版本信息：
版本:REV2020
时间:30/01/2020

米联客 2020 版 FPGA 课程(PCIE-win 系统篇)

Milianke(msxbo) 2020 FPGA Course(MA703FA-35T 5 个 demo)

电子版自学资料配视频课程

常州一二三电子科技有限公司
溧阳米联电子科技有限公司
版权所有

米联客(MSXBO)04QQ 群：516869816
米联客(MSXBO)03QQ 群：543731097
米联客(MSXBO)02QQ 群：86730608
米联客(MSXBO)01QQ 群：34215299

微信扫码注册米联客技术论坛 www.uisrc.com 免费享受更多资源

扫码关注微信公众平台“MSXBO(米联客)”掌握更多信息动态

版本	时间	描述
Rev2016	2016-07-03	第一版
Rev2019	2019-05-09	第二版
Rev2020	2020-05-20	第三版

序：

FPGA 芯片是硬件技术而 FPGA 编程又称为硬件编程语言和流行的各类软件编程语言 C/C++、JAVA、python 等相比，掌握基础的硬件编程语言不是难事，难点是 FPGA 在每个专业领域的应用，只有充分理解了 FPGA，并且具有对自己所处行业专业背景认知，才能真正理解 FPGA 应该用在什么场合更加合适。

从业多年来，亲身经历了 FPGA 的发展历程，也深刻体会到未来 FPGA 应用领域可能发生的深刻变革，FPGA 从简单的逻辑门，发展到现在具备很多高速通信接口，而且最新 SOC 中也集成了 FPGA 单元，实现了 ARM 和 FPGA 单芯片。目前 XILINX 代表了业内领先的 FPGA 技术，已经可以把 FPGA\ARM\GPU\RFDID 等集成到单芯片。FPGA 的逻辑资源也是达到了前所未有的密度以及超大容量。

很多人问我学习 FPGA 是否有前途，这个问题着实难以回答。我们可以一起来探讨以下几种情况，会是 FPGA 发挥作用的场合：

1)、数字 IC 设计工作

数字 IC 设计还是主要以硬件编程语言去设计数字 IC 芯片，通过硬件编程语言，软件可以把语言翻译成电路，自动布线工具可以完成布局布线，之后流片。由于流片费用非常贵，前期也可以用 FPGA 芯片模拟设计的数字 IC 芯片的功能。

2)、高速模数字信号采集分析

高速的 ADC,DAC 的数模信号处理的领域也是必然需要用到 FPGA，在无线通信、雷达信号处理等领域也都会用到。

3)、数字信号高速通信

FPGA 具备的高速接口也非常适合用于高速通信，比如 PCIE 通信、光通信、以太网通信

4)、视频图像

包括图像的拼接、缩放、高效的实时传输等， 4K 视频 、8K 视频领域

5)、硬件加速算法

FPGA 的硬件加速领域也是目前的热门研究，也是 FPGA 未来最有前景的一个应用领域，已经有很多公司利用 FPGA 的硬件加速实现了很好的经济效应，但是目前 FPGA 的加速还没有做到普及，和传统的 GPU 相比，主要难度还是在开发难度上，一般小公司很难有实力取得突破。

6)、通用 CPU GPU 无法完成的工作

如果通用的 CPU 和 GPU 无法完成的工作任务，可以考虑下 FPGA 或者带 FPGA 的 SOC.

FPGA 到目前为止依然是一个小众的领域，如果专门为了学习 FPGA 而学习 FPGA 而不知道如何应用 FPGA，那么这是非常悲哀的一件事情，学习 FPGA 只是学习一门技能，而结合自己专业背景选择最合适的解决方案，解决问题才是最终的目的。

米联客团队励志在 FPGA 领域可以贡献一份自己的力量，我们的目标是，可以减轻 FPGA 从业者基础学习难度、FPGA 应用难度，为中小型 FPGA 团队提供必要有价值的技术资料和硬件支持。我们希望可以和广大客户形成紧密的合作伙伴关系，一起创造共赢，各自实现自己的价值目标。

2020 年注定是不平凡的一年，是米联客团队继续成长的一年，也是米联客发展非常关键的一年，其中 2020 版本教程的研发也是新一年中米联客最重要的核心工作之一。

2020 版本教程是适应当前 FPGA 技术发展，SOC 技术发展，新形势下，米联客做出的战略决策。2020 版本教程需要解决以下几个问题：

1)、FPGA 基础课程，需要解决长期以来没有认真解决好的课程内容，包括：FPGA 的构架、FPGA 常用 IP 使用、硬件编程经验、通信接口应用、代码规范、时序分析

2)、新版本 vitis 软件的使用技巧

3)、更多更详细的讲解高速通信的基础知识和更多的应用解决方案

4)、对于 SOC (ZYNQ 和 MPSOC)Linux 课程做到适合初级入门，并提供丰富的应用 demo

5)、适应未来发展趋势，增加 FPGA 高层次加速算法编程领域的课程内容

6)、教程的构架能够支持 7 系列 FPGA、UltraScale 系列 FPGA、UltraScale+系列 FPGA 和 MPSOC

不管有多麻烦，不管困难多大，面对挑战我们坚信胜利！

米联客团队
2020 年 1 月 30 日

目录

01XDMA 实现 PCIE 通信入门

软件版本：VIVADO2019.2
操作系统：WIN10 64bit
硬件平台：适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA
登录“米联客”FPGA 社区-www.uisrc.com 视频课程、在线答疑！

1.1 概述

为了让 PCIE 通信变的更加简单，XILINX 提供了 XDMA IP 以及配套的驱动程序，并且支持目前主流的 windows 系统，以及 LINUX 系统。而且驱动源码是开源的 ，方便我们自行修改升级。米联客 PCIE 部分教程内容，分 windows 系统篇和 linux 系统篇，windows 系统篇以 WIN10 作为测试系统，linux 系统以 Ubuntu16.04 作为测试系统。

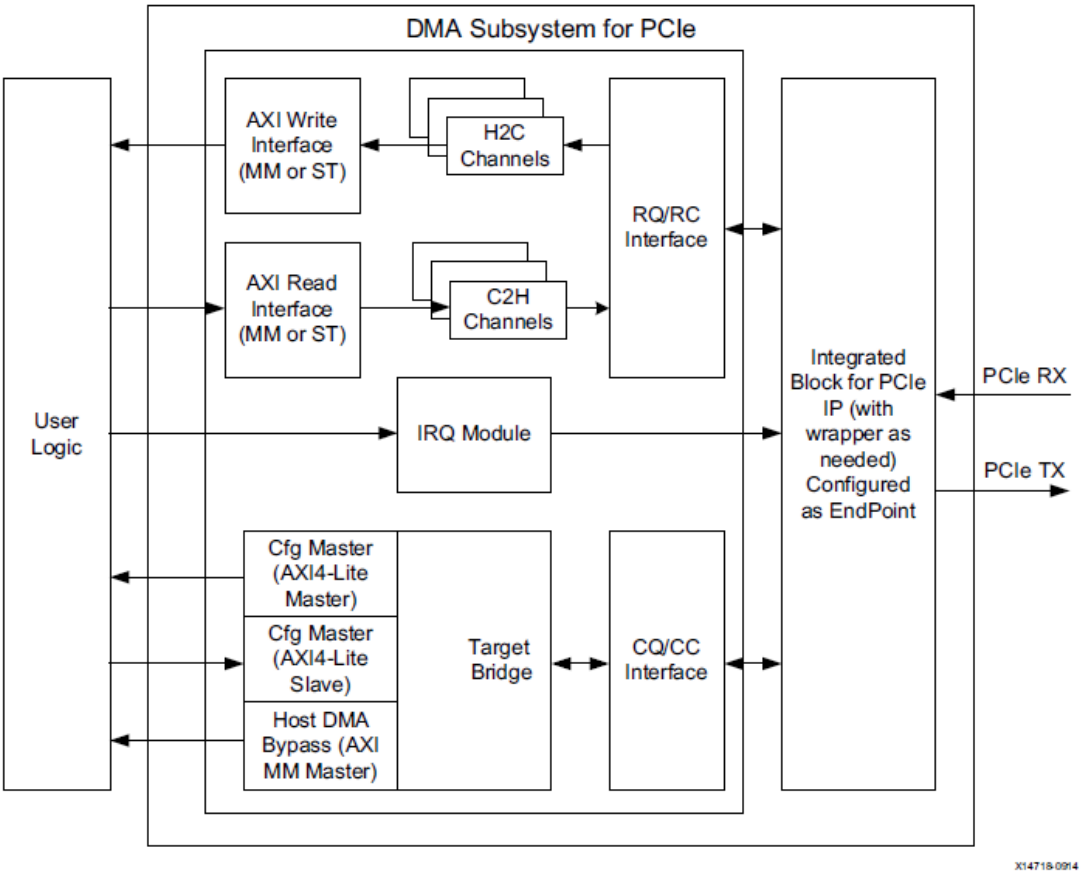
本文内容作为通用的教程内容，适合 XILINX 各类支持 PCIE 通信的板卡。并且米联客在 XDMA 中使用了自己编写的 FDMA 控制 IP，可以简单方便的完成数据之间的交换。

演示用的板卡是一款低端板卡 MA703FA-100T，但是演示的例子方法可以适用于所有的米联客开发板，米联客带有 PCIE 接口的开发板也都会提供配套的演示源码。

1.2XMDA 概述

Xilinx 提供的 DMASubsystem for PCIExpressIP 是一个高性能，可配置的适用于 PCIE2.0，PCIE3.0 的 SG 模式 DMA，提供用户可选择的 AXI4 接口或者 AXI4-Stream 接口。一般情况下配置成 AXI4 接口可以加入到系统总线互联，适用于大数据量异步传输，通常情况都会使用到 DDR(对于 MA703A-35T 资源不够无法 PCIE 下使用 DDR，所以可以用 BRAM 替换 DDR 但是只能缓存少量数据)，AXI4-Stream 接口适用于低延迟数据流传输。

XDMA 是 SGDMA，并非 Block DMA，SG 模式下，主机会把要传输的数据组成链表的形式，然后将链表首地址通过 BAR 传送给 XDMA，XDMA 会根据链表结构首地址依次完成链表所指定的传输任务。

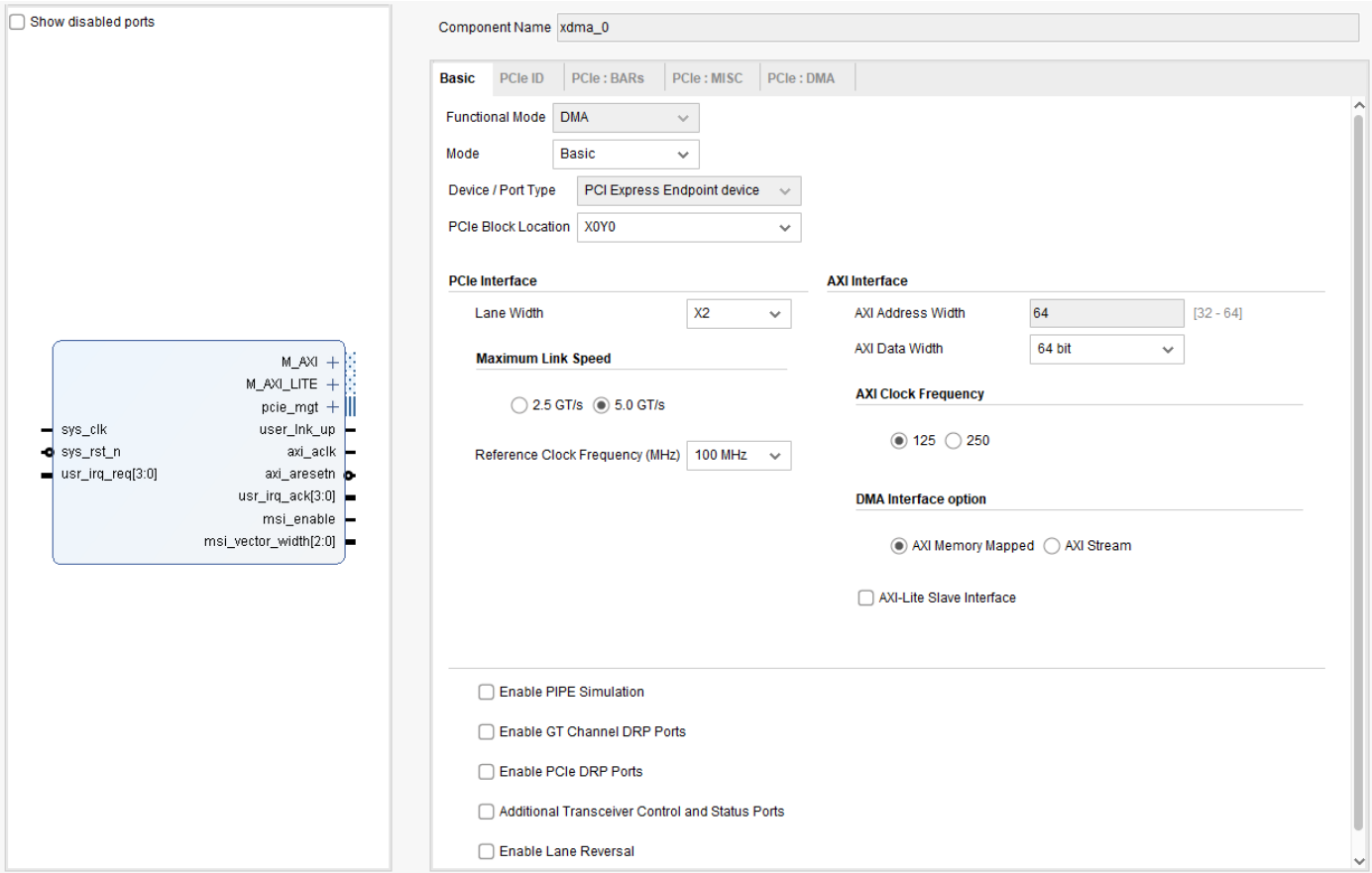


1.3XDMA 提供的接口

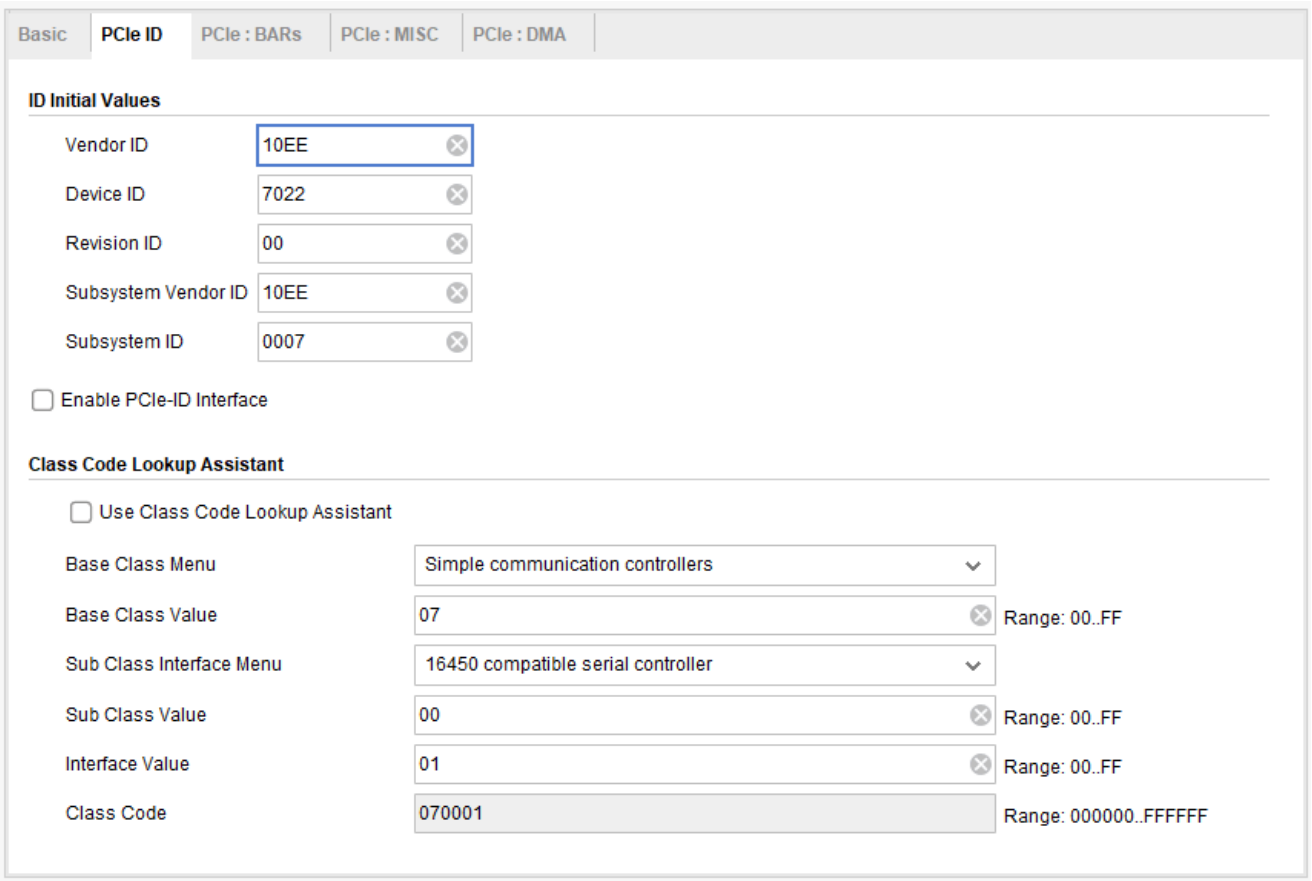
AXI4、AXI4-Stream，必须选择一个，用于数据传输
AXI4-Lite Master 可选，用于实现 PCIE BAR 地址到 AXI4-lite 寄存器地址的映射，可以用于读写用户逻辑寄存器。
AXI4-Lite Slave 可选，用来将 XDMA 内部寄存器开放给用户逻辑，用户逻辑可以通过此接口访问 XDMA 内部寄存器，不会映射到 BAR。
AXI4 Bypass 接口，可选，用来实现 PCIE 直通用户逻辑访问，可用于低延迟数据传输。

1.4XDMA IP 配置

Mode：配置模式，选择 BASE 配置
Lane Width：选择 PCIE 的通道数量对于 MA703FA 为 2 个通道，每个开发板支持通道数量不一样，通道数量越多通信速度越快，用户需要根据硬件的实际通达数量选择正确的通道数。
Max Link Speed：选择 5.0GT/s 即 PCIE2.0，对于 ultrascale 或者 ultrascale+的 FPGA 可以选择 PCIE3.0 实现更高速度
Reference Clock ： 100MHZ，参考时钟 100M
DMA Interface Option：接口选择 AXI4 接口
AXI Data Width：128bit，即 AXI4 数据总线宽度为 64bit
AXI Clock ： 125M，即 AXI4 接口时钟为 125MHZ
DMA Interface option 设置为 AXI Memory Mapped 方式



PCIE ID 配置，这里选择默认的配置就可以，默认的设备类型是 Simple communication controllers

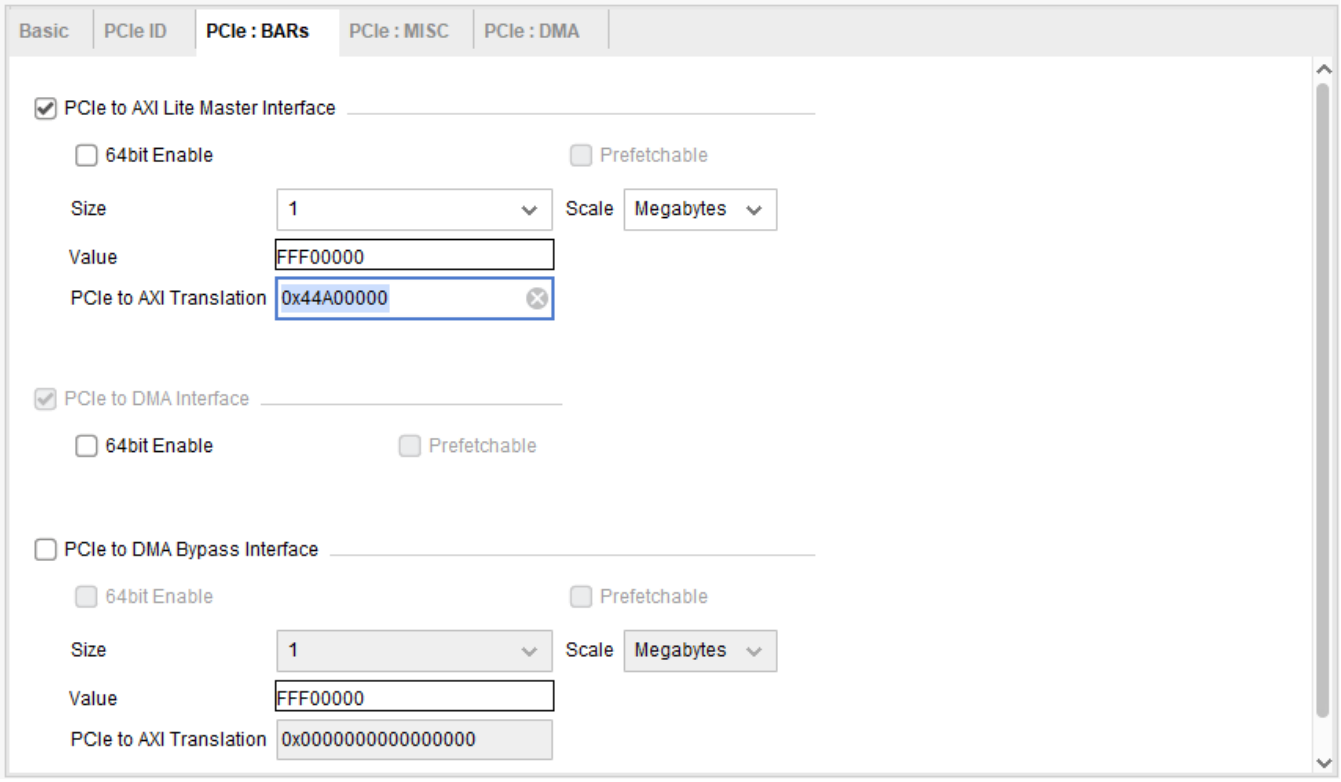


PCIE BAR 配置，这里面的配置比较重要, 首先使能 PCIE to AXI Lite Master Interface ，这样可以在主机一侧通过 PCIE 来访问用户逻辑侧寄存器或者其他 AXI4-Lite 总线设备映射空间选择 1M，当然用户也可以根据实际需要来自定义大小。

PCIE to AXI Translation：这个设置比较重要，通常情况下，主机侧PCIE BAR 地址与用户逻辑侧地址是不一样的， 这个设置就是进行BAR 地址到AXI 地址的转换，比如主机一侧 BAR 地址为0，IP 里面转换设置为 0x44A00000， 则主机访问 BAR 地址 0 转换到AXI LIte 总线地址就是0x44A00000

PCIE to DMA Interface ：选择64bit 使能

DMA Bypass 暂时不用



PCIE 中断设置

User Interrupts:用户中断，XDMA提供16条中断线给用户逻辑，这里面可以配置使用几条中断线。

Legacy Interrupt:XDMA 支持 Legacy 中断, 我们这么不选

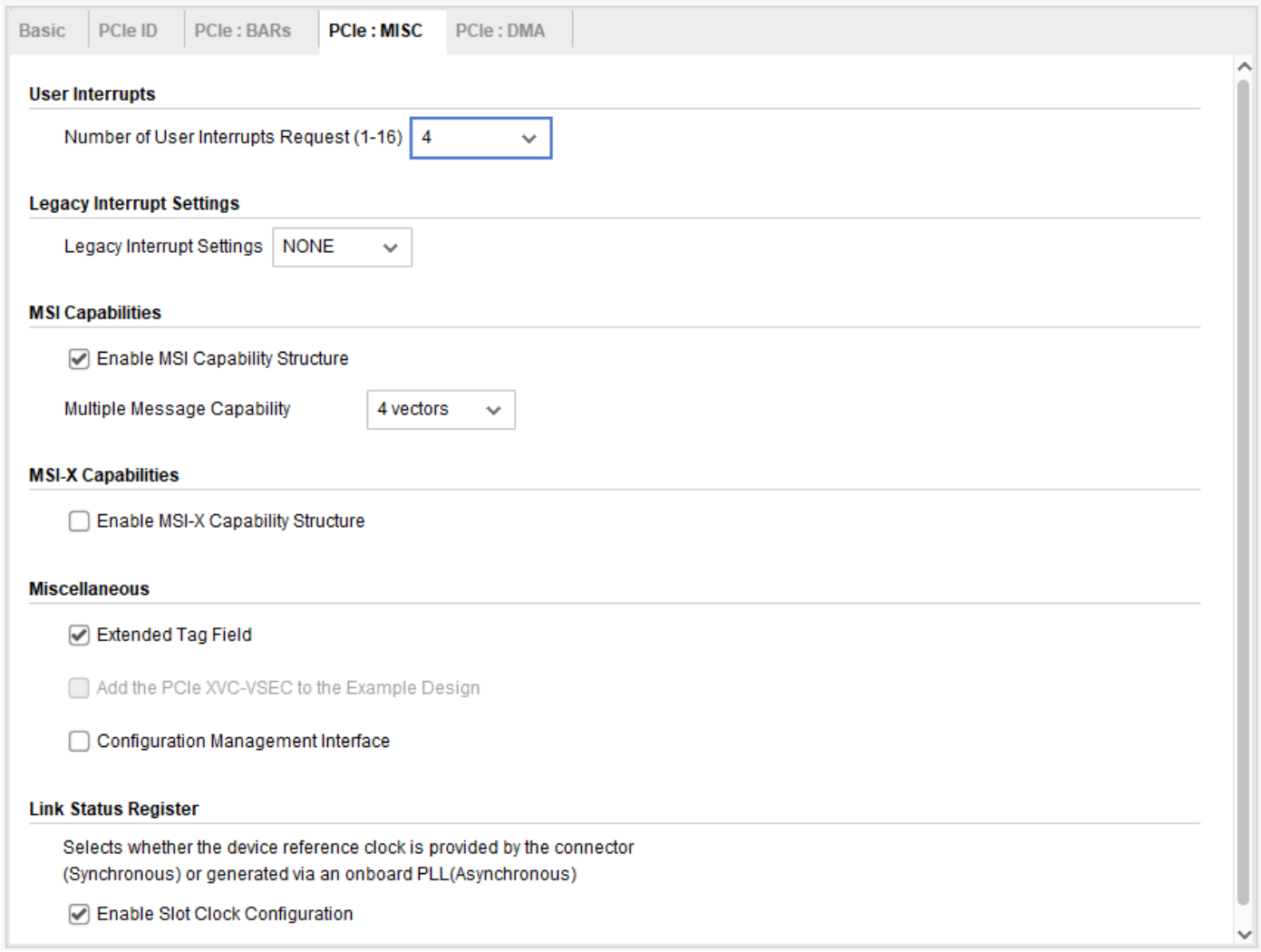
MSI Capabilities:选择支持MSI中断 ，支持4个中断消息向量

注意：MSI 中断和 MSI-X 中断只能选择一个，否则会报错，如果选择了 MSI 中断，则可以选择 Legacy 中断， 如果选择了 MSI-X 中断，那么 MSI 必须取消选择，同时 Legacy 也必须选择 None。此 IP 对于 7 系列设置有这没 个问题，如果使用 Ultrascale 系列，则可以全部选择

MSI-X Capabilities:不选

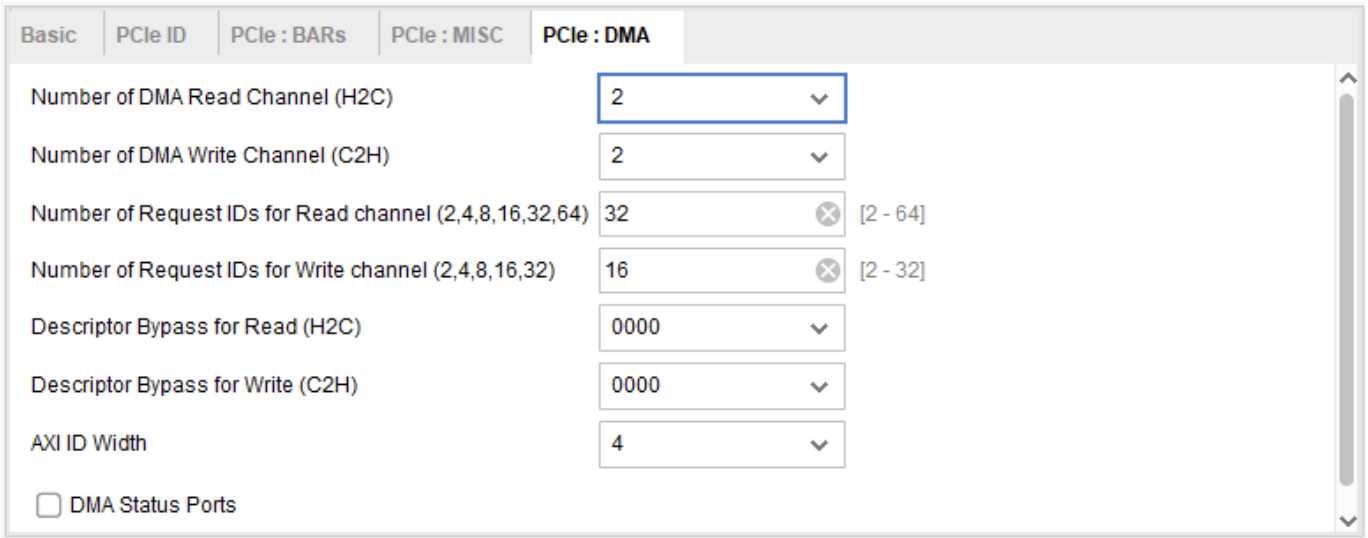
Miscellaneous:选 Extended Tag Field

Link Status Register:选 Enable Slot Clock Configuration

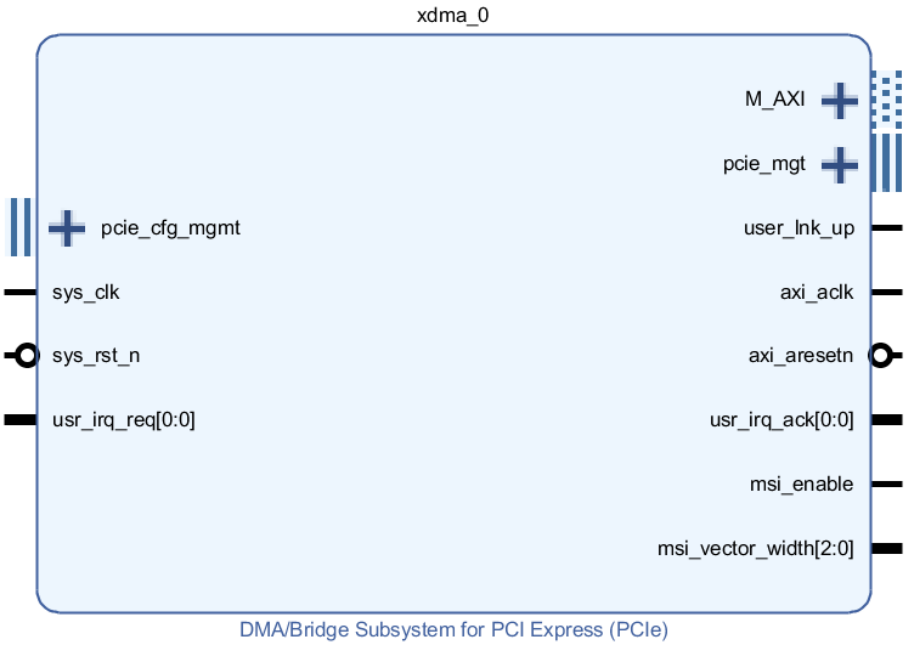


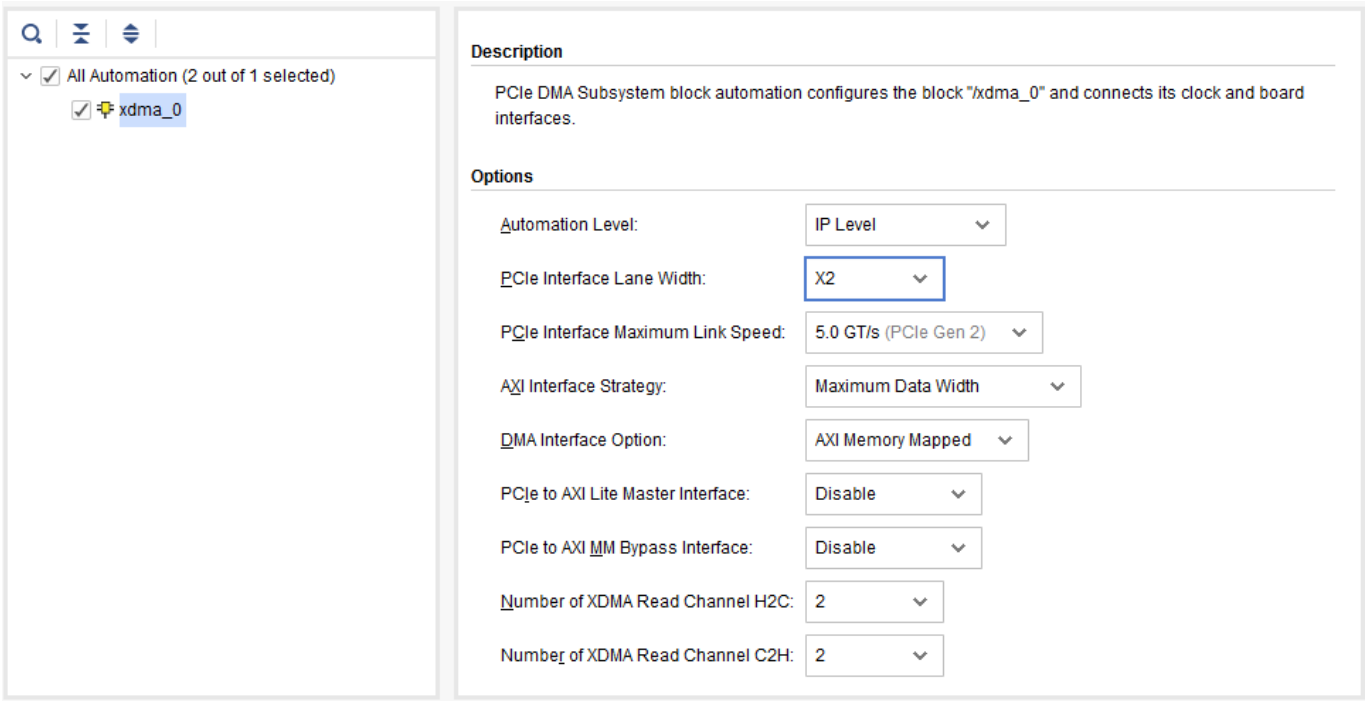
配置 DMA 相关内容

Number of DMA Read Channel（H2C）和 Number of DMA Write Channel（C2H）通道数，对于 PCIE2.0 来说最大只能选择 2，也就是 XDMA 可以提供最多两个独立的写通道和两个独立的读通道，独立的通道对于实际应用中有很大的作用，在带宽允许的前提前，一个 PCIE 可以实现多种不同的传输功能，并且互不影响。这里我们选择 2 Number of Request IDs for Read （Write）channel ：这个是每个通道设置允许最大的 outstanding 数量，按照默 认即可

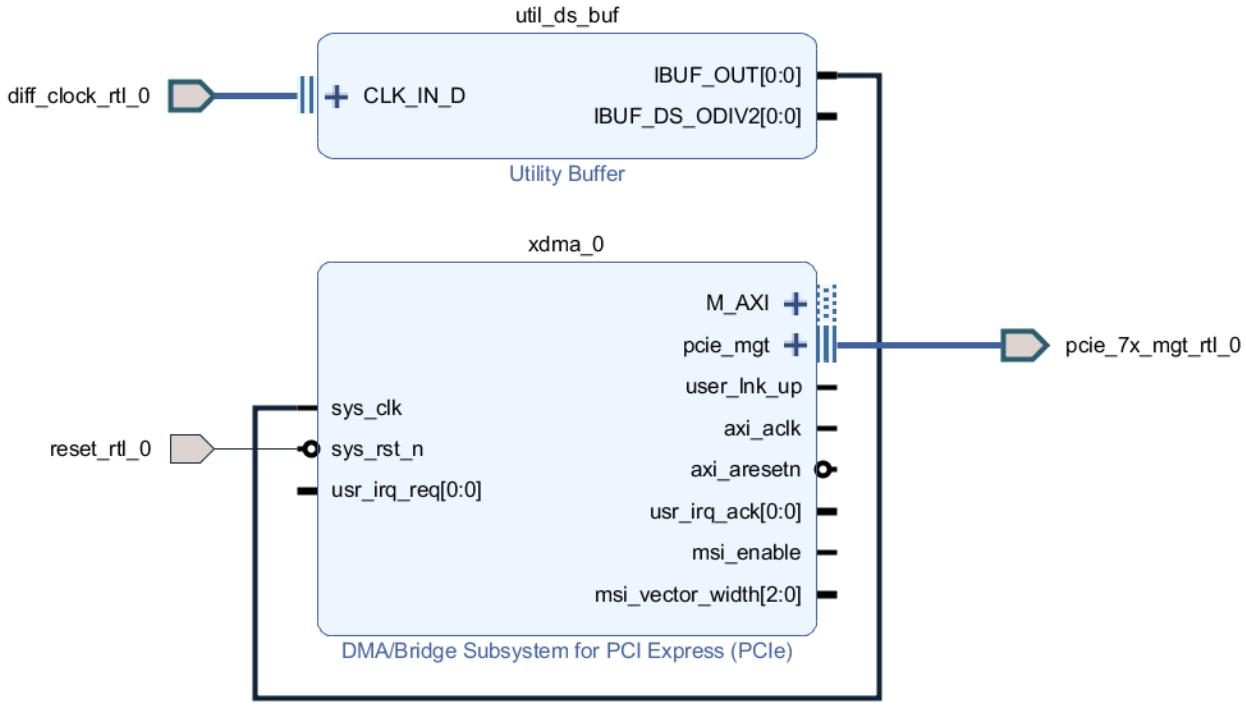


配置完成以后，点击 Run Block Auto，可以看到之前的配置信息，如果有发现和目标配置不一样的，需要手动修改，点击 OK，完成配置。



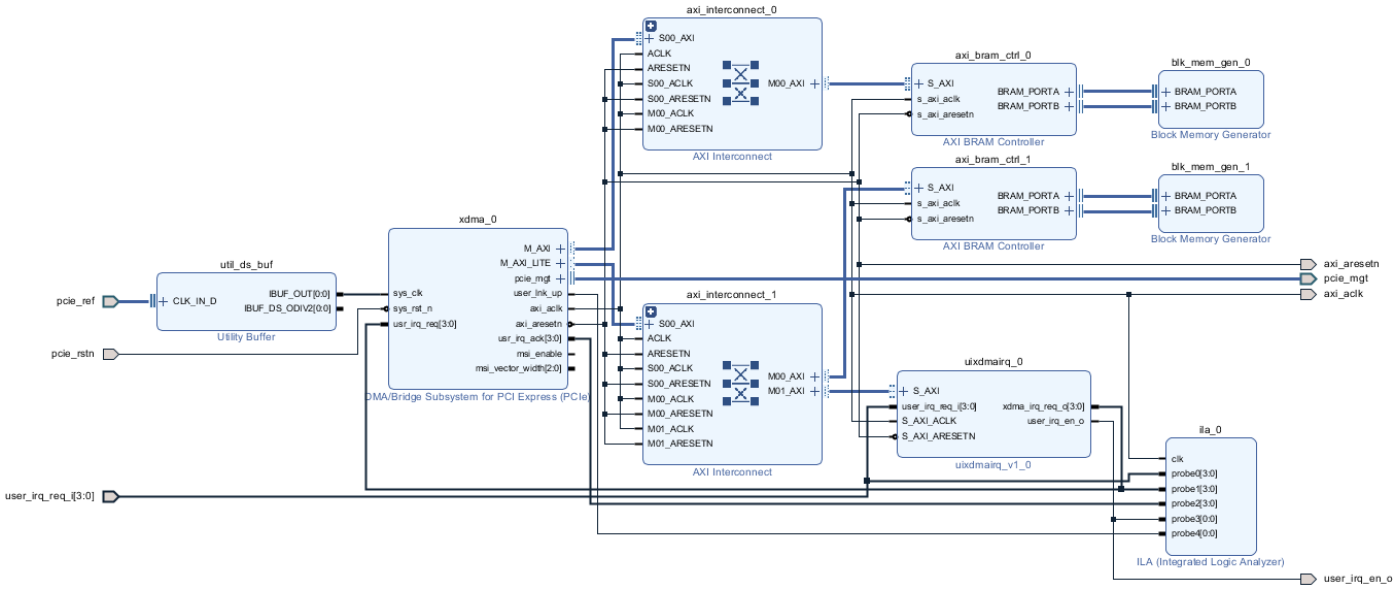


配置完成后，VIVADO 会自动进行必要的连线



到此为止，XDMA IP 配置就完成了。为了让 XDMA 和上位机可以密切配和工作，我们还要继续搭建其他部分的功能模块。

1.5 包含中断演示的 XDMA 工程



进行地址分配：

这里我们把挂在 M_AXI 上的 DDR(对于 MA703FA-35T 挂 BRAM)地址分配从 0 开始(对于 widnows 系统必须为 0), M_AXI 是需要进行 DMA 操作的。而 M_AXI_LITE 挂载的 BRAM 和 uixdmairq 中断控制单元是映射到用户 BAR 地址空间，这个地址就是前面我们 XDMA IP 里面设置的地址。默认情况下，需要设置 uixdmairq 中断控制单元的地址和 XDMA 里面设置的用户 BAR 地址空间一致。如下图 0x44A0_0000。BRAM 的地址空间 0x44A01_0000。关于地址空间的具体含义，结合软件的使用会更加清晰，初学者暂且根据教程设置。

Address Editor

Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
xdma_0					
M_AXI (64 address bits : 16E)					
axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0000_0000	32K	0x0000_0000_0000_7FFF
M_AXI_LITE (32 address bits : 4G)					
axi_bram_ctrl_1	S_AXI	Mem0	0x44A1_0000	8K	0x44A1_1FFF
uixdmairq_0	S_AXI	reg0	0x44A0_0000	64K	0x44A0_FFFF

1.6 FPGA 程序下载以及固化

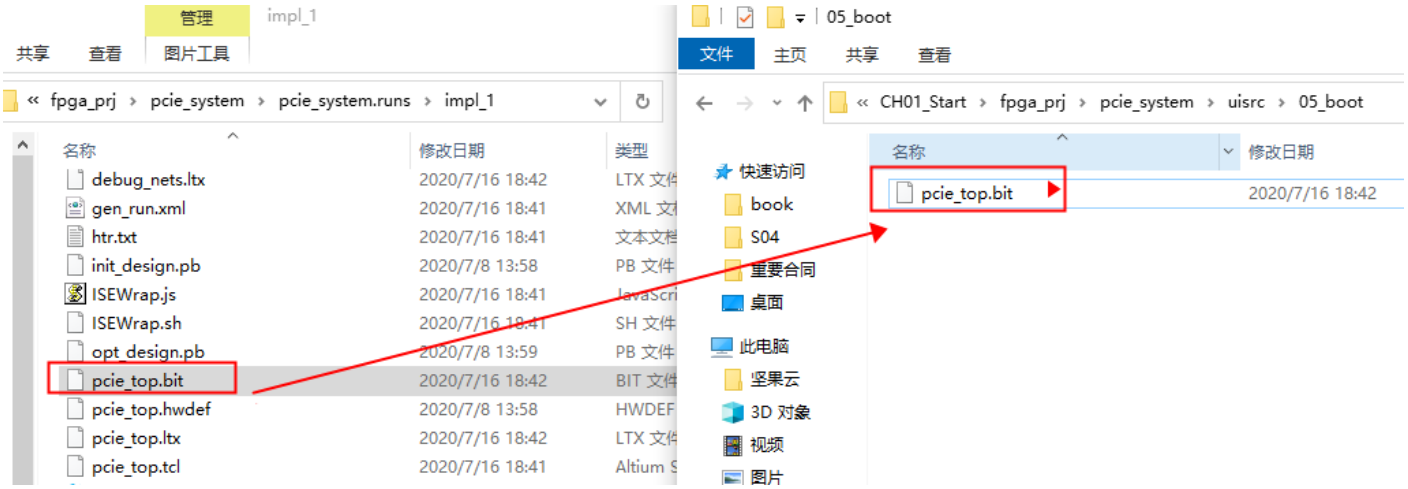
固化 FPGA 程序到 FLASH 在前面章节的课程中有详细介绍，这里再次简要介绍下。下载 bit 完成后重启电脑（有些电脑需要关机重启），对于 bit 文件确保开发板不掉电。下载 bit 很简单，下面对固化到 FLASH 步骤做一些说明。固化 MCS 或者 BIN 文件到 QSPI FLASH 后需要开发板断电重启。读者需要注意，目前的电脑一般开机速度很快，有可能在 FPGA 还没加载完成程序前就开机了，导致 PCIE 无法识别，如果有这种情况，建议禁止 WIN10 系统的快速开机模式，以及尝试设置 BISO 的延迟开机。

1、首先设置 xdc 文件，设置 bit 为压缩模式，以及 QSPI 的加载方式和速度，在 xdc pin 脚约束中增加如下代码后编译 bit。

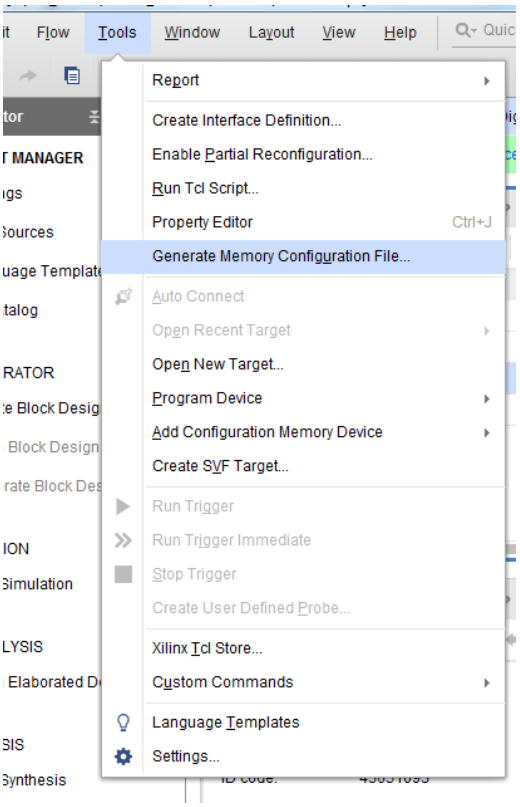
```
#bit compress spix4 speed up

set_property CFGBVS VCCO [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]

set_property BITSTREAM.GENERAL.COMPRESS true [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]

set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]

set_property BITSTREAM.CONFIG.SPI_FALL_EDGE Yes [current_design]
```

2、单击产生 bit

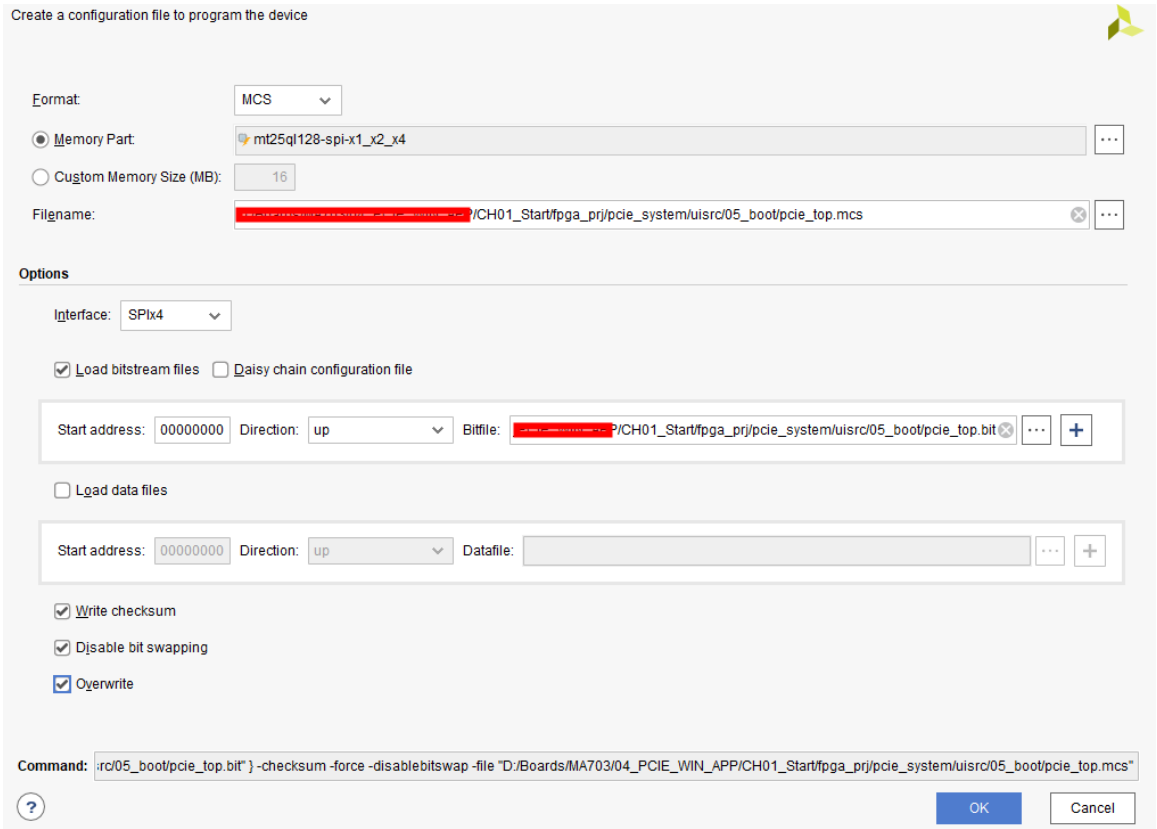
3、复制 pcie_top.bit 到路径 uisrc/05_boot 路径，方便我们管理程序



3、点击 tool，然后选择 Generate Memory Configuration File



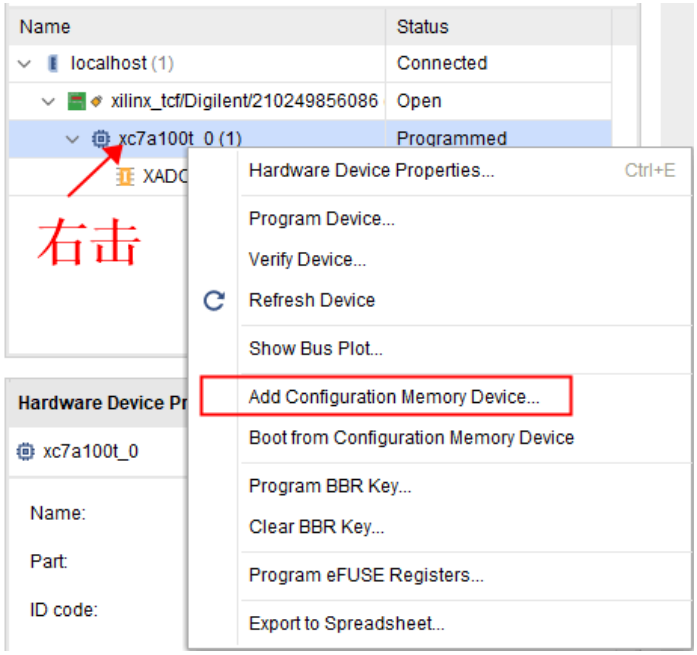
4、选择正确的 FLASH 型号，指定 MCS 的保存路径



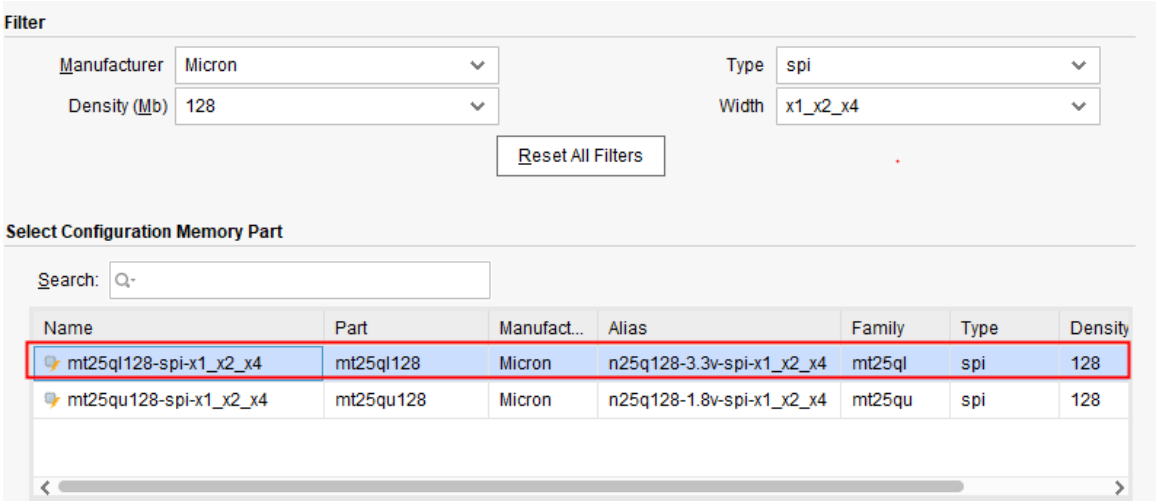
3、单击产生 bit 和 bin 文件

4、连接硬件，然后右击芯片型号，单击 Add Configuration Memory Device

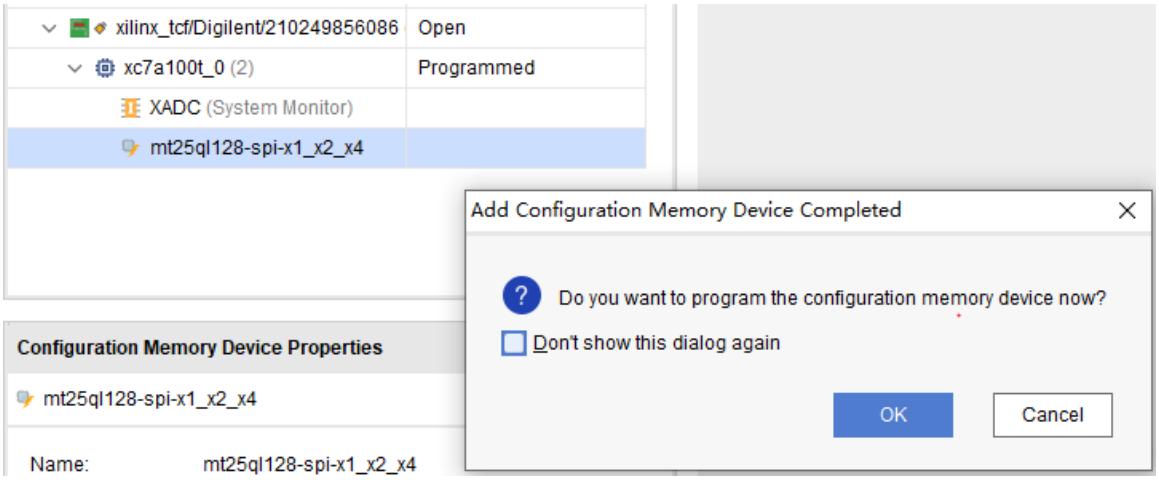
以下演示用的截图是 100T 的芯片，实际你的板卡可能是其他芯片型号。



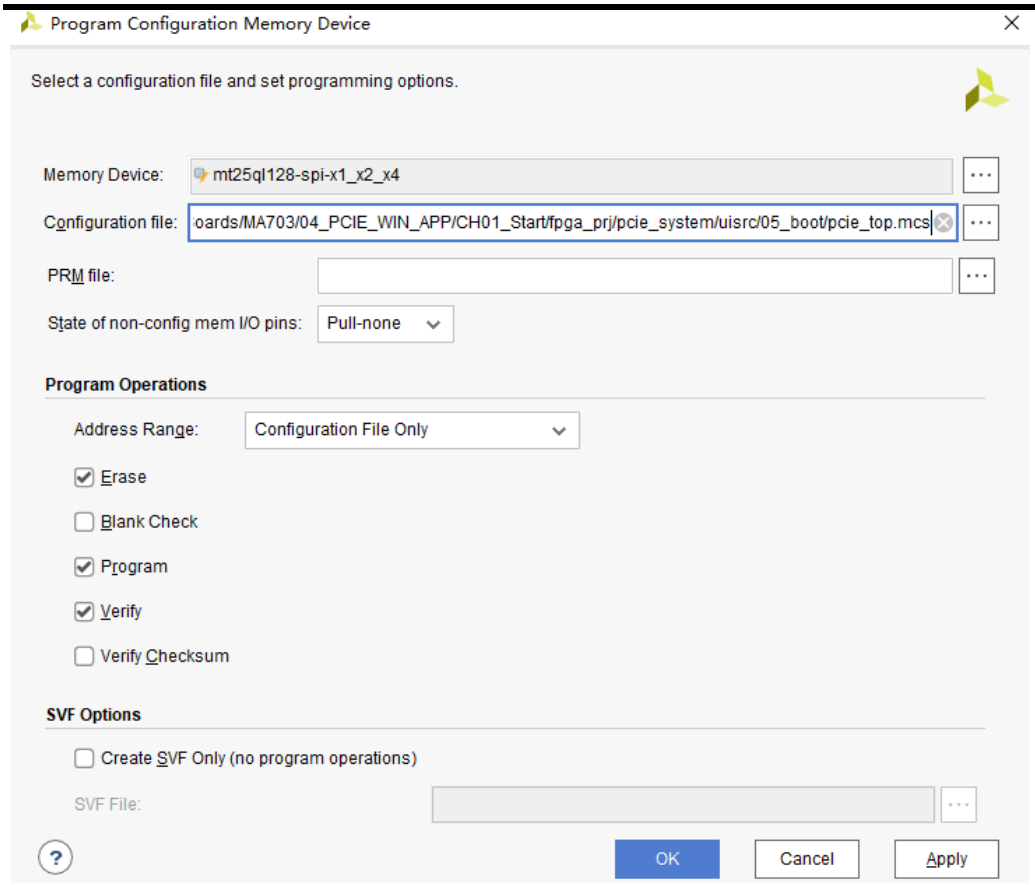
7、选择正确的 FLASH 型号，不同的开发板型号可能不一致，用户需要注意。



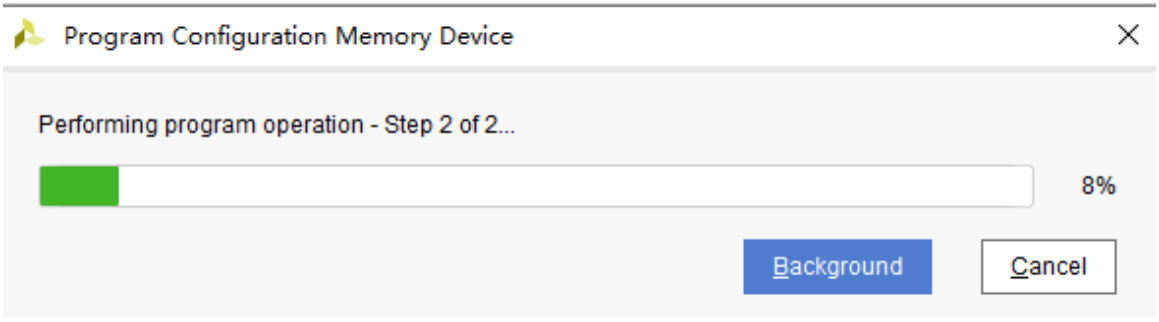
8、单击 OK



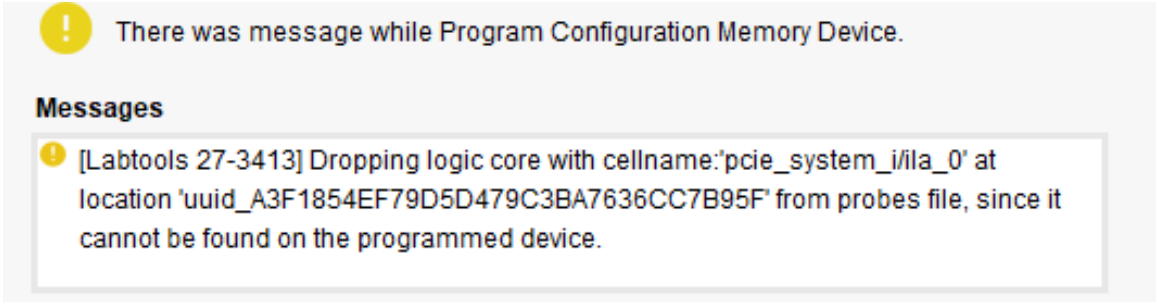
9、选择刚才我们保存到 05_boot 路径中的 mcs 文件，单击 OK 烧录。



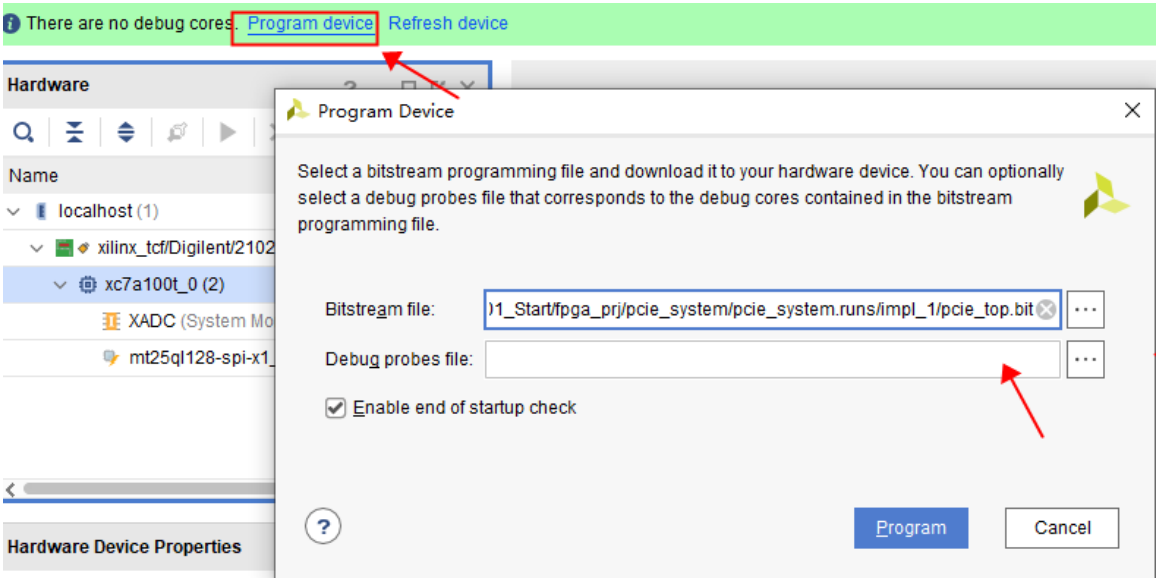
10、等待烧录完成既可以



由于添加了 ila 在线逻辑分析 IP CORE 而 MCS 文件是固化到 FLASH 的没有这个功能，所以如果正在调试后下载固化程序到 FLASH 会有一条 waring，导致下载失败。



点击 program device 并且把 debug probes file 设置为空，重新下载 FLASH 就可以了。



1.7 WIN7/WIN10 系统下开发环境搭建

请用户在我们提供的网盘里面下载VS2015，WINSDK，WDK 开发包，这是经过我们验证的，如果SDK 和WDK 版本不匹配会造成编译失败。

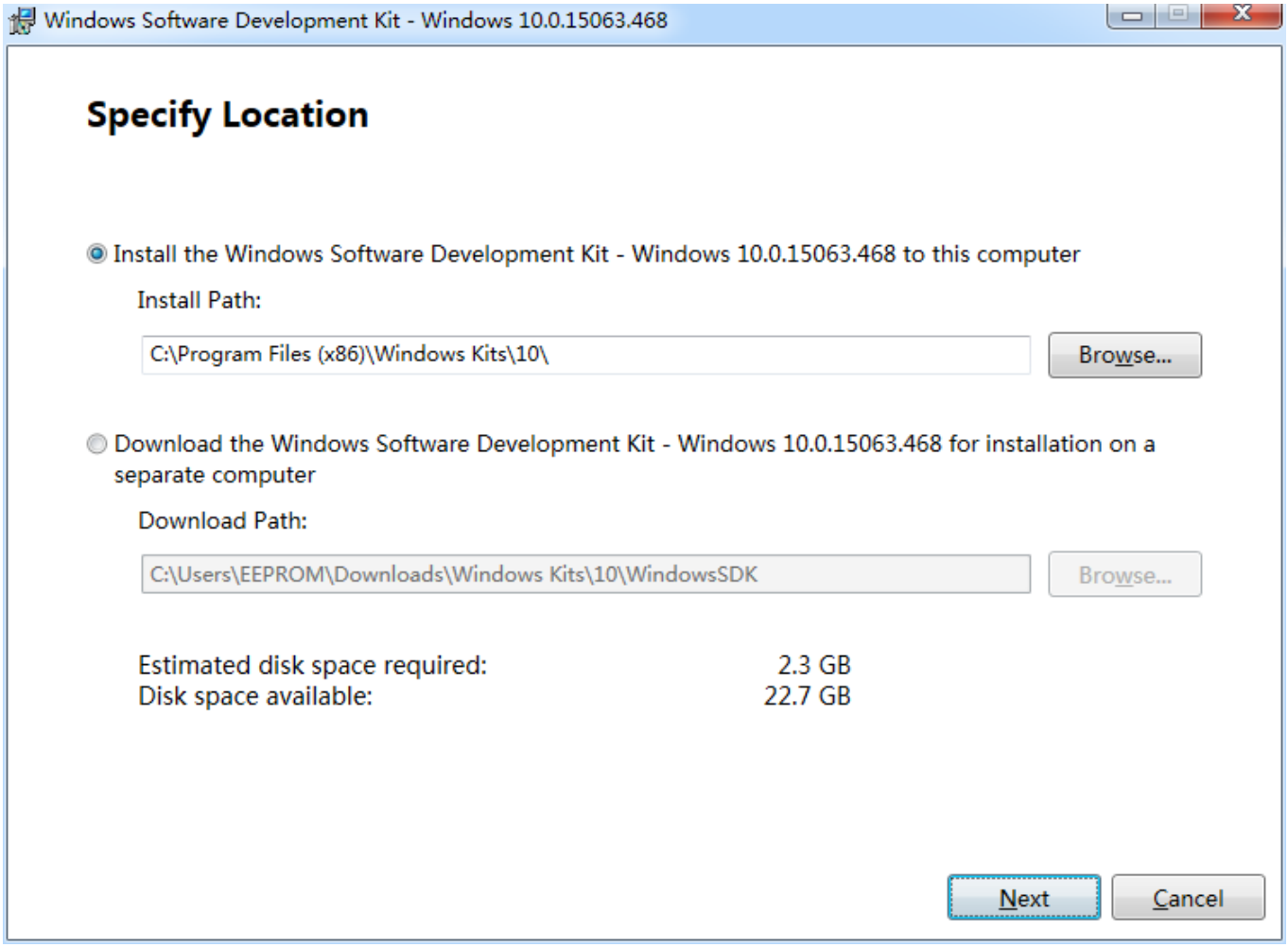
首先是VS2015 的安装，如果已经安装了VS2010，或者其他版本，建议先进行卸载，卸载过程不要使用360等软件工具卸载，而是使用VS 自带的卸载工具（注意：操作过程中可能会出现问题，导致该系统无论如何都无法编译程序，本教程不对此情况负责，我们测试宿主机是一台刚刚安装了WIN7/WIN10 系统的机器，为了避免因搭建环境而造成宿主机原系统算坏的情况，建议用户可以花几十块钱购买一块320G 以下的硬盘，在新硬盘安装系统搭建环境测试）

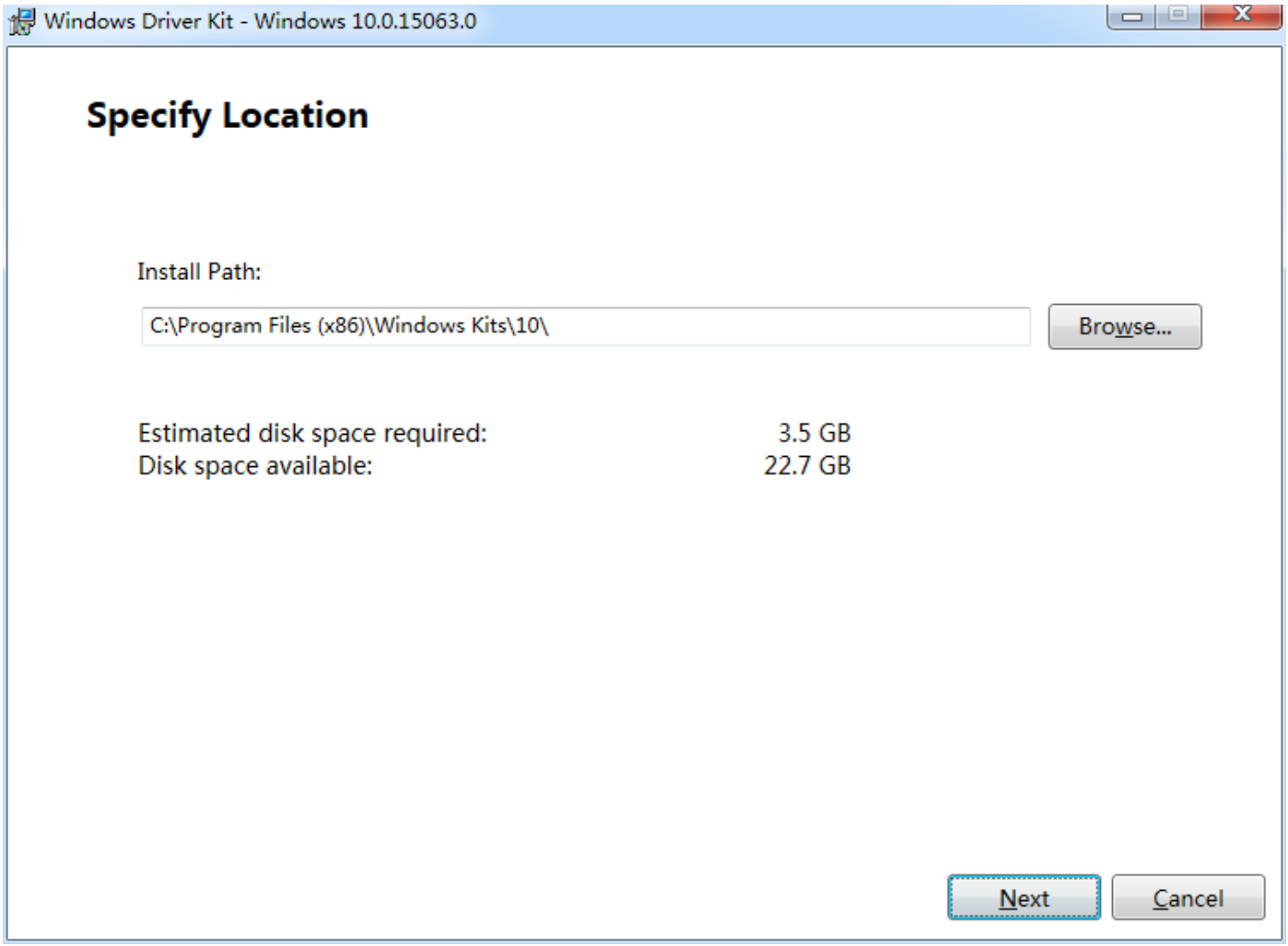
大家请注意软件的安装顺序一定要是先安装 vs2015 再安装 WINSDK，最后安装 WDK。如果一不小心顺序错了，卸载干净后重新安装，否则肯定不会成功的。

这里一定要选择自定义，默认情况下是不安装 VC++语言的，不知道 VS 是怎么搞的。

然后在编程语言里面勾选vc++，vs2015 更新3 也勾选上，否则也可能造成编译错误，这些错误主要是由于版本匹配问题引起的。下面就是一路安装，不多说了。安装好以后，记得去百度搜个序列号，你们懂得。

下面安装 WINSDK。


Windows Software Development Kit - Windows 10.0.15063.468
Specify Location
Install the Windows Software Development Kit - Windows 10.0.15063.468 to this computer
Install Path:
C:\Program Files (x86)\Windows Kits\10\
Browse...
Download the Windows Software Development Kit - Windows 10.0.15063.468 for installation on a separate computer
Download Path:
C:\Users\EEPROM\Downloads\Windows Kits\10\WindowsSDK
Browse...
Estimated disk space required: 2.3 GB
Disk space available: 22.7 GB
Next
Cancel



接下来还有一项重要的设置，根据官方文档的说法，XDMA 的驱动没有提供一个验证过的证书，所以必须让系统进入测试模式才能安装驱动。

使用如下命令可以开关测试模式。

The TESTSIGNING boot configuration option is enabled or disabled through the BCDEdit command. To enable test-signing, use the following BCDEdit command:

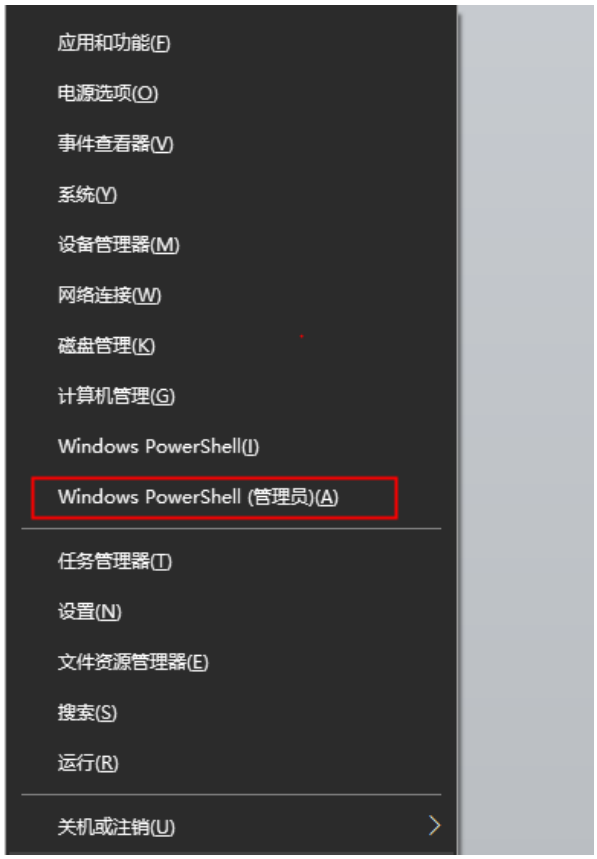
```
Bcdedit.exe -set TESTSIGNING ON
```

To disable test-signing, use the following BCDEdit command:

```
Bcdedit.exe -set TESTSIGNING OFF
```

Note After you change the TESTSIGNING boot configuration option, restart the computer for the change to take effect.

bcdedit /set testsigning on 打开测试模式
bcdedit /set testsigning off 关闭测试模式
在WIN7/WIN10系统下打开终端，一定要使用管理员权限
设置完成后需要重启系统。



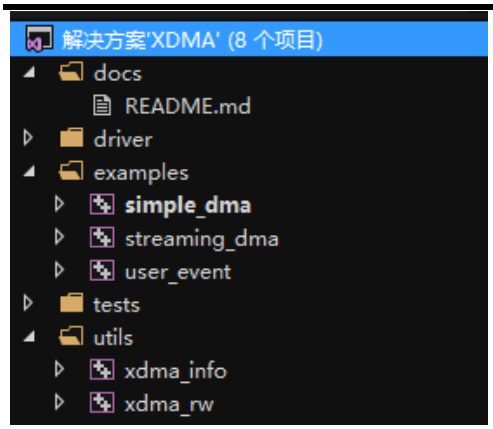
操作成功后，电脑桌面右下角可以看到，系统进入测试模式的提示信息。

那么有些客户问，驱动没有签名，能否让驱动有签名呢？这个当然可以花钱向微软公司购买认证签名既可。

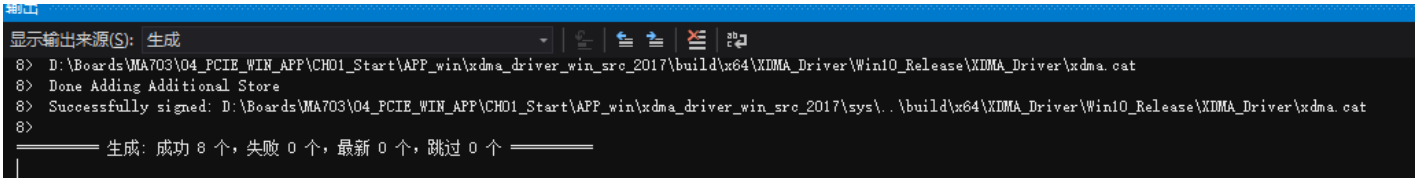
1.8 驱动程序编译以及安装

1.8.1 驱动编译

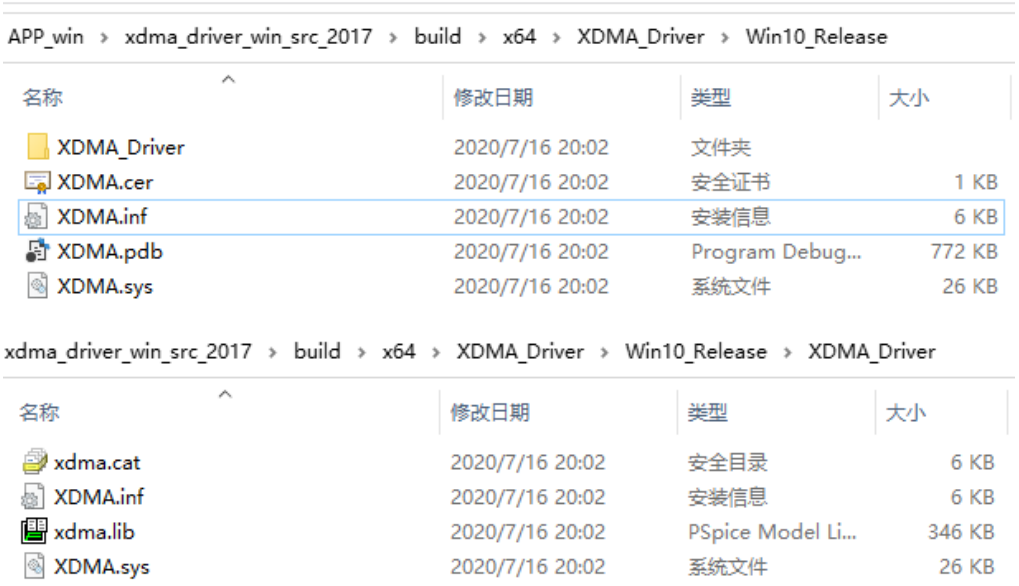
驱动默认已经编译好，可以直接使用，当然用户也可以体验下如何编译驱动。Xilinx 提供的 XDMA 驱动源码，直接编译可能会由于 WPP 的原因导致编译失败，WPP 是驱动调试功能，用户可以自行修改，本教程中不使用 WPP 功能，同时对驱动源码进行了一些修改，可以直接使用，请参考我们提供的工程。工程文件如下：



driver 里面是驱动源码，其他项目都是应用程序。拿到工程以后，先进行清理，然后进行编译。

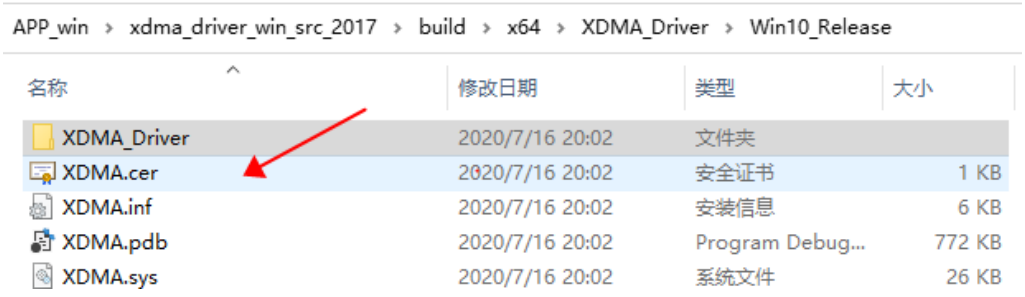


编译成功后，可以看到

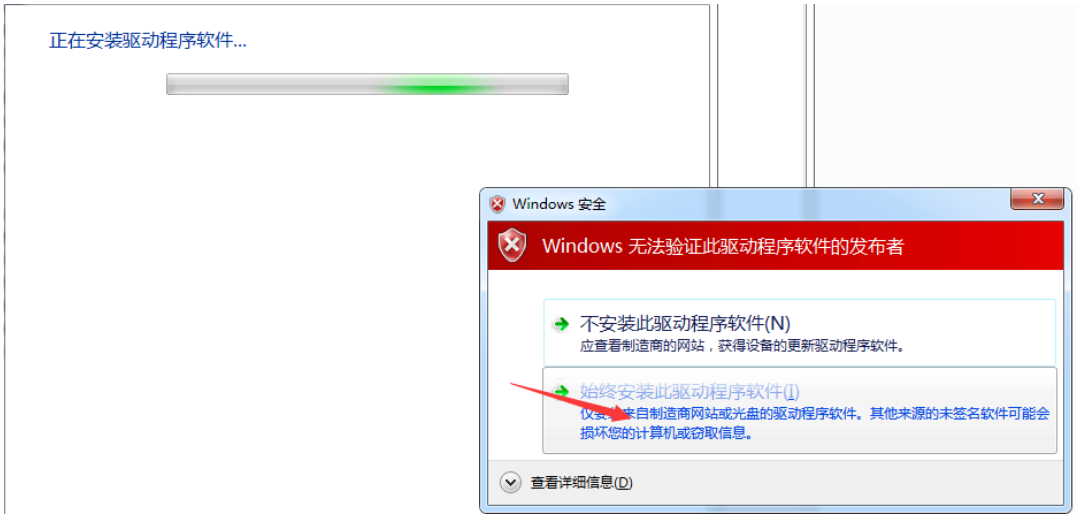


1.8.2 驱动安装

右击安装证书



右击安装驱动



1.9 应用程序测试

1.9.1 简单读写测试

我们打开一个终端（如果双击运行会很快退出来），进入到上一节编译生成的应用程序目录找到 xdma_rw.exe，这个应用程序是操作 pcie 的所有设备的，我们在终端只输入 xdma_rw.exe，可以看到提示信息，告诉用户这个程序如何使用。

```
D:\Boards\M[redacted]\04_example_PCIE\CH01_Start\APP_win\xdma_driver_win_src_2017\build\x64\bin>xdma_rw.exe
xdma_rw.exe usage:

xdma_rw.exe <DEVNODE> <read|write> <ADDR> [OPTIONS] [DATA]
- DEVNODE : One of: control | user | event_* | h2c_* | c2h_*,
            where the * is a numeric wildcard (0-15 for events, 0-3 for h2c and c2h).
- ADDR :    The target offset address of the read/write operation.
            Can be in hex or decimal.
- OPTIONS :
            -a set alignment requirement for host-side buffer (default: PAGE_SIZE)
            -b open file as binary
            -f use contents of file as input or write output into file.
            -l length of data to read/write (default: 4 bytes or whole file if '-f' flag is used)
            -v more verbose output
- DATA :    Space separated bytes (big endian) in decimal or hex,
            e.g.: 17 34 51 68
            or:   0x11 0x22 0x33 0x44
```

可以看到在当前目录有一个 datafile4k.bin 文件，那就测试一下将这个文件传输到 FPGA 的 DDR 或者(MA703FA-35T 是 BRAM)，然后读出来。首先在终端输入指令：xdma_rw.exe h2c_0 write 0x0000000 -b -f datafile4K.bin -l 4096 意思就是使用 h2c_0 设备以二进制的形式读取文件 datafile4k.bin 写入到 BRAM 内存地址 0x0000000 长度为 4096 字节。

```
D:\Boards\M[redacted]\04_example_PCIE\CH01_Start\APP_win\xdma_driver_win_src_2017\build\x64\bin>xdma_rw.exe h2c_0 write 0x0000
000 -b -f datafile4K.bin -l 4096
4096 bytes read from file datafile4K.bin
4096 bytes written in 0.000094s
```

接下来再读回来，使用命令 xdma_rw.exe c2h_0 read 0x0000000 -b -f datafile4K_recv.bin -l 4096

```
D:\Boards\M[redacted]\04_example_PCIE\CH01_Start\APP_win\xdma_driver_win_src_2017\build\x64\bin>xdma_rw.exe c2h_0 read 0x00000
00 -b -f datafile4K_recv.bin -l 4096
4096 bytes received in 0.000071s
```

接下来我们可以使用 winhex 等软件来检查一下两个文件数据是否一直，经过检查，是一致的则说明传输功能正常。

1.9.2 中断 FPGA 程序

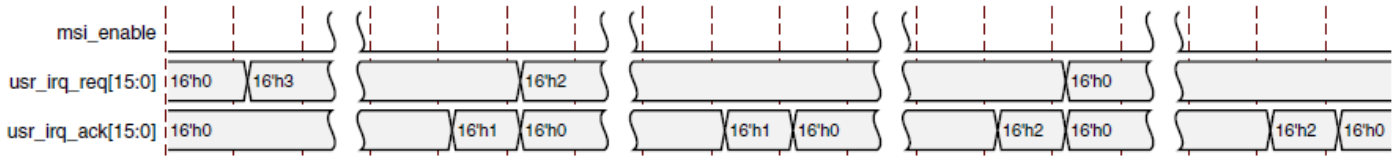
首先我们要理解下 XDMA 的中断类型，以及控制时序：

1)、Legacy Interrupts：

对于 Legacy Interrupts 中断，当 user_irq_ack 第一次为 1 的时候 usr_irq_req 可以清 0，当 user_irq_ack 第二次为 1 的时候，可以重新设置 usr_irq_req 发起中断。

在 PCI 总线里面 INTx 中断是由四条可选的中断线决定的，这种中断方式是共享式的，所有的 pci 设备把中断信号在一条中短线上相与，再上报给 cpu，cpu 收到中断以后再查询具体是哪个设备产生了中断。

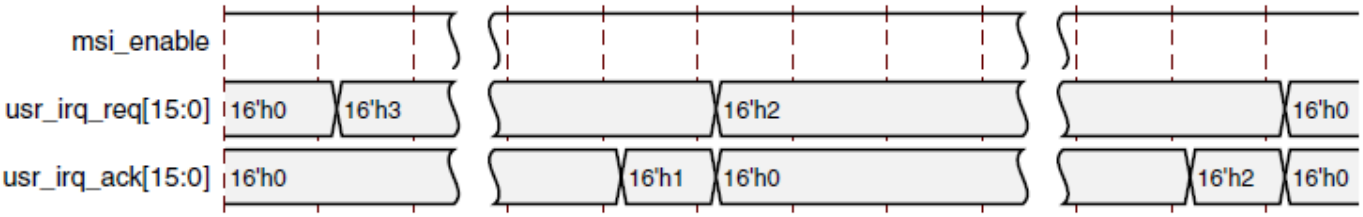
在 PCIE 总线里面已经没有了实体的 INTx 物理中断线了，PCIE 标准使用专门的 Message 事务包来实现 INTx 中断，这是为了兼容以前的 PCI 软件。INTx 是共享式的，cpu 相应中断后还需要查询具体中断源，效率比较低



2)、MSI Interrupts：

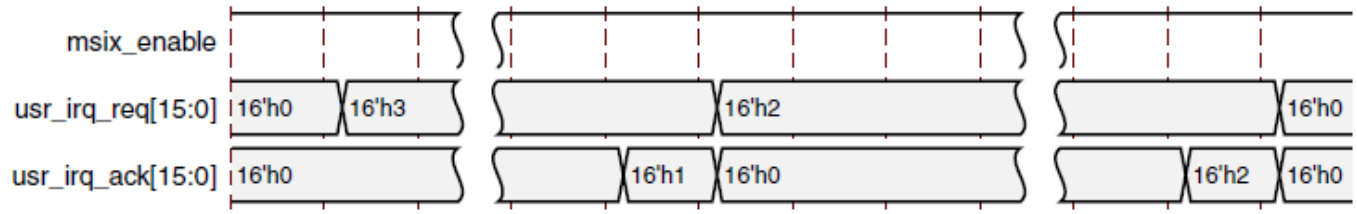
MSI 发出 usr_irq_req 中断请求后，user_irq_ack 为 1 只是说明中断已经被主机接收了，但是不代表已经处理，软件或者驱动层可以去清零 usr_irq_req。

MSI 中断和 MSI-X 都是往配置的 CPU 中断寄存器里进行 memory 写操作，来产生中断，效率比 INTx 是共享式高，其中 MSI 最多支持 32 个中断向量，而 MSI-X 最多支持 2048 个中断向量。

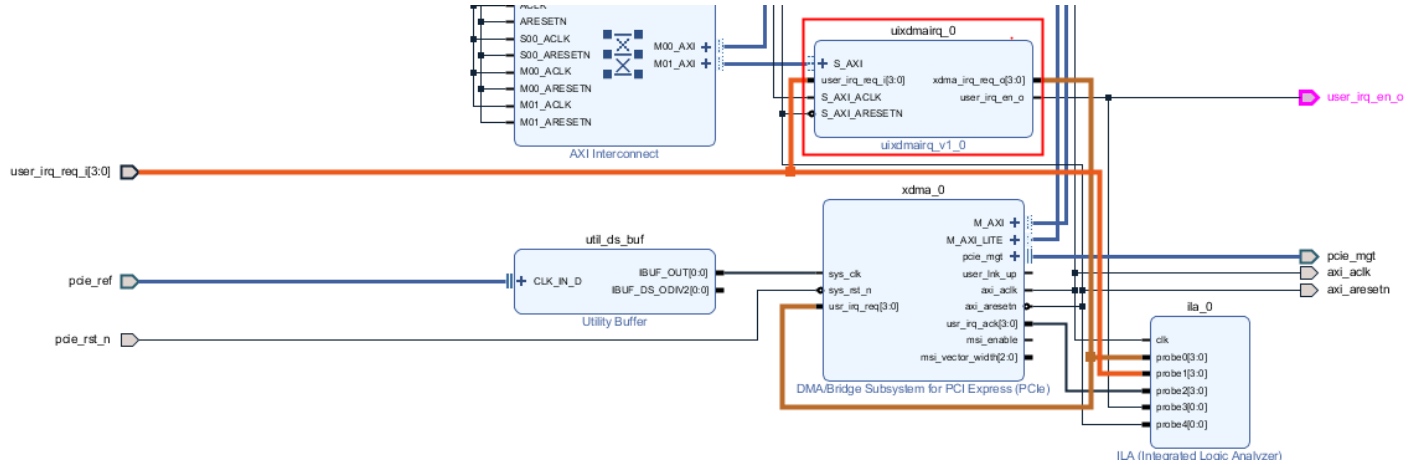


3)、MSI-X Interrupts：

当 usr_irq_req 中断请求后，只要 user_irq_ack 为 1 就可以清零 usr_irq_req，但是没说明扫码时候可以置 1，重启下次中断。



经过以上所有中断方式测试，发目前使用 Legacy 和 MSI 已经够用，而且相对稳定，上位机驱动通过访问用户 bar 地址空间和米联客编写的 Uixdmairq ip-core 一起管理接收的中断。



Uixdmairq.v 源码

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand:  米联客(msxbo)
Technical forum：uisrc.com
```

```
taobao: osrc.taobao.com
Create Date: 2019/05/20
Module Name: uixdmairq
Description:
XDMA interrupt control
Copyright: Copyright (c) msxbo
Revision: 1.0
Signal description：
1) _i input
2) _o output
3) _n activ low
4) _dg debug signal
5) _r delay or register
6) _s state mechine
*/
//////////////////////////////////////////////////////////////////////////////////

    module uixdmairq #
    (
        parameter integer XMDA_REQ_NUM = 8
    )
    (
        // Users to add ports here
        input    wire [XMDA_REQ_NUM-1 :0] user_irq_req_i,
        //input    wire [XMDA_REQ_NUM-1 :0] xdma_irq_ack_i,
        output wire [XMDA_REQ_NUM-1 :0] xdma_irq_req_o,
        output wire user_irq_en_o,
        input wire    S_AXI_ACLK,
        input wire    S_AXI_ARESETN,
        input wire [3 : 0] S_AXI_AWADDR,
        input wire [2 : 0] S_AXI_AWPROT,
        input wire    S_AXI_AWVALID,
        output wire    S_AXI_AWREADY,
        input wire [31 : 0] S_AXI_WDATA,
        input wire [3    : 0] S_AXI_WSTRB,
        input wire    S_AXI_WVALID,
        output wire    S_AXI_WREADY,
        output wire [1 : 0] S_AXI_BRESP,
        output wire    S_AXI_BVALID,
        input wire    S_AXI_BREADY,
        input wire [3 : 0] S_AXI_ARADDR,
        input wire [2 : 0] S_AXI_ARPROT,
        input wire    S_AXI_ARVALID,
        output wire    S_AXI_ARREADY,
        output wire [31 : 0] S_AXI_RDATA,
        output wire [1    : 0] S_AXI_RRESP,
        output wire    S_AXI_RVALID,
        input wire    S_AXI_RREADY
```

```
    );



reg   [XMDA_REQ_NUM -1 :0] user_irq_req;
reg   [XMDA_REQ_NUM -1 :0] user_irq_req_r1;
reg   [XMDA_REQ_NUM -1 :0] user_irq_req_r2;
reg   [XMDA_REQ_NUM -1 :0] user_irq_req_r3;
reg   [XMDA_REQ_NUM -1 :0] xdma_irq_ack_r1;
reg   [XMDA_REQ_NUM -1 :0] xdma_irq_ack_r2;
reg   [XMDA_REQ_NUM -1 :0] xdma_irq_ack_r3;
//reg   [XMDA_REQ_NUM -1 :0] xdma_irq_ack;
reg   [XMDA_REQ_NUM -1 :0] xdma_irq_req;
    // AXI4LITE signals
    reg [3:0]           axi_awaddr;
    reg          axi_awready;
    reg          axi_wready;
    reg [1 : 0]       axi_bresp;
    reg          axi_bvalid;
    reg [3:0]         axi_araddr;
    reg          axi_arready;
    reg [31 : 0]      axi_rdata;
    reg [1    : 0]    axi_rresp;
    reg          axi_rvalid;

    // Example-specific design signals
    // local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
    // ADDR_LSB is used for addressing 32/64 bit registers/memories
    // ADDR_LSB = 2 for 32 bits (n downto 2)
    // ADDR_LSB = 3 for 64 bits (n downto 3)
    localparam integer ADDR_LSB = 2;
    localparam integer OPT_MEM_ADDR_BITS = 1;
    //------------------------------------------------
    //-- Signals for user logic register space example
    //------------------------------------------------
    //-- Number of Slave Registers 4
    reg [31:0]slv_reg0;
    reg [31:0]slv_reg1;
    wire   slv_reg_rden;
    wire   slv_reg_wren;
    reg [31:0]  reg_data_out;
    integer      byte_index;
    reg     aw_en;

    // I/O Connections assignments

    assign S_AXI_AWREADY    = axi_awready;
    assign S_AXI_WREADY     = axi_wready;
```

```
    assign S_AXI_BRESP   = axi_bresp;
    assign S_AXI_BVALID = axi_bvalid;
    assign S_AXI_ARREADY = axi_arready;
    assign S_AXI_RDATA   = axi_rdata;
    assign S_AXI_RRESP   = axi_rresp;
    assign S_AXI_RVALID = axi_rvalid;
    // Implement axi_awready generation
    // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
    // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
    // de-asserted when reset is low.

    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_awready <= 1'b0;
          aw_en <= 1'b1;
        end
      else
        begin
          if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
            begin
              // slave is ready to accept write address when
              // there is a valid write address and write data
              // on the write address and data bus. This design
              // expects no outstanding transactions.
              axi_awready <= 1'b1;
              aw_en <= 1'b0;
            end
            else if (S_AXI_BREADY && axi_bvalid)
                begin
                  aw_en <= 1'b1;
                  axi_awready <= 1'b0;
                end
          else
            begin
              axi_awready <= 1'b0;
            end
        end
    end

    // Implement axi_awaddr latching
    // This process is used to latch the address when both
    // S_AXI_AWVALID and S_AXI_WVALID are valid.

    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
```

```
            begin
               axi_awaddr <= 0;
            end
         else
            begin
               if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
                  begin
                     // Write Address latching
                     axi_awaddr <= S_AXI_AWADDR;
                  end
            end
      end

      // Implement axi_wready generation
      // axi_wready is asserted for one S_AXI_ACLK clock cycle when both
      // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
      // de-asserted when reset is low.

      always @( posedge S_AXI_ACLK )
      begin
         if ( S_AXI_ARESETN == 1'b0 )
            begin
               axi_wready <= 1'b0;
            end
         else
            begin
               if (~axi_wready && S_AXI_WVALID && S_AXI_AWVALID && aw_en )
                  begin
                     // slave is ready to accept write data when
                     // there is a valid write address and write data
                     // on the write address and data bus. This design
                     // expects no outstanding transactions.
                     axi_wready <= 1'b1;
                  end
               else
                  begin
                     axi_wready <= 1'b0;
                  end
            end
      end

assign slv_reg_wren = axi_wready && S_AXI_WVALID && axi_awready && S_AXI_AWVALID;

      always @( posedge S_AXI_ACLK )
      begin
         if ( S_AXI_ARESETN == 1'b0 )begin
               slv_reg0 <= 0;
         end
```

```
    else if(slv_reg_wren)begin
            if (axi_awaddr[3:2] == 2'd0)
              slv_reg0[31:0] <= S_AXI_WDATA[31:0];
            else if (axi_awaddr[3:2] == 2'd1)
              slv_reg1[31:0] <= S_AXI_WDATA[31:0];
    end else begin
              slv_reg0 <= 0;
              slv_reg1 <= slv_reg1;
    end
end

// Implement write response logic generation
// The write response and response valid signals are asserted by the slave
// when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
// This marks the acceptance of address and indicates the status of
// write transaction.

always @( posedge S_AXI_ACLK )
begin
   if ( S_AXI_ARESETN == 1'b0 )
      begin
         axi_bvalid  <= 0;
         axi_bresp   <= 2'b0;
      end
   else
      begin
         if (axi_awready && S_AXI_AWVALID && ~axi_bvalid && axi_wready && S_AXI_WVALID)
            begin
               // indicates a valid write response is available
               axi_bvalid <= 1'b1;
               axi_bresp  <= 2'b0; // 'OKAY' response
            end                                 // work error responses in future
         else
            begin
               if (S_AXI_BREADY && axi_bvalid)
                  //check if bready is asserted while bvalid is high)
                  //(there is a possibility that bready is always asserted high)
                  begin
                     axi_bvalid <= 1'b0;
                  end
            end
      end
end

// Implement axi_arready generation
// axi_arready is asserted for one S_AXI_ACLK clock cycle when
// S_AXI_ARVALID is asserted. axi_awready is
// de-asserted when reset (active low) is asserted.
```

```
    // The read address is also latched when S_AXI_ARVALID is
    // asserted. axi_araddr is reset to zero on reset assertion.

    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_arready <= 1'b0;
          axi_araddr  <= 32'b0;
        end
      else
        begin
          if (~axi_arready && S_AXI_ARVALID)
            begin
              // indicates that the slave has acceped the valid read address
              axi_arready <= 1'b1;
              // Read address latching
              axi_araddr  <= S_AXI_ARADDR;
            end
          else
            begin
              axi_arready <= 1'b0;
            end
        end
    end

    // Implement axi_arvalid generation
    // axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
    // S_AXI_ARVALID and axi_arready are asserted. The slave registers
    // data are available on the axi_rdata bus at this instance. The
    // assertion of axi_rvalid marks the validity of read data on the
    // bus and axi_rresp indicates the status of read transaction.axi_rvalid
    // is deasserted on reset (active low). axi_rresp and axi_rdata are
    // cleared to zero on reset (active low).
    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_rvalid <= 0;
          axi_rresp  <= 0;
        end
      else
        begin
          if (axi_arready && S_AXI_ARVALID && ~axi_rvalid)
            begin
              // Valid read data is available at the read data bus
              axi_rvalid <= 1'b1;
              axi_rresp  <= 2'b0; // 'OKAY' response
```

```
                end
            else if (axi_rvalid && S_AXI_RREADY)
                begin
                    // Read data is accepted by the master
                    axi_rvalid <= 1'b0;
                end
        end
    end

    // Implement memory mapped register select and read logic generation
    // Slave register read enable is asserted when valid address is available
    // and the slave is ready to accept the read address.
    assign slv_reg_rden = axi_aready & S_AXI_ARVALID & ~axi_rvalid;
    always @(*)
    if(axi_awaddr[3:2] == 2'd0)
                reg_data_out[31 : 0] <= slv_reg0;
    else if(axi_awaddr[3:2] == 2'd1)
                reg_data_out[31 : 0] <= slv_reg1;
    else
                reg_data_out <= reg_data_out;

    // Output register or memory read data
    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_rdata    <= 0;
        end
      else
        begin
          // When there is a valid read address (S_AXI_ARVALID) with
          // acceptance of read address by the slave (axi_aready),
          // output the read dada
          if (slv_reg_rden) begin
                axi_rdata <= reg_data_out;          // register read data
            end
        end
    end

// Add user logic here

reg    [4 :0] i;
reg    [4 :0] j;
reg    [4 :0] k;
assign xdma_irq_req_o = xdma_irq_req;

assign user_irq_en_o = slv_reg1[31];
```

```
wire [XMDA_REQ_NUM-1:0] xdma_irq_ack = slv_reg0[XMDA_REQ_NUM-1:0];

always @( posedge S_AXI_ACLK ) begin
    if ( S_AXI_ARESETN == 1'b0 || user_irq_en_o == 1'b0 )begin
        user_irq_req_r1 = 0;
        user_irq_req_r2 = 0;
        user_irq_req_r3 = 0;
    end
    else begin
        user_irq_req_r1 <= user_irq_req_i;
        user_irq_req_r2 <= user_irq_req_r1;
        user_irq_req_r3 <= user_irq_req_r2;
    end
end

always @( posedge S_AXI_ACLK ) begin
      if ( S_AXI_ARESETN == 1'b0 || user_irq_en_o == 1'b0)begin
          j <= 5'd0;
          user_irq_req <= 0;
        end
        else begin
          for(j = 0; j <=XMDA_REQ_NUM-1; j = j +1 )
            user_irq_req[j] <= !user_irq_req_r3[j] & user_irq_req_r2[j];
        end
end

always @( posedge S_AXI_ACLK ) begin
      if ( S_AXI_ARESETN == 1'b0 || user_irq_en_o == 1'b0)begin
          i <= 5'd0;
          xdma_irq_req <= 0;
        end
        else begin
        for(i = 0; i <=XMDA_REQ_NUM-1; i = i +1 )begin
          if(xdma_irq_ack[i]) begin
                xdma_irq_req[i] <= 1'b0;
          end
          else if(user_irq_req[i])begin
                xdma_irq_req[i] <= 1'b1;
          end
        end
        end
end
    // User logic ends

    endmodule
```

为了方便测试中断，在 pcie_top.v 中，增加定时产生用户中断的程序。

```
wire axi_aclk;
wire axi_aresetn;
```

```
wire user_irq_en_o;
reg    [21:0]timer_cnt;

always @(posedge axi_aclk)begin
      if(!axi_aresetn||!user_irq_en_o)begin
                  timer_cnt <= 22'd0;
      end
      else begin
                  timer_cnt <= timer_cnt + 1'b1;
      end
end

reg timer_r1,timer_r2;
wire inter = !timer_r2 && timer_r1;

always @(posedge axi_aclk)begin
      if(!axi_aresetn||!user_irq_en_o)begin
                  timer_r1 <= 1'd0;
                  timer_r2 <= 1'd0;
      end
      else begin
                  timer_r1 <= timer_cnt[20];
                  timer_r2 <= timer_r1;
      end
end

reg [1:0] int_p;
reg [3:0]user_irq_req_i;

always @(posedge axi_aclk)begin
      if(!axi_aresetn||!user_irq_en_o)begin
                  int_p[1:0]    <= 4'd0;
                  user_irq_req_i <= 4'd0;
      end
      else begin
            if(inter)    int_p <= int_p + 1'b1;
            user_irq_req_i <= 4'd0;
            user_irq_req_i[int_p] <= 1'b1;
      end
end
```

1.9.3 中断上位机程序

实现中断程序的源码 intr_event.c：

```
#include <Windows.h>
#include <assert.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <strsafe.h>
#include <stdint.h>
#include <SetupAPI.h>
#include <INITGUID.H>
#include <WinIoCtl.h>
//#include <AtlBase.h>
#include <io.h>
#include "xdma_public.h"

#pragma comment(lib, "setupapi.lib")
#pragma warning(disable:4996)
BYTE      start_en;
HANDLE h_user;
DWORD    user_irq_ack[1];
char      base_path[MAX_PATH + 1] = "";


#define MAX_BYTES_PER_TRANSFER 0x800000

static int verbose_msg(const char* const fmt, ...) {

      int ret = 0;
      va_list args;
      if (1) {
            va_start(args, fmt);
            ret = vprintf(fmt, args);
            va_end(args);
      }
      return ret;

}
static BYTE* allocate_buffer(size_t size, size_t alignment) {

      if (size == 0) {
            size = 4;
      }

      if (alignment == 0) {
            SYSTEM_INFO sys_info;
            GetSystemInfo(&sys_info);
            alignment = sys_info.dwPageSize;
            //printf("alignment = %d\n",alignment);
      }
      verbose_msg("Allocating host-side buffer of size %d, aligned to %d bytes\n", size, alignment);
      return (BYTE*)_aligned_malloc(size, alignment);

}
```

```
static int get_devices(GUID guid, char* devpath, size_t len_devpath) {

    SP_DEVICE_INTERFACE_DATA device_interface;
    PSP_DEVICE_INTERFACE_DETAIL_DATA dev_detail;
    DWORD index;
    HDEVINFO device_info;
    wchar_t tmp[256];
    device_info = SetupDiGetClassDevs((LPGUID)&guid, NULL, NULL, DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);
    if (device_info == INVALID_HANDLE_VALUE) {
        fprintf(stderr, "GetDevices INVALID_HANDLE_VALUE\n");
        exit(-1);
    }

    device_interface.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);
    // enumerate through devices

    for (index = 0; SetupDiEnumDeviceInterfaces(device_info, NULL, &guid, index, &device_interface); ++index) {

        // get required buffer size
        ULONG detailLength = 0;
        if (!SetupDiGetDeviceInterfaceDetail(device_info, &device_interface, NULL, 0, &detailLength, NULL) && GetLastError() != ERROR_INSUFFICIENT_BUFFER) {
            fprintf(stderr, "SetupDiGetDeviceInterfaceDetail - get length failed\n");
            break;
        }

        // allocate space for device interface detail
        dev_detail = (PSP_DEVICE_INTERFACE_DETAIL_DATA)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, detailLength);
        if (!dev_detail) {
            fprintf(stderr, "HeapAlloc failed\n");
            break;
        }
        dev_detail->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

        // get device interface detail
        if (!SetupDiGetDeviceInterfaceDetail(device_info, &device_interface, dev_detail, detailLength, NULL, NULL)) {
            fprintf(stderr, "SetupDiGetDeviceInterfaceDetail - get detail failed\n");
            HeapFree(GetProcessHeap(), 0, dev_detail);
            break;
        }

        StringCchCopy(tmp, len_devpath, dev_detail->DevicePath);
        wcstombs(devpath, tmp, 256);
        HeapFree(GetProcessHeap(), 0, dev_detail);
```

```
    }

    SetupDiDestroyDeviceInfoList(device_info);

    return index;
}


HANDLE open_devices(char* device_base_path, char* device_name, DWORD accessFlags)
{
    char device_path[MAX_PATH + 1] = "";
    wchar_t device_path_w[MAX_PATH + 1];
    HANDLE h;

    // extend device path to include target device node (xdma_control, xdma_user etc)
    verbose_msg("Device base path: %s\n", device_base_path);
    strcpy_s(device_path, sizeof device_path, device_base_path);
    strcat_s(device_path, sizeof device_path, device_name);
    verbose_msg("Device node: %s\n", device_name);
    // open device file
    mbstowcs(device_path_w, device_path, sizeof(device_path));
    h = CreateFile(device_path_w, accessFlags, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (h == INVALID_HANDLE_VALUE)
    {
        fprintf(stderr, "Error opening device, win32 error code: %ld\n", GetLastError());
    }

    return h;
}

static int read_device(HANDLE device, long address, DWORD size, BYTE *buffer)
{
    DWORD rd_size = 0;

    unsigned int transfers;
    unsigned int i;
    if (INVALID_SET_FILE_POINTER == SetFilePointer(device, address, NULL, FILE_BEGIN)) {
        fprintf(stderr, "Error setting file pointer, win32 error code: %ld\n", GetLastError());
        return -3;
    }
    transfers = (unsigned int)(size / MAX_BYTES_PER_TRANSFER);
    for (i = 0; i<transfers; i++)
    {
        if (!ReadFile(device, (void *)(buffer + i*MAX_BYTES_PER_TRANSFER),
(DWORD)MAX_BYTES_PER_TRANSFER, &rd_size, NULL))
        {
            return -1;
        }
```

```
            if (rd_size != MAX_BYTES_PER_TRANSFER)
            {
                return -2;
            }
        }
        if (!ReadFile(device, (void *)(buffer + i*MAX_BYTES_PER_TRANSFER), (DWORD)(size - i*MAX_BYTES_PER_TRANSFER), &rd_size, NULL))
        {
            return -1;
        }
        if (rd_size != (size - i*MAX_BYTES_PER_TRANSFER))
        {
            return -2;
        }

        return size;
}

static int write_device(HANDLE device, long address, DWORD size, BYTE *buffer)
{
        DWORD wr_size = 0;
        unsigned int transfers;
        unsigned int i;
        transfers = (unsigned int)(size / MAX_BYTES_PER_TRANSFER);
        //printf("transfers = %d\n",transfers);
        if (INVALID_SET_FILE_POINTER == SetFilePointer(device, address, NULL, FILE_BEGIN)) {
            fprintf(stderr, "Error setting file pointer, win32 error code: %ld\n", GetLastError());
            return -3;
        }
        for (i = 0; i<transfers; i++)
        {
            if (!WriteFile(device, (void *)(buffer + i*MAX_BYTES_PER_TRANSFER), MAX_BYTES_PER_TRANSFER, &wr_size, NULL))
            {
                return -1;
            }
            if (wr_size != MAX_BYTES_PER_TRANSFER)
            {
                return -2;
            }
        }
        if (!WriteFile(device, (void *)(buffer + i*MAX_BYTES_PER_TRANSFER), (DWORD)(size - i*MAX_BYTES_PER_TRANSFER), &wr_size, NULL))
        {
            return -1;
        }
        if (wr_size != (size - i*MAX_BYTES_PER_TRANSFER))
        {
```

```
            return -2;
        }
        return size;
}


DWORD WINAPI thread_event0(LPVOID lpParam)
{
        BYTE val0[1] = "";
        DWORD i = 0;
        BYTE statu;
        char* device_name1 = "\\event_0";
        HANDLE h_event0 = open_devices(base_path, device_name1, GENERIC_READ);
            while (1)
            {
                if (start_en) {
                    read_device(h_event0, 0, 1, val0);//waite irq
                    Sleep(1);
                    if (val0[0] == 1)
                        printf("event_0 done!\n");
                    else
                        printf("event_0 timeout!\n");
                    i++;

                }
        }
        CloseHandle(h_event0);
        return 0;
}

DWORD WINAPI thread_event1(LPVOID lpParam)
{
        BYTE val0[1] = "";
        DWORD i = 0;
        BYTE statu;
        char* device_name1 = "\\event_1";
        HANDLE h_event1 = open_devices(base_path, device_name1, GENERIC_READ);
        while (1)
        {
            if (start_en) {
                read_device(h_event1, 0, 1, val0);//waite irq
                Sleep(1);
                if (val0[0] == 1)
                    printf("event_1 done!\n");
                else
                    printf("event_1 timeout!\n");
                i++;
```

```
            }
        }
        CloseHandle(h_event1);
        return 0;
}

DWORD WINAPI thread_event2(LPVOID lpParam)
{
        BYTE val0[1] = "";
        DWORD i = 0;
        BYTE statu;
        char* device_name1 = "\\event_2";
        HANDLE h_event2 = open_devices(base_path, device_name1, GENERIC_READ);
        while (1)
        {
            if (start_en) {
                read_device(h_event2, 0, 1, val0);//waite irq
                Sleep(1);
                if (val0[0] == 1)
                    printf("event_2 done!\n");
                else
                    printf("event_2 timeout!\n");
                i++;

            }
        }
        CloseHandle(h_event2);
        return 0;
}


DWORD WINAPI thread_event3(LPVOID lpParam)
{
        BYTE val0[1] = "";
        DWORD i = 0;
        BYTE statu;
        char* device_name1 = "\\event_3";
        HANDLE h_event3 = open_devices(base_path, device_name1, GENERIC_READ);
        while (1)
        {
            if (start_en) {
                read_device(h_event3, 0, 1, val0);//waite irq
                Sleep(1);
                if (val0[0] == 1)
                    printf("event_3 done!\n");
                else
                    printf("event_3 timeout!\n");
                i++;
```

```
            }
        }
        CloseHandle(h_event3);
        return 0;
}

int __cdecl main(int argc, char* argv[])
{

        HANDLE h_event0;
        HANDLE h_event1;
        HANDLE h_event2;
        HANDLE h_event3;
        HANDLE h_event4;
        HANDLE h_event5;
        HANDLE h_event6;
        HANDLE h_event7;


        char* device_name = "\\user";
        DWORD num_devices = get_devices(GUID_DEVINTERFACE_XDMA, base_path, sizeof(base_path));

        verbose_msg("Devices found: %d\n", num_devices);
        if (num_devices < 1)
        {
                printf("error\n");
        }

        h_user = open_devices(base_path, device_name, GENERIC_READ | GENERIC_WRITE);
        if (h_user == INVALID_HANDLE_VALUE)
        {
                fprintf(stderr, "Error opening device, win32 error code: %ld\n", GetLastError());
        }

        h_event0 = CreateThread(NULL, 0, thread_event0, NULL, 0, NULL);
        h_event1 = CreateThread(NULL, 0, thread_event1, NULL, 0, NULL);
        h_event2 = CreateThread(NULL, 0, thread_event2, NULL, 0, NULL);
        h_event3 = CreateThread(NULL, 0, thread_event3, NULL, 0, NULL);

        user_irq_ack[0] = 0xffff0000;
        write_device(h_user, 0x00004, 4, (BYTE*)user_irq_ack);//start irq
        start_en = 1;

        printf("start\n");
        WaitForSingleObject(h_event0, INFINITE);
        WaitForSingleObject(h_event1, INFINITE);
        WaitForSingleObject(h_event2, INFINITE);
```

```
    WaitForSingleObject(h_event3, INFINITE);

    user_irq_ack[0] = 0x00000000;
    write_device(h_user, 0x00004, 4, (BYTE*)user_irq_ack);//stop    irq
    CloseHandle(h_user);
    CloseHandle(h_event0);
    CloseHandle(h_event1);
    CloseHandle(h_event2);
    CloseHandle(h_event3);
    return 0;
}
```

以下指令启动中断

```
user_irq_ack[0] = 0xffff0000;
write_device(h_user, 0x00004, 4, (BYTE*)user_irq_ack);
```

以下指令关闭中断

```
user_irq_ack[0] = 0x00000000;
write_device(h_user, 0x00004, 4, (BYTE*)user_irq_ack);
```

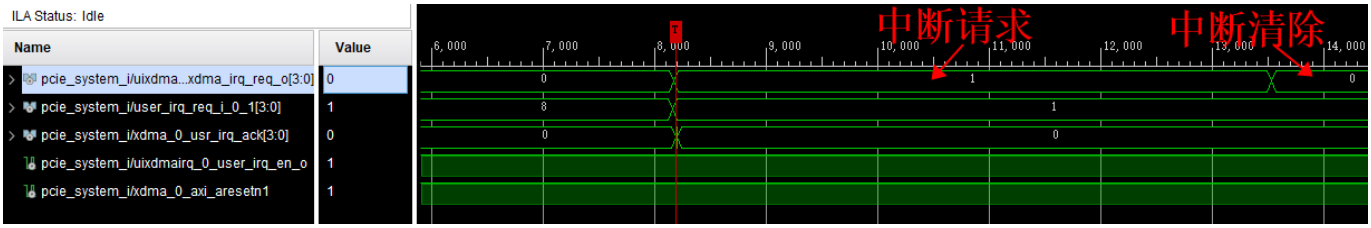
以上程序设置了 4 个中断事件，每个事件开启了一个线程，当中断等待的时候线程是挂起的，当中断产生后，继续执行线程。对于 XDMA 最大可以支持 32 个中断

1.9.4 中断测试

执行 xdma_event.exe 程序

可以看到运行结果是 4 个中断事件，实际上 XMDA 最大支持 32 个中断事件。更多的中断时间可以更好的发挥 CPU 多核多线程的性能。

看 FPGA 抓的波形信号



好了，到此 XDMA 的 PCIE 方案核心内容就已经讲完了。XILINX 官方给的资料往往没有细化，我们米联客已

经对以上驱动稍加修改，以更好的支持中断。

02QT 环境搭建及 PCIE 测速软件

软件版本：VIVADO2019.2
操作系统：WIN10 64bit
硬件平台：适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA
登录“米联客”FPGA 社区-www.uisrc.com 视频课程、在线答疑！

2.1 概述

经过前面章节的学习，如果读者认真学习应该已经掌握了 PCIE XDMA 方案的使用，那么我们知道 QT 可以设计出华丽的界面，那么本章就是设计一个简单的测速码表程序，比起前面的章节测试，这个小程序界面非常酷。

2.2QT 开发环境搭建

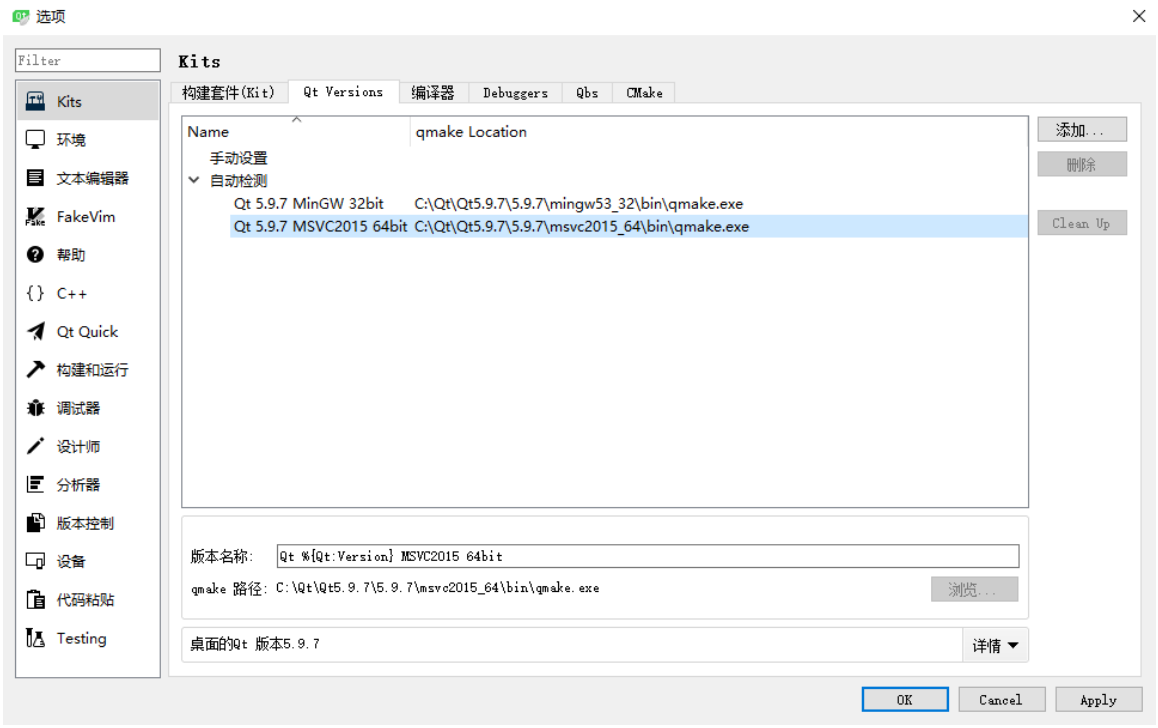
在 WIN 下，我们同样使用 QT Creator 工具。首先需要安装 qt 功能软件，读者可在米联网盘下载教程所使用的 QT，之所以选用 QT5，主要是为了配合 VS2015. 下载完软件以后，进行安装，注意，这个 QT 软件自带了 QT Creator，在安装的时候注意勾选即可，没有别的要注意的地方。

首先需要安装 qt 功能软件，读者可在米联客(MSXBO)网盘下载教程所使用的 QT，之所以选用 QT5，主要是为了配合 VS2015.

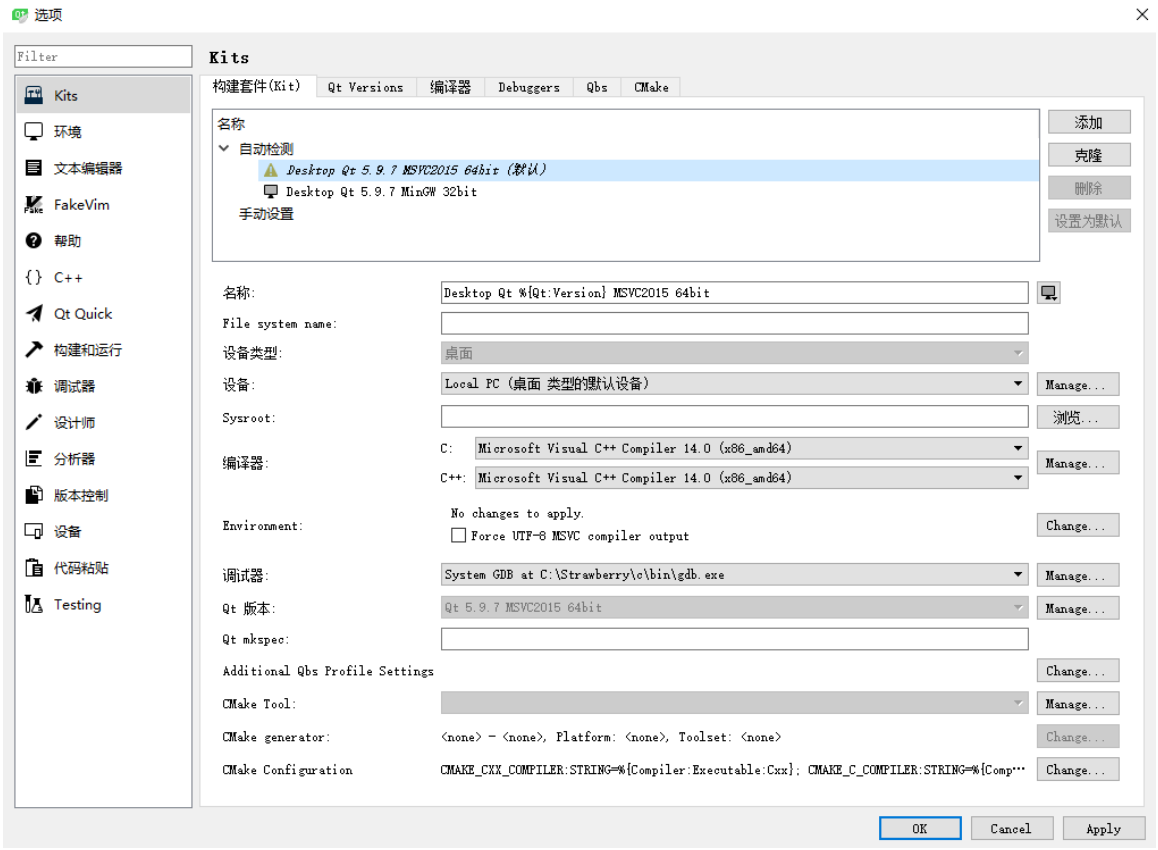
下载完软件以后，进行安装，注意，这个 QT 软件自带了 QT Creator，在安装的时候注意勾选即可，没有别的要注意的地方。

安装 qt-opensource-windows-x86-5.9.7.exe
按照安装步骤一步一步进行即可。用户可能会需要联网注册账号后才能进一步安装，根据软件向导提示进行。

安装完成以后，我们要对 QT Creator 进行配置。打开 QT Creator，工具，选项，选择构建和运行。首先是 QT Version 配置，一般 Creator 自动搜索了，如果没有搜索到，需要手动配置



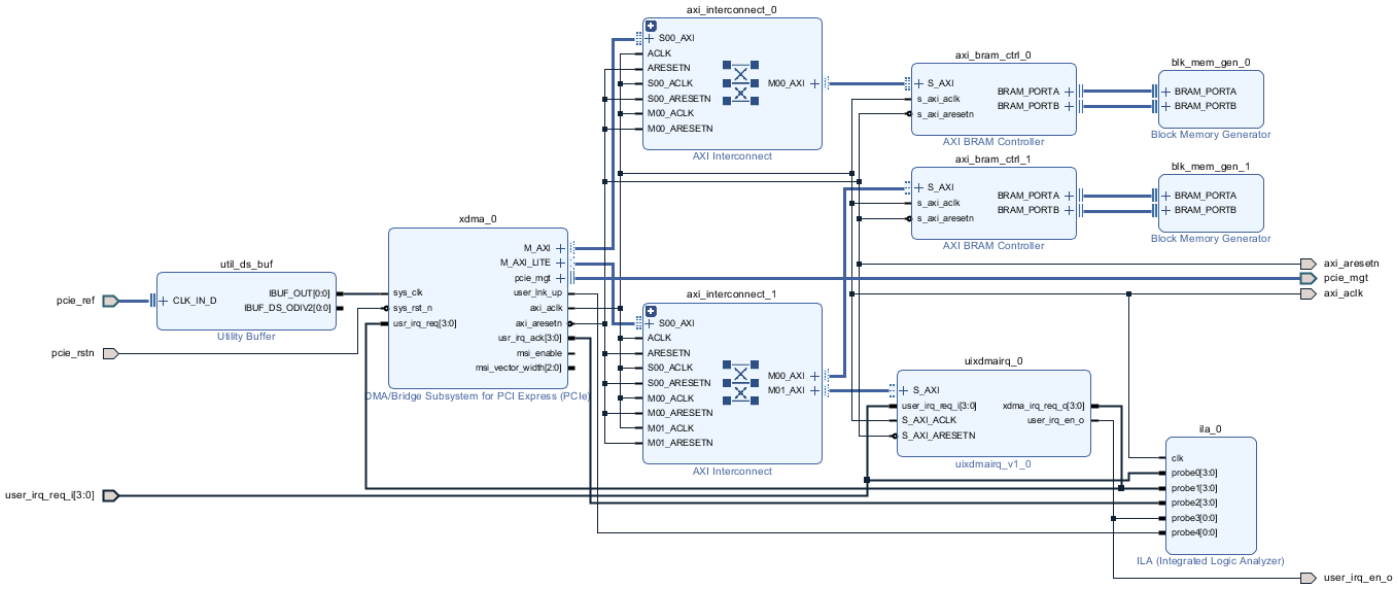
下面选择构建套件（Kit）按照如下配置：



配置完成以后即可，然后打开本教程提供的 QT Creator 工程进行编译即可。

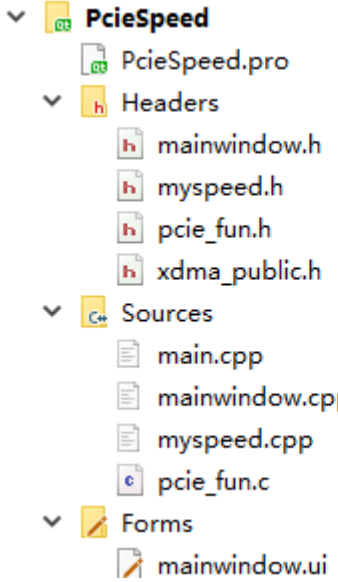
2.3FPGA 代码

FPGA 可以用任何一个章节的，这里就用 CH01 的代码，简单。具体就不重复了，不清楚的可以看 CH01 部分教程内容。本文中的上位机没有使用中断功能。



2.4 测试码表上位机程序设计

首先感谢网友贡献的测试码表控件源码，笔者修改了几个参数就可以正常使用到本次的例子中了。设计思路也很简单，在 QT 中开启了 2 个定时器，分别用于 h2c 和 c2h 通道，每过 100ms 定时器进行一次读操作或者写操作。在 pcie_fun.c 文件中，有测试函数，完成传输测试后，把结果的值输出到 myspeed 测速码表控件。

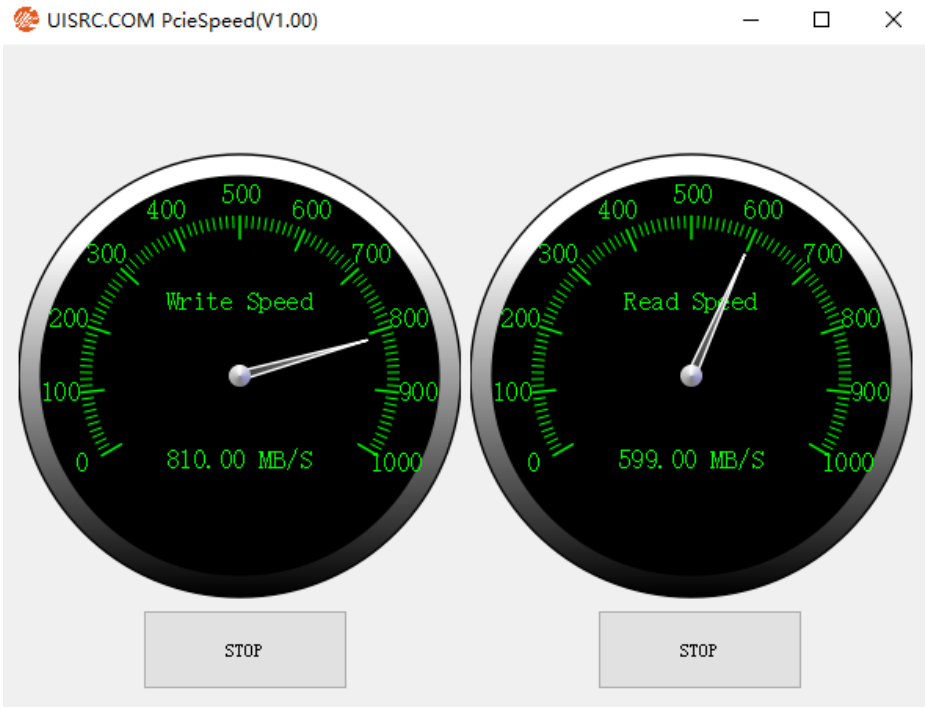


以上代码中，显示速度的控件代码是myspeed.c和myspeed.h

关于 PCIE 通信的核心代码是 pcie_fun.c 和 pcie_fun.h 详细的实现过程读者可以阅读程序源码。

2.5 测速结果

不同的板卡，PCIE 的最大带宽不一样，以实际为准。下面的是 PCIEX2 2.0 的测速指标。


UISRC.COM PcieSpeed(V1.00)
Write Speed
810.00 MB/S
Read Speed
599.00 MB/S
STOP
STOP

03QT 上位机读写 FPGA 内存

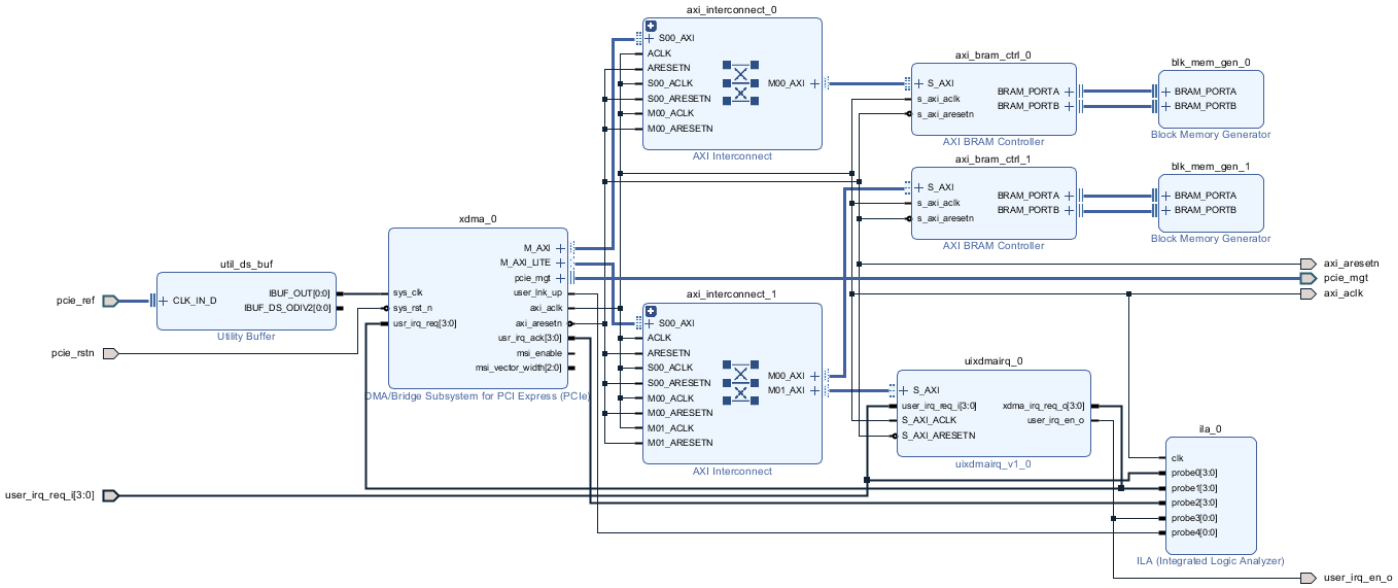
软件版本：VIVADO2019.2
操作系统：WIN10 64bit
硬件平台：适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA
登录“米联客”FPGA 社区-www.uisrc.com 视频课程、在线答疑！

3.1 概述

如果读者对于 01 和 02 还有一些疑惑，那么本节课的内容，可以让你更加简单地弄明白什么是 BAR 地址空间操作，什么是对开发板 DDR 内存地址空间操作。如果你掌握了如何通过 XDMA 读写 BAR 地址空间和 DMA 内存地址空间操作，那么基于 XMDA 的所有原理性设计都会变的 so easy!

3.2FPGA 代码

FPGA 可以用任何一个章节的，这里就用“01 驱动安装及 XDMA 实现 PCIE 通信”的代码，简单。具体就不重复了，不清楚的可以看 01 部分教程内容。本文中的上位机没有使用中断功能。



3.3QT 程序设计

如果你足够细心，你会发现在 CH01 里面的 BRAM 和 DDR 我们在前面的课程中没有用到。那么这节课就可以用起来了。测试的原理是分别往 BAR 地址空间和 DDR 地址空间写入测试数据，然后读出看看写入的和读出的十分一致。同理，。程序比较简单，读者可以自己阅读 QT 源码。

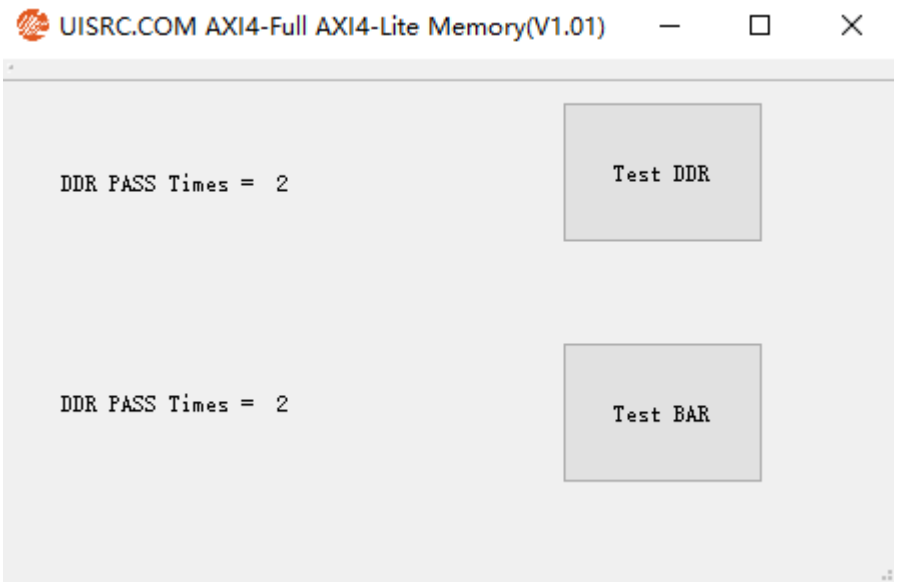
读写代码,以下代码中 on_TestDDR_clicked 是上是对 AXI4 接口的 DDR 或者 BRAM 测试。on_TestBAR_clicked 是对用户的 BAR 空间测试。

```
void MainWindow::on_TestDDR_clicked()
{
    unsigned int buf1[1024];
    unsigned int buf2[1024];
    unsigned int i=0;

    unsigned int error_cnt =0;
```

```
    for(i=0;i<1024;i++)
    {
        buf1[i]=i;
    }

    h2c_transfer(0,1024*4,(unsigned char *)buf1);

    c2h_transfer(0,1024*4,(unsigned char *)buf2);

    for(i=0;i<1024;i++)
    {
        if(buf1[i] != buf2[i])
            error_cnt++;
    }
    if(error_cnt)
    {
        QString str  =  QString("%1 %2").arg("DDR bad data = ").arg(error_cnt);
        ui->labelDDRPASS->setText(str);
    }
    else
    {
        m_pass1++;
        QString str  =  QString("%1 %2").arg("DDR PASS Times = ").arg(m_pass1);
        ui->labelDDRPASS->setText(str);
    }

}

void MainWindow::on_TestBAR_clicked()
{
    unsigned int val;
    unsigned int i=0;

    unsigned int error_cnt =0;

    for(i=0;i<1024;i++)
    {
        user_write(0x10000+i*4,4,(unsigned char *)&i);
    }

    for(i=0;i<1024;i++)
    {
        user_read(0x10000+i*4,4,(unsigned char *)&val);
        if(i!= val)
            error_cnt++;
    }
```

```
    if(error_cnt)
    {
        QString str  =  QString("%1 %2").arg("DDR bad data = ").arg(error_cnt);
        ui->labelBARPASS->setText(str);
    }
    else
    {
        m_pass2++;
        QString str  =  QString("%1 %2").arg("DDR PASS Times = ").arg(m_pass2);
        ui->labelBARPASS->setText(str);
    }


}
```

3.4 测试结果



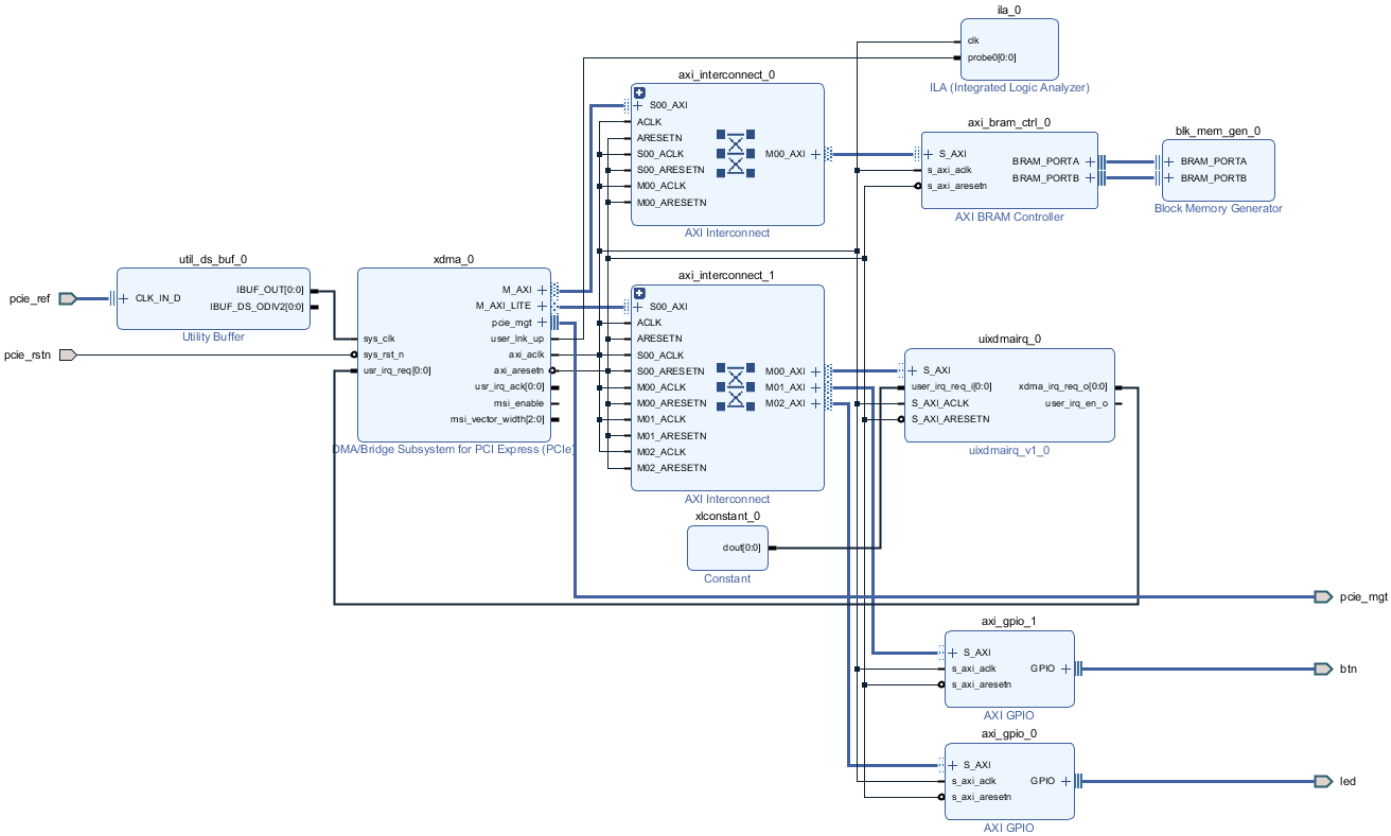
04PCIE 的 GPIO 控制卡

软件版本：VIVADO2019.2
操作系统：WIN10 64bit
硬件平台：适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA
登录“米联客”FPGA 社区-www.uisrc.com 视频课程、在线答疑！

4.1 概述

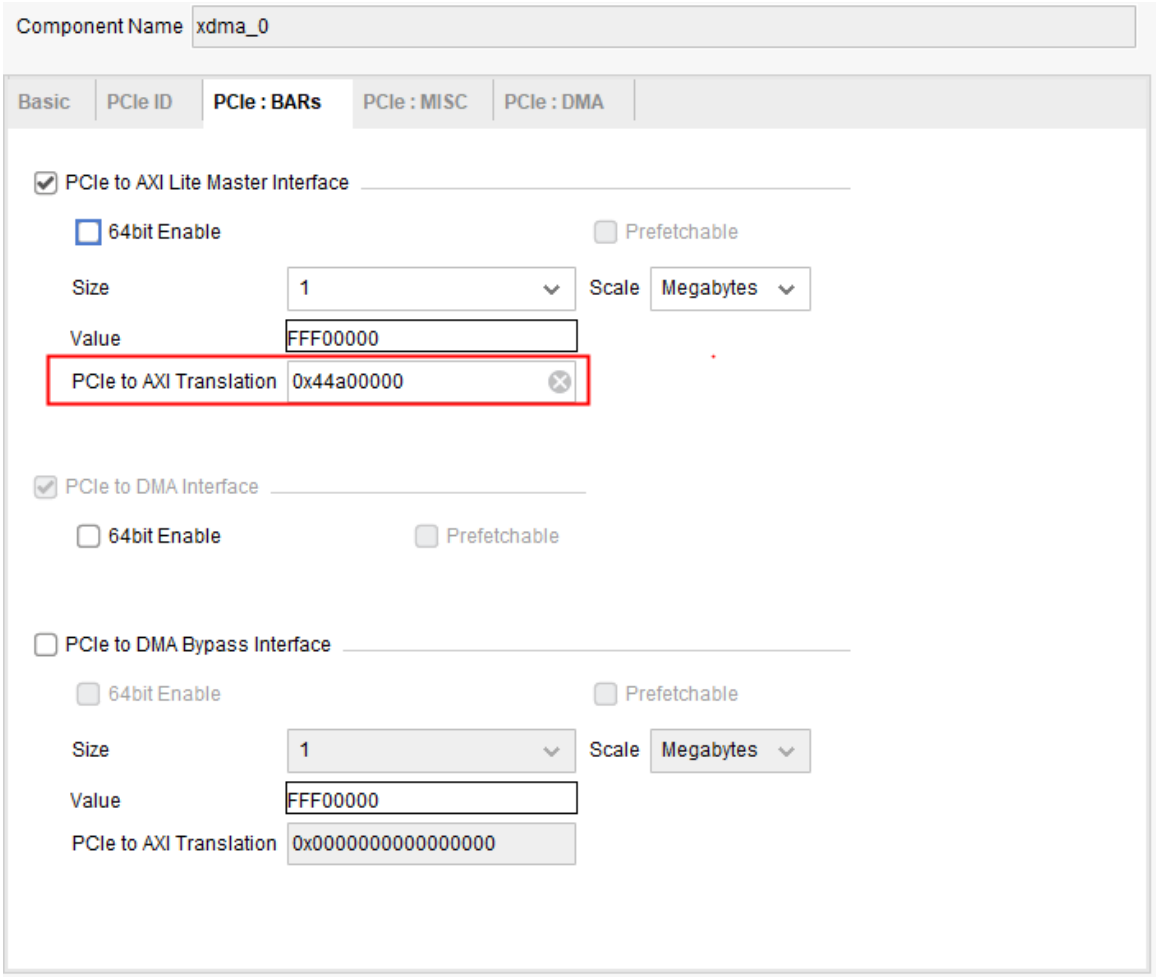
还记得 2008 年左右刚刚参加工作，买一个简单的 PCI 的 DAQ 数据卡，或者 IO 卡都要好几千元。市面上主要是研华，凌华的数据卡。现在我们也可以用 PCIE 做出更好的数据卡了。那么本节课就是设计一个 GPIO 的 IO 卡方案。

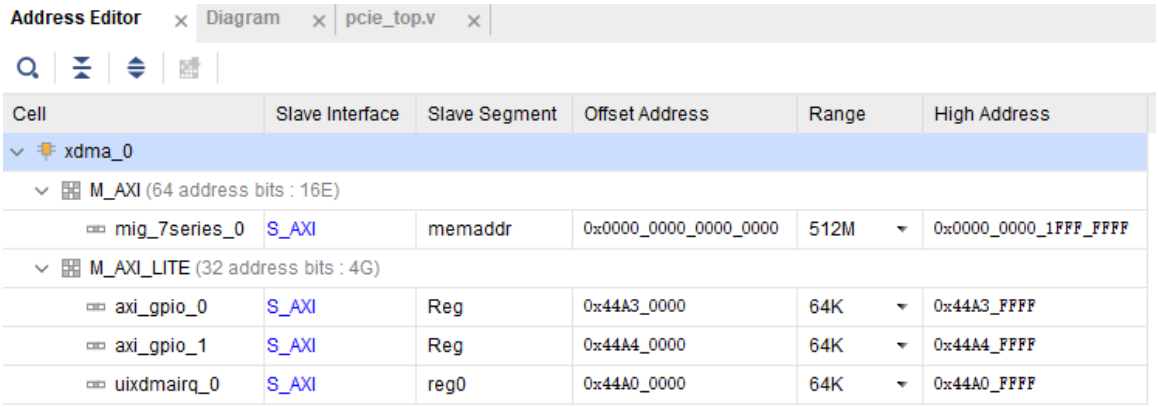
4.2FPGA 代码



上面接的 DDR(MA703-35T 是 BRAM)实际上没有用到。我们这里使用了 2 个 AXI GPIO，分别定义了 4 个 LED 输出，和 4 个 BTN 按钮输入。AXI-GPIO 大家应该很熟悉了吧，之前在 ZYNQ 上操作，很简单，先我们把 AXI-GPIO 的 IP 地址空间映射到了 BAR 空间，这样就可以直接访问控制这些 IP 了。

首先是 XDMA 的地址空间分配



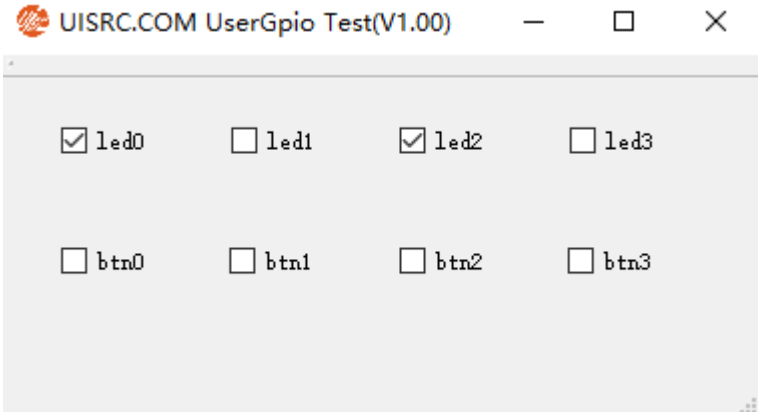


4.3QT 程序设计

本课程的 QT 程序超级简单，我们在界面上设计了一些 check box 用于 LED 和 BTN 的状态显示。程序比较简单，读者可以自己阅读 QT 源码。

4.4 测试结果

装在机箱里面的开发板的上 LED 不容易看清楚，我们点击 LED check box 可以看到 LED 灯光的变化。装在机箱的按钮也不容易按到，所以测试的时候要小心些，以下是测试结果。


UISRC.COM UserGpio Test(V1.00)
led0
led1
led2
led3
btn0
btn1
btn2
btn3

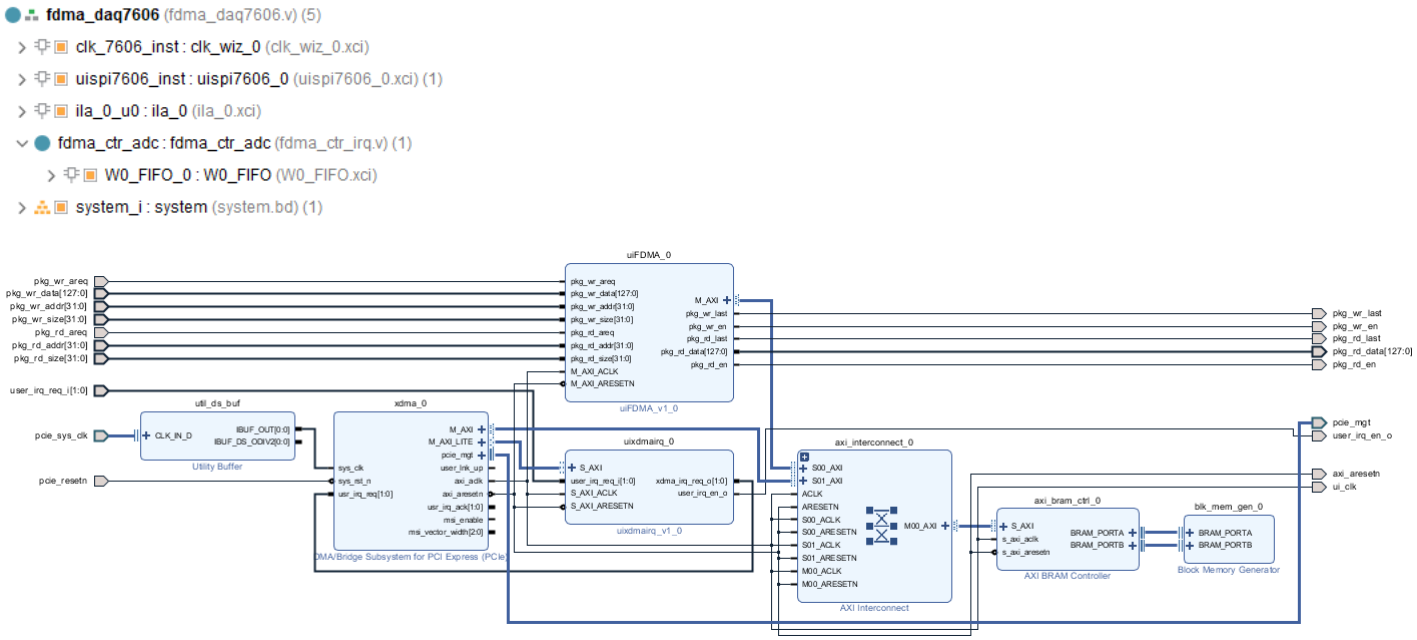
05PCIE 数据卡 BRAM 缓存中断采集

软件版本：VIVADO2019.2
操作系统：WIN10 64bit
硬件平台：适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA
登录“米联客”FPGA 社区-www.uisrc.com 观看免费视频课程、在线答疑解惑！

5.1 概述

在上一个版本 PCIE 教程中，使用的 M_AXI 接口接到自定义的 AXI-SLAVE 接口,然后经过 FIFO 读取 ADC 数据。在本版本中，使用基于 AXI4 实现的 FDMA 来实现数据的缓存。通过切换缓存的地址，实现 2 帧以上缓存数据的读取。这种构架更加方便高效。帧的管理，使用到了 PCIE 中断。本文中缓存使用了 FPGA 芯片自带的 BRAM，可以在目前硬件配置最低的 MA703FA-35T 上实现本方案。

5.2FPGA 工程



新版本的 DAQ7606 只支持 SPI 串行采样源码封装成 uispi7606 IPCORE，教程的第一部分课程中有详细的讲解分析。采用 SPI 接口可以省下更多的 FPGA IO。

FDMA 缓存构架方式是米联客成熟的构架，可以适合一切多帧缓存的应用需求。

```
`timescale 1ns / 1ps
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand:  米联客(msxbo)
Technical forum:uisrc.com
taobao: osrc.taobao.com
Create Date: 2019/12/17
Module Name: fdma_ctr_adc
Description:
Copyright: Copyright (c) msxbo
Revision: 1.0
Signal description:
```

```
1) _i input
2) _o output
3) _n activ low
4) _dg debug signal
5) _r delay or register
6) _s state mechine
*/
//////////////////////////////////////////////////////////////////////////////////

module fdma_ctr_adc#
(
parameter   integer   ADDR_OFFSET = 0,
parameter   integer   AXI_BURST_LEN   = 256,
parameter   integer   AXI_DATA_WIDTH = 128,
parameter   integer   FDMA_BUF_SIZE = 3,
parameter   integer   FDMA_BUF_LEN   = 1024// FDMA_BUF_LEN*4 BYTES
)
(
      input                  ui_clk,
      input                  ui_rstn,
//sensor input -W0_FIFO----------
//       input                  W0_FS_i,
      input                  W0_wclk_i,
      input                  W0_wren_i,
      input   [31:0]     W0_data_i,
//----------fdma signals write-------
      output   reg       pkg_wr_areq,
      input                  pkg_wr_en,
      input                  pkg_wr_last,
      output   [31:0]   pkg_wr_addr,
      output   [31:0]   pkg_wr_data,
      output   [31:0]   pkg_wr_size,
      output   reg [1 :0]   xdma_irq_req
      );

parameter FBUF_SIZE     = FDMA_BUF_SIZE -1'b1;
parameter BURST_SIZE   = AXI_BURST_LEN*AXI_DATA_WIDTH/8;//BYTES 设置 FDMA 每次 burst 大小正好等于 AXI4 总线 burst 大小,如果这里设置成 4 倍的 AXI_BURST_LEN*AXI_DATA_WIDTH/8，那么只要 burst 1 次，FIFO 也得改
parameter PKG_SIZE       = AXI_BURST_LEN;// lenth of AXI_DATA_WIDTH
parameter BURST_TIMES = FDMA_BUF_LEN/(BURST_SIZE/4);//计算为了传输 FDMA_BUF_LEN 量 数据，需要多少次 FDMA burst
assign pkg_wr_size = PKG_SIZE;//256*128 =4KB 和 FDMA 的参数设置一致，每次最大传输 4KB,总线效率最高
//------------vs 滤波---------------
//reg   W0_FIFO_Rst;
parameter S_IDLE   =   2'd0;
parameter S_RST    =   2'd1;
parameter S_DATA1 =   2'd2;
```

```
parameter S_DATA2 =    2'd3;

reg [1 :0]    W_MS;
reg [13:0]    W0_addr;
reg [31 :0] W0_fcnt;
reg [10 :0] W0_bcnt;
wire[9 :0]    W0_rcnt;
reg W0_REQ;

reg [6:0] W0_Fbuf;
wire    [3:0] Fbuf = (W0_Fbuf==0) ? FBUF_SIZE : (W0_Fbuf - 1);

always @(posedge ui_clk)begin
      if(!ui_rstn)begin
            xdma_irq_req <= 2'd0;
      end
      else begin
            xdma_irq_req <= 0;
            xdma_irq_req[Fbuf] <= 1'b1;
      end
end

assign pkg_wr_addr = {12'd0,W0_Fbuf,W0_addr}+ ADDR_OFFSET;
//assign pkg_wr_data = {32'hffff0000,32'hffff1111,32'haaaa0000,W0_fcnt};
//--------一副图像写入 DDR------------
  always @(posedge ui_clk) begin
      if(!ui_rstn)begin
            W_MS <= S_IDLE;
            W0_addr <= 14'd0;
            pkg_wr_areq <= 1'd0;
            W0_fcnt    <= 0;
            W0_bcnt    <= 0;
            W0_Fbuf    <= 7'd0;
      end
      else begin
         case(W_MS)
           S_IDLE:begin
               W0_addr    <= 14'd0;
               W0_fcnt    <= 0;
               W0_bcnt    <= 11'd0;
               W_MS <= S_RST;
           end
           S_RST:begin
               if(W0_fcnt > 8'd30 ) W_MS <= S_DATA1;
               W0_fcnt <= W0_fcnt +1'd1;
            end
            S_DATA1:begin
                  if(W0_bcnt == BURST_TIMES) begin
```

```
                    if(W0_Fbuf == FBUF_SIZE)
                        W0_Fbuf <= 7'd0;
                     else
                        W0_Fbuf <= W0_Fbuf + 1'b1;
                     W_MS <= S_IDLE;
                end
                else if(W0_REQ) begin
                    W0_fcnt <=0;
                    pkg_wr_areq <= 1'b1;
                    W_MS <= S_DATA2;
                end
            end
            S_DATA2:begin
                pkg_wr_areq <= 1'b0;
                if(pkg_wr_last)begin
                    W_MS <= S_DATA1;
                    W0_bcnt <= W0_bcnt + 1'd1;
                    W0_addr <= W0_addr + BURST_SIZE;
                end
            end
        endcase
    end
end


always@(posedge ui_clk)
begin
      W0_REQ       <= (W0_rcnt    > PKG_SIZE - 2);
end

wire [31:0]fifo_data;
assign pkg_wr_data=fifo_data;//{
                            // fifo_data[31 : 0],
                            //    fifo_data[63 : 32],
                            //    fifo_data[95 : 64],
                            //    fifo_data[127: 96]
                            // };

W0_FIFO W0_FIFO_0 (
  .rst(~ui_rstn),    // input wire rst
  .wr_clk(W0_wclk_i),    // input wire wr_clk
  .din(W0_data_i),              // input wire [31 : 0] din
  .wr_en(W0_wren_i),        // input wire wr_en
  .rd_clk(ui_clk),    // input wire rd_clk
  .rd_en(pkg_wr_en),        // input wire rd_en
  .dout(fifo_data),           // output wire [31 : 0] dout
  .rd_data_count(W0_rcnt)    // output wire [10 : 0] wr_data_count
);
```

```
endmodule
```

5.3 测试结果

QT程序主要用到了QWT进行波形显示，以下链接为QWT的编译和安装，也可以使用本文已经编译好的qwt库。

http://www.uisrc.com/forum.php?mod=viewthread&tid=1873&extra=page%3D1

上位机代码比较简单，可以自己阅读。下面给出测试结果。