HLT-XC7Z020-AD9363-B开发套件资料

快速测试及使用指南

开发软件VIVADO2018和LICENSE安装

AD936x裸机驱动时序讲解及例程

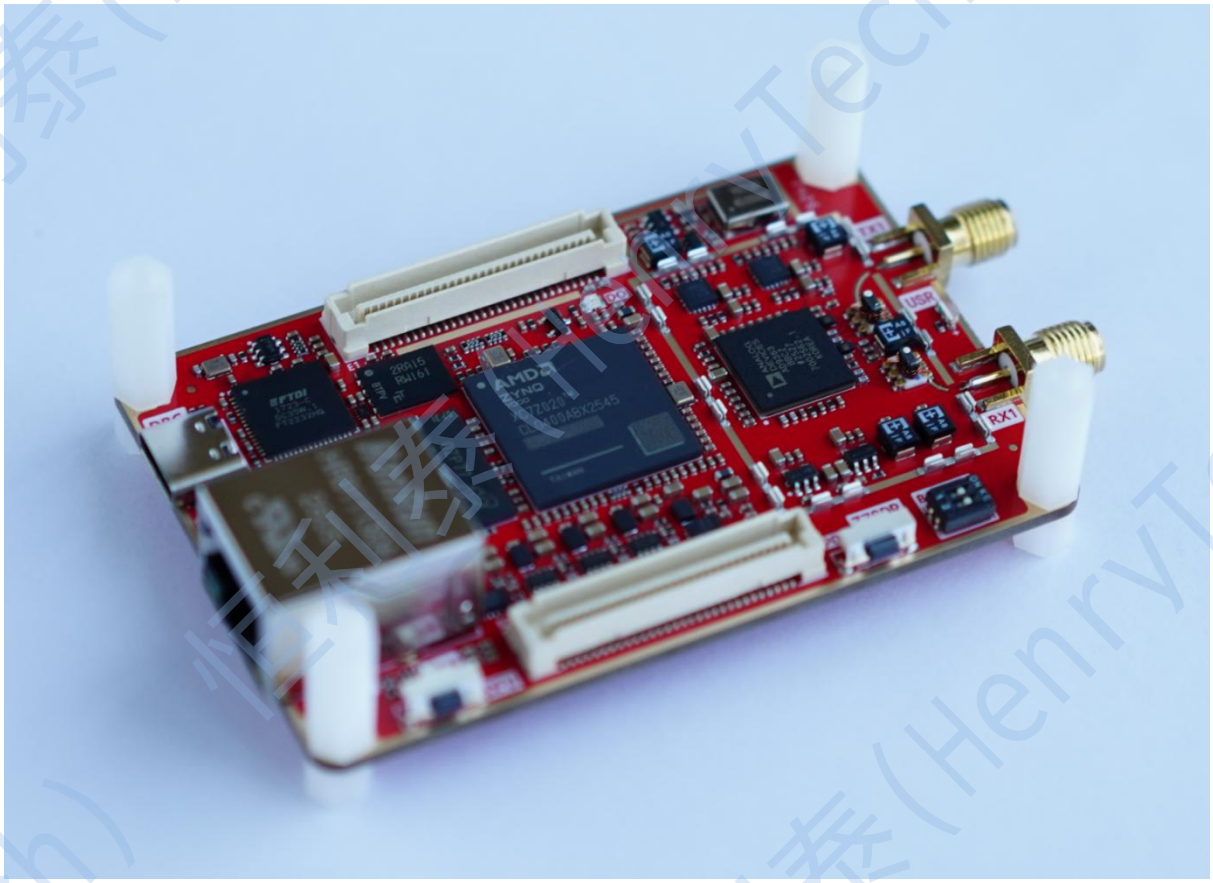
ubuntu18.4虚拟机的安装

ubuntu安装和使用GNURadio

使用matlab2018a的硬件支持包安装

快速测试及使用指南

------ 基于Z7SDRMicro开发平台



目录

一、文档实现功能介绍

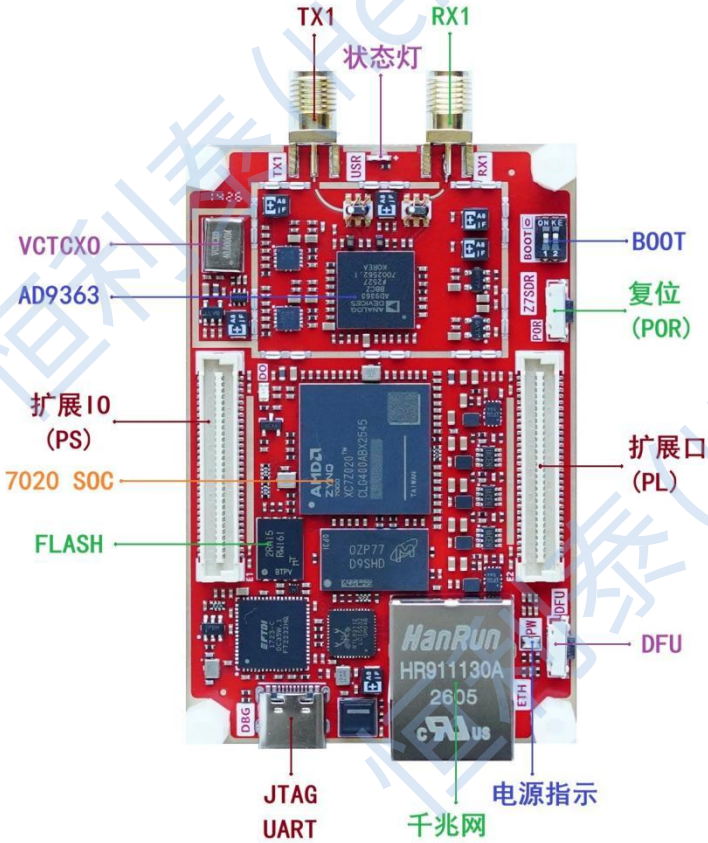
本文档旨在大家拿到开发板后，可以进行参考固件烧写测试以及快速使用，我们可以将Z7SDRMicro这款开发板作为软件无线电开发平台。批量应用本SDR，性能比原版提升，并且带有扩展接口BTB连接器，扩展出约40个以上引脚，包括PL和PS部分。

本文档可能使用到与FPGA环境相关的VIVADO SDK软件，这类使用需要用户具备一定的XILINX FPGA工具使用经验和技能，涉及需要使用的部分，如果不具备VIVADO使用经验，建议提前自行学习，在本产品资料中不做基础开发学习参考。

另外如果是SDR开发者但不具备软件开发能力，可直接使用出厂固件进行使用，因为提供固件支持常见的SDR软件。

文档最后有重刷出厂固件操作，一般情况下，非FPGA和嵌入式开发者，普通无线电开发用不到，如果万一出现固件误操作丢失变砖，可以按照《02开发软件VIVADO2018和LICENSE安装》搭建vivado环境，然后进行出厂固件烧写恢复。相关操作后续我们可能也会有视频演示。文档开始之前，大家根据这个图先了解一下SDR的功能接口，可以清楚地知道每个功能接口所处位置：

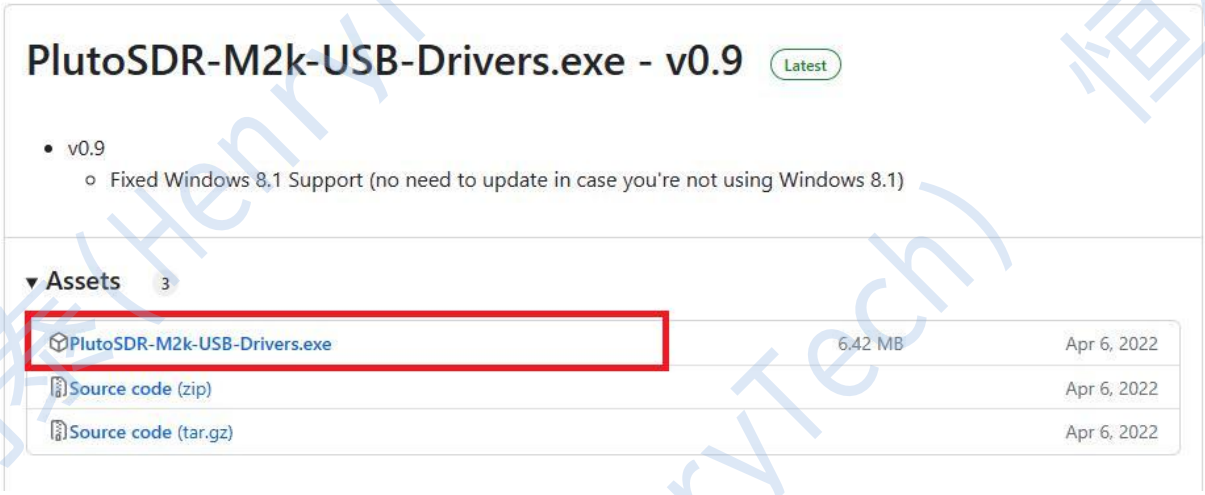
Z7SDRMicro 功能接口标注



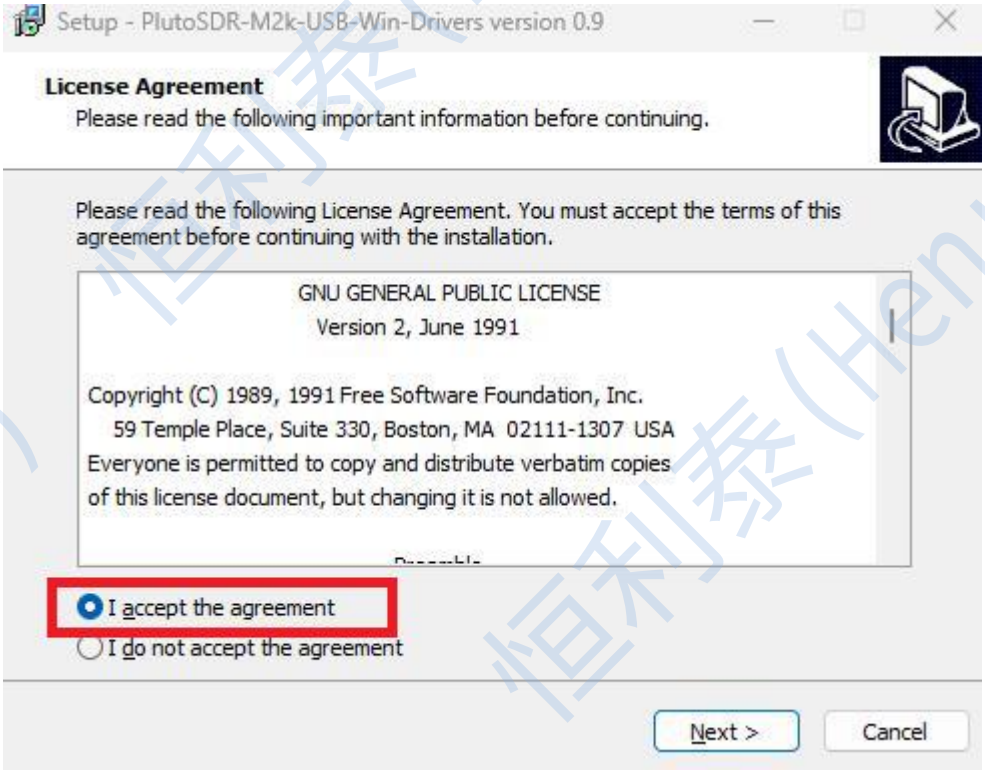
二、Pluto SDR USB驱动安装

Pluto SDR USB驱动下载链接：https://github.com/analogdevicesinc/plutosdr-m2k-drivers-win/releases

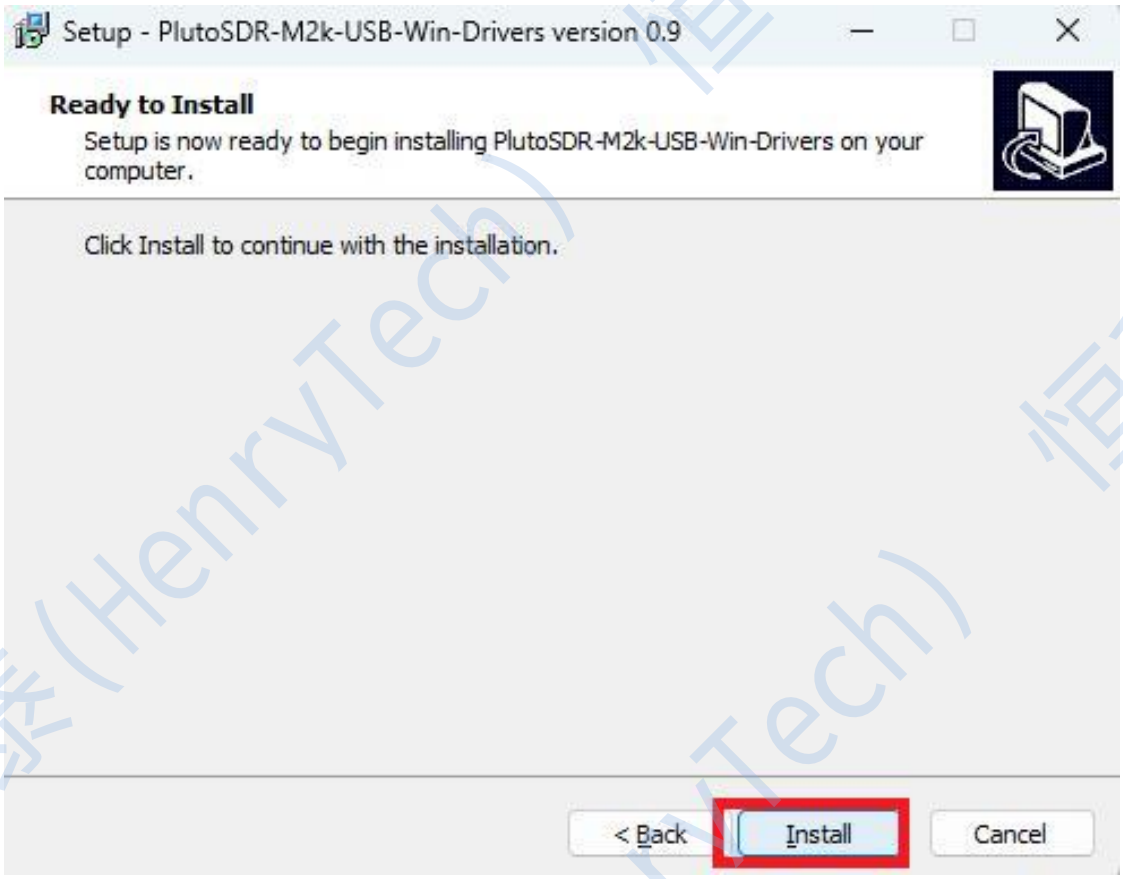
找到如下文件点击下载即可（也可以在提供的百度云网盘资料里的《Z7SDRMicro\01工具软件\Pluto驱动》找到此文件）：



Git访问可能会比较慢或者打不开，大家可以多尝试几个时间段下载。下载之后双击运行，推荐右键以管理员身份运行，第一个页面默认接受，点击next：

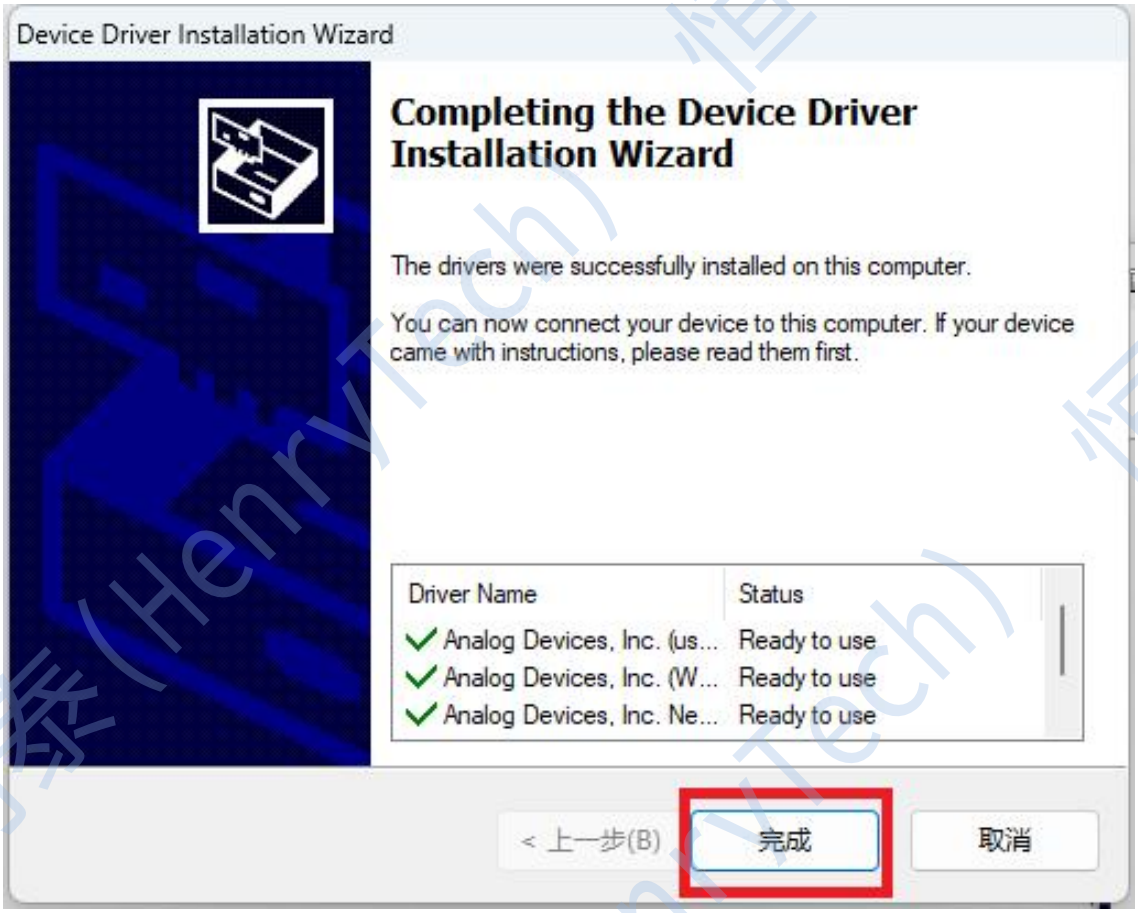


点击安装：

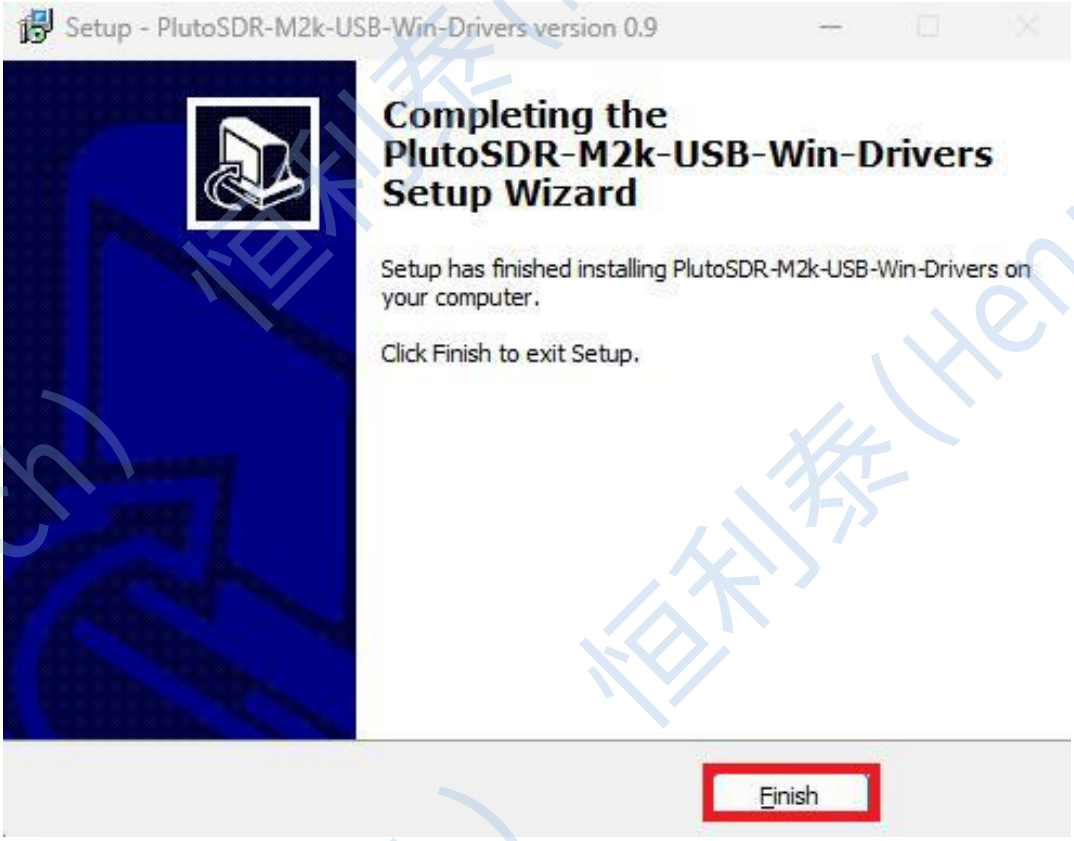


弹出新的安装界面点击下一页：

最后点击完成：



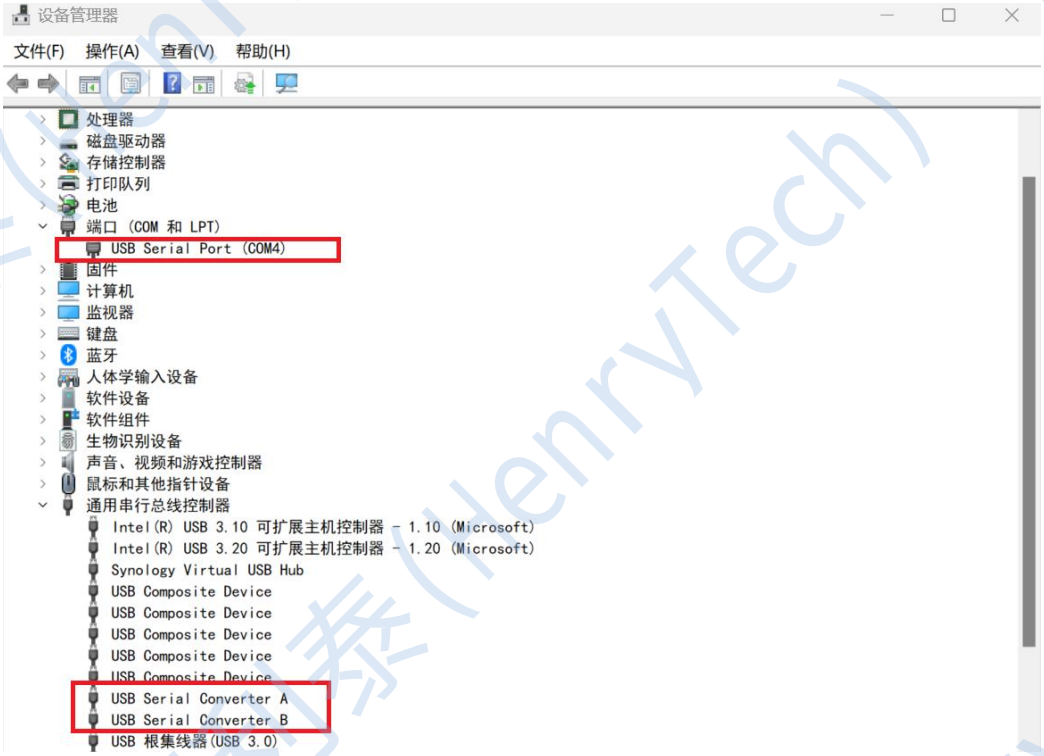
回到上个界面也点击finish完成安装：



到此我们就完成了USB驱动的安装。除此之外我们还需要安装JTAG/UART调试口驱动。

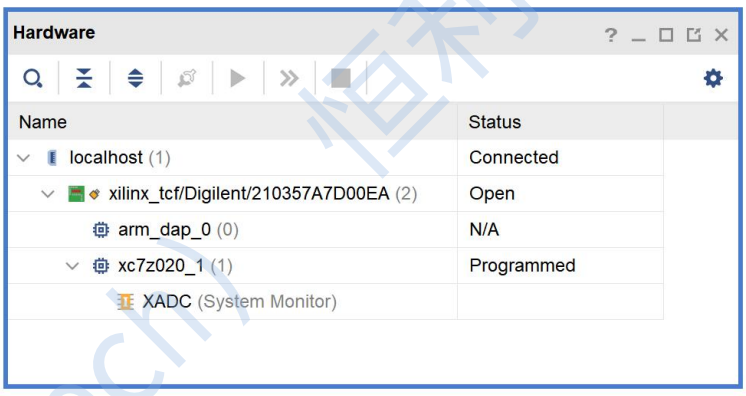
三、调试接口（串口和JTAG）驱动

一般情况下，按照《02开发软件VIVADO2018和LICENSE安装》安装好VIVADO，JTAG/UART驱动即自动适配。针对不做FPGA开发的用户，可以单独安装驱动，安装包位于《Z7SDRMicro\01工具软件\下载器驱动》。 正常情况下，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sdk软件代码，建议BOOT设置为JTAG模式开机上电。

设备插上DEBUG口连接到电脑USB，我们使用vivado打开硬件管理然后open target后可以看到开发板识别到了ZYNQ芯片：



四、BOOT拨码开关设置

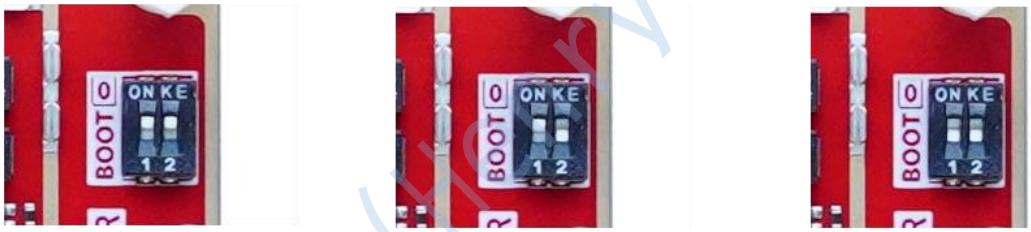
开发板有三种模式设置，分别是JTAG、FLASH、SD模式。其中JTAG模式一般是调试FPGA或者下载调试ARM程序用。需要注意，DEBUG口连接电脑，打开vivado，使用XADC监控，必须是JTAG启动模式才可以读取到芯片温度，其他模式读取到温度应该是-273.15度。并且大家如果要烧写FLASH，也必须是JTAG模式，烧写完FLASH后再设置成FLASH启动即可。

SD模式仅用于用户自己设计扩展模块，使用TF卡的场景。这里不做讨论。

FLASH是用于板载FLASH启动固件程序。FLASH内的固件出厂有烧写。用户如果固件掉了或者有过FLASH操作丢失固件，可以按照本文档后续的启动固件烧写操作恢复FLASH内的固件。

三种模式如下设置：

JTAG　　FLASH　　SD



需注意，启动开关必须在断电模式下设置。或者带电设置后，按下POR按钮重启复位之后才能生效。一般情况下，包括出厂，默认FLASH启动。

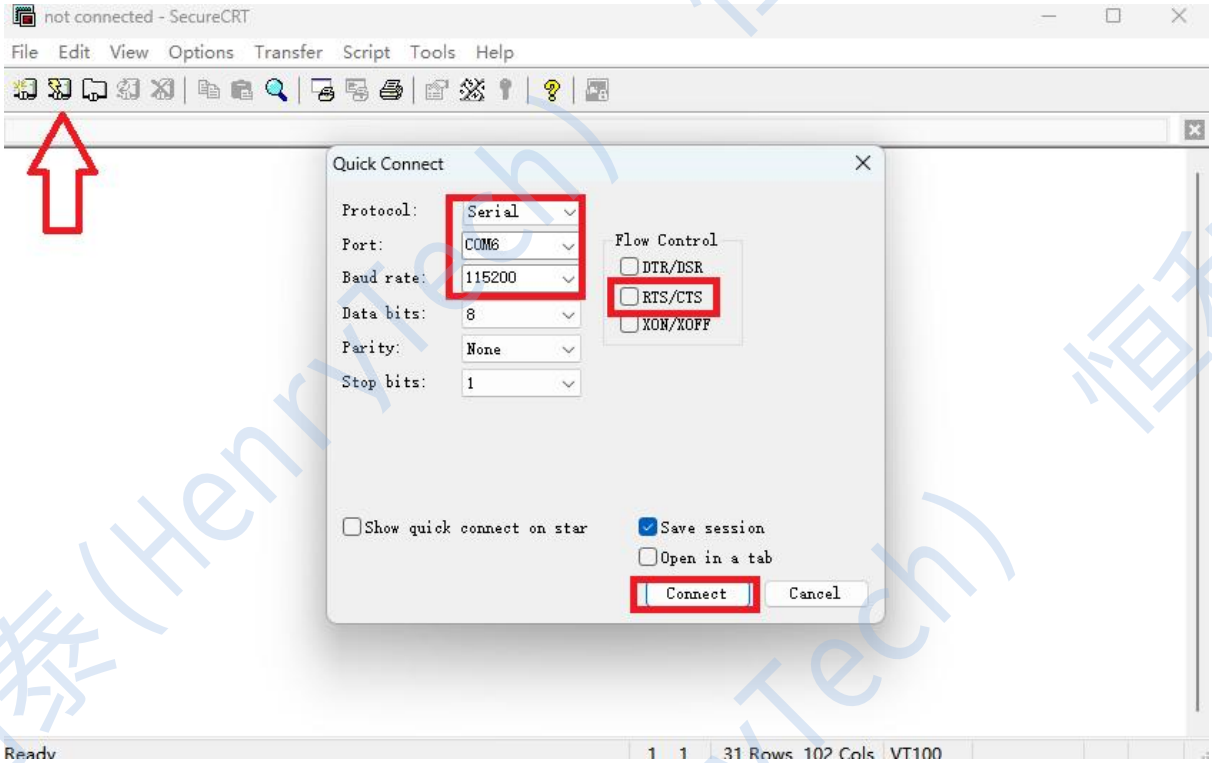
五、FLASH启动

首先，将开发板的启动BOOT设置为FLASH模式，然后将DBG（JTAG UART）TYPE-C USB口连接到电脑。根据第三节中，Windows设备管理器看到的串口，可以通过串口登录系统终端。

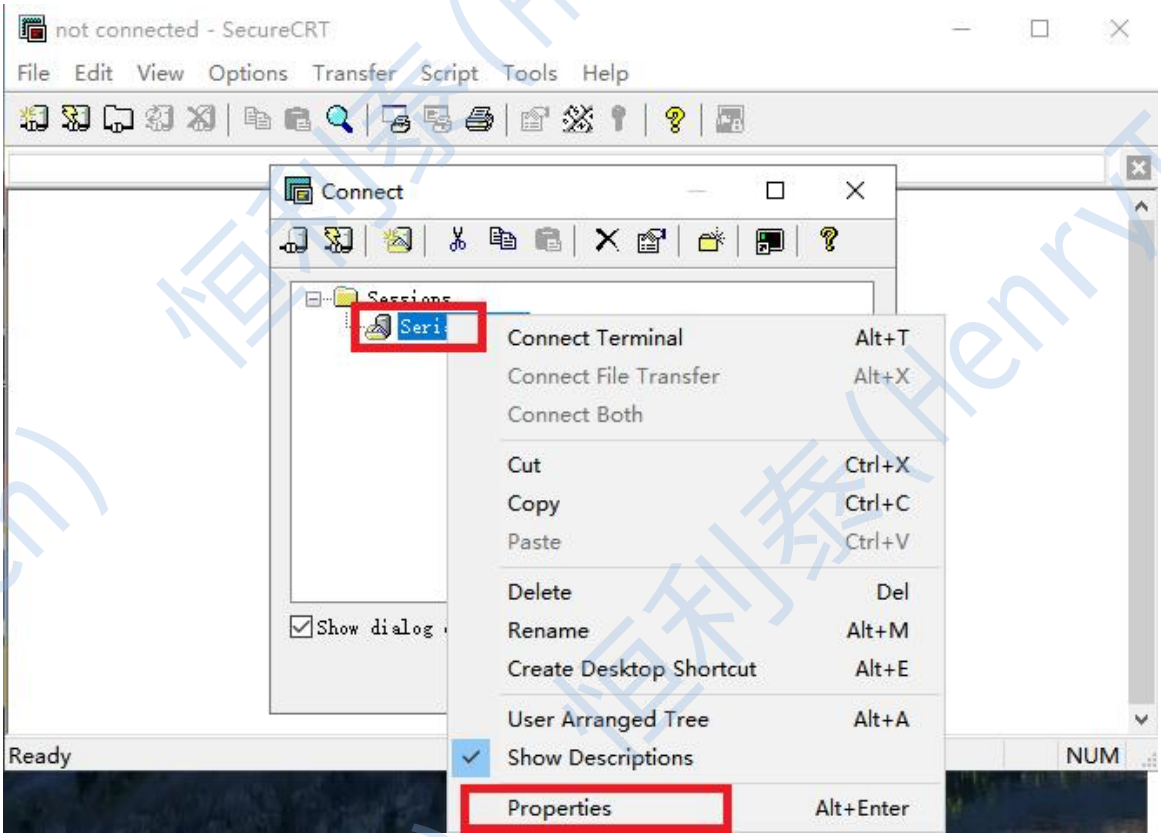
串口终端工具我们推荐使用SecureCRT5.5。资料中有软件包，位于目录《Z7SDRMicro\01工具软件\SecureCRT5.5.zip》。解压后打开目录双击SecureCRT.EXE启动，此软件不需要安装。

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第三节中截图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，笔者这里换了一个USB变成COM6了，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

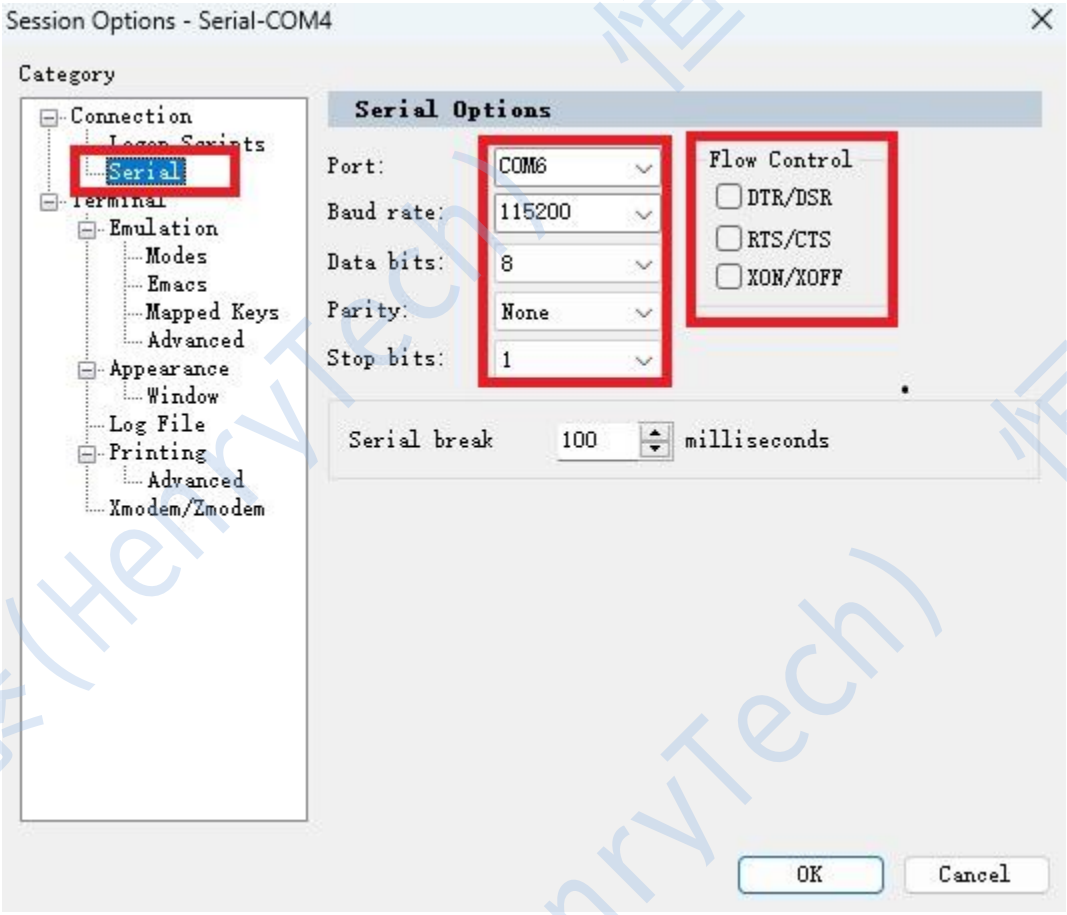
里务必要注意。选择好之后点击Connect：



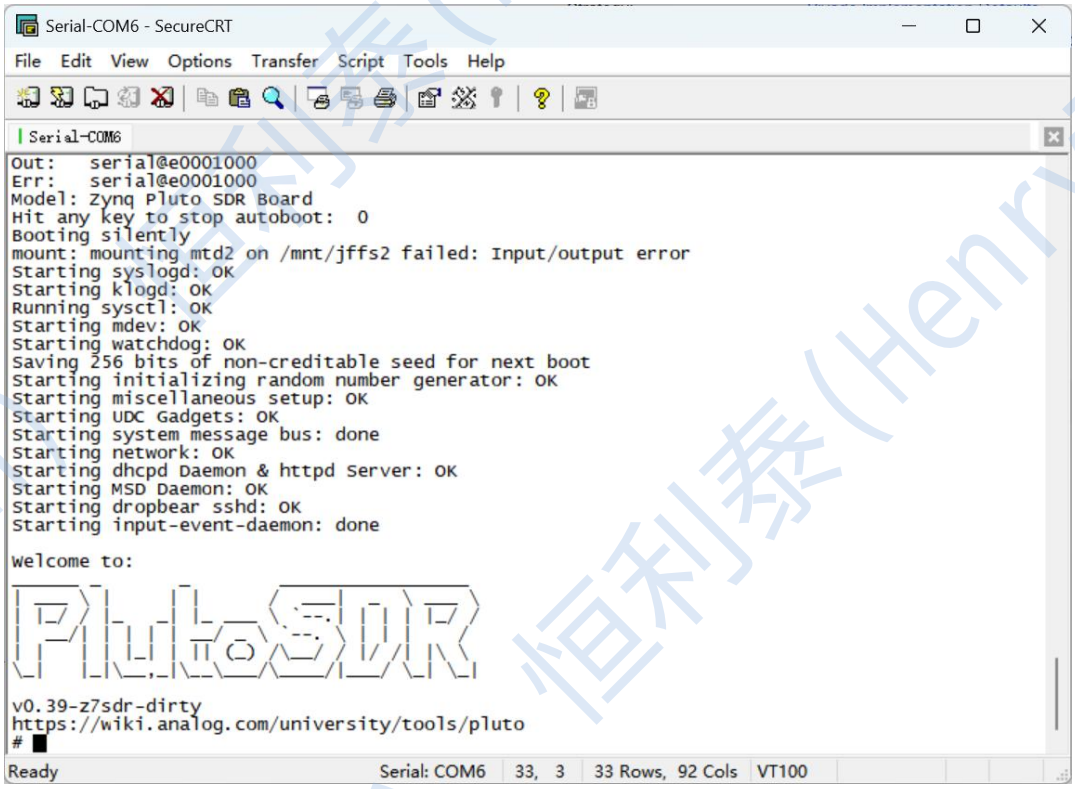
如果不是第一次连接，打开串口工具弹出的连接选择框右键最后一项：



我们连接配置如下：



点击OK完成配置就可以使用了。保证启动设置为FLASH，我们按下POR按钮重启，开始打印启动信息，对于官方原版固件需要登录，但是出厂固件不需要登录：

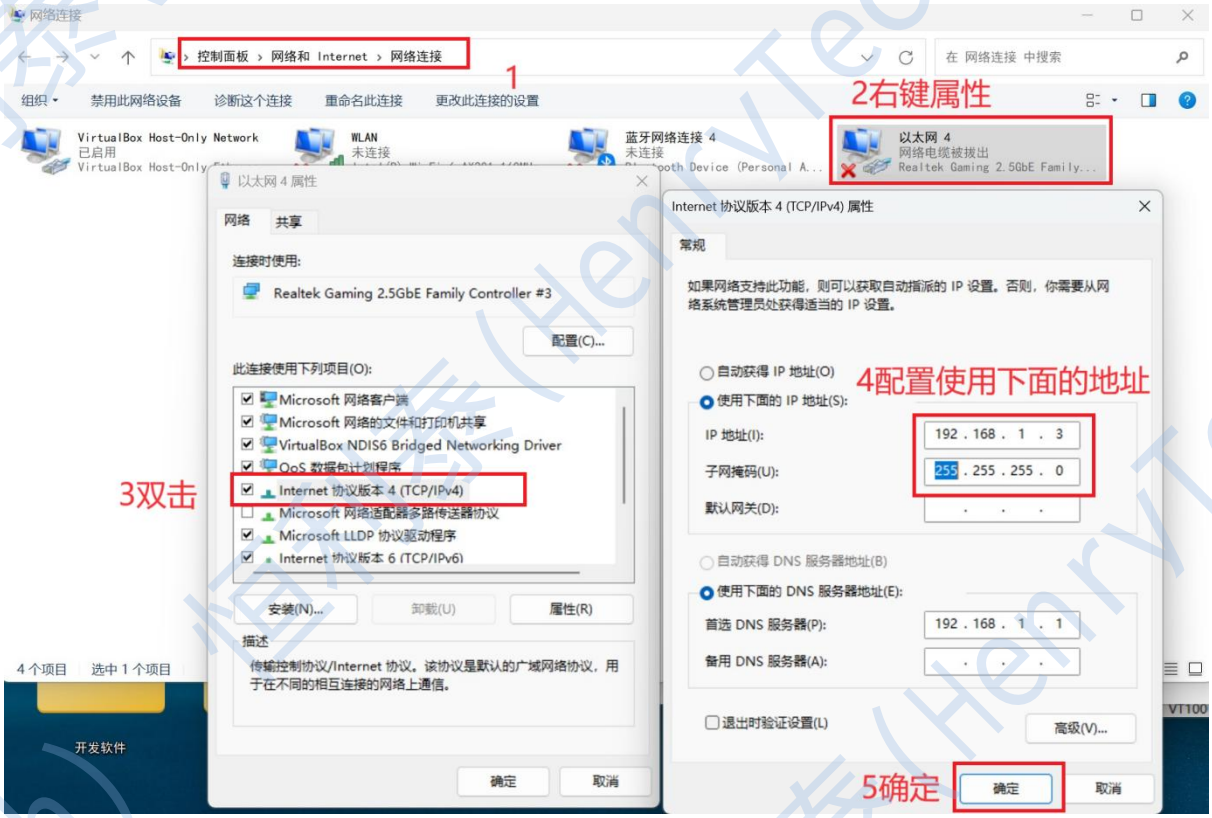


至此我们SDR固件就通过FLASH成功启动。我们接下来可以进行测试SDR基本功能。终端这里实际后续一般用不到，仅演示一下启动的打印信息。

六、SDR#快速测试

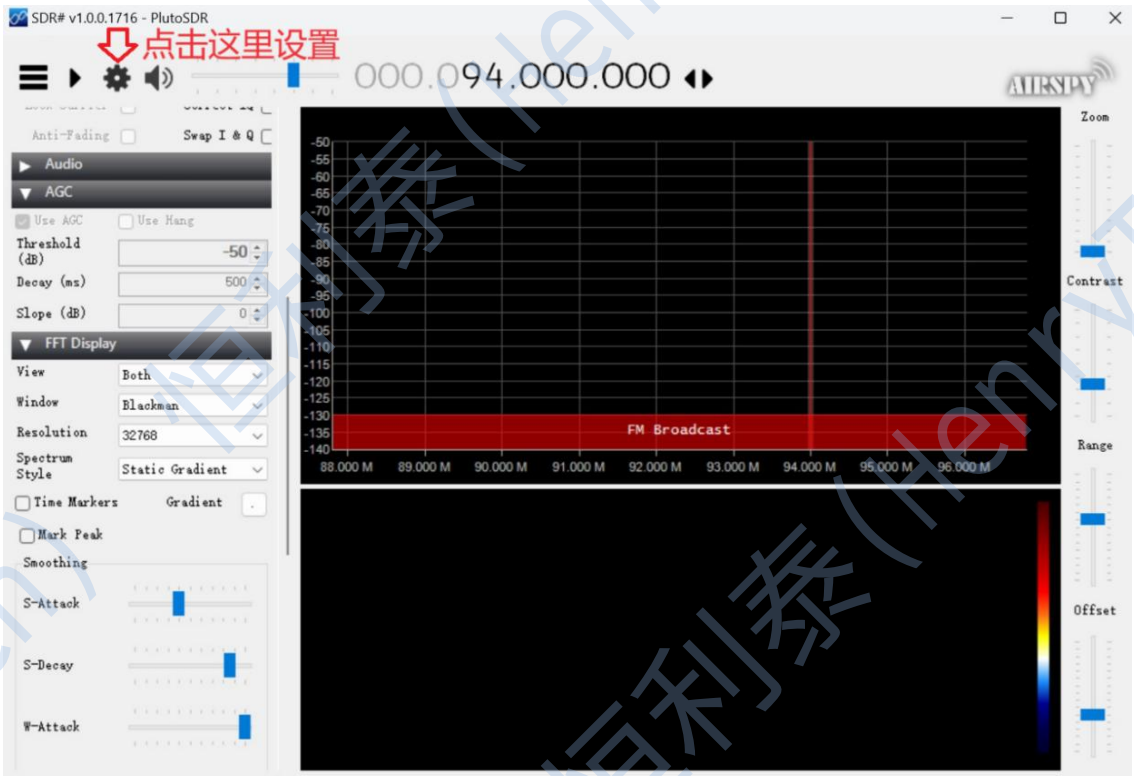
我们在打开连接的串口终端输入ifconfig eth0 192.168.1.10配置SDR的IP地址，我们后续连接上位机软件均为此地址进行：

然后将电脑的网口使用千兆网线与SDR网口直连，电脑网口IP配置于同一个网段，比如192.168.1.3，子网掩码255.255.255.0，电脑端网口IP配置如下图所示：

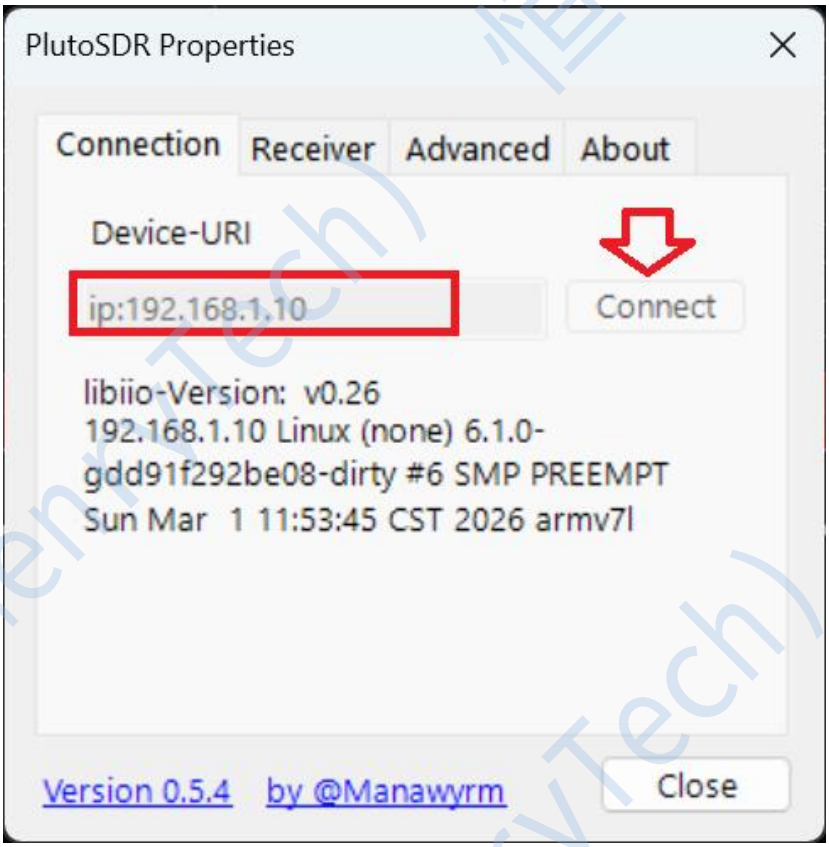


电脑完成上述以太网配置后，开发板启动约一分钟完成，我们Ping一下看是否通上。我们在windows打开命令提示符，开始菜单栏搜索cmd即可找到，并且打开它，打开之后我们输入ping 192.168.1.10（Z7SDRMicro刚才设置的地址），然后可以看到网口是通的：

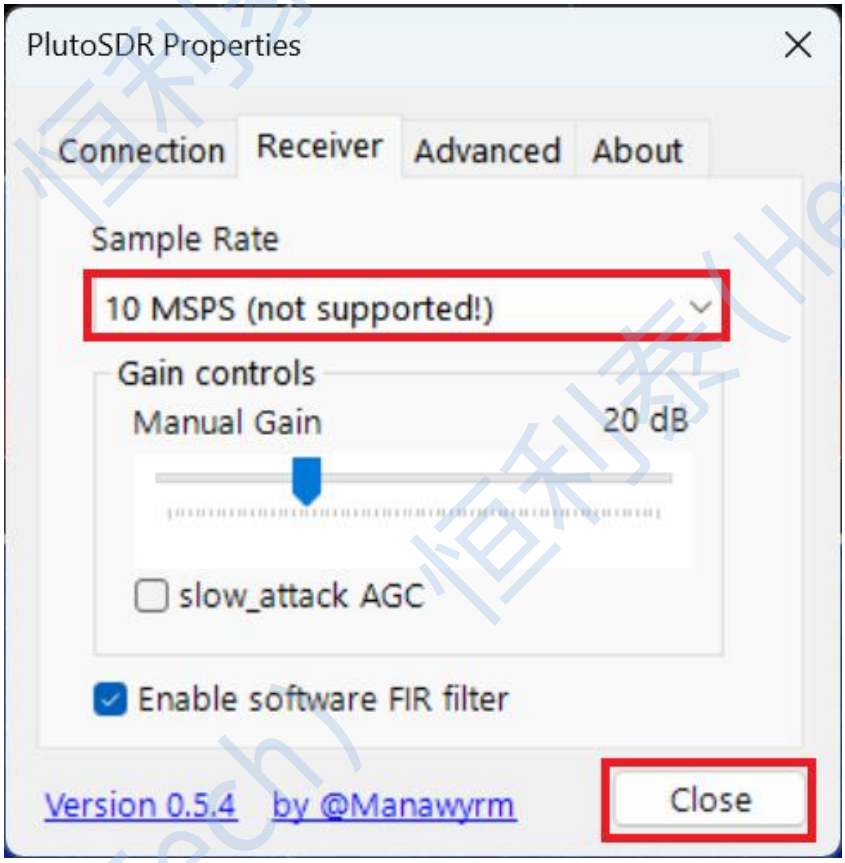
配置好网口连通后，即可进行FM接收测试，收听广播。接下来我们打开SDR#软件。这个工具的路径位置在《Z7SDRMicro\01工具软件\SDR常用WINDOWS软件\SDR#\sdrsharp-x86-noskin》内，下载之后直接运行SDRSharp.exe，打开界面如下，我们找到设置按钮点进去：



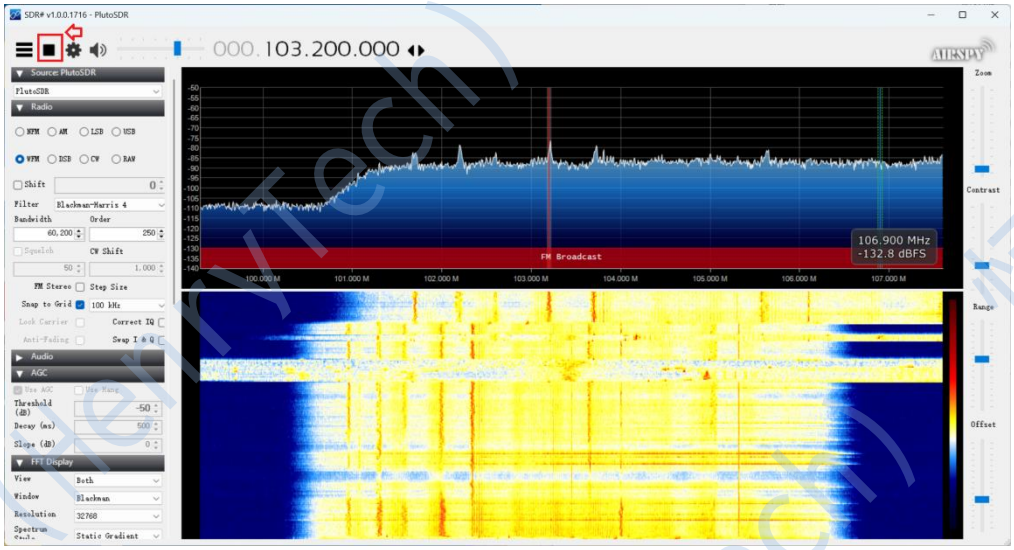
点击进去之后，Device URI填入IP地址。以太网使用的之前配置的192.168.1.10这个IP，我们填入后点击右边Connect，会看到下方的libiio版本及设备信息，看到这些信息说明我们连接成功，可以使用了：



采样率设置，我们默认是5MSPS，可以下拉选择10M，配置提示不支持，这是由于原版PLUTO是USB接口，速度限制最大5M，但是我们Z7SDRMicro使用千兆网口，可以达到更高采样率，实测最高可以设置到12M，最大采样率还与电脑配置和网线质量有一定关系：



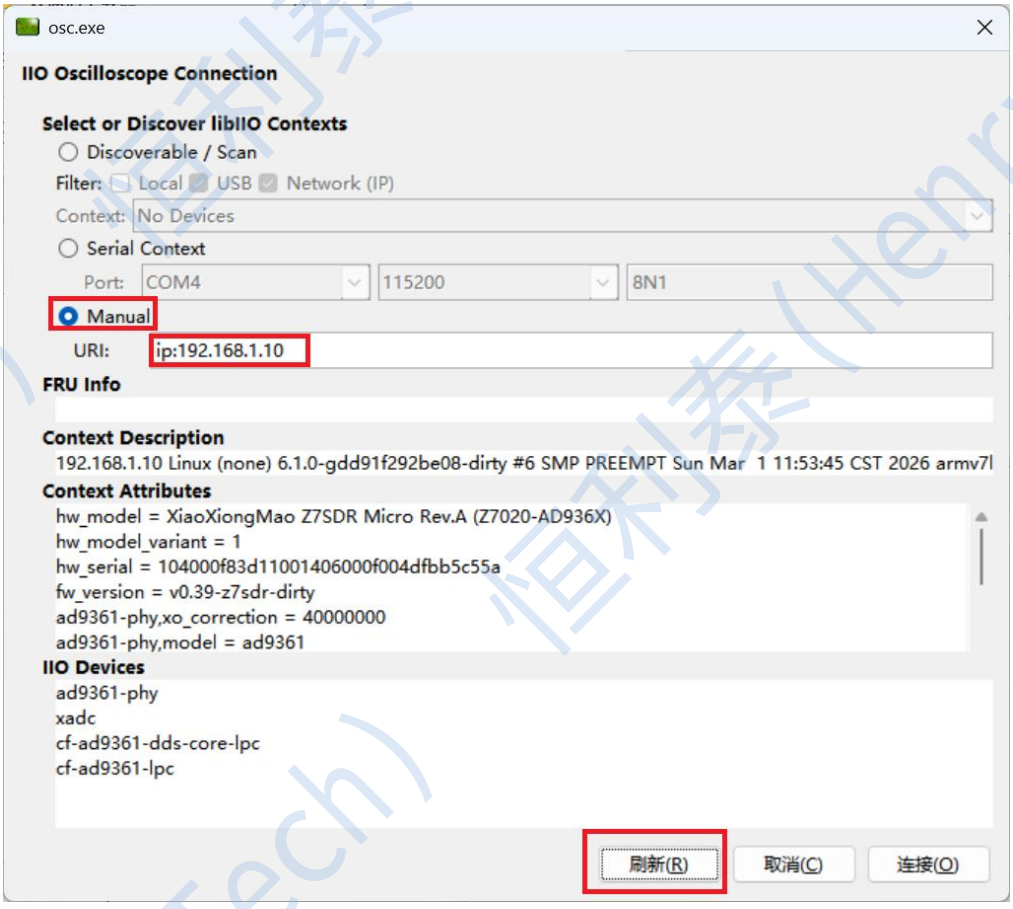
设置好后点击CLOSE关掉，回到主界面点击播放按钮 进行接收。点击后，按钮变为方块，再点击一次就可以停止。我们插上FM拉杆天线到RX1上，拧紧后拉到最长，开始收听：



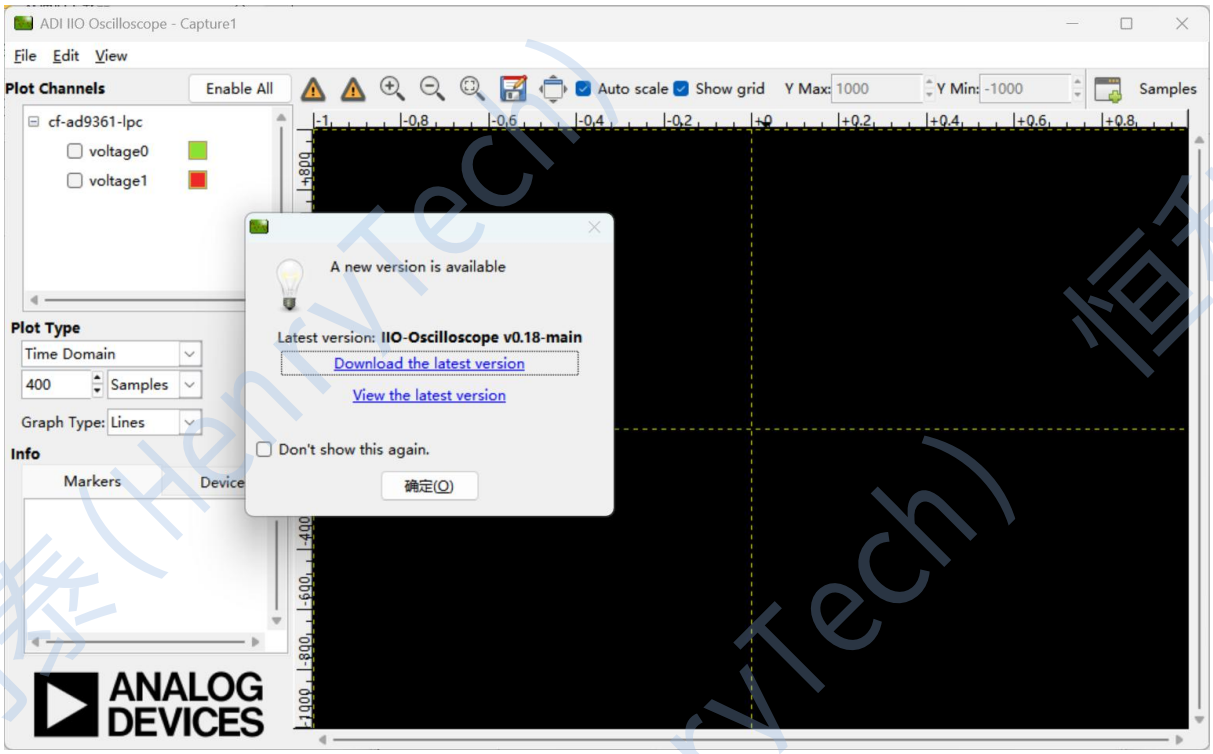
在尖峰大的地方一般都是广播信号，下方瀑布图中，对应颜色深的地方。

七、IIO_SCOPE示波器使用

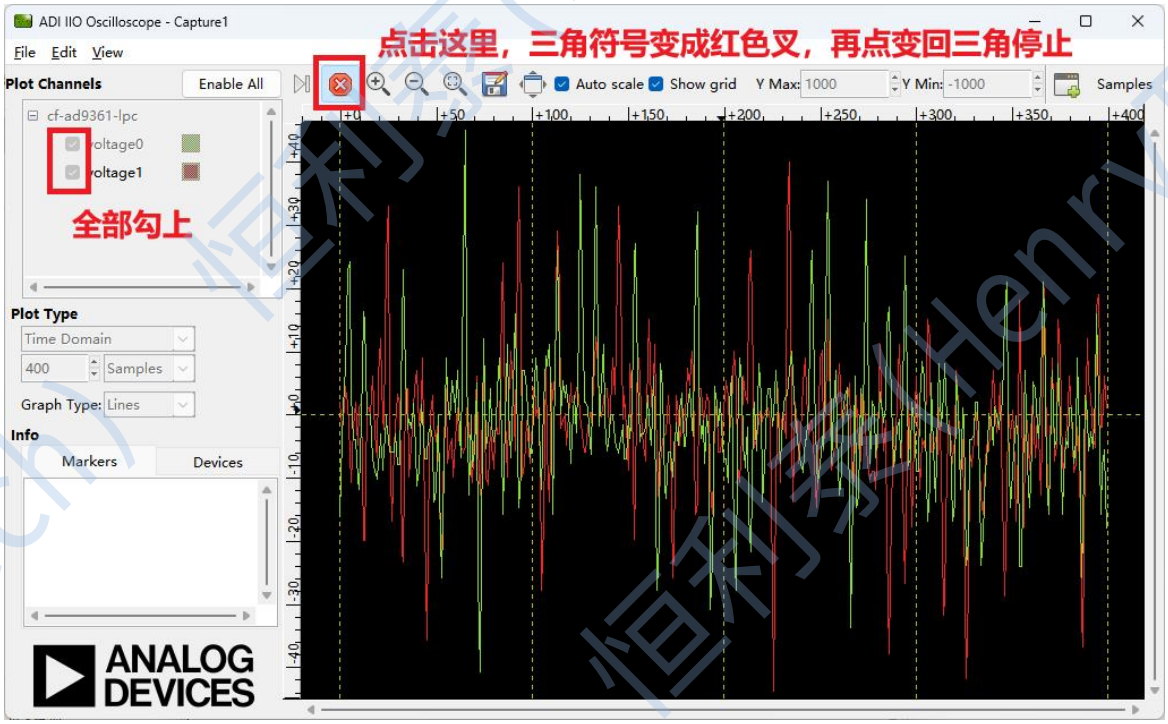
我们安装IIO_Scope示波器，安装过程不做讲解，这是ADI官方提供的基于IIO协议的示波器，安装包位于《Z7SDRMicro/01工具软件/SDR常用WINDOWS软件/IIO_Scope》。打开配置如下，选择Manual手动输入配置。我们手动填入字符串<ip:12.168.1.10>后点击刷新可识别：



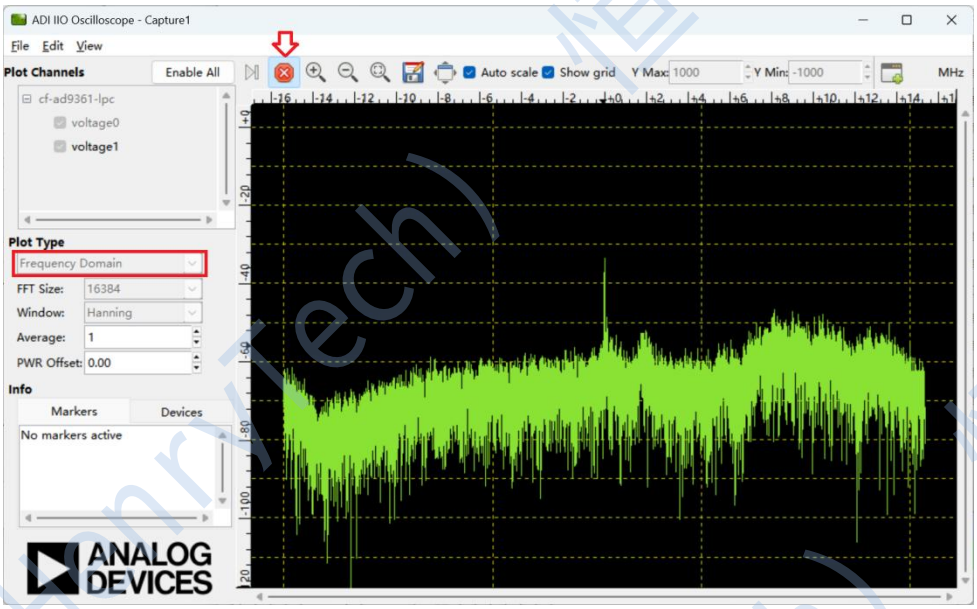
识别到后点击连接，即可连接使用。连接之后进入主界面，可能会弹出一个提示窗口和显示图形界面，提示窗口点确定即可关掉：



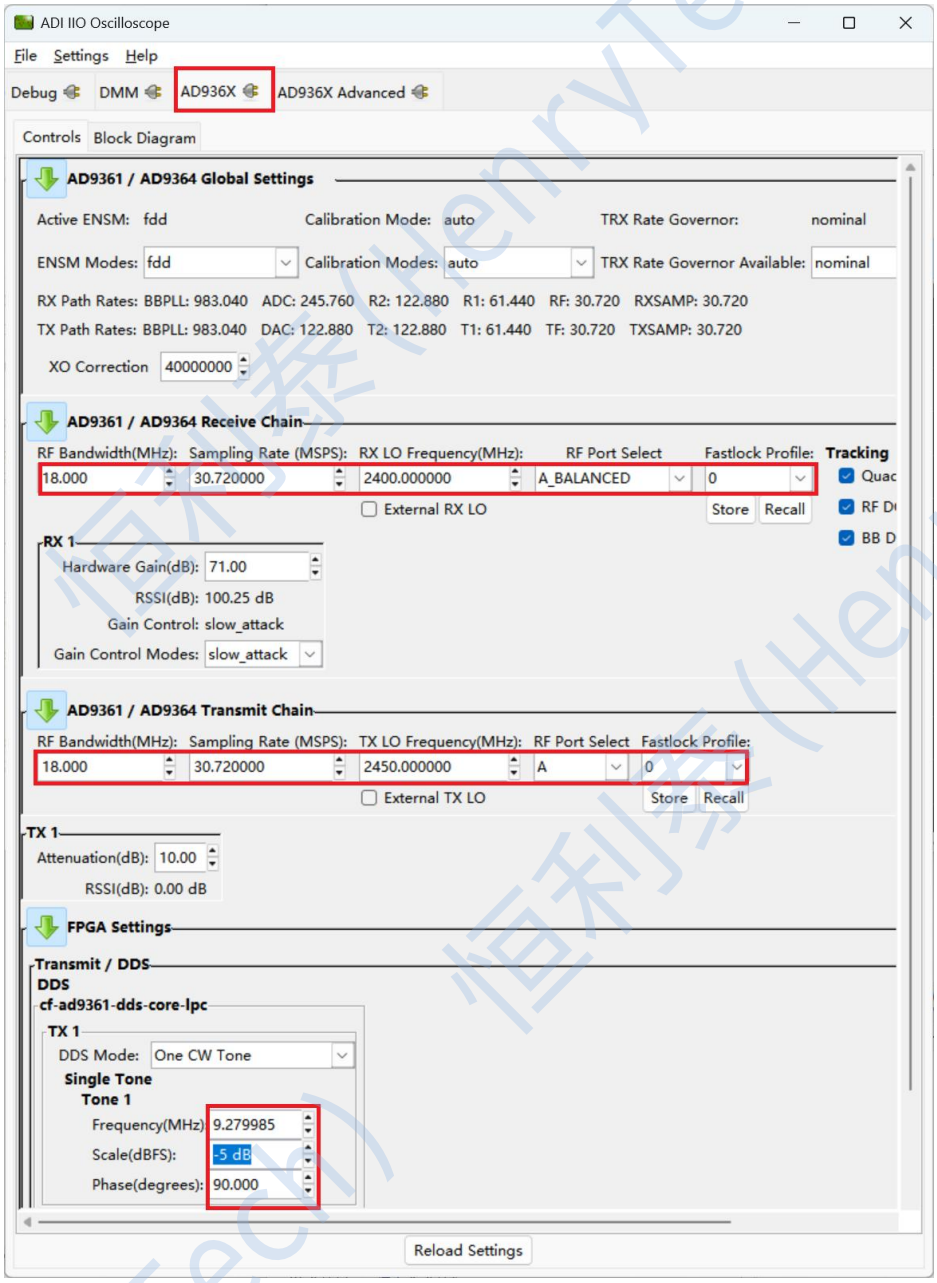
最后弹出的主界面，就是示波器显示界面，用于采集波形和频谱显示等。勾选左边几路votage电压电极上方三角符号开始采集：



我们可以在左边切换Plot Type选择Frequency Domain选择使用频谱显示，然后点击三角符号 开始运行，然后可以看到频谱：



IIO示波器的配置额绵中AD936X配置，有常见的本振，带宽，采样率，和DDS输出，增益，衰减等配置，常用设置如下红色标注：



八、IIO_SCOPE更多使用参考

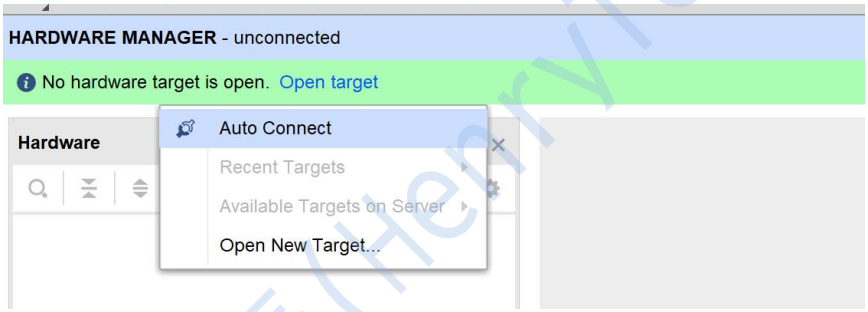
关于IIO Scope使用更多功能请参照官方wiki如下链接：

https://wiki.analog.com/resources/tools-software/linux-software/iio_oscilloscope

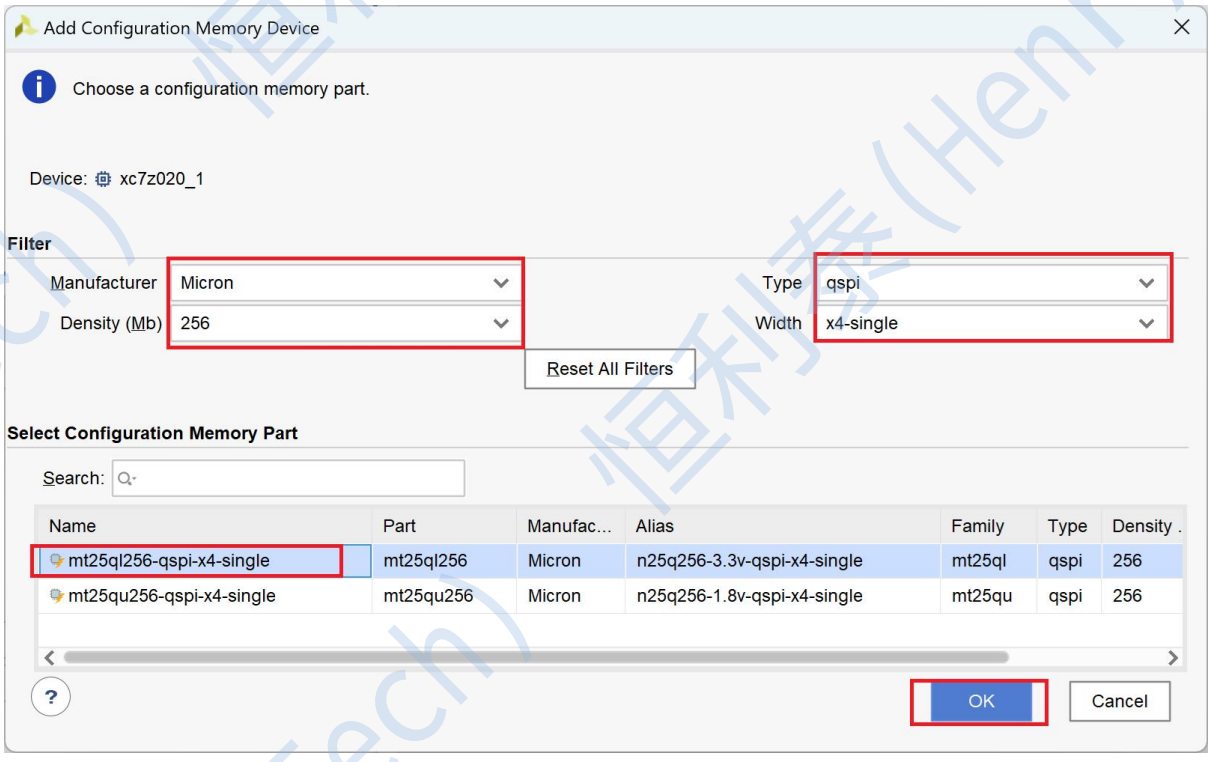
九、固件烧写/恢复出厂

固件文件有两个，一个是unify.bin，是固件文件；还有一个fsbl.elf是bootloader文件。位于《Z7SDRMicro/04例程代码/固件和镜像/编译好的固件启动文件》中，下载待用。注意，文件存放于一个英文路径文件夹中，不要有中文。

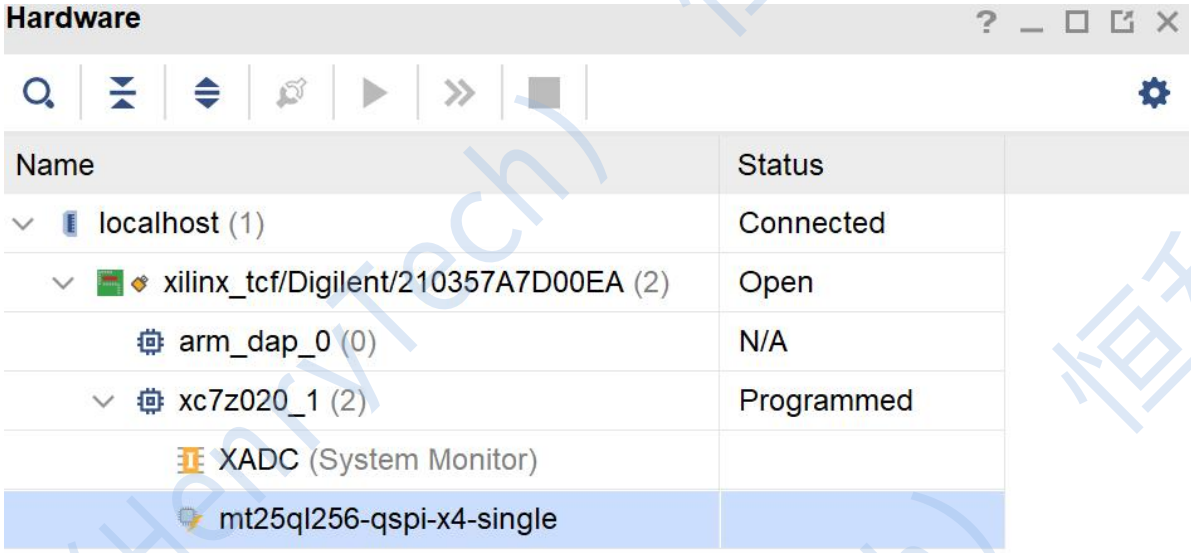
首先，参考第四节中BOOT开关设置为JTAG模式，然后连接DBG（UART/JTAG）到电脑上，打开vivaodo，临时新建一个工程或者打开任意一个Vivado工程，打开Vivado硬件管理连接。打开后点击Open Target，弹出选择Auto connect打开连接目标板：



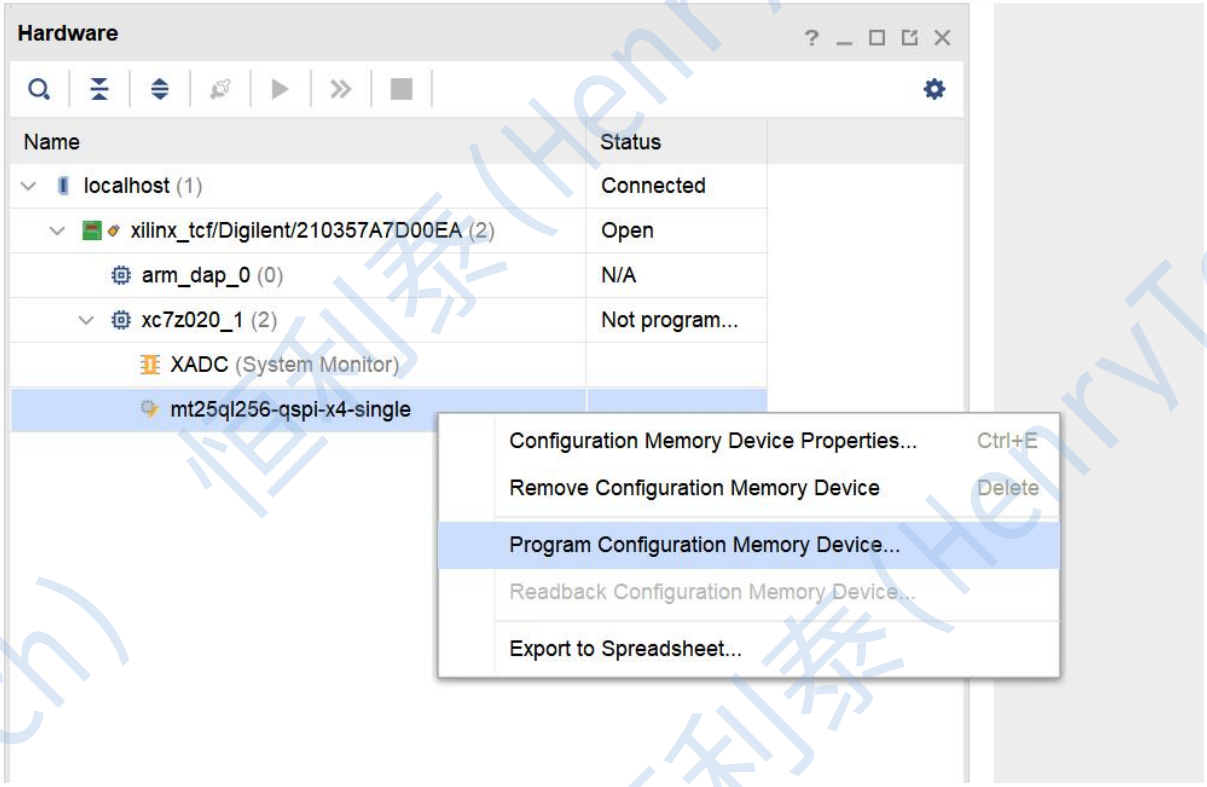
连接好识别到芯片后（列表可能显示不全可以手动拖动调整），选中xc7z020芯片右键选择Add Configuration Memory Device添加FLASH，配置如下，mt25ql256-qspi-x4-single，板载使用的是一颗MT25QL256的镁光32MByte容量的FLASH，选择后点击OK：



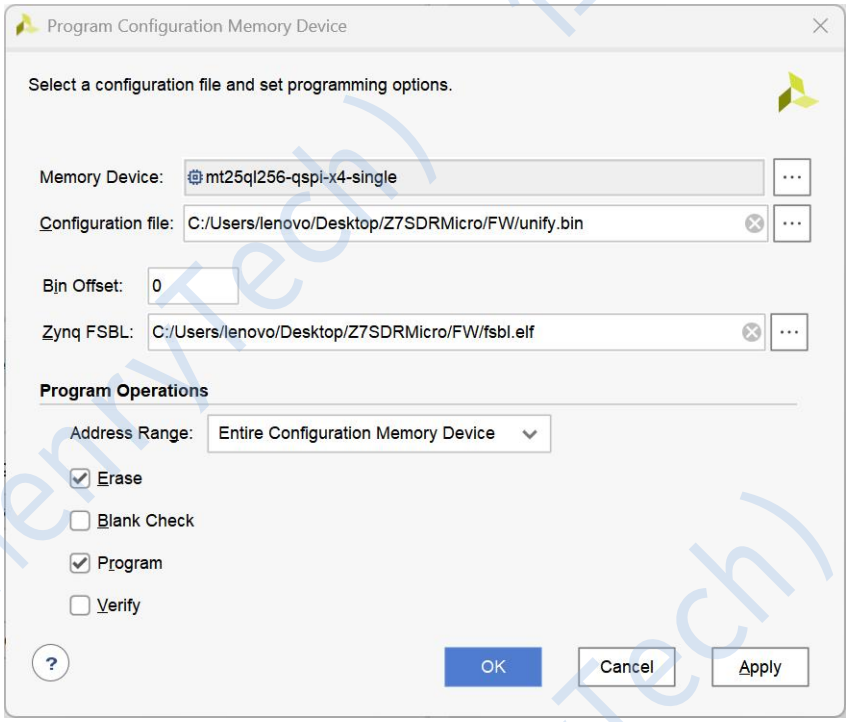
点击OK后，弹出窗口点击Cancel取消，就可以看到FLASH已经被加进来：



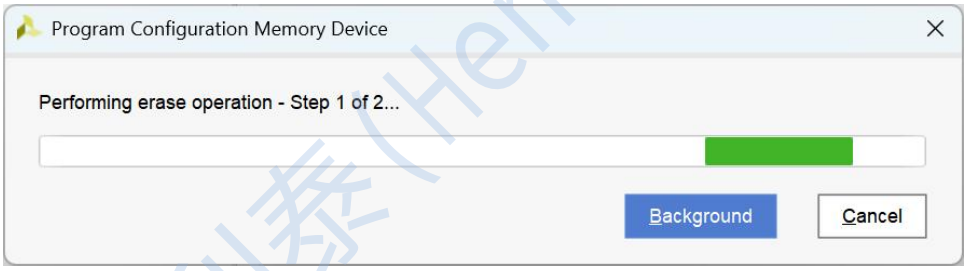
选中mt25ql256器件右键选择Program Configuration Memory Device：



然后弹出的界面中Configuration file配置文件选择刚才下载的unify.bin，Zynq FSBL文件选择刚才下载的fsbl.elf。勾选可以将Verify去掉，只保留Erase和Program，点击OK开始进行烧写：



烧写开始后，弹出进度框，一共两个步骤，先进行擦除，然后再写入：



我们等待几分钟后操作完成，我们将BOOT设置回FLASH模式，重新上电启动，即完成了出厂设置。

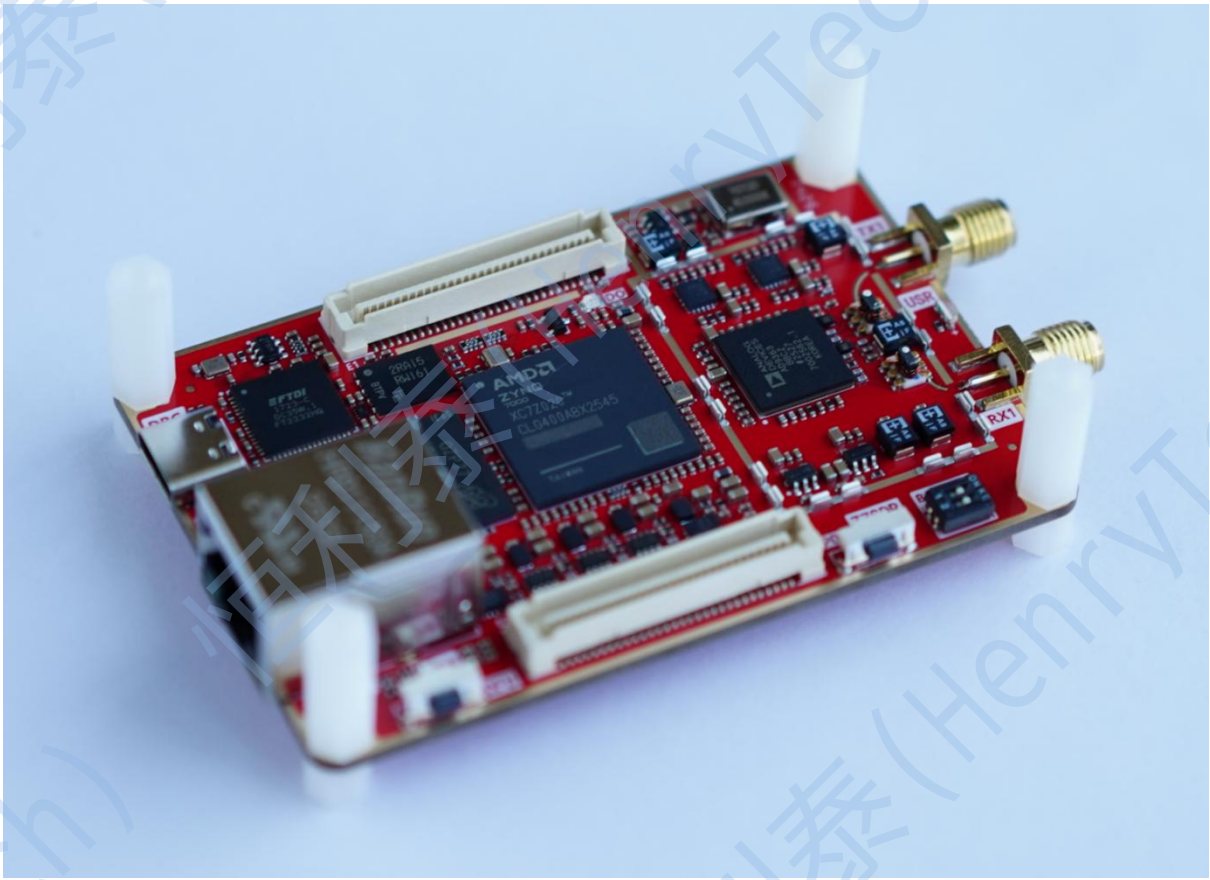
十、常见问题

1、发热：AD9363和ZYNQ工作发热量大，属于正常情况；

2、关于板载调试器：可完成FPGA\ARM全部开发。

开发软件 VIVADO2018 和 LICENSE 安装

------ 基于Z7SDRMicro开发平台



目录

一、注意事项

1)、教程使用的操作系统平台：WIN10 专业版，win7 类似，也可参考

2)、支持的系统：WINDOWS,LINUX

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 250G 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 ubuntu 的 LINUX 系统安装 VIVADO，软件效率略高于 Windows。Ubuntu 建议使用 14.4，高版本亲测安装有问题，依赖包的关系比较复杂。

4)、安装前，务必确认安装包解压到非中文路径！Vivado 不支持中文路径，安装包存放也是。

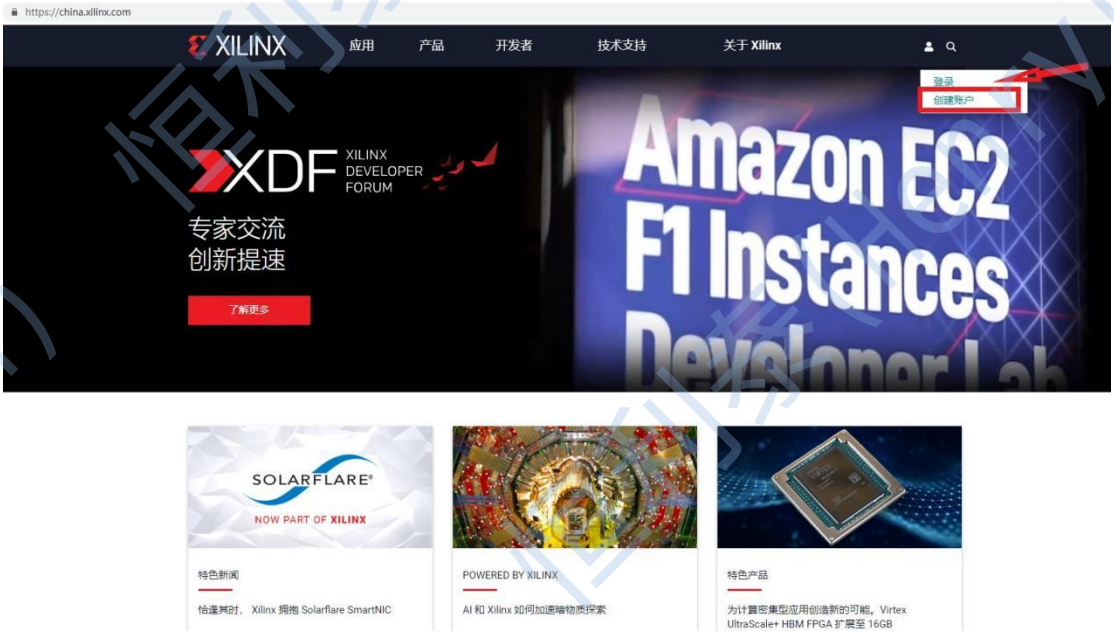
5)、安装前，务必确认 license 解压存放到非中文路径。否则会加载失败。

二、注册账户

由于 VIVADO 软件下载需要 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。

XILINX 国内网站：https://china.xilinx.com/

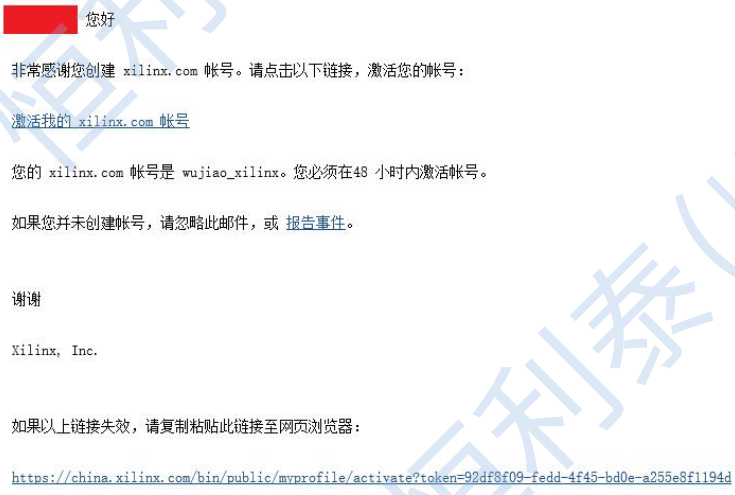
这里本文档初版较早，由早期开发板改版，这里做个说明：由于赛灵思已经被 AMD 收购，所以目前最新官网 https://www.amd.com/zh-cn.html，官网注册可能与下方不太一样，注册方法一致。大家可以以最新 AMD 页面为准。



点进去，可以看到如下注册页面，将每一行的信息填写，注意：密码需要 8 个字符，其中必须包含一个字母，一个特殊字符，一个数字。邮箱需要填写可用邮箱。如下图所示：

提交注册之后提示创建账号成功，需要打开自己的邮箱找到激活邮件根据提示激活。

打开邮箱，收到邮件如下，点击激活我的 Xilinx.com 账号，然后打开网页页面显示激活成功，即可登录。



三、安装软件

1、获取安装包

安装包获取有 3 方式，如下所示：

下载 VIVADO 2018.3 安装包，然后解压，通过以下链接：

https://china.xilinx.com/support/download.html

目录下，找到 Vivado 存档。

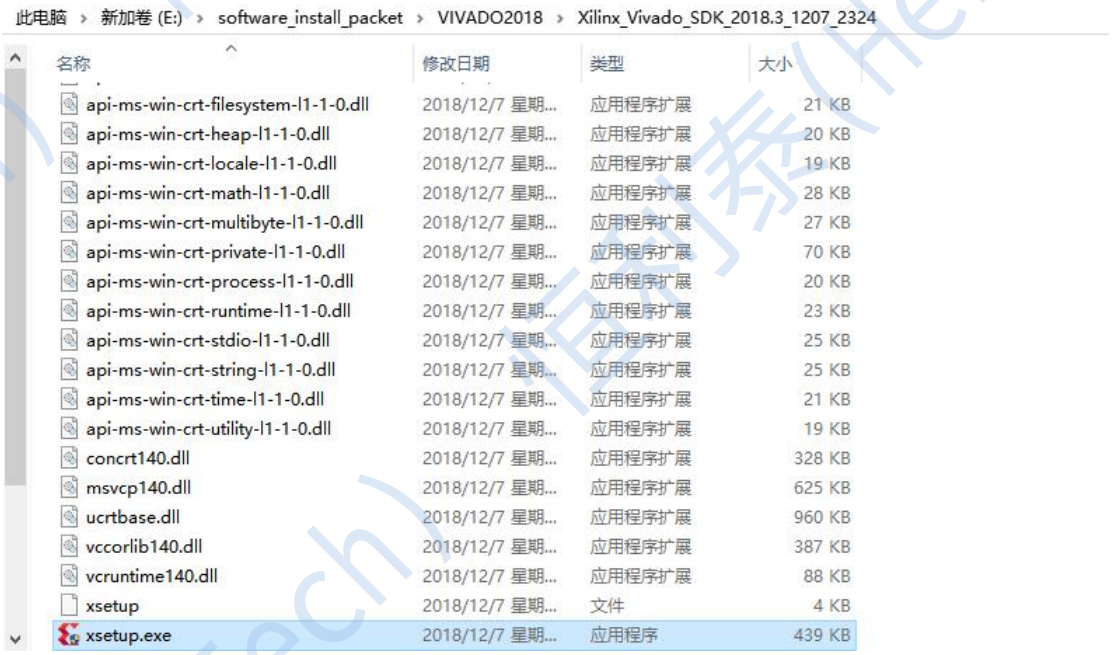
下载可以使用迅雷或者 EagleGet。迅雷可能会掉链子，建议使用 EagleGet。

或者通过我们提供的百度云网盘下载 VIVADO 2018.3。

2、安装程序

解压安装包，注意路径必须英文，不能出现中文路径。

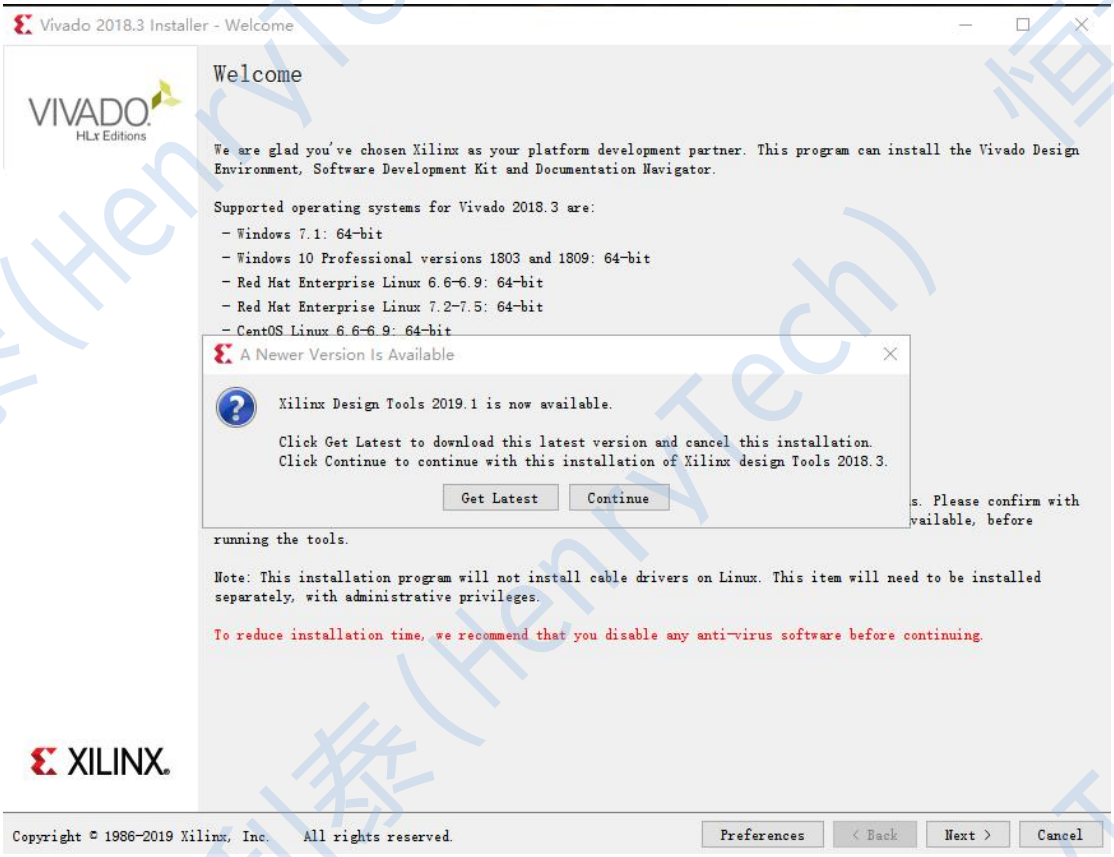
安装包大小约 18G，文件较大，建议采用 2345 好压进行解压，解压到选择需要解压到的目的文件夹。解压完成后，进入文件夹，找到 xsetup.exe



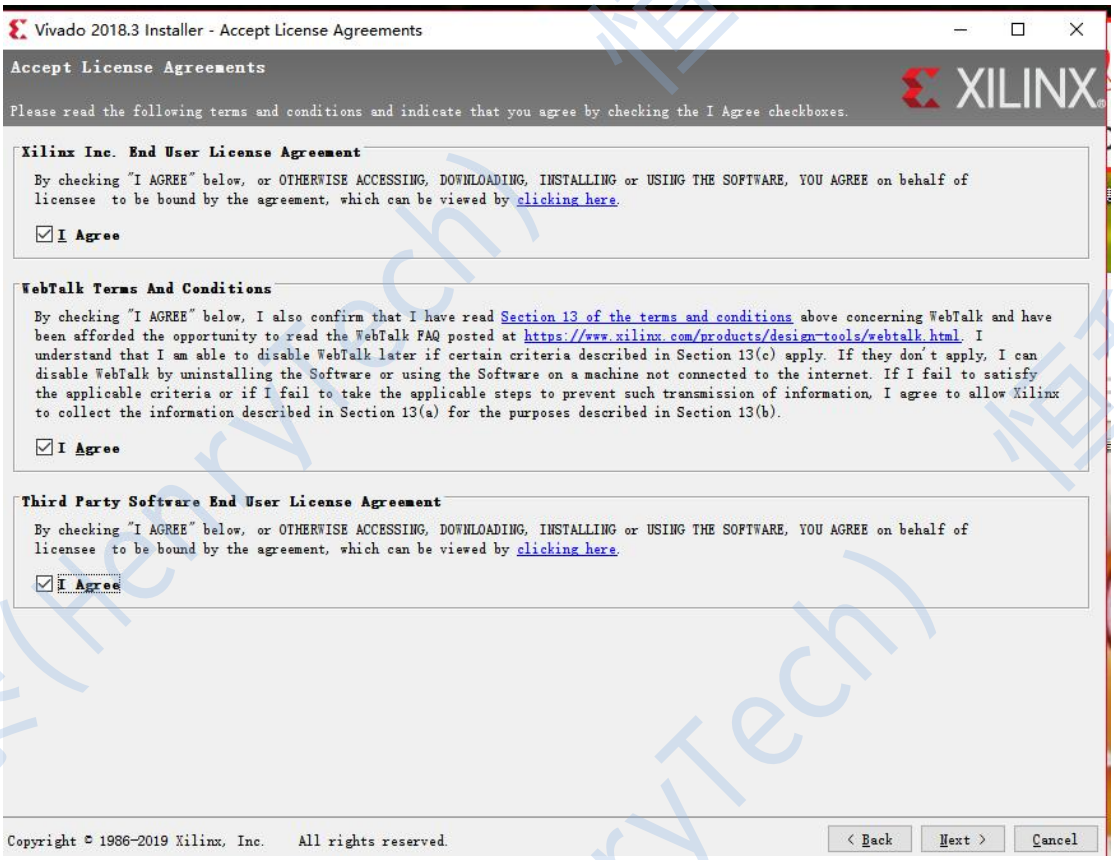
安装步骤

注：安装之前保证安装目录盘至少有 36G 可用空间，软件空间需求比较大！

双击打开（建议以管理员身份运行），弹出如下对话框，这里提示 2019.1 版本最新版可用。我们不建议使用最新版 2019.1，由于实际测试发现 2019 版本还有较多 bug 和不稳定原因（一般情况下不要使用最新版本软件，软件有待市场检验）。我们点 continue 继续，然后点击 NEXT。具体步骤如下图所示：

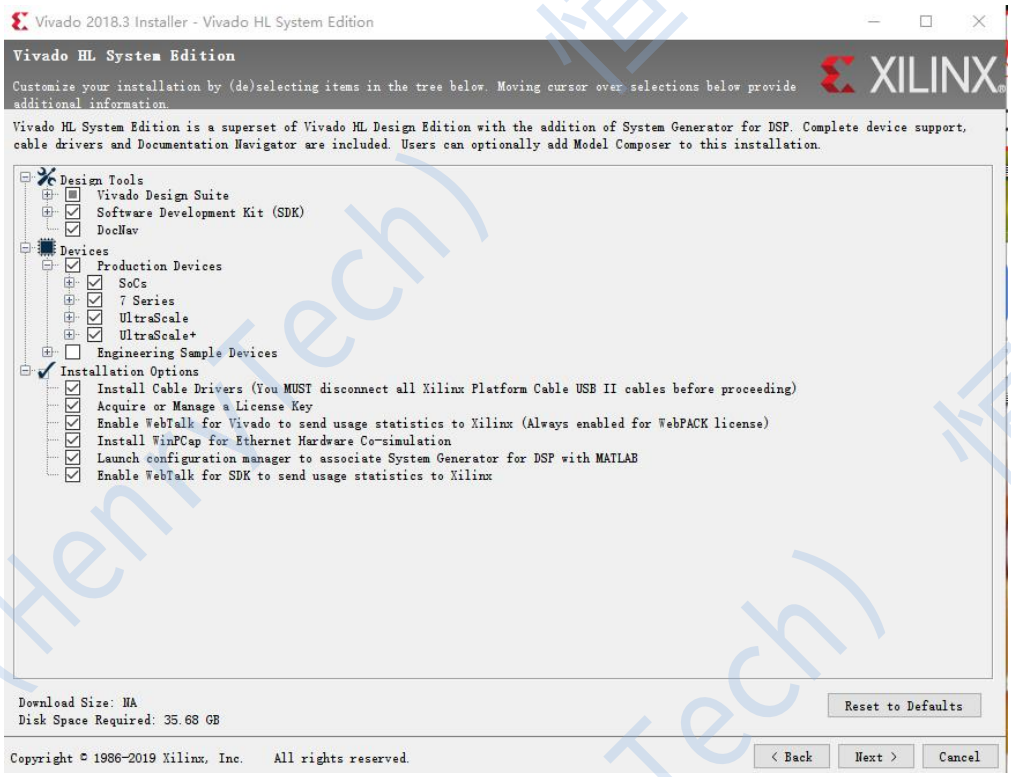


在这个界面，I Agree 全部点上，然后再点击 NEXT：

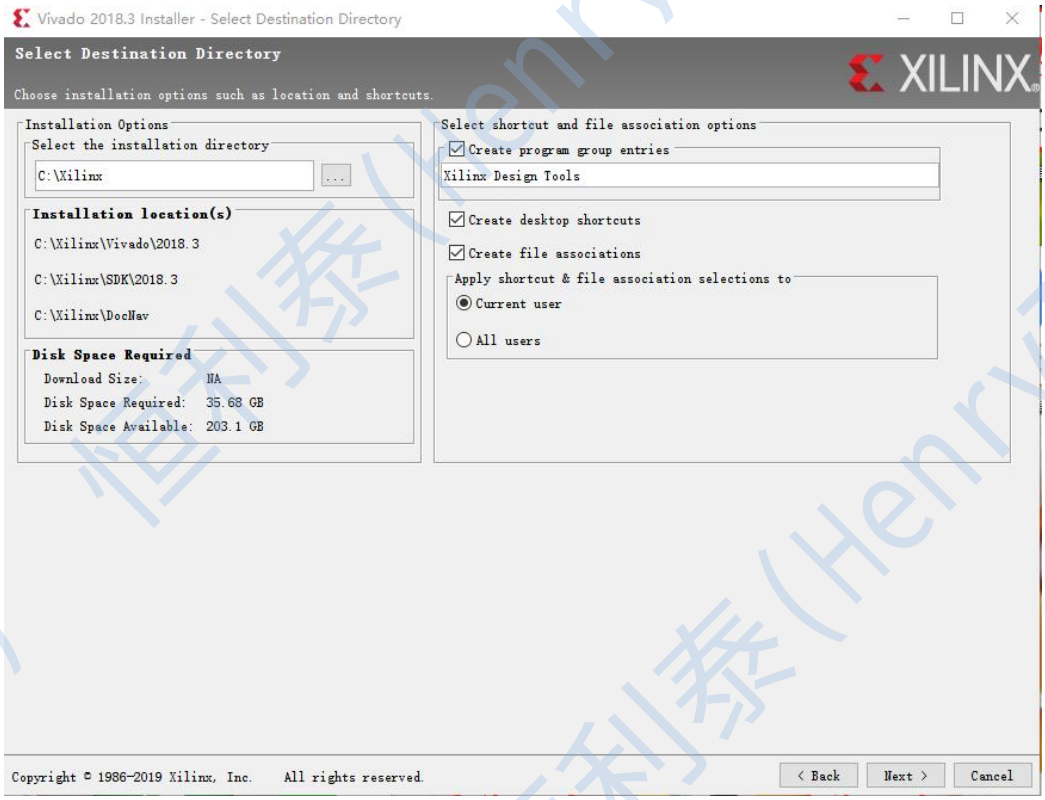


选择安装的版本，这里我们选择第 3 项，Vivado HL System Edition，然后点击 NEXT：

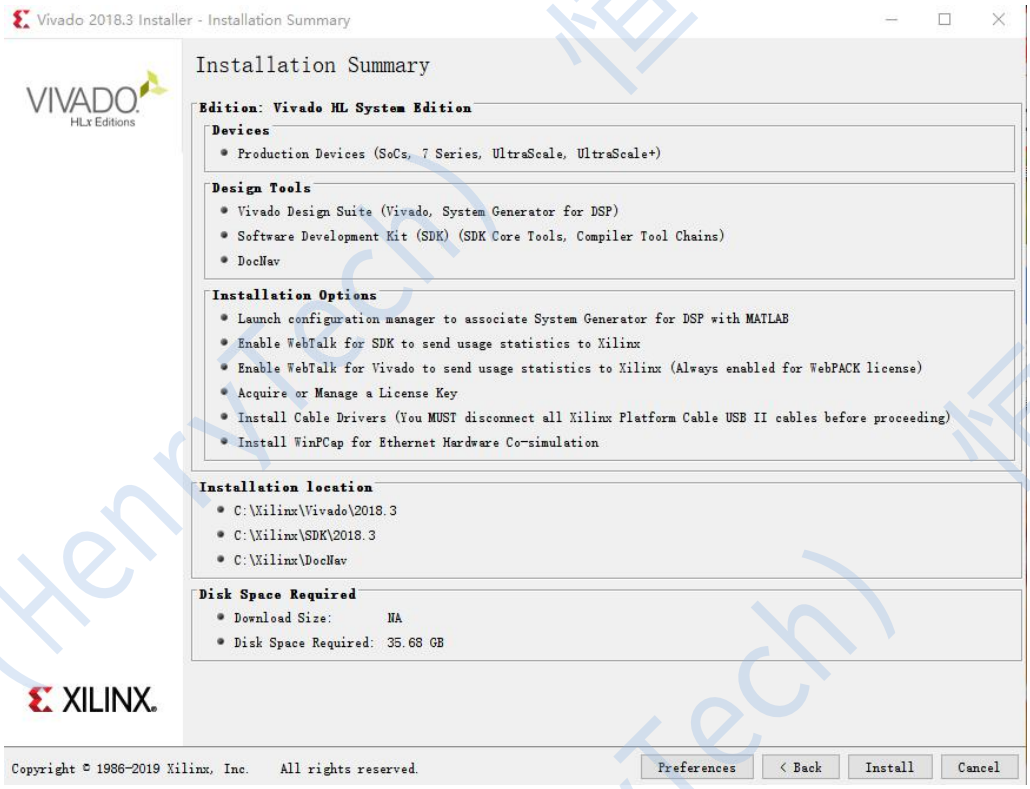
进去勾选需要安装的组件页面，默认不变，然后点击 NEXT：



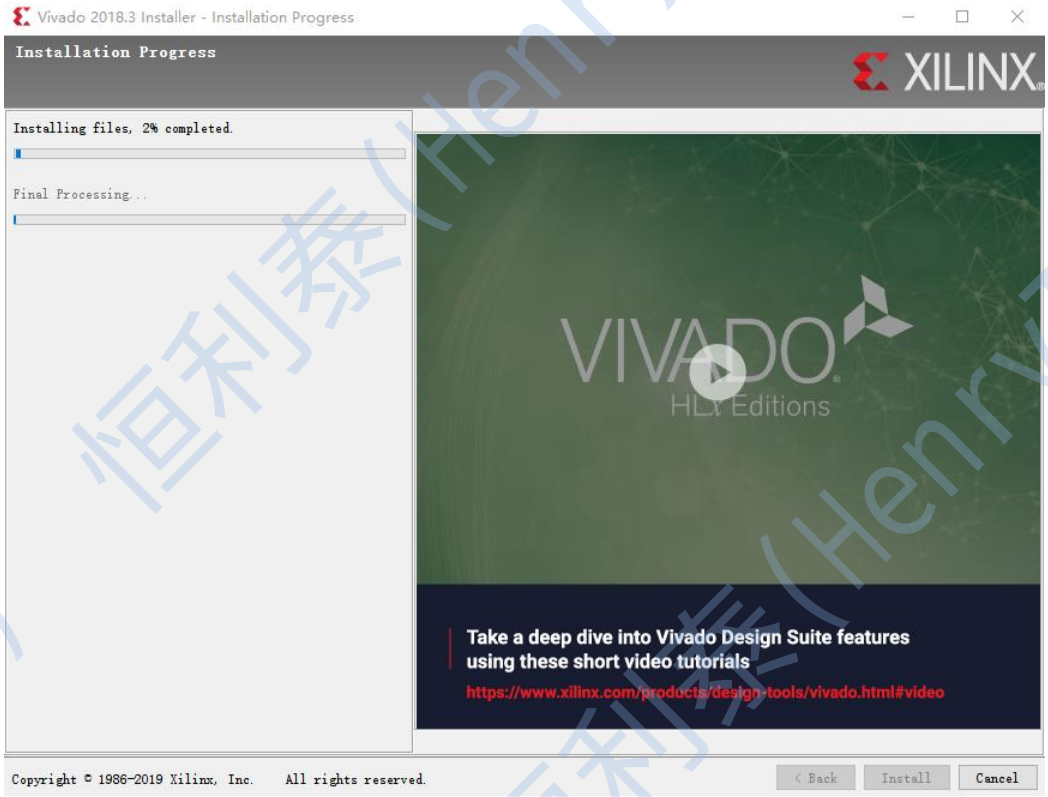
选择安装目录文件夹，默认不变，点击 NEXT：



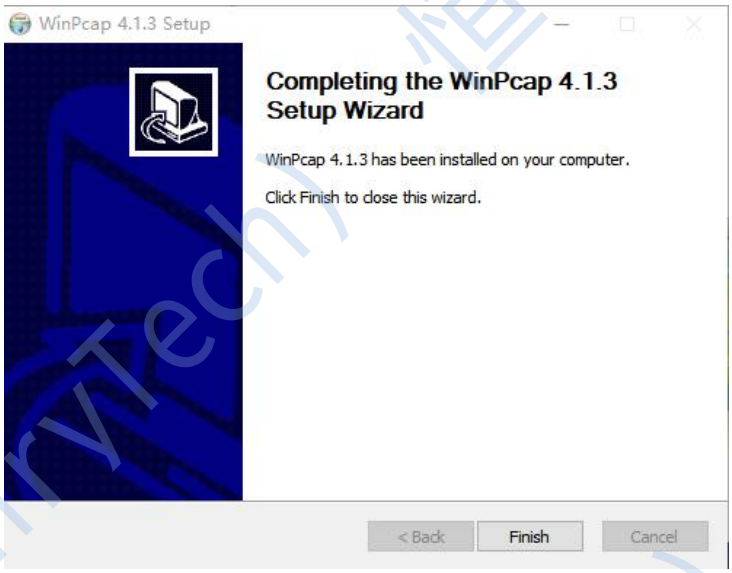
弹出如下界面，确认好安装信息，然后再点击 install：



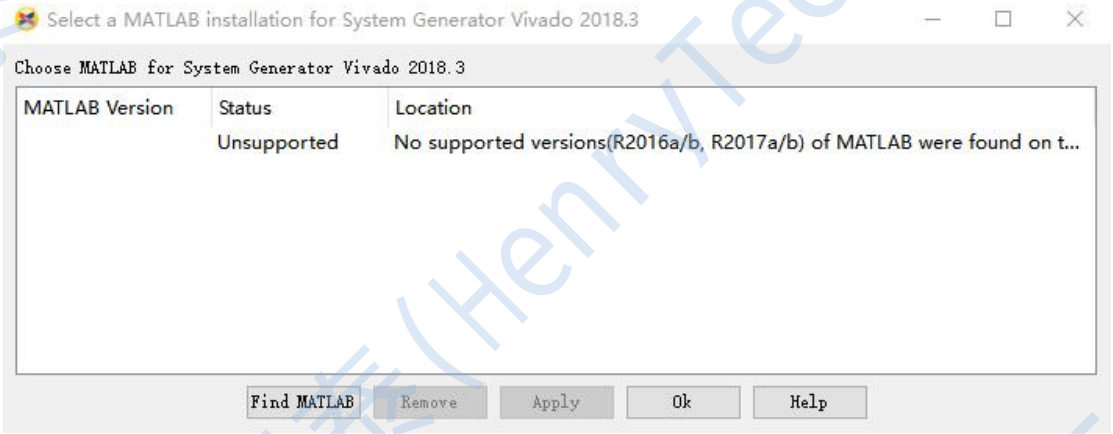
等待安装完成，时间很长，视电脑配置而变，请耐心等待。



中途弹出 WINPCAP 驱动安装界面，点击 NEXT，然后点击 I Agree，一路默认最后 finish，完成安装，安装完驱动界面：

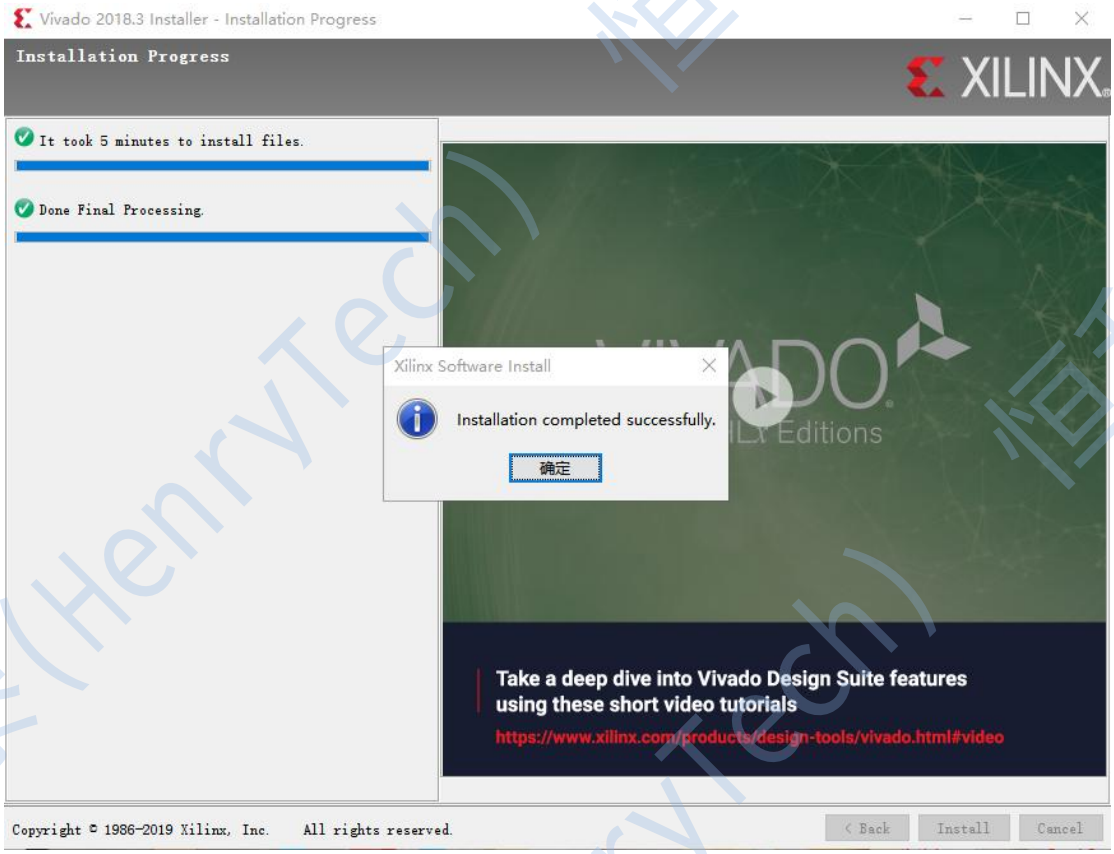


弹出 MATLAB 的支持，这里直接点 OK：



备注：如果是安装了 MATLAB 并且是 2018.3 支持的版本，会出现在里面可供选择。这里我们暂时用不到 MATLAB，我们会在后面逐渐支持更高级的开发，到时候会有新的教程详细讲解。

安装成功，VIVADO 已完成安装。

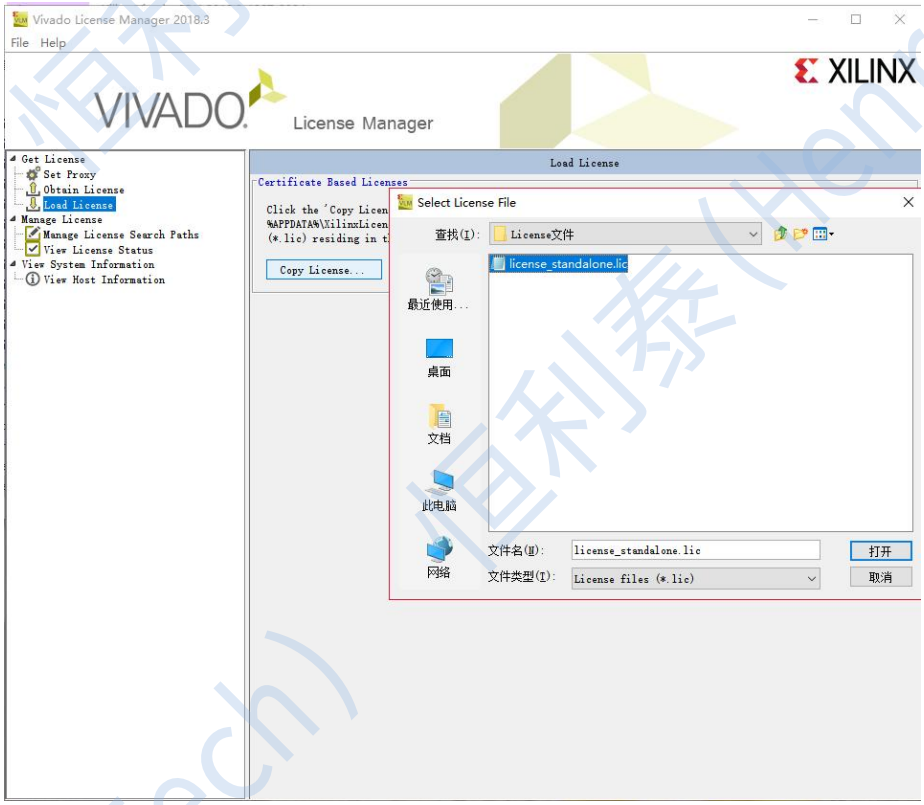


四、License 的配置

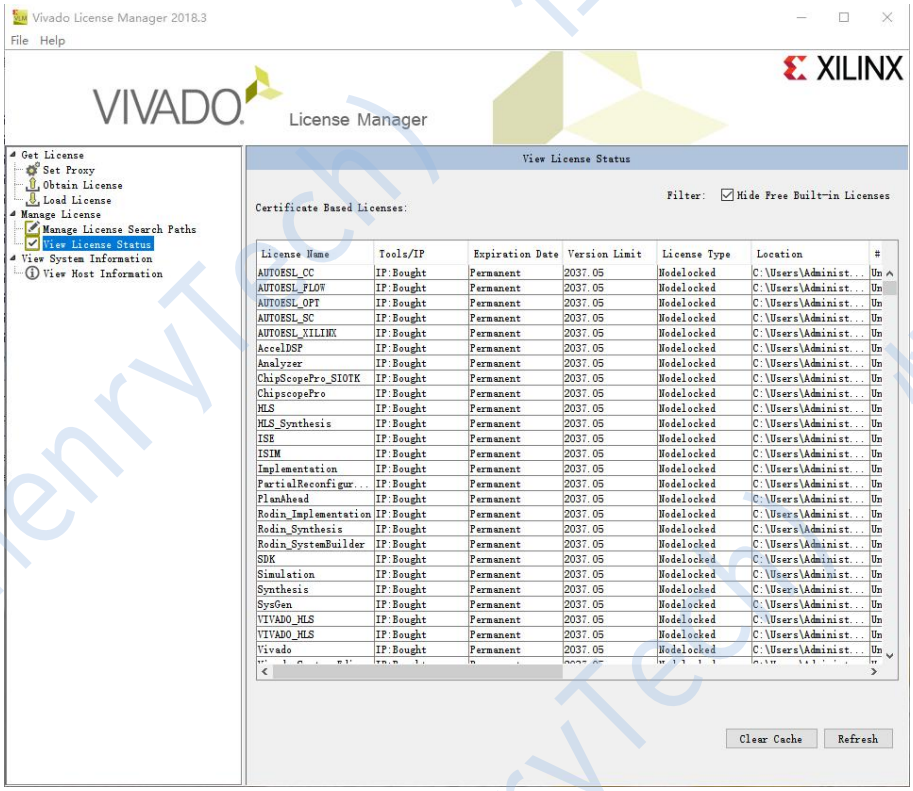
有两种方式，一种是找代理商要，另外一种，针对普通的初学者和工程师用于自学非商用，可以再 XILINX 官网注册账户，然后自己下载免费 License，基本满足学习的所有需求。

这里我们提供了几个 License 文件，附在开发板资料里，注意解压出来放到一个非中文路径中，不能出现中文路径，否则加载会失败。License 的加载界面在安装完 VIVADO 之后自动弹出，此界面是 XILINX 的 License 管理工具，如果有新的 License，均可通过这个工具加载，并且可以查看到 License 状态信息。如果后面发现开发过程中有因为 License 原因无法使用的工具和 IP，可通过此工具查看。

我们点击左边菜单栏的 Load License 然后点击右边 Copy License...，找到我们提供的 license 文件，打开。然后就加载进来了。这里截图的 licens_standalone 是我们自己账户申请的，我们给大家提供了多个 license，建议全部都添加一遍！



然后我们点击左边的菜单栏的 Viw License Status，可以看到 License 状态信息如下：



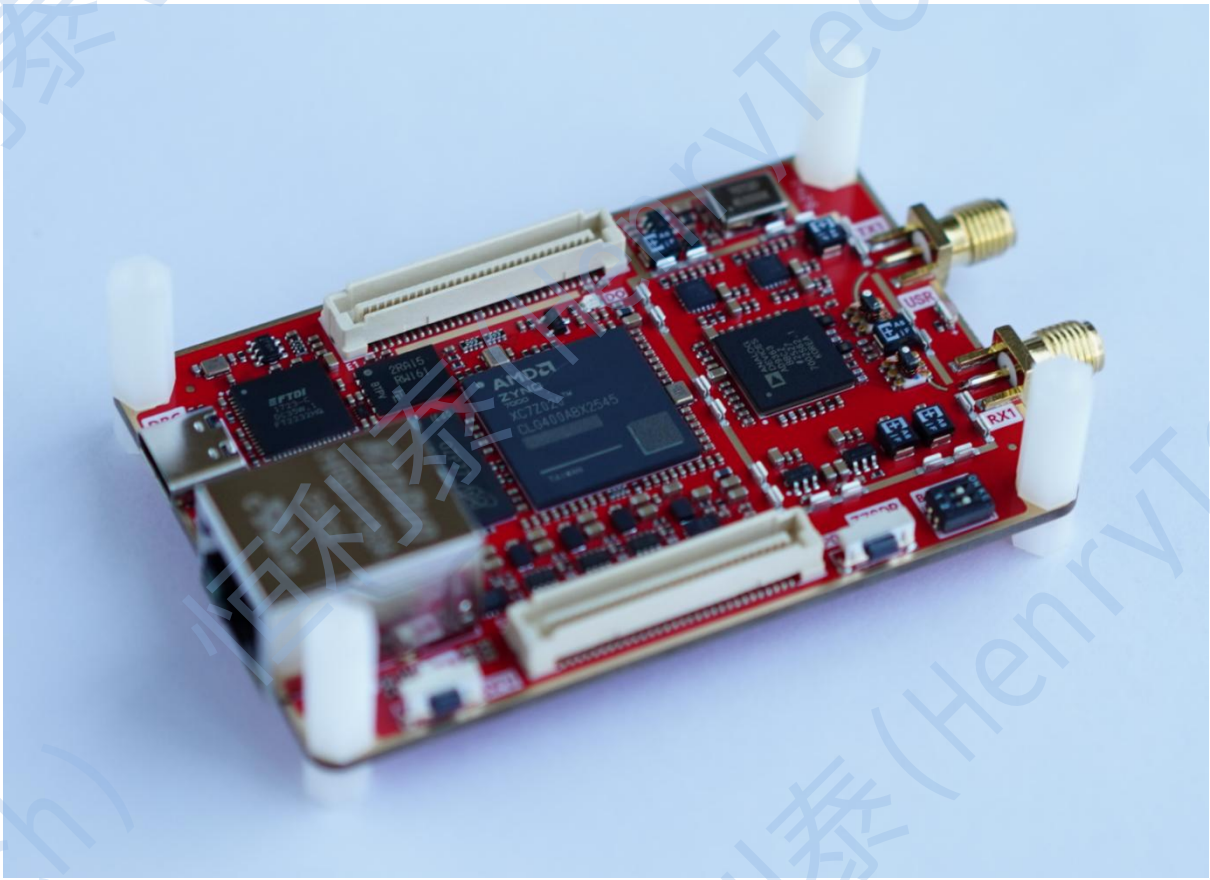
我们找到桌面的 VIVADO 图标，打开软件，到此，VIVADO 开发环境全部安装完成。

四、下载器驱动

大家根据自己使用的下载器安装厂家对应的驱动。一般 vivado 安装后会默认自动适配常见的比如 FT232,FT2232 等方案下载器驱动，如果没有自动适配，可手动安装，资料中我们提供了这两种最常用驱动，位于如下目录《PlutoSDR_Mini 软件无线电资料\01 工具软件及固件\下载器驱动》。

AD936x裸机驱动时序讲解及例程

------ 基于Z7SDRMicro开发平台



目录

二、文档介绍

在教程开始前，我们先给大家说明一下，本文档基于AD9361来讲解。AD9363与9361可以认为是这样一种关系，出厂使用同一个晶圆，实际上出货9363，测试指标范围官方是325M-3.8G频率，范围外以9361频率范围，实际使用也是支持，只不过官方参数外的官方不做任何指标上的保证。

本教程给大家介绍我们无线收发器芯片AD9361的基本功能参数，以及接口时序驱动等。并且讲解一个以Z7SDRMicro为例的裸机工程代码来给大家学习AD9361的基本使用。9361作为一款经典的无线收发芯片，被广泛应用在各个领域，尤其是作为通信设备，基站等。当然，目前5G时代基站已经使用更高规格的收发器通信,不过作为一款非常经典的射频芯片，9361依然用的很广泛。全世界范围内，无线电爱好者和一些基础的无线应用，到处都有AD9361的身影。另外，作为AD9361的姊妹型号9363，与9361一样，使用上几乎是兼容的，除了参数频率范围等不一样，性能指标差一些。类似的还有一些比如AD9364。另外目前比AD9361规格更高，参数更好的有AD9371,ADRV9009等，从采样率，灵敏度，ADC和DAC的分辨率位数，及数字接口吞吐量等都有更好的性能。不过作为初学入门，9361/9363依旧是很适合的型号。

软件无线电SDR（Software Defination Radio)是上个世纪末提出的一种无线电实现的新方案。本质是一种利用软件算法来控制实现物理无线电波的接收发送，利用无线电基础硬件，尽量多的将软件算法融合到无线通信中。软件无线电可以通过基本的硬件平台，安装不同的软件，通过软件上层进行控制驱动，并且可以快速的升级修改算法，以实现新的无线电调制通信需求。对于9361芯片，在SDR中作用是作为最底层物理层数据信号调制发送和接收的前端收发器。对于信号的输入算法处理和解析，均为上层来处理。

AD9361作为一个具有功能集成度高的无线芯片，我们为了帮助用户更快地了解它的基本功能和一些基础使用，我们编写了本文档。通过本文档可以简单的完成测试使用一个AD9361的驱动代码，以及对底层接口逻辑进行更深刻的了解。

关于adi官方的相关资源，git hub的链接如下：

https://github.com/analogdevicesinc

Git hub关于ADI的类似AD9361的资源，官方评估板的裸机代码驱动，逻辑HDL工程和库，针对linux系统的设备树和驱动库源代码等等均有。其所有资料说明git页面均有详细说明。这里我们不做讨论，大家直接参考上面的链接即可。

三、主芯片介绍

我们PlutoSDR Mini子卡采用的AD9361芯片，是由ADI（Analog Devices Inc.）亚德诺半导体公司推出的一款用于无线电收发的通用RF芯片。9361有几个衍生器件，有9361/9363/9364等。具体参数指标有所区别，大家可以到WIKI自己对比。链接如下：

https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/ad9361。

ADI官方的链接如下：

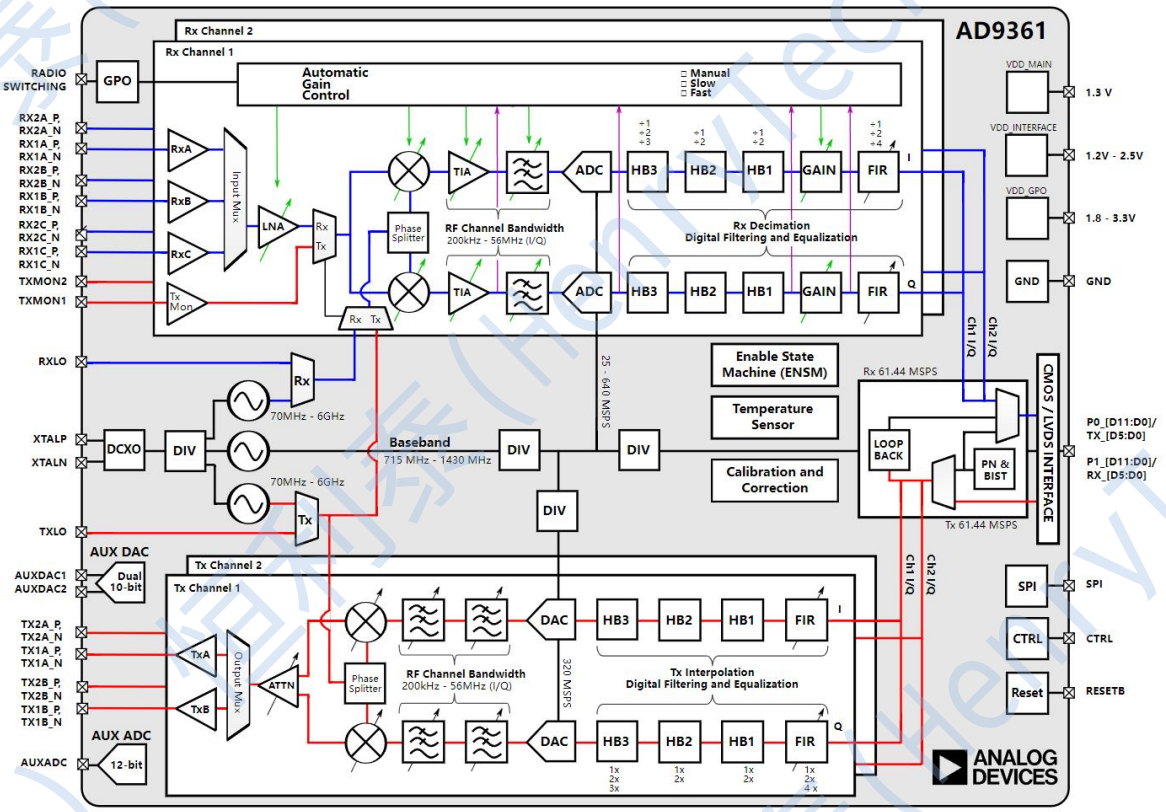
https://www.analog.com/en/products/ad9361.html

ADI官方的评估子卡FMCOMMS3链接如下：

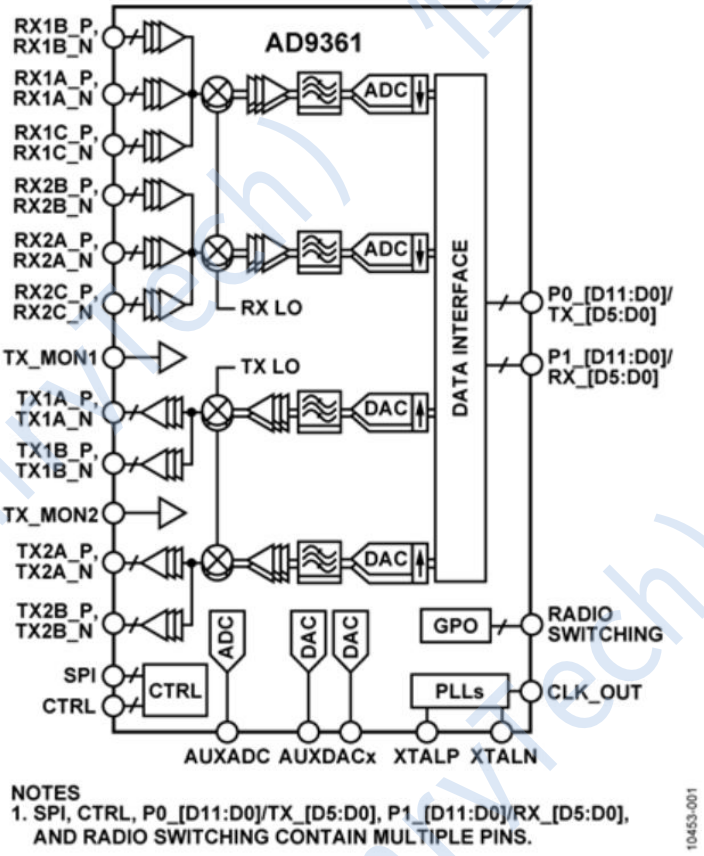
https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcomms3-ebz.html

此外ADI官方有详细的文档和数据手册，大家有需要可以自己官网下载。

其中AD9361和9363芯片型号在内部结构上除了参数指标，基本一致，所以使用的时候，9363可以使用参数范围更大性能更好的9361替代。而两者在某些应用频率参数范围基本可以PIN TO PIN替代，我们在参考设计资料的时候可以查看9363的相关配置。我们从wiki维基百科上可以看到如下一个芯片的结构图，这个图比较详细的展示了AD9361芯片内部的每个部分：



对于用户来说，我们较为关心的就是数据接收发送等接口图，开发者往往需要接口定义来进行相关时序底层驱动代码的编写，来进行使用驱动。我们在ADI官网的9361芯片页面可以找到接口功能框图：



通过上图我们可以看到，对于用户来说，9361包含2个ADC和2个DAC，分别用于接收和发送。其中接收部分每个通道ADC前端有3个输入；发送部分前端每个通道有两个输出。实际上，我们在设计的时候不需要这么多个输入和输出，每个输入输出通道我们实际只用到一个，对于我们PlutoSDR Mini，我们输入输出RX TX均只使用A。

信号接收路径：输入信号从天线进入，连接到其中一个通道的三个输入（任意一个），以RX1为例，信号可以从RX1A RX1B RX1C输入。对于前端的ABC输入，频率范围对于9361是70Mhz-6Ghz范围。注意，RX1的ABC三个输入都支持差分输入，也可以配置为单端输入。实际上使用差分可实现更好的性能，对于我们的模块子卡，使用的差分。三个输入中，如果频率低于3Ghz，每个通道的ABC三路输入性能指标基本差不多；但是如果输入频率3G-6G，为了更好的性能，推荐使用A路输入最佳，也就是RX1A,RX2A。1和2两个通道均适用这一原则。然后通过内部的解调器，输出给ADC进行采样。经过ADC之后，通过P1端口输出数据。P1端口可配置为单端或者差分。我们子卡模块默认为差分。关于TX RX引脚详细说明设计原则我们在硬件手册中有过详细讲解，大家可以进行参考。

信号发送路径：信号经过P0端口，经过DAC输出，到射频调制之后，通过TX1和TX2发送出去。TX1和TX2各有两个输出，以TX1为例，有TX1A和TX1B。我们模块只使用到TX1A。注意TX发送数据的范围与接收不同，我们发送的频率范围是47Mhz-6Ghz之间。

四、芯片时钟和辅助/控制接口以及控制接口引脚

AD9361包含一个时钟管理。我们输入时钟支持无源晶振，有源晶振（外部时钟）。我们模块默认是由一个40Mhz/0.5ppm的TCXO晶振，给芯片提供40Mhz时钟。关于时钟，在9361的驱动嵌入式软件代码中可以配置。

AD9361包含辅助ADC和DAC功能。该功能可以帮助用户实现一个监测和控制功能。具体这里不做讲解，有兴趣想要更详细的说明可以参阅9361的数据手册。

9361的控制接口有很多个信号，其中有一个SPI接口，可用于9361初始化配置，这个是最重要的接口，由于9361参数和所有初始化均为SPI进行。另外还有4个控制输入信号和8个用于状态检测的输出信号引脚。

另外还有一系列的其他控制信号，比如说ENABLE使能信号，EN_AGC使能增益，RESETN复位信号，CLKOUT时钟输出，SYNC_IN芯片间同步信号等等，大家具体参阅AD9361的数据手册的引脚和信号注释部分详细了解。官方手册9361的16-20页附图了芯片的BGA视图下的每一个引脚的标注图，以及列表清单对整个芯片每一个引脚的详细阐述说明，大家设计9361的硬件也是基于这部分说明：

	1	2	3	4	5	6	7	8	9	10	11	12
A	RX2A_N	RX2A_P	NC	VSSA	TX_MON2	VSSA	TX2A_N	TX2A_P	TX2B_N	TX2B_P	VDDA1P1_TX_VCO	TX_EXT_LO_IN
B	VSSA	VSSA	AUXDAC1	GPO_3	GPO_2	GPO_1	GPO_0	VDD_GPO	VDDA1P3_TX_LO	VDDA1P3_TX_VCO_LDO	TX_VCO_LDO_OUT	VSSA
C	RX2C_P	VSSA	AUXDAC2	TEST/ENABLE	CTRL_IN0	CTRL_IN1	VSSA	VSSA	VSSA	VSSA	VSSA	VSSA
D	RX2C_N	VDDA1P3_RX_RF	VDDA1P3_RX_TX	CTRL_OUT0	CTRL_IN3	CTRL_IN2	P0_D9/TX_D4_P	P0_D7/TX_D3_P	P0_D5/TX_D2_P	P0_D3/TX_D1_P	P0_D1/TX_D0_P	VSSD
E	RX2B_P	VDDA1P3_RX_LO	VDDA1P3_TX_LO_BUFFER	CTRL_OUT1	CTRL_OUT2	CTRL_OUT3	P0_D11/TX_D5_P	P0_D8/TX_D4_N	P0_D6/TX_D3_N	P0_D4/TX_D2_N	P0_D2/TX_D1_N	P0_D0/TX_D0_N
F	RX2B_N	VDDA1P3_RX_VCO_LDO	VSSA	CTRL_OUT6	CTRL_OUT5	CTRL_OUT4	VSSD	P0_D10/TX_D5_N	VSSD	FB_CLK_P	VSSD	VDDD1P3_DIG
G	RX_EXT_LO_IN	RX_VCO_LDO_OUT	VDDA1P1_RX_VCO	CTRL_OUT7	EN_AGC	ENABLE	RX_FRAME_N	RX_FRAME_P	TX_FRAME_P	FB_CLK_N	DATA_CLK_P	VSSD
H	RX1B_P	VSSA	VSSA	TXNRX	SYNC_IN	VSSA	VSSD	P1_D11/RX_D5_P	TX_FRAME_N	VSSD	DATA_CLK_N	VDD_INTERFACE
J	RX1B_N	VSSA	VDDA1P3_RX_SYNTH	SPI_DI	SPI_CLK	CLK_OUT	P1_D10/RX_D5_N	P1_D9/RX_D4_P	P1_D7/RX_D3_P	P1_D5/RX_D2_P	P1_D3/RX_D1_P	P1_D1/RX_D0_P
K	RX1C_P	VSSA	VDDA1P3_TX_SYNTH	VDDA1P3_BB	RESETB	SPI_ENB	P1_D8/RX_D4_N	P1_D6/RX_D3_N	P1_D4/RX_D2_N	P1_D2/RX_D1_N	P1_D0/RX_D0_N	VSSD
L	RX1C_N	VSSA	VSSA	RBIAS	AUXADC	SPI_DO	VSSA	VSSA	VSSA	VSSA	VSSA	VSSA
M	RX1A_P	RX1A_N	NC	VSSA	TX_MON1	VSSA	TX1A_P	TX1A_N	TX1B_P	TX1B_N	XTALP	XTALN

ANALOG I/O　DC POWER
DIGITAL I/O　GROUND
NO CONNECT

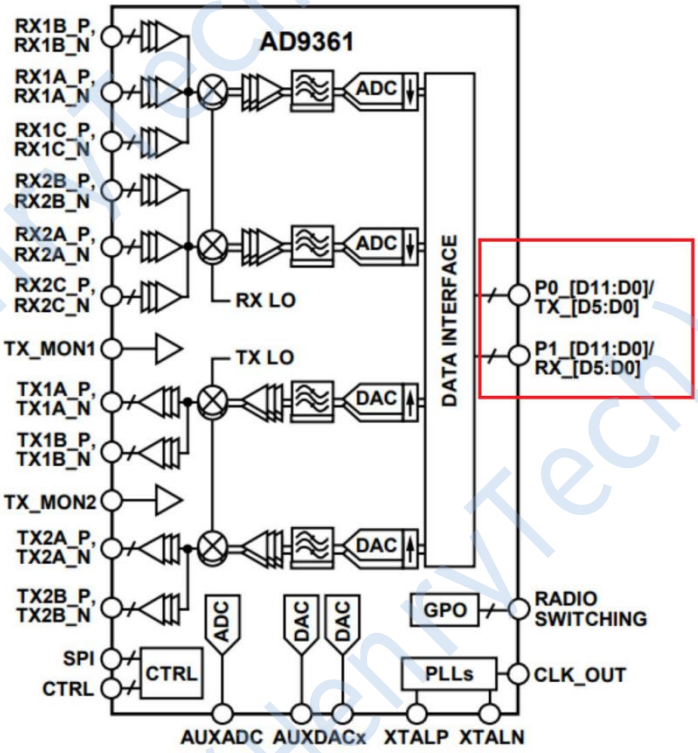
10453-002

Figure 2. Pin Configuration, Top View

关于引脚阐述由于篇幅太多，这里不再截图。大家直接参阅AD9361数据手册的16-19页table3.

五、数据收发的数据接口

从官方页面上的接口框图右边的数据接口，有P0 P1两个端口。其中P0是TX的发送接口，P1是数据集接收端口。我们整个无线发送接收的原始数据都通过这两个端口进出。



我们接下来重点讲解P0和P1端口。对于FPGA技术人员来说最关心的就是数据交互接口，我们需要了解接口的数据收发时序。

AD9361支持两种模式的接口，一种是单端的CMOS另一种是LVDS差分。两种模式下都可以通过SPI接口进行配置。区别就是时序有所差异，PCB设计走线也是不一样的。CMOS的单端模式下，P0 P1接口以12位并口传输数据；LVDS模式下，为6对差分，通过时钟上升沿下降沿均传输6位数据，然后通过DDR原语，最终拼接12为数据，其中注意，LVDS模式下，时钟和同步信号均为差分信号,并且接口需要使用一些原语来进行差分转单端或者单端转差分输出等处理。而CMOS单端模式下，我们除了P0 P1的并行数据接口，时钟和数据同步FRAME引脚有N和P极性，我们只需要P极性引脚，N忽略悬空。LVDS接口时序稍微复杂一些，不过熟悉了接口时序也不难。我们PlutoSDR Mini采用单端走线，电平使用LVCMOS1.8V标准。很多朋友可能会问，为何不支持差分？我们从原理上看，差分是一对并行在一起走线的信号，为了抗干扰；但是，官方PLUTO项目标准设计的HDL源码采用标准LVCMOS单端接口，单端信号跑差分走线，必然会相邻信号串扰，导致信号干扰问题。所以，不采用差分走线。另外单端走线不能使用差分电平标准，违背了差分的设计初衷，抗干扰能力下降，反而更容易引入干扰。在CMOS单端设计走线条件下，我们使用CMOS引脚接口模式。

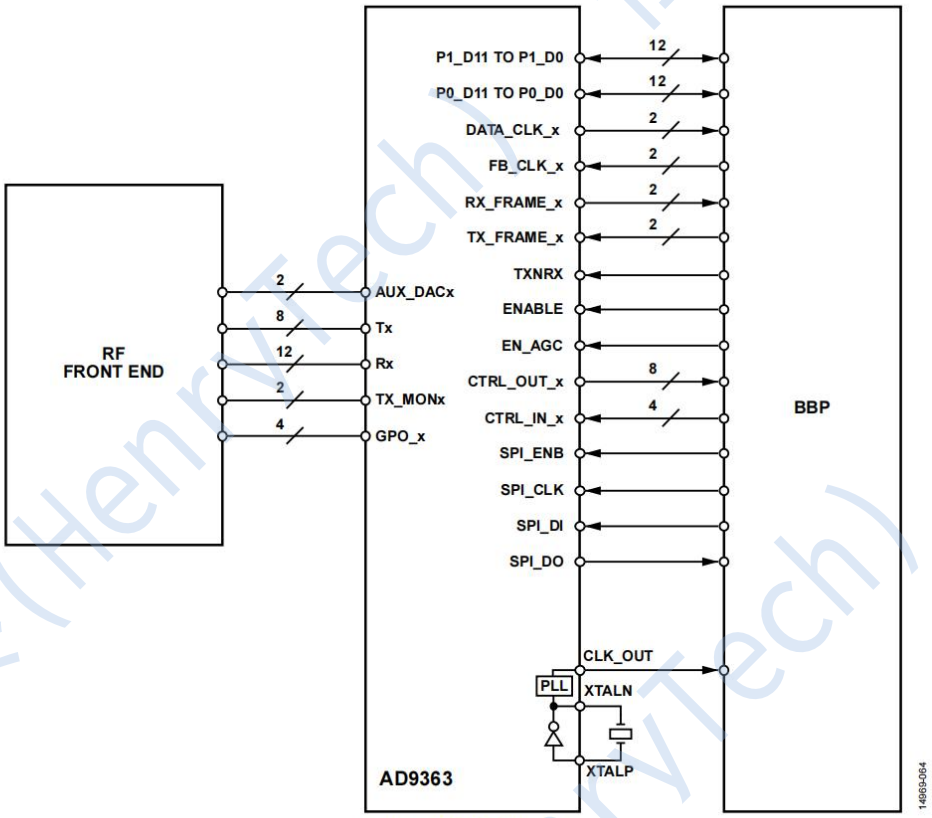


Figure 58. AD9363 Interface

我们查阅UG1040文档《AD9363-Reference-Manual-UG-1040.pdf》，第74页，我们看到上图就是芯片9361与RF前端连接以及BBP（控制器，这里可以认为是FPGA）。数据接口支持的CMOS单端和LVDS模式下，9361支持两种数据信号传输模式，一个叫做FDD(Frequency Division Duplexing)，这里如果简单的理解可以看做通信里的全双工，另一个叫做TDD(Frequency Division Duplexing)可以看做是半双工。前者发送接收可以同时进行；但是TDD模式下，数据发送接收在不同的时间进行，并且时间上错开，避免干扰影响，也就是单位时间只能发送或者接收，有点类似于单车道，交错出行。

关于接口传输数据，我们引入一个叫做DDR的概念。所谓DDR，就是数据在时钟上升沿和下降沿均传输数据，这样就可以使得一根信号可以达到两倍时钟的带宽速度。对于DDR，我们仅仅在时钟上升沿或者下降沿传输数据的接口，我们称之为SDR。SDR相对于DDR来说只有一半带宽，并且SDR数据速率与时钟频率一致。我们接收和发送端口均使用DDR模式。常见的DDR有输入和输出之分。用于输入信号双边沿转换，使用IDDR原语;用于输出信号做双边沿信号发送，使用ODDR原语。原语的概念这里就不多说，类似于一个FPGA内部底层的IO硬件单元。调用类似IP或者模块例化，但是实际上找不到对应的源代码，综合器在综合及布局布线的时候直接当最基础的元器件逻辑单元处理。

六、CMOS单端模式下的接口信号及时序

我们接口包含的信号，在单端CMOS模式下与FPGA的接口连接,工作在TDD（半双工）模式下，连接如下图所示，我们这里均讨论双端口P0 P1均使用的模式：

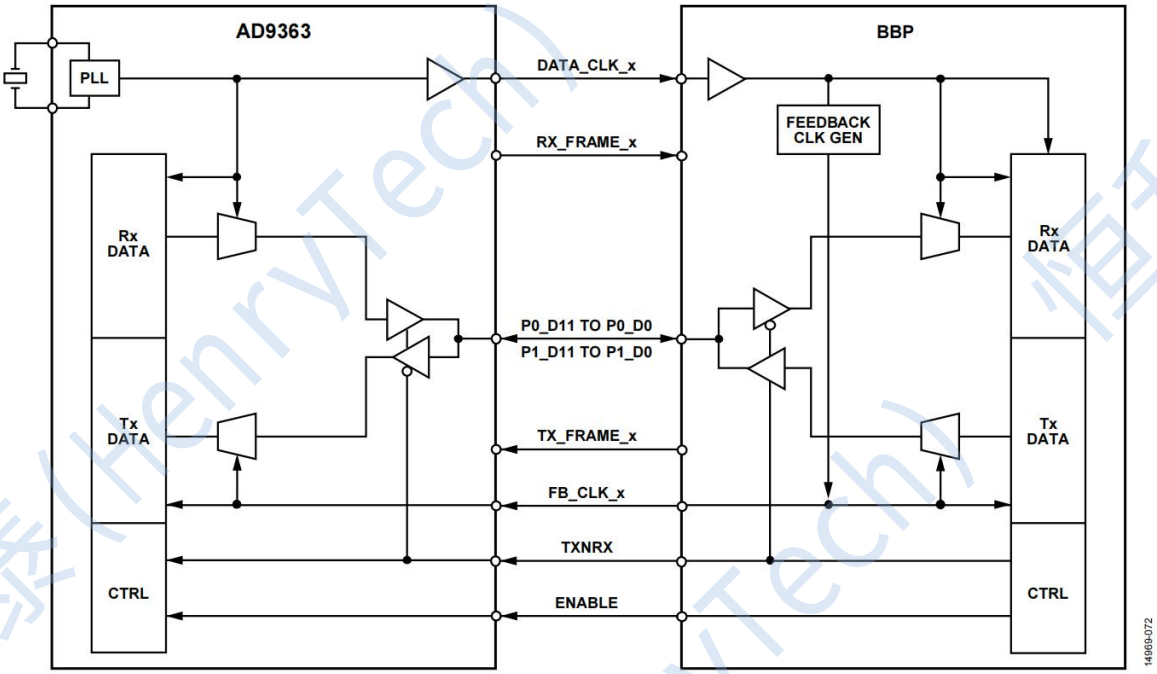


Figure 65. Dual-Port TDD Mode

在FDD（全双工）模式下连接如图所示：

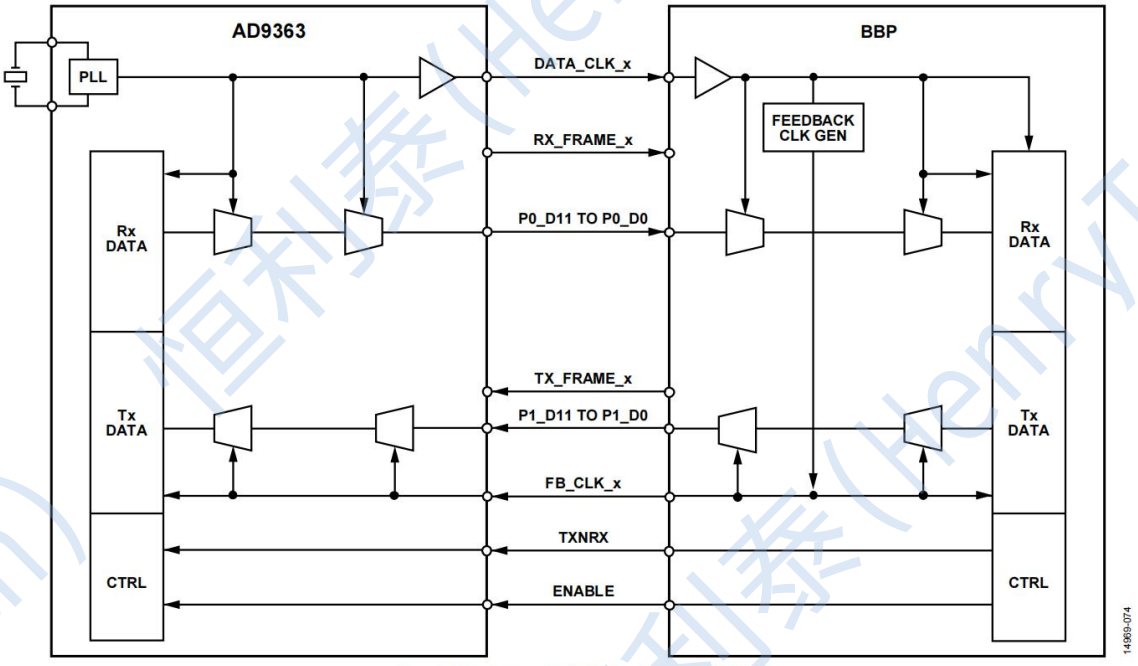


Figure 68. Dual-Port, Full Duplex Mode (Full Port)

其中我们这里讲解一下上图对应标注连接的信号。

- DATA_CLK_x：9361输出的时钟，给BBP(FPGA)，并且是作为数据接收的主时钟，频率由9361的配置寄存器决定；单端模式下，仅使用DATA_CLK_P，N悬空不使用，忽略。
- RX_FRAME_x：9361输出作为数据同步的有效信号，给BBP(FPGA)，指示当前总线数据有效，单端模式下仅使用RX_FRAME_P,N不使用，悬空即可，忽略。
- P0:TDD模式下，D0-D11为RX接收和发送数据公用。FDD下作为RX接收数据接口；

- P1:TDD模式下，D0-D11为RX接收和发送数据公用。FDD下作为TX接收数据接口；
- TX_FRAME_x：9361的发送同步的有效信号，由BBP(FPGA)给9361芯片，用作数据的发送有效和同步指示。单端模式下仅使用TX_FRAME_P,不使用N，悬空即可，忽略；
- FB_CLK_x:这个时钟用于发送数据，从BBP（FPGA）给到9361用于同步数据发送端的时钟信号。与接收的DATA_CLK_x类似。只不过一个是接收时钟一个是发送时钟。
- TXNRX:用于TDD模式下的分时同步信号，主要用于使能状态机控制信号，用于控制数据信号的方向，是做输入还是输出，更直接点表示就是，当前应该发送还是接收。FDD全双工模式下忽略这个。
- ENABLE：AD9361内部有一个使能状态机（ENSM），结合TXNRX和ENABLE控制实时状态。

我们从UG1040《AD9363-Reference-Manual-UG-1040.pdf》上可以看到，我们的9361在单端模式下存在两种工作方式，我们之前讲过，AD9361内部有两个无线收发通道，这两个物理上基本独立，所以引申出1R1T，2R2T两种模式分别表示一收一发，2收2发。我们对应子卡，1R1T仅使用RX1和TX1接口，仅使用一路数据收发；在2R2T下，我们RX1 RX2 TX1 TX2四个接口都用到了，可以同时进行两路数据收发。在CMOS单端模式下可以达到的最大数据采样和发送的带宽如下：

Table 47. Maximum Data Rates (SDR and DDR) and Signal Bandwidths

	1R1T Configurations						1R2T/2R1T/2R2T Configurations					
	Maximum Data Rate (Combined I and Q Words)		Maximum RF Channel Signal Bandwidth				Maximum Data Rate (Combined I and Q Words)		Maximum RF Channel Signal Bandwidth (Per Channel)			
			Using Minimum Sample Frequency		Using 2× Oversampling				Using Minimum Sample Frequency		Using 2× Oversampling	
Operating Mode	SDR (MSPS)	DDR (MSPS)	SDR Bus (MHz)	DDR Bus (MHz)	SDR Bus (MHz)	DDR Bus (MHz)	SDR (MSPS)	DDR (MSPS)	SDR Bus (MHz)	DDR Bus (MHz)	SDR Bus (MHz)	DDR Bus (MHz)
Single Port												
Half Duplex	30.72	61.44	30.72	56[1]	15.36	30.72	15.36	30.72	15.36	30.72	7.68	15.36
Full Duplex	15.36	30.72	15.36	30.72	7.68	15.36	7.68	15.36	7.68	15.36	3.84	7.68
Dual Port												
Half Duplex	61.44	122.88	56[1]	56[1]	30.72	56[1]	30.72	61.44	30.72	56[1]	15.36	30.72
Full Duplex	30.72	61.44	30.72	56[1]	15.36	30.72	15.36	30.72	15.36	30.72	7.68	15.36

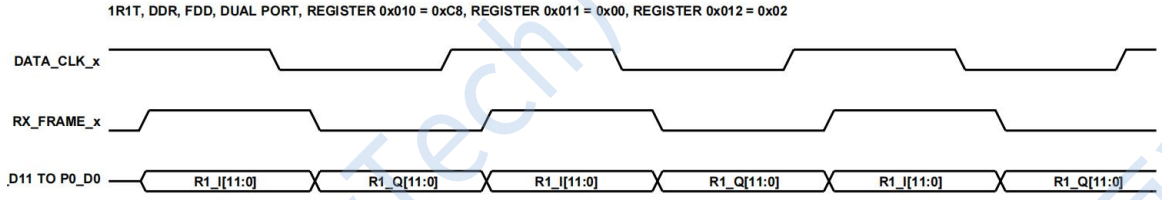
[1] Limited by the analog filter bandwidth.

上述表格中可以看到，对应的CMOS的单端引脚可以工作在SDR(时钟单边沿比如上升沿)采样模式，以及DDR(始终上升沿下降沿都采样)双边传输模式。上图中的Half duplex和Full duplex模式可以对应我们TDD和FDD模式。可以从表格看出，我们9361最大数据带宽FDD（全双工）模式可以达到56M，数据接口频率可以达到61.44M。这个是针对1R1T一收一发；对于2R2T模式下，最大数据带宽在FDD（全双工）条件下仅30.72Mhz带宽。所以我们实际使用的情况下合理选择工作模式接口很重要。

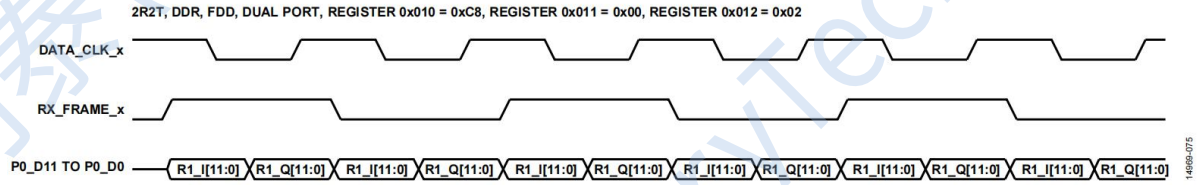
我们对于本模块的单端CMOS接口方式，后续所有讨论均以FDD全双工模式进行讲解，TDD半双工不做讲解，具体自己参阅UG1040《AD9363-Reference-Manual-UG-1040.pdf》。对于FDD全双工P0和P1同时使用的模式下，我们讲解一下UG1040手册的83页。大家除了本文档，这个1040手册是9361讲解接口时序和参数等技术细节最详细的一手资料。

单端CMOS在1R1T模式下的FDD全双工工作接口的接收RX时序。这个模式下，数据在RX_FRAME=1的时候，传输一个数据的I路调制信号；在RX_FRAME=0的时候传输Q路调制信号。其

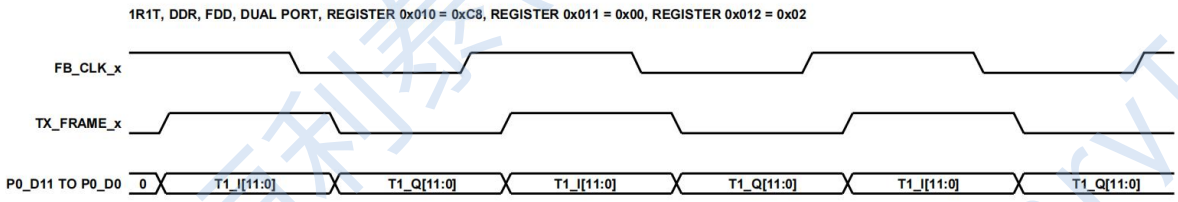
中I数据和Q数据分别在DATA_CLK_x时钟的下降沿和上升沿采样。注意，我们9361虽然只有最多2收2发2T2R，实际上期中一个通道可以包含两个分量的数据，芯片内部使用了正交调制，输出空间90度相位的两路数据，我们称之为I分量数据和Q分量数据（或者D数据和Q数据）：



单端CMOS在2R2T模式下的全双工工作接口的接收RX时序。这个模式下，当RX_FRAME=1时，时钟下降沿传输第一路数据的I分量；上升沿传输Q路的分量；RX_FRAME=0的时候，传输我们的第二路数据，时钟下降沿为第二路的I分量，上升沿为Q分量。注意，这里可能是官方UG1040手册编写问题，RX_FRAME=0的对应数据应该标注为R2表示第二路。



单端CMOS在1R1T模式下的全双工工作接口的发送TX时序。这个模式下，TX_FRAME=1的时候输出一个通道的I分量；TX_FRAME=0的时候输出Q分量；I分量在时钟下降沿有效，Q在时钟上升沿有效：



单端CMOS在2R2T模式下的全双工工作接口的发送TX时序。TX_FRAME=1的时候，时钟下降沿输出第一通道的I分量；上升沿输出Q分量。RX_FRAME=0的时候，时钟下降沿输出第二路的I分量，上升沿输出第二路的Q分量：

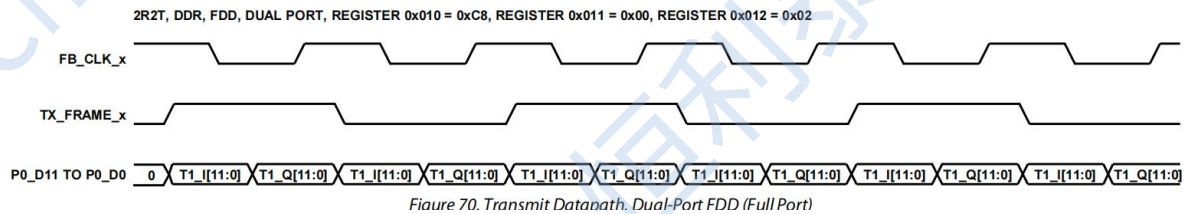


Figure 70. Transmit Datapath, Dual-Port FDD (Full Port)

以上就是当9361工作在单端数据接口CMOS，在FDD全双工，将P0和P1均使用上的模式下的1R1T和2R2T一收一发两收两发的所有时序。接下来我们讲解LVDS信号模式的FDD，P0和P1均使用的情况下的时序。

七、LVDS差分引脚接口模式下的接口信号及时序

相对于CMOS单端接口，LVDS低压差分信号能够获得更好的数据传输稳定性和抗干扰能力，并且，信号由于幅值小，能够有更好的EMI性能，电磁兼容性更友好。唯一的不足之处就是我们在设计PCB走线，狭小的芯片引脚扇出更具挑战性，而且需要PCB走线做差分布线，并且为了更好地稳定性，还需要针对差分做差分走线阻抗。在LVDS接口下，可以获得在1R1T和2R2T都能够达到芯片最大采样带宽。我们LVDS差分与BBP(FPGA)连接如下图所示：

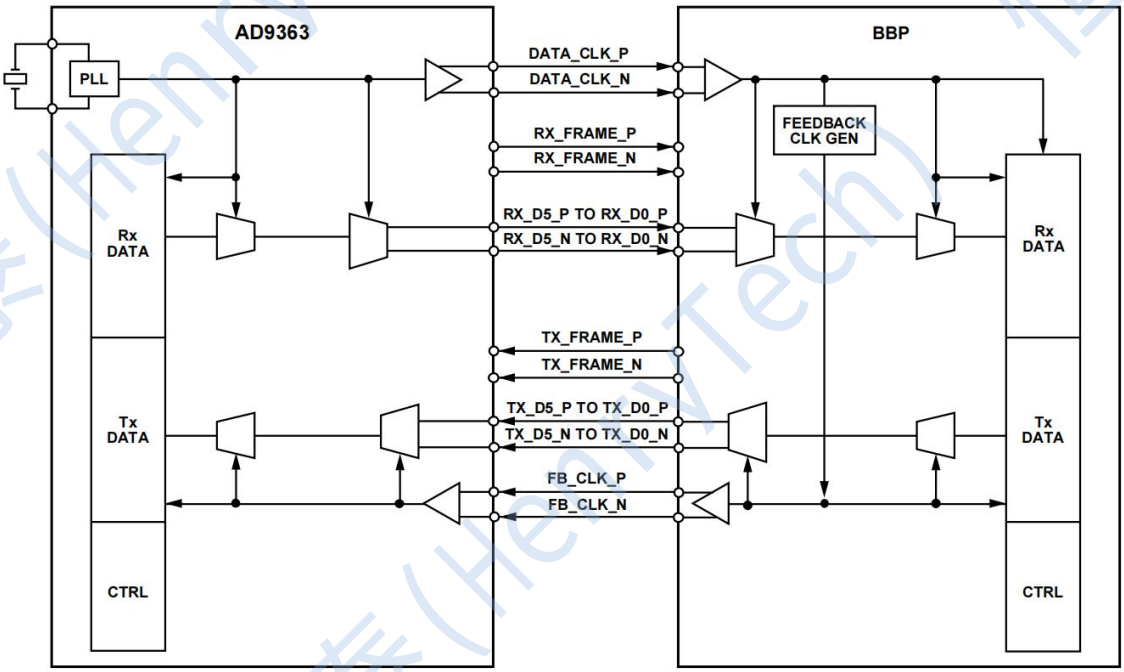


Figure 73. AD9363 Datapath, LVDS Mode

其中我们的数据和时钟及同步信号都是采用差分成对连接。另外，P0和P1，均为差分，并且与CMOS单端不同，我们差分数据由于P0 P1只有12根线，原来单端现在变成差分，就变成了6对差分信号。所以数据在传输的时候，以6Bit进行先后传输12BIT数据的高低位。同样的，差分LVDS传输也用到DDR双边沿传输。最后进行拼接还原12位数据。与P0 P1接口对用差分和线序关系，TX接口（P0）：

- P0_D0->TX_D0_N P0_D1->TX_D0_P
- P0_D2->TX_D1_N P0_D3->TX_D1_P
- P0_D4->TX_D2_N P0_D5->TX_D2_P
- P0_D6->TX_D3_N P0_D7->TX_D3_P
- P0_D8->TX_D4_N P0_D9->TX_D4_P
- P0_D10->TX_D5_N P0_D11->TX_D5_P

RX接口（P1）：

- P1_D0->RX_D0_N P1_D1->RX_D0_P
- P1_D2->RX_D1_N P1_D3->RX_D1_P
- P1_D4->RX_D2_N P1_D5->RX_D2_P

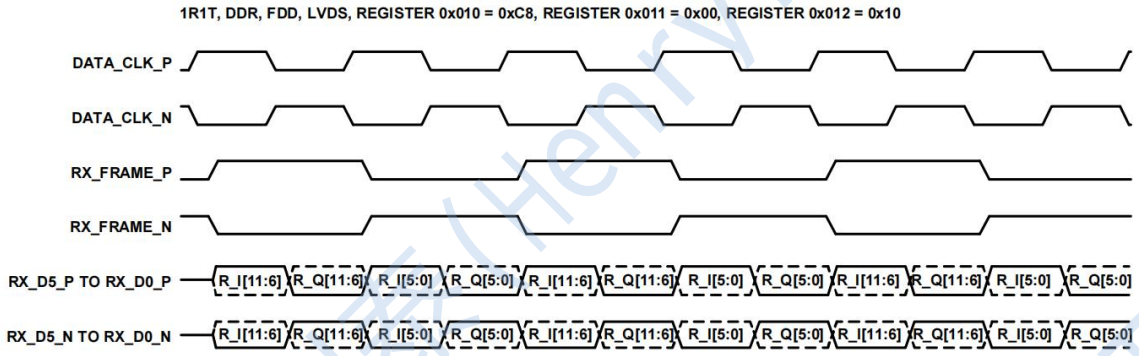
- P1_D6->RX_D3_N P1_D7->RX_D3_P
- P1_D8->RX_D4_N P1_D9->RX_D4_P
- P1_D10->RX_D5_N P1_D11->RX_D5_P

使用到LVDS差分模式的数据传输速率如下表所示，可以看出，1R1T和2R2T均可以达到56M采样极限性能：

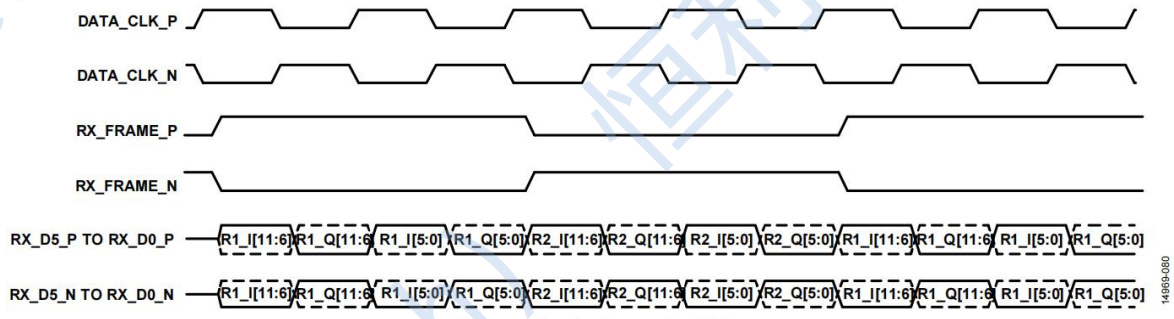
Table 49. Maximum LVDS Data Rates and Signal Bandwidths

	1R1T Configurations			1R2T/2R1T/2R2T Configurations		
		Maximum RF Channel Signal Bandwidth			Maximum RF Channel Signal Bandwidth (MHz) Per Channel	
Operating Mode	Maximum Data Rate (Combined I and Q Words)	Using Minimum Sample Frequency	Using 2× Oversampling	Maximum Data Rate (Combined I and Q Words)	Using Minimum Sample Frequency	Using 2× Oversampling
Dual-Port, Full Duplex	61.44	56[1]	56[1]	61.44	56[1]	30.72

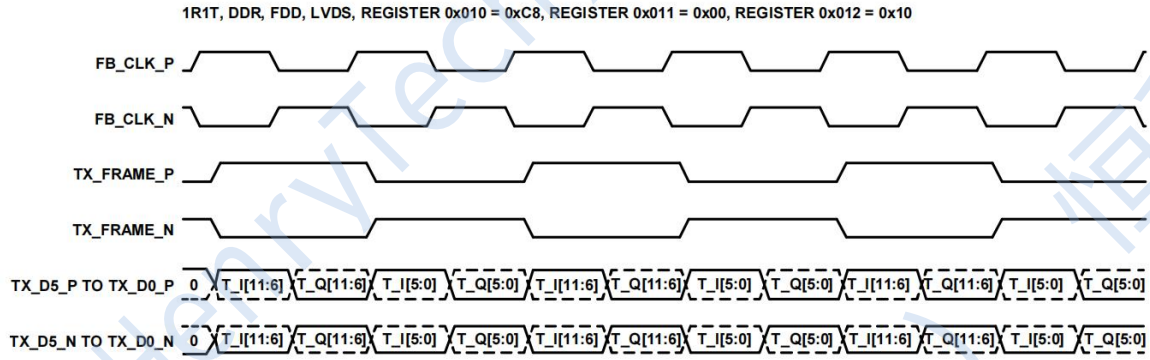
LVDS接口下1R1T的FDD（全双工）工作接口的接收RX时序。此模式下，RX_FRAME=1（差分信号只看P极），时钟下降沿传输一路数据的I分量高6位，时钟上升沿传输一路数据的Q分量高6位；当RX_FRAME=0的时候，时钟下降沿传输I分量的低6位，时钟上升沿传输Q分量低6位：



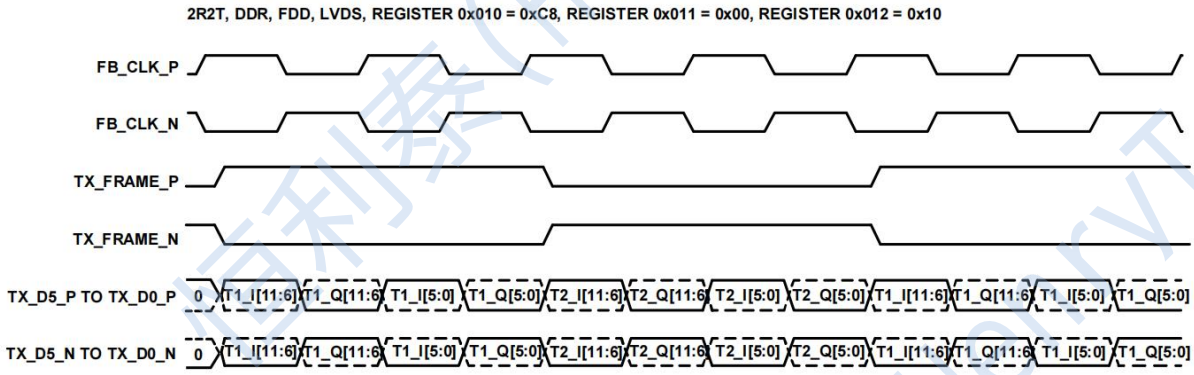
LVDS接口下2R2T的FDD（全双工）工作接口的接收RX时序。当RX_FRAME=1的时候传输的第一路数据：时钟第一个下降沿传输第一路数据的I分量高6位，时钟第一个上升沿传输第一路数据的Q分量高6位；时钟第二个下降沿传输第一路数据的I分量低6位，时钟第二个上升沿传输第一录数据的Q分量低6位；同样的，RX_FRAME=0的时候，我们传输的是第二路数据：与第一路数据排列方式相同，仅仅是R_FRAME=0的时候代表第二路数据：



LVDS接口下1R1T的FDD（全双工）工作接口的发送TX时序。这个模式下我们TX_FRAME=1的时候，时钟下降沿传输一路数据的I分量高6位；时钟上升沿传输Q分量的高6位；当TX_FRAME=0的时候，时钟下降沿传输一路数据的I分量低6位，时钟上升沿传输一路数据的Q分量低6位：



LVDS接口下2R2T的FDD全双工工作接口的发送TX时序。此模式下，TX_FRAME=1的时候，第一个时钟周期下降沿传输第一路数据的I分量高6位，第一个时钟上升沿传输第一录数据的Q分量高6位，第二个时钟周期下降沿传输第一个数据的I分量低6位，第二个时钟上升沿传输第一个数据的Q分量低6位。同理TX_FRAME=0的时候，与第一路数据排列方式相同，仅仅是R_FRAME=0的时候代表第二路数据：



以上就是我们AD9361的LVDS差分模式下的接口时序图了。关于LVDS模式，由于我们SDR采用CMOS单端走线，所以后续LVDS差分不做讨论。

八、CMOS引脚接口模式的接口时序代码RX接收编写

本节讨论第五节<单端CMOS在1R1T模式下的FDD全双工工作接口的接收RX时序>。通过第五节中请参考对应时序图。由于我们SDR仅官方默认固件支持1R1T，单通道，所以对于2R2T双通道模式不做讨论。

讲解编写的最终的代码位于《Z7SDRMicro\04例程代码\936x接口模块代码》中。也可以在资料得到裸机工程《Z7SDRMicro\04例程代码\裸机驱动工程\Z7SDRMicro_noos》解压后在

/repo/ad9361_cmos/src中找到。

首先我们来编写接收部分。接收部分代码顶层模块信号列表部分如下，模块名为ad936x_cmos_rx：

```
module ad936x_cmos_rx(
        input    wire                    ref_clk   ,
        input    wire                    rst       ,
        //adc rx input from port
        input    wire                    rx_clk_in ,
        input    wire                    rx_frame_in ,
        input    wire     [11:0]    rx_data_in  ,
    //adc rx result &valid
        output   wire                    adc_valid,
        output   wire     [11:0]    adc_data_i0  ,
        output   wire     [11:0]    adc_data_q0  ,
    //clock & delay ctrl
        output   wire                    data_clk ,//ad936x data clk to user logical
        input    wire     [4:0]     delay_tap,//delay_value  tap
        input    wire                    delay_reload//1 reload
        );
```

上述代码中信号解释：

- ref_clk：由于数据和控制信号因为936x芯片内部延迟和走线延迟可能存在相位差，所以需要一个就叫做i_delay的模块进行采样相位延迟，i_delay需要一个200M参考时钟；注意，针对ZYNQ7000系列这个是200M，ZU系列是300M；
- rst：接收模块的复位信号，高有效；
- rx_clk_in：接收数据时钟信号，来自AD936X输出；
- rx_frame_in:接收数据同步信号；
- rx_data_in：接收数据信号，位宽12比特；
- adc_valid：输出ADC结果数据有效标志。拉高时刻表示这通道的数据当前有效，通道包含I和Q分量。
- adc_data_i0，adc_data_q0：AD936x的第一个通道的I,Q数据采样的ADC结果输出。
- data_clk:通过AD936X输出给FPGA信号rx_clk_in，经过BUFG在输出的时钟，可以给到用户做用户逻辑时钟。
- delay_tap：io_delay模块配置延迟多少节拍。范围0-31。
- delay_reload：接收数据使用到i_delay模块，delay_reload用于使能i_delay的输入配置延迟节拍；

接下来我们做了一些参数定义和线信号：

```
//delay the input clock
wire                        rx_clk_delay      ;//经过I DELAY之后的时钟
wire      [11:0]    rx_data_delay     ;//经过I DELAY后的数据DATA
wire                        rx_frame_delay    ;//经过I DELAY后的FRAME信号
//iddr
wire      [1:0]     rx_frame          ;//经过DDR双边转单边原语的FRAME信号
wire      [11:0]    rx_data_i         ;//经过DDR双边转单边原语的I路DATA信号
wire      [11:0]    rx_data_q         ;//经过DDR双边转单边原语的Q路DATA信号
//data & valid output reg
reg                 valid_reg ;//数据有效标志
reg       [11:0]    adc_data_i0_reg   ;//I路数据临时寄存器
reg       [11:0]    adc_data_q0_reg   ;//Q路数据临时寄存器
//assign to output
assign adc_valid = valid_reg;//连接到模块接口，数据有效标志
assign adc_data_i0 = adc_data_i0_reg;//连接到模块接口，i路数据
assign adc_data_q0 = adc_data_q0_reg;//连接到模块接口，q路数据
```

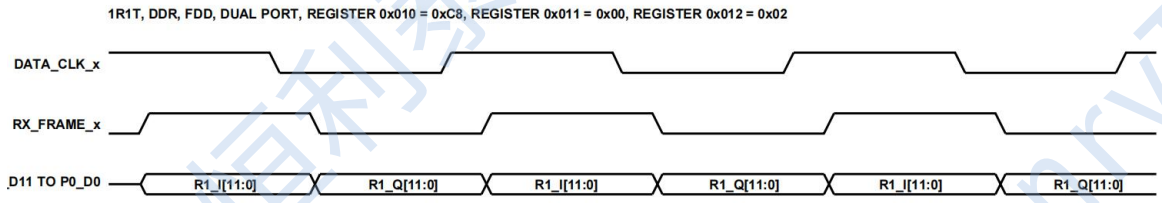
定义了上述信号后我们将接口信号做原语处理。首先，输入rx需要I DELAY，也就是将输入信号做一些延迟，以便调整数据和时钟的采样关系，上升沿下降沿调整对齐相位，以便调整到一个稳定的输入采样结果。另外，我们将对输入rx_clk_in时钟经过一个BUFG增强驱动，以便给其他逻辑使用这个时钟：

```
// IDELAY CTRL
IDELAYCTRL IDELAYCTRL_inst (
          .RDY(),        // 1-bit output: Ready output
          .REFCLK(ref_clk),// 1-bit input: Reference clock input
          .RST(rst)         // 1-bit input: Active high reset input
);
//input phy clock buf
BUFG BUFG_inst (
  .O(data_clk), // 1-bit output: Clock output
  .I(rx_clk_in)  // 1-bit input: Clock input
);
//input delay
IDELAYE2 #(
          .CINVCTRL_SEL("FALSE"),          // Enable dynamic clock inversion (FALSE, TRUE)
          .DELAY_SRC("IDATAIN"),           // Delay input (IDATAIN, DATAIN)
```

```
        .HIGH_PERFORMANCE_MODE("FALSE"), // Reduced jitter ("TRUE"), Reduced power ("FALSE")
        .IDELAY_TYPE("VAR_LOAD"),        // FIXED, VARIABLE, VAR_LOAD, VAR_LOAD_PIPE
        .IDELAY_VALUE(6),                // Input delay tap setting (0-31)
        .PIPE_SEL("FALSE"),              // Select pipelined mode, FALSE, TRUE
        .REFCLK_FREQUENCY(200.0),        // REF input frequency in MHz(190.0-210.0, 290.0-310.0).
        .SIGNAL_PATTERN("DATA")          // DATA, CLOCK input signal
)
//frame delay
IDELAYE2_inst_frame_delay (
        .CNTVALUEOUT(),                          // 5-bit output: Counter value output
        .DATAOUT(rx_frame_delay),      // 1-bit output: Delayed data output
        .C(ref_clk),                   // 1-bit input: Clock input
        .CE(1'b0),                     // 1-bit input: Active high enable increment/decrement
input
        .CINVCTRL(1'b0),               // 1-bit input: Dynamic clock inversion input
        .CNTVALUEIN(delay_tap),        // 5-bit input: Counter value input
        .DATAIN(1'b0),                           // 1-bit input: Internal delay data input
        .IDATAIN(rx_frame_in),         // 1-bit input: Data input from the I/O
        .INC(1'b0),                    // 1-bit input: Increment / Decrement tap delay input
        .LD(delay_reload),             // 1-bit input: Load IDELAY_VALUE input
        .LDPIPEEN(1'b0),               // 1-bit input: Enable PIPELINE register to load data input
        .REGRST(1'b0)                  // 1-bit input: Active-high reset tap-delay input
);
//data delay
genvar i;
generate
        for (i = 0; i < 12; i = i + 1) begin
        IDELAYE2 #(
        .CINVCTRL_SEL("FALSE"),// Enable dynamic clock inversion (FALSE, TRUE)
        .DELAY_SRC("IDATAIN"),// Delay input (IDATAIN, DATAIN)
        .HIGH_PERFORMANCE_MODE("FALSE"),// Reduced jitter ("TRUE"), Reduced power ("FALSE")
        .IDELAY_TYPE("VAR_LOAD"),// FIXED, VARIABLE, VAR_LOAD, VAR_LOAD_PIPE
        .IDELAY_VALUE(0),// Input delay tap setting (0-31)
        .PIPE_SEL("FALSE"),// Select pipelined mode, FALSE, TRUE
        .REFCLK_FREQUENCY(200.0),// REF input frequency in MHz(190.0-210.0, 290.0-310.0).
        .SIGNAL_PATTERN("DATA")          // DATA, CLOCK input signal
        )
        IDELAYE2_inst_frame_delay (
        .CNTVALUEOUT(),                          // 5-bit output: Counter value output
```

```
        .DATAOUT(rx_data_delay[i]),    // 1-bit output: Delayed data output
        .C(ref_clk),                   // 1-bit input: Clock input
        .CE(1'b0),           // 1-bit input: Active high enable increment/decrement input
        .CINVCTRL(1'b0),           // 1-bit input: Dynamic clock inversion input
        .CNTVALUEIN(delay_tap),    // 5-bit input: Counter value input
        .DATAIN(1'b0),                         // 1-bit input: Internal delay data input
        .IDATAIN(rx_data_in[i]),     // 1-bit input: Data input from the I/O
        .INC(1'b0),          // 1-bit input: Increment / Decrement tap delay input
        .LD(delay_reload),   // 1-bit input: Load IDELAY_VALUE input
        .LDPIPEEN(1'b0),   // 1-bit input: Enable PIPELINE register to load data input
        .REGRST(1'b0)                  // 1-bit input: Active-high reset tap-delay input
        );
        end
endgenerate
```

经过IDELAY调整数据和FRAME的延迟之后，我们可以进行下一步，就是将输入信号双边沿采样转换成单边沿。由于FPGA在时钟上升沿和下降沿均可传输出具，所以，针对只在上升沿或者下降沿传输的信号叫做SDR单边沿，针对上升沿下降沿均传输数据，我们叫做DDR双边沿。由于AD936x在CMOS单端模式使用的双边沿DDR模式，上升沿时钟传输I路数据，下降沿传输Q路，所以我们需要IDDR模块将其分离。FRAME信号也是，结合时序图可以看出下降沿时钟传输FRAME=1表示I路数据，上升沿FRAME=0表示Q路数据：



这部分DDR代码如下：

```
//SDR TO DDR FRAME
IDDR #(
        //"OPPOSITE_EDGE","SAME_EDGE"//or"SAME_EDGE_PIPELINED"
        .DDR_CLK_EDGE("SAME_EDGE_PIPELINED"),
        .INIT_Q1(1'b0), // Initial value of Q1: 1'b0 or 1'b1
        .INIT_Q2(1'b0), // Initial value of Q2: 1'b0 or 1'b1
        .SRTYPE("SYNC") // Set/Reset type: "SYNC" or "ASYNC"
)       IDDR_inst_frame (
        .Q1(rx_frame[1]),    // 1-bit output for positive edge of clock
        .Q2(rx_frame[0]),    // 1-bit output for negative edge of clock
        .C(data_clk),        // 1-bit clock input
```

```
        .CE(1'b1),                                  // 1-bit clock enable input
        .D(rx_frame_delay), // 1-bit DDR data input
        .R(1'b0),                          // 1-bit reset
        .S(1'b0)                           // 1-bit set
);
//SDR TO DDR DATA
generate
        for (i = 0; i < 12; i = i + 1)
        begin:data_iddr
        IDDR #(
          //"OPPOSITE_EDGE","SAME_EDGE"or"SAME_EDGE_PIPELINED"
                .DDR_CLK_EDGE("SAME_EDGE_PIPELINED"),
                .INIT_Q1(1'b0), // Initial value of Q1: 1'b0 or 1'b1
                .INIT_Q2(1'b0), // Initial value of Q2: 1'b0 or 1'b1
                .SRTYPE("SYNC") // Set/Reset type: "SYNC" or "ASYNC"
        )       IDDR_inst_data (
                .Q1(rx_data_i[i]),              // 1-bit output for positive edge of clock
                .Q2(rx_data_q[i]),              // 1-bit output for negative edge of clock
                .C(data_clk),                   // 1-bit clock input
                .CE(1'b1),                                      // 1-bit clock enable input
                .D(rx_data_delay[i]),           // 1-bit DDR data input
                .R(1'b0),                               // 1-bit reset
                .S(1'b0)                                // 1-bit set
        );
        end
endgenerate
```

最后我们简单的打一拍，拉高valid_reg寄存器，表示数据有效：

```
//rx fram & data shift output
always @(posedge data_clk) begin
        if (rst==1'b1) begin
                adc_data_i0_reg <= 'd0;
                adc_data_q0_reg <= 'd0;
                valid_reg <= 1'b0;
        end
        else if(rx_frame==2'b10)begin
      adc_data_i0_reg <= rx_data_i;
      adc_data_q0_reg <= rx_data_q;
```

```
        valid_reg <= 1'b1;
          end
end
```

打一拍后的的寄存器最后在前面定义寄存器那部分assign到模块接口上，到此，关于RX接收部分就讲完了。接下来我们讲解发送数据接口，TX部分。

九、CMOS引脚接口模式的接口时序代码TX发送编写

本节讨论第五节中<单端CMOS在1R1T模式下的全双工工作接口的发送TX时序>发送部分代码模块ad936x_cmos_tx，模块接口如下：

```
module ad936x_cmos_tx(
          input     wire                          fb_clk              ,//drive user logic
          input     wire                          rst                 ,
          //tx out to port
          output    wire                          tx_clk_out          ,
          output    wire                          tx_frame_out,
          output    wire  [11:0]        tx_data_out ,
    //dac data in
          input     wire                          dac_valid ,
          input     wire      [11:0]    dac_data_i0         ,
          input     wire      [11:0]    dac_data_q0
          );
```

上述模块列表信号解释：

- fb_clk数据发送逻辑的同步时钟；
- rst：模块复位信号,高有效；
- tx_clk_out：输出到接口的LVCMOS发送数据时钟；
- tx_frame_out:输出到接口的LVCMOS差分同步信号；
- tx_data_out输出到接口的LVCMOS差分数据，12Bit；
- dac_valid：输入的用于指示当前的输入DAC数据有效标志；
- dac_data_i0，dac_data_q0:要发送的单个通道数据的iq分量；

接下来定义一些寄存器和参数，简单来说就是将dac两路数据和valid信号定义两个缓冲寄存器，用于后面打拍两次：

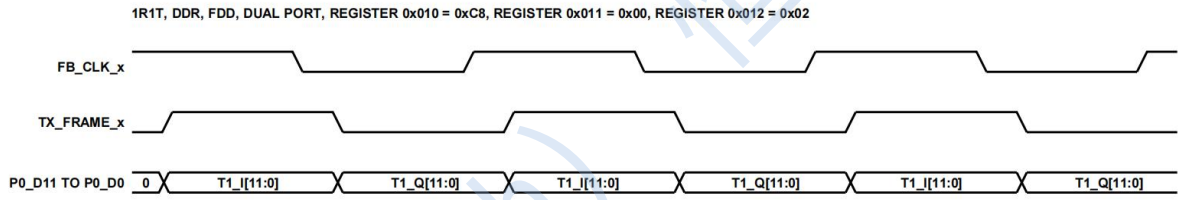
```
reg                           dac_valid_reg       ;
reg       [11:0]    dac_data_i0_reg     ;
```

```
reg         [11:0]     dac_data_q0_reg      ;
reg         [1:0]      tx_frame_reg                 ;
reg         [11:0]     tx_data_i_reg                ;
reg         [11:0]     tx_data_q_reg                ;
```

结合时序图，dac数据valid打一拍的寄存器判断等于1，就正常将dac i q数据打两拍，我们frame需要上升沿输出1，下降沿输出0，所以tx_frame_reg直接就给2’b10。代码如下：

```
//valid=1,output data
always @(posedge fb_clk ) begin
        if (rst==1'b1) begin
                dac_valid_reg <= 1'b0;
                dac_data_i0_reg <= 'd0;
                dac_data_q0_reg <= 'd0;
            tx_data_i_reg <= 'd0;
                tx_data_q_reg <= 'd0;
            tx_frame_reg <= 2'b00;
        end
        else begin
                dac_valid_reg <= dac_valid;
                if(dac_valid_reg==1)begin
                        dac_data_i0_reg <= dac_data_i0;
                        dac_data_q0_reg <= dac_data_q0;
                        tx_data_i_reg <= dac_data_i0_reg;
                        tx_data_q_reg <= dac_data_q0_reg;
                        tx_frame_reg <= 2'b10;
                end
        end
end
```

上述代码仅仅做了最简单的frame寄存器赋值，iq输入的数据做个打拍。真正要输出，还要经过我们ODDR原语，将tx_frame_reg=2’b10输出去，高位1给下降沿，表示frame在时钟下降沿的时候是输出1，低位0表示frame在时钟上升沿的时候是输出0。然后是数据i，q，我们时钟下降沿输出tx_data_i_reg，上升沿输出tx_data_q_reg。我们结合一下时序图；



代码如下：

```
//oddr frame
genvar i;
ODDR #(
        .DDR_CLK_EDGE("SAME_EDGE"), // "OPPOSITE_EDGE" or "SAME_EDGE"
        .INIT(1'b0),                                    // Initial value of Q: 1'b0 or 1'b1
        .SRTYPE("SYNC")                                 // Set/Reset type: "SYNC" or "ASYNC"
)       ODDR_inst_frame (
        .Q(tx_frame_out),       // 1-bit DDR output
        .C(fb_clk),                         // 1-bit clock input
        .CE(1'b1),                              // 1-bit clock enable input
        .D1(tx_frame_reg[1]),       // 1-bit data input (positive edge)
        .D2(tx_frame_reg[0]),       // 1-bit data input (negative edge)
        .R(1'b0),                               // 1-bit reset
        .S(1'b0)                                // 1-bit set
);
//oddr clock
ODDR #(
        .DDR_CLK_EDGE("SAME_EDGE"), // "OPPOSITE_EDGE" or "SAME_EDGE"
        .INIT(1'b0),                                    // Initial value of Q: 1'b0 or 1'b1
        .SRTYPE("SYNC")                                 // Set/Reset type: "SYNC" or "ASYNC"
)       ODDR_inst_clock (
        .Q(tx_clk_out),         // 1-bit DDR output
        .C(fb_clk),                         // 1-bit clock input
        .CE(1'b1),                          // 1-bit clock enable input
        .D1(1'b1),                          // 1-bit data input (positive edge)
        .D2(1'b0),                          // 1-bit data input (negative edge)
        .R(1'b0),                           // 1-bit reset
        .S(1'b0)                            // 1-bit set
);
//oddr data
generate
        for (i = 0; i < 12; i = i + 1) begin
                ODDR #(
                        .DDR_CLK_EDGE("SAME_EDGE"), // "OPPOSITE_EDGE" or "SAME_EDGE"
                        .INIT(1'b0),            // Initial value of Q: 1'b0 or 1'b1
                        .SRTYPE("SYNC")         // Set/Reset type: "SYNC" or "ASYNC"
```

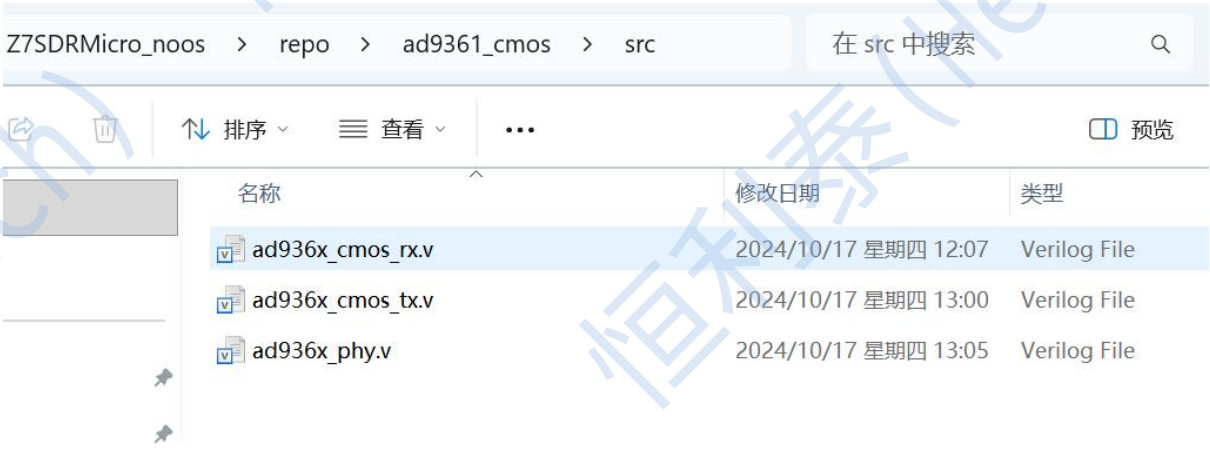
```
                    )          ODDR_inst_clock (
                                .Q(tx_data_out[i]),     // 1-bit DDR output
                                .C(fb_clk),             // 1-bit clock input
                                .CE(1'b1),              // 1-bit clock enable input
                                .D1(tx_data_i_reg[i]), // 1-bit data input (positive edge)
                                .D2(tx_data_q_reg[i]), // 1-bit data input (negative edge)
                                .R(1'b0),               // 1-bit reset
                                .S(1'b0)                // 1-bit set
                    );
            end
endgenerate
```

以上就是我们FPGA与9361的通信接口CMOS单端1R1T单通道收发代码说明。如需完整的代码和完整的裸机实例我们通过后续章节给大家讲解。

十、ZYNQ裸机驱动实例讲解

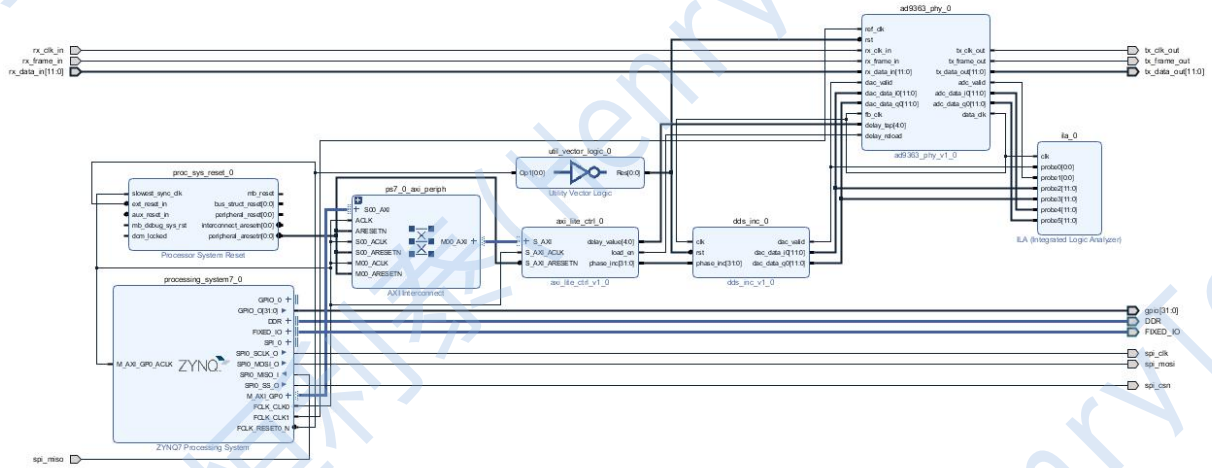
我们网盘资料链接里的《Z7SDRMicro\04例程代码\裸机驱动工程》目录中，提供了一个我们配套的裸机驱动例程《Z7SDRMicro_noos》。这个例程工程是基于VIVADO2018.3版本。很多小伙伴可能会问，为什么不建议使用高版本VITIS？这是因为高版本的调试VIVADO和VITIS步骤繁琐，如果大家修改了vivado设计，重新更新vitis会比较麻烦，而sdk会更方便。这里不推荐大家使用新版本。

我们在第7-8节中所讲解涉及到的源代码在裸机驱动例程《Z7SDRMicro_noos》中如下位置，其中这个位置是将接收和发送代码，加上了一个叫做ad936x_phy的模块顶层，将ad936x_cmos_rx和ad936x_cmos_tx代码模块一起封装，最终打包的ad936x_phy的IP核（注意这里仅仅给大家展示代码IP位置，例程要自己解压放到英文路径）：

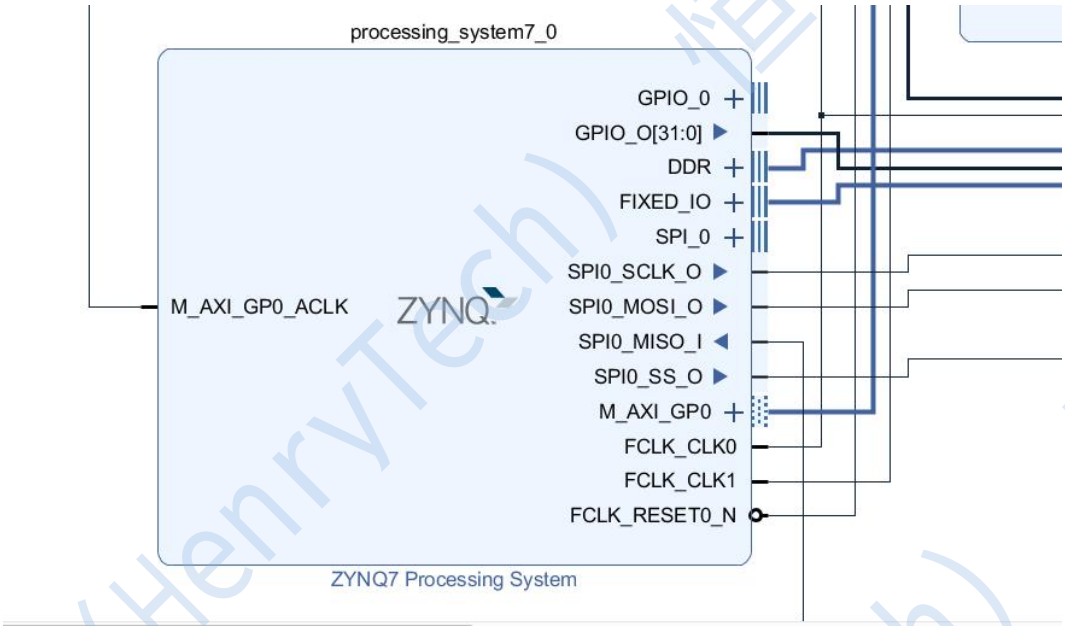


例程所有用到的额外的IP库位置如下（注意这里仅仅给大家展示代码IP位置，例程要自己解压放到英文路径）：

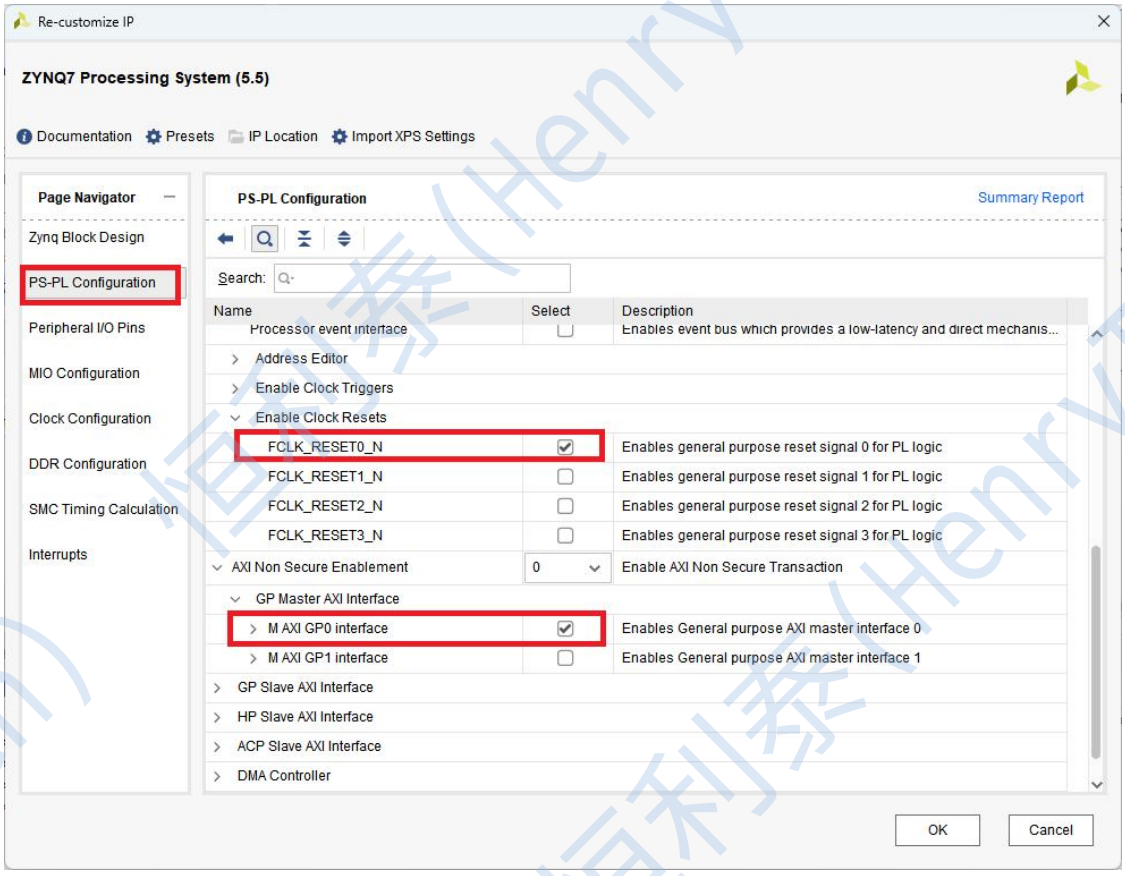
我们将例程解压到一个英文路径，建议直接解压放到桌面或者硬盘根目录，使用vivado2018.3即可打开。如果大家电脑安装多个版本vivado，建议开始菜单栏找到vivado2018.3启动后找到解压的例程然后打再打开它。不要用其他版本，版本不兼容不能直接使用。我们打开工程，我们打开bd文档即可看到如下的设计逻辑框图：



关于逻辑框图的具体设计过程这里不做详细讲解，我们对逻辑的主要的几个模块进行讲解一下。首先是ZYNQ处理器的配置，双击ZYNQ核打开配置可以看一下：



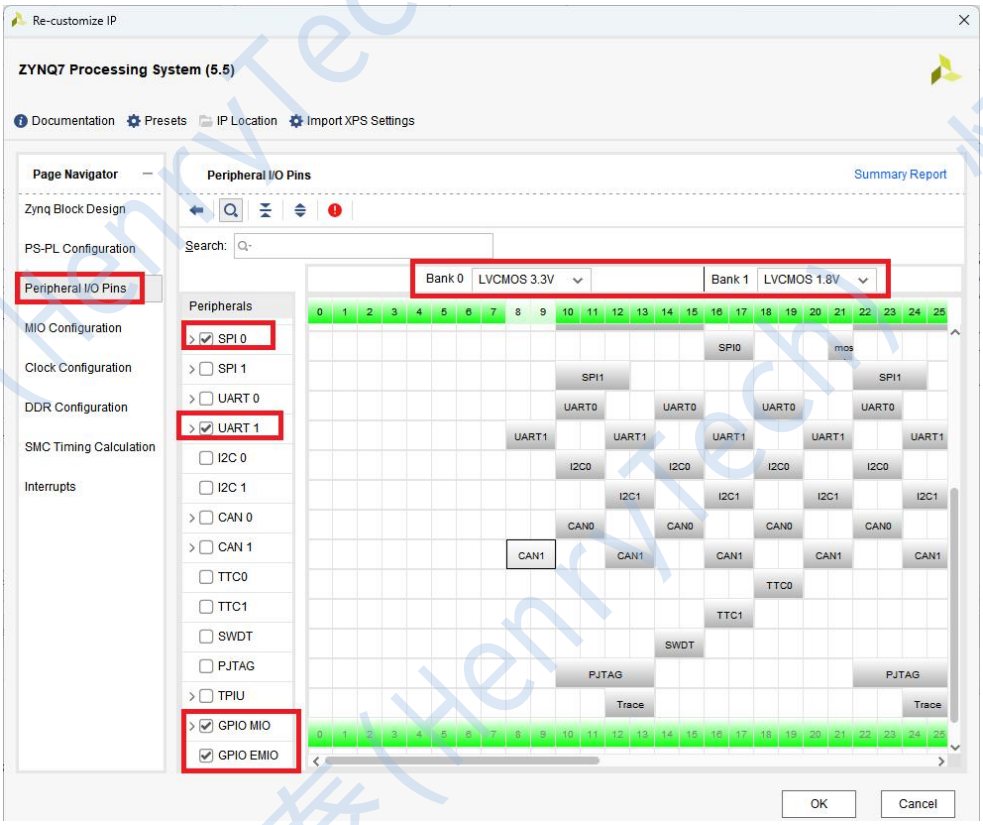
PS-PL配置页面，我们需要一个FCLK_RESET0_N提供给FPGA部分逻辑做复位，以及一个GP Master端口，用于AXI接口配置：

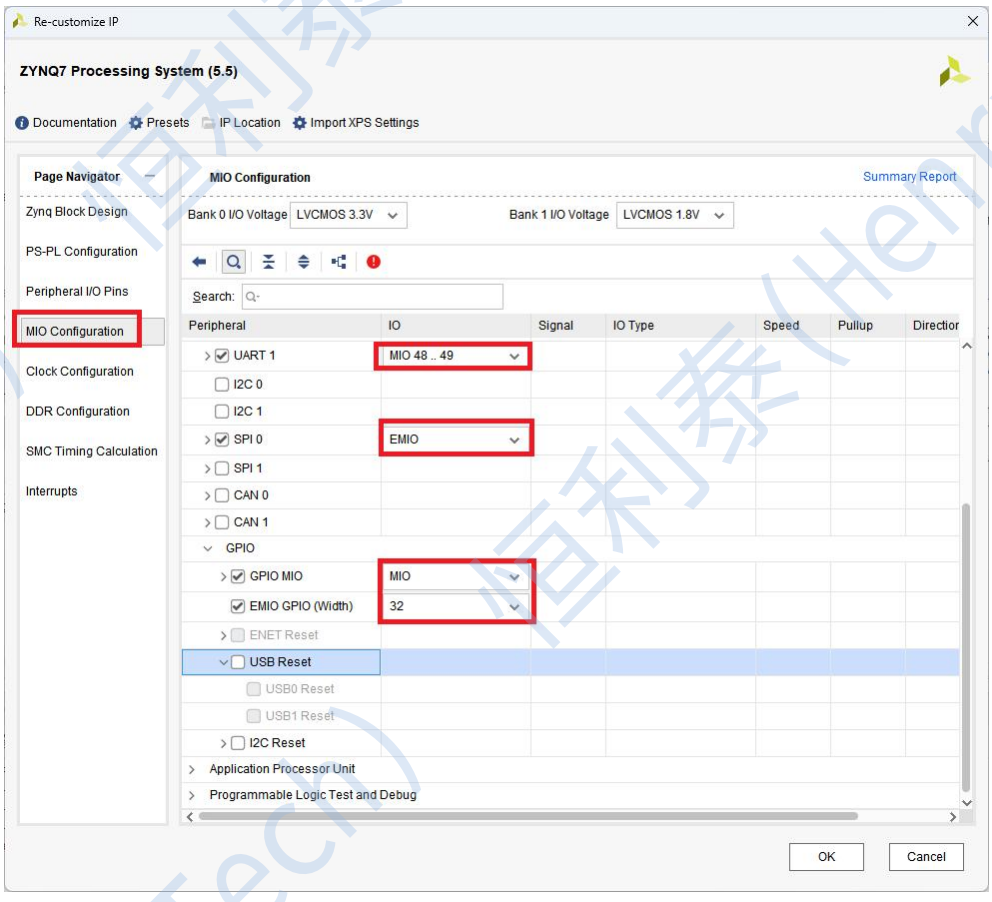


接着是我们PS外设端口配置。我们驱动9361，需要用到串口进行信息打印，需要SPI进行9361的参数和初始化配置，另外还需要EMIO做普通的控制信号端口，我们这部分端口包括有RESETN, TXNRX, ENABLE, EN_AGC信号等等，最终需要绑定连接到9361的控制端口上。我们这些IO也可以不使用PS EMIO，使用AXI_GPIO模块进行扩展也是可以的，但是PS 使用EMIO更方便，也不需要额外占用AXI接口资源。关于普通的一些IO，使用很灵活，这里没有太多讲究的地方。

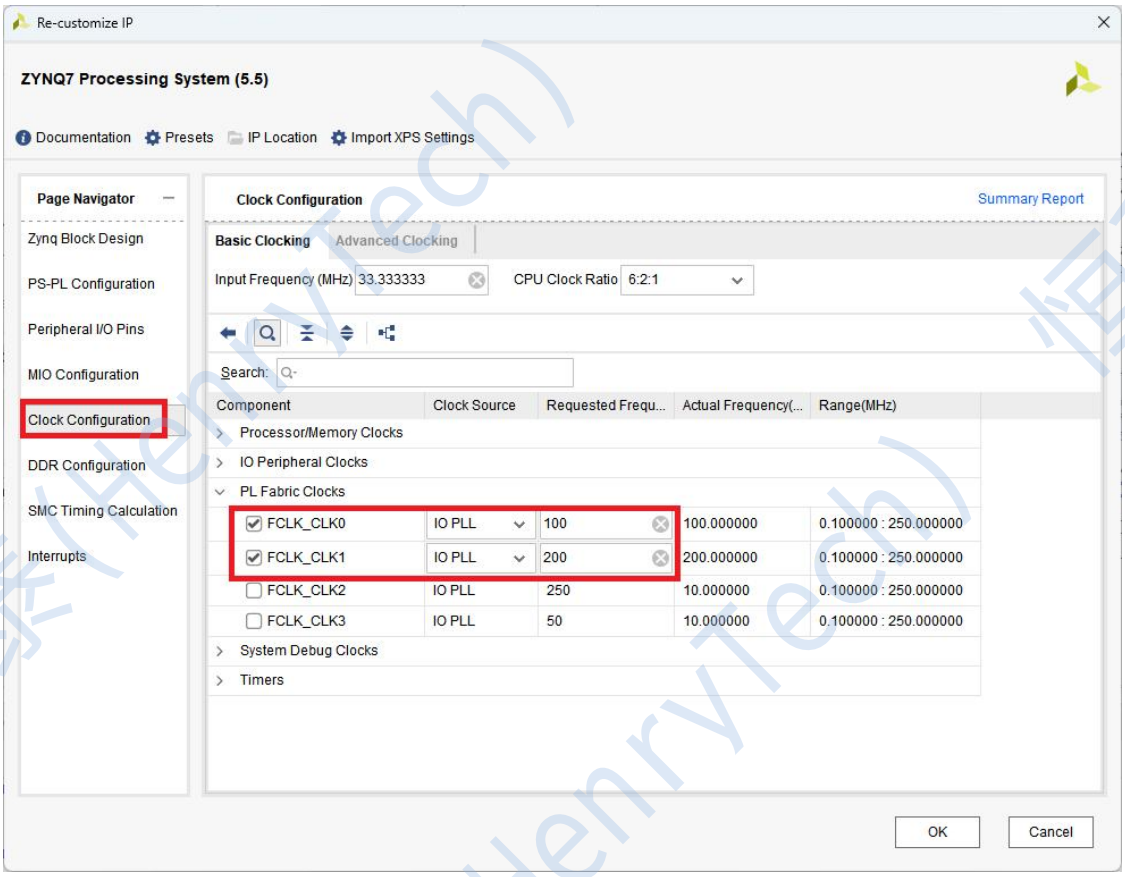
EMIO其实就是ZYNQ给ARM提供一个最便捷的使用FPGA引脚的方式了。其中ARM还有MIO可用，但是我们实际上没有开发板单独引出MIO引脚，所以MIO不做讨论。

外设配置勾选如下，UART1,SPI0以及GPIO EMIO，并且MIO配置页面配置如下，除了串口是MIO42 43,SPI则勾选EMIO，我们9361的SPI连接到FPGA的引脚的,另外需要GPIO EMIO，配置界面最上面，BANK0为LVCMOS 3.3V,BANK1是LVCMOS 1.8V：

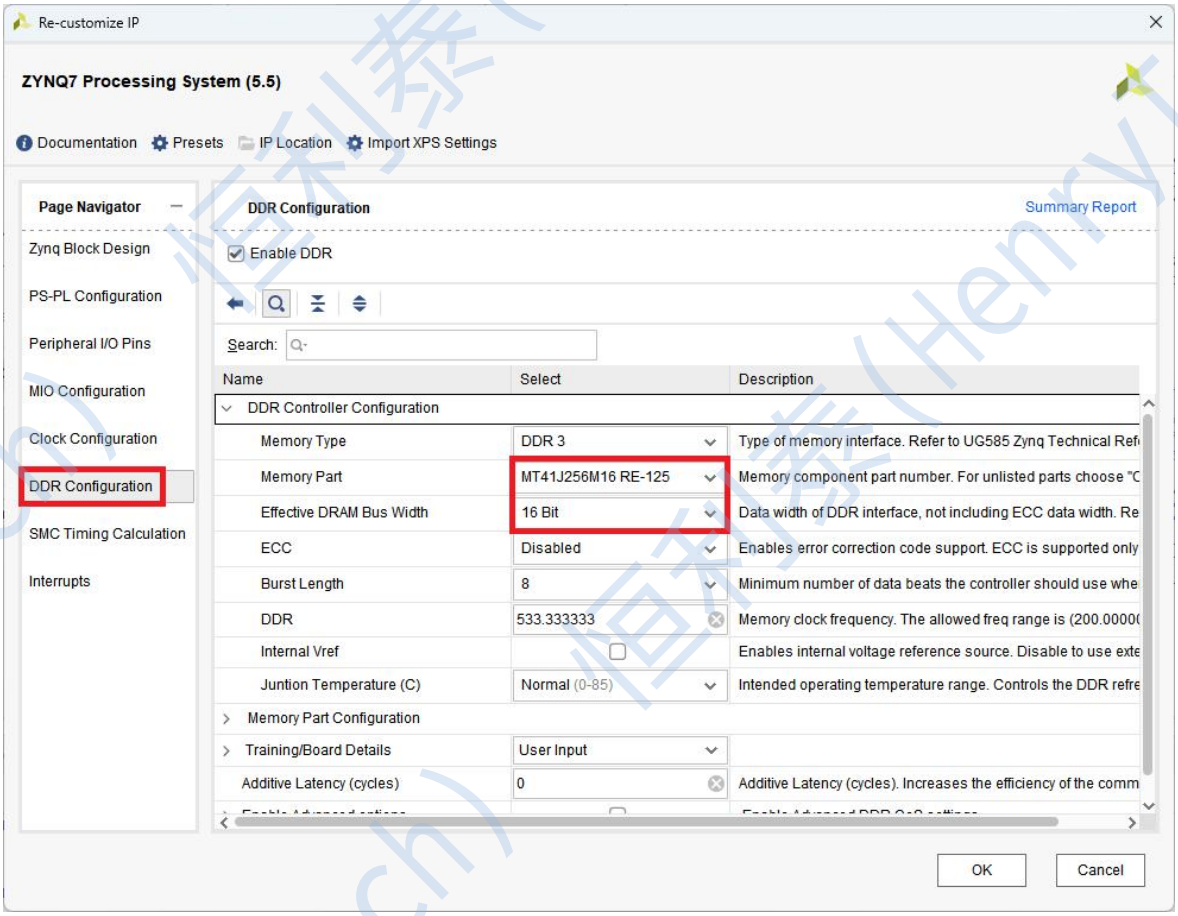




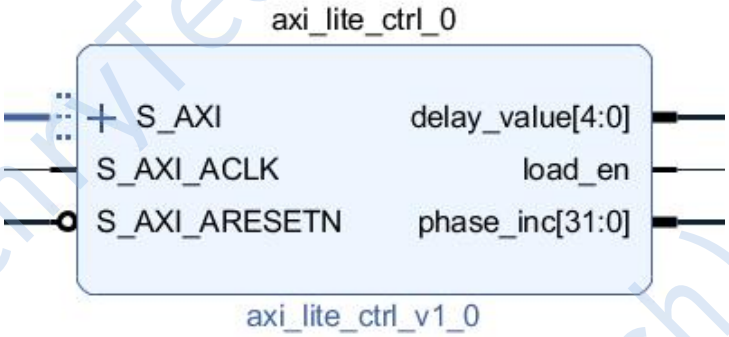
时钟输出这一页，我们需要PS提供2路，一路100M和2路200M，100M给到AXI接口相关模块使用，200M给到9361的IO_DELAY做参考时钟：



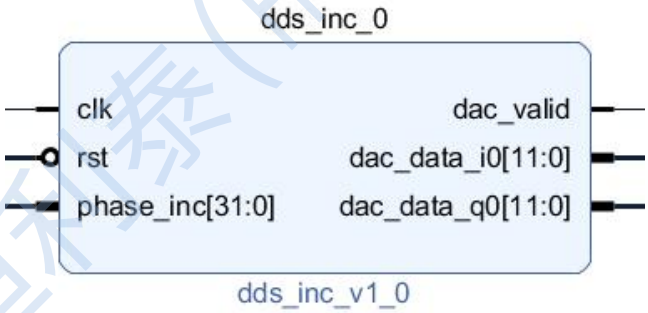
最后是DDR3配置。我们ZYNQ运行程序使用的内存DDR3规格型号是一颗16位颗粒：



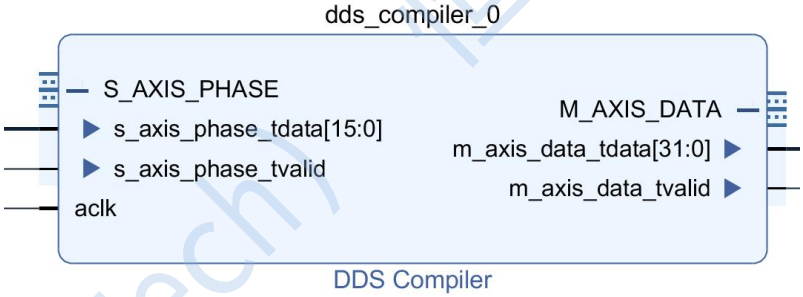
以上就是ZYNQ PS的IP配置。接下来我们讲解一下axi_lite_ctrl这个模块。这个模块用于通过AXI接口，输出控制ad9361的i_delay的load_en和delay_value值，以及用于控制DDS模块的相位增量phase_inc。这个模块是通过vivado的IP向导自己生成的AXI Lite接口的模块，最终修改代码接口来生成的，这里不细讲：



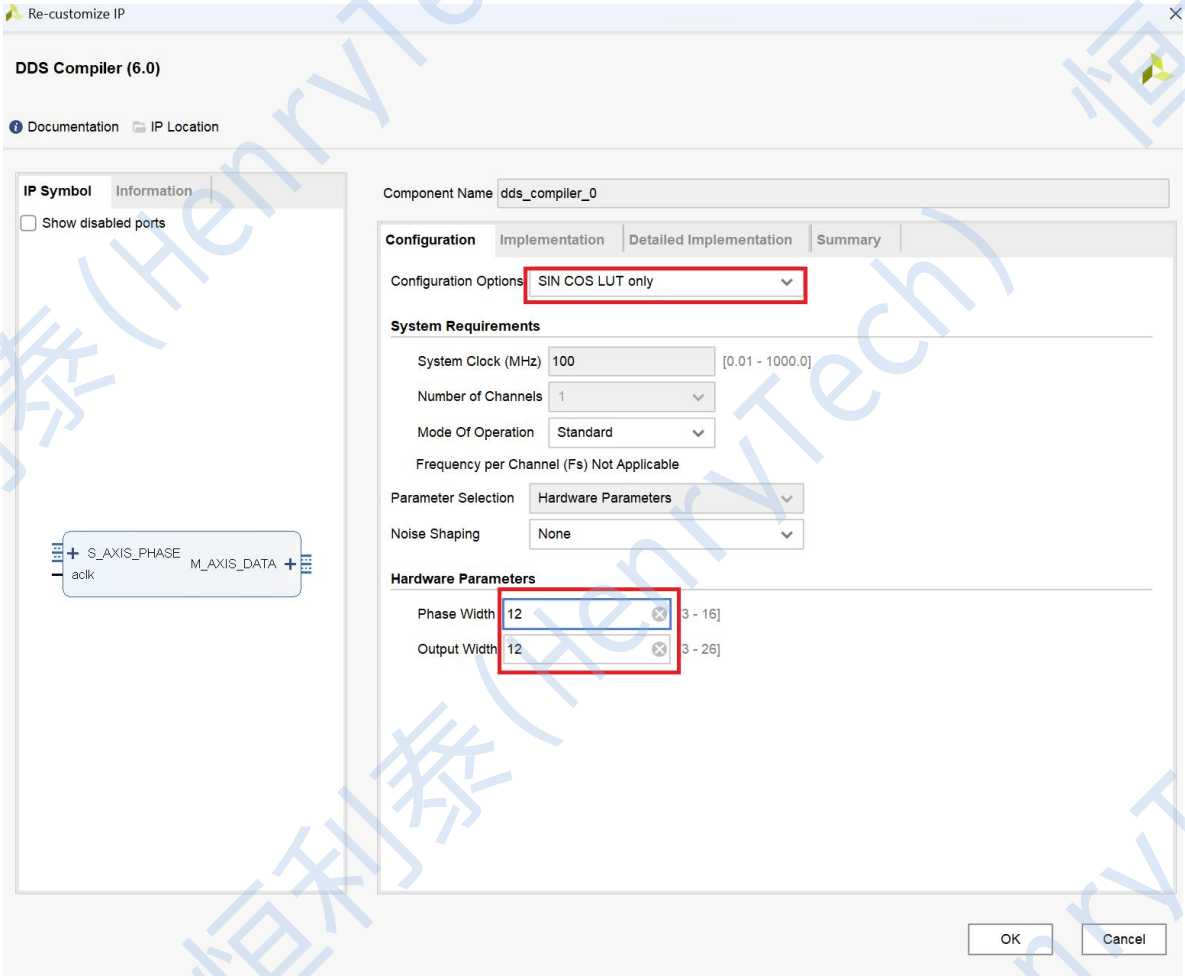
接着就是我们的dds_inc模块，这个模块给定一个相位增量，通过phase_inc输入，可以输出合成的数字正余弦数字波形输出。这个模块内部封装了一个DDS IP核，我们调试接口为了方便简单直观的观察测试，我们一般用DDS模块来给9361的输入，最后9361的接收端口读取测试验证数据是否一致。dds_inc模块如下：



关于DDS IP核，如需更详细的了解请参阅《DDS Compiler v6.0文档》，我们例程在IP的s_axis_phase_tdata给定一个16位的线性变化的数据，我们DDS的m_axis_data_tdata输出正余弦变化的数据。注意DDS模块其实本质就是一个线性0-2π的线性数据转换成-1到+1的函数变换而已，只不过DDS是数字的，输入输出不是小数，而是被放大的整数。dds_inc内部包含的核心DDS模块如下：



关于DDS的IP，我们配置如下，我们输出两路数据，正余弦两路输出，并且输出的正余弦数据为12位宽。注意，输出实际上是有32BIT宽度，正余弦同时输出，就是把32位输出拆分成两个16位的输出，输出的m_axis_data_tdata[31:16]为一路，另一路为m_axis_data_tdata[15:0]。这里一路为正弦一路为余弦。注意，我们输出数据设置有效宽度是12，所以，两个16位的接口数据仅低12位有效，高4位无效。



我们dds_inc的IP代码如下，代码很简单，仅仅将phase_inc连接到DDS IP核的相位增量，然后DDS IP的s_axis_config_tvalid拉高，每个时钟周期都按照正余弦输出dds_out信号，最后这个信号简单的通过打拍，赋值给dac_data_i0和dac_data_q0：

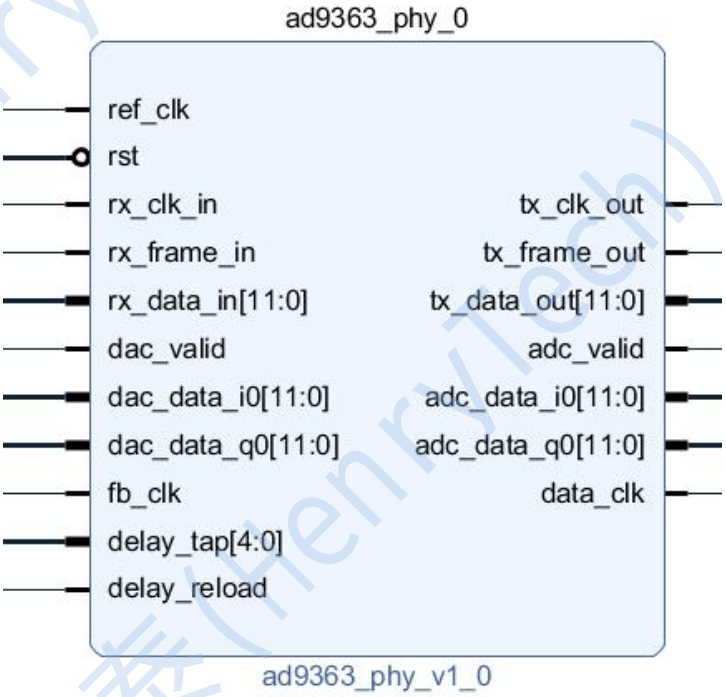
```
//////////////////////////////////////////////////////////////////////////////////
// Company: bochenjingxin
// Engineer: yang
// Create Date: 2024/05/16 12:35:02
// Design Name:
// Module Name: dds_inc
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description: dds module data increament driver
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////
module dds_inc (
          input     wire                clk       ,// Clock
          input     wire                rst       ,// Asynchronous reset active high

          input     wire [31:0]         phase_inc ,
          output    wire                dac_valid ,
          output    reg [11:0]dac_data_i0         ,
          output    reg [11:0]dac_data_q0
);

wire      [31:0]    dds_out    ;
//get sin cos data
always @(posedge clk ) begin
          if (rst==1'b1) begin
                    dac_data_i0 <= 'd0;
                    dac_data_q0 <= 'd0;
          end
          else if (dac_valid == 1'b1) begin
                    dac_data_i0 <= dds_out[27:16];
                    dac_data_q0 <= dds_out[11:0];
          end
end
//dds ip
dds_compiler_0 dds_inst
(
          .aclk(clk),                                 // input wire aclk
          .s_axis_config_tvalid(1'b1),   // input wire s_axis_config_tvalid
          .s_axis_config_tdata(phase_inc),          // input wire [31 : 0] s_axis_config_tdata
          .m_axis_data_tvalid(dac_valid),           // output wire m_axis_data_tvalid
          .m_axis_data_tdata(dds_out)               // output wire [31 : 0] m_axis_data_tdata
);
```

```
endmodule
```

上述代码就完成了我们输出给9361的DAC模拟的数据了。

最后我们来说明一下我们的ad9363_phy这个IP。这个IP就是我们之前第七节中的CMOS接口的时序接收发送代码合并之后例化到一起打包的IP核。这里不再细讲，大家直接参阅第七节源代码，主要就是通过CMOS接口转换出2路ADC采样结果，同时将给定的2路DAC数据通过CMOS输出到AD9361。



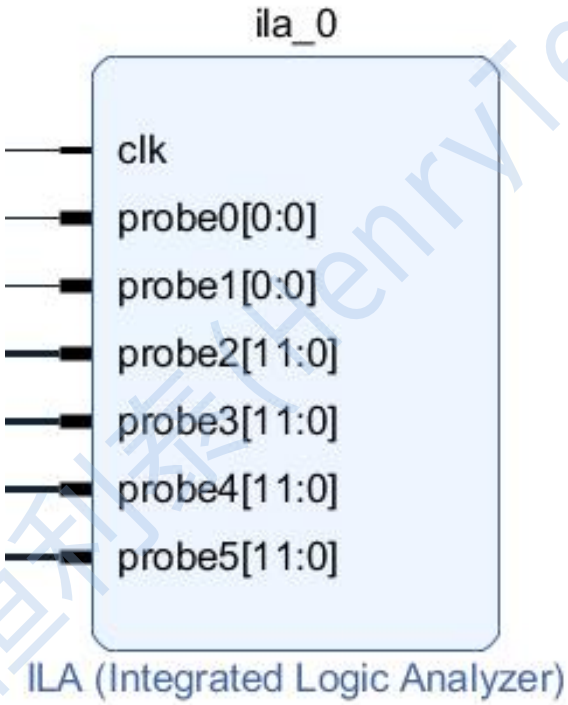
9363_phy这个IP代码顶层如下，仅仅简单例化了TX RX两部分代码，整合在一起，信号直接连接到顶层，没有额外的额代码：

```
module ad936x_phy(
        input     wire                          ref_clk   ,//200M IODELAY REF CLOCK
        input     wire                          rst                ,//system reset
        //rx input
        input     wire                rx_clk_in ,
        input     wire                rx_frame_in       ,
        input     wire  [11:0]        rx_data_in,
        //tx output
        output    wire                          tx_clk_out          ,
        output    wire                          tx_frame_out,
        output    wire  [11:0]        tx_data_out ,
        //get adc data output
        output    wire                          adc_valid ,
        output    wire      [11:0]    adc_data_i0         ,
```

```
        output    wire      [11:0]    adc_data_q0         ,
        //dac output to phy
        input     wire                  dac_valid ,
        input     wire      [11:0]    dac_data_i0         ,
        input     wire      [11:0]    dac_data_q0         ,
        //ctrl & clock
        output    wire                  data_clk ,//user_tx_clk
        input   wire                  fb_clk  ,
        input     wire      [4:0]     delay_tap,//delay_value
        input     wire                  delay_reload//enable delay load
        );
//rx instance
ad936x_cmos_rx ad936x_cmos_rx_inst
(
        .ref_clk      (ref_clk      ),
        .rst          (rst          ),
        //adc rx input from port
        .rx_clk_in    (rx_clk_in    ),
        .rx_frame_in  (rx_frame_in  ),
        .rx_data_in   (rx_data_in   ),
    //adc rx result &valid
        .adc_valid    (adc_valid    ),
        .adc_data_i0  (adc_data_i0  ),
        .adc_data_q0  (adc_data_q0  ),
    //clock & delay ctrl
        .data_clk     (data_clk     ),
        .delay_tap    (delay_tap    ),
        .delay_reload (delay_reload )
);
//tx instance
ad936x_cmos_tx ad936x_cmos_tx_inst
(
        .fb_clk       (fb_clk       ),
        .rst          (rst          ),
        //tx out to port
        .tx_clk_out   (tx_clk_out   ),
        .tx_frame_out (tx_frame_out ),
        .tx_data_out  (tx_data_out  ),
    //dac data in
```

```
        .dac_valid    (dac_valid    ),
        .dac_data_i0  (dac_data_i0  ),
        .dac_data_q0  (dac_data_q0  )
);

endmodule
```

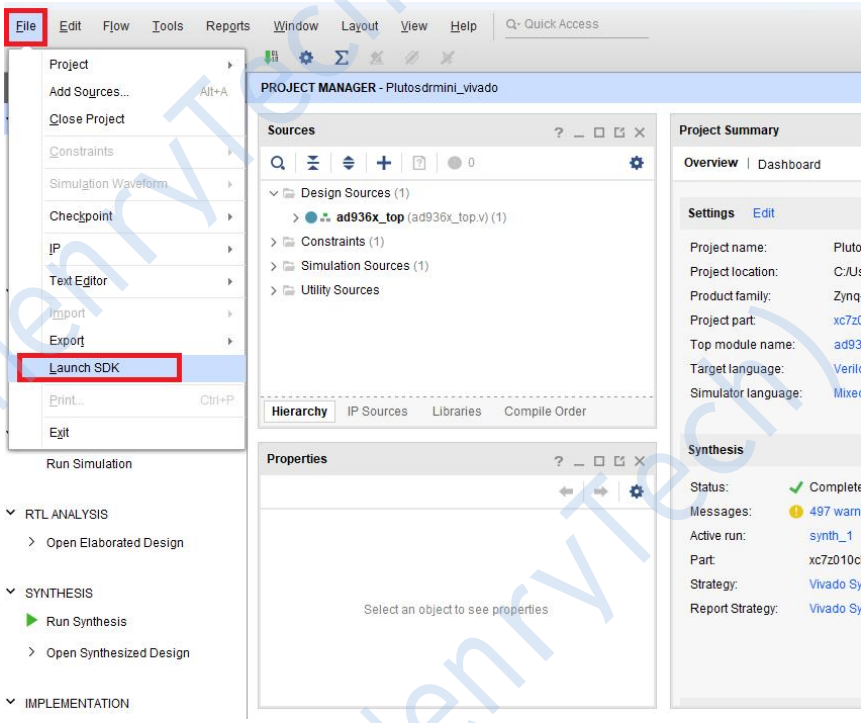
我们设计中还用到一个ILA模块，ILA模块用于连接抓取我们的ad9363_phy IP的输入DAC数据和从端口接收到的输出ADC数据。我们裸机代码就是通过它，抓取的信号数据来进行验证的。ILA是很常规的IP，大家可以参阅《Integrated Logic Analyzer v6.2》手册。



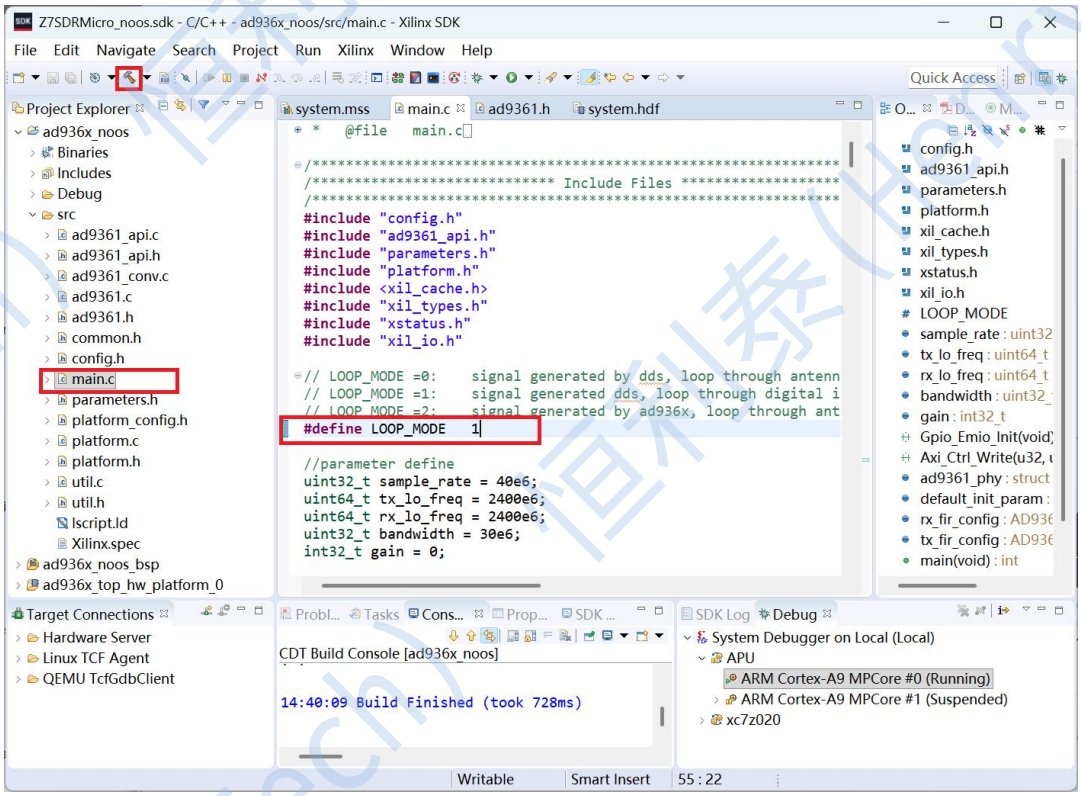
以上就是我们逻辑主要设计部分。其余未提及的IP模块大家可以自己打开BD进行查看，其余的没有讲解的都是一些通用的IP模块，比如proc_sys_reset复位模块和AXI接口转换的axi_interconnect用于AXI接口转换适配的模块。关于FPGA的设计，需要有一定的开发基础，否则无法看懂，想要开发必须具备相关基础。到此我们FPGA部分整体设计就告一段落，后续部分与sdk嵌入式代码和调试相关。我们下一节讲解通过SDK调试测试，准备上板进行实验。通过实验，可以更改一些配置查看效果。

十一、SDK调试测试

我们工程生成BIT文件后（网盘下载后就已经是完整生成的），我们在Vivado的菜单栏下拉File->Launch SDK启动我们的例程SDK：

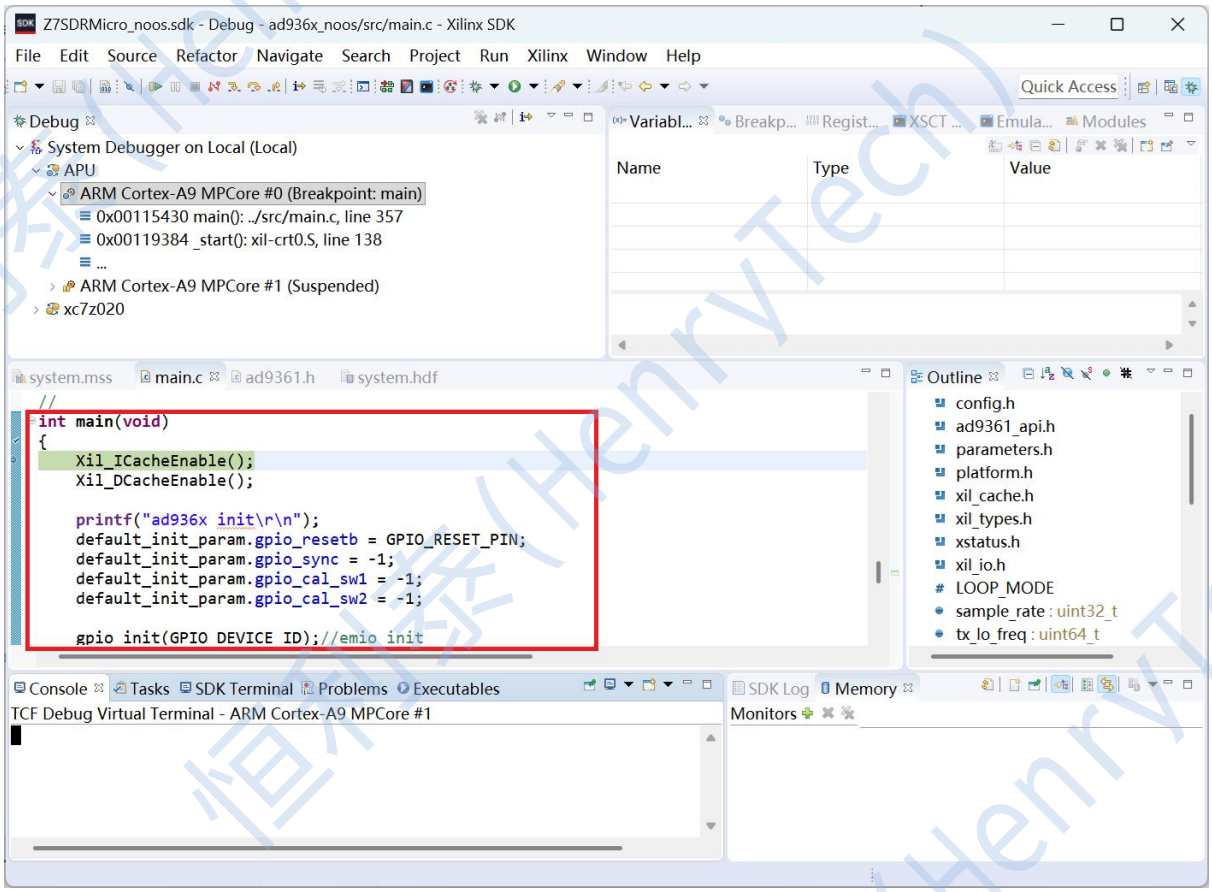


启动SDK后，一般情况下软件会自动导入一个新的硬件平台，可能sdk硬件平台会重复，大家不用理会。打开启动后我们SDK工程如下我们可以将ad9361工程中的src目录展开，就可以看到我们整个arm的驱动源代码，其中包含我们9361的驱动原文件和头文件，以及配置头文件等一些参数定义等。我们找到main.c主程序更改定义LOOP_MODE为1，重新编译：

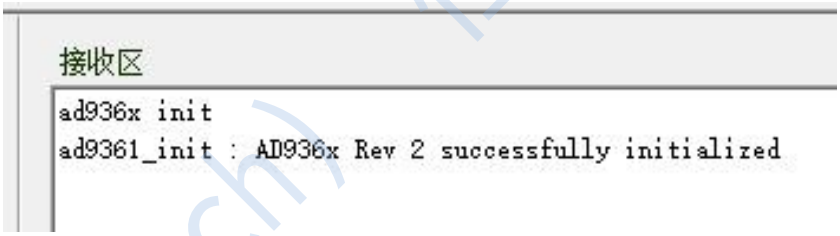


我们将开发板插上电源，将BOOT设置为JTAG模式（必须设置成JTAG模式）然后开关按下后开机，将DEBUG口连接电脑，接好之后如下，这里还需要一根SMA内针内螺纹的同轴线，我们可以将RX1A和TX1A连起来，后面我们实验可以用到，我们出厂自带一根SMA线可以试验。

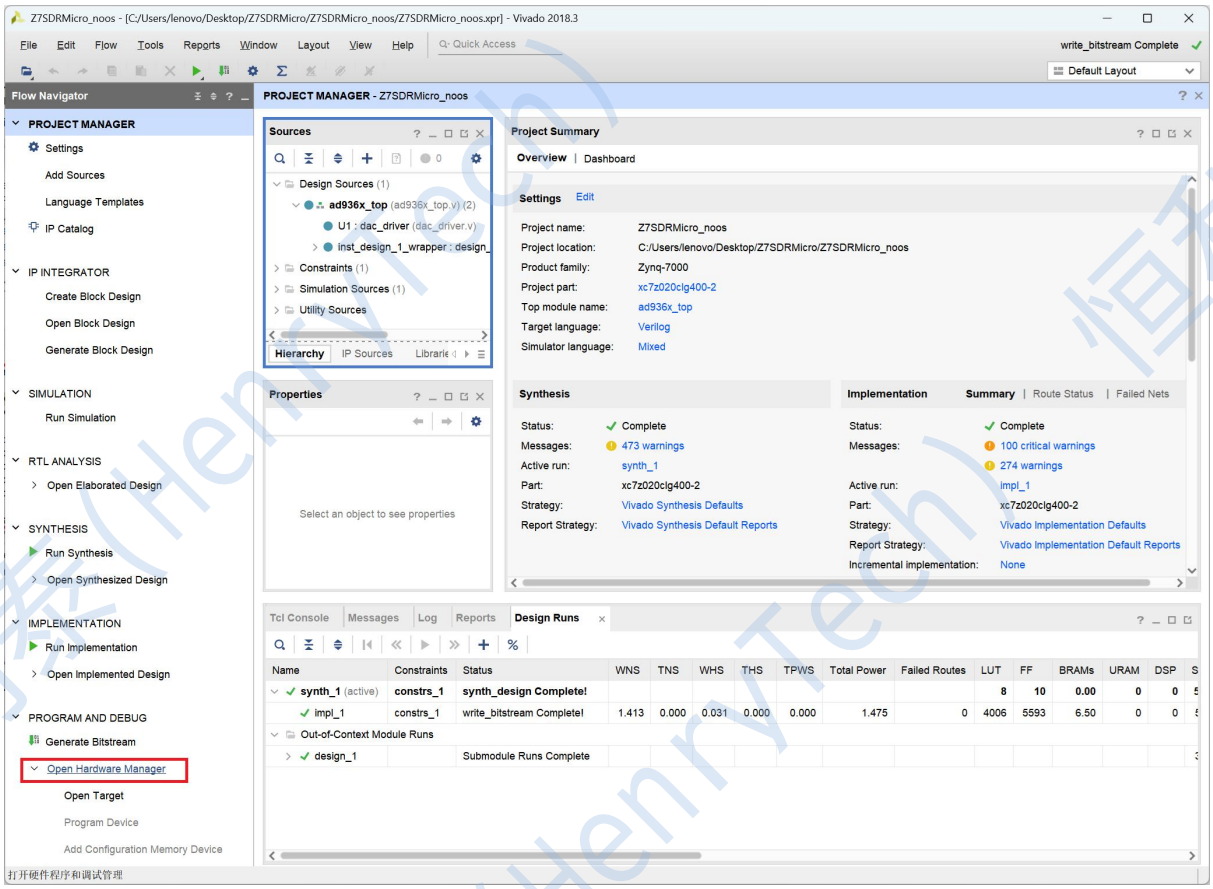
我们在SDK上方点击小虫子符号 开始进行调试。SDK我们已经配置好DEBUG。本例程使用到FPGA，SDK调试会烧写BIT文件然后运行SDK代码，我们DEBUG配置中已经勾选烧写FPGA，DEBUG大家不用单独再配置。注意，下方红色显示可能会出现找不到代码，无法定位，因为例程解压后路径变了，大家点击Locate File重新选择SDK的ad936x_noos工程目录下的src里面对应的main.c文件即可。或者大家可以从头将sdk删掉重新导出硬件重新新建工程最后将代码全部添加一遍也可以解决。建议大家裸机例程工程存放到桌面，与笔者保持一致：



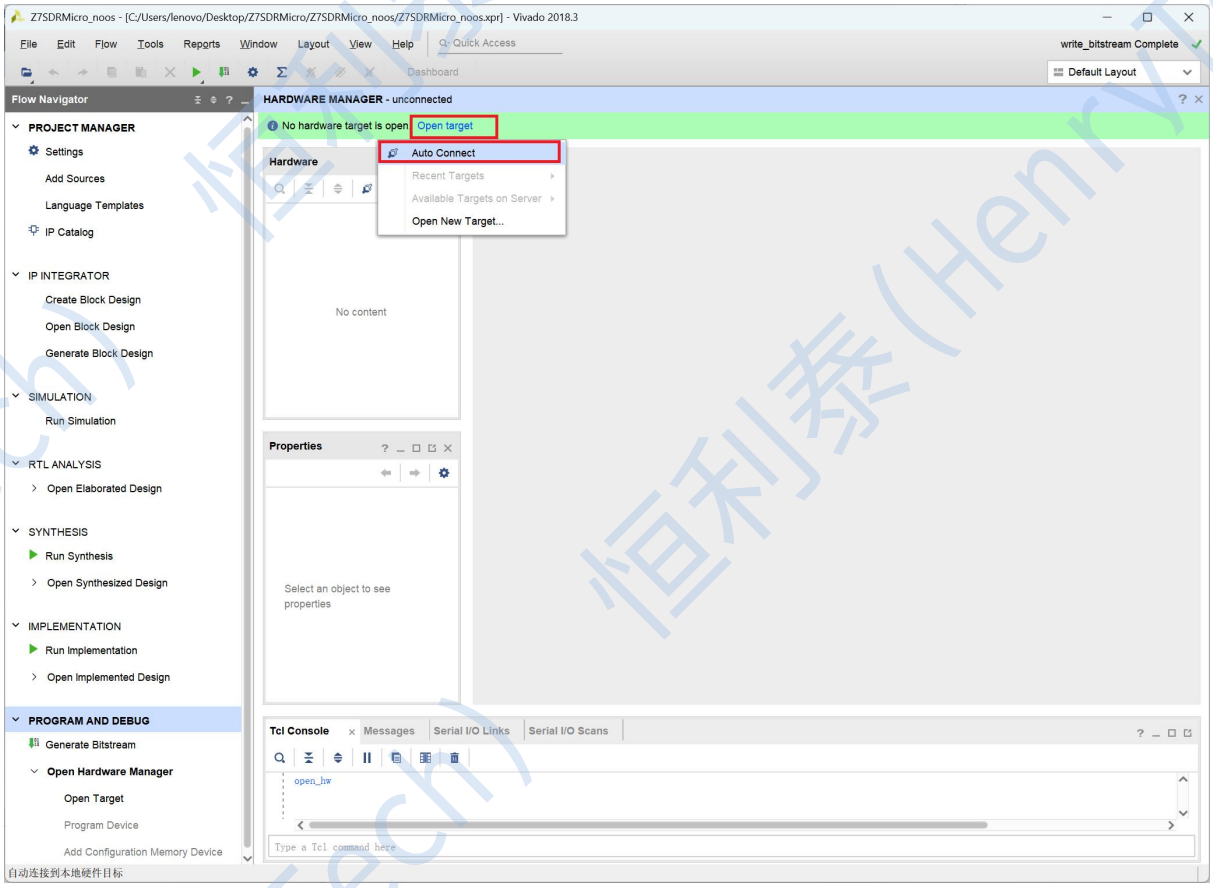
我们在点击运行按钮之前可以打开一个我们自己熟悉的的串口助手，设置好波特率115200，串口号，然后打开串口。这里串口如何查看设备管理器的串口号，串口工具如何配置等不做讲解，大家根据自己电脑系统和自己平时使用习惯的串口工具来定。我们点击了小虫子之后SDK先烧写BIT文件，我们点击小三角符号 运行。运行后，串口助手会打印出如下初始化信息，表明已成功初始化：



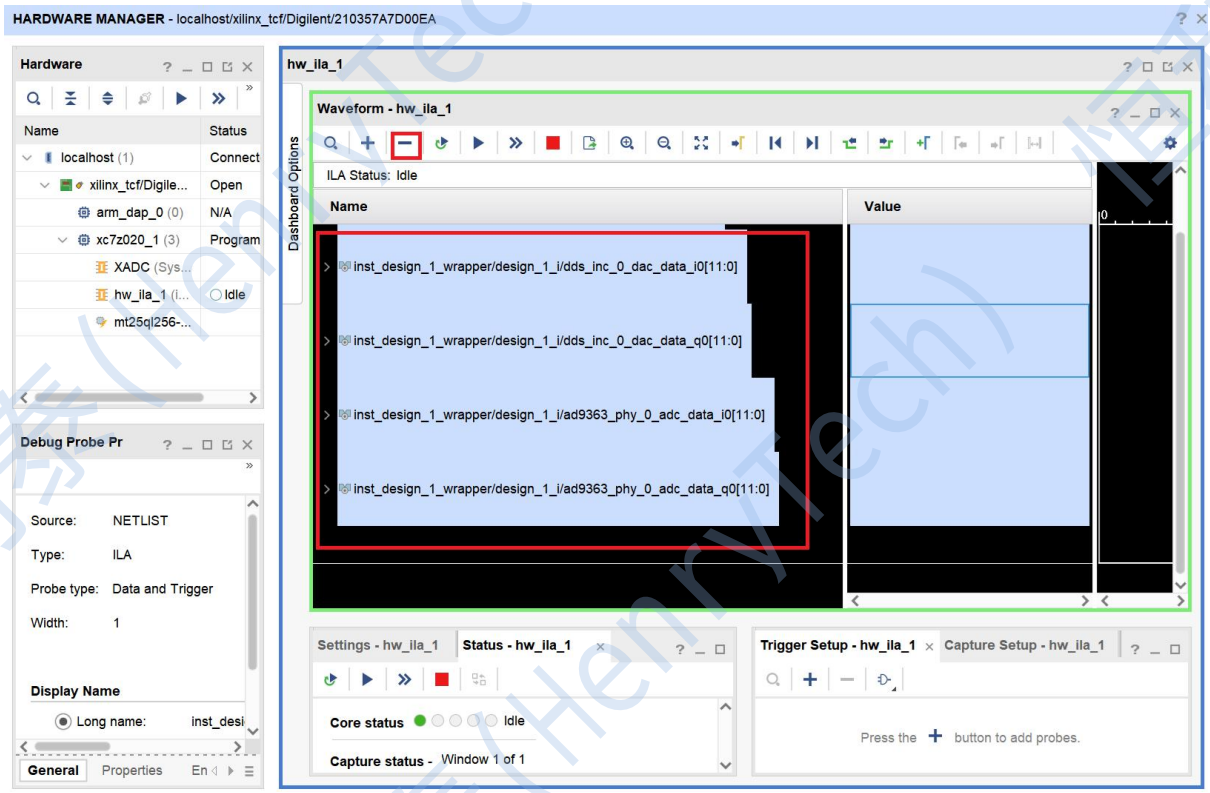
我们到这里，SDK就成功运行。接着我们查看FPGA的接口数据，进行信号抓取。我们回到VIVADO软件，我们在软件左边最下方找到 Open Hardware Manager硬件管理器打开：



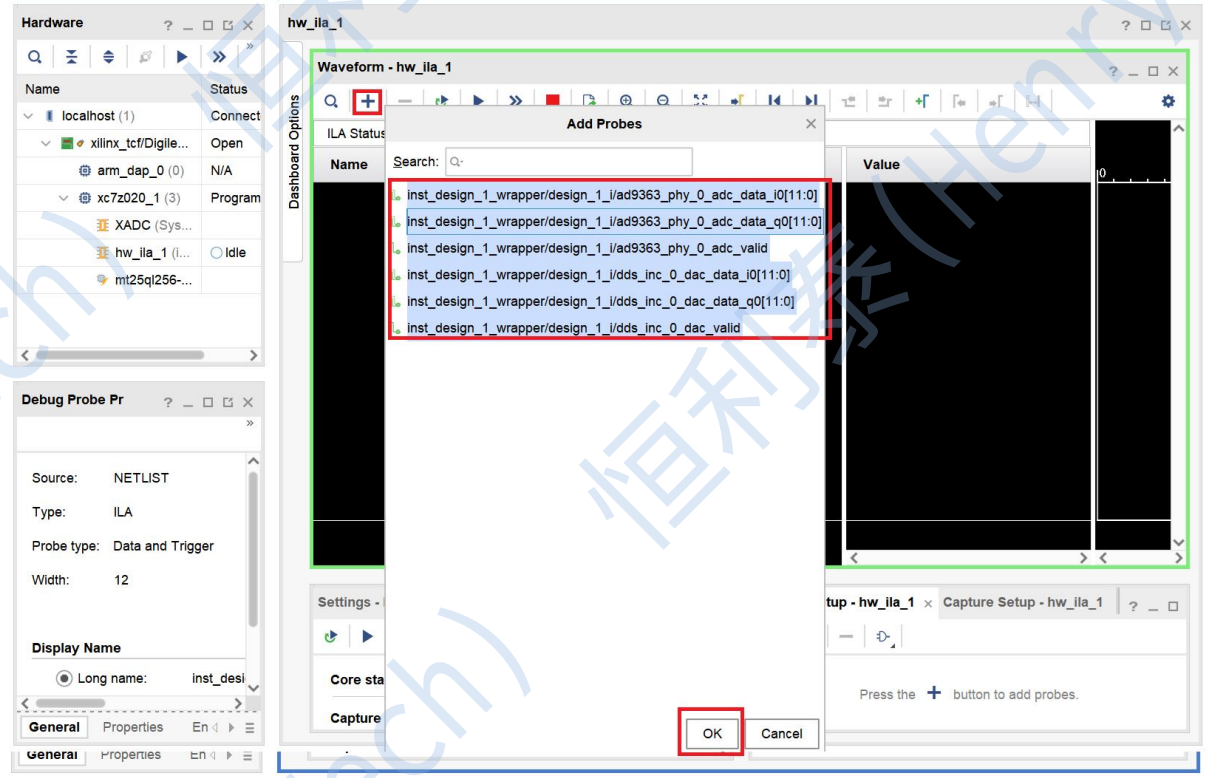
弹出的界面点击Open target：



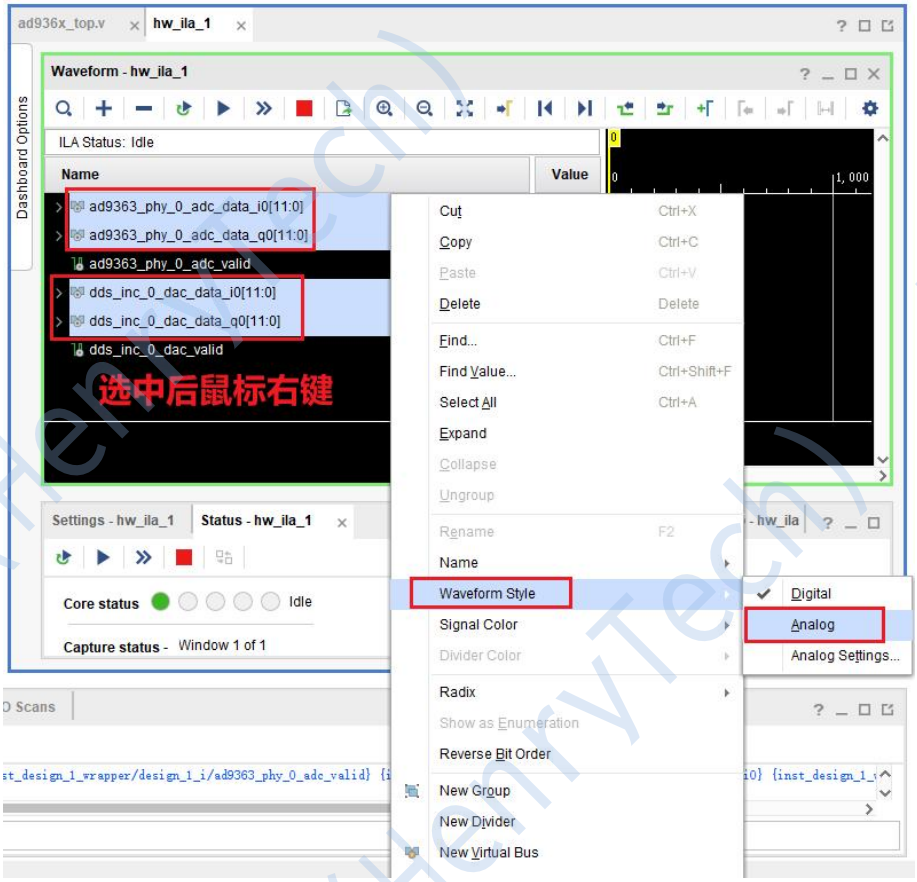
然后等待连接打开。打开之后界面如下，我们可以看到已经识别了FPGA的调试ILA模块。注意，打开的ILA界面左边是要抓取的信号列表，我们左上方有+ 和-信号，可以往当前界面添加或者删减信号。关于这里建议大家可以将信号列表全选，按ctrl+a，点击 - 号全部删除之后可以点击 + 号将所有信号全部重新添加进来：



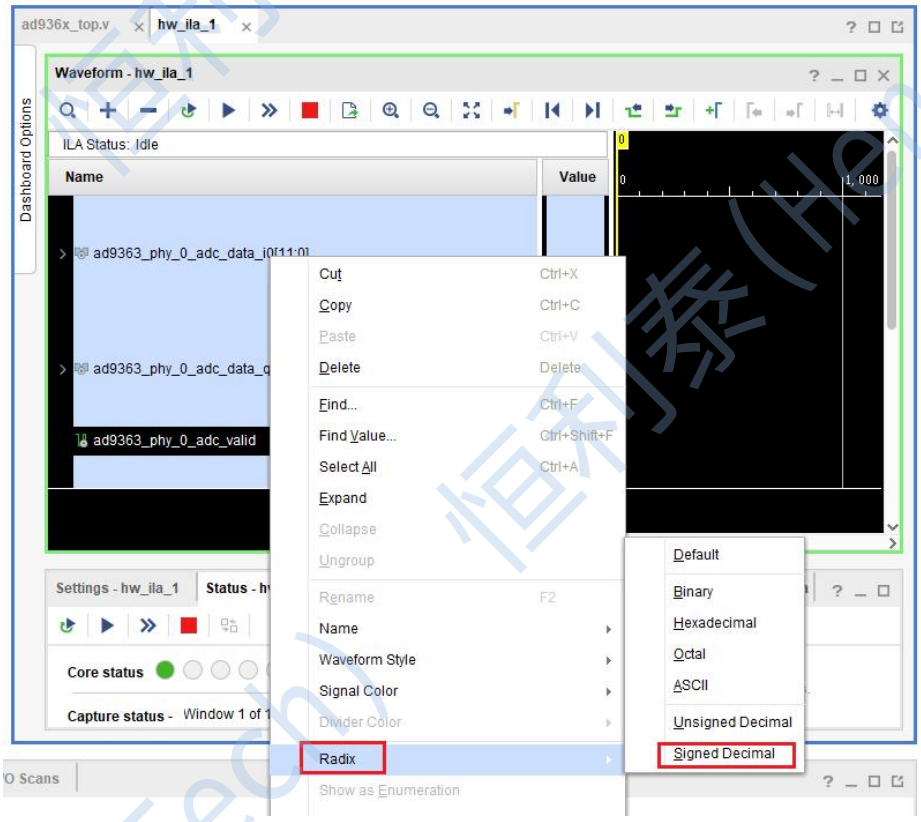
全部删掉之后点击+号，信号可以ctrl+a全选，然后点击OK添加：



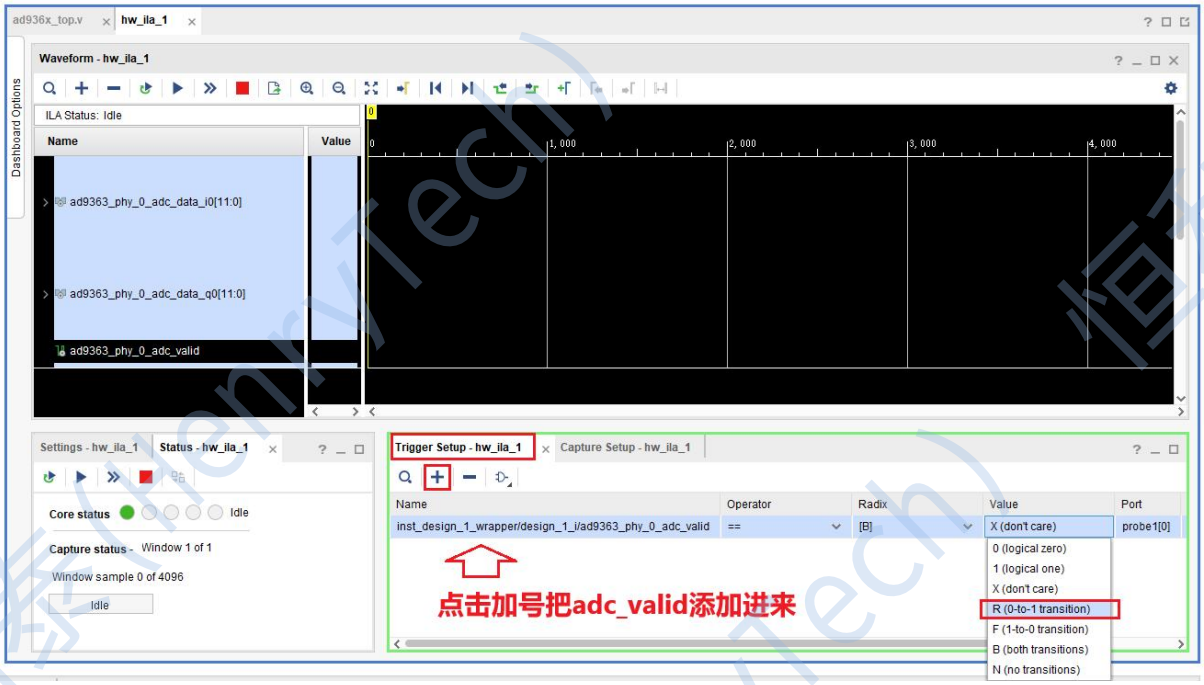
接着我们设置信号显示方式，我们将2组adc和2组dac等选中，右键找到Waveform style设置成Analog显示，即可显示为模拟曲线：

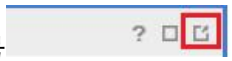


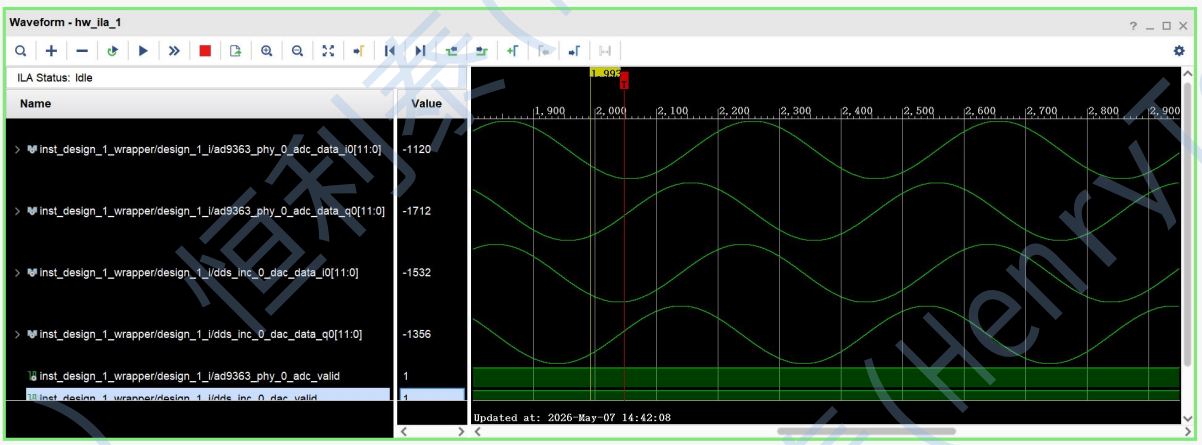
完成波形显示设置之后，大家可以再设置一下符号。上图中设置了信号模拟显示之后，同样的选中刚设置的这几个信号，大家可以再检查一下Radix这一项，然后选择Signed Decimal表示有符号数，如果不是这一项可能波形显示会出错：



接着配置采样。Triger setup设置我们添加adc_valid采样有效信号进来，然后右边的值选择R上升沿触发，表示adc有效上升沿触发采样抓取：



配置好后我们点击三角符号 ▶ 抓取，抓取后可以看到我们抓取的2路ADC和DAC数据，我们窗口不好调整查看可以点击上角这里红色标记的符号  将窗口独立出来，再调整窗口大小以便查看，最后看到波形如下，能看到在波峰处幅度iq几乎一致：

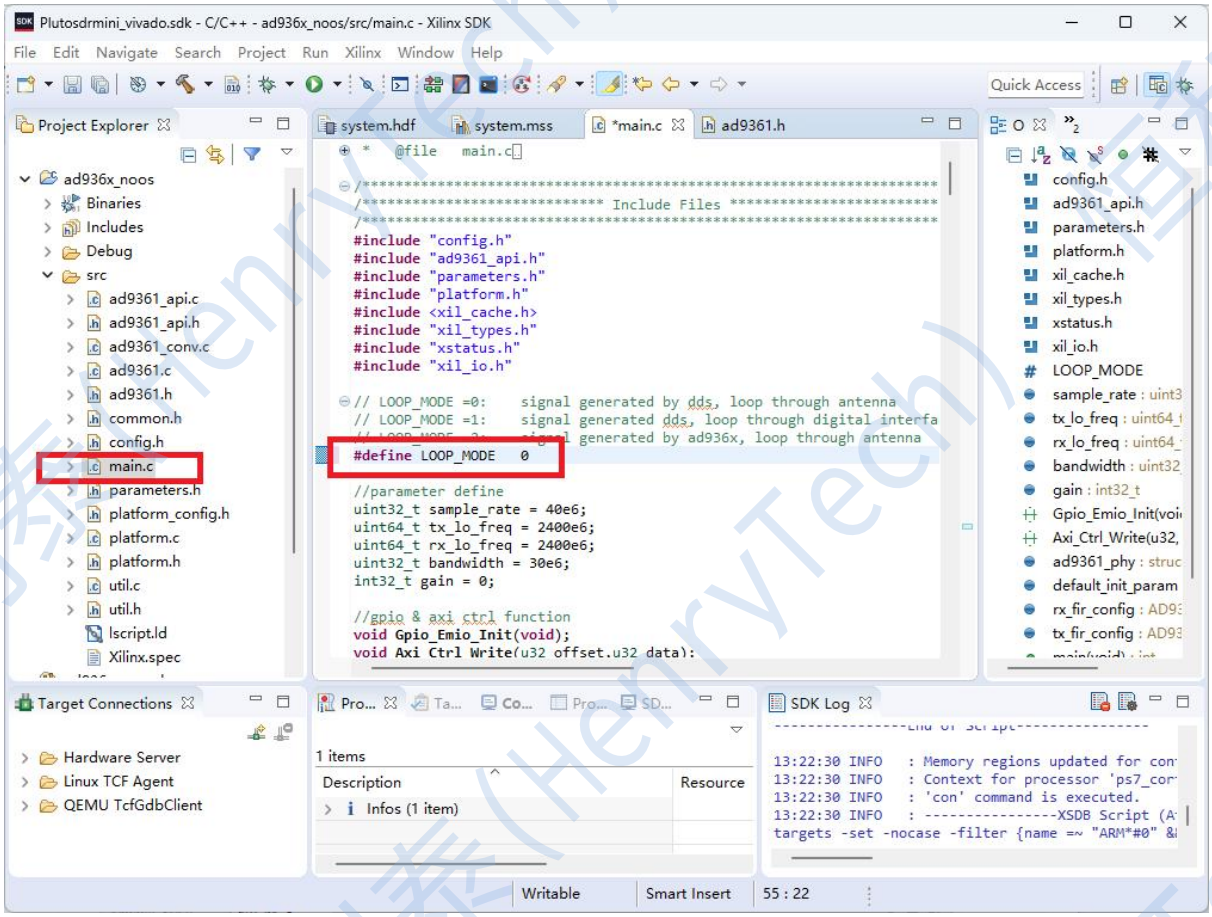


通过上述代码我们验证了我们给9361输出和输入的数据波形基本匹配。关于上述测试波形输入输出完全一致，由于9361发射信号，我们知道，信号可能会有环境影响或者衰减，为什么会出现完全一致的情况呢？这是因为我们代码中，打开了数据接口直接回环测试功能。我们不经过无线收发器前端ADC和DAC，直接接口数据回传回来，方便用户调试接口时序。接下来一节我们讲一下如何使用天线进行信号传输采样。

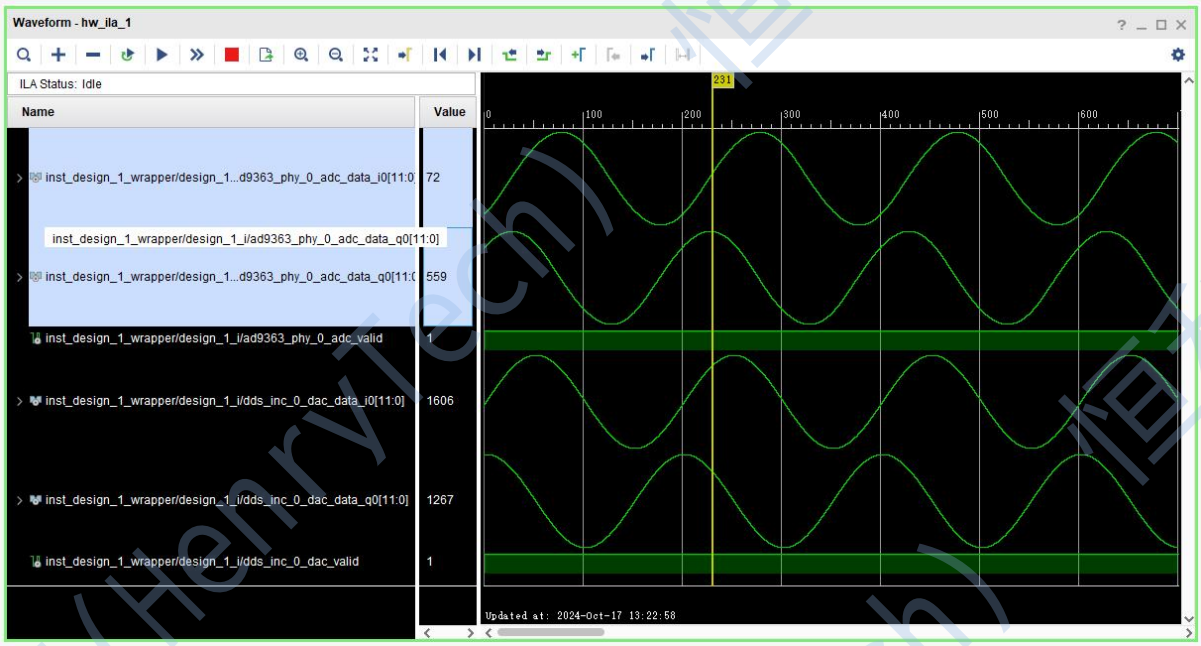
十二、SDK常见配置修改

接上一节，我们通过信号接口数据直接回环，我们接下来修改一下代码，数据通过

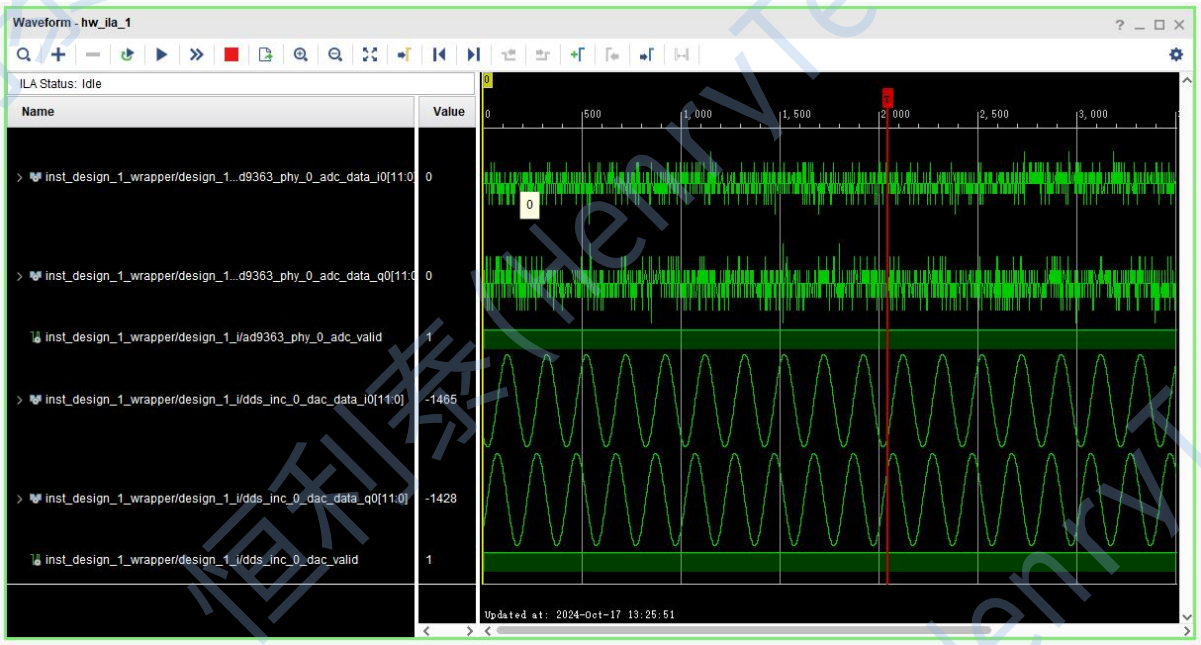
9361无线接口进行回环，所谓接口回环的话指的是数据不通过天线接口传输，直接在内部CMOS进行回环，给进去什么数据，出来的就是什么数据。我们将SDK工程的main.c中开头部分的LOOP_MODE宏定义改为0即可，含义在上方三行注释可以看到，这个宏定义取值范围0，1，2分别对应的测试关系：



更改测试模式之后我们点击SDK上方的锤子 工具编译之后再次点击小虫子运行，然后烧写BIT后我们点击 运行符号运行，我们再次回到vivado，重新打开硬件管理或者刷新一下，重新抓取信号，可以看到我们抓到的数据不一样了，数据不是接近12bit满量程的，这是由于ADC DAC输入输出经过接口，可能由于配置功率，衰减，增益等，不可能刚好达到满量程的情况，数据曲线任然是正余弦。关于信号曲线，理论上dac i q和adc i q应该几乎重合，但是由于936x芯片无线发射出去回环，可能存在被反转180°问题，也就是波峰变成波谷，属于正常现象，另外由于输出到接口，有数据打拍，以及输入也有打拍，所以抓取的数据有几个时钟周期的延迟，dac和adc数据无法完全重合，属于正常现象：



拔掉SMA，抓取数据如下，就是幅度很小的噪声：



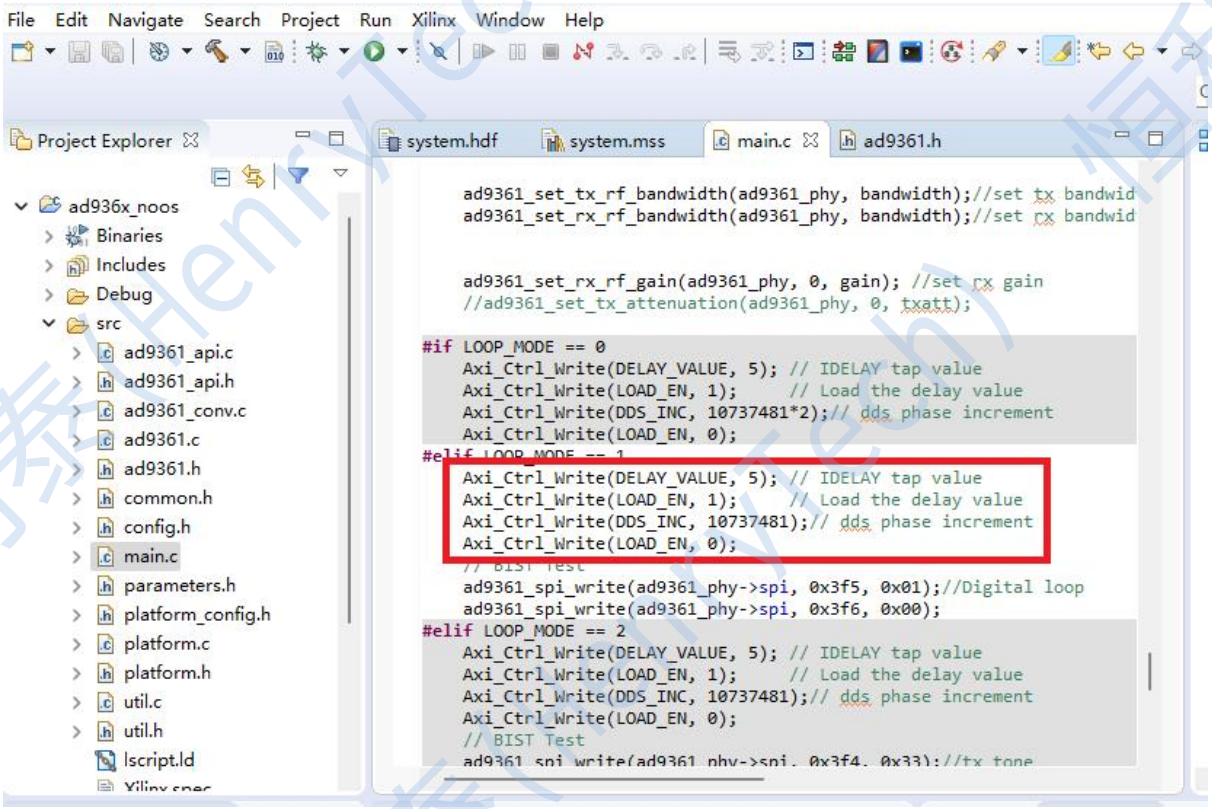
除了使用SMA信号做无线回环实验，我们还可以尝试不同的天线进行实验。注意，天线需要自己购买内针内螺纹SMA天线。实验的时候可能会出现天线太近或者信号太强，出现波形变形削顶的情况，大家可以用SMA延长线将天线接长一点即可。

十三、常见参数配置修改

例程程中定义了一些参数和结构体，我们讲几个关键的配置。首先是采样频率，无线发射和接收频率，以及滤波带宽和增益等，这些在main.c最上方的参数定义中，另外在整个代码中最重要的配置参数结构体就是AD9361_InitParam default_init_param这个结构体，其中包含9361的接口时序模式，电平标准，晶振频率等，以及无线参数，IO的信号delay时序。大家注意，9361我们在接收发送时序代码中，RX部分使用到i_delay来调整时序，实际上9361可

以通过配置修改这个结构体相关的配置项，来调整9361的接口时序。对于接口的时序调整比较灵活。关于结构体每一项，都有相应的注释，大家可以对照查看。

另外补充一点，关于FPGA逻辑中的RX部分的I_DELAY的Tap延迟值，和DDS的相位增量设置。IODELAY设置需要将LOAD_EN写1,然后写0生效，DDS_INC值默认10737481，这个决定DDS输出的正余弦频率，关于这个值如何得来，这里不做介绍，大家详细参阅DDS IP核手册即可：



最后再讲一下几个常见变量，位于main.c开头部分：

```
//parameter define
uint32_t sample_rate = 40e6;
uint64_t tx_lo_freq = 2400e6;
uint64_t rx_lo_freq = 2400e6;
uint32_t bandwidth = 30e6;
int32_t gain = 0;
```

- sample_rate：采样率。这个参数决定ADC DAC每秒钟采样多少个点；
- tx_lo_freq:发射无线电本振频率；
- rx_lo_freq：接收无线电本振频率；
- bandwidth：带宽：以本振为中心频点，其无线电的射频带宽。
- gain:接收增益。值越大，RX幅度越大。

十四、其他

我们在《Z7SDRMicro\02硬件资料\芯片手册》资料如下目录提供的数据手册列表：

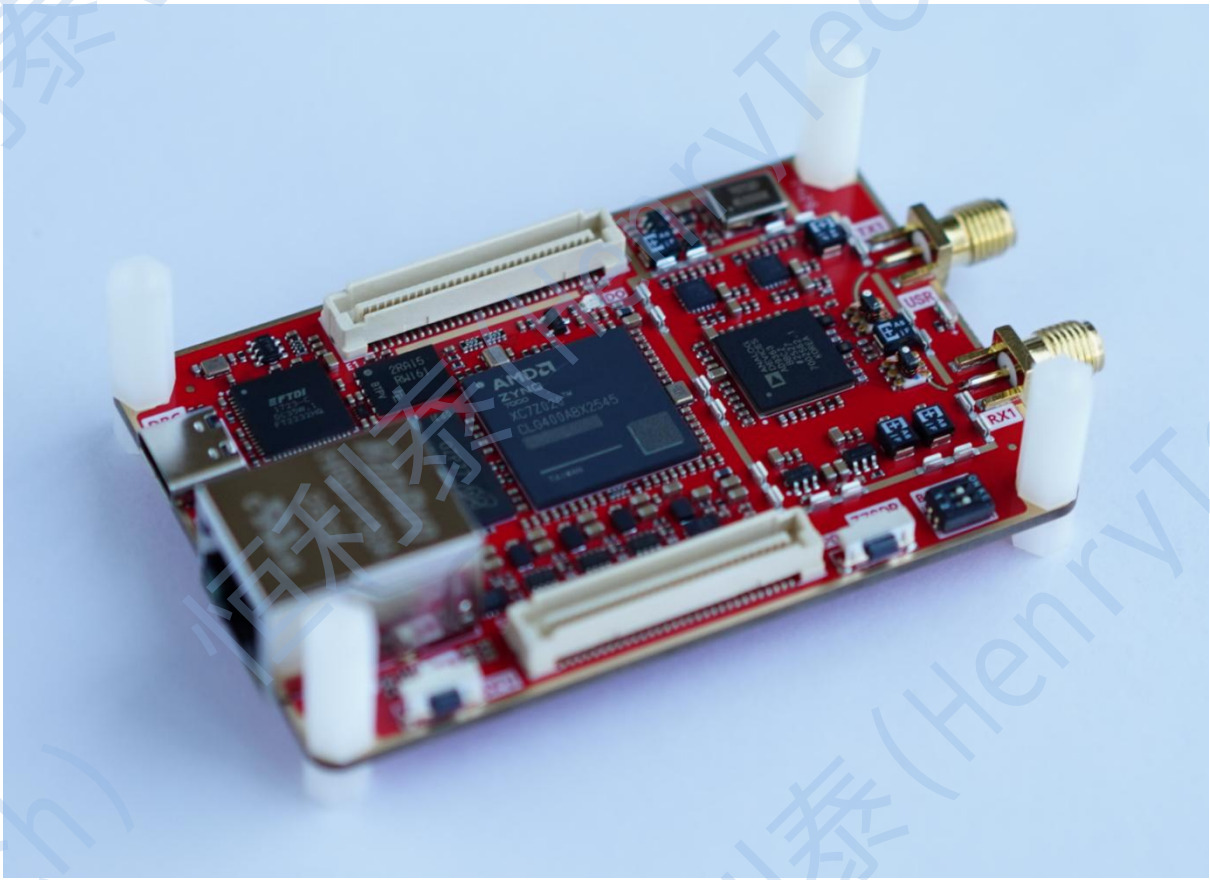
ZYNQ主芯片	2024-09-22 20:41	文件夹	
其他xilinx官方手册	2024-09-22 20:41	文件夹	
AD7291.pdf	2024-09-22 20:21	Adobe Acroba...	526 KB
AD9361.pdf	2024-09-22 20:21	Adobe Acroba...	633 KB
AD9363-Reference-Manual-UG-1040.pdf	2024-09-22 20:21	Adobe Acroba...	2,732 KB
AD9363-Register-Map-Reference-Manual-UG-1057.pdf	2024-09-22 20:21	Adobe Acroba...	1,024 KB
CH340.PDF	2022-11-15 10:23	Adobe Acroba...	169 KB
MAX809T.pdf	2022-11-15 10:23	Adobe Acroba...	1,950 KB
MP2143DJ-LF-Z.pdf	2022-11-15 10:23	Adobe Acroba...	499 KB
MP2145.pdf	2024-09-22 20:21	Adobe Acroba...	1,391 KB
MT41J256M16.pdf	2022-11-15 10:23	Adobe Acroba...	2,912 KB
Software-Defined-Radio-Solutions-From-ADI.pdf	2024-09-22 20:21	Adobe Acroba...	842 KB
TCM1-63AX+.pdf	2024-09-22 20:21	Adobe Acroba...	414 KB
tps74801.pdf	2024-09-22 20:21	Adobe Acroba...	1,899 KB
UG570.pdf	2024-09-22 20:21	Adobe Acroba...	2,564 KB
usb3320c.pdf	2022-11-15 10:23	Adobe Acroba...	1,398 KB

由于9363和9361配置寄存器和使用一致，除了参数性能上的区别。所以资料里配置寄存器和技术手册名字为9363，这个不影响，大家均可参考这一套。

我们作为带大家简单入门学习如何使用为目的，受限于笔者的水平和经验，笔者接触9361也时间不长，也非专业无线开发人员，所以文档中更深的一些专业知识没有涉及，需要大家自己花更多时间精力去研究，笔者后续有更好的例子再分享，本文档到此结束。

ubuntu18.4虚拟机的安装

------ 基于Z7SDRMicro开发平台



目录

十五、文档介绍

本教程给大家讲解如何安装使用VirtualBox虚拟机，然后配置好虚拟机，安装ubuntu18.4版本。我们后续将会在虚拟机ubuntu18.4搭建我们的GNURadio软件无线电开发环境。针对绝大多数学习9361的软件无线电，虚拟机+ubuntu是首选，因为很少有人安装实体机ubuntu，所以我们编写本教程也是为了方便大家。如果有条件或者闲置电脑也可以安装实体机，但是需要自己研究如何制作优盘启动ubuntu镜像进行安装。这里不对实体机安装ubuntu进行说明。

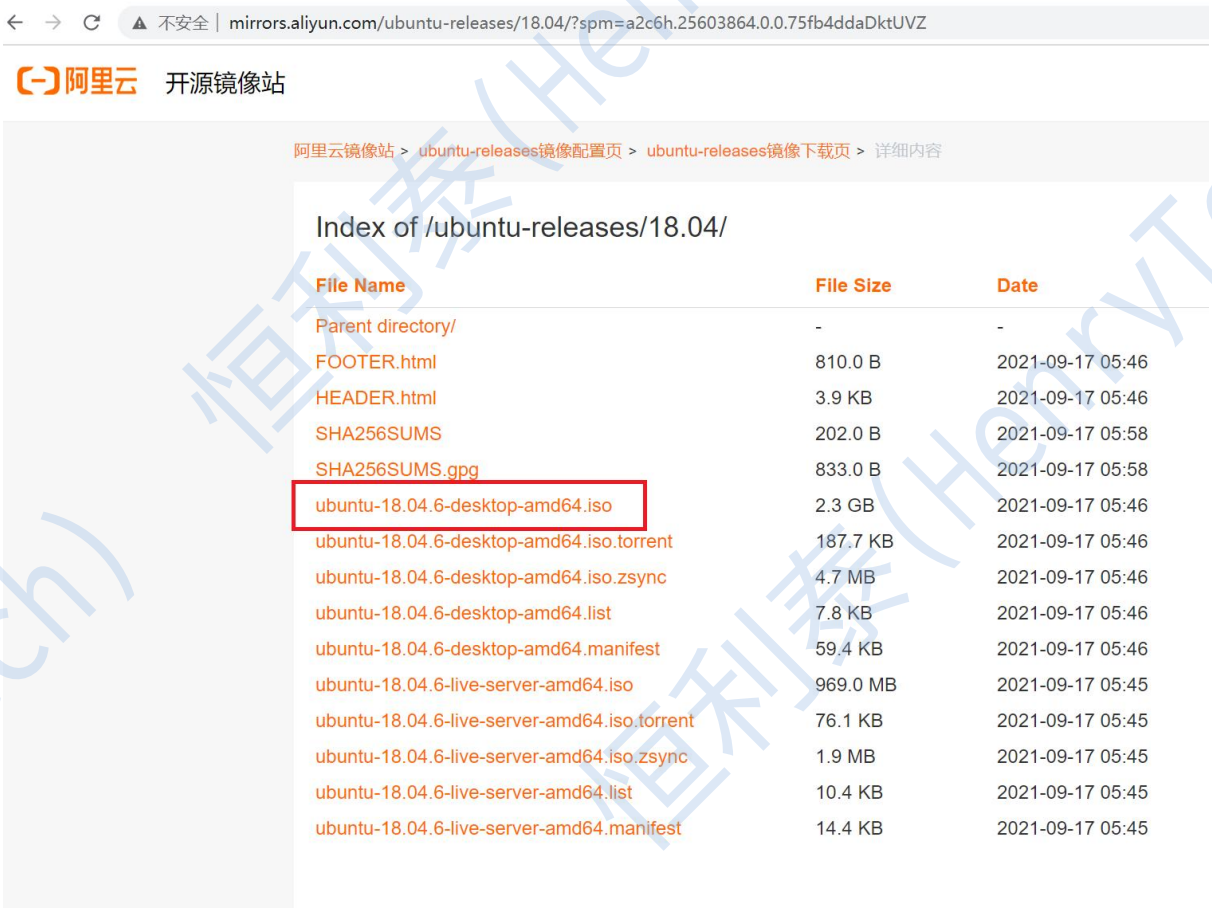
一、Ubuntu镜像获取

我们通过百度云提供ubuntu-18.04.6的镜像。下载地址很多，关于ubuntu18.4下载。我们提供的版本全名是：ubuntu-18.04.6-desktop-amd64。对于本教程，大家只需要

关于镜像文件下载，大家可以在以下链接下载：

http://mirrors.aliyun.com/ubuntu-releases/18.04

我们打开下载链接找到一个大约2.3G的ISO镜像文件下载即可：

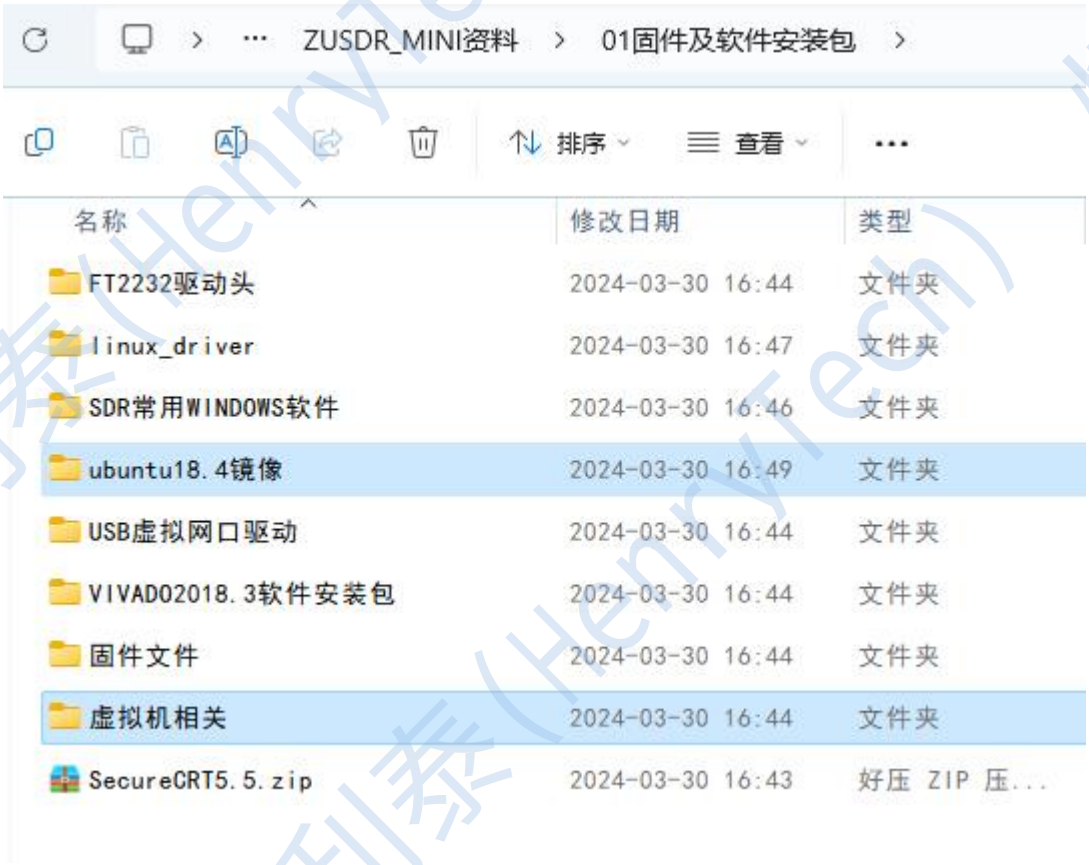


关于ubuntu镜像，笔者这里没有尝试其他版本，建议与我们文档保持一致的版本。由于不同版本之间的工具链版本一致性问题。当然也有一些开发者习惯使用一些特殊版本，如果在安装过程中出现任何不一致的情况，需要自行解决，新手初学建议与我们文档一致。

二、虚拟机安装

虚拟机有好几种，这里我们只介绍Oracle VM VirtualBox，我们的安装包所在位置，其中包含一个扩展包iso，这里暂时用不到。我们只需要安装VirtualBox-6.0.12-133076-Win即可。

本节的安装相对灵活，虚拟机软件有最新版的，不推荐使用其他版本，大家最好保持一致，遇到问题可以根据我们教程进行核对检查。我们安装所需要的几个安装包和镜像位于如下位置：



我们双击《虚拟机相关/VirtualBox-6.0.12-133076-Win.exe》启动一路next，中途弹出任何对话框均点击YES或者是就可以了：

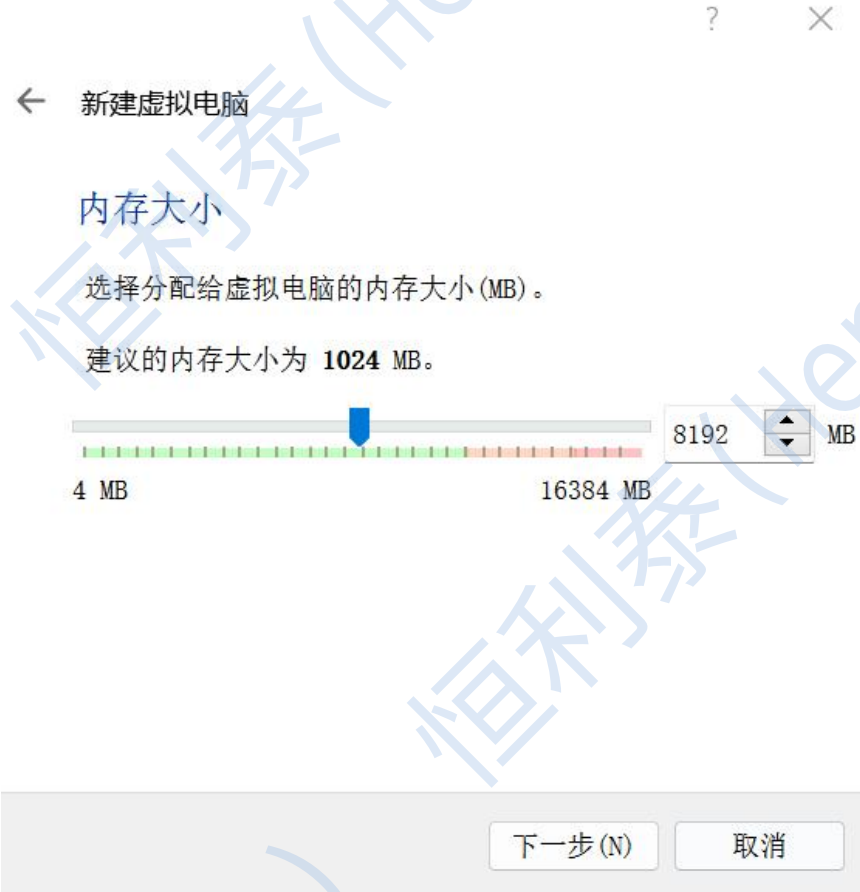
安装完成之后点击完成，就启动了。启动之后会弹出有更新版本，我们不用理会。

三、虚拟电脑配置新建

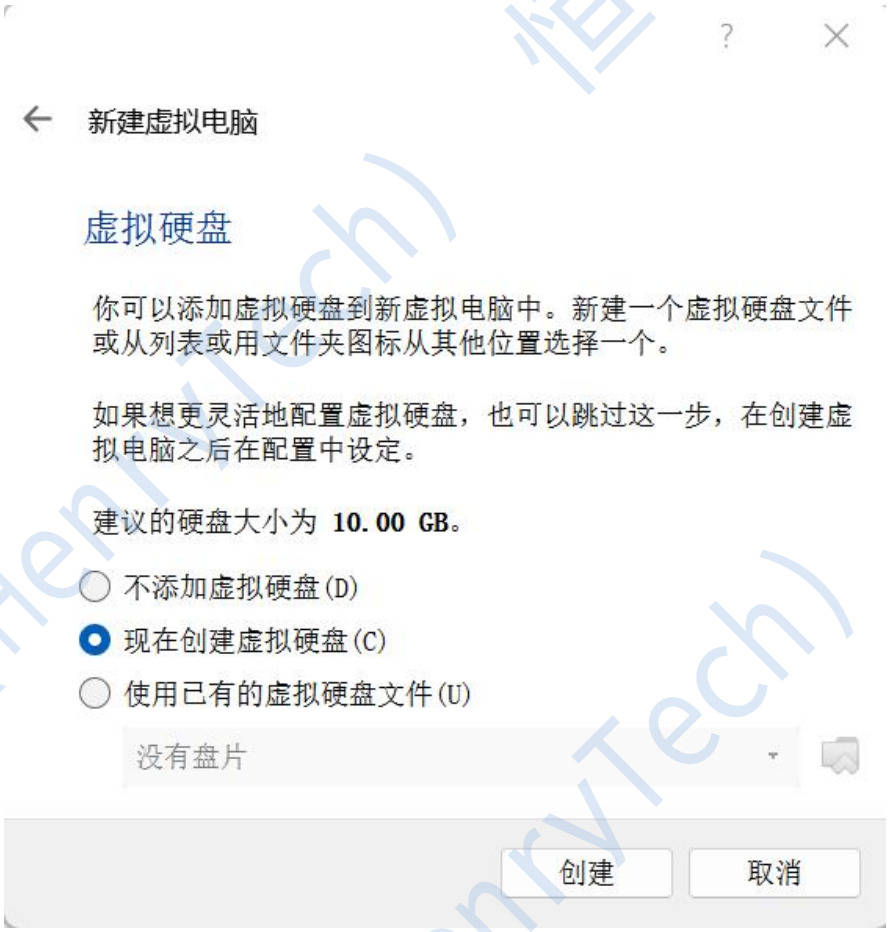
我们点击新建，建一个虚拟电脑：

然后新建的名字自己取一个，这里我们是ubuntu18.4，类型默认linux，版本Ubuntu(64bit)然后点击下一步：

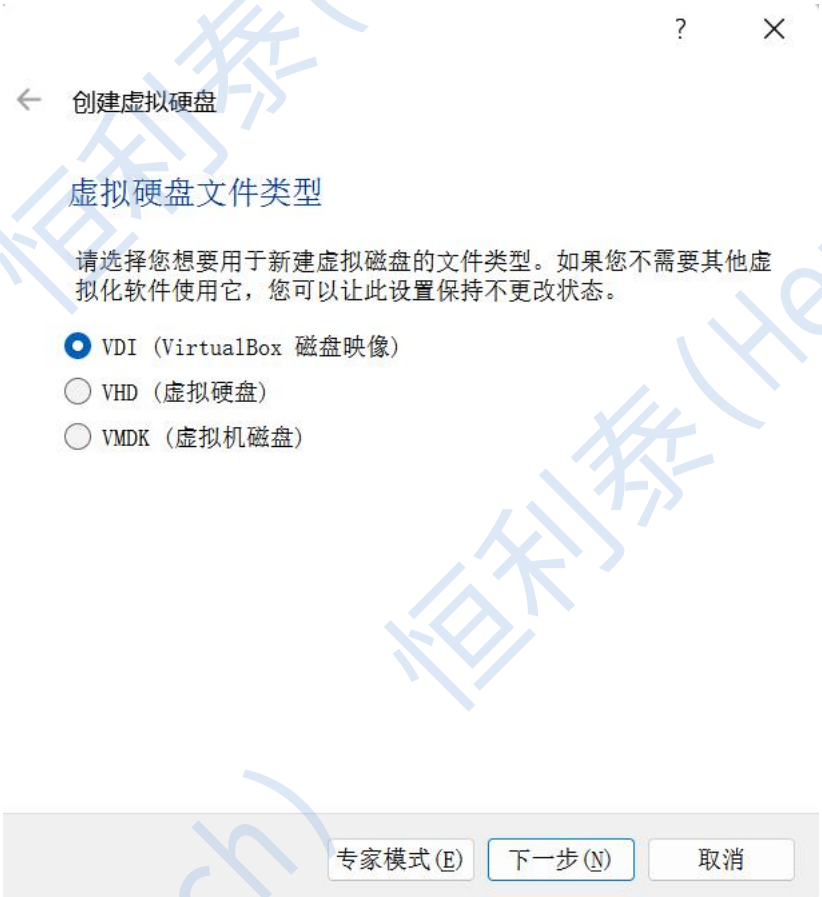
接下来这一页决定配置虚拟机运行内存。个人建议大家配置大一点，一般为电脑实际内存的一半。笔者电脑的内存是16G。我们分配8192大小，配置好了继续点击下一步：



这一页让创建虚拟硬盘。虚拟硬盘就是一个很大的文件，用于运行虚拟机的ubuntu的虚拟硬盘。我们选择默认的现在创建虚拟硬盘即可，点击创建：

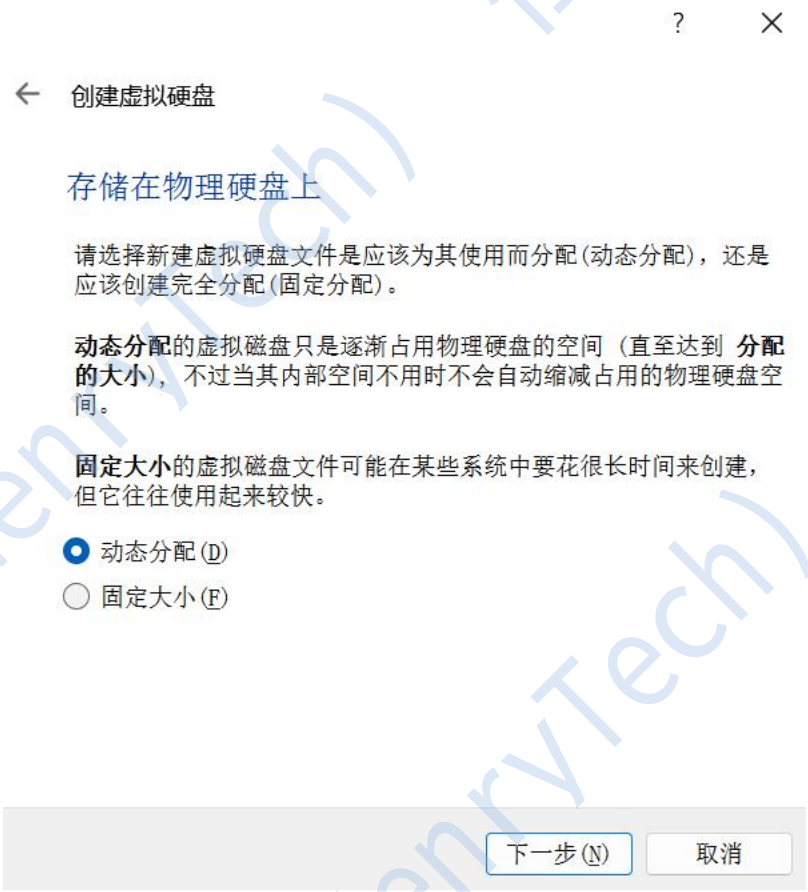


接下来的一页我们默认第一项就可以，使用VDI格式的磁盘映像：



接下来一页我们选择动态分配。动态分配就是可以节约空间，会根据实际使用逐步占

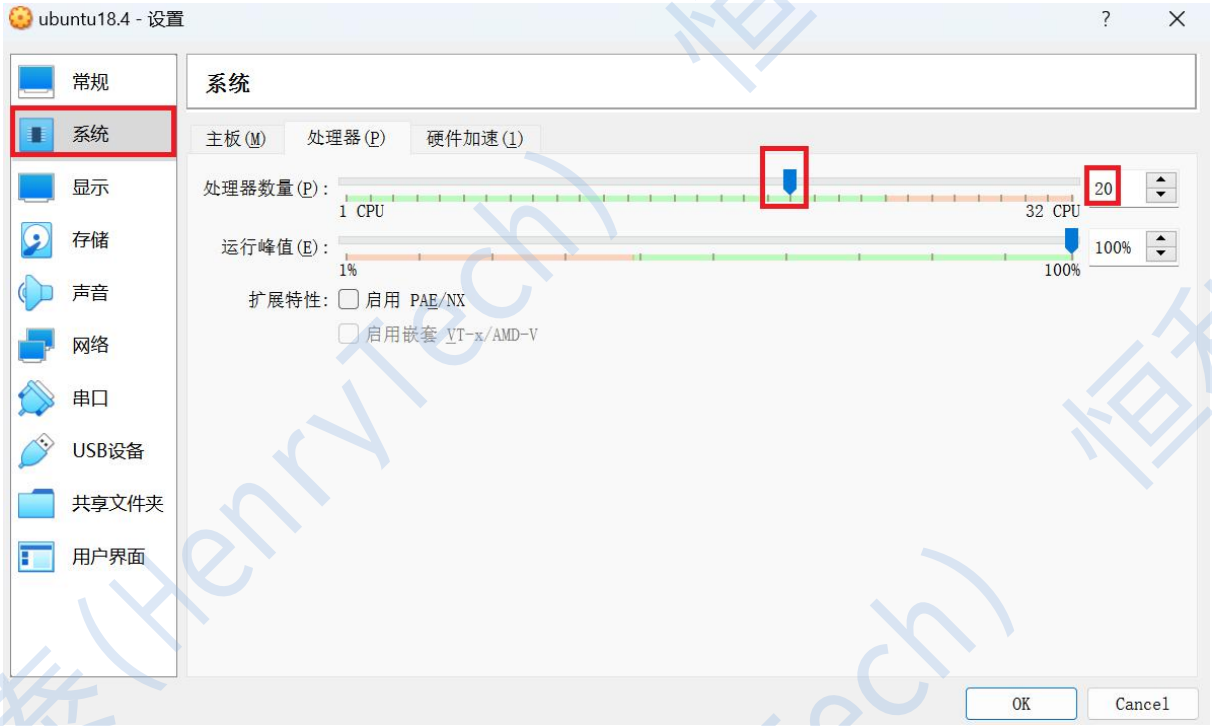
用配置的大小容量：



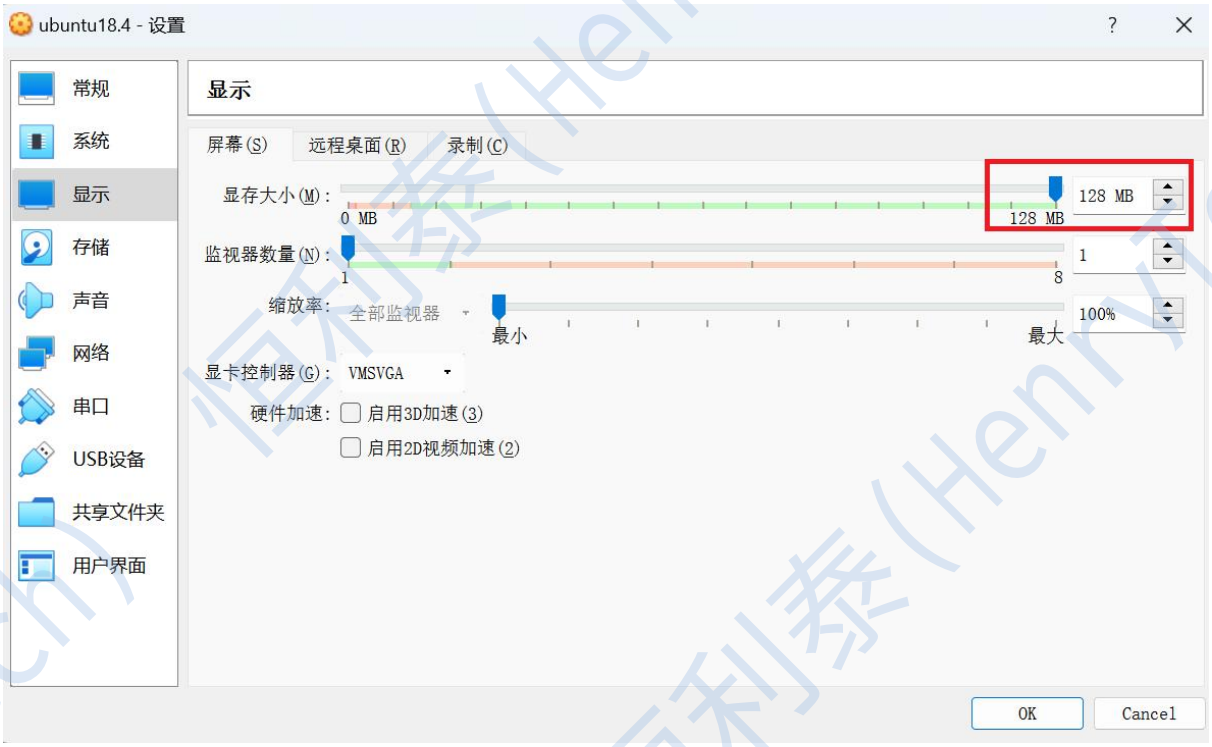
选择好动态分配后点击下一步，我们配置新建的vdi文件路径及大小。我们可以在硬盘新建一个叫做vm_diskfile的文件夹存放我们虚拟磁盘文件。建议使用固态硬盘的目录下，可以提高运行性能速度，大小可以转动滑块，我们大约90G就够用了，大家硬盘空间不够的话也可以小一点，建议至少保证50G：

点击创建即可完成创建，我们就回到主界面，可以看到左边出现了我们新建的ubuntu18.4虚拟电脑，我们这个时候再做一些设置，我们点击设置按钮：

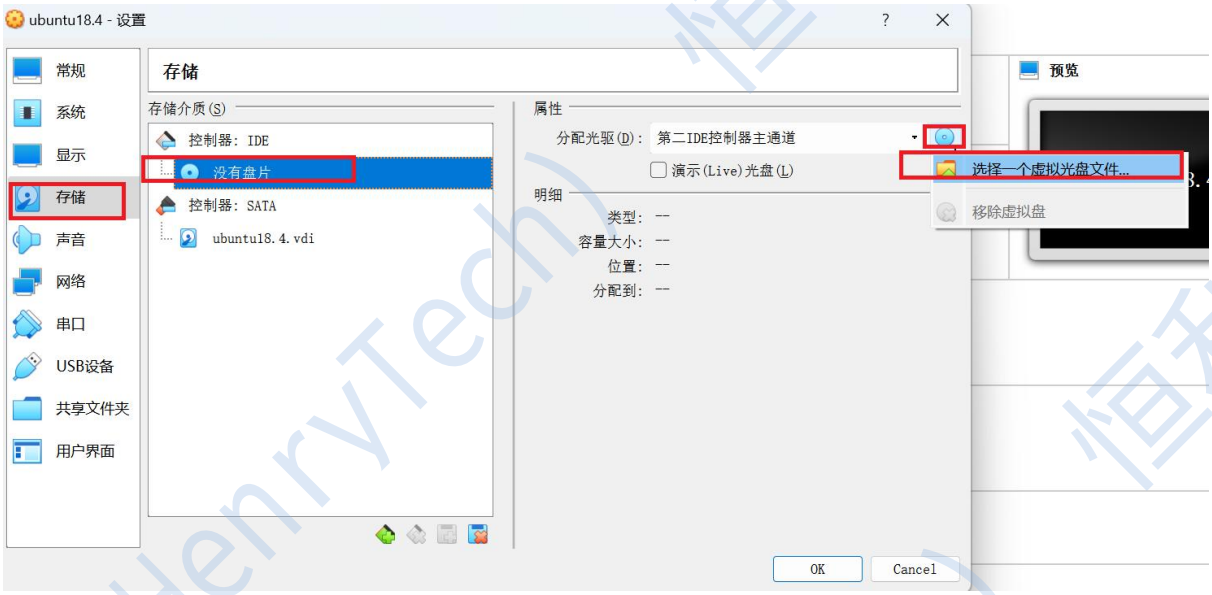
我们首先配置处理器，我们笔者这里使用的是I9 13900，所以可配置的处理器数量很多。大家根据自己电脑实际配置，虚拟机不要占完就可以了，针对其他处理器比如四核的，大多数CPU还有线程之分，这里CPU数量实际上是线程数，4核的处理器一般有8个线程，也有4个线程的，假如大家是8线程，建议配置6个：



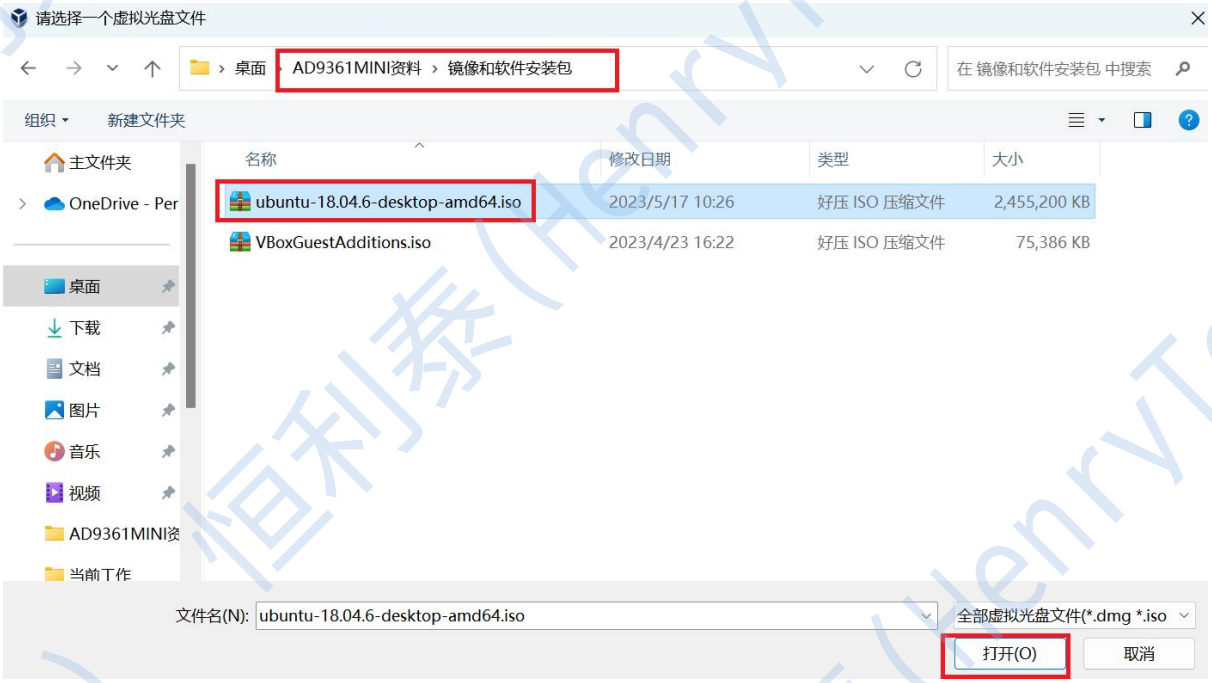
显存大小配置直接拉到最大，128M，这里不要太小，否则显示不够流畅或者出现卡顿等情况，虚拟机对显存容量大小要求不高，但是同时建议也不要太小，推荐128M：



接下来配置存储。我们知道，在之前的步骤我们新建了一块空的硬盘，还没有安装系统。所以我们要加载ubuntu18.4镜像的iso文件到控制器上，然后启动系统，安装到我们的虚拟硬盘。这里配置如下，我们右侧属性分配光驱点击光盘符号，选择一个虚拟光盘文件：



我们找到网盘下载的资料里面的镜像和软件安装包目录下的ubuntu-18.04.6-desktop-amd64.iso文件点击打开：

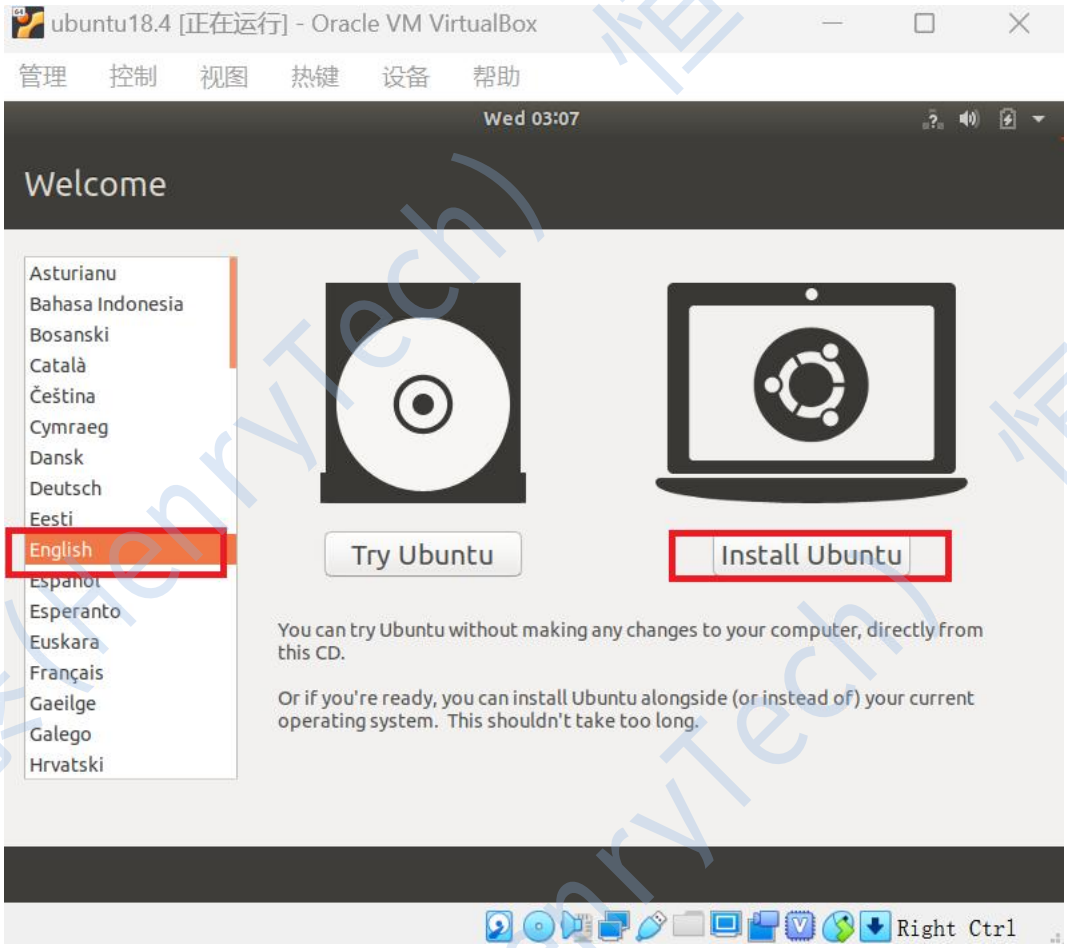


打开之后就可以看到我们控制器下方识别到ubuntu18.4的安装镜像盘片了。当我们启动虚拟机就可以从这里启动加载进行ubuntu安装。我们点击OK完成配置，退出回到我们虚拟机的主界面：

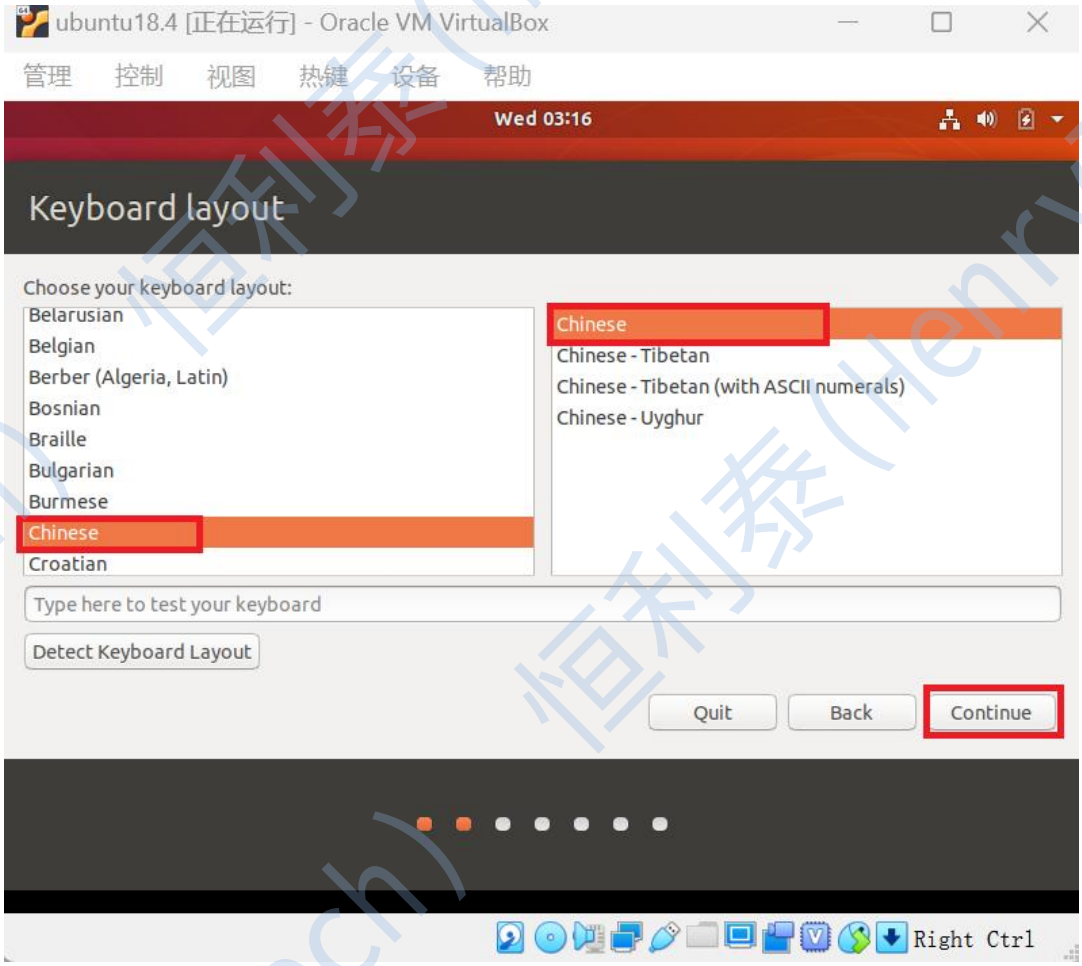
四、虚拟机启动安装ubuntu

回到主界面后我们点击启动，打开我们的虚拟机：

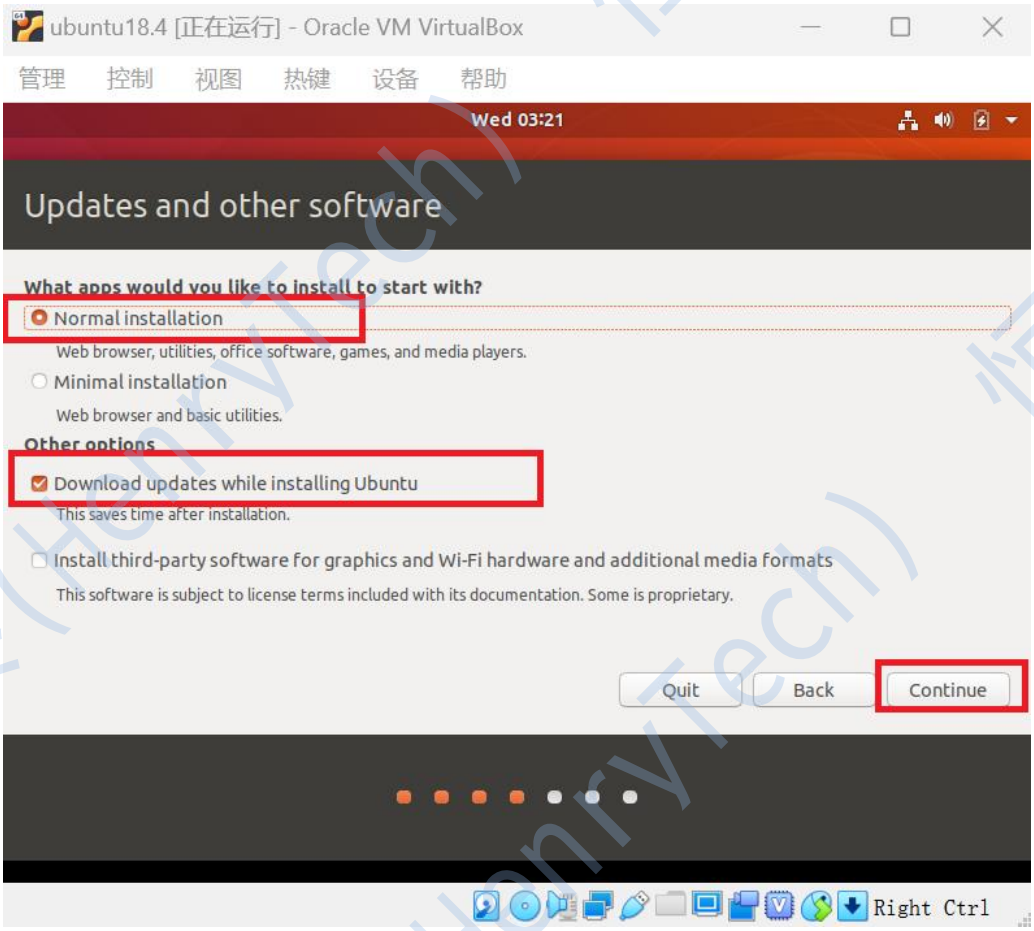
虚拟机刚开机的时候鼠标可能没办法移动，我们等待启动界面的安装系统出现，出现后，我们安装界面左边可以选择语言，我们下拉找到English英文，我们右边点击选择install ubuntu选项：


ubuntu18.4 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Wed 03:07
Welcome
Asturianu
Bahasa Indonesia
Bosanski
Català
Čeština
Cymraeg
Dansk
Deutsch
Eesti
English
Español
Esperanto
Euskara
Français
Gaeilge
Galego
Hrvatski
Try Ubuntu
Install Ubuntu
You can try Ubuntu without making any changes to your computer, directly from this CD.
Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.
Right Ctrl

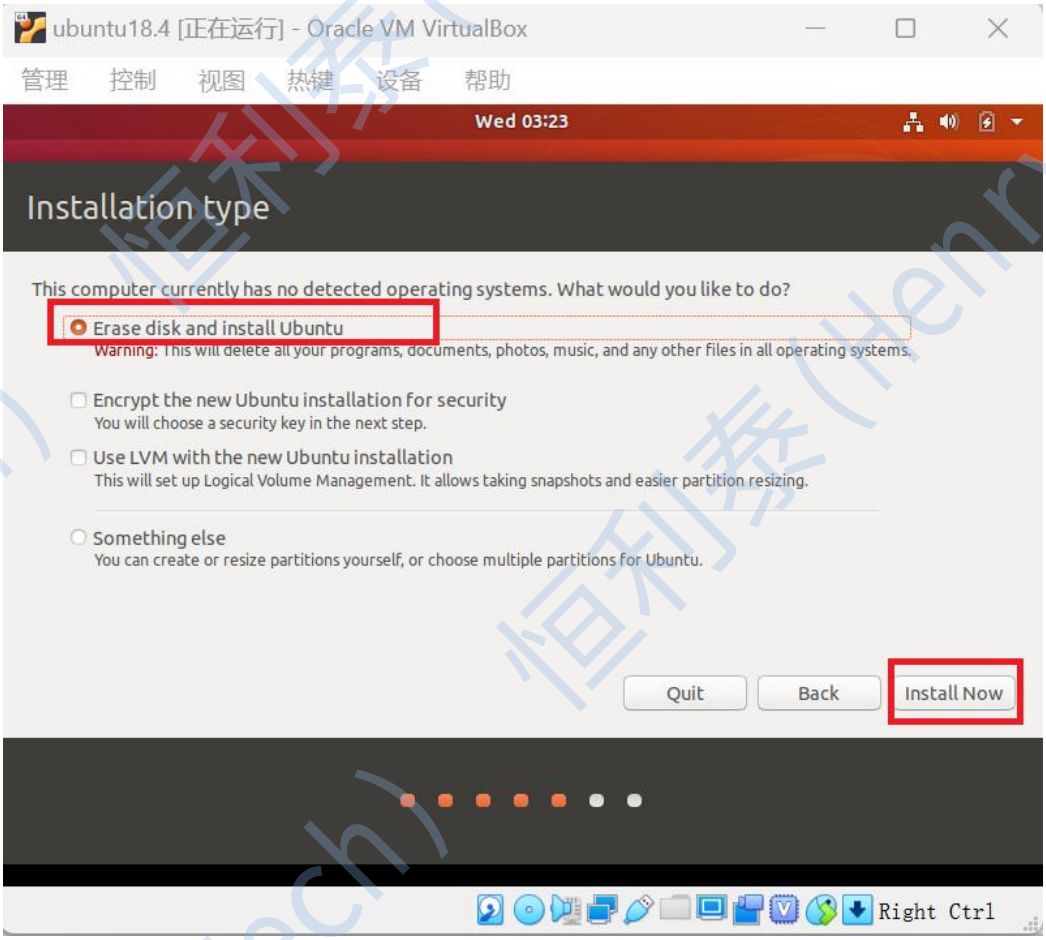
键盘布局默认：


ubuntu18.4 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Wed 03:16
Keyboard layout
Choose your keyboard layout:
Belarusian
Belgian
Berber (Algeria, Latin)
Bosnian
Braille
Bulgarian
Burmese
Chinese
Croatian
Chinese
Chinese - Tibetan
Chinese - Tibetan (with ASCII numerals)
Chinese - Uyghur
Type here to test your keyboard
Detect Keyboard Layout
Quit
Back
Continue
Right Ctrl

这一页默认勾选即可，点击Continue：

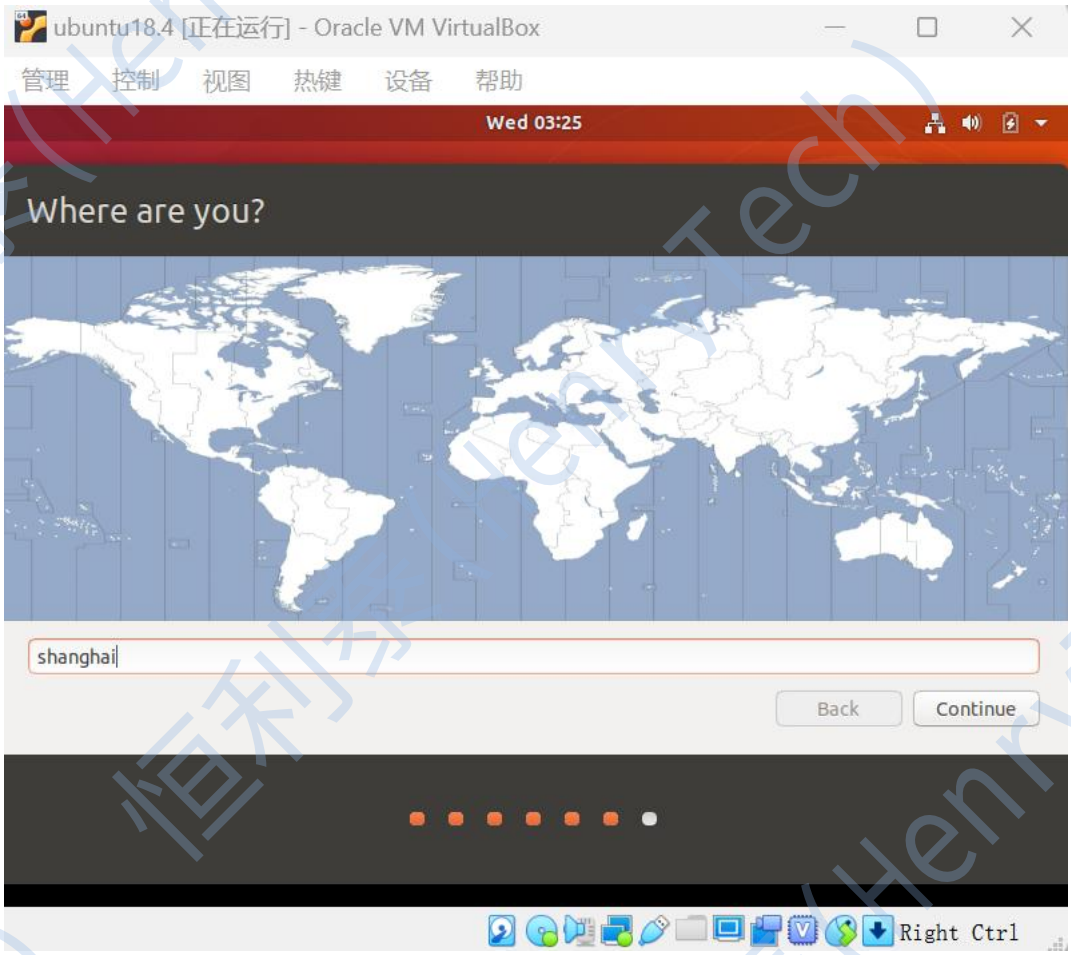


这一页默认第一项，点击Install now：

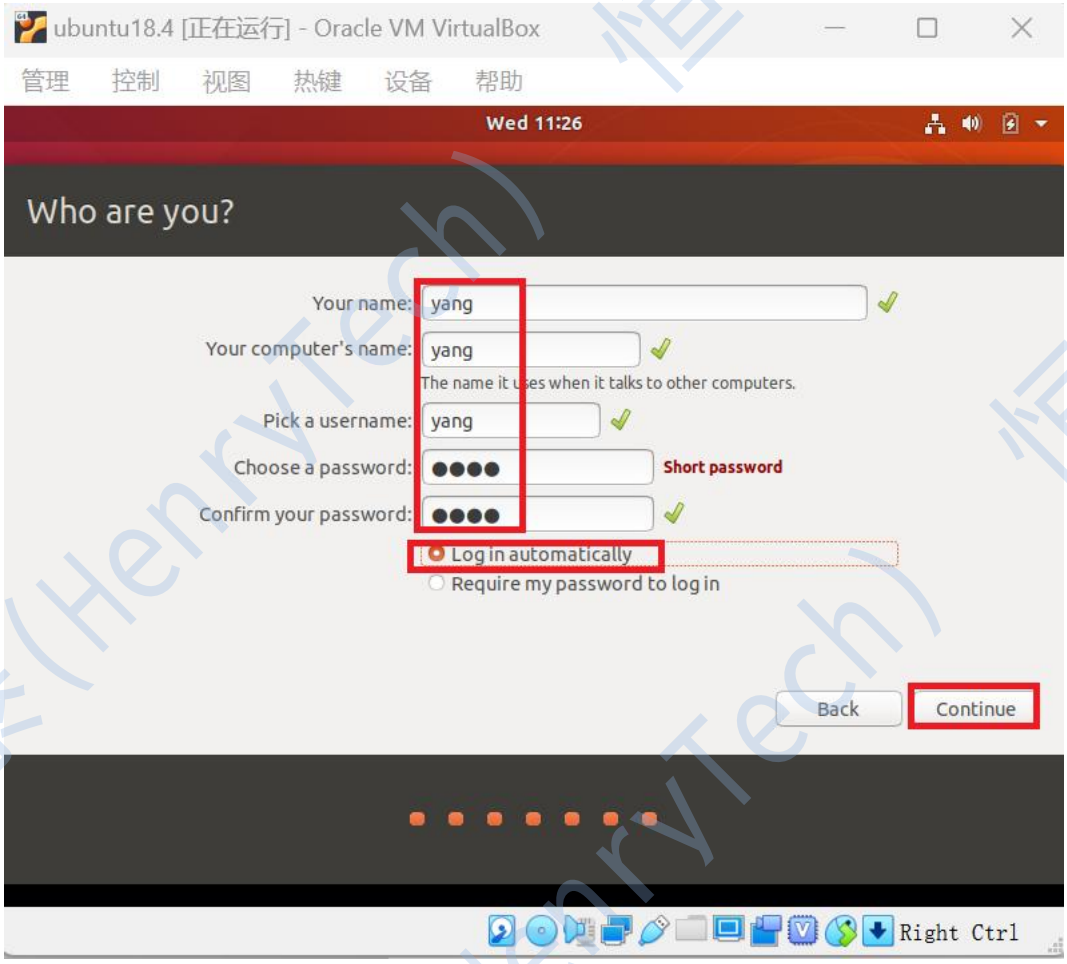


可能弹出对话框，点击Continue即可：

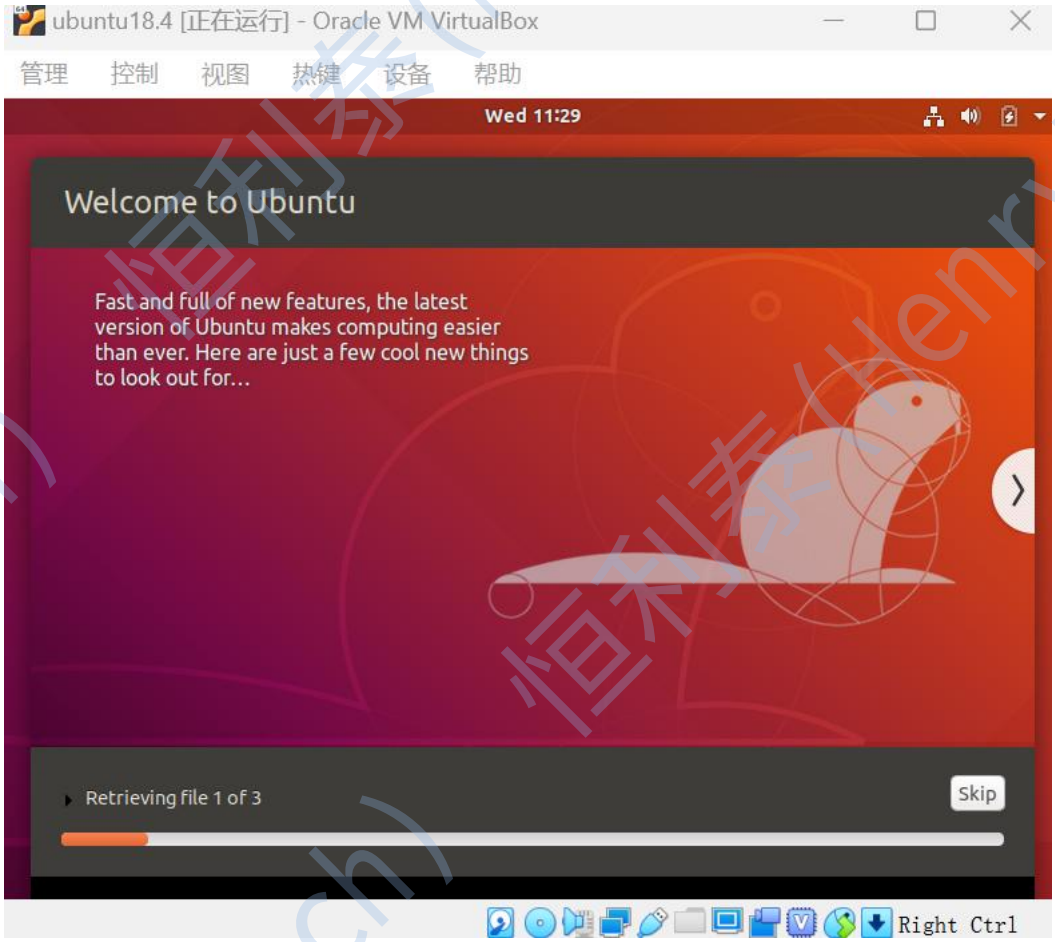
安装开始前，会提示输入所在城市，随便输入即可：



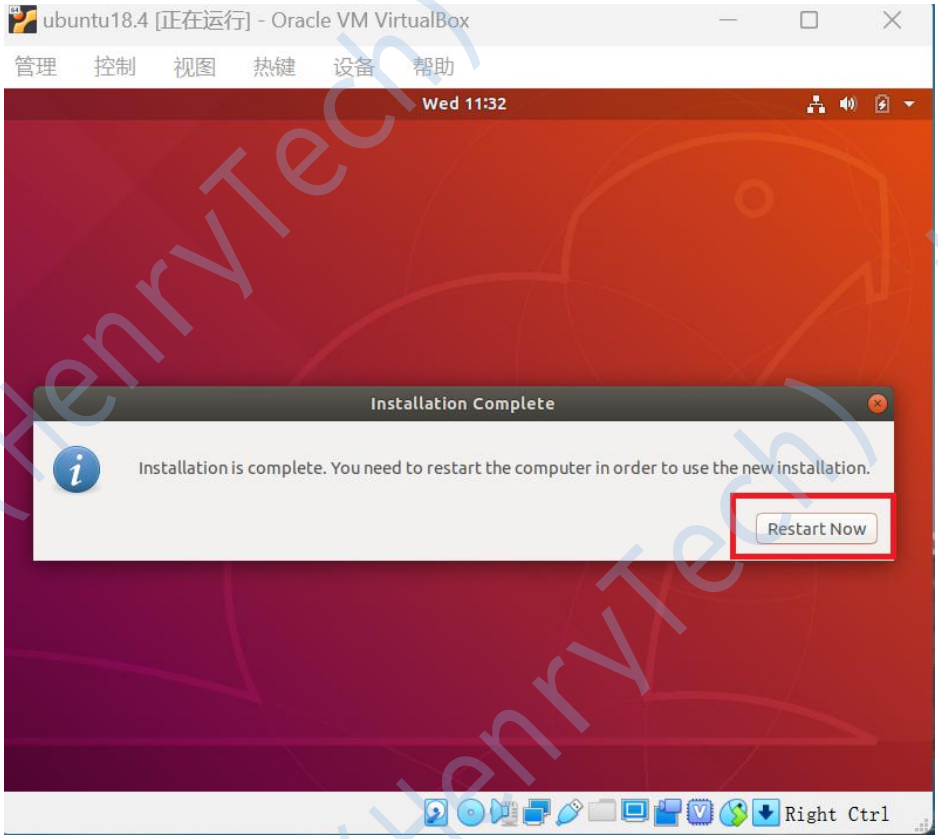
接下来这一页会让你输入用户名，计算机名，以及密码设置。这个密码是上方设置的用户名对应的密码，我们下方登录方式勾选Log in automatically自动登录方式，也就是说开机启动无需输入用户名和密码，系统自动进入登录状态，设置好后点击Continue继续：



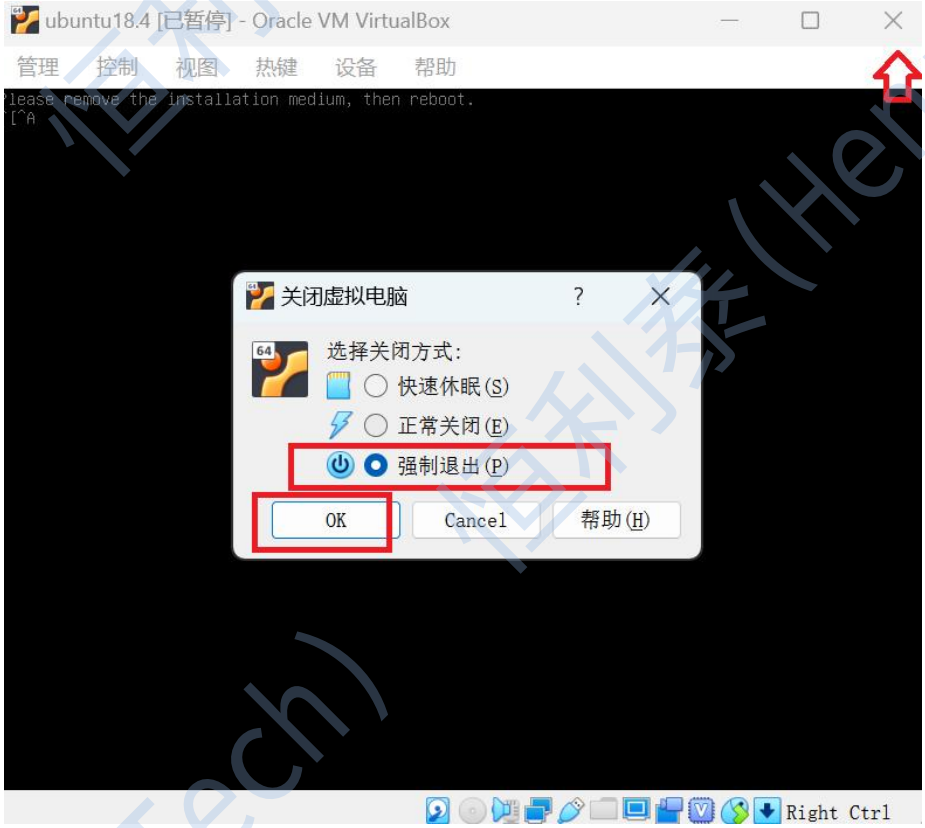
安装开始，我们耐心等待安装完成：



整个安装过程视网络稳定和电脑配置，时间不同，笔者大概6分钟时间完成安装，一般情况下普通电脑大约30分钟左右。大家等待安装完成即可，安装完成后弹出的对话框我们点击Restart now重启。

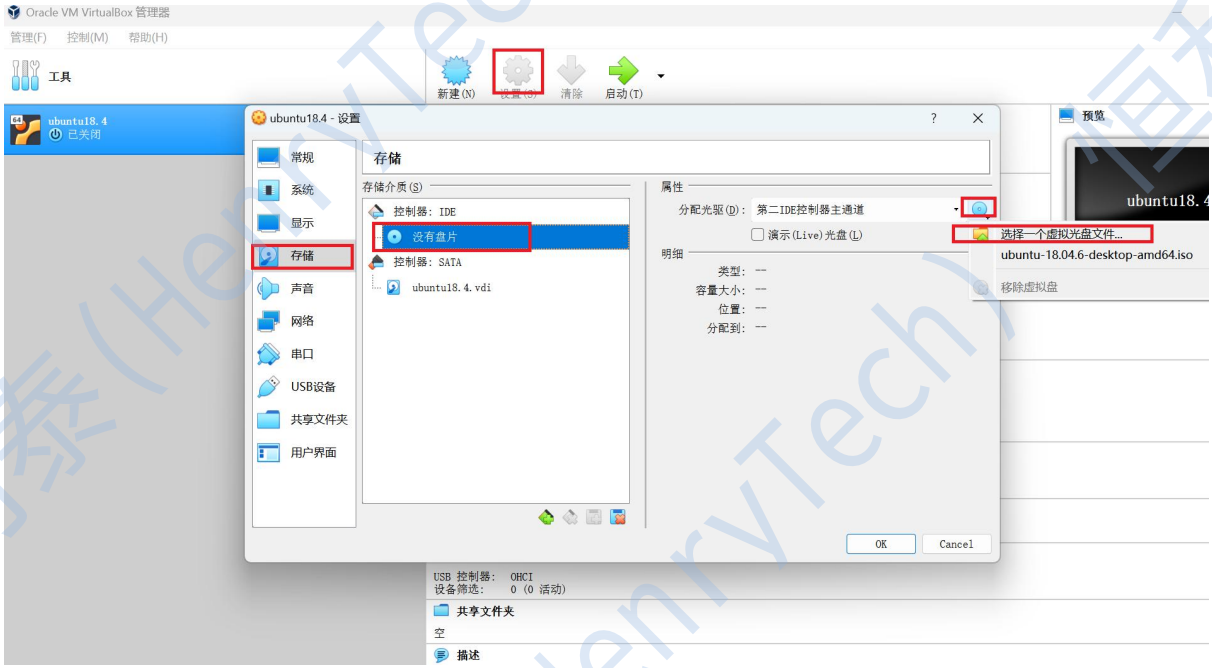


点击后我们弹出黑屏，无法关机，要用户弹出安装介质之后才会重启，但是我们是虚拟机，这种情况下怎么办呢？我们可以将虚拟机点击右上角的关闭按钮，然后选择强制关机，点击OK即可强制关掉退出：

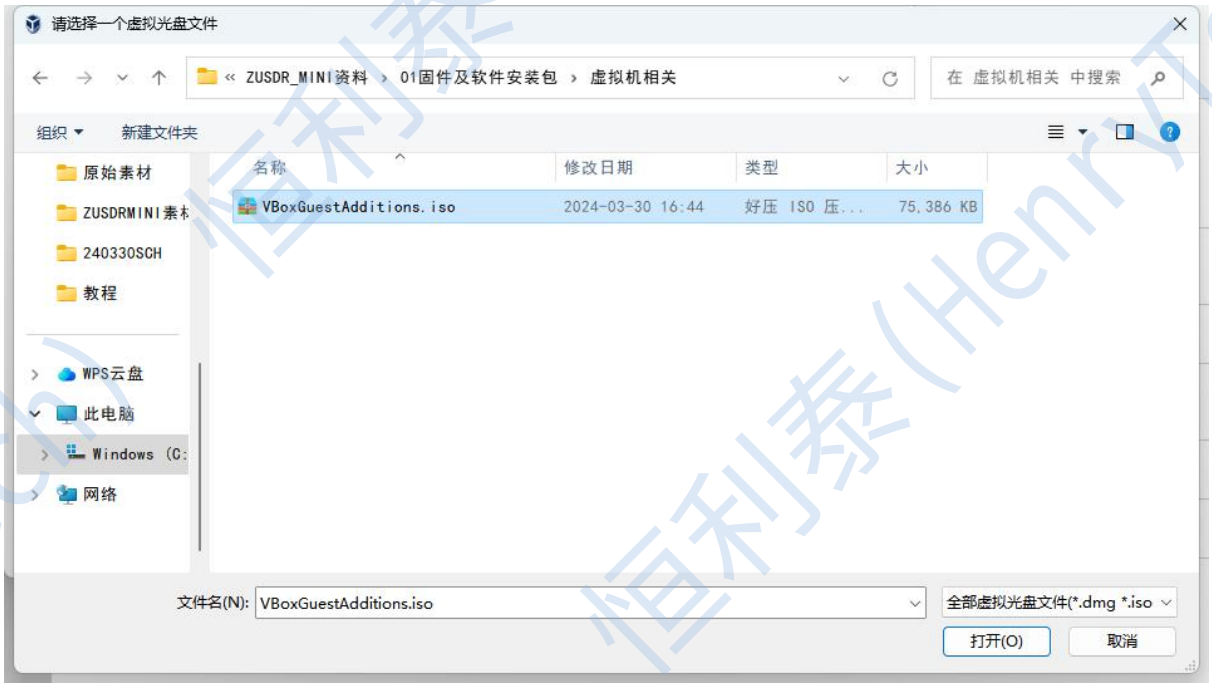


五、增强功能安装

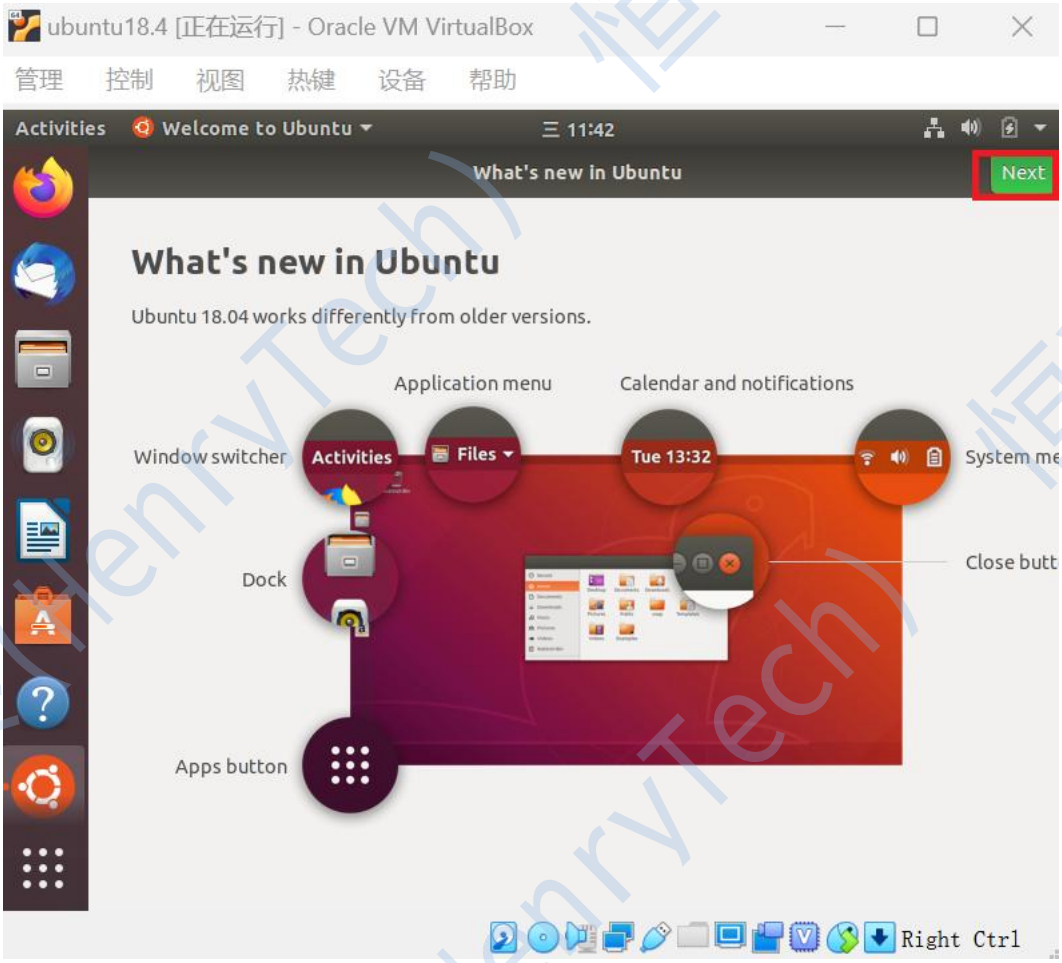
我们回到虚拟机软件点击设置，找到存储，控制器已经自动被移除安装镜像盘片文件了，这个时候我们再次选中右侧选择一个虚拟光盘文件，我们需要安装增强功能，这样虚拟机界面分辨率和后续设置Windows共享文件夹就很方便了：



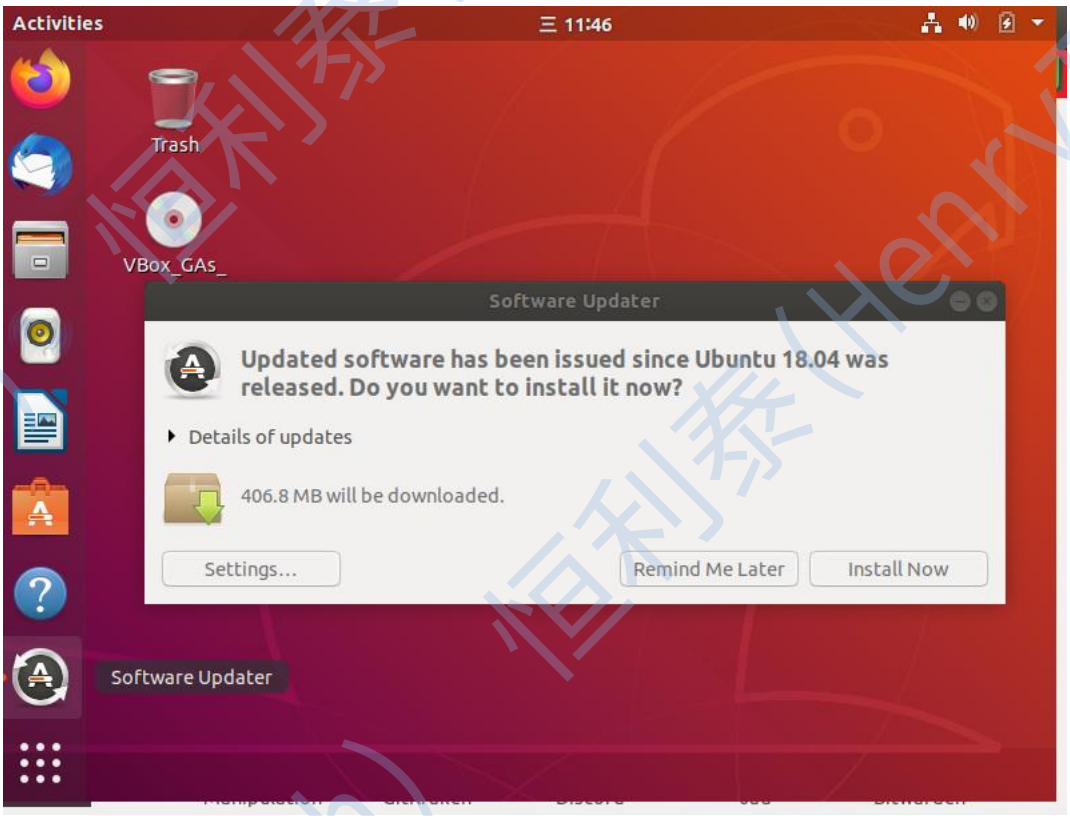
选择我们网盘的如下文件镜像点击打开：



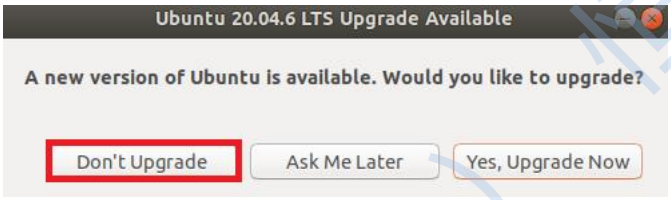
选择打开之后回到设置界面点击OK完成配置，我们回到虚拟机主界面点击启动，开始运行虚拟机。我们等待虚拟机启动后，开始安装虚拟机的ubuntu增强功能。注意，我们安装完成之后，第一次启动后ubuntu会有欢迎界面，大家可以点击右上角的next，直到显示DONE完成。欢迎页面大家可以不做任何配置更改，每一页默认即可。

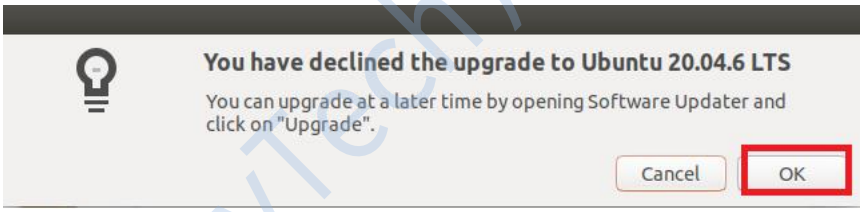


完成欢迎页面配置后我们可能会弹出升级提示，我们可以选择当前安装或者晚点提示都可以，这个不影响，大家自行决定。

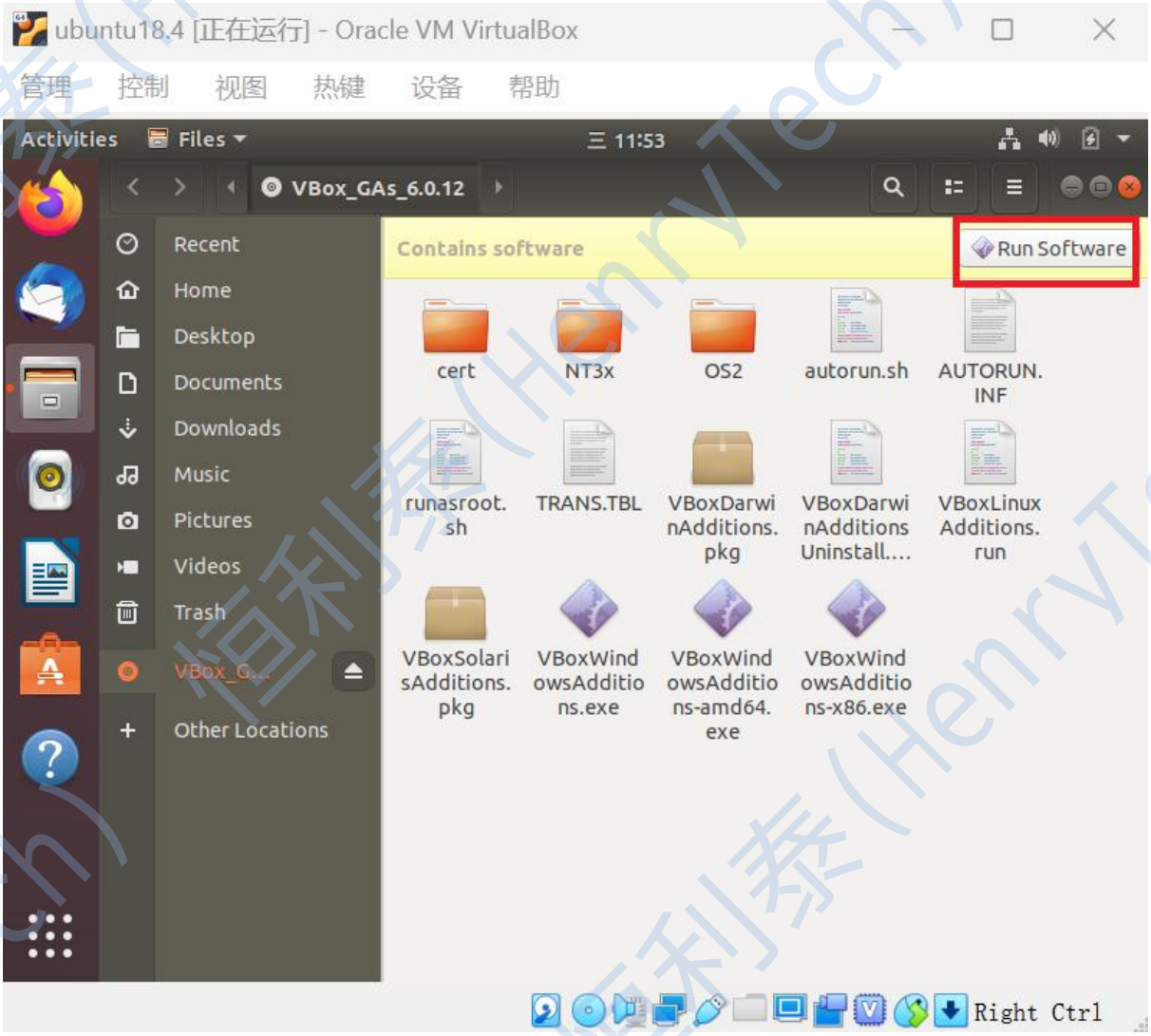


另外可能还会提示升级到20.4版本，我们直接拒绝即可：

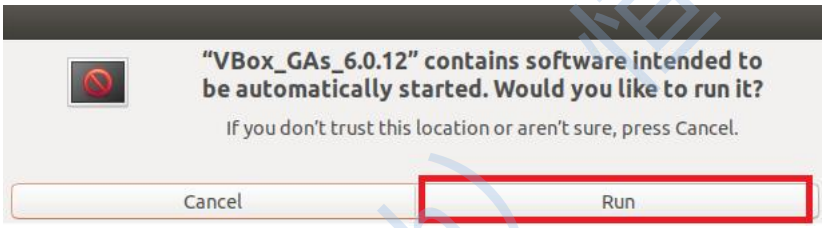




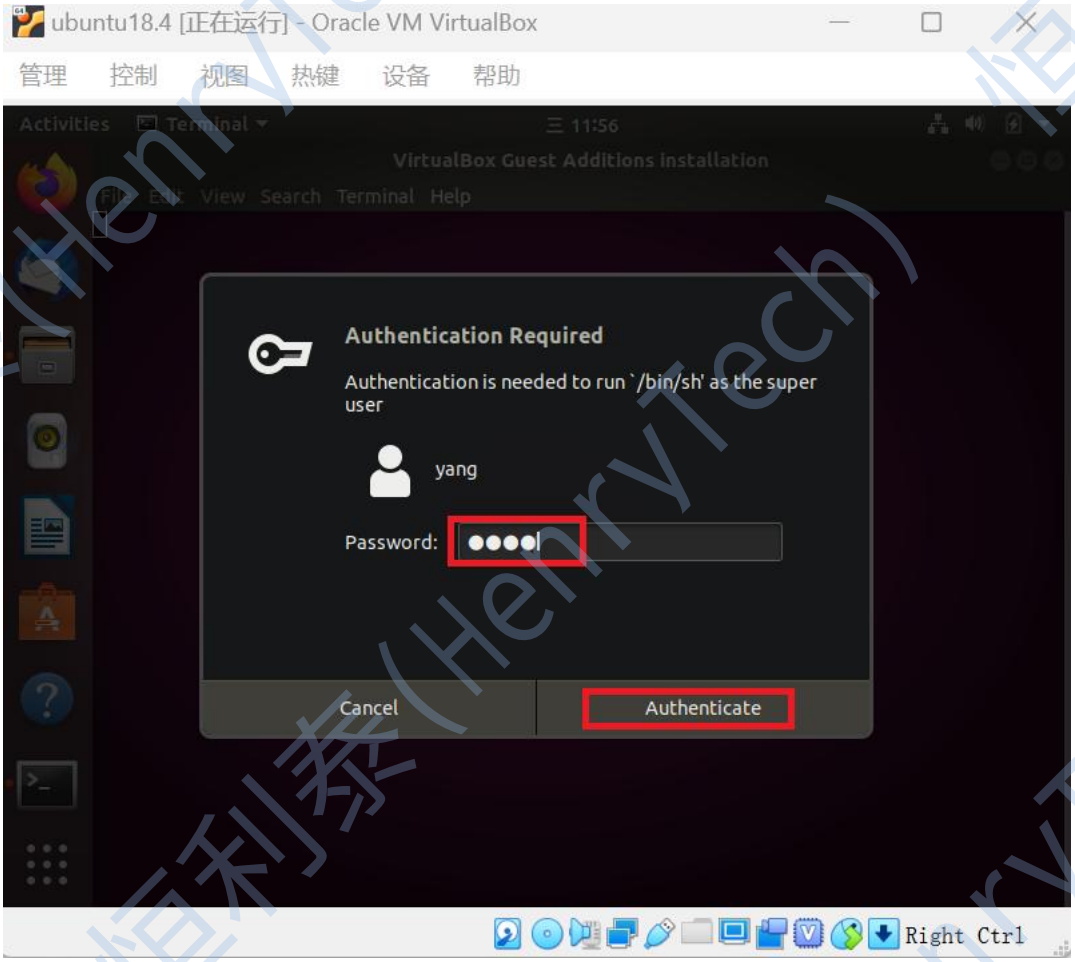
到这里我们安装启动基本完成。我们安装增强功能，我们双击虚拟机桌面的虚拟光驱打开界面，点击右上角的Run Software进行运行安装虚拟光驱的安装程序：



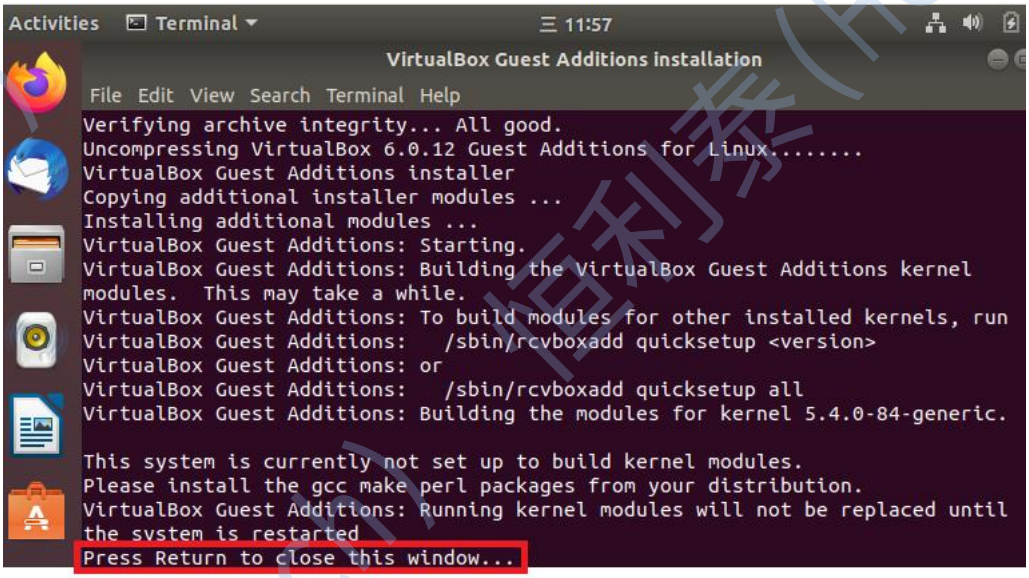
弹出的安装对话框点击RUN进行安装：



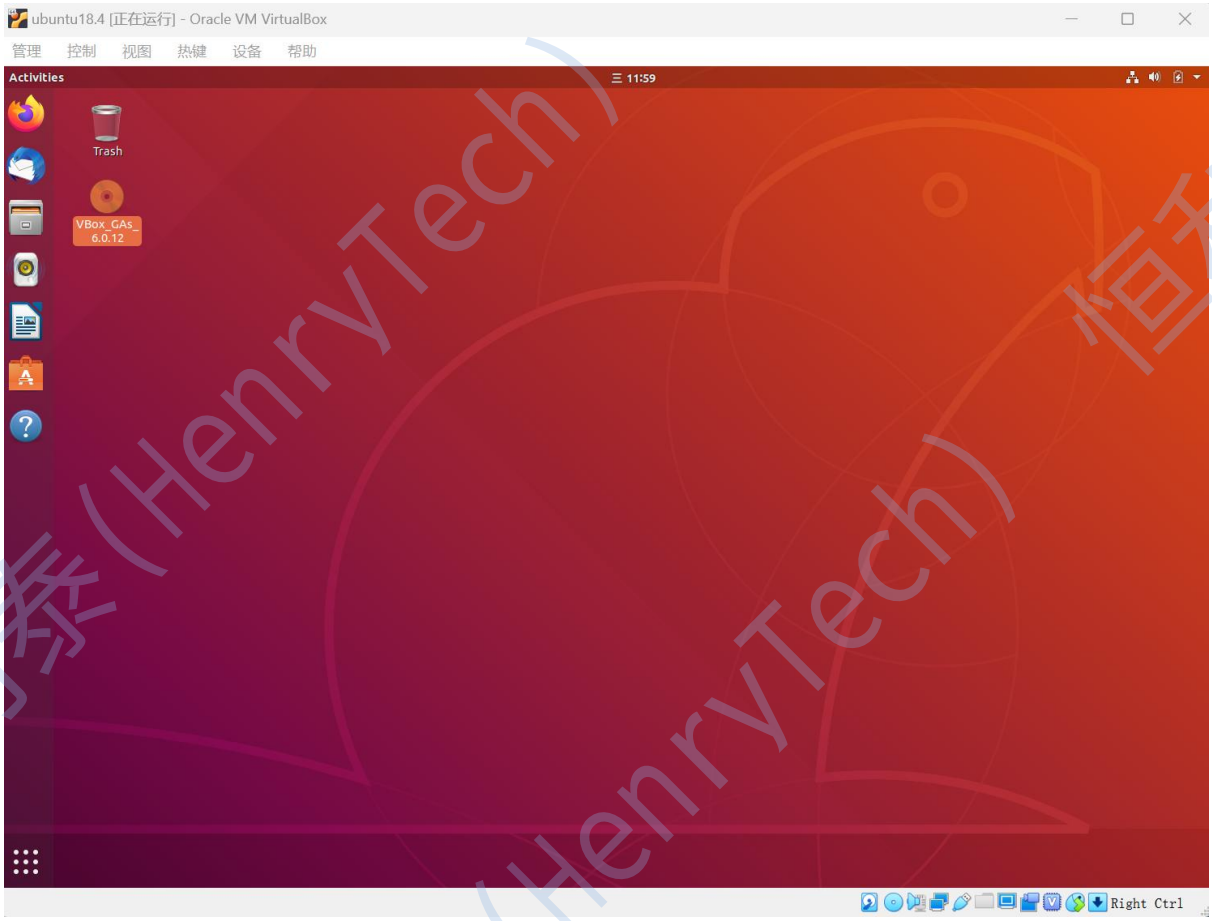
接着还会弹出输入用户名密码，大家输入我们在安装ubuntu的时候设置的密码进行安装授权，点击Authenticate即可：



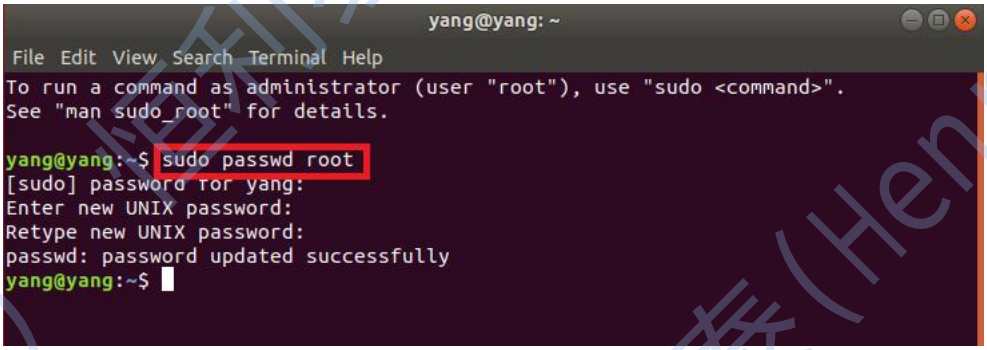
安装完成，我们按下回车即可退出安装。



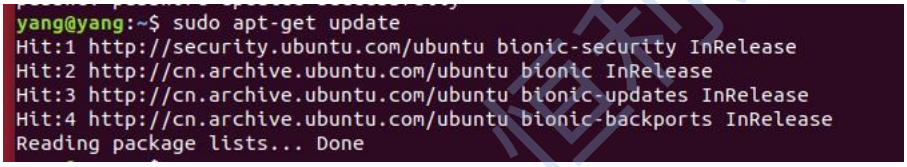
退出安装后我们关闭虚拟光驱窗口。我们尝试将虚拟机ubuntu窗口拉大缩小，可以看到分辨率显示已经可以自动缩放适配窗口了。说明我们增强功能已经成功安装：



我们在窗口空白处右键找到最后一项Open terminal打开终端，输入sudo passwd root设置一个root密码，先输入用户密码，然后输入两次设置的root密码完成设置：



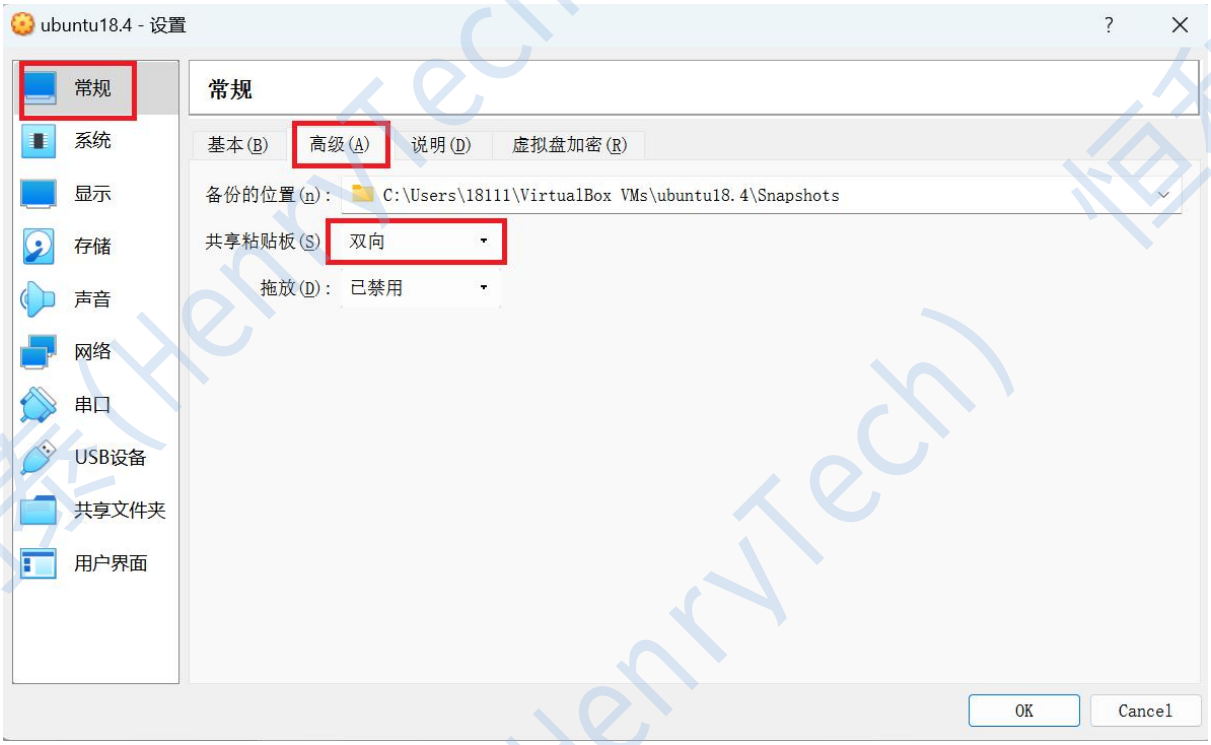
接着输入sudo apt-get update进行更新一下：



到此我们ubuntu虚拟机完成全部的配置和安装。

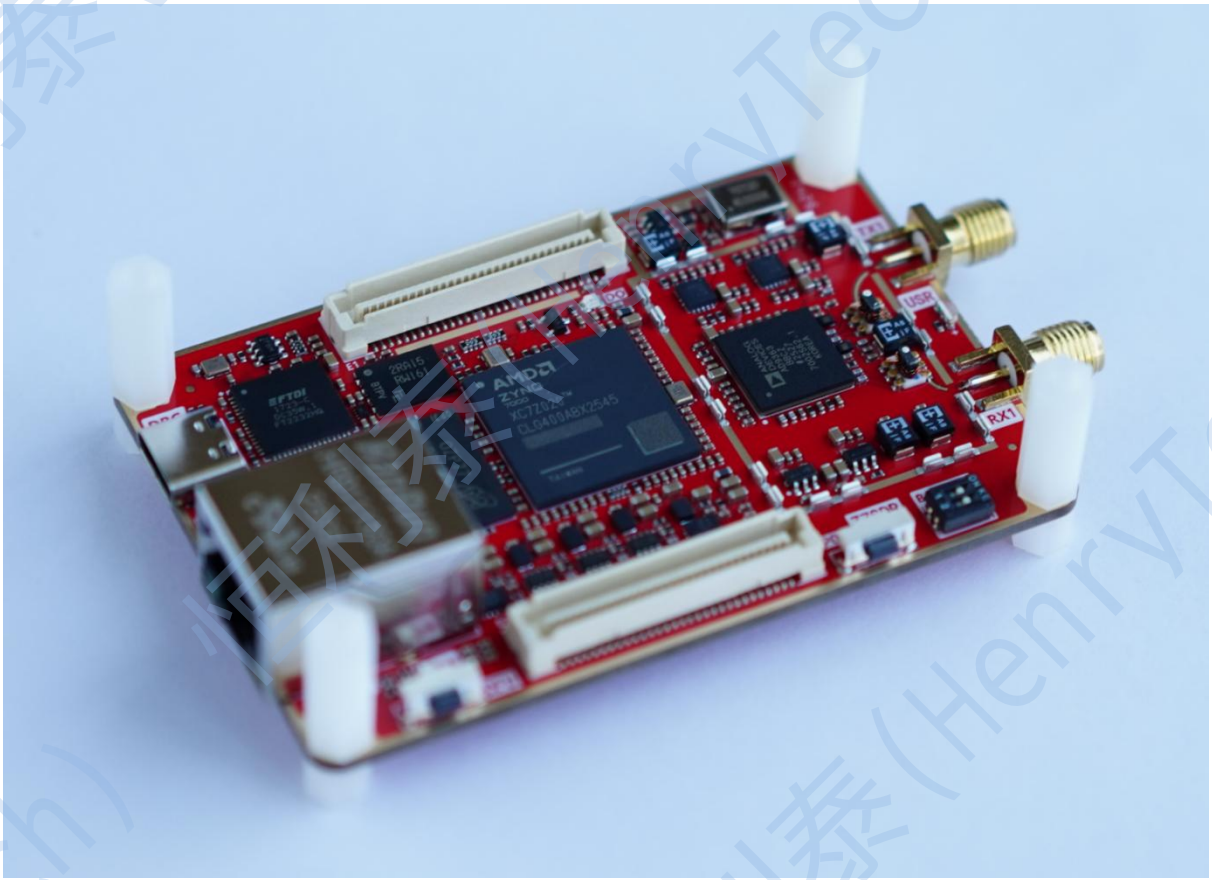
六、其他

对于大家可能会在windows中与虚拟机ubuntu进行复制粘贴文本等功能，我们在虚拟机关闭的情况下，配置如下项，可以支持windows和虚拟机之间进行复制粘贴：



ubuntu安装和使用GNUradio

------ 基于Z7SDRMicro开发平台



目录

十六、文档介绍

本教程给大家介绍如何使用虚拟机环境下安装GNURadio开源软件无线电软件的安装和使用，并且如何连接到我们使用Pluto_Sdr Mini套件。我们本教程将会通过安装虚拟机所需要用到的linux驱动，然后配置USB虚拟网口网络，连接到SDR开发板，GNURadio通过USB虚拟网口连接进行无线收发实验。

十七、GNURadio介绍

GNURadio是一个开源的软件无线电软件。官方地址：https://www.gnuradio.org/
我们可以通过官方网站做更详细的了解。我们这里使用虚拟机给大家进行演示如何安装GNURadio，并且需要安装支持子卡9361的lib iio等驱动和库，才能实际连接到我们硬件驱动我们子卡收发工作。关于虚拟机进行开发，对于学习和验证足够了，受制于运行效率和网络转换性能延迟，虚拟机可能会出现一些数据收发短暂异常等情况，大家如需更加稳定可靠的验证环境，有条件可以考虑安装实体机ubuntu进行。

我们这里GNURadio安装在我们之前一个教程ubuntu18.4安装的环境中进行的。等于是承接上一个教程。请务必按照配套版本进行实验。

GNURadio是一个免费的开源软件开发包工具，里面有大量的信号生成处理模块，可以用来实现SDR软件无线电的验证测试和学习。并且，可以支持安装软件无线电SDR的驱动和开发库等，来支持连接硬件完成实际物理的收发。针对无线电研究学者，业余爱好者，在校大学生以及相关专业的课题研究，均为不错的选择。

针对开发者，GNURadio提供了逻辑模块开发功能，并且支持大量的基本模块，提供了运行模拟SDR运行和信号处理等基本的框架，模块与模块之间的连接，形成了一系列流程处理和数据流处理。很多模块，其底层给予C++或者python编写，所以安装库会依赖python和c++等。并且，GNURadio本身是一个很灵活的系统，可以随时修改重新配置，不针对特定硬件，所以其开发方式对软件定义无线电SDR是一个很不错的诠释。

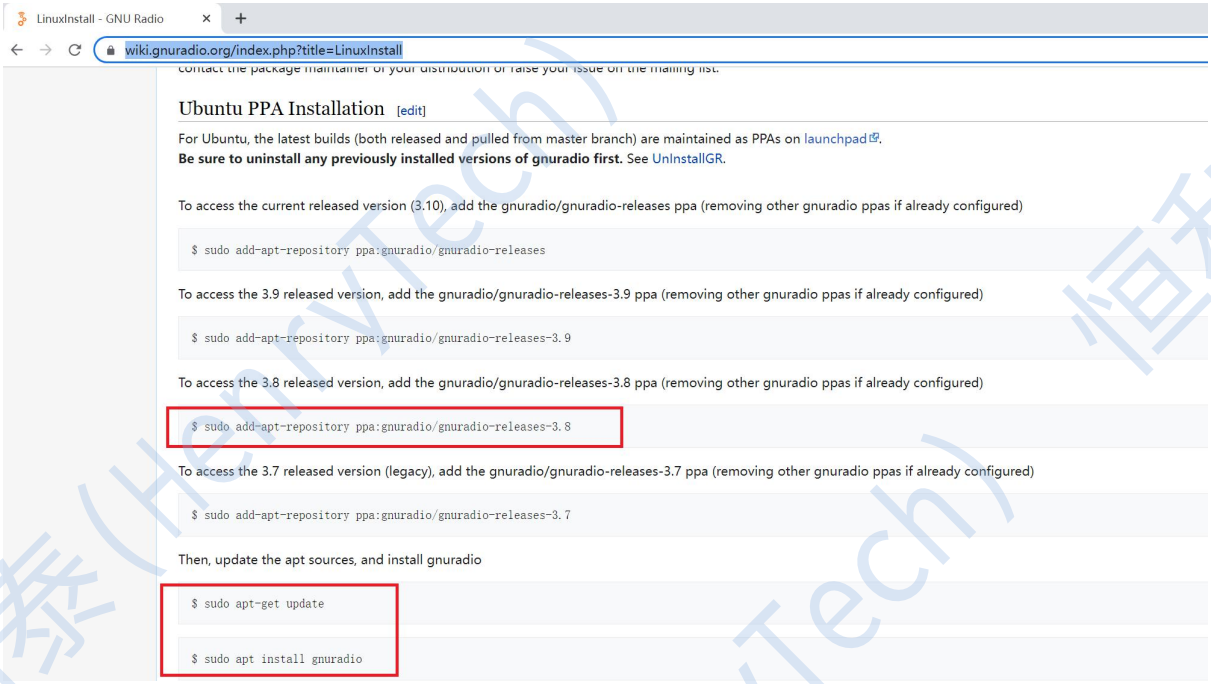
Gnuradio的Git链接如下：https://github.com/gnuradio/gnuradio

十八、GNURadio安装

关于具体的安装在wiki有更详细的说明。针对不同的ubuntu版本，GNURadio也不同。我们使用的ubuntu18.4，实际上wiki目前给的最低版本是ubuntu20.04，这里不影响，笔者使用18.4.6的ubuntu实际测试没有任何问题，安装指南页面如下：

https://wiki.gnuradio.org/index.php?title=LinuxInstall

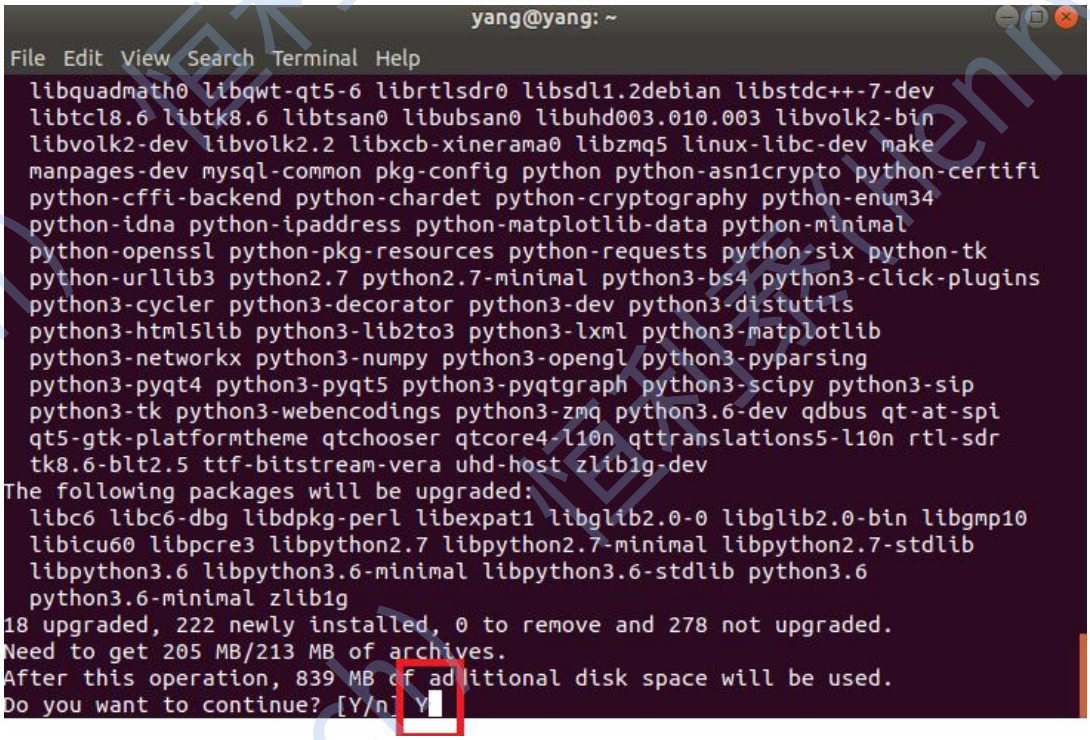
我们在上面链接可以找到gnuradio3.8版本的相关库添加指令，我们依照这几条指令即可完成安装：



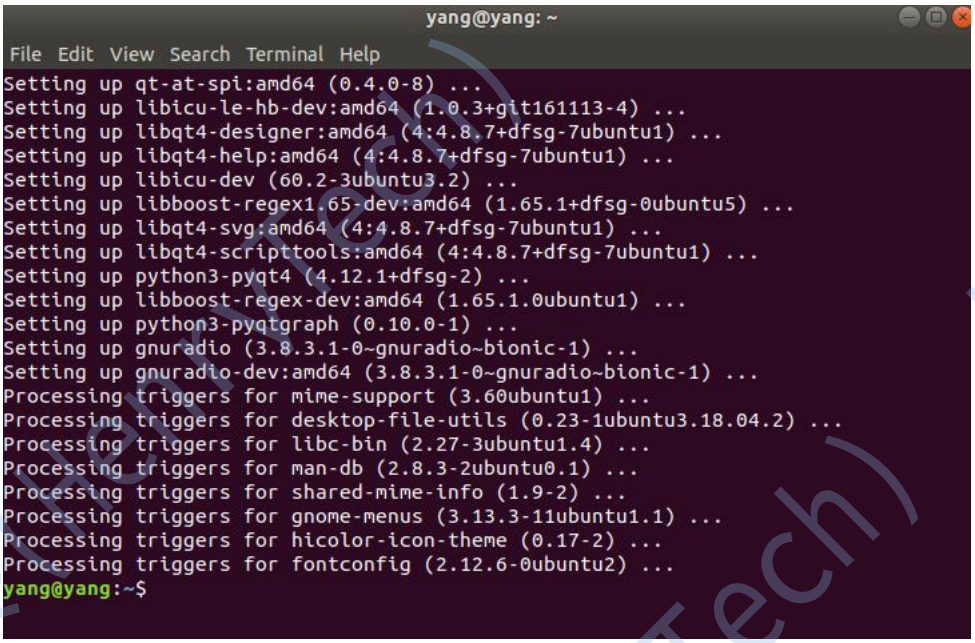
我们保证电脑已经联网。我们打开虚拟机终端，输入以下指令：

```
sudo add-apt-repository ppa:gnuradio/gnuradio-releases-3.8

sudo apt-get update

sudo apt install gnuradio
```

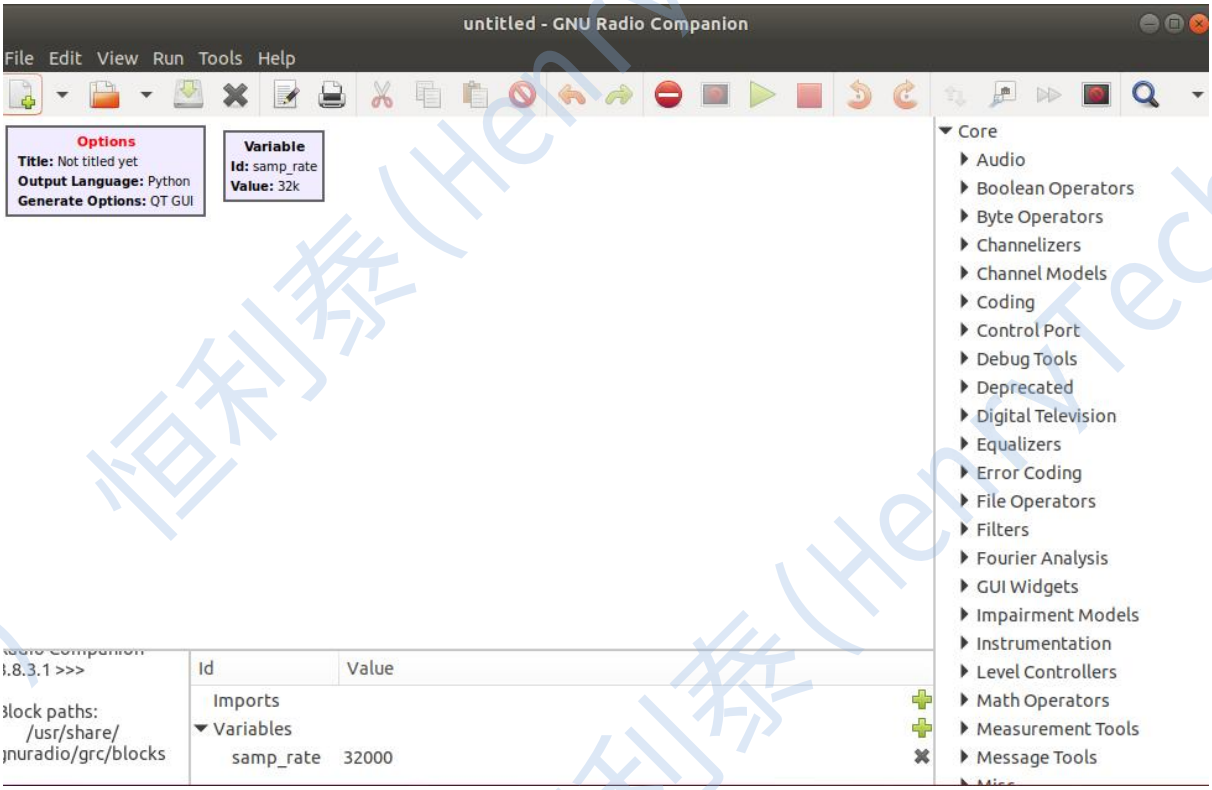
上述指令最后一条就是安装gnuradio。安装会弹出让输入Y或者n选择继续安装还是退出。我们输入Y即可：



我们等待安装结束完成安装视网速性能而定，大概需要5—10分钟时间，有时候网速慢一些需要更久，大家耐心等待，当显示如下之后就说明完成安装了：



安装完成后输入 gnuradio-companion就可以启动软件了。启动后如下界面：



以上就是我们安装的过程。如果大家做仿真验证，不需要连接硬件的话，安装结束即可使用。但是我们需要连接开发板，需要硬件支持，我们还需要安装驱动库。

十九、驱动库安装支持硬件

在安装之前我们需要安装如下一些依赖包：

sudo apt install -y gnuradio-dev libxml2 libxml2-dev bison flex cmake git libaio-dev libboost-all-dev swig libavahi-common-dev libavahi-client-dev g++ libpcre3 libpcre3-dev doxygen libusb-1.0

以上的库可以复制到虚拟机终端安装，注意复制的指令最后敲一下回车就可以。特别需要注意的是，libavahi-common-dev libavahi-client-dev以及libusb-1.0这三个库要保证是安装OK的。

安装库的过程时间也很长，视配置和网络而定。我们这些包大约10分钟。完成安装后我们开始下载编译必需的驱动。首先是libiio驱动。这个是9361底层支持的基础，下方几条指令一条条执行。注意，如果执行失败，或者甚至第一条git指令都克隆不下来，大家检查版本配套，网盘资料《PlutoSDR_Mini软件无线电资料\01工具软件及固件\linux_driver.zip》是我们笔者所使用的配套下载下来的，拷贝到虚拟机使用即可解决，跳过第一条git下载指令。其原因有可能出现git克隆下来的最新版本代码有更新，导致编译make失败：

```
git clone https://github.com/analogdevicesinc/libiio.git
cd libiio
mkdir build & cd build
cmake -DPYTHON_BINDINGS=ON -DCMAKE_INSTALL_PREFIX:PATH=/usr .
make
sudo make install
sudo ldconfig
```

另外，指令敲错可能会导致问题，注意上面路径。最后执行结束我们使用cd .回到用户目录下。注意cd后面是两次.所以返回两级目录，因为我们刚才操作是cd进入了libiio然后进入build文件夹。

接着同样的方法安装libad9361-iio。我们依次输入如下指令一条条执行：

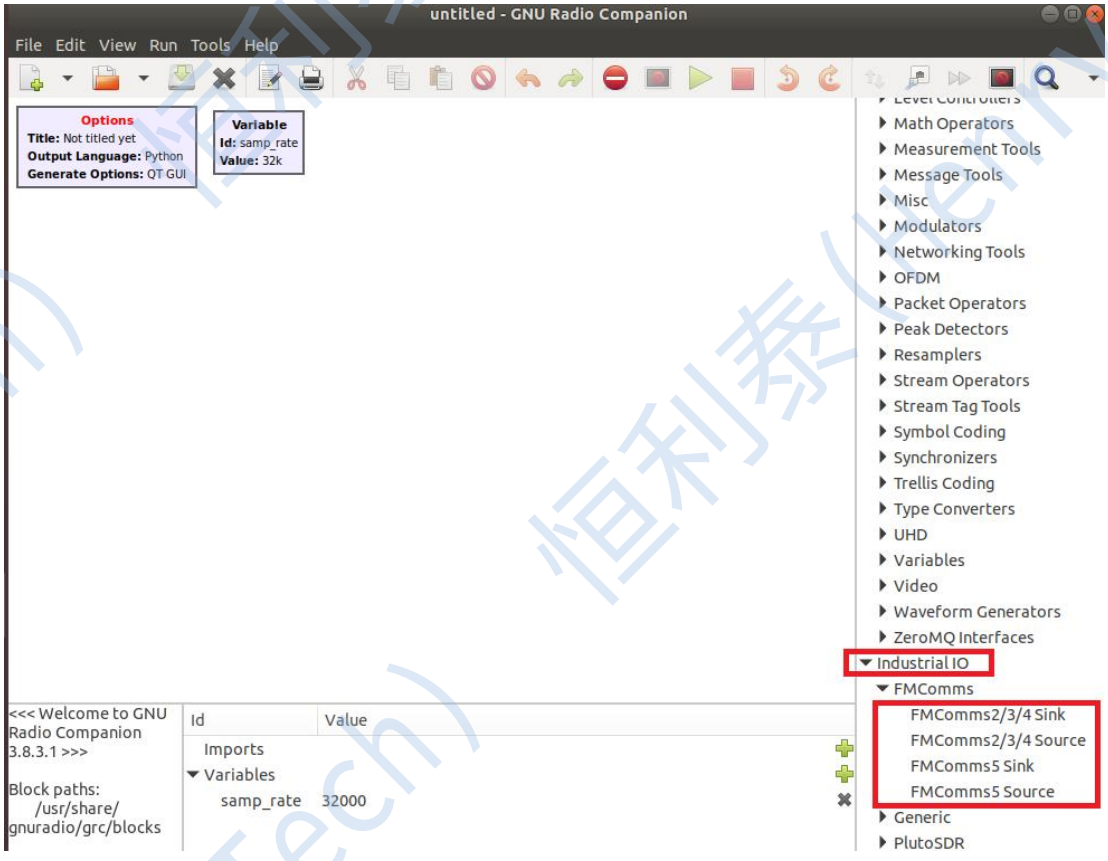
```
git clone https://github.com/analogdevicesinc/libad9361-iio.git
cd libad9361-iio
mkdir build & cd build
cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
make
sudo make install
sudo ldconfig
```

上述代码执行完我们依旧cd . 回到用户目录，准备下载编译安装最后一项，gr-iio模块。我们依次如下指令，注意，克隆完成进入gr-iio目录一定要checkout一下版本，使用3.8，否则可能make编译出错：

```
git clone https://github.com/analogdevicesinc/gr-iio.git
cd gr-iio
git checkout upgrade-3.8
mkdir build & cd build
cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
make
sudo make install
sudo ldconfig
```

上述操作如果自己没有开发经验能自己解决版本问题的话，一定要使用我们网盘给的虚拟机以及ubuntu配套镜像，不要自己去下载其他版本，linux驱动git不下来或者执行出错有任何问题，直接换成网盘里的《PlutoSDR_Mini软件无线电资料\01工具软件及固件\linux_driver.zip》，跳过git指令，直接开始后续指令。

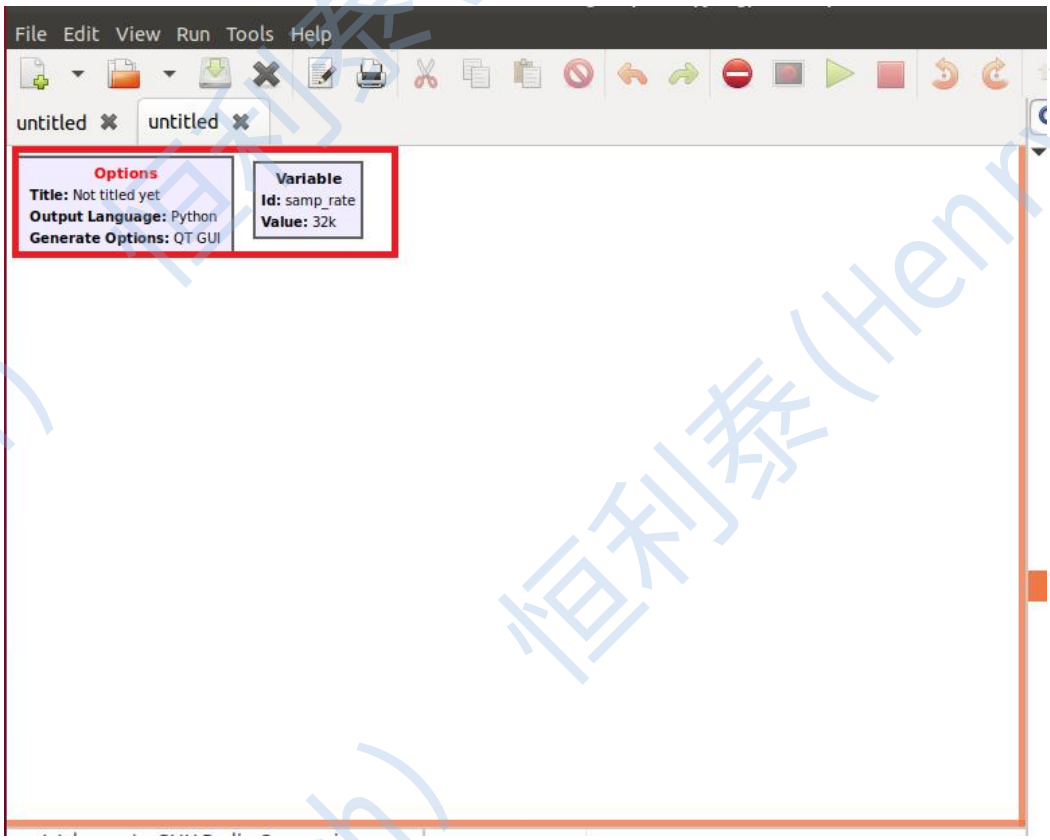
至此我们所有的驱动支持就安装完成了。我们终端输入gnuradio-companion再次启动GNURadio，右侧所有模块控件下拉到最后可以看到Industrial IO，可以找到FMCOMMS系列官方AD9361子卡支持，以及PlutoSDR支持：



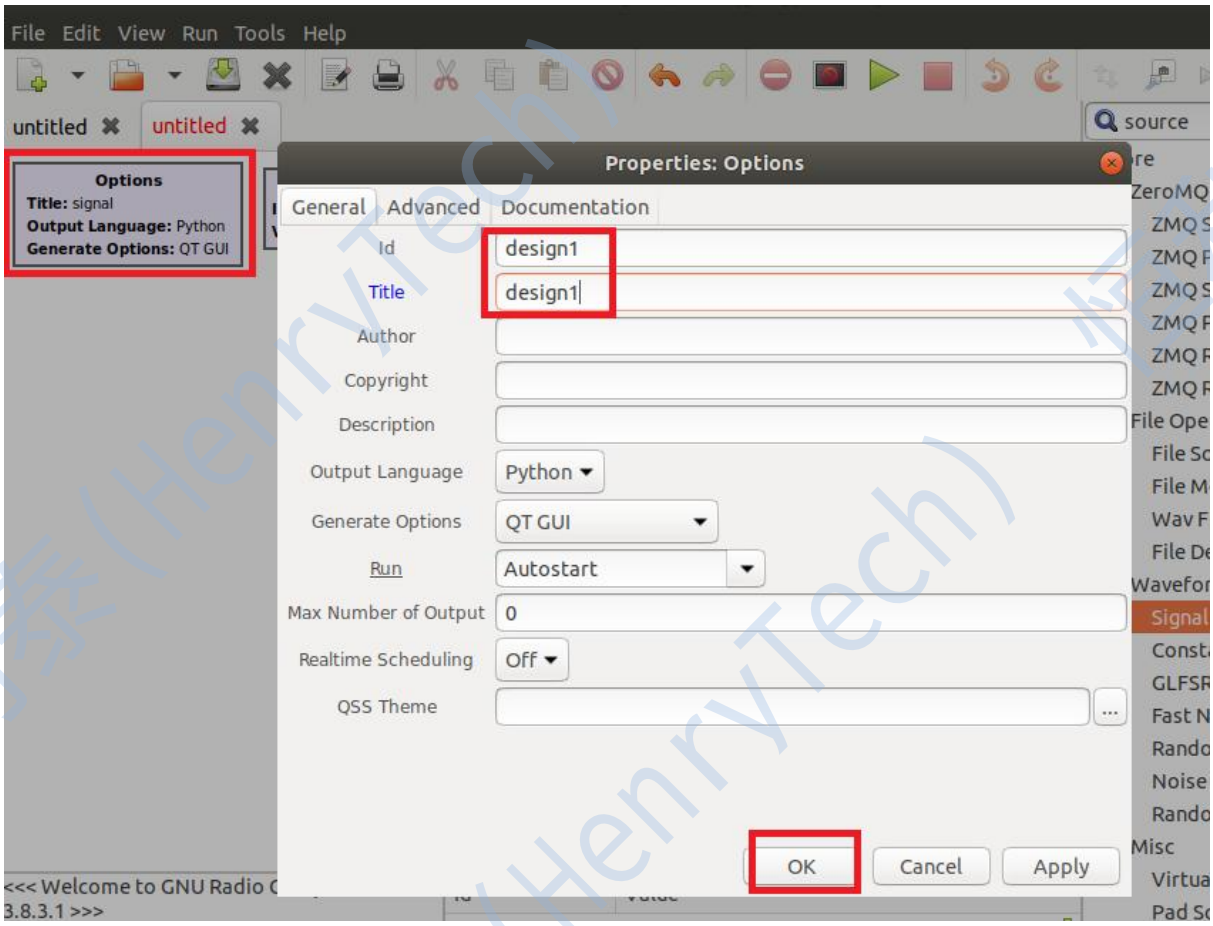
到此我们开发环境就搭建完成了。关于GNURadio界面，左边的最大区域是我们的工作区，大家可以在这里搭建自己的算法模块；右边显示各种列表模块的是模块控件区域，并且支持第三方的模块集成进来。前面我们安装编译的模块就是ADI官方提供的SDR硬件及驱动支持模块。左下角是我们的输出打印信息显示区域，右下角Value窗口是我们的变量显示区域。关于GNURadio我们暂且介绍这么多，我们下面直接进入主题，使用GNURadio搭建几个最简单的例子供大家入门参考。

二十、最简单的单音信号测试

作为SDR开发平台或者算法验证，GNURadio可以脱离硬件进行验证。我们使用GNURadio生成一个最简单的单音信号例子，来给大家演示如何调用几个简单的模块快速搭建信号输入输出。我们在GNURadio启动，就会自动生成一个新的设计文件，我们如果需要重新新建或者另存为，可以在下拉File菜单栏选择New->QT GUI新建，或者File->Save as选择另存为当前的设计。在下一次启动软件，自动打开上一次关闭退出软件时候的设计。我们终端输入gnuradio-companion启动GNURadio后，默认打开一个空白的设计，注意设计页面左上角自动添加两个标签和选项，Options是本设计的属性，包括标题，本设计输出的语言，以及生成选项为QT GUI，采用QTUI作为界面。右边的Varible是一个变量标签，这个标签是表示一个叫作samp_rate，并且值等于32k的参数变量。设置这个标签就表示，在本设计中任何地方想要使用32k参数，可以直接填入samp_rate这个名称就代表32k：

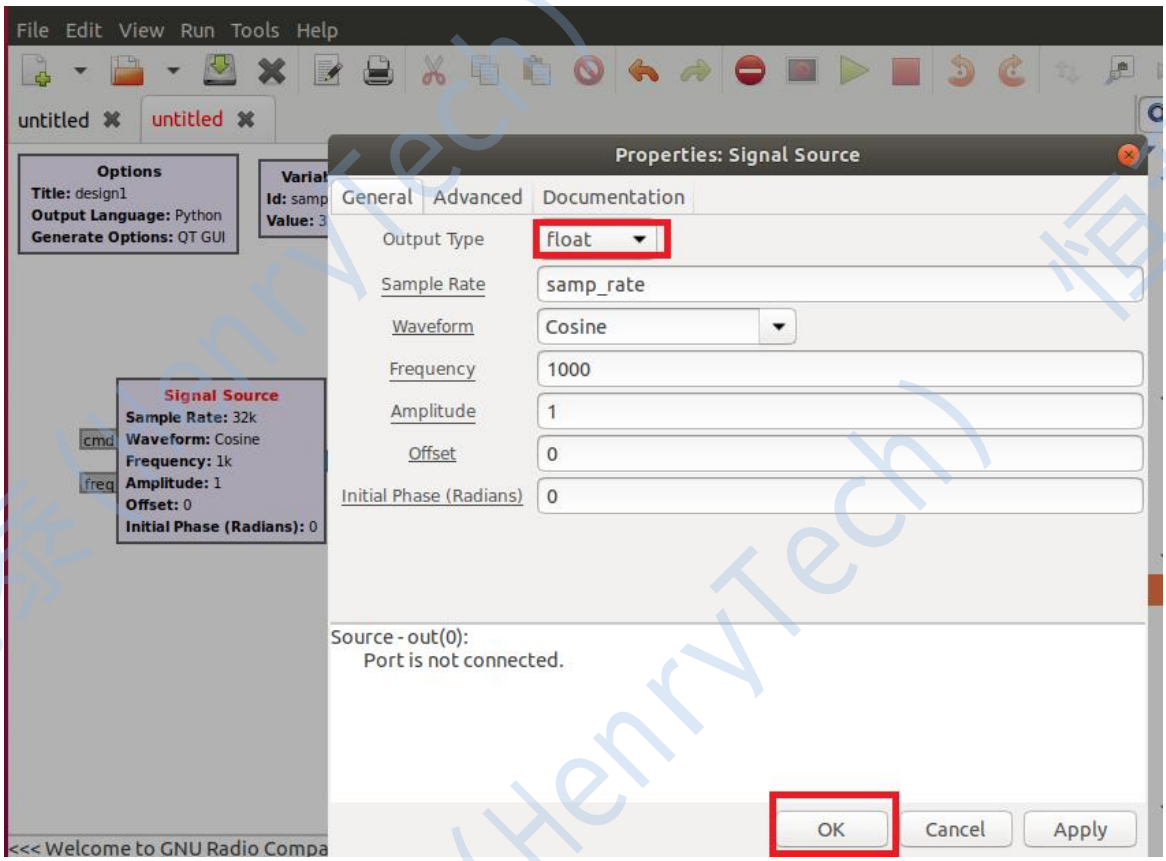


上面默认两个标签，Options我们双击进去做个简单设置，id和title我们随便填入一个名称即可，然后其他的配置默认，不用修改，点击OK完成配置：



同理大家可以修改Varible名字和参数。我们这里不做修改，默认即可。接下来我们调用一个信号模块。我们在右侧输入框输入source，找到Waveform Generators下面的signal Source双击添加：

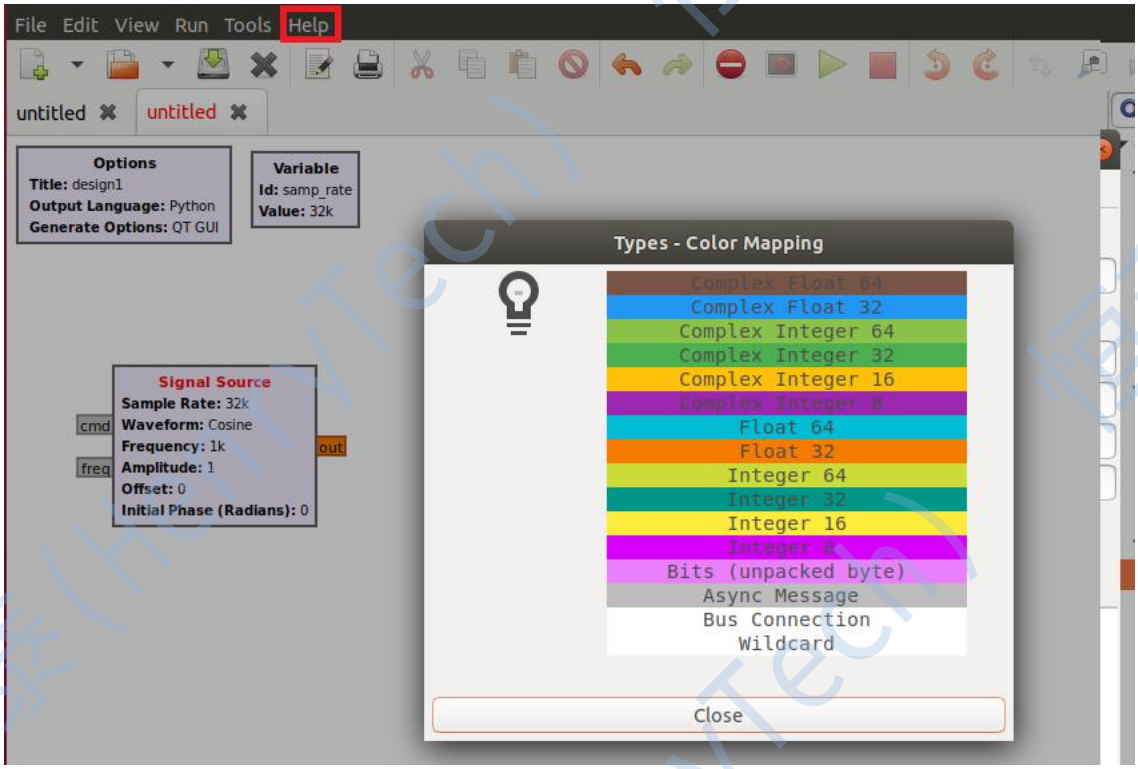
添加进来之后我们看到已经在设计中出现了这个模块。我们双击这个模块打开进行配置。我们仅需修改output type为float浮点数，其他的保持默认：



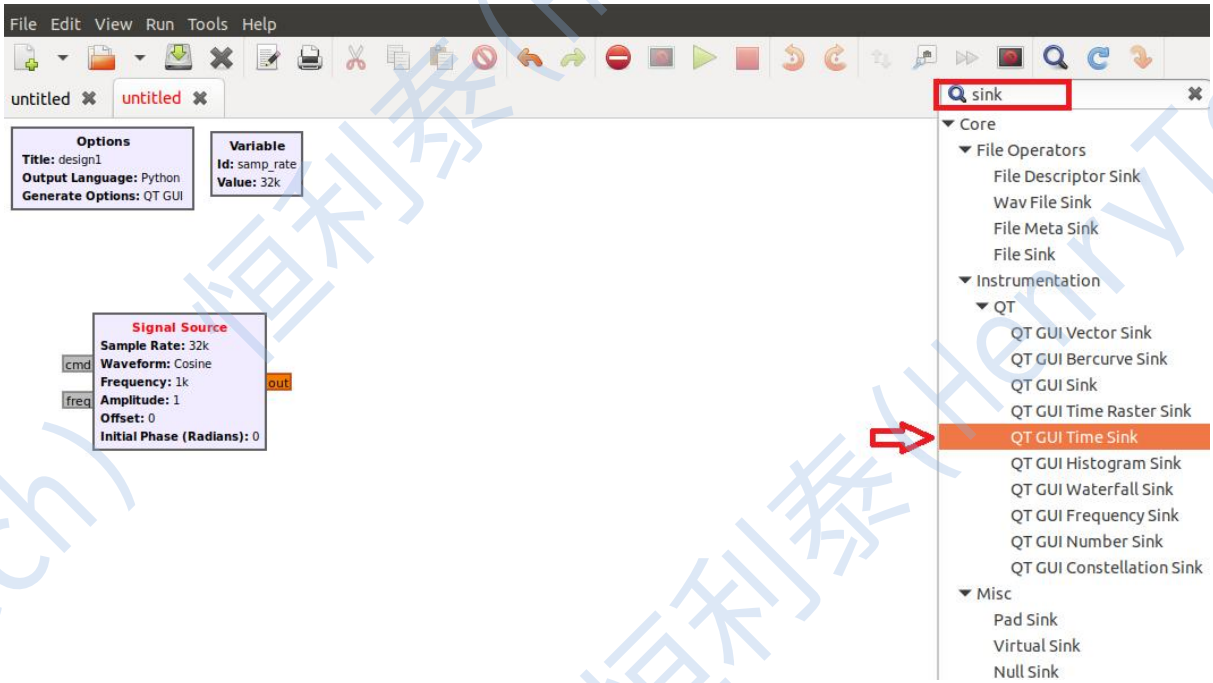
上面的参数解释：

- Output type：输出的数据类型。可选complex复数，float浮点数，int整数，short短整数，byte字节；
- Sample_rate：采样率，我们可以直接填入数字，默认是设计中的Varible自带的samp_rate变量，表示32K。注意GNURadio中，很多参数均提前定义好标签，后续就直接填入标签的id名字就代表参数；
- Waveform：表示数据波形类型。我们默认Cosine余弦。可选Constant, sine, Square, Triangle, Saw tooth;
- Frequency：频率。我们这里配置1000表示1K；
- Amplitude：振幅，表示信号幅度。我们默认为1；
- Offset：偏移量：默认0；
- Initial phase（radians）：初始相位，以弧度为单位。我们默认为0。

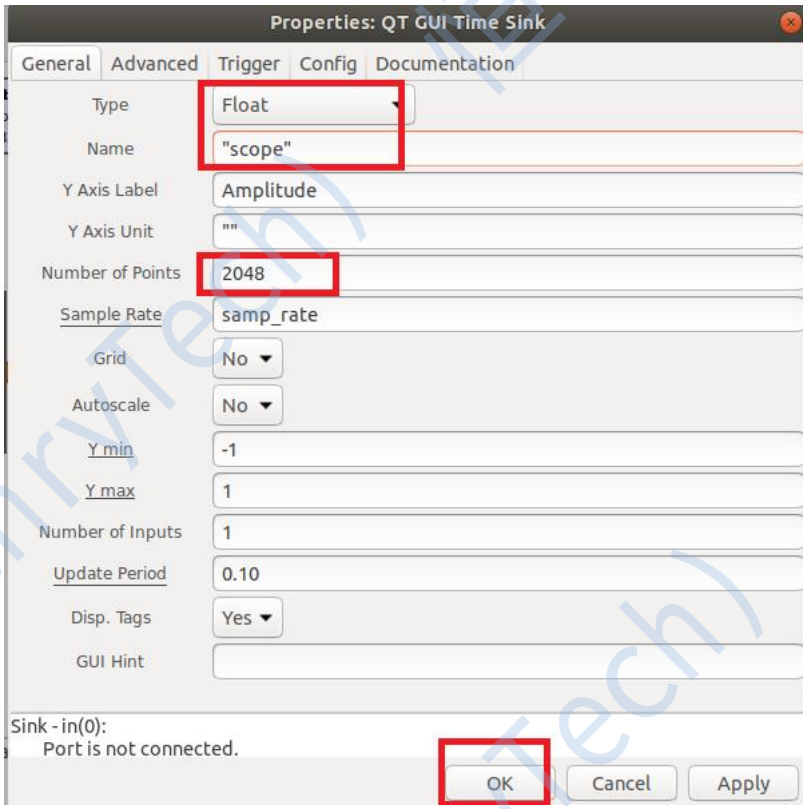
关于信号输入输出接口，我们不同类型会有不同的颜色显示，上图配置float输出后会发现我们out端口是橙色。信号颜色和类型对应关系在菜单栏下拉help->types即可查看：



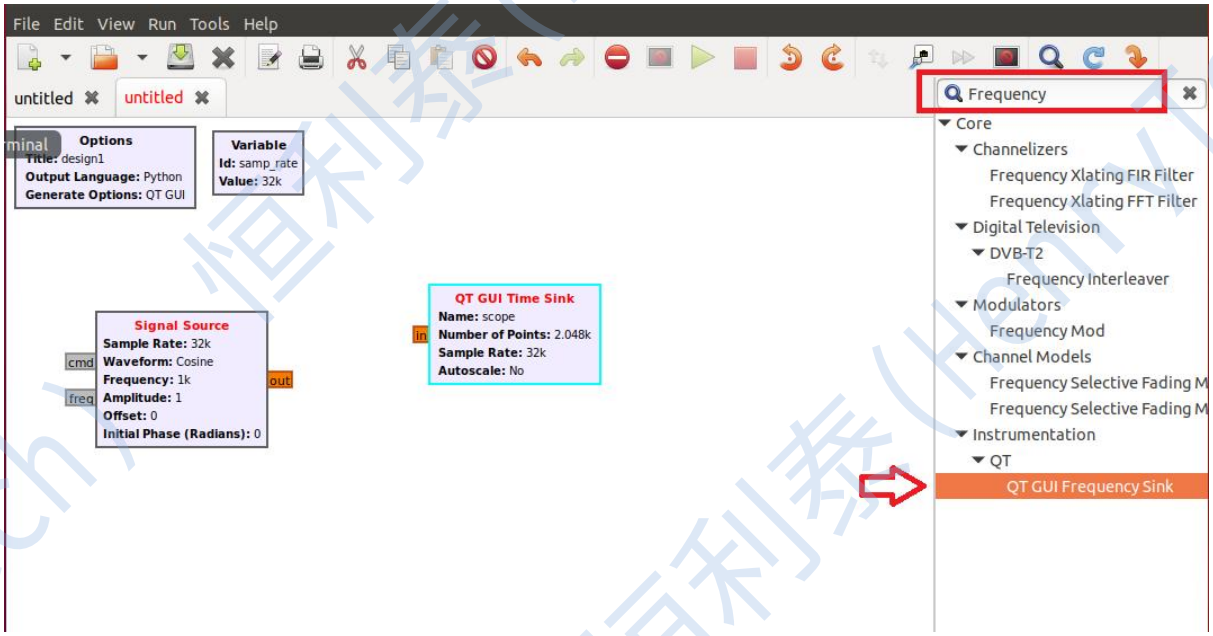
我们配置完成点击OK退出配置界面，接下来我们再添加一个观测时域的模块，简单来说就是示波器功能，可以查看波形。我们搜索sink找到QT GUI Time Sink双击添加：



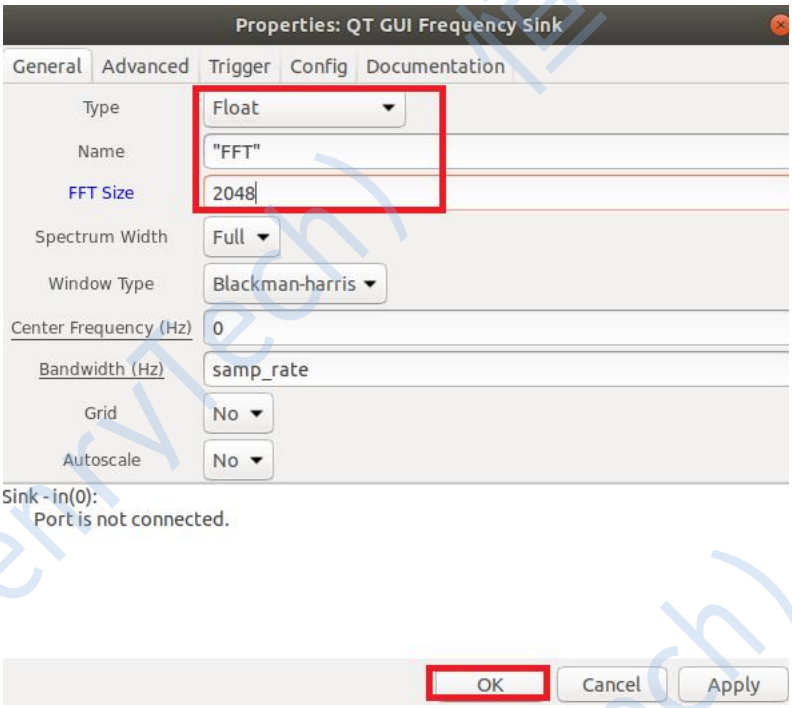
双击添加进来之后我们设计中出现了这个模块，我们双击这个模块打开配置页面进行配置。注意，这一页配置项可能因为窗口原因没有显示完，大家可以把窗口往下拉长一点就可以看到完整的配置项了。我们仅仅做如下配置修改，type选择Float，名字“scope”，名字可以随便取，Number of points点数2048（这里大家可以改其他参数，值越大表示点越多，图越精细），其余的均保持默认，最后点击OK保存配置退出：



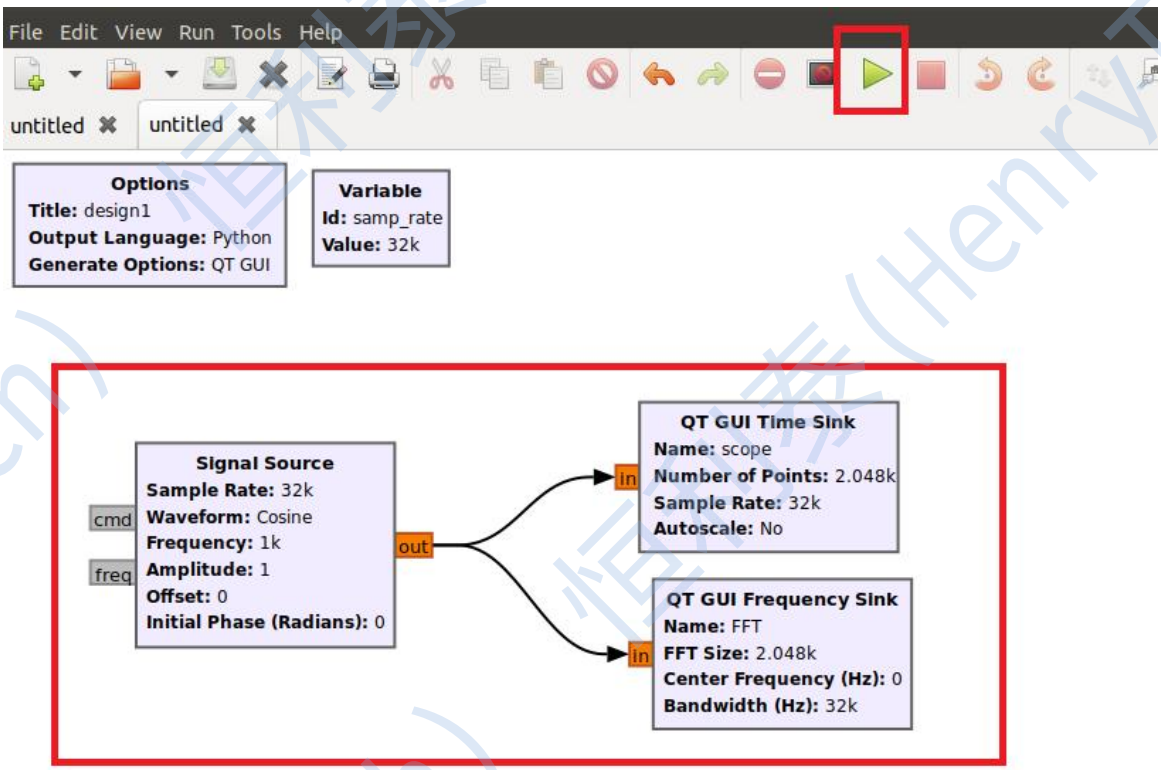
配置了示波器模块后我们再调用一个频域显示的频谱模块用来分析频谱，类似于频谱分析仪的功能。我们找到instrumetation下面的QT分类中的QT GUI Frequency Sink模块双击添加调用：



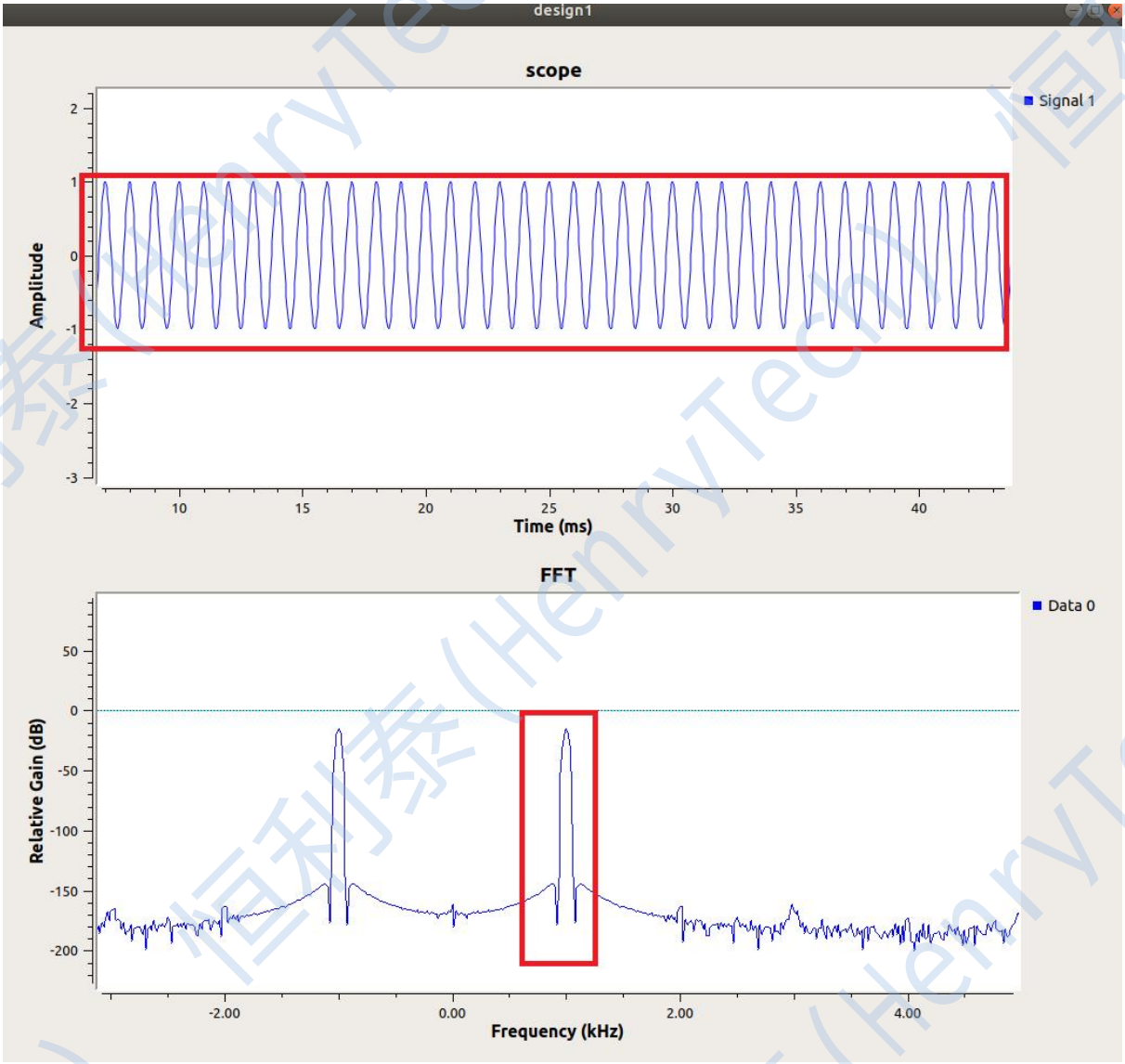
双击添加进来之后我们双击打开配置。首先是type改为float，接着名字大家随意设置，这里我们叫作FFT，FFT尺寸我们设置为2048个采样点（这里也可以改大），这样可以使得FFT运算结果显示精细一些，最终设置如下，设置完成点击OK退出：



接下来我们开始连线。我们鼠标单击Signal Source的out信号，再单机QT GUI Time Sink的in端口就完成这两端连线，同样再次单机Signal Source的Out，再单机QT GUI Frequency Sink的in，完成连线。我们设计的目标就是将Signal Source的输出信号连接到示波器输入进行显示同时要给FFT做输入，最终波形和频谱同时可以查看。连接好如下，我们最后点击上方的绿色小三角可以进行运行。注意，设计配置没有问题的情况下，运行的按钮是绿色；如果连线有问题或者设计有问题，那么这个按钮颜色很淡，无法运行，如果无法点击运行大家需要自己检查一下配置：



我们点击  之后，可能第一次运行会弹出提示警告一类的对话框。我们将对话框关掉即可，然后就可以看到运行的信号显示界面了，我们在信号波形和FFT波形上可以拉动鼠标滚轮缩放波形显示，我们适当缩放拉伸一下波形，可以看到如下显示，上面是波形图，下面是对波形进行FFT的傅里叶变换频谱图。频谱可以看到，我们频率在1K左右的尖峰，在坐标轴小于0的-1K地方也有一个对称的尖峰，大家看频谱只需要看右侧的即可，因为正负的数据对称：



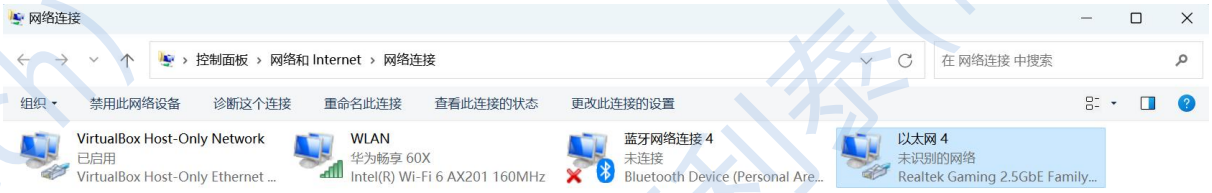
到此我们就完成了一个最简单的单音信号生成显示实验。我们如何与SDR的硬件联系起来，到PlutoSDR Mini发板进行实验？接下来一节我们教大家如何连接我们PlutoSDR Mini到电脑，通过USB接口，将我们GNURadio实验信号驱动输出。

二十一、虚拟机网口配置连接开发板

我们这一节讲解如何上板测试单音信号。我们连接DBG(UART+JTAG）口到电脑，千兆网线直连电脑。我们在实验之前先做一些准备工作。我们按照《01快速测试及使用指南。pdf》教程，按第六节<SDR#快速测试>在串口终端中配置好SDR的IP地址为192.168.1.10，电脑与SDR

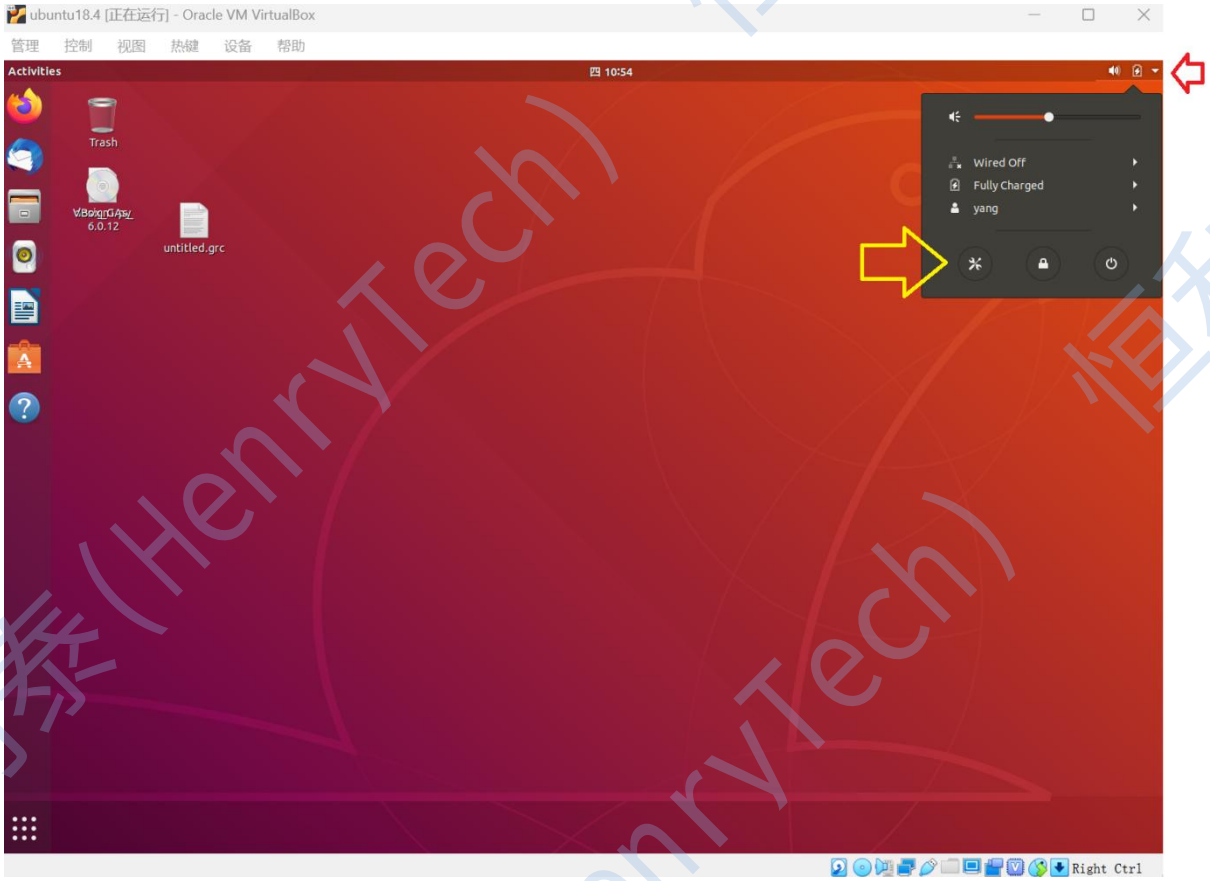
连接的网口IP地址192.168.1.3。在虚拟机中进行实验之前我们先关掉虚拟机，按照《01快速测试及使用指南。pdf》第六节直连网线的方式先跑通进行SDR#的FM收听实验确保网口可以正常通信，并且在我们windows下可以使用cmd命令提示符完成与开发板的ping通，保证开发板IP网络配置和网口连接成功。然后我们关掉SDR#软件以免影响占用，配置虚拟机网口方式。我们打开虚拟机的设置网络如下，选择桥接网卡，网卡配置选择电脑的与SDR连接的网口点击OK完成虚拟机网口配置：

注意，我们网口怎么看是哪个？我们在《01快速测试及使用指南。pdf》中直连电脑配置过IP的那个接口，在网络连接中，我们与开发板连接网线后显示未识别的网络就是我们电脑的有线网口，上面可以看到一个2.5GE的型号网口。注意，笔者使用的电脑网口最大支持2.5G，对于大家实际使用的电脑，市面上大部分均为1.0G速率的千兆网。大家实际网口应该是1.0G的网口。因为笔者电脑配置版本较新，所以是2.5G。

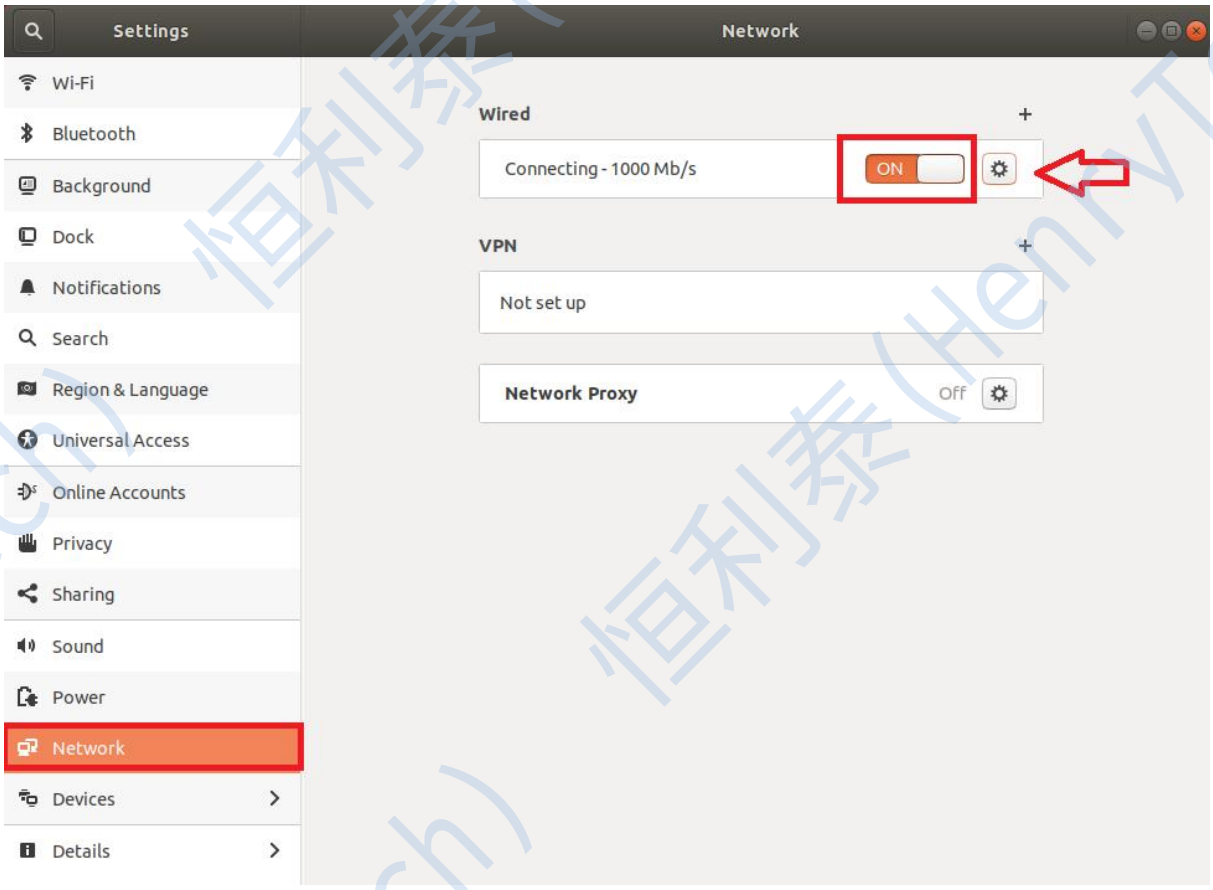


我们虚拟机配置好网络之后点击OK完成配置之后我们退出，接下来启动虚拟机，我们再对虚拟机进行一些网络配置。我们在虚拟机ubuntu中，右上角小箭头展开找到扳手图标的设

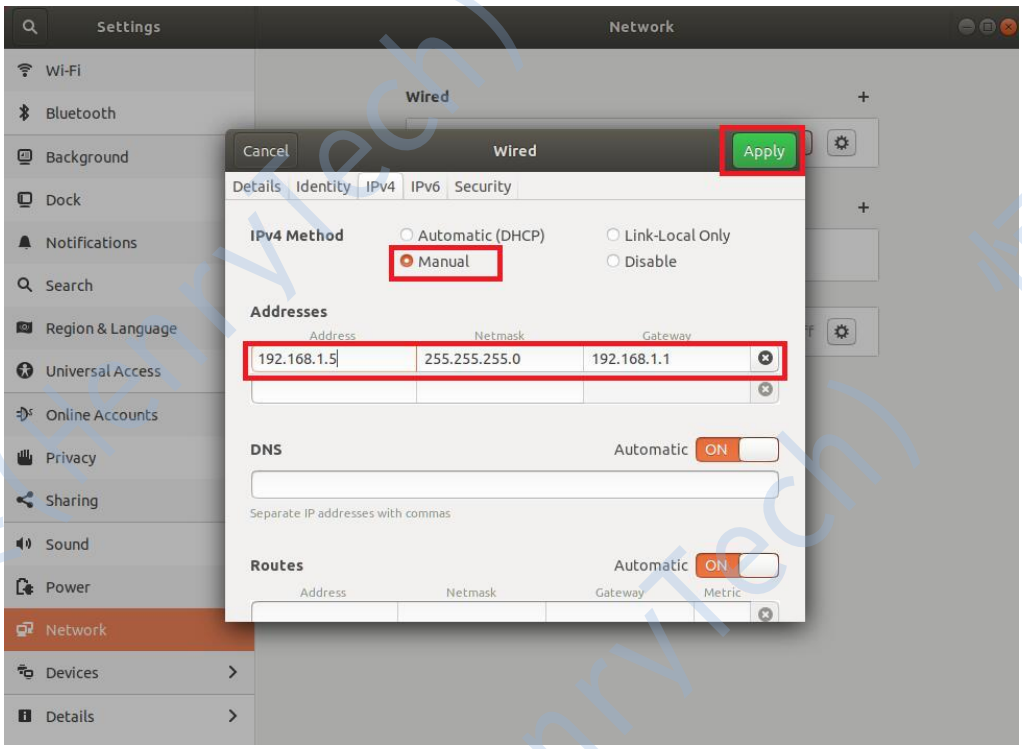
置点击进去：



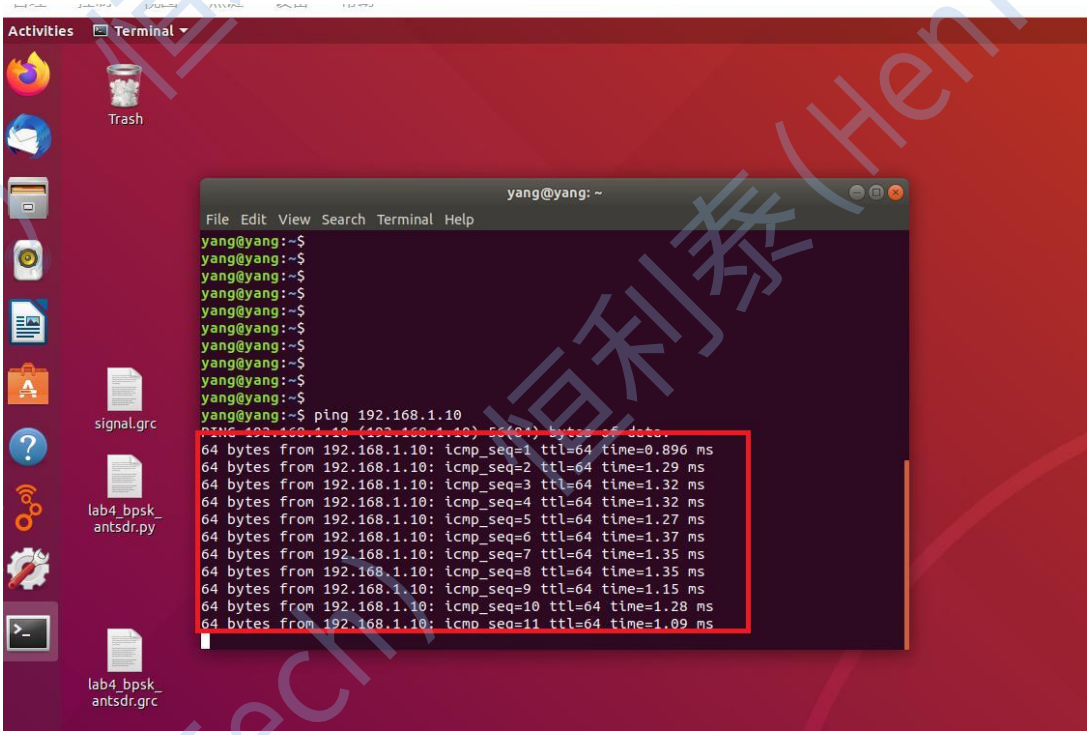
找到Network，右边连接1000m的开关打开，然后点击最右侧齿轮图标进一步设置：



我们选择手动设置manual选项，最后IP，NET MASK,GATEWAY等设置如下，设置好了之后大家再次确认一遍，点击右上角Apply应用即可完成设置，然后退出：

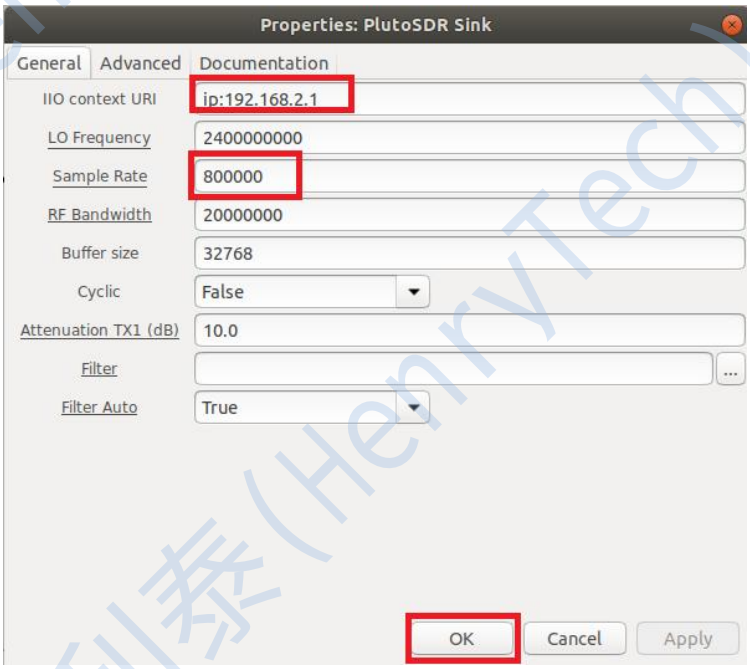


到此我们打开终端，输入ping 192.168.1.10测试一下与开发板的连接。如果没有通过，建议可以按下开发板电源，重启一下开发板，检查一下虚拟机网络设置，然后检查一下开发板网口配置文件是否生效。最简单的办法就是，将《01快速测试及使用指南。pdf》教程的网口直连电脑配置好，windows下与SDR#连接完成FM广播收听，或者windows下完成ping通，然后再检查虚拟机网口配置，再次强调，要把windows下用到这个网口的软件一定要关掉，比如测试过SDR#之后关掉软件。这里我们虚拟机ping开发板的IP已经通了：

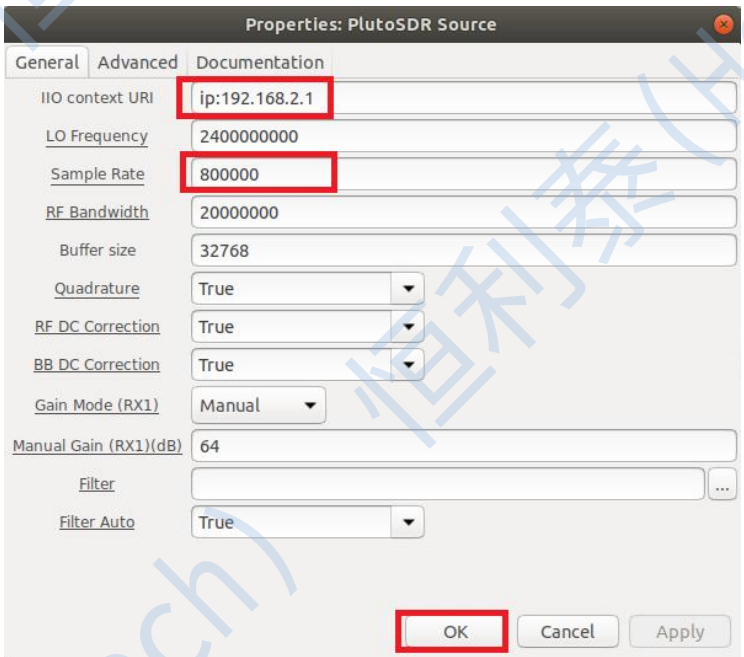


二十二、GNURadio调用开发板

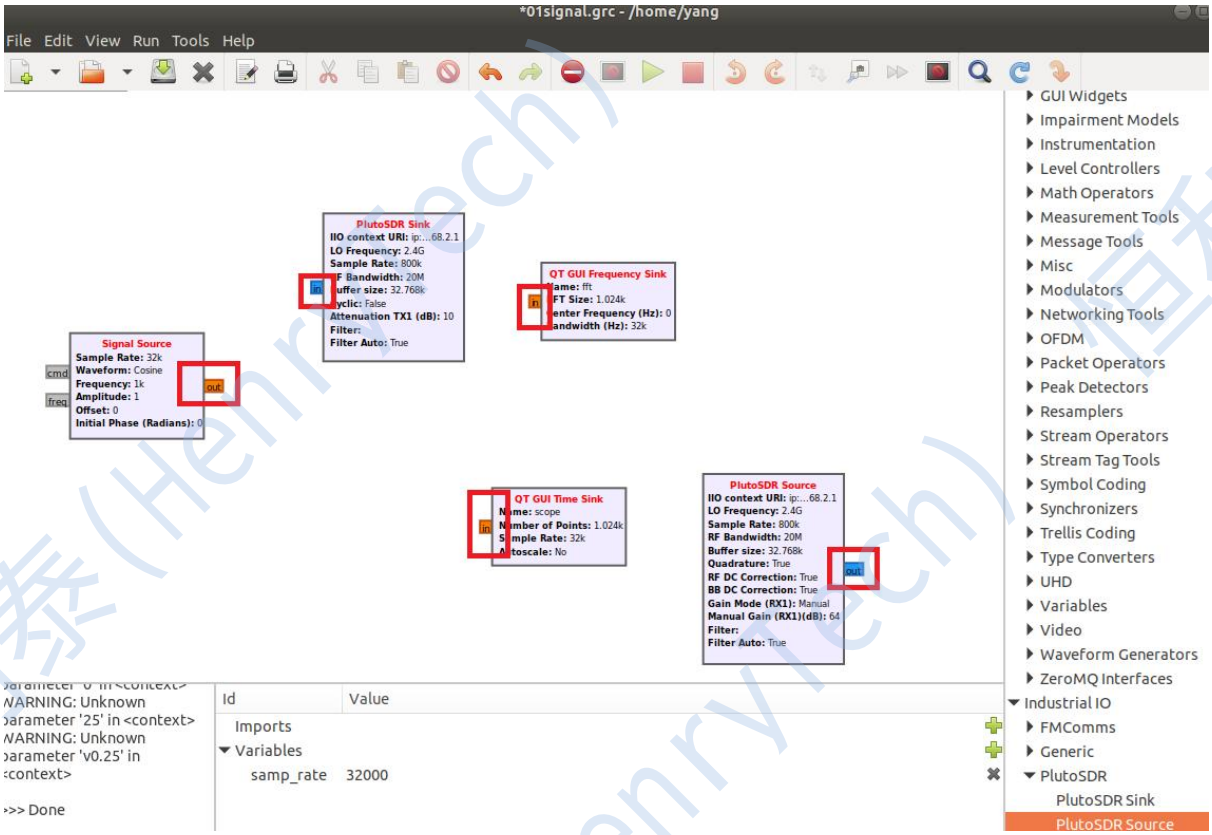
我们上面一节完成了开发板与虚拟机的网络连通测试。接下来我们就可以调用相关的硬件驱动模块将信号给我们的SDR硬件。我们输入gnuradio-companion启动GNURadio，接第五节的单音信号，我们在这个基础上增加硬件支持模块进行实验。我们从右边界面选件中，在Industrial IO找到，PlutoSDR Sink模块添加进来（双击添加），这个模块负责输入1路数据给9361的DAC将无线数据发送出去，配置如下，就需要把第一项IIO context URI改为ip:192.168.1.10对应我们开发板的IP地址，采样率800K然后点击OK完成配置退出（注意截图中使用的是其他产品的配置，这里按ip:192.168.1.10地址，没有重新截图）：



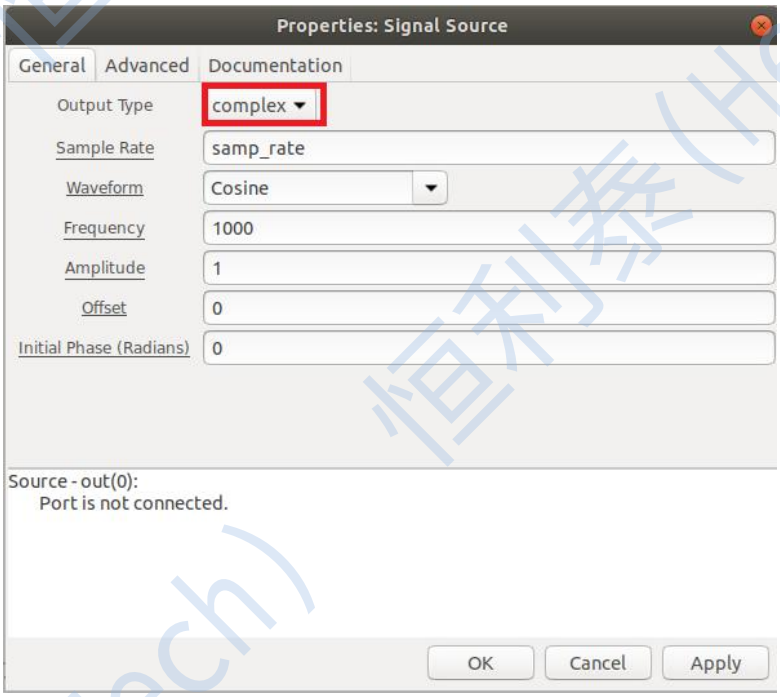
接着我们添加PlutoSDR Source模块。这个模块负责获取接收从9361上接收采集的ADC数据。我们配置也仅仅配置ip和采样率，最后点击OK完成配置：



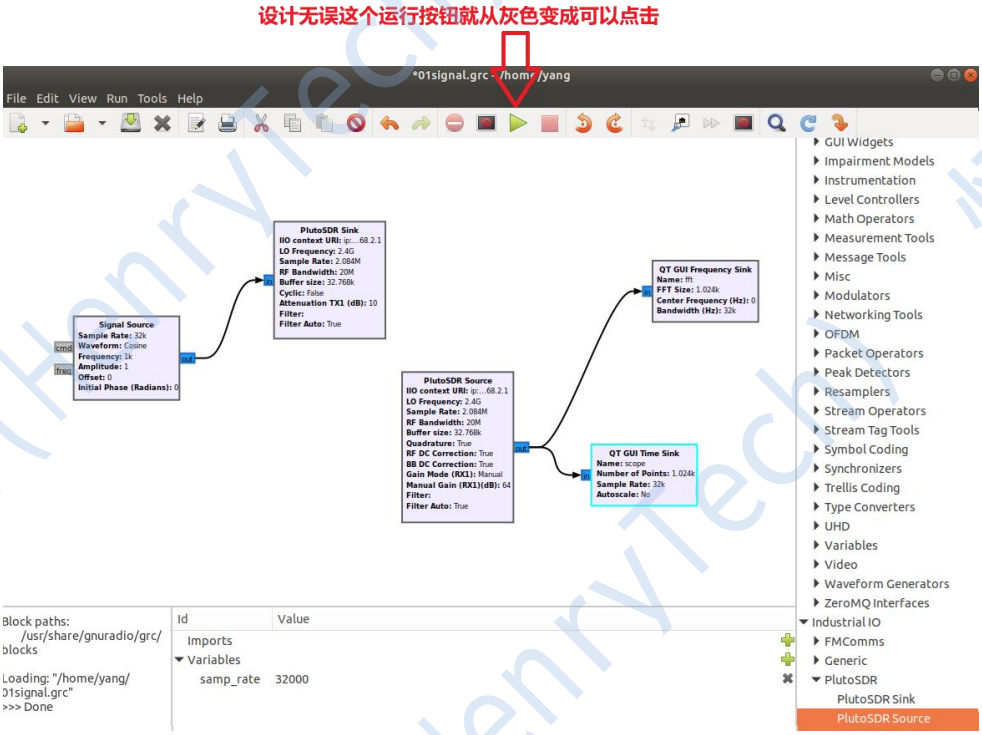
到此就有硬件支持了。将设计中的连线全部删掉，看一下各个模块的接口颜色：



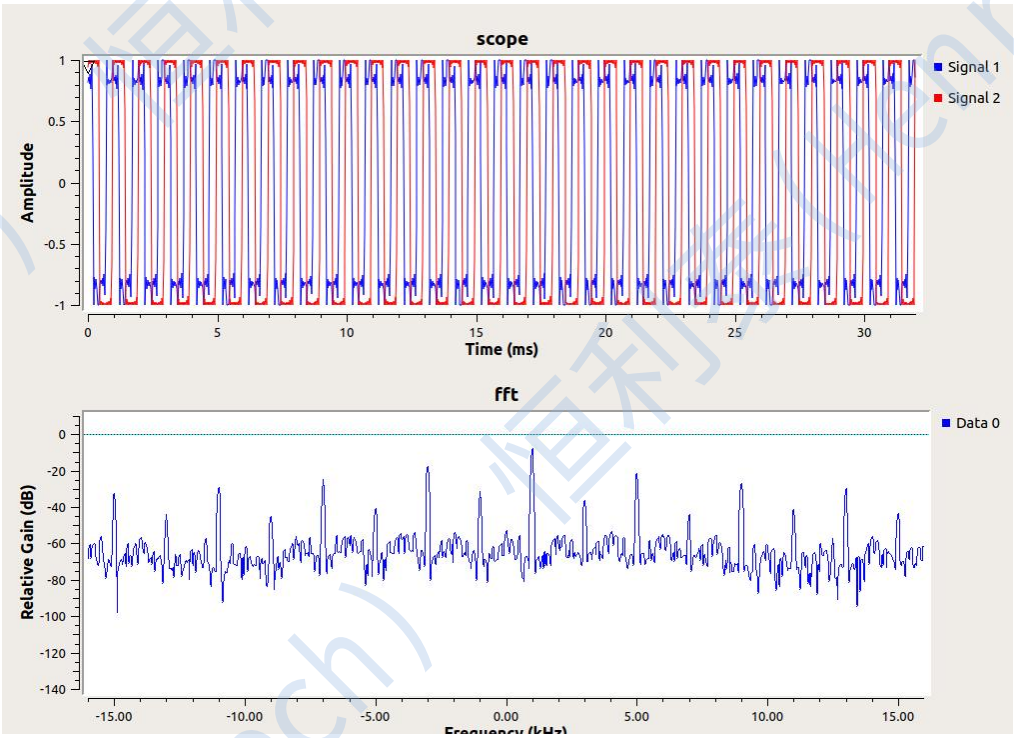
我们在之前讲过，菜单栏的help下拉可以查看type，上面模块中，橙色是float，蓝色是复数（complex）。复数代表什么？我们可以回忆一下，在9361接口接收发送中，一个通道可以传输两个分量的数据，I和Q，这两个是空间上90度相位的关系，这就类似于我们复数的表示关系。所以我们PlutoSDR Source和PlutoSDR Sink模块输入输出均为复数接口，复数为蓝色。我们需要把其他的几个模块接口全部改成复数，首先是Singal Source，第一项改为complex即表示复数，点击OK完成配置：



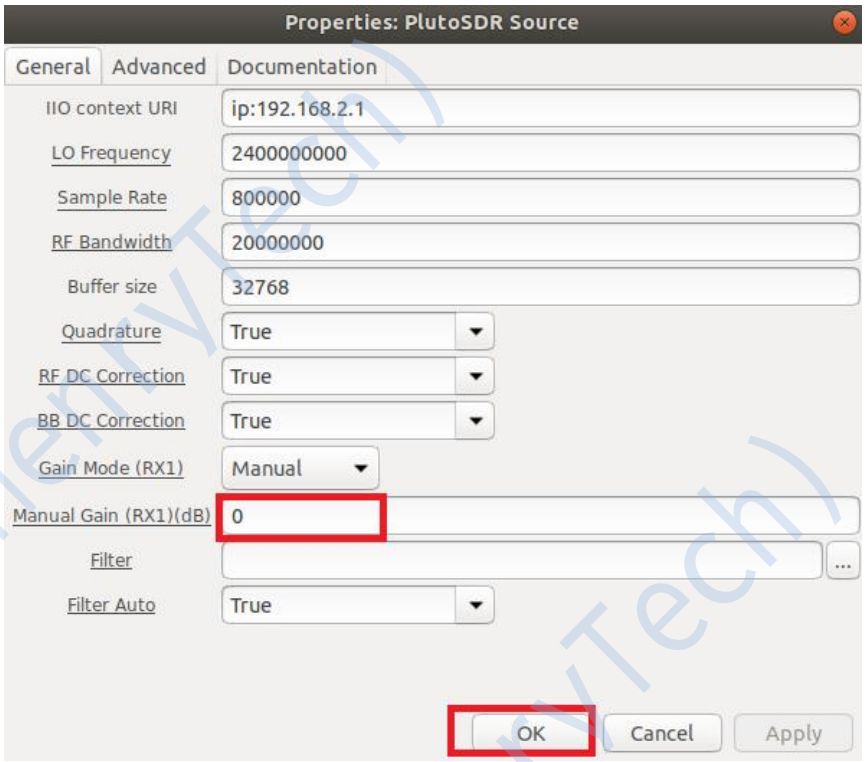
同样地将QT GUI Frequency Sink和QT GUI Time Sink接口Type改为复数complex，接着就可以开始连线。将Signal Source的 OUT连接到PlutoSDR Sink的IN，将PlutoSDR Source的OUT连接到QT GUI Frequency Sink，同时连接到QT GUI Time Sink，模块整理拖拽一下，最终如下：



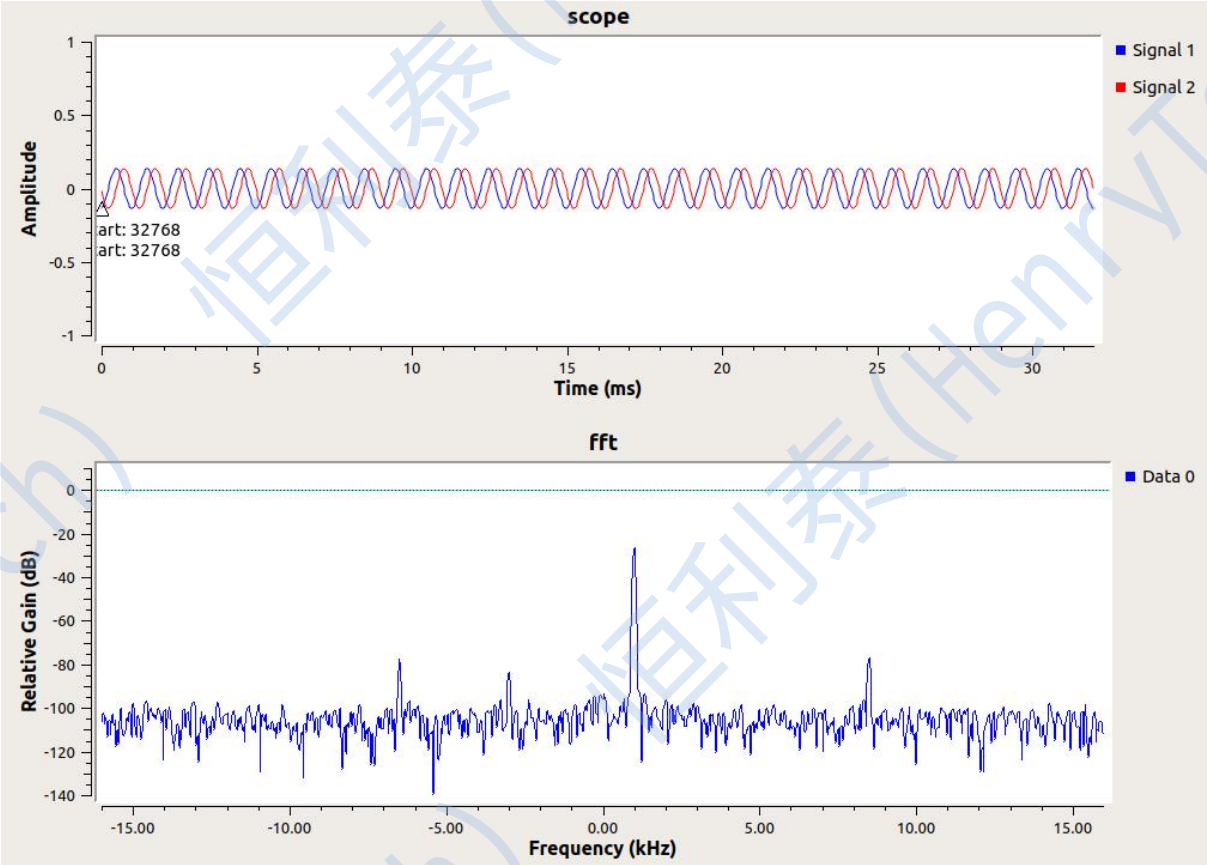
在运行之前，我们将SMA线连接TX1A和RX1A准备实验。设计无误gnuradio有自动检查功能，运行按钮设计没有完成，或者有错误，没有正确连接的信号，就是点不动的灰色  。所以能点动就说明设计无误，我们点击绿色按钮  运行，第一次运行有可能弹出警告，点击OK即可，没弹出就不管：



上图中信号有点饱和。我们关掉显示窗口，重新配置一下PlutoSDR Source，我们将Manual Gain （RX1）（dB）改小点，改成0：

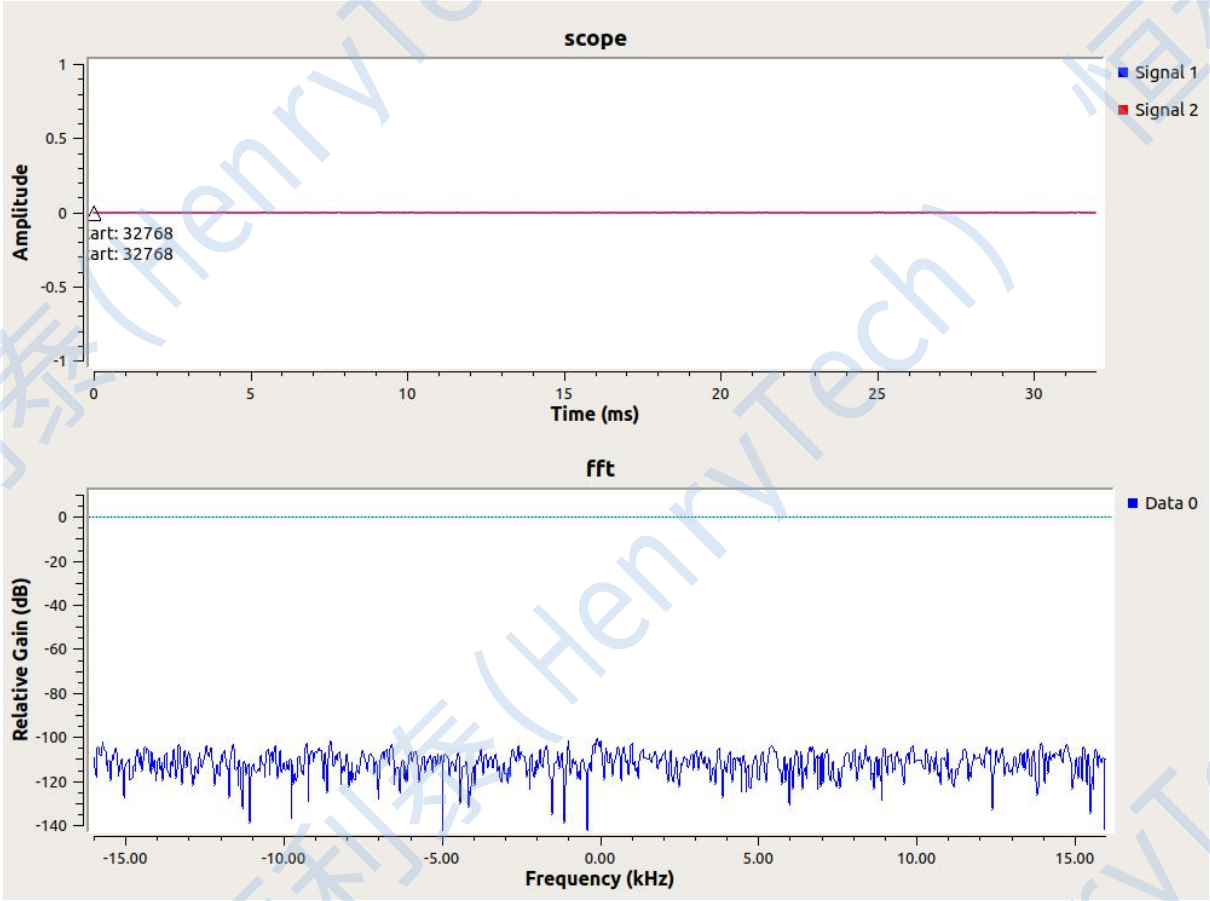


再次点击 运行，可以看到，信号幅度小了很多：



频谱显示我们1Khz的地方有一个主要尖峰，其余的地方有少量的其他品类的干扰，属

于正常现象。由于我们信号源输出1Khz，所以测到也是1K。这个频率与射频2.4G无关，是调制输入基波信号，通过混频2.4G载波发射出去，最后接收后还原，ADC再采样回来的。信号TXA天线连接到频谱仪，看到的实际上是2.4G无线载波信号，是用于承载无线电发射的信号，不是1K。两个概念不要弄混了。要更改发射的载波信号，更改模块中的LO Frequency配置参数即可。注意9361范围是70M-6Ghz，换算成Hz单位。我们可以尝试将SMA线拔掉，可以看到信号和频谱就没有了：



PlutoSDR模块常见几个参数解释：

- LO Frequency：调制的无线频率，载波；
- Sample Rate:AD9361的传输带宽；
- RF Bandwidth：无线滤波频率；
- Buffer size：缓冲的buffer大小。

注意，更改了PlutoSDR Sink的无线配置，一定要同步修改PlutoSDR Source的无线参数一致，否则收发信号实际测试会出错。

本教程到此结束，以上就是带大家简单入门的使用实际操作的内容。GNURadio实际上有很多的功能，我们这里仅仅简单地调用了一个单一的信号进行实验，还有更多的功能和设计大家自己研究。关于GNURadio更多的详细使用直接参阅https://www.gnuradio.org/。

二十三、一些常见问题

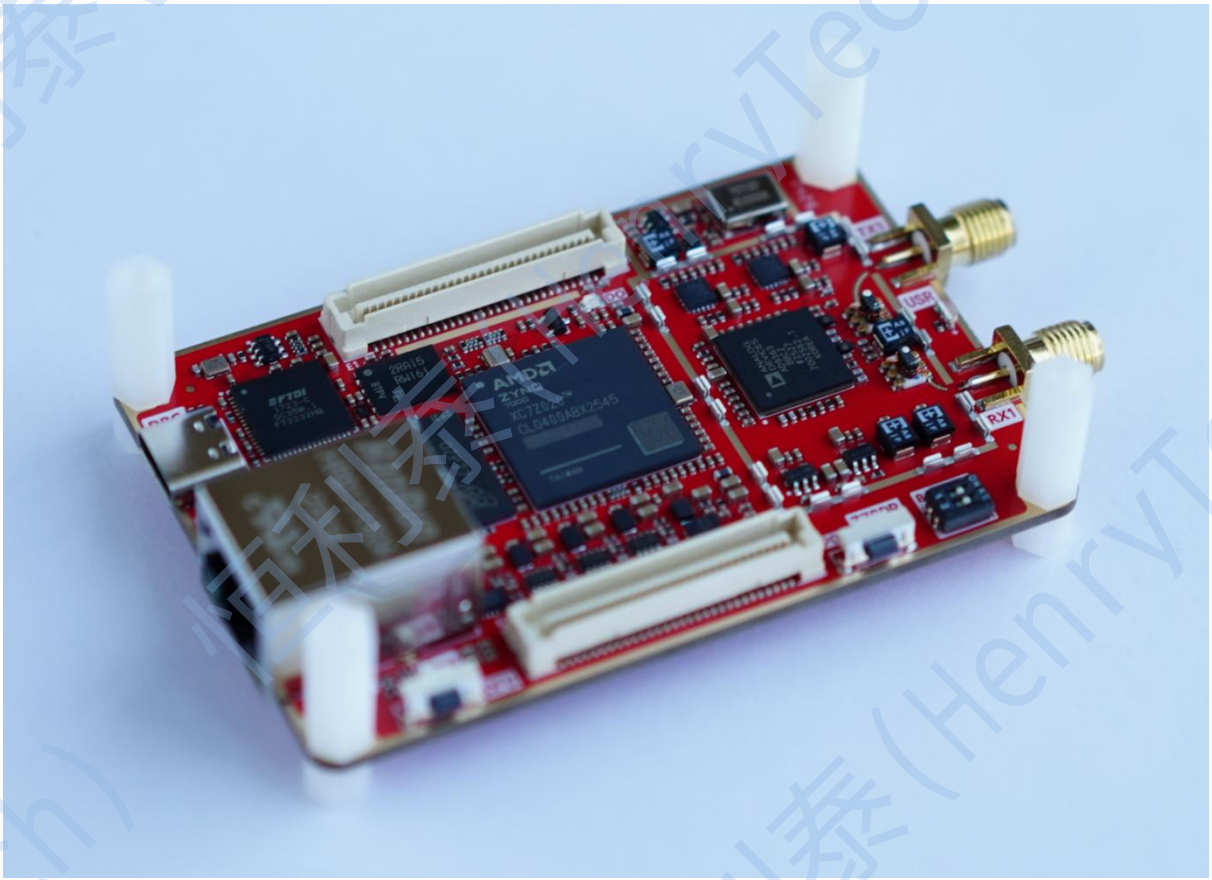
1. 网络问题：常见的虚拟机可能会出现网口不通等情况，注意IP地址。

2、Gnuradio数据带宽问题：采样率不能太大。

3. 上图开发GNURadio，基于ZEDBOARD+fmcomms3也可以使用相同的开发。

使用matlab2018a的硬件支持包安装

------ 基于Z7SDRMicro开发平台



目录

二十四、文档介绍

本教程给大家介绍如何使用matlab2018a进行安装硬件支持包，可以使用simulink进行软件无线电做基本算法验证。关于matlab安装，这里不做介绍，用户需自行下载安装。用户可以在matlab官方注册并且下载matlab，注意，本教程给出的是2018a版本，相关硬件支持包也是基于此版本。

二十五、安装驱动和开发板IP配置

硬件开机启动后需要配置好IP地址，详见《01快速测试及使用指南》第六节的SDR#快速测试配置方法，开发板地址为默认192.168.1.10，电脑与开发板一个网段，192.168.1.3，配置过程方法这里不赘述。

二十六、Simulink简介

Matlab Simulink 是一个图形化界面，并且以模块化设计为基础的仿真工具，集成 了Matlab的基础组件。针对一些设计更适合以模块为基础，而不是以代码为基础的场合，由于针对各行各业的科学计算和工程算法等等，很多时候跨领域应用的科学工作者不擅长编程，这种情况下图形模块化界面就显得非常友好。并且由于matlab中集成Simulink，simulink不作为一个完全独立于matlab的软件 ，有一些机制的算法可以实现模块和代码共同工作，协同完成一个复杂的工程算法验证。举个例子，我们在Matlab 中的代码可以放在 Simulink 的模块里运行工作，并且也可以在模型加载的时候，自动执行 Matlab 的代码，也可以将我们的自定义变量作为模型的基本参数，这是因为 Simulink 是集成在Matlab中的，所以是共享workspace工作空间。在matlab常见使用中，混合Simulink图形模块化设计与源代码一起使用，是比较常见而且很方便的。我们经常可以看到，一些基础科学的数学验证，其仿真模型等等验证算法都会有matlab simulink的影子。

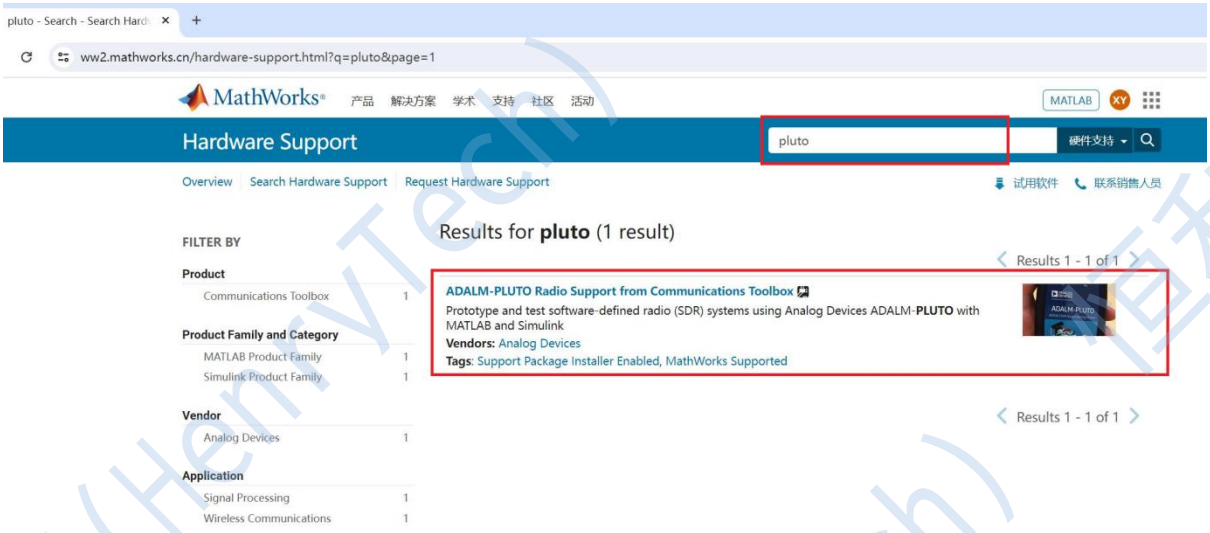
二十七、在线安装硬件支持包

由于官方的AD9363的标准libiio驱动设备均可使用，所以我们直接使用ADI官方的ADALM-PLUTO硬件支持包。与我们Z7SDRMicro软件无线电相比，唯一不同的是，在连接方式上，前者使用USB，而我们使用以太网口进行。以太网口可获得更高带宽。需要了解ADALM-PLUTO，可以在wiki官方了解更多：

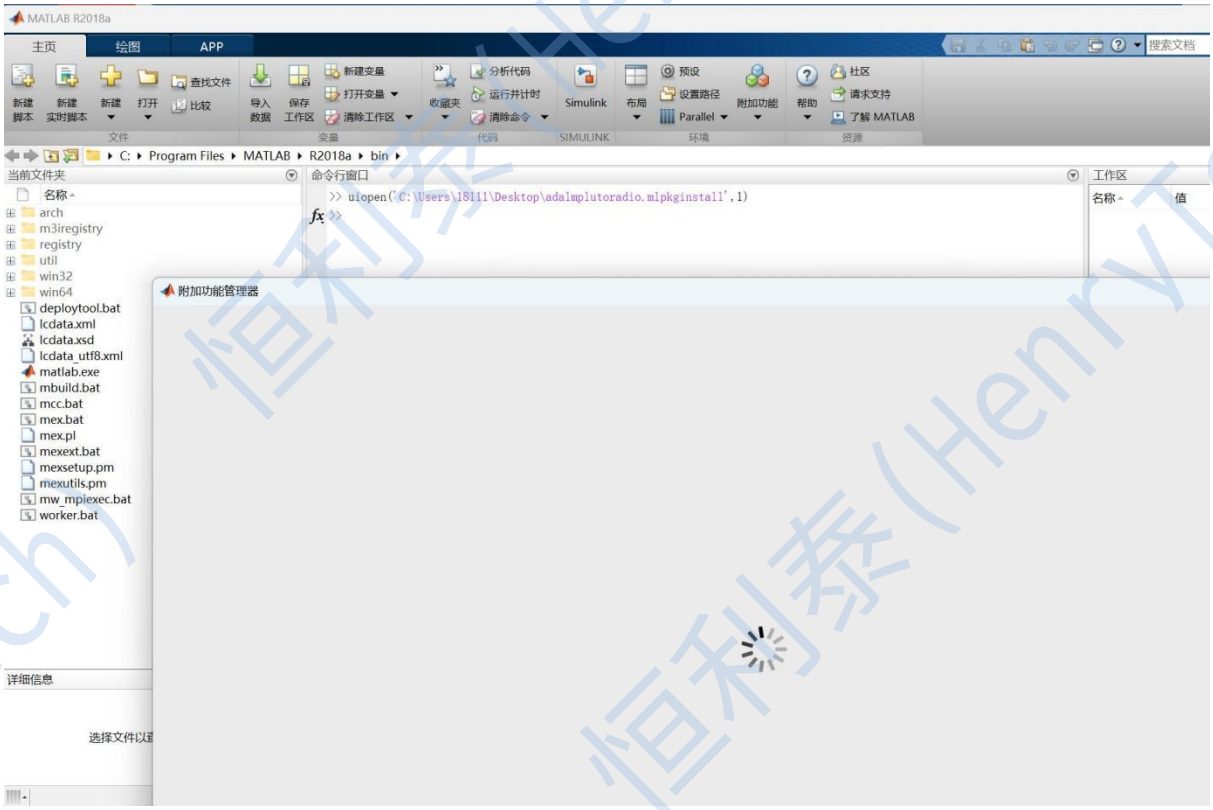
https://wiki.analog.com/university/tools/pluto/hacking/hardware

安装支持包方法多种，我们第一种方法，就是没有离线安装包的时候，在线安装，我

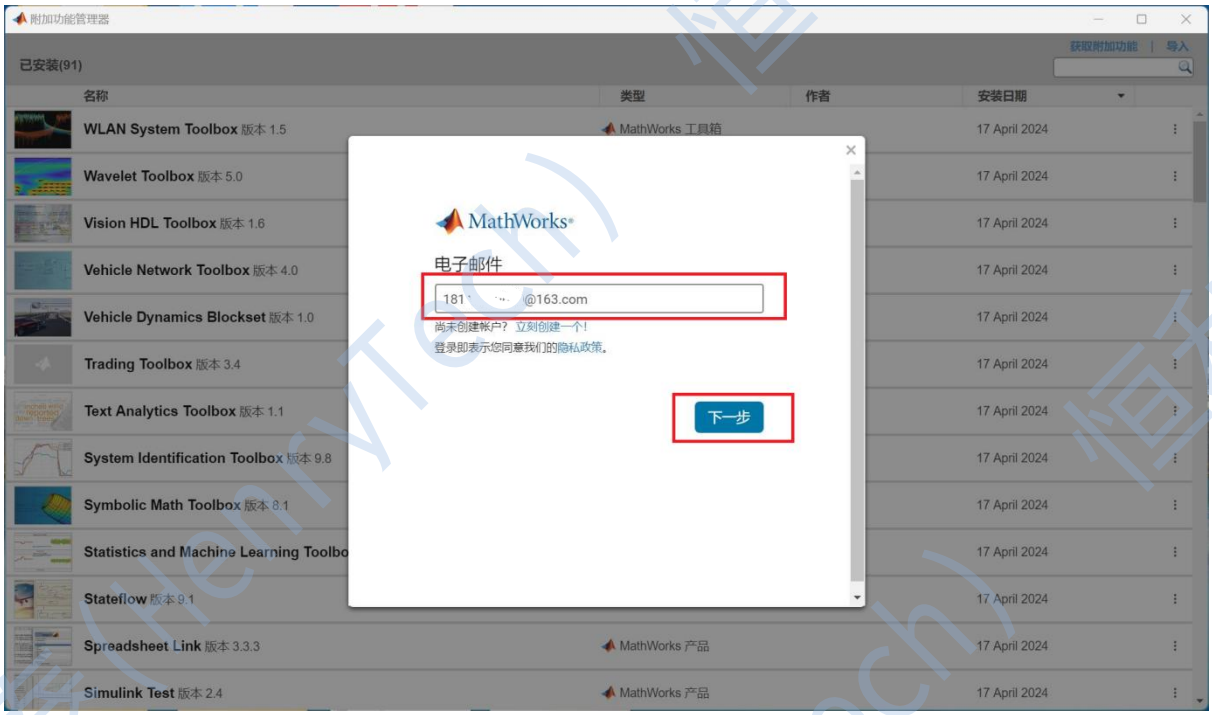
们在matlab官方网址https://ww2.mathworks.cn/hardware-support.html?q=&page=1找到如下链接，或者上方搜索ADALM-PLUTO（可直接输入pluto），找到如下支持之后点击进去：



点击进入后就是关于这个硬件支持包的完整链接页面。我们在页面上Get support package点击后即可下载一个安装程序，浏览器自动下载。下载的文件adalmplutoradio.mlpkginstall我们在启动matlab主界面后将这个文件拖进matlab主界面中打开进行安装，注意，拖进来会自动打开弹出一个新的界面，比较慢，耐心等几分钟：



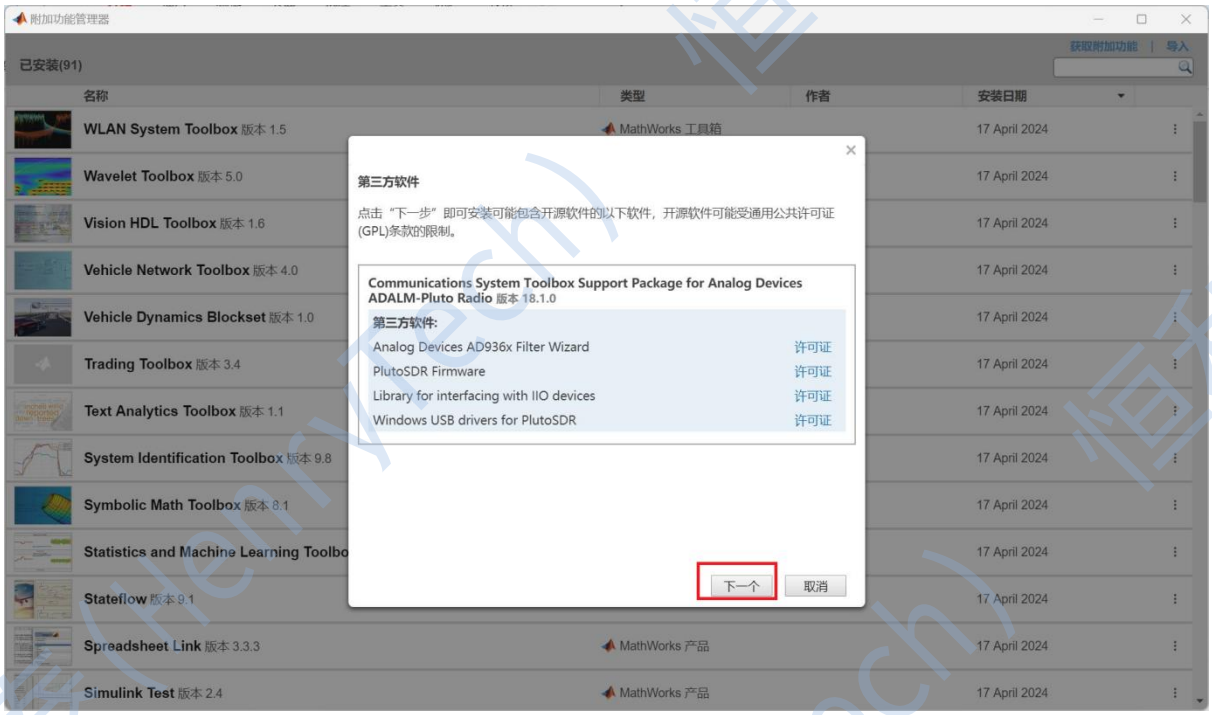
弹出的界面出现登录框，需要登录matlab账号，我们填入账号邮箱信息进行登录：



上图中点击下一步输入密码完成登录，登录时间较长，耐心等待。

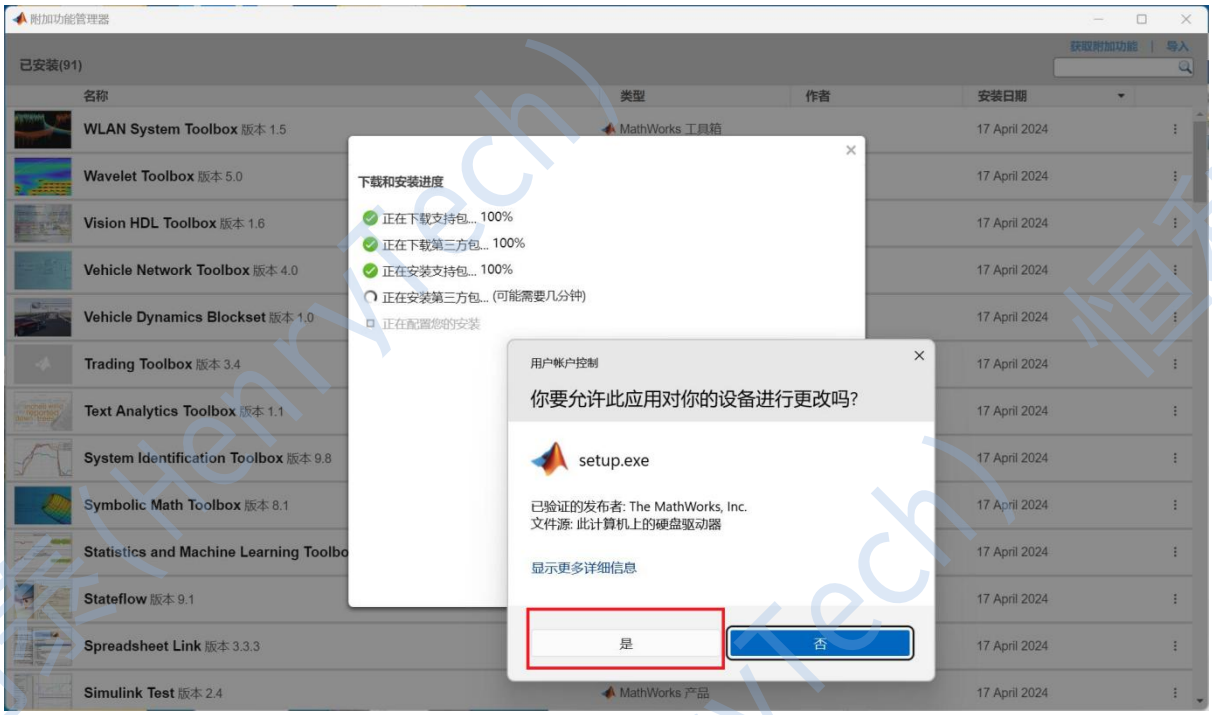
登录完成后，弹出的许可协议点击我接受：

接着弹出的界面点击下一个：

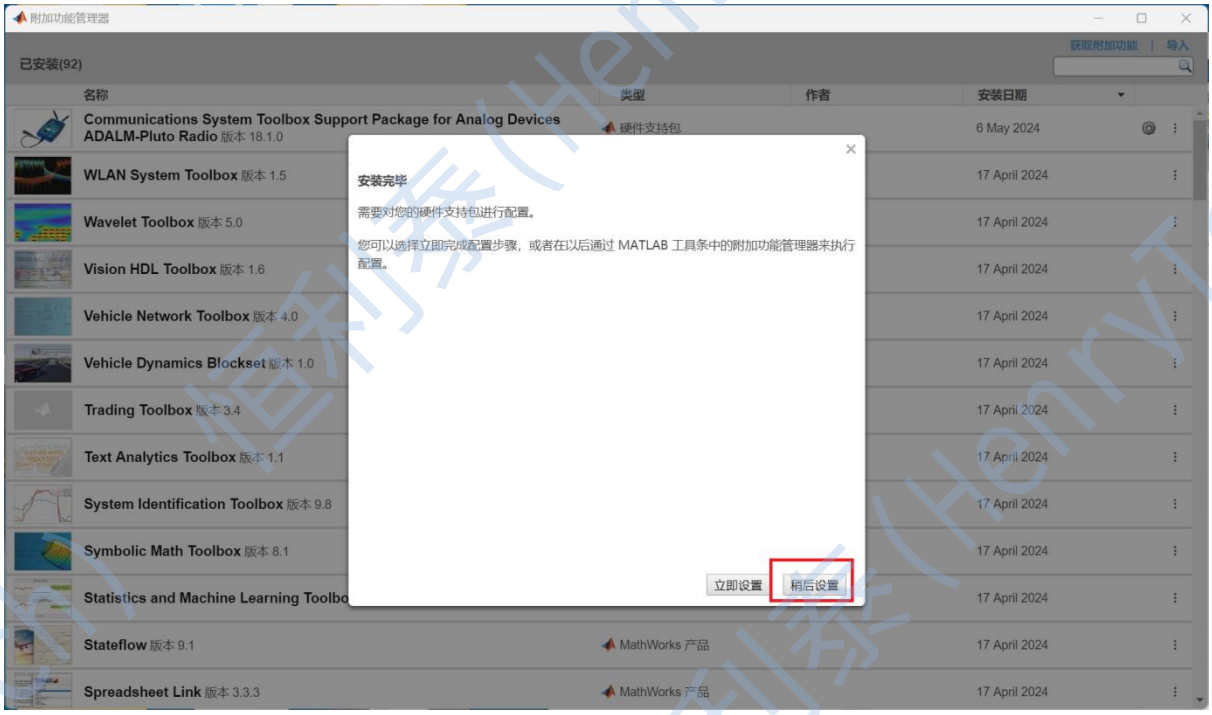


接下来才是开始安装，首先会下载支持包：

中途下载完安装包会弹出界面，点击是即可：



最后弹出的界面选择稍后设置：

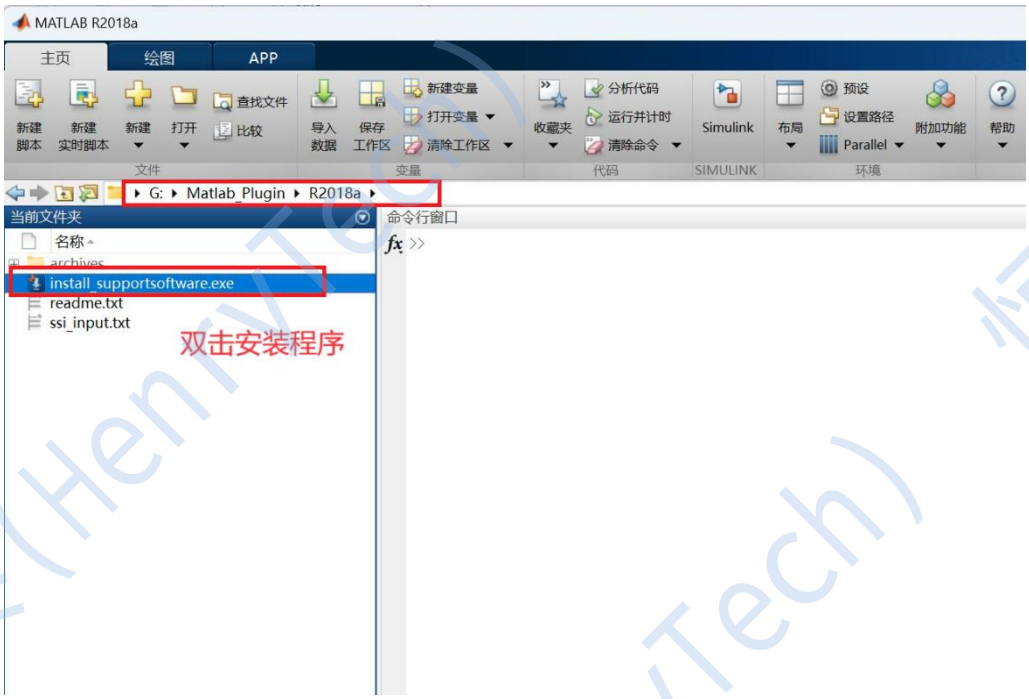


上述方法是第一种标准安装流程，适合任何支持包安装。但是依赖于网络，有的包可能下载失败，无法安装。我们接下来介绍第二种方法。

二十八、离线安装硬件支持包

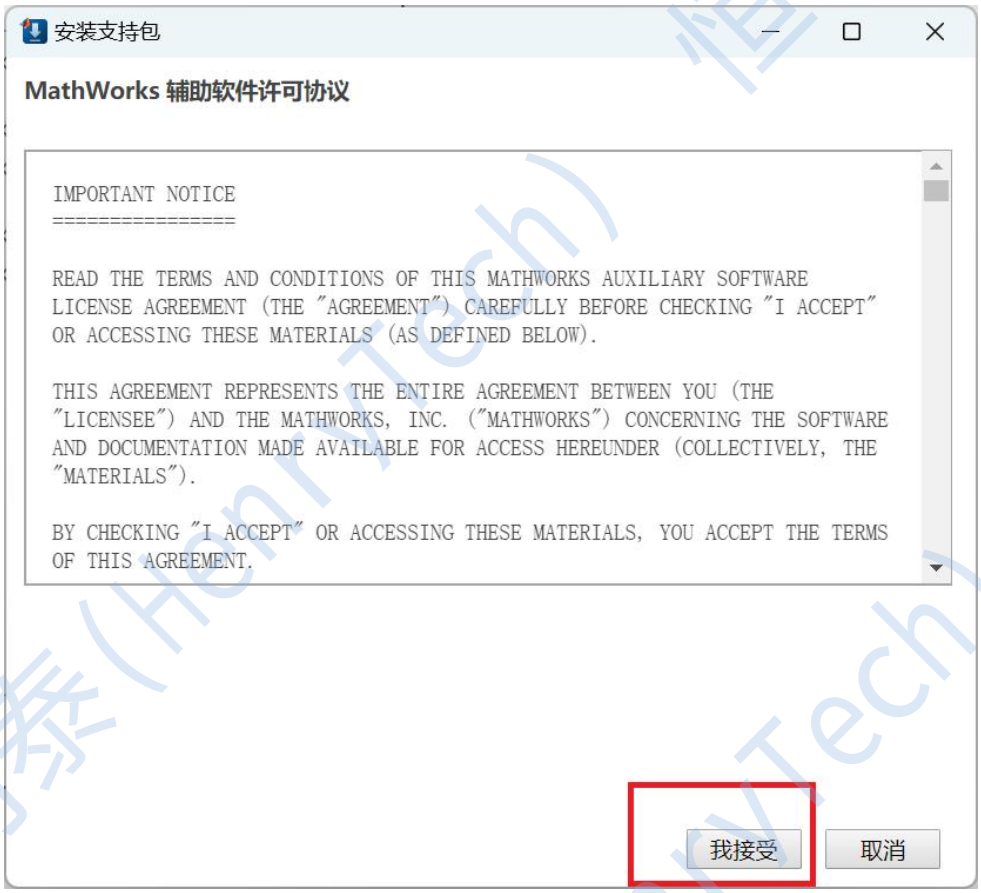
网盘目录中提供了硬件支持包。位于网盘资料的《Z7SDRMicro\01工具软件》中我们解

压《R2018a_Hardware_Support_Package.zip》待用。我们打开matlab主界面左边，最上方的路径定位到解压的目录中找到安装程序并且双击安装：

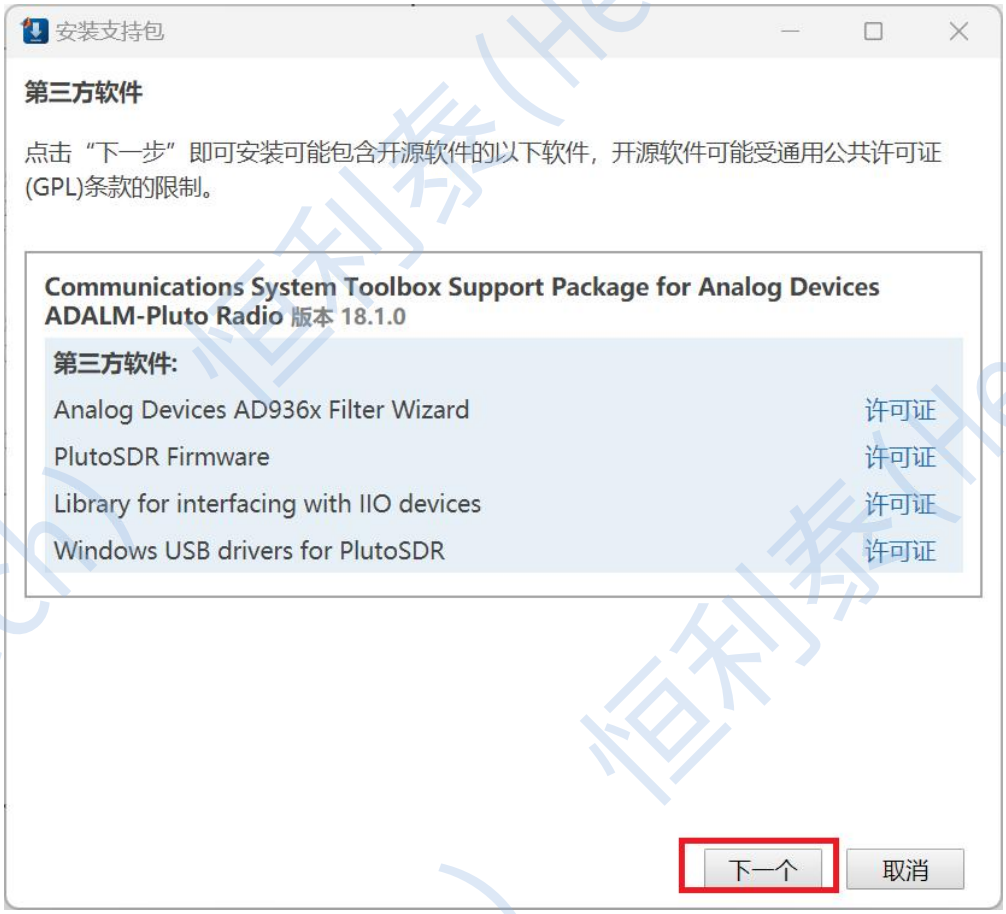


弹出界面可能会让登录，大家登录matlab账号即可，如果matlab已经登录，就直接出现如下界面，我们选择第一项，点击下一个：

点击我接受：

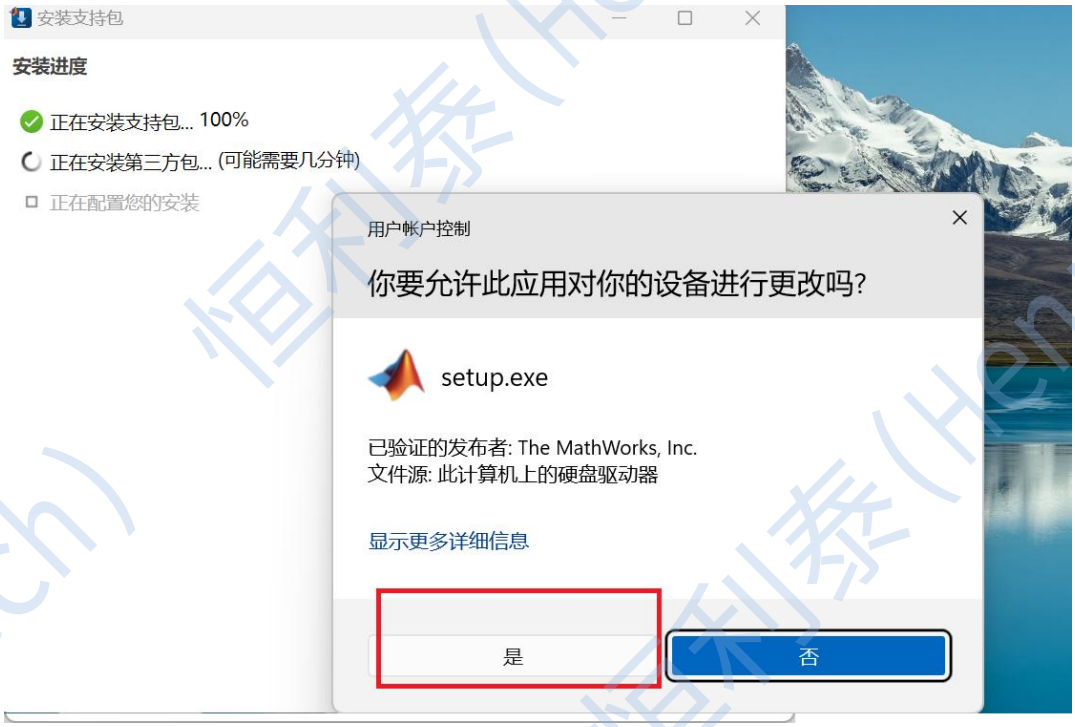


接下来点击下一个：



这里安装过程少了几个下载的步骤，所以不会出现因为包下载失败导致安装失败：

弹出界面点击是：

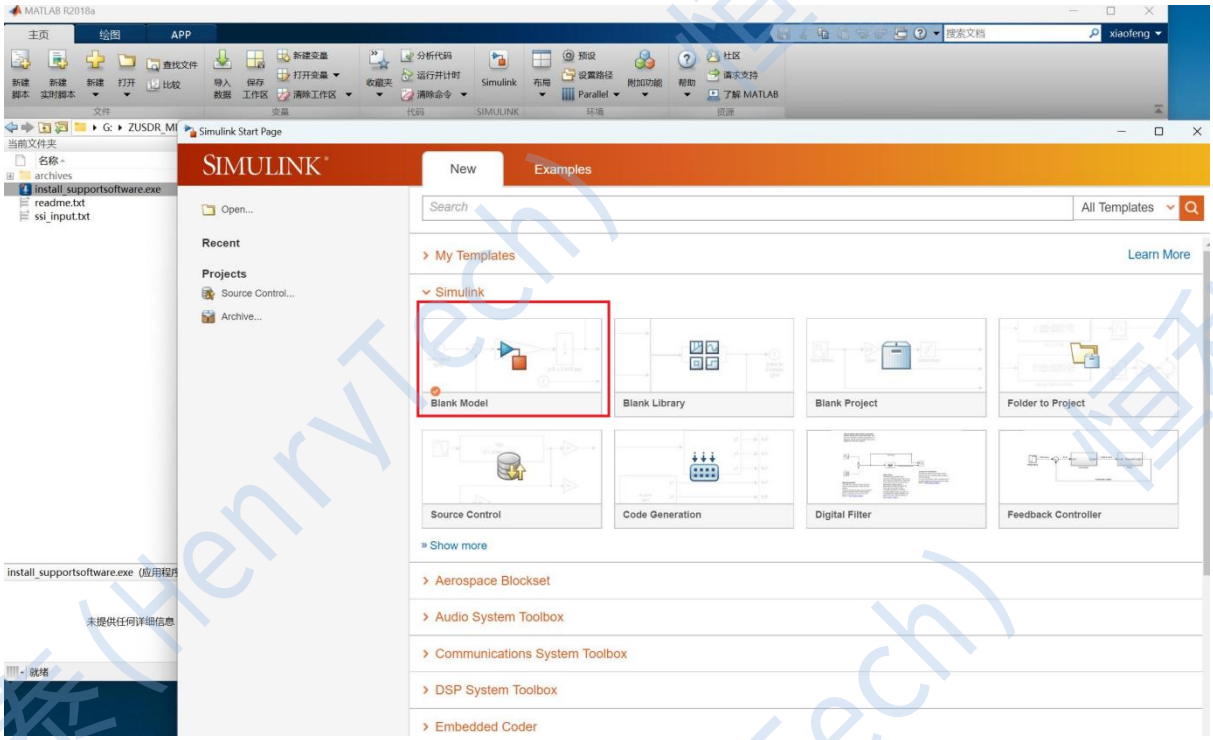


最后一步，还是点击稍后设置：

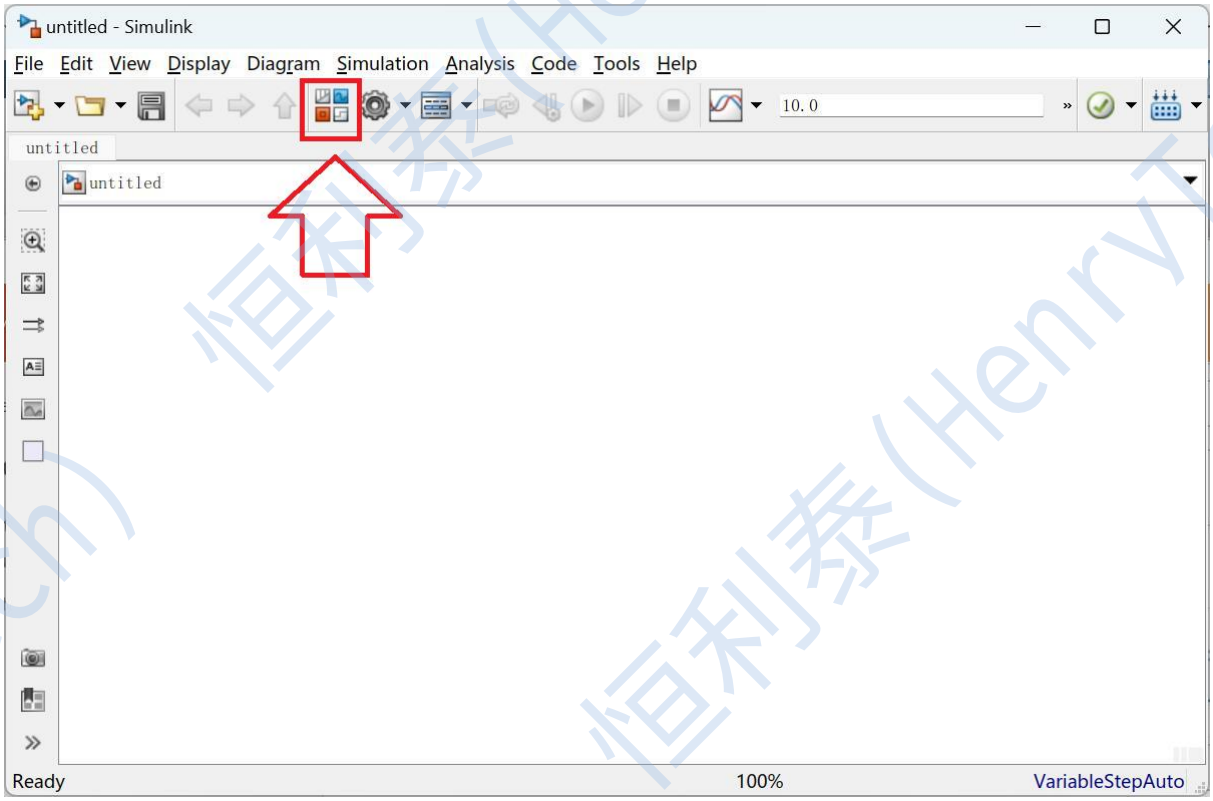
到这里我们安装完毕。

二十九、Simulink查看硬件支持库

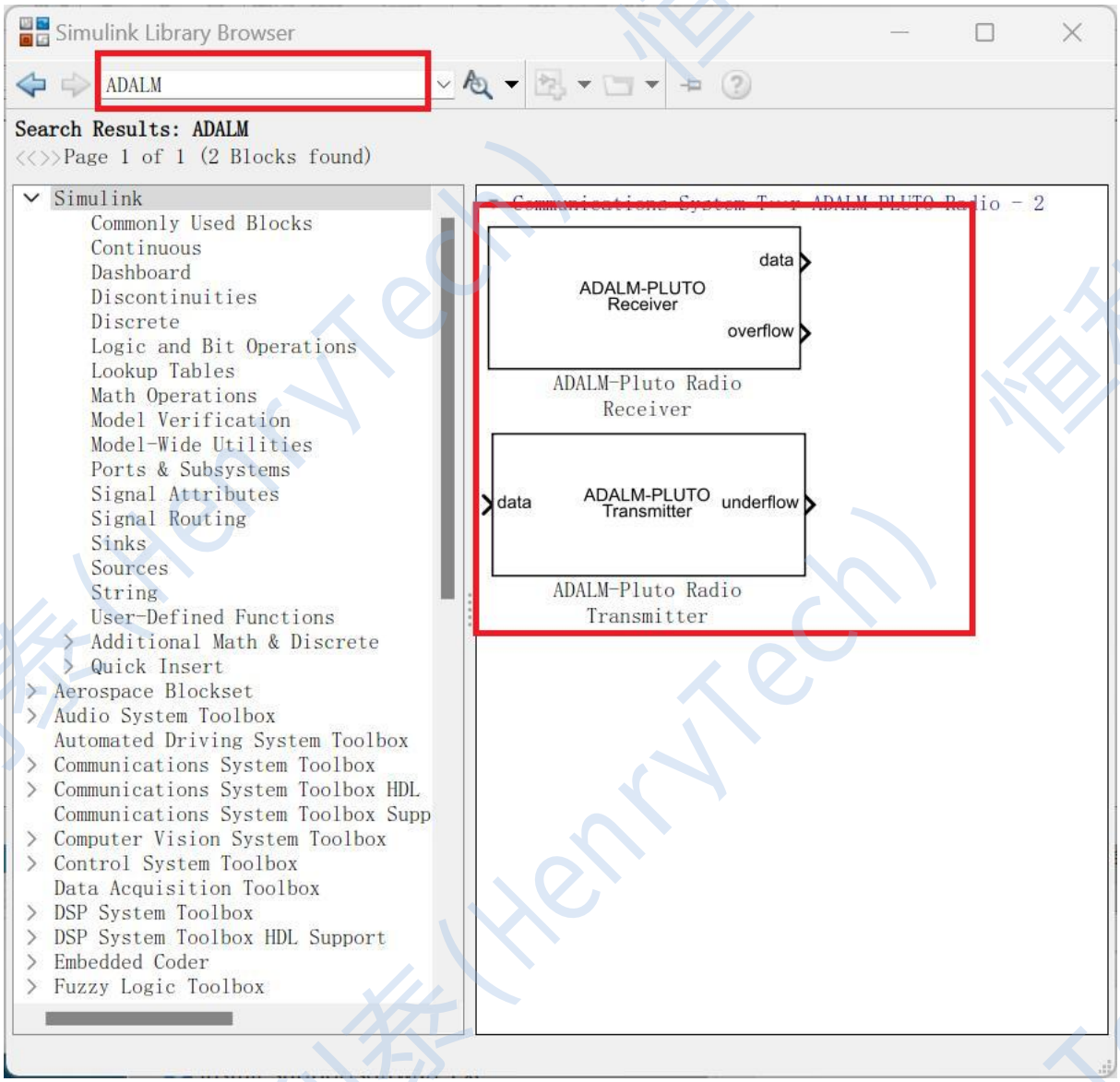
我们启动simulink，点击右边窗口中的blank model：



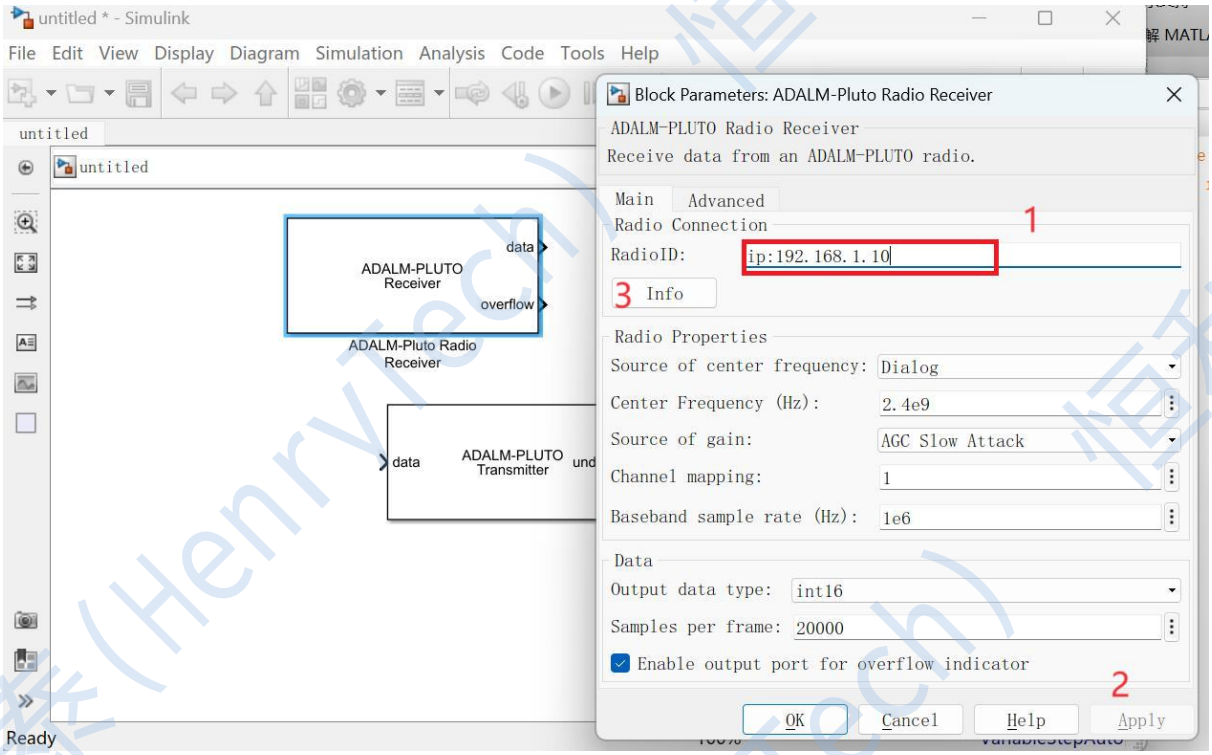
然后会弹出一个打开的Simulink模型空白界面，我们点击最上方的Library Browser打开库：



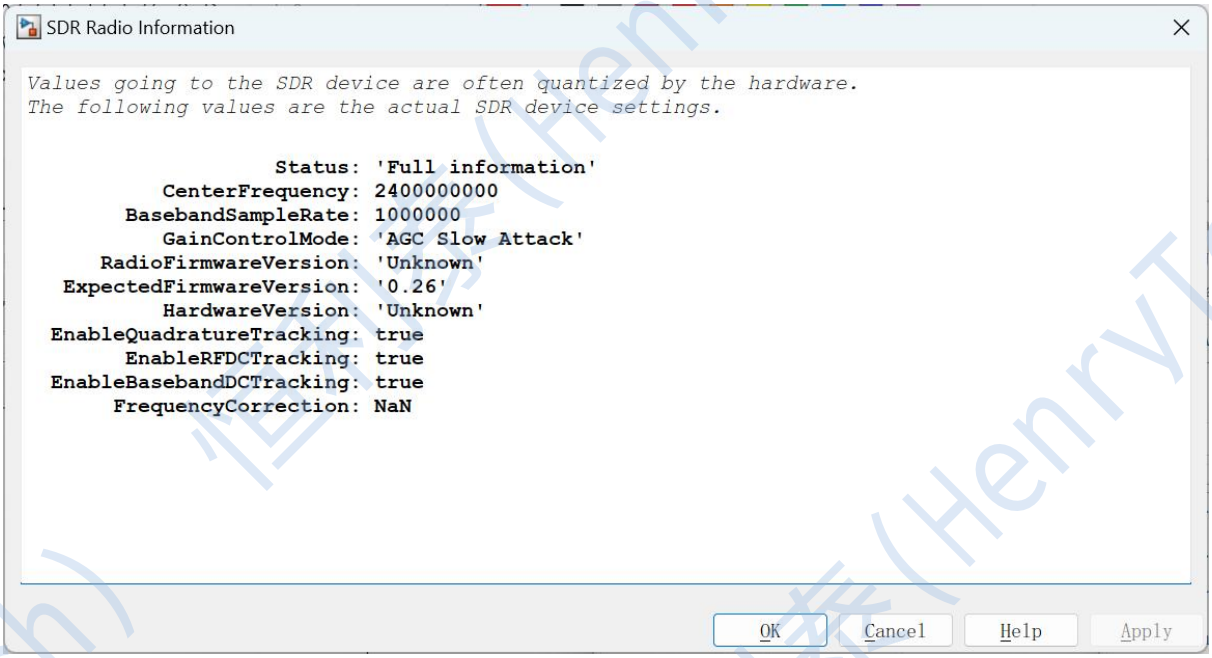
打开的库支持如下，可以看到各种模块，我们可以在上方的搜索栏输入ADALM找到两个模块：



这就是我们simulink所需要的SDR数据收发的子模块。我们将其拖入空白的simulink的文档中，双击其中一个打开配置界面。我们在配置之前，我们将Z7SDRMicro连接千兆网线到电脑，开机启动，大家先按照《01快速测试及使用指南》中第六节，配置好以太网，保证Z7SDRMicro能够和电脑通信，过程这里不再重复。配置好后，我们在ADLM-PLUTO配置界面将Radio ID改为ip:192.168.1.10(zusdr网络配置的时候自己配置的ip），再点击apply，最后再点击info（这里请注意顺序，ip填好必须先点击apply再点击info才会识别）：



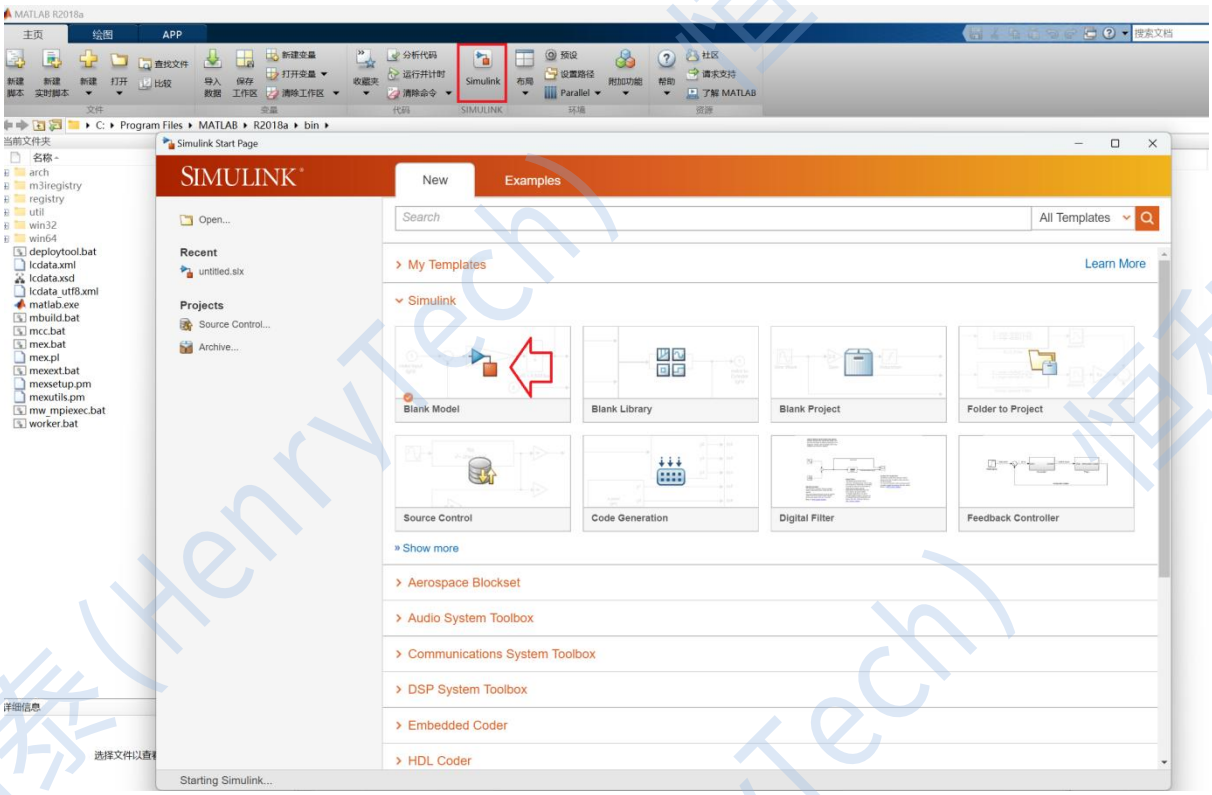
Info点击后弹出如下信息，表明已经识别到SDR了：



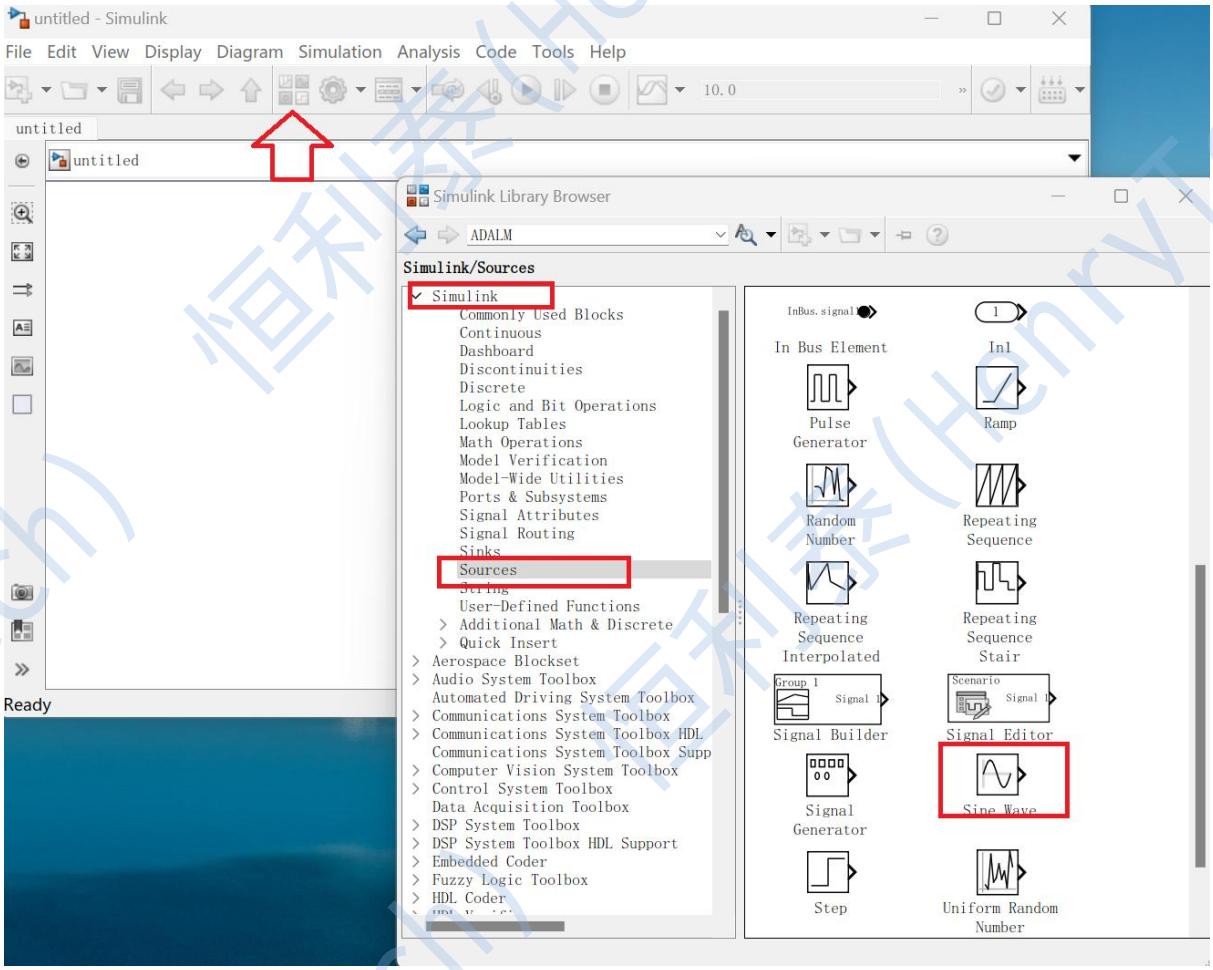
至此我们matlab的simulink的Z7SDRMicro开发环境就搭建好了。

三十、Simulink最简单的正弦单音信号仿真

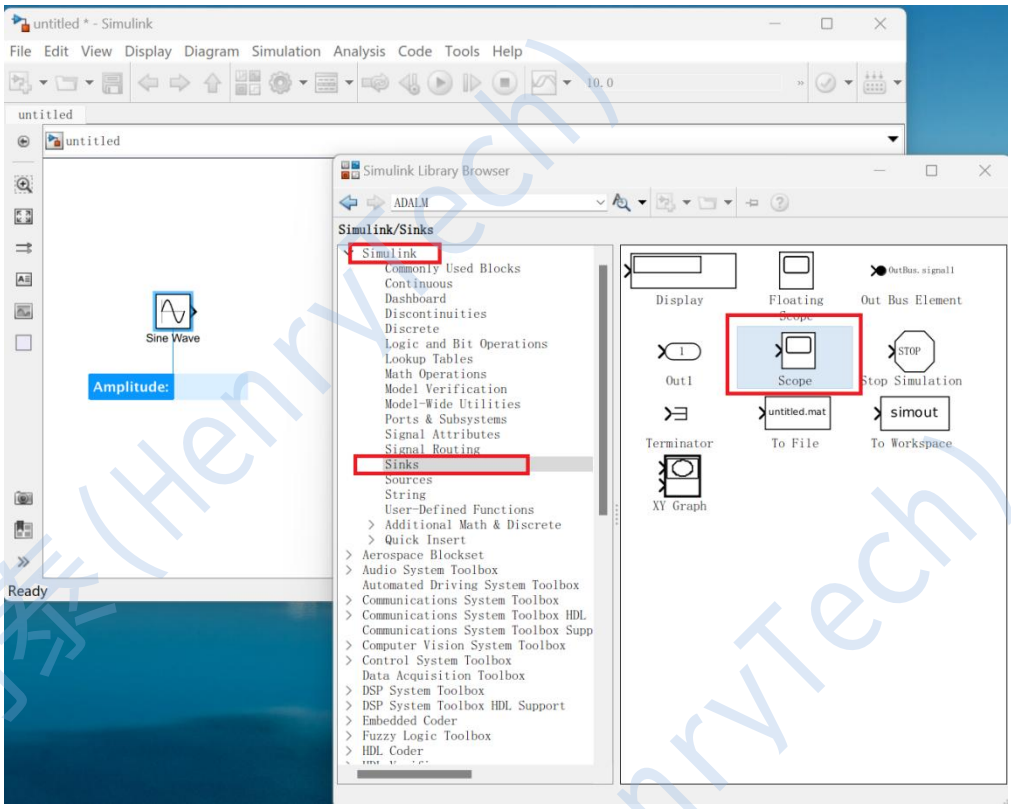
我们按照第6节打开Simulink新建一个空白文档：



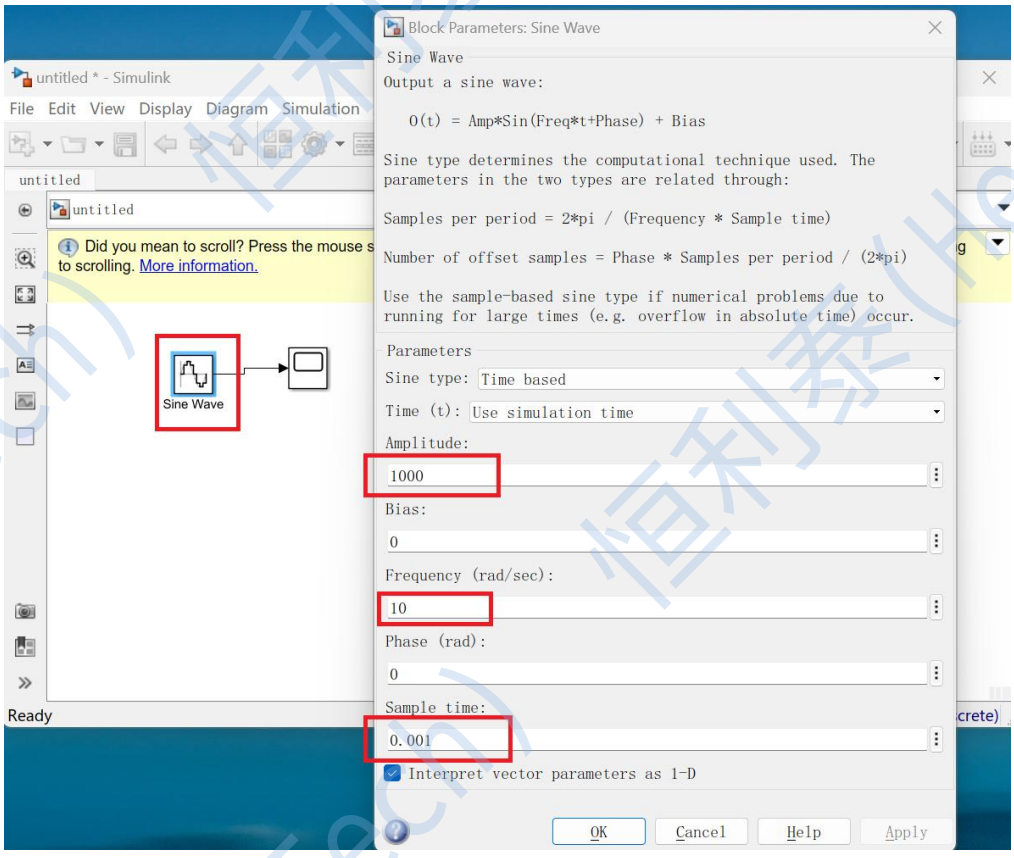
接着打开的文档我们点击仿真库，找到sine wave正弦信号，位于simulink的source信号分类列表中如下位置：



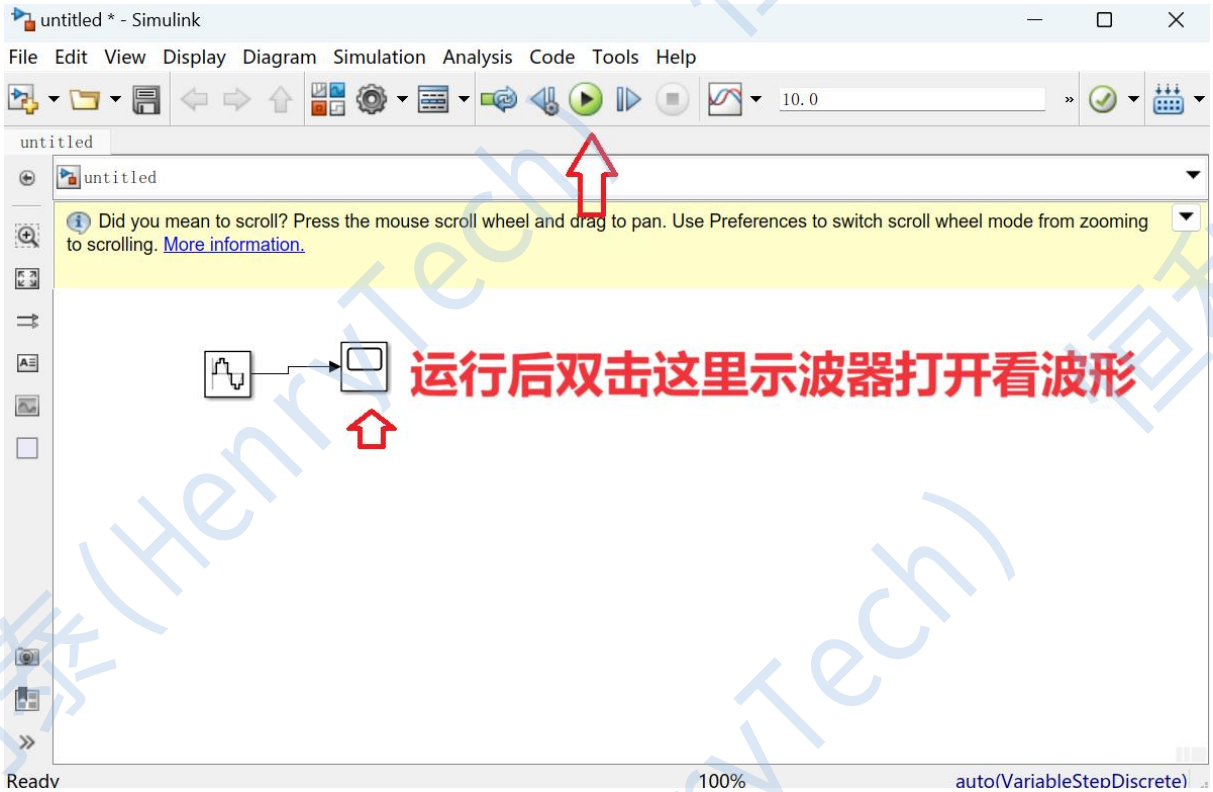
我们将sine wave模块拖至新建的空白simulink的文档中，然后再添加一个示波器，我们在simulink的sinks分类中找到scope拖进文档中：



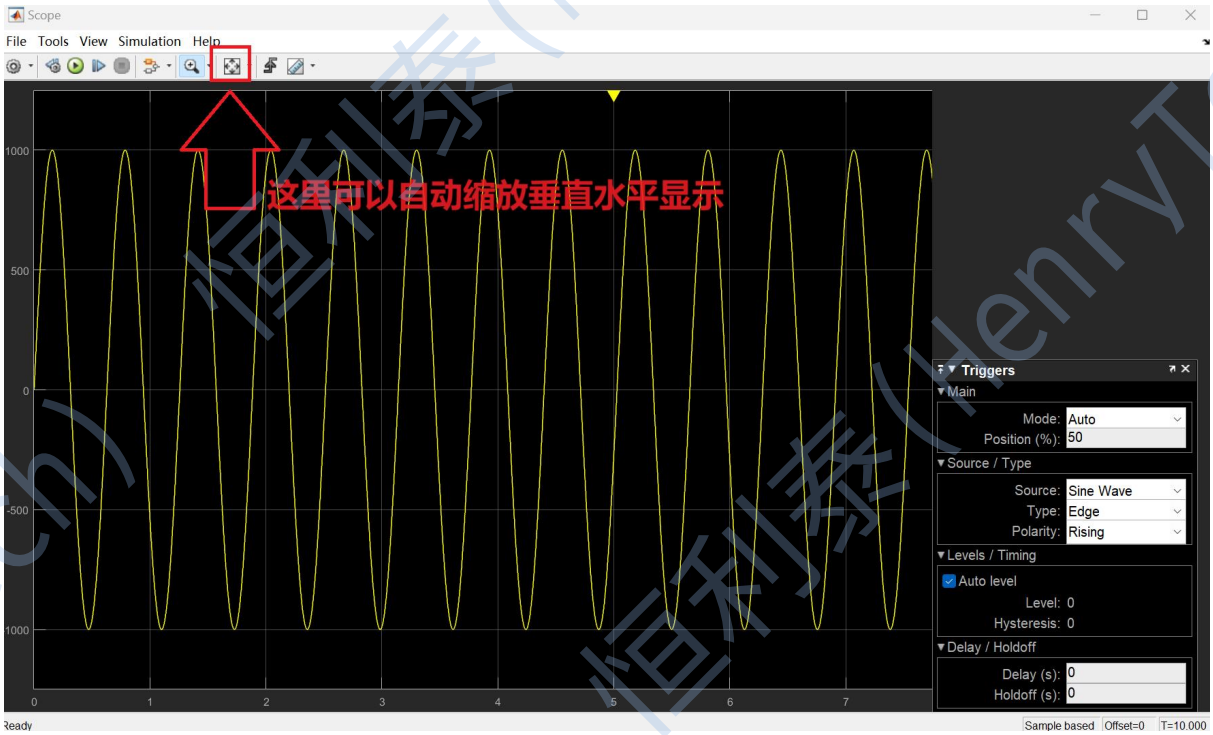
最后点击sine wave的输出，再点击scope的输入完成连线。我们打开sine wave配置界面，这里参数默认即可，幅值1000，偏移0，相位0，频率我们可以改成10，采样时间0.001表示1ms（毫秒）采样一次，关于采样时间，可认为是采样细分精度时间，越小波形越细腻，设置为0表示最高细分，类似示波器采样频率的倒数，对应的采样一个点的时间周期：



最后点击文档上方运行，运行结束我们双击示波器模块打开看运行结果：



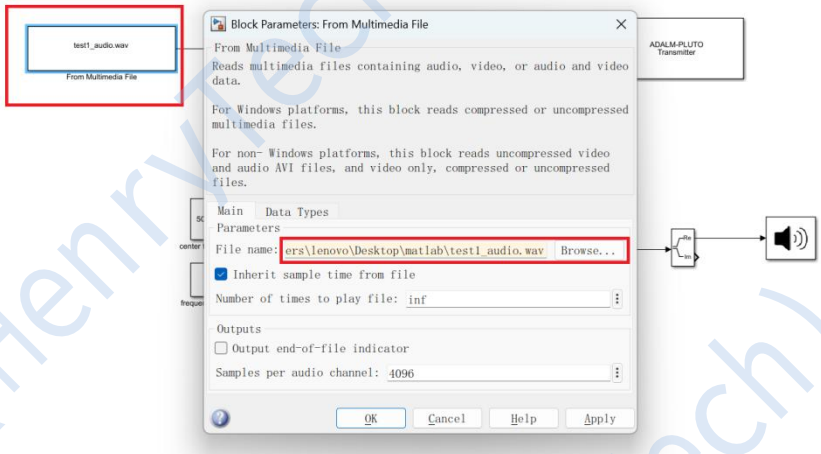
可以看到示波器中的正弦信号就输出了，上方的狮子箭头点击可以自动缩放垂直水平缩放，大家可以点击一下调整显示，可以看到调整后显示了多个周期的正弦波形：



到这里我们的单音信号输出仿真实验就完成了。

三十一、例子说明

以am_tx_rx为例，matlab以管理员身份运行，打开simulink，然后打开am_tx_rx.slx设计，我们重点需要设置音频文件路径，如下，我们matlab例程目录中的test_audio.wav：



然后连接好网口设置好了，按照《01快速测试及使用指南》中第六节进行了测试，然后就可以直接运行simulink查看结果。天线可以直接使用SMA回环线进行测试。