

PCIE 之 XDMA 进阶

黑金动力社区 2019-04-26

1 实验简介

本实验采用的平台是 AX7103 开发板，例程中通过配置 XDMA IP 的多种接口来实现了对 XILINX 的 XDMA IP 的测试，让大家更好地了解 XDMA IP 的各种接口的用途。

2 实验原理

2.1 例程简介

例程中 FPGA 端程序采用 XILINX 的 XDMA 进行设计。主要实现的功能是：添加了用户事件响应，增加了 M_AXI_LITE 及 M_AXI_BYPASS 端口外设通信的应用；PC 端的驱动和上位机的程序都是 XILINX 官方提供，资料路径在下方做了说明。

例程由三部分组成：FPGA 端程序、PCle 卡驱动、PCle 上位机测试程序。

FPGA 端程序：负责建立与 PCle 通信需具备的 FPGA 框架，PCle 通信协议的构建；

PCle 卡驱动：负责上位机测试程序与 PCle 卡的数据交换；

PCle 上位机测试程序：对例程的各个功能进行测试。

在进行 PCle 测速例程之前，确保计算机为 WIN7（64 位）或 WIN10（64 位）系统。

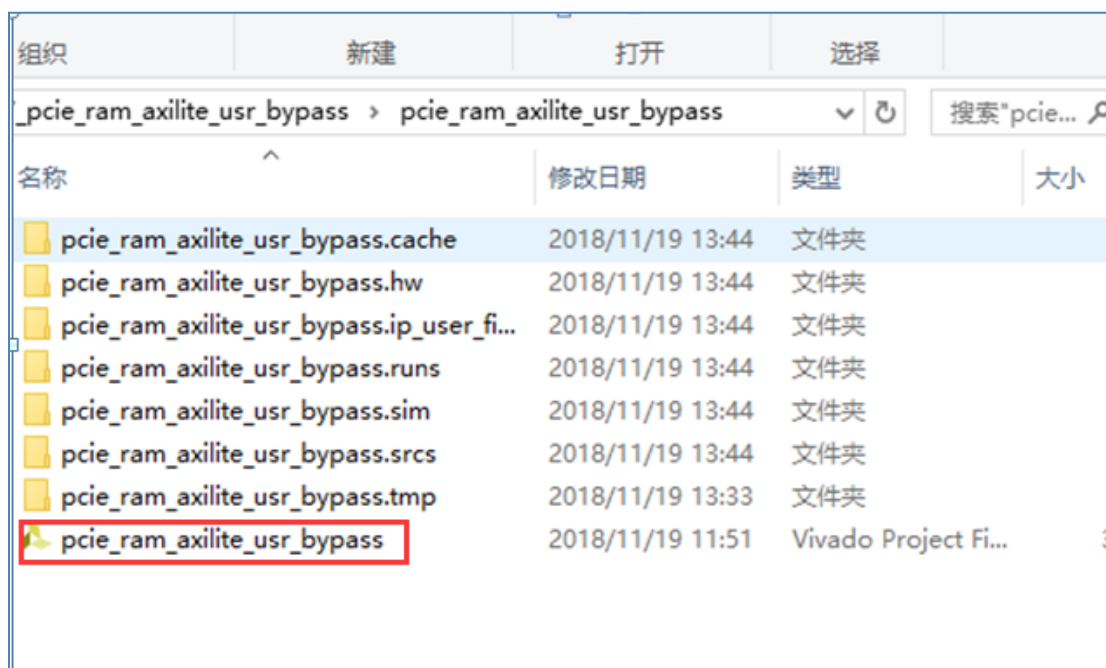
2.2 硬件描述

这里忽略，见《PCle 速度测试例程》教程相关章节。

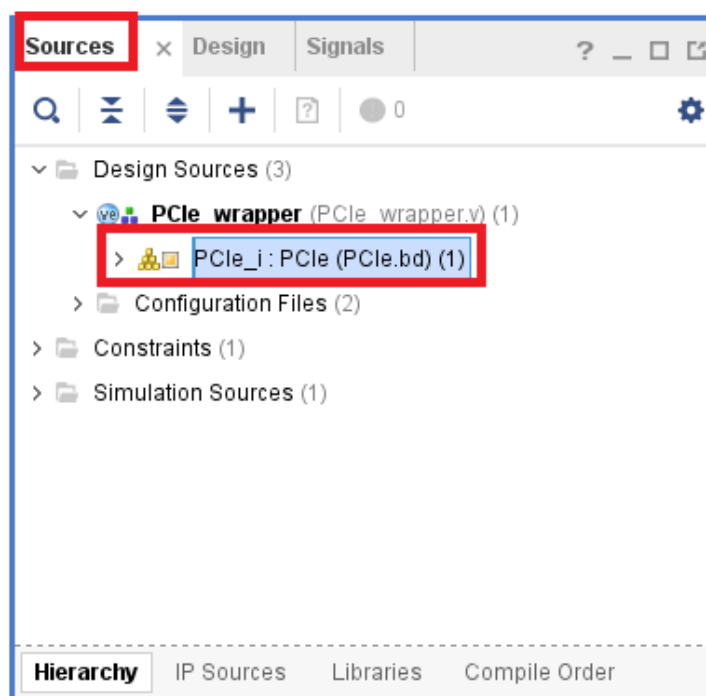
3 程序设计

3.1 FPGA 程序

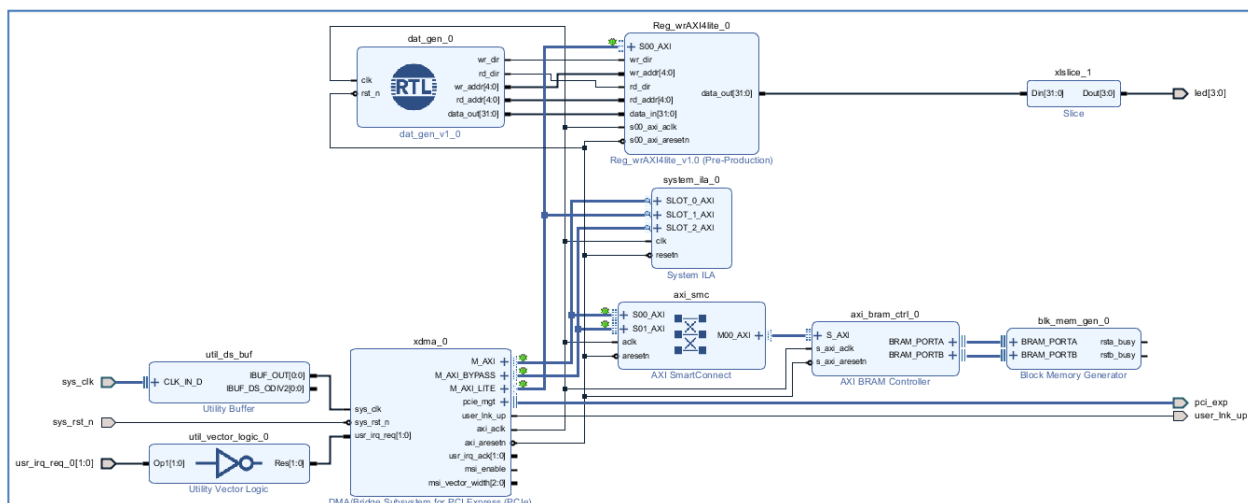
打开 Vivado 的 FPGA 工程，FPGA 程序位于如下图位置



设计中采用 XILINX 的 Block Design，Block Design 的程序设计可参考 PCIe_test 例程的文档，打开 PCIe.bd：



可看到打开后的 PCIe 如下



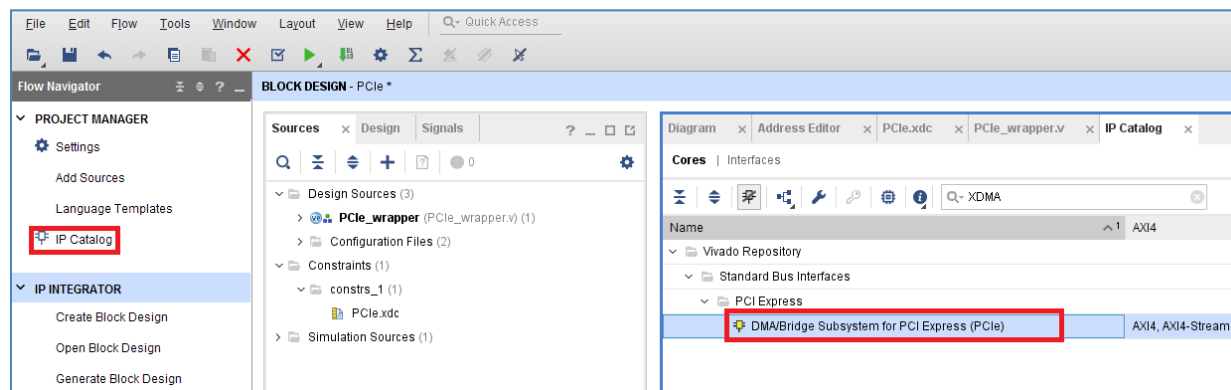
程序模块说明：PCIe 通信程序由 util_ds_buf、axi_smc、axi_bram_ctrl_0、blk_mem_gen_0、util_vector_logic_0、Reg_wrAXI4lite_0、dat_gen_0、xlslice_1、system_ila_0 及 xdma_0 组成。

util_ds_buf 是对外部的 PCIe 输入时钟进行 buffer；axi_smc 是 Master 和 Slave 接口设备互联的协议模块，例程中的作用是通过 M_AXI 总线和 M_AXI_BYPASS 来控制 blk_mem_gen_0 RAM；axi_bram_ctrl_0 是 blk_mem_gen_0 的控制器；util_vector_logic_0 是逻辑非门，对用户信号处理；dat_gen_0 模块作用是产生数据及地址的测试信号，里面逻辑比较简单，用户自己可以去完善，用来验证 Reg_wrAXI4lite_0 的读写功能；Reg_wrAXI4lite_0 模块是自定义 IP，连接到 XDMA 模块，其功能是定义了一些用户可操作的寄存器，以便在进行 PCIe 通信时进行一些简单信号的交互，双字地址 0 到 9 为 FPGA 端可写寄存器，双字地址 10 到 19 为 FPGA 端可读寄存器；xlslice_1 用来进行数据宽度变换，在这里连接 LED 灯（LED1、LED2、LED3、LED4），可用上位机控制控制 LED 状态；system_ila_0 是信号分析模块，用来进行调试的作用；XDMA_0 模块是 PCIe 通信模块，内部具备 DMA 功能。

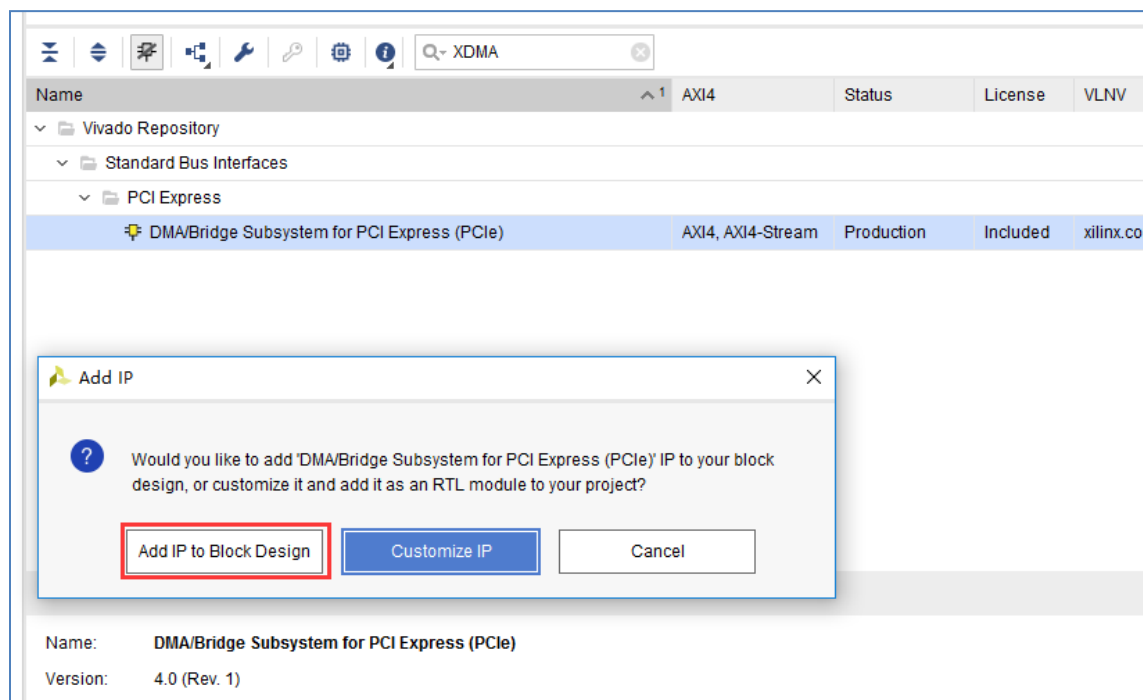
下面主要介绍下 XDMA_0 模块配置和 dat_gen_0 模块的产生，其它不做介绍：

XDMA_0 模块配置如下：

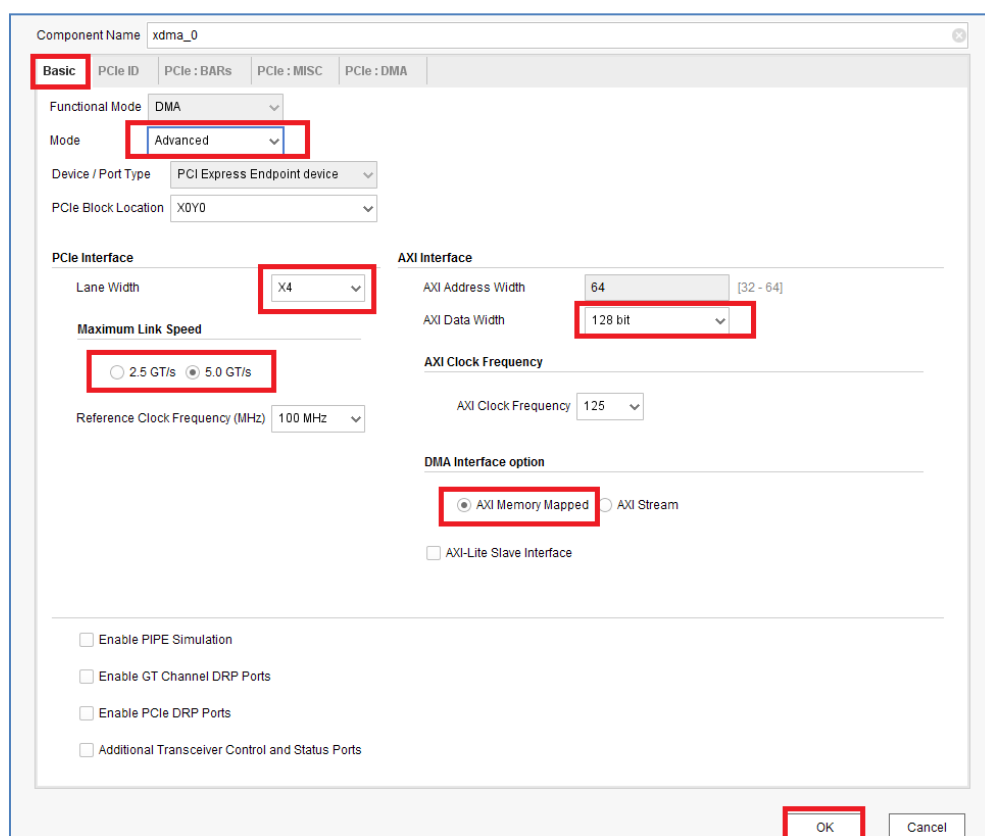
1) 在 IP Catalog 中找到如下图中的 PCIe，并双击。



2) 在弹出的对话框中选择自定义 IP:



3) 按下图进行 Basic 栏设置, 这里选择主要是 Mode、PCIe Lane、Link Speed、DMA 接口方式选择等, 按图中设置即可。



4) PCIe ID 栏按下图红色框中设置，其它默认：

Component Name: xdma_0

Basic **PCIe ID** PCIe : BARs PCIe : MISC PCIe : DMA

ID Initial Values

Vendor ID	10EE	
Device ID	7024	
Revision ID	00	
Subsystem Vendor ID	10EE	
Subsystem ID	0007	

Class Code Lookup Assistant

☐ Use Class Code Lookup Assistant

Base Class Menu	Simple communication controllers	
Base Class Value	05	Range: 00..FF
Sub Class Interface Menu	Generic XT compatible serial controller	
Sub Class Value	80	Range: 00..FF
Interface Value	00	Range: 00..FF
Class Code	058000	Range: 000000..FFFFFF

5) PCIe:BARs 如下设置、PCIe:MISC 中保持默认设置。

Re-customize IP

DMA/Bridge Subsystem for PCI Express (PCIe) (4.0)

Documentation IP Location

☐ Show disabled ports

Component Name: xdma_0

Basic PCIe ID **PCIe : BARs** PCIe : MISC PCIe : DMA

☒ PCIe to AXI Lite Master Interface

☐ 64bit Enable ☐ Prefetchable

Size: 1 Scale: Megabyte

Value: FFF00000

PCIe to AXI Translation: 0x0000000000000000

☒ PCIe to DMA Interface

☐ 64bit Enable ☐ Prefetchable

Size: 1 Scale: Megabyte

Value: FFF00000

PCIe to AXI Translation: 0x0000000000000000

Background task running on the design. Customization changes are not allowed

Component Name: xdma_0

Basic | PCIe ID | PCIe : BARs | **PCIe : MISC** | PCIe : DMA

User Interrupts

Number of User Interrupts Request (1-16): 2

Legacy Interrupt Settings

Legacy Interrupt Settings: INTA

MSI Capabilities

☒ Enable MSI Capability Structure

Multiple Message Capability: 1 vector

MSI-X Capabilities

☐ Enable MSI-X Capability Structure

Miscellaneous

☐ Finite Completion Credits

☒ Extended Tag Field

☐ Add the PCIe XVC-VSEC to the Example Design

☐ Configuration Management Interface

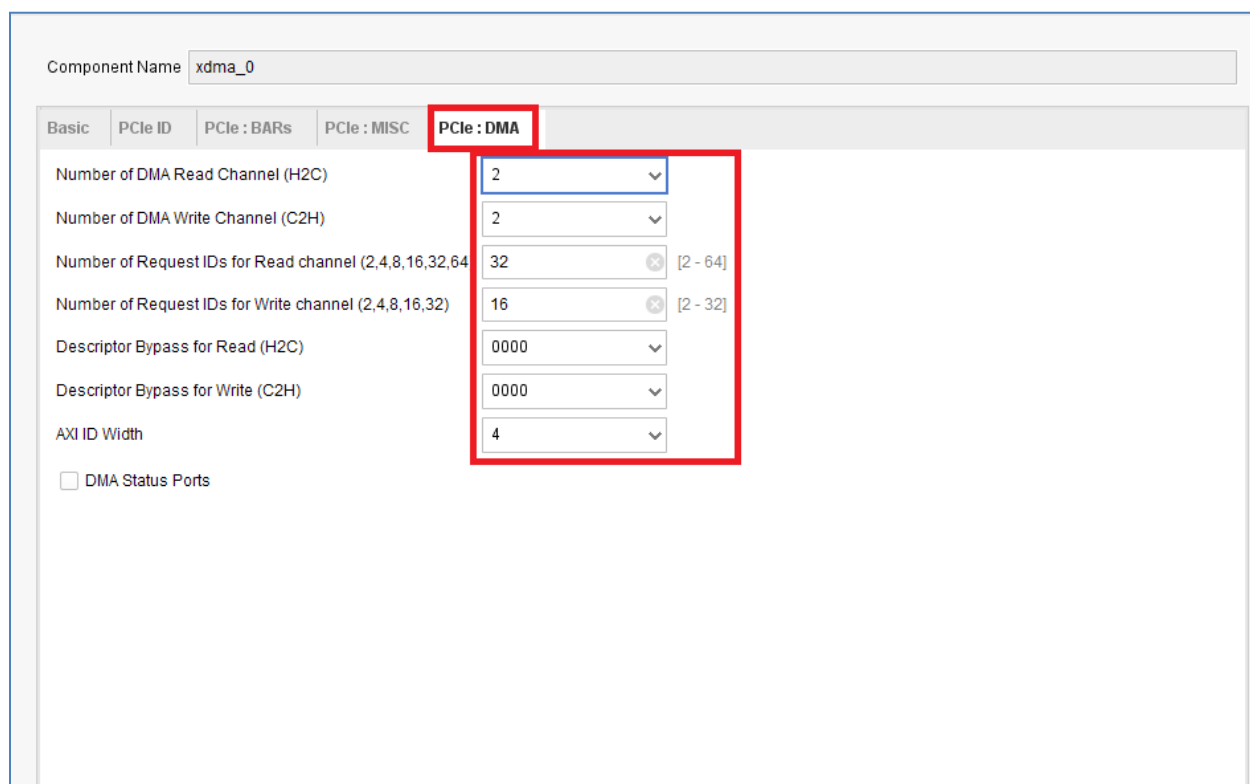
Link Status Register

Selects whether the device reference clock is provided by the connector (Synchronous) or generated via an onboard PLL(Asynchronous)

☒ Enable Slot Clock Configuration

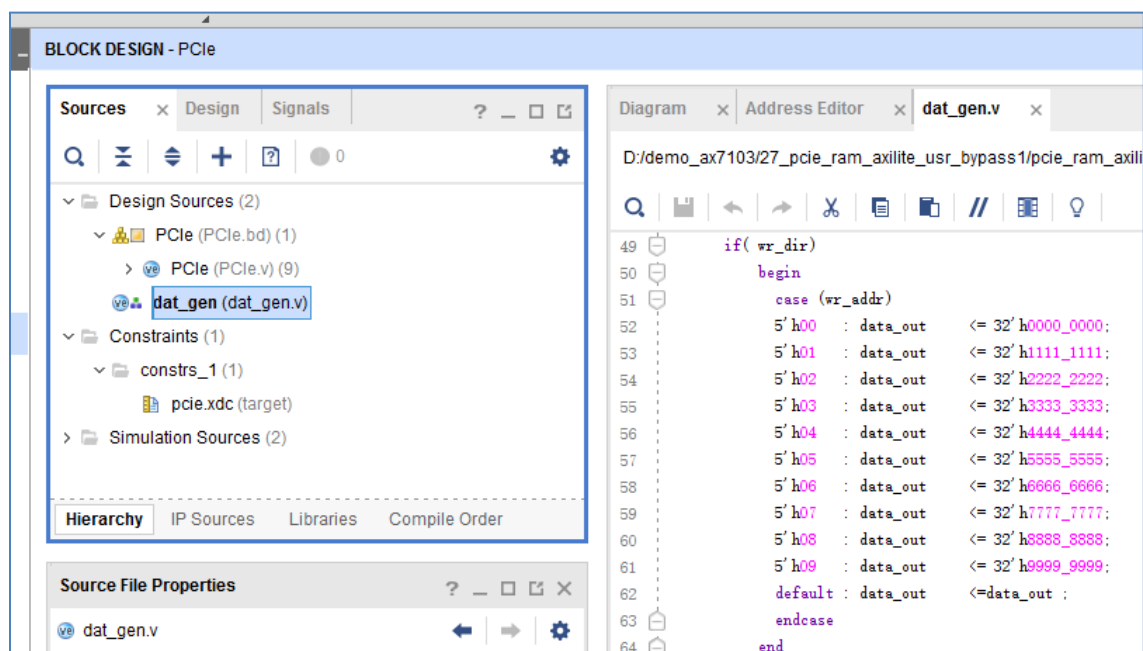
OK Cancel

6) PCIe:DMA 主要设置 DMA 的参数: H2C 通道数、C2H 通道数、ID 设置, 采用下图设置即可, 然后单击 OK 即可完成设置。

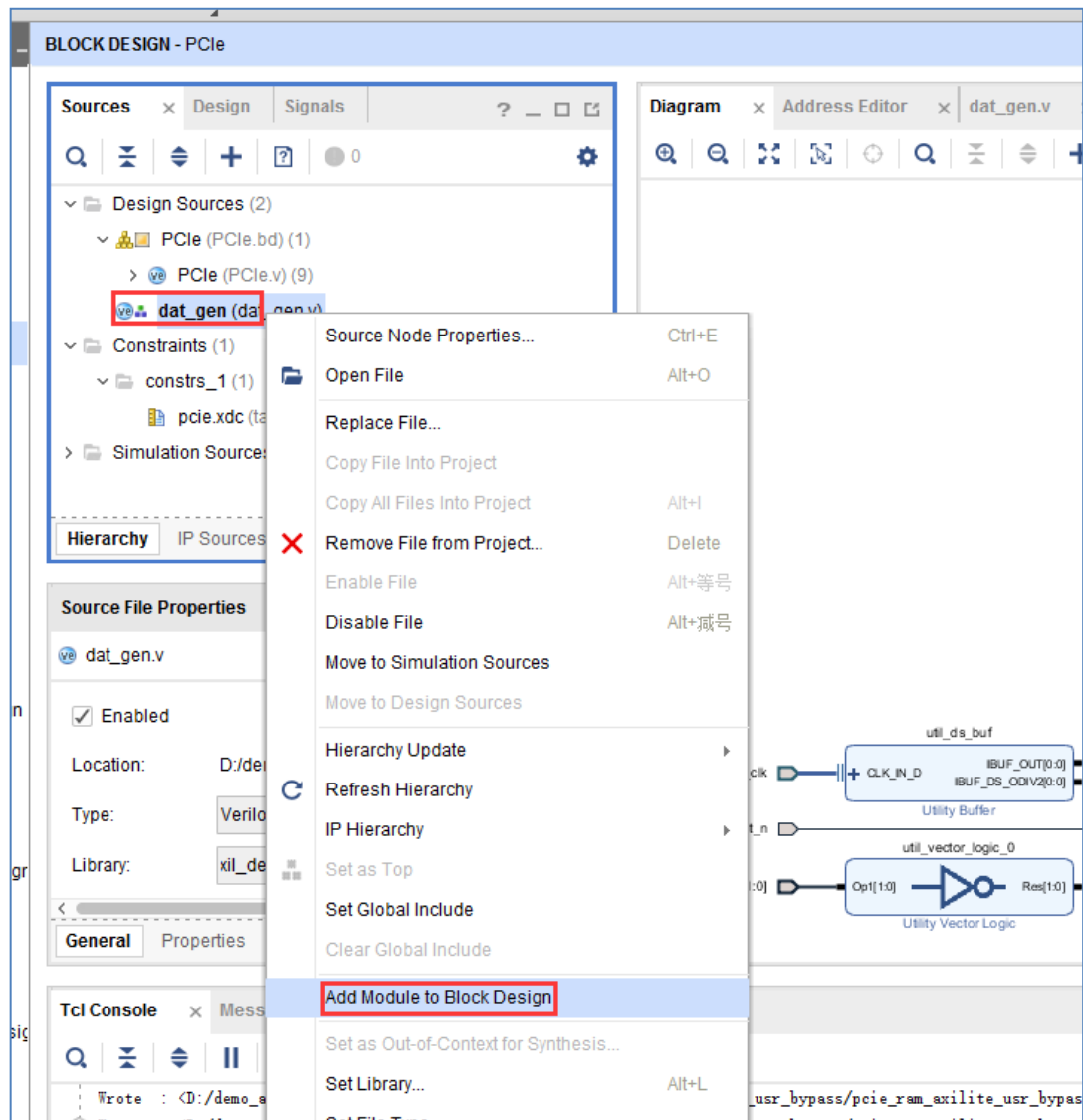


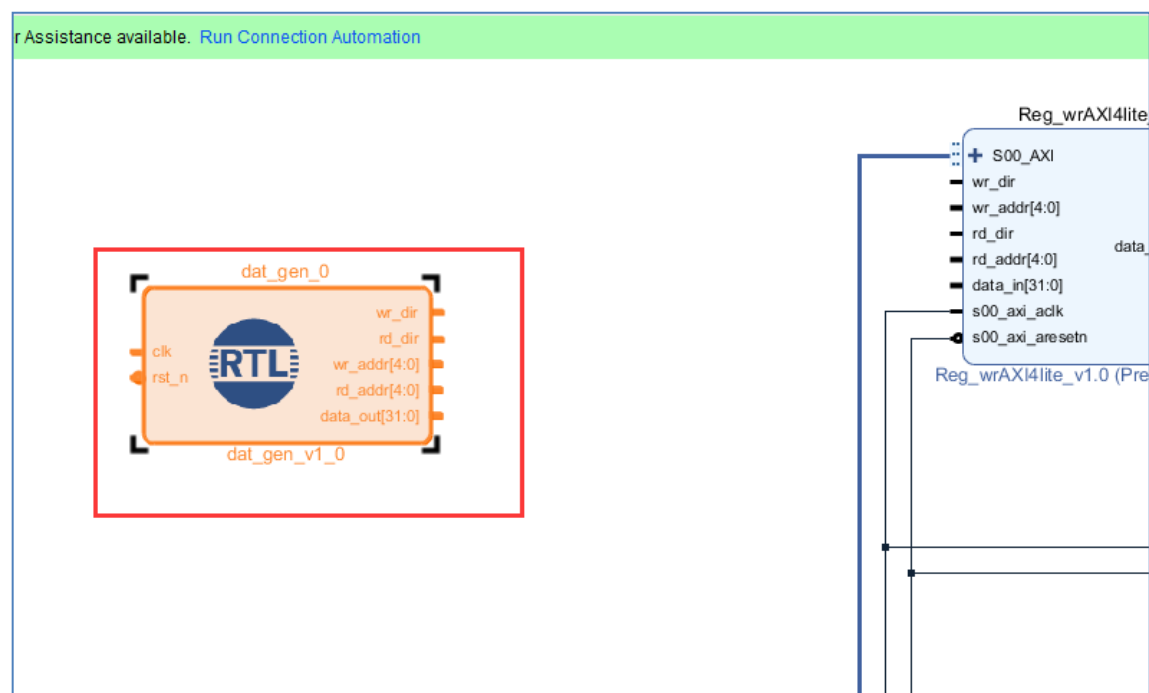
dat_gen_0 模块:

新建 dat_gen.v 文件并添加到工程中如下图，内容这里不做介绍:



然后右击文件选择“Add Module to Block Design”即可添加到 PCIe.bd 中，如下图:





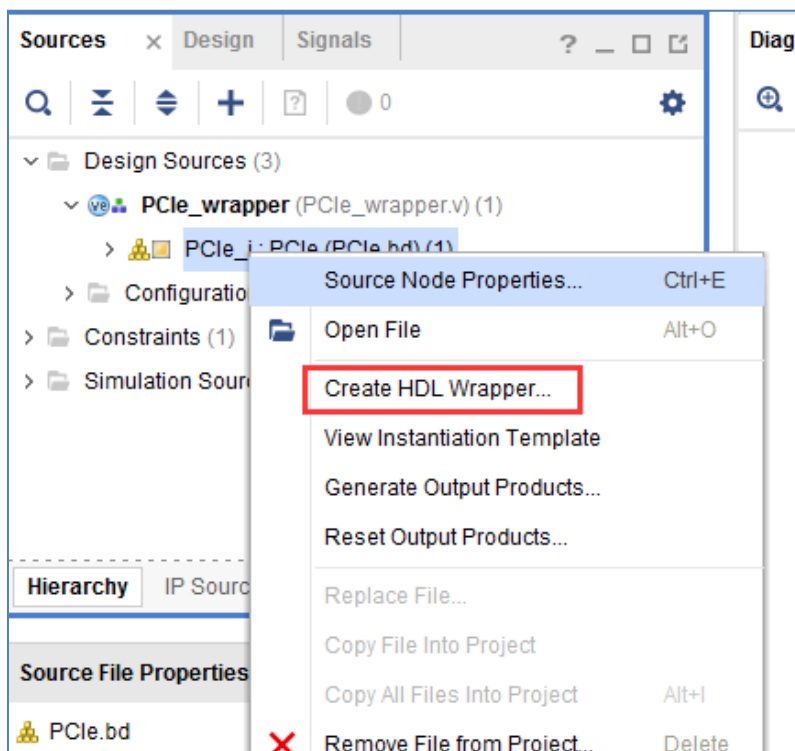
再把相应的线连接起来。

其它模块比较简单按工程设置设置即可，这里不做介绍。

然后在如下图中对各 AXI 总线模块的映射地址进行如下分配：

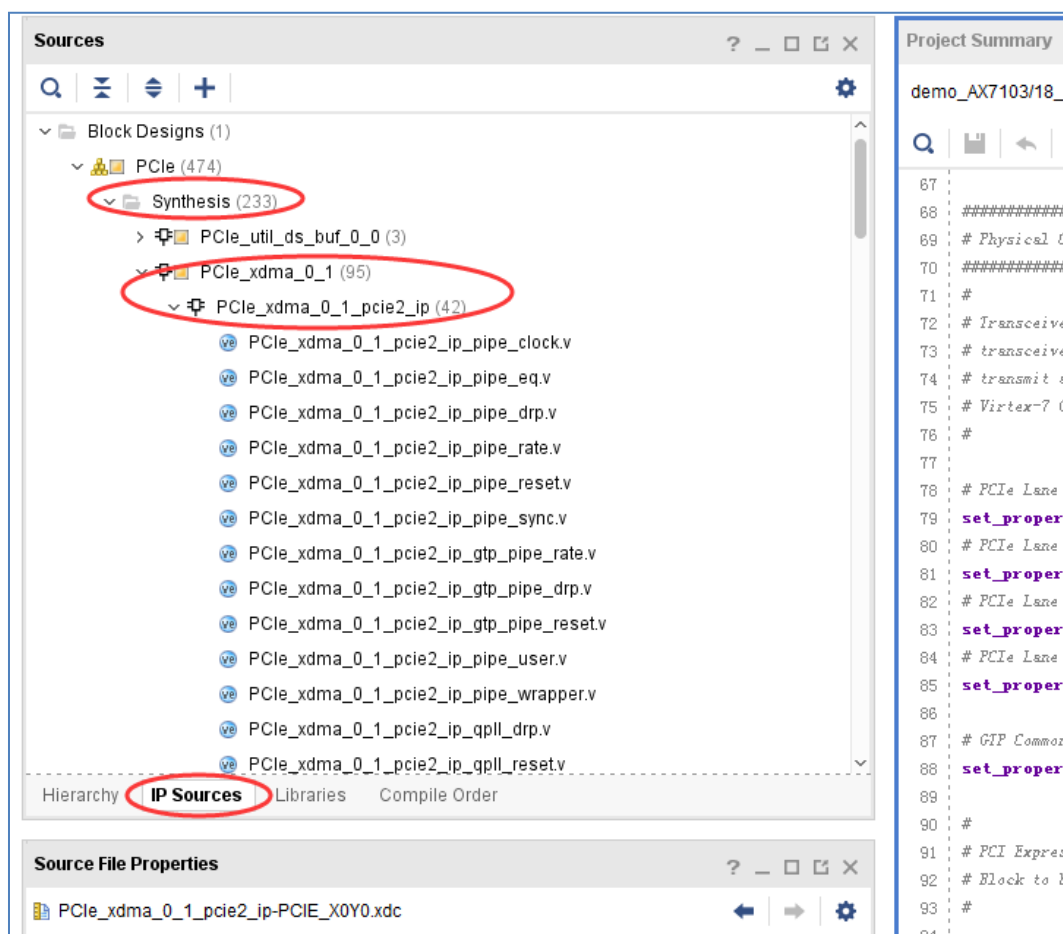
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
xdma_0					
M_AXI (64 address bits : 16E)					
axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0000_0000	8K	0x0000_0000_0000_1FFF
M_AXI_LITE (32 address bits : 4G)					
Reg_wrAXI4lite_0	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
M_AXI_BYPASS (64 address bits : 16E)					
axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0000_0000	8K	0x0000_0000_0000_1FFF

完成上述步骤后，右键单击 PCIe.bd 模块选择如下图中“Create HDL Wrapper...”的进行顶层文件创建：

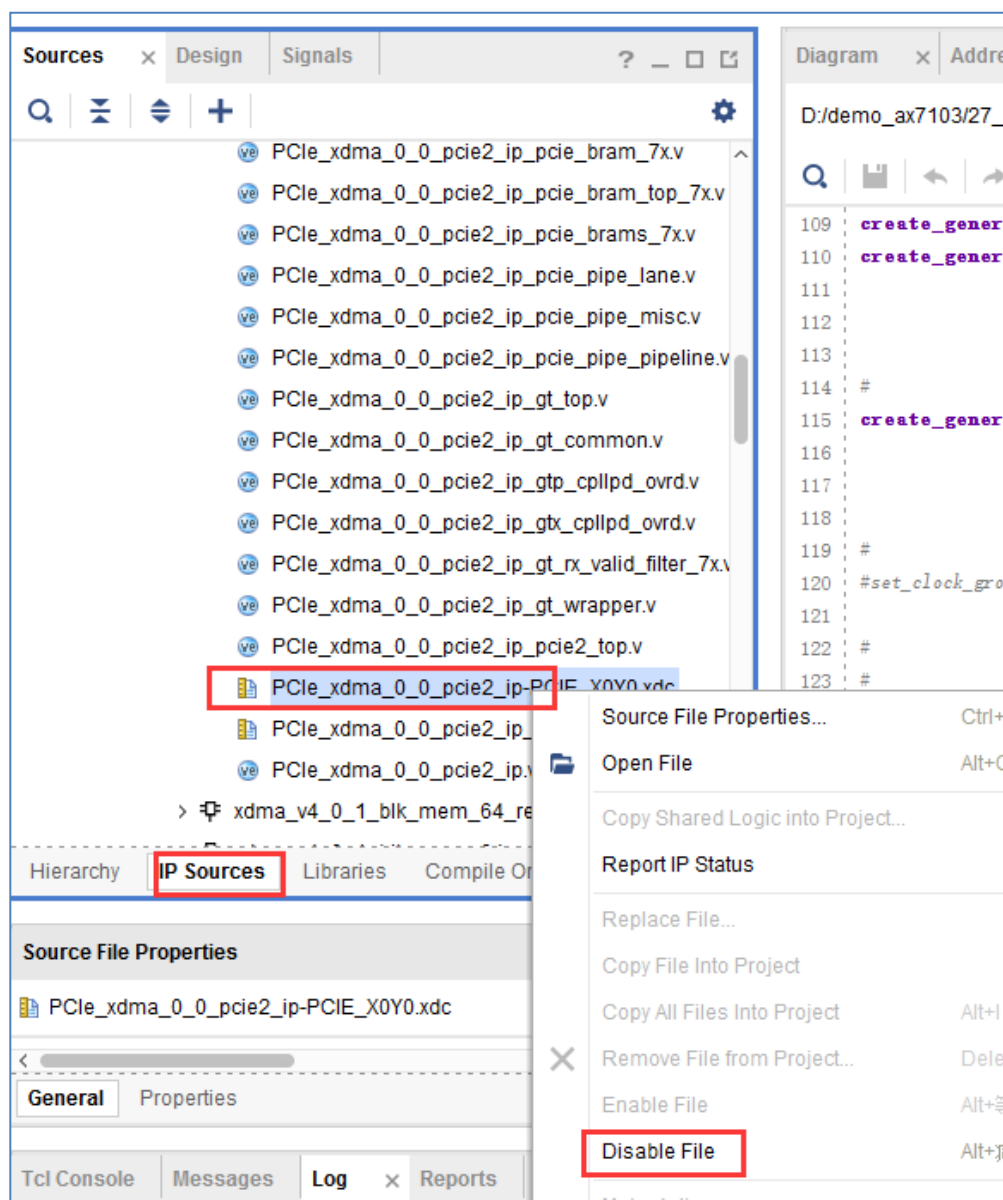


编译综合下载到 AX7103 的 FLASH 之前，还需做最后一个步工作，修改 XDMA IP 的 PCIe 管脚分配约束，该约束说明适应于 AX7103 开发板的 PCIe 所有例程。参考如下：

由于 vivado 软件生成的 XDMA IP PCIe 管脚约束是默认的，与 AX7103 开发板的硬件不相符，需自己修改 PCIe 管脚约束。打开 PCIe 相关例程，这里以 PCIe_test 进行说明，按照红色框提示找到 PCIe_xdma IPcore，如下图：



在打开的文件清单中找到下图中红色标注的.xdc PCIe 管脚约束文件，并“Disable File”,在顶层的 XDC 文件中进行约束。




修改完成后进行编译综合，然后下载到 AX7103 的 FLASH 中。把开发板插入计算机 PCIe 插槽（断电操作）。

3.2 PCIe 驱动安装

驱动安装这里不做介绍，如已安装请忽略，否则请参考教程《PCIe 速度测试例程》相关驱动安装这一章节的内容。

3.3 上位机测试程序

上位机程序采用 XILINX 提供的 XDMA 的 WINDOW 测试程序进行测试，资料目录在“01_学习教程之例程篇\demo\xilinx PCIe 驱动相关资料\Windows 驱动源文件”下，如下图：

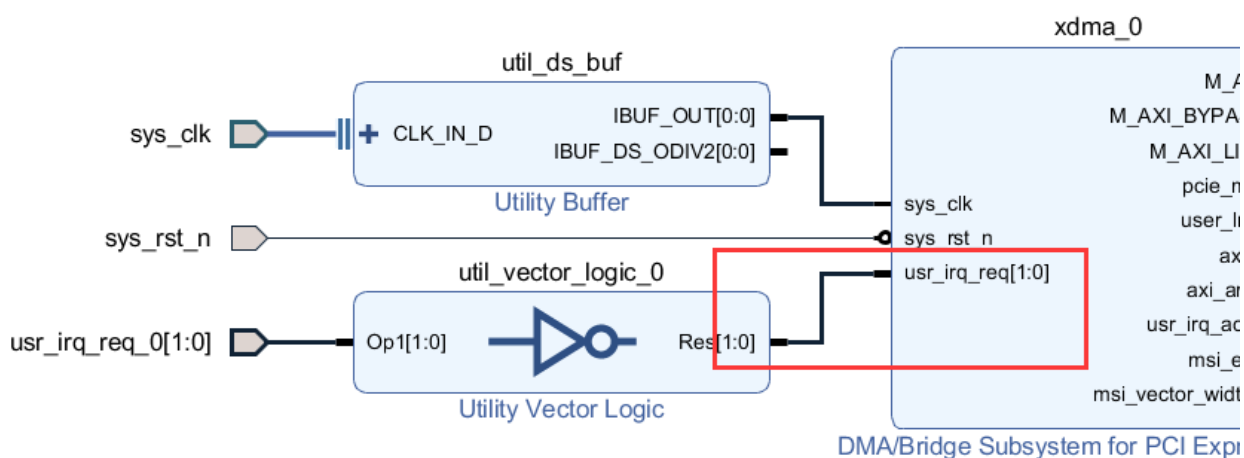
01_学习教程之例程篇 > demo > xilinx PCIe驱动相关资料 > Windows驱动源文件 >			
名称	修改日期	类型	大小
 xdma_driver_win_src_2017_4	2018/8/16 16:32	ZIP 文件	77 KB

程序的打开软件为 VS2015 以上的版本，官方采用的是 VS2015。

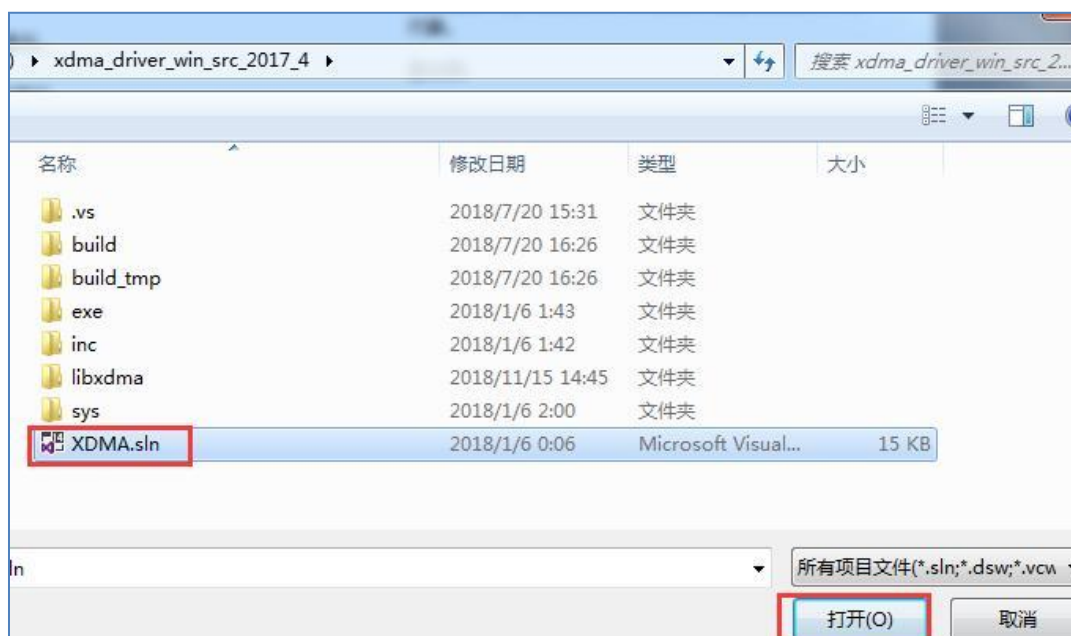
4 实验现象

接下来对例程的功能进行测试，测试之前先确认 PCIe 板卡已插入到电脑的 PCIe 插槽（务必断电插入）：

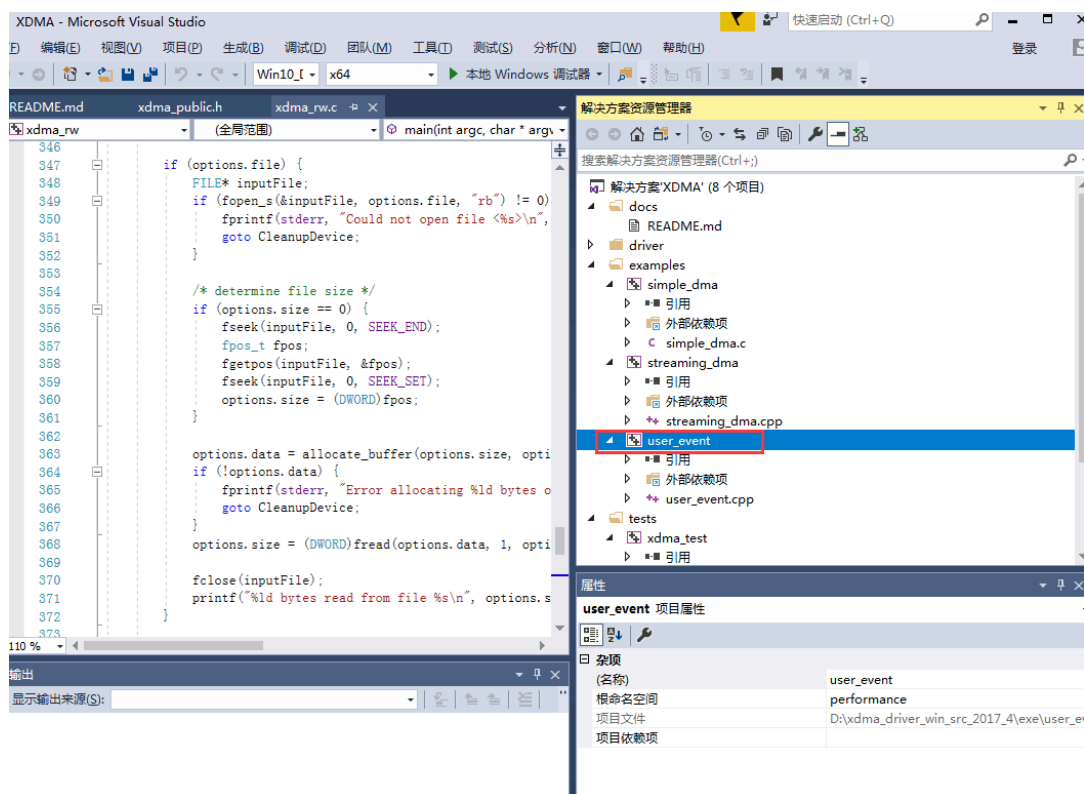
1) 用户事件响应测试 也就是如下图中 FPGA 中程序中标红的这一部分，功能为用户按下开发板上的 KEY1 和 KEY2 键，这个用户事件会通过 XDMA 的 PCIe 发送端口发送信号，上位机接收到这个用户事件后可以进行相应的动作，这里上位机的功能只是简单地显示相应的中断号。



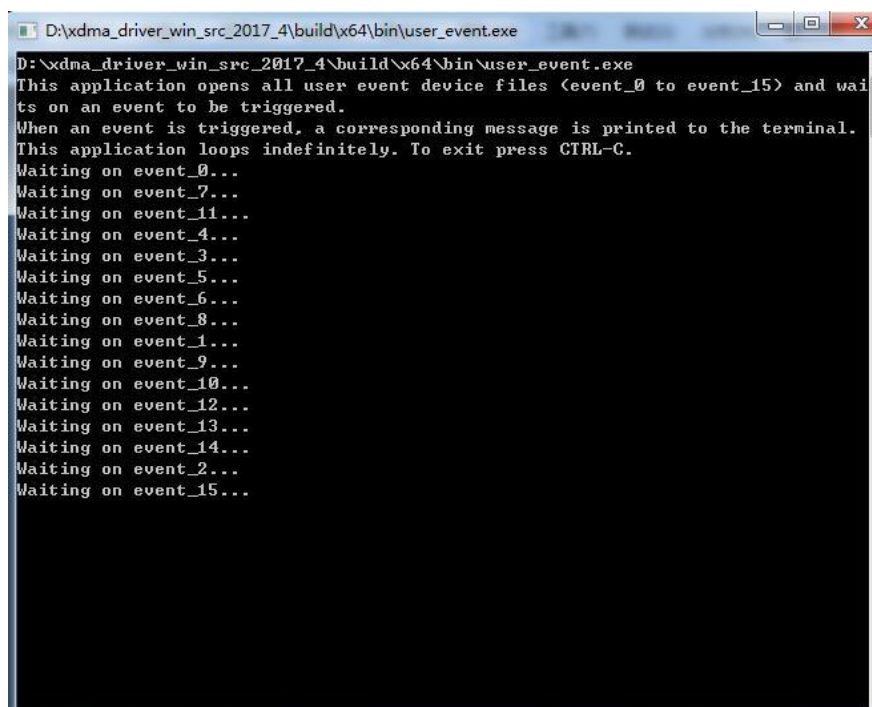
打开上位机程序：



在这个界面下选择 user_event 工程，VS 开发平台方面的问题这里不做支持。

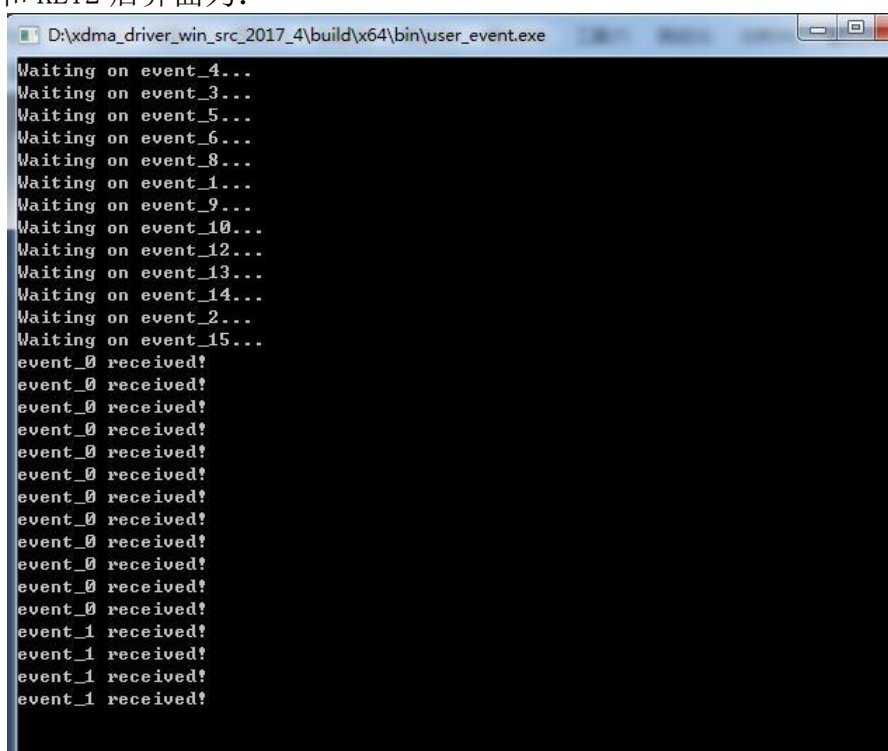


运行后测试界面如下：



```
D:\xdma_driver_win_src_2017_4\build\x64\bin\user_event.exe
D:\xdma_driver_win_src_2017_4\build\x64\bin\user_event.exe
This application opens all user event device files (event_0 to event_15) and waits
on an event to be triggered.
When an event is triggered, a corresponding message is printed to the terminal.
This application loops indefinitely. To exit press CTRL-C.
Waiting on event_0...
Waiting on event_7...
Waiting on event_11...
Waiting on event_4...
Waiting on event_3...
Waiting on event_5...
Waiting on event_6...
Waiting on event_8...
Waiting on event_1...
Waiting on event_9...
Waiting on event_10...
Waiting on event_12...
Waiting on event_13...
Waiting on event_14...
Waiting on event_2...
Waiting on event_15...
```

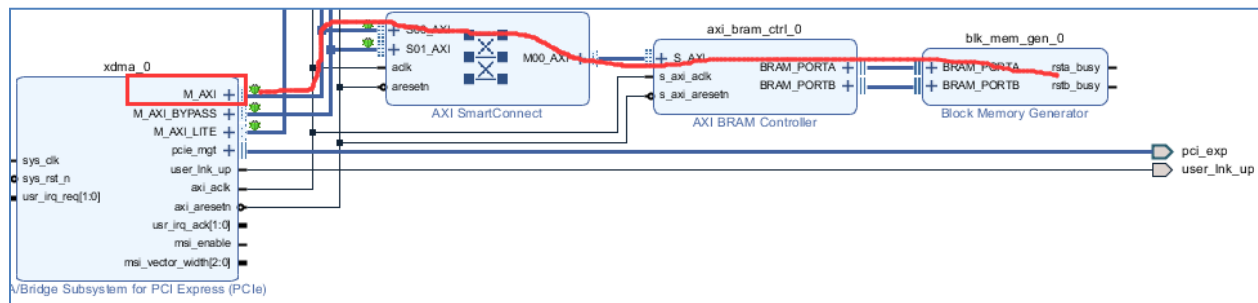
按下 KEY1 和 KEY2 后界面为:



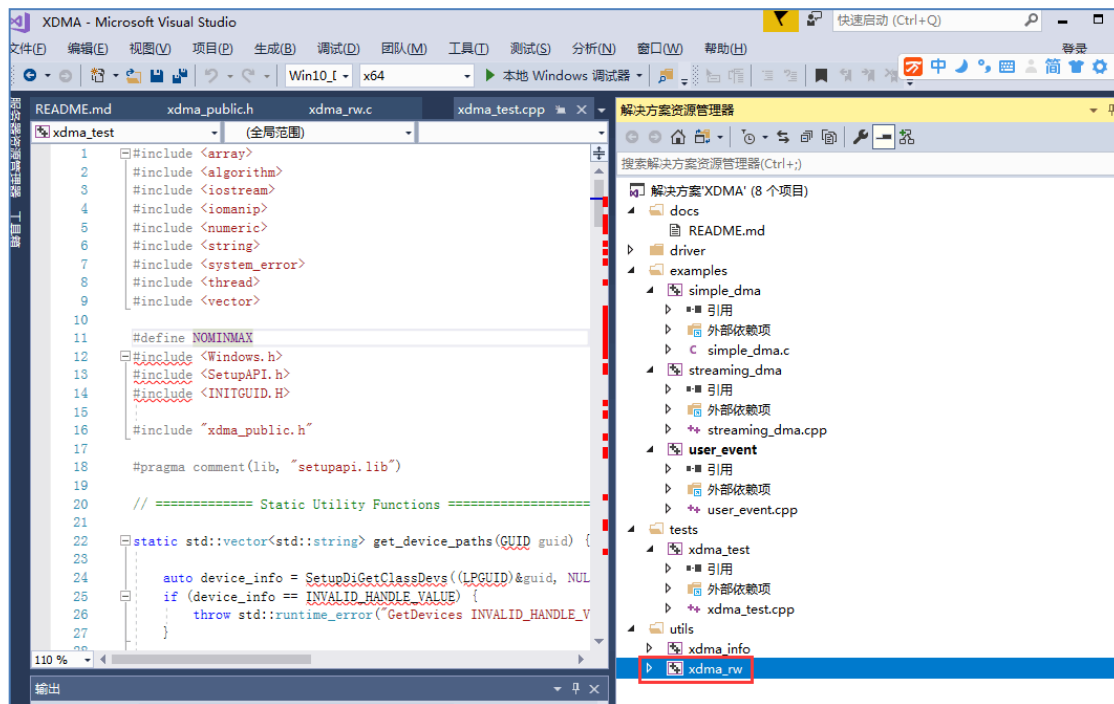
```
Waiting on event_4...
Waiting on event_3...
Waiting on event_5...
Waiting on event_6...
Waiting on event_8...
Waiting on event_1...
Waiting on event_9...
Waiting on event_10...
Waiting on event_12...
Waiting on event_13...
Waiting on event_14...
Waiting on event_2...
Waiting on event_15...
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_0 received!
event_1 received!
event_1 received!
event_1 received!
event_1 received!
```

注意：由于按键直接相连，没有进行防抖处理，所以会出现界面中连续触发的情况，有兴趣的可以结合开发板的防抖程序试试效果。

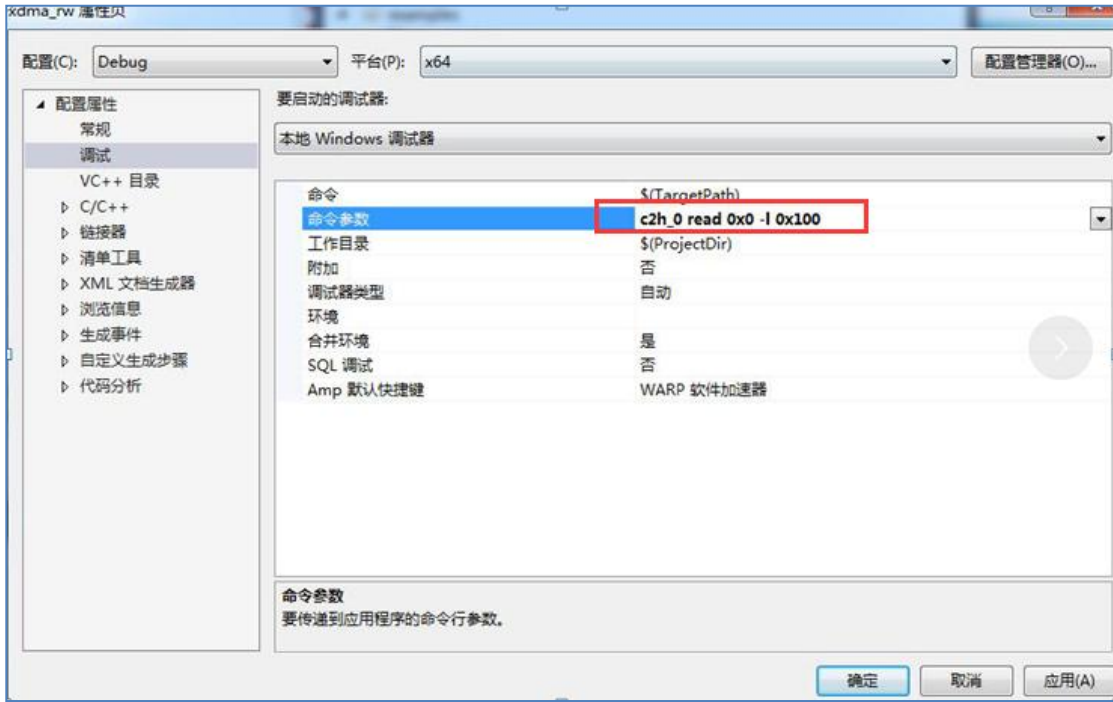
2) M_AXI 接口测试 也就是如下图中 FPGA 中程序中标红的这一部分，功能是上位机程序通过 PCIe 写入一定的数据到 FPGA 的 RAM 中，在 FPGA 中调试窗口中可以看到写入的数据，同时通过读取命令可以读出相应数据。



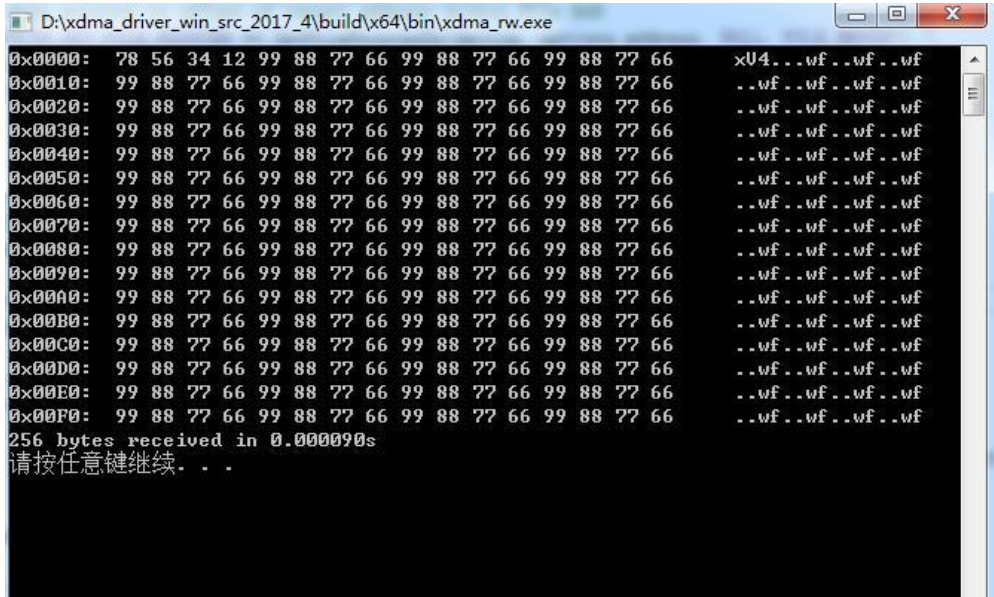
上位机选择 xdma_rw 工程



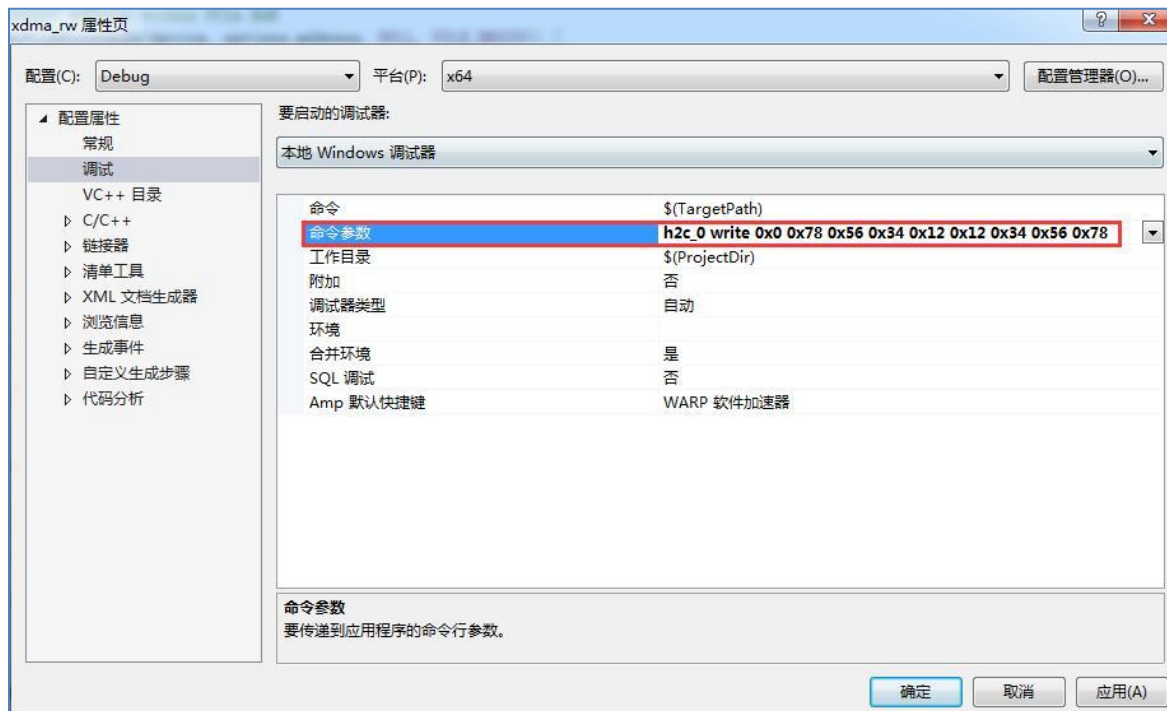
先进行读 RAM 测试，从 RAM 的 0 地址起读取 256Bytes，界面设置为：



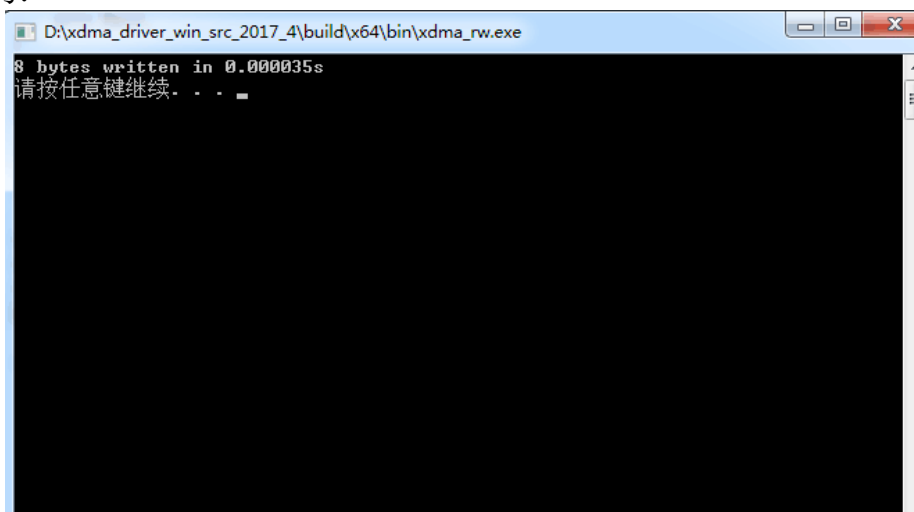
运行后测试界面如下：



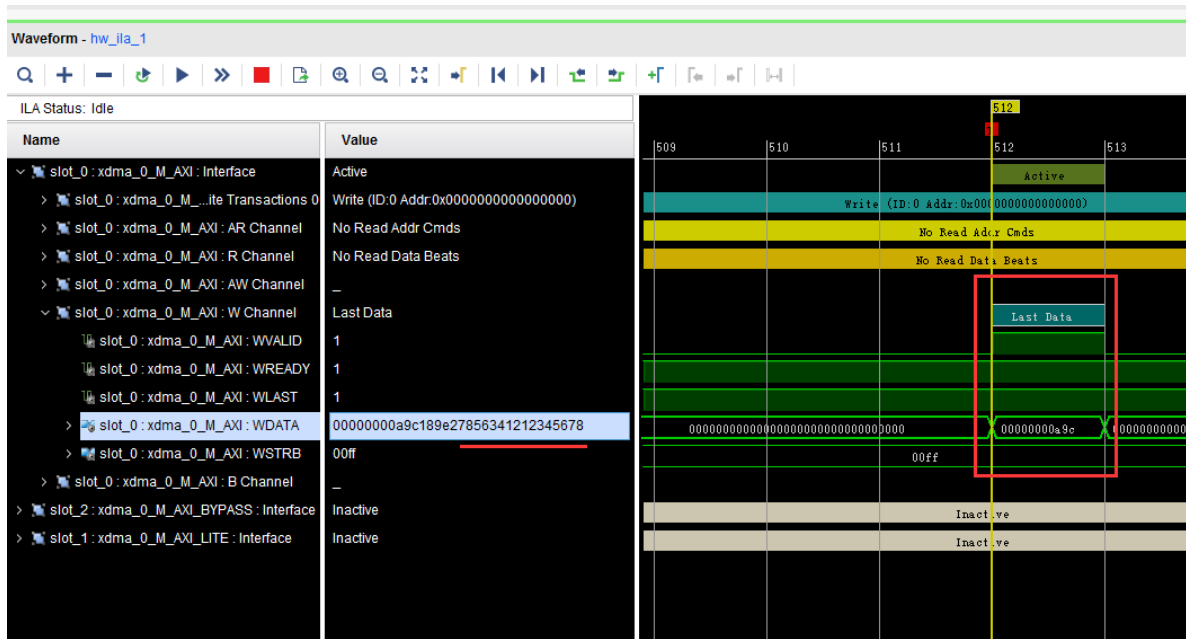
然后进行写 RAM 测试，从 RAM 的 0 地址起写 8Bytes（0x78 0x56 0x34 0x12 0x12 0x34 0x56 0x78），界面设置为：



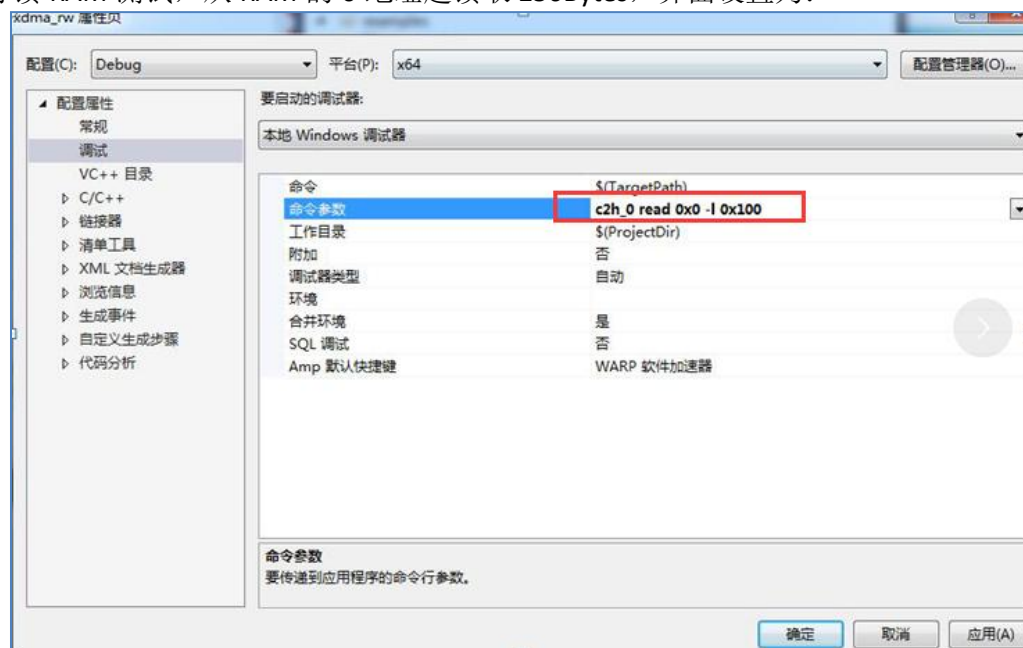
运行的界面为:



如果同时进行调试，可以在 VIVADO 调试窗口看到如下图所示数据，是上位机写入的数据，低 8Bytes 有效。



再进行读 RAM 测试，从 RAM 的 0 地址起读取 256Bytes，界面设置为：



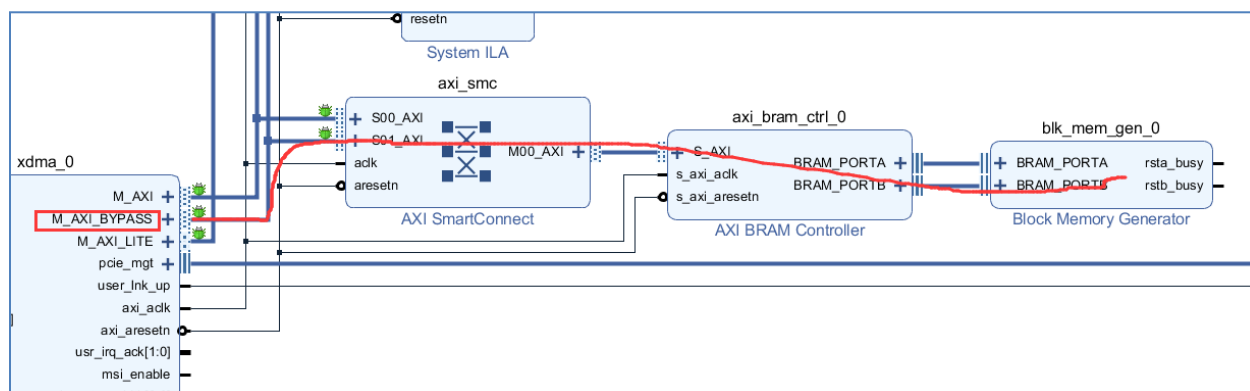
运行后可以看到如标红的所示 PCIe 上位机所写的内容，与写的内容一致：

```

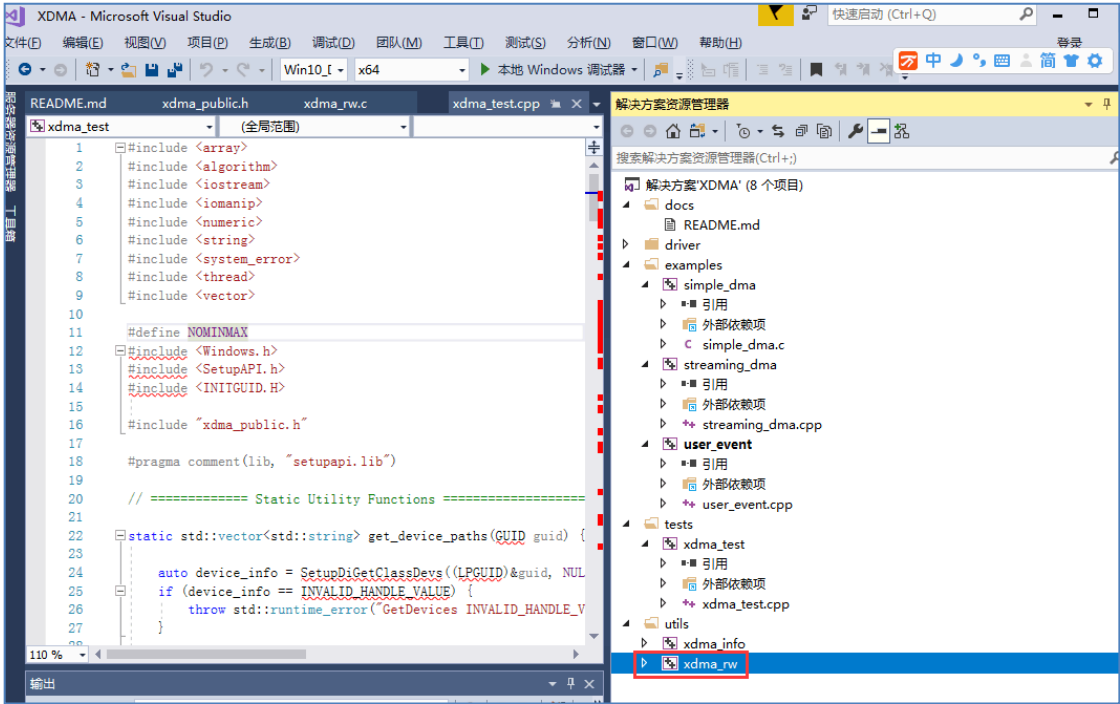
D:\xdma_driver_win_src_2017_4\build\x64\bin\xdma_rw.exe
0x0000: 78 56 34 12 12 34 56 78 99 88 77 66 99 88 77 66 x04. .4Ux. .wf. .wf
0x0010: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0020: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0030: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0040: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0050: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0060: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0070: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0080: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x0090: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00A0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00B0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00C0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00D0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00E0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
0x00F0: 99 88 77 66 99 88 77 66 99 88 77 66 99 88 77 66 . .wf. .wf. .wf. .wf
256 bytes received in 0.000027s
请按任意键继续. . .

```

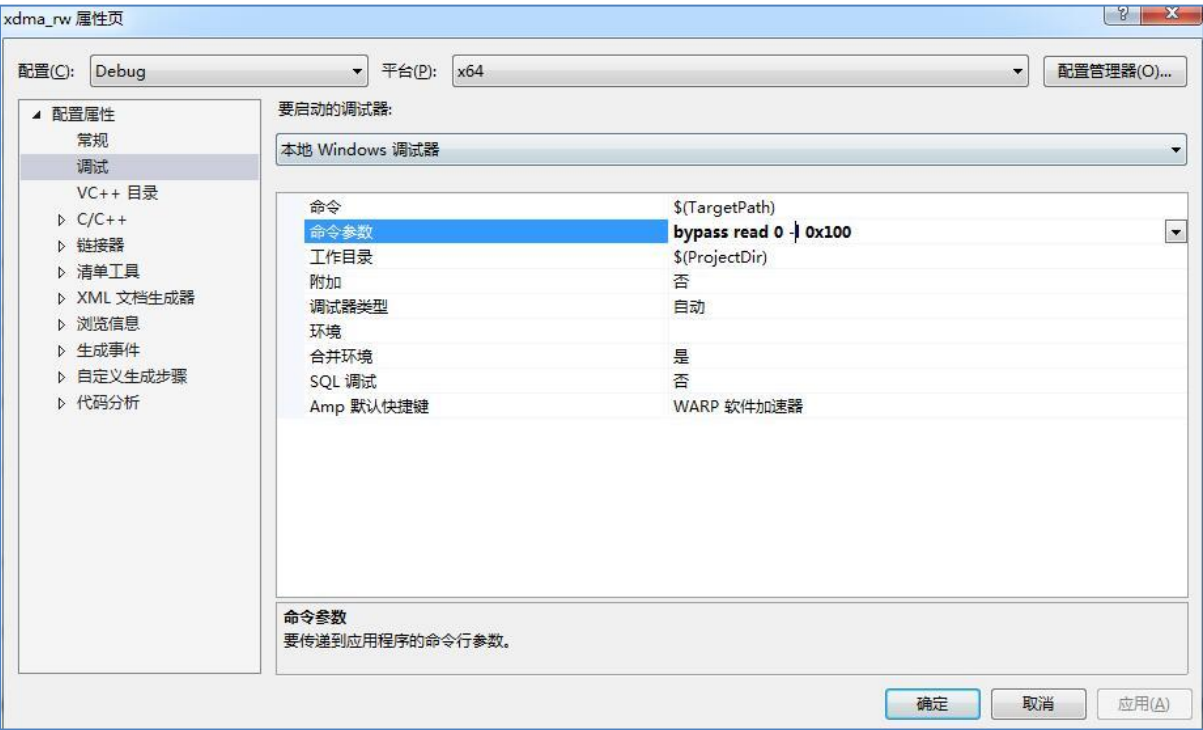
3) M_AXI_BYPASS 接口测试 也就是如下图中 FPGA 中程序中标红的这一部分，功能是上位机程序通过 PCIe 写入或读出一定的数据，然后在 FPGA 中通过 M_AXI_BYPASS 接口写入或读取 RAM 中的数据，在 FPGA 中调试窗口中可以看到写入的数据，同时通过读取命令可以读出相应数据。数据传输不需要 DMA 功能时可采用此接口。



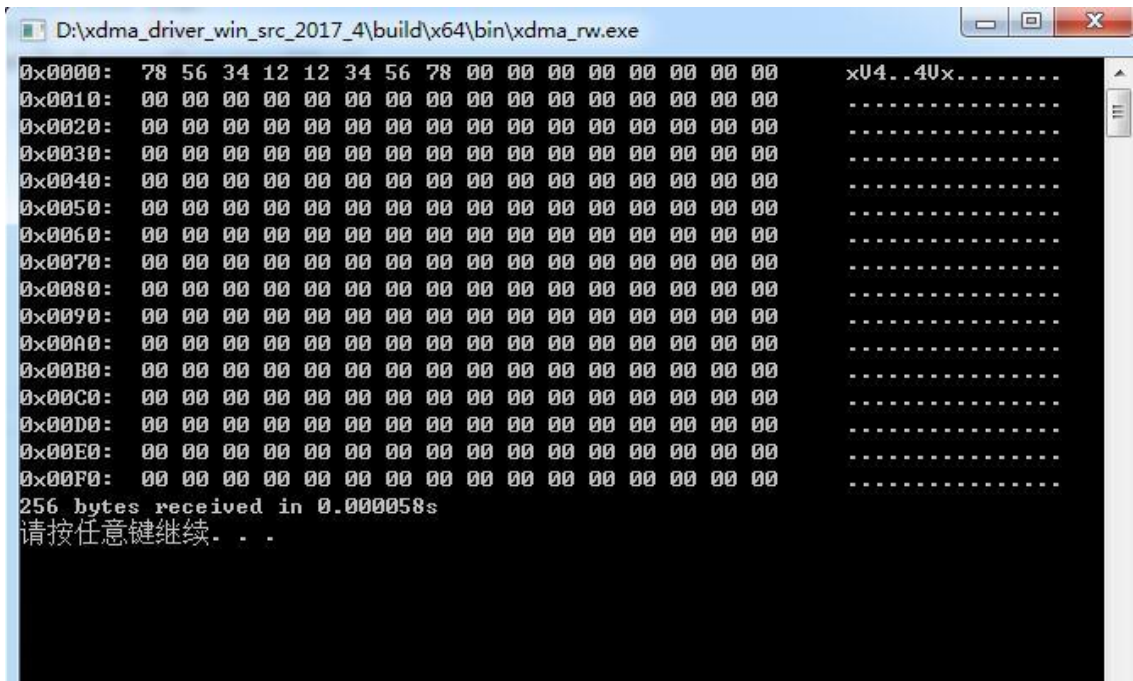
上位机选择 xdma_rw 工程



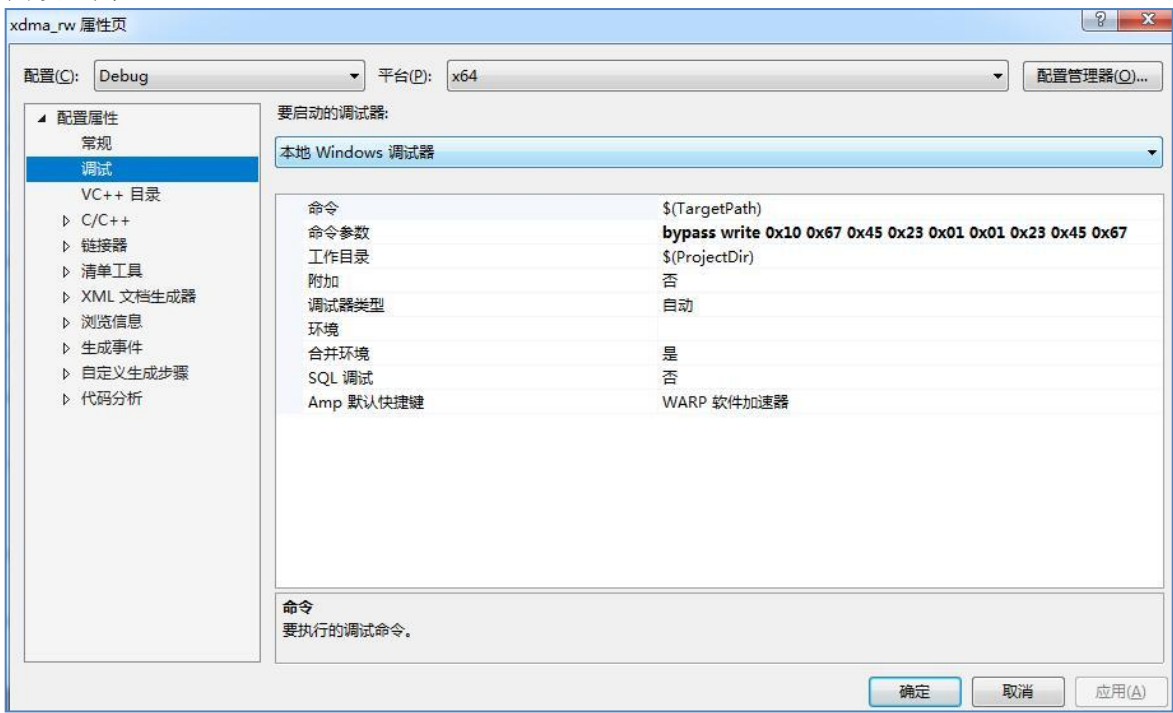
先进行读 RAM 测试，从 RAM 的 0 地址起读取 256Bytes，界面设置为：



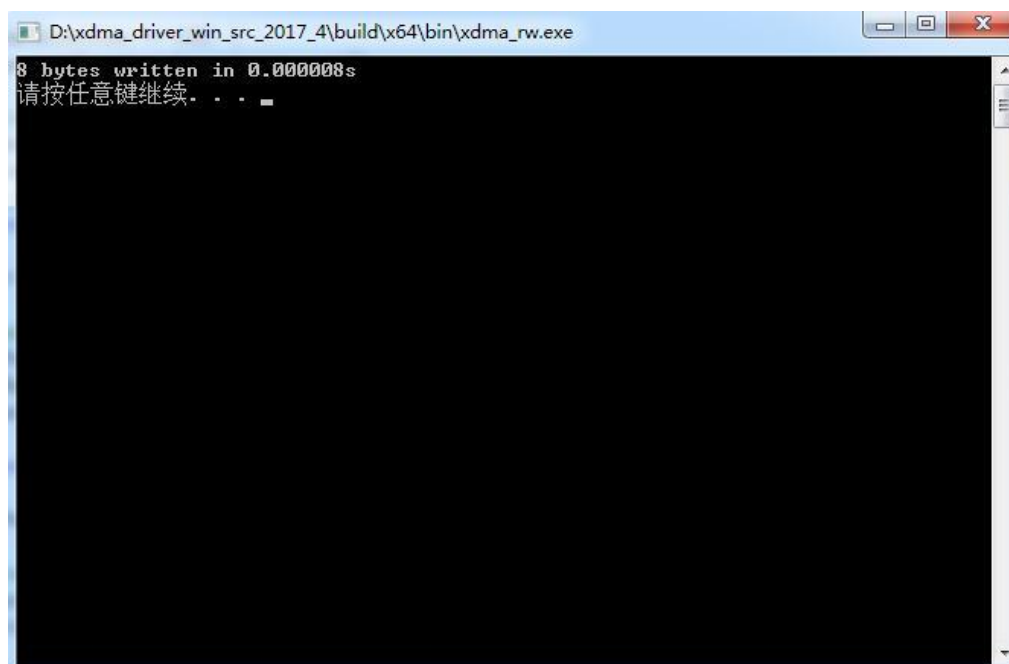
运行后测试界面如下：



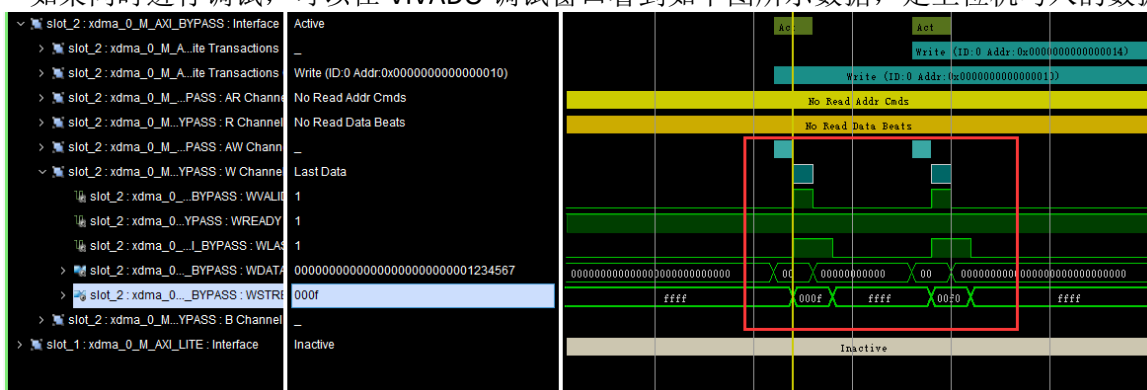
进行写 RAM 测试，从 RAM 的 0x10 地址起写 8Byte,(0x67 0x45 0x23 0x01 0x01 0x23 0x45 0x67)，
界面设置为：



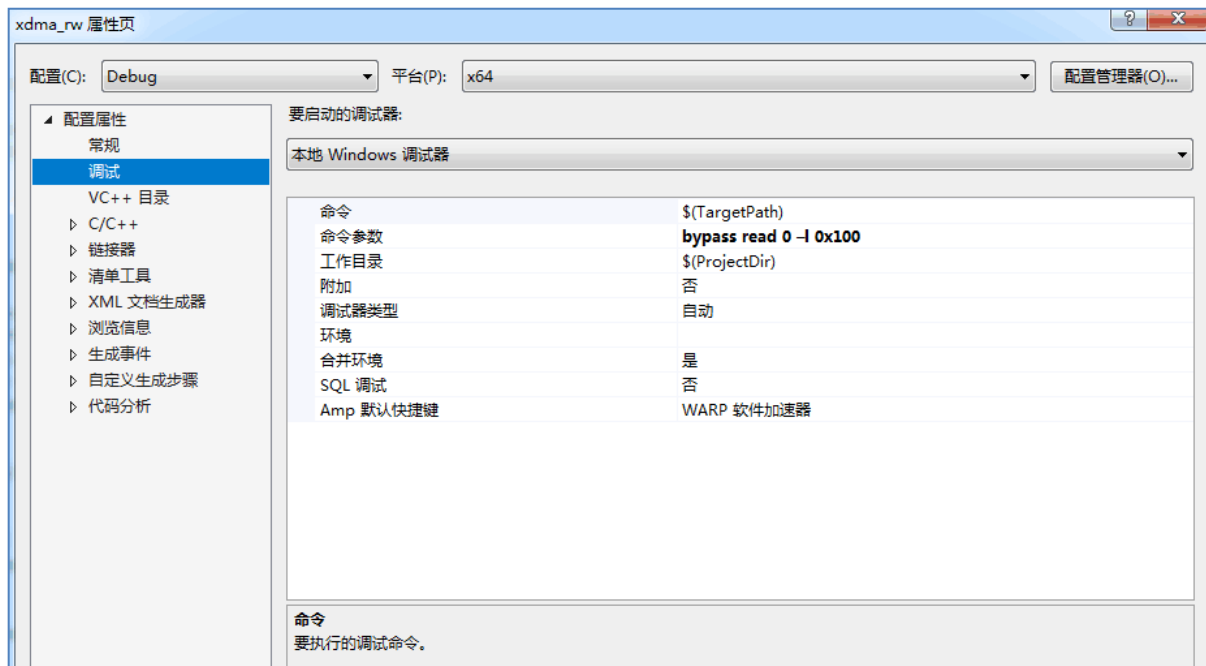
运行的界面为：



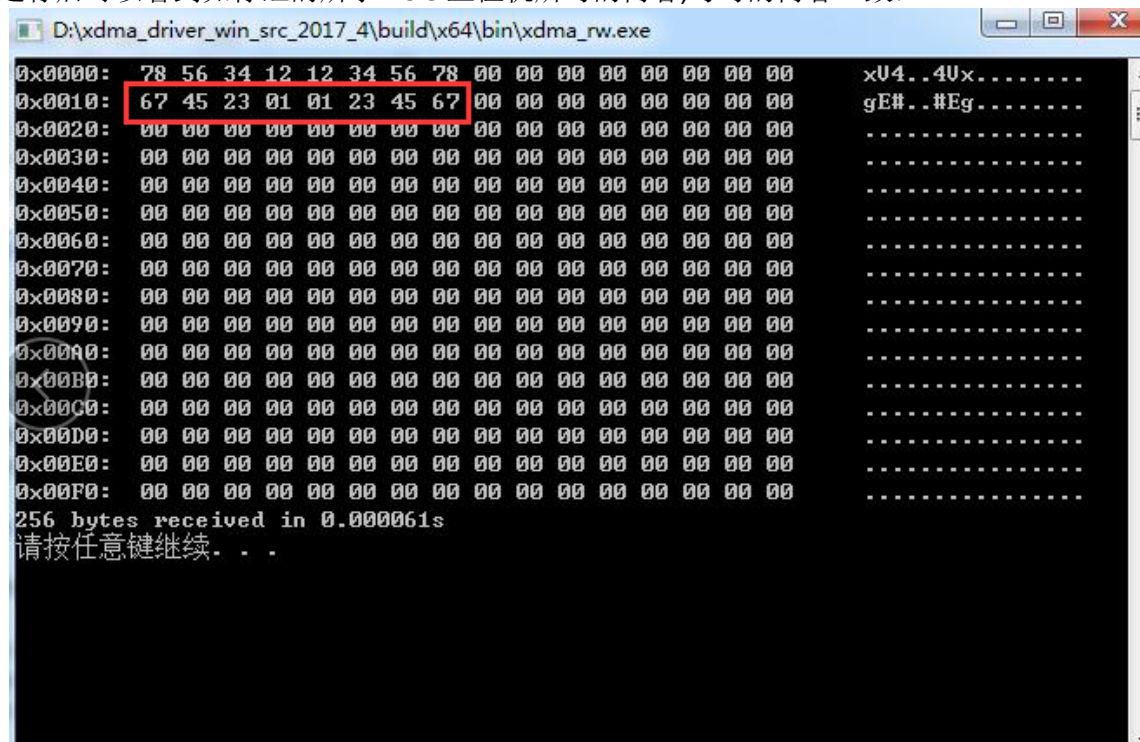
如果同时进行调试，可以在 VIVADO 调试窗口看到如下图所示数据，是上位机写入的数据。



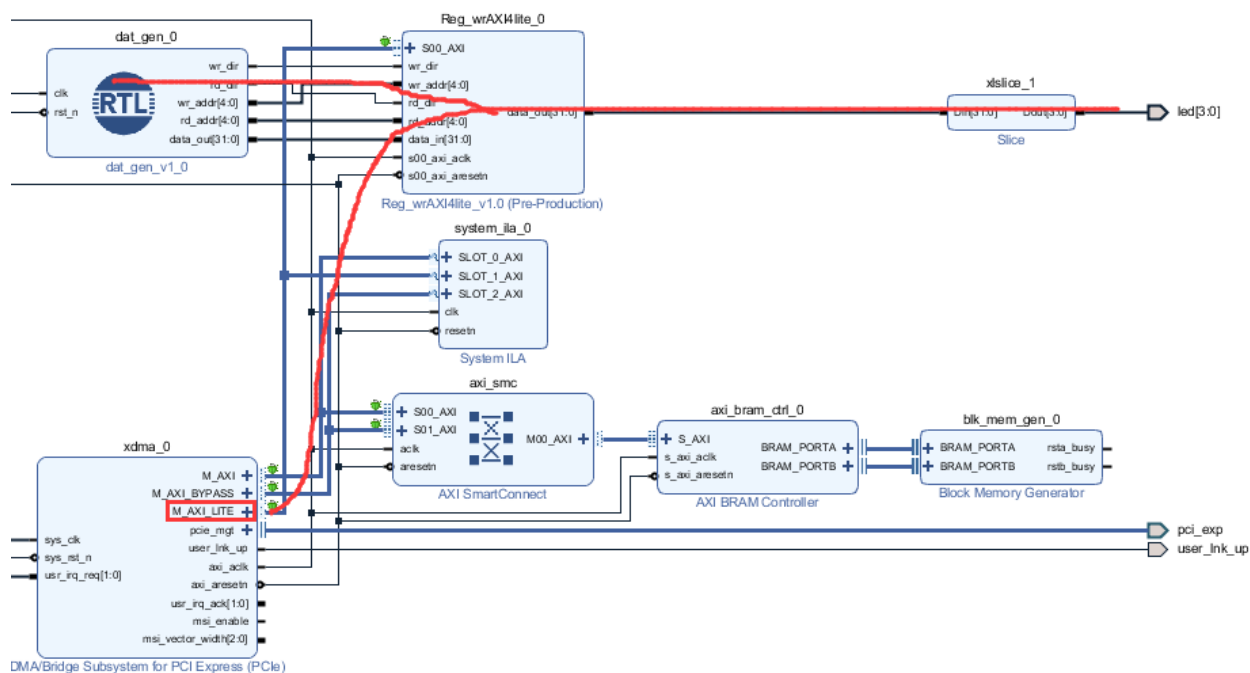
再进行读 RAM 测试，从 RAM 的 0 地址起读取 256Bytes，界面设置为：



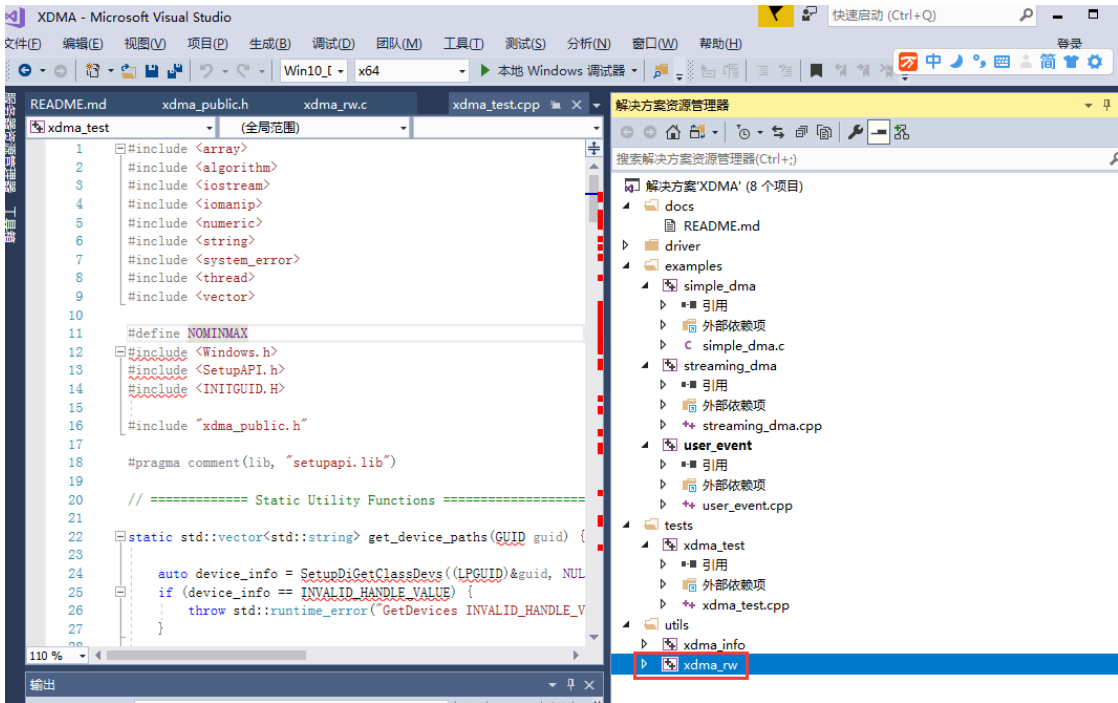
运行后可以看到如标红的所示 PCIe 上位机所写的内容,与写的内容一致:



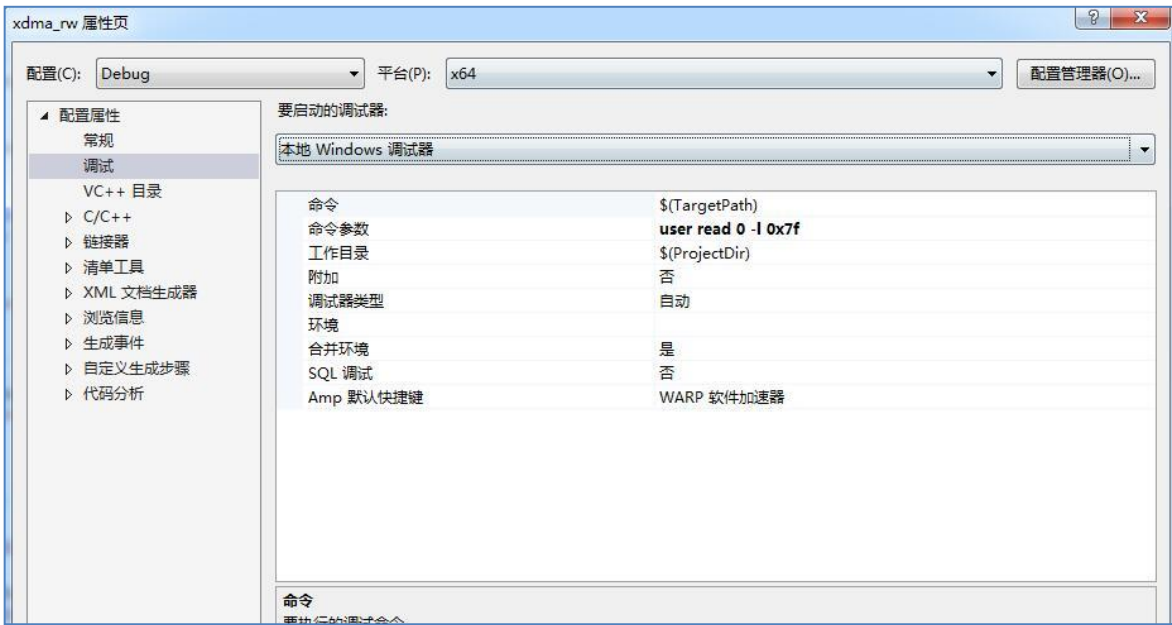
4) M_AXI_LITE 接口测试 也就是如下图中 FPGA 中程序中标红的这一部分，功能是上位机程序通过 PCIe 写入或读出一定的数据，然后 FPGA 通过 M_AXI_LITE 接口写入到 Reg_wrAXI4lite_0 或读取数据。Reg_wrAXI4lite_0 模块功能：定义了 20 个 32 位寄存器，双字地址（wr_addr）0 到 9 为 FPGA 端可写寄存器，双字地址（rd_addr）10 到 19 为 FPGA 端可读寄存器；xlslice_1 用来进行数据宽度变换，在这里连接 LED 灯（LED1、LED2、LED3、LED4），可用上位机控制 LED 状态；dat_gen_0 模块：产生读写 Reg_wrAXI4lite_0 的地址和数据，前 10 个寄存器产生数据为 32'h0000_0000、32'h 1111_1111、32'h2222_2222..... 32'h9999_9999, 后 10 个寄存器地址中在这里只定义了双字地址为 12 的地址进行简单验证，复杂的测试功能自己去完成。此接口主要用于 PCIe 通信时进行自定义寄存器控制。



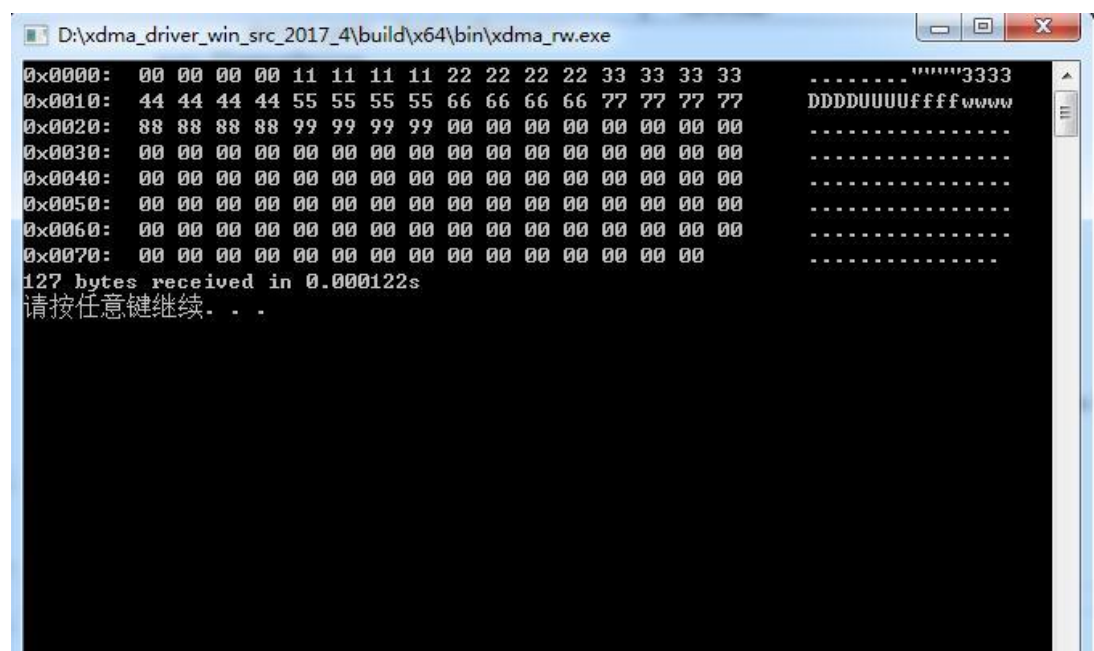
上位机选择 xdma_rw 工程



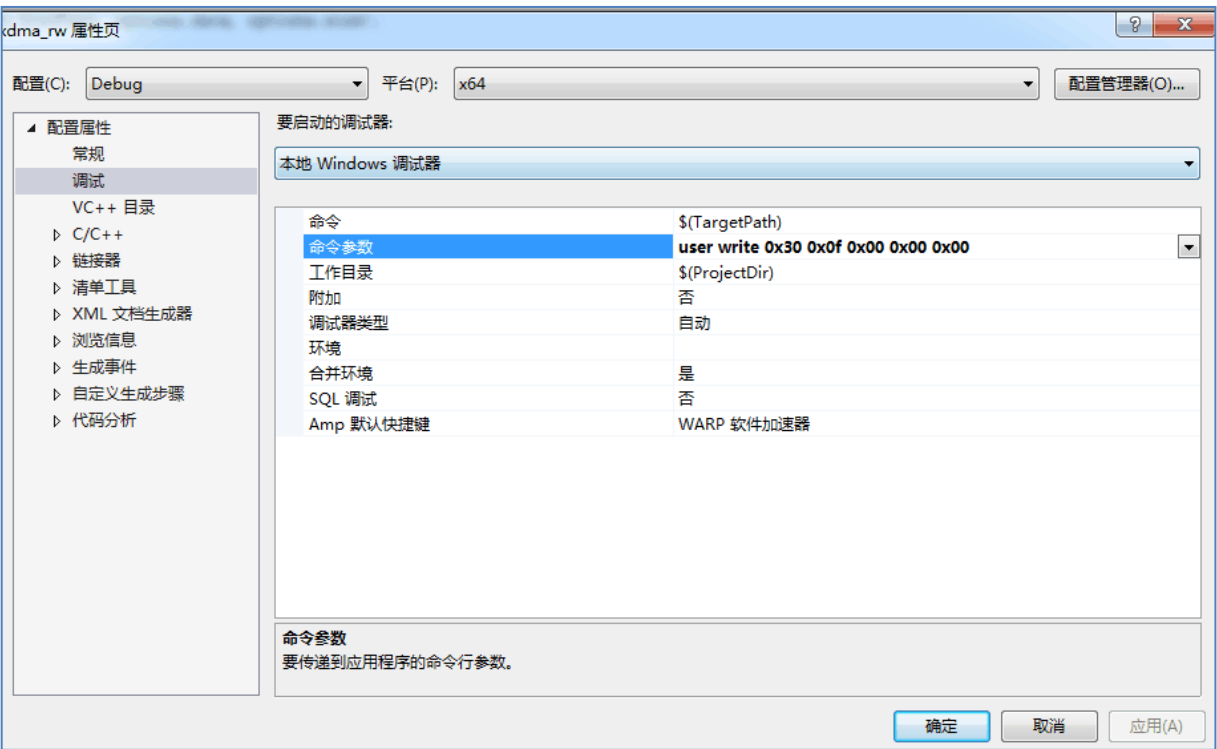
上位机先进行读寄存器测试，从 0 地址起读取 128Bytes，界面设置为：



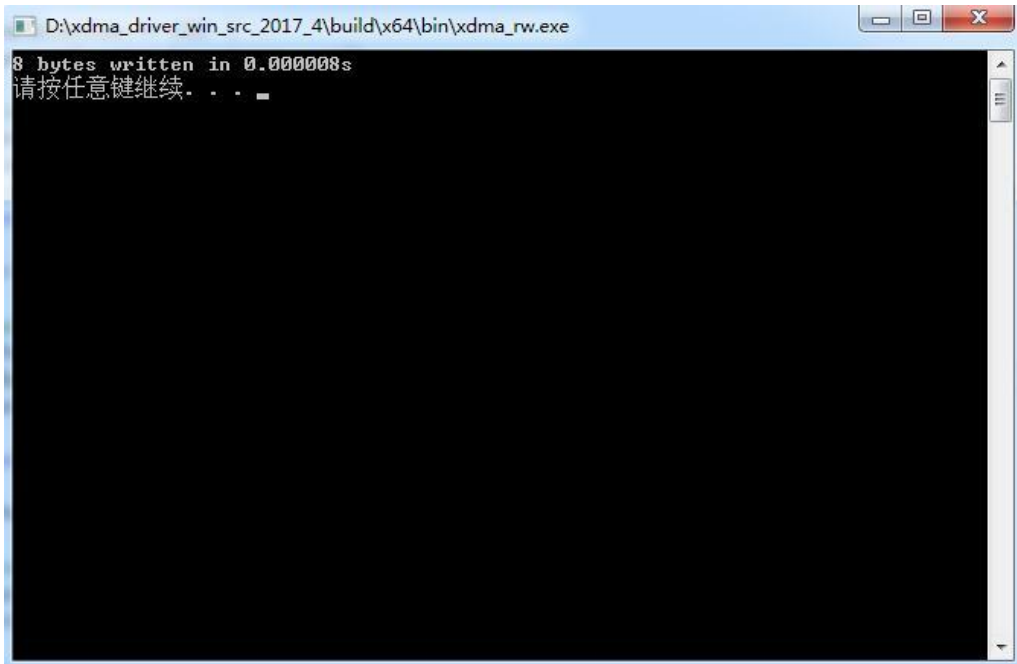
运行后测试界面如下，与测试模块产生的数据一致：



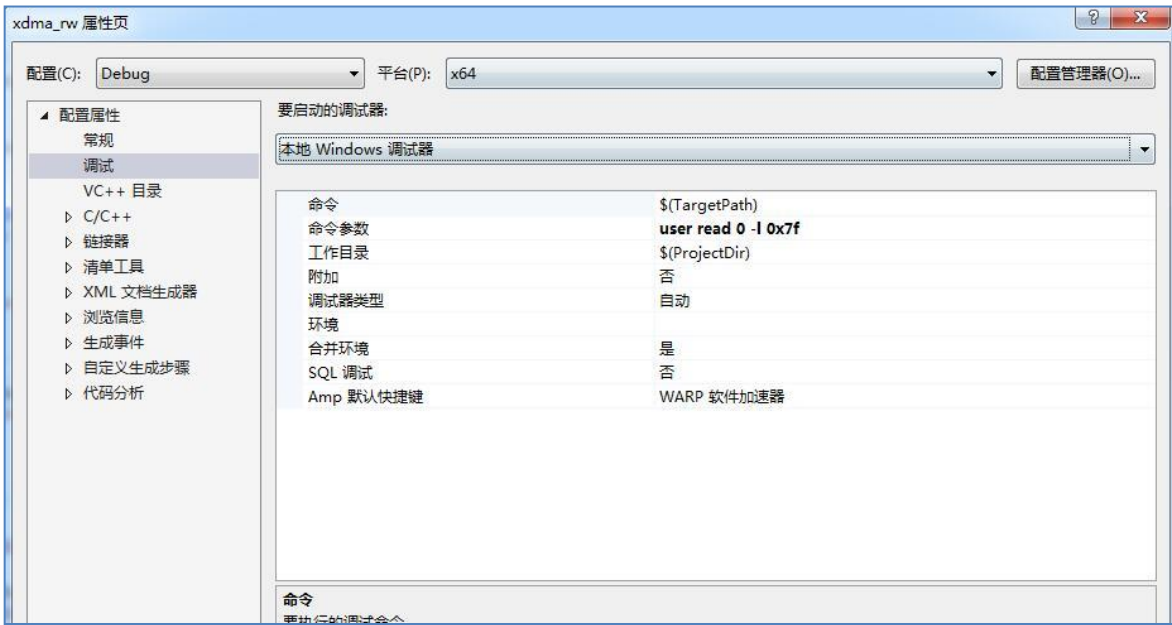
然后再进行写寄存器测试，从字节地址 0x30 起写入 0x0f,可以看到开发板上的 4 个 led 灯（LED1、LED2、LED3、LED4）灭，界面设置为：

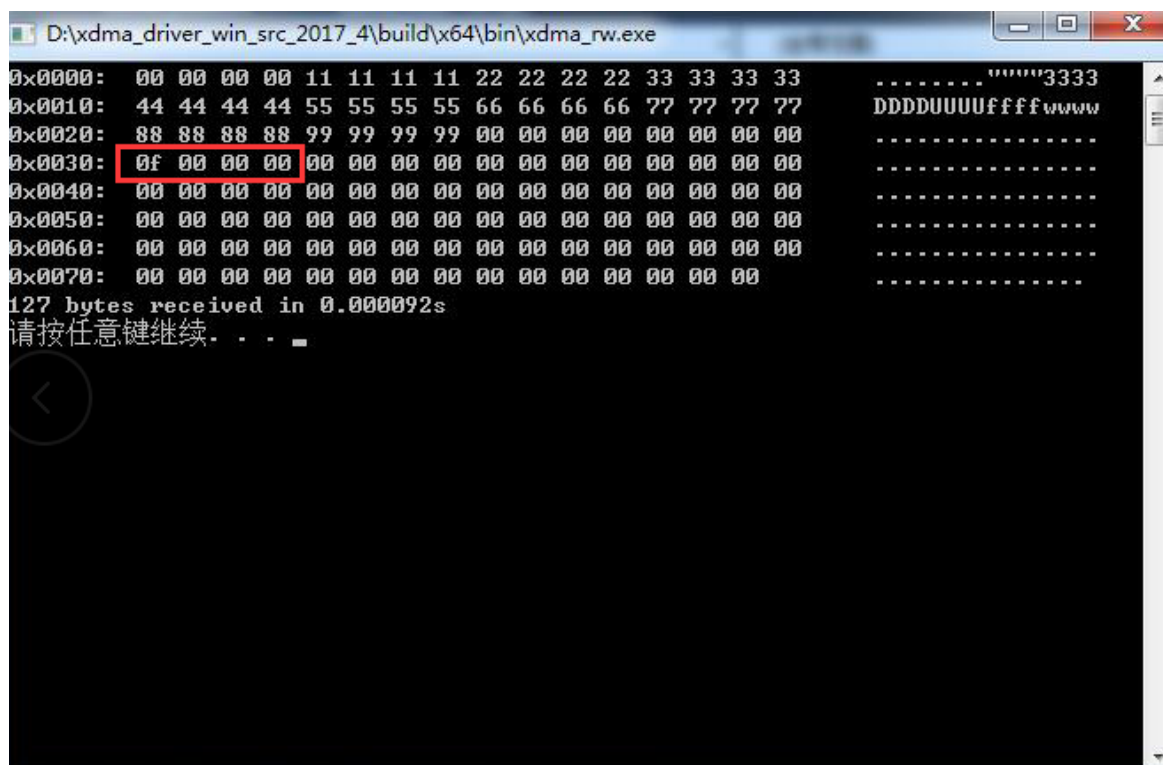


运行的界面为：



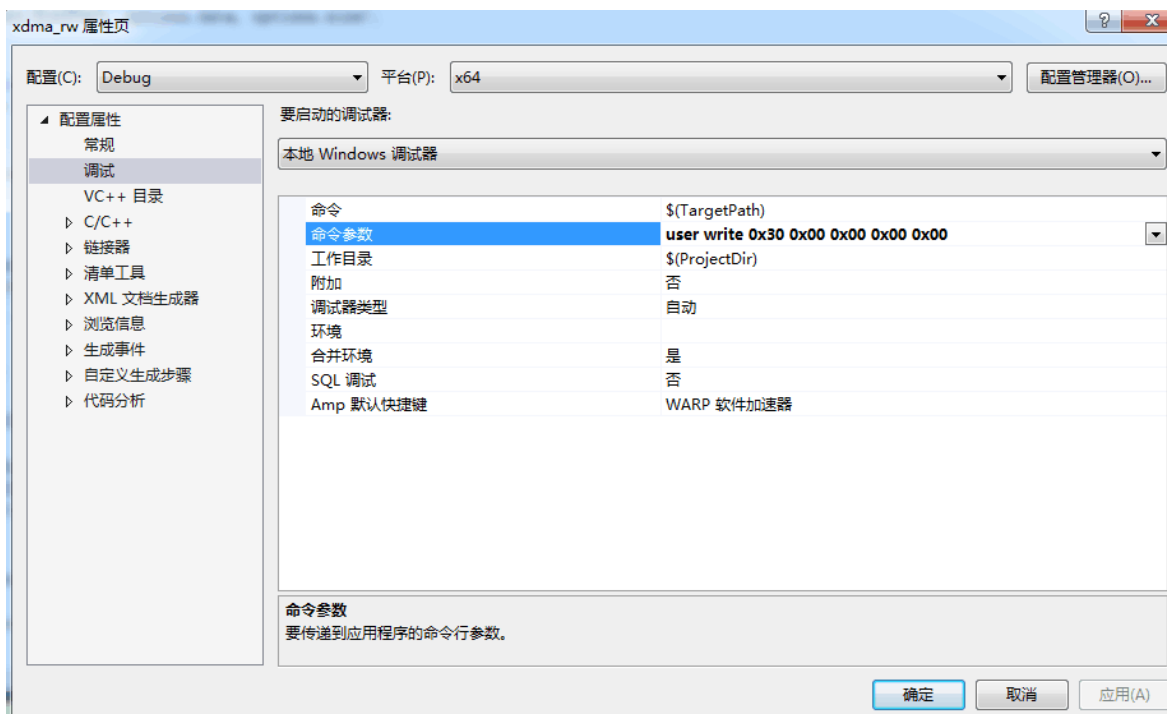
然后看到开发板上的 4 个 LED（LED1、LED2、LED3、LED4）是熄灭的。再进行读寄存器测试，从 0 地址起读取 128Bytes，数据正确，界面设置为：



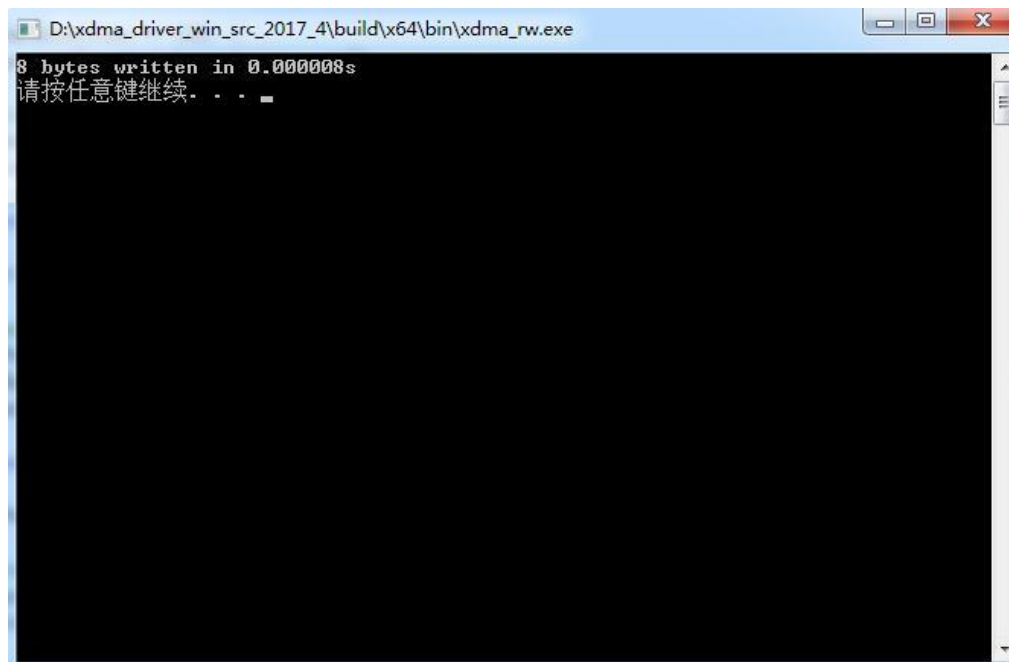


```
D:\xdma_driver_win_src_2017_4\build\x64\bin\xdma_rw.exe
0x0000: 00 00 00 00 11 11 11 11 22 22 22 22 33 33 33 33 ..... "3333
0x0010: 44 44 44 44 55 55 55 55 66 66 66 66 77 77 77 77 DDDDUUUUffffWWWW
0x0020: 88 88 88 88 99 99 99 99 00 00 00 00 00 00 00 00 .....
0x0030: 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
127 bytes received in 0.000092s
请按任意键继续. . .
```

从字节地址 0x30 起再写入 0x00, 可以看到开发板上的 4 个 led (LED1、LED2、LED3、LED4) 灯亮, 界面设置为:



运行的界面为:



然后看到开发板上的 4 个 LED 是亮的。

5 总结

上面介绍了 XDMA 的接口用法，在工程实践中自己需根据需求进行选择对应的方式，文中把这个几种方式统一在工程中，下面再归纳下：

- 1) 用户事件接口 用于 FPGA 事件通知上位机响应；
- 2) M_AXI 接口 用于 PCIe 需 DMA 传输的场合，大量数据的快速传输；
- 3) M_AXI_BYPASS 接口 数据不需 DMA 传输的场合，数据传输效率比较低；
- 4) M_AXI_LITE 接口测试 此接口主要用于 PCIe 通信时进行自定义寄存器控制。

XDMA 的用法就介绍到这里了，如果想要连接 DDR3 内存，有兴趣的同学可以实验一下，这里不做介绍。