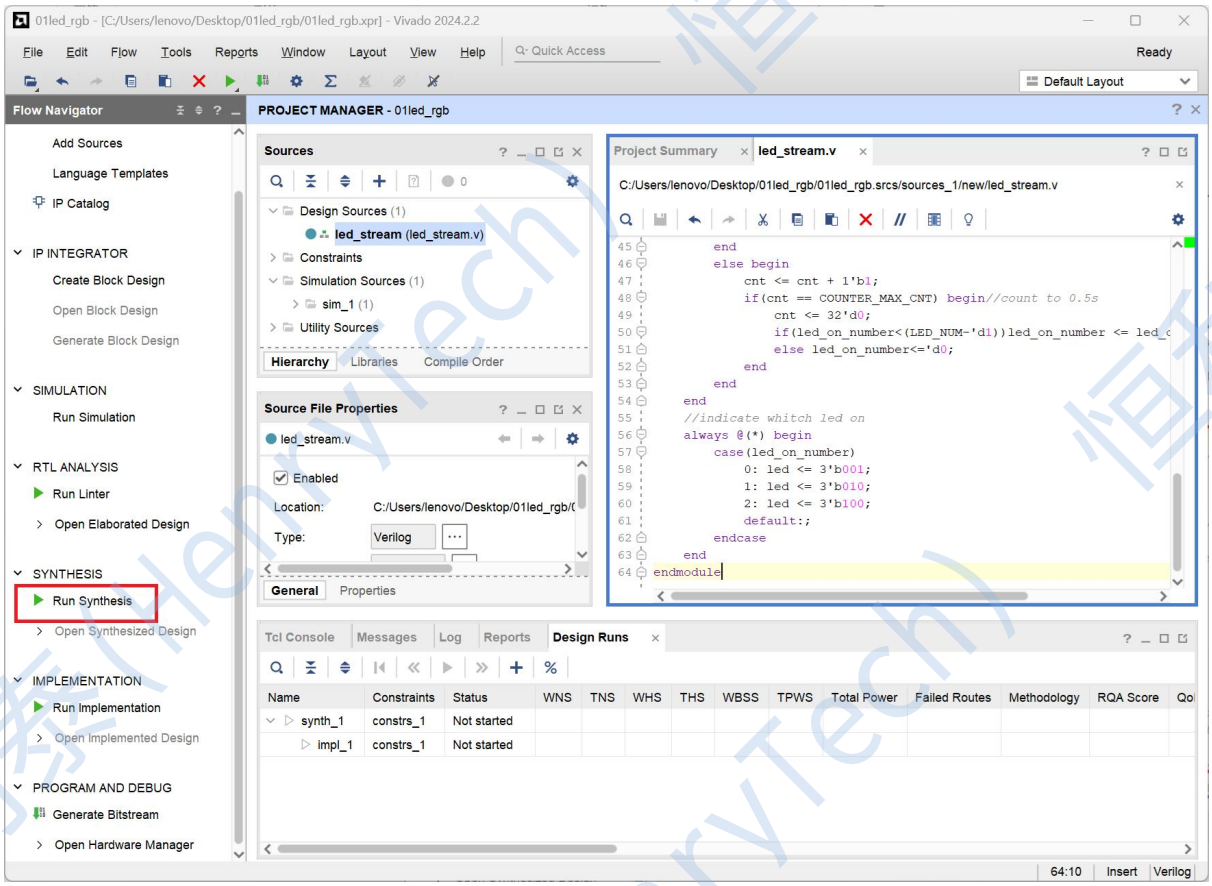
我们双击打开 led_stream 然后对其进行编辑，添加如下代码（代码请从对应的demo里面拷贝，这里拷贝pdf文字格式可能会发生变化）：

```
//////////////////////////////////////////////////////////////////////////////////
// Company: bochenjingxin
// Engineer: yang
// Create Date: 2024/05/16 12:35:02
// Design Name:
// Module Name: led_stream
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////
module led_stream(
    output reg [2:0] led,  // LED4 to LED1, 1 on, 0 off
    input            clk_n,  // FPGA PL clock n, 100 MHz
    input            clk_p,  // FPGA PL clock p, 100 MHz
    input            rst_n // FPGA reset pin
);

parameter LED_NUM='d3;
wire clk;

   IBUFDS IBUFDS_inst (
      .O(clk),   // 1-bit output: Buffer output
      .I(clk_p),   // 1-bit input: Diff_p buffer input (connect directly to top-level port)
      .IB(clk_n)  // 1-bit input: Diff_n buffer input (connect directly to top-level port)
   );

reg [31:0] cnt;
reg [2:0]  led_on_number;
parameter CLOCK_FREQ  =100000000; //clock input 100000000,100Mhz
```

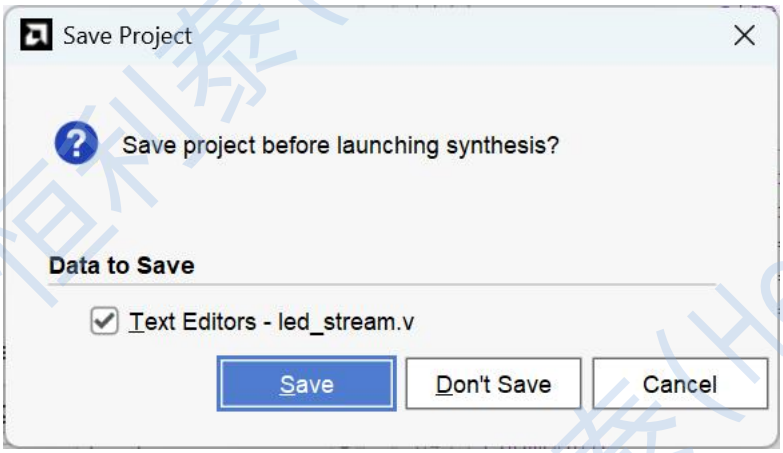
```
parameter COUNTER_MAX_CNT=CLOCK_FREQ/2-1;//change time 0.5s
//counter 1s
    always @(posedge clk, negedge rst_n) begin
        if(!rst_n) begin
            cnt <= 32'd0;
            led_on_number <= 3'd0;
        end
        else begin
            cnt <= cnt + 1'b1;
            if(cnt == COUNTER_MAX_CNT) begin//count to 0.5s
                cnt <= 32'd0;
                if(led_on_number<(LED_NUM-'d1))led_on_number <= led_on_number + 1'b1;
                else led_on_number<='d0;
            end
        end
    end
    //indicate whitch led on
    always @(*) begin
        case(led_on_number)
            0: led <= 3'b001;
            1: led <= 3'b010;
            2: led <= 3'b100;
            default:;
        endcase
    end
endmodule
```

我们将流水灯代码简单的讲解一下：代码中，晶振使用的外部晶振为我们100M频率的晶振,由于是差分晶振，需要使用IBUFDS原语转换为单端时钟clk，代码中我们定义CLOCK_FREQ为100*1000*1000=100000000（100M频率计数一秒钟达到的值），我们让一个计数器cnt 从0计数到100M晶振频率下的半秒钟时间的值改变led输出，所以计数器最大值等于CLOCK_FREQ/2-1=49999999，即让led_on_number自动加1，注意最大值是2，接着清零。也就是说这个数字会在0，1，2三个值变化，时间周期为0.5秒。最终通过代码的case语句来选择led_on_number点亮哪一个led，将3个LED输出信号对应的LED位设置为1，点亮LED。

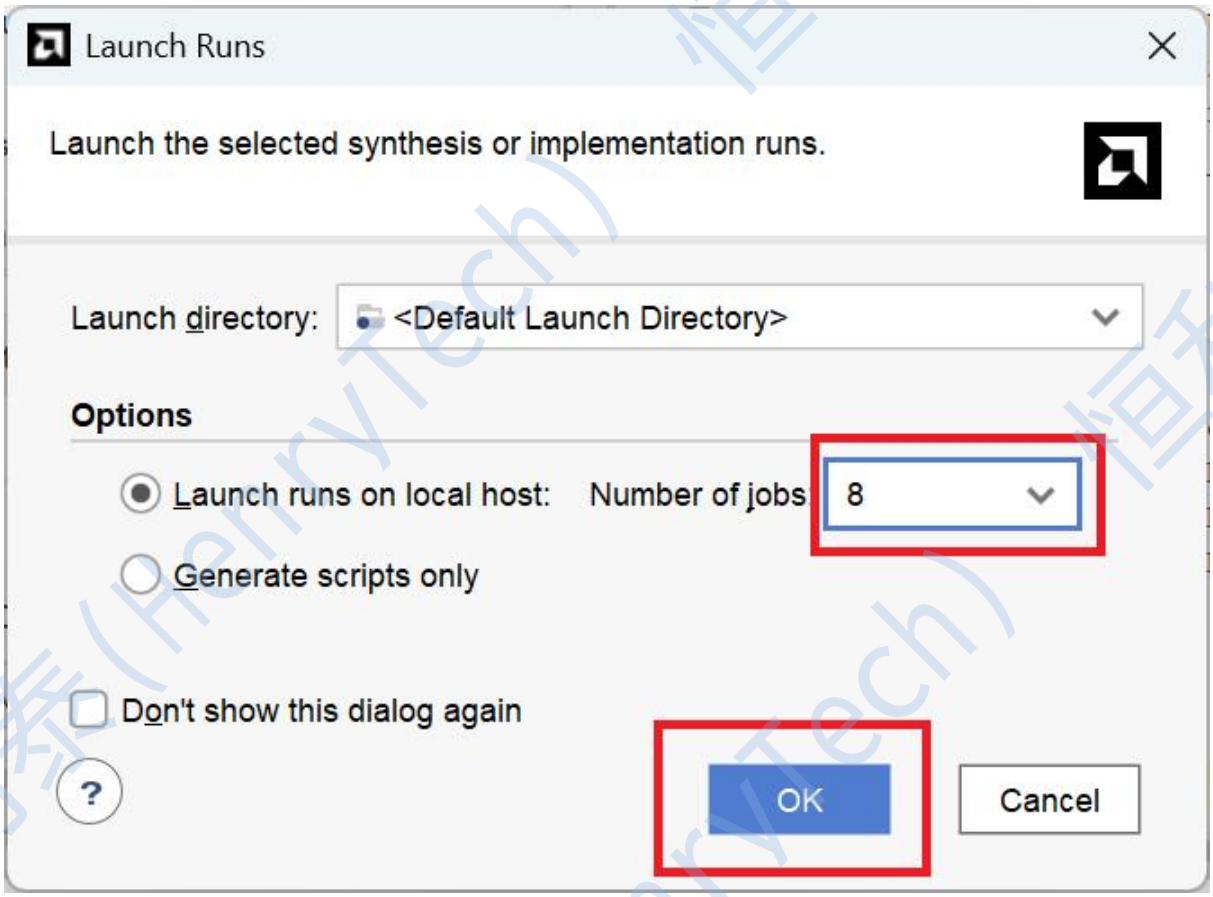
开始综合，点击工程管理栏的 Run Synthesis：



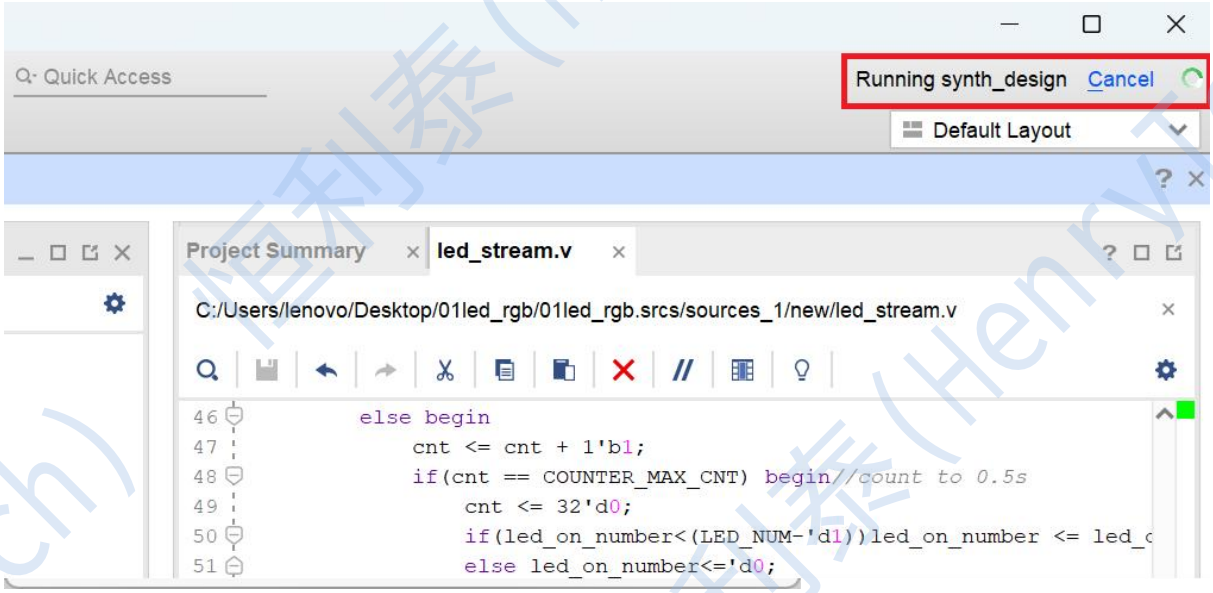
然后弹出的对话框点击 Save：



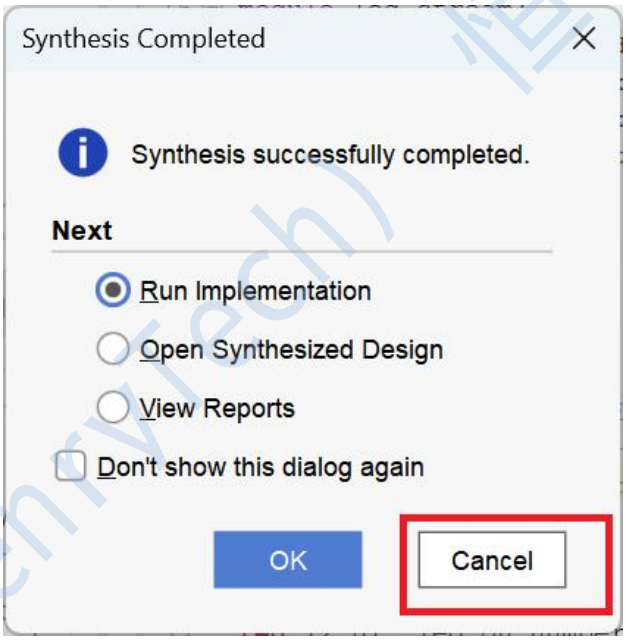
然后弹出的 Launch Runs 对话框根据电脑配置选择 Jobs 数量（代表综合消耗的进线程，VIVADO 目前已验证过支持 32 线程，采用I9-13900处理器，因为它是 20 核 32 线程）， 由于这里电脑配置是 I7-7700K，四核八线程，所以最多可以选择 8 个。综合过程很耗 CPU 性能，所以如果想要综合过程中流畅使用电脑，可以少选两个线程。选完线程数量，点击 OK 进行综合。（综合过程很慢，请耐心等待）



综合状态可以在软件右上角看到已经开始运行。

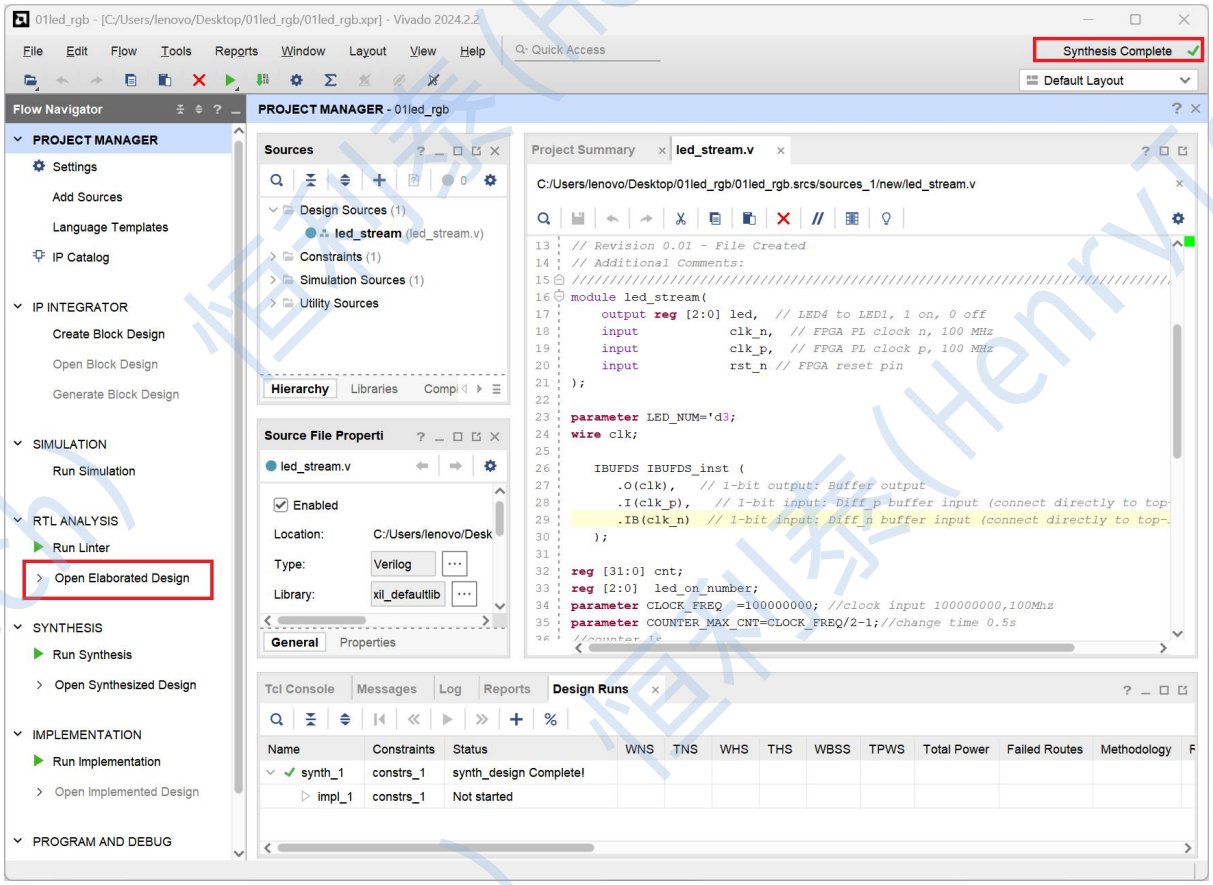


完成综合后，**弹出的综合完成对话框中我们点击 Cancel 取消。不要进行 Implementation。**

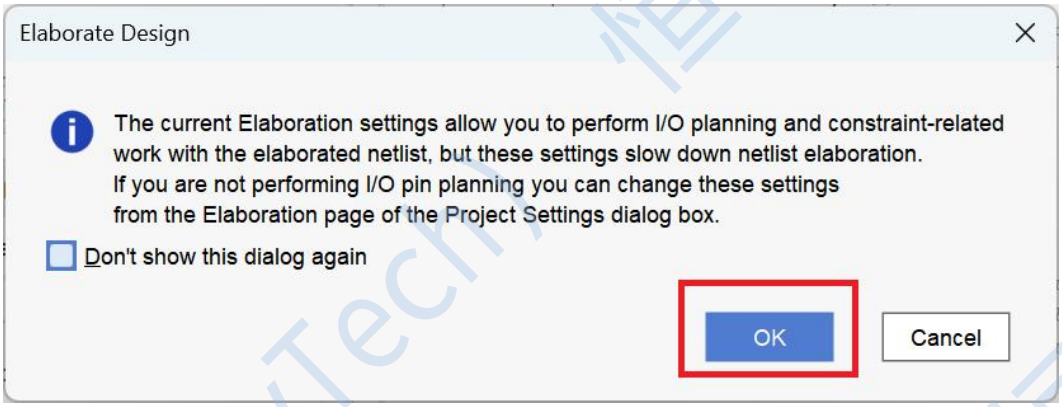


四、绑定引脚约束

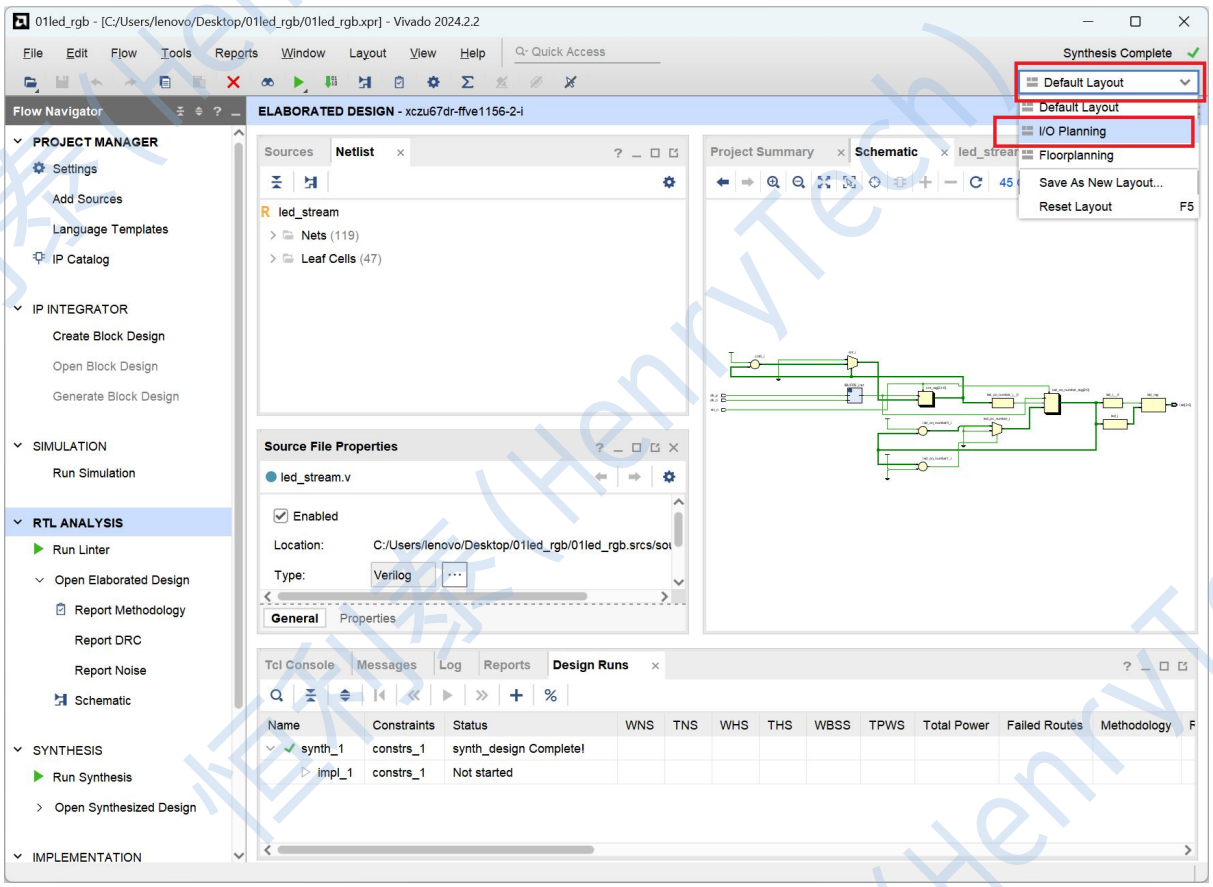
由于我们还没有进行管脚绑定，所以我们需要先绑定管脚在进行 Implementation 实现，最后 布局布线最后生成 BIT 文件。我们接下来进行绑定管脚。可以看到右上角状态已经是综合完成的。我们点击工程管理栏的 RTL ANALYSIS 下面的>open Elaborated Design：



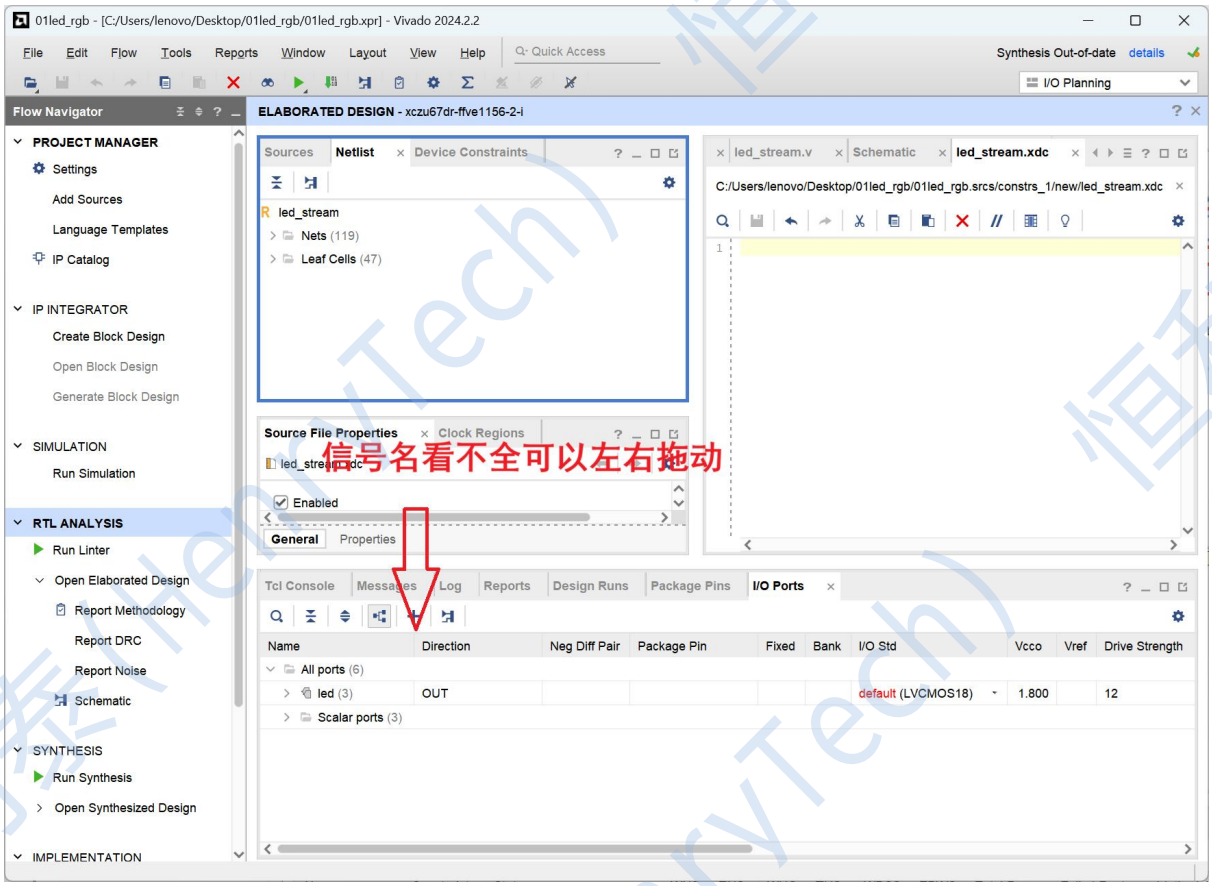
弹出的对话框我们选择 OK，接着等待打开：



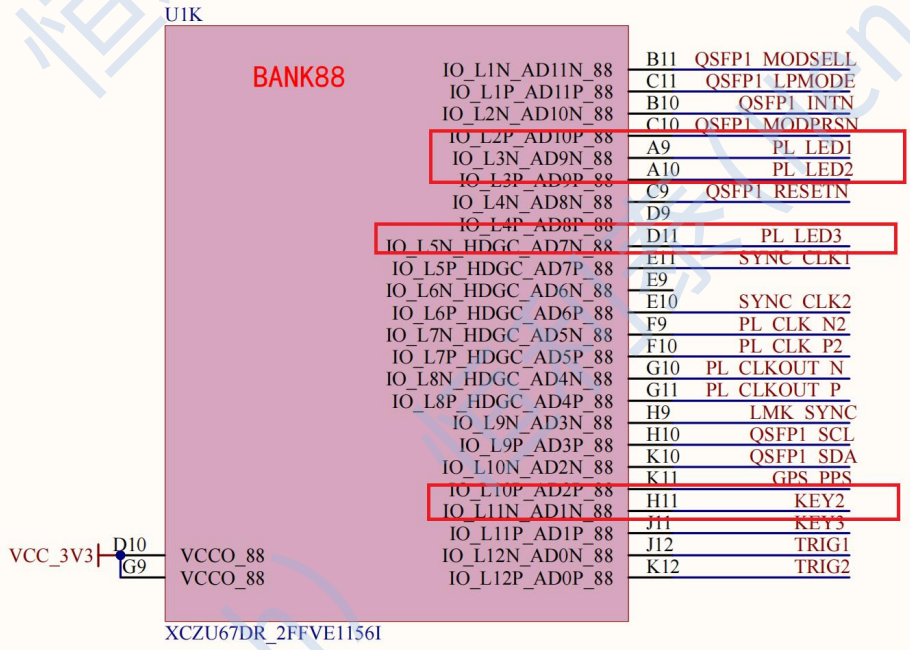
然后右上角，软件运行状态下面的视图选择。我们默认是 Default Layout，但是我们要进行 IO 绑定，我们选择 I/O Planning。

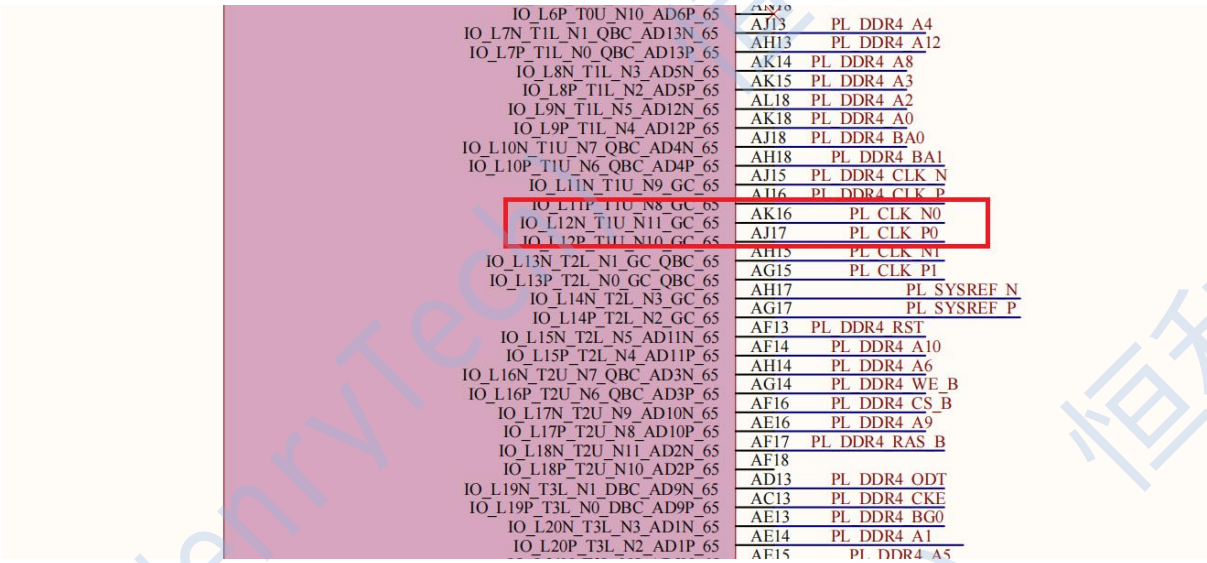


选择 I/O Planning 之后出现如下视图，在下面窗口部分，I/O Ports 即可进行 IO 管脚绑定。 注意，有的时候VIVADO引脚绑定下面的窗口信号名看不清楚，显示不完整，窗口每一列都是可以手动拖动调整的。

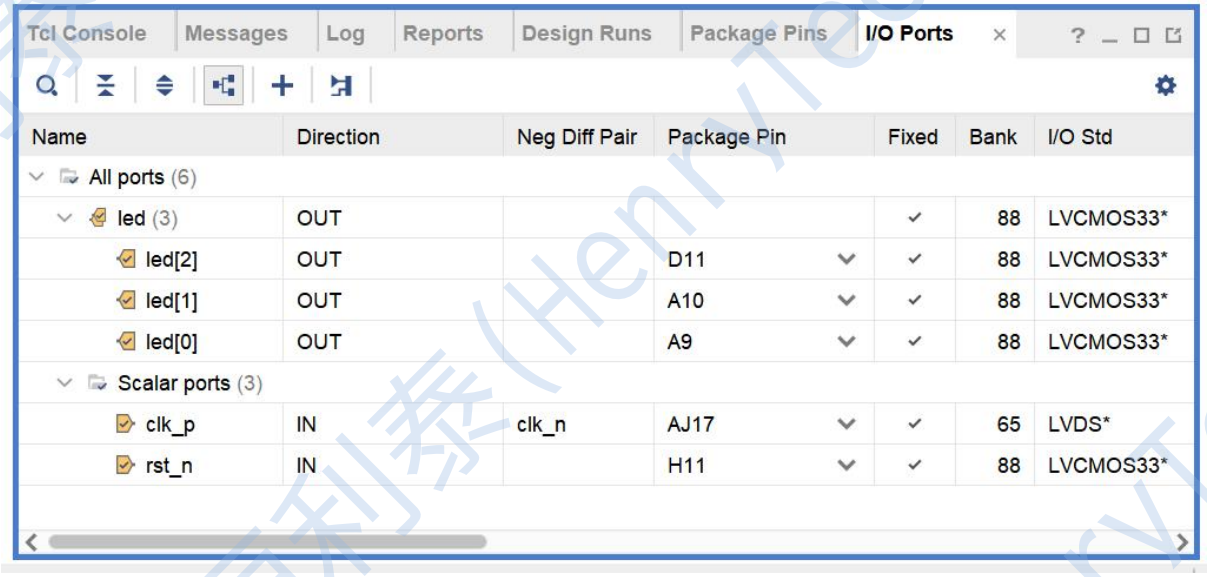


根据原理图绑定的管脚如下所示，引脚大家查阅一下原理图可或等更详尽的引脚信息，按钮复位信号rst_n用的位于原理图第十部分的BANK89上的KEY2按钮引脚H11，电平标准LVCMOS33，LED位于BANK88的B12,A12,A14三个脚分别对应RGB LED的红绿蓝,电平标准也是LVCMOS33，时钟使用原理图的第十三部份的BANK65的AJ17引脚为差分时钟P极性，AK16为N极性，绑定仅仅需要绑定P极性也就是说AJ17,电平标准LVDS：

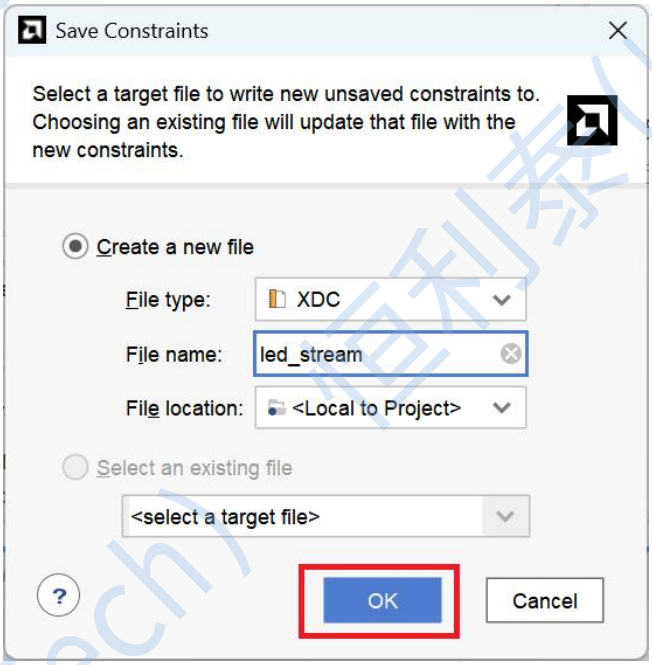




绑定最终如下所示：



按下 ctrl+s 保存，弹出的保存对话框，填入管脚约束文件名。我们这里取为 led_stream：



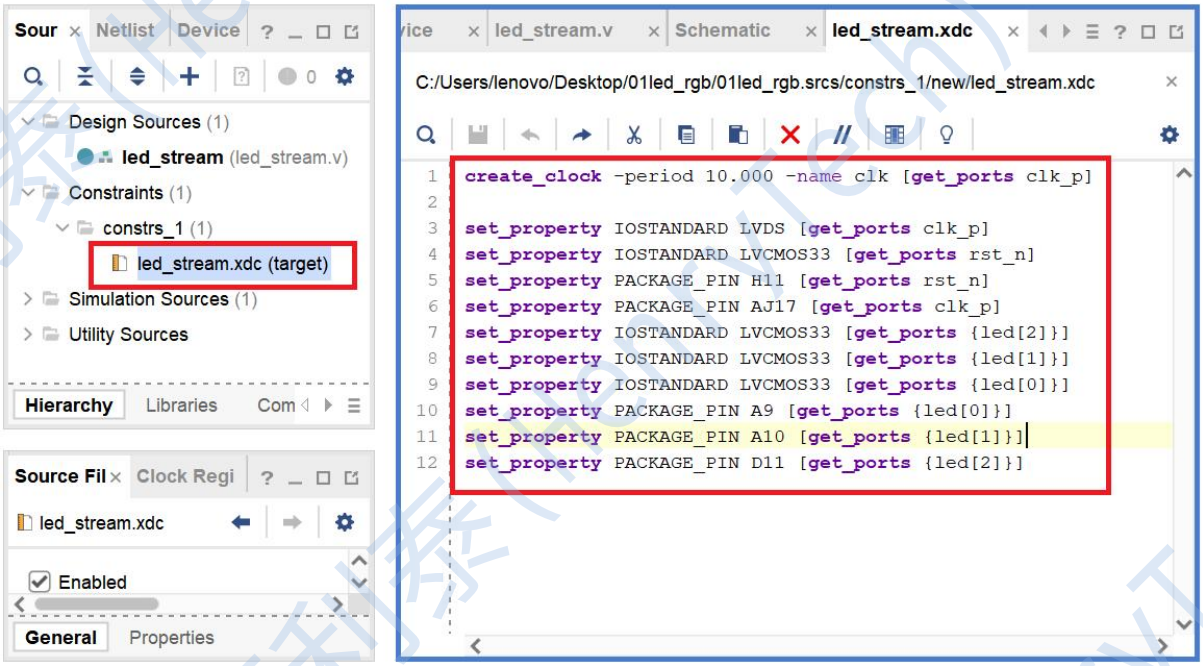
后续绑定引脚，有了约束xdc文件，大家后续项目中也可以自己按照约束文件的格式使用约束语句来绑定引脚。

五、约束

我们打开约束文件看一下，两种基本约束格式：

```
set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
set_property PACKAGE_PIN H11 [get_ports rst_n]
```

第一句是约束引脚编号，第二句是约束电平标准。方括号最后的clk是信号名。



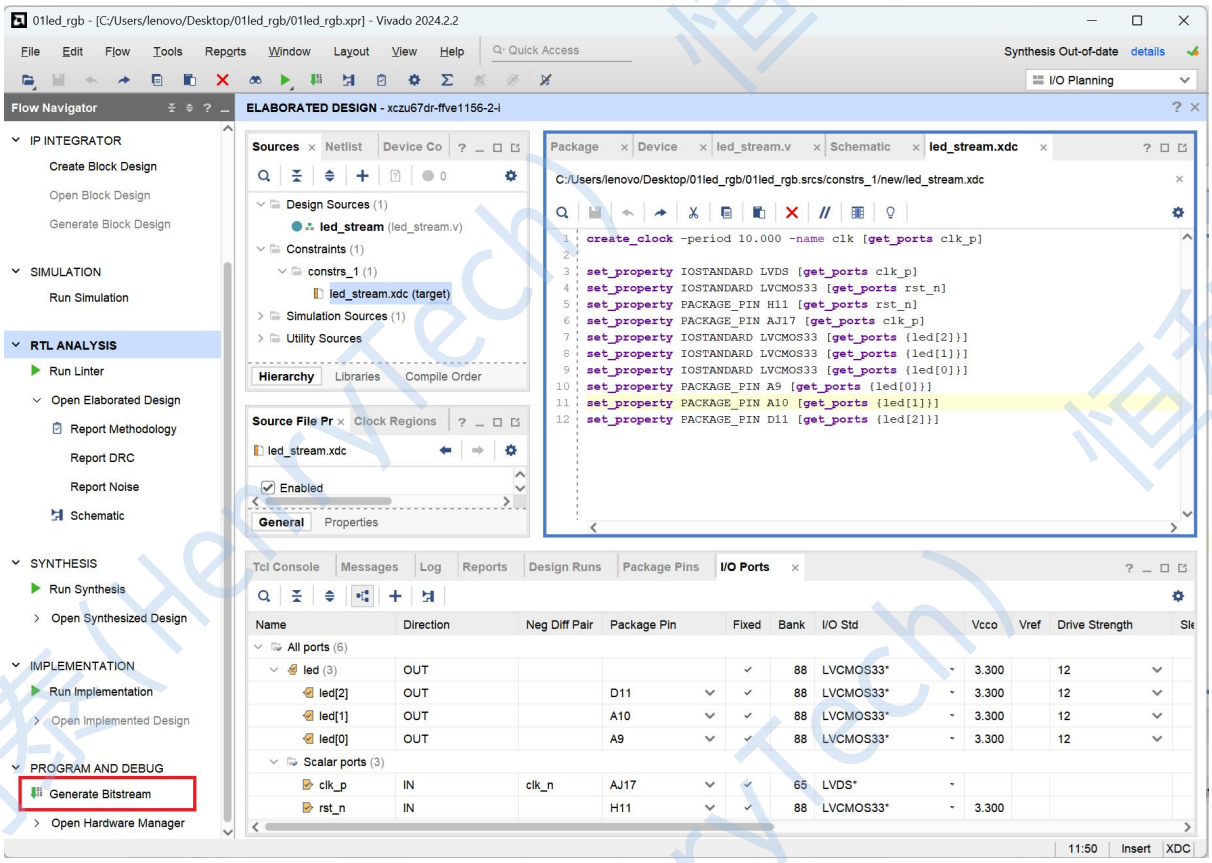
我们约束代码最上方加了一句约束时钟：

```
create_clock -period 10.000 -name clk [get_ports clk_p]
```

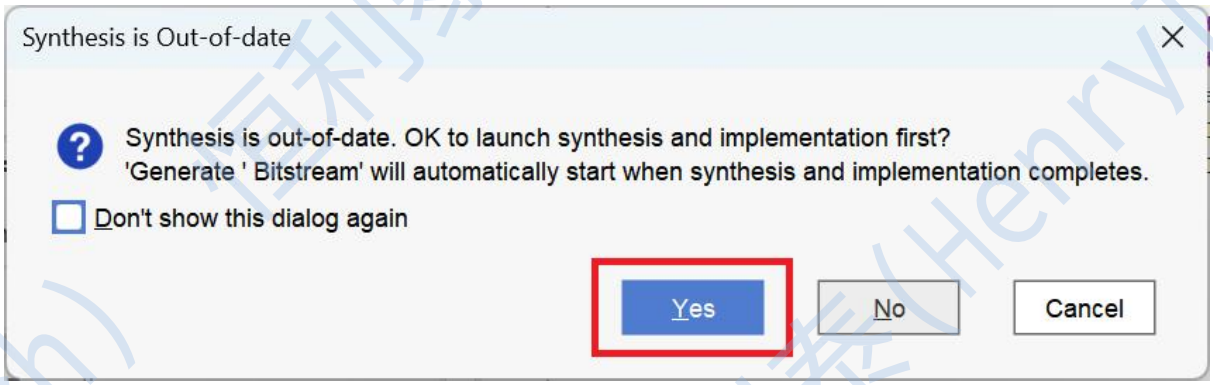
上面一句可以告诉软件，我们代码的时钟是100M的，是引脚端口clk_p信号路径，布局布线需要工作在这个频率。有了这一句约束，我们软件就可以对布局布线的结果进行约束路径，使得我们生成的BIT文件可以满足100M频率的要求。

六、生成BIT上板验证

我们点击工程管理栏的 PROGRAM AND DEBUG 下面的 Generate Bitestream 生成 BIT 文件。（也可以先点击 IMPLEMENTATION 下面的 Run implementation 先进行实现，实现包括布局布线，这里直接省车了，直接生成bit 文件包含了实现操作）。

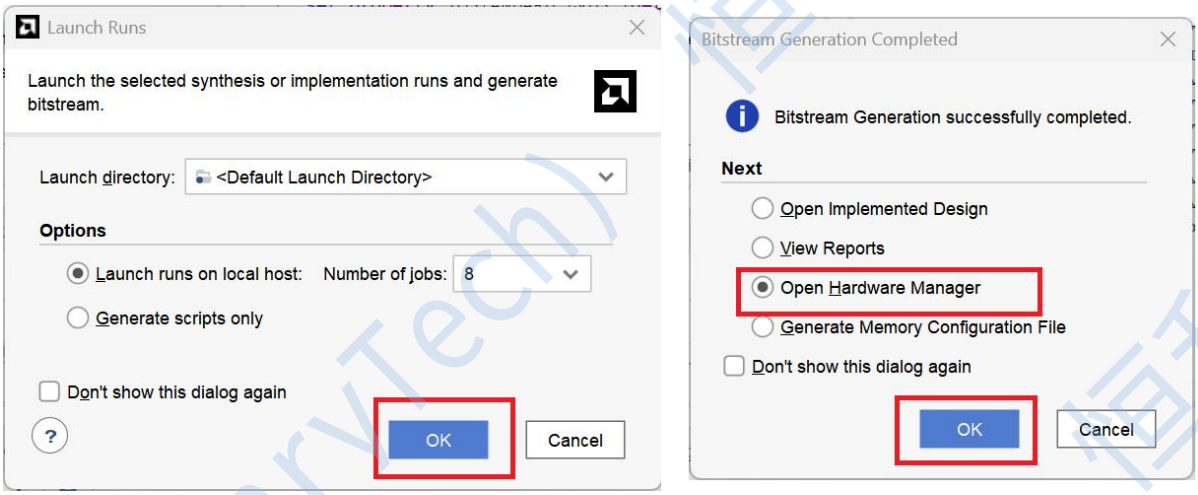


如果弹出的对话框为综合信息过期，这里 默认点击 OK（由于软件综合后才能有模块的顶层引脚信息，才能进行绑定管脚。我们绑定 管脚之后，软件认为工程信息发生了变更所以需要再次综合）。 如果没有弹出，就不用理会：

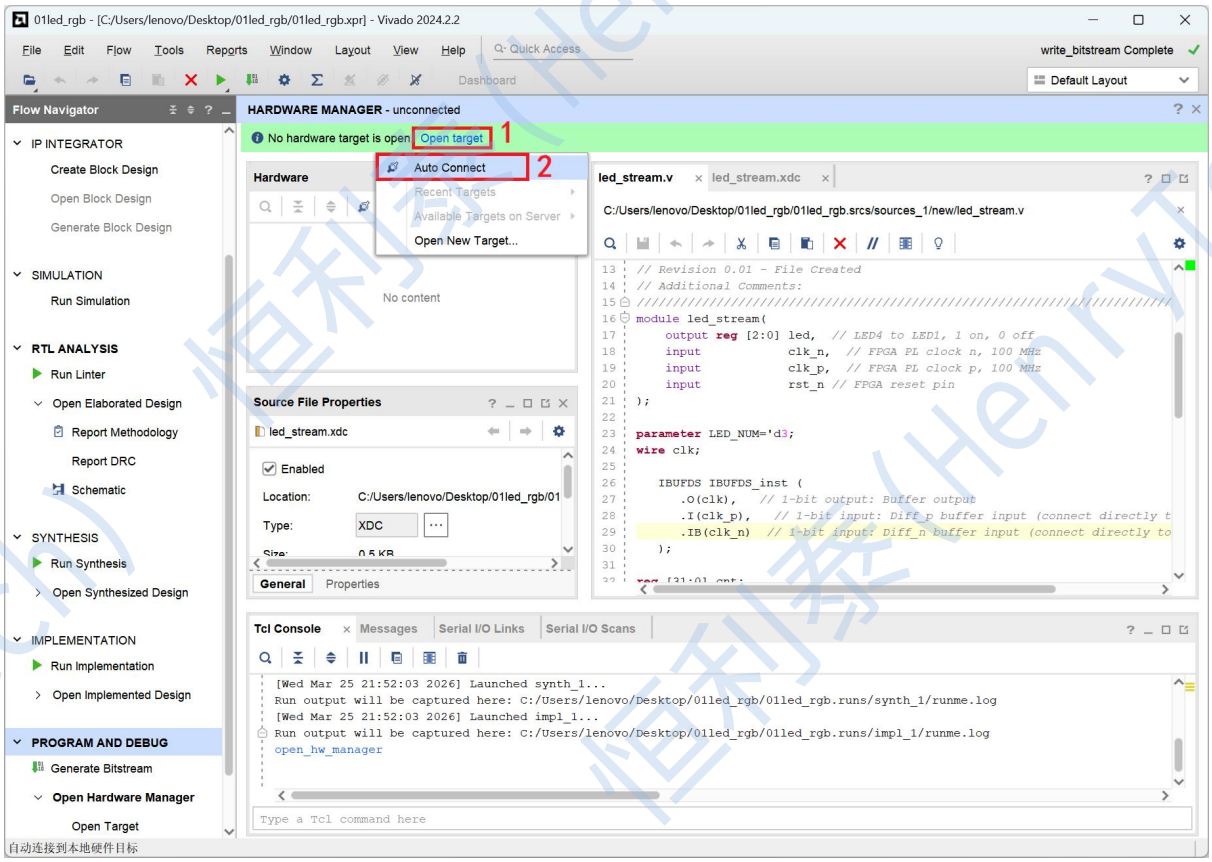


接着还会弹出综合线程数量对话框。我们也选择默认（这里使用的电脑最大是 8），然后点 击 OK。然后软件开始了生成比特流的一系列工作。由于我们约束文件创建了，所以绑定完引脚软件可能还要进行一次综合操作。

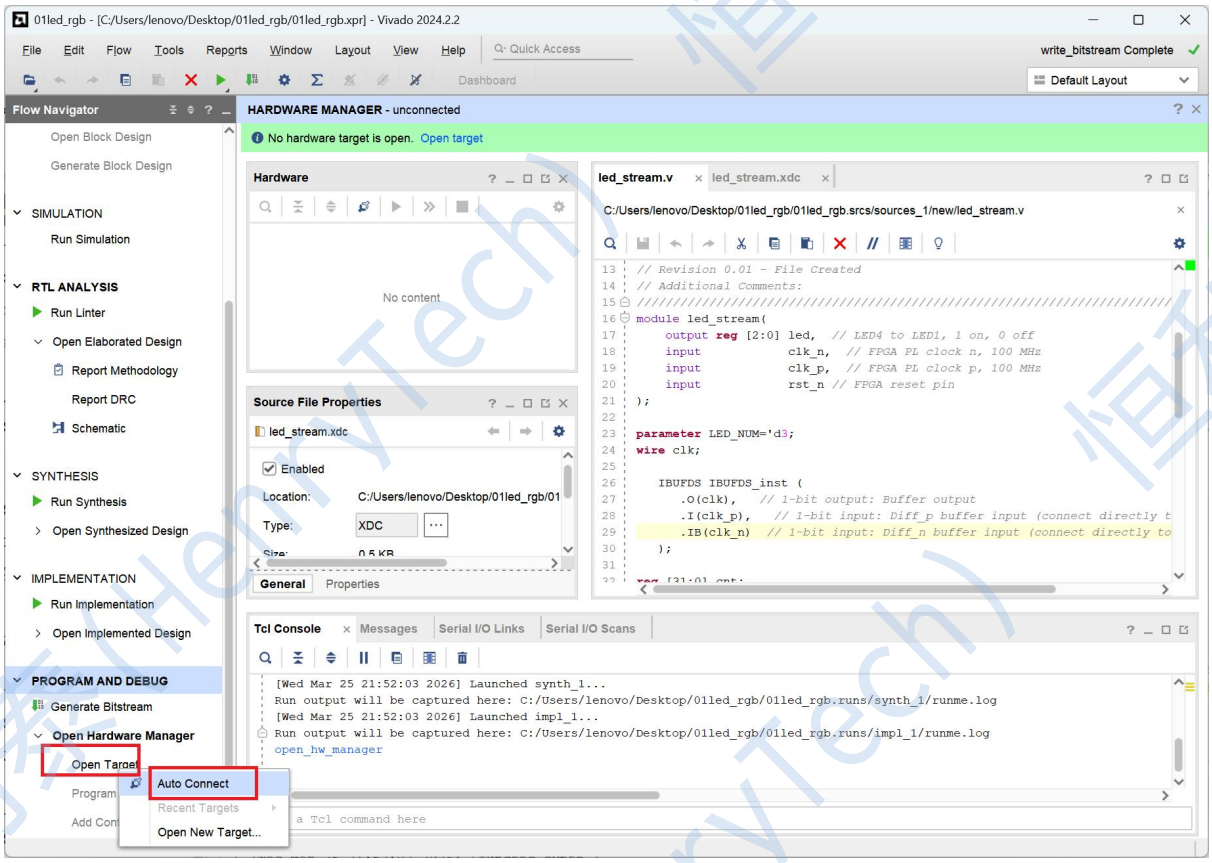
总流程是：综合->实现（布局布线）->生成 bit 文件。我们等待软件运行这些操作结束，结束后弹出如下完成BIT文件生成的对话框，这里我们默认打开硬件管理器 Open Hardware Mnager：



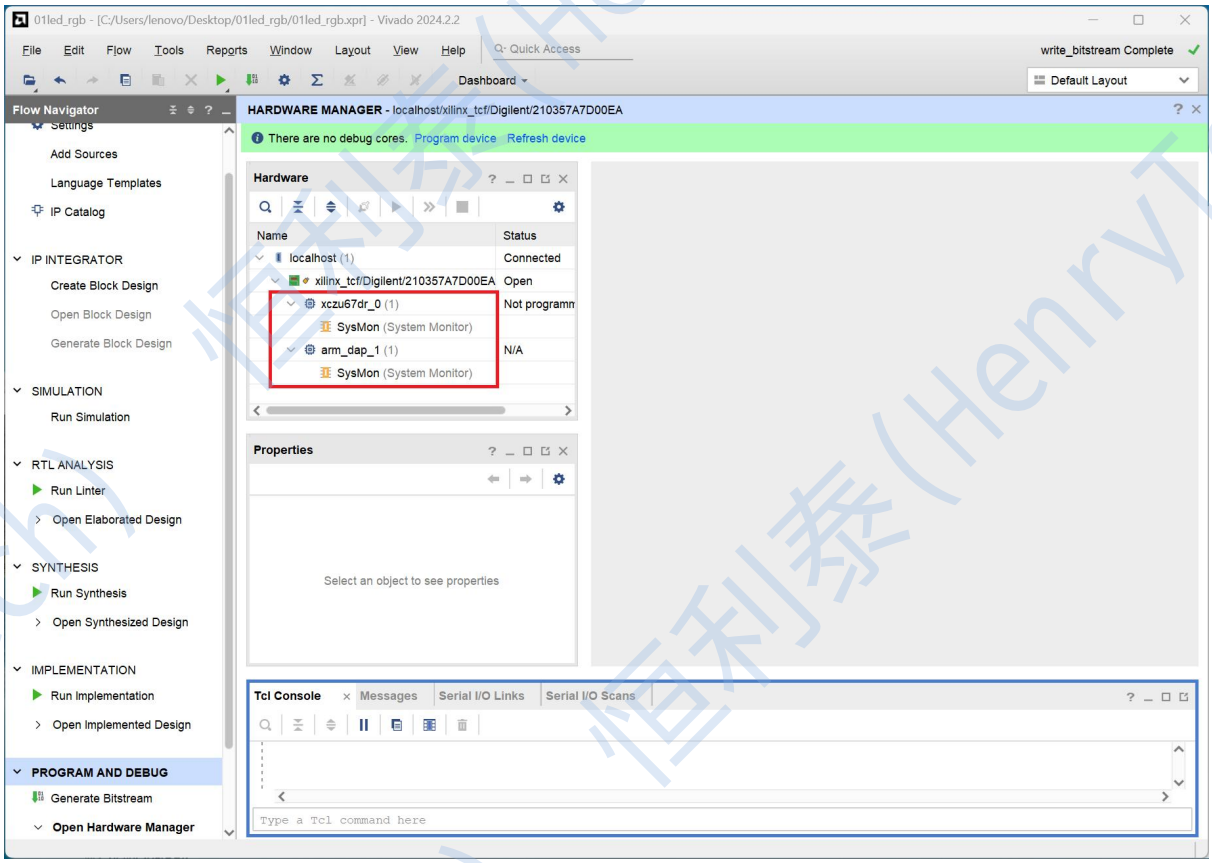
到这一步了，我们连接电源到开发板，以及DEBUG接口到电脑，设置好BOOT开关为JTAG（BOOT设置请参考《02RF67DR-Lite快速测试使用指南》中的启动设置部分）长按POWER按钮开机上电（大约2-3秒松手就启动了），接着我们可以使用VIVADO连接板子，我们上面的对话框默认选择的使Open Hardware Manager，我们对话框下方点击OK之后，出来的界面点击上方的 Open Target，然后弹出的下拉界面选择点击 Auto Connect，自动连接目标开发板。点击之后等待连接完成。注意，中途弹出任何网络访问的对话框一定要都勾选允许软件访问，否则可能无法使用下载器，并且可能会导致重装软件才能解决。这一点很关键！！



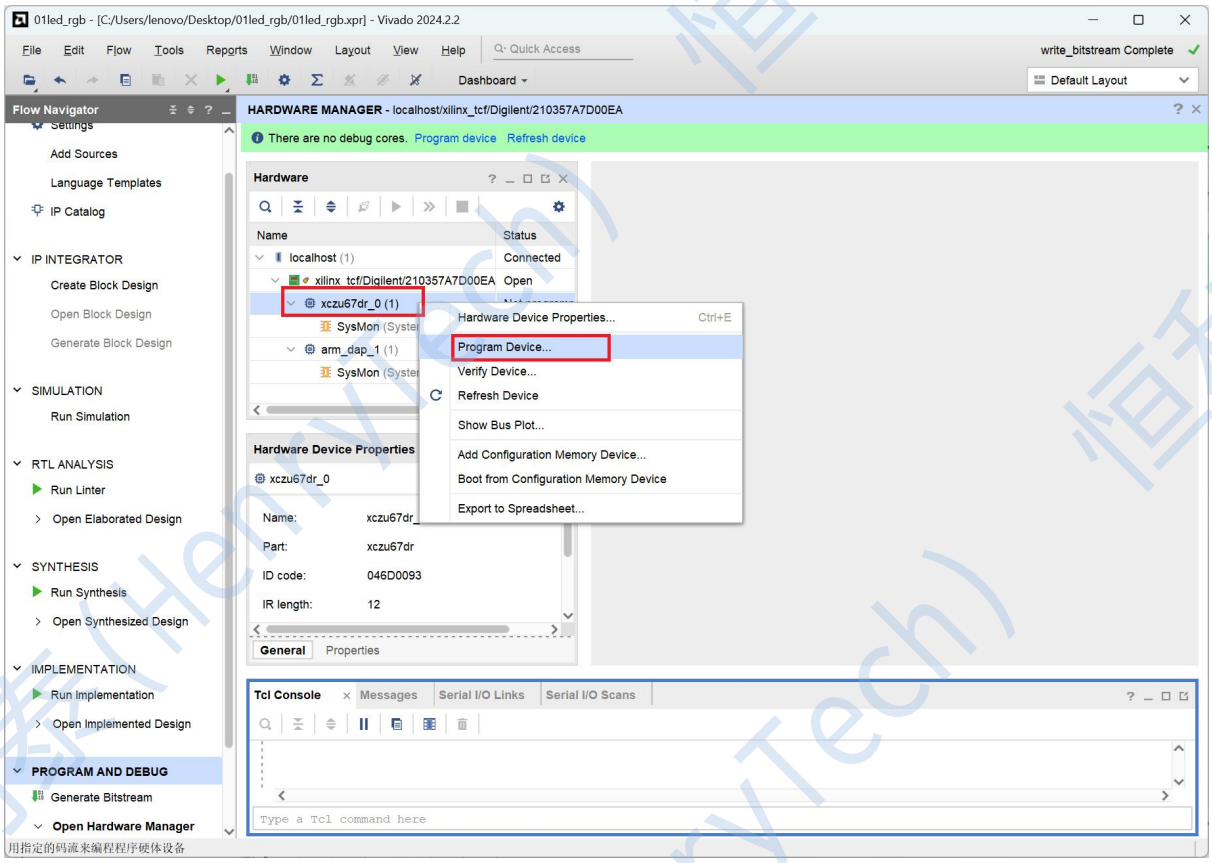
如果BIT文件生成后弹出的对话框不小心右边点取消Cancel，我们可以在软件左下角找到Open Hardware Manager下拉点击Open target右边点击Auto Connect打开：



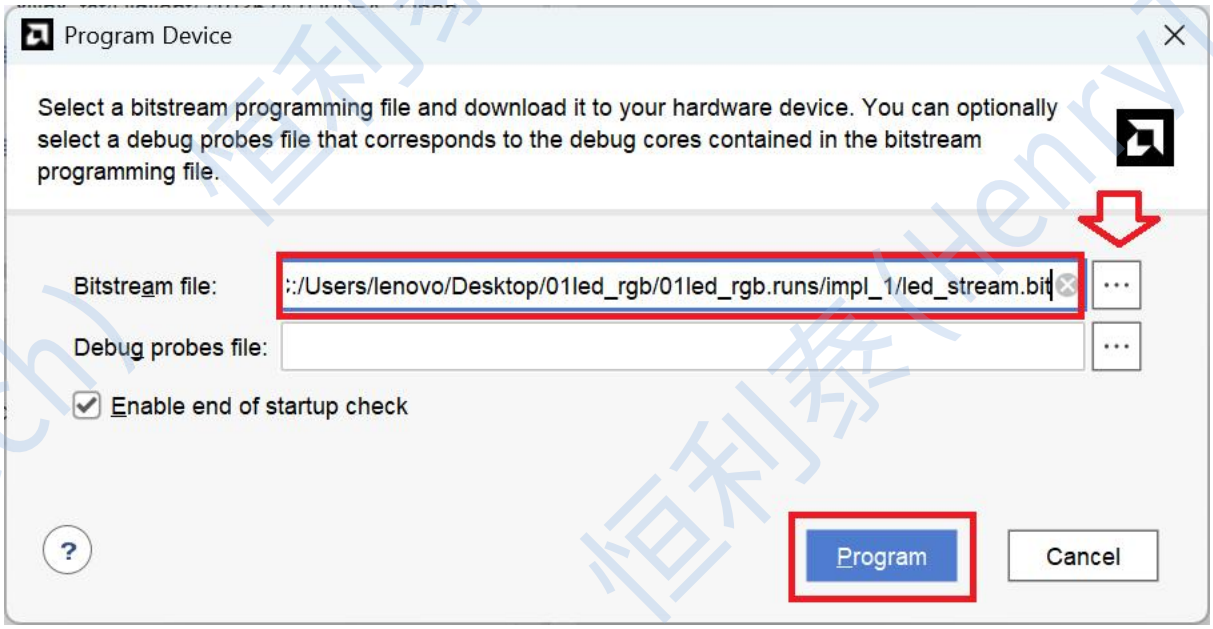
连接完成之后，可以看到 Hardware 窗口出现了我们的XCZU47DR芯片设备，说明已经连接并且识别到了芯片：



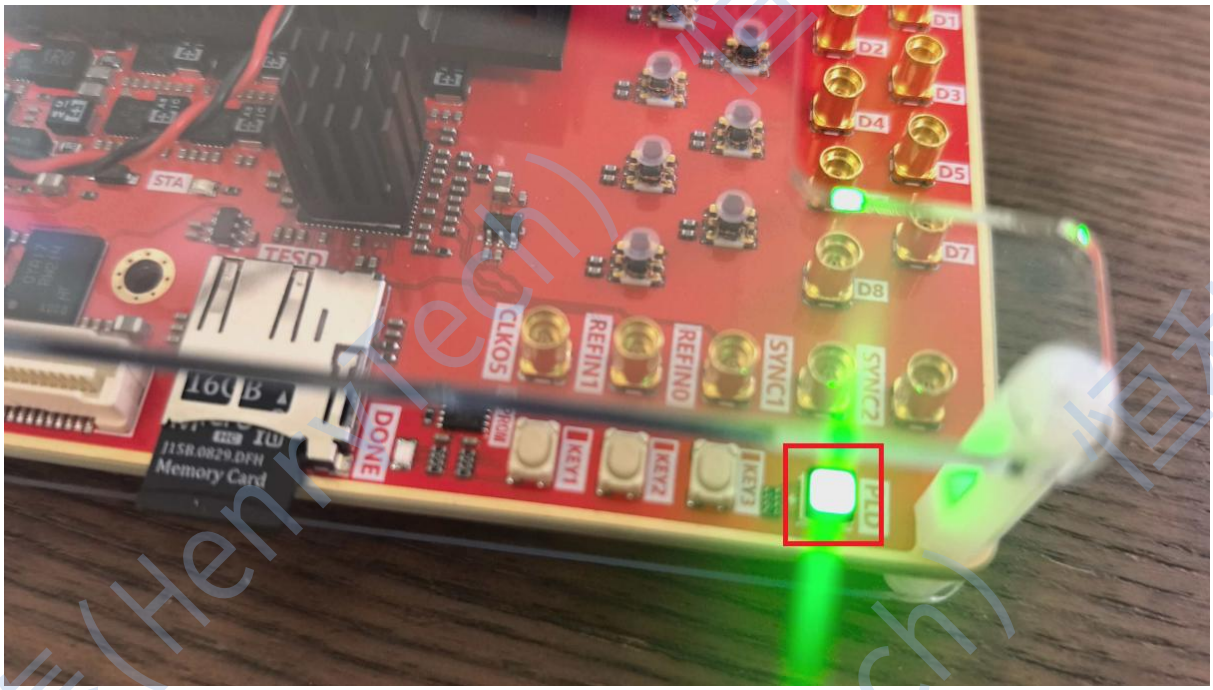
们鼠标在xczu47dr右键点击 Program Device 进行烧写配置（断电就丢失）：



弹出的烧写对话框，，其中 Bitstream file 表示选择的 bit 文件，也就是上面我们生成的 bit 文件。这里要检查一下路径，是否是我们本工程的。有时候不是本工程的，烧写找不到文件,烧写会报错，这种情况点击BITstream file右边的路径按钮（...），找到我们本工程路径下面的：/01led_rgb.runs/impl_1/led_stream.bit，然后点击Program开始烧写BIT：



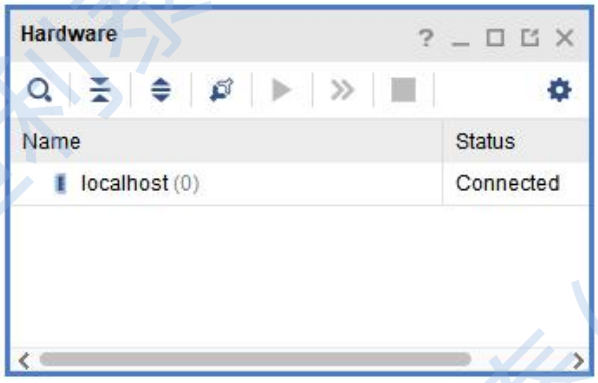
烧写完成。我们可以看到开发板的几个按钮旁边的那个RGB灯三种颜色轮流显示。



我们可以按下KEY2复位。

七、常见问题

1、烧写失败，或者找不到芯片或者下载器：如果硬件管理器打开，只有localhost，可能是开发板没有上电连接电脑。如果连localhost都找不到，那就参阅软件安装的教程，最后提到要允许网络访问，如果已经禁止网络访问，那么你可能要重新安装软件。



2、RGB灯没有按照预期变化：请检查约束引脚，核对原理图。

VIVADO仿真的使用之呼吸灯

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 VIVADO工程建立 3
三、 呼吸灯的实现代码 3
四、 新建 verilog 的testbench文件 7
五、 验证效果 14
六、 常见问题 14

恒利泰(HenryTech)

一、文档实现功能介绍

本文档实现对设计代码进行仿真，使用VIVADO自身的工具进行仿真，通过本教程学习如何使用VIVADO仿真功能。 关于仿真，很多小伙伴一般都知道使用modelsim等工具，但是VIVADO自带的仿真可以完成所有的仿真工作和验证。工程新建方法请参考文档《04VIVADO新建工程流程和软件介绍》。

特殊说明：由于VIVADO使用2024.2版本，可能显示界面等有所区别，但是开发过程一致。素材图片未做更新，沿用47DR的版本VIVADO2022.2的截图。

二、VIVADO工程建立

起始页（或 file->Project->New）	创建新工程（Create New Project）
向导起始页面	点击 Next->
Project Name（工程名）	工程名：02breath_led 工程路径：（自己选择，不要有中文路径） 勾选 Create Project Subdirectory，点击 Next->
Add Source（添加设计源文件）	点击 Next->
Add Exsixting IP（添加已有的 IP）	点击 Next->
Add Constraints（添加已有约束文件）	点击 Next->
Default Part（默认配置，芯片选型）	Family->Zynq UltraScale_RFSoCs Package->FFVE1156 Speed->-2 按具体型号选择 XCZU67DR-FFVE1156-2-i
New Project Summary（新建工程概况）	确认工程信息，选型等，点击 Finish 完成

三、呼吸灯的实现代码

新建verilog文档方法请参考《05最简单的led流水灯》。这里不再重新进行详细说明，我们新建的文档名：breath_led.v，模块名为breath_led，模块实现RGB LED三个颜色的轮流呼吸灯效果。breath_led.v代码如下（代码不要从这里拷贝，请从对应的demo拷贝，pdf文档拷贝出来格式会发生变化）：

```
//////////////////////////////////////////////////////////////////////////////////
// Company: bochenjingxin
// Engineer: yang
```

```
// Create Date: 2024/05/16 12:35:02
// Design Name:
// Module Name: breath_led
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////
module breath_led #
    (
        parameter    CLOCK_FRQ=100000000,时钟频率
        parameter    PWM_FRQ=1000,PWM频率
        parameter    BREATH_ERIOD=2,呼吸周期
        parameter    SET_COMPARE_FRQ=1000,设置比较的频率，一秒钟比较多少次
        parameter    PWM_COUNTER_MAX=CLOCK_FRQ/PWM_FRQ,PWM计数器最大值
        parameter        BREATH_COUNTER_MAX=CLOCK_FRQ,呼吸计数器最大值
        parameter        SET_COMPARE_COUNTER_MAX=CLOCK_FRQ/SET_COMPARE_FRQ,设置比较计数器最大值
        parameter  COMPARE_VALUE_STEP=PWM_COUNTER_MAX/SET_COMPARE_FRQ//比较值的增加步长
    )
    (
        input wire clk_p,clk p 100m
        input wire clk_n,clk n 100m
        input wire rst_n,KEY2
        output wire [2:0]led //led
    );

//parameter reg define
reg [31:0]counter_pwm;
reg [31:0]counter_breath;
reg [31:0]counter_compare;
reg [31:0]compare_value;
reg pwm_period_clk_view;
reg breath_period_clk_view;
reg compare_period_clk_view;
reg [4:0]led_number;

reg led_breath_view;PWM最终输出的寄存器
reg breath_dir;呼吸方向，向上还是向下
reg [2:0]led_reg;
assign  led_clk_pwm_period=pwm_period_clk_view;
```

```
assign  led_clk_breath_period=breath_period_clk_view;
assign  led_clk_compare_period=compare_period_clk_view;
assign     led=led_reg;
assign   pll_reset=~rst_n;

wire clk;

   IBUFDS #(
      .CCIO_EN_M("TRUE"),
      .CCIO_EN_S("TRUE")
   )
   IBUFDS_inst (
      .O(clk),   // 1-bit output: Buffer output
      .I(clk_p),   // 1-bit input: Diff_p buffer input (connect directly to top-level port)
      .IB(clk_n)  // 1-bit input: Diff_n buffer input (connect directly to top-level port)
   );

//pwm generation
always @(posedge clk or negedge rst_n)
begin
           if(rst_n==0)
           begin
                        counter_pwm<=0;
                        pwm_period_clk_view<=0;
           end
           else
           begin
                        counter_pwm<=counter_pwm+1;
                        if(counter_pwm<compare_value)led_breath_view<=1;
                        else led_breath_view<=0;
                        if(counter_pwm>PWM_COUNTER_MAX-1)
                        begin
                                     counter_pwm<=0;
                                     pwm_period_clk_view<=~pwm_period_clk_view;
                        end
           end
end

reg [4:0]led_number_state;led编号
reg breath_dir_pre_reg;
reg breath_dir_now_reg;
//shift led breath direction
always @(posedge clk or negedge rst_n)
begin
```

```
        if(rst_n==0)
                begin
                        led_number='d0;
                        counter_breath<=0;
                        breath_period_clk_view<=0;
                        breath_dir<=0;
                        breath_dir_pre_reg<='b0;
                        breath_dir_now_reg<='b0;
                        led_number_state<=0;
                end
        else
        begin
                counter_breath<=counter_breath+1;
                if(counter_breath>BREATH_COUNTER_MAX-1)begin
                        counter_breath<=0;
                        breath_period_clk_view<=~breath_period_clk_view;
                        breath_dir<=~breath_dir;
                end
                breath_dir_pre_reg<=breath_dir_now_reg;
                breath_dir_now_reg<=breath_dir;
                if((breath_dir_pre_reg==1'b1)&(breath_dir_now_reg==1'b0))begin
         case (led_number_state)
         0:      begin      led_number_state=1;led_number=1;   end
         1:      begin      led_number_state=2;led_number=2;   end
         2:      begin      led_number_state=3;led_number=1;   end
         3:      begin      led_number_state=0;led_number=0;   end
         default:begin led_number_state=0;led_number=0;  end
         endcase
            end
        end
end
//breath compare
always @(posedge clk or negedge rst_n)
begin
        if(rst_n==0)
                begin
                        counter_compare<=0;
                        compare_period_clk_view<=0;
                        compare_value<=0;
                end
        else
        begin
                counter_compare<=counter_compare+1;
                if(counter_compare>SET_COMPARE_COUNTER_MAX-1)
```

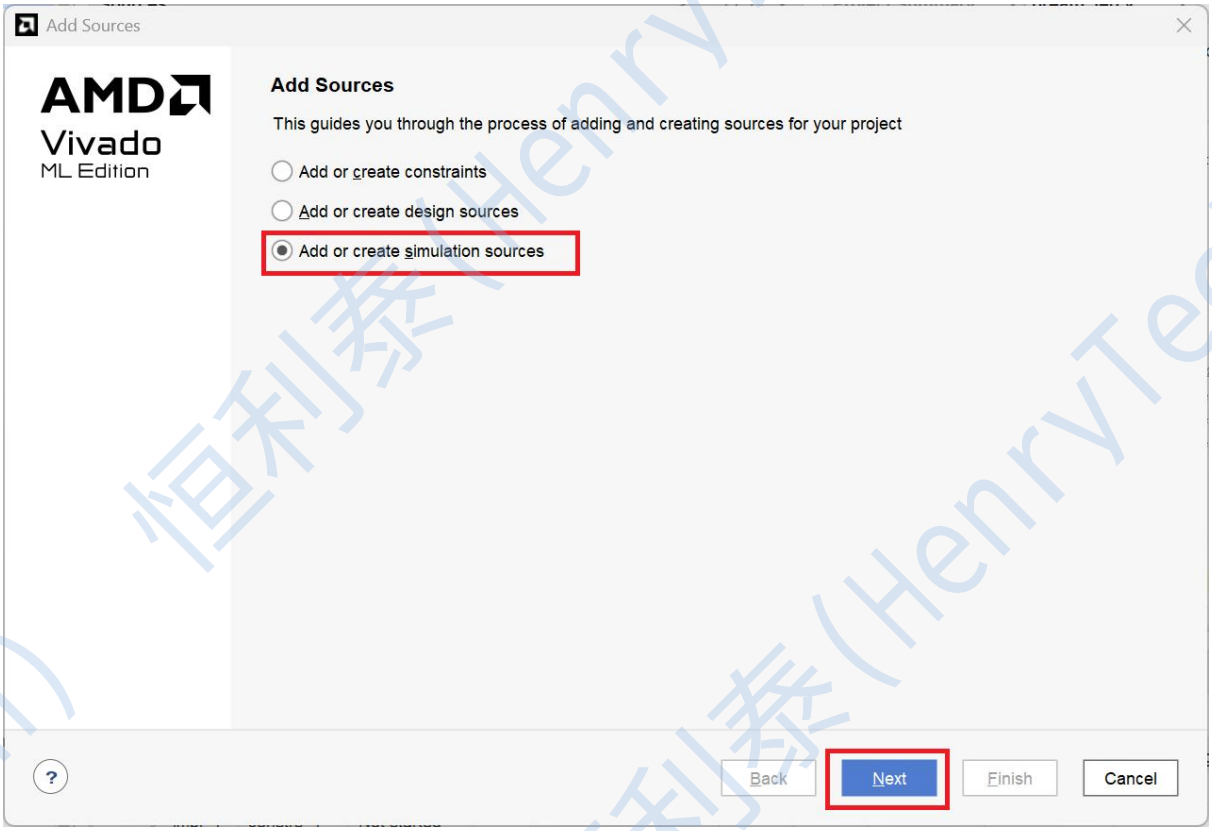
```
                    begin
                                counter_compare<=0;
                                if(breath_dir==0)begin

            if(compare_value<PWM_COUNTER_MAX)compare_value<=compare_value+COMPARE_VALUE_STEP;
                                end
                                else if(breath_dir==1)begin
                                            if(compare_value>0)compare_value<=compare_value-COMPARE_VALUE_STEP;
                                end
                                compare_period_clk_view<=~compare_period_clk_view;
                    end
            end
end
// led output
always @(posedge clk)
begin
            if(rst_n==0)led_reg<=0;
            else begin
        case (led_number)//PWM输出到对应通道的LED引脚
        'd0:            led_reg[0]<=led_breath_view;
        'd1:            led_reg[1]<=led_breath_view;
        'd2:            led_reg[2]<=led_breath_view;
        default:led_reg<='d0;
        endcase
            end
end
endmodule
```

代码中，有呼吸PWM比较输出模块，有PWM生成计数模块，以及最终PWM对应输出到哪个LED的译码模块。代码我们不做过多讲解，有兴趣的小伙伴可以通过接下来的仿真研究代码的呼吸灯实现逻辑。代码新建完先综合一下。

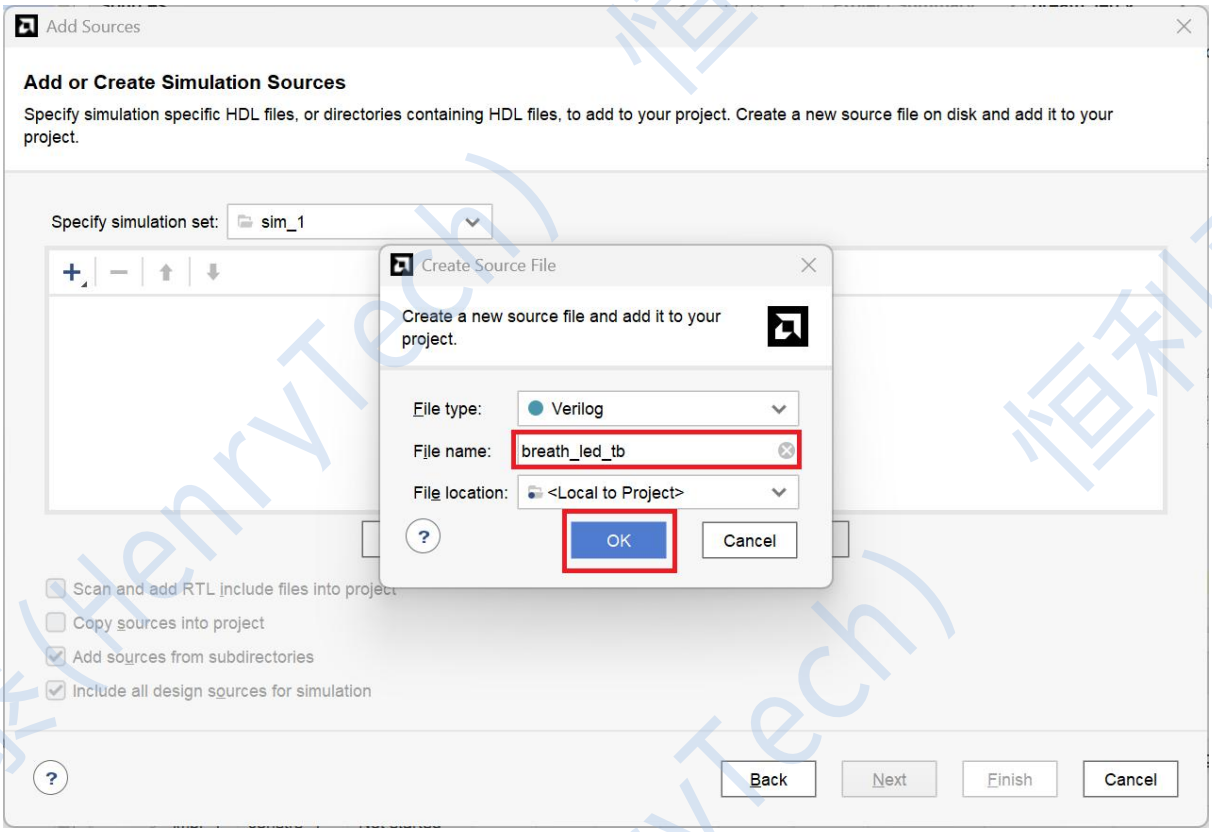
四、新建 verilog 的testbench文件

我们可以看到，新建了verilog文件breath_led.v之后，Sources文件栏下的simulation sources->sim_1下也出现了breath_led.v文件。新建的设计文件也会被同步到仿真文件目录。

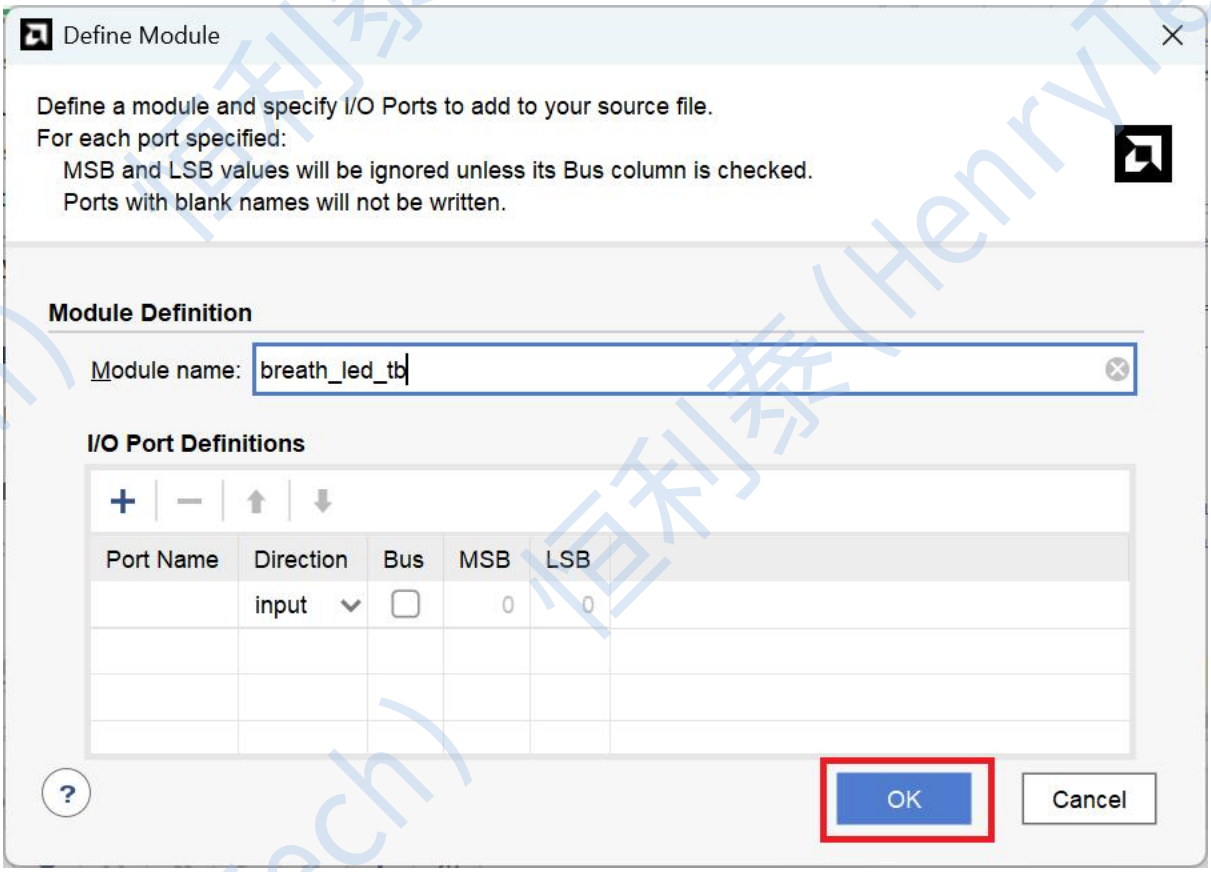
我们在simulation soruces右键Add Sources，弹出添加源文件界面，默认勾选Add or Create simulation sources然后点击next->



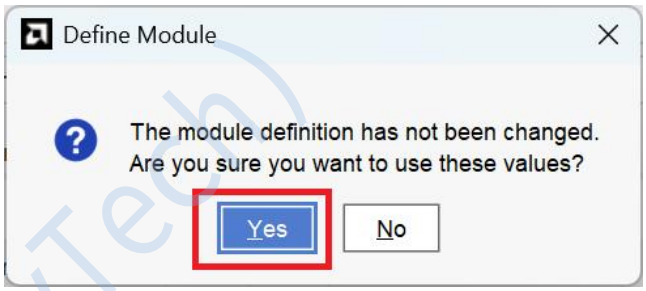
然后点击Create File弹出新建文件窗口，我们这里默认File Type和File location不变，然后File name填入breath_led_tb，然后点击OK：



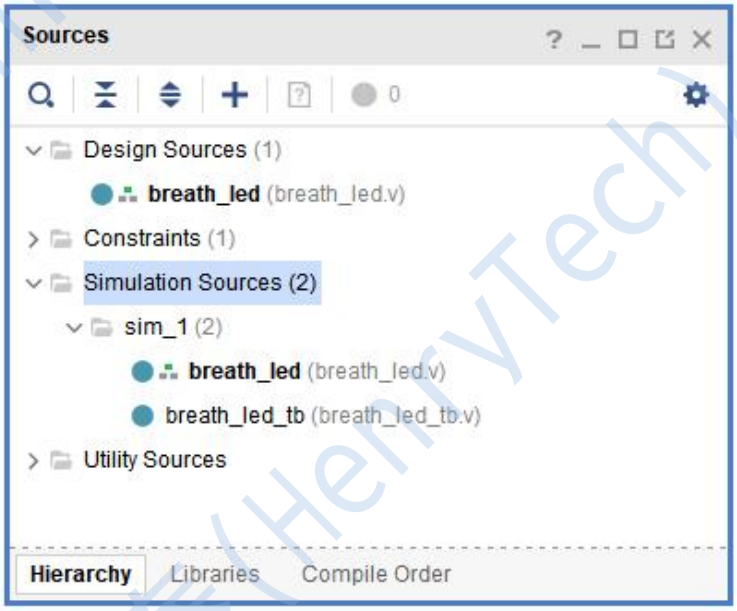
然后点击Finish完成testbench文件添加。最后弹出定义testbench顶层模块名窗口，这里我们默认不修改，然后点击OK：



弹出的对话框点击YES：



最后我们看到sim_1下面我们新建的breath_led_tb.v新建好了。

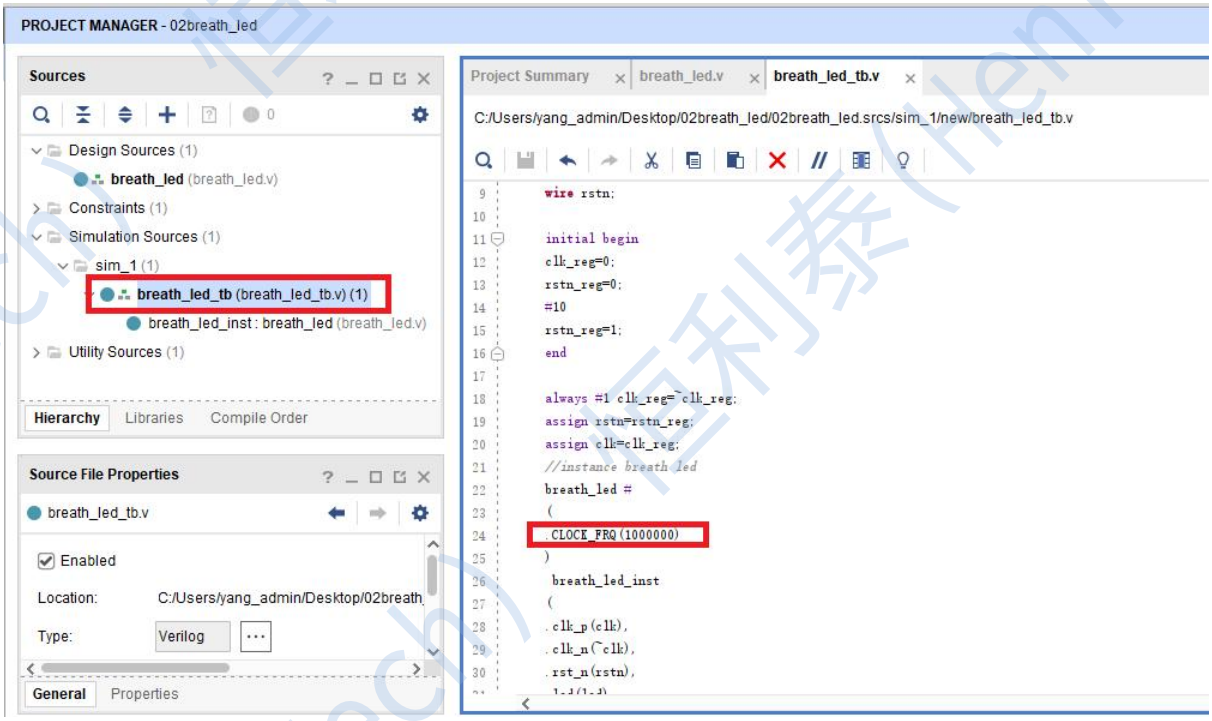


我们双击breath_led_tb.v进行编辑。我们先将要进行测试的breath_led模块例化进来，然后编写一系列的信号初始化，和时钟驱动代码。最终，breath_led_tb.v文件代码如下：（再次重申代码不要从这里拷贝，请从例程工程中拷贝）

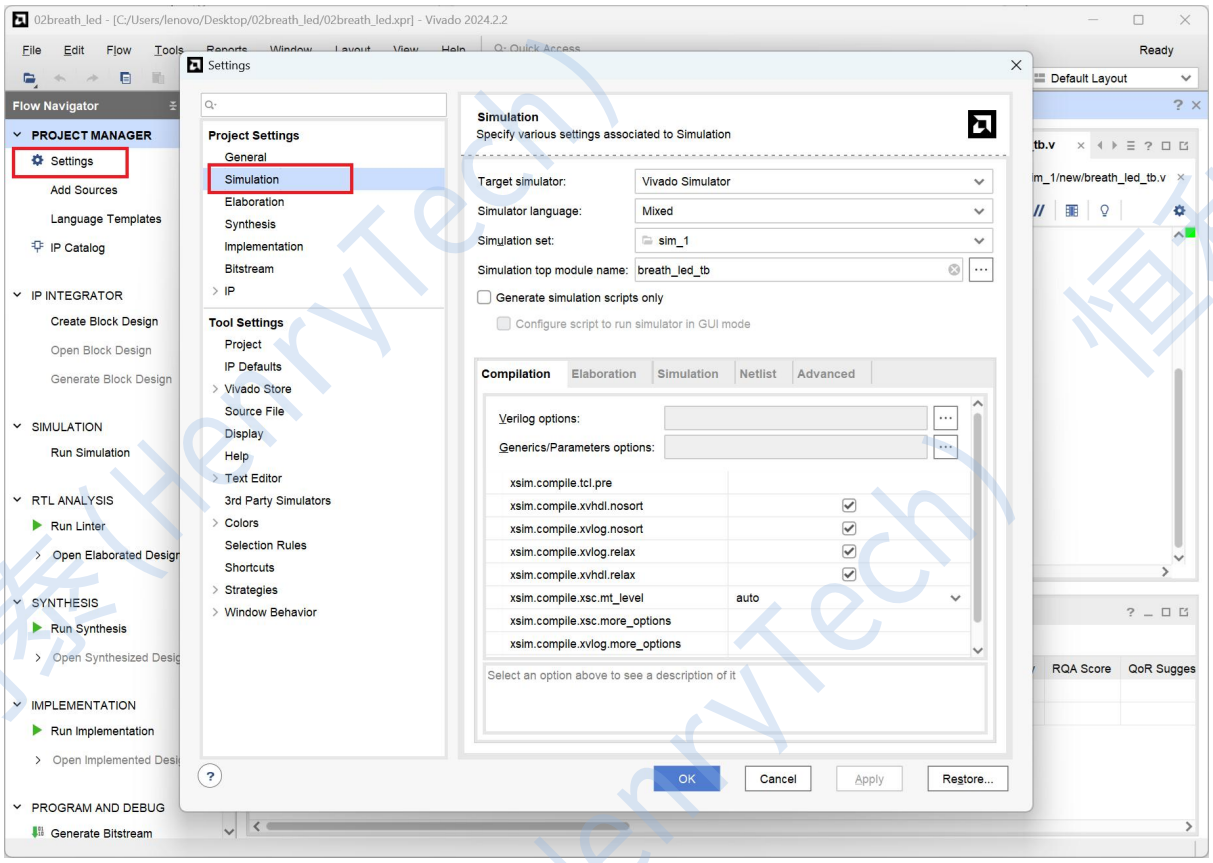
```
`timescale 1ns / 1ns
module breath_led_tb();

    reg clk_reg;
    reg rstn_reg;
    wire [2:0]led;

    wire clk;
    wire rstn;

    initial begin
    clk_reg=0;
    rstn_reg=0;
    #10
    rstn_reg=1;
```

```
    end

    always #1 clk_reg=~clk_reg;
    assign rstn=rstn_reg;
    assign clk=clk_reg;
    //instance breath led
    breath_led #
    (
    .CLOCK_FRQ(1000000)
    )
     breath_led_inst
    (
    .clk_p(clk),
    .clk_n(clk),
    .rst_n(rstn),
    .led(led)
    );
endmodule
```

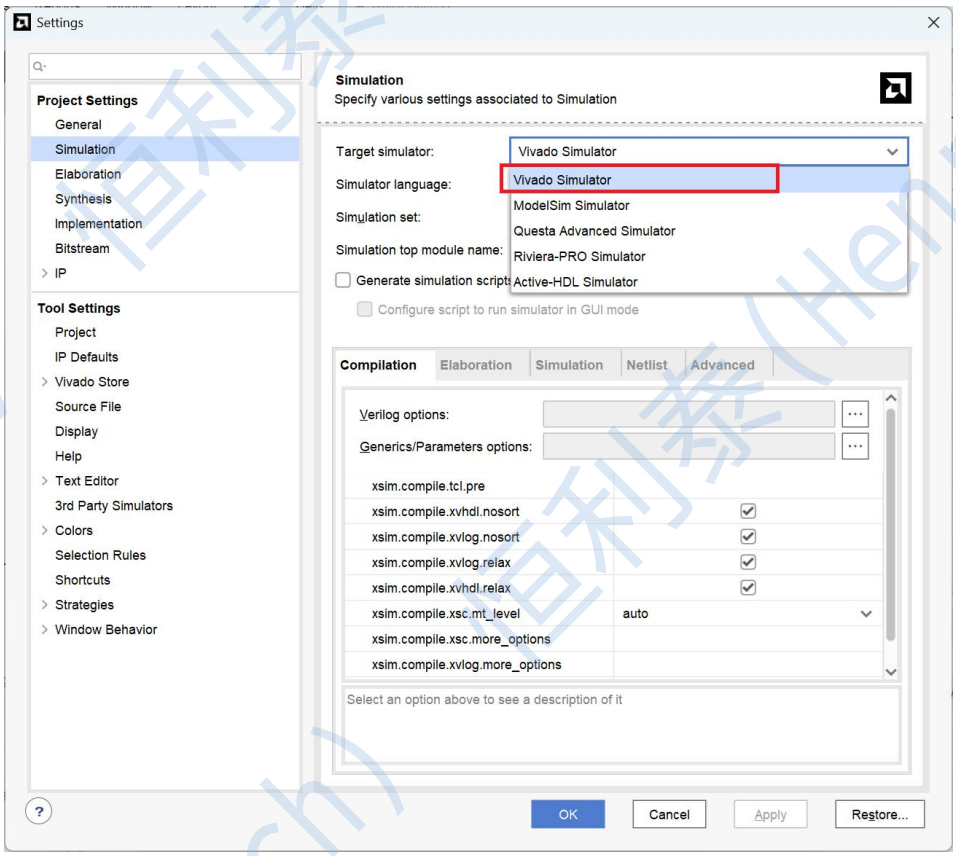
上述代码简单地产生了时钟和复位，然后调用我们呼吸灯模块进行驱动。我们为了提高仿真验证速度，将时钟信号参数CLOCK_FRQ改为100000HZ，通过参数传递的形式修改。如上代码标红的地方。这种技巧可以在仿真的时候使用，提高开发效率，大大节约时间。可能会有朋问为什么将时钟计数的这个参数改小就能加快仿真验证速度？这个很容易理解，计数CLOCK_FRQ到100000花的时间长还是计数到100000000所花的时间长？



我们设置仿真。首先，我们在工程管理栏的Settings，点击进去。在弹出的设置里面可以进行很多设置，有工程设置，型号设置，仿真设置等。

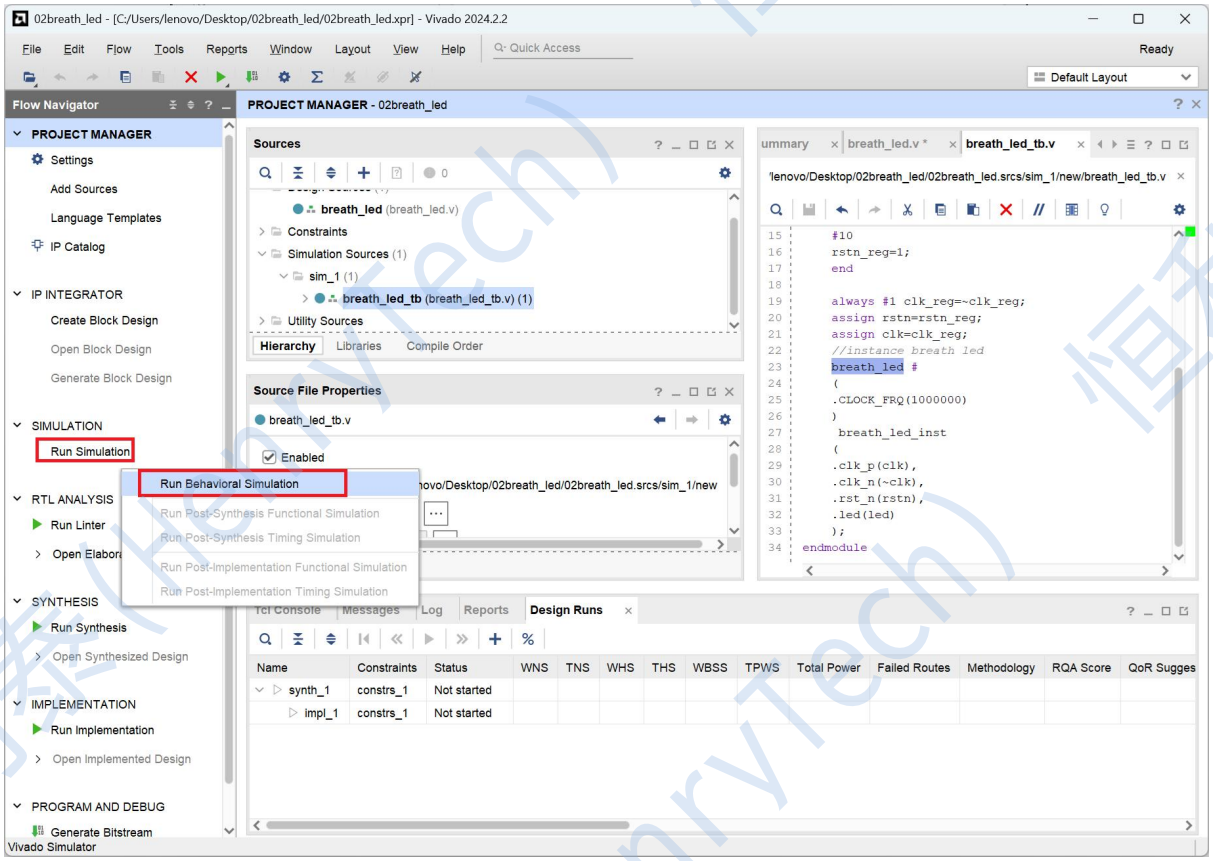


我们点击simulation进去，默认选择Vivado Simulator。然后点击OK退出。使用自带仿真器优点是不用编译仿真库，缺点是没有modelsim专业。点击OK就可以退出设置：

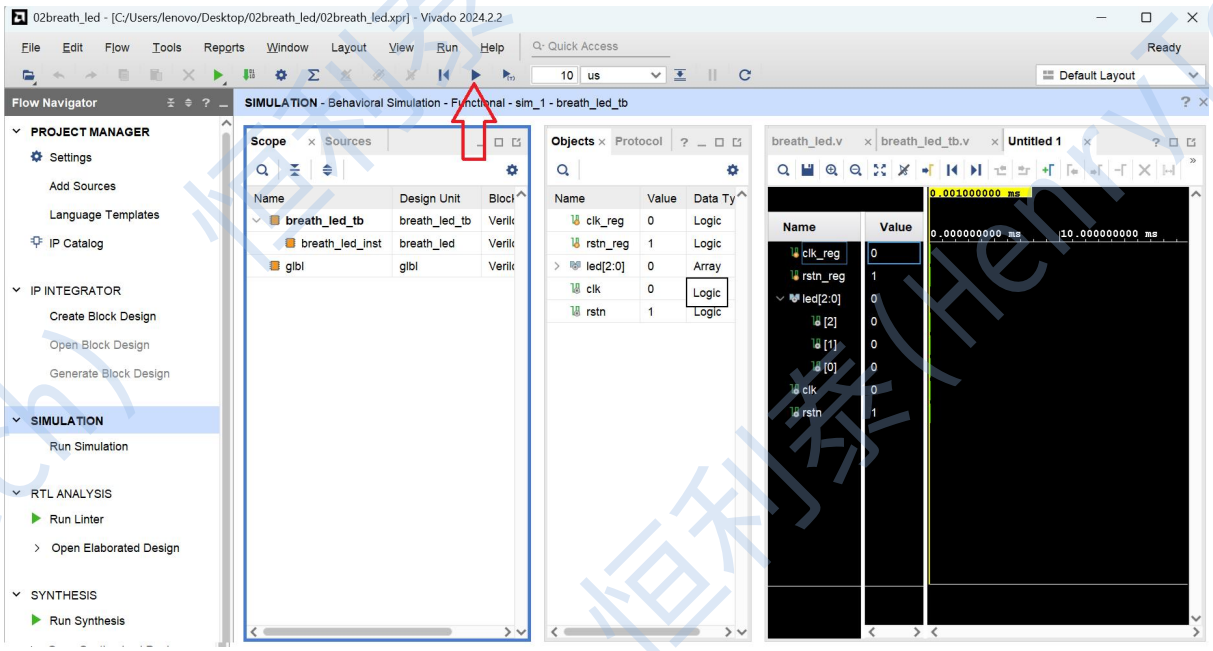


回到主界面之后，工程管理懒得SIMULATION下面的Run Simulation右键，选择第一个 Run

Behavioral Simulation进行逻辑行为仿真：

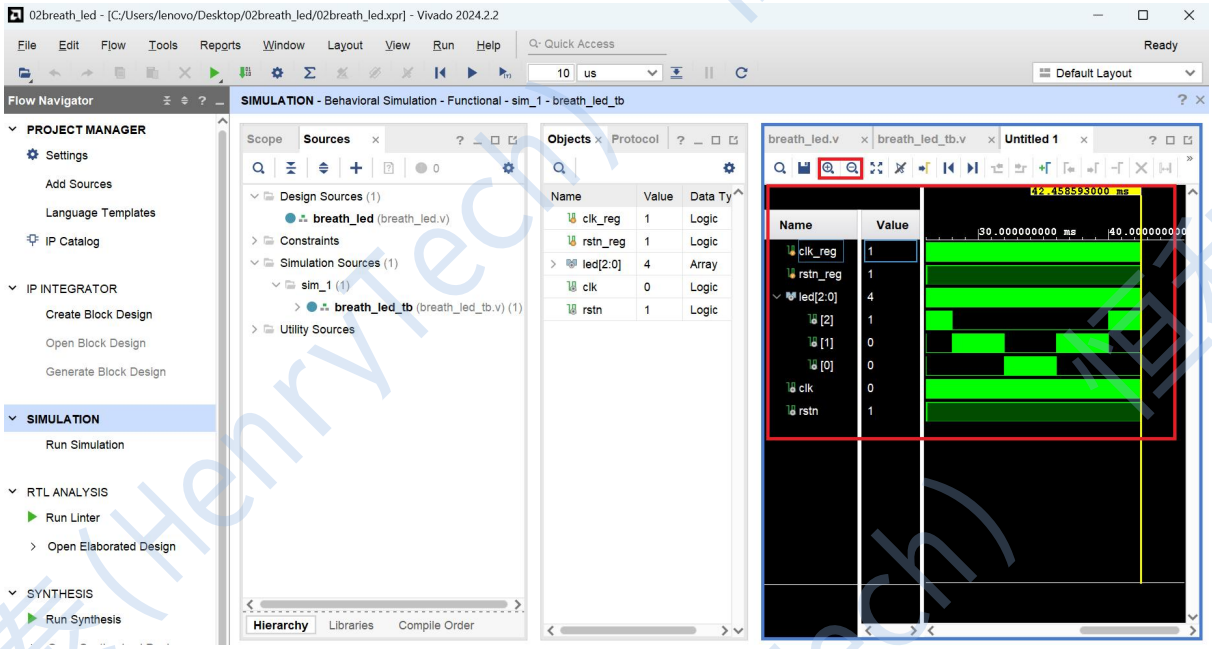


最后，仿真启动。通过上面的控制仿真启停按钮可以开始仿真。点击中间那个右三角符号启动仿真。如果综合过代码，还可以点击启动综合后仿真。这里我们就进行逻辑行为仿真。



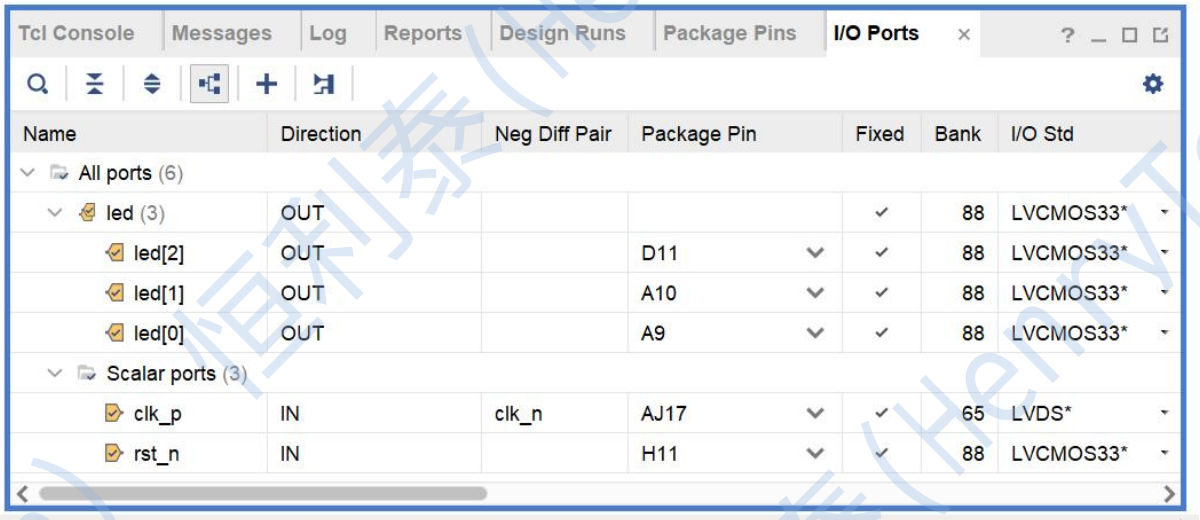
可以通过拖拽的方式适当重新布局，以及点击 可进行波形缩放。我们调整窗口和缩放，等待运行一段时间。可以看到，我们的LED信号输出随着时间，流动进行呼吸灯波形输出。注意，仿真时间多跑一会儿，大约30秒以上，然后再去点缩放调整看波形。要想暂停，

可以点击上方 ⏸ 符号。



五、验证效果

绑定管脚和综合布局布线生成比特文件请参考《07最简单的led流水灯》文档。这里附上管脚约束图：



然后直接点击生成比特文件，等待生成完成，下载到板子查看效果。本节教程到此结束。如果需要用到第三方modelsim仿真，可以在设置里面更改使用的仿真软件选项，然后安装modelsim进行仿真。但是可能配置modelsim仿真软件破解操作比较麻烦，本教程不做讨论。

六、常见问题

1. 看不到波形：多运行一会儿，缩放多调整尝试一下；

2. 下载到板子没有呼吸灯效果：检查一下是不是仿真的时候频率计数器CLOCK_FRQ是不是在代码中做了修改，改成100000000（100M晶振频率值）然后重新编译生成BIT下载到开发板。

在线调试进阶之MARK DEBUG

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 VIVADO工程新建 3
三、 在线调试和ILA简介 3
四、 使用mark debug语法标记 4
五、 准备工作之先删除原来的DEBUG设置 5
六、 重新设置我们的DEBUG在线调试 9
七、 配置触发抓取信号 15
八、 常见问题和经验总结 18

恒利泰(HenryTech)

一、文档实现功能介绍

本文档实现如何使用MARK DEBUG方式标记debug信号，然后设置在线调试，最后生成比特文件进行在线信号抓取。相信很多人之前使用过各个厂家，比如ila，chipscope等等。这里使用mark debug方式更简单而且更加灵活。而且，信号名字可以很大程度上保持不变，被保留，再抓取出来做信号逻辑分析有很大的方便。工程新建方法请参考文档《04VIVADO新建工程流程和软件介绍》。

二、VIVADO工程新建

本教程无需建立新工程，大家简单地编写自己的代码或者使用我们任意一个简单的例程即可做实验学习。

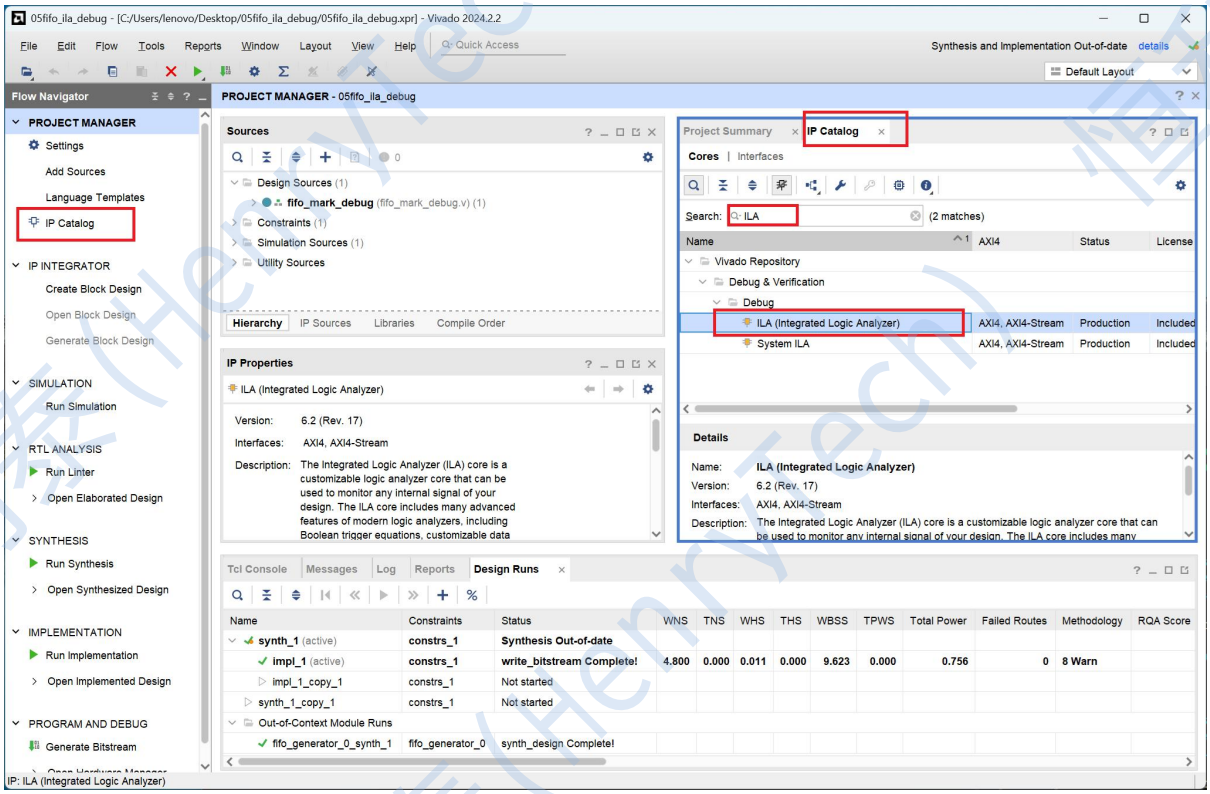
起始页（或 file->Project->New）	创建新工程（Create New Project）
向导起始页面	点击 Next->
Project Name（工程名）	工程名：任意（无需建工程） 工程路径：（自己选择，不要有中文路径） 勾选 Create Project Subdirectory，点击 Next->
Add Source（添加设计源文件）	点击 Next->
Add Exsixting IP（添加已有的 IP）	点击 Next->
Add Constraints（添加已有约束文件）	点击 Next->
Default Part（默认配置，芯片选型）	Family->Zynq UltraScale_RFSoCs Package->FFVE1156 Speed->-2 按具体型号选择 XCZU67DR-FFVE1156-2-i
New Project Summary（新建工程概况）	确认工程信息，选型等，点击 Finish 完成

三、在线调试和ILA简介

我们例程有很多都可以做在线调试学习。我们就拿FPGA例程的《05fifo_ila_debug》这个例程，从头到尾详细讲一遍整个过程。由于我们在工程开发中需要抓取实际的逻辑代码的运行状态和值，在某个条件触发的情况下，确定逻辑是否实际正确。很多时候我们虽然仿真没有问题，但实际上运行出现了问题，这个时候我们必须做在线调试，在线调试可以把实际情况抓出来研究。相信很多朋友都知道逻辑分析仪，在线调试就是FPGA芯片级的逻辑分析仪，比真

实的好用，而且高效。

Xilinx厂家使用了一个叫作ILA的逻辑模块，也就是一个集成的在线调试IP。大家可以打开IP管理器输入ILA，可以看得到。大家在线调试可以直接调用这个IP，可以自己研究一下怎么使用。我们本教程使用MARK DEBUG方式，虽然代码没有调用这个IP，但是实际上软件自己帮我们调用了这个IP连接到我们设计的代码中。



四、使用mark debug语法标记

标记需要被debug的信号方法：（*mark_debug="true"*），也就是在需要被抓取的信号前添加这一句标记语，记住，这个语法不要写错了，建议直接拷贝。对于可以标记的信号，建议不要标记模块的信号列表，直接在定义的wire 或者reg 变量名字前面添加。比如：

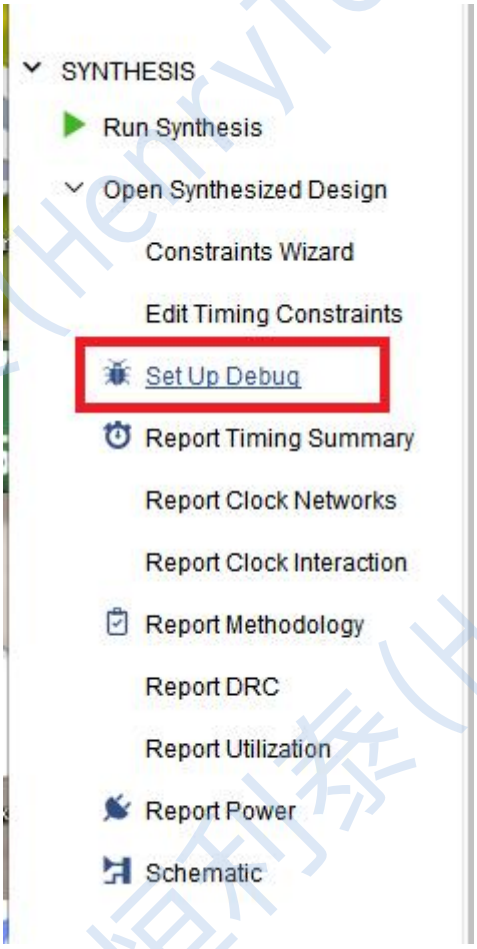
```
(*mark_debug="true"*) wire clk_input;
(*mark_debug="true"*)  reg valid_reg;
```

标记语法还可以保留这个信号名字。由于FPGA代码被综合后，很多内部的信号会关联一些缓冲器BUF信号，信号名字转换成网表的时候名字会发生变化，这个时候调试我们就无法对应的真实信号名称。所以，加了这一句综合的时候，网表对于这个信号名字也不会改变。当然，对于某些特殊信号，比如inout信号，加了之后也会被综合工具网表名字添加输入输出BUF的名称字样，当然这是少数情况。

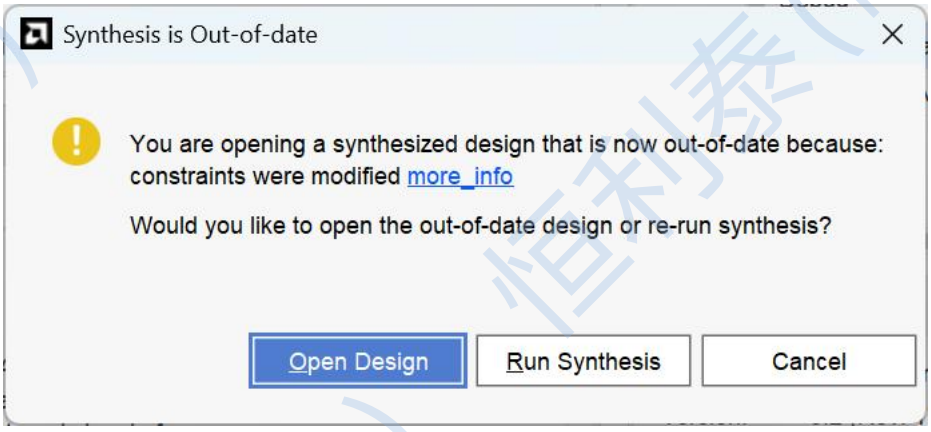
标记之后，我们的在线调试工作还没有结束，还需要一些步骤。

五、准备工作之先删除原来的DEBUG设置

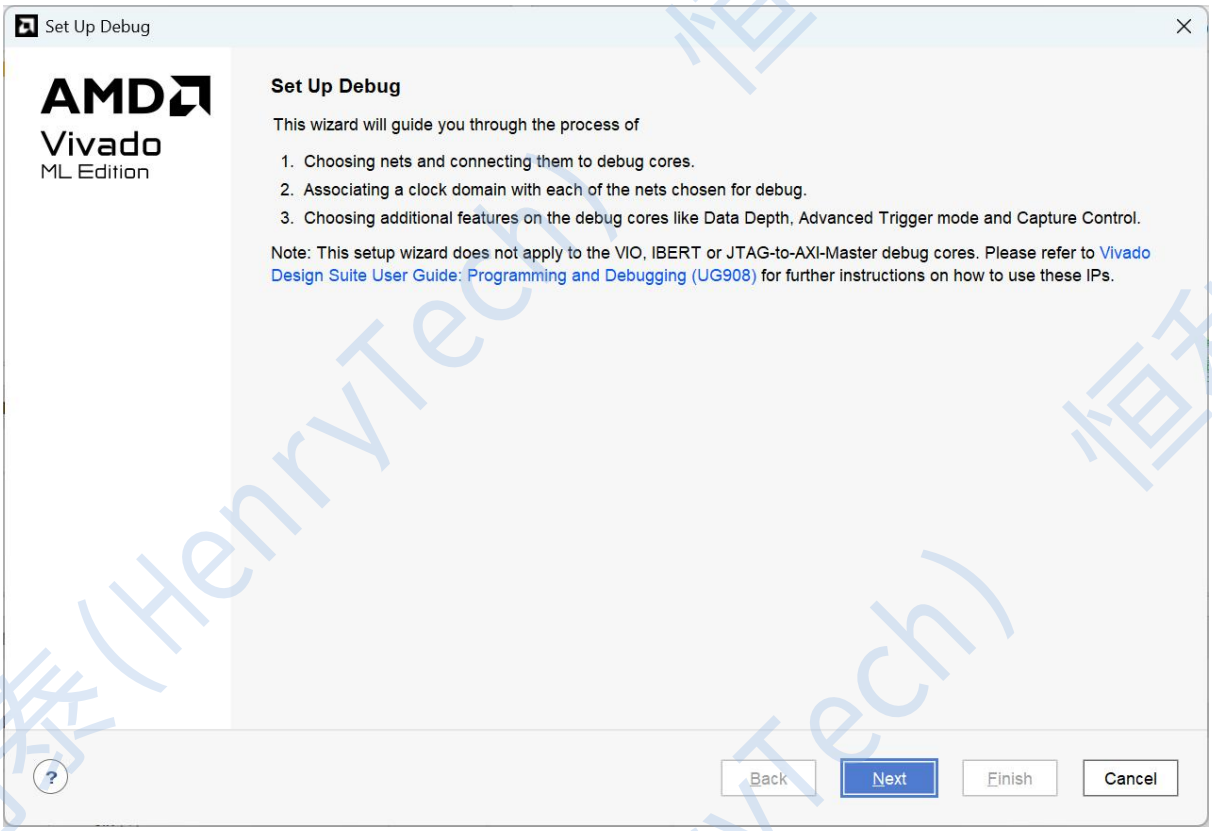
设置在线调试之前，我们先给大家做一遍如何清除在线调试的过程。在后续代码在线调试结束后，不需要在线调试逻辑功能了，可以根据这部分来清除在线调试。我们给的例程05fifo_ila_debug已经有了在线调试逻辑，我们以此演示。首先，我们打开SYNTHESIS下面的Open synthesized design，找到set up debug打开它：



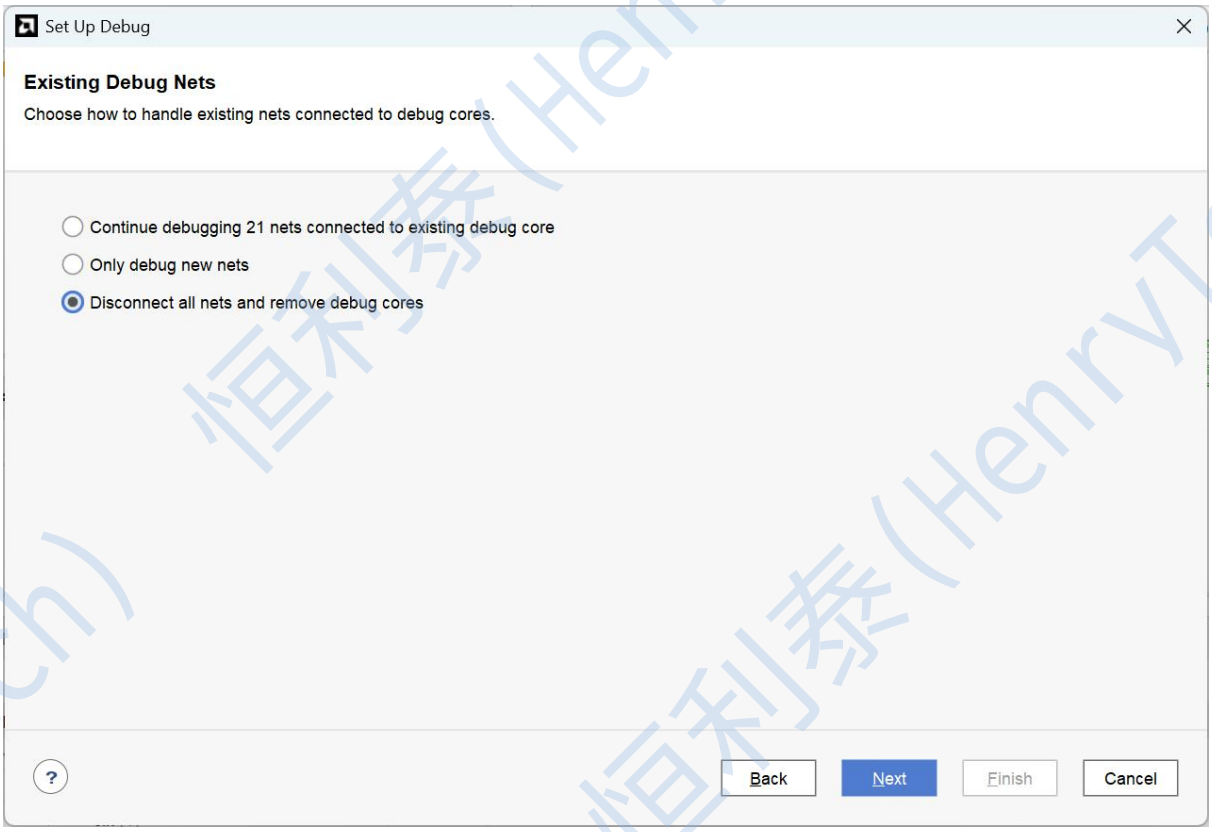
如果弹出如下界面，这个界面表示，如果你上次综合结束，改过代码，没有再次综合就会弹出，这里我们新综合一下吧！点击中间的Run synthesis然后等待综合结束：



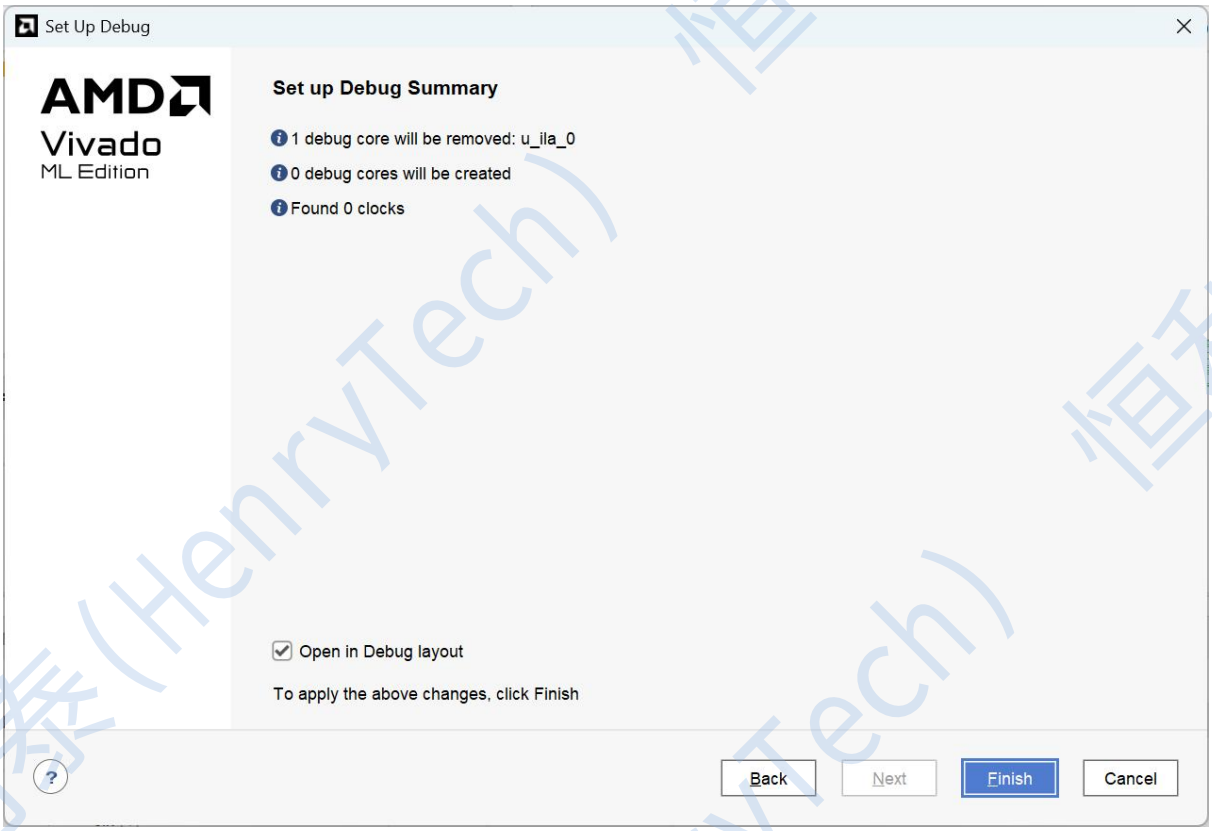
结束之后再次点击set up debug打开设置在线调试，第一页点击NEXT：



点击第三行，移除所有的debug信号再点击NEXT：



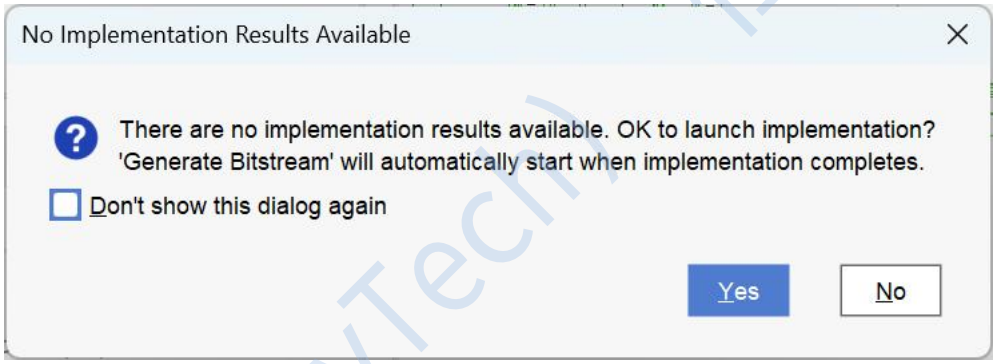
然后直接点击Finish：



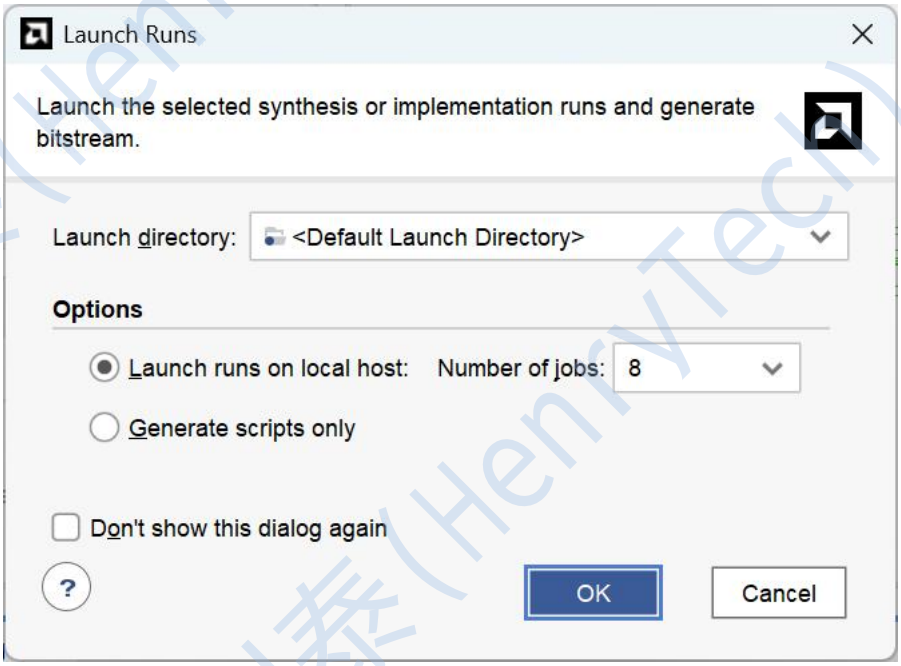
移除之后，直接点击生成bit文件。生成之前，还会弹出对话框询问是否保存xdc约束。这个约束文件被系统将debug信息的约束全部删掉了，所以我们点击Save：

接着会弹出提醒综合过期，点击OK：

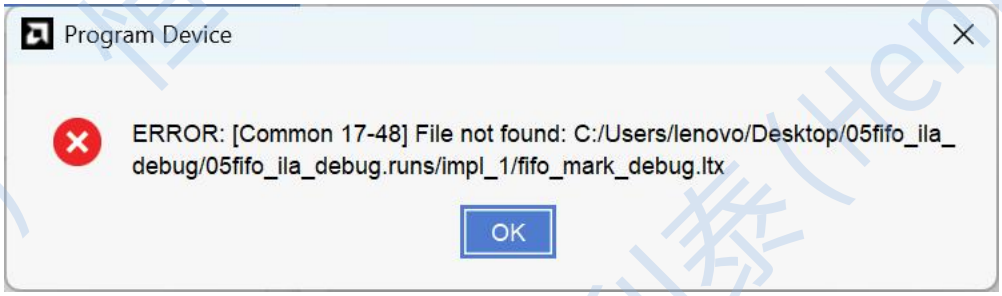
接着弹出的对话框直接点击YES：



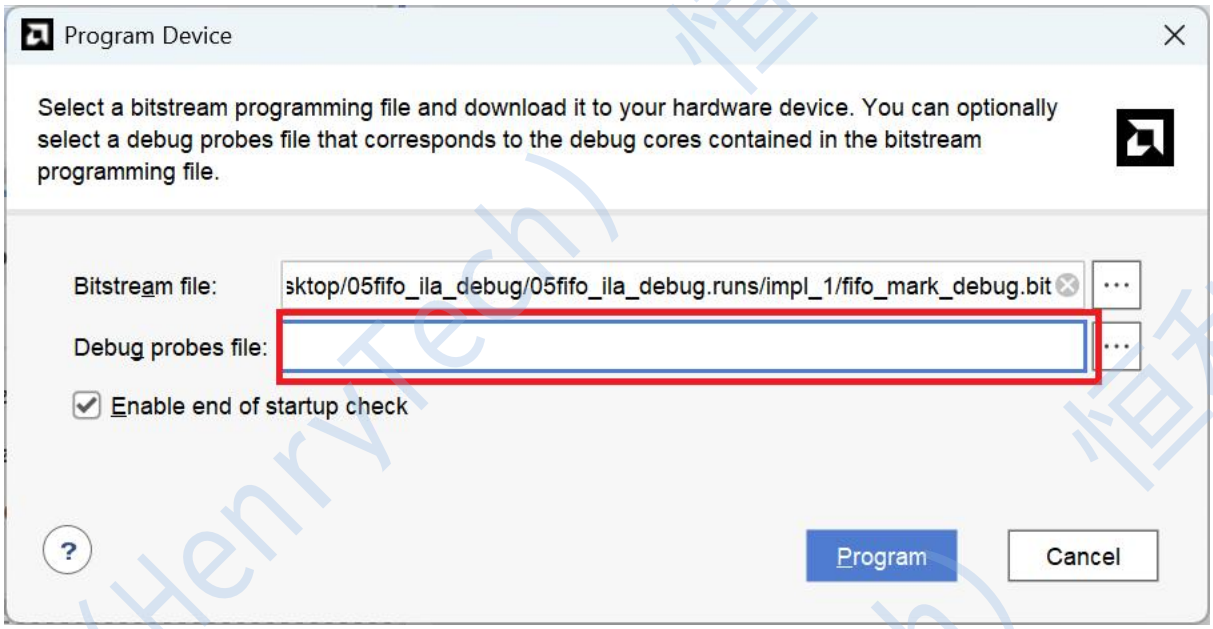
然后点击OK即可。



等待重新生成bit文件完成。然后打开硬件管理器，连接开发板，芯片右键选择烧写program，点击烧写，提示错误，因为那个debug的文件已经被删掉了。



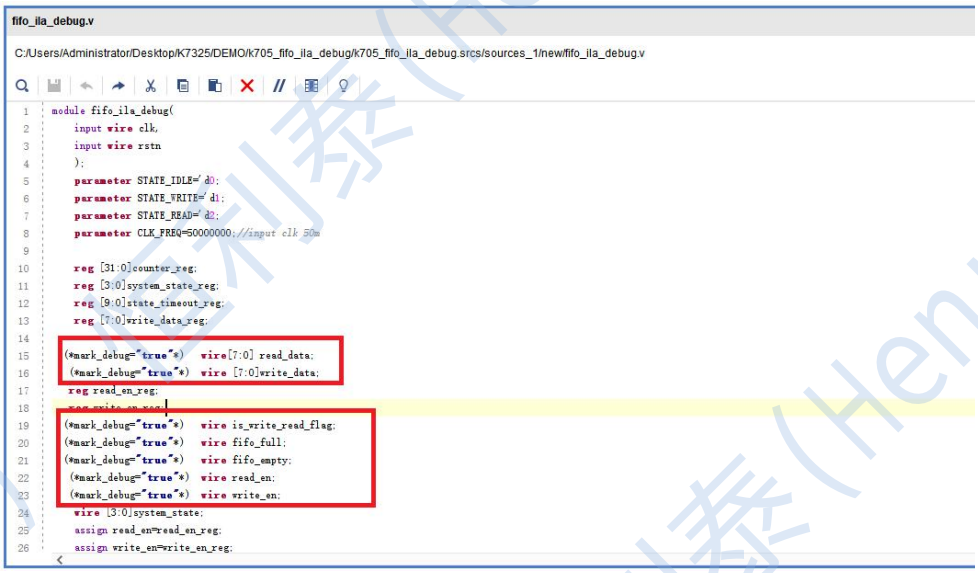
我们重新点击芯片右键program，将第二项文件路径清空再烧写即可烧写了：



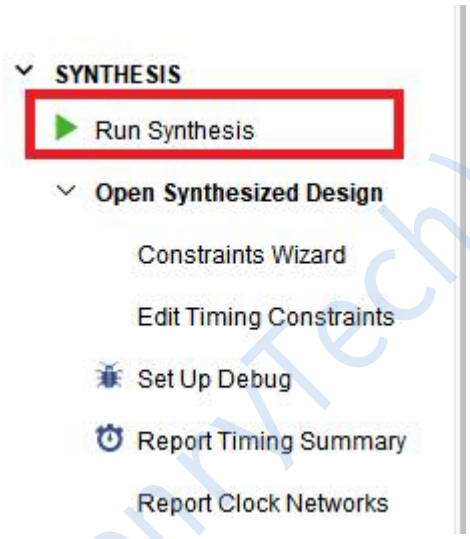
至此我们工程的DEBUG设置全部清除干净了，接着我们开始正式从头开始设置debug：

六、重新设置我们的DEBUG在线调试

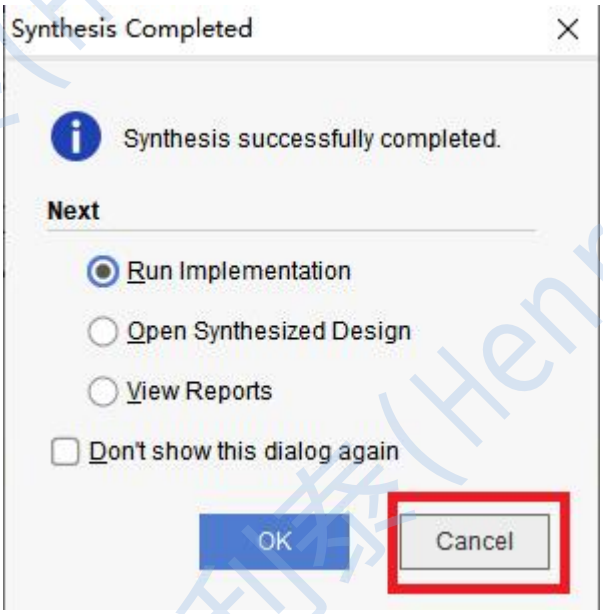
到此为止，根据上一节，我们将提供的demo的在线调试已清除干净。我们开始设置。首先，我们代码里面已经把所需要查看的信号前面已经标记了mark debug：



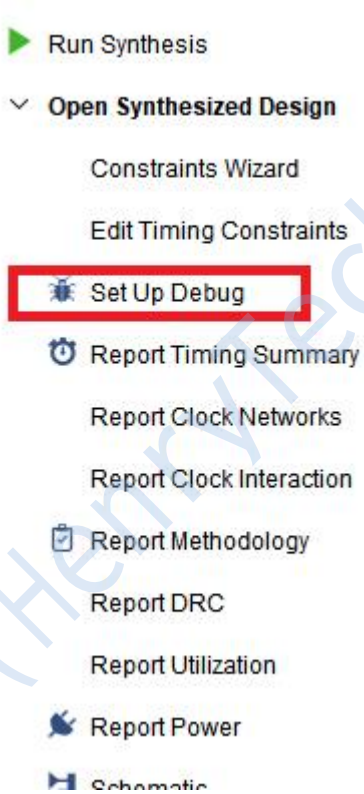
然后我们开始综合，点击综合按钮：



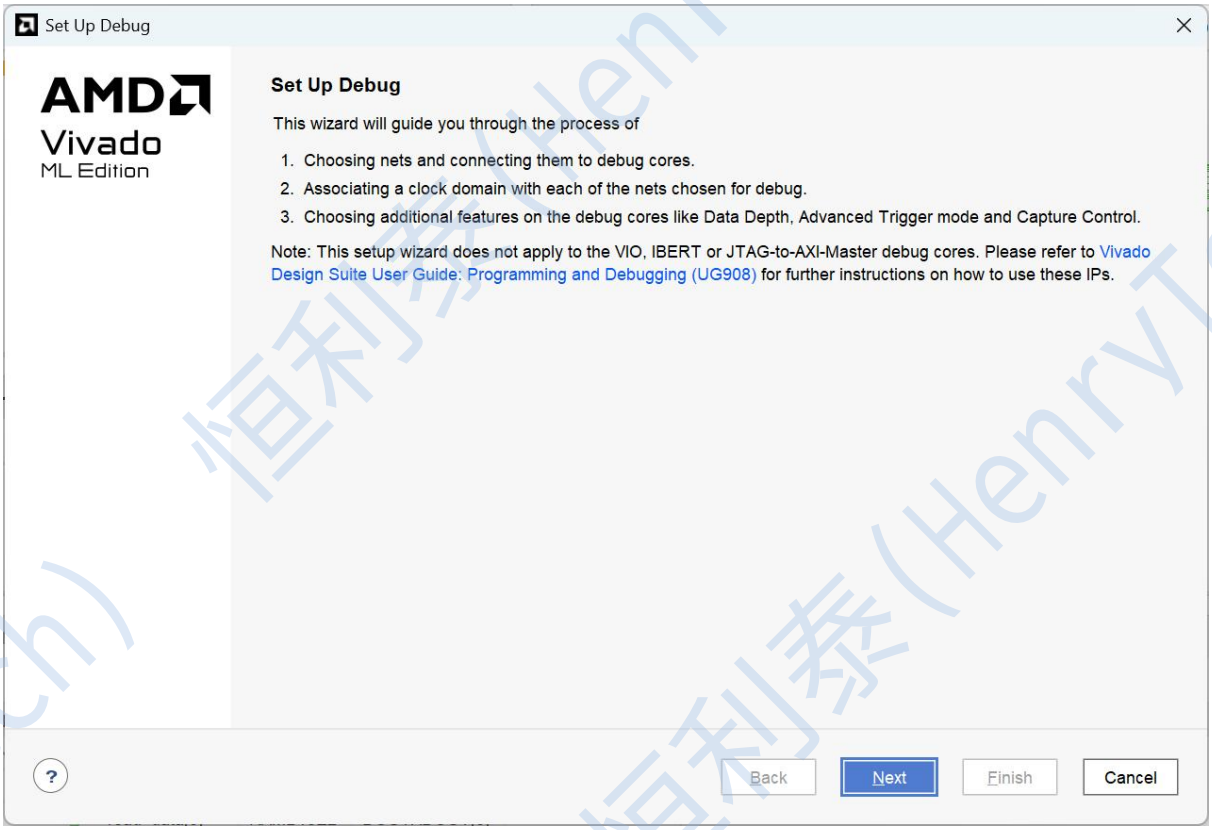
综合结束，点击cancel取消实现：



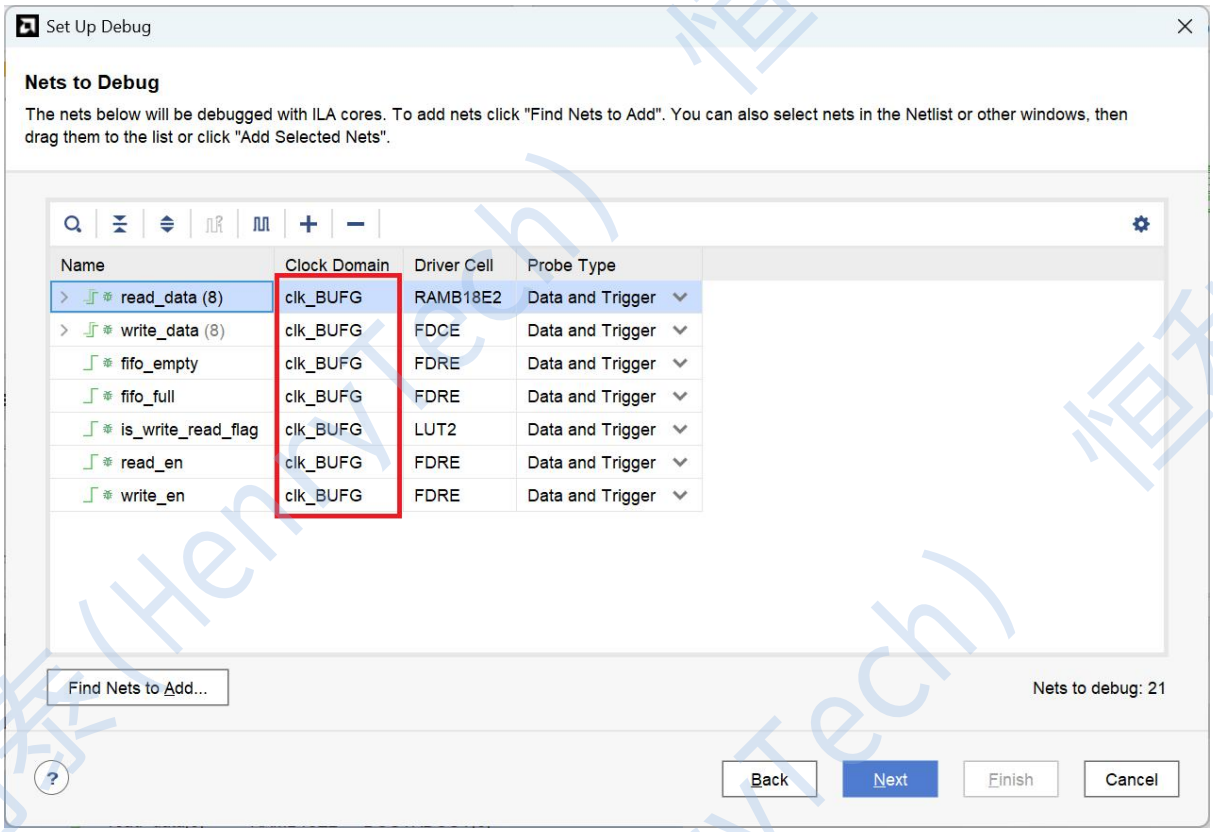
然后我们再点set up debug设置在线调试：



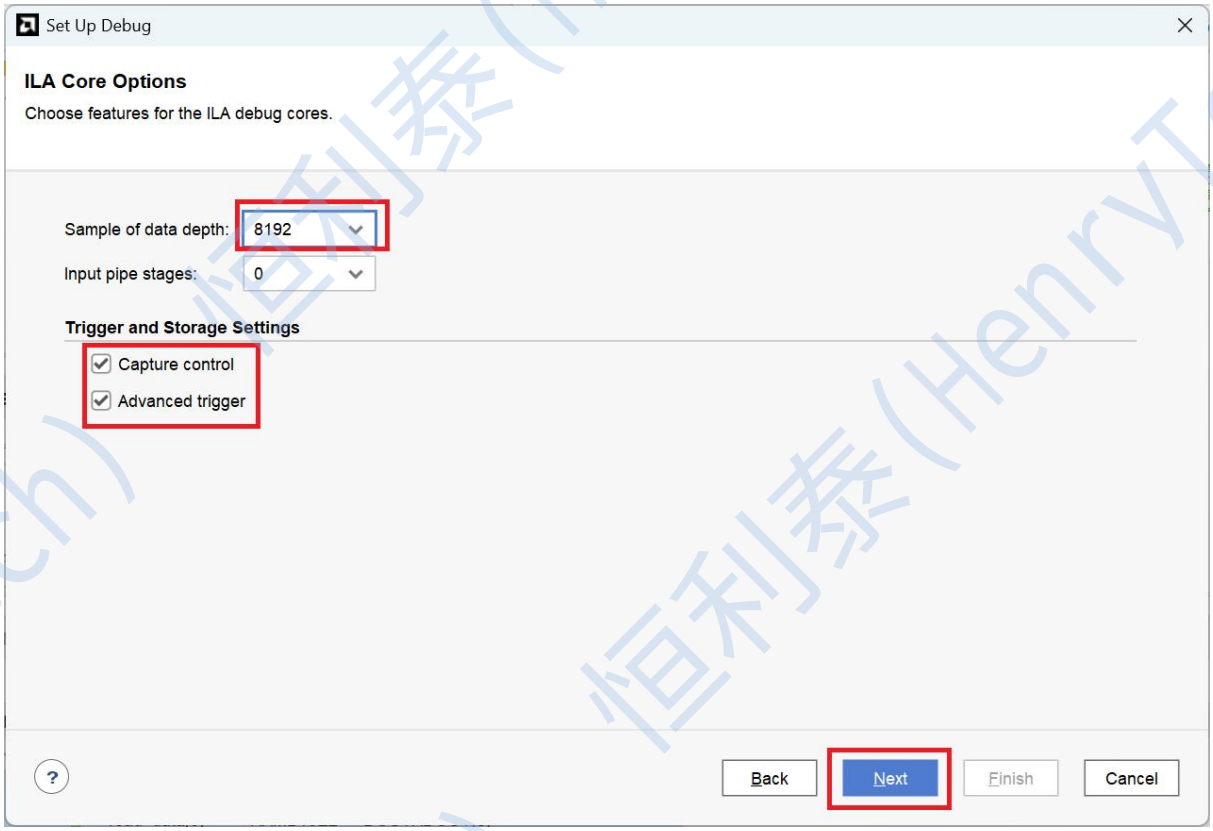
点击下一步Next：



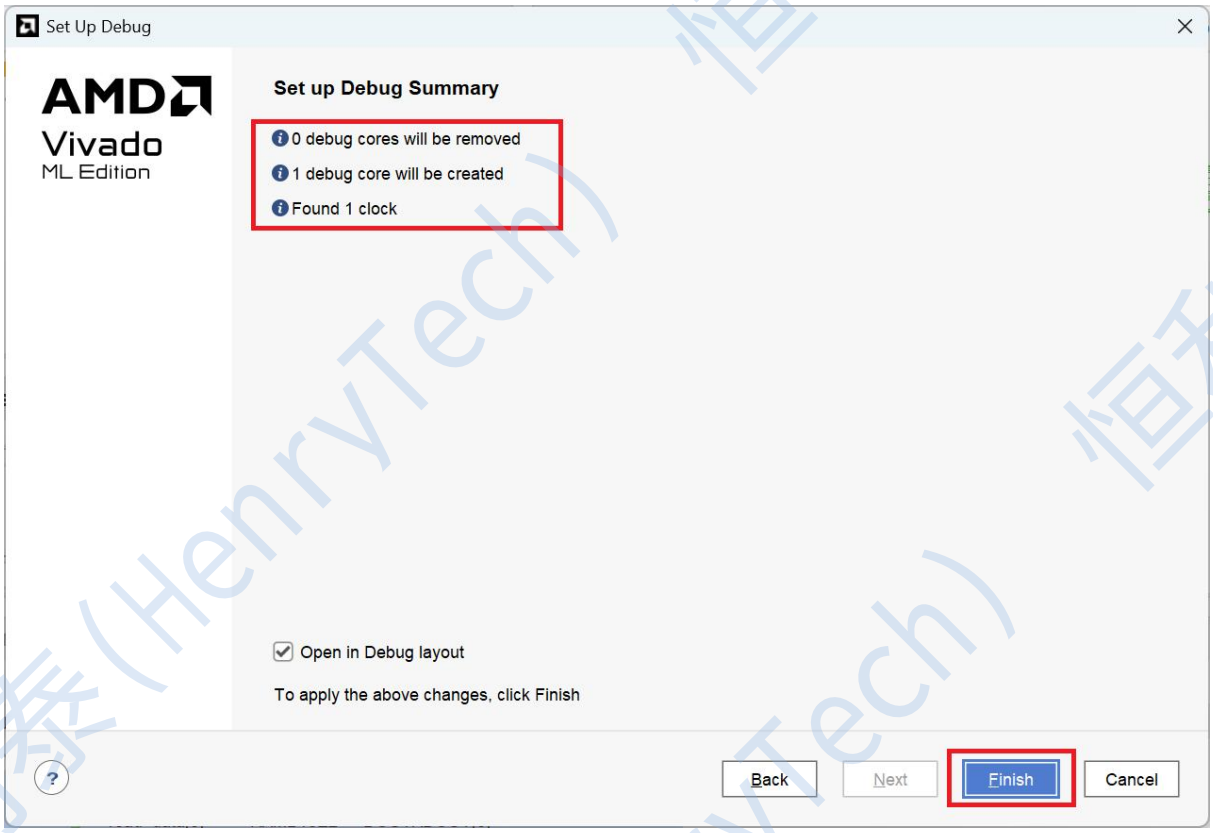
这里我们可以看到出来的信号列表。有的工程中，有多个时钟的，要debug的信号后面可能时钟域不一致。要注意，时钟域表示这些信号工作的参考时钟，如果不是同一个，可能抓取的信号不正确。我们本教程的演示用的demo只有一个时钟，所以都默认是这个时钟。由于我们没有对这个时钟标记mark debug，这里看到的就是综合后网表里面的带BUF字样的信号。然后点击下一步：



设置采样深度和其他选项。勾选下面两项，表示信号同时可以做捕捉控制和更多触发，也就是支持任意信号的触发条件设置，然后点击next进行下一步：



看一下debug最后一个页面信息，找到了一个时钟，生成一个debug core 。点击finish完成设置：

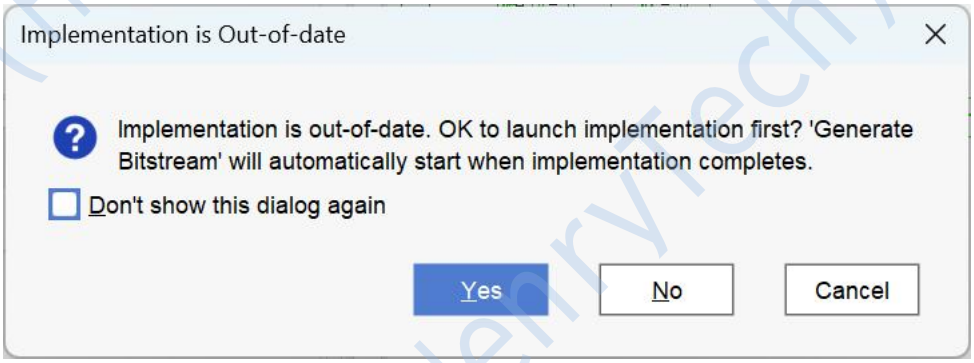


直接点击生成bit文件：

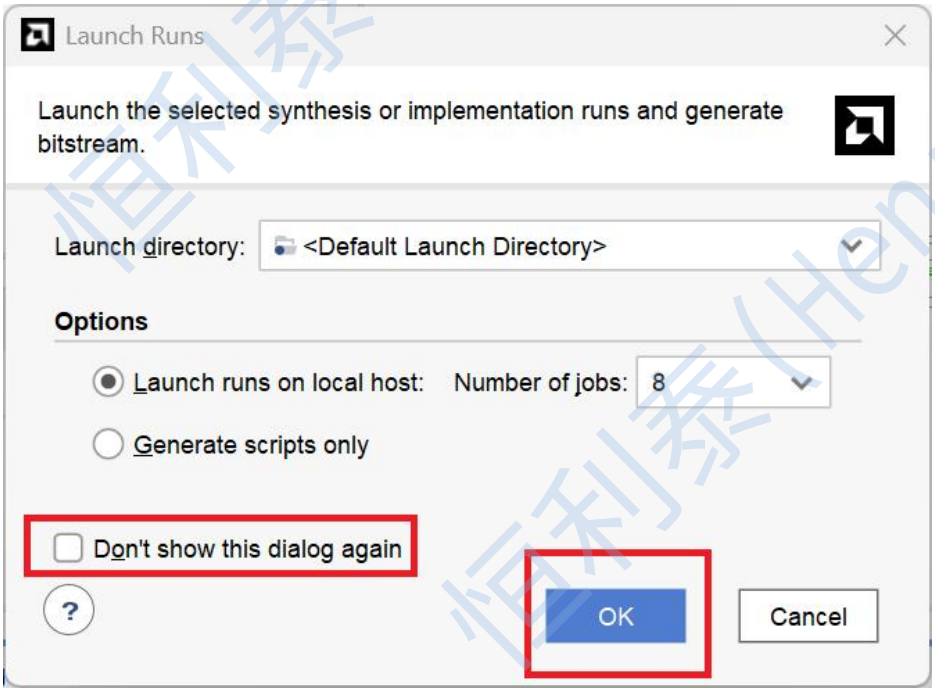
弹出的对话框点击save：

再次弹出对话框点击OK即可。

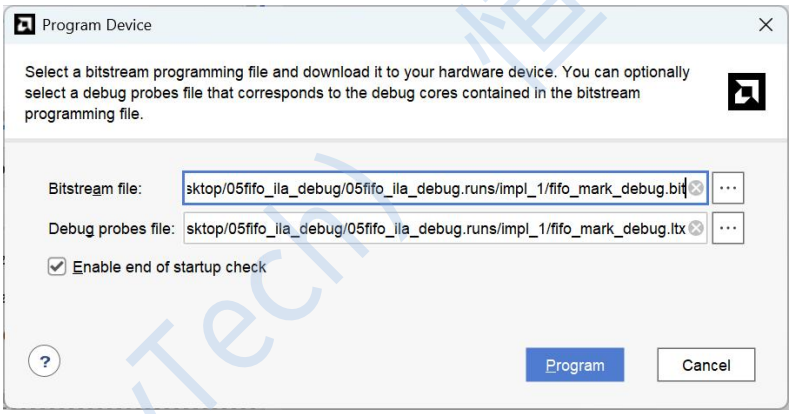
再次提示没有实现（布局布线）过程由于我们设置完debug还没有点击implementation进行布局布线，直接生成比特文件，所以提示你需要布局布线的implementation，直接点击OK。VIVADO很多时候会弹出各种对话框，大家看清楚读一遍，然后点击OK或者YES即可。



然后弹出设置线程。这里很多窗口大家不想每次都点，可以勾选上不要再提示这个界面即可：



等待生成比特文件完成。生成完成，打开硬件管理器，打开目标硬件找到xczu47dr芯片右键program烧写，弹出的界面可看到，第二项的debug文件又出现了。然后烧写进去，弹出在线抓取的界面。

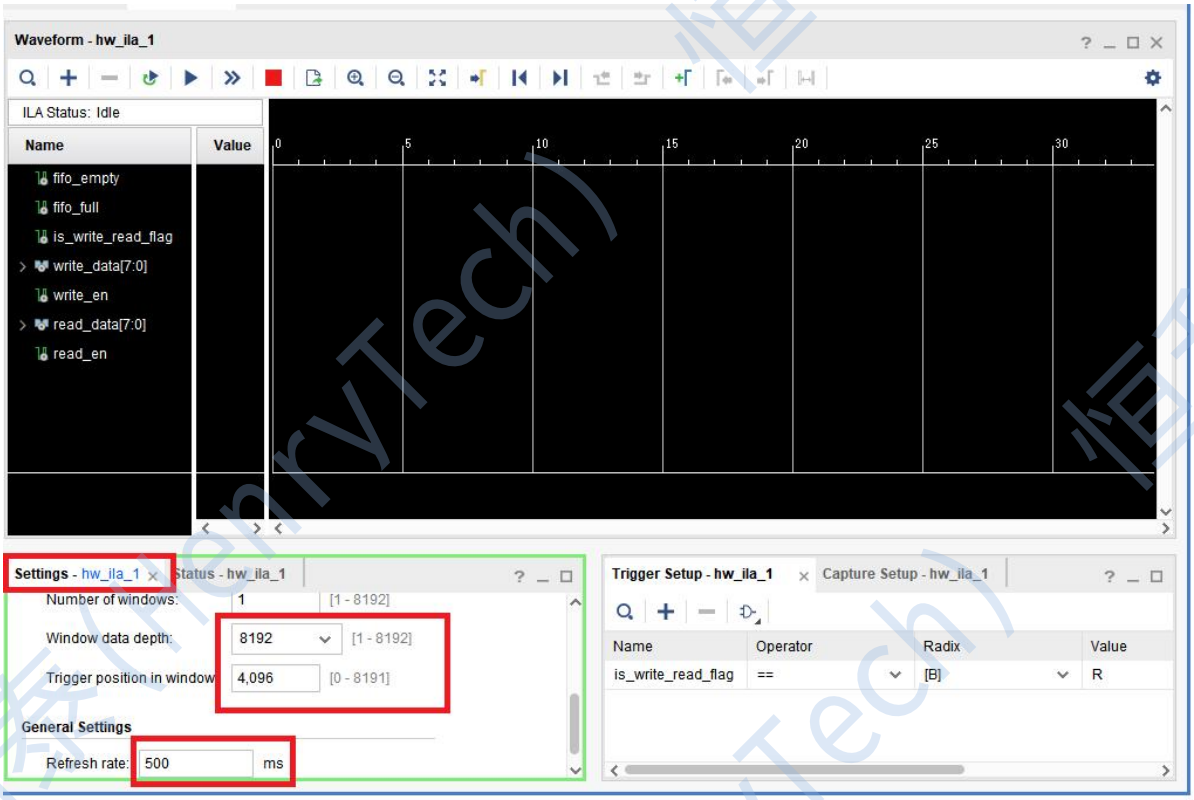


接着我们设置触发条件和采样配置。

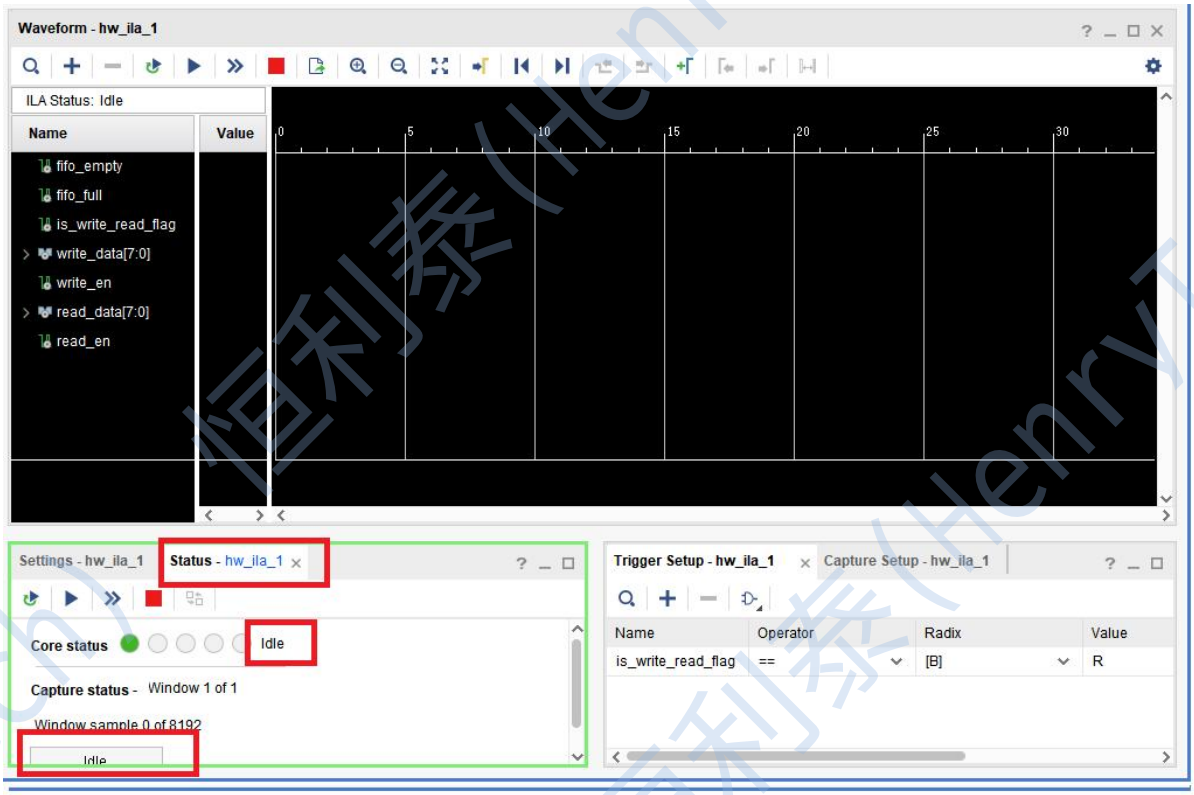
七、配置触发抓取信号

在线调试抓取，有几个概念。第一，触发深度。触发深度指的是触发条件来了，存取的信号的多少。1024代表存储1024个采样时钟周期的数据，8192表示8192个。第二，触发位置。触发位置为0，表示触发条件满足，从0时刻开始存储接下来的采样数据，也就是从条件满足那一刻+采样的深度X时钟周期的数据会被存起来。如果是从采样深度中间位置触发，则条件满足的前一段时间的数据也会存储起来，这样可以看到触发条件产生前后所发生的数据变化。根据自己实际情况设置。第三，触发条件。触发条件可以是某个信号等于多少，或者某个信号上升沿，下降沿等。

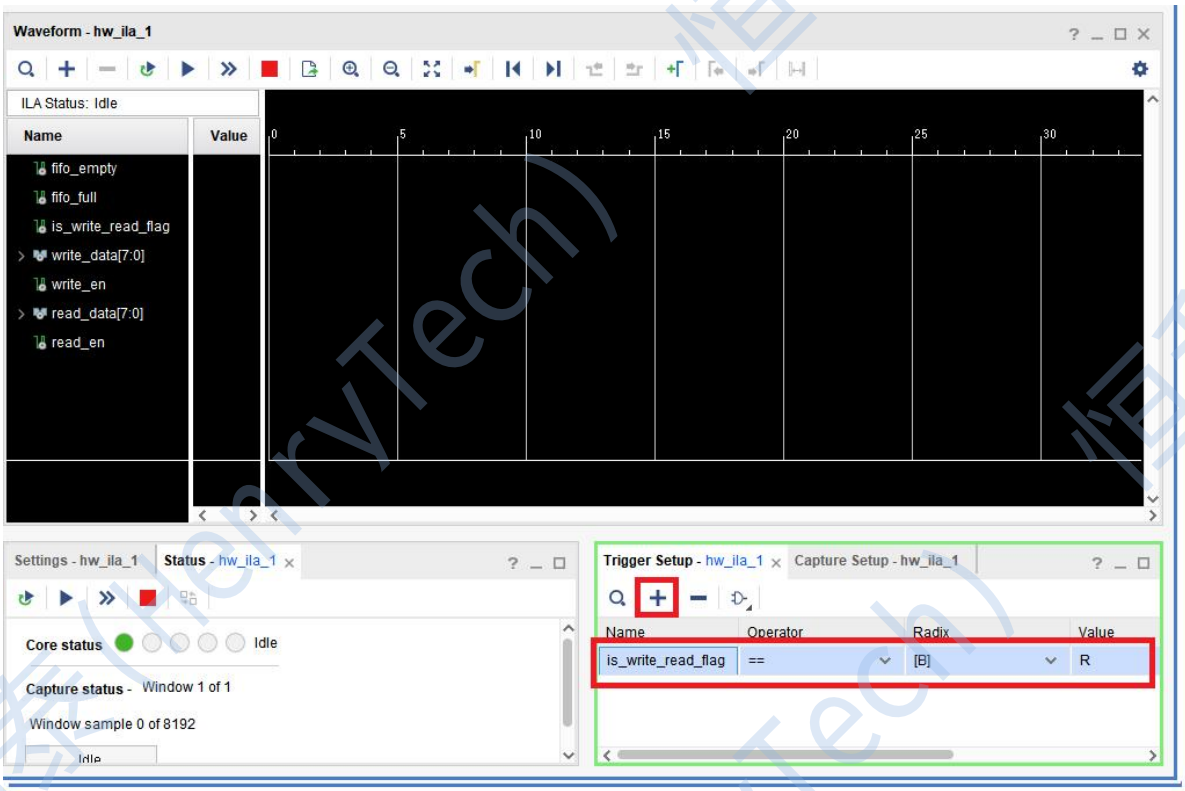
我们先设置基本的采样。首先示采样深度和触发位置。一般来说，采样深度在debug配置的时候就设置好了深度，这里不用修改。然后是触发位置，我们默认是从中间位置触发，也就是可以看到发条件前后发生的事情。这个也可以改成0表示触发条件开始之后的数据进行采样。下面的rresh rate 刷新频率，也就是波形刷新速度，越小越快，默认我们不修改。



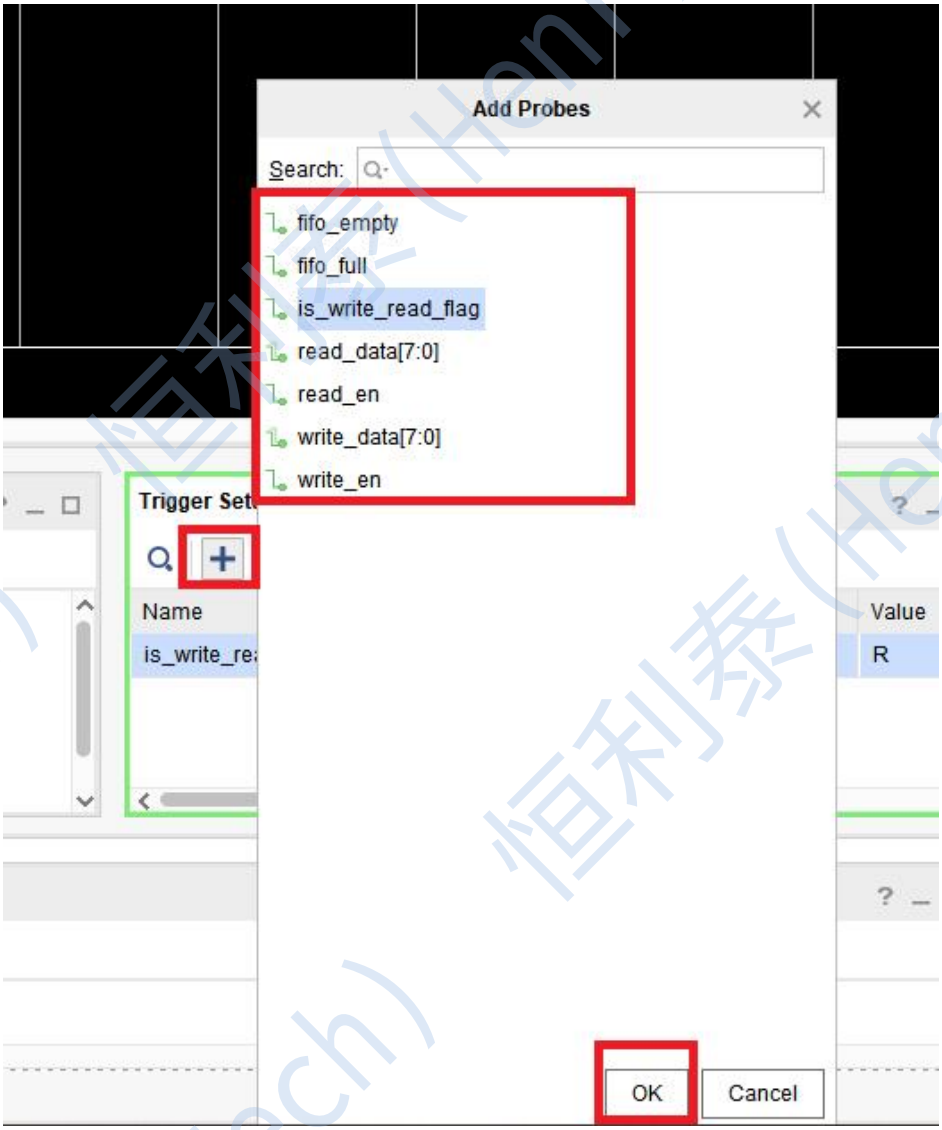
触发采样状态，当前显示idle标志正在空闲。没有等待触发。



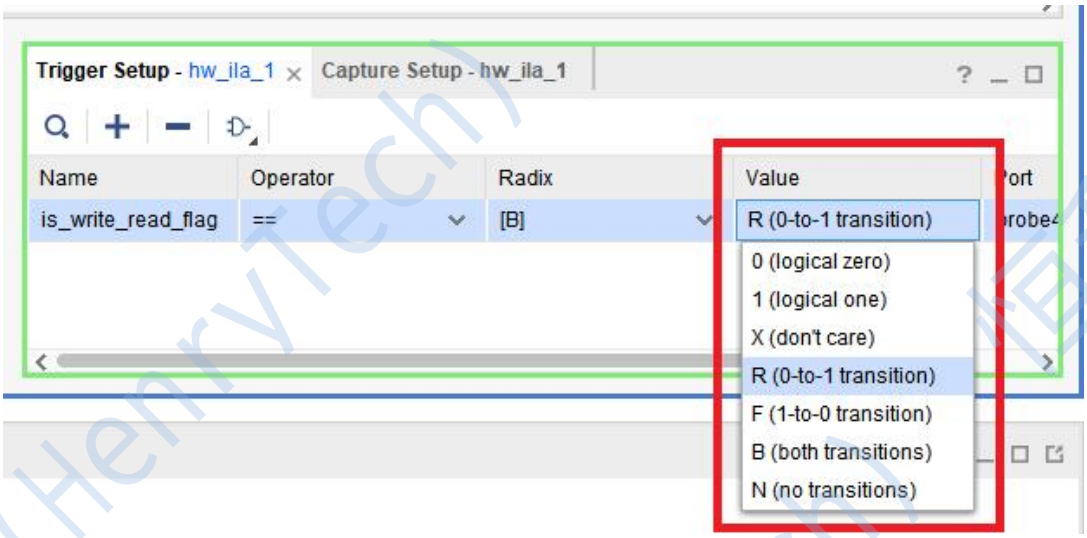
触发条件设置，触发条件是自己设置的，这里我们设置为正在读写操作信号。这个信号在代码中拉高表示在读写，空闲的时候拉低，所以我们可以使用它做触发，VALUE值设置为R表示上升沿：



点击触发设置的+号可以看到能够设置触发的列表，这里任何信号都可以做触发设置：



再看一下可以设置的触发条件值，其中R,F比较常用，表示上升沿下降沿，也可以是0或者1，表示等于0或者1的时候就触发：



触发条件了解完，我们还是只保留那个is_read_write_flag的上升沿R做触发条件，我们点击三角符号 抓取（也可以点击带圆形箭头的三角形 连续抓取，可以一直刷新波形采样）：

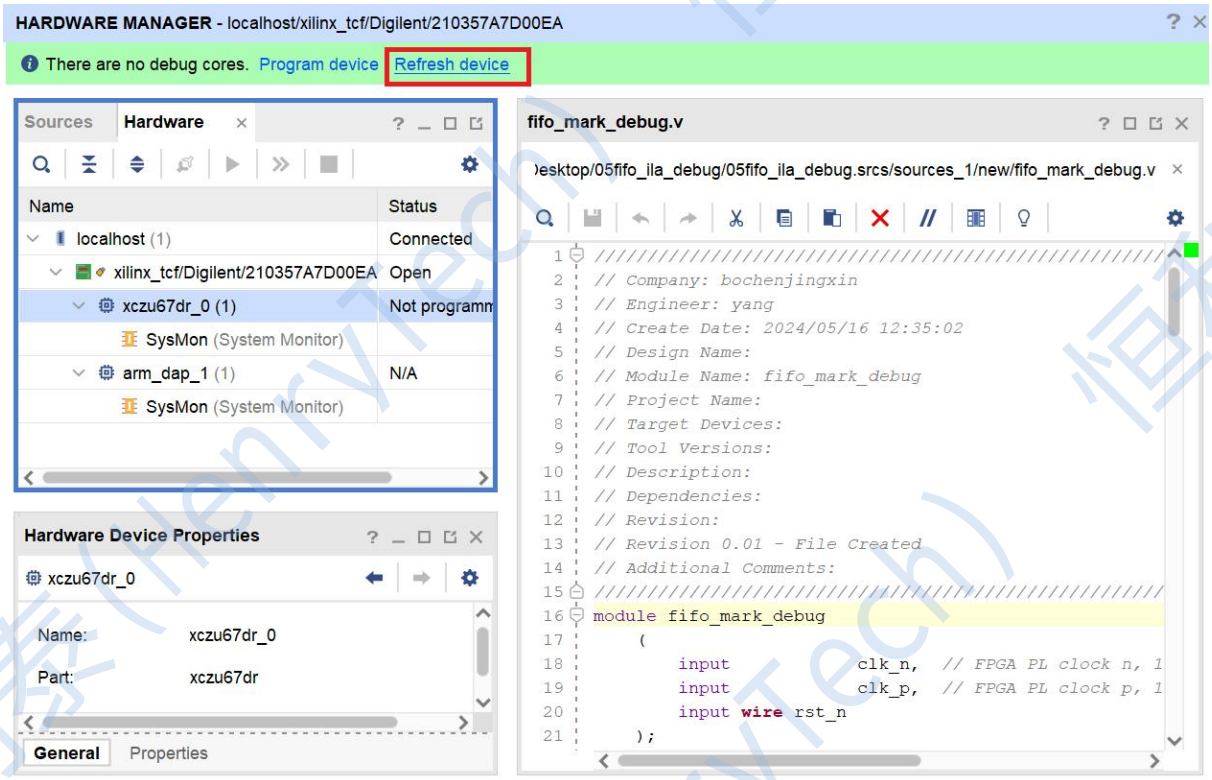
上面可以看到，抓到波形了，红色T线表示触发位置。大家可以放大波形详细查看。

八、常见问题和经验总结

1、我们抓取信号的代码中有很多个时钟怎么办？解决办法；建议只关注所需要关注的信号，将这些信号选择它工作的时钟作为时钟域。或者，需要查看所有信号的时候建议系统添加一个比所有时钟快两倍以上的时钟作为采样时钟，可以抓到慢时钟域信号。

2、下载之后不出现在线调试界面？解决办法；赛灵思的在线调试有个机制，下载之后会检测这个采样时钟有没有翻转，如果没有，就不会启动在线抓取界面。如果这个时钟在下载之后过一会儿才启动，则在下载之后过一会儿，芯片型号右键刷新设备refresh device即可出

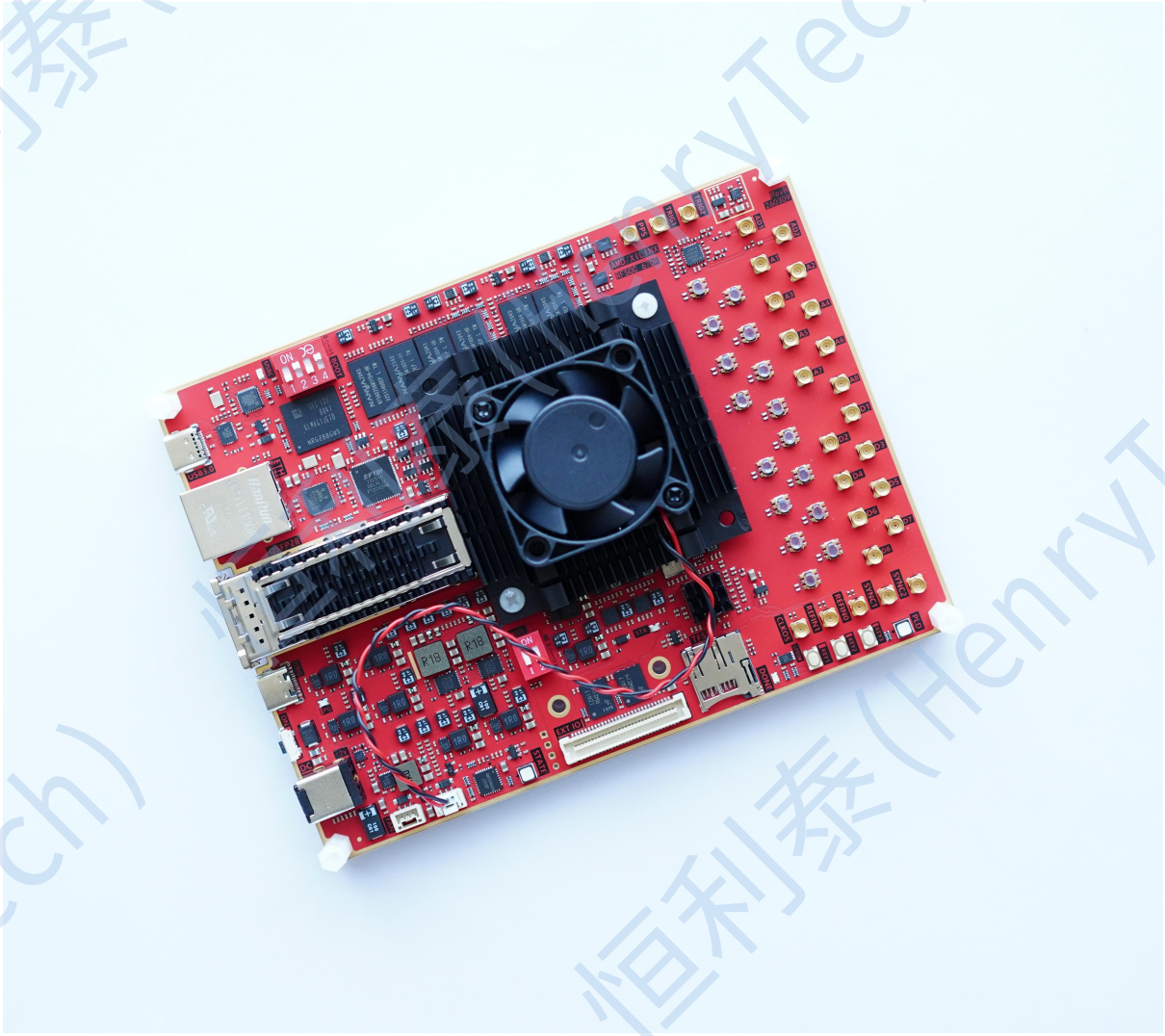
现，接着就可以进行抓取：



很多在线调试的技巧，大家多多使用，在实践中成长，本教程到此结束。

08petalinux开发简介

------ 基于RF67DR-Lite开发板套件



恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 配套版本和petalinux下载 3
三、 系统配套 3
四、 参考文档 3
五、 开发建议 3
六、 提供的Demo运行 4
七、 设备树 4
八、 USB3.0 SLAVE测试 4

恒利泰(HenryTech)

一、文档实现功能介绍

本文档简单介绍petalinux的开发，不做具体步骤讲解，由于petalinux编译基于ubuntu安装petalinux工具包，25年后官方可能停止支持更新，开发环境可能存在很多不支持情况，所以对开发过程不做讲解。我们简单说明一下本开发板配套petalinux的编译开发环境版本，以及其相关开发环境搭建。

二、配套版本和petalinux下载

Petalinux新版本管饭已停止更新维护，链接如下，后续可能会失效。

https://www.amd.com/zh-cn/products/software/adaptive-socs-and-fpgas/embedded-software/petalinux-sdk.html

Petalinux工具基于2024.2版本的下载，通过我们提供的百度云进行下载。

三、系统配套

虚拟机：vitualbox 7.2.6

Ubuntu:22.04.4

主机windows：win11专业版

电脑配置越高越好。

四、参考文档

Petalinux2024.2参考官方文档：ug1144-petalinux-tools-reference-guide-en-us-2024.2.pdf，位于《RF67DR_Lite开发板资料\02软件和相关linux包》目录中。

五、开发建议

针对没有开发经验的朋友，我们建议，ubuntu可以下载安装trae（字节跳动旗下的AI开发工具），给他提要求，检查当前系统环境是否满足编译开发要求，如果需要安装依赖包，还可以把ug1144-petalinux-tools-reference-guide-en-us-2024.2.pdf提供给他，让他进行分析生成依赖包安装指令。

编译过程可能由于网络问题编译失败，可尝试科学上网，或分时错高峰，在早上的时候进行编译，下载会比较顺利。关于安装petalinux、编译该过程任何报错，均可尝试给trae进行分析，然后提出建议解决方法。

六、提供的Demo运行

位于网盘的《RF67DR_Lite开发板资料\04例程demo\Petalinux\01petalinux\编译好的启动文件》中的三个文件，BOOT.BIN、boot.scr、image.ub复制到一张经过格式化FAT32，容量建议32G以内的TF卡上（注意必须格式化为FAT32，以及容量必须小于32G），然后设置BOOT模式为SDCARD，开机启动。

七、设备树

设备树用户需修改文件位于petalinux工程如下位置：

</petalinux67dr/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi>

针对基本驱动，SD（EMMC或者SD卡）以及USB3.0接口，设备树内容修改如下：

```
/include/ "system-conf.dtsi"
/ {
};

/* SD */
&sdhci1 {
        disable-wp;
        no-1-8-v;
};

/* USB  */
&dwc3_0 {
    status = "okay";
    dr_mode = "host";
};
```

八、USB3.0 SLAVE测试

针对USB3.0 SLAVE，上一节设备树中dr_mode = "host"；需修改为

dr_mode = "peripheral";

我们将TF卡格式化为FAT32之后，复制<RF67DR_Lite开发板资料\04例程demo\Petalinux\02petalinux_usb_slave\编译好的启动文件>中的所有文件到TF卡，设置

SDCARD启动后，USB3.0数据线连接到电脑支持USB3.0的接口上，然后开发板串口输入运行如下指令：

```
mount /dev/mmcblk1p1 /mnt/
cd /mnt/
dd if=/dev/zero of=/tmp/test.img bs=1M count=512
insmod g_mass_storage.ko file=/tmp/test.img removable=1
```

然后电脑中会显示一个新盘符。将其格式化后，即可复制拷贝文件做测试。数据线是3.0，且运行成功，盘符读写速度理论上大于100MBPS。建议使用雷电线（10G以上线材）保证稳定可靠。

petalinux-config -c kernel修改如下项确认被选中：

Device Drivers---> USB Support---> USB Gadget Support -->Serial Gadget

编译后，cd到image->linux中使用如下指令打包启动文件：

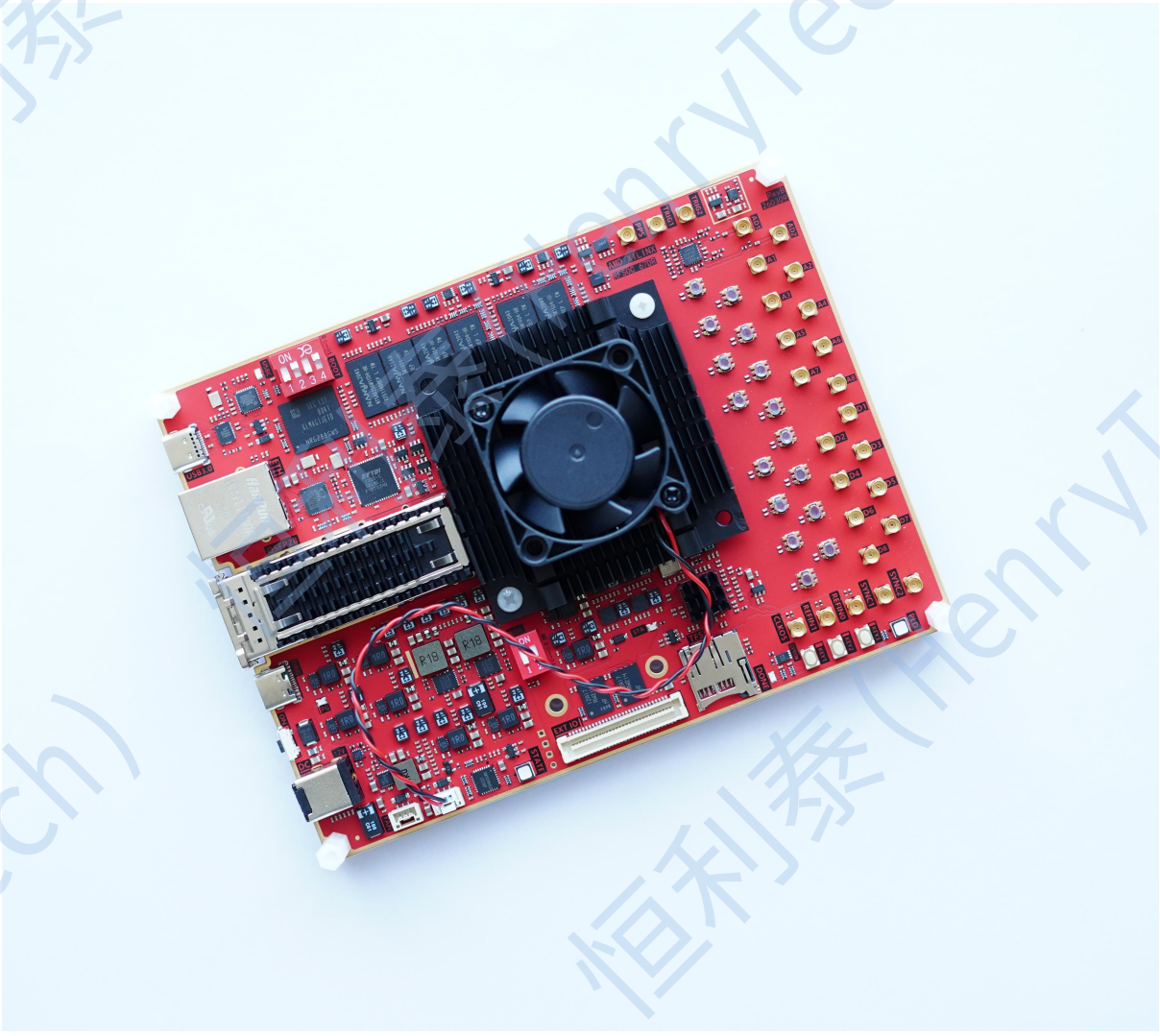
petalinux-package --boot --u-boot --fpga --force

上述路径执行后生成BOOT.BIN\boot.scr\image.ub。

g_ether.ko\g_mass_storage.ko\g_serial.ko这几个文件可以在petalinux工程中搜索到，复制出来即可。

光口开发之Ibert眼图及光通信测试

------ 基于RF67DR-Lite开发板套件



恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 眼图简介 3
三、 VIVADO工程新建 8
四、 IP调用参数配置 8
五、 IP例子生成和代码修改 11
六、 绑定引脚生成BIT文件 13
七、 硬件连接 14
八、 链路状态查看 15
九、 眼图扫描 17
十、 如何使用官方的GTY收发器IP核例子 19

恒利泰(HenryTech)

一、文档实现功能介绍

本文档实现在 VIVADO 下面新建一个 FPGA 开发工程，然后FPGA开发调用Ibert的IP核，实现使用赛灵思提供的Ibert的IP进行GTY收发器通过光纤传输的收发眼图测试,我们使用到开发板的100G QSFP光口。通过眼图测试可以对硬件和收发器的信号质量评估。比如数据错误率，以及开口面积等，以及如何使用GTY收发器IP核配置使用IP的官方例子进行通信学习测试。

工程新建方法请参考文档如何新建工程参考《04VIVADO新建工程流程和软件介绍》；

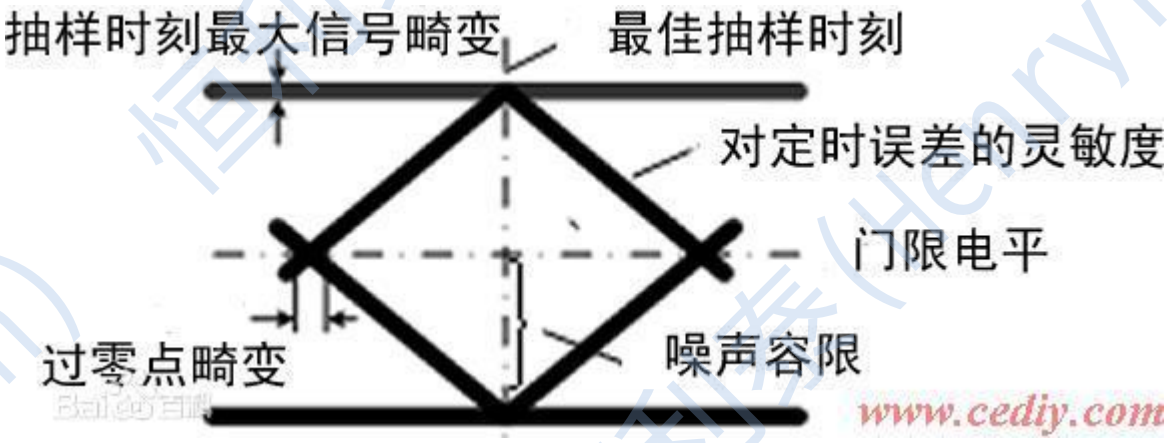
配套使用的光模块型号和光纤回环模块在文档后面上板测试会有介绍。

二、眼图简介

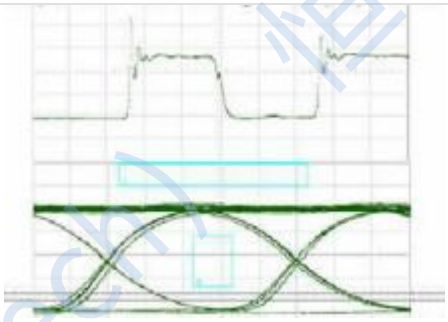
以下一段内容摘自百度百科的一部分：

眼图，是由于示波器的余辉作用，将扫描所得的每一个码元波形重叠在一起，从而形成眼图。其是指利用实验的方法估计和改善（通过调整）传输系统性能时在示波器上观察到的一种图形。观察眼图的方法是：用一个示波器跨接在接收滤波器的输出端，然后调整示波器扫描周期，使示波器水平扫描周期与接收码元的周期同步，这时示波器屏幕上看到的图形像人的眼睛，故称 为 “眼图”。

眼图中包含了丰富的信息，从眼图上可以观察出码间串扰和噪声的影响，体现了数字信号整体的特征，从而可以估计系统优劣程度，因而眼图分析是高速互连系统信号完整性分析的核心。另外也可以用此图形对接收滤波器的特性加以调整，以减小码间串扰，改善系统的传输性能。

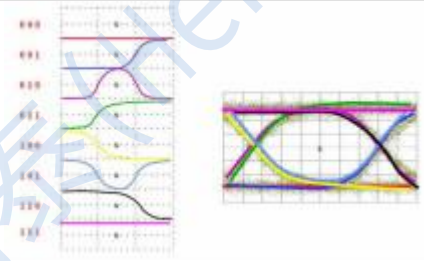


一般均可以用示波器观测到信号的眼图，其具体的操作方法为：将示波器跨接在接收滤波器的输出端，然后调整示波器扫描周期，使示波器水平扫描周期与接收码元的周期同步，这时示波器屏幕上看到的图形就称为眼图。示波器一般测量的信号是一些位或某一段时间的波形，更多的反映的是细节信息，而眼图则反映的是链路上传输的所有数字信号的整体特征，两者对比如下图所示：[1]

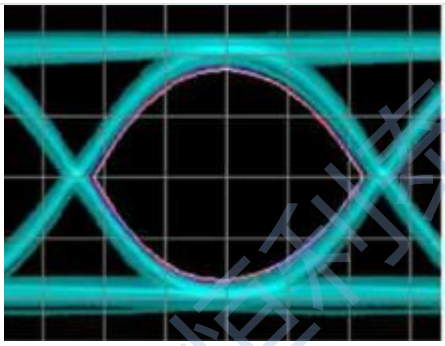


如果示波器的整个显示屏幕宽度为100ns，则表示在示波器的有效频宽、取样率及记忆体配合下，得到了100ns下的波形资料。但是，对于一个系统而言，分析这么短的时间内的信号并不具有代表性，例如信号在每一百万位元会出现一次突波（Spike），但在这100ns时间内，突波出现的机率很小，因此会错过某些重要的信息。如果要衡量整个系统的性能，这么短的时间内测量得到的数据显然是不够的。设想，如果可以以重复叠加的方式，将新的信号不断的加入显示屏幕中，但却仍然记录着前次的波形，只要累积时间够久，就可以形成眼图，从而可以了解到整个系统的性能，如串扰、噪声以及其他的一些参数，为整个系统性能的改善提供依据。

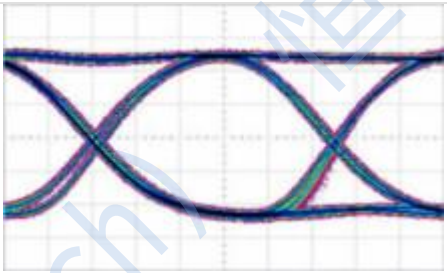
分析实际眼图，再结合理论，一个完整的眼图应该包含从“000”到“111”的所有状态组，且每一个状态组发生的次数要尽量一致，否则有些信息将无法呈现在屏幕上，八种状态形成的眼图如下所示：



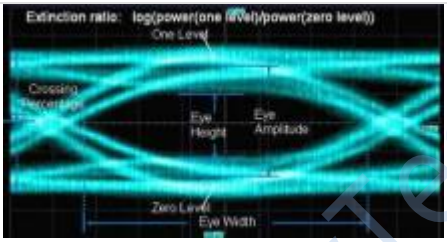
由上述的理论分析，结合示波器实际眼图的生成原理，可以知道一般在示波器上观测到的眼图与理论分析得到的眼图大致接近（无串扰等影响），如下所示：



如果这八种状态组中缺失某种状态，得到的眼图会不完整，如下所示：



眼图中通常显示的是1.25UI 的时间窗口，眼图的形状各种各样，通过眼图的形状特点可以快速地判断信号的质量。可以根据眼图的相关参数来判别眼图的好坏，从而可以衡量系统的性能。眼图相关的参数有很多，如眼高、眼宽、眼幅度、眼交叉比、“1”电平，“0”电平，消光比，Q 因子，平均功率等，各个参数如下图中所示：



眼图中的“1”电平（ top P）与“0”（ base P ）电平即是表示逻辑为1 或0 的电压位准值，实际中选取眼图中间的20% UI 部分向垂直轴投影做直方图，直方图的中心值分别为“1”电平和“0”电平。

眼幅度表示“1”电平信号分布与“0”电平信号分布平均数之差，其测量是通过在眼图中央位置附近区域（通常为零点交叉时间之间距离的20%）分布振幅值进行的。

眼宽反映信号的总抖动，即是眼图在水平轴所开的大小，其定义为两上缘与下缘交汇的点（Crossing Point）间的时间差。交叉点之间的时间是基于信号中的两个零交叉点处的直方图平均数计算而来，每个分布的标准偏差是从两个平均数之间的差值相减而来。

眼高即是眼图在垂直轴所开的大小，它是信噪比测量，与眼图振幅非常相似。

下面详细介绍一些复杂的概念，以帮忙理解眼图的性能。

（1）消光比（Extinction Ratio）消光比定义为眼图中“1”电平与“0”电平的统计平均的比值，其计算公式可以是如下的三种：

$$Ratio = \frac{P_{top}}{P_{base}}$$

$$Ratio\% = 100 \times \frac{P_{top}}{P_{base}}$$

$$Ratio = 100 \log \frac{P_{top}}{P_{base}} (dB)$$

消光比在光通信发射源的量测上是相当重要的参数，它的大小决定了通信信号的品质。消光比越大，代表在接收机端会有越好的逻辑鉴别率；消光比越小，表示信号较易受到干扰，系统误码率会上升。

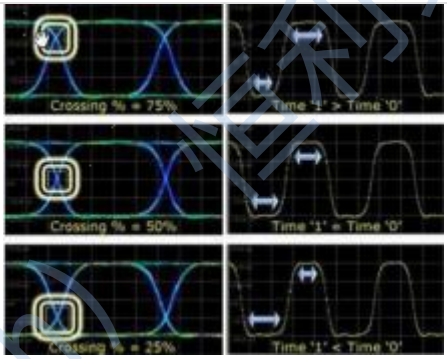
消光比直接影响光接收机的灵敏度，从提高接收机灵敏度的角度希望消光比尽可能大，有利于减少功率代价。但是，消光比也不是越大越好，如果消光比太大会使激光器的图案相关抖动增加。因此，一般建议实际消光比与最低要求消光比大 0.5~1.5dB。

（2）眼交叉比

眼图交叉比，是测量交叉点振幅与信号“1”及“0”位准之关系，因此不同交叉比例关系可传递不同信号位准。一般标准的信号其交叉比为50%，即表示信号“1”及“0”各占一半的位准。为了测量其相关比率，使用如下图所示的统计方式。交叉位准是依据交叉点垂直统计的中心窗口而计算出来的平均值，其比例方程式如下（其中的1 及0 位准是取眼图中间的20%为其平均值，即从40%~60%中作换算）：

$$100 \times \frac{\text{交叉位准} - 0\text{位准}}{1\text{位准} - 0\text{位准}}$$

随着交叉点比例关系的不同，表示不同的信号1 或0 传递质量的能耐。如下图所示，左边图形为不同交叉比例关系的眼图，对应到右边相关的1 及0 脉冲信号。同时也可以了解到在不同脉冲信号时间的宽度与图交叉比例的关系。



对于一般的信号而言，平均分布信号位准1 及0 是最常见的。一般要求眼图交叉比为50%，即以相同的信号脉冲1 与0 长度为标准，来作相关参数的验证。因此，根据眼交叉比关系的分布，可以有效地测量因不同1 及0 信号位准的偏差所造成的相对振幅损失分析。例如，眼交叉比过大，即传递过多1 位准信号，将会依此交叉比关系来验证信号误码、屏蔽及其极限值。眼交叉比过小，即传递过多0 位准信号，一般容易造成接收端信号不易从其中抽取频率，导致无法同步，进而产生同步损失。

（3）信号上升时间与下降时间

一般测量上升及下降时间是以眼图占20%～80%的部分为主，其中上升时间如下图，分别以左侧交叉点左侧(20%)至右侧(80%)两块水平区间作此传递信号上升斜率时间之换算，计算公式如下：

上升时间=平均（80%时间位准）-平均（20%时间位准）

由于时间位准20%及80%是与信号位准1 及0 有着相关性的。当然，如果上升时间愈短，即愈能表现出眼图中间的白色区块，即代表可传递的信号及容忍误码比率较好。

（4）Q 因子（Q Factor）

Q 因子是用于测量眼图信噪比的参数，它的定义是接收机在最佳判决门限下信号功率和噪声功率的比值，可适用于各种信号格式和速率的数字信号，其计算公式如下：

$$Q_{factor} = \frac{P_{top} - P_{base}}{\sigma_1 + \sigma_0}$$

其中，“1”电平的平均值top P与“0”电平的平均值base P 的差为眼幅度，“1”信号噪声有效值1s与“0”信号噪声有效值0 s 之和为信号噪声有效值。Q 因子综合反映眼图的质量问题。Q 因子越高，眼图的质量就越好，信噪比就越高。Q 因子一般受噪声、光功率、电信号是否从始端到终端阻抗匹配等因素影响。一般来说，眼图中1 电平的这条线越细、越平滑，Q 因子越高。在不加光衰减的情况下，发送侧光眼图的Q 因子不应该小于12，接收测的Q 因子不应该小于6 。

（5）平均功率通过眼图反映的平均功率，即是整个数据流的平均值。与眼图振幅测量不同，平均功率则是直方图的平均值。如果数据编码正常工作，平均功率应为总眼图振幅的50%。

（6）抖动抖动是在高速数据传输线中导致误码的定时噪声。如果系统的数据速率提高，在几秒内测得的抖动幅度会大体不变，但在位周期的几分之一时间内测量时，它会随着数据速率成比例提高，进而导致误码。因此，在系统中尽可能的减少这种相关抖动，提升系统总体性能。

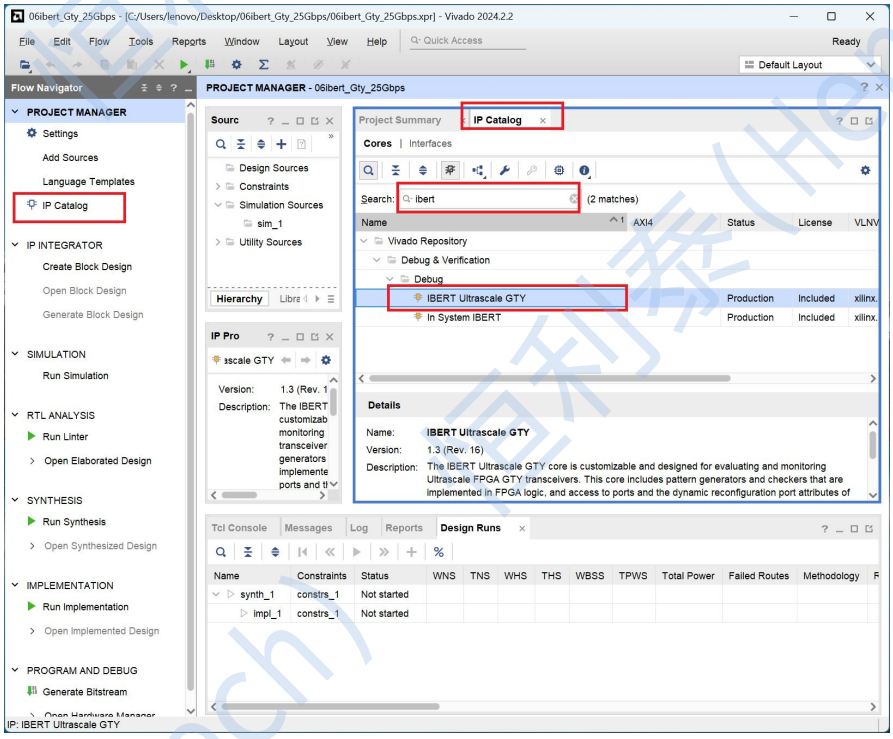
三、VIVADO工程新建

工程新建步骤和配置如下：

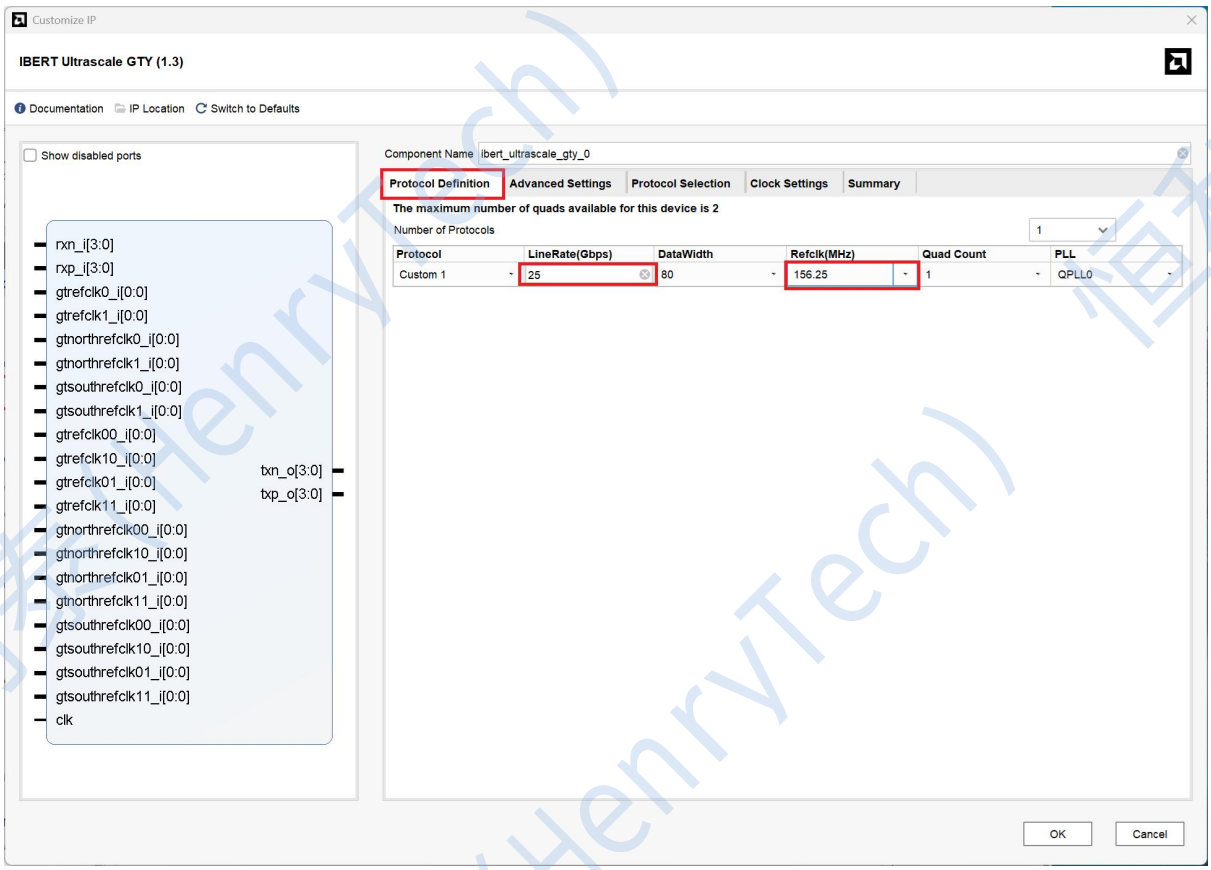
起始页（或 file->Project->New）	创建新工程(Create New Project)
向导起始页面	点击 Next->
Project Name(工程名)	工程名：06ibert_Gty_25Gbps 工程路径：(自己选择，尽量不要有中文路径) 勾选 Create Project Subdirectory,点击 Next->
Add Source(添加设计源文件)	点击 Next->
Add Exsixting IP(添加已有的 IP)	点击 Next->
Add Constraints(添加已有约束文件)	点击 Next->
Default Part(默认配置，芯片选型)	Family->Zynq UltraScale_RFSoCs Package->FFVE1156 Speed->-2 按具体型号选择 XCZU67DR-FFVE1156-2-i
New Project Summary(新建工程概况)	确认工程信息，选型等，点击 Finish 完成

四、IP调用参数配置

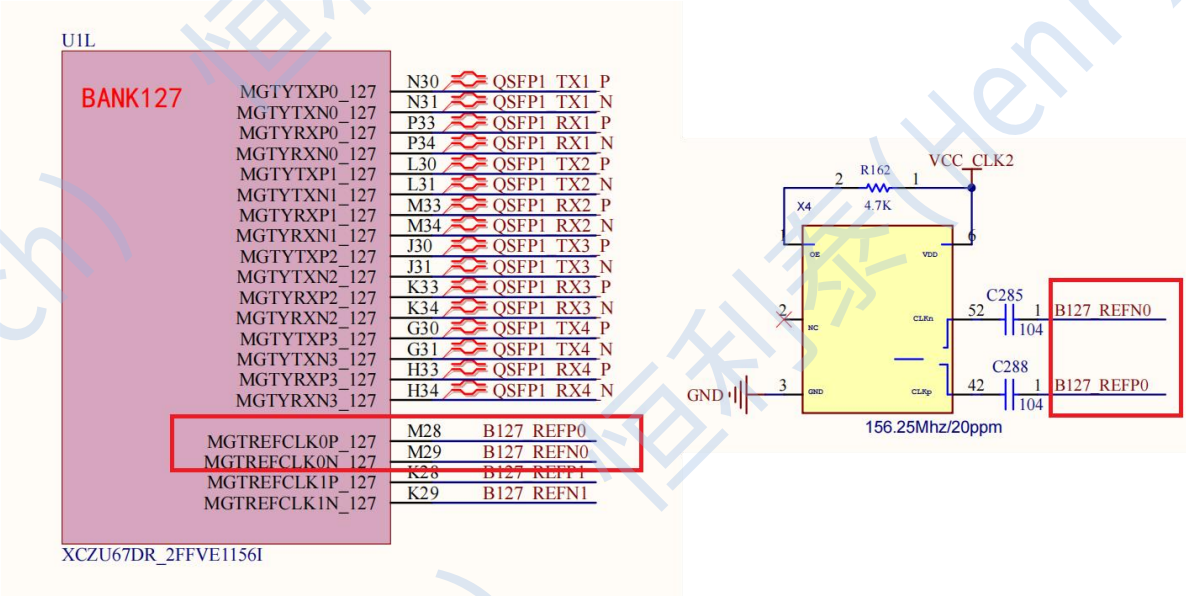
我们点击IP Catalog调用IP。我们搜索栏输入ibert，找到如下IP核然后双击配置调用：



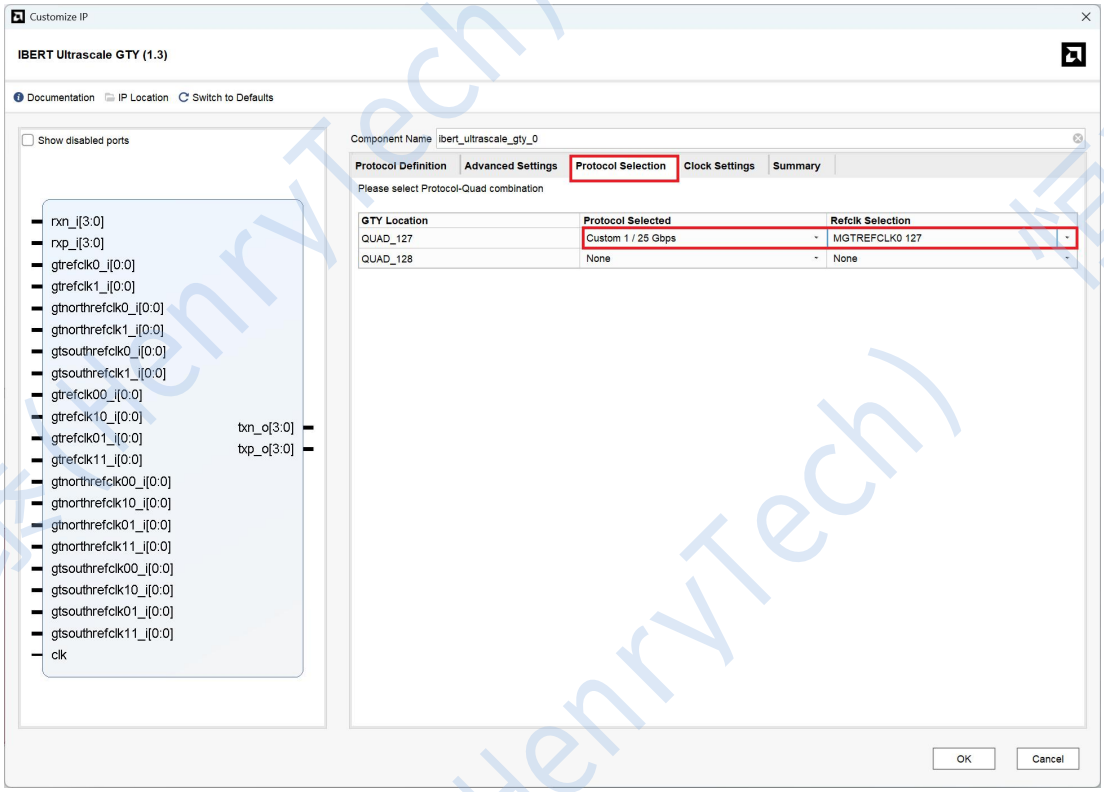
配置中第一页protocal definition,我们配置速率，以及参考时钟，我们使用了板子上的一个GTY的156.25M差分时钟作为参考：



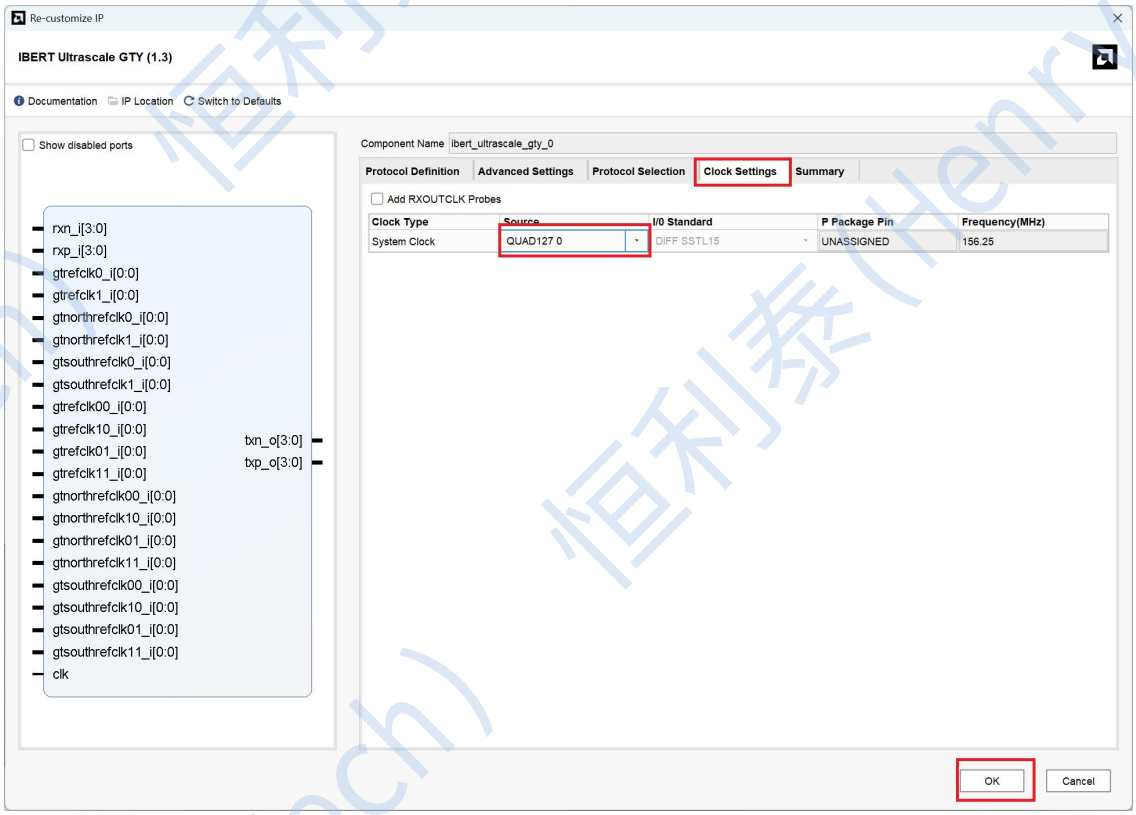
在配置第三页之前我们先看一下原理图的光口连接的哪个BANK，以及时钟信号用的哪一对参考输入，从原理图的第十一部分BANK127上，REF0引脚的标注B127_REFP0 N0是连接到原理图第十九部分的X4晶振，156.25M上：



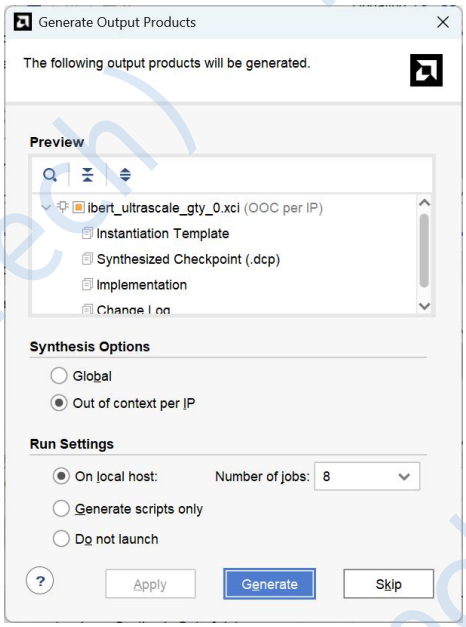
我们第三页Protocal selection,看一下配置，我们开发板光口和156.25M时钟连接到我们的BANK127上，时钟连接到我们的REF0差分输入引脚，所以我们配置如下,只选择QUAD_127：



第四页clock settings，时钟source用我们的BANK127的这个QUAD127，GTY的PLL同时可以用作逻辑时钟输出,不用额外输入时钟，配好后我们点击OK完成IP配置：



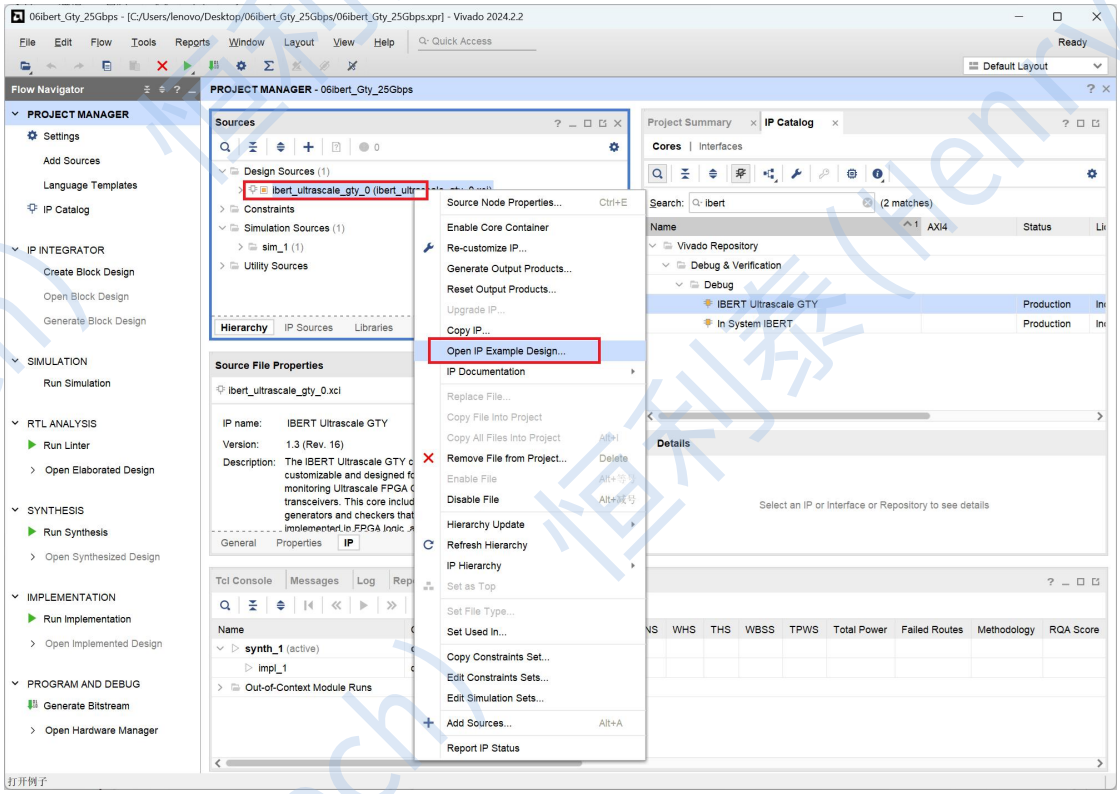
弹出生成IP，点击generate即可：



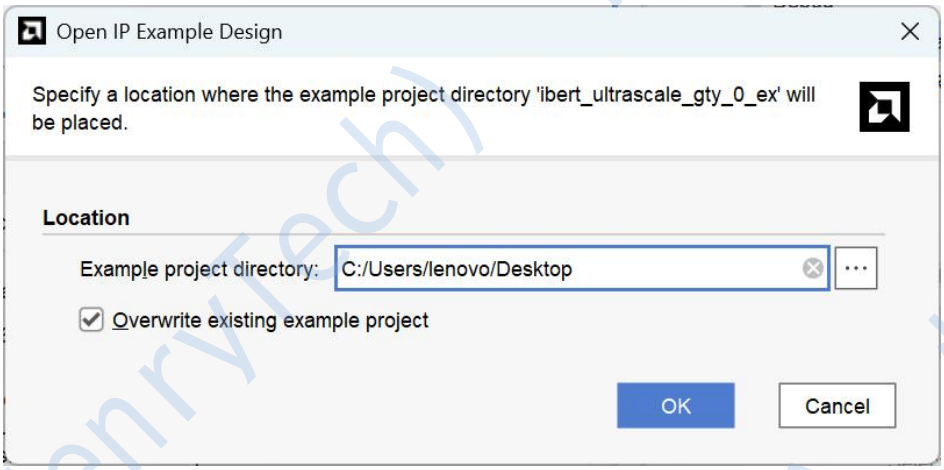
接着生成输出等等一系列操作，然后VIVADO调用IP会自动综合。我们等待IP生成完成。生成完成并且综合完，VIVADO右上角显示READY就表示IP调用生成成功。

五、IP例子生成和代码修改

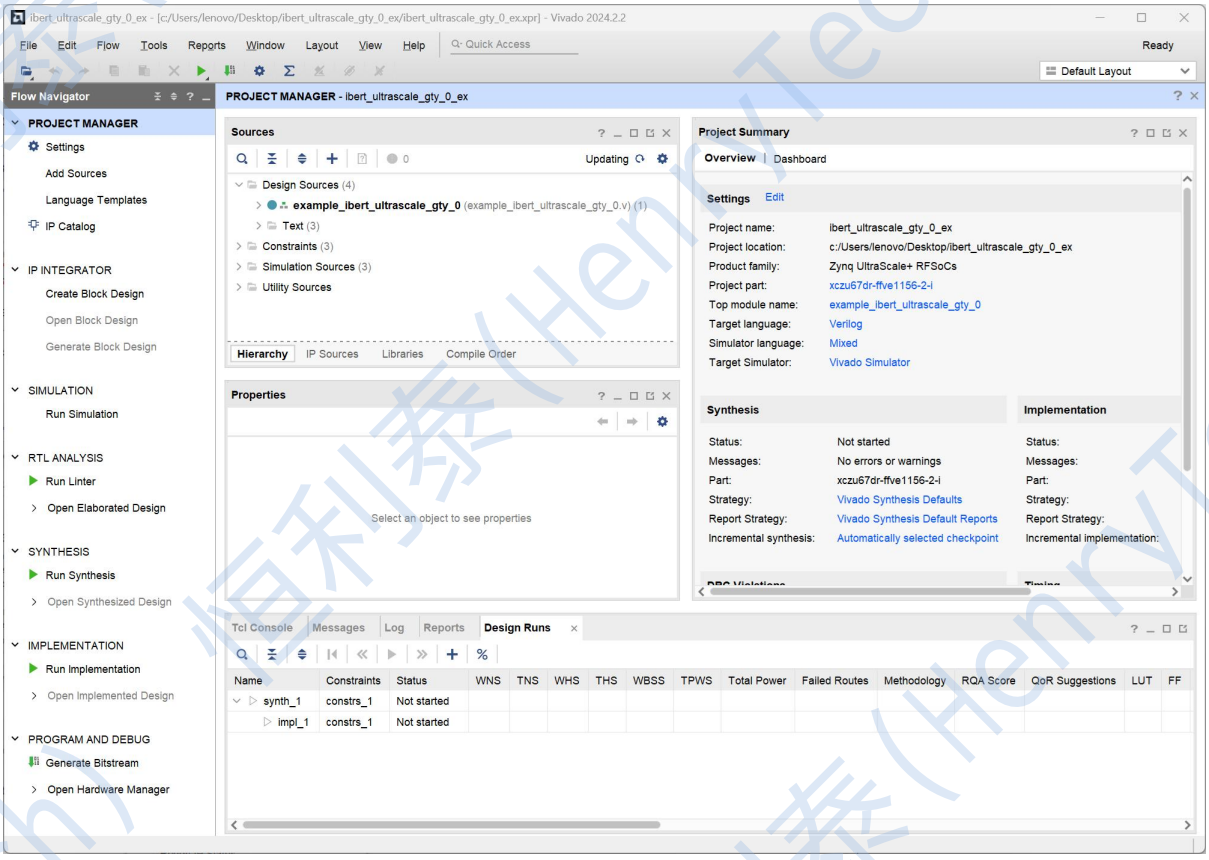
接下来我们使用IP生成例子然后做些修改使用。IP调用后软件右上角ready显示，我们选中IP右键打开IP例子：



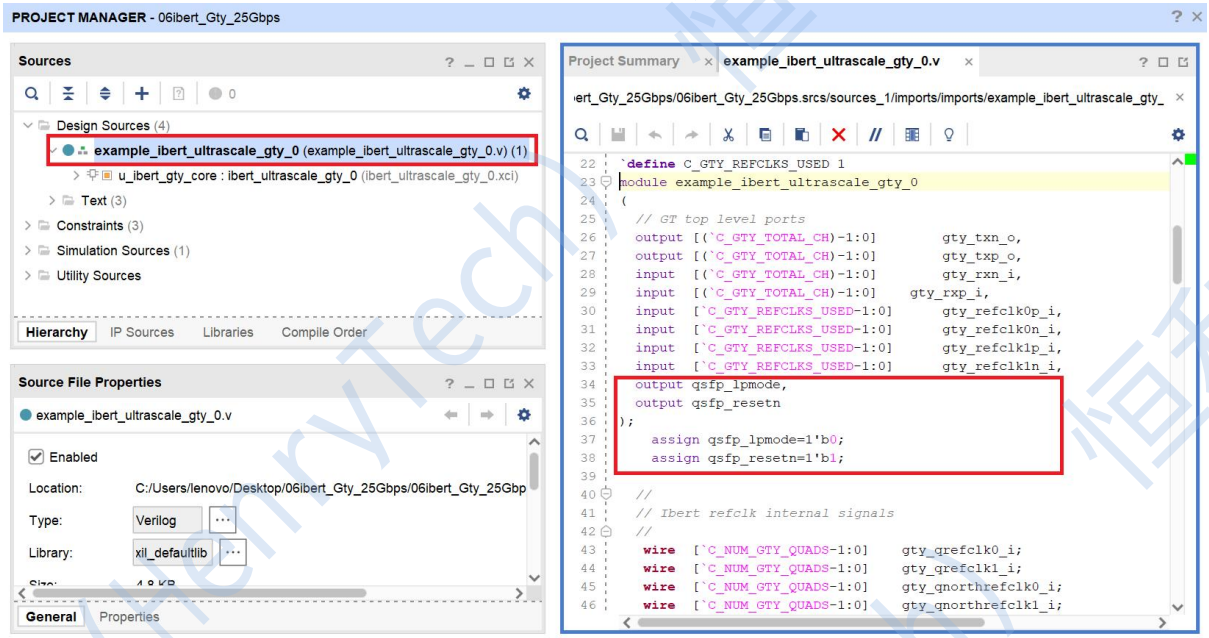
例子存放工程位置我们自己指定：



然后我们等待打开例子工程，时间比较长，耐心等待，打开的工程如下：

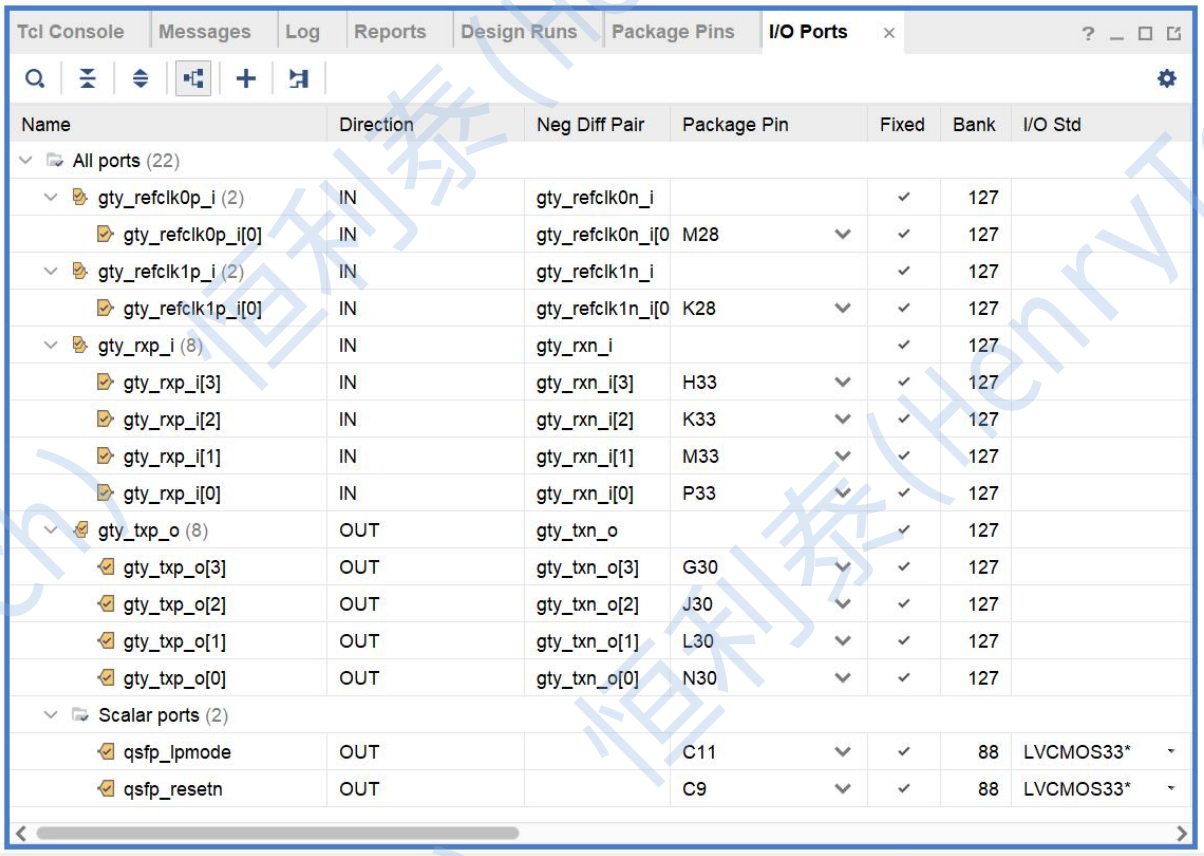


我们打开顶层代码，在顶层信号列表添加两个信号，qsfp_lpmode和qsfp_resetn，一个用于控制模块正常工作模式，一个用于模块复位信号，正常工作模式下lpmode信号需要拉低，resetn需要拉高，其余信号不用管，我们只做简单的光模块通信连路测试：



六、绑定引脚生成BIT文件

我们将代码综合后绑定引脚。引脚绑定如下，请注意，GTY高速信号引脚不需要自己绑定电平标准，并且差分对只需要绑定一个信号，另一个自动分配固定：



Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (22)						
gty_refclk0p_i (2)	IN	gty_refclk0n_i		✓	127	
gty_refclk0p_i[0]	IN	gty_refclk0n_i[0	M28	✓	127	
gty_refclk1p_i (2)	IN	gty_refclk1n_i		✓	127	
gty_refclk1p_i[0]	IN	gty_refclk1n_i[0	K28	✓	127	
gty_rxp_i (8)	IN	gty_rxn_i		✓	127	
gty_rxp_i[3]	IN	gty_rxn_i[3]	H33	✓	127	
gty_rxp_i[2]	IN	gty_rxn_i[2]	K33	✓	127	
gty_rxp_i[1]	IN	gty_rxn_i[1]	M33	✓	127	
gty_rxp_i[0]	IN	gty_rxn_i[0]	P33	✓	127	
gty_txp_o (8)	OUT	gty_txn_o		✓	127	
gty_txp_o[3]	OUT	gty_txn_o[3]	G30	✓	127	
gty_txp_o[2]	OUT	gty_txn_o[2]	J30	✓	127	
gty_txp_o[1]	OUT	gty_txn_o[1]	L30	✓	127	
gty_txp_o[0]	OUT	gty_txn_o[0]	N30	✓	127	
Scalar ports (2)						
qsfp_lpmode	OUT		C11	✓	88	LVCMOS33*
qsfp_resetn	OUT		C9	✓	88	LVCMOS33*

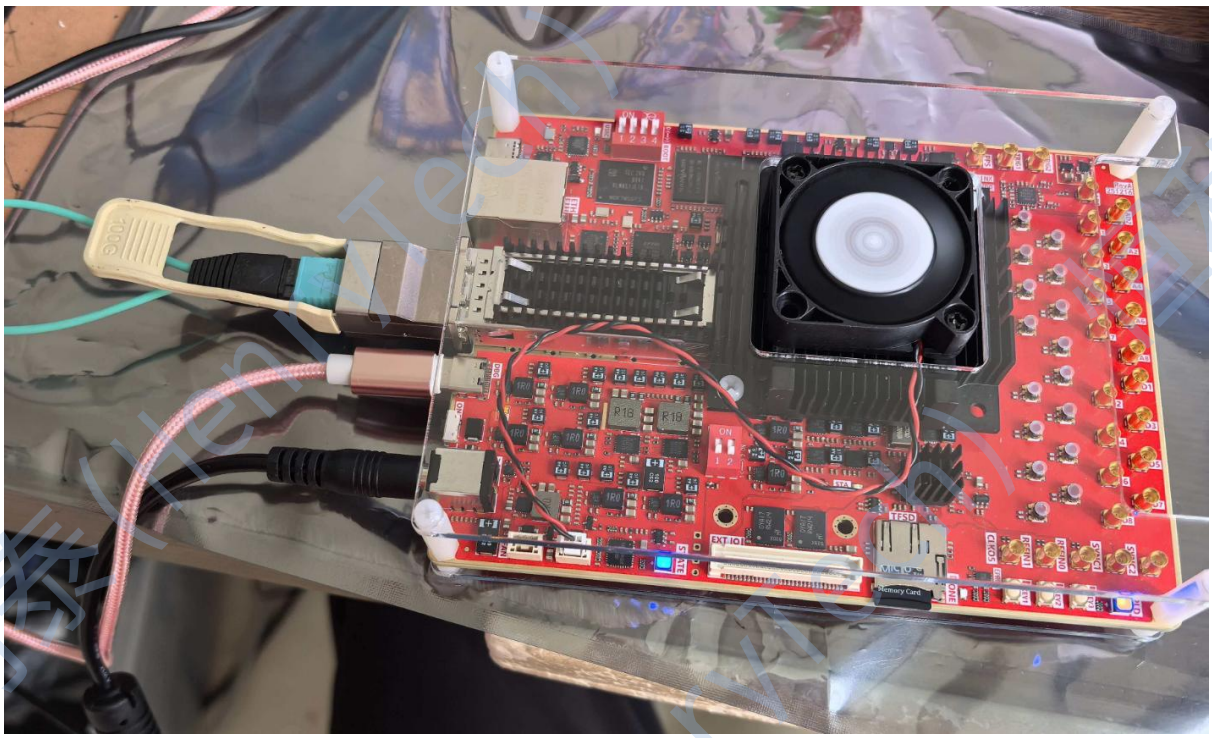
七、硬件连接

我们准备一只QSFP支持100G（QSFP28,单路lane最高支持28Gbps,QSFP为4lane所以总带宽超过100G）的光模块，型号是**HZ-QSFP28-100G-SR4**，推荐型号实物如下：

除了光模块，我们还需要一个回路器模块，将QSFP发出的信号环回接到接收端。我们QSFP模块包含4对发送，4对接收通道，所以，对于QSFP接口所使用的光纤里面是8根芯。我们环路器也是将4收4发做个回环。环路器全名叫做MPO loopback（又叫MPO回路器，MPO光纤回路器等）模块，模块有很多种规格，有8芯，12芯，或者其他，我们买8芯即可，模块外形如下，价格大概100rmb左右：

或者使用外部带有光纤环回的模块：

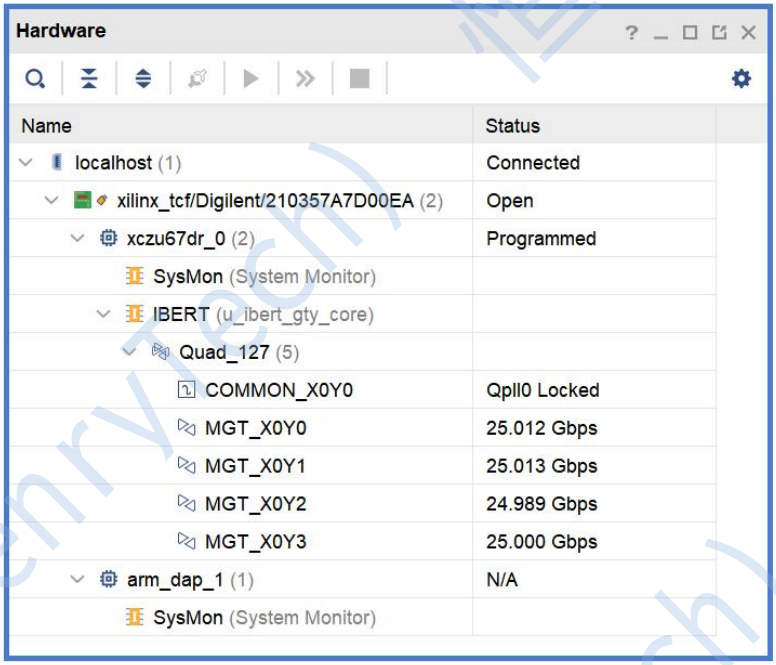
准备好后我们插到板子上,模块和MPO环路器都是有方向的（环路器上面还有盖子，记得取下来再插上），大家插上的时候注意下，插到位了会听到咔嚓一身卡住了：



模块拔的时候拉住黄色的拉手往外拉即可。MPO环路器扣住蓝色套壳往外拉可以分离模块和环路器，注意环路器插入也是有方向的。插好连接DEBUG口到电脑，开机上电，然后我们烧写BIT到板子上。

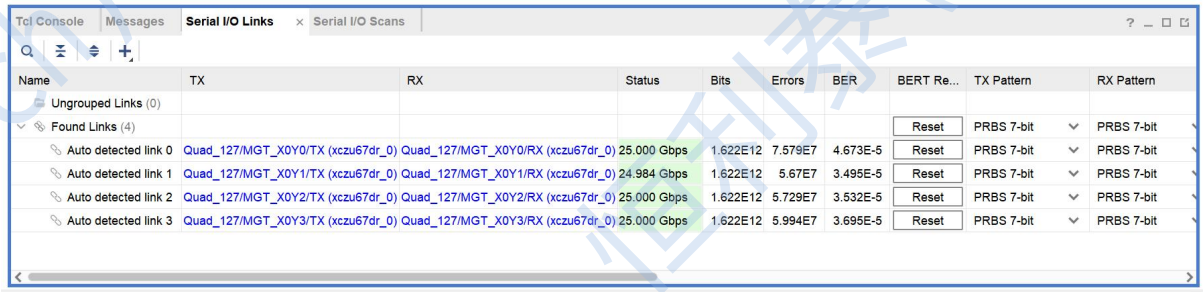
八、链路状态查看

我们下载BIT后，我们硬件界面就可以看到我们的IBERT的连接了，可以看到我们的后面有四个通道收发器后面显示25G的速率（这里可能速率会小范围波动，属于正常），说明这四个收发器链路已经在通信了，并且可以看到COMMON_X0Y1的QPLL已经Locked状态：



我们找到下方窗口的Serial I/O Links，点击中间的Auto detect links检测连接：

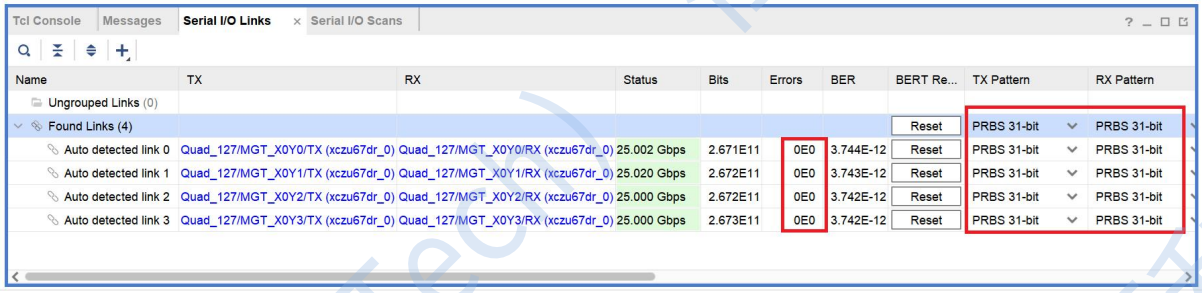
点击后，可以看到我们四个连接的状态信息，和数据收发信息，其中包括已发送的数据，当前通信速率，以及Erros错误计数，这里由于下载BIT文件后链路还没稳定，刚开始通信可能会出现少量误码，正常现象，大家可以点击右边BERT Reset复位一下这一路，就可以清零从头开始，然后就不会再出现误码或者很久才有误码：



可以看到我们速率可能在小范围，千分之一位波动，传输数据随着时间在增大，注意，这里单位是E，是10的多少次方，表示当前已经传输了多少数据。

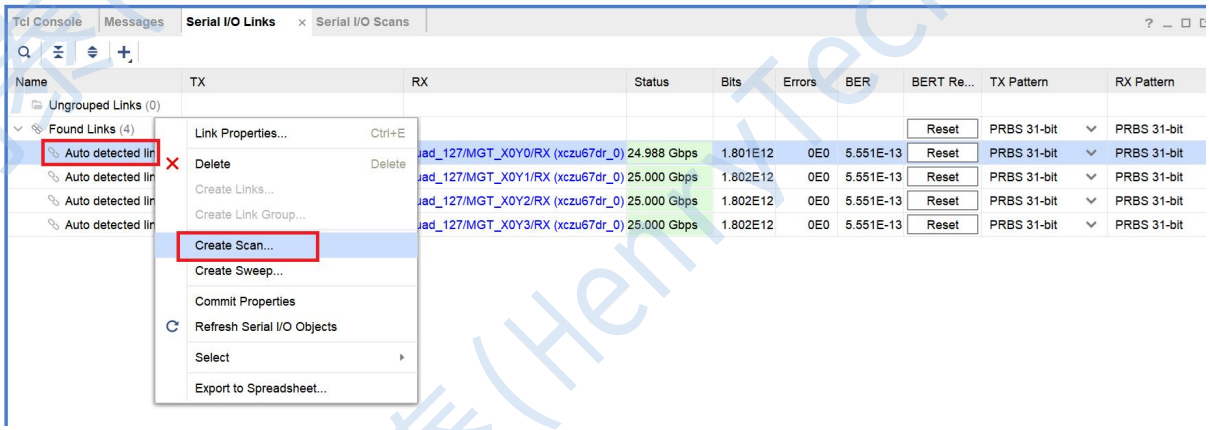
默认传输的TX RX Ppattern为7bit模式，我们可以将TX RX改为31再看数据误码。注意TX 和RX pattern都改为31bit之后，点击一下Reset复位清理一下，这个时候持续一段时间没

有出现误码或者极少量错误，说明链路稳定：

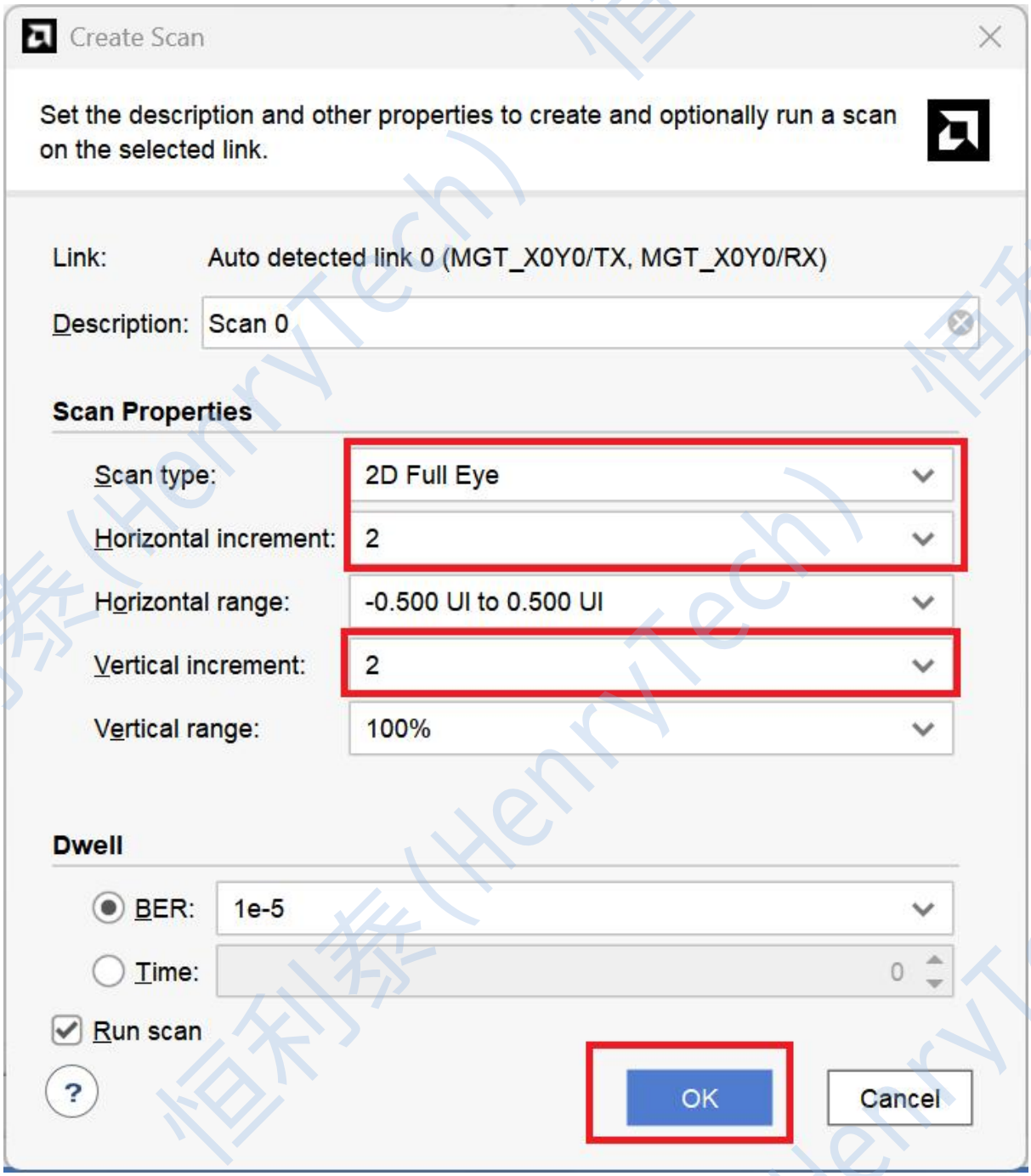


九、眼图扫描

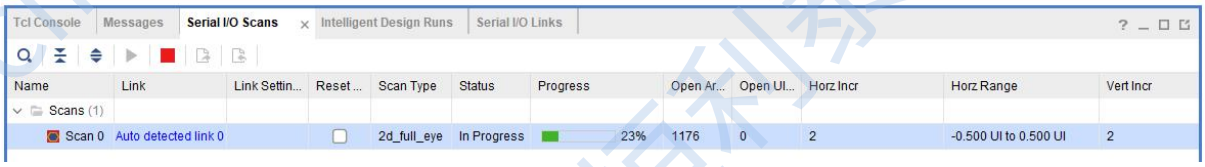
我们接下来对四个通道进行眼图测试。我们在Serial I/O Links选中其中一个通道右键选择Create scan，创建扫描：



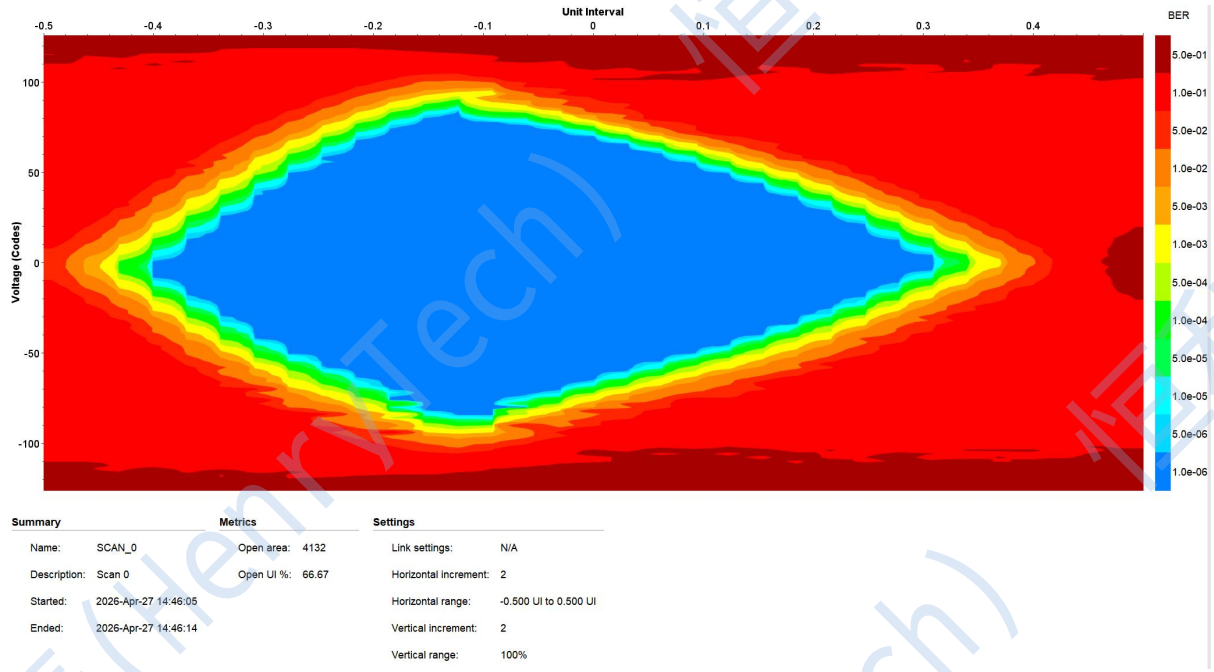
弹出配置界面如下配置，配置为2D扫描图，垂直和水平增量都设置为2，扫描出来的图更加细腻更加清晰，然后我们点击OK开始扫描：



我们扫描过程可以在下面的Serial IO Scan界面看到进度：



扫描完成后，我们上方可以看到扫描眼图结果了：



眼图可以看到，我们Open UI为百分之66.67。眼图清晰，说明通信链路稳定。同样的方法可以扫描其他三路，扫描结果如下：

Scan 0	Auto detected link 0	☐	2d_full_eye	Done	100%	4132	66.67	2
Scan 1	Auto detected link 1	☐	2d_full_eye	Done	100%	3848	63.64	2
Scan 2	Auto detected link 2	☐	2d_full_eye	Done	100%	3960	66.67	2
Scan 3	Auto detected link 3	☐	2d_full_eye	Done	100%	4000	66.67	2

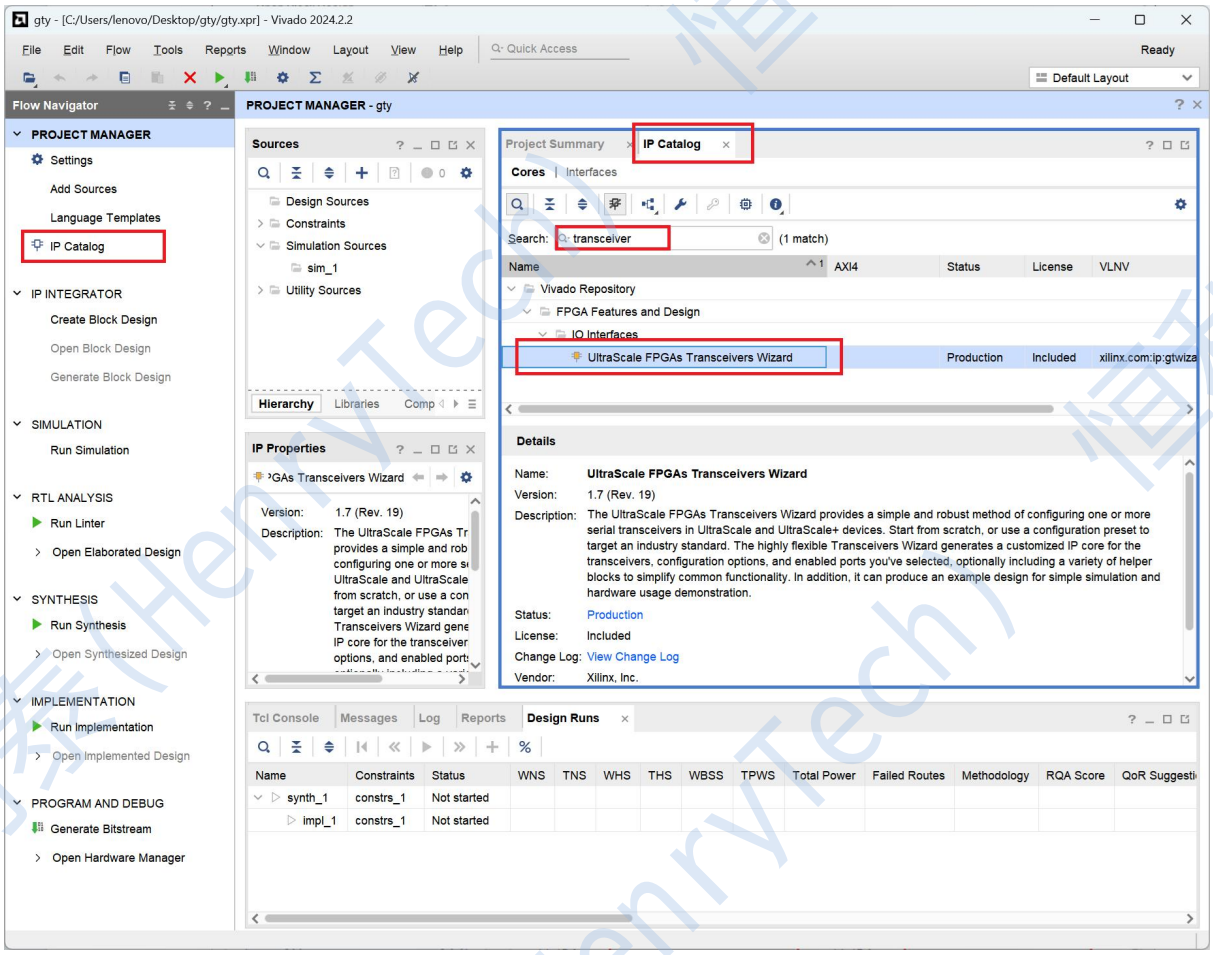
可以看到几乎每一路开口都接近百分之66左右UI，链路稳定。每一路性能差异有一些区别，由于光模块自身每一路性能，PCB板的走线，以及光纤等等都有一定影响。但是记住一点，PRBS31模式测试无误码，就说明链路稳定。我们TX RX Pattern模式对眼图影响比较大。我们将TX RX的Pattern都改为7Bit默认的参数后再次每一路扫描结果如下，眼图开口有明显的本质提升：

Scans (4)								
Scan 0	Auto detected link 0	☐	2d_full_eye	Done	100%	4824	72.73	2
Scan 1	Auto detected link 1	☐	2d_full_eye	Done	100%	4572	69.70	2
Scan 2	Auto detected link 2	☐	2d_full_eye	Done	100%	4616	78.79	2
Scan 3	Auto detected link 3	☐	2d_full_eye	Done	100%	4552	66.67	2

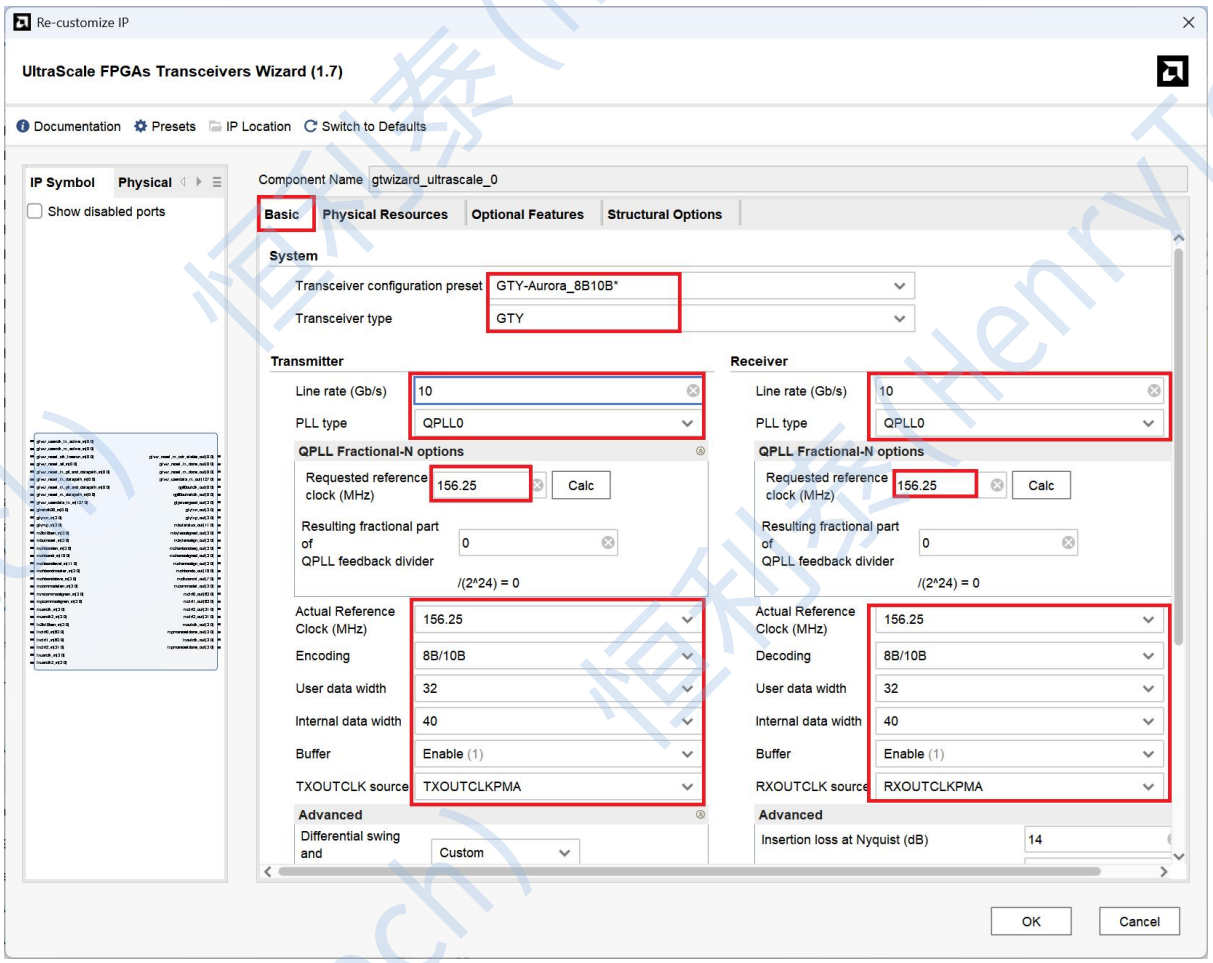
另外有时候眼图有问题，还有可能是接口的光纤损坏，或者接口有油污等引起，会影响通信质量。

十、如何使用官方的GTY收发器IP核例子

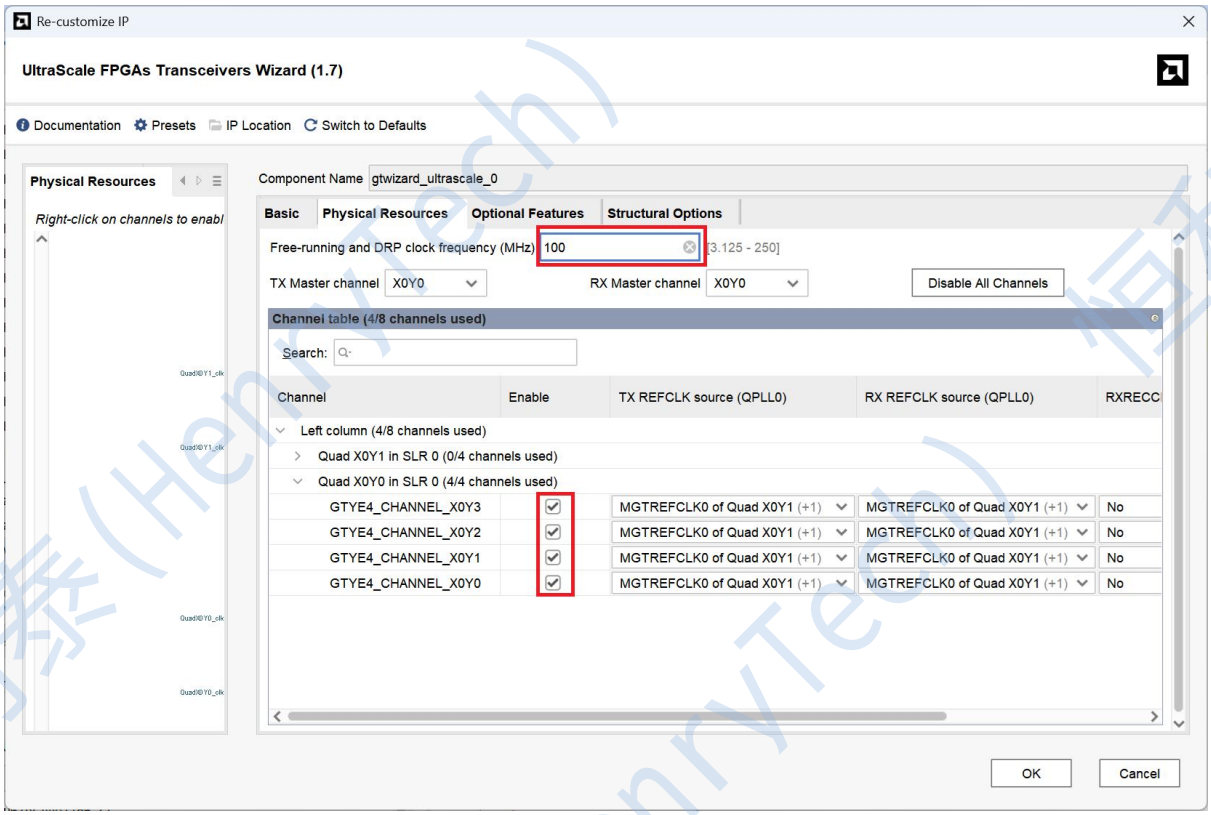
我们重新新建一个vivado工程，IP调用如下，搜索transceiver找到如下IP：



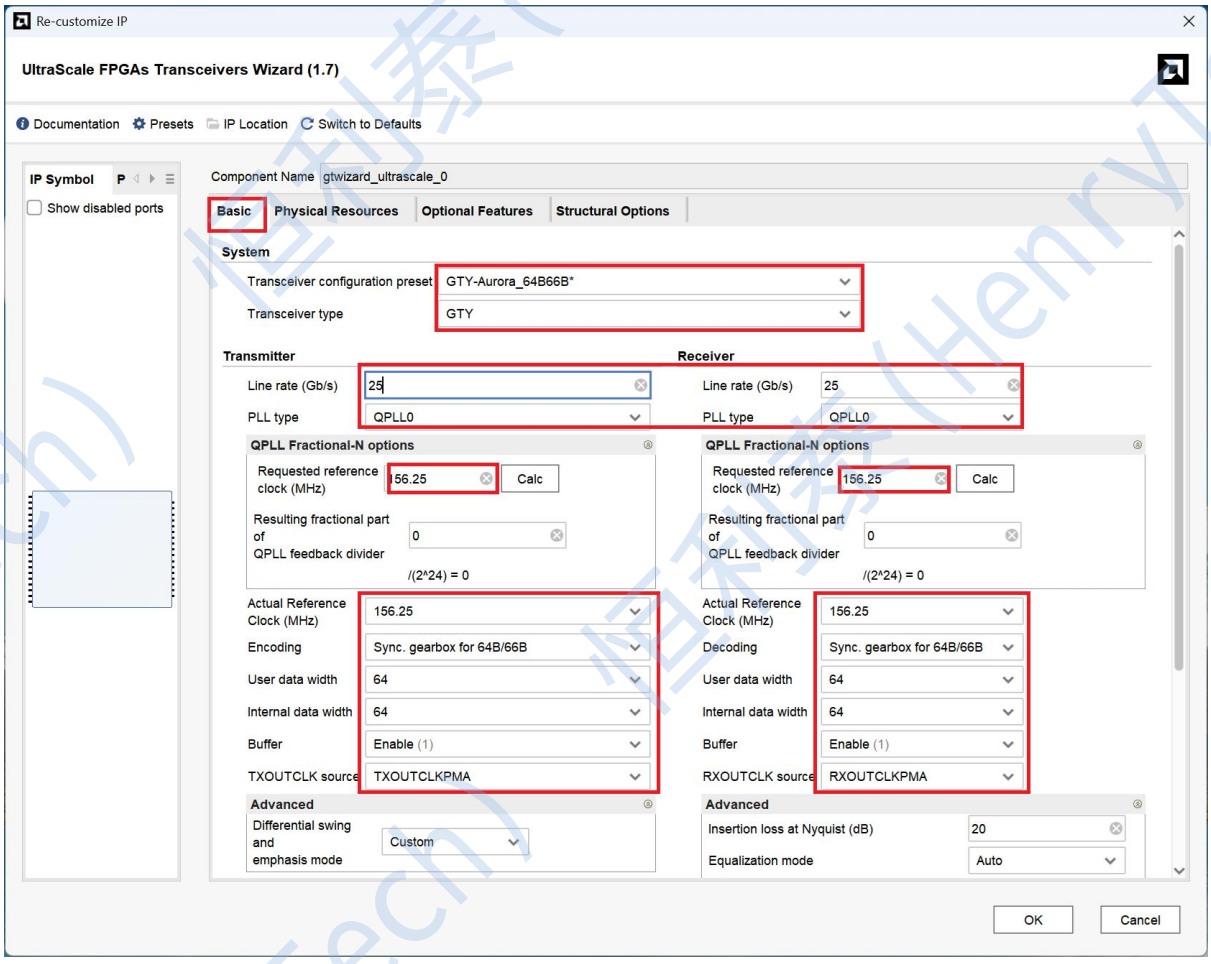
双击进入配置，10G的速率可以如下配置，第一页：



第二页，DRP时钟配置100M，X0Y0-Y3都勾上：



如果是25G可选配置如下，第一页：



同样的，配置完成点击OK，等待IP生成输出完成，source中找到IP右键生成例子即可。生成后我们可以对代码中收发的部分接口输入输出数据进行标记mark debug,来学习如何收发，这里我们不对官方例子进行研究，有兴趣的小伙伴可以自己解读官方IP example例子代码。我们生成的最终10G和25G的IP例子也在FPGA例程中提供了:

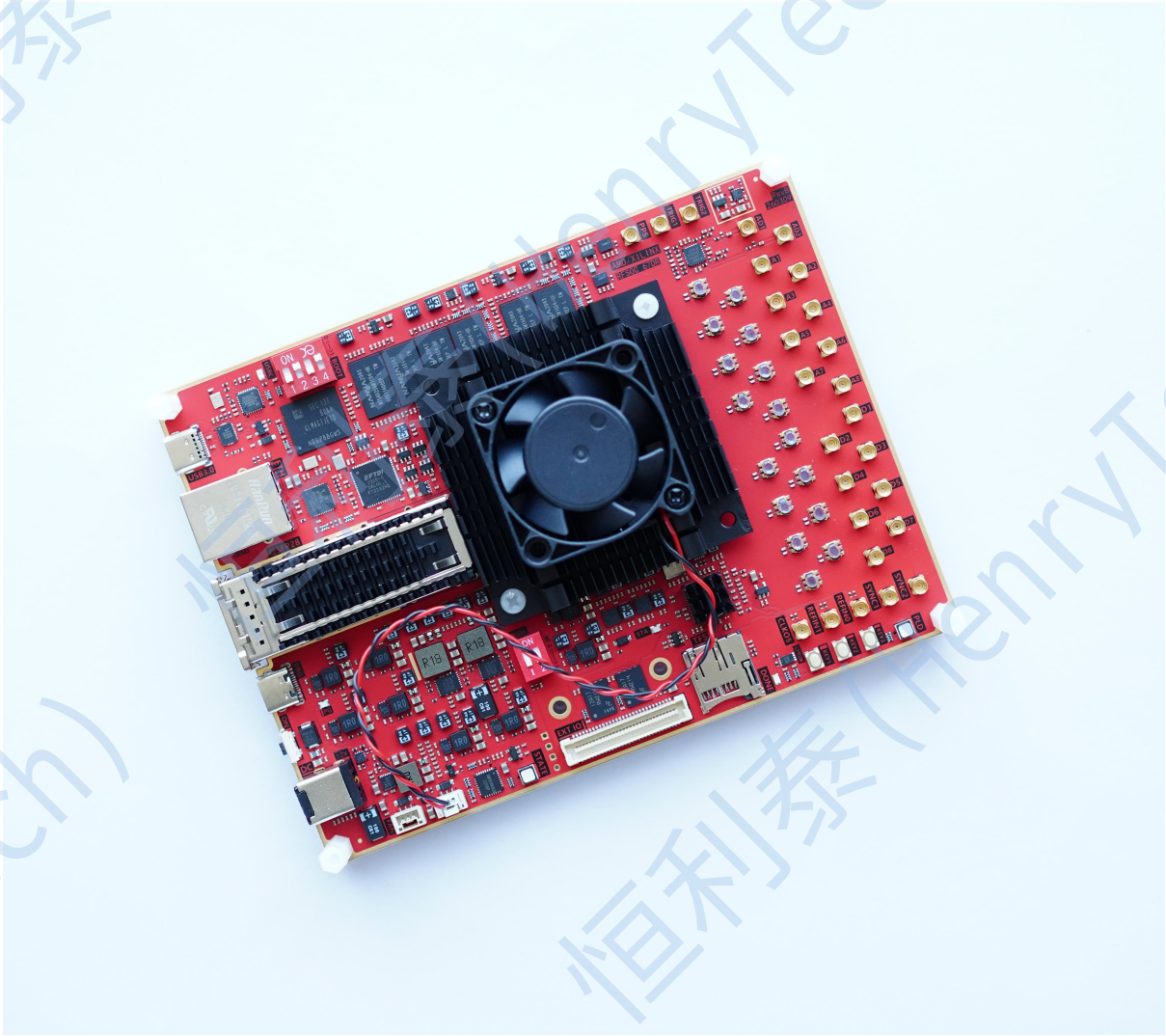
- 07qsfp_10G_loop
- 08qsfp_25G_loop
- 09qsfp_10g_data

注意，FPGA例子中这两个demo顶层也需要控制lpmode和resetn。0909qsfp_10g_data例子经过代码修改，实际传输测试数据。

恒利泰(HenryTech)

PS处理器配置及之helloword打印

------ 基于RF67DR-Lite开发板套件



恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 VIVADO的工程建立 3
三、 BD文档创建和ZYNQ的PS调用 4
四、 配置裁剪ZYNQ处理器外设IO 5
五、 ZYNQ 时钟配置 11
六、 ZYNQ DDR4配置 11
七、 添加AXI GPIO 12
八、 生成输出文件 16
九、 绑定引脚生成BIT文件 18
十、 导出硬件文件 19
十一、 VITIS导入硬件文件新建平台和工程 22
十二、 新建Platform Component 24
十三、 导入helloworld模板 27
十四、 运行调试代码 31
十五、 直接运行代码 35
十六、 可能的问题及解决方法 39

恒利泰(HenryTech)

一、文档实现功能介绍

本文档实现在 VIVADO 下面新建一个工程，然后调用ZYNQ IP，配置好我们的ZYNQ处理器ARM核功能参数接口，其中包括基本外设，存储接口及DDR4，网口和USB等，最后导出硬件，交由VITIS进行软件开发。通过vitis导入硬件，然后创建一个helloword的应用application工程，通过串口来输出我们的helloword。 工程新建方法请参考《04VIVADO新建工程流程和软件介绍》。我们习惯上把ZYNQ的FPGA部分叫做PL(Program Logical)，ZYNQ处理器部分叫做PS(Program System)，后续我们会用到这两个术语，行业里面也是这个简称。

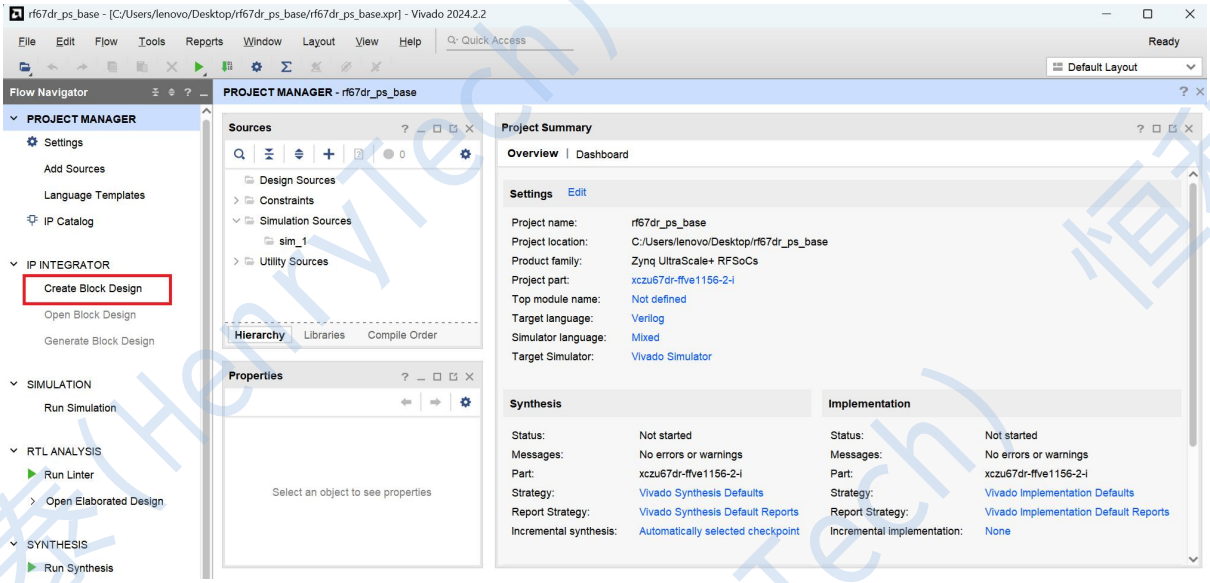
二、VIVADO的工程建立

工程新建步骤和配置如下,这里的vivado工程适用于我们后续只有zynq处理器，没有fpga逻辑部分的开发。比如后续的文件系统，rtos，sd卡或者flash启动，lwip，rtc等等例程都基于此。我们相关配置都在本helloword教程中教大家配置好相关的接口。后续不再赘述我们的PS配置，如果有其他的不一样的开发我们将再单独讲解，额外添加配置，比如运行linux的相关配置，需要配置更多的Zynq相关参数接口，如有我们再继续新增讲解。

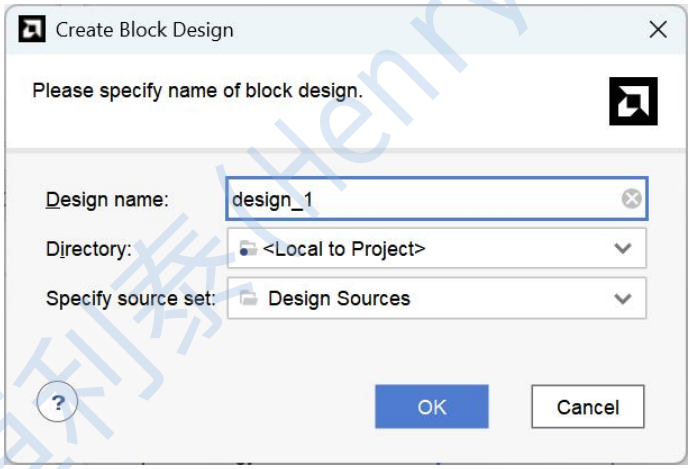
起始页（或 file->Project->New）	创建新工程(Create New Project)
向导起始页面	点击 Next->
Project Name(工程名)	工程名：rf67dr_ps_base 工程路径：（自己选择，尽量不要有中文路径） 勾选 Create Project Subdirectory,点击 Next->
Add Source(添加设计源文件)	点击 Next->
Add Exsixting IP(添加已有的 IP)	点击 Next->
Add Constraints(添加已有约束文件)	点击 Next->
Default Part(默认配置，芯片选型)	Family->Zynq UltraScale_RFSoCs Package->FFVE1156 Speed->-2 按具体型号选择 XCZU67DR-FFVE1156-2-i
New Project Summary(新建工程概况)	确认工程信息，选型等，点击 Finish 完成

三、BD文档创建和ZYNQ的PS调用

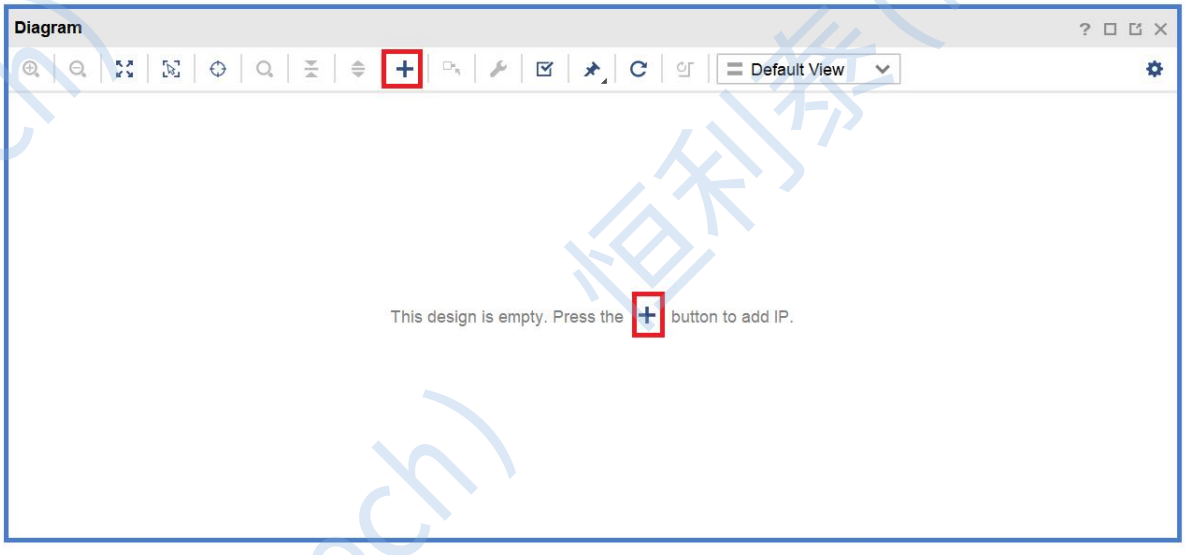
工程管理栏左边找到Create Block Design,点击创建一个bd文件：



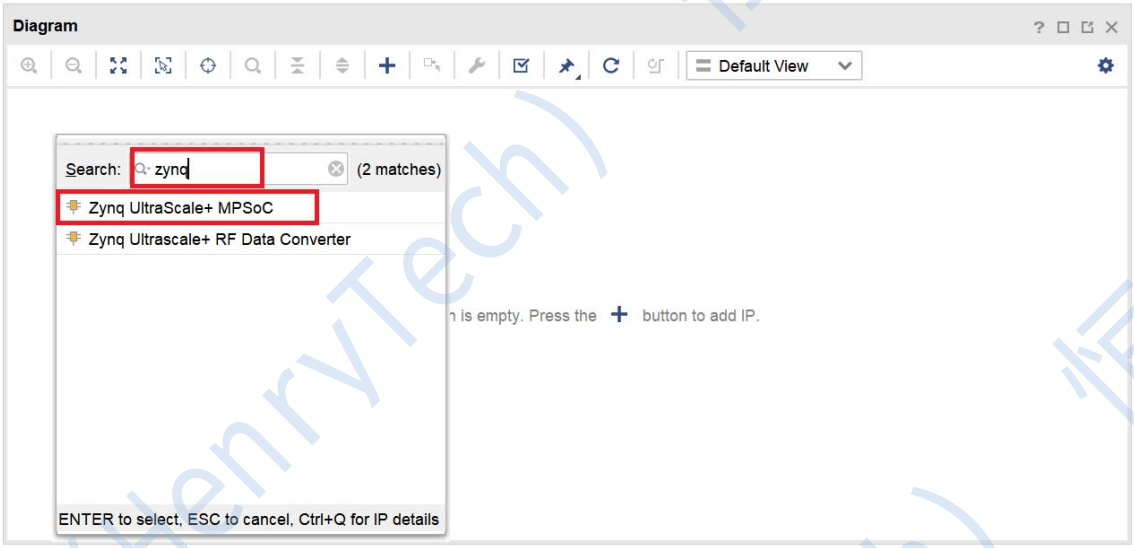
点击后弹出的对话框，默认新建的bd文件名，然后点击OK：



点击OK之后，bd文件打开了，是一个空白的窗口，我们可以点击中间的+号或者上方的+号添加IP进来：



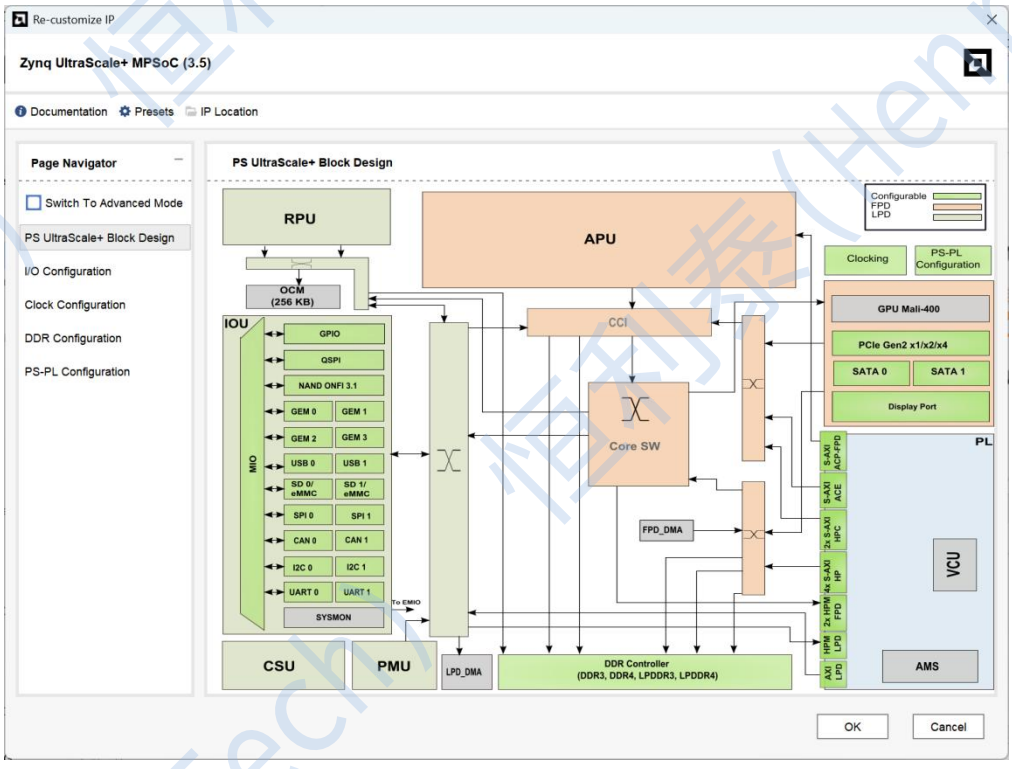
点击+号之后，我们输入zynq,双击第一项zynq ultrascale+mpsoc添加进来：



添加之后我们bd中就有了我们的zynq了：

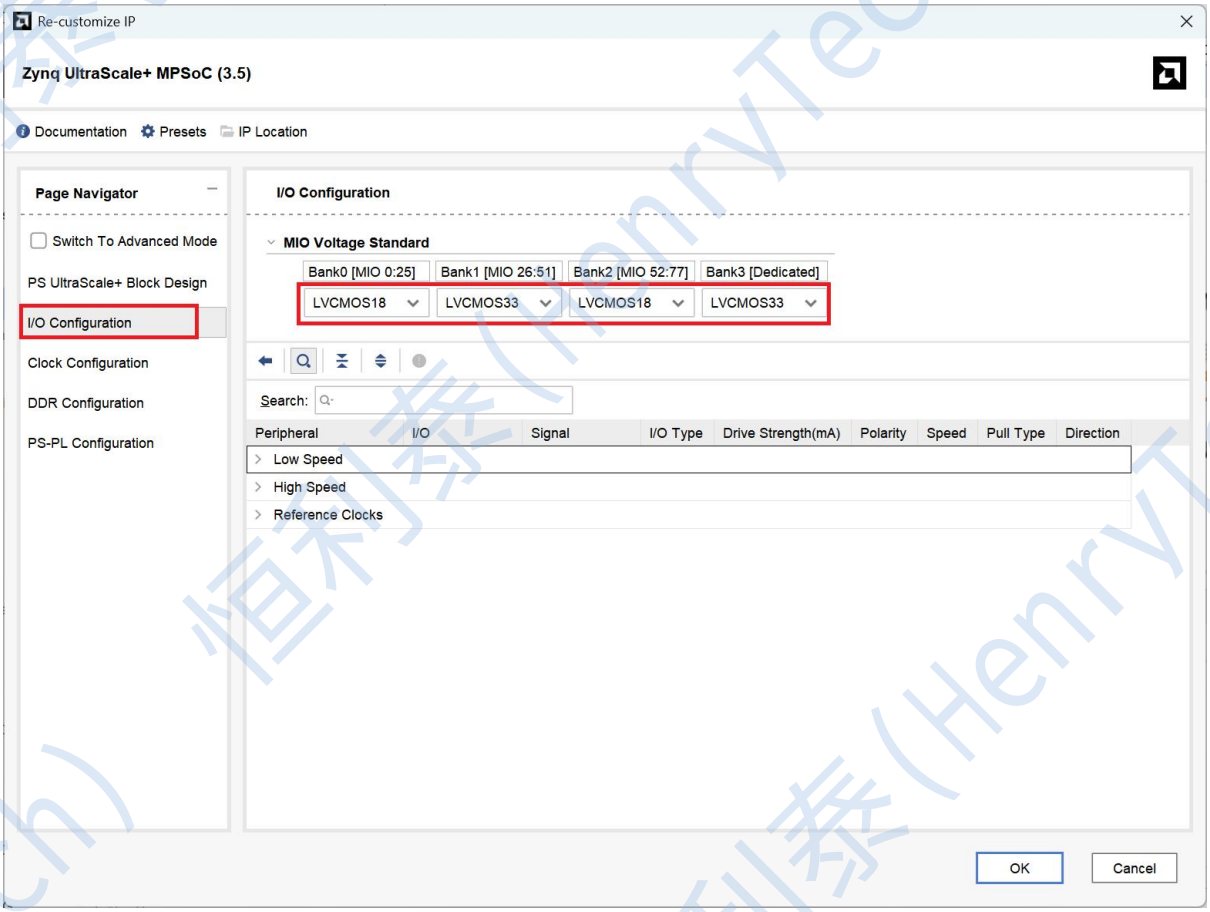
四、配置裁剪ZYNQ处理器外设IO

我们双击上面添加进来的ZYNQ处理器模块（PS），然后进入配置。启动配置界面后第一个页面我们可以看一下整个架构：

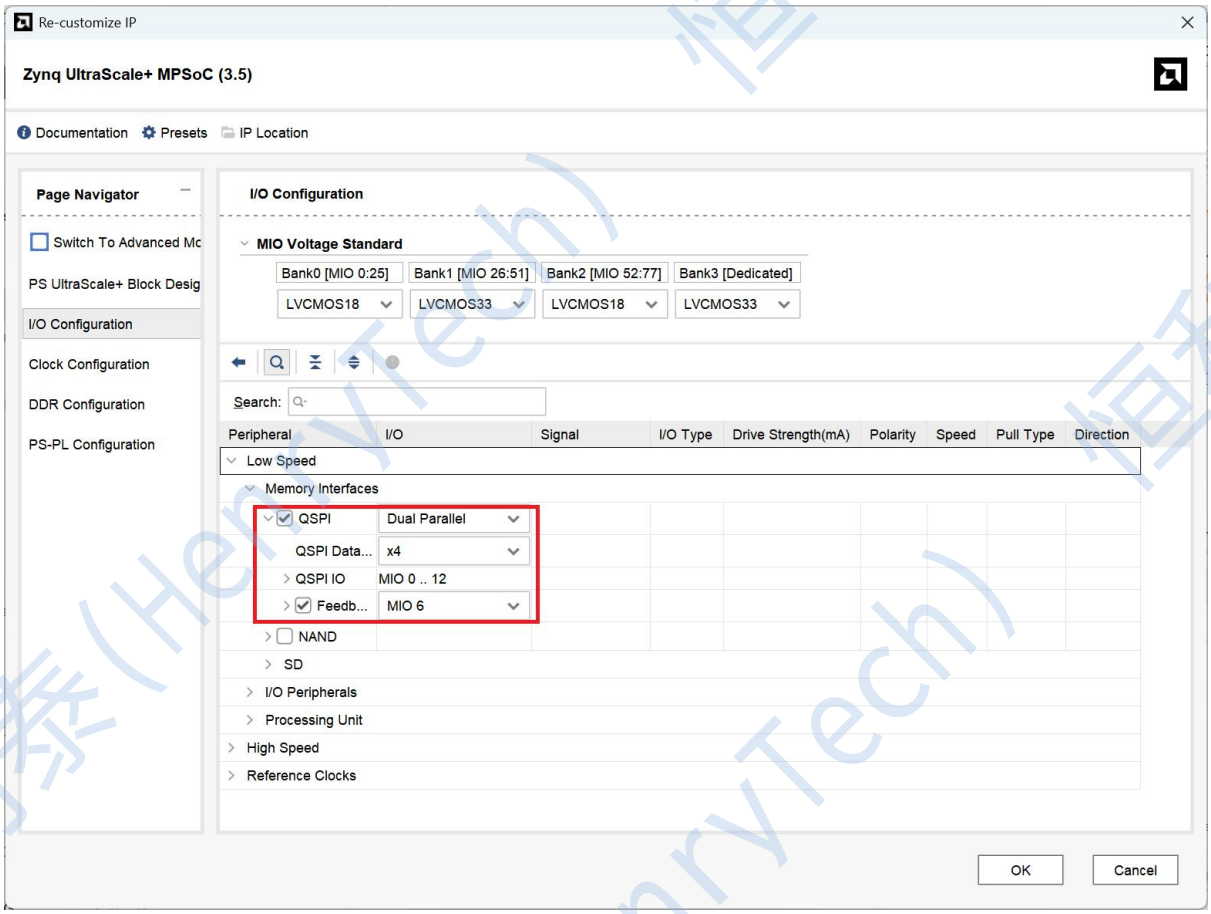


上面这个图可以看到我们整个ZYNQ部分，包括我们的处理器APU（应用处理器核心），RPU（实时处理器核心）等等，还包含我们芯片的一些外设，比如串口，SD卡，以太网，SPI等等，右边可以看到我们还有一些高速设备接口，像PCIE，SATA，Display port 一类。另外，ZYNQ还包含DDR3,DDR4存储接口，以及PS和PL的配置接口逻辑，时钟模块。

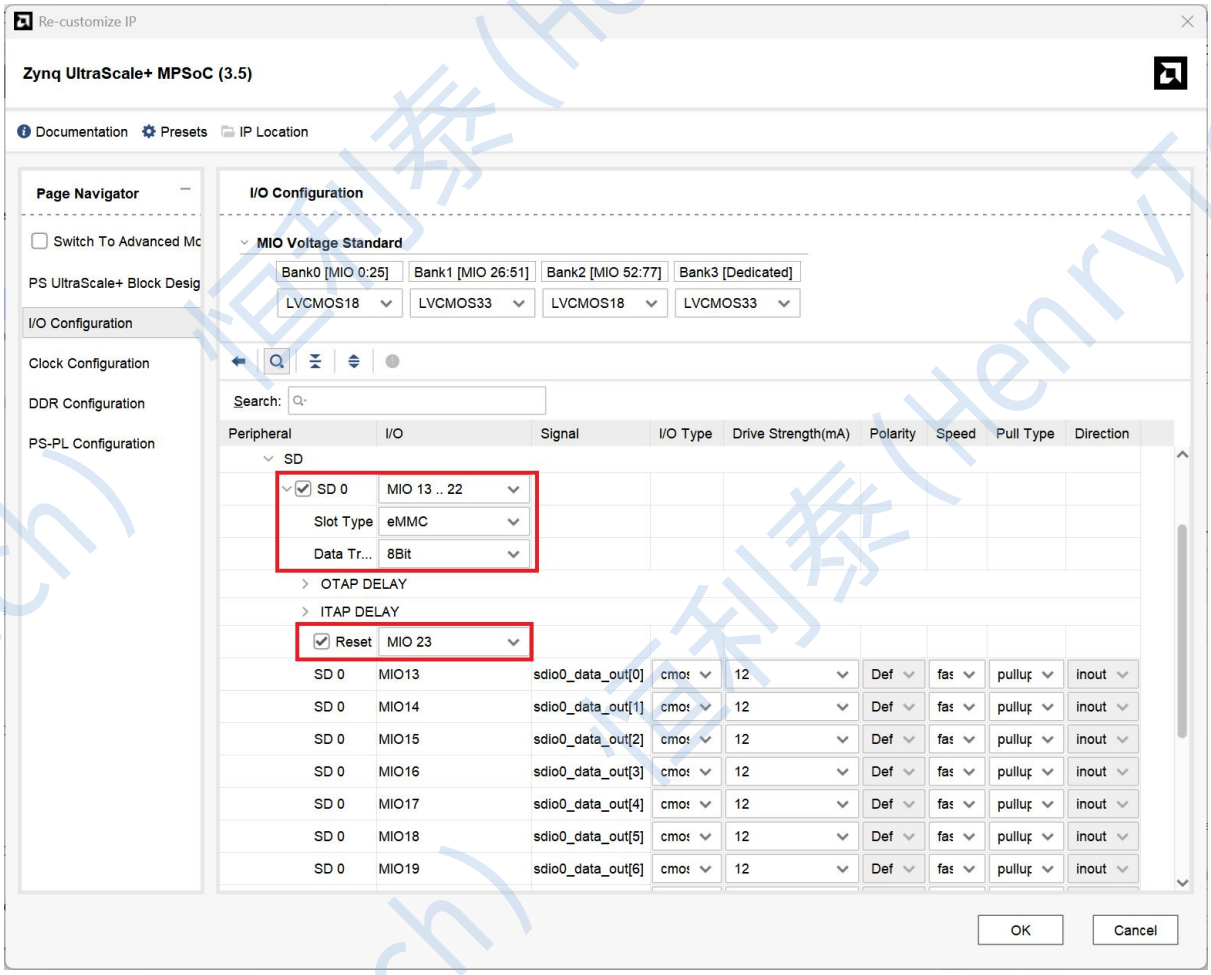
我们首先配置我们IO。左边选择I/O Configuration，然后上方的几个PS部分的bank电压选择一下，上方的四个芯片的bank都是我们ZYNQ处理器部分的，BANK500为1.8V，BANK501为3.3V，BANK502为1.8V，BANK503为3.3V，具体可以查看原理图的BANK500到BANK503。另外我们PS还包含一个BANK505，这个BANK是高速收发器接口，像SATA,PCIE,Display port等等，都是使用到这个BANK的高速信号，引脚电压不可选择。所以，我们配置电压，仅仅配置BANK500-BANK503的四个。在我们ZYNQ的PS配置界面，对应我们的Bank0-bank3。注意，在整个芯片上，这几个bank名字是500-503。实际上PS还有个BANK504，但是这个BANK是给DDR内存专用，不单独写出来。



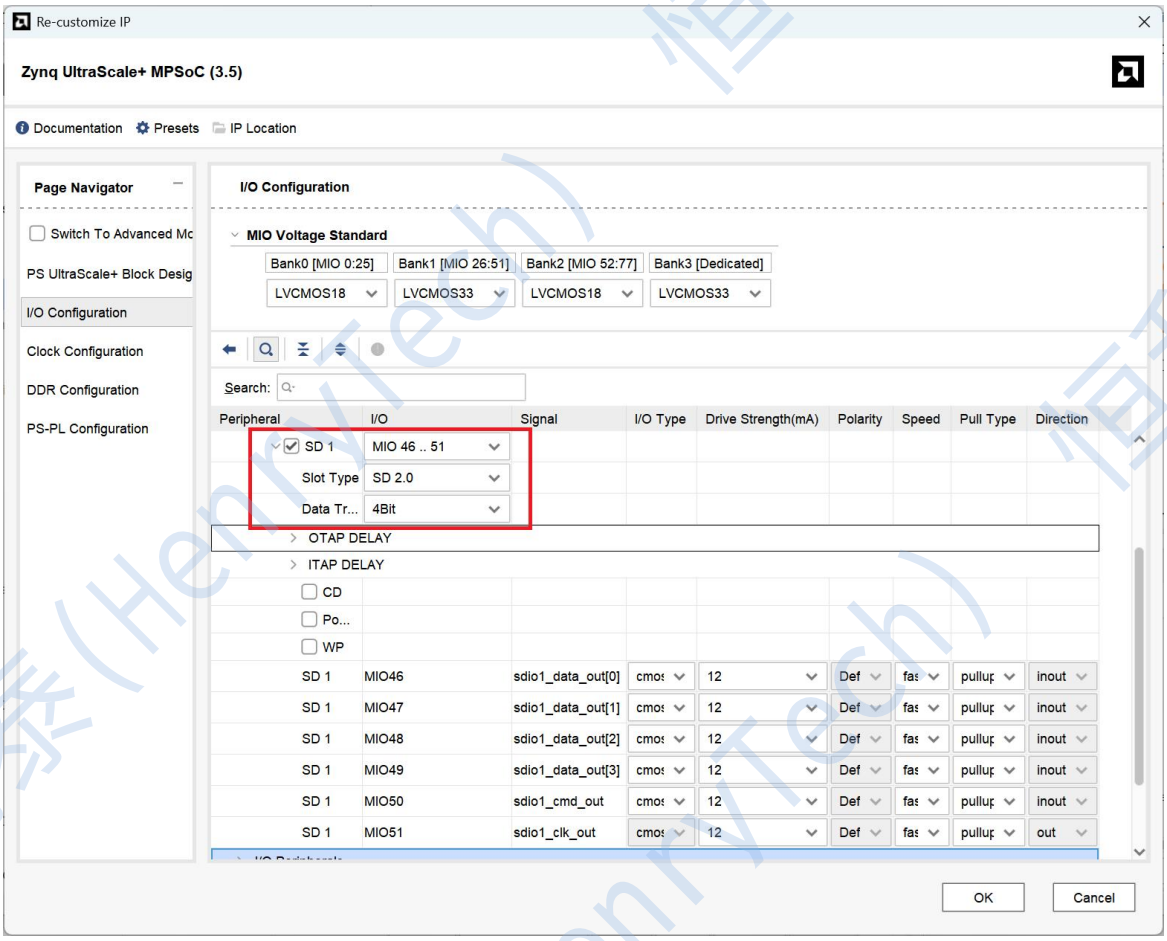
配置好Bank电压，我们下面的外设开始配置，首先，我们点开low speed,首先要配置的是QSPI接口，使用我们的板子上的FLASH存储。单颗FLASH是X4位宽，dual parallel模式，所以FLASH实际上是8位宽度，IO使用MIO0-12,feedback clk可以勾选：



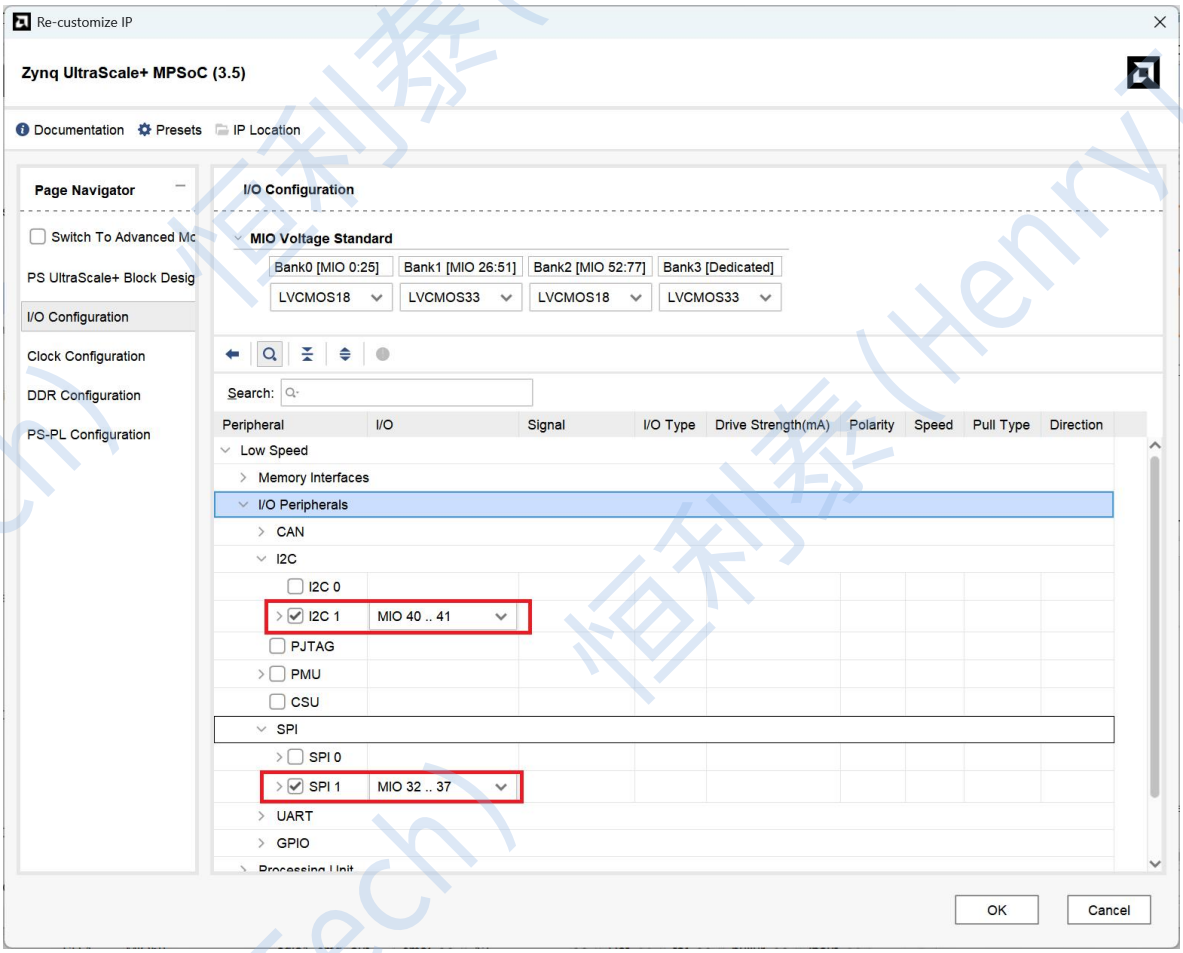
SD0接口，选择EMMC,IO12-22,8bit，reset使用MIO23:



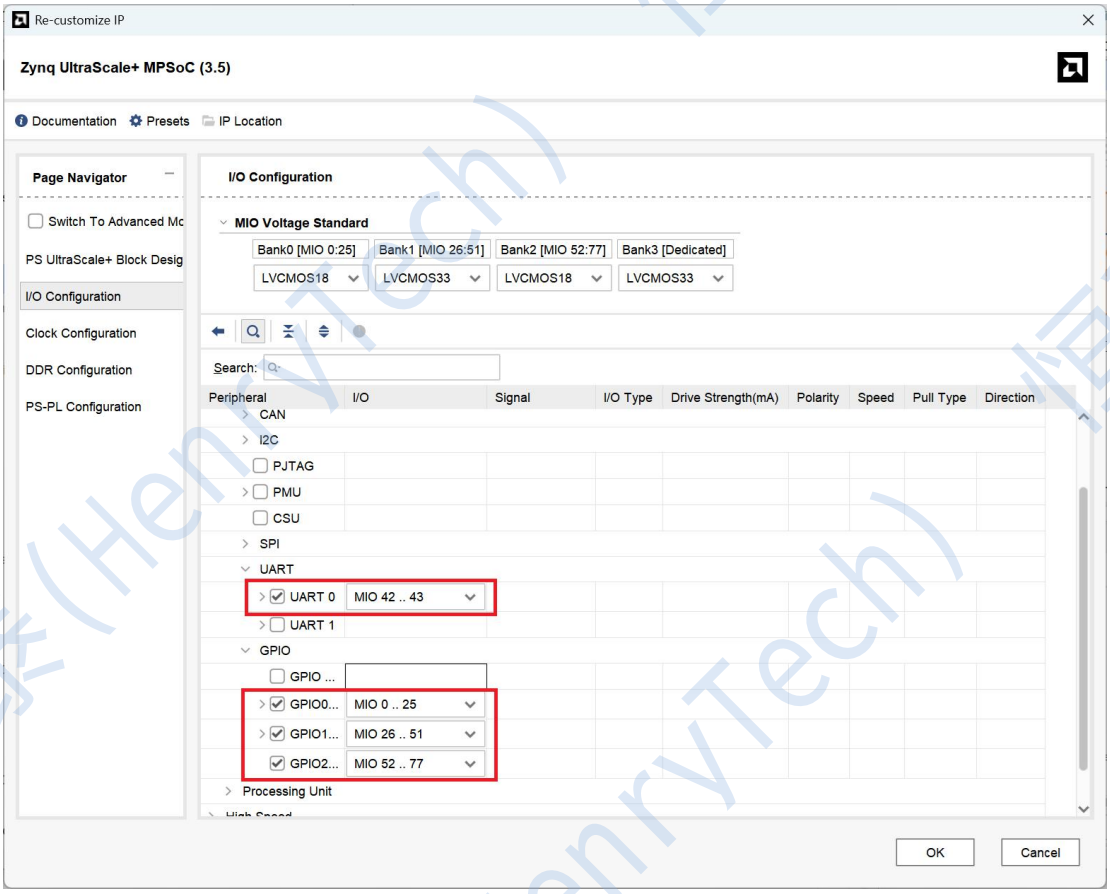
SD1选择MIO46-51,SD2.0，4bit:



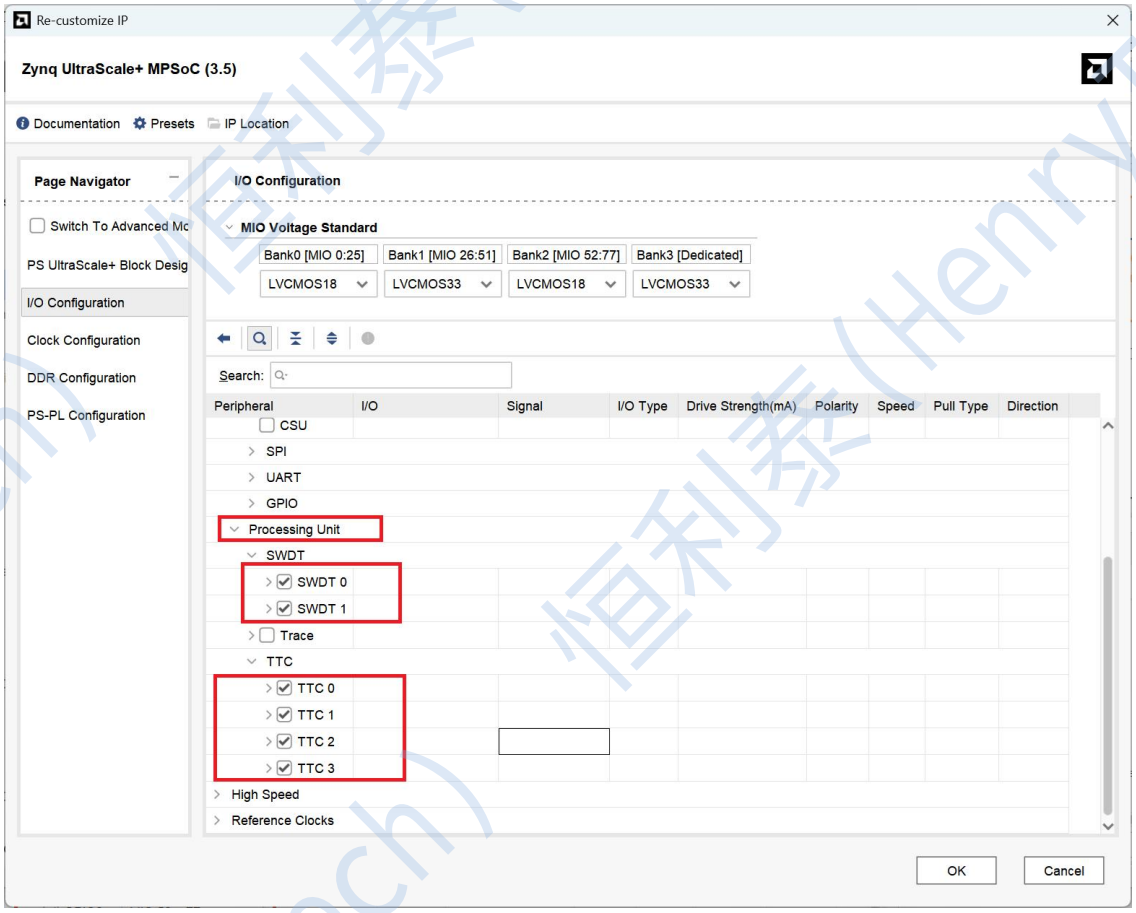
I2C配置了两个没用的MIO40-41引脚，不配置的话，以太网lwip例子可能会有点问题，SPI配置MIO32-37，实际上裸机用不到：



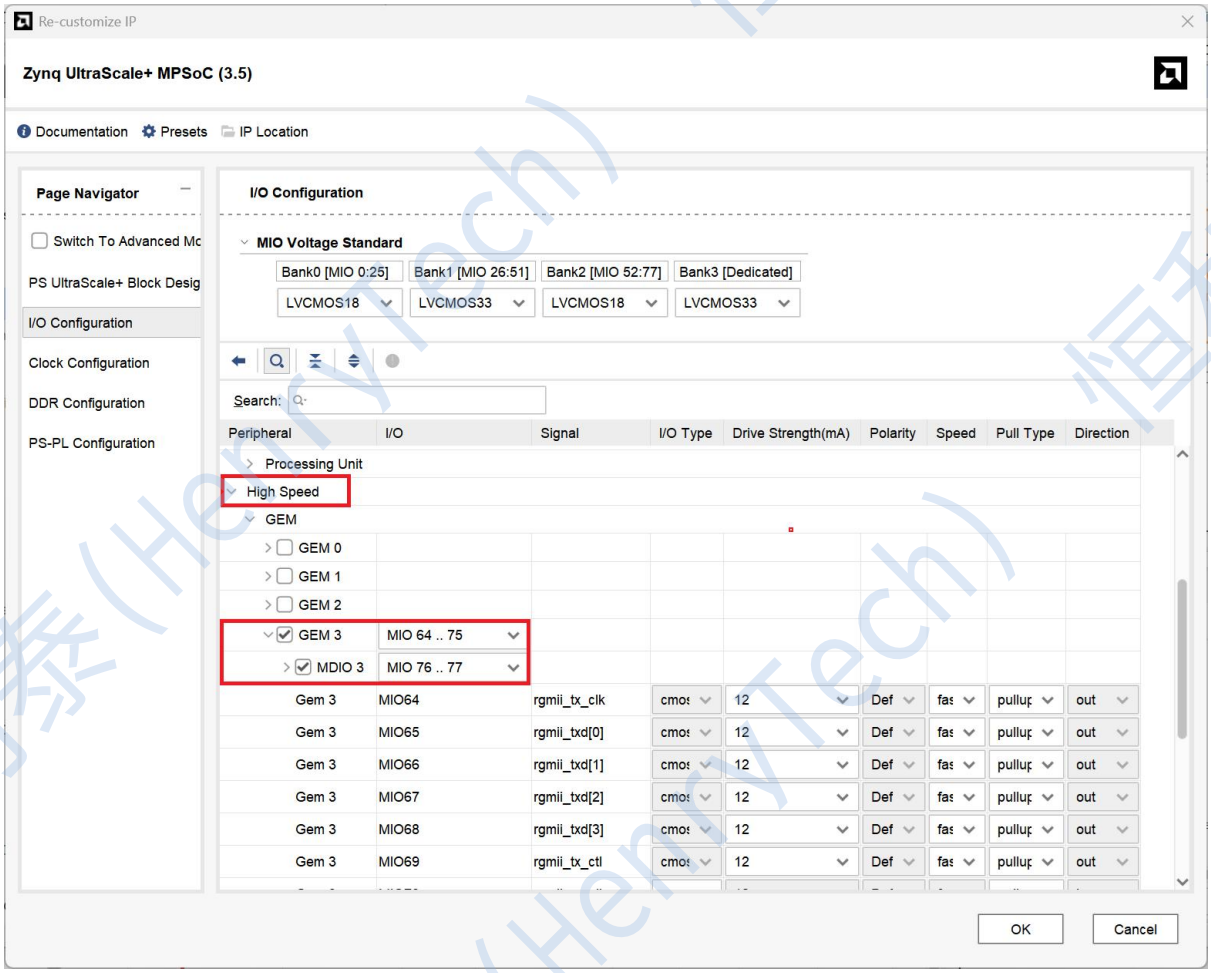
串口和GPIO，我们配置UART0为MIO42-43,GPIO勾选GPIO0-GPIO3,并且IO号配置如下:



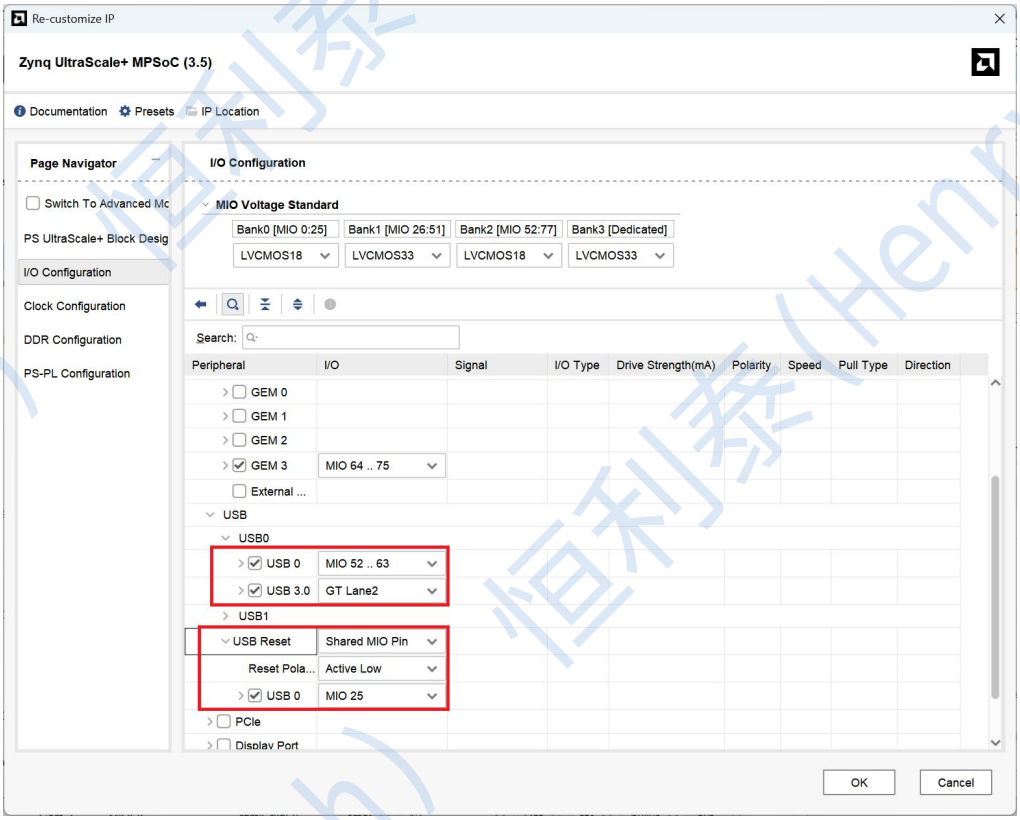
Processing unit配置勾选SWDT 0和1，这两个是看门狗，TTC全部勾选，这几个是定时器可能会用到：



千兆网口配置勾选GEM 3，配置IO如下：

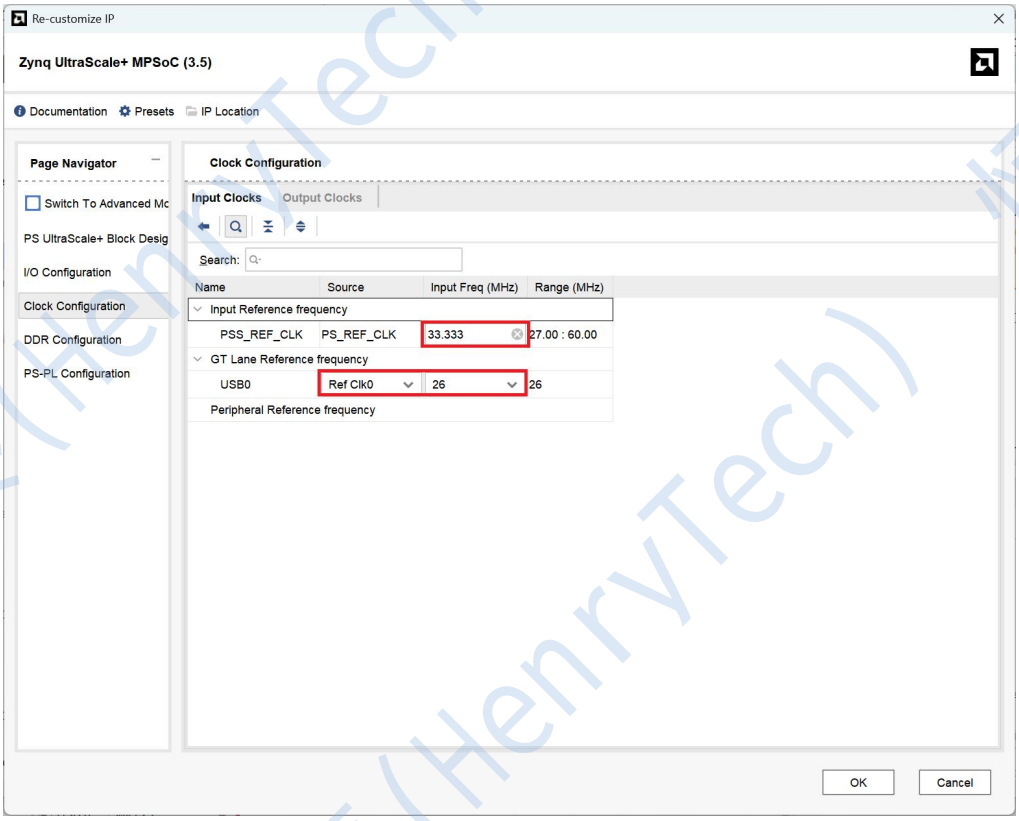


USB配置使用USB0，勾选USB3.0使用GT Lane2，复位使用MIO25：



五、ZYNQ 时钟配置

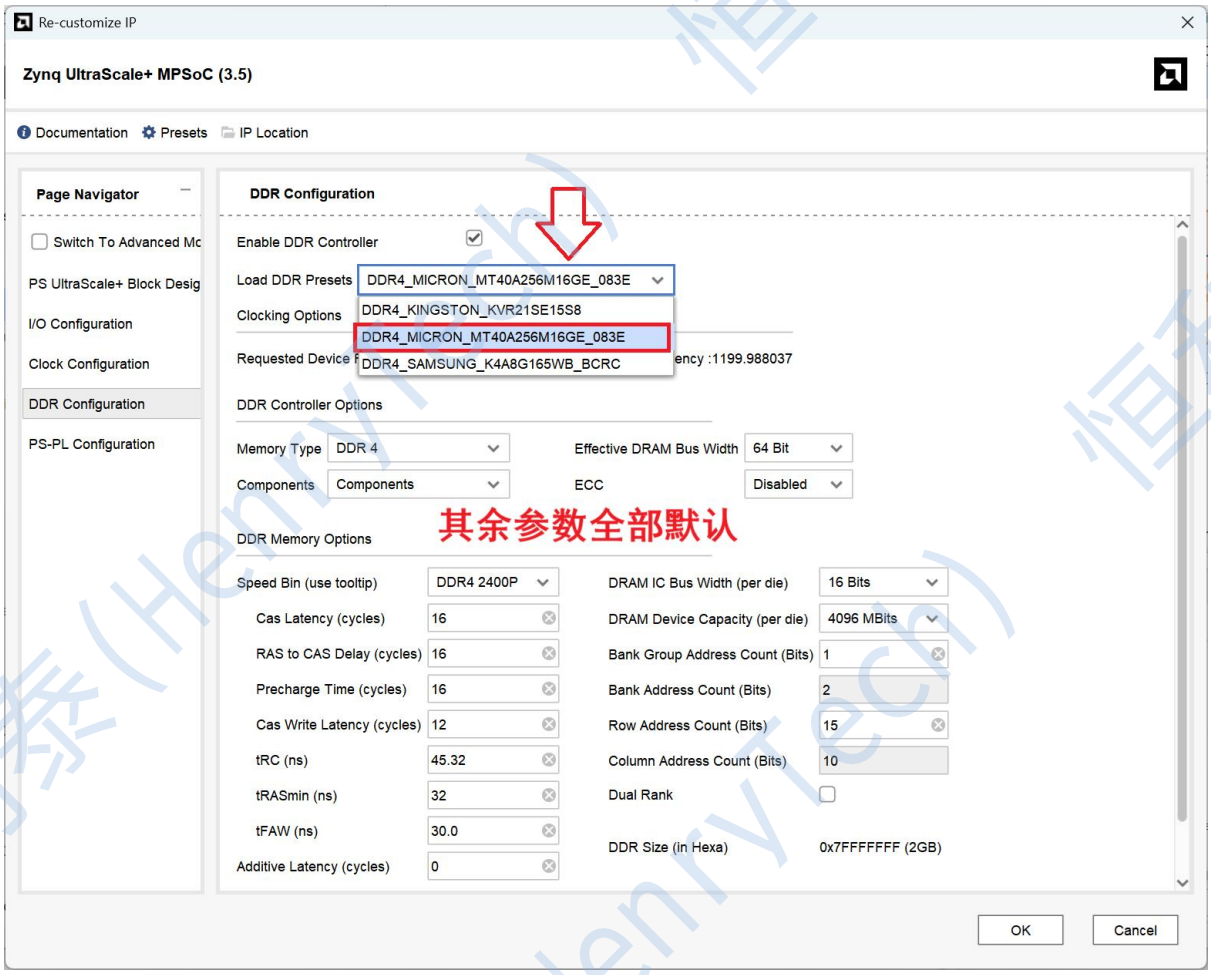
时钟配置，输入时钟配置如下，处理器使用33.333M晶振，USB0使用26M时钟，给USB3.0部分使用，注意输入从ref0输入：



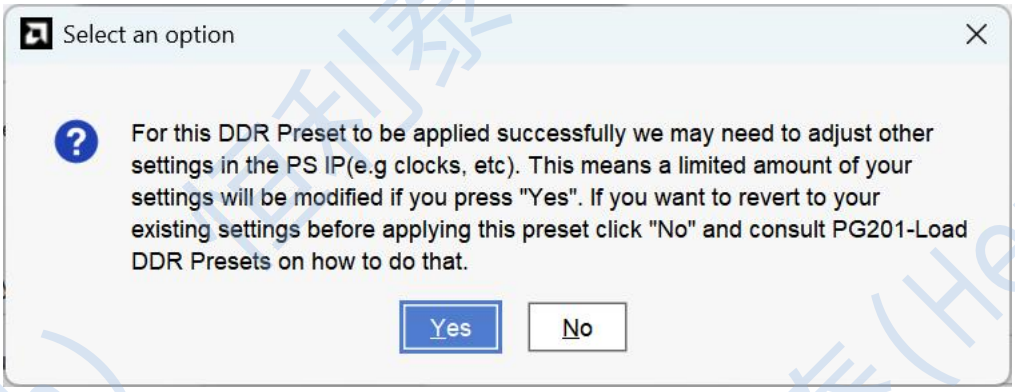
其余不用更改。时钟配置到此完成。

六、ZYNQ DDR4配置

首先，DDR4预加载选择如下型号：



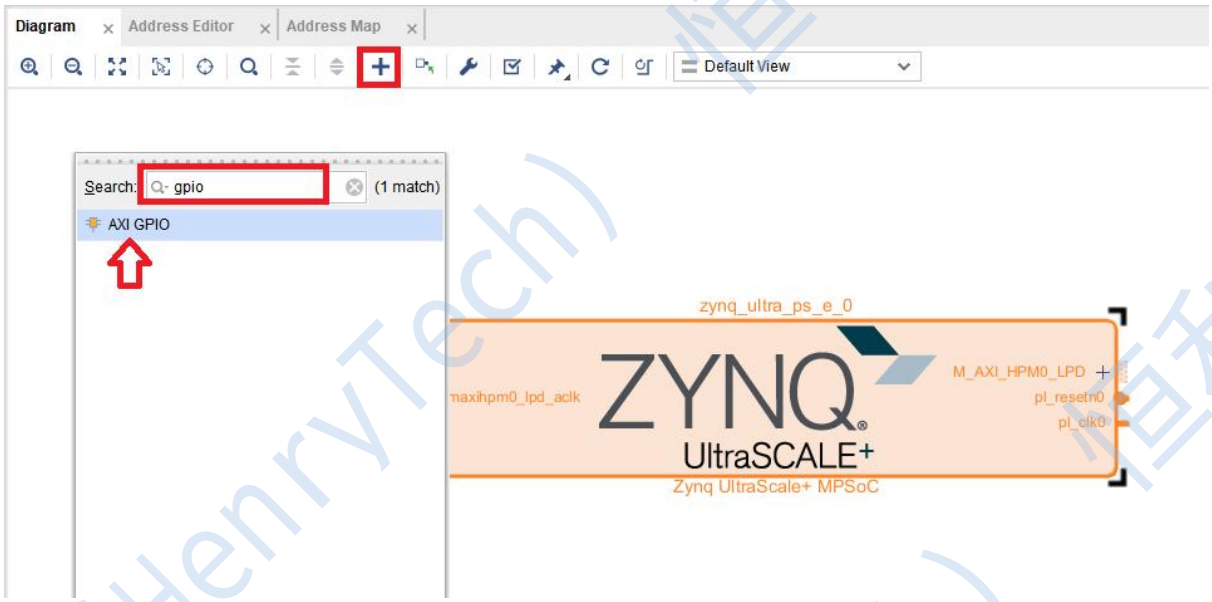
选择后会弹出对话框点击YES即可：



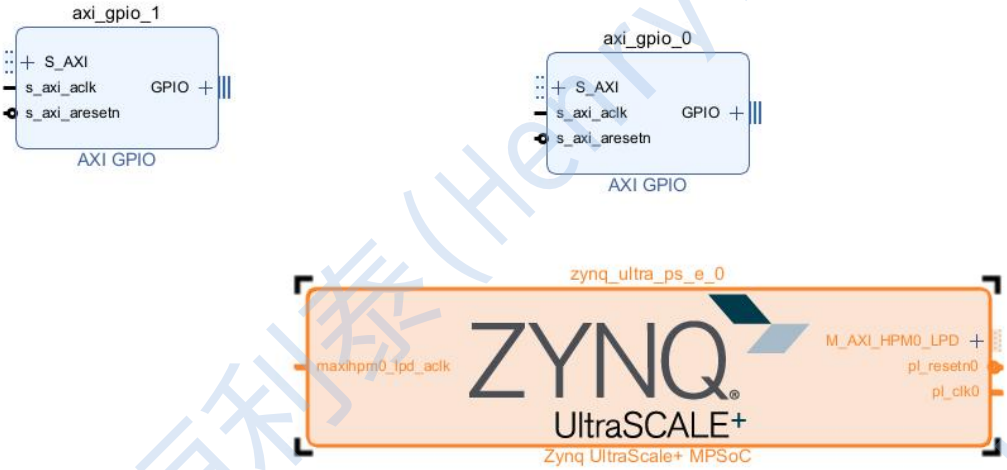
最后我们点击OK完成，到此PS的ZYNQ核就完成所有配置。

七、添加AXI GPIO

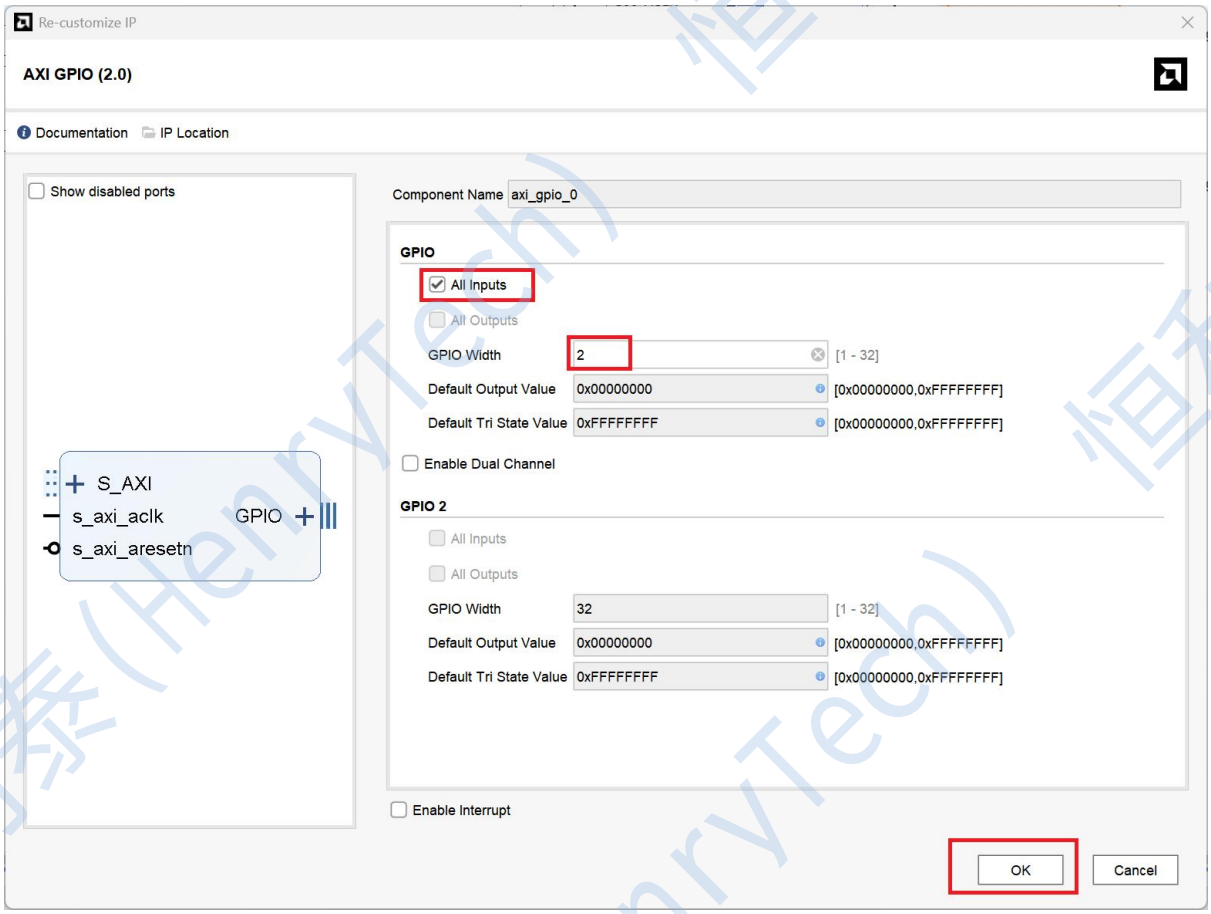
为了方便带有FPGA和ARM的同时开发，我们再设计中添加两个AXI GPIO模块，因为ARM和FPGA之间通信一般采用AXI，GPIO作为最简单的模块，我们给大家简单演示如何调用，我们在bd上方的+号这里点击，弹出的搜索框输入gpio，找到AXI GPIO,双击添加：



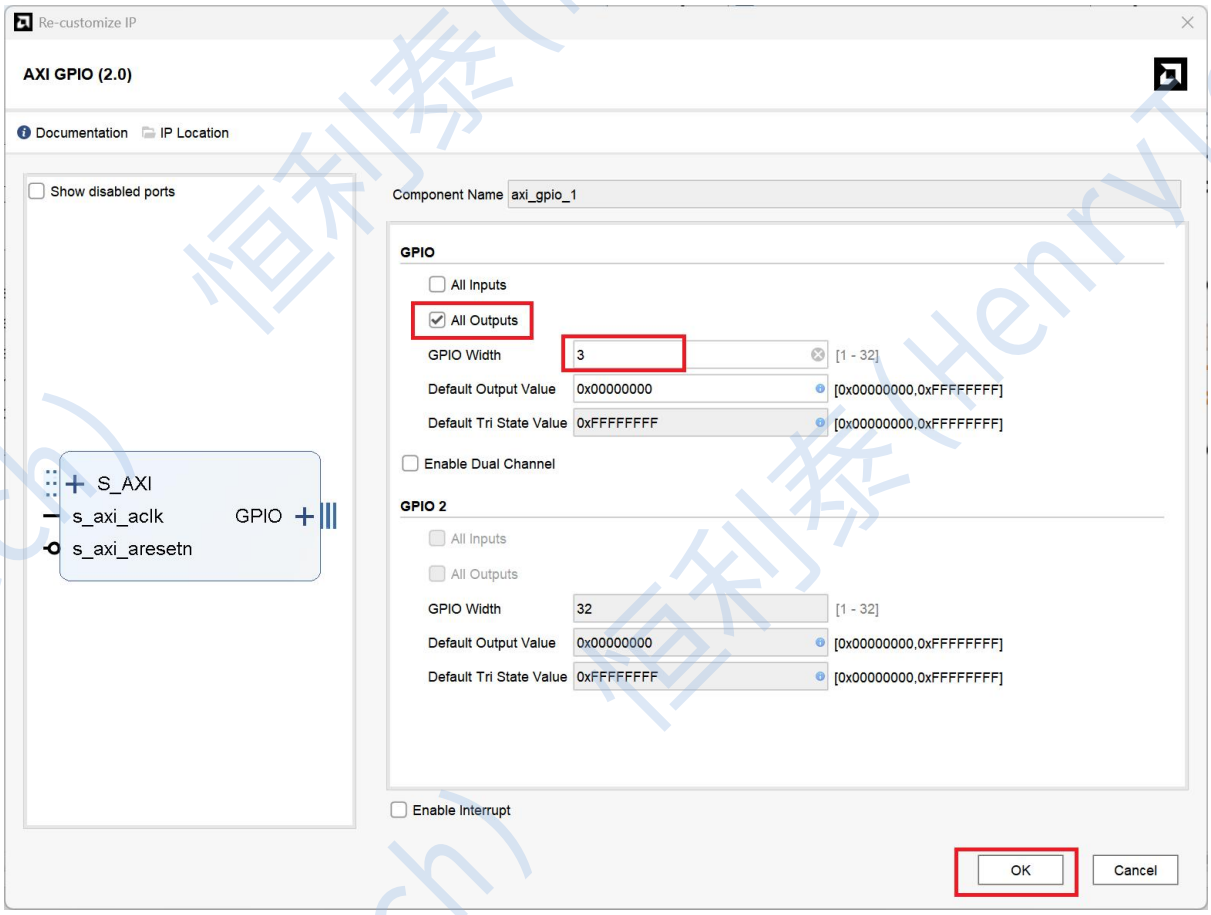
添加两个进来，一个是axi_gpio_0，一个是axi_gpio_1：



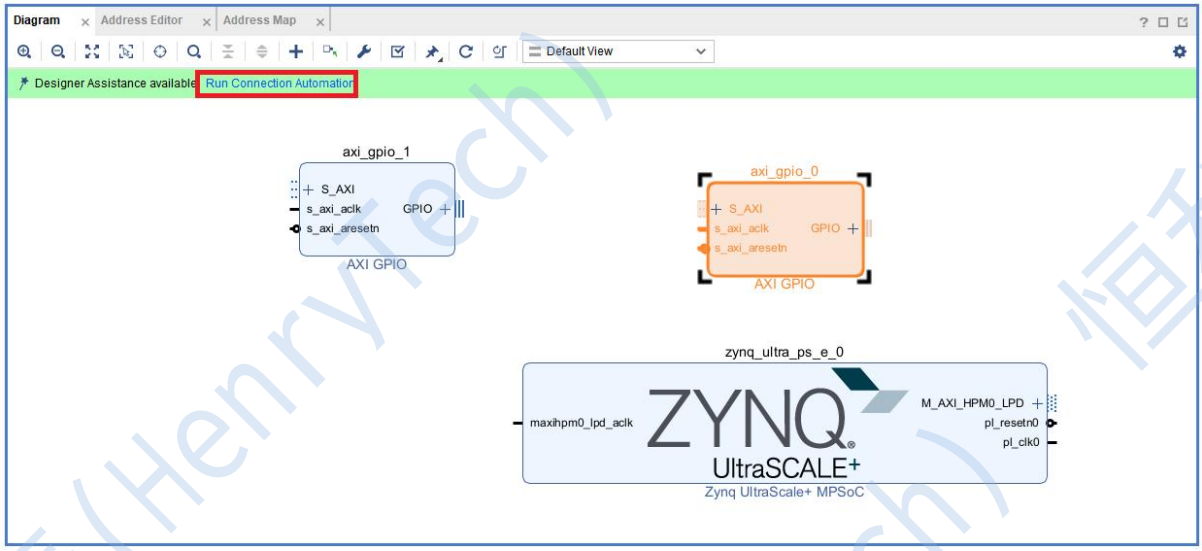
两个gpio模块双击进去配置。axi_gpio_0配置如下，配置为输入，宽度2，可用作PL两个按钮的输入：



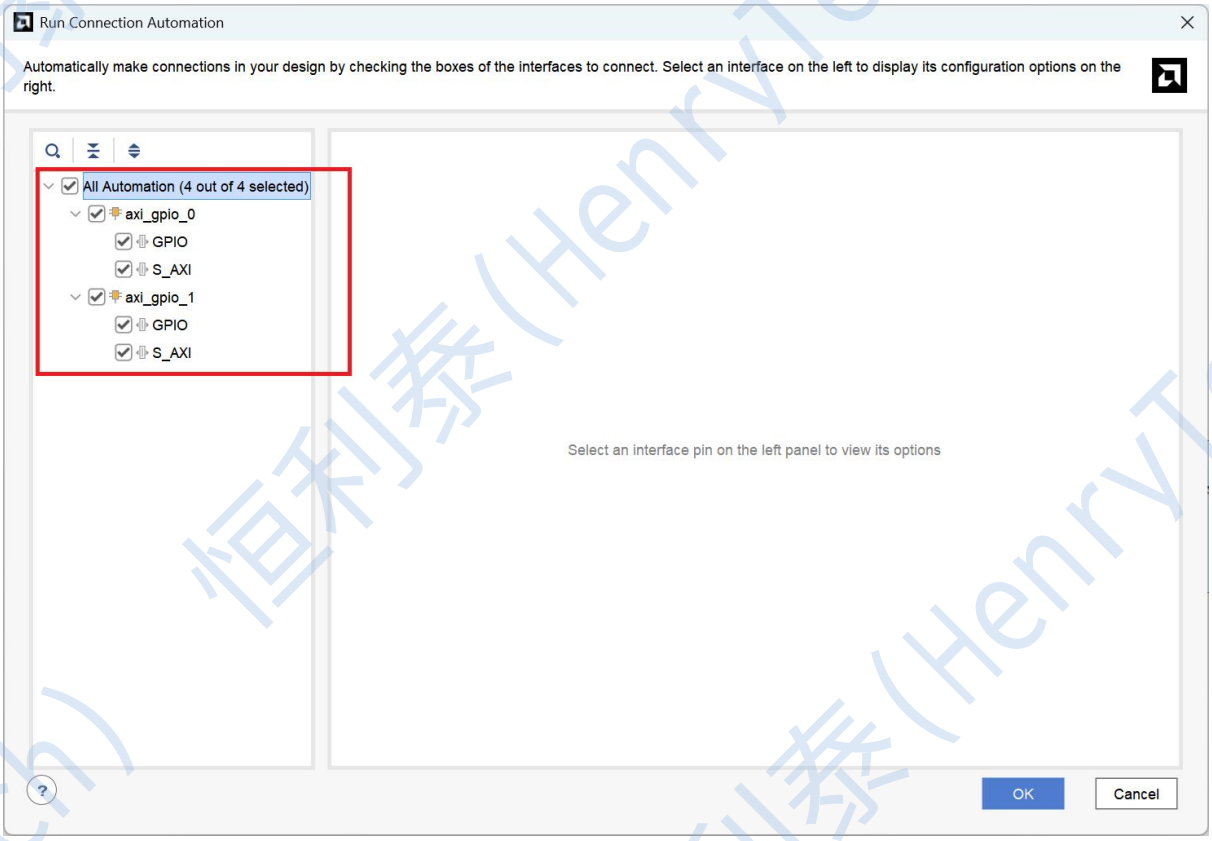
接着是配置axi_gpio_1，配置为输出，宽度3，可以用作PL的RGB LED控制：



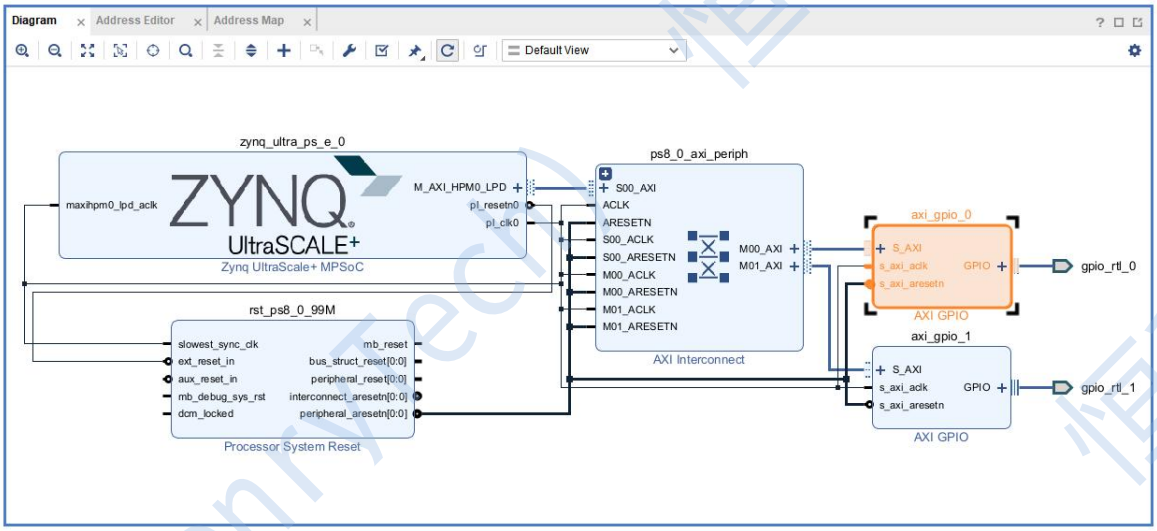
配好后点击OK完成配置，回到bd界面，上方的Run connection automation点击运行自动连接：



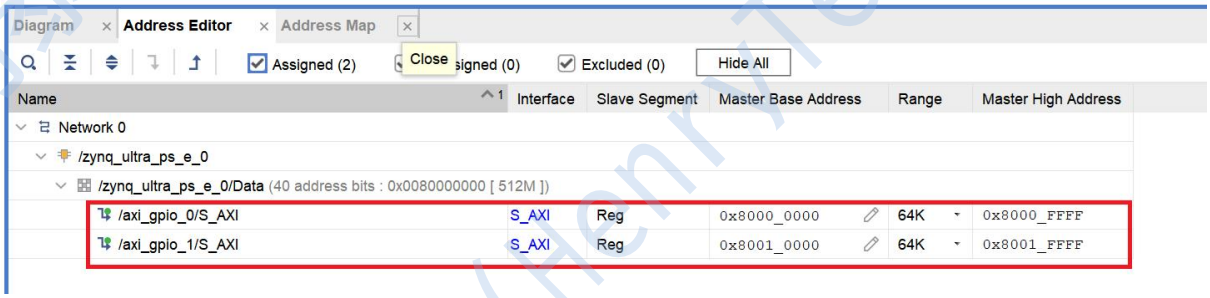
弹出的窗口左边全部勾上，点击OK：



自动连接后bd布局有点乱，我们点击上方的圆圈箭头重新布局，然后整齐多了：

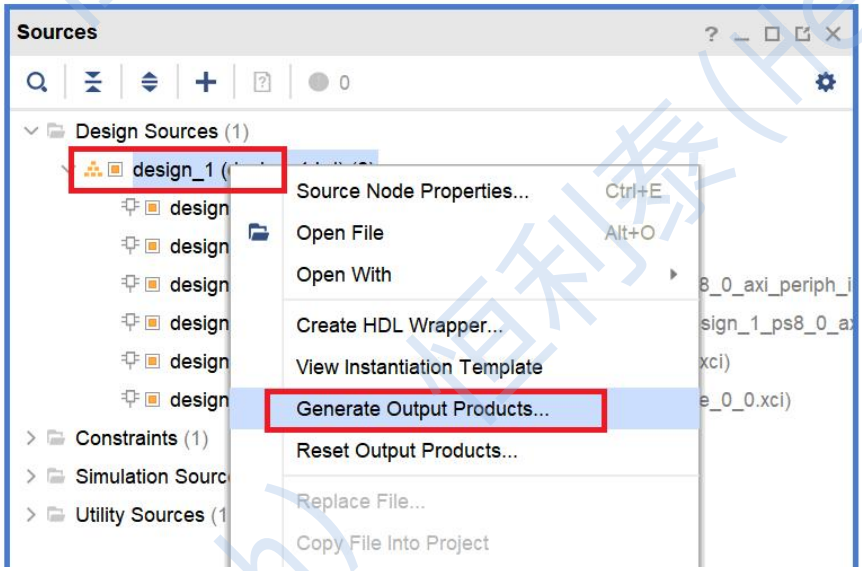


最后我们来地址分配。由于AXI接口是靠地址进行访问的，所以我们需要分配地址，我们bd上方有Address Editor，点了之后界面可以看到已经分配了，我如果没有分配可以再点击向下箭头分配：

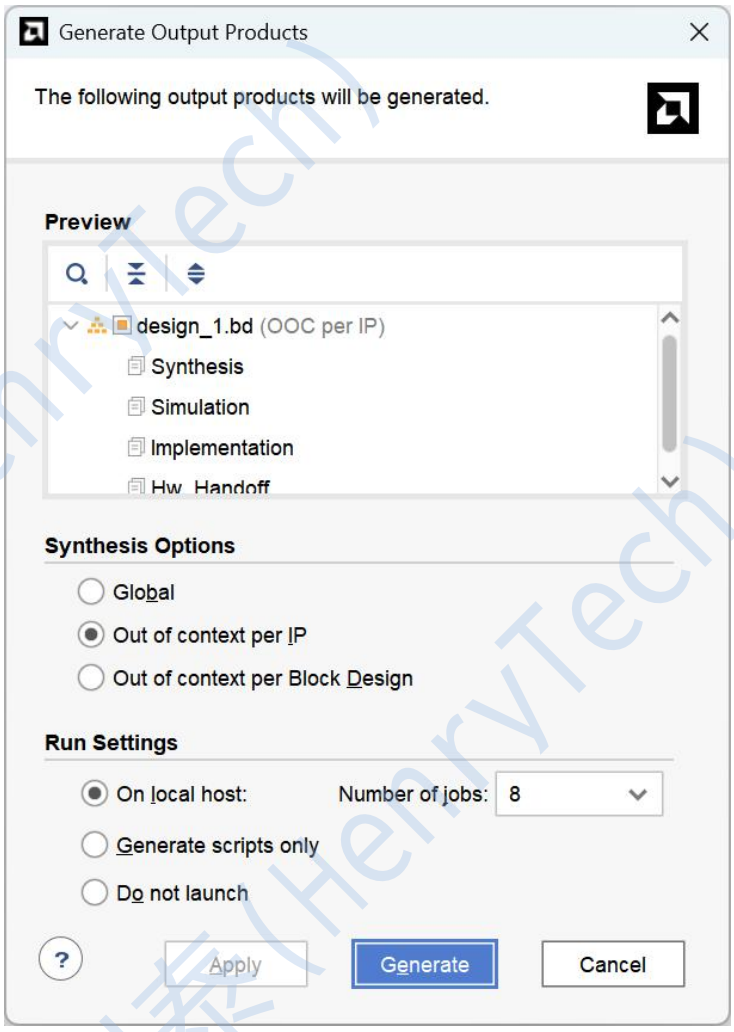


八、生成输出文件

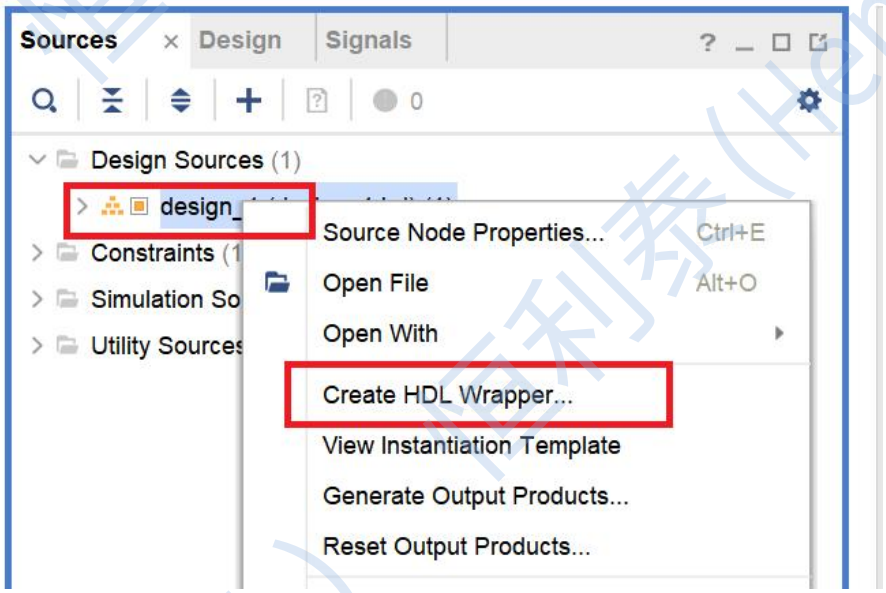
接着我们回到source窗口选中source下面的design sources找到我们bd文档右键选择Generate Output Product，生成我们BD文档内所有模块以及相关配置，源代码等等文件：



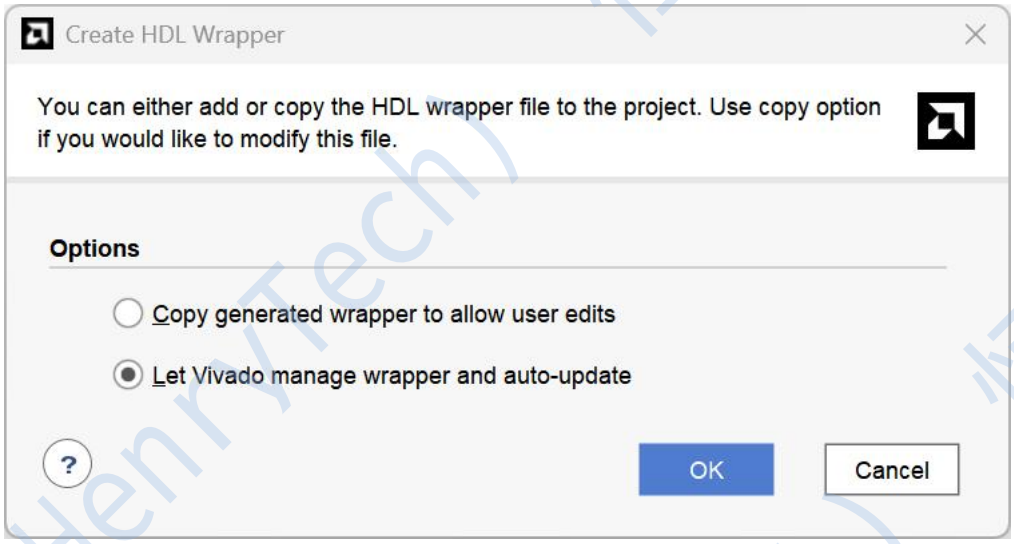
接着弹出对话框让选择生成输出文件，以及电脑的CPU线程使用多少，我们默认就好，点击Generate生成：



弹出任何对话框，我们不用管，点击OK即可。生成输出之后我们再次选中bd文件，右键生成顶层文件，右键选择Create HDL Wrapper：



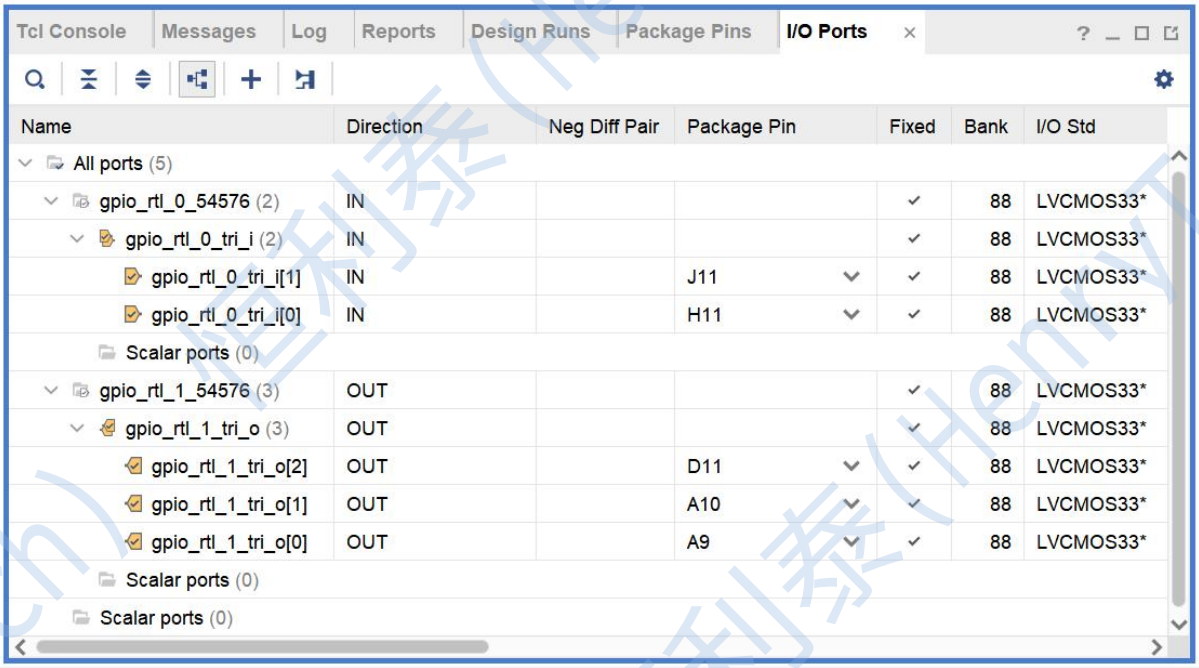
创建顶层弹出对话框，我们选则第二项让软件自己管理并启动更新，然后点击OK：



我们关于ZYNQ的处理器PS部分就配置好了。

九、绑定引脚生成BIT文件

绑定引脚方法可以参考《05最简单的led流水灯》这里不再赘述，绑定引脚如下：

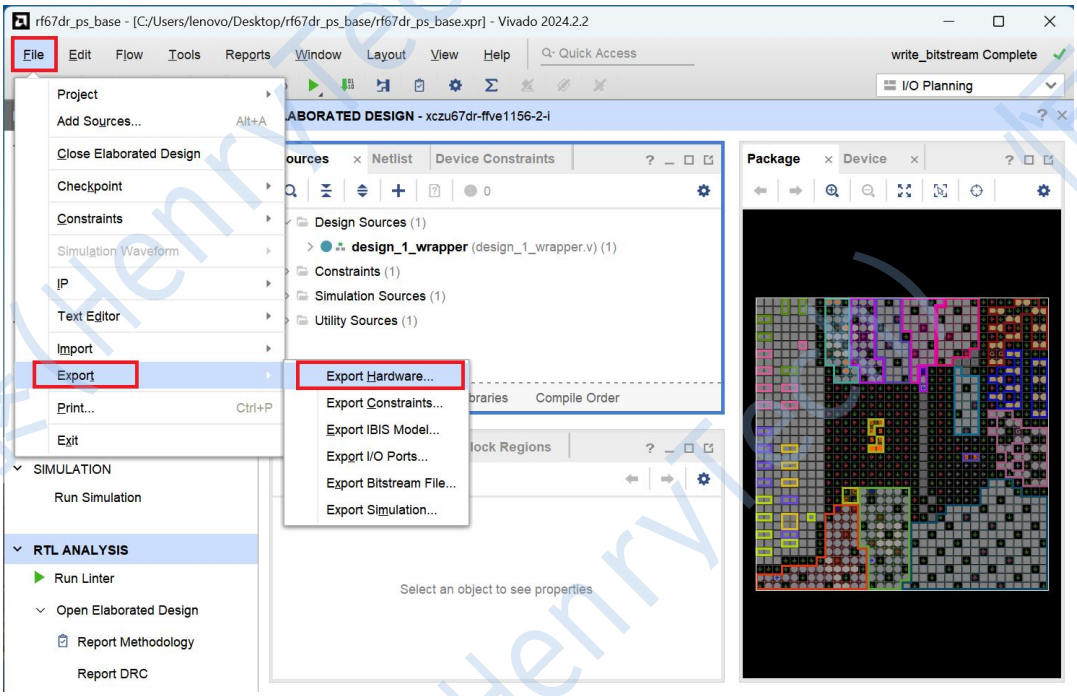


Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (5)						
gpio_rtl_0_54576 (2)	IN			✓	88	LVCMOS33*
gpio_rtl_0_tri_i (2)	IN			✓	88	LVCMOS33*
gpio_rtl_0_tri_i[1]	IN		J11	✓	88	LVCMOS33*
gpio_rtl_0_tri_i[0]	IN		H11	✓	88	LVCMOS33*
Scalar ports (0)						
gpio_rtl_1_54576 (3)	OUT			✓	88	LVCMOS33*
gpio_rtl_1_tri_o (3)	OUT			✓	88	LVCMOS33*
gpio_rtl_1_tri_o[2]	OUT		D11	✓	88	LVCMOS33*
gpio_rtl_1_tri_o[1]	OUT		A10	✓	88	LVCMOS33*
gpio_rtl_1_tri_o[0]	OUT		A9	✓	88	LVCMOS33*
Scalar ports (0)						
Scalar ports (0)						

绑定好后保存，直接生成BIT文件，注意，如果使用到FPGA部分的AXI GPIO模块，必须要生成BIT。如果仅仅使用ARM部分，无需生成，为了避免失败，建议等待先生成BIT文件。

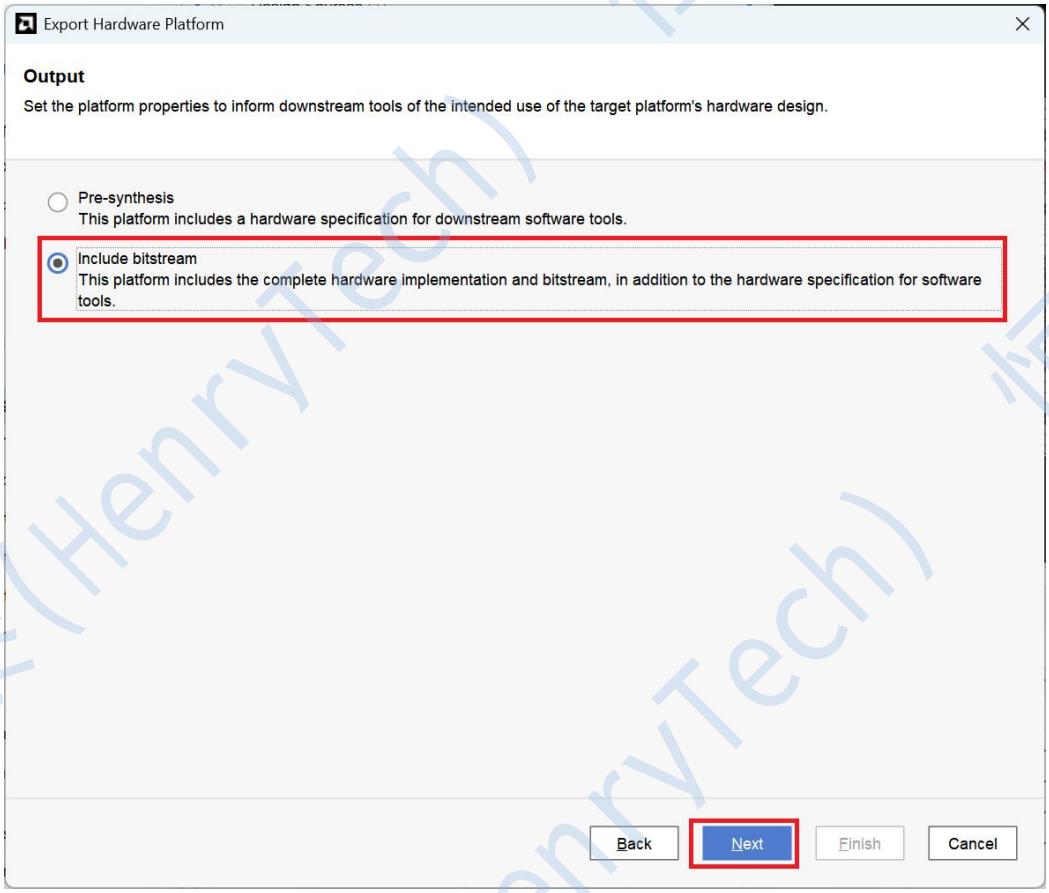
十、导出硬件文件

我们上一节完成了ZYNQ处理器（PS）配置，接着我们要导出一个文件，这个文件交给vitis软件进行软件开发。我们VIVADO左上角选择file下拉，找到export，右边选择export hardware导出硬件：

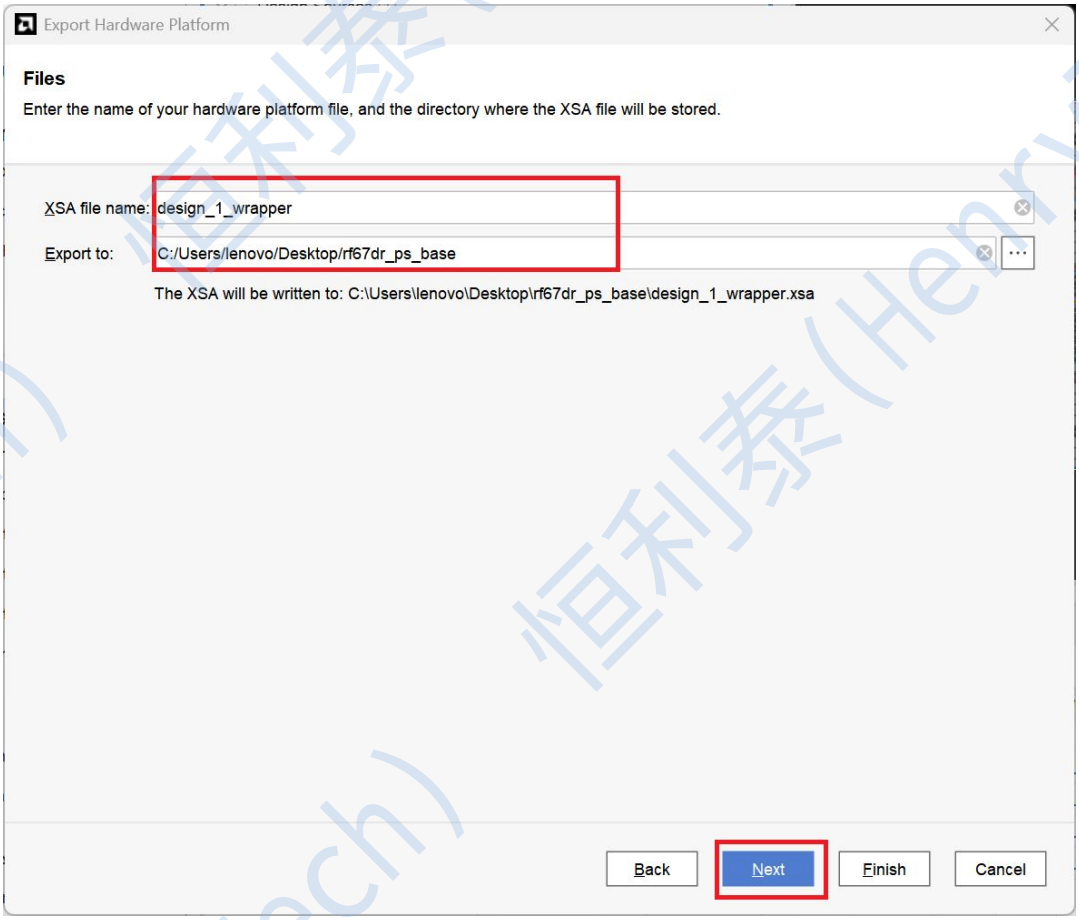


弹出第一个界面点击next：

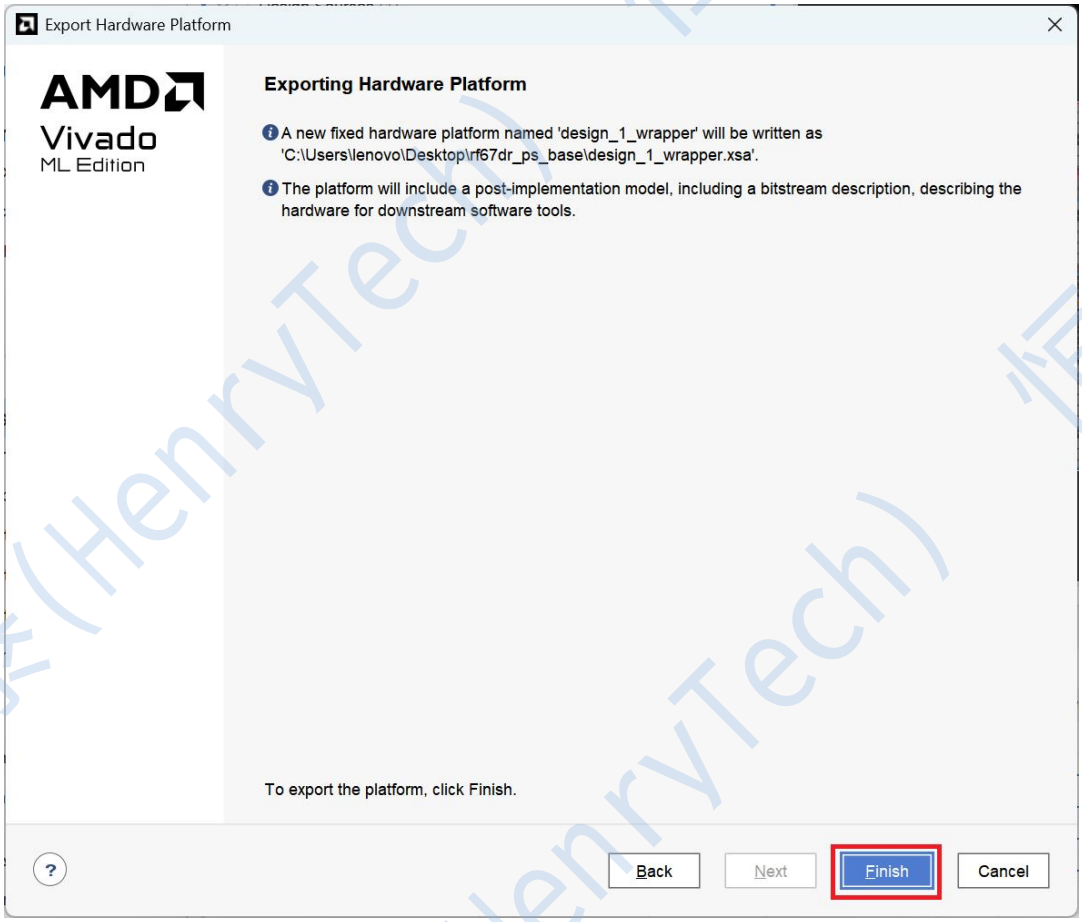
第二页勾选 include bitstream，点击Next：



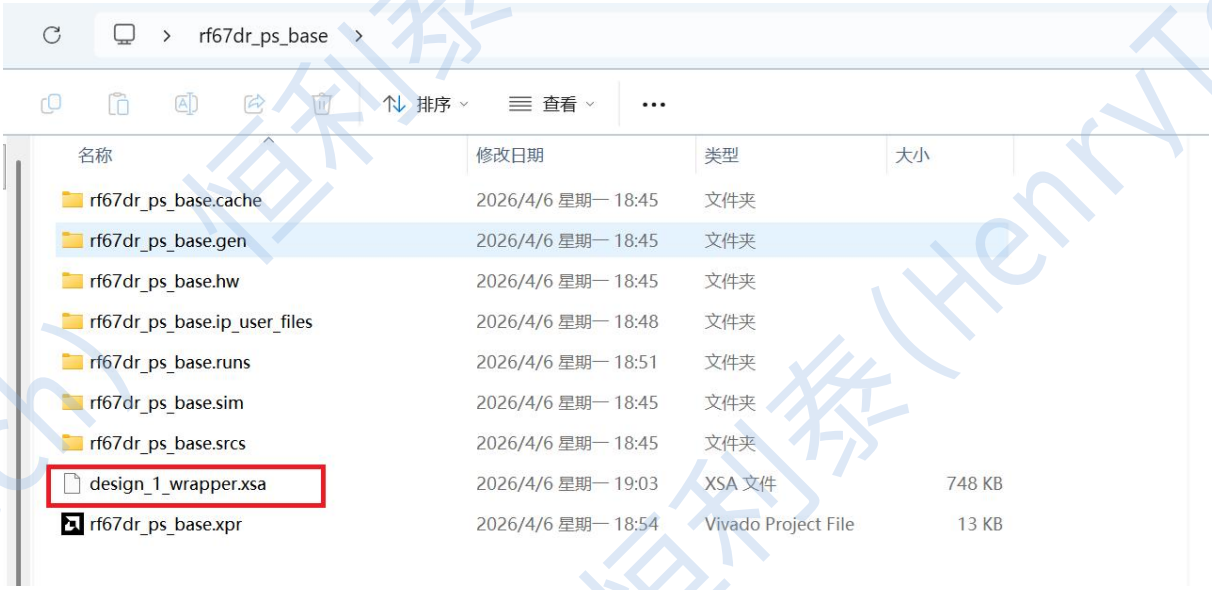
接着这一页是我们导出硬件文件的名字和路径，默认就在我们的工程目录路径。我们默认就可以了，然后点击next：



最后一页我们点击Finish完成导出。导出需要花一点时间：



导出完成，我们会在工程下面找到一个.xsa后缀的文件就是我们所需要的：



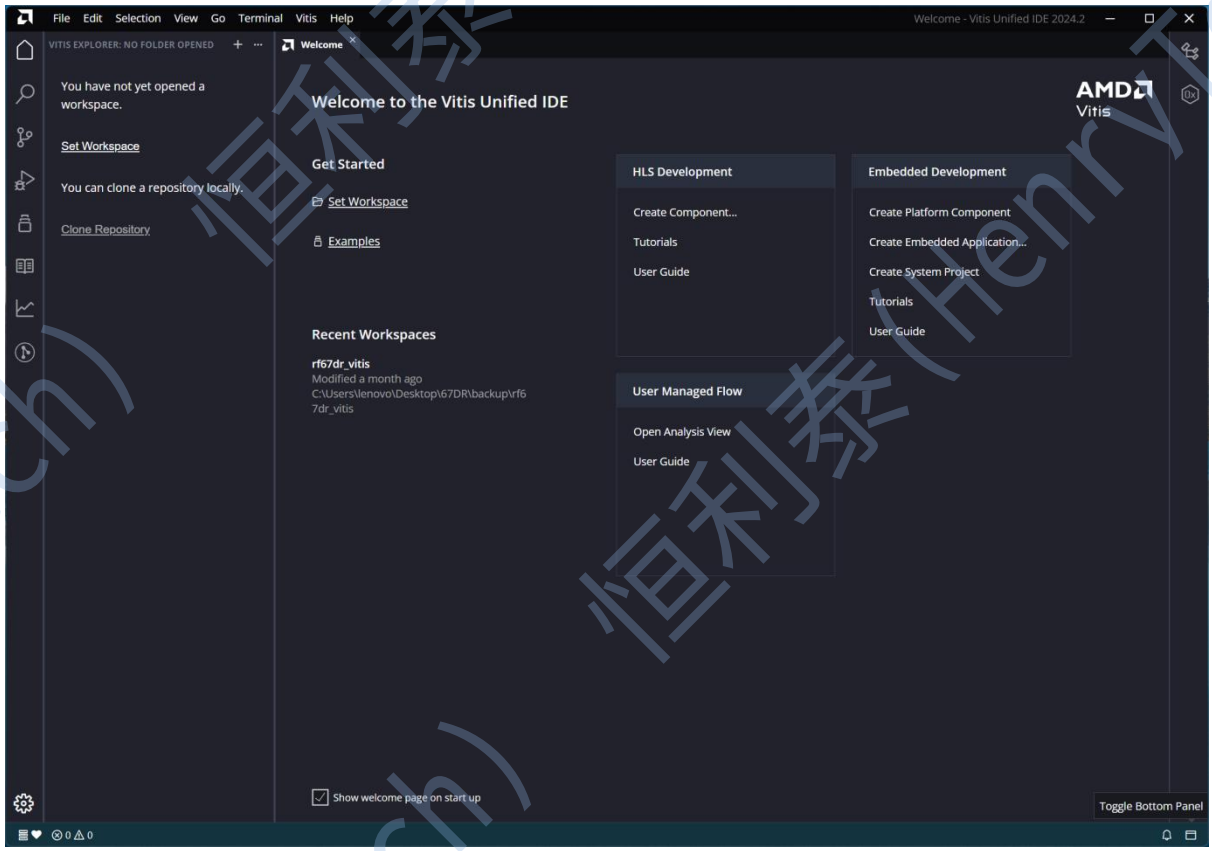
到这里我们可以将这个xsa文件拷贝出来待用，我们下一节讲解如何使用vitis。

十一、VITIS导入硬件文件新建平台和工程

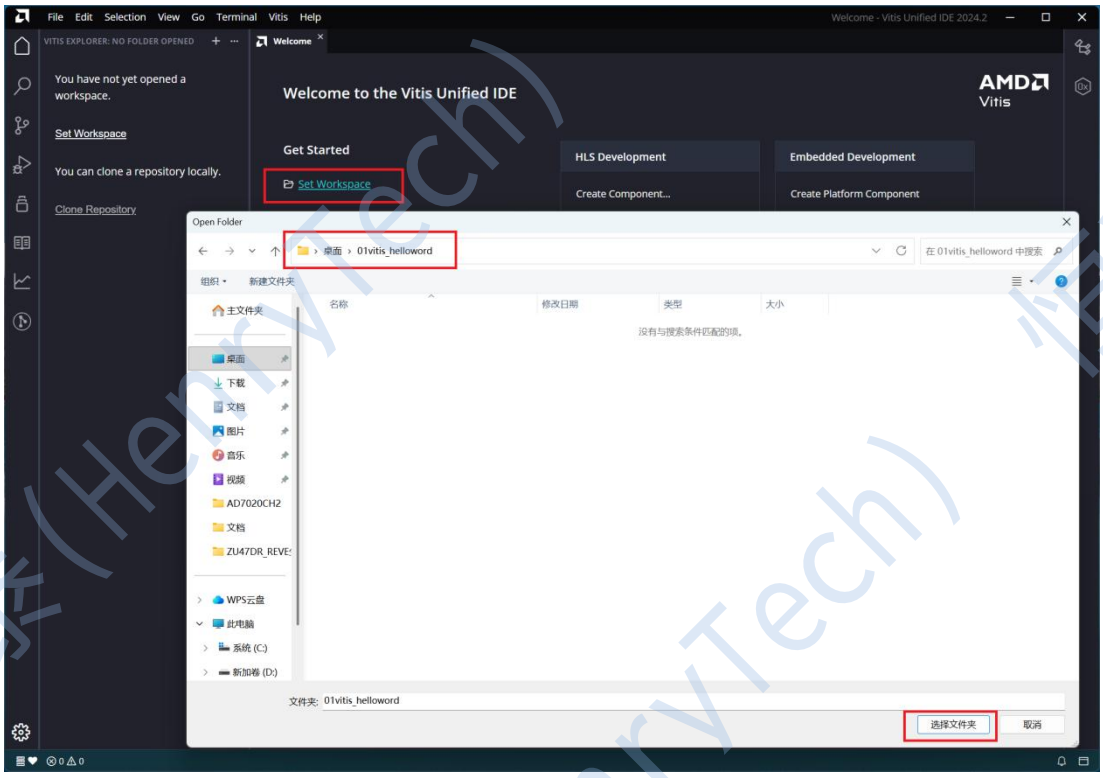
上一节我们导出一个xsa硬件文件。这一节我们使用这个文件新建一个vitis工程，需要用到这个文件。我们启动VITIS软件，找到Xilinx vitis双击启动或者从开始菜单栏找到vitis图标打开它：

启动后弹出工作路径选择窗口，我们在桌面上或者硬盘根目录下新建一个文件夹叫做01vitis_helloword，将我们上一节导出的xsa文件复制到新建的这个文件夹下：

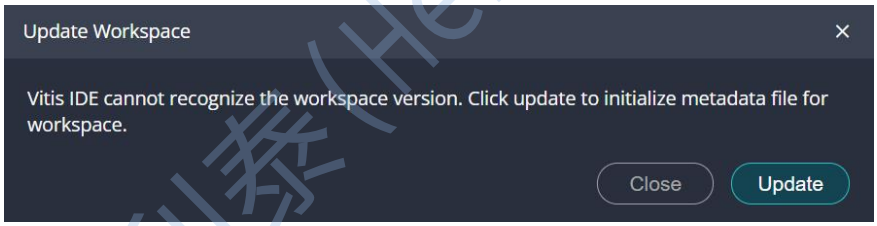
Vitis启动后界面如下：



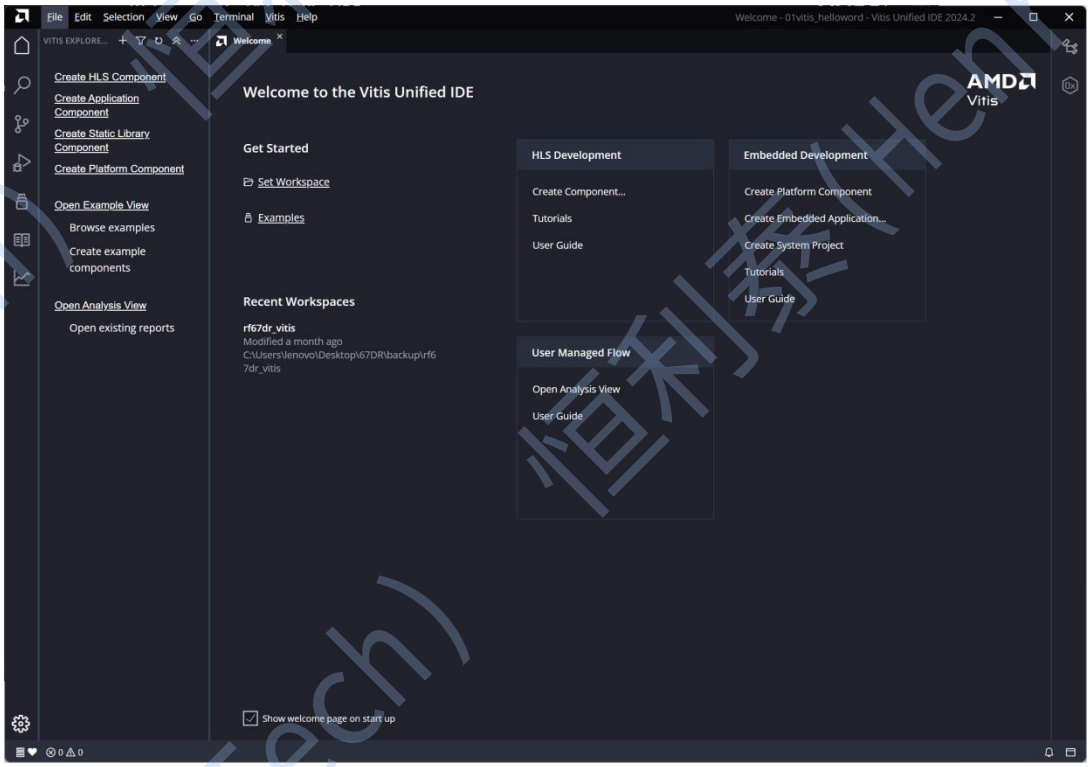
我们首先设置Workspace，界面点击Set Workspace，找到刚才复制了xsa文件的01vitis_helloword文件夹，点击下方选择文件夹完成目录设置：



接着弹出提示，未识别到工作簿的版本，我们点击Update更新即可：

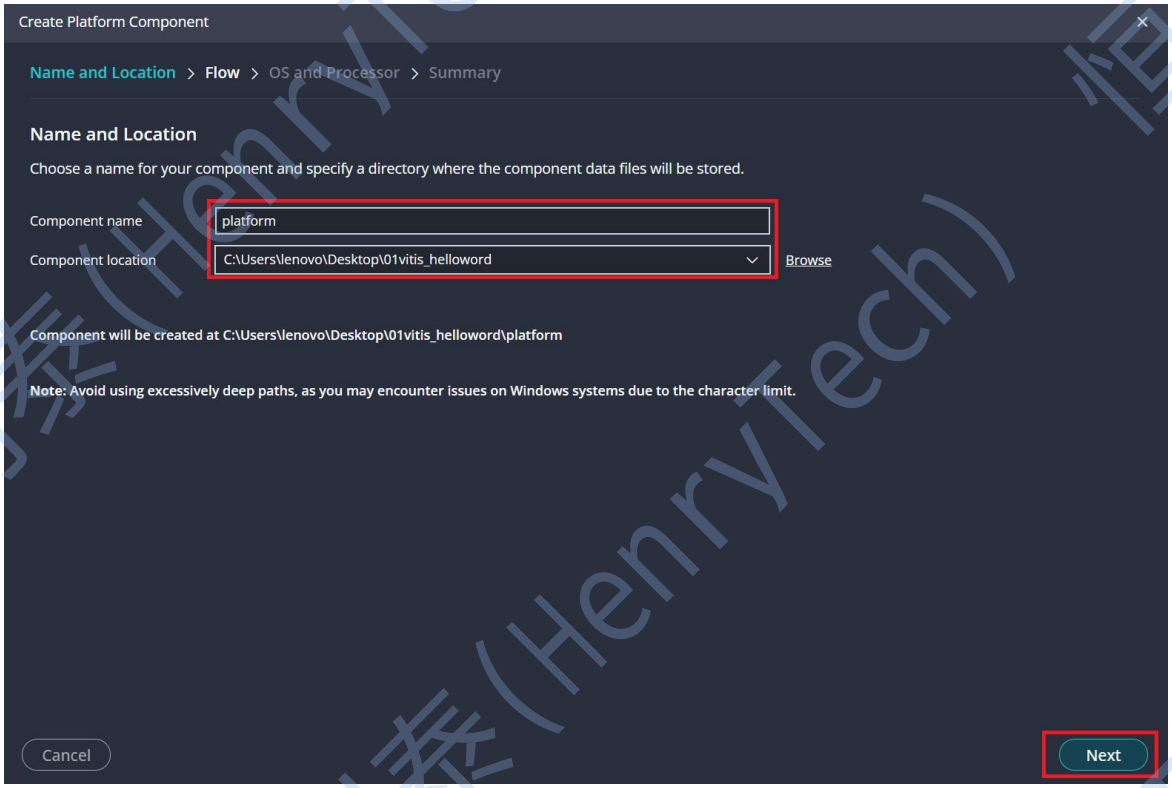


到此我们工程文件夹建立完成：

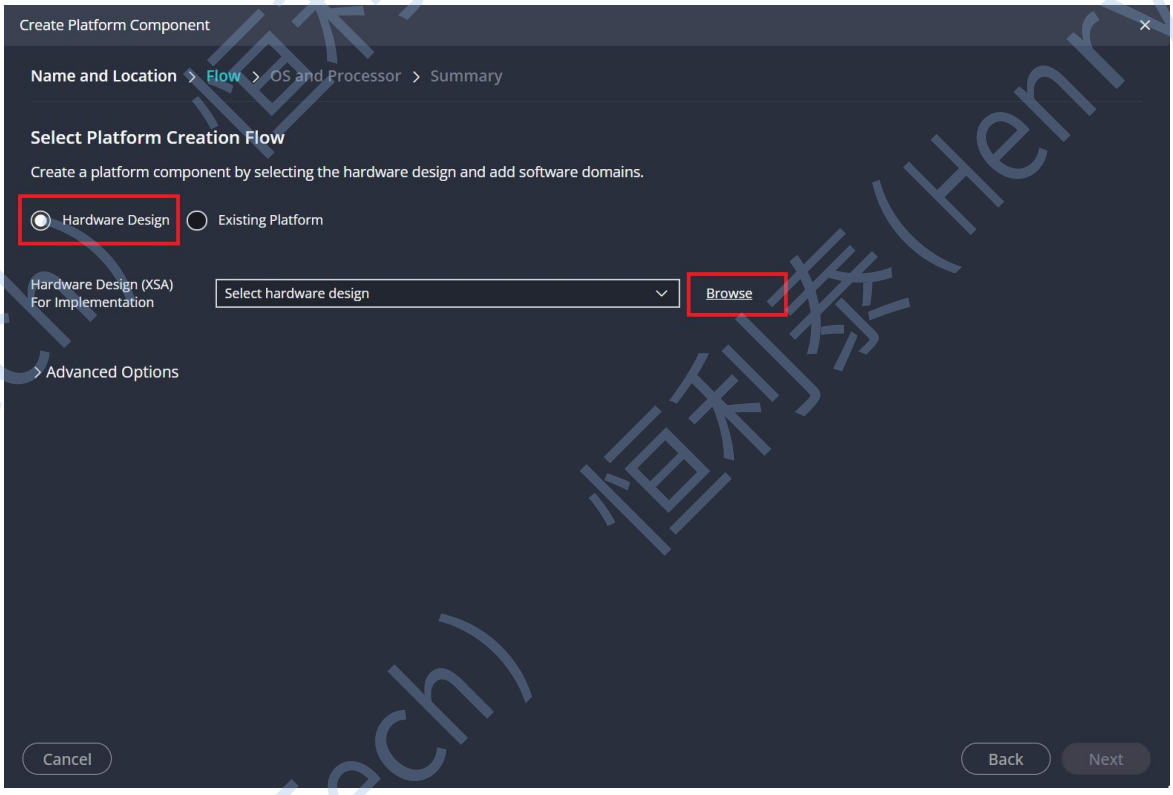


十二、新建Platform Component

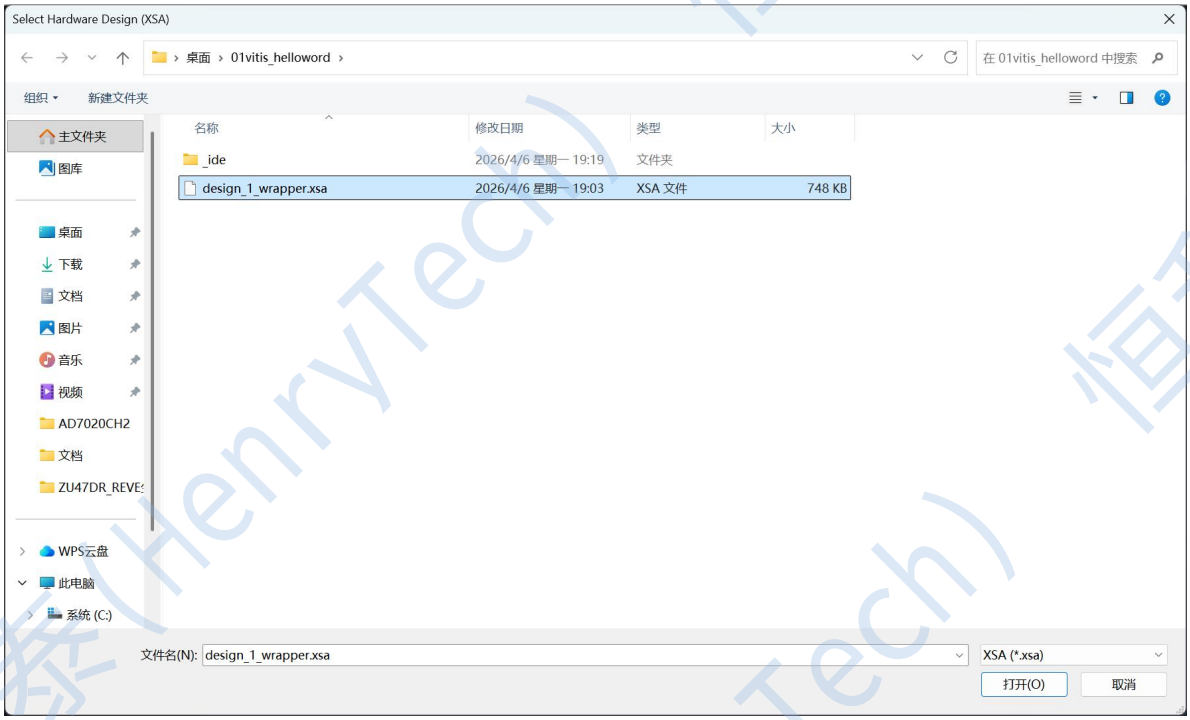
针对嵌入式工程开发，我们需要一个硬件工程平台(Platform component)。基于此平台，再新建应用工程。我们在新建好的工程界面中找到Create Platform Component，弹出新建界面，第一页，cpmponet name和component location（硬件平台名称和目录）这两项设置默认，不做更改，默认为当前vitis的workspace位置，我们直接点击Next：



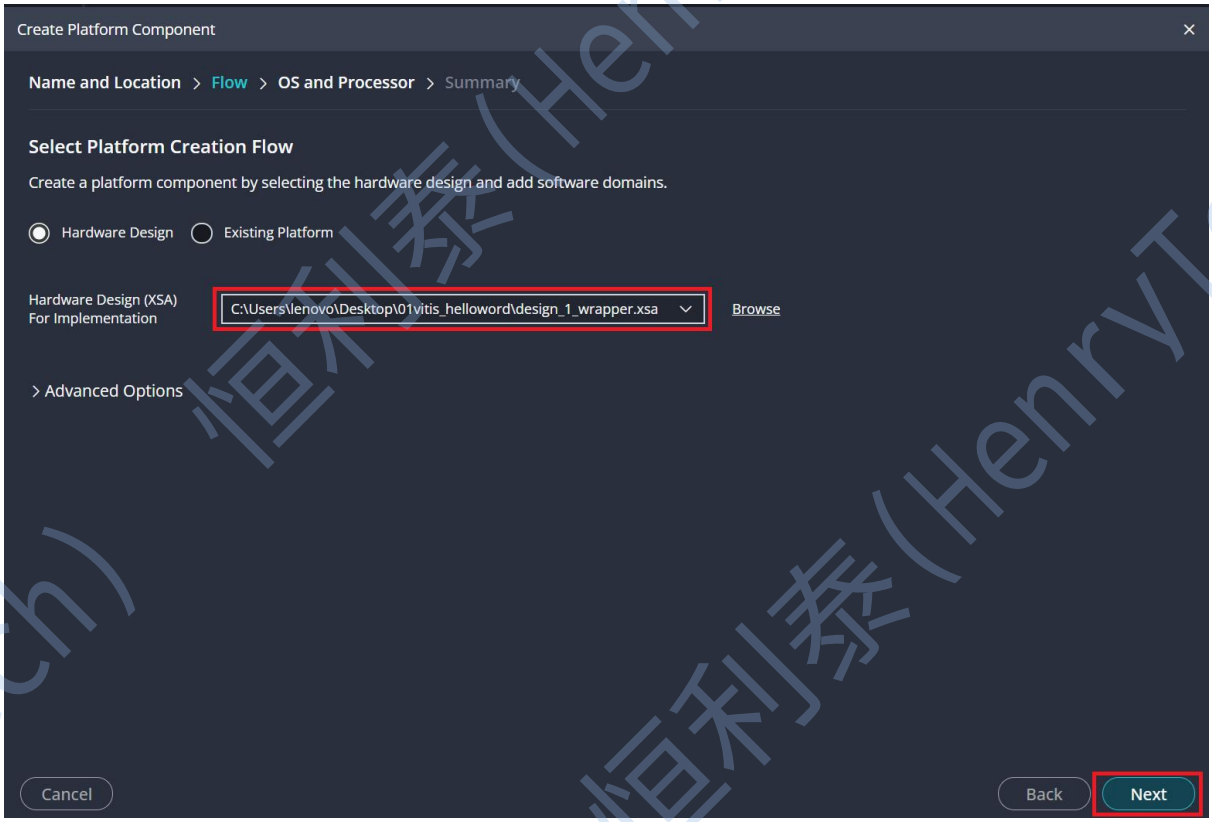
第二页，默认选Hardware Design(硬件设计)，下方选择XSA文件我们点击右侧Brose浏览：



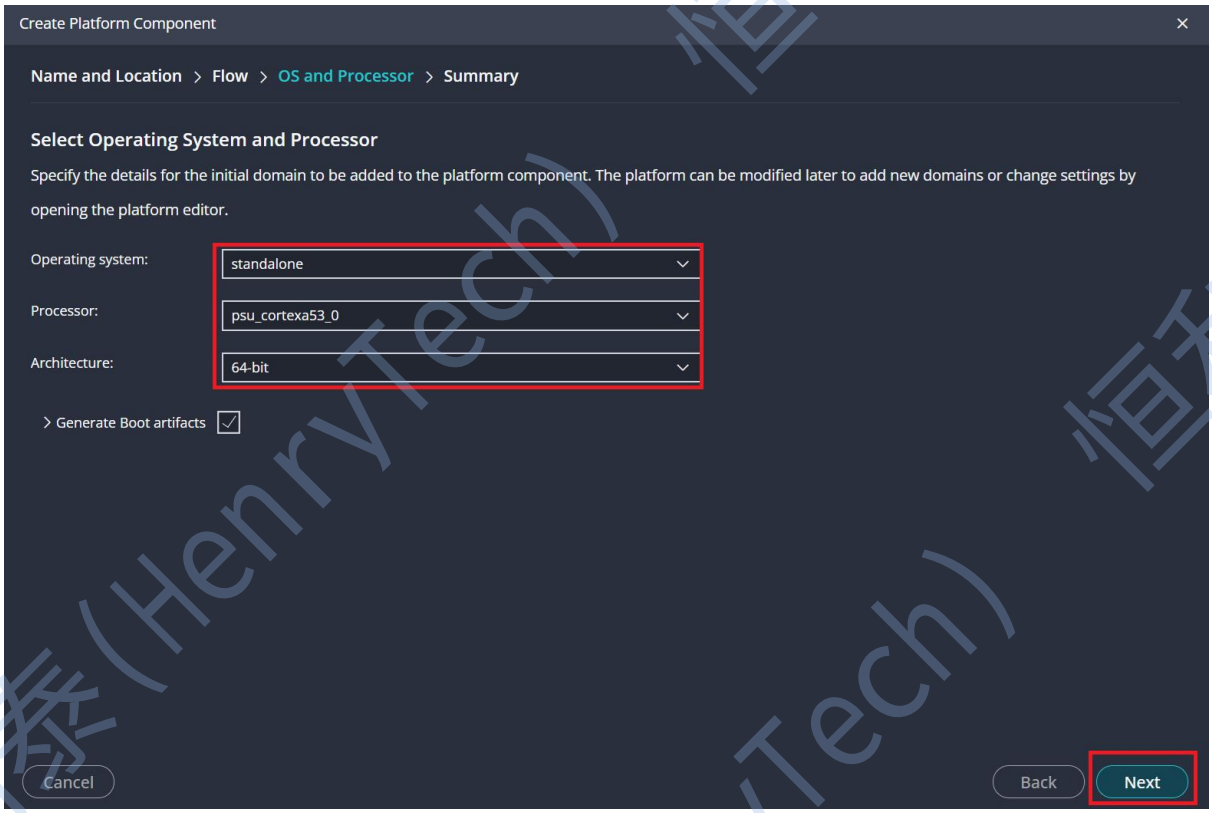
找到我们之前复制的xsa文件点击打开：



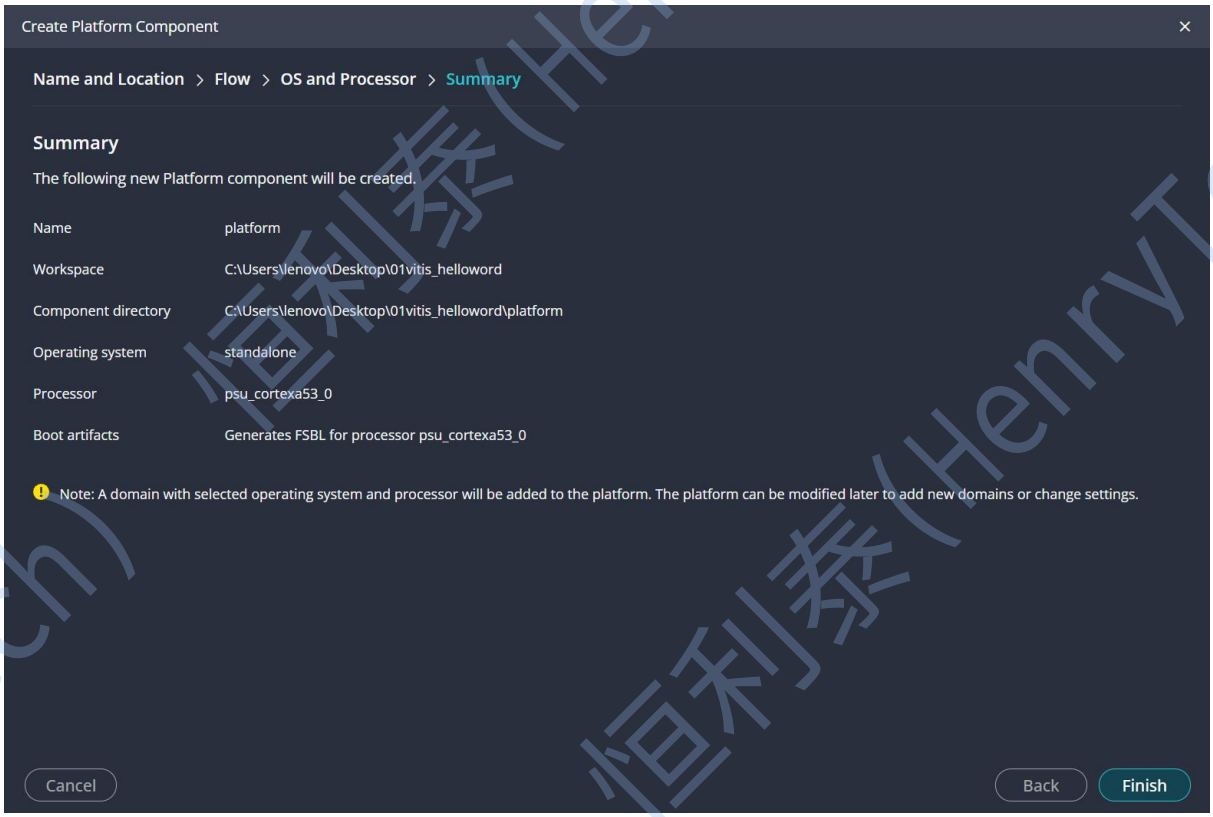
选择xsa之后我们点击Next：



进入到选择平台的操作系统和处理器设置页面，此页面一般不做更改。默认的，Operation system为standalone表示裸机（不使用操作系统），Processor处理器设置为psu_cortexa53_0表示程序运行在A53处理器的第一个核上。我们知道ZYNQ MPSOC有cortex a处理器和r处理器。最后一行的Architecture架构，默认64-Bit。点击Next：



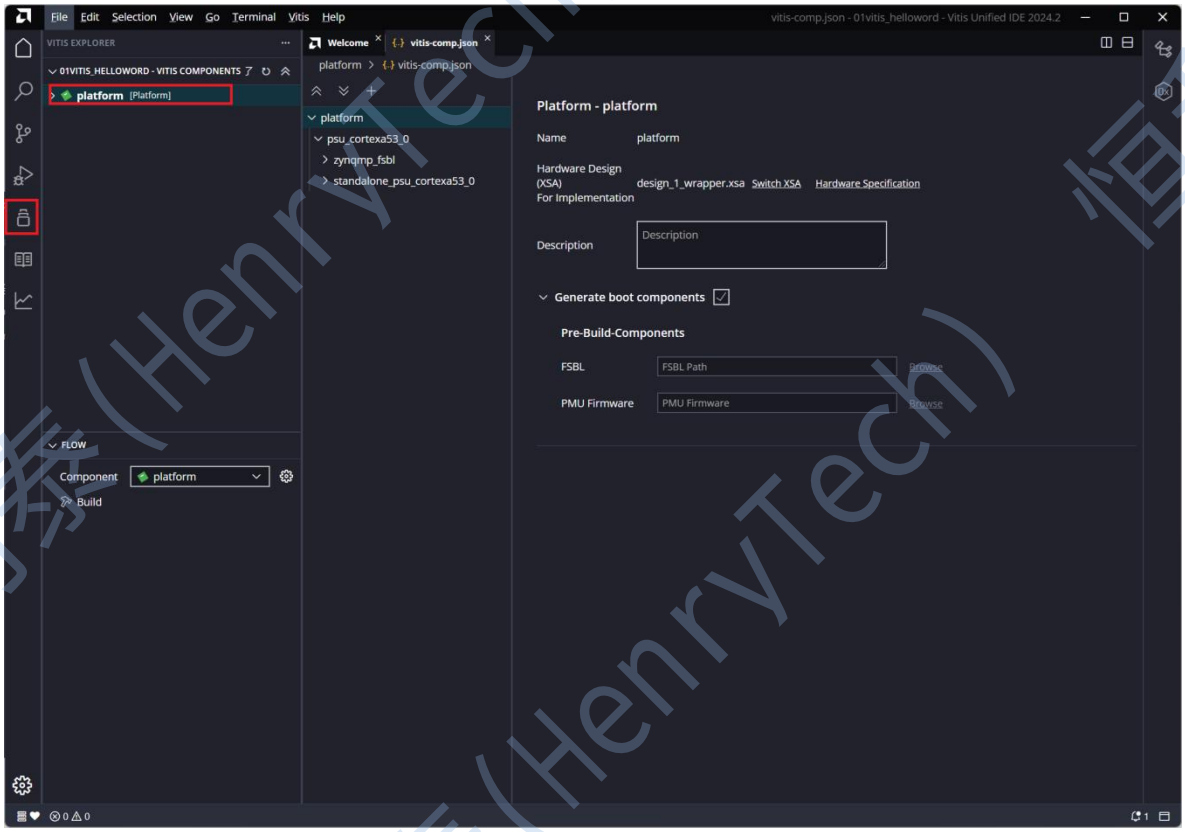
最后一页，确认配置信息无误，点击Finish:



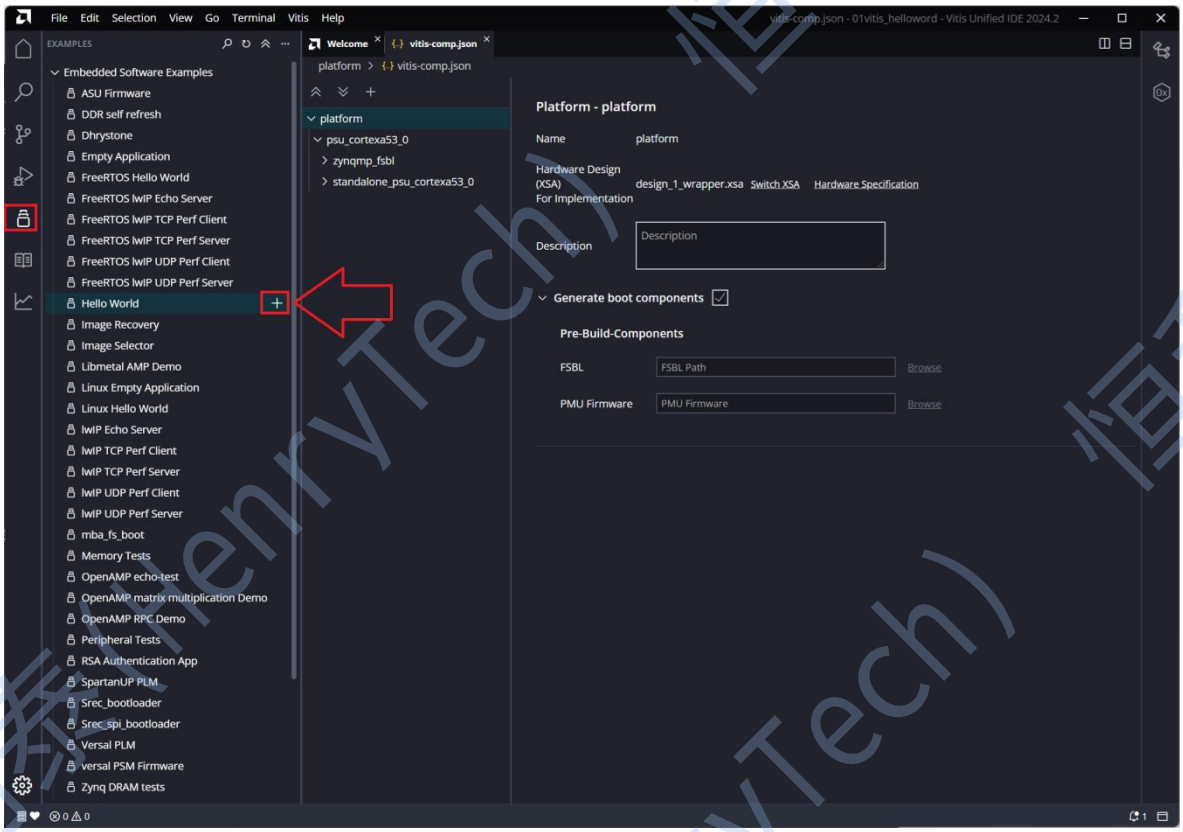
到此我们的platform component平台创建完成。

十三、导入helloworld模板

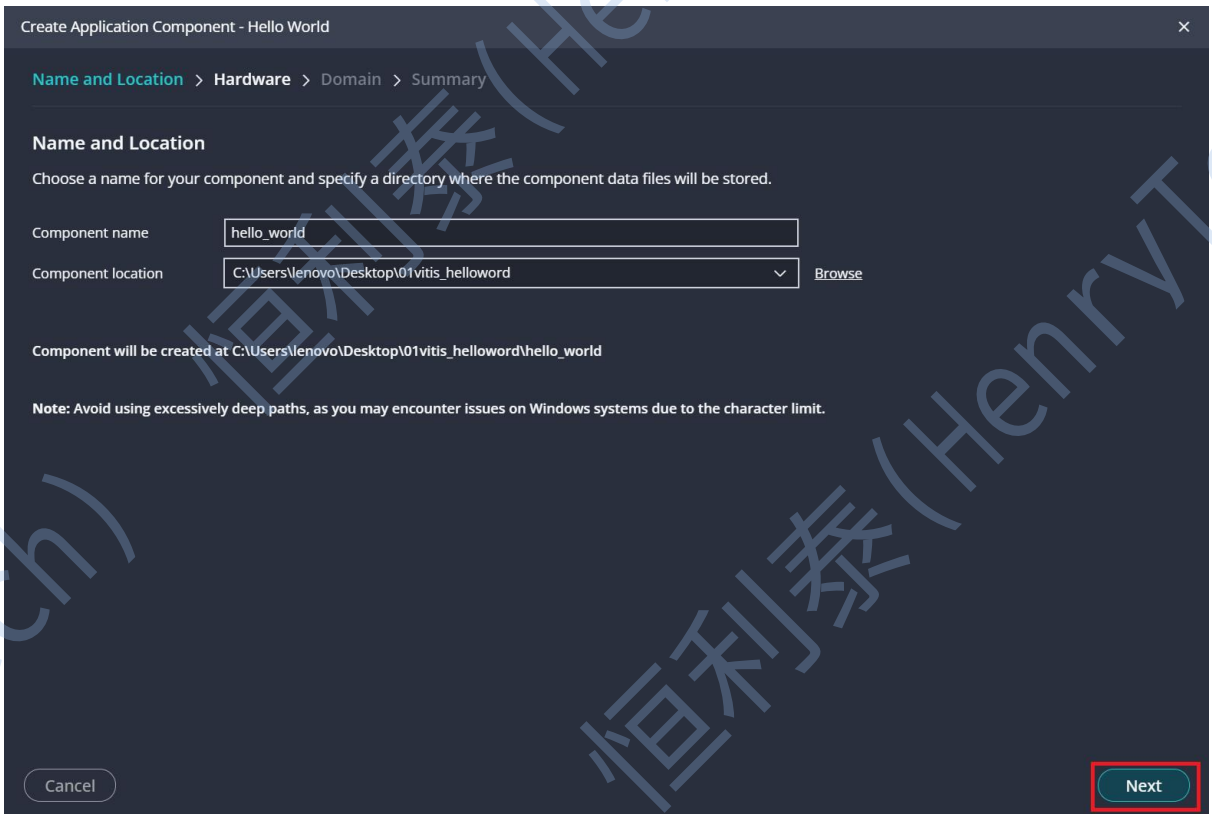
接上一节，我们可以看到工程中已经有了platform，接着我们左侧找到倒数第三个，像优盘一样的图标可以导入application模板：



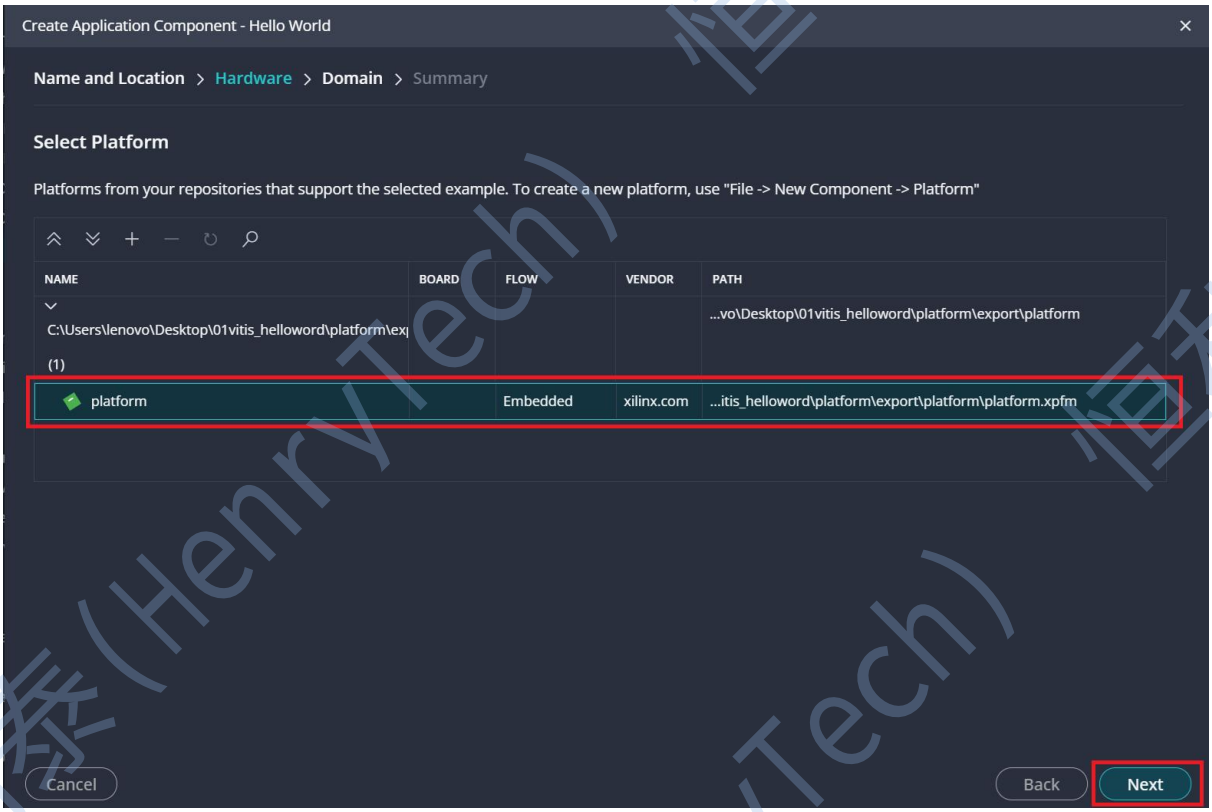
找到Helloworld模板，点击右侧“+”号导入：



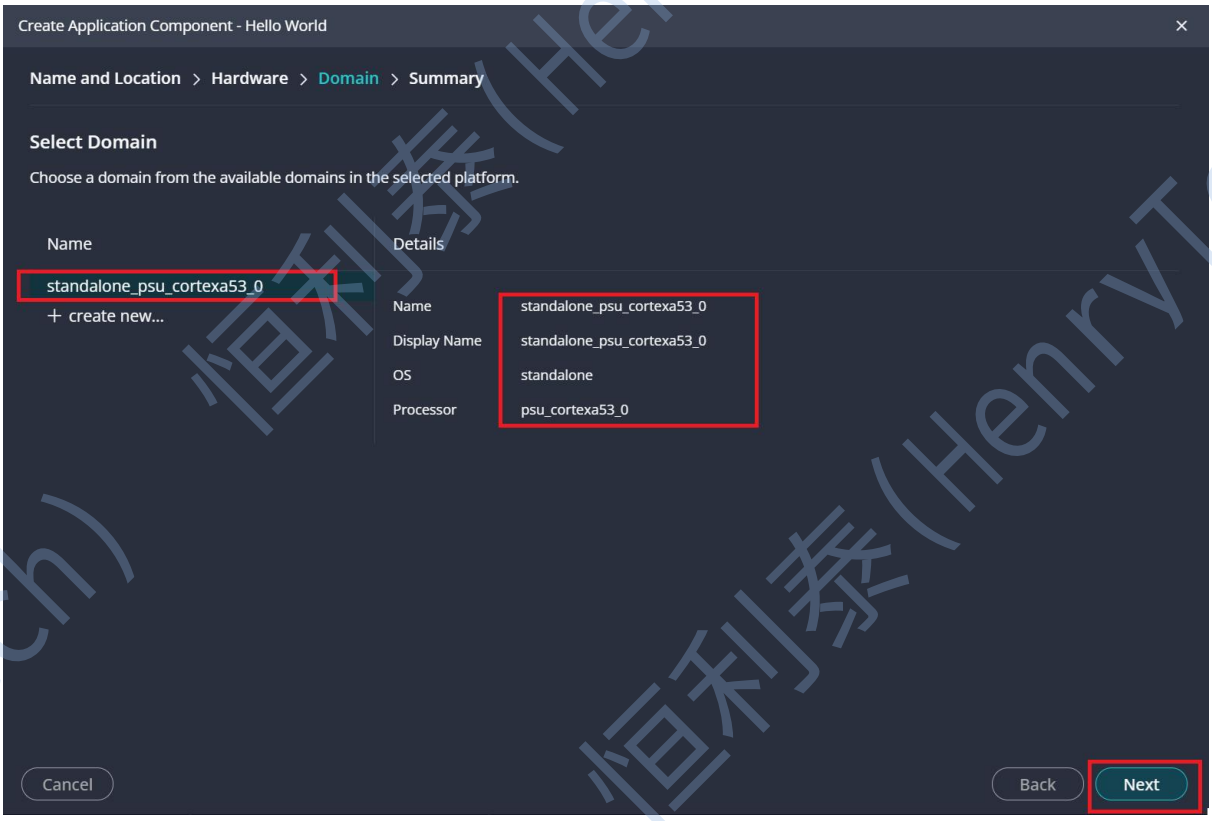
接着弹出创建Application Component应用程序界面，默认的，不做任何更改，直接点击Next进行下一页：



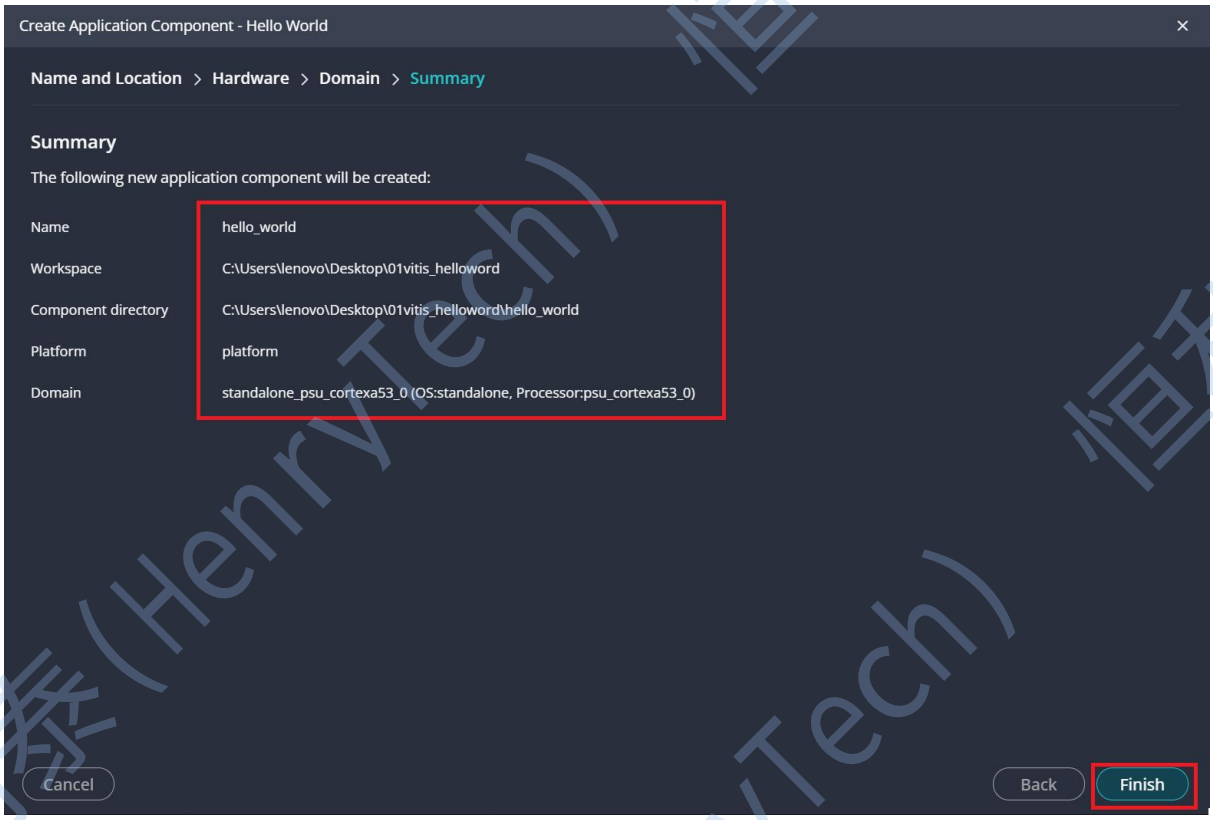
第二页要求选择一个platform，我们选择下方的默认这个即可，点击Next：



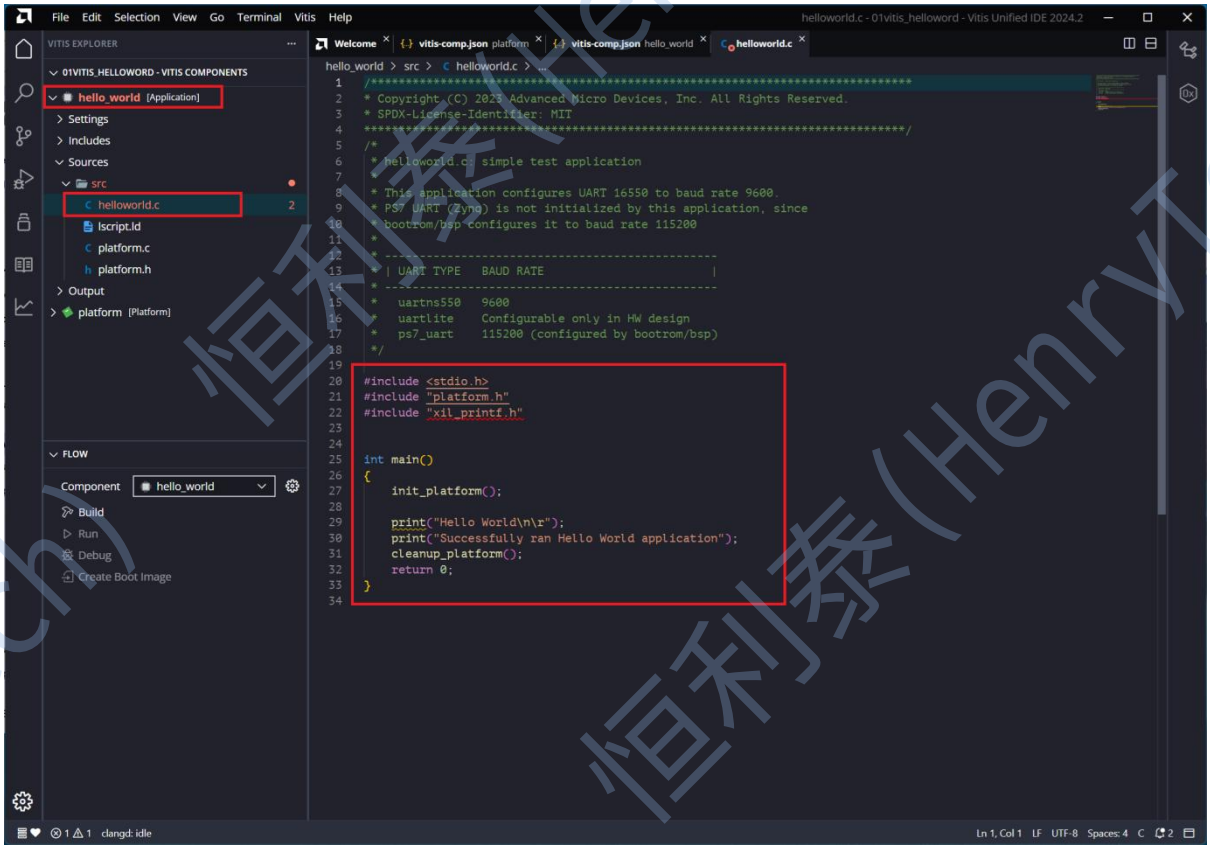
这一页选择Domain，默认的，设置使用我们在之前platform component的时候设置的处理器和操作系统配置，使用裸机，cortexa53_0处理器上。点击Next:



最后一页概况，确认信息后，直接点击Finish完成helloworld模板工程创建：



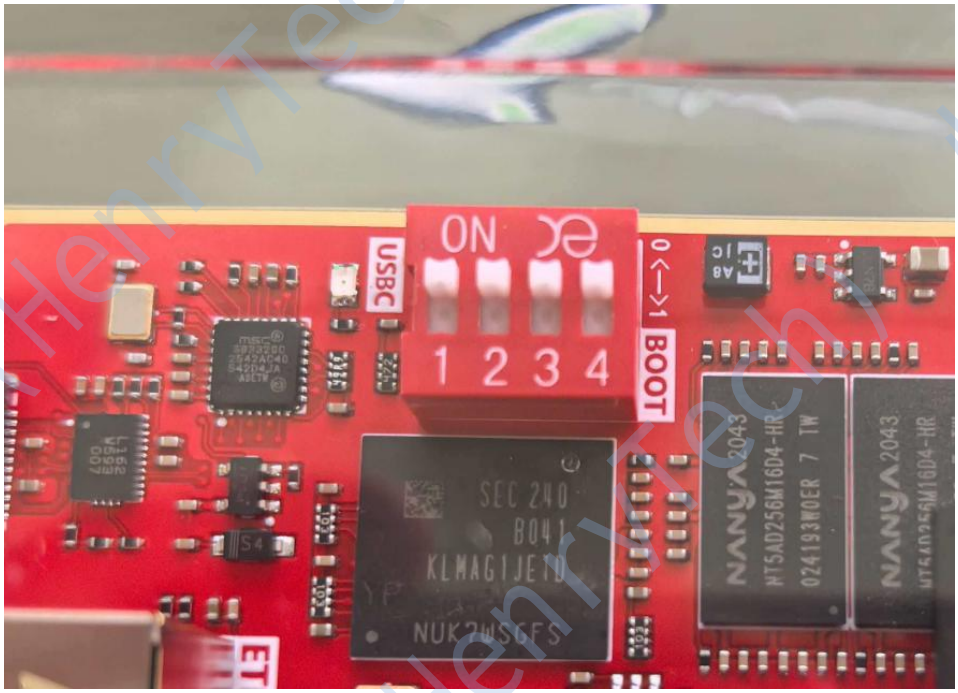
到这里，就完成了helloworld裸机例程创建。工程建好后，左边打开helloworld目录的helloworld.c即可看到源代码：



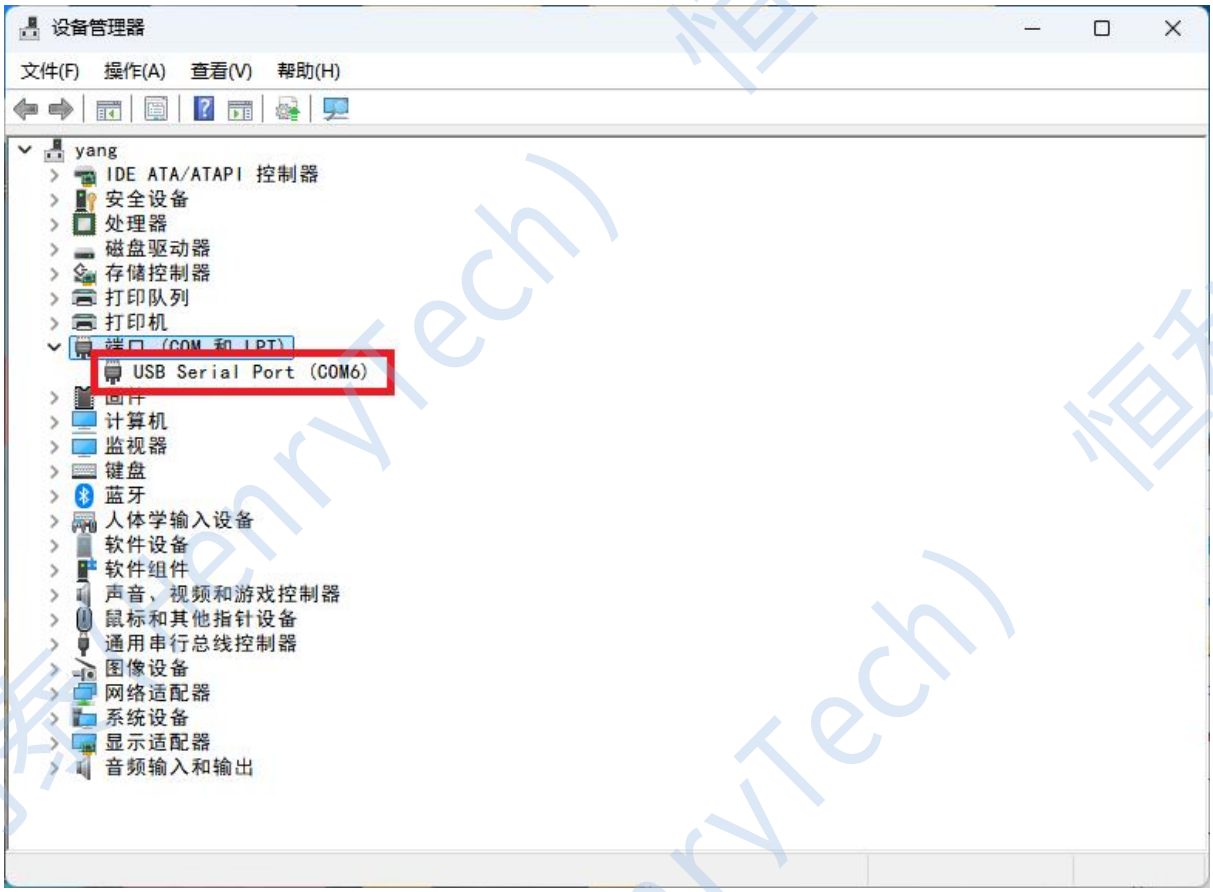
下一节我们将helloworld运行起来。

十四、运行调试代码

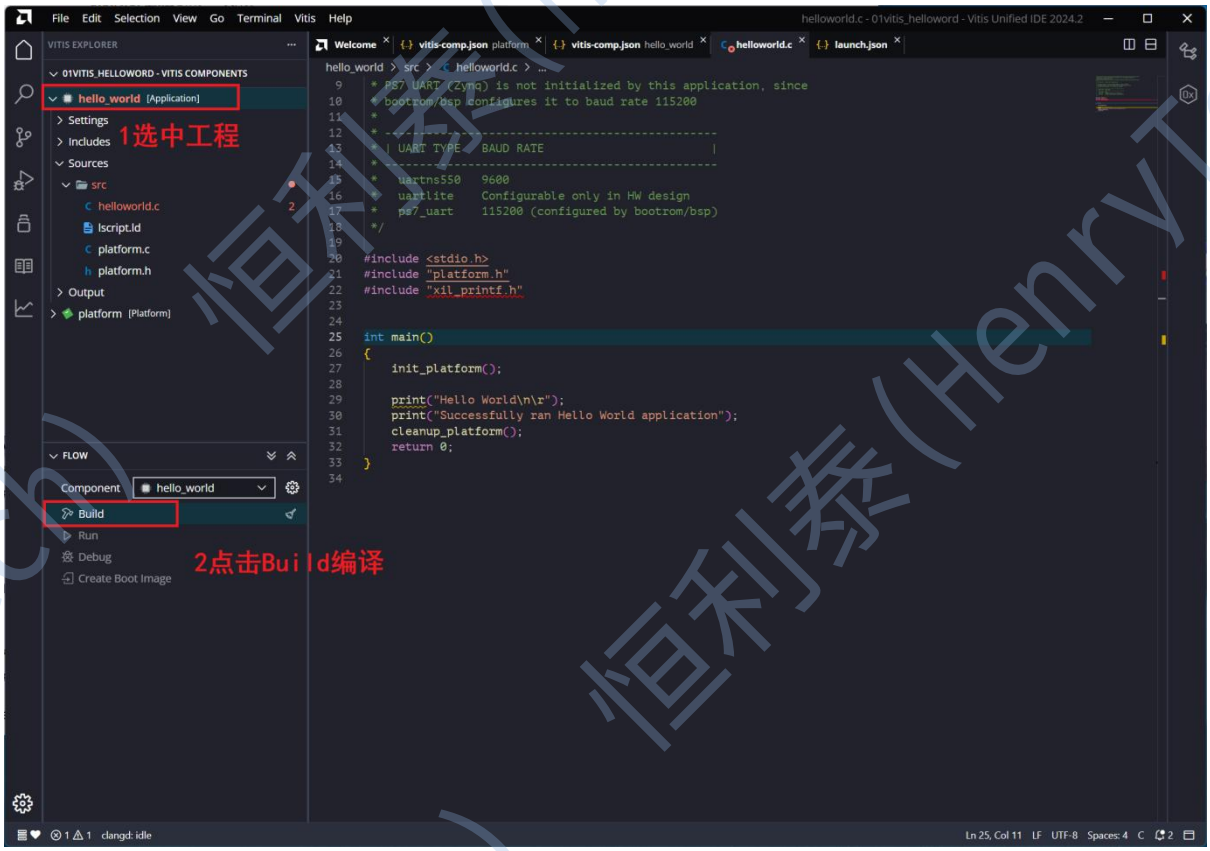
我们上一节完成vitis的helloword工程新建，完成了代码编译，接着我们开始准备上板调试。在下载到板子调试之前，我们首先设置一下BOOT开关。我们在开发板上电开机之前先将BOOT全部设置为0，也就是0000，对应开发板上BOOT丝印是JTAG启动模式，拨码都在ON KE一侧即可：



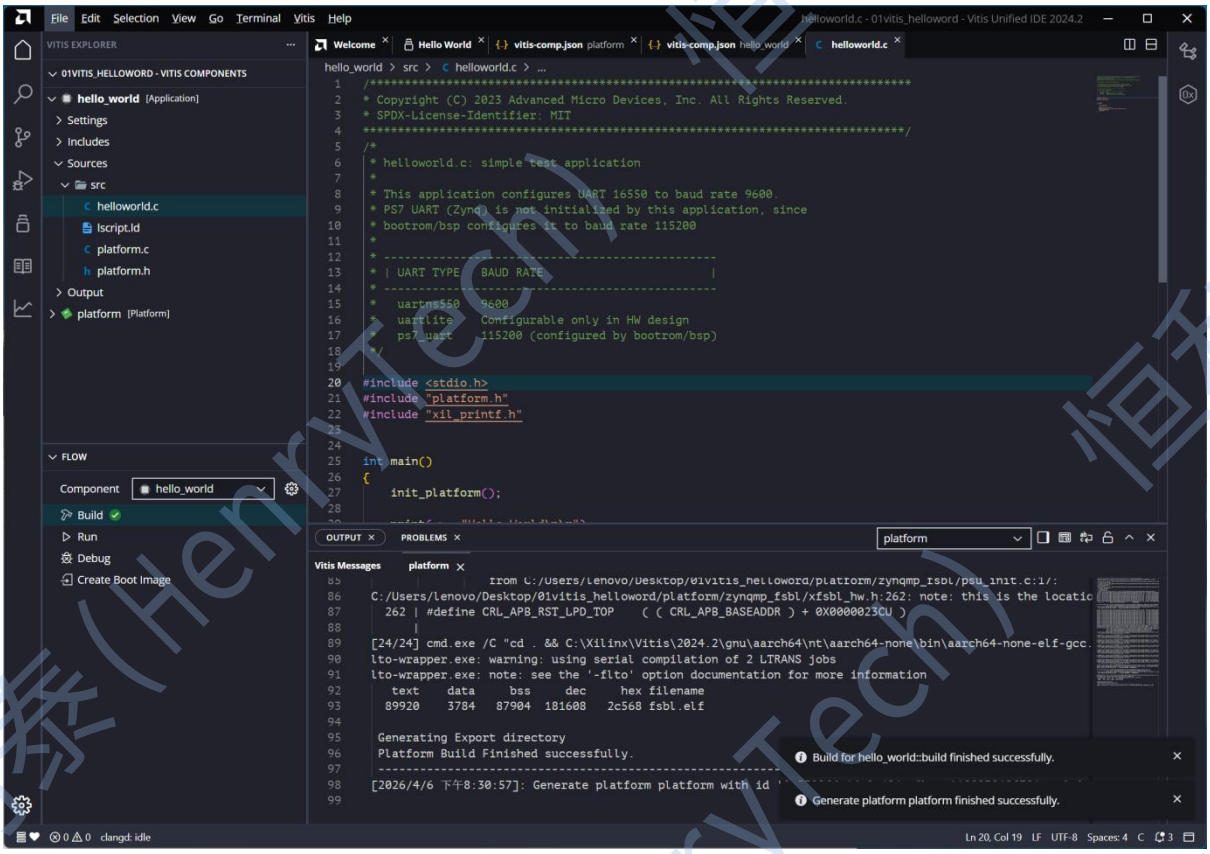
我们插上开发板的DEBUG的TYPE-C USB接口到电脑。我们helloword打印就通过DEBUG口的串口功能打印。连接我们的DC-IN电源接口上电，按下POWER按钮，长按大约2-3秒给开发板启动开机后，我们在windows设备管理器下，查看我们串口设备号：



我们在运行调试前先编译hello_world工程。我们选中左上角的hello_world工程后，点击下方的FLOW中的Build开始编译：

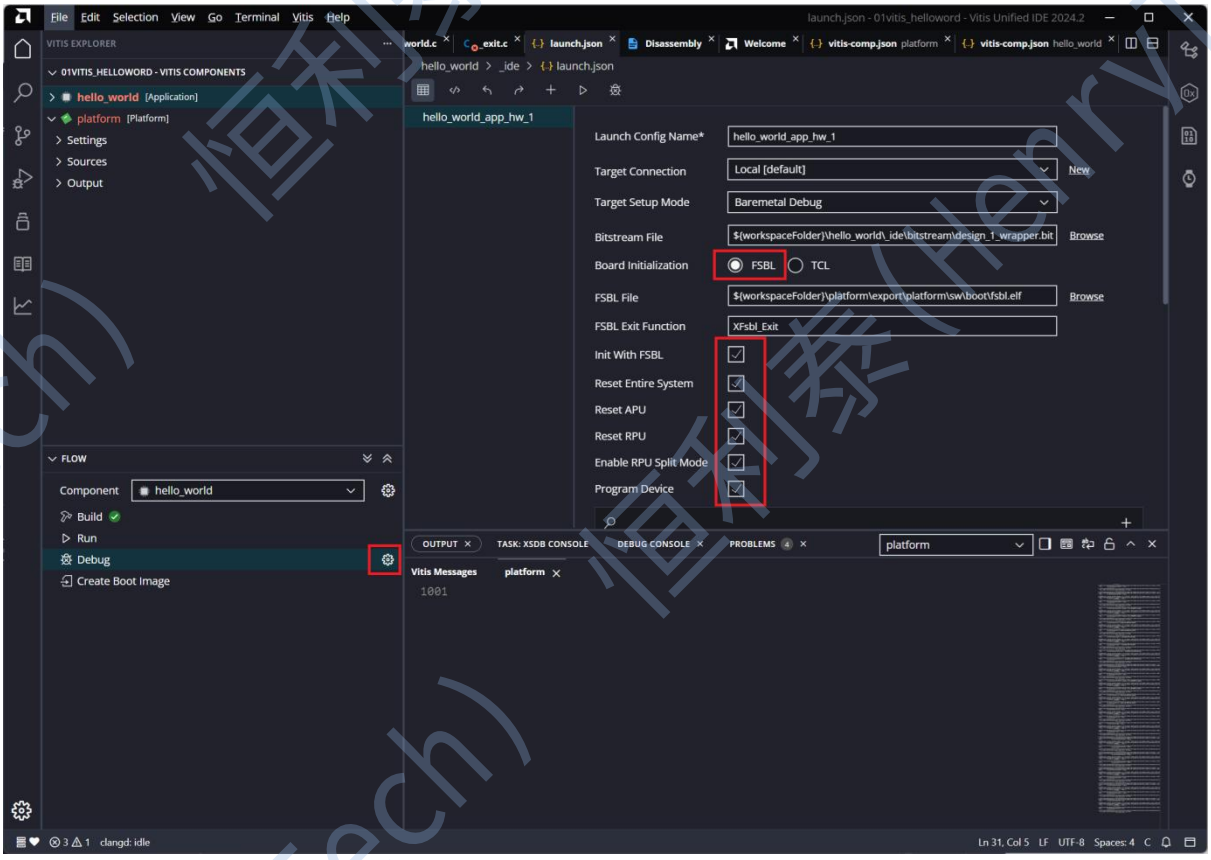


编译完成后下方output显示打印信息，右下角同时会提示编译成功：

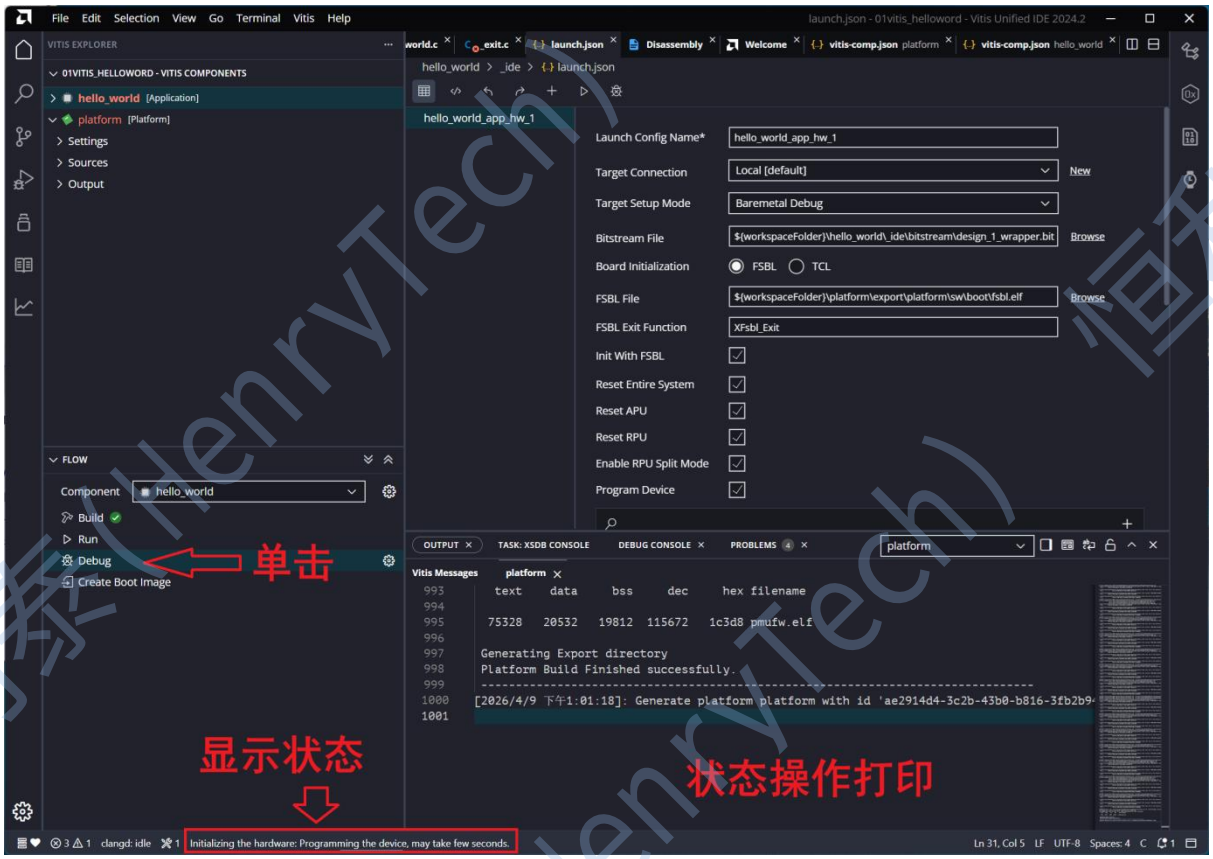


到这里我们就可以上板运行DEBUG。我们运行调试前，打开一个自己熟悉的串口助手工具，这里以《串口助手.exe》为例，工具位于《02软件和相关linux包》中。

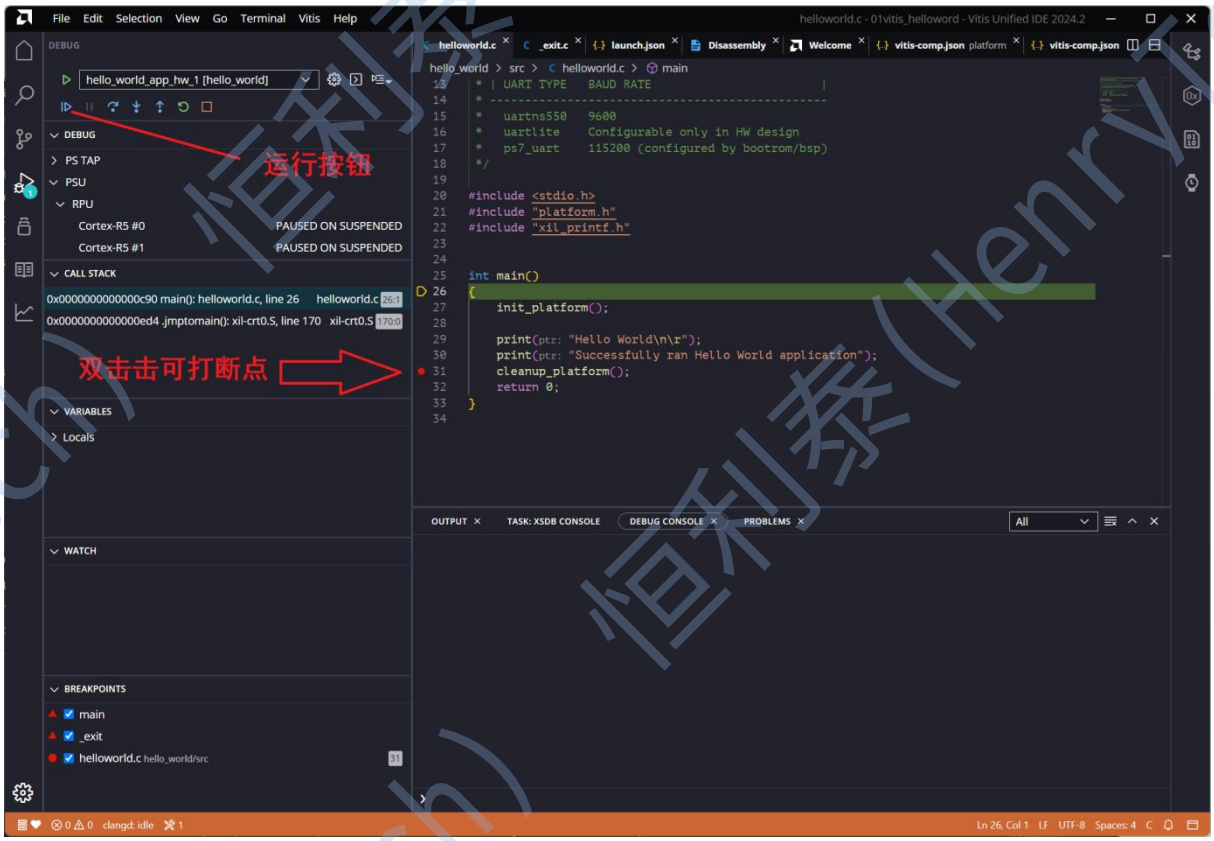
Vitis中，左下方的FLOW当中Debug右侧设置图标（ ）可进行DEBUG设置。DEBUG设置如下，Board initlization初始化选择FSBL方式，下方操作全部勾选。这里特别说明，如果xsa导出不包含Bit文件，上图不需要，或者无法勾选Program Device，或者PL Power up:



我们打开串口助手，然后单FLOW中的Debug启动调试，软件下方提示操作正在初始化，复位系统：



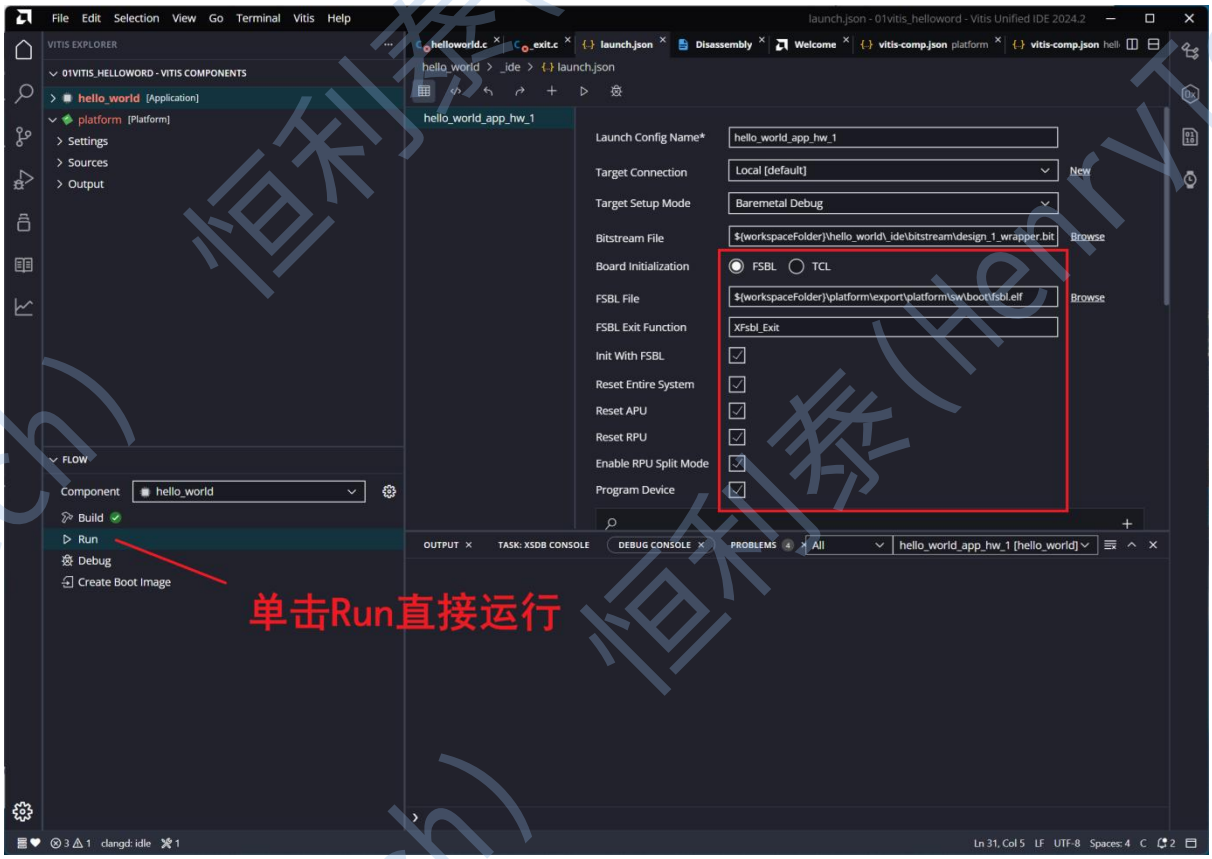
操作初始化完成，进入DEBUG界面：



点击一次运行，代码停止到断点处。同时，串口助手显示打印结果helloworld，打印hello World之前，有FSBL和PMU的调试信息：

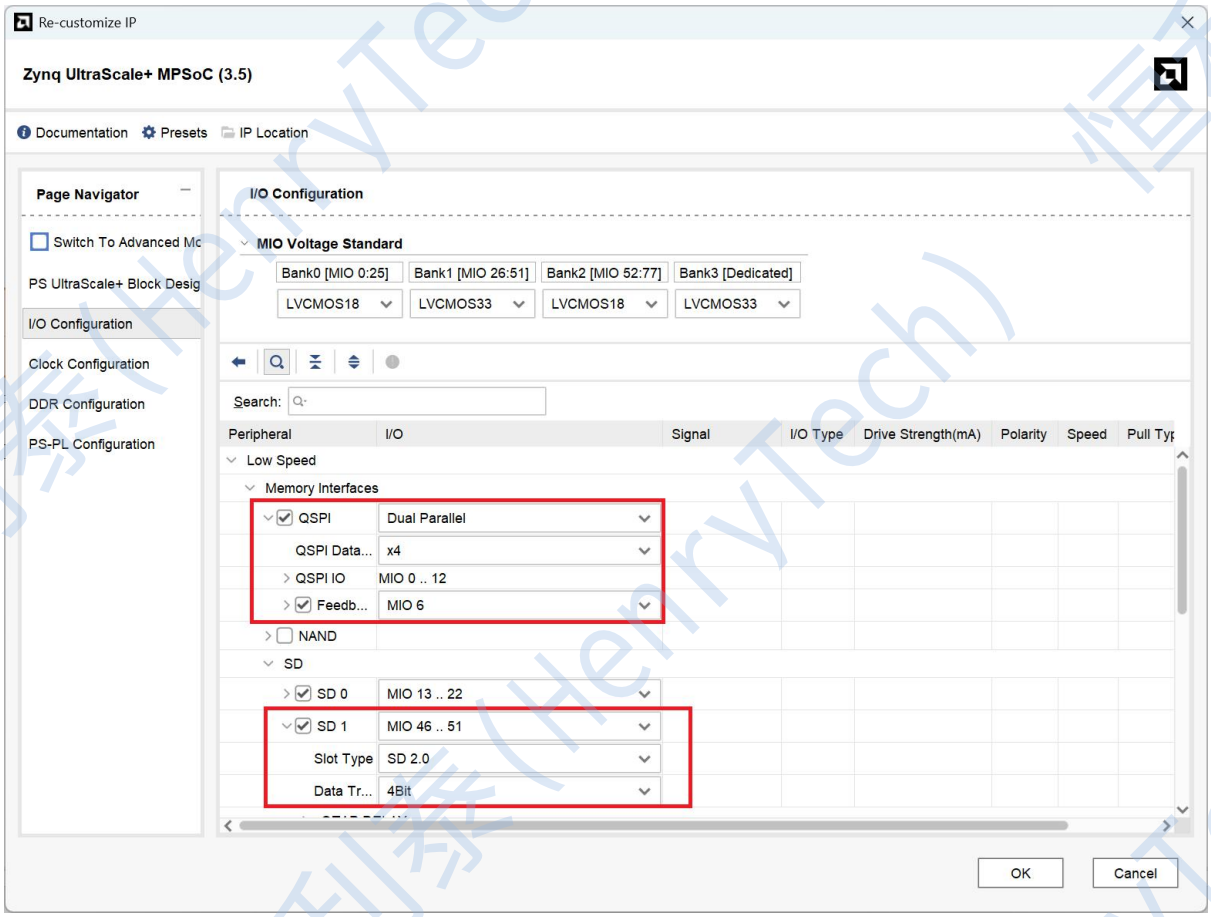
十五、直接运行代码

同样的，FLOW下方Run，右侧设置（⚙）与Debug一致，然后单击Run运行。等待执行完成，串口助手打印结果。注意，由于开发环境原因，Run可能会失败。建议直接Debug进行运行。



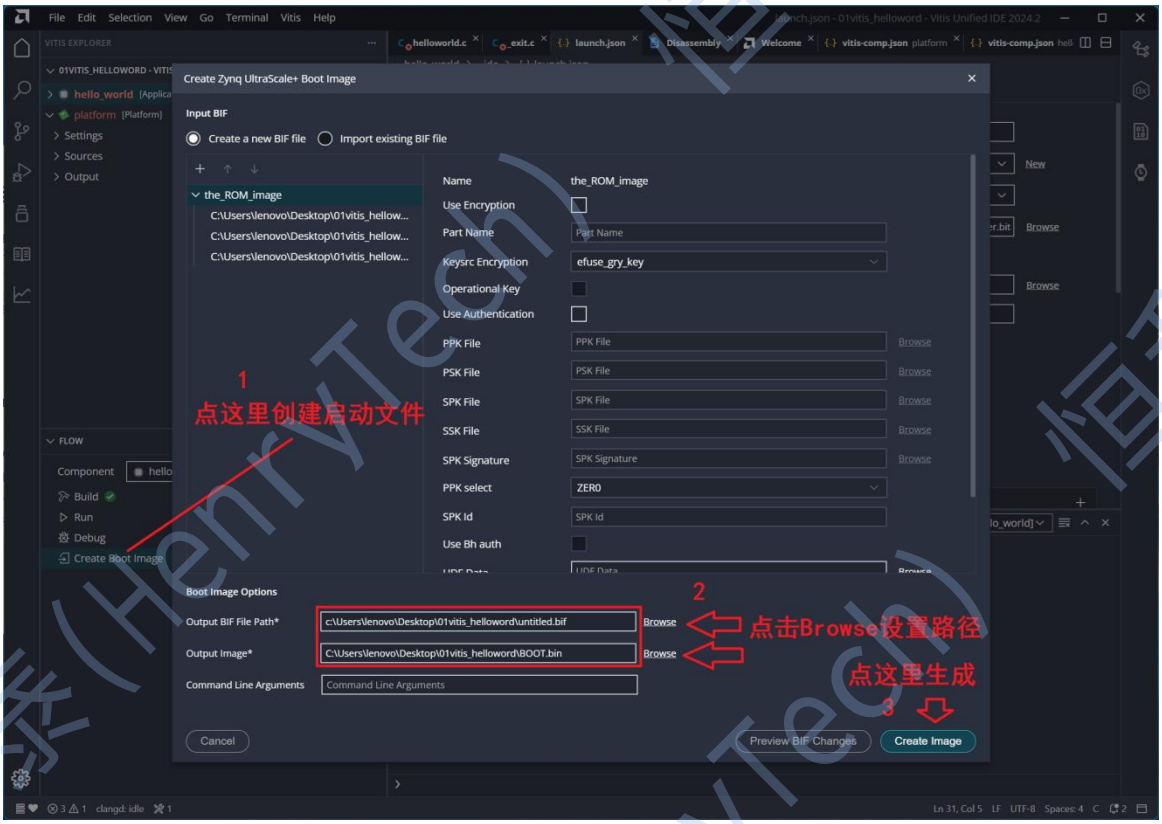
十六、创建SD或者FLASH启动BIN文件

我们如需生成BIN文件烧写到FLASH或者拷贝到FAT32格式化的SD卡（TF卡，容量建议32G内）进行启动，需要制作启动文件。在此之前，先检查ZYNQ配置。基于Vivado工程《rf67dr_ps_base》打开ZYNQ配置，检查如下项：



如果是考虑后续使用EMMC启动，需要配置SD0，与本工程SD0一致。注意，EMMC启动，由于出厂开发板颗粒全新，没有文件系统，需要先移植一个linux系统，操作SD0分区创建以及格式化后才能使用，具体交给有开发经验的客户自行开发。本教程不做讨论。

检查后，我们回到Vitis的FLOW，选择最下方的Create Boot Image创建启动映像开始生成。注意，弹出的前端界面我们需要设置Output BIF File Path路径，bif文件是生成启动文件配置信息的文件，由Vitis自动生成；Output Image就是我们需要的BOOT.BIN启动文件了。同样可以点击Browse浏览选择路径，生成后位于此路径可以找到BOOT.BIN文件。



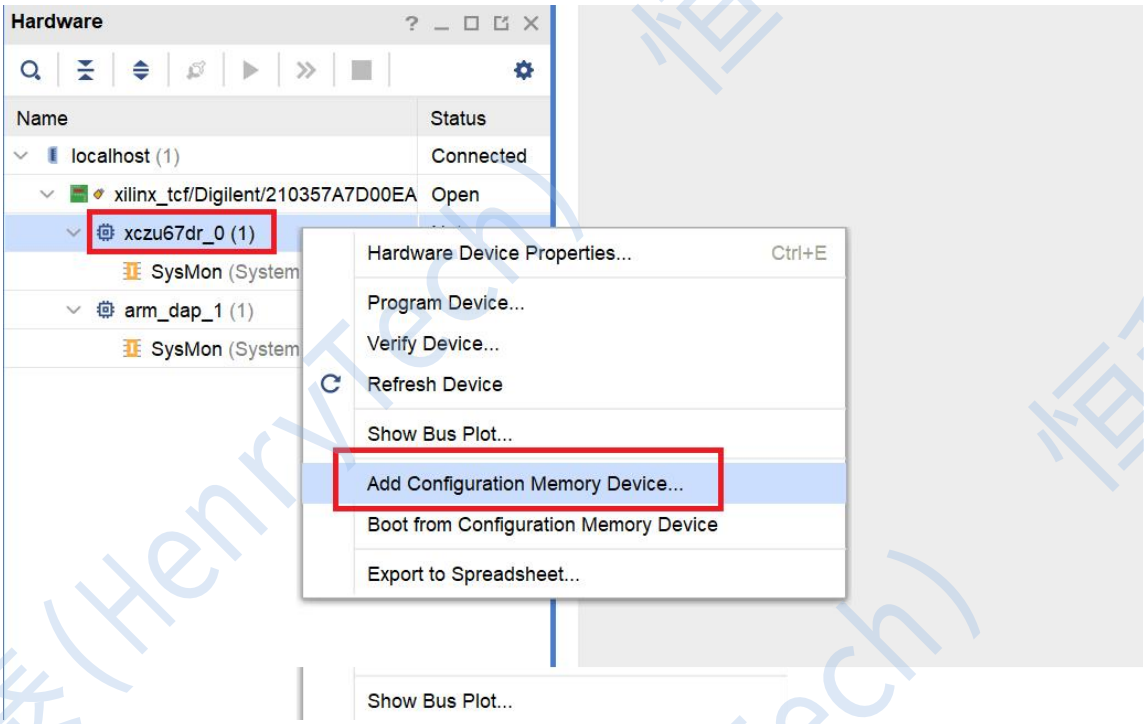
生成后，BOOT.BIN直接拷贝到TF卡（注意必须格式化为FAT32，其他不行）然后设置BOOT开关为SDCARD模式，插入内存卡到TFSD卡座，开机启动，即可运行程序。如需烧写到FLASH，还需找到fsbl文件。

十七、烧写到FLASH

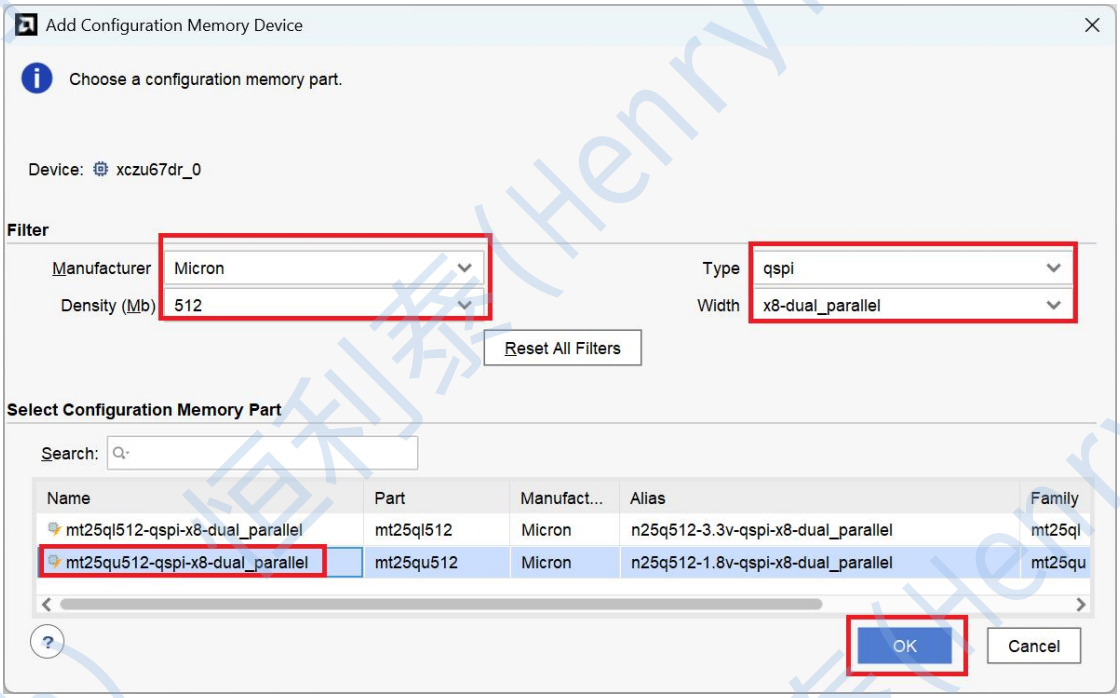
Fsbl编译后位于《01vitis_helloword\platform\zynqmp_fsbl\build\fsbl.elf》位置下。如何烧写FLASH？可以使用Vitis也可以使用Vivado。这里介绍如何使用Vivado进行烧写。Vitis也可以烧写，这里限于篇幅，不做讨论。

开始之前，我们先将开发板的BOOT开关设置到0000全部在ON一侧，为JTAG启动模式，然后开发板再上电开机。

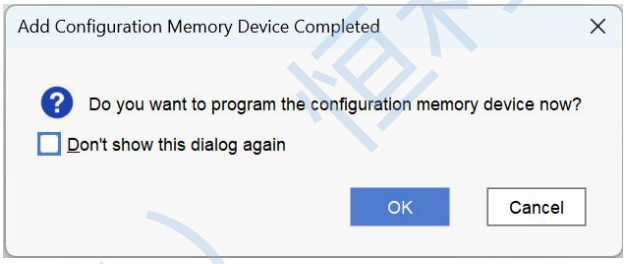
我们打开vivado工程（任意），打开硬件管理，开发板连接DBG接口到电脑，连接开发板。找到FPGA器件（xczu67dr_0）鼠标右键选择Add Configuration Memory Device添加FLASH进来：



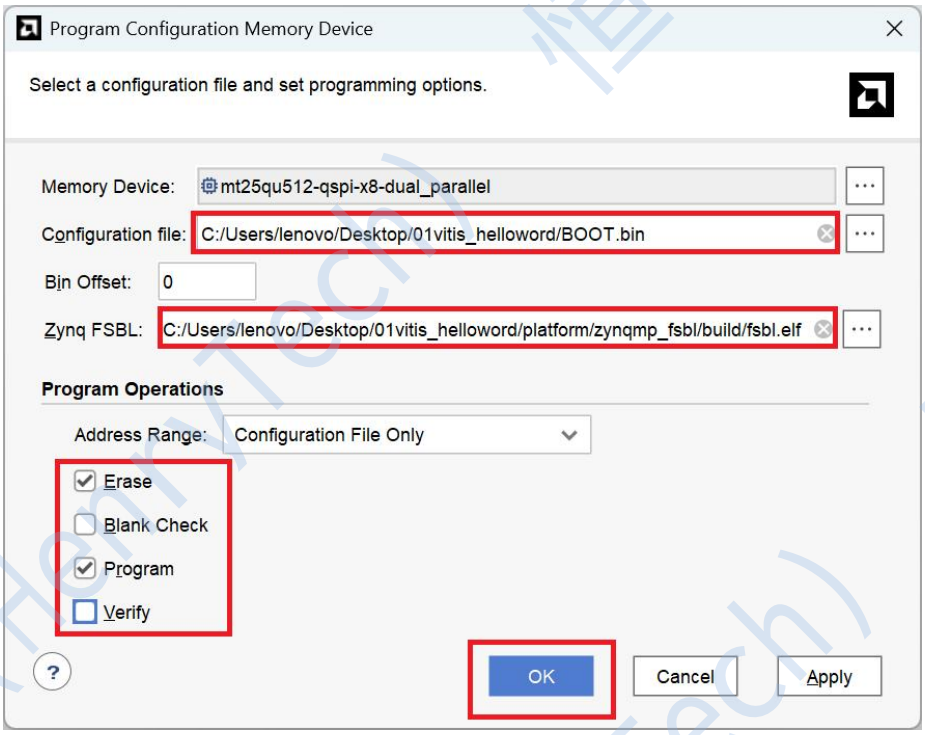
型号筛选如下，选择mt25qu512-qspi-x8-dual_parallel，点击OK完成添加：



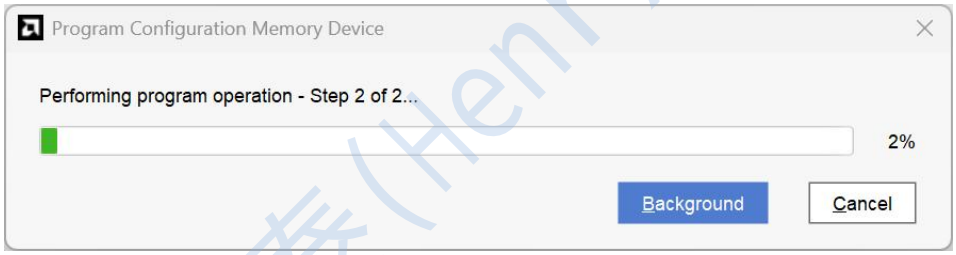
添加后弹出对话框问是否现在烧写，点击OK（或者点击Cancel取消，硬件管理找到Flash设备右键Program Configuration Memory Device）：



烧写设置界面配置，主要是Configuration file和Zynq FSBL文件路径设置如下，路径咱们生成的时候提到过，最后点击OK开始烧写：



开始烧写后，可以看到进度条显示，操作有两个步骤，第一步是擦除，然后才是烧写数据，整个过程可能持续几分钟：



烧写完成FLASH后，设置BOOT开关为QSPI模式，然后重新上电开机即可运行。如果看不到helloworld打印现象，建议先连接串口助手，然后再开机，或者，短按ON开机按钮进行复位重启板子，即可看到程序运行打印结果。

十八、可能的问题及解决方法

笔者从2022.2升级为2024.2版本，遇到过诸多问题，这里列举给大家闭坑：

1、新建工程，设置好workspace后提示未识别到工程版本，必须选择update；

2、新建完platform后等待操作完成，不要过快创建导入helloworld工程；

3、调试Debug设置，这个是最为重要的，Board initlization务必不能使用FSBL！！！！！笔者尝试多次发现不稳定，无法正常使用。所以，这里要检查，建议TCL脚本指令了进行初始化。

使用我们的PS处理器运行RTOS系统

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 VIVADO的工程 3
三、 VITIS导入硬件文件新建平台 3
四、 运行调试代码 9

恒利泰(HenryTech)

一、文档实现功能介绍

本文档实现在我们的ZYNQ的PS(Program System）上，使用我们的ARM核，运行我们的RTOS操作系统，完成多任务运行。我们ZYNQ ZU47DR的PS一共有六个ARM核处理器，其中四个是我们的cortex-A53，另外两个是我们的cortex-R5F。均可运行软件代码。本文档同时对vitis自带的例子如何导入做了说明。

如何使用和最基础的配置ZYNQ的PS请参考《11PS处理器配置及helloword打印》。

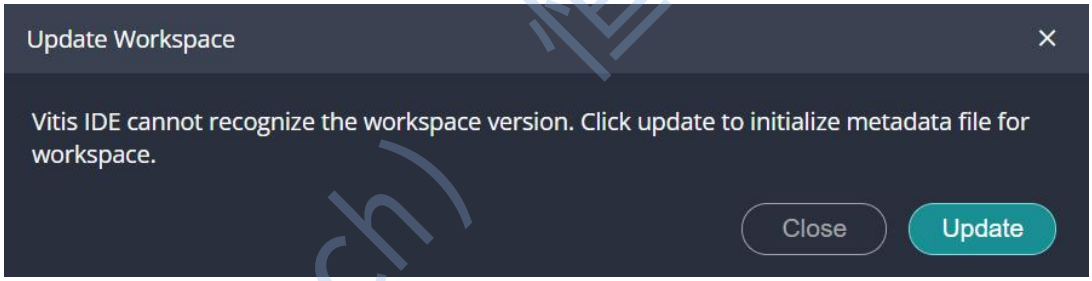
二、VIVADO的工程

工程新建步骤和配置直接沿用我们的ARM部分《11PS处理器配置及helloword打印》配套的工程《rf67dr_ps_base》，不做任何更改。大家后续只需要用到PS部分的工程，可以直接用这个复制然后修改配置。后续我们如果有例程需要更改配置我们会讲解修改的配置。我们将《rf67dr_ps_base》程序导出的xsa文件复制一份备用。

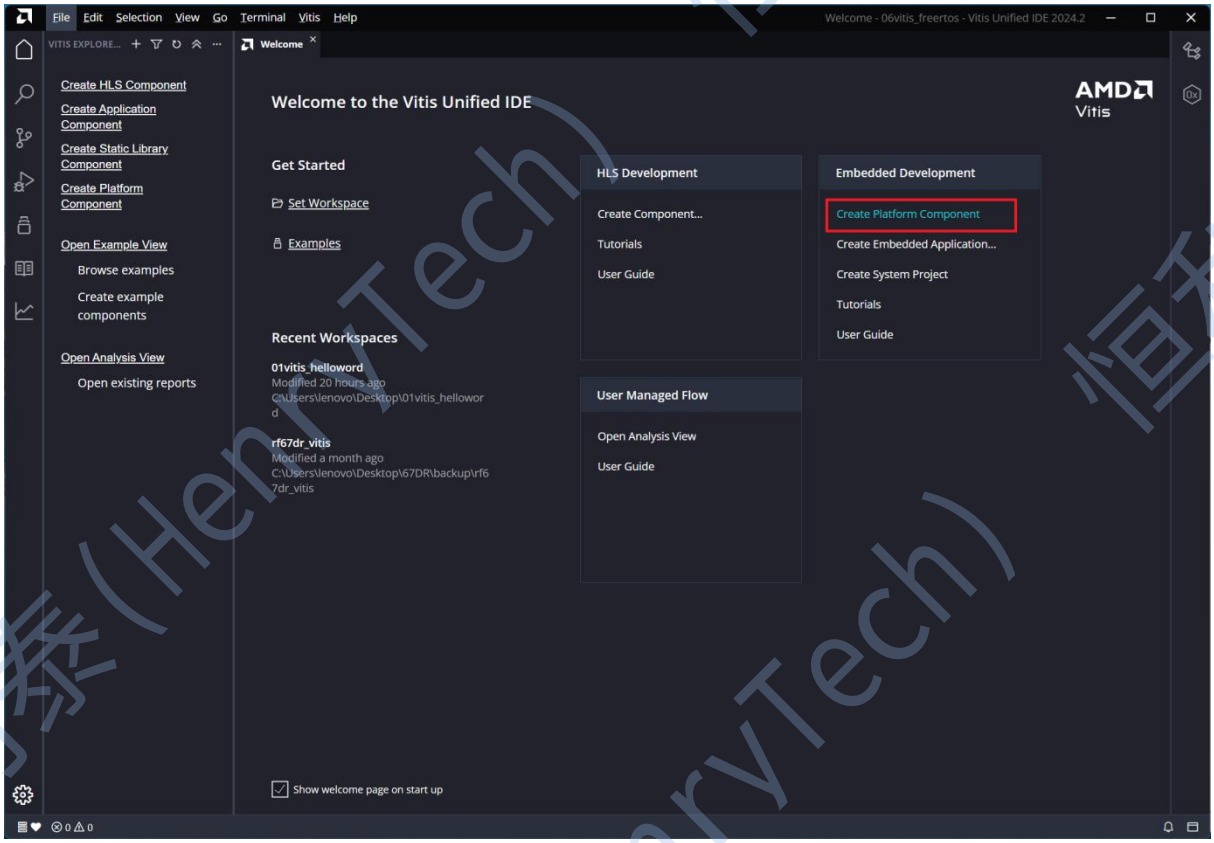
三、VITIS导入硬件文件新建平台

上一节我们导出一个xsa硬件文件。这一节我们使用这个文件新建一个vitis工程，需要用到这个文件。我们启动VITIS软件，接着弹出工作路径选择窗口，我们在桌面上或者硬盘根目录下新建一个文件夹叫作06vitis_freertos，将我们上一节导出的xsa文件复制到新建的这个文件夹下：

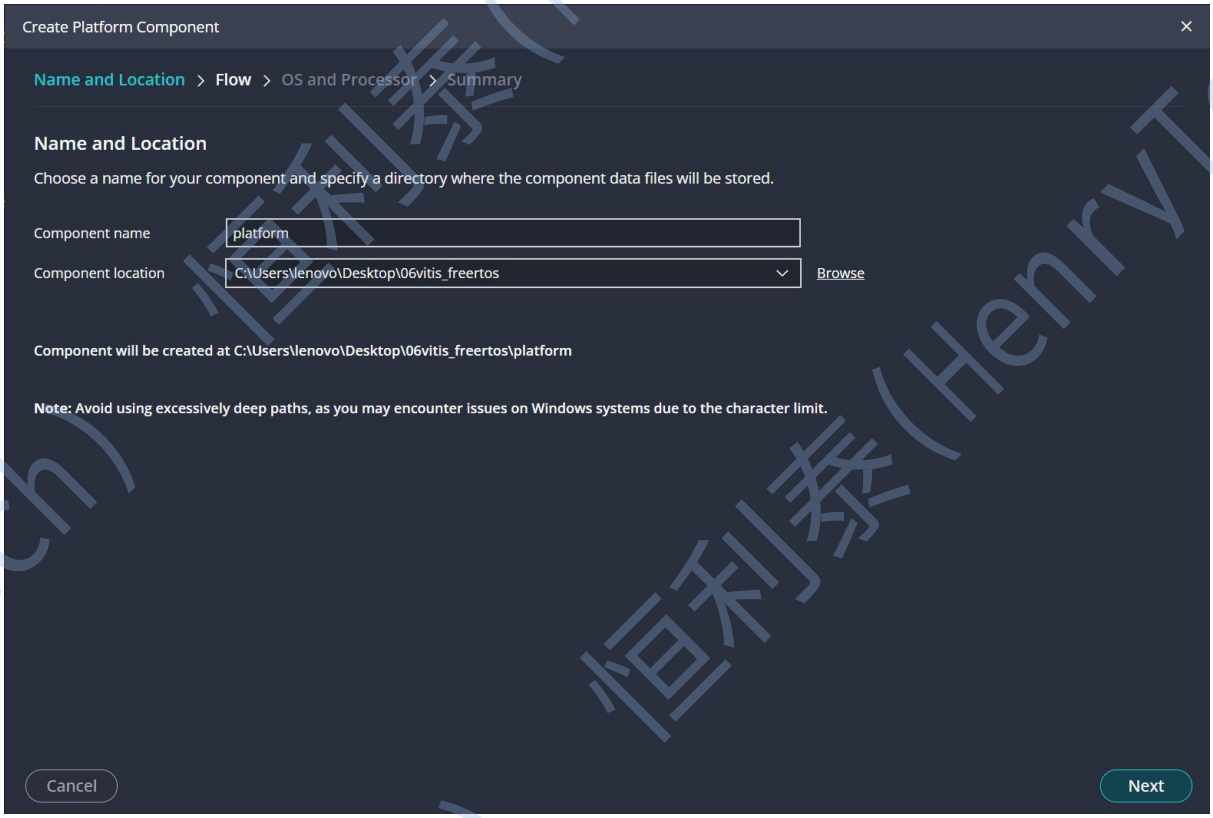
然后我们启动的工程配置workspace选择这个文件夹。同样的，记得弹出update workspace更新选择update：



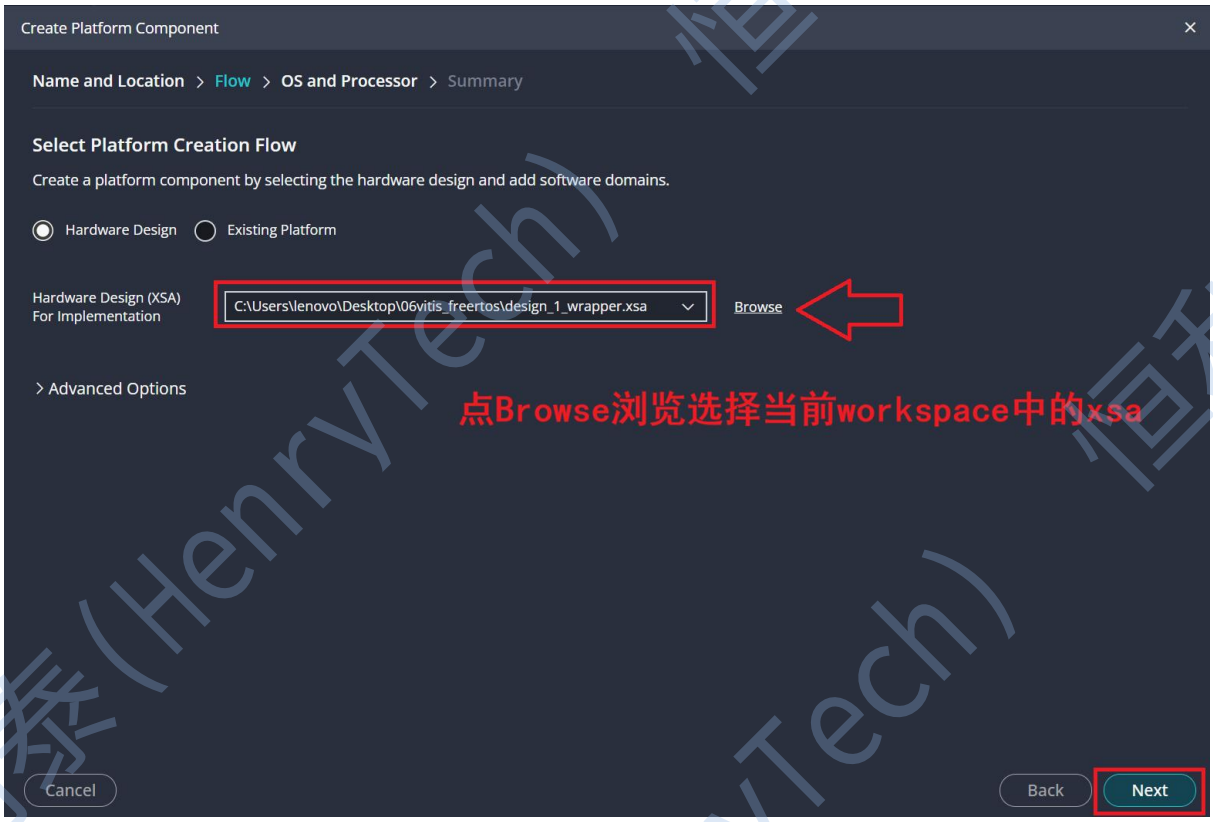
主界面点击Create Platform Component:



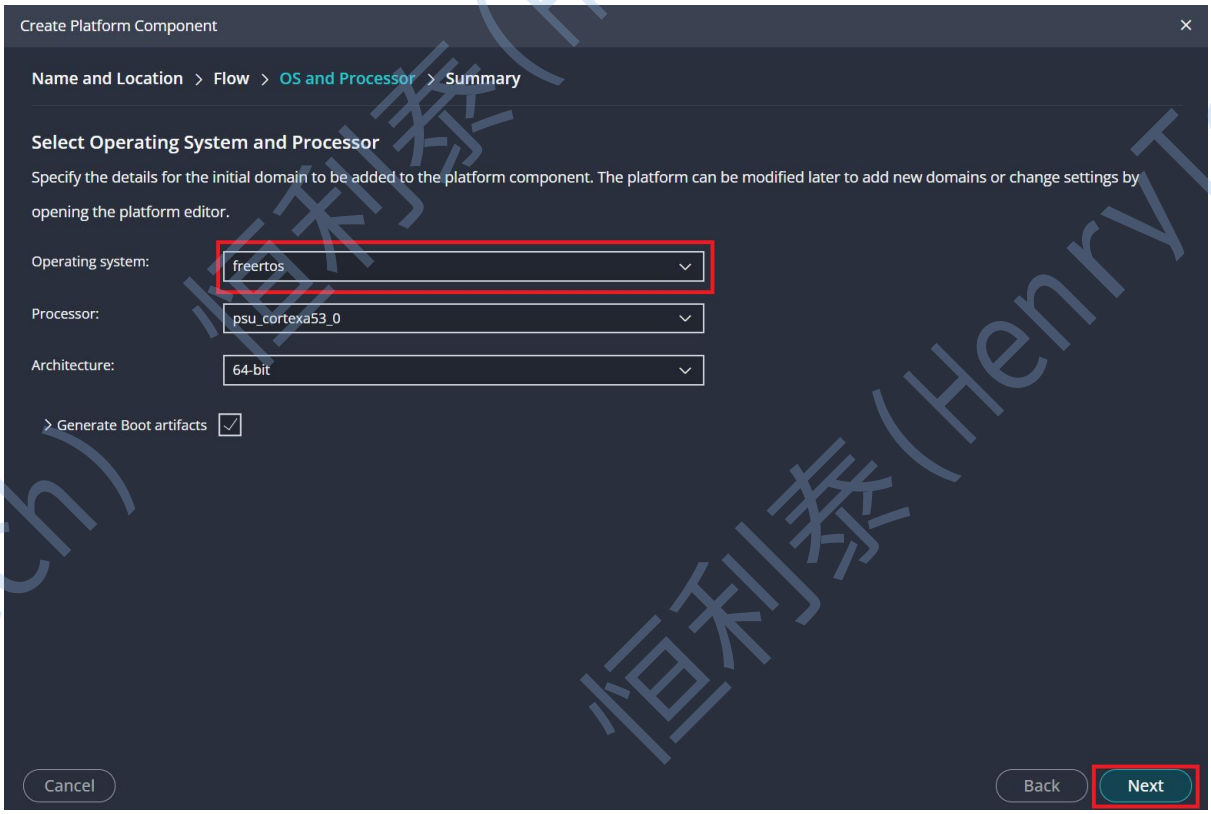
第一页，点击NEXT:



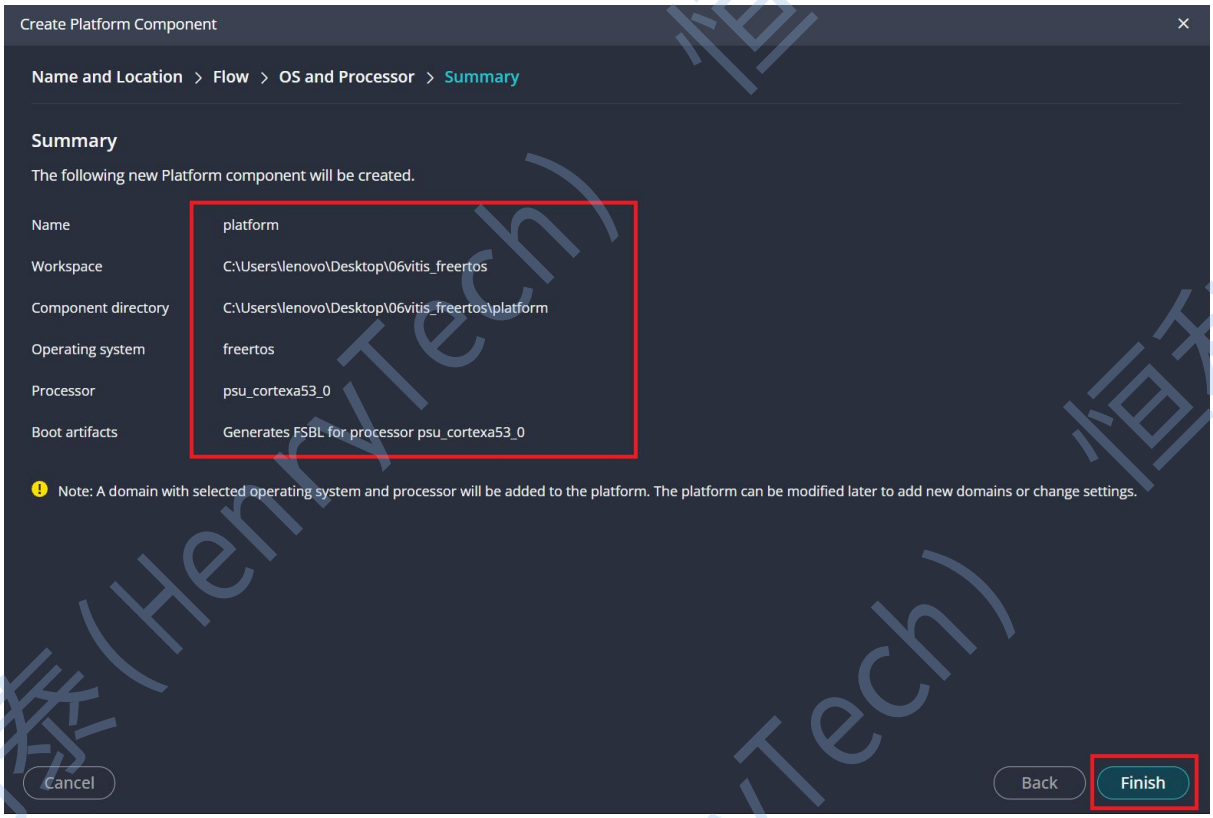
第二页，与helloworld一样，先点浏览（Browse）选择当前workspace的xsa文件，接着点击Next:



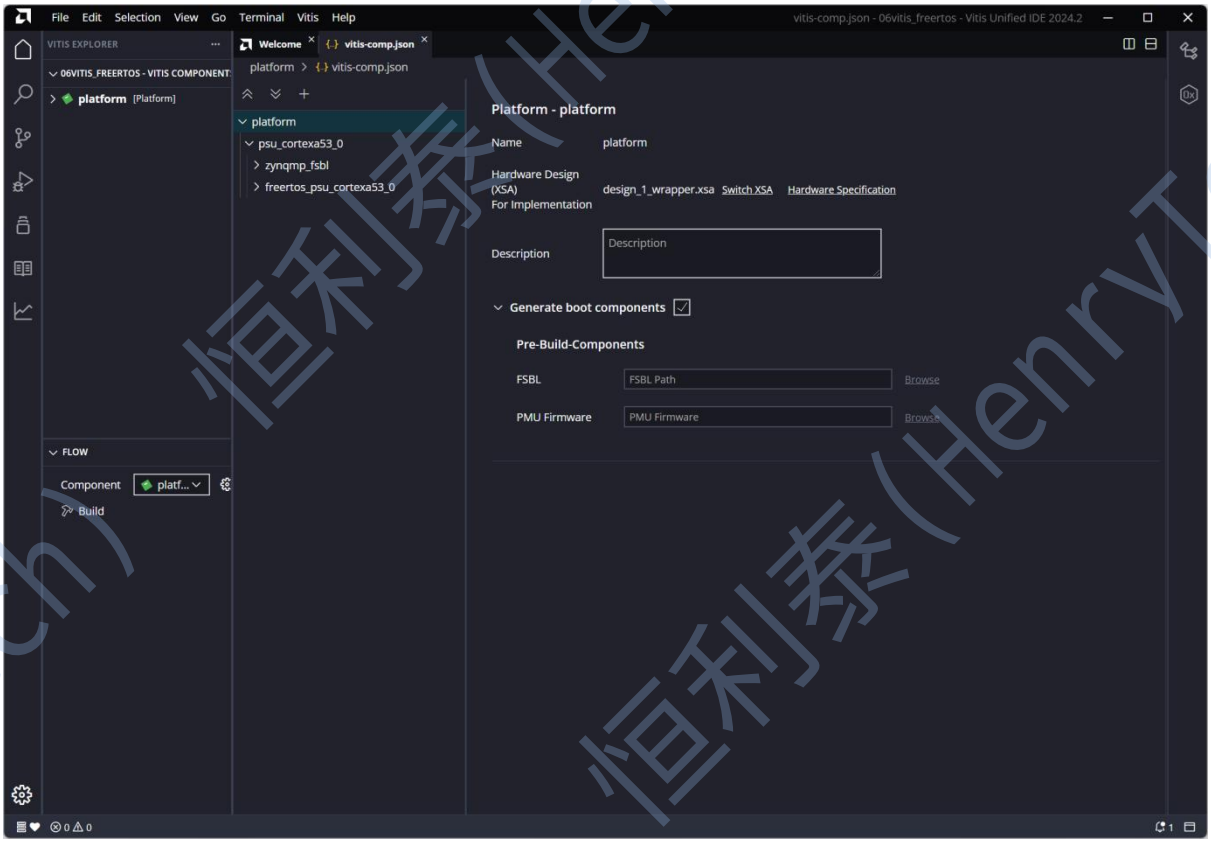
OS and Processor系统和处理器设置界面，我们配置如下，第一行Operation System设置为freertos，其余默认：



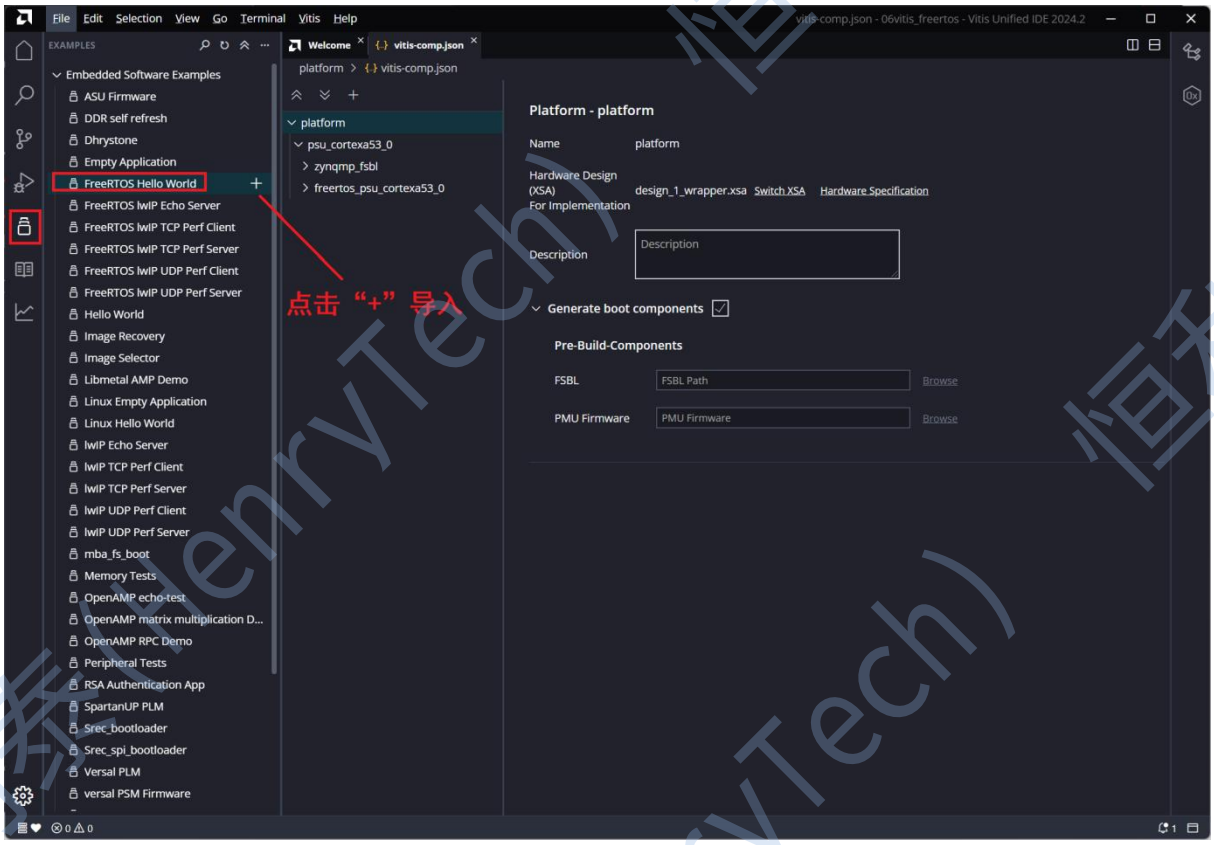
最后一页确认配置信息点击Finish完成：



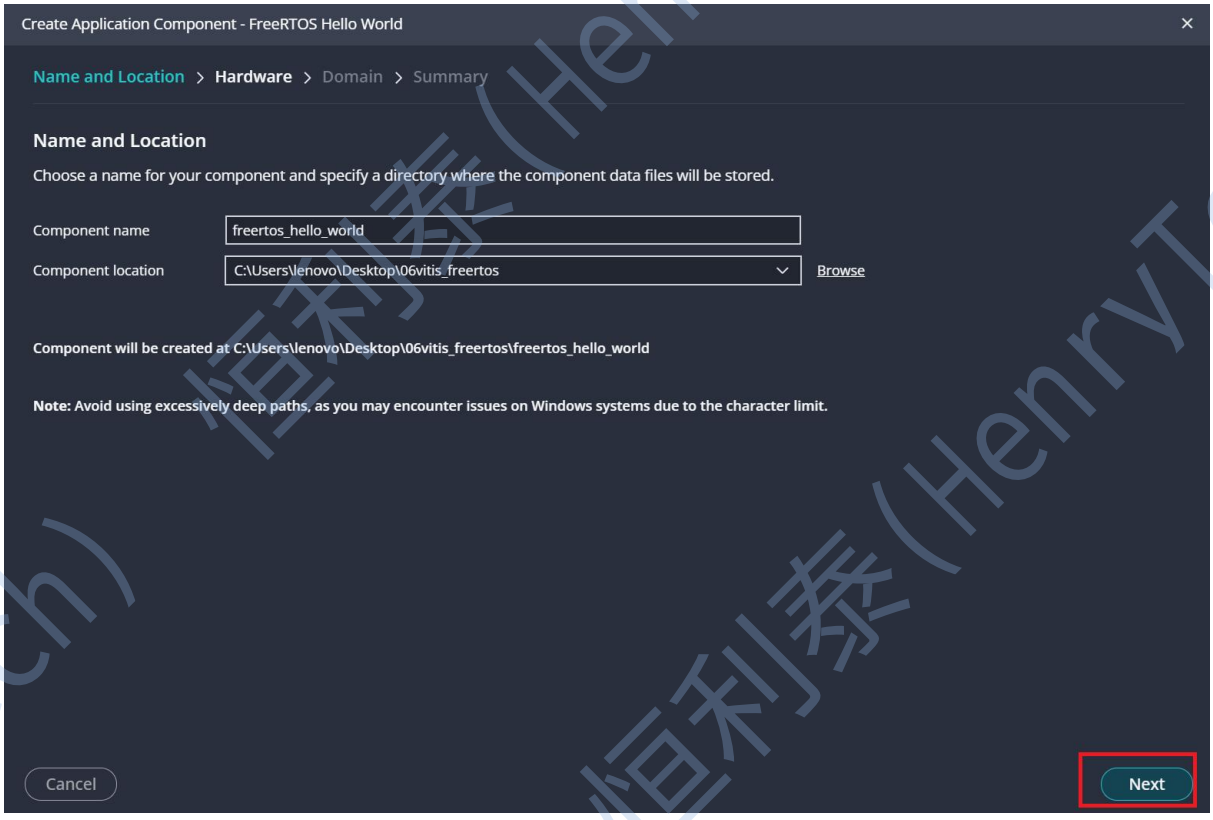
等待创建完成，工程如下：



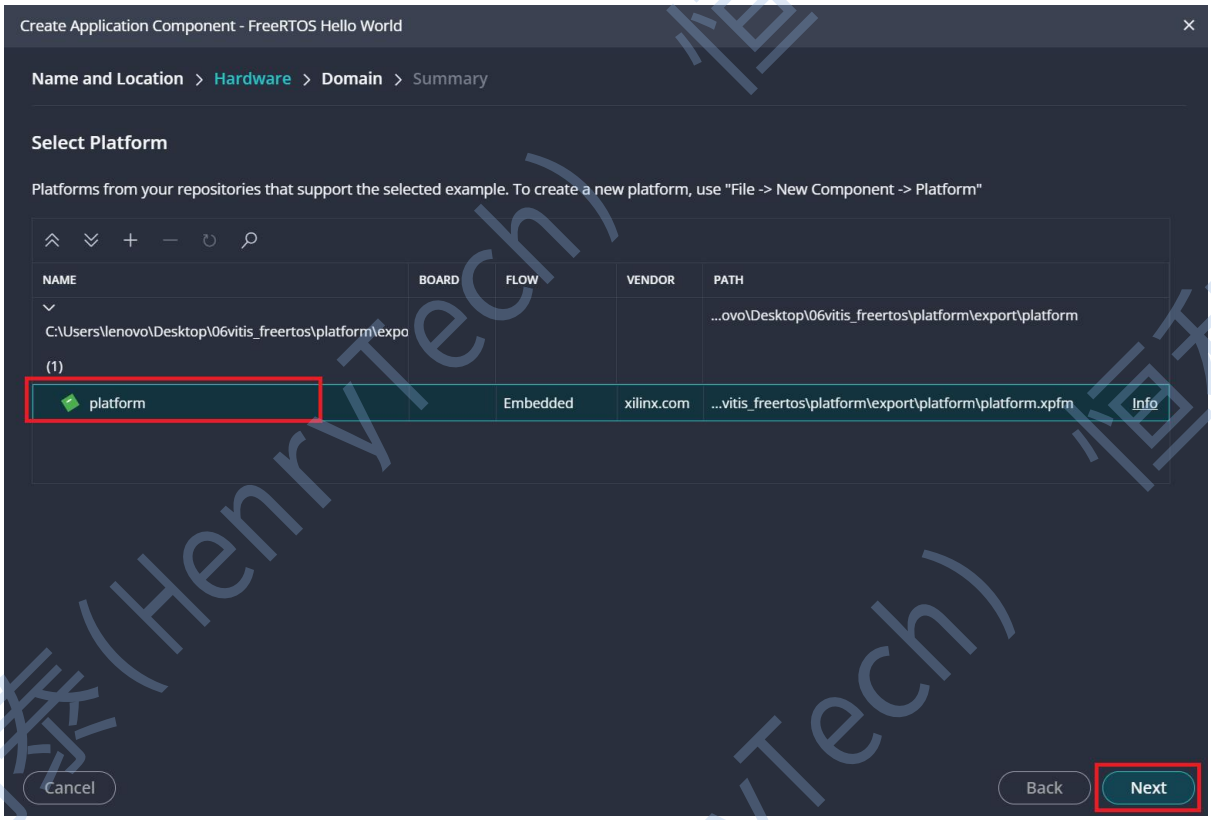
找到左侧图标倒数第三项点击进去，找到freertos模板，选中它，点击右侧“+”导入：



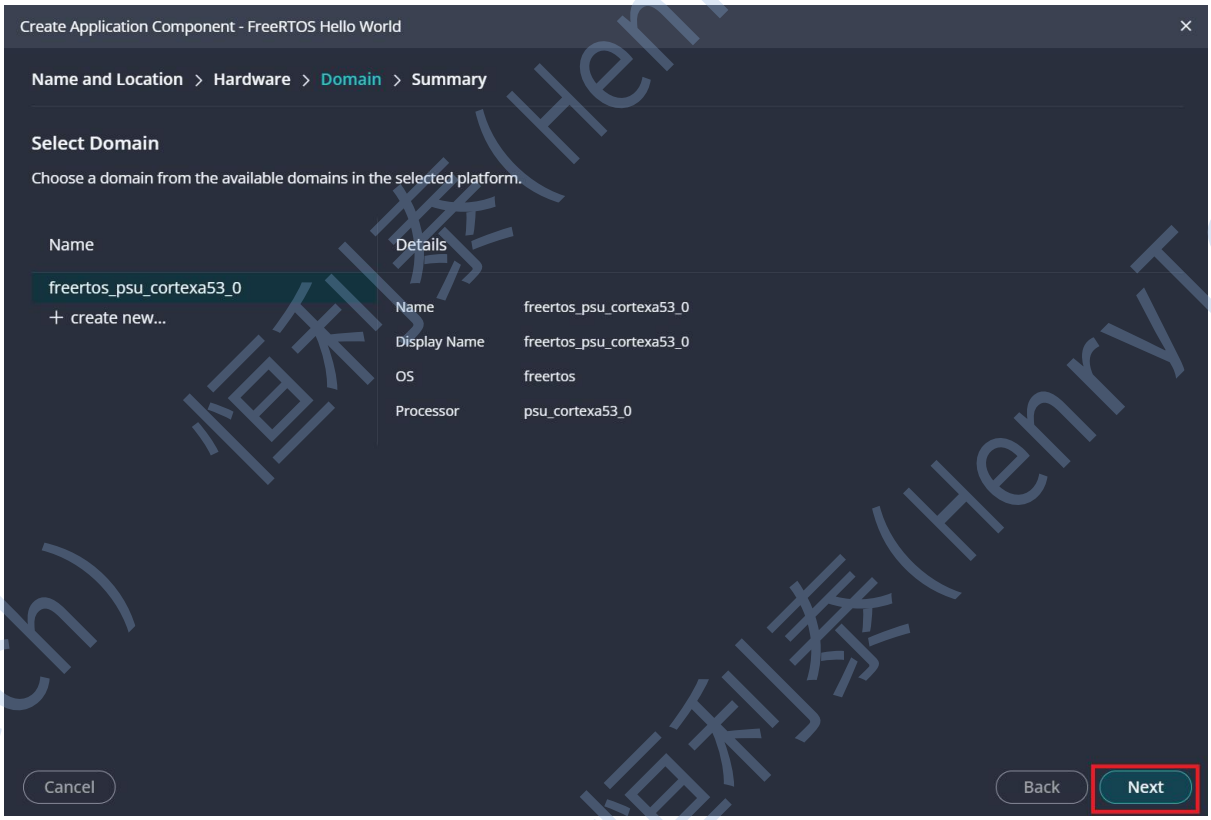
弹出新建application界面，第一页默认，点击next：



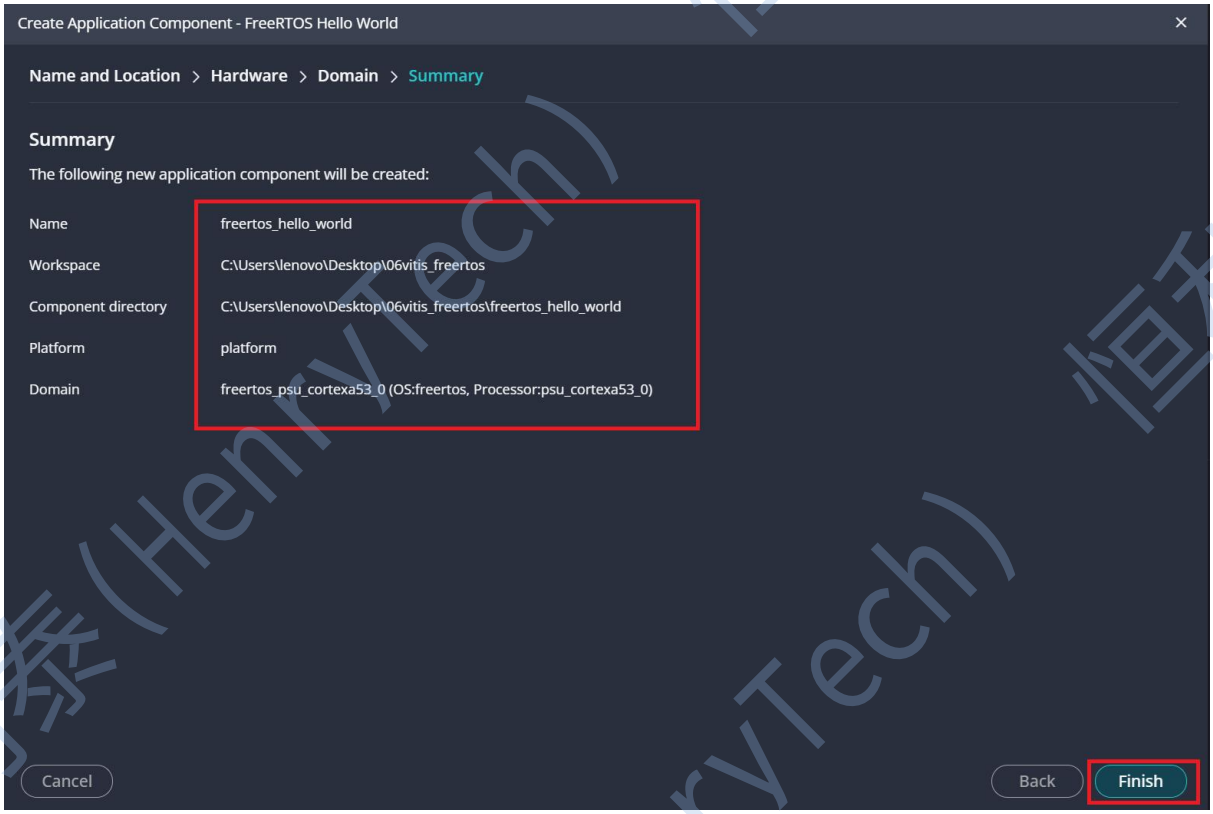
第二页，选择默认的platform点击Next：



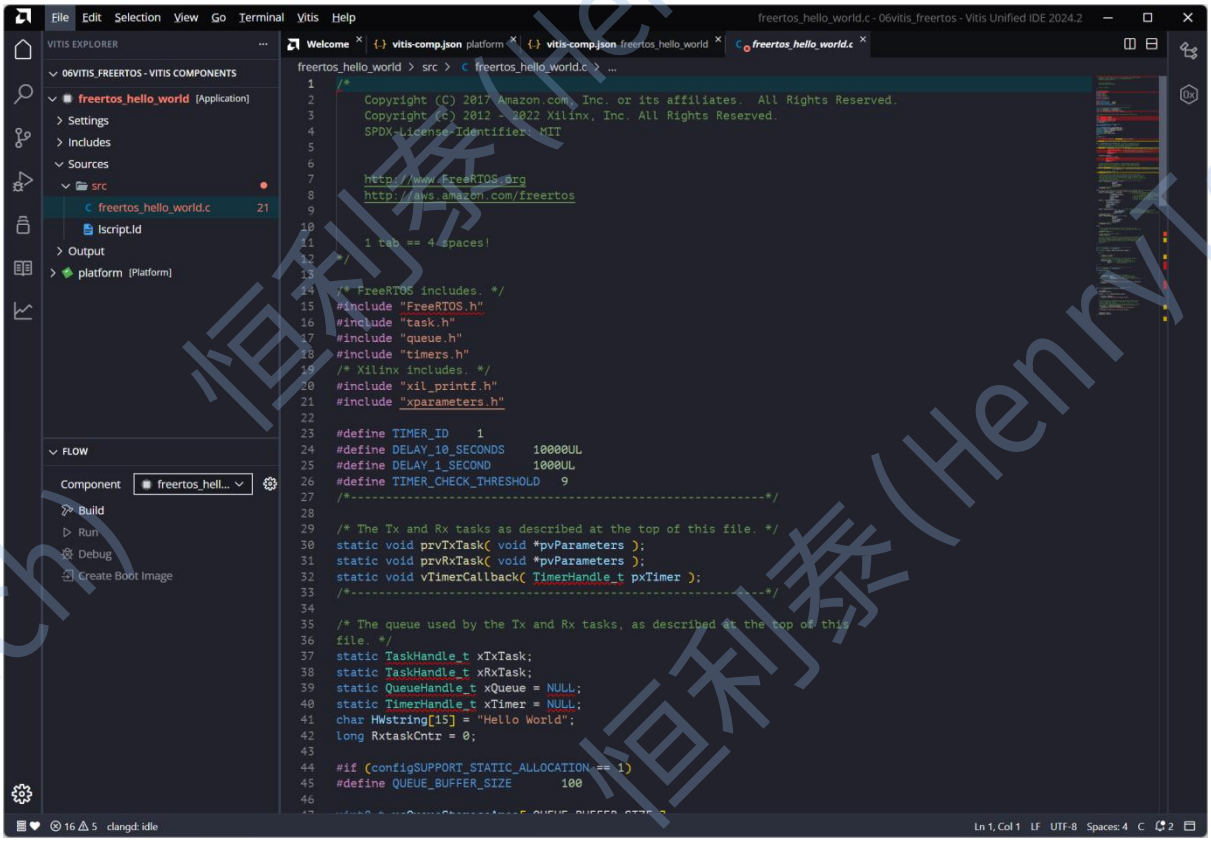
第三页domain选择，默认，点击Next：



最后一页，确认信息，点击Finish：



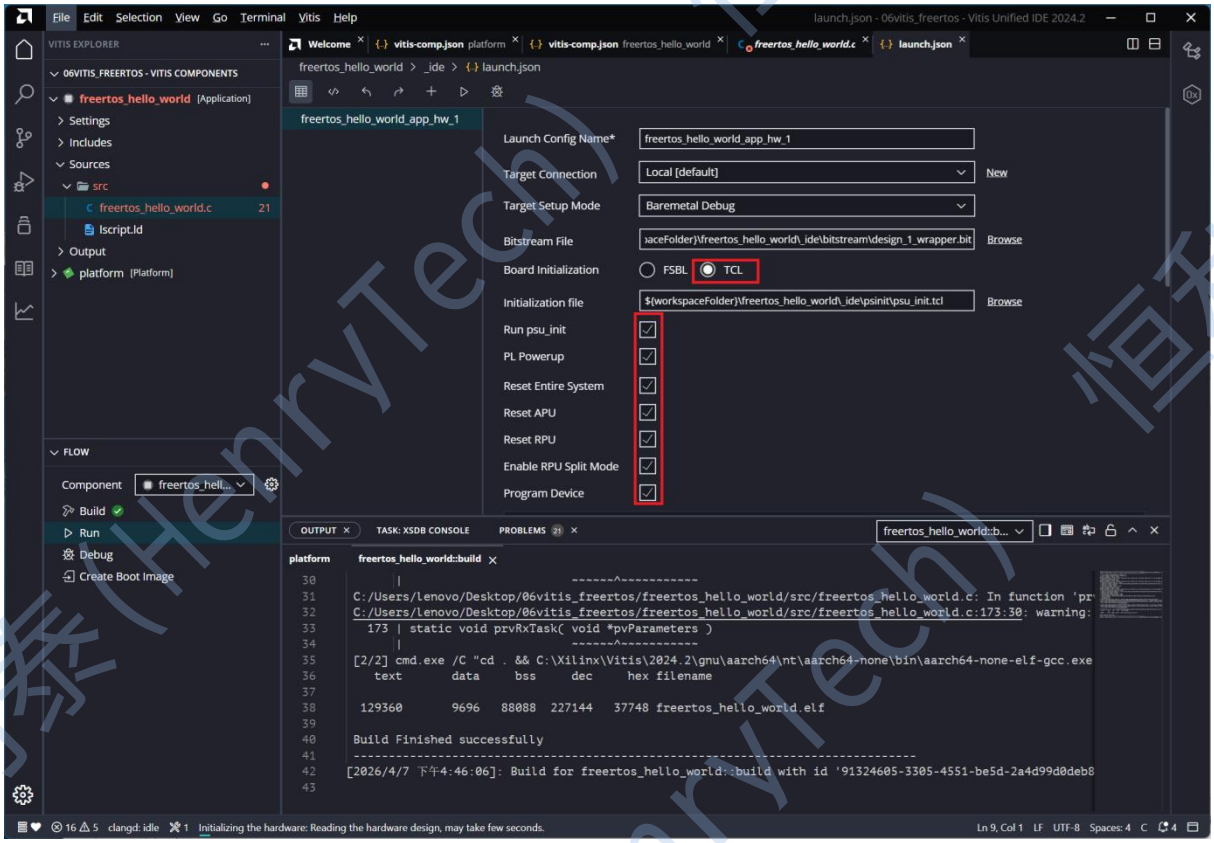
完成后等待工程创建完成，工程如下：



四、运行调试代码

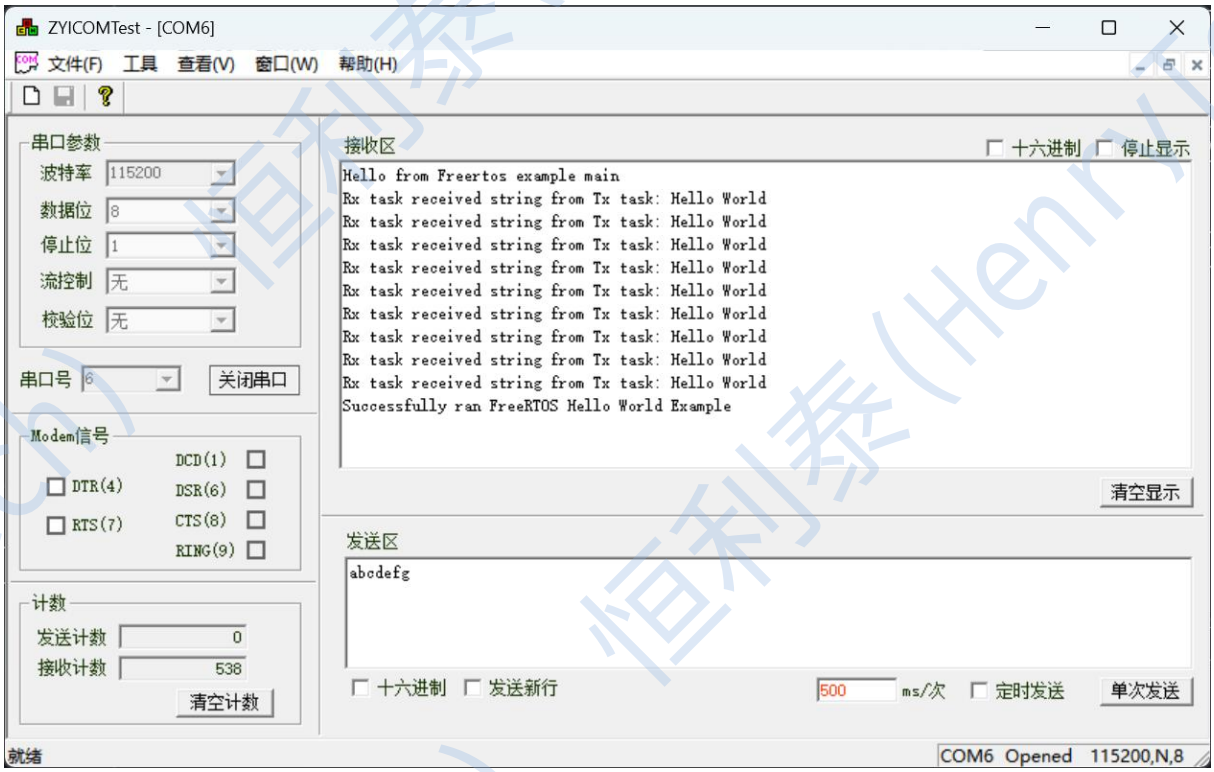
参考《11PS处理器配置及之helloword打印》，编译工程，设置好DEBUG，这里再强调

一下Debug和Run的配置：



这里特别说明，如果xsa导出不包含Bit文件，上图不需要，或者无法勾选Program Device，或者PL Power up。

我们单击Run，打开串口助手，打印如下：



LMK04828时钟配置

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 导入默认频率配置 3
三、 导出hex文件待用 8

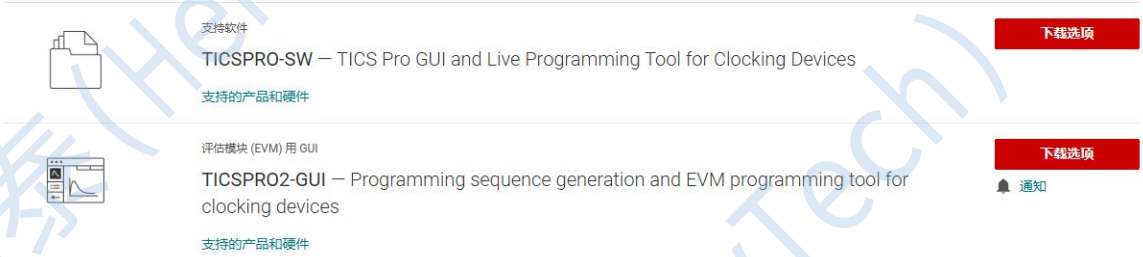
恒利泰(HenryTech)

一、文档实现功能介绍

本文档通过讲解使用TI的TICS Pro工具来配置使用LMK04828时钟芯片。由于ADC DAC需要工作的时钟输入和同步时钟等，而且时钟数量较多，所以，我们采用LMK04828这款型号。对应需要使用到的软件工具叫作TICSPro，我们网盘资料目录的《02软件和相关linux包/TICSPro_1.7.7.6_29-Oct-2024.exe》，大家可以自行下载安装，除了网盘提供，也可以从TI官方下载，地址：

https://www.ti.com.cn/tool/cn/TICSPRO-SW

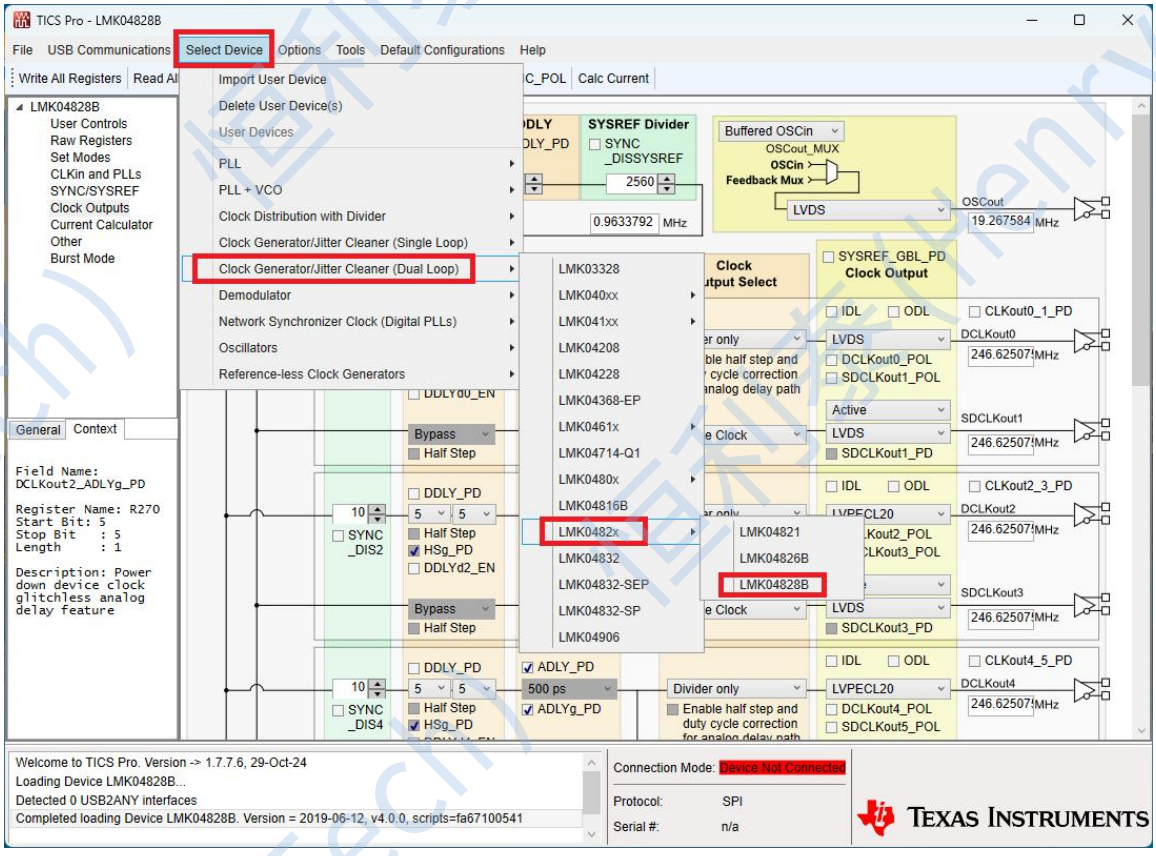
注意，TICSPro有两个软件，下载第一个即可，下载需要TI账号，无法注册或者登录，直接使用我们百度云提供的安装包：



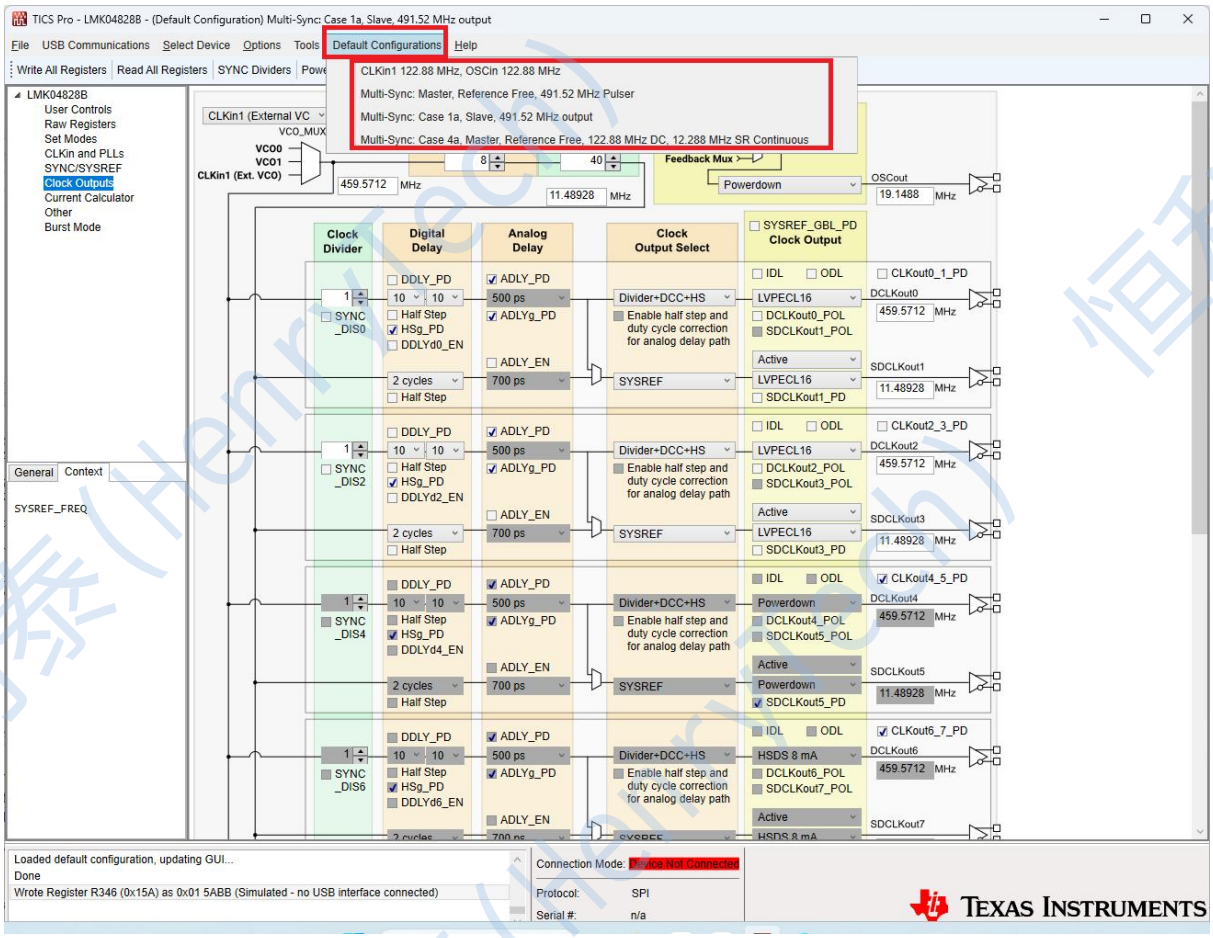
关于软件安装这里不细说。接下来我们讲一下具体配置过程。

二、导入默认频率配置

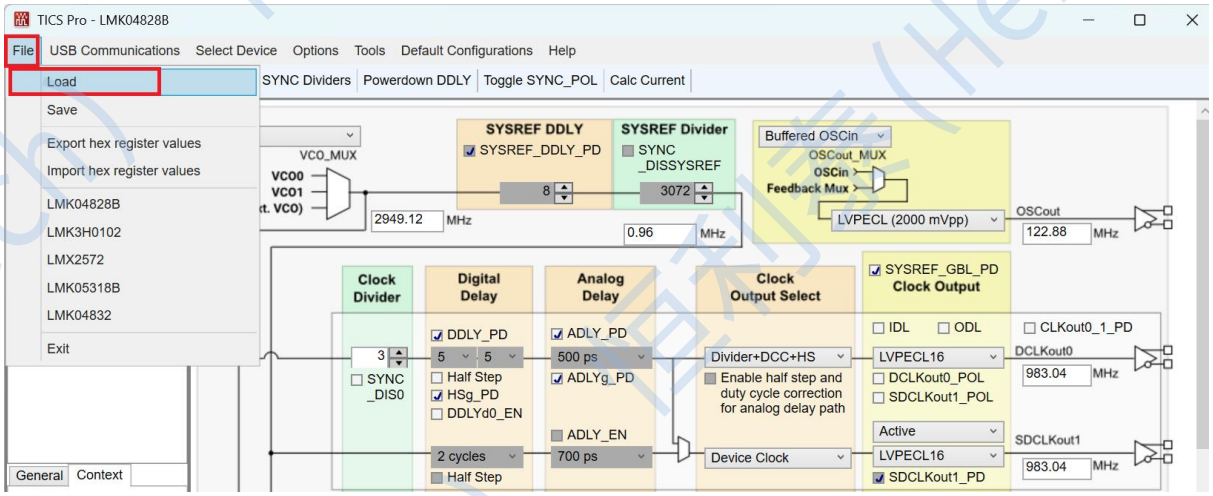
安装完成后我们双击打开软件，启动后进入主界面。第一次启动，型号可能默认不是我们的LMK04828，我们可以选择，菜单栏select device下拉选择如下：



选择型号LMK04828B之后，我们接着进行时钟具体配置。由于软件工具提供了一些常见配置预设模式，我们可以在Default Configuration下拉找到好几项：

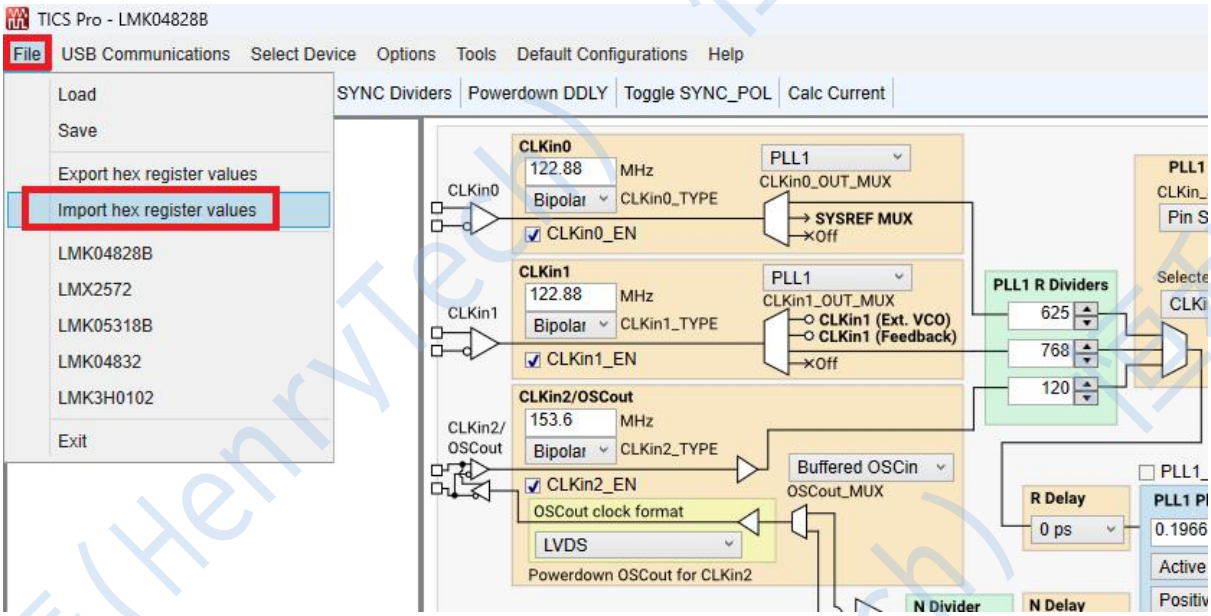


我们可以选择一项，比如第一项为预设，然后开始配置。由于配置项比较多，我们这里给大家提供了一个配置好的工程文件，可以直接加载导入。建议在此基础配置上如有需要可以进行修改再导出。我们下载网盘《RF67DR_Lite开发板资料/04例程demo/rf_adda_loop》目录中的<rf67_default_config.tcs>文件，TICS Pro中上方菜单栏选择file->load加载进来：

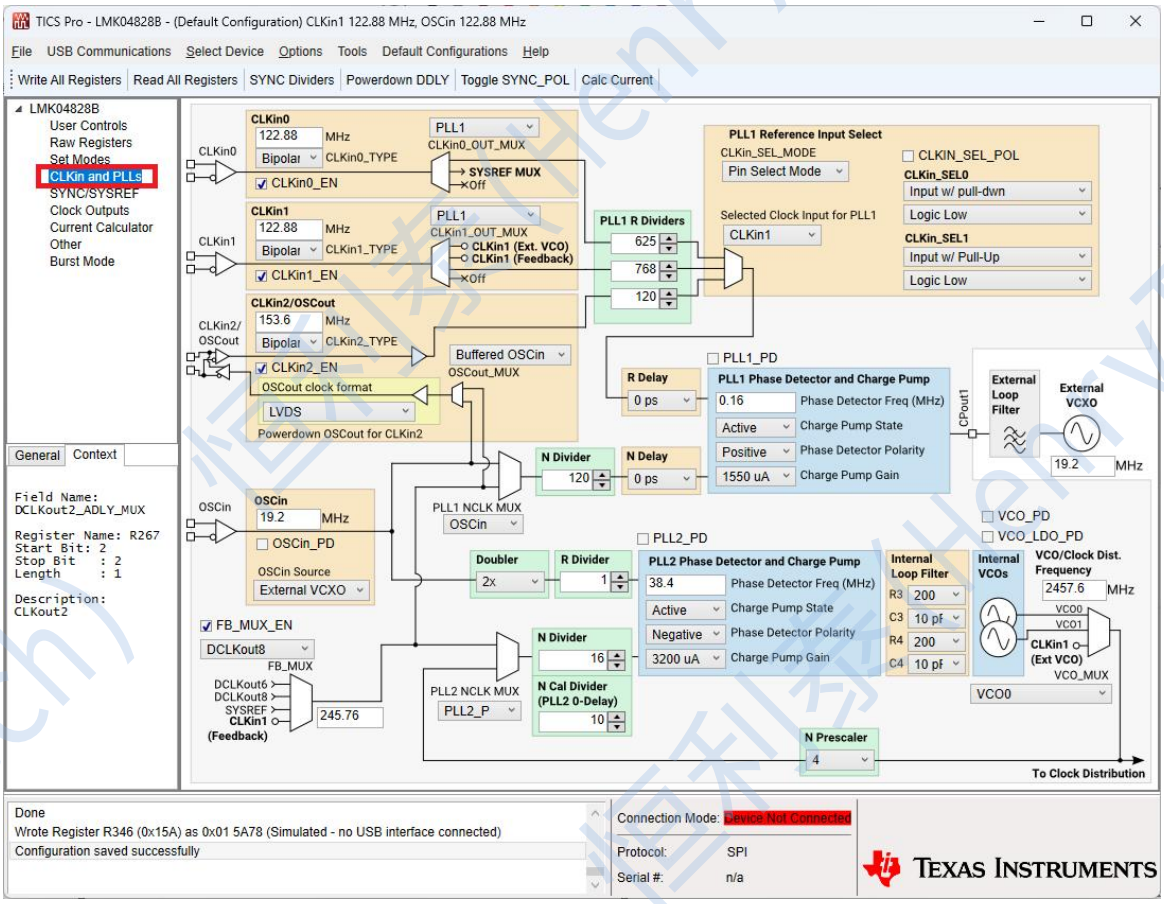


基于<rf67_default_config.tcs>配置修改后，如需恢复或者重新修改，可导入我们默认配置寄存器数据。txt文本文件位于《03例程demo/rf_adda_loop/HexRegisterValues.txt》，我们下载后存放到比如桌面，TICSPro的菜单栏选择File->import hex register values然后

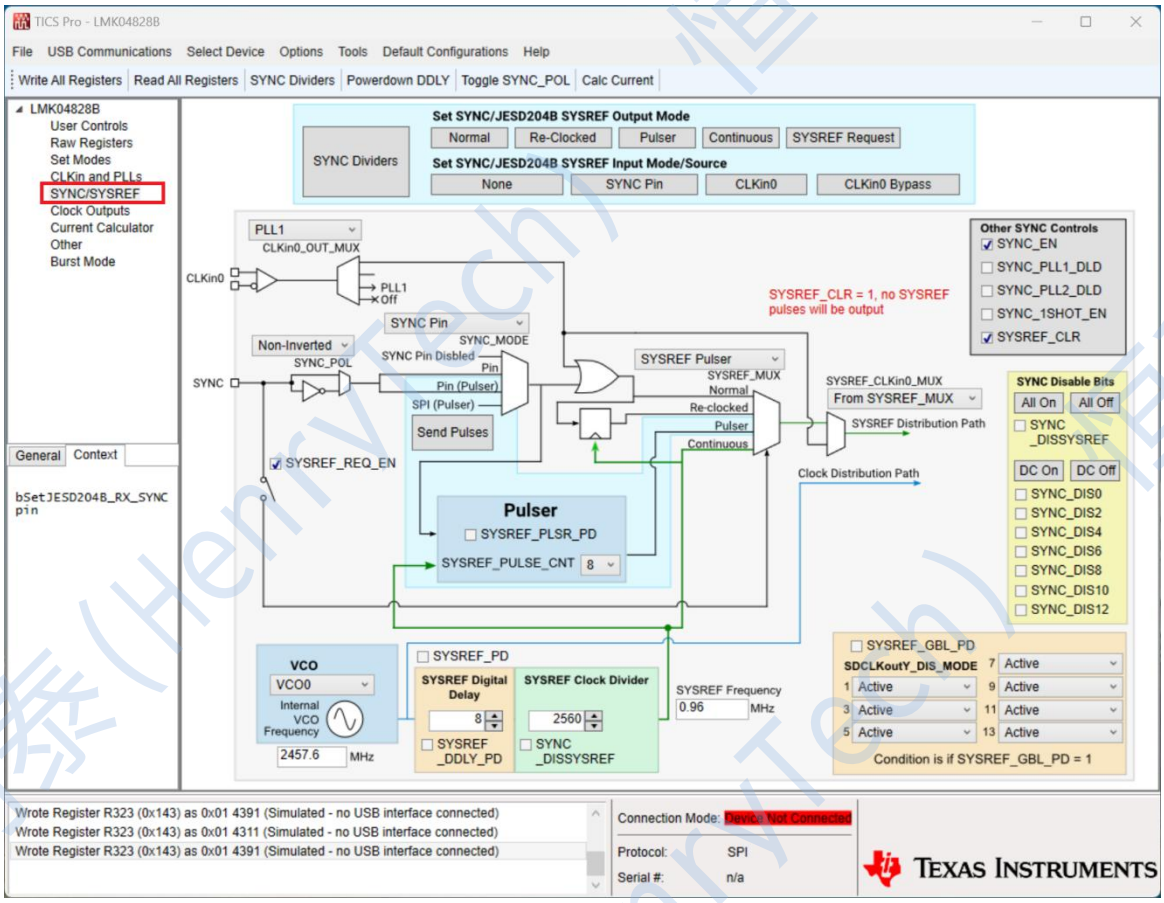
选择这个文件即可导入配置：



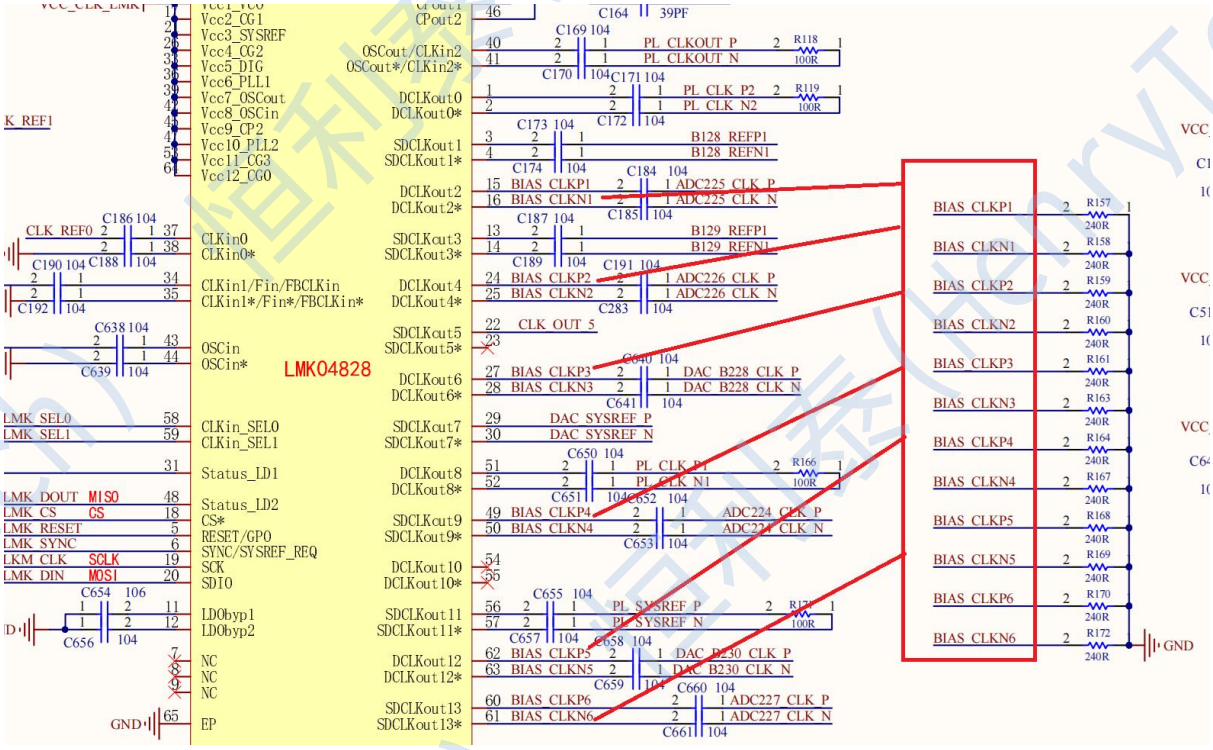
导入后，可以左边选择配置内容，其中主要的几个配置项如下，首先是CLKin and PLLs，主要涉及使用的PLL，输入输出的分频参数等：

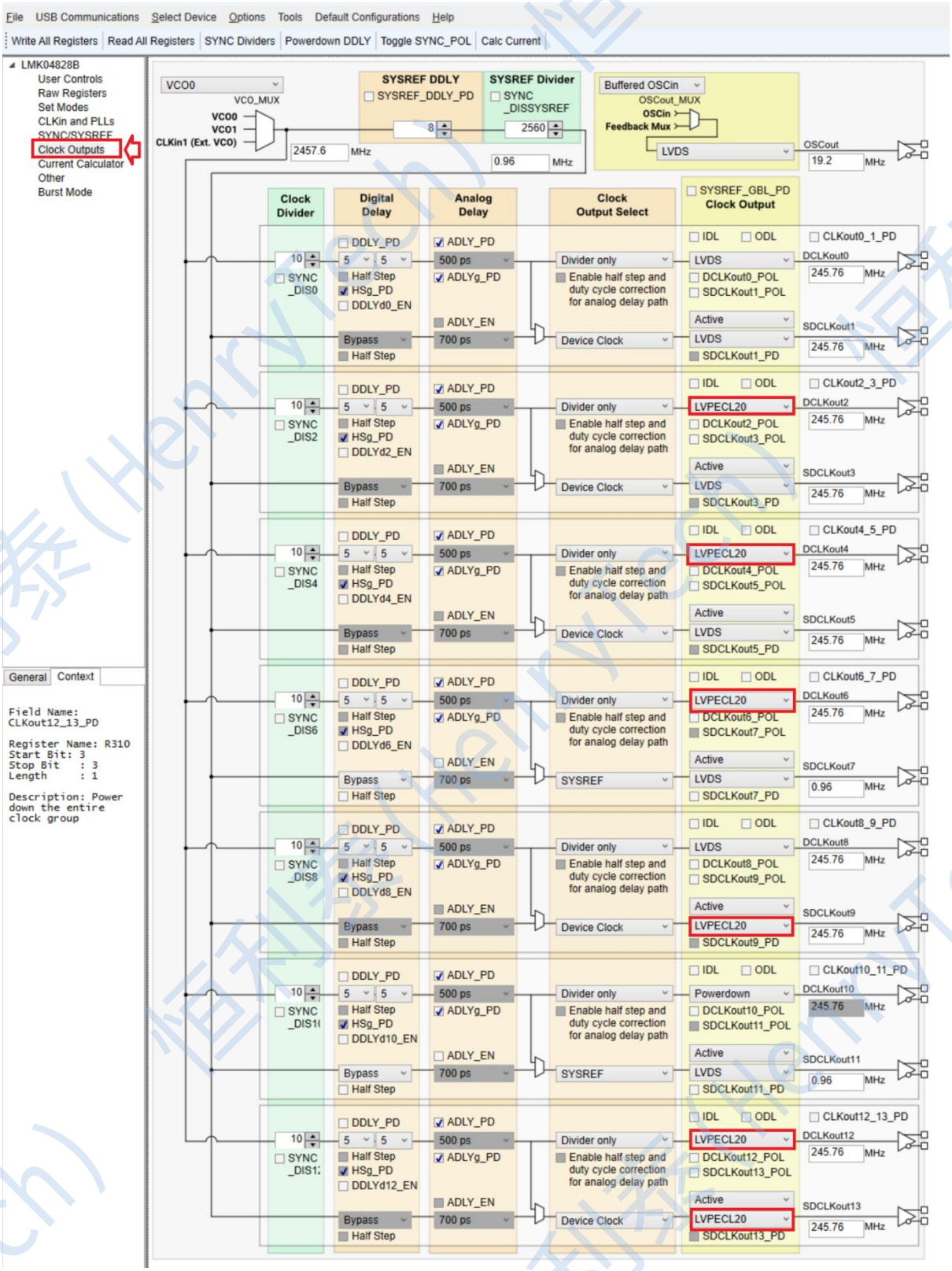


接着是同步设置页面：



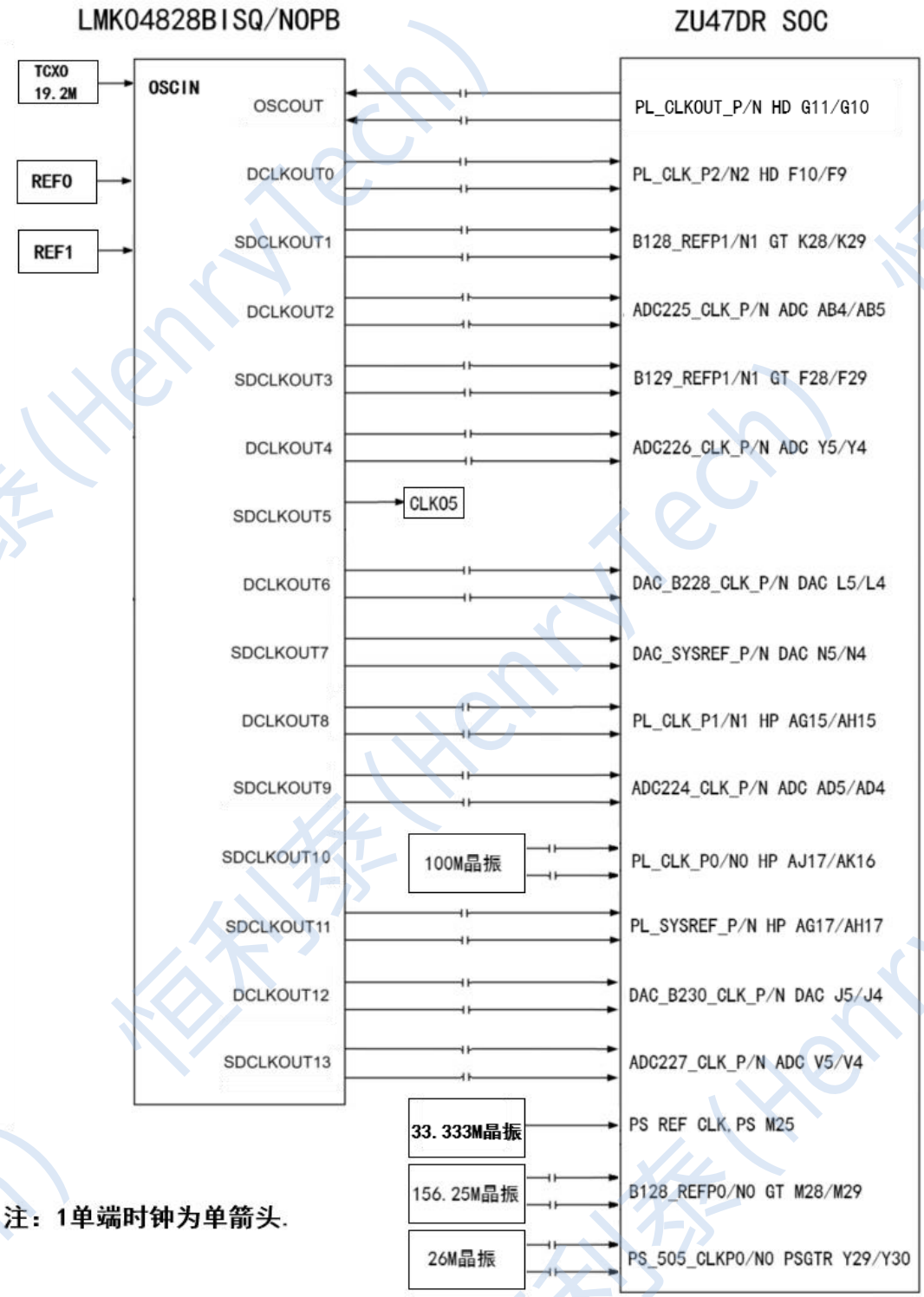
最后是每一路输出，注意，原理图中有一些差分输出连接了240欧姆下拉，这部分时钟选择电平是LVPECL20，其余是LVDS电平，可以从原理图第十九部分查看：





一般情况下上述时钟大家仅做一些小的改动，由于其中一些引脚可用于GTY的收发器或者PL的逻辑时钟，大家可以修改对应的这部分用户可用的时钟频率等参数。

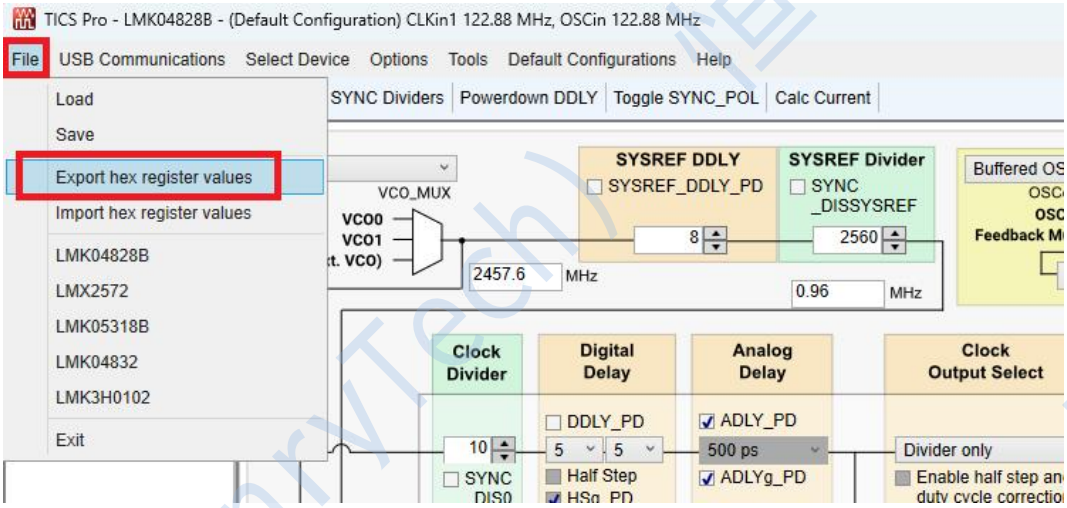
时钟我们整个开发板架构如下，我们硬件手册也有这个图：



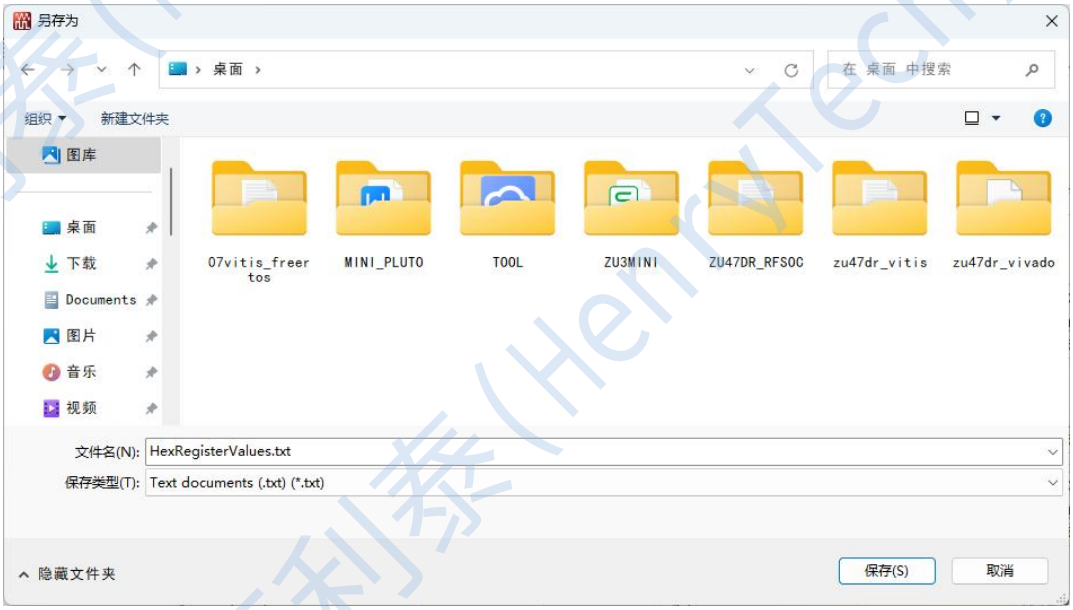
注：1单端时钟为单箭头.

三、导出hex文件待用

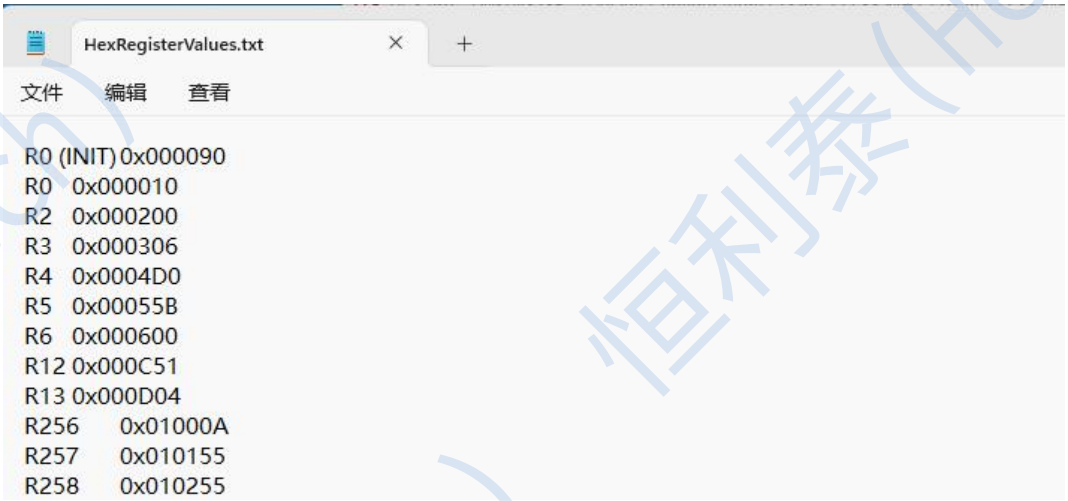
同导入文件，我们可以导出二进制。菜单栏file->export hex register values：



选择后我们弹出的对话框选择存放位置即可存放，比如桌面，点击保存：



最后打开保存到桌面的《HexRegisterValues.txt》文件格式如下：



至此，我们文件寄存器配置就完成输出。这个文件保存待用，后续我们在Vitis的adc dac例子中会介绍如何使用。

RF_Data_Converter射频回环实验

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

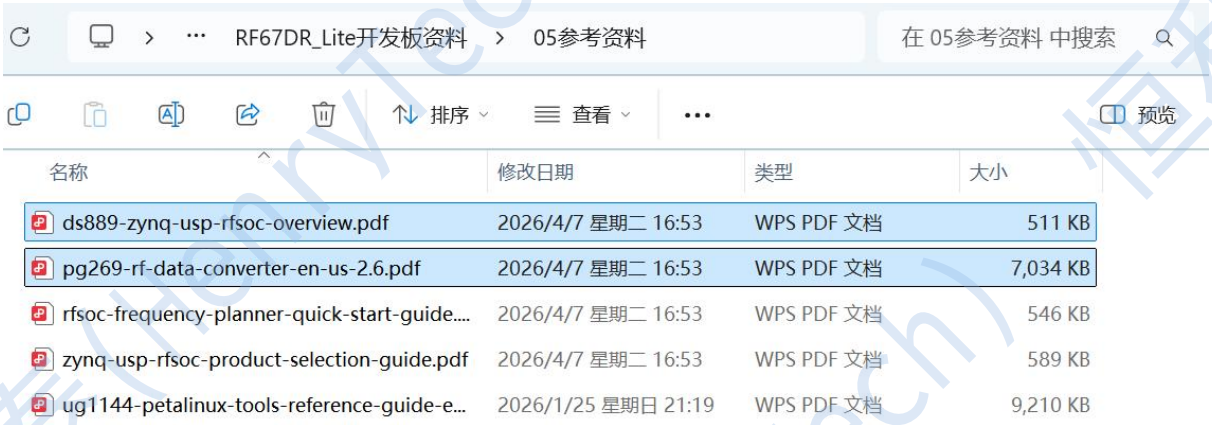
目录

一、 文档实现功能介绍 3
二、 RF Data Converter IP核基本配置 3
三、 调用ZYNQ配置 11
四、 BD设计介绍 13
五、 FPGA顶层驱动代码讲解 17
六、 编译生成bit文件导出硬件 28
七、 Vitis工程建立 32
八、 vitis代码说明 40
九、 联合VIVADO测试回环 44
十、 时钟更改配置 49
十一、 其他 50

恒利泰(HenryTech)

一、文档实现功能介绍

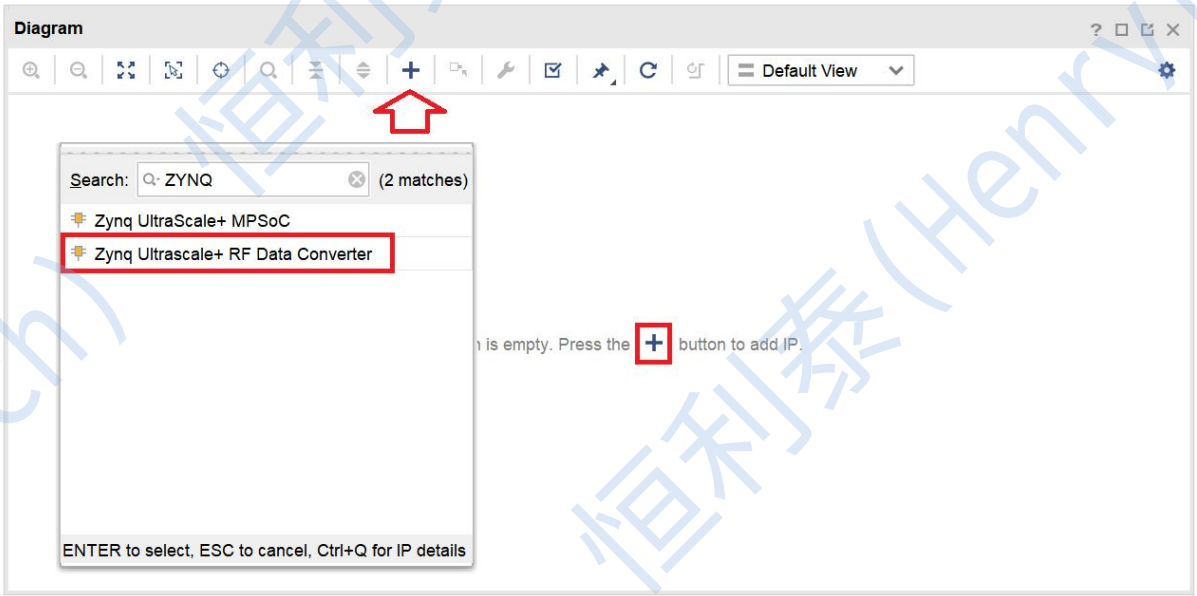
本文档实现射频RF Data Converter IP核做射频的ADC采集以及DAC输出的简单的射频回环实验。由于RFSOC的核心就是这个IP核，其余的ZYNQ系列不具备，仅RFSOC系列才有。本文档仅做裸机例程的使用参考，更加详细的相关开发，可以参考官方的PG269手册，位于《05参考手册》中：



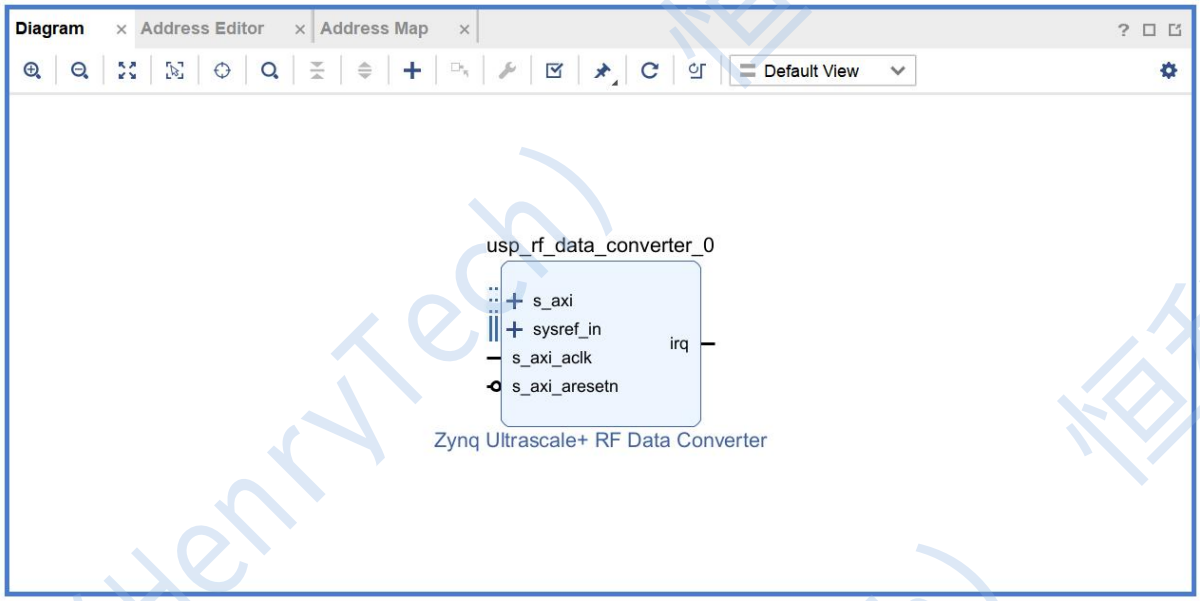
关于RF Data Converter这个IP，里面包含所有的详细介绍和开发信息。本文档仅作抛砖引玉，能够让大家先简单快速地跑起来RF ADC DAC。更详细的开发信息直接参阅PG269。

二、RF Data Converter IP核基本配置

RFSOC的 ADC以及DAC均使用此IP核进行配置调用。我们启动vivado2024.2新建一个任意工程，然后新建一个bd文档，添加IP符号中点击+号添加IP，输入搜索ZYNQ，找到如下IP核：



双击即可添加进来：



关于这个IP核，RF-ADC部分结构，根据PG269的53页可以看到内部框图如下，这里的简化图实际上对应的有两种。最常用的型号中，比如27DR，39DR的看上图，47DR属于GEN3,67DR属于DFE：

Figure 14: **RF-ADC Functionality Block Diagram (Simplified) (Gen 1/Gen 2)**

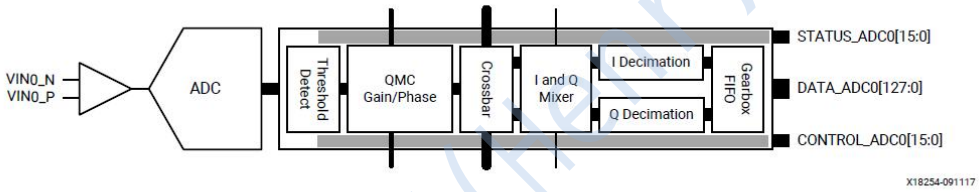
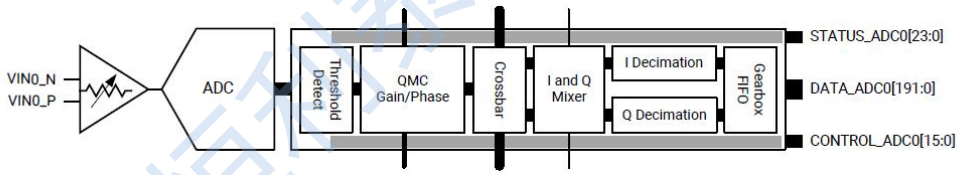


Figure 15: **RF-ADC Functionality Block Diagram (Simplified) (Gen 3/DFE)**



另外是RF DAC部分，PG269的109页如图67所示：

Figure 67: **Simplified RF-DAC Functionality Block Diagram (Gen 1/Gen 2)**

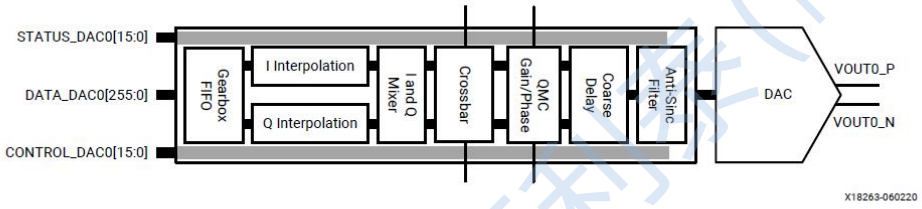
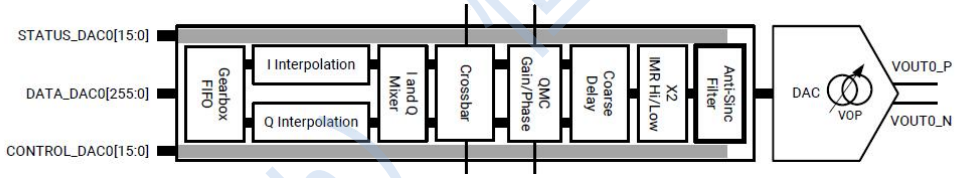
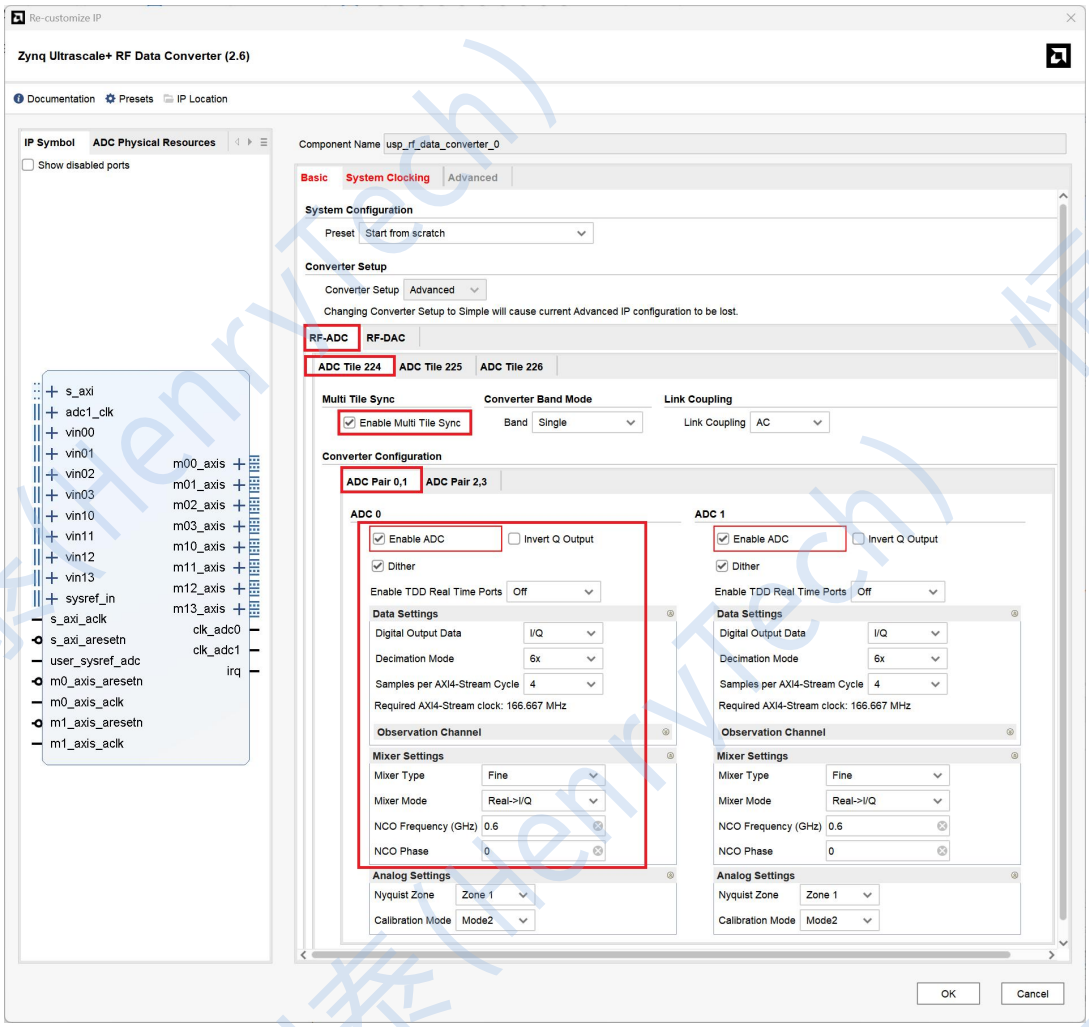


Figure 68: **Simplified RF-DAC Functionality Block Diagram (Gen 3/DFE)**

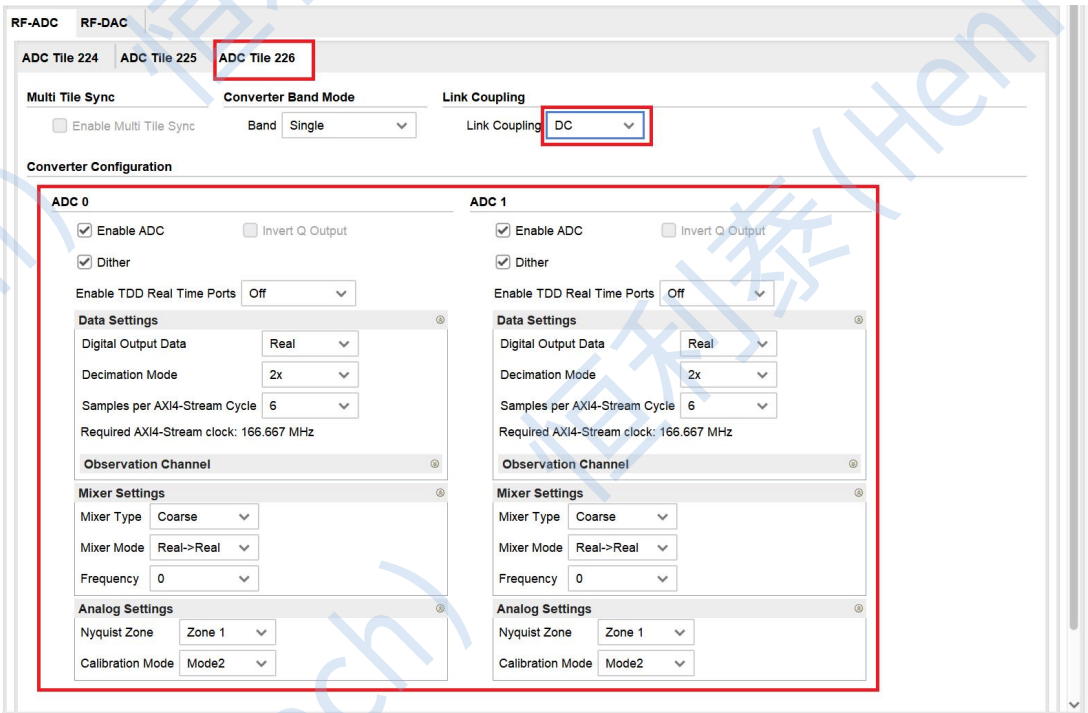


我们双击这个IP核，进入IP配置界面。关于配置，很多参数这里配置后，在嵌入式软件可以修改，这里可以认为是预设，但是针对某些特定的比如时钟分配等等，需要IP配置好。

打开配置界面，首先配置Basic基础部分。我们将RF-ADC的配置，每一个Title（ADC/DAC BANK）配置如下，ADC Title 224和225配置按照下图中所示配置，一个Title有4个ADC，需一致：



由于ADC Title 224 225采样率最大均为2.95GSPS，各4个通道，Title 226最大采样率5.9GSPS，有两个通道，所以我们针对Title 226配置如下：



3个ADC Title配置，224和225作为8通道，最大采样率2.95GSPS的RF ADC输入，主要设置了数据路径为I/Q格式，抽样点，axi接口数据宽度是几个点，以及混频方式，NCO本振频率，以及勾选了Enable ADC使能已经勾选了Enable Multi Title SYNC多通道同步等。

Title 226比较特殊，在硬件上，我们将前端设计成运算放大器，将连接器单端信号进行输入，放大，转为差分信号。放大器输出是采用DC耦合到ADC输入，针对高速信号采集应用（非射频）。但是运放输入信号到运放输入端的时候，硬件默认是交流单端耦合（针对输入单端信号接口是交流耦合）。

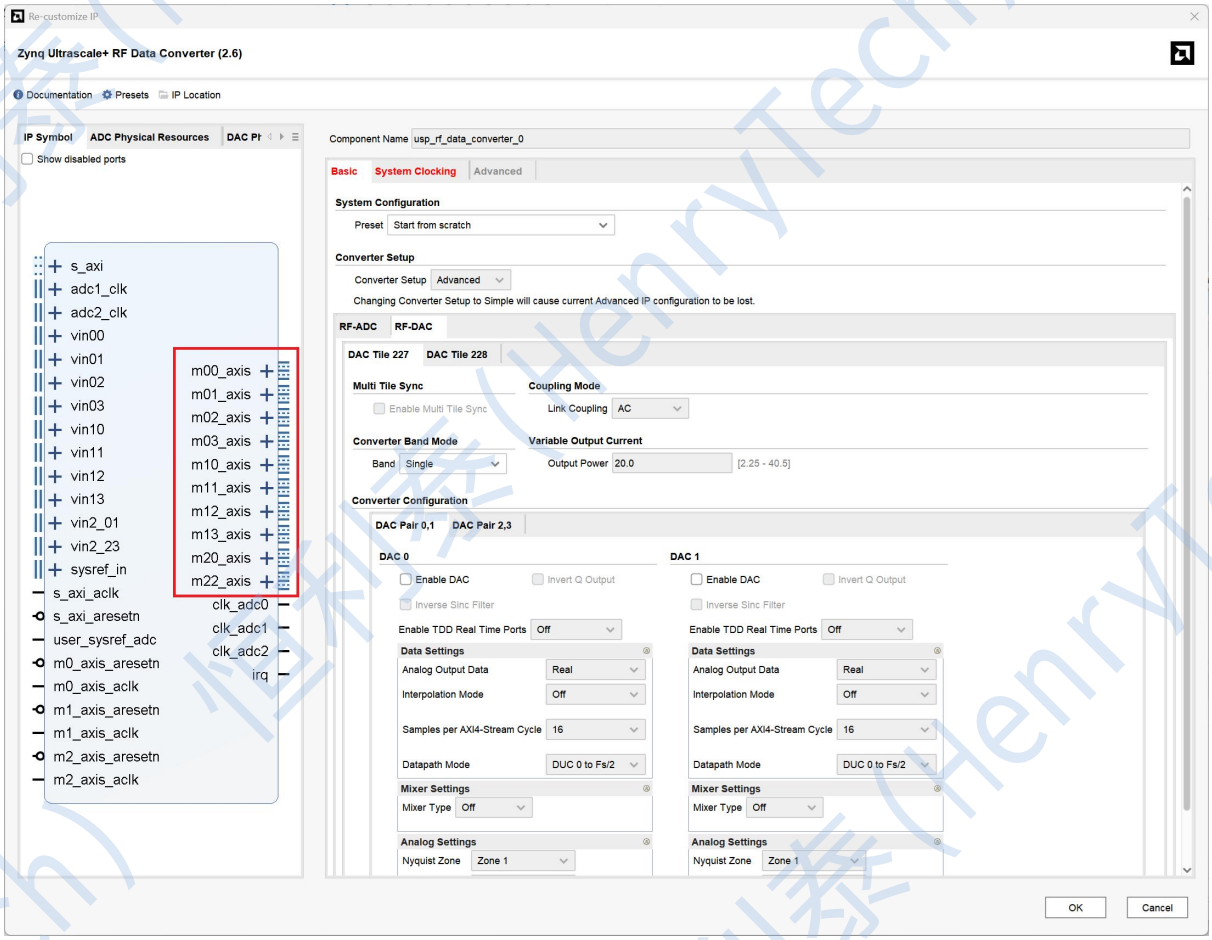
这里要搞清楚一个概念，ADC设置中的Link couplig指的是差分ADC输入信号与前端的耦合方式，前端指的巴伦或者运算放大器的输出端。Title 224 225的link coupling由于前端是采用巴伦+电容输入，默认AC。

配置参数释义：

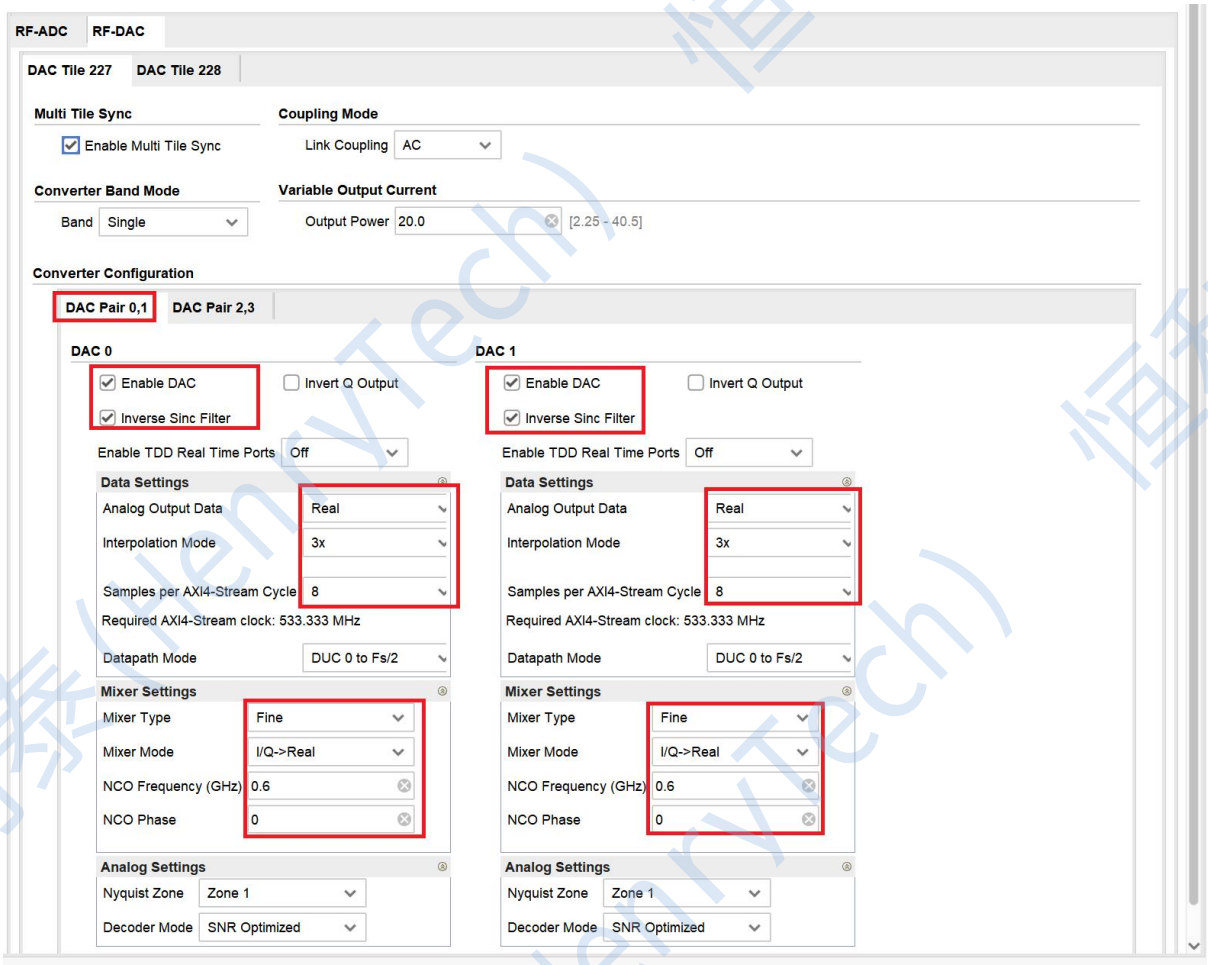
- Multi title Sync：多个titile同步；勾选表示几个ADC BANK可以同步，不勾选不同步。
- Converter band mode：转换器频带模式，默认为single选项，单独使用，multi表示多个一起使用。
- Link coupling：耦合模式，AC表示交流耦合，DC表示直流耦合。如果选择DC模式，需要结合VCM引脚，配合外部差分放大器等作为共模电压。本开发板硬件不支持DC模式。前端采用巴伦直接连接电容AC耦合到差分输入引脚。
- Enable ADC：使能。表示使用这个ADC。
- Invert Q Output：当使能时，转换器的Q输出被取反。仅在勾选 I/Q输出数据时可设置。后续的模拟混频器实现减法，Ix cos(f) - Qx sin（f）数据输出。
- Dither：抖动设置。勾选表示使用抖动。
- Enable TDD Real Time Ports 启用 TDD 实时端口， 启用后，tdd_mode 端口将添加到 IP。 可以通过关闭 RF-DAC 实现节能。
- Digital Output Data：设置所选片内所选转换器的数据类型，该参数仅在启用转换器时可配置。有效值为Real和I/Q，当转换器0被设置为I/Q时，转换器1也必须被设置为I/Q。
- Decimation Mode： 设置所选片内所选转换器的抽取值，可以选择 1x、2x、3x、4x、5x、6x、8x、10x、12x、16x、20x、24x、40x。 由于按照ADC采样率直接采样，时钟频率太高，我们可以从采样数据硬件抽取，比如5X就是表示每5个ADC采样点通过AXI输出一个点，其余不要，软件处理不过来。可以设置为1x就是全部采样，但是时钟到FPGA内部频率太高，时序可能不一定能够处理。不推荐。
- Samples per AXI4-Stream Word： 设置每个周期的字数，当对应转换器已启用时，此参数是可配置的，简单理解就是一个AXI时钟周期有多少个采样点数据。结合上一个抽取参数，这个可选配置项也是不固定的。
- Mixer Type：设置混频器类型，有效的选项有coarse, 表示粗混频，fine精混频。当然fine输出混频更好一些。

- Mixer Mode：设置转换器的混合器模式，可以选择是一个通道只有一路信号还是有正交调制的IQ信号，在空间向量垂直90度发射。
- NCO Frequency(GHz)：NCO本振频率。
- NCO Phase：本振相位。
- Nyquist Zone：偶数和奇数奈奎斯特区域选项。
- Calibration Mode：校准模式。

我们配置好四个ADC Title（BANK）后显示IP如下，IP端口出现了m开头的axi master接口，这部分接口是主动往外发的ADC采样结果输出，m00-m22一共10个通道。每个AXI接口对应一路数据，我们67DR是10路ADC，其中M00-M03对应Title 224,M10-13对应Title225，M20-21对应226。



接着配置DAC部分，RF-DAC部分界面上，我们开始配置，配置如下，我们67DR的DAC是有2个Title(BANK),227 228 ，这部分配置需要有一定先后顺序，我们先勾选每一个DAC使能：



参数释义：

- Multi title Sync：多个titile同步；勾选表示几个DAC BANK可以同步，不勾选不同步。
- Converter band mode：转换器频带模式，默认为single选项，单独使用，multi表示多个一起使用。
- Link coupling：耦合模式，AC表示交流耦合，DC表示直流耦合。
- Enable DAC：使能。表示使用这个DAC。
- Invert Q Output：当使能时，转换器的Q输出被取反。仅在勾选 I/Q输出数据 时可设置。后续的模拟混频器实现减法，Ix cos(f) - Qx sin(f)。
- Variable Output Current：可调输出电流，输出电流按照设置大小输出。
- Inverse Sinc Filter：是否启用 Sinc逆变滤波器。
- Enable TDD Real Time Ports 启用 TDD 实时端口， 启用后，tdd_mode 端口将添加到 IP。 这可以通过关闭 RF-DAC 的部分来实现节能。
- Digital Output Data：设置所选片内所选转换器的数据类型。可设置 Real 和 I/Q。当转换器 0 被设置为 I/Q时，转换器 1 也必须被启用。
- Interpolation Mode：设置所选片内所选转换器的插值。可选择 2x、3x、4x、5x、6x、8x、10x、12x、16x、20x、24x、40x。与ADC相反，这里是DAC输入比如DDS数据速率跟不上，硬件插入的采样点，让输出DAC更加平滑。
- Samples per AXI4-Stream Word： 设置每个周期的字数，当特定转换器已启用时，此参

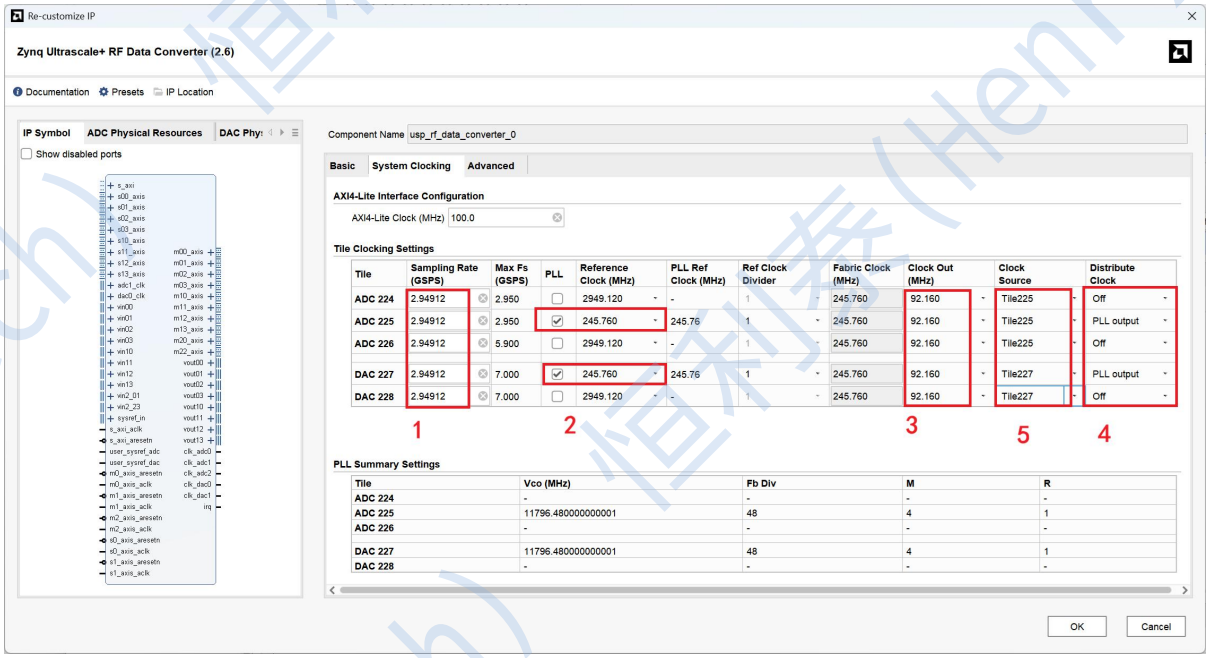
数是可配置的。简单理解就是一个AXI时钟周期有多少个采样点数据。另外，与插值Interpolation Mode联动，不同的插值，会导致不同的选项可选择差异。比如当前设置5x插值，我们可以选4-16。

- Datapath Mode：DAC数据路径模式，这些模式适用于数字数据的路径，可以使用模拟设置奈奎斯特区域选择来自第一或第二区域输出该数据，以下是可选项：

  1）DUC 0 to Fs/2：全奈奎斯特DUC模式；
  2）DUC 0 to Fs/4：低通IMR滤波器的半奈奎斯特DUC模式；
  3）DUC Fs/4 to Fs/2：高通IMR滤波器的半奈奎斯特DUC模式；
  4）No DUC 0 to Fs/2：DUC旁路模式。

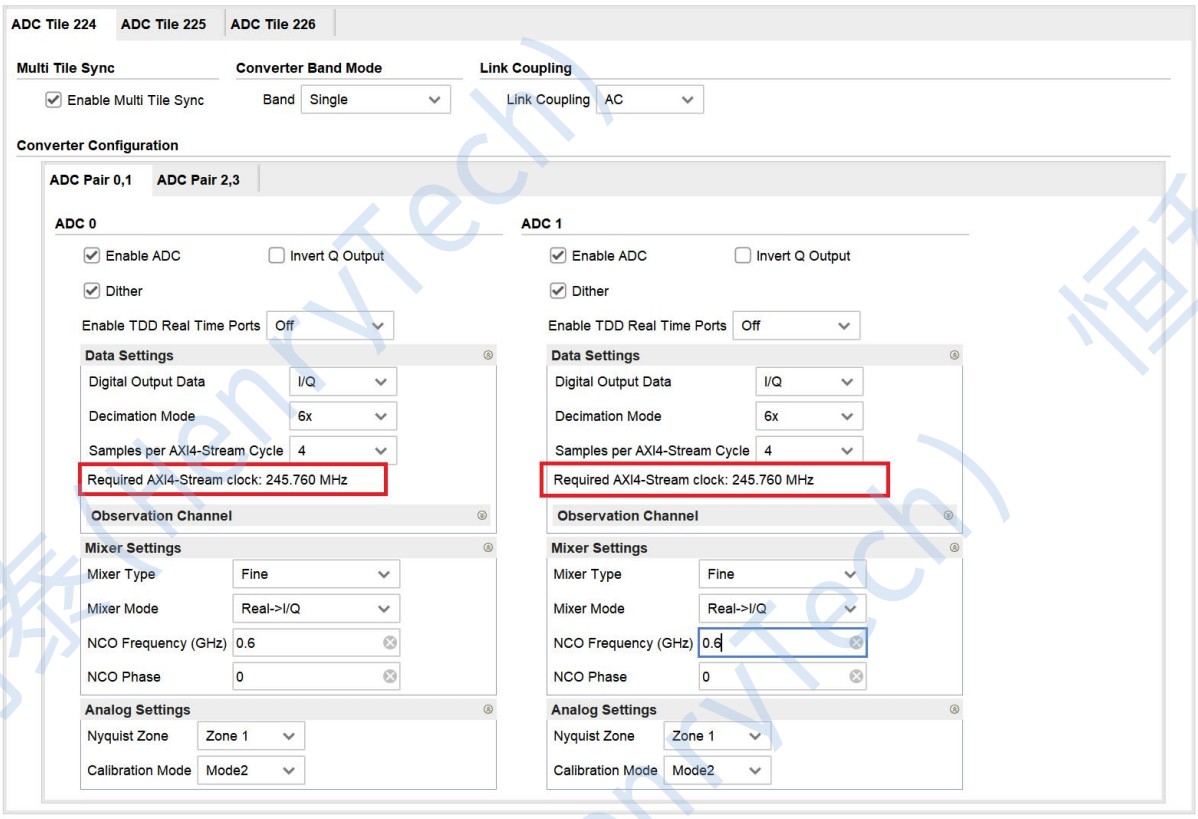
- Mixer Type：设置混频器类型，有效的选项有coarse, fine表示粗混频，精混频。
- Mixer Mode：设置转换器的混合器模式，对于精混频，只有I/Q-Real可选。粗混频还有Real-Real可选。
- NCO Frequency(GHz)：NCO本振频率。
- NCO Phase：本振相位。
- Nyquist Zone：偶数和奇数奈奎斯特区选项。
- Decode Mode：解码可以选择优化为低信噪比或高线性度选项。

配置好DAC后，我们配置时钟。时钟是关键的部分，目前ADC和DAC配置还有参数不对，比如AXI时钟频率。我们在上方System Clocking页面，对时钟进行如下配置，注意，按照数字顺序依次配置。从配置界面可以看到，每个Title最高采样率：

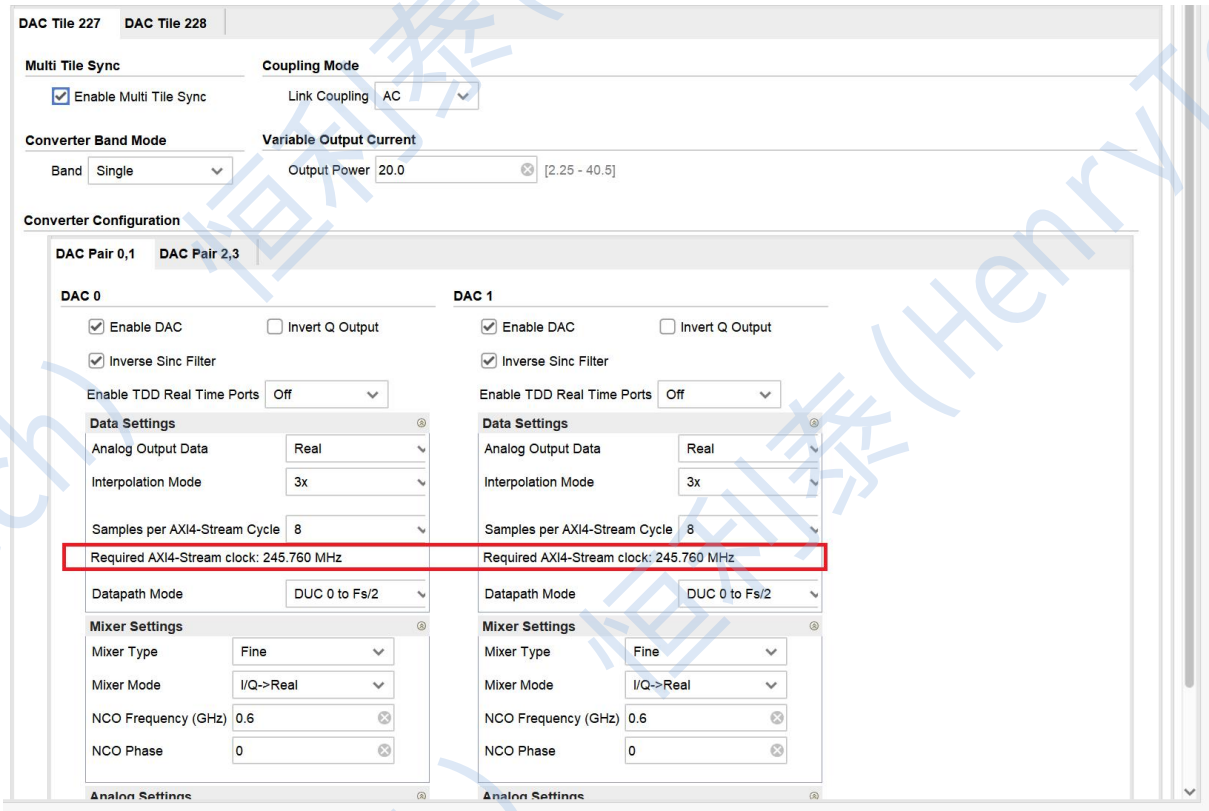


上述配置中，clock source是指Title的工作时钟源，Distribute clock我们选择了一个Title

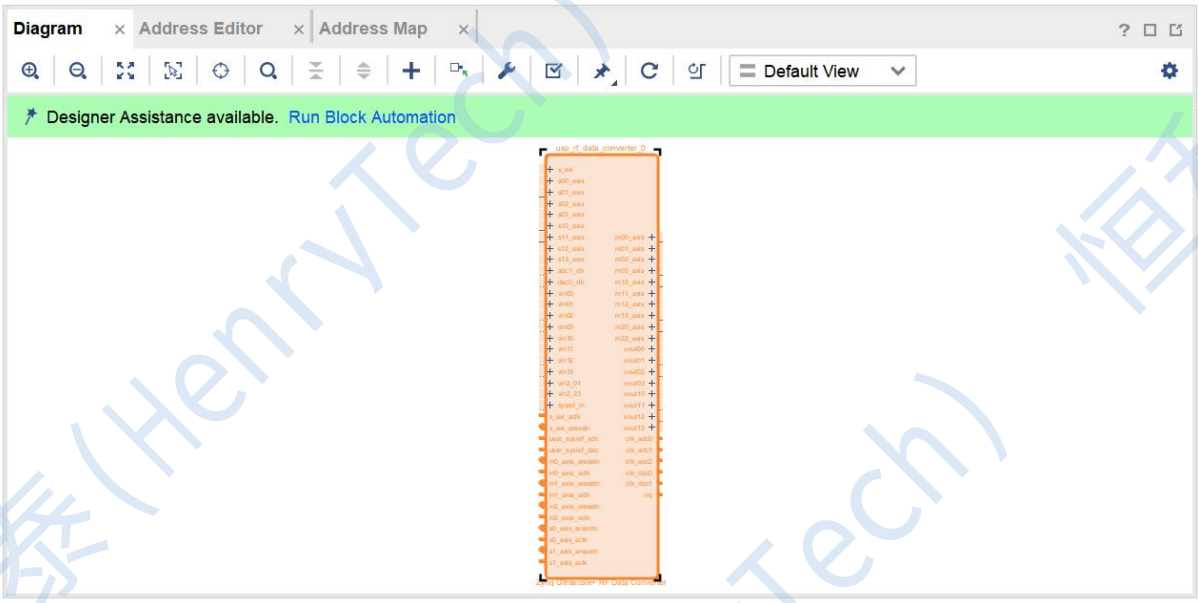
作为PLL output，所以clock source可以选择对应这个TITLE。针对多通道同步，这样配置是非常方便的。配置好时钟后，回到ADC和DAC配置界面，可以看到AXI时钟显示245.76Mhz了：



DAC配置也可以看到AXI时钟：

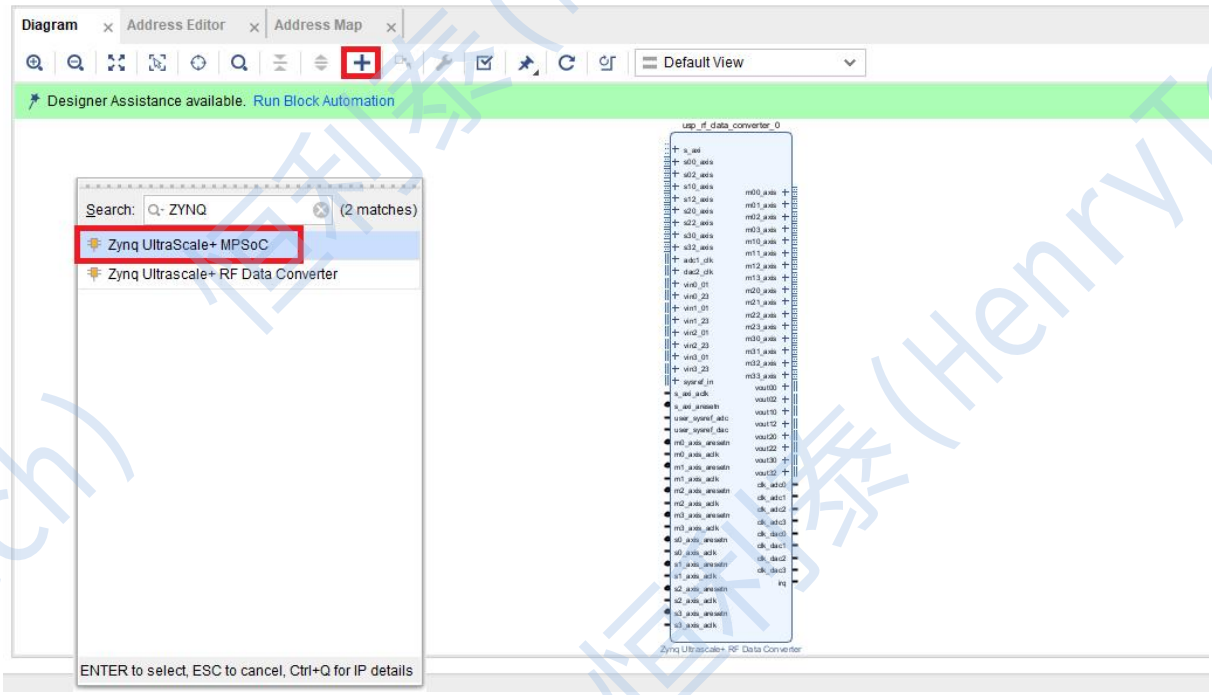


关于AXI接口时钟，我们在逻辑设计中还需要提供一个245.76M的时钟用于ADC采集和DAC输出。至此，RF data_converter核就完成配置。

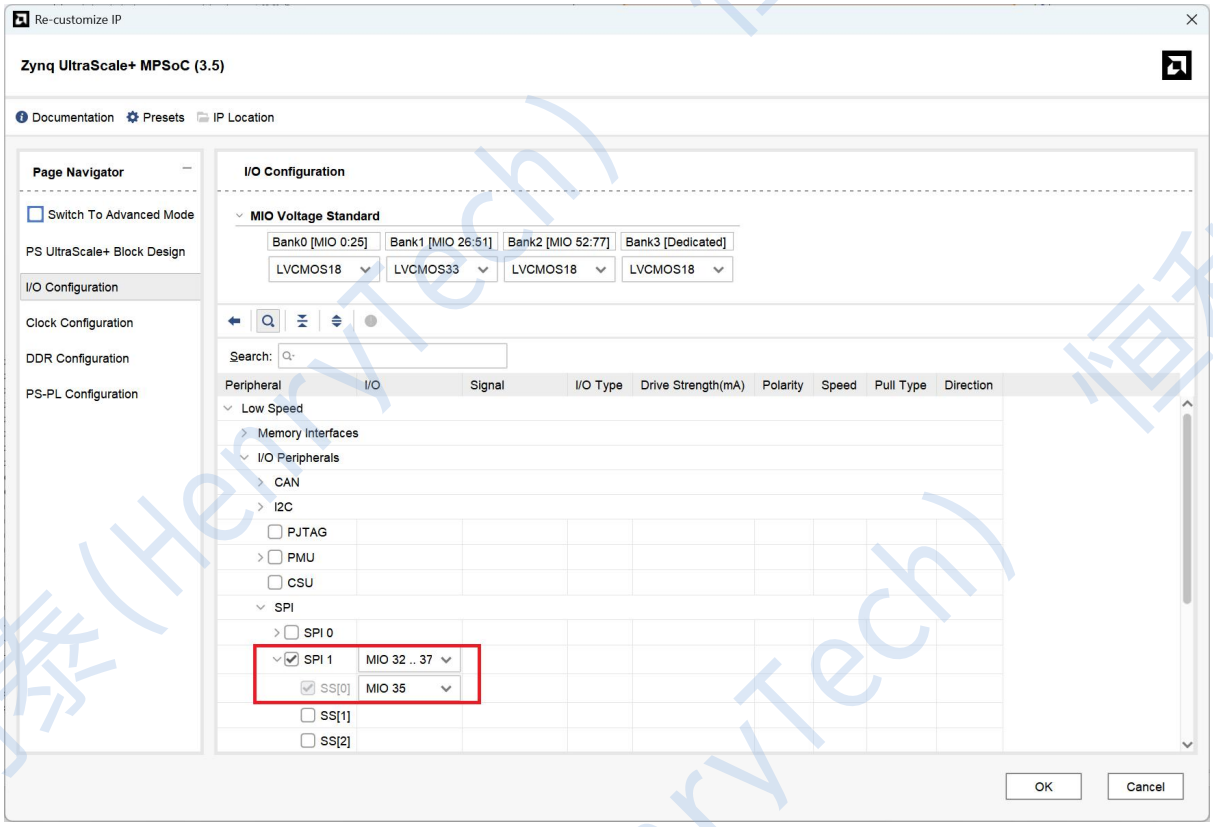


三、调用ZYNQ配置

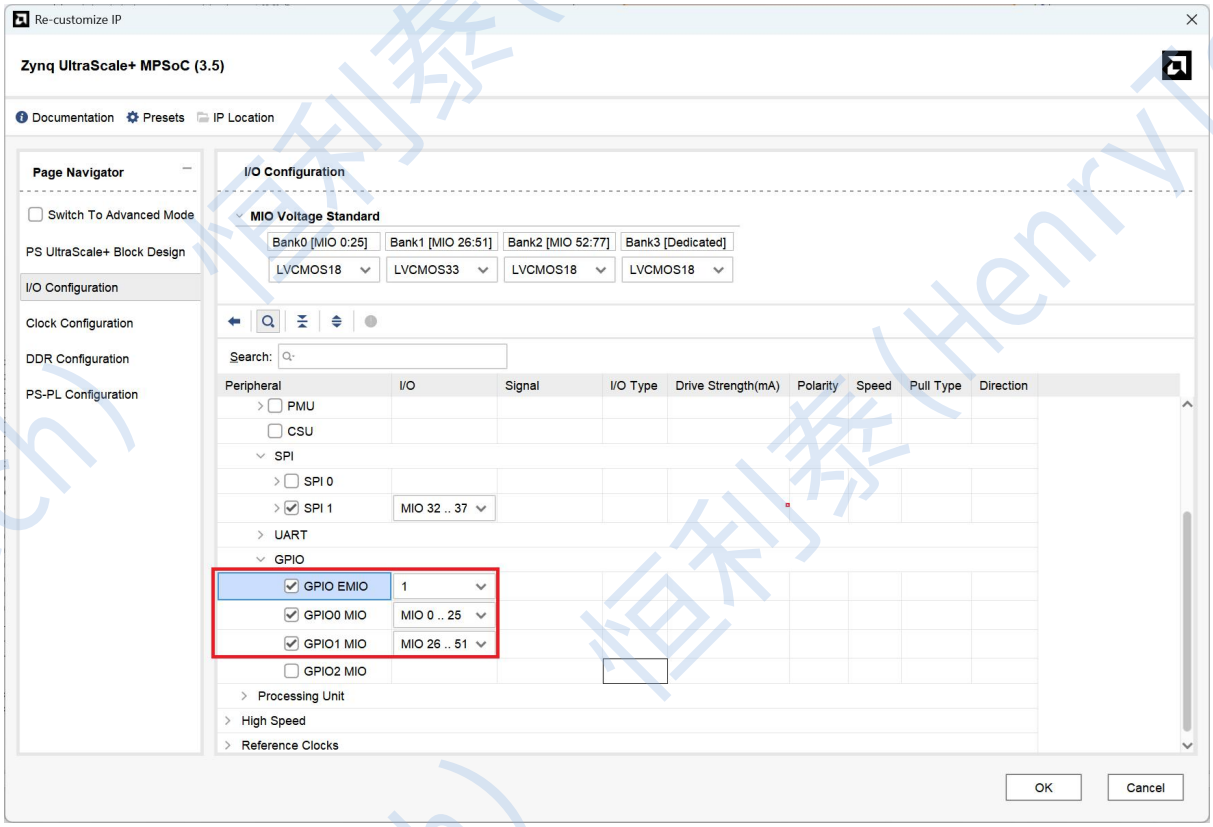
我们再添加一个ZYNQ核，bd点击+号搜索ZYNQ，找到ZYNQ处理器核双击添加：



关于ZYNQ的详细配置，我们可以参考《11PS处理器配置及之helloword打印》文档中详细配置，这里不再重复，DDR4，串口，IO电压等等。在helloword配置基础上，我们需要检查这几个配置，是必需的，如果helloword没有配置，我们要配置一下，首先是SPI，需要用到SPI1,MIO32-37引脚：



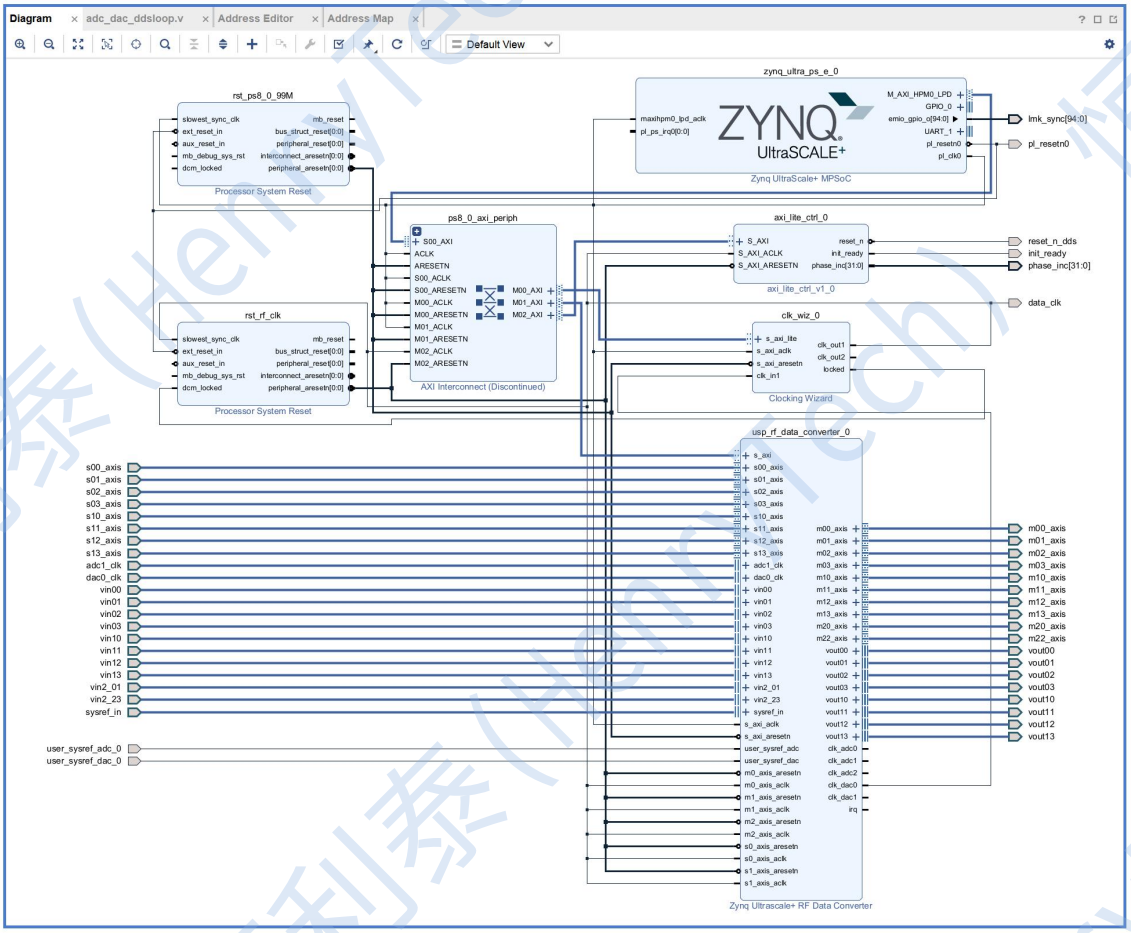
GPIO配置如下，因为MIO会用到，控制LMK04828,EMIO也需要，因为LMK04828的LMK_SYNC同步引脚接在FPGA上，需要EMIO进行扩展：



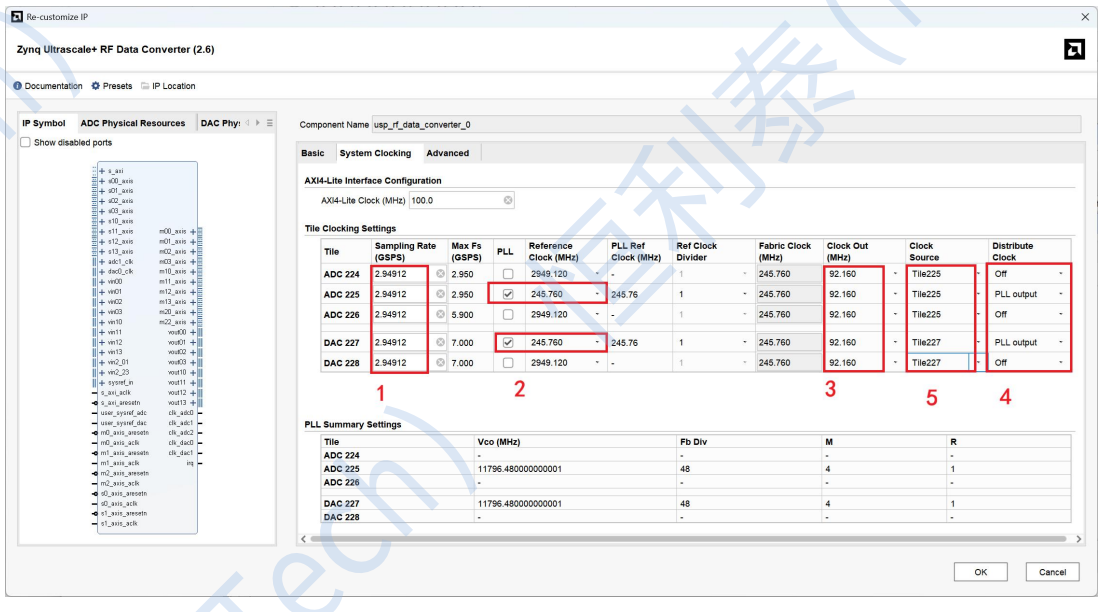
ZYNQ配置到此结束。我们配置好后点击OK完成配置。

四、BD设计介绍

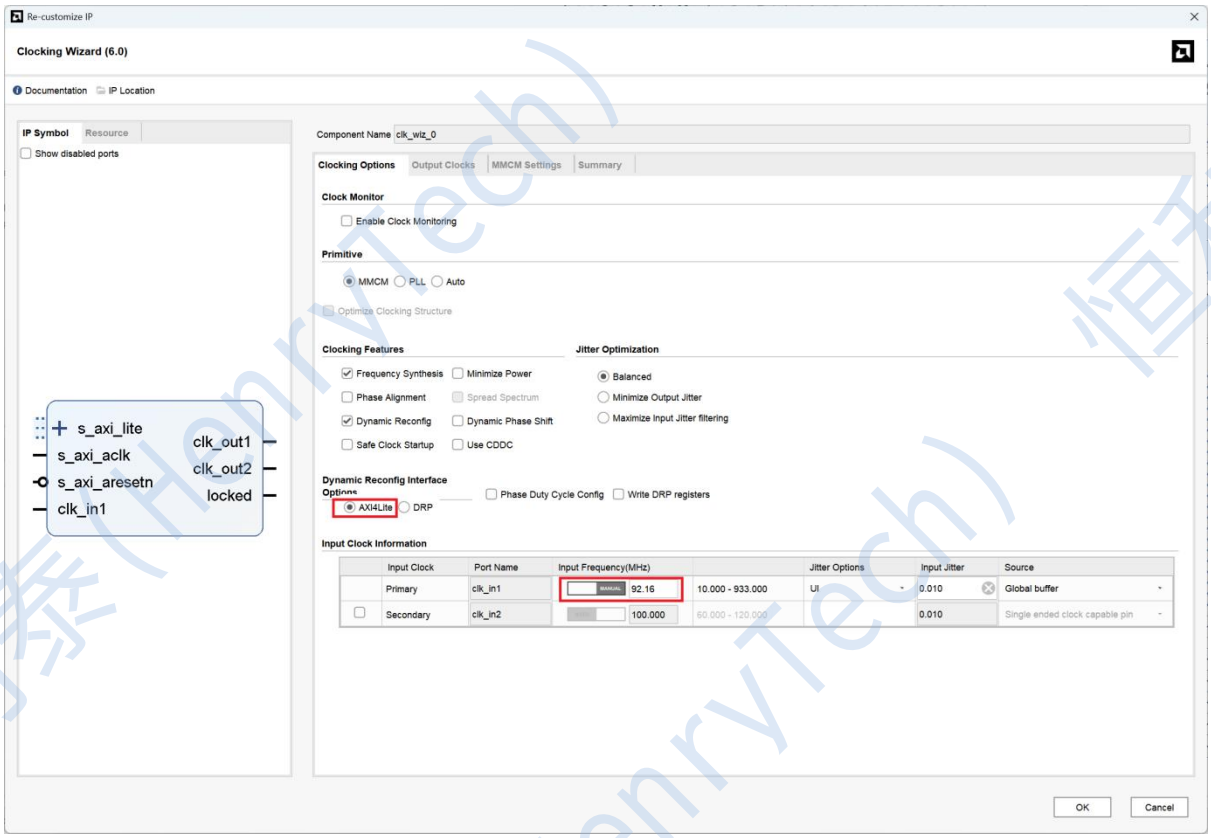
由于设计过程较为烦琐，我们前面两节讲解了RF Data Converter IP以及ZYNQ的调用。对完整BD设计，我们不对具体的设计过程做完整的介绍，我们打开我们例程中《04例程demo/rf_adda_loop/rf67_adda_vivado》工程解压后，打开bd可以看到整个完整设计如下：



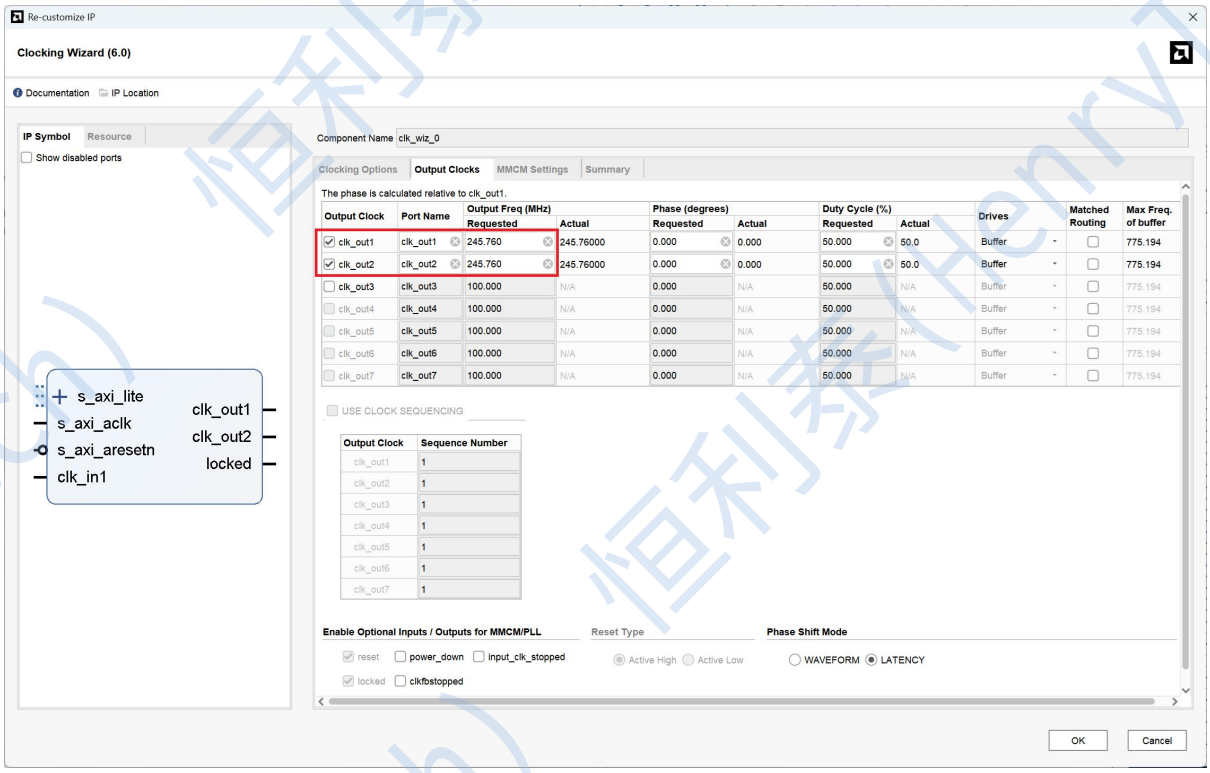
我们对其中的一些功能模块讲解一下。Bd中用到一个时钟模块，clk_wiz模块。时钟输入来自usp_rf_data_converter_0这个IP核的clk_dac0时钟。我们之前第二节中配置过的clock out时钟为92.16M：



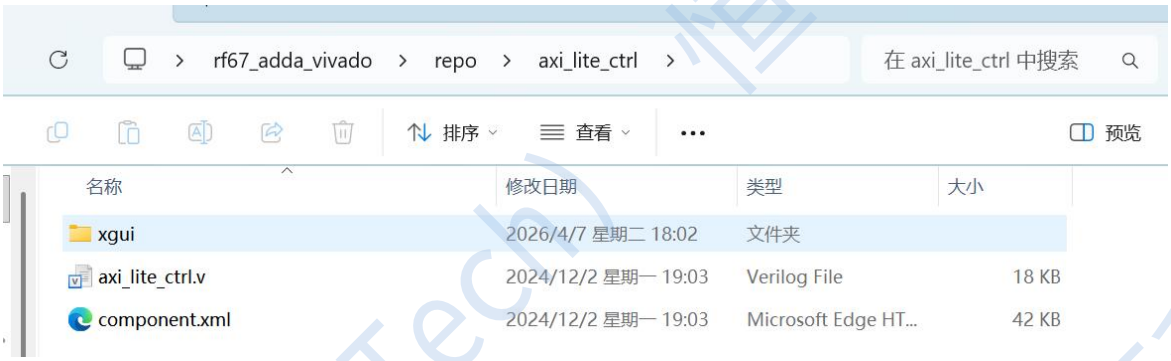
所以我们时钟模块输入配置如下，由于设计中后续可能有动态重新配置时钟需求，所以勾选了AXILite接口：



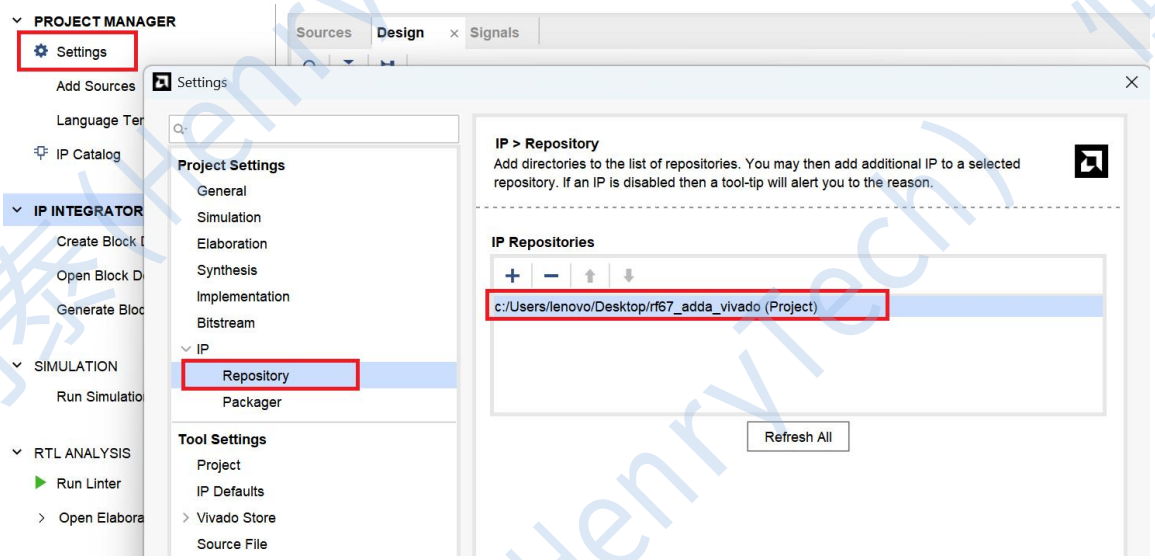
关于输出，由于我们ADC DAC的工作时钟都是245.76MHZ，所以我们时钟输出使用第1和2路clk_out1输出245.76M用于数据接口时钟，实际上只用到clk_out1，第二路待用，后续其他应用预留：



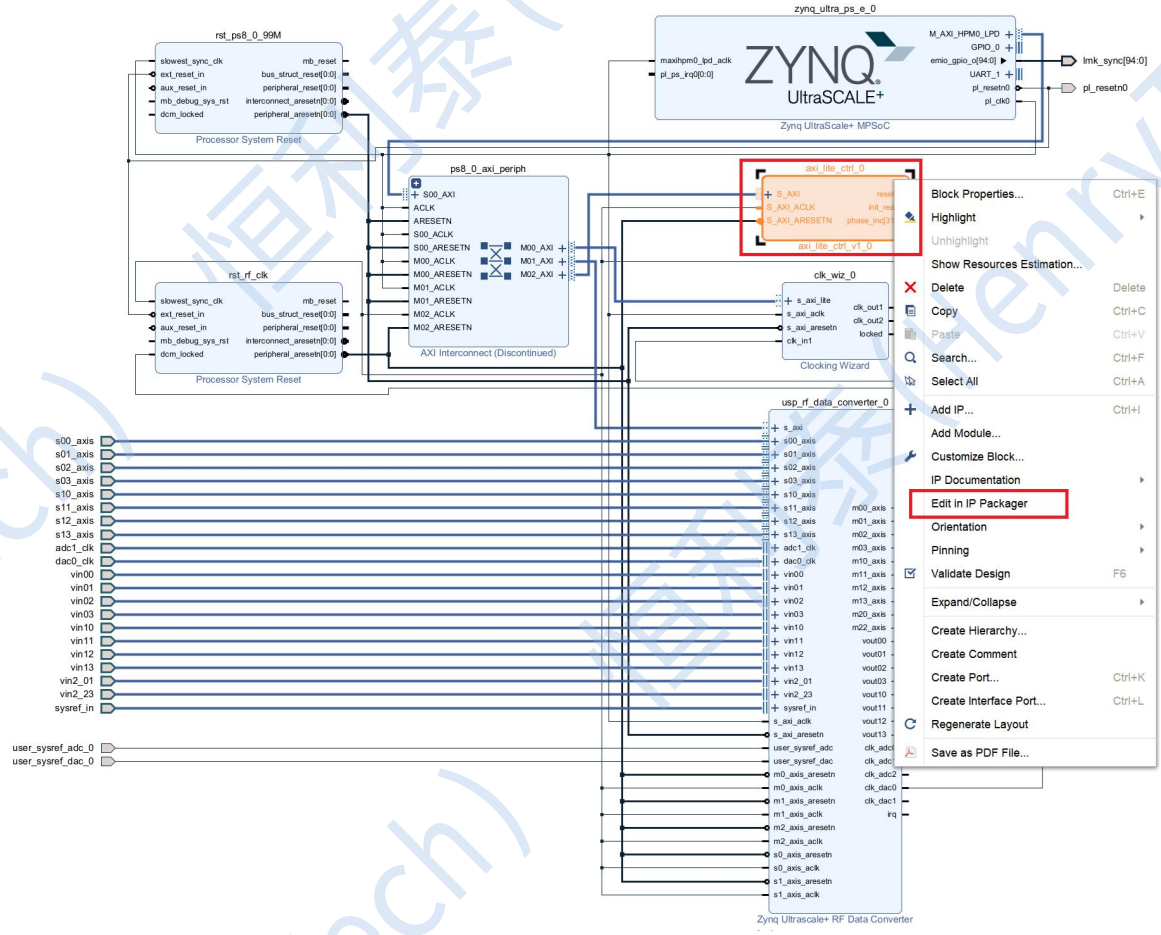
这是时钟模块部分。接着是一个用户自定义的AXI接口IP，叫作axi_lite_ctrl_0，IP位于工程如下位置：



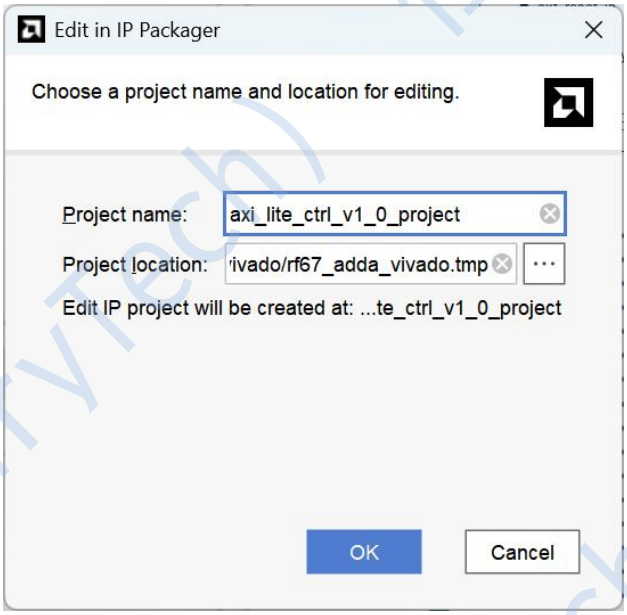
在vivado设置中有这个IP库路径：



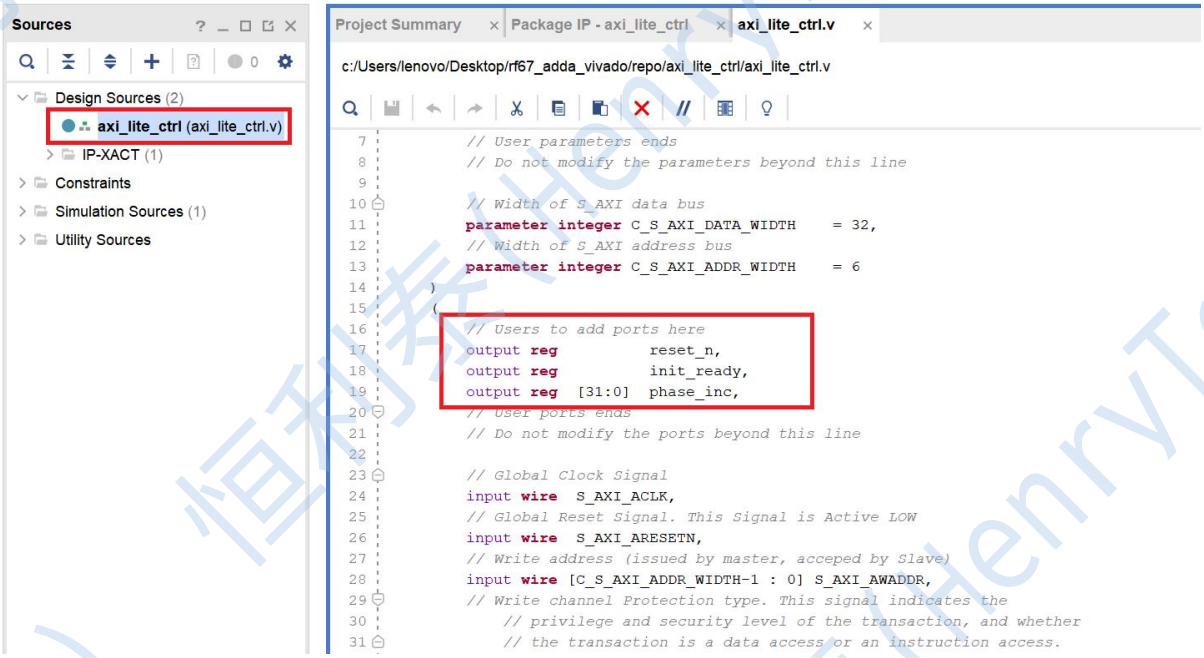
这个模块用户可以自行修改，我们bd选中这个模块右键可以编辑：



启动编辑弹出一个默认的IP编辑工程，点击OK即可：



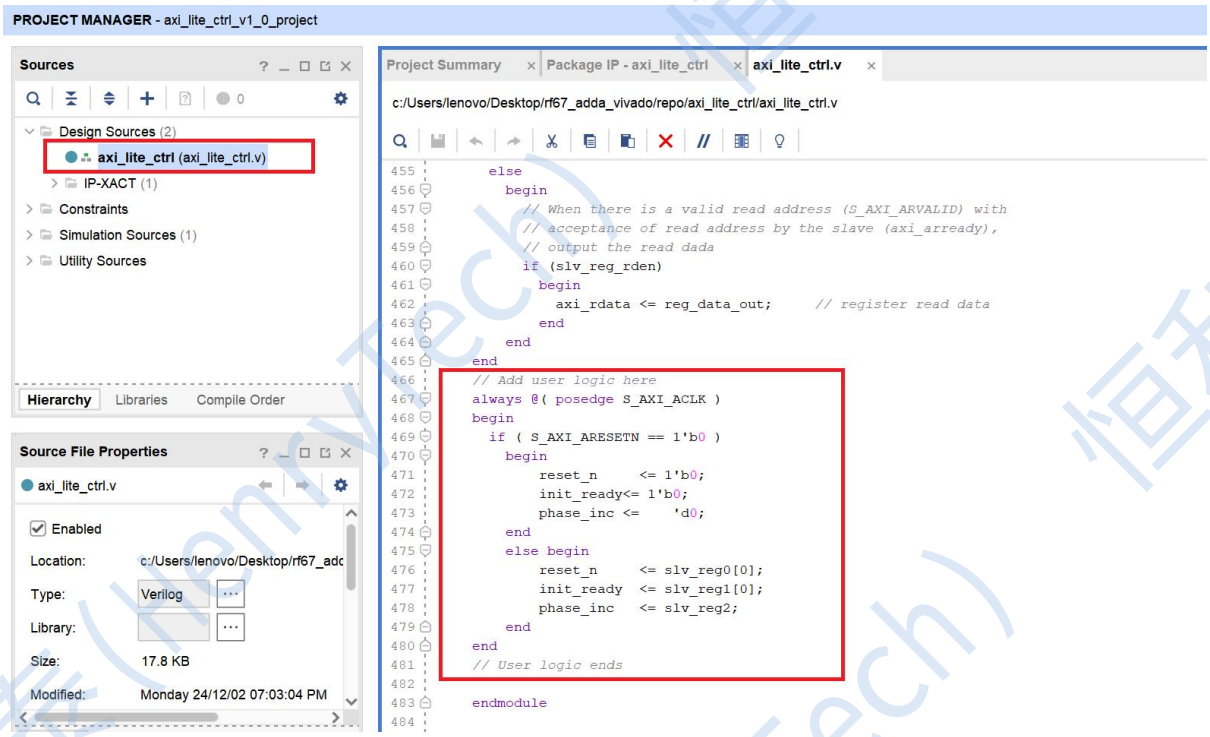
打开进行编辑后，我们可以看到IP代码工程，找到源代码，从接口列表中可以找到几个信号列表：



其中信号释义：

- reset_n是用于用户上层vitis代码控制逻辑复位的信号；
- init_ready是嵌入式代码用户用于指示配置初始化完成的信号；
- phase_inc是我们ADC DAC 射频回环实验，用于控制逻辑的DDS模块正余弦信号波形快慢的增量，越大，正余弦频率越高。

IP代码中466行开始，我们就可以看到，这几个信号直接接到AXI的寄存器上：



用户想要自己更改这部分可以研究一下如何修改，我们这里点到为止。AXI外设的IP可以通过vivado得到IP打包功能创建axi ip模板。

五、FPGA顶层驱动代码讲解

上一节的bd整体作为一个模块，被例化到我们工程的顶层代码模块adc_dac_ddsloop。我们顶层负责两个事情，第一个就是将一些信号引脚连接到IO，用来绑定；第二个作用就是，我们定义了一些debug标记，可以抓取ADC DAC的数据，以及顶层包含DDS模块，可以合成我们DAC的数据输出信号。代码中大量的信号定义和axi接口数据信号定义，顶层完整代码如下：

```
//////////////////////////////////////////////////////////////////////////////////
// Company: adc_dac_ddsloop
// Engineer: yang
// Create Date: 2024/05/16 12:35:02
// Design Name:
// Module Name: adc_dac_ddsloop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////
```

```
`timescale 1ns / 1ps
module adc_dac_ddsloop(
//global clock
input           pl_clk_n,
input           pl_clk_p,
input           pl_sysref_n,
input           pl_sysref_p,
//rf signal
input           adc1_clk_n,
input           adc1_clk_p,

input           dac0_clk_n,
input           dac0_clk_p,
input           sysref_in_n,
input           sysref_in_p,

input           vin00_v_n,
input           vin00_v_p,
input           vin01_v_n,
input           vin01_v_p,
input           vin02_v_n,
input           vin02_v_p,
input           vin03_v_n,
input           vin03_v_p,
input           vin10_v_n,
input           vin10_v_p,
input           vin11_v_n,
input           vin11_v_p,
input           vin12_v_n,
input           vin12_v_p,
input           vin13_v_n,
input           vin13_v_p,
input           vin2_01_v_n,
input           vin2_01_v_p,
input           vin2_23_v_n,
input           vin2_23_v_p,

output          vout00_v_n,
output          vout00_v_p,
output          vout01_v_n,
output          vout01_v_p,
output          vout02_v_n,
output          vout02_v_p,
output          vout03_v_n,
```

恒利泰(HenryTech)

```
output          vout03_v_p,
output          vout10_v_n,
output          vout10_v_p,
output          vout11_v_n,
output          vout11_v_p,
output          vout12_v_n,
output          vout12_v_p,
output          vout13_v_n,
output          vout13_v_p,
//
output          lmk_sync,
output [2:0]    led
);
/*************************************************/
//adc axi master,adc result output bus
wire [63:0]     m00_axis_tdata;
wire            m00_axis_tvalid;
wire [63:0]     m01_axis_tdata;
wire            m01_axis_tvalid;
wire [63:0]     m02_axis_tdata;
wire            m02_axis_tvalid;
wire [63:0]     m03_axis_tdata;
wire            m03_axis_tvalid;
wire [63:0]     m10_axis_tdata;
wire            m10_axis_tvalid;
wire [63:0]     m11_axis_tdata;
wire            m11_axis_tvalid;
wire [63:0]     m12_axis_tdata;
wire            m12_axis_tvalid;
wire [63:0]     m13_axis_tdata;
wire            m13_axis_tvalid;
wire [95:0]     m20_axis_tdata;
wire            m20_axis_tvalid;
wire [95:0]     m22_axis_tdata;
wire            m22_axis_tvalid;
/*********************************************/
//dac axi slave,dac data output bus
wire [127:0]    s00_axis_tdata;
wire            s00_axis_tready;
wire [127:0]    s01_axis_tdata;
wire            s01_axis_tready;
wire [127:0]    s02_axis_tdata;
wire            s02_axis_tready;
wire [127:0]    s03_axis_tdata;
```

恒利泰(HenryTech)

```
wire            s03_axis_tready;
wire [127:0]    s10_axis_tdata;
wire            s10_axis_tready;
wire [127:0]    s11_axis_tdata;
wire            s11_axis_tready;
wire [127:0]    s12_axis_tdata;
wire            s12_axis_tready;
wire [127:0]    s13_axis_tdata;
wire            s13_axis_tready;
// Local Parameters derived from user selection
parameter CHANLE_DAC          = 8;
wire [127:0]dds_data;
wire [CHANLE_DAC-1:0]dac_tready;
wire [9:0]adc_tvalid;
/*****************************************************/
//ADC result define
(*mark_debug="true"*)wire [15:0]adc_data_i_ch1;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch1;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch2;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch2;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch3;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch3;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch4;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch4;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch5;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch5;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch6;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch6;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch7;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch7;
(*mark_debug="true"*)wire [15:0]adc_data_i_ch8;
(*mark_debug="true"*)wire [15:0]adc_data_q_ch8;
(*mark_debug="true"*)wire [15:0]adc_data_ch9;
(*mark_debug="true"*)wire [15:0]adc_data_ch10;
//8 ch iq rf
assign adc_data_i_ch1 = m00_axis_tdata[15:0];
assign adc_data_q_ch1 = m00_axis_tdata[31:16];
assign adc_data_i_ch2 = m01_axis_tdata[15:0];
assign adc_data_q_ch2 = m01_axis_tdata[31:16];
assign adc_data_i_ch3 = m02_axis_tdata[15:0];
assign adc_data_q_ch3 = m02_axis_tdata[31:16];
assign adc_data_i_ch4 = m03_axis_tdata[15:0];
assign adc_data_q_ch4 = m03_axis_tdata[31:16];
assign adc_data_i_ch5 = m10_axis_tdata[15:0];
```

```
assign adc_data_q_ch5 = m10_axis_tdata[31:16];
assign adc_data_i_ch6 = m11_axis_tdata[15:0];
assign adc_data_q_ch6 = m11_axis_tdata[31:16];
assign adc_data_i_ch7 = m12_axis_tdata[15:0];
assign adc_data_q_ch7 = m12_axis_tdata[31:16];
assign adc_data_i_ch8 = m13_axis_tdata[15:0];
assign adc_data_q_ch8 = m13_axis_tdata[31:16];
//2 ch analog
assign adc_data_ch9   = m20_axis_tdata[15:0];
assign adc_data_ch10  = m22_axis_tdata[15:0];
/*************************************************************/
//dac tx valid
assign adc_tvalid[3:0] = {m03_axis_tvalid,m02_axis_tvalid,m01_axis_tvalid,m00_axis_tvalid};
assign adc_tvalid[7:4] = {m13_axis_tvalid,m12_axis_tvalid,m11_axis_tvalid,m10_axis_tvalid};
assign adc_tvalid[9:8] = {m20_axis_tvalid,m22_axis_tvalid};

assign s00_axis_tdata = dds_data;
assign s01_axis_tdata = dds_data;
assign s02_axis_tdata = dds_data;
assign s03_axis_tdata = dds_data;
assign s10_axis_tdata = dds_data;
assign s11_axis_tdata = dds_data;
assign s12_axis_tdata = dds_data;
assign s13_axis_tdata = dds_data;

assign dac_tready[0] = s00_axis_tready;
assign dac_tready[1] = s01_axis_tready;
assign dac_tready[2] = s02_axis_tready;
assign dac_tready[3] = s03_axis_tready;
assign dac_tready[4] = s10_axis_tready;
assign dac_tready[5] = s11_axis_tready;
assign dac_tready[6] = s12_axis_tready;
assign dac_tready[7] = s13_axis_tready;

//DAC output assign
(*mark_debug="true"*)wire [15:0]dac_data_i_point1;
(*mark_debug="true"*)wire [15:0]dac_data_q_point1;
wire [15:0]dac_data_i_point2;
wire [15:0]dac_data_q_point2;
wire [15:0]dac_data_i_point3;
wire [15:0]dac_data_q_point3;
wire [15:0]dac_data_i_point4;
wire [15:0]dac_data_q_point4;
assign dac_data_i_point1 = dds_data[15:0];
```

恒利泰(HenryTech)

```
assign dac_data_q_point1 = dds_data[31:16];
assign dac_data_i_point2 = dds_data[47:32];
assign dac_data_q_point2 = dds_data[63:48];
assign dac_data_i_point3 = dds_data[79:64];
assign dac_data_q_point3 = dds_data[95:80];
assign dac_data_i_point4 = dds_data[111:96];
assign dac_data_q_point4 = dds_data[127:112];

reg             user_sysref_adc;
reg             user_sysref_dac;
wire init_ready;
wire reset_n_dds;
wire [31:0]phase_inc;
system system_i
(
    .adc1_clk_clk_n      (adc1_clk_n),
    .adc1_clk_clk_p      (adc1_clk_p),

    .dac0_clk_clk_n      (dac0_clk_n),
    .dac0_clk_clk_p      (dac0_clk_p),

    .data_clk            (data_clk),
    .init_ready          (init_ready),
    .lmk_sync            (lmk_sync),

    .m00_axis_tdata      (m00_axis_tdata),
    .m00_axis_tready     (1'b1),
    .m00_axis_tvalid     (m00_axis_tvalid),
    .m01_axis_tdata      (m01_axis_tdata),
    .m01_axis_tready     (1'b1),
    .m01_axis_tvalid     (m01_axis_tvalid),
    .m02_axis_tdata      (m02_axis_tdata),
    .m02_axis_tready     (1'b1),
    .m02_axis_tvalid     (m02_axis_tvalid),
    .m03_axis_tdata      (m03_axis_tdata),
    .m03_axis_tready     (1'b1),
    .m03_axis_tvalid     (m03_axis_tvalid),
    .m10_axis_tdata      (m10_axis_tdata),
    .m10_axis_tready     (1'b1),
    .m10_axis_tvalid     (m10_axis_tvalid),
    .m11_axis_tdata      (m11_axis_tdata),
    .m11_axis_tready     (1'b1),
    .m11_axis_tvalid     (m11_axis_tvalid),
    .m12_axis_tdata      (m12_axis_tdata),
```

```
    .m12_axis_tready    (1'b1),
    .m12_axis_tvalid    (m12_axis_tvalid),
    .m13_axis_tdata     (m13_axis_tdata),
    .m13_axis_tready    (1'b1),
    .m13_axis_tvalid    (m13_axis_tvalid),

    .m20_axis_tdata     (m20_axis_tdata),
    .m20_axis_tready    (1'b1),
    .m20_axis_tvalid    (m20_axis_tvalid),
    .m22_axis_tdata     (m22_axis_tdata),
    .m22_axis_tready    (1'b1),
    .m22_axis_tvalid    (m22_axis_tvalid),

    .phase_inc          (phase_inc),
    .pl_resetn0         (fclk_reset0),
    .reset_n_dds        (reset_n_dds),

    .s00_axis_tdata     (s00_axis_tdata),
    .s00_axis_tready    (s00_axis_tready),
    .s00_axis_tvalid    (1'b1),
    .s01_axis_tdata     (s01_axis_tdata),
    .s01_axis_tready    (s01_axis_tready),
    .s01_axis_tvalid    (1'b1),
    .s02_axis_tdata     (s02_axis_tdata),
    .s02_axis_tready    (s02_axis_tready),
    .s02_axis_tvalid    (1'b1),
    .s03_axis_tdata     (s03_axis_tdata),
    .s03_axis_tready    (s03_axis_tready),
    .s03_axis_tvalid    (1'b1),

    .s10_axis_tdata     (s10_axis_tdata),
    .s10_axis_tready    (s10_axis_tready),
    .s10_axis_tvalid    (1'b1),
    .s11_axis_tdata     (s11_axis_tdata),
    .s11_axis_tready    (s11_axis_tready),
    .s11_axis_tvalid    (1'b1),
    .s12_axis_tdata     (s12_axis_tdata),
    .s12_axis_tready    (s12_axis_tready),
    .s12_axis_tvalid    (1'b1),
    .s13_axis_tdata     (s13_axis_tdata),
    .s13_axis_tready    (s13_axis_tready),
    .s13_axis_tvalid    (1'b1),

    .sysref_in_diff_n   (sysref_in_n),
```

```
    .sysref_in_diff_p   (sysref_in_p),
    .user_sysref_adc_0  (user_sysref_adc),
    .user_sysref_dac_0  (user_sysref_dac),

    .vin00_v_n          (vin00_v_n),
    .vin00_v_p          (vin00_v_p),
    .vin01_v_n          (vin01_v_n),
    .vin01_v_p          (vin01_v_p),
    .vin02_v_n          (vin02_v_n),
    .vin02_v_p          (vin02_v_p),
    .vin03_v_n          (vin03_v_n),
    .vin03_v_p          (vin03_v_p),
    .vin10_v_n          (vin10_v_n),
    .vin10_v_p          (vin10_v_p),
    .vin11_v_n          (vin11_v_n),
    .vin11_v_p          (vin11_v_p),
    .vin12_v_n          (vin12_v_n),
    .vin12_v_p          (vin12_v_p),
    .vin13_v_n          (vin13_v_n),
    .vin13_v_p          (vin13_v_p),
    .vin2_01_v_n          (vin2_01_v_n),
            .vin2_01_v_p              (vin2_01_v_p),
            .vin2_23_v_n              (vin2_23_v_n),
            .vin2_23_v_p              (vin2_23_v_p),

    .vout00_v_n         (vout00_v_n),
    .vout00_v_p         (vout00_v_p),
    .vout01_v_n         (vout01_v_n),
    .vout01_v_p         (vout01_v_p),
    .vout02_v_n         (vout02_v_n),
    .vout02_v_p         (vout02_v_p),
    .vout03_v_n         (vout03_v_n),
    .vout03_v_p         (vout03_v_p),
    .vout10_v_n         (vout10_v_n),
    .vout10_v_p         (vout10_v_p),
    .vout11_v_n         (vout11_v_n),
    .vout11_v_p         (vout11_v_p),
    .vout12_v_n         (vout12_v_n),
    .vout12_v_p         (vout12_v_p),
    .vout13_v_n         (vout13_v_n),
    .vout13_v_p         (vout13_v_p)
);

// Multi-tile sync logic
```

恒利泰(HenryTech)

```
  IBUFDS pl_clk_i
  (
    .I          (pl_clk_p),
    .IB         (pl_clk_n),
    .O          (pl_clk)
    );

  IBUFDS pl_sysref_i
  (
    .I          (pl_sysref_p),
    .IB         (pl_sysref_n),
    .O          (pl_sysref)
    );
reg pl_sysref_reg;
always @(posedge pl_clk)pl_sysref_reg <= pl_sysref;
always @(posedge pl_clk)user_sysref_adc <= pl_sysref_reg;
always @(posedge pl_clk)user_sysref_dac <= pl_sysref_reg;
//tx dds
(*mark_debug="true"*)reg [63:0] phase_dds_in;
(*mark_debug="true"*)reg [15:0]phase_inc_reg;
always @ (posedge data_clk)begin
    if(!reset_n_dds)begin
        phase_inc_reg<=phase_inc;
        phase_dds_in[16*0 +:16]<=16'd0;//point 1
        phase_dds_in[16*1 +:16]<=phase_inc_reg;//point 2
        phase_dds_in[16*2 +:16]<=phase_inc_reg+phase_inc_reg;//point 3
        phase_dds_in[16*3 +:16]<=phase_inc_reg+phase_inc_reg+phase_inc_reg;point 4
    end
    else begin
        phase_dds_in[16*0 +:16]<=phase_dds_in[16*0 +:16]+{phase_inc_reg,2'd0};point 1
        phase_dds_in[16*1 +:16]<=phase_dds_in[16*1 +:16]+{phase_inc_reg,2'd0};point 2
        phase_dds_in[16*2 +:16]<=phase_dds_in[16*2 +:16]+{phase_inc_reg,2'd0};point 3
        phase_dds_in[16*3 +:16]<=phase_dds_in[16*3 +:16]+{phase_inc_reg,2'd0};point 4
  end
end

genvar ch;
generate
    for (ch = 0; ch < 4; ch = ch + 1) begin
        dds_compiler_0 dds_compiler_0
        (
            .aclk                 (data_clk          ), input wire aclk
            .aresetn              (reset_n_dds       ), input wire aresetn
            .s_axis_phase_tvalid  (1'b1              ), input wire s_axis_phase_tvalid
```

恒利泰(HenryTech)

```
            .s_axis_phase_tdata    (phase_dds_in[16*ch +:16] ), input wire [7 : 0] s_axis_phase_tdata
            .m_axis_data_tvalid    ( ),  // output wire m_axis_data_tvalid
            .m_axis_data_tdata     (dds_data[32*ch +:32])// output wire [31 : 0] m_axis_data_tdata
        );
    end
endgenerate

//led
assign led[0]=1'b0;
assign led[1]=1'b0;
assign led[2]=init_ready;

endmodule
```

代码讲解（代码行位置请具体参考vivado工程）：

1.代码17-72行是整个工程的顶层信号，连接到芯片RFSOC 67DR外面，包含pl_clk_p/n差分时钟，pl_sysref_p/n的pl ref参考时钟，adc1_clk_p/n和dac0_clk_p/n是ADC DAC工作时钟。另外针对ADC DAC同步，sysref_in_p/n是用于ADC DAC内部多title同步的参考信号。其余信号，vin字样的是ADC输入信号引脚。vout部分是DAC输出引脚。列表中lmk_sync是时钟芯片lmk04828的同步操作控制引脚。最后信号列表还有led。

2.代码73-94行是我们ADC的AXI数据输出接口，输出10路ADC采样信号结果。

3.96行-112行代码，是我们的DAC输出的AXI数据输入接口。

4.114-117定义了dac通道数量和输出的数据和有效信号。这几个信号最终连接到DAC的AXI端口。

5.119行-162行，定义了ADC信号，assign连接到axi接口上，并且连接adc的axi_tvalid有效信号。

6.164行-198行，信号将dds连接到axi上，作为dac通道输出，以及定义了dac的i和q的四个采样点的数据信号，连接到dds上作为观测；

7.200-204定义了几个信号。其中，user_sysref_adc user_sysref_dac是连接到bd中，用于ADC DAC内部片内多Title同步的时钟参考；reset_n_dds，phase_inc则是来自bd中，axi lite ctrl IP核用于控制DDS的复位，相位快慢的控制信号，init_ready则是表示vitis中嵌入式代码初始化完成后的标志。

8.205-321行为我们bd设计的顶层例化实例；

9.324到340行，代码中使用了两个BUF用于时钟和参考时钟的差分转单端，然后通过pl_clk时钟对pl_sysref打拍，最后分出user_sysref_adc和user_sysref_dac两个参考信号，最后给到bd设计中的adc dac同步输入，用作Title 同步用。

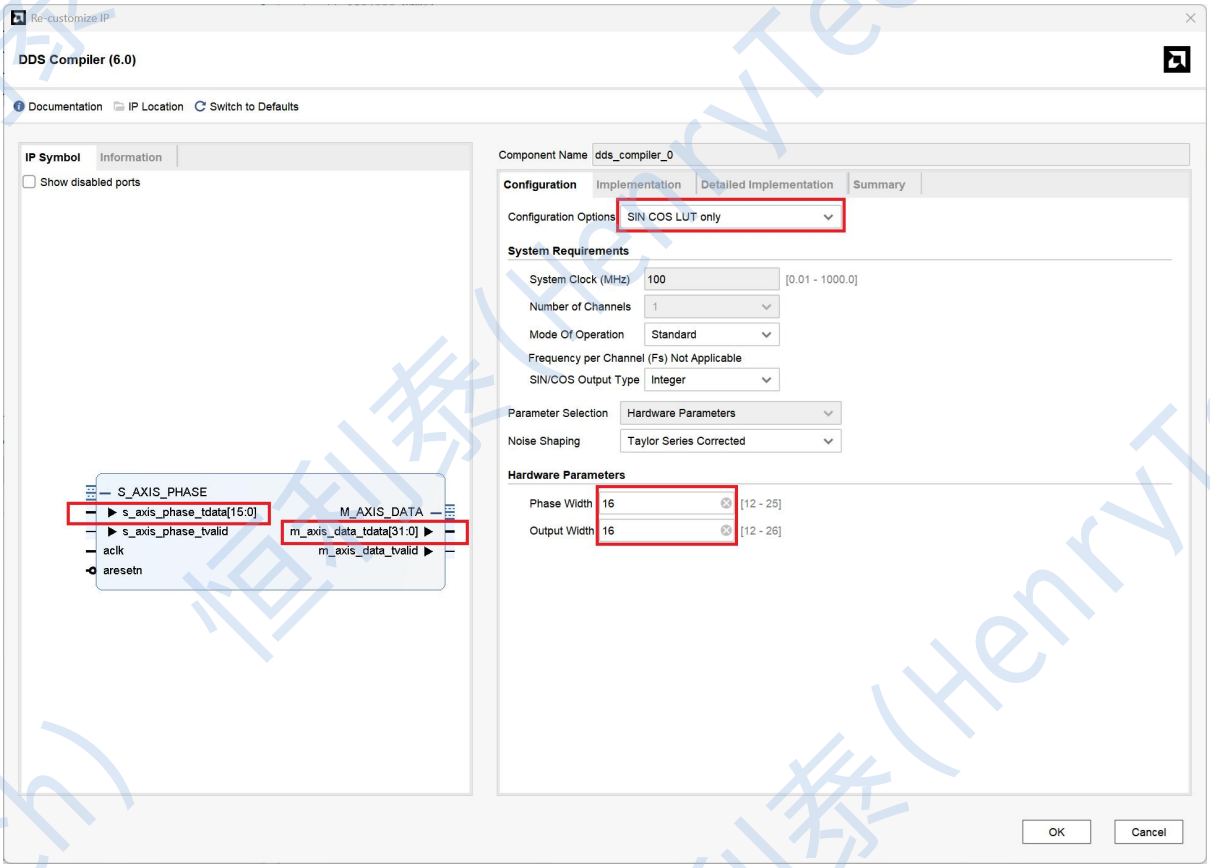
10.我们重点讲解一下342-373行，这部分是我们生成DDS信号输出正余弦的DAC数据的发生逻辑。这部分代码中，345行-350行的敏感块中，我们在复位reset_n_dds为0的时候，初始化了初

值phase_inc_reg为phase_inc，假设phase_inc为512，那么嵌入式代码中reset_n_dds拉低后，我们phase_inc_reg就是512。并且phase_dds_in输入的四路数据，会被初始化为四个相邻增量的相位DDS输入数据：

```
phase_dds_in[16*0 +:16]<=16'd0;point 1
phase_dds_in[16*1 +:16]<=phase_inc_reg;point 2
phase_dds_in[16*2 +:16]<=phase_inc_reg+phase_inc_reg;point 3
phase_dds_in[16*3 +:16]<=phase_inc_reg+phase_inc_reg+phase_inc_reg;point 4
```

上述四个16bit 的phase_dds_in最终输入到360行-373行的dds中。dds例化了4个，因为generate的for循环到4。简单来说就是，我们初始化的四个phase_dds_in相位依次相差一个点的输入相位增量，所以dds输出就是四个相邻的采样点。为什么要构成四个相邻采样点？因为dac的axi接口，从我们之前配置IP中可以知道，一个时钟周期要给4个采样点（8个I/Q数据）。

11. 360行-373行代码就是例化的4个DDS模块，通过phase_dds_in不停地线性变化增加做输入，dds输出dds_data正余弦数据。我们DDS IP配置如下：

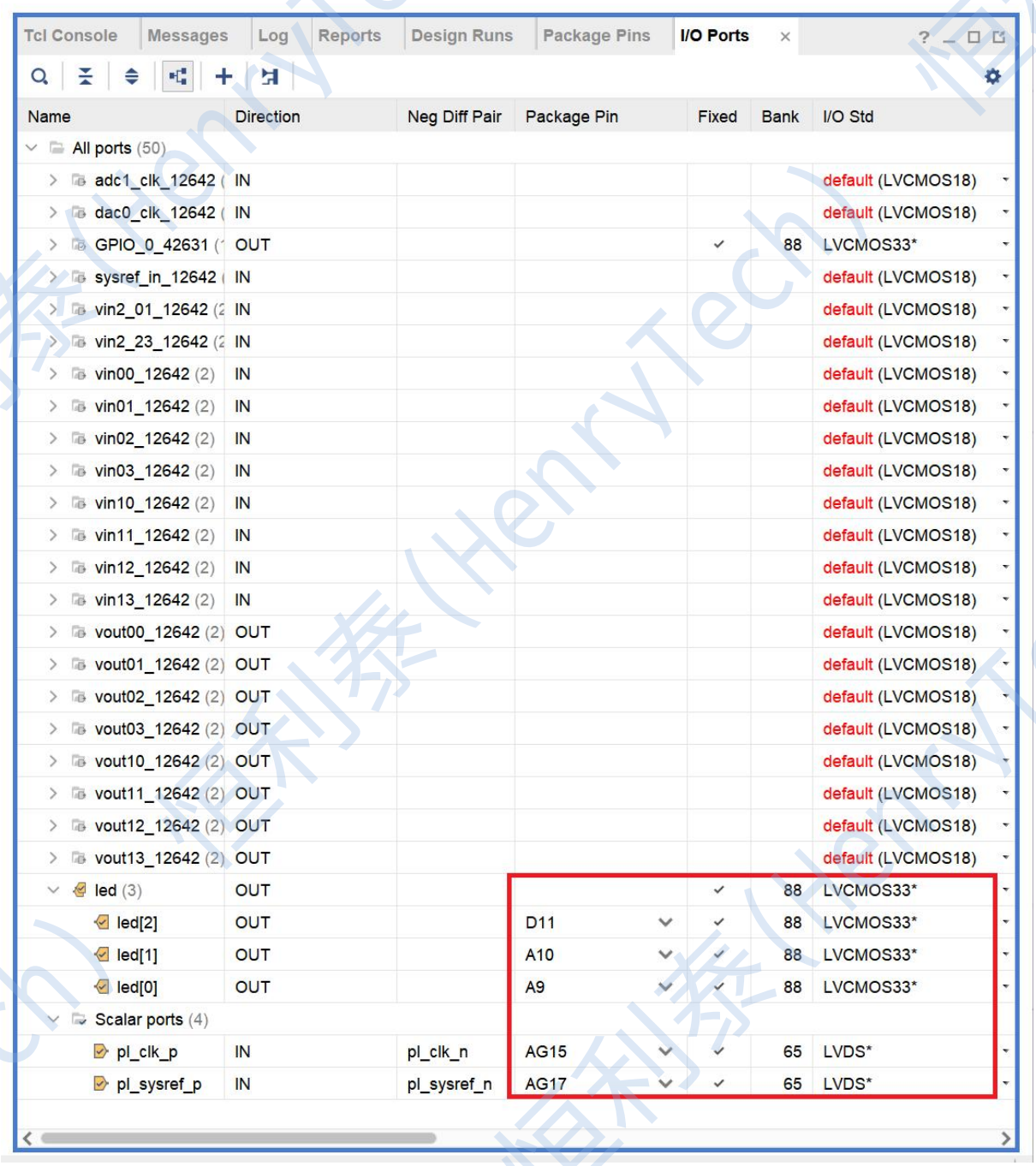


其中Configuration option是SIN COS LUT only表示正余弦输出，LUT查找表方式合成。下面的phase width和output width为16位。但是左边IP观察可以看到，axi接口输入是16位宽，输出却是32位，为什么？因为32位宽0-15位是SIN正弦波输出，16-31是COS余弦波输出。由于DAC输入数据要求I/Q输入，axi输入是8个数据，I/Q交错的，所以DDS正余弦刚好适合直接连接IQ数据输出到DAC方便观察。通过394行-401行敏感块，给到dac_tdata。dac_tdata前面代码通过assign连接到了DAC的以s开头的axi接口上。

至此我们代码简单讲解完成。

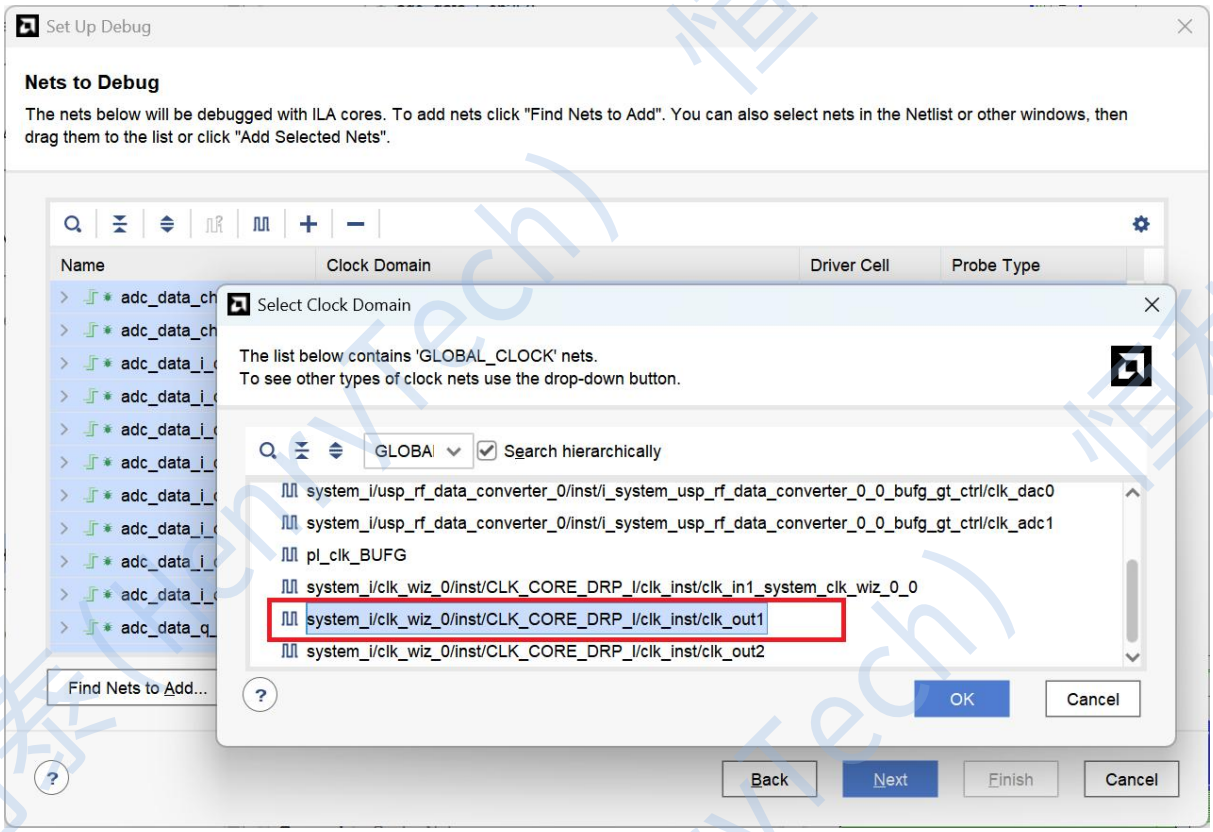
六、编译生成bit文件导出硬件

我们将工程编译，绑定引脚，我们只需要绑定如下几个引脚，由于ADC DAC属于芯片固定引脚，类似ZYNQ IP核的MIO引脚，预先分配好，硬件固定，不需要绑定，引脚约束中不绑定最终也能正常生成BIT文件，这些可以理解为是固定分配，vivado软件还未对RFSOC的ADC DAC固定分配引脚做优化，所以还是会被显示，只不过我们可以不理会：

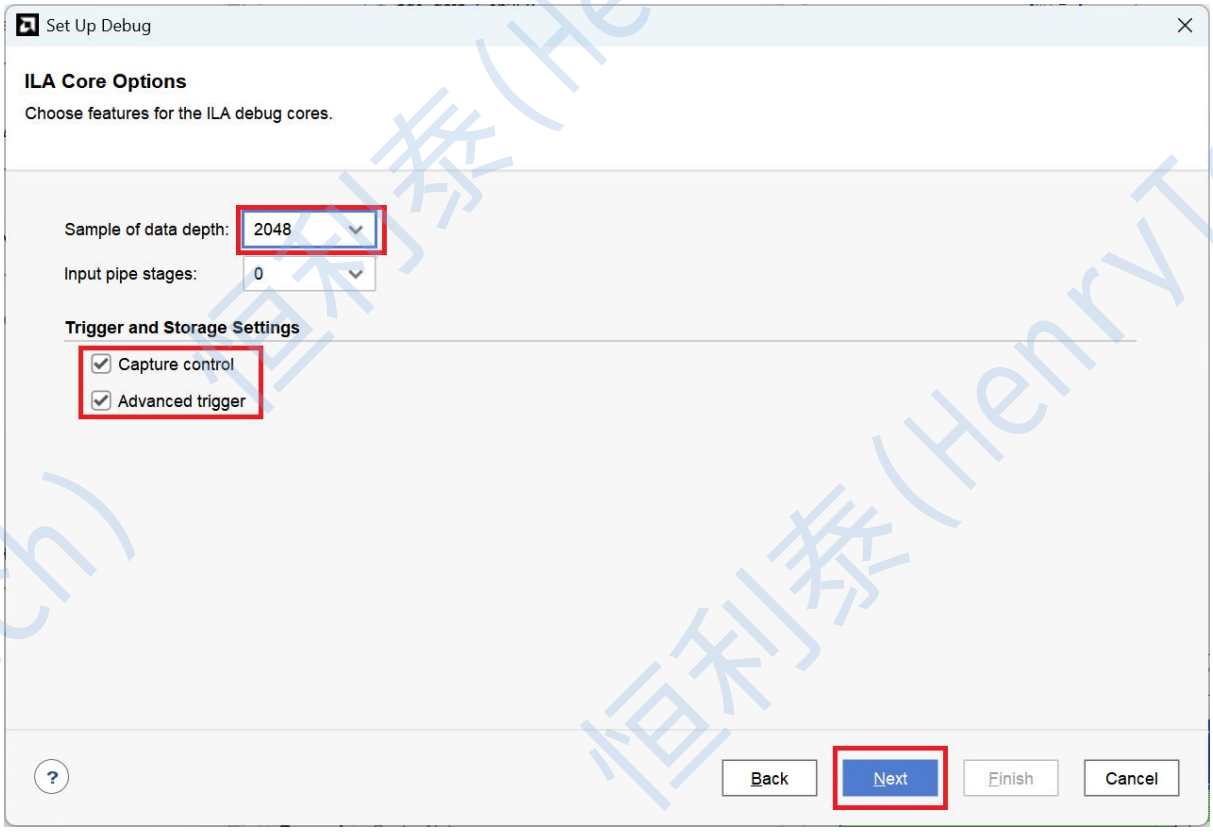


Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (50)						
adc1_clk_12642 (	IN					default (LVCMOS18)
dac0_clk_12642 (	IN					default (LVCMOS18)
GPIO_0_42631 (1	OUT			✓	88	LVCMOS33*
sysref_in_12642 (	IN					default (LVCMOS18)
vin2_01_12642 (2	IN					default (LVCMOS18)
vin2_23_12642 (2	IN					default (LVCMOS18)
vin00_12642 (2)	IN					default (LVCMOS18)
vin01_12642 (2)	IN					default (LVCMOS18)
vin02_12642 (2)	IN					default (LVCMOS18)
vin03_12642 (2)	IN					default (LVCMOS18)
vin10_12642 (2)	IN					default (LVCMOS18)
vin11_12642 (2)	IN					default (LVCMOS18)
vin12_12642 (2)	IN					default (LVCMOS18)
vin13_12642 (2)	IN					default (LVCMOS18)
vout00_12642 (2)	OUT					default (LVCMOS18)
vout01_12642 (2)	OUT					default (LVCMOS18)
vout02_12642 (2)	OUT					default (LVCMOS18)
vout03_12642 (2)	OUT					default (LVCMOS18)
vout10_12642 (2)	OUT					default (LVCMOS18)
vout11_12642 (2)	OUT					default (LVCMOS18)
vout12_12642 (2)	OUT					default (LVCMOS18)
vout13_12642 (2)	OUT					default (LVCMOS18)
led (3)	OUT			✓	88	LVCMOS33*
led[2]	OUT		D11	✓	88	LVCMOS33*
led[1]	OUT		A10	✓	88	LVCMOS33*
led[0]	OUT		A9	✓	88	LVCMOS33*
Scalar ports (4)						
pl_clk_p	IN	pl_clk_n	AG15	✓	65	LVDS*
pl_sysref_p	IN	pl_sysref_n	AG17	✓	65	LVDS*

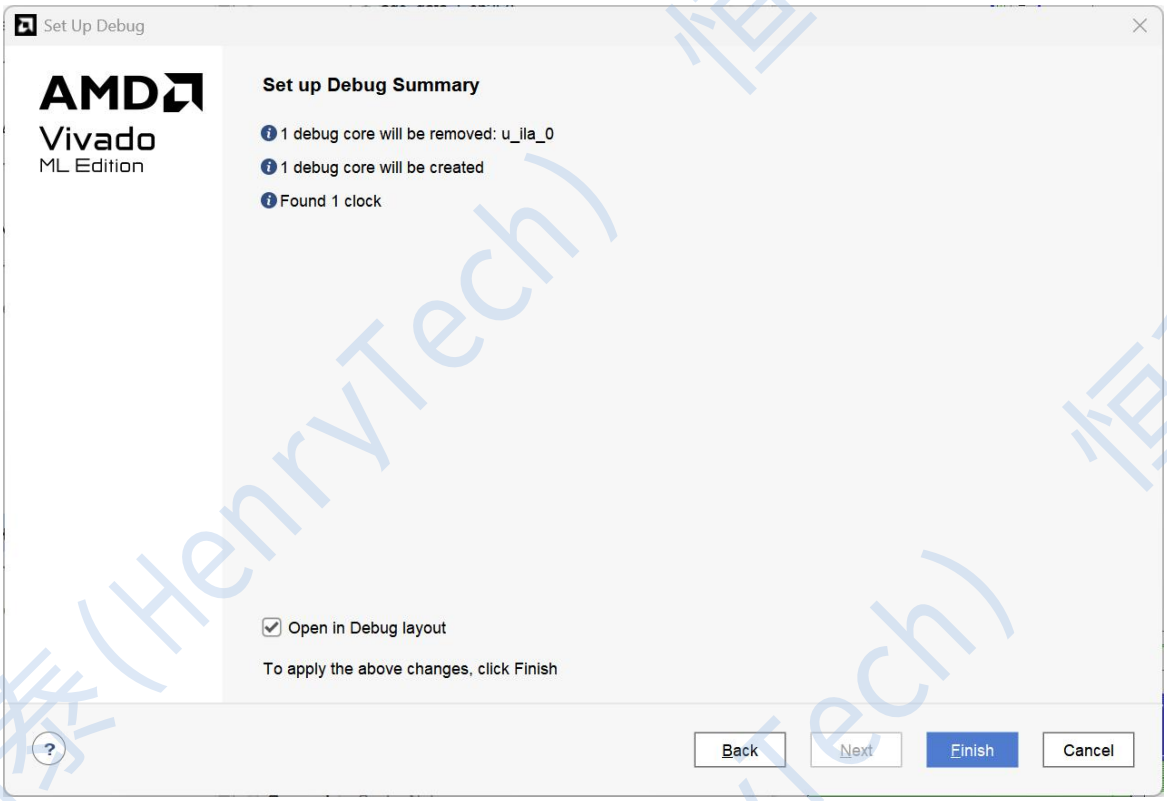
绑定好后，我们设置DEBUG。代码顶层设置好了mark debug标记，我们debug设置如下，首先是采样时钟，是clk_wiz_0这个IP的clk_out1：



采样深度和触发储设置如下：

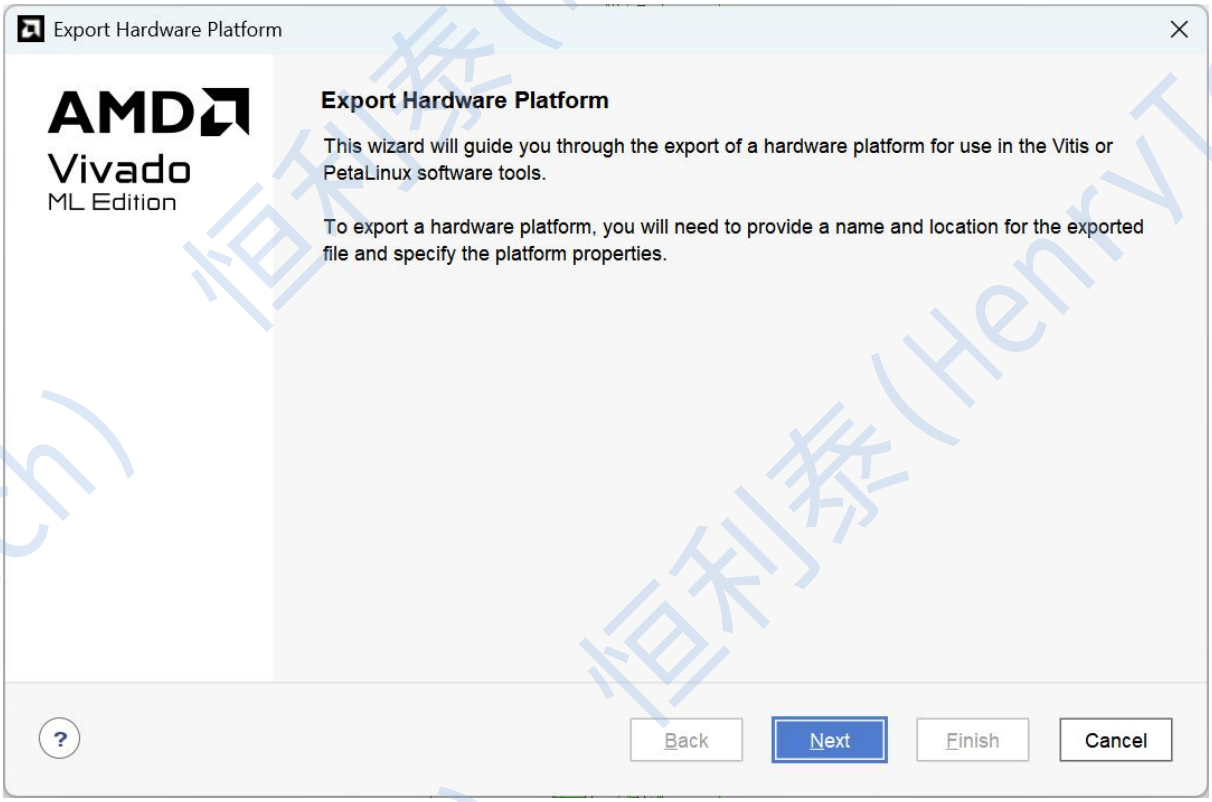


最后一页直接点击Finish:

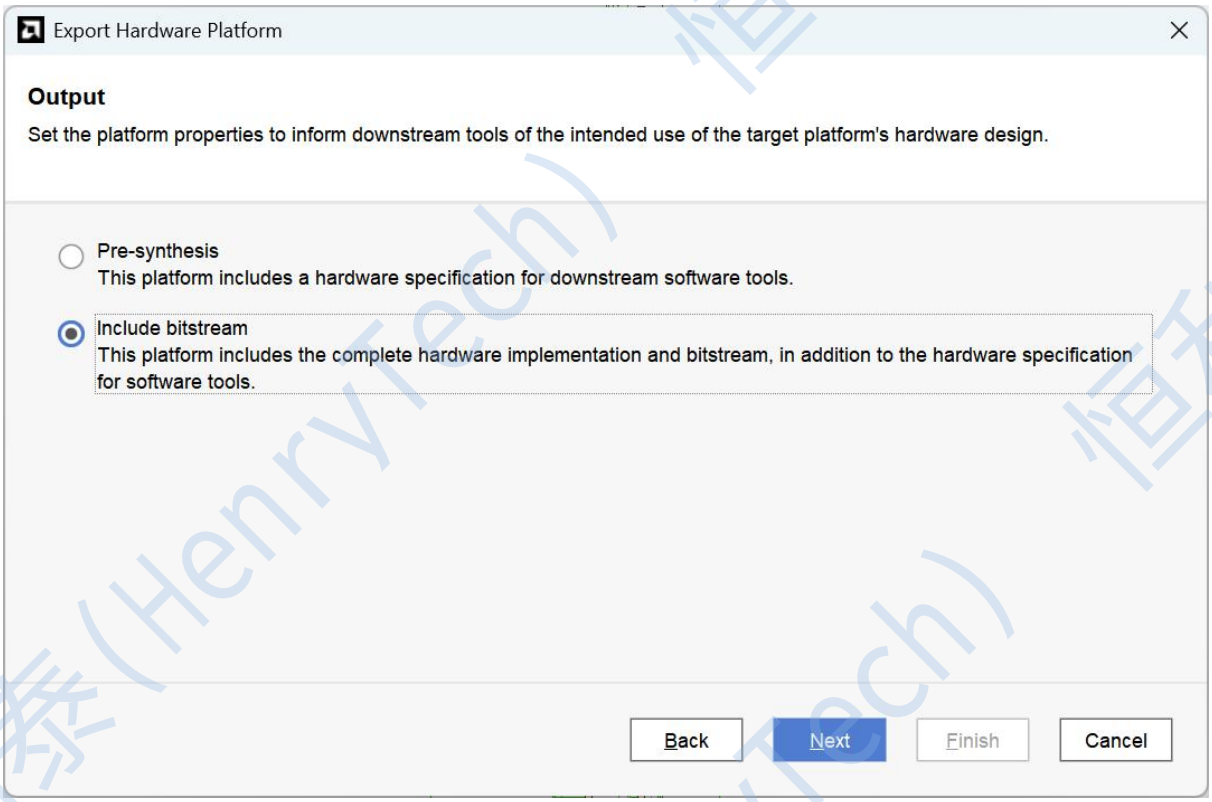


配置好ila按ctrl+s保存一下。我们直接生成bit。如果需要更改采样debug可以自己重新修改。如何使用mark debug参考《07在线调试之进阶MARK_DEBUG》，这里不重复。

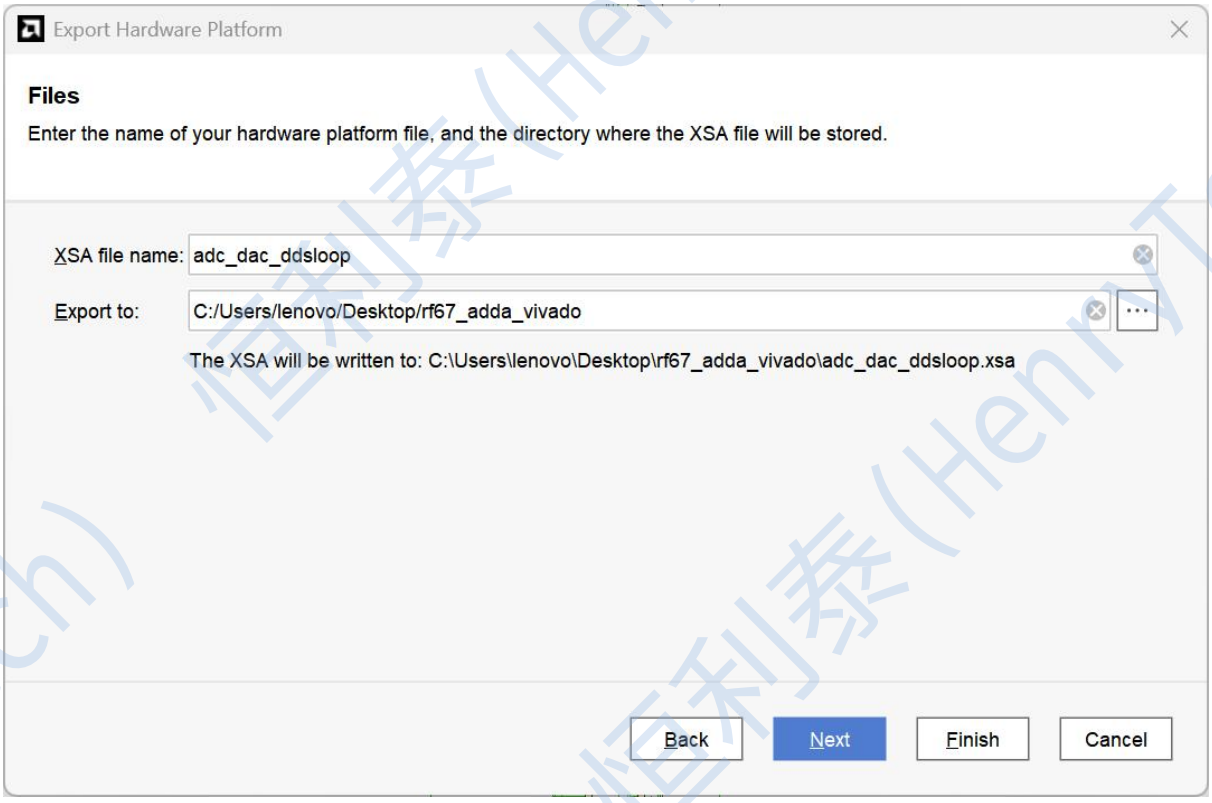
生成好bit文件后我们导出硬件xsa文件。Vivado菜单栏file->export->export hardware，第一页点击Next：



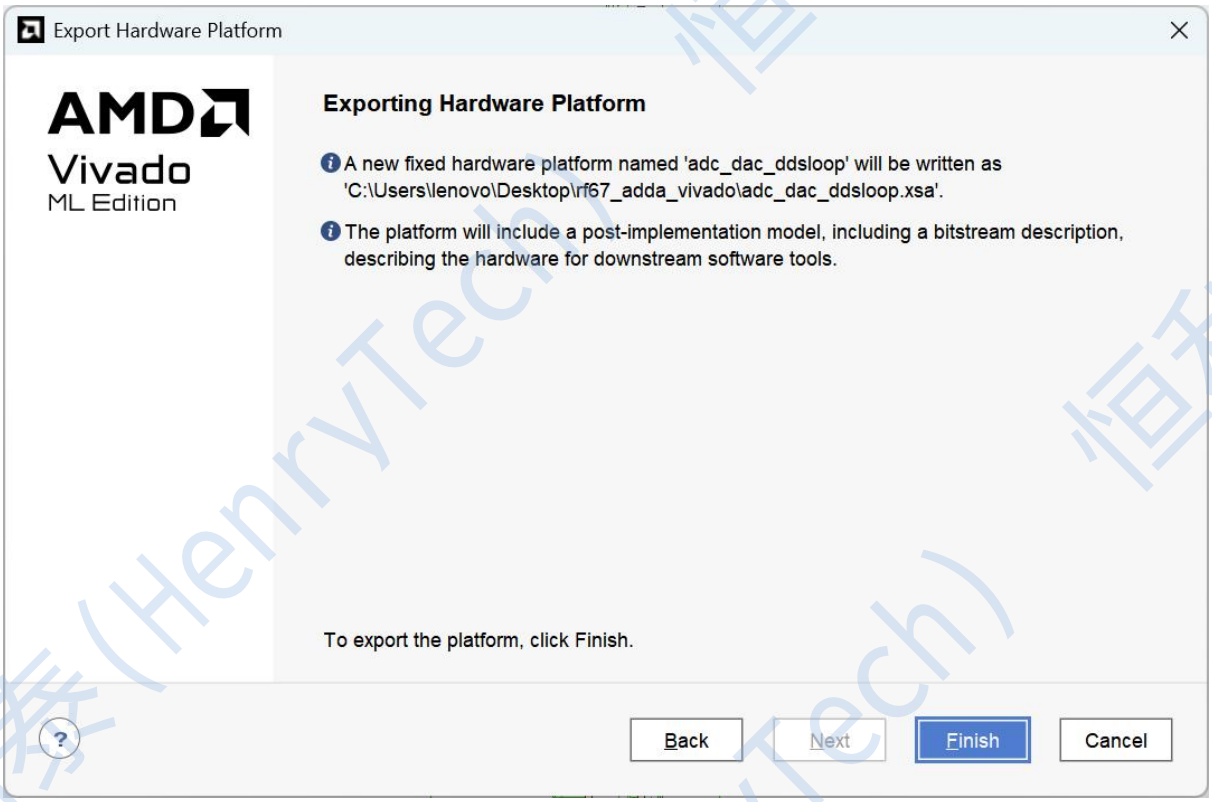
注意一定要选择导出包含bit文件，点击Next：



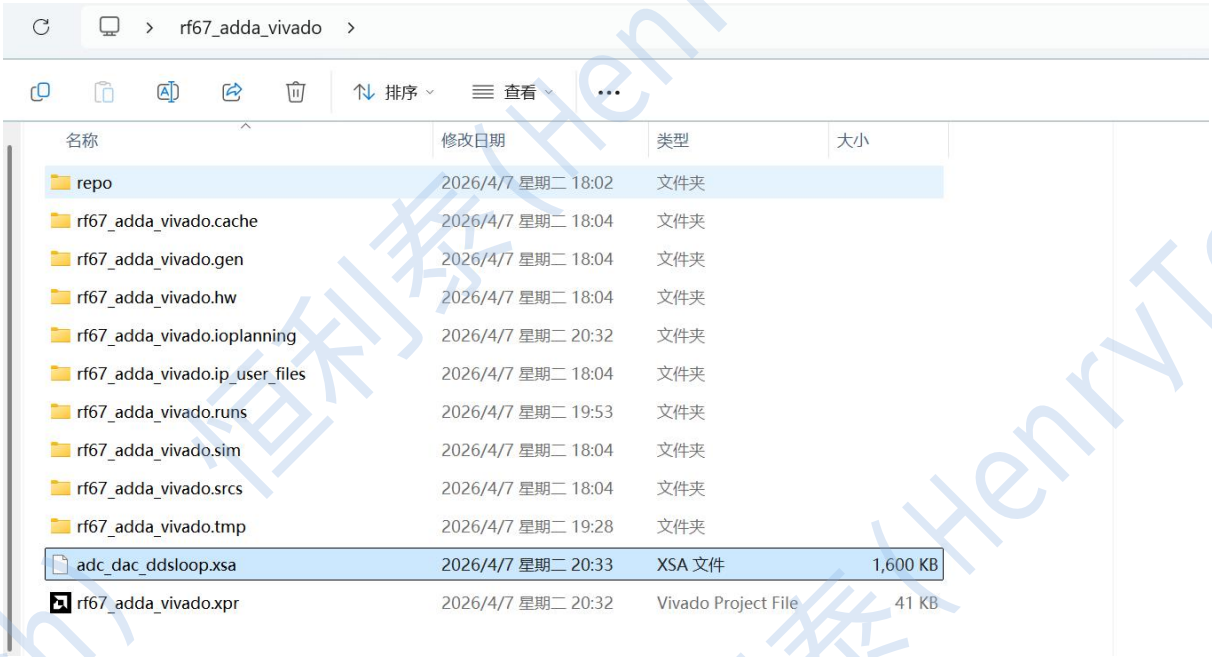
点击Next：



最后点击Finish完成导出：



最终如下文件就是我们的硬件文件了：



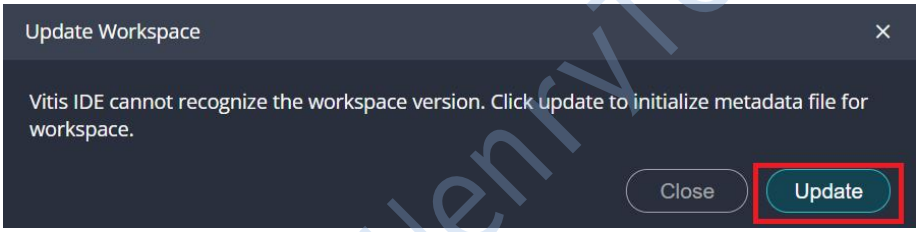
七、Vitis工程建立

我们上一节讲了如何导出xsa文件，这一节我们讲解如何建立vitis工程。我们首先在桌面或者硬盘根目录新建一个叫作rf67dr_vitis的文件夹，我们将上一节导出的xsa文件复制进去：

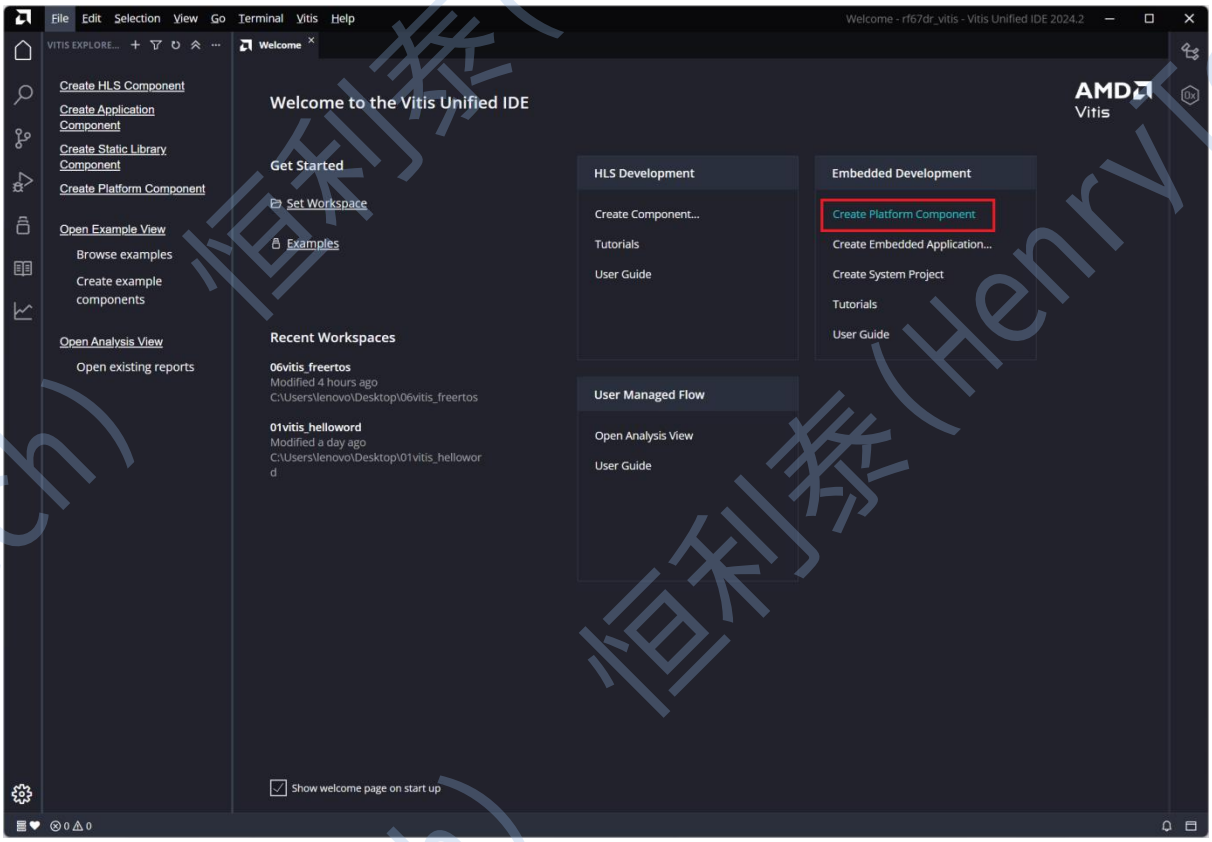
接着我们找到2024.2版本的vitis启动：

Vitis 2024.2

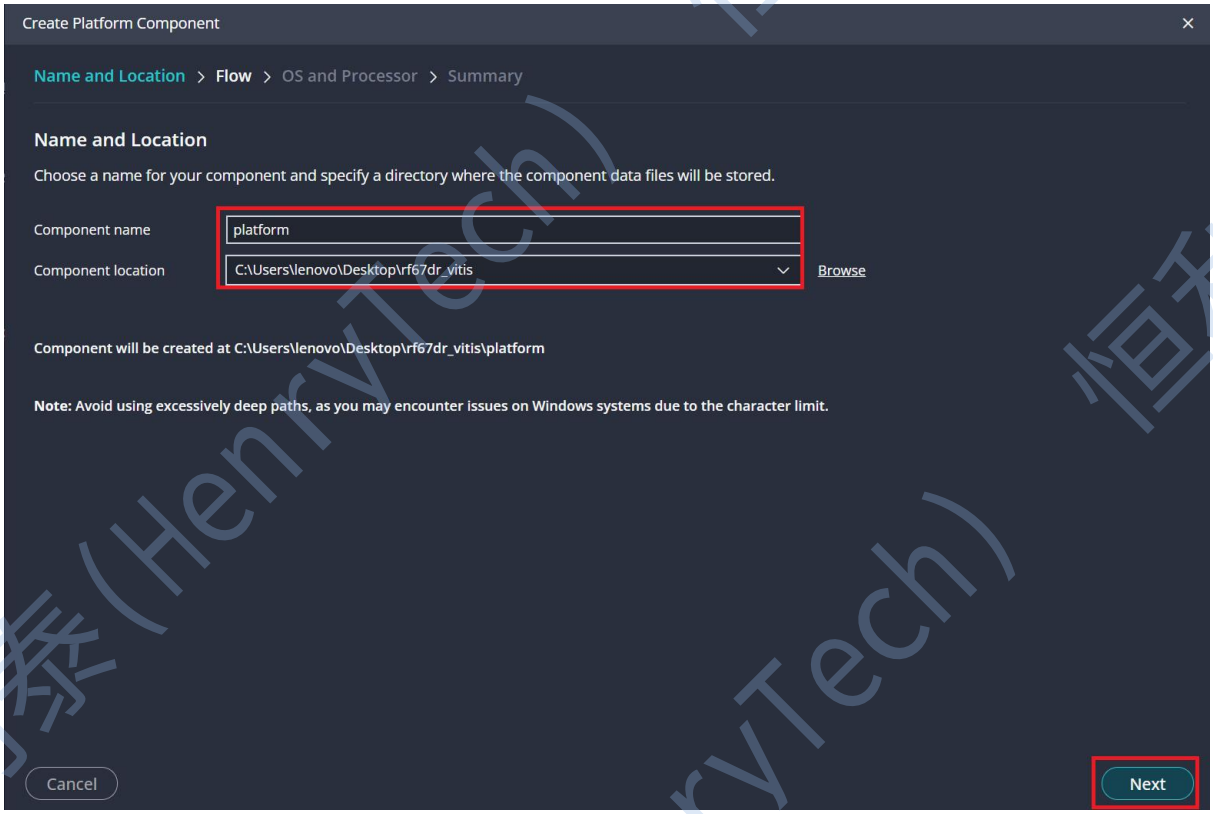
启动后，主界面选择Set Workspace的目录为刚才新建的文件夹。然后弹出Update workspace更新目录，点击Update：



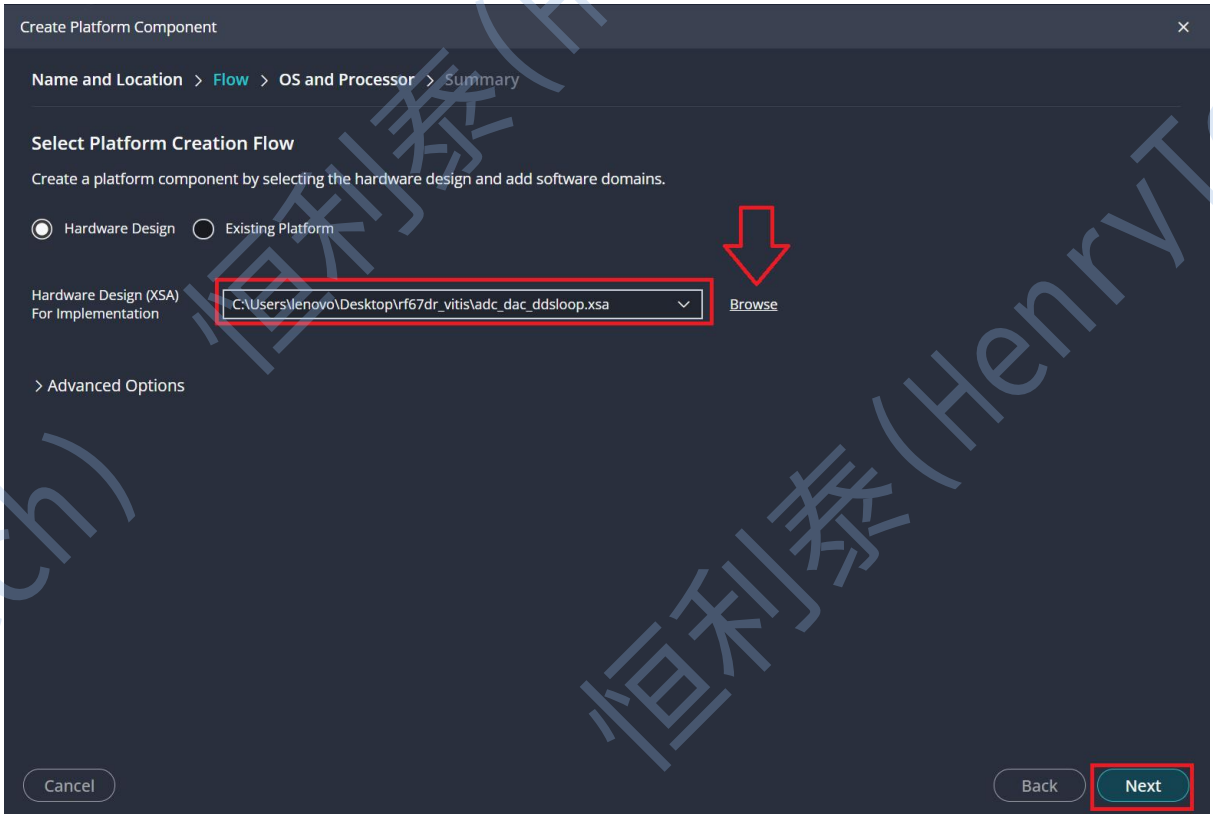
主界面点击Create Platform Component创建硬件平台：



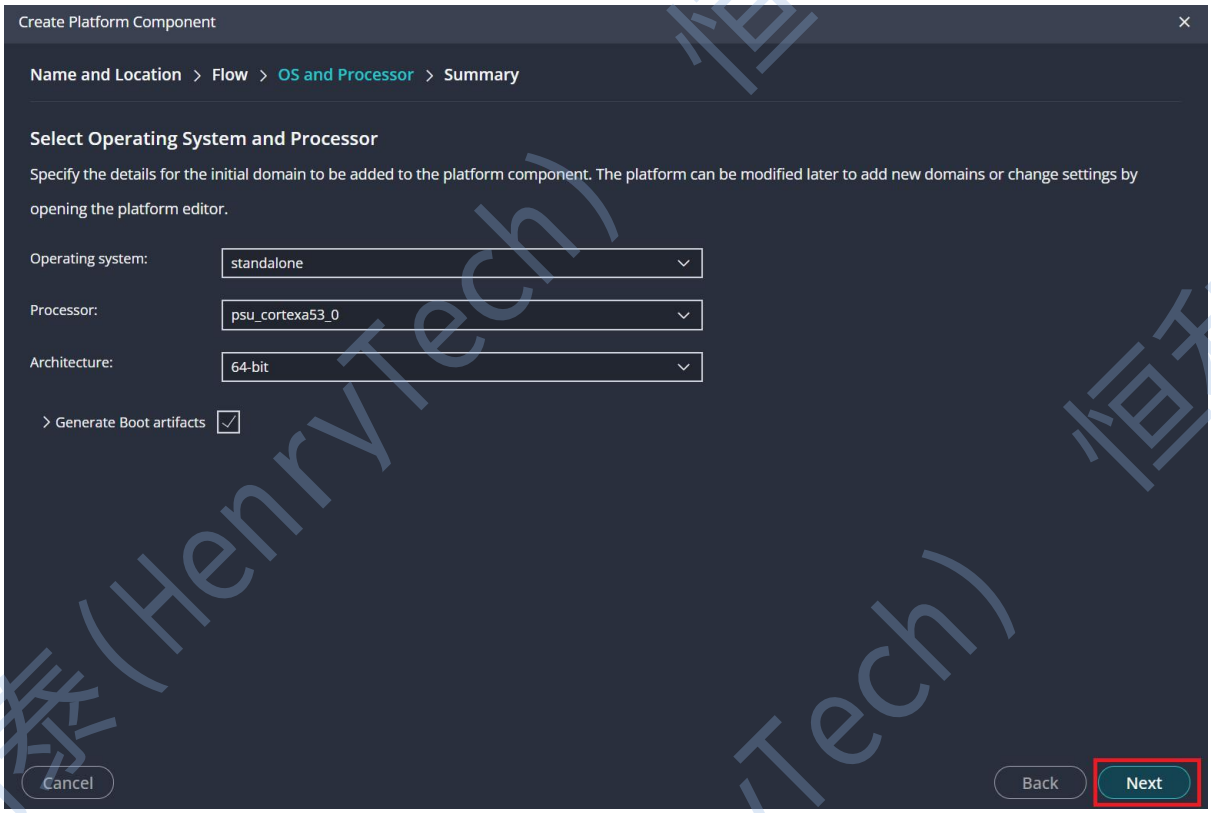
第一页，名字和路径默认在当前workspace中，直接点击Next：



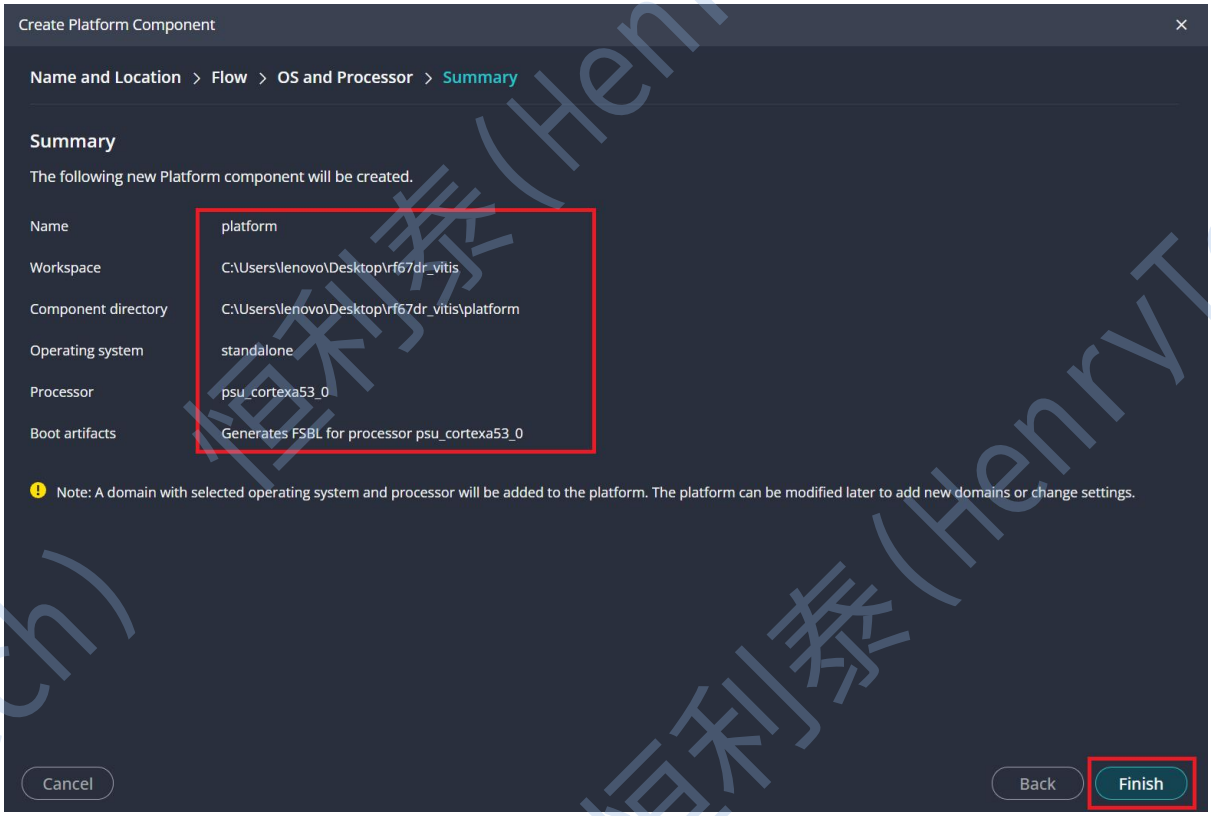
Flow这一页，xsa右侧的Browse浏览，选择当前workspace中的xsa文件（文件夹创建的时候复制的xsa文件）进来，点击Next：



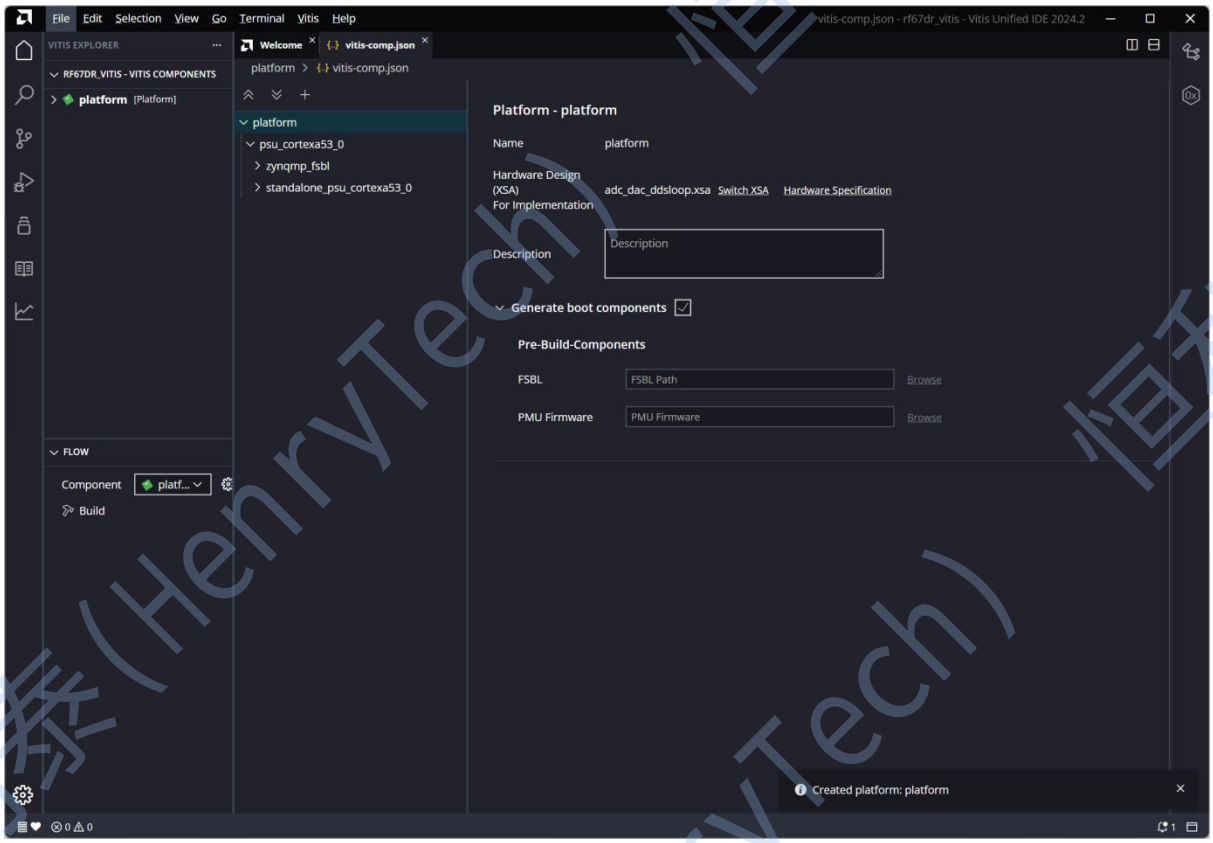
OS and Processor操作系统和处理器页面，保持默认，点击Next：



最后一页Summary概况，核对好配置信息点击Finish:



等待platform创建完成，工程如下：

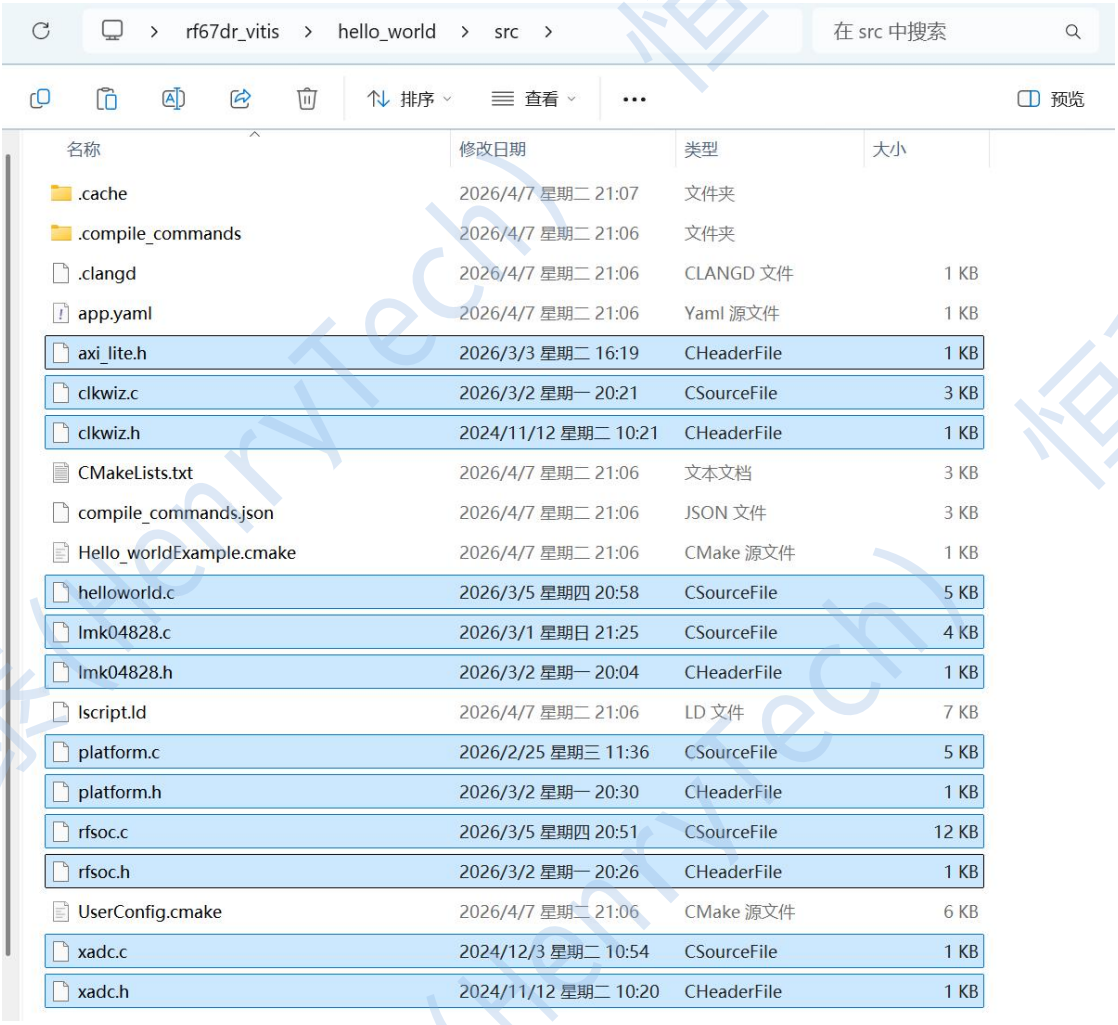


八、导入helloworld模板

参考教程《11PS处理器配置及helloword打印》文档，这里不再赘述。所有步骤均一致，没有特殊更改。

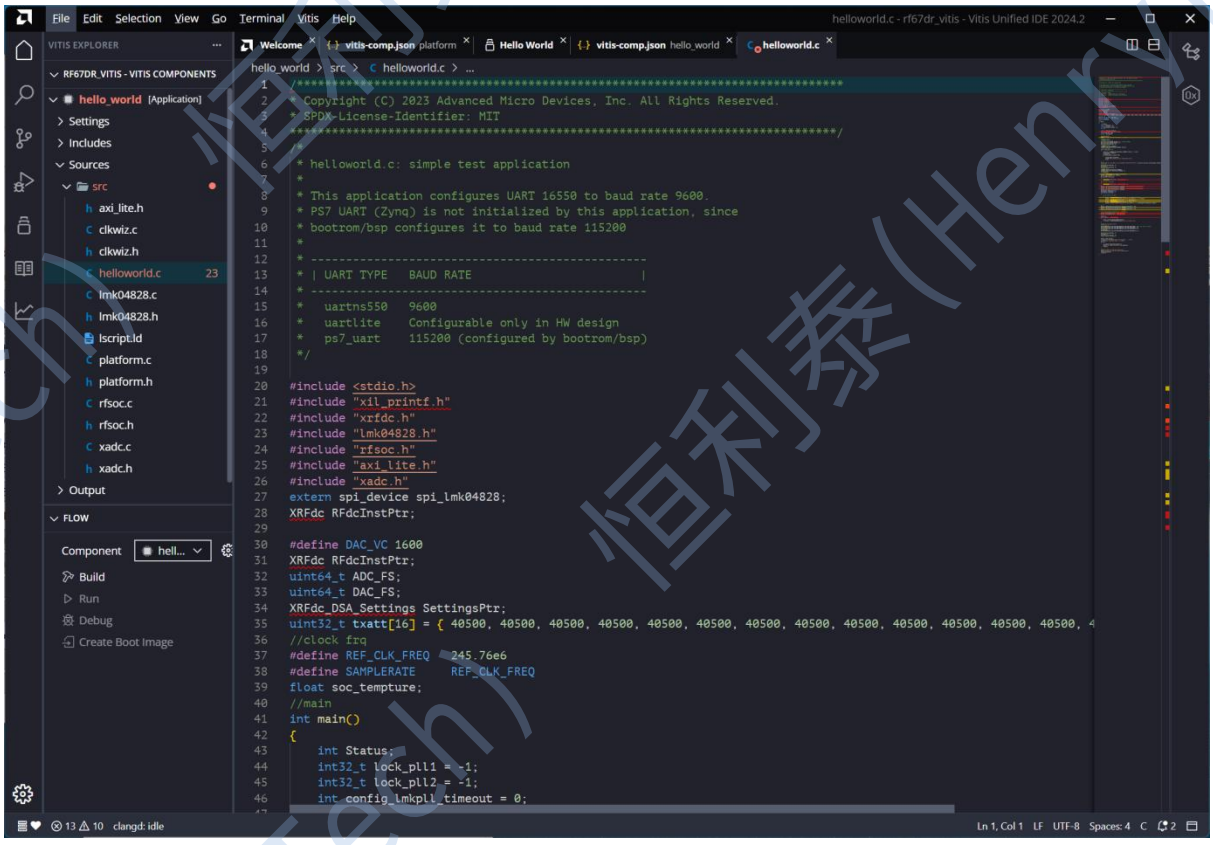
九、拷贝源码替换helloworld目录代码

按照helloworld模板导入后的工程，我们需要将源代码添加进来。Vitis源代码位于《RF67DR_Lite开发板资料\04例程demo\rf_adda_loop\vitis_src.zip》，解压后，复制到当前工程中的《rf67dr_vitis\hello_world\src》目录中，提示替换，全部选择替换。代码复制后，工程目录如下：



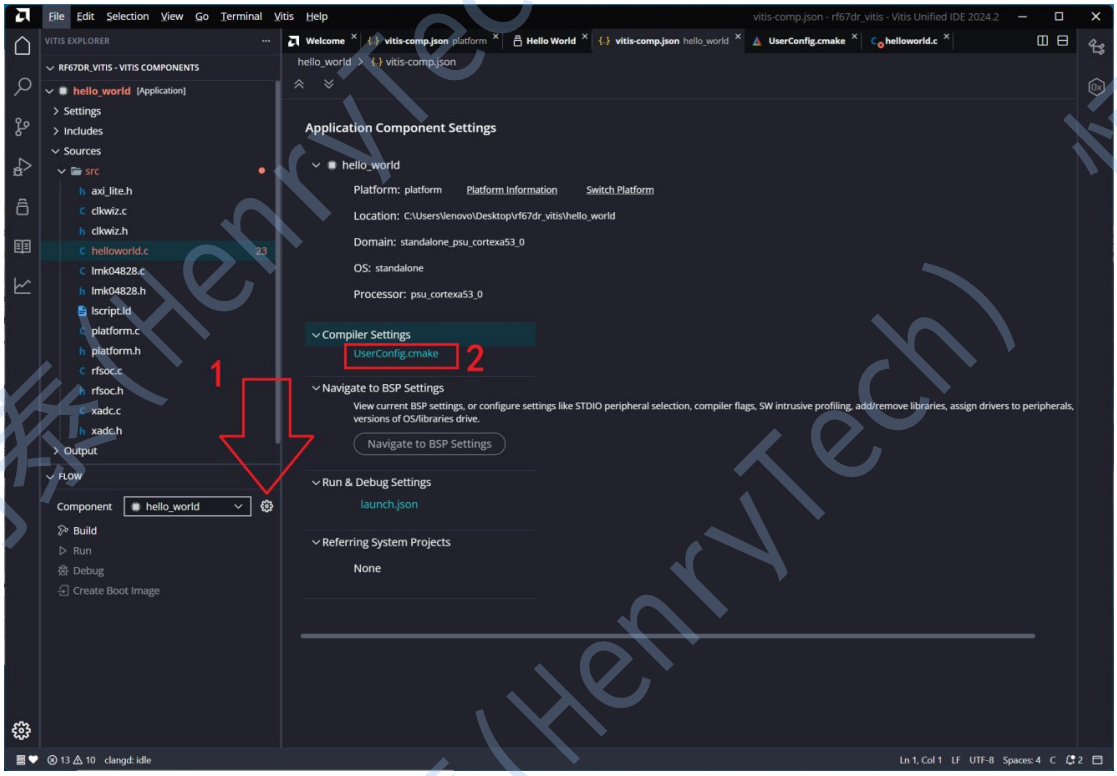
名称	修改日期	类型	大小
.cache	2026/4/7 星期二 21:07	文件夹	
.compile_commands	2026/4/7 星期二 21:06	文件夹	
.clangd	2026/4/7 星期二 21:06	CLANGD 文件	1 KB
app.yaml	2026/4/7 星期二 21:06	Yaml 源文件	1 KB
axi_lite.h	2026/3/3 星期二 16:19	CHeaderFile	1 KB
clkwiz.c	2026/3/2 星期一 20:21	CSourceFile	3 KB
clkwiz.h	2024/11/12 星期二 10:21	CHeaderFile	1 KB
CMakeLists.txt	2026/4/7 星期二 21:06	文本文档	3 KB
compile_commands.json	2026/4/7 星期二 21:06	JSON 文件	3 KB
Hello_worldExample.cmake	2026/4/7 星期二 21:06	CMake 源文件	1 KB
helloworld.c	2026/3/5 星期四 20:58	CSourceFile	5 KB
lmk04828.c	2026/3/1 星期日 21:25	CSourceFile	4 KB
lmk04828.h	2026/3/2 星期一 20:04	CHeaderFile	1 KB
lscript.ld	2026/4/7 星期二 21:06	LD 文件	7 KB
platform.c	2026/2/25 星期三 11:36	CSourceFile	5 KB
platform.h	2026/3/2 星期一 20:30	CHeaderFile	1 KB
rfsoc.c	2026/3/5 星期四 20:51	CSourceFile	12 KB
rfsoc.h	2026/3/2 星期一 20:26	CHeaderFile	1 KB
UserConfig.cmake	2026/4/7 星期二 21:06	CMake 源文件	6 KB
xadc.c	2024/12/3 星期二 10:54	CSourceFile	1 KB
xadc.h	2024/11/12 星期二 10:20	CHeaderFile	1 KB

Vitis中目录如下，此时代码无法正确编译，由于vitis编译还有设置需要做：

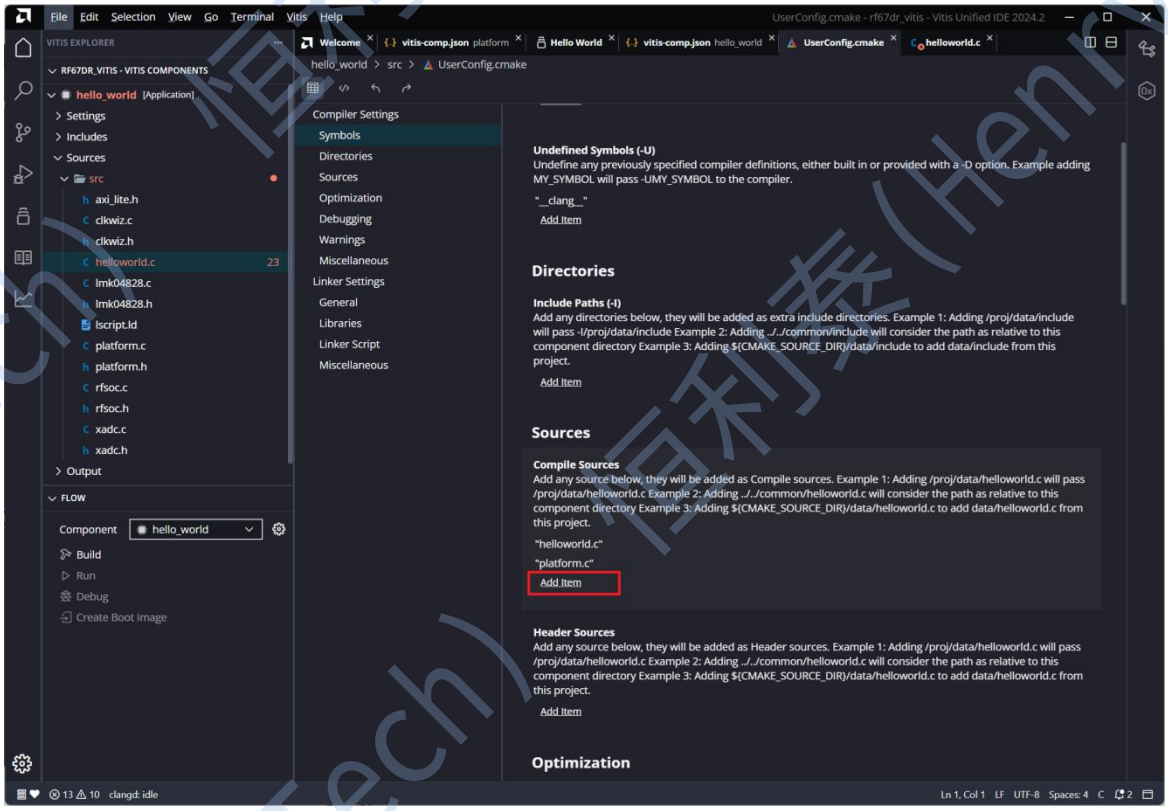


十、vitis编译设置

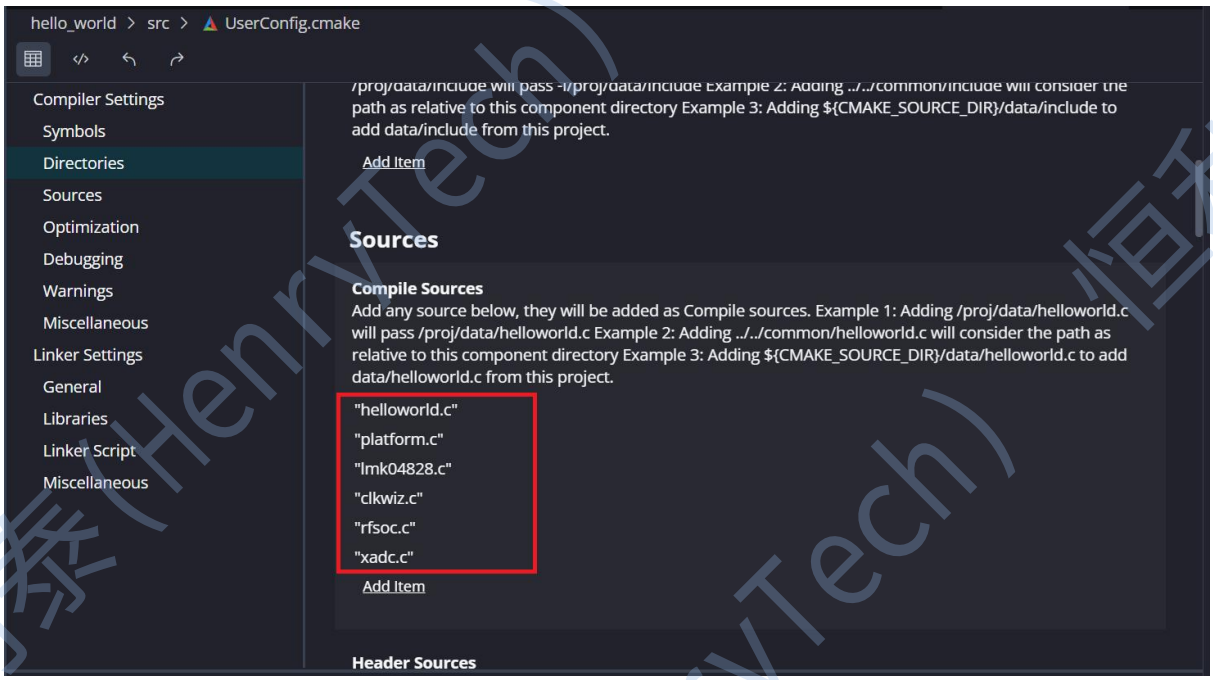
我们在FLOW下方的Component右侧点击设置图标进入编译设置。打开主界面中找到Compile settings设置，点击UserConfig.cmake进入设置：



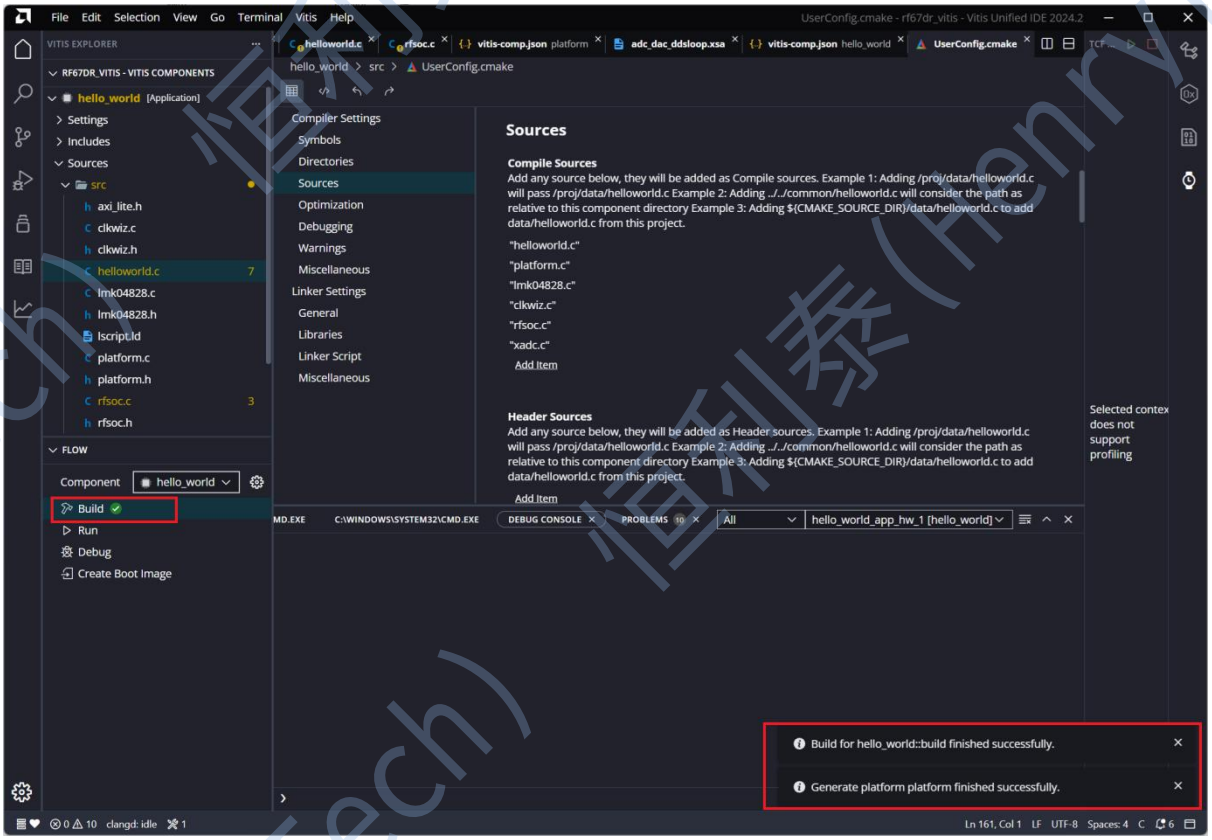
设置中，找到Sources->Compile Sources，我们点击Add Item添加文件。



文件添加如下，所使用到的每一项。c文件都要添加，为了方便后期路径维护，建议只保留双引号内的文件名即可：



与低版本Vitis不一样，低版本只需要替换复制源代码到src目录，就可以直接编译。但是2023.2以上的版本就不行，必须做此设置。另外Header Sources头文件设置中，可以不添加。做完代码配置后，我们点击FLOW中的Build进行编译，编译成功，Build右侧显示绿色的√，并且右下方提示编译成功：



十一、vitis代码说明

主程序helloworld.c代码如下：

```
/******************************************************************************
* Copyright (C) 2023 Advanced Micro Devices, Inc. All Rights Reserved.
* SPDX-License-Identifier: MIT
******************************************************************************/
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *  uartns550   9600
 *  uartlite    Configurable only in HW design
 *  ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "xil_printf.h"
#include "xrfdc.h"
#include "lmk04828.h"
#include "rfsoc.h"
#include "axi_lite.h"
#include "xadc.h"
extern spi_device spi_lmk04828;
XRFdc RFdcInstPtr;
```

```
#define DAC_VC 1600
XRFdc RFdcInstPtr;
uint64_t ADC_FS;
uint64_t DAC_FS;
XRFdc_DSA_Settings SettingsPtr;
uint32_t txatt[16] = { 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000, 40000,
40000 };
//clock frq
#define REF_CLK_FREQ   245.76e6
#define SAMPLERATE    REF_CLK_FREQ
float soc_tempture;
//main
int main()
{
    int Status;
    int32_t lock_pll1 = -1;
    int32_t lock_pll2 = -1;
    int config_lmkpll_timeout = 0;
    int i,j;
```

```
uint32_t Tile_Id, Block_Id;
XRFdc_Mixer_Settings Mixer_Settings;
double Freq_set = 600;
uint32_t nco_nyquist;
```

```
Platform_Init();
print("RF-67DR_Lite RevA init start……\n\r");
```

```
Lmk04828_reset();
//lmk04828 ref clk set
Gpio_Value_Set(LMK_REF_SELECT0, 1); //CLKIN1 EXTERNAL
Gpio_Value_Set(LMK_REF_SELECT1, 0);
Gpio_Value_Set(LMK_SYNC, 0);
//lmk04828 init
Lmk04828_Init(spi_lmk04828);
Lmk04828_Update_VC_DAC(spi_lmk04828, DAC_VC);
//wait pll locked
while (lock_pll2 != 1)
{
   lock_pll2 = Lmk04828_Spi_Read(&spi_lmk04828, 0X183) >> 1 & 0x1;
   config_lmkpll_timeout++;
   ms_delay(1);
   if(config_lmkpll_timeout>=100)
   {
      config_lmkpll_timeout=0;
      printf("LMK pllconfig faild!,timeout=100.\r\n");
      break;
   }
}
printf("LMK pll1_lock=0x%x, pll2_lock=0x%x,try=%d,init Done\r\n", lock_pll1,lock_pll2,config_lmkpll_timeout);
//LMK SYNC
Gpio_Value_Set(LMK_SYNC, 1);
ms_delay(1);
Gpio_Value_Set(LMK_SYNC, 0);
//update sysref
Lmk04828_Update_Sysref(&spi_lmk04828);
Lmk04828_Set_Sysref_Shots(&spi_lmk04828, 0);
ms_delay(100);
   //RF Init
Status = AXI_REG_READ(0XFF5E0000, 0X00000200U);
printf("boot mode=%x\r\n", Status);
RF_init(&RFdcInstPtr, 0);
for (i = 0; i < 2; i++)
{
   XRFdc_Reset(&RFdcInstPtr, XRFDC_DAC_TILE, i);
}
for (i = 0; i < 3; i++)
{
   XRFdc_Reset(&RFdcInstPtr, XRFDC_ADC_TILE, i);
}
Status = AXI_REG_READ(XPAR_XRFDC_0_BASEADDR, XRFDC_ADC_PATHS_ENABLED_OFFSET);
printf("ADC enable status=%x\r\n", Status);
Status = AXI_REG_READ(XPAR_XRFDC_0_BASEADDR, XRFDC_DAC_PATHS_ENABLED_OFFSET);
printf("DAC enable status=%x\r\n", Status);
```

恒利泰(HenryTech)

```
    SettingsPtr.Attenuation = 0;
    SettingsPtr.DisableRTS = 1;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 4; j++)
        {
            Status = XRFdc_SetDACVOP(&RFdcInstPtr, i, j, txatt[i * 4 + j]);
            printf("tile=%d,block=%d,XRFdc_SetDACVOP,Status=%d\r\n", i, j, Status);
            Status = XRFdc_SetDSA(&RFdcInstPtr, i, j, &SettingsPtr);
            Status = XRFdc_GetDSA(&RFdcInstPtr, i, j, &SettingsPtr);
            Status = XRFdc_SetDither(&RFdcInstPtr, i, j, 1);
            printf("tile%d,block%d,Attenuation %f,Status=%d\r\n", i, j,SettingsPtr.Attenuation, Status);
        }
    }
    ADC_FS = RFdcInstPtr.ADC_Tile[0].PLL_Settings.SampleRate * 1e9;
    printf("ADC Samplerate is%ld Hz\r\n", ADC_FS);
    DAC_FS = RFdcInstPtr.DAC_Tile[0].PLL_Settings.SampleRate * 1e9;
    printf("DAC Samplerate is%ld Hz\r\n", DAC_FS);
//DAC OUT
    if (Freq_set * 1e6 < DAC_FS / 2)
    {
        nco_nyquist = 1;
    }
    else
    {
        nco_nyquist = 2;
        Freq_set = -Freq_set;
    }
    for (Tile_Id = 0; Tile_Id < 2; Tile_Id++)
    {
        for (Block_Id = 0; Block_Id < 4; Block_Id++)
        {
            Status = XRFdc_SetNyquistZone(&RFdcInstPtr, XRFDC_DAC_TILE, Tile_Id,Block_Id, nco_nyquist);
            if (Status != XST_SUCCESS)printf("Tile_Id=%d,Block_Id=%d,XRFdc_SetNyquistZone fail\r\n",Tile_Id, Block_Id);
```

```
            RF_ChangeNCOFreq(&RFdcInstPtr, XRFDC_DAC_TILE, Tile_Id,Block_Id, Freq_set);
```

```
            Status = XRFdc_GetMixerSettings(&RFdcInstPtr, XRFDC_DAC_TILE, Tile_Id,Block_Id, &Mixer_Settings);
            if (Status != XST_SUCCESS)printf("Tile_Id=%d,Block_Id=%d,XRFdc_GetMixerSettings fail\r\n",Tile_Id, Block_Id);
            printf("DAC NCO frequency of Tile%d chk%d is%f\r\n",Tile_Id, Block_Id, Mixer_Settings.Freq);
        }
    }
    //ADC IN
    if (Freq_set * 1e6 < (ADC_FS / 2))
    {
        nco_nyquist = 1;
        Freq_set = -Freq_set;
    }
    else
    {
        nco_nyquist = 2;
        Freq_set = Freq_set;
    }
    for (Tile_Id = 0; Tile_Id < 2; Tile_Id++)
```

```
    {
        for (Block_Id = 0; Block_Id < 4; Block_Id++)
        {
            Status = XRFdc_SetNyquistZone(&RFdcInstPtr, XRFDC_ADC_TILE, Tile_Id,Block_Id, nco_nyquist);
            if (Status != XST_SUCCESS)printf("Tile_Id=%d,Block_Id=%d,XRFdc_SetNyquistZone fail\r\n",Tile_Id, Block_Id);
            RF_ChangeNCOFreq(&RFdcInstPtr, XRFDC_ADC_TILE, Tile_Id,Block_Id, Freq_set);
            Status = XRFdc_GetMixerSettings(&RFdcInstPtr, XRFDC_ADC_TILE,Tile_Id, Block_Id, &Mixer_Settings);
            if (Status != XST_SUCCESS)printf("Tile_Id=%d,Block_Id=%d,XRFdc_GetMixerSettings fail\r\n",Tile_Id, Block_Id);
            printf("ADC NCO frequency of Tile%d ch%d is%f\r\n",Tile_Id, Block_Id, Mixer_Settings.Freq);
        }
    }
```

```
    RF_ChangeSampleConfg(RFdcInstPtr, SAMPLERATE);
    RF_Qmc_Config(RFdcInstPtr, XRFDC_ADC_TILE);
    RF_Qmc_Config(RFdcInstPtr, XRFDC_DAC_TILE);

    Status=-1;
    while (Status != XRFDC_SUCCESS)
    {
        Status = RF_Multi_Title_SYNC(RFdcInstPtr);
        if (Status != XRFDC_SUCCESS)printf("RF init failed!\r\n");
    }

    printf("RF init done.\r\n");
    //DDS START UP
    AXI_REG_WRITE(AXI_LITE_CTRL_BASE_ADDR,AXI_LITE_CTRL_INIT_READY, 0); dds phase increment
    AXI_REG_WRITE(AXI_LITE_CTRL_BASE_ADDR,AXI_LITE_CTRL_PHASE_INC1, 64);
    AXI_REG_WRITE(AXI_LITE_CTRL_BASE_ADDR,AXI_LITE_CTRL_DDS_RESET_N, 0);
    ms_delay(10);
    AXI_REG_WRITE(AXI_LITE_CTRL_BASE_ADDR,AXI_LITE_CTRL_DDS_RESET_N, 1);
```

```
    Gpio_Value_Set(LMK_SYNC, 1);
    ms_delay(1);
    Gpio_Value_Set(LMK_SYNC, 0);
    //xadc tempture read&printf

    Status = Xadc_Init(0);
    if (Status == 0) printf("Xadc Setup Complete……\r\n");
    printf("Start rfsoc xadc read……\r\n");
    while(1)
    {
        soc_tempture=Xadc_Read_Tempture();
        printf("soc tempture=%.2f\r\n",soc_tempture);
        sleep(1);
    }
```

```
    print("RF-67DR_Lite RevA init Done\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

```
}
```

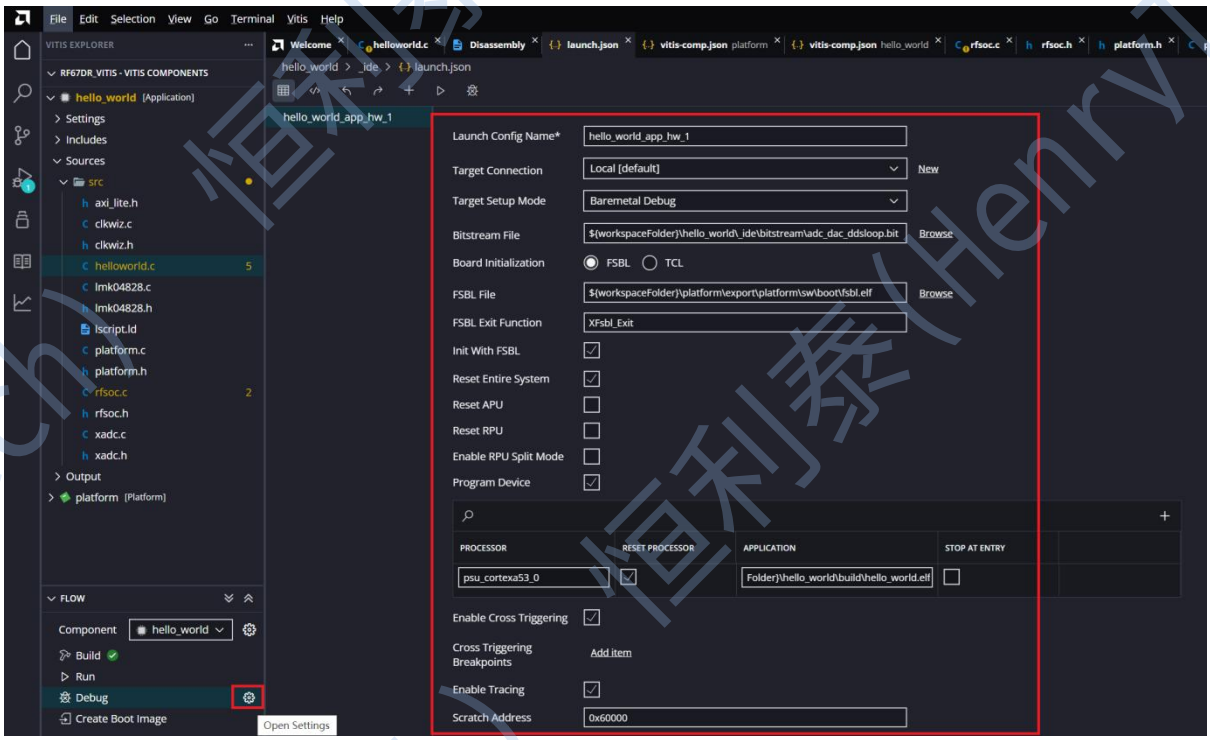
代码中：

- 54-86行，我们对硬件和时钟LMK04828做了初始化，并且配置输出了sysref参考时钟。
- 88-102行，读取打印启动设置模式，初始化了RF的IP的射频初始化，并且读取了使能配置状态。
- 103-120行我们读取了IP核相关信息，以及时钟检测，并且读取了采样率配置。
- 121-144初始化DAC本振频率及nyquist区域，设置DAC混频器；
- 146-167初始化了ADC的本振及nyquist区域，设置ADC混频器；
- 169-171，配置了采样率和QMC；
- 173-178进行了多通道同步操作；
- 182-186行，完成DDS输出相位增量phase_inc设置，这个参数决定DDS输出的正余弦频率，同时，对DDS复位信号做了先设置为0再设置为1的操作，复位DDS相关逻辑。
- 193-201行，我们初始化了XADC，然后循环读取一秒一次打印芯片温度。

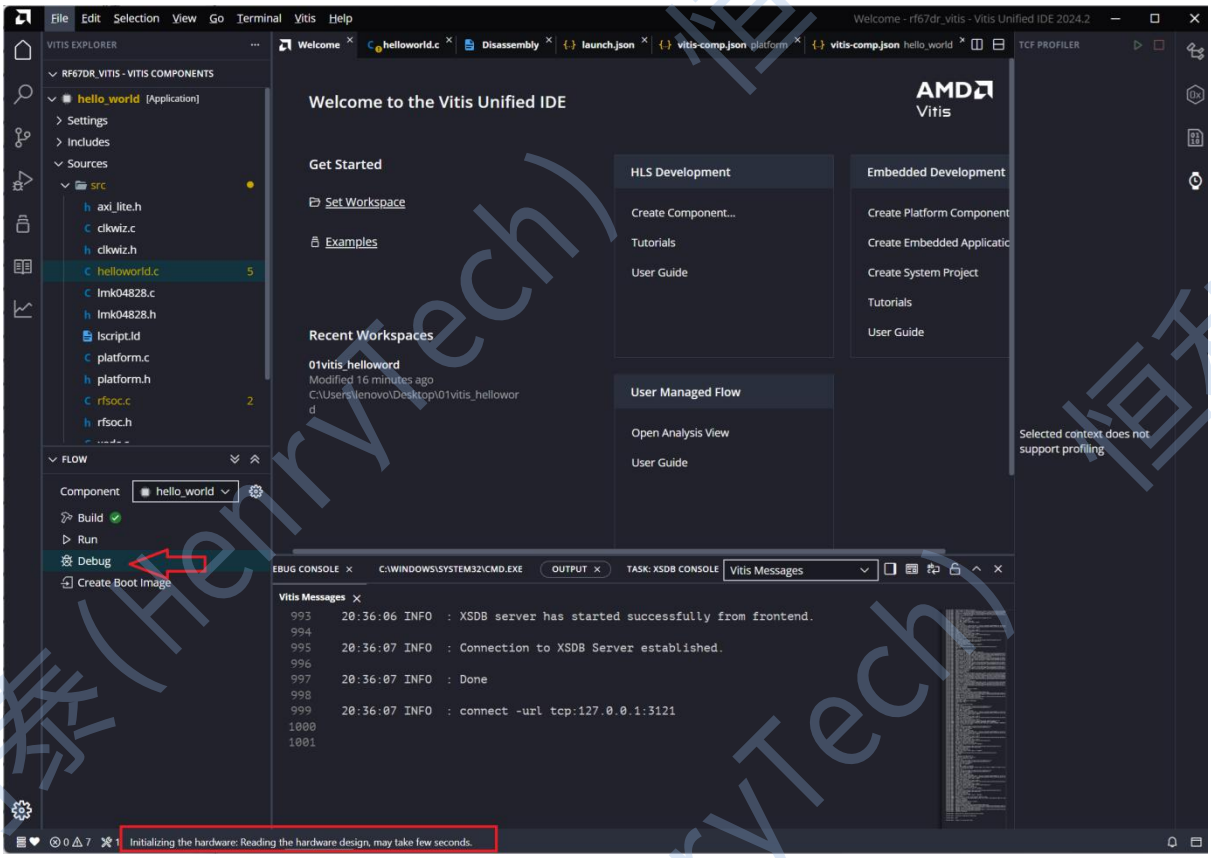
更多的开发，请具体参考《04参考手册》中的PG269手册。就是本文档开头提及的文档，里面包含所有的RF射频核的相关信息，有条件的话，建议全篇阅读此官方文档。

十二、联合VIVADO测试回环

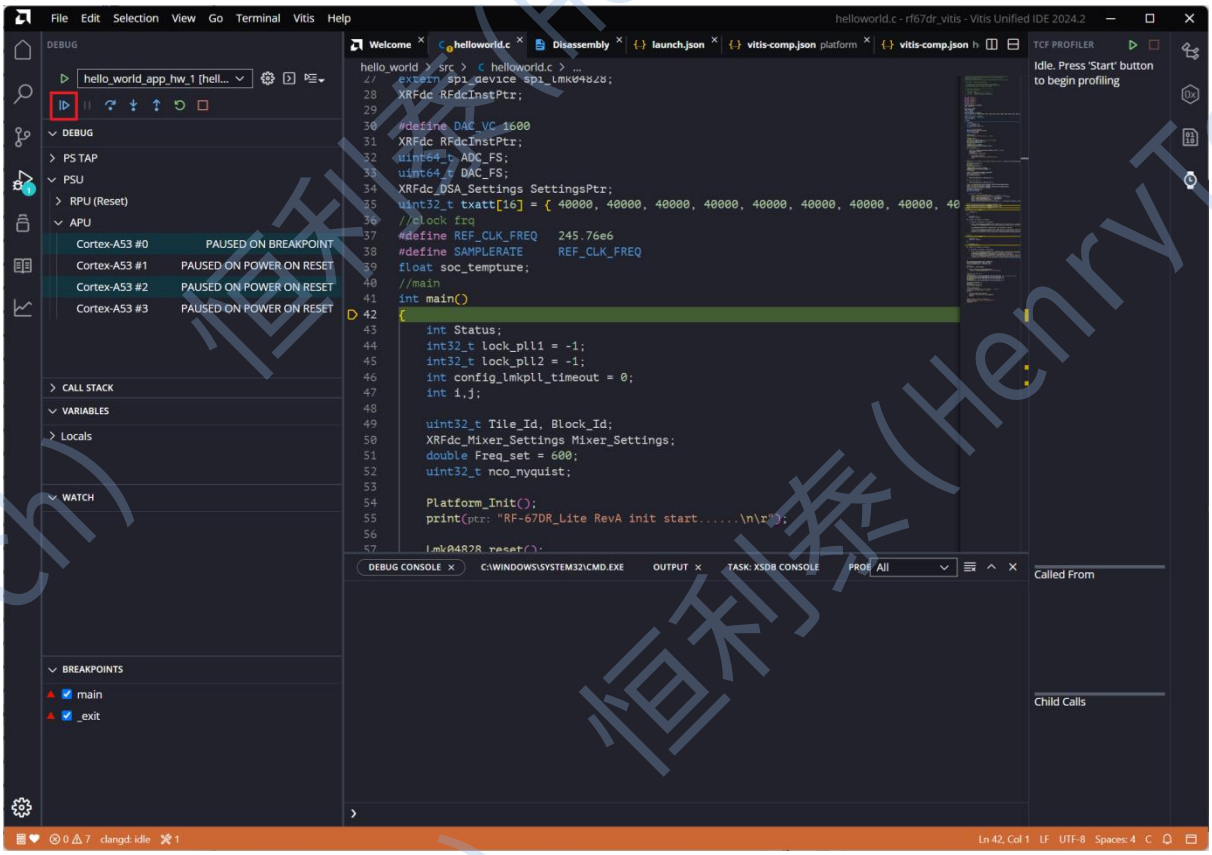
我们Vitis的FLOW点击DEBUG右侧的设置，配置如下，注意勾选项，其余的配置保持默认即可：



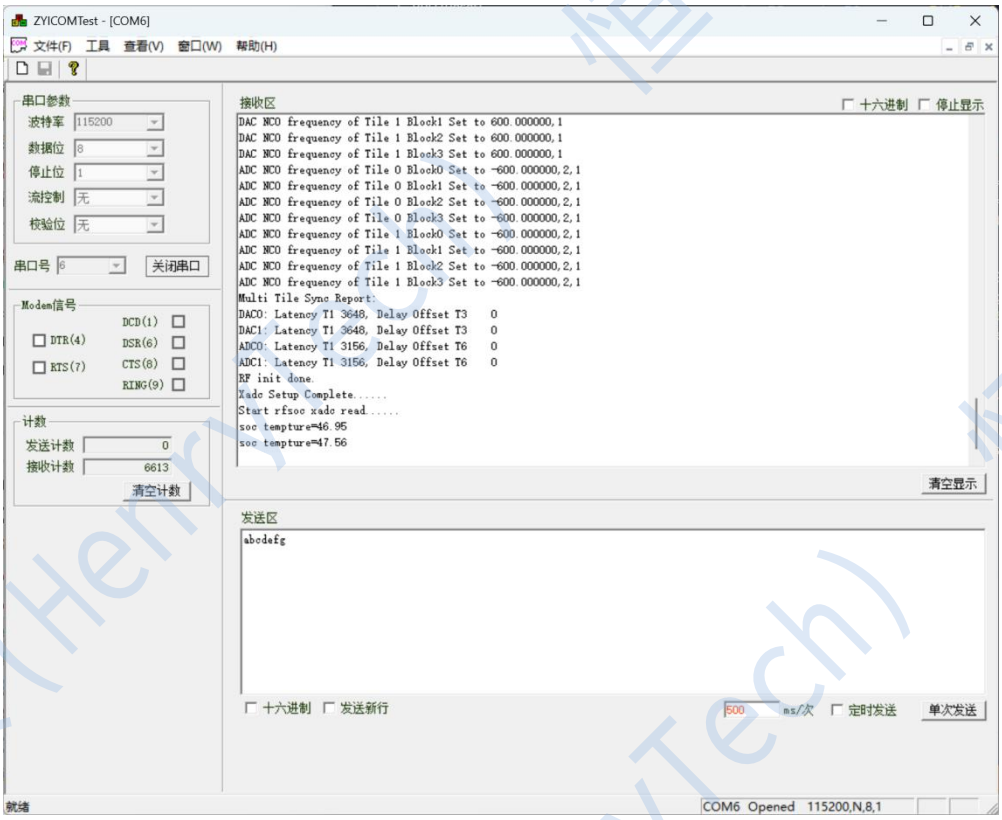
我们点击Debug即可开始调试。等待烧写程序完成：



进入调试界面，打开串口助手连接，点击运行按钮 进行运行：



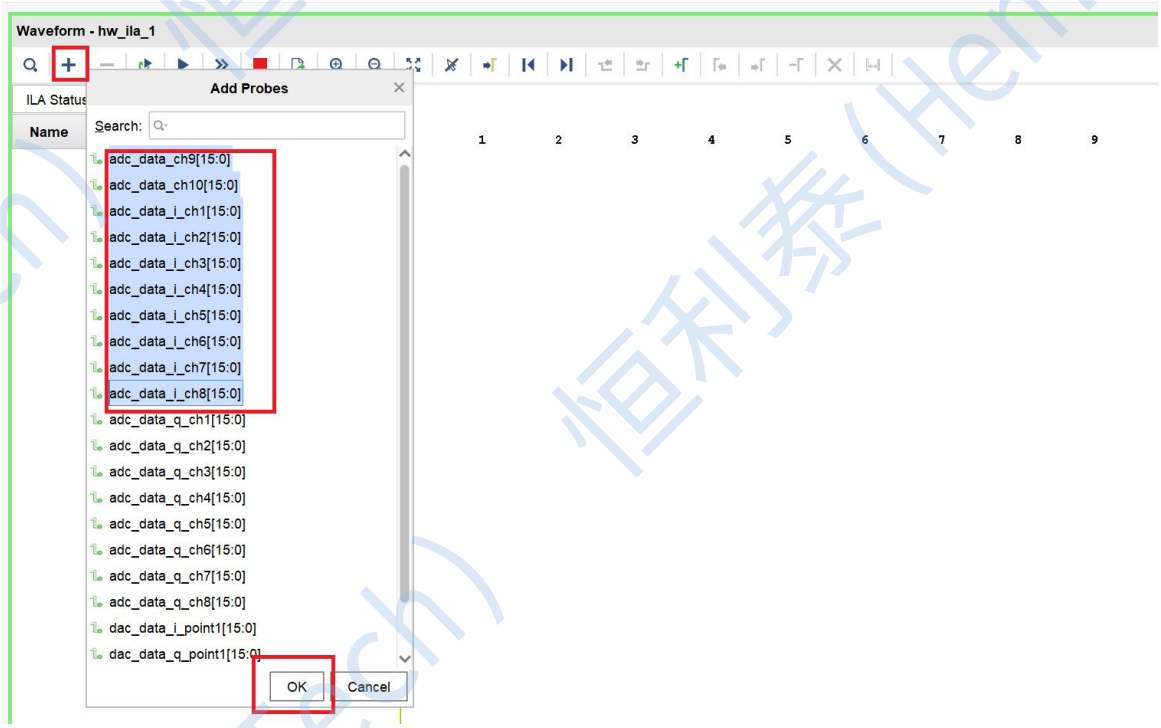
运行后，串口助手打印信息如下，已经完成初始化，并且打印器件XADC温度信息：



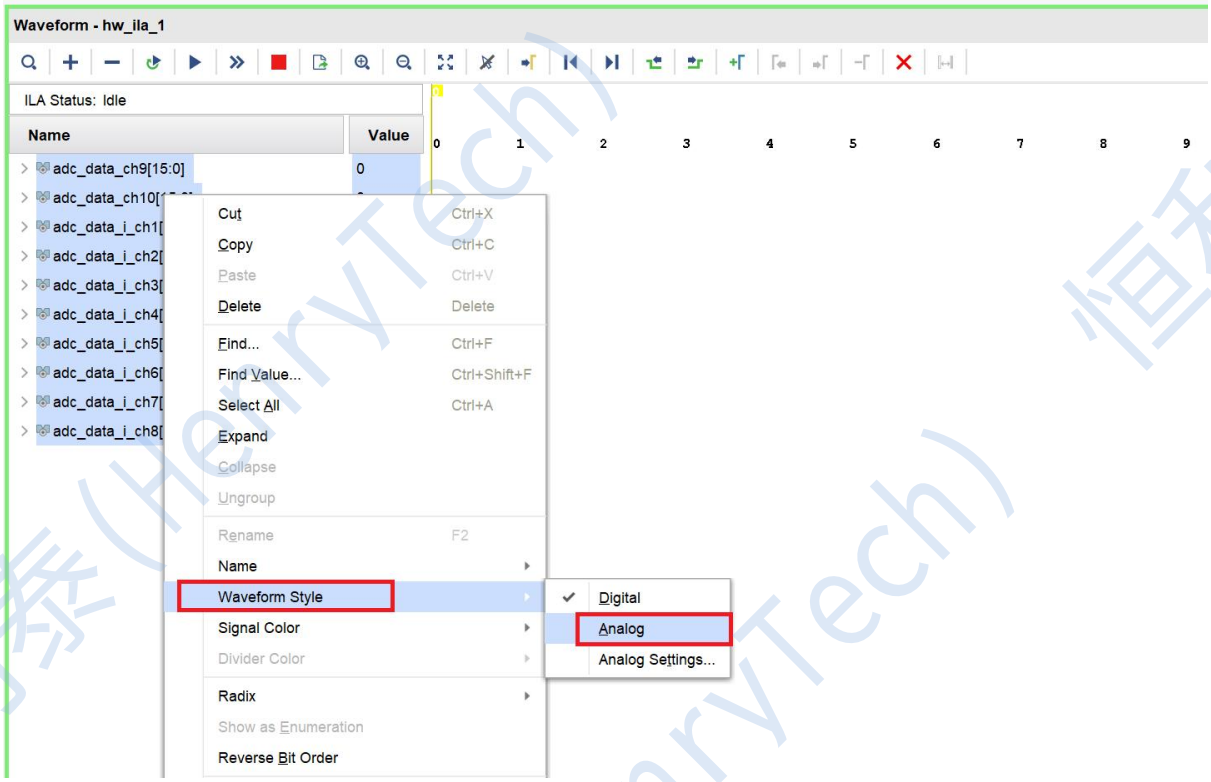
到这一步我们vitis配置嵌入式代码就运行成功。

十三、VIVADO采样抓取验证

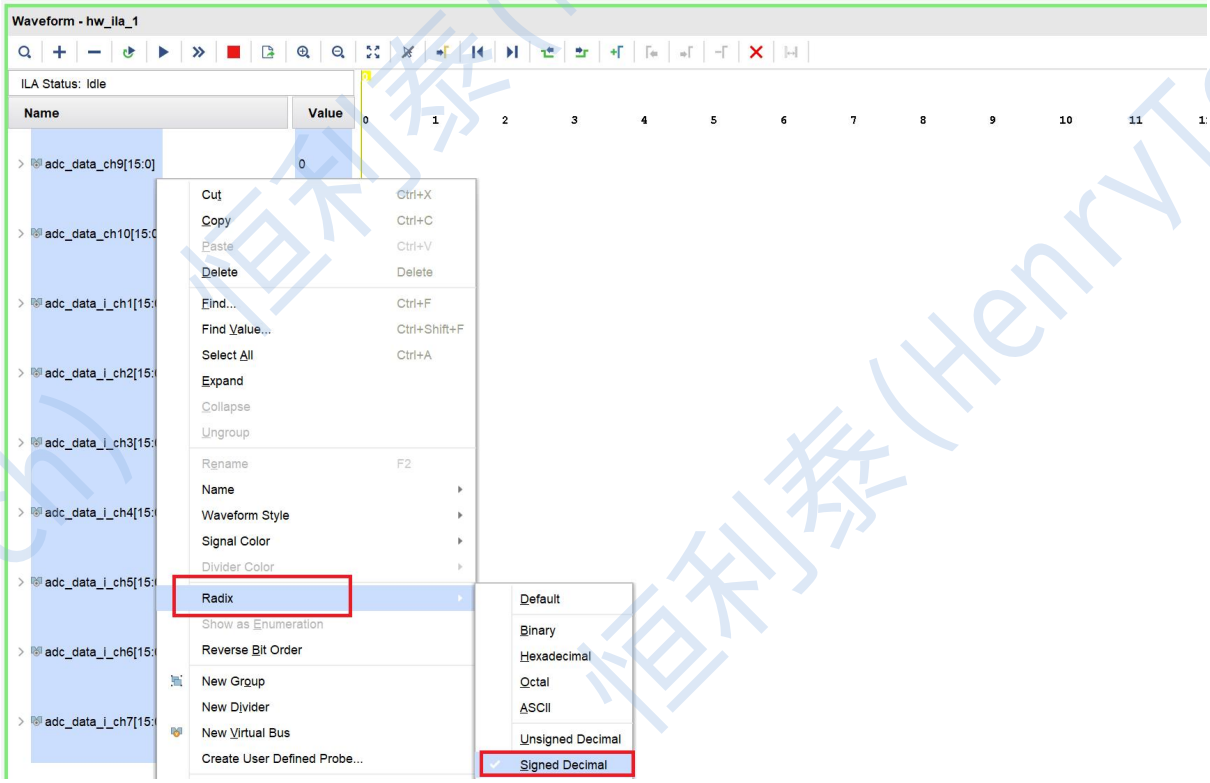
上一节成功运行后，我们打开Vivado2024.2，打开硬件管理，连接开发板，就可以进行在线调试。连接后自动弹出ila界面，如果右侧波形界面有数据，可以ctrl+a全选然后点击一号全部删除，然后点击+号添加adc_data_i_ch的1-8通道，以及adc_data_ch9和10信号进来观察。



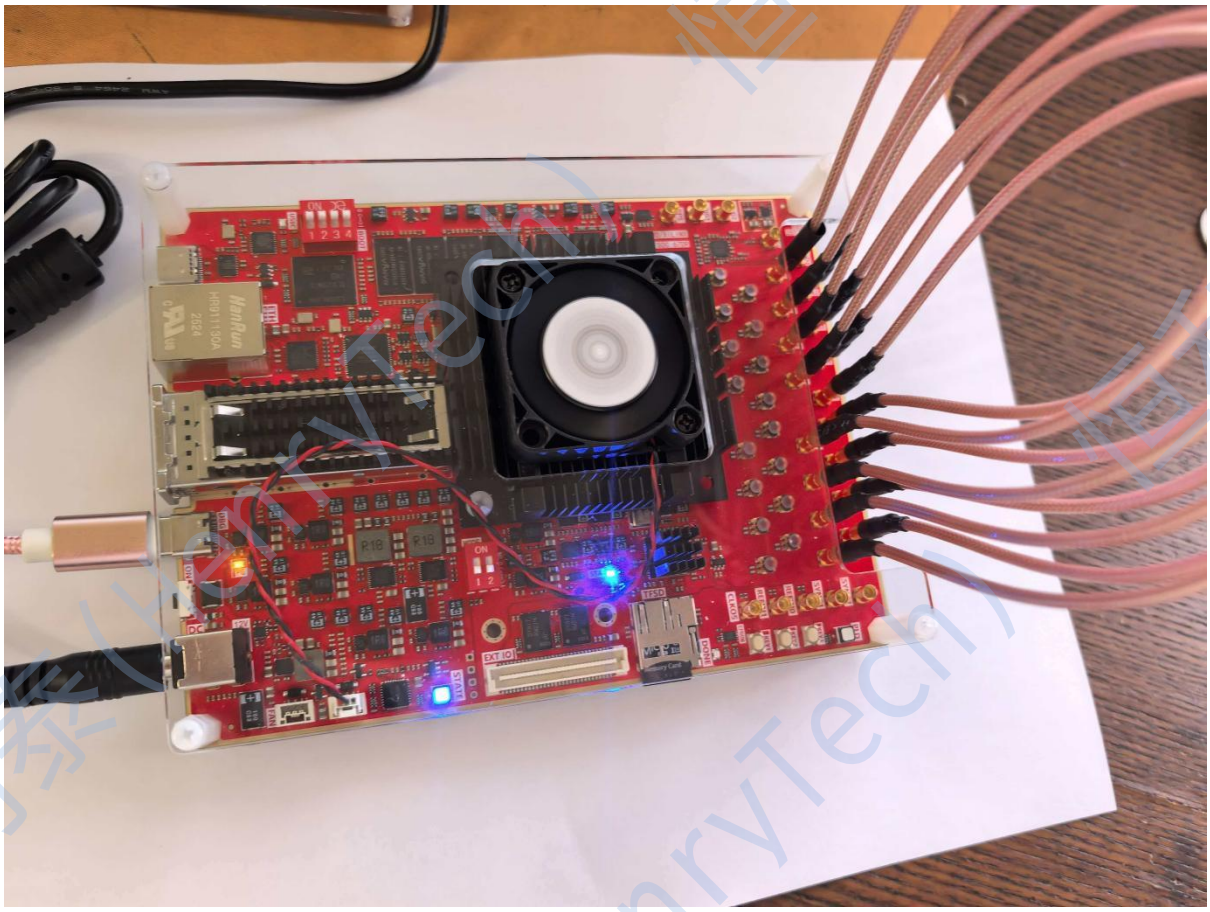
添加进来后，选中信号，右键选择Wavefom Style波形风格，选择Analog表示模拟曲线：



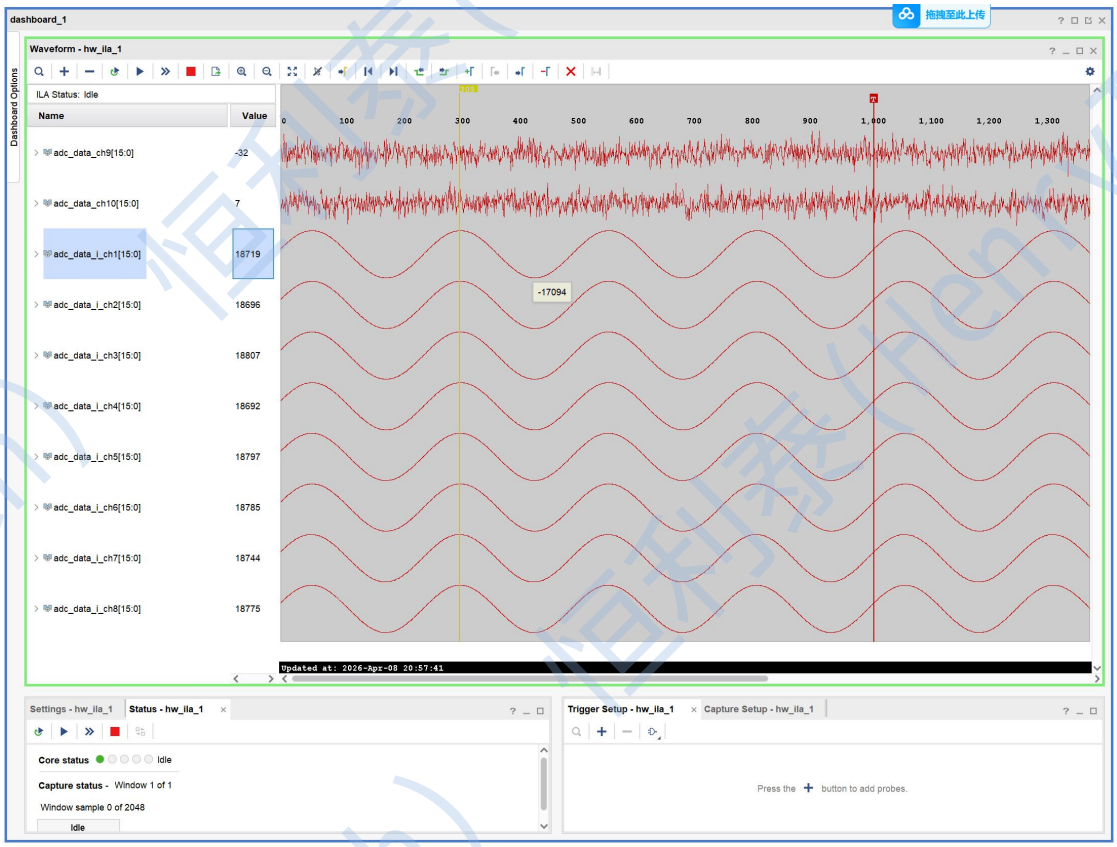
配置好后再配置数据格式，再全选列表信号，右键Radix选择Signed Decimal表示有符号十进制数：



抓取之前，我们将mmcx回环线，依次对应连接A1-D1,A2-D2....以此类推，8通道ADC和DAC连接好，如下图所示：



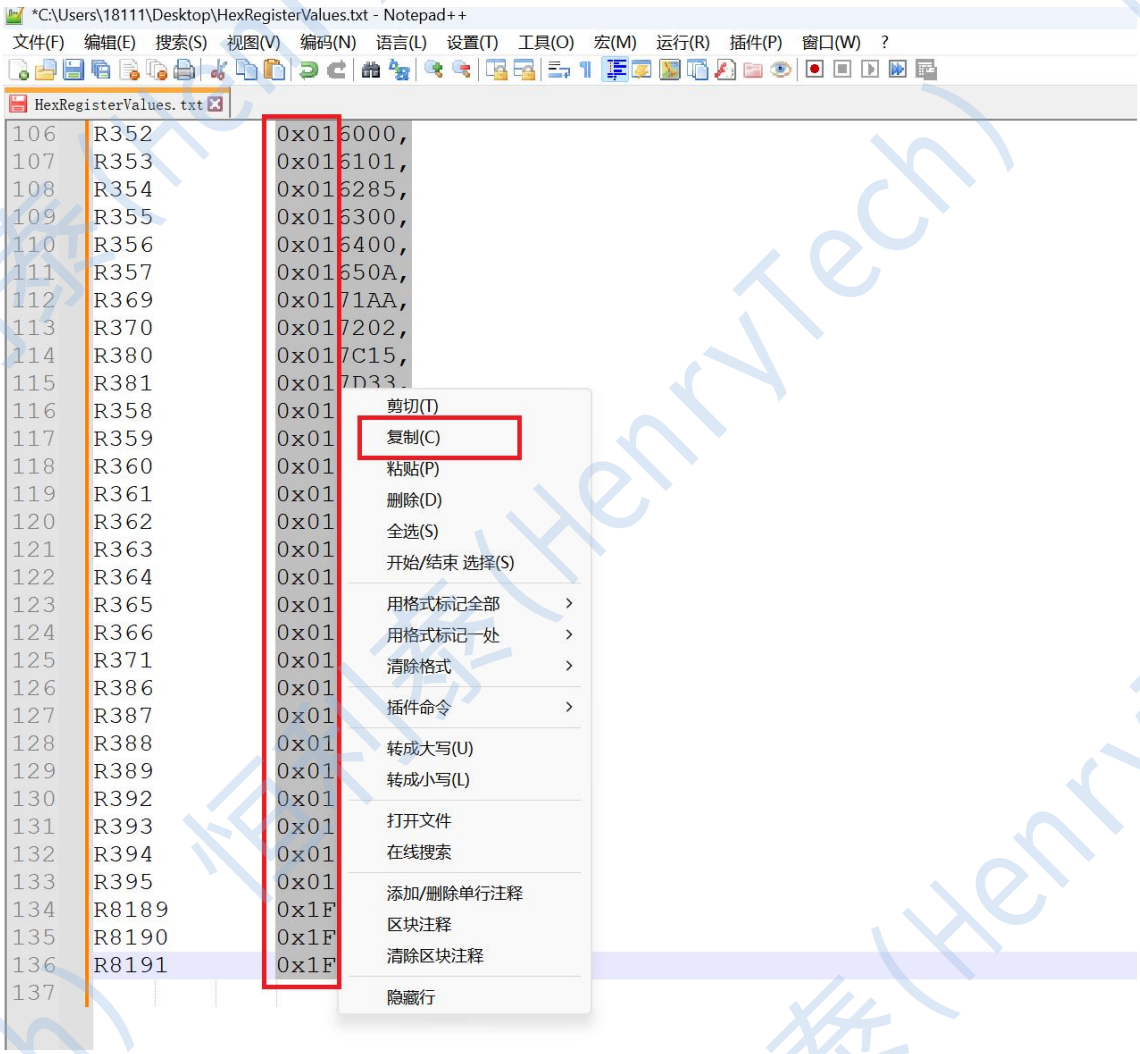
最后我们点击抓取 ▶ 符号抓取。由于未设置Capture等条件，随时可以进行触发抓取信号，采样后显示波形如下：



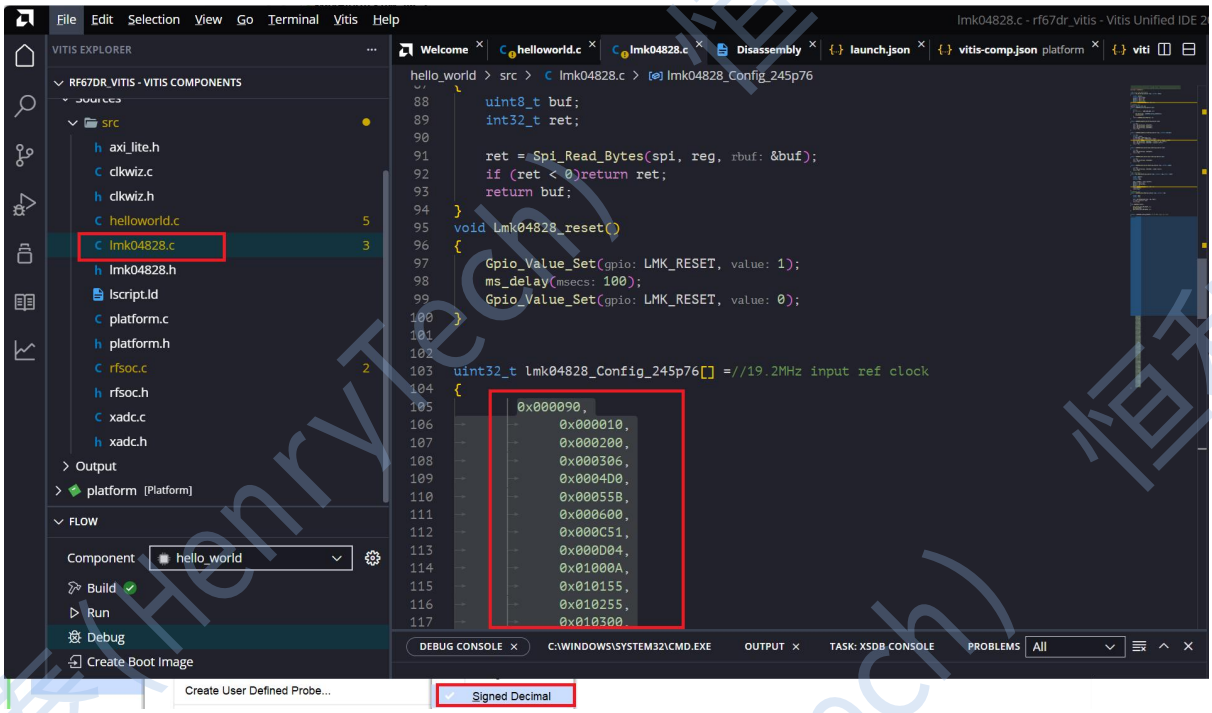
我们缩放一下，将标线拉到峰值处，可以看到，峰峰值均位于同一时刻对齐。并且，幅度值高度一致，可以说明，射频通道同步性和信号幅度一致性非常好，已完成同步。

十四、时钟更改配置

我们在《15LMK04828时钟配置》中讲解如何导入导出配置，我们对时钟做了修改后导出的txt，稍作编辑即可使用。我们推荐使用notepad++打开编辑，网盘资料中有提供此推荐的文本编辑器安装包《02软件和相关linux包/Notepad++安装程序》。我们将右侧的二进制数据每一行对齐。注意notepad++支持多行编辑，按alt再向下滑动鼠标即可选中多行同一列，我们将二进制数据最后都加上英文的逗号，然后选中所有二进制数据，右键选择复制：



接着我们在vitis中打开lmk04828.c将配置的时钟数组数据全部替换：



替换后vitis重新编译即可。注意复制要完整，配置文本TXT里面每一行寄存器都要编辑复制进来。

十五、其他

关于RFSOC，更多信息请参阅《04参考资料/pg269-rf-data-converter-en-us-2.6》，我们不过多说明，后续将不定期提供一些简单的例子进行学习。

ADC的AD1 AD2为两通道模拟输入，需要配合信号源进行实验。在工程中ila抓取界面是adc的第九通道和第十通道数据。

VIVADO对设计工程的芯片功耗分析和报告

------ 基于RF67DR-Lite开发板套件

恒利泰(HenryTech)

目录

一、 文档实现功能介绍 3
二、 例程分析实例 3
三、 一些器件温度基本概念 6

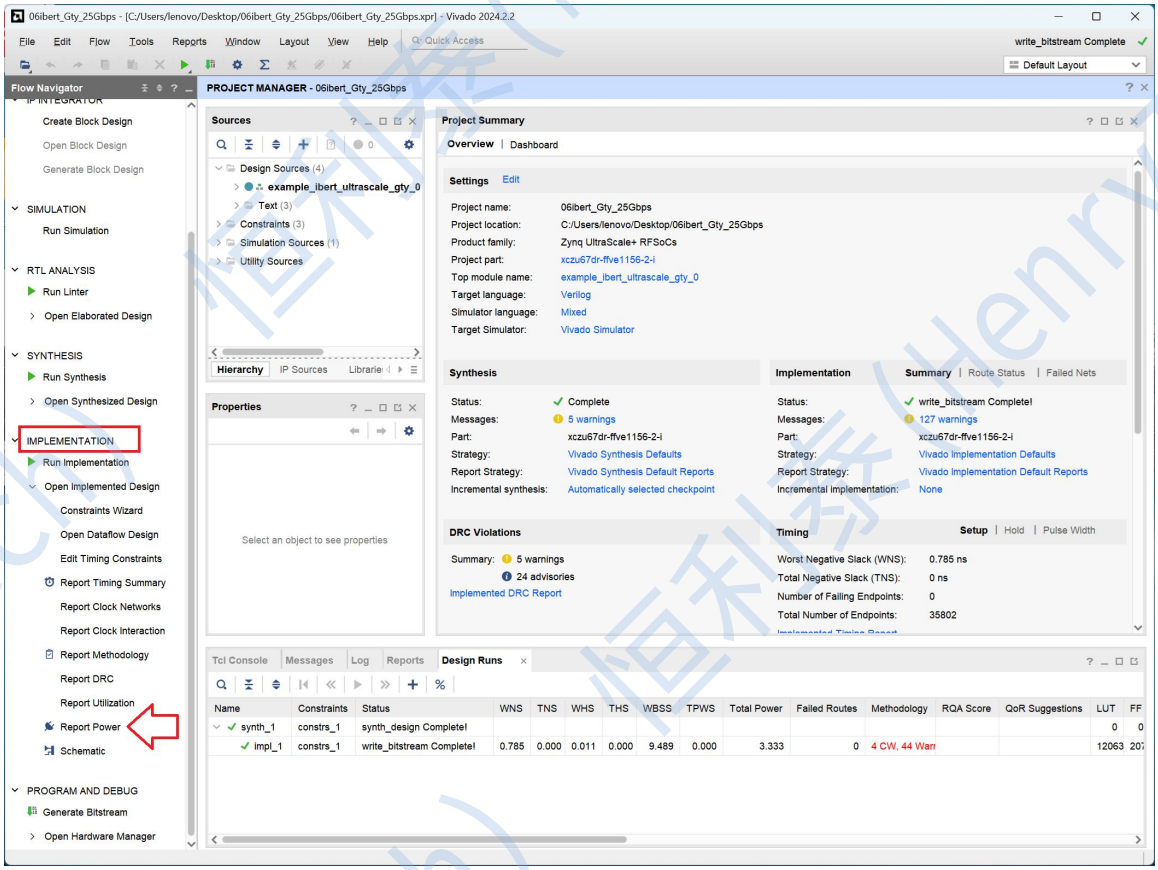
恒利泰(HenryTech)

一、文档实现功能介绍

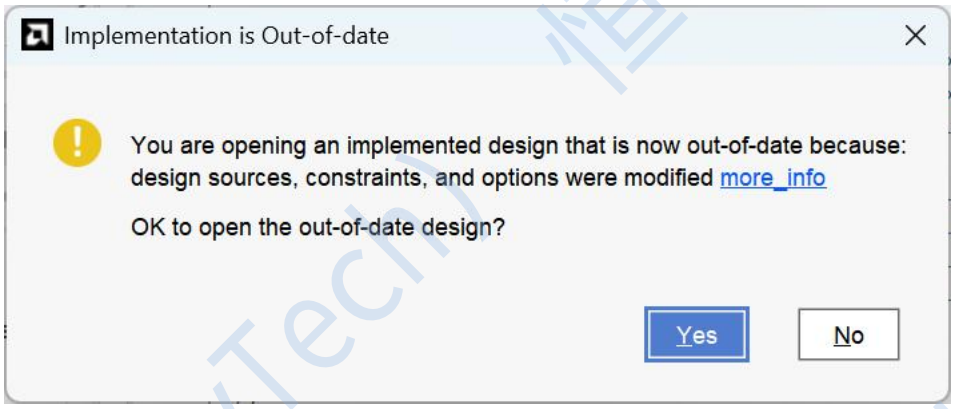
本文档简单教大家如何使用VIVADO进行功耗评估。我们在设计完硬件代码之后，工程使用了多少逻辑，这些逻辑每多一个少一个都会引起功耗变化。比如说，一个代码中使用的RAM，FIFO越多，所驱动的存储逻辑就会越多，代码中使用的时钟频率越高，信号跳变越快会引起动态功耗变高等等。而我们VIVADO软件自带功耗评估功能。

二、例程分析实例

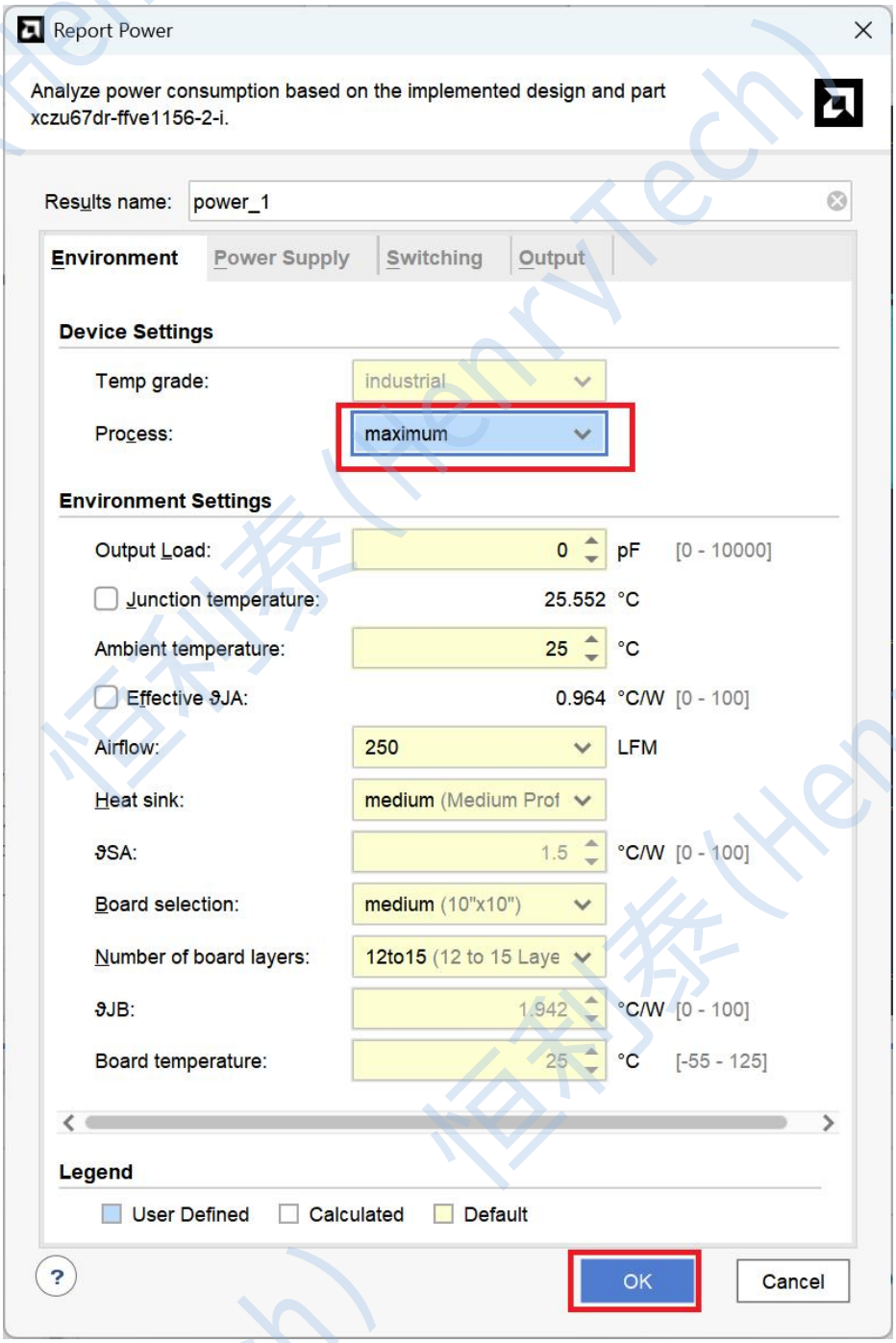
以我们的RF-67DR-Lite开发板的FPGA部分《06ibert_Gty_25Gbps》例程为例进行分析。功耗可以分析的要求是，我们代码完整，我们工程需要能被综合，无代码错误。有一个关键点，大家如果使用了时钟模块，或者某些IP有时钟配置，那么我们综合器可以获得我们代码的时序信息，频率信息，就可以正确估算功耗。我们所有的可以下载到板子的demo工程都可以进行功耗评估。什么情况下不能进行功耗估算？就是假设我们代码用了很多资源，但是没有用到PLL一类的时钟模块，我们使用一个比如100M晶振给逻辑做时钟输入，我们也没有时钟约束语句对时钟进行约束，那么，综合器不知道你频率和占空比等信息是多少，就不能估算，因为功耗是基于时钟，资源综合进行计算评估得到的结果。这种情况下，大家可以约束时钟来告诉综合器时钟参数，就能正常估算。我们打开IMPLEMENTATION下面的Report Power：



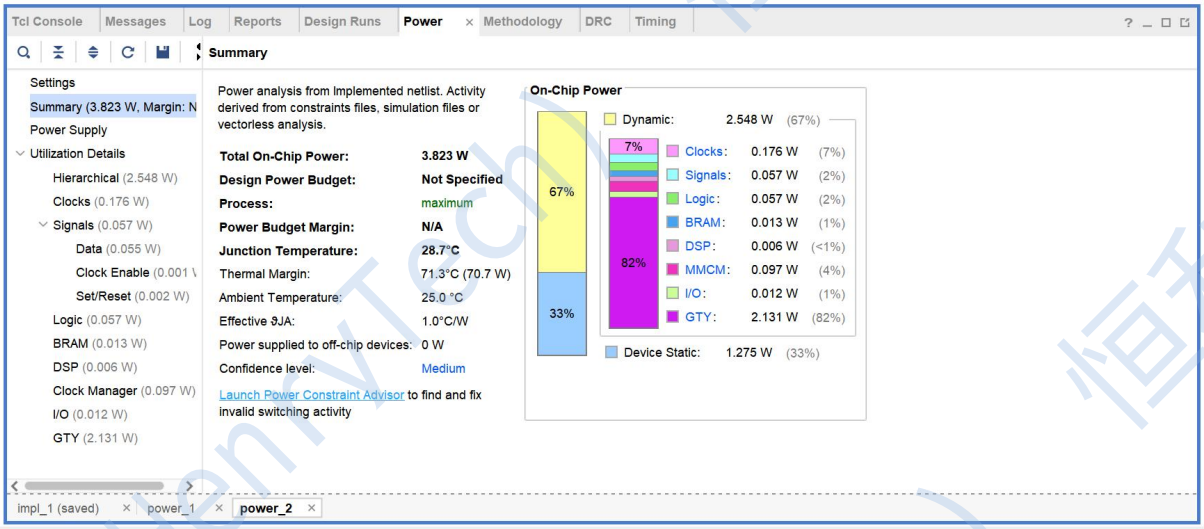
如果打开的例子很久不用，或者修改过代码，软件提示过期。我们点击NO，然后先重新综合生成Bit文件后再打开：



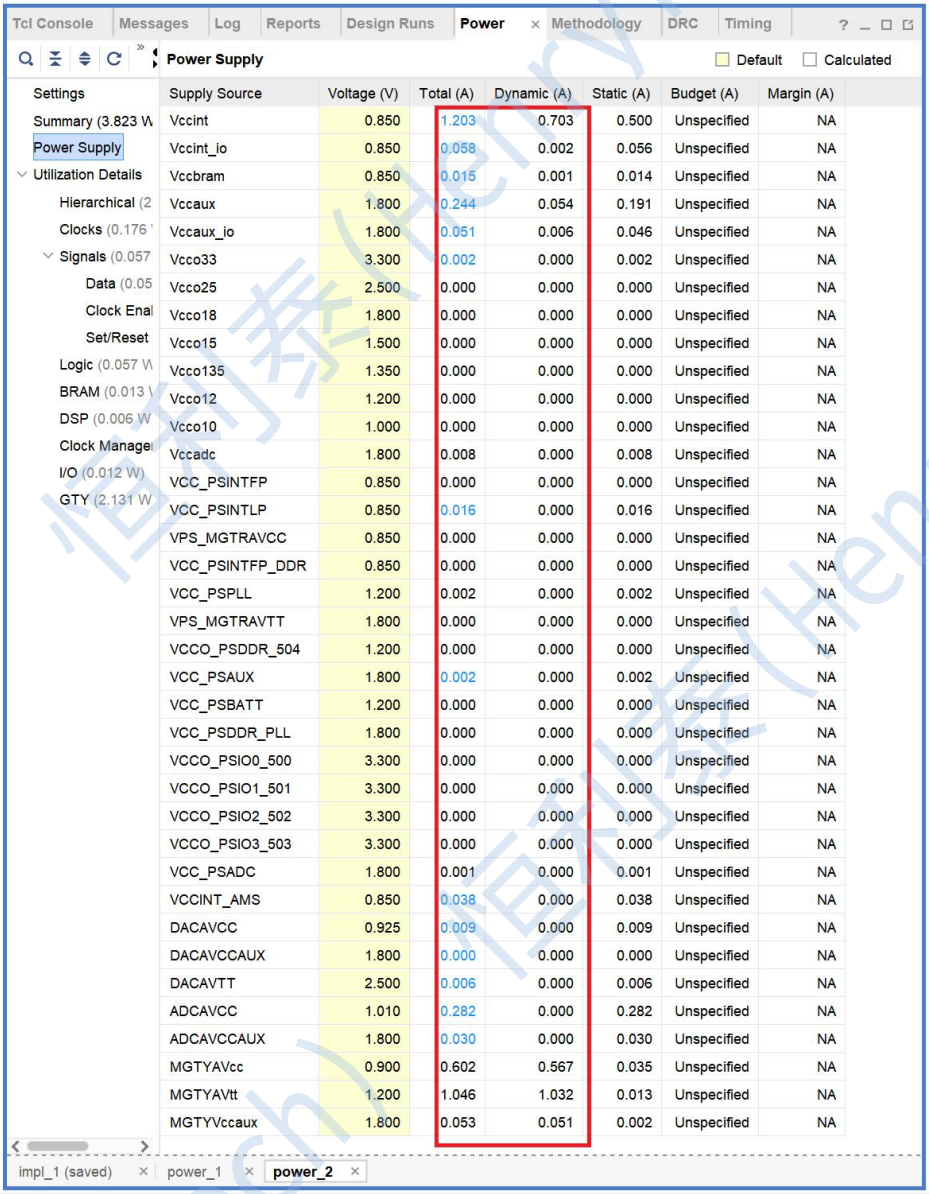
重新综合后，我们重新打开report power，等待一会儿，弹出功耗分析配置界面，配置界面配置如下：



配置默认，直接点击OK，等待生成Power报告窗口：



大家可以看到，GTY部分占了很大一部分功耗，整个Ibert工程功耗大概是4.321W。最后大家再看一下各路电流大小，可以知道电流都去哪儿了。除了GTY高速收发器，内核也占了很大一部分：



三、一些器件温度基本概念

结温（junction temperature）：是处于电子设备中实际半导体芯片（晶圆、裸片）的最高温度。它通常高于外壳温度和器件表面温度。结温可以衡量从半导体晶圆到封装器件外壳间的散热所需时间以及热阻。

热余裕（Thermal Margin）：指处理器安全工作的温度范围，超过处理的Thermal Margin将产生过热，将使处理器损害，或者损害其他器件。

恒利泰(HenryTech)