

1 FPGA 波形抓取

1.1 跑示例工程

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug936-vivado-tutorial-programming-debugging.pdf 的 lab1。

1) 按步骤添加文件、路径，由于有约束文件，选择文中的 xc7k325tffg900-2 创建工程。

2) 作为可选步骤，在“设置”对话框中，从左侧选择“综合”，然后将展平层次结构更改为无。将此设置更改为 none 的原因是为了防止综合工具对本教程进行任何边界优化。

3) 综合后打开综合设计，有三种方式设置 Debug：①verilog 代码指定；
②在 Netlist 窗口右击设置；③TCL 命令设置。

Tips：②③可能会在软件优化中被吸收、合并。

4) 生成 bitstream

5) 连接板子，可以烧写 bit 文件并看到波形。

6) Q:对于 zynq 板怎么实现？

A:实验得到，生成 bitstream 后，用 overlay 烧写，在 vivado 中打开 HARDWARE MANAGER，加载 ltx 文件（与 bit 流文件同路径）

2 ps 对 pl 传输过多数据情况

2.1 ps 一次性给 pl 传输 20 个数，pl 一次只读取十个

1) 由于 pl 只读取十个就给出 last 信号，所以输出只有前 10 个数。

```
In [1]: from pynq import Overlay
        from pynq import Xlnk
        import numpy as np
        from time import time
        import multiprocessing as mp

In [3]: xlnk = Xlnk()
        overlay = Overlay('./AXIS_test_ultra96-v2.bit')
        dma_in = overlay.axi_dma_in
        dma_out = overlay.axi_dma_out
        dma_out.sendchannel.start()
        dma_in.recvchannel.start()

        in_buf = xlnk.cma_array(shape=(20), dtype=np.uint16)
        out_buf = xlnk.cma_array(shape=(20,), dtype=np.uint16)

        for i in range(20):
            in_buf[i] = i

        dma_in.sendchannel.transfer(in_buf)
        dma_out.recvchannel.transfer(out_buf)
        dma_in.sendchannel.wait()
        dma_out.recvchannel.wait()
        out_buf.flush()

        print(out_buf)

[ 5  6  7  8  9 10 11 12 13 14  0  0  0  0  0  0  0  0  0  0]
```

图 2-1 前 10 个结果的输出

2) 但之前传入的数还剩余十个，因此可以直接继续处理。

```
In [3]: out_buf_1 = xlnk.cma_array(shape=(20,), dtype=np.uint16)
        dma_out.recvchannel.transfer(out_buf_1)
        dma_out.recvchannel.wait()
        out_buf_1.flush()

        print(out_buf_1)

[15 16 17 18 19 20 21 22 23 24  0  0  0  0  0  0  0  0  0  0]
```

图 2-2 后 10 个结果的输出

3) Q : ps 给 pl 一次性传输 20 个可以正常运行，1010 个可以正常运行，

2010 程序会在 sendchannel.wait()处卡死

2.2 板上调试并抓取波形

- 1) 搭建工程，运行 Synthesis
- 2) 在 Synthesis design 中设置 debug 的端口




| Unassigned Debug Nets (36) | | | |
|--|------|---|--|
| >  design_1_i/AXIS_test_0/in_stream_TDATA (16) | FDRE | Q | |
| >  design_1_i/AXIS_test_0/out_stream_TDATA (16) | FDRE | Q | |
|  design_1_i/AXIS_test_0/out_stream_TREADY | FDRE | Q | |
|  design_1_i/AXIS_test_0/out_stream_TVALID | FDRE | Q | |
|  design_1_i/AXIS_test_0/in_stream_TREADY | LUT3 | O | |
|  design_1_i/AXIS_test_0/in_stream_TVALID | FDRE | Q | |

图 2-3 设置 debug 端口

我设置为输入与输出 axis 的 data，ready，valid

- 3) Set up Debug(select both **Capture Control** and **Advanced Trigger**)，Generate Bitstream
- 4) Open Hardware Manager，连接板子，会提示 PL 没上电

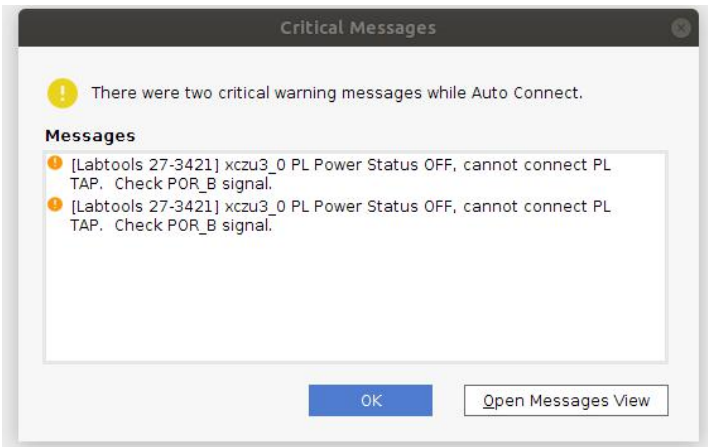


图 2-4 Hardware Manager Warning

5) ipynb 执行到 overlay

```
In [1]: from pynq import Overlay
        from pynq import allocate
        import numpy as np
        from time import time
        import multiprocessing as mp

In [2]: overlay = Overlay('./AXIS_Debug.bit')
```

图 2-5 overlay 代码

刷新设备，可以看到板子上的 ila

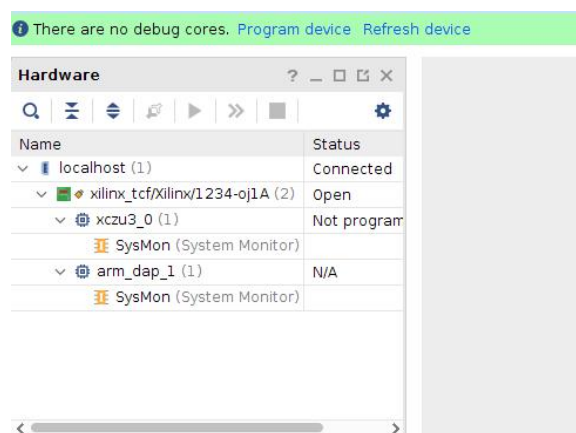


图 2-6 刷新前

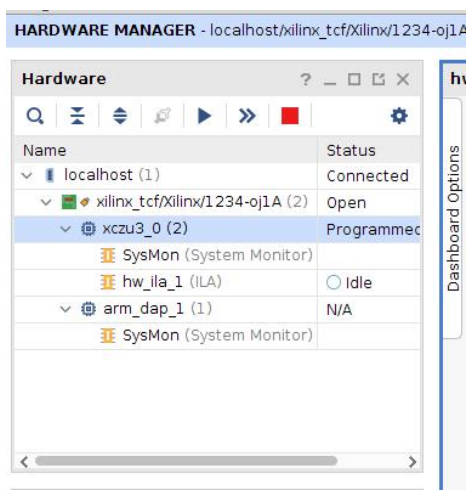


图 2-7 刷新后

6) 制定 ltx 文件 (与 bit 文件同一路径), 可以看到波形图左侧的信号名称, 正是我们之前指定的

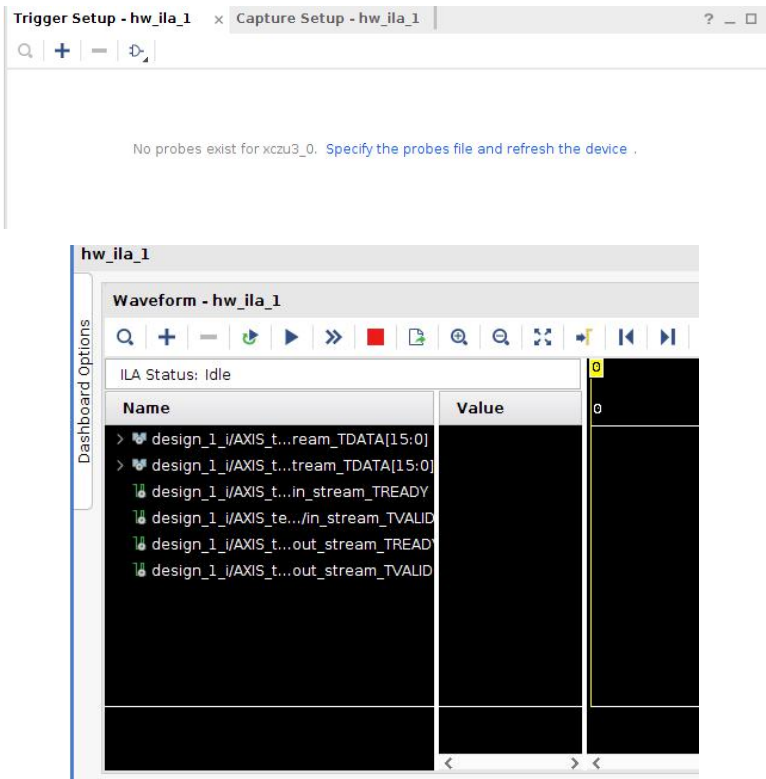


图 2-8 指定 ltx 文件

7) 点击 Add probe, 设置激励, R 代表 0 到 1

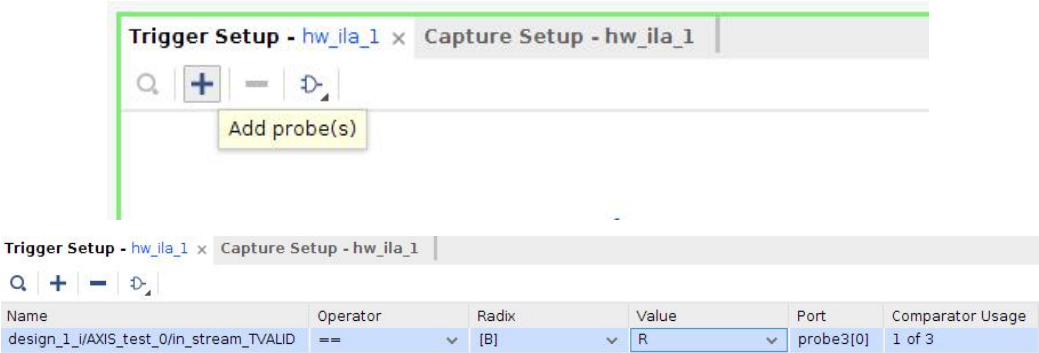


图 2-9 设置激励

8) 右击左侧 ila , 选择 Enable Auto Re-trigger

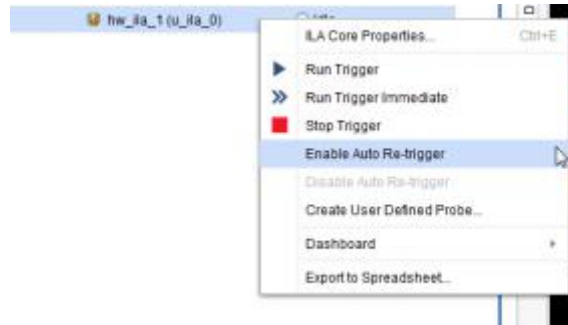


图 2-10 Enable Auto Re-trigger

9) 点击 Run Trigger , Tcl Console 显示

INFO: [Labtools 27-1964] The ILA core 'hw_ila_1' trigger was armed at 2020-Aug-14 22:37:19

10) 运行 ipynb , dma 发送和接收数据。波形图中出现波形，红线代表激励。

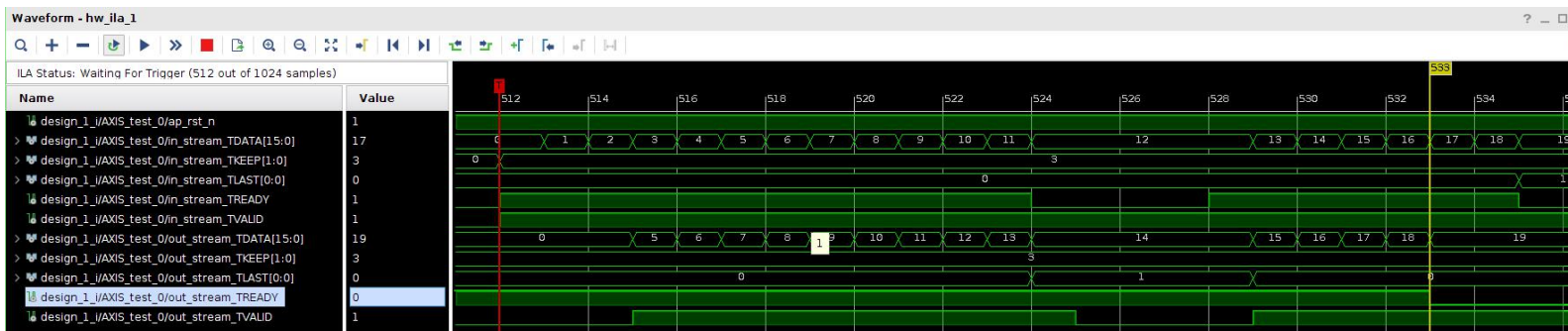


图 2-11 波形图

11) 发现输出只到 19 (15+4) 就被堵住，后面应该还有 20 ~ 24，堵住的原因是 DMA 给的 TREADY 降低，因此当 DMA 再次接收数据时，pl 才会继续处理。