汇编与接口课程设计 模块三

1120141831 朴泉宇

1. 实验目的

随着FPGA 容量的增大，FPGA 的设计日益复杂，设计调试成为一个很繁重的任务。为了使得设计尽快投入市场，设计人员需要一种简易有效的测试工具，以尽可能的缩短测试时间。本实验介绍利用Vivado 内部功能强大的逻辑分析仪ILA 来协助进行FPGA 开发的流程。该工具满足了FPGA 开发中硬件调试的要求，具有无干扰、便于升级、使用简单、价格低廉等特点。本实验以一个实例介绍该分析仪具体的操作方法和步骤，学习并掌握Vivado 内部逻辑分析仪ILA来协助FPGA开发的方法。

1. 实验过程

通过流水灯实验，利用Vivado 内部功能强大的逻辑分析仪ILA来协助在指定开发板上进行FPGA开发。

基本步骤如下：

1. 通过编写HDL文件的方式创建Vivado设计。
2. 通过添加XDC文件进行管脚约束。
3. 通过Mark Debug的方式进行ILA的添加。
4. 生成Bitstream文件。
5. 将生成的Bitstream文件下载到FPGA开发板里。
6. 观察ILA抓取到信号线数据。
7. 实验流程
8. 创建工程

创建一个新的工程“hw\_debug”。

1. 添加源文件和约束文件
2. 创建源文件：“flow\_led\_top.v”，“clk\_div.v”和“flow\_led.v”，并用Verilog语言输入流水灯设计。各模块源代码如下：

flow\_led\_top.v：

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2017/09/07 11:32:54
7. // Design Name:
8. // Module Name: flow\_led\_top
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module flow\_led\_top
22. (
23. input clk, // 100Mhz
24. output [15:0] led
25. );
26. wire clk\_pulse;
27. clk\_div clk\_div
28. (
29. .clk(clk),
30. .clk\_pulse(clk\_pulse) // 1Hz
31. );
32. flow\_led flow\_led
33. (
34. .clk(clk),
35. .clk\_pulse(clk\_pulse),
36. .led\_r(led)
37. );
38. endmodule

flow\_led\_top.v：

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2017/09/07 11:32:54
7. // Design Name:
8. // Module Name: clk\_div
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module clk\_div
22. (
23. input clk, // 100Mhz
24. output clk\_pulse // 1Hz
25. );
26. reg clk\_pulse = 0;
27. reg [25:0] div\_counter = 0;
28. always @(posedge clk) begin
29. if(div\_counter >= 50000000)
30. begin
31. clk\_pulse <= 1;
32. div\_counter <= 0;
33. end
34. else
35. begin
36. clk\_pulse <= 0;
37. div\_counter <= div\_counter + 1;
38. end
39. end
40. endmodule

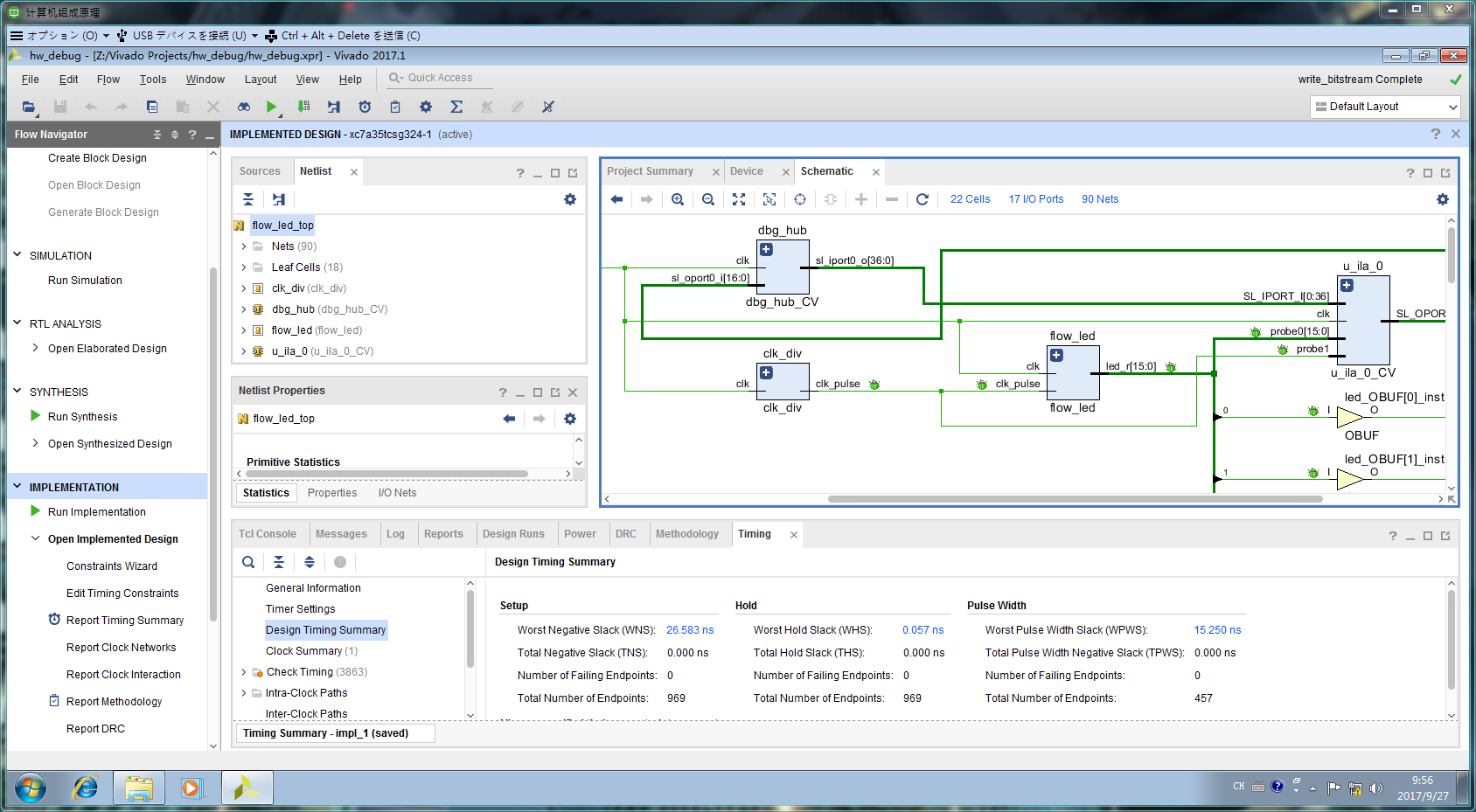
flow\_led.v：

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2017/09/07 11:32:54
7. // Design Name:
8. // Module Name: flow\_led
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module flow\_led
22. (
23. input clk,
24. input clk\_pulse, // 100Mhz
25. output reg [15:0] led\_r
26. );
27. reg [15:0] led\_r = 16'h000f;
28. always @(posedge clk) begin
29. if(clk\_pulse == 1)
30. led\_r <= {led\_r[11:0], led\_r[15:12]};
31. else
32. led\_r <= led\_r;
33. end
34. endmodule
35. 添加约束文件“flow\_led.xdc”：
36. set\_property IOSTANDARD LVCMOS33 [get\_ports clk]
37. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[15]}]
38. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[14]}]
39. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[13]}]
40. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[12]}]
41. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[11]}]
42. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[10]}]
43. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[9]}]
44. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[8]}]
45. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[7]}]
46. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[6]}]
47. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[5]}]
48. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[4]}]
49. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]
50. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]
51. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]
52. set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]
53. set\_property PACKAGE\_PIN P17 [get\_ports clk]
54. set\_property PACKAGE\_PIN F6 [get\_ports {led[15]}]
55. set\_property PACKAGE\_PIN G4 [get\_ports {led[14]}]
56. set\_property PACKAGE\_PIN G3 [get\_ports {led[13]}]
57. set\_property PACKAGE\_PIN J4 [get\_ports {led[12]}]
58. set\_property PACKAGE\_PIN H4 [get\_ports {led[11]}]
59. set\_property PACKAGE\_PIN J3 [get\_ports {led[10]}]
60. set\_property PACKAGE\_PIN J2 [get\_ports {led[9]}]
61. set\_property PACKAGE\_PIN K2 [get\_ports {led[8]}]
62. set\_property PACKAGE\_PIN K1 [get\_ports {led[7]}]
63. set\_property PACKAGE\_PIN H6 [get\_ports {led[6]}]
64. set\_property PACKAGE\_PIN H5 [get\_ports {led[5]}]
65. set\_property PACKAGE\_PIN J5 [get\_ports {led[4]}]
66. set\_property PACKAGE\_PIN K6 [get\_ports {led[3]}]
67. set\_property PACKAGE\_PIN L1 [get\_ports {led[2]}]
68. set\_property PACKAGE\_PIN M1 [get\_ports {led[1]}]
69. set\_property PACKAGE\_PIN K3 [get\_ports {led[0]}]
70. 综合

在“Flow Navigator”栏中的“Synthesis”下点击“Run Synthesis”。右上角的进度条“Running synth\_design”指示正在对工程进行综合。

综合完成之后在弹出的对话框中点击“Cancel”取消。在“Flow Navigator”一栏中，找到“Synthesis” -> “Open Synthesised Design” -> “Schematic”，点击“Schematic”。

1. Mark Debug
   1. 在“Schematic”标签页中，点击左侧工具栏中的放大镜图标，将电路图放大到合适的大小。找到“clk\_div”模块和“flow\_led”模块之间的连线“clk\_pulse”，选中后右击，选择“Mark Debug”。
   2. 同样，找到与“flow\_led”模块的输出端口相连的信号线“led\_OBUF”，选中后右击，选择“Mark Debug”。



1. Set Up Debug
   1. 在“Debug”窗口中（可通过菜单栏中点击“Layout” -> “Debug”打开），单击选中“Unassigned Debug Nets”，然后右击选择“Set Up Debug”。
   2. 在“Set Up Debug”向导中连续点击“Next”，最后点击“Finish”。
   3. 在菜单栏中点击“File” -> “Save Constraints”。在弹出的对话框中点击“OK”。然后在“Source”窗口中打开“flow\_led.xdc”，在文档的地步可以看到“Mark Debug”及“Set Up Debug”的相关信息被添加上去了。
2. set\_property MARK\_DEBUG true [get\_nets clk\_pulse]
3. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[6]}]
4. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[2]}]
5. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[0]}]
6. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[1]}]
7. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[3]}]
8. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[4]}]
9. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[5]}]
10. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[11]}]
11. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[12]}]
12. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[7]}]
13. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[8]}]
14. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[15]}]
15. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[13]}]
16. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[14]}]
17. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[9]}]
18. set\_property MARK\_DEBUG true [get\_nets {led\_OBUF[10]}]
19. create\_debug\_core u\_ila\_0 ila
20. set\_property ALL\_PROBE\_SAME\_MU true [get\_debug\_cores u\_ila\_0]
21. set\_property ALL\_PROBE\_SAME\_MU\_CNT 1 [get\_debug\_cores u\_ila\_0]
22. set\_property C\_ADV\_TRIGGER false [get\_debug\_cores u\_ila\_0]
23. set\_property C\_DATA\_DEPTH 1024 [get\_debug\_cores u\_ila\_0]
24. set\_property C\_EN\_STRG\_QUAL false [get\_debug\_cores u\_ila\_0]
25. set\_property C\_INPUT\_PIPE\_STAGES 0 [get\_debug\_cores u\_ila\_0]
26. set\_property C\_TRIGIN\_EN false [get\_debug\_cores u\_ila\_0]
27. set\_property C\_TRIGOUT\_EN false [get\_debug\_cores u\_ila\_0]
28. set\_property port\_width 1 [get\_debug\_ports u\_ila\_0/clk]
29. connect\_debug\_port u\_ila\_0/clk [get\_nets [list clk\_IBUF\_BUFG]]
30. set\_property PROBE\_TYPE DATA\_AND\_TRIGGER [get\_debug\_ports u\_ila\_0/probe0]
31. set\_property port\_width 16 [get\_debug\_ports u\_ila\_0/probe0]
32. connect\_debug\_port u\_ila\_0/probe0 [get\_nets [list {led\_OBUF[0]} {led\_OBUF[1]} {led\_OBUF[2]} {led\_OBUF[3]} {led\_OBUF[4]} {led\_OBUF[5]} {led\_OBUF[6]} {led\_OBUF[7]} {led\_OBUF[8]} {led\_OBUF[9]} {led\_OBUF[10]} {led\_OBUF[11]} {led\_OBUF[12]} {led\_OBUF[13]} {led\_OBUF[14]} {led\_OBUF[15]}]]
33. create\_debug\_port u\_ila\_0 probe
34. set\_property PROBE\_TYPE DATA\_AND\_TRIGGER [get\_debug\_ports u\_ila\_0/probe1]
35. set\_property port\_width 1 [get\_debug\_ports u\_ila\_0/probe1]
36. connect\_debug\_port u\_ila\_0/probe1 [get\_nets [list clk\_pulse]]
37. set\_property C\_CLK\_INPUT\_FREQ\_HZ 300000000 [get\_debug\_cores dbg\_hub]
38. set\_property C\_ENABLE\_CLK\_DIVIDER false [get\_debug\_cores dbg\_hub]
39. set\_property C\_USER\_SCAN\_CHAIN 1 [get\_debug\_cores dbg\_hub]
40. connect\_debug\_port dbg\_hub/clk [get\_nets clk\_IBUF\_BUFG]
41. 生成bit文件

在“Flow Navigator”一栏中的“Program and Debug”下点击“Generate Bitstream”，此时会提示工程没有实现，点击“Yes”，会自动执行实现过程。

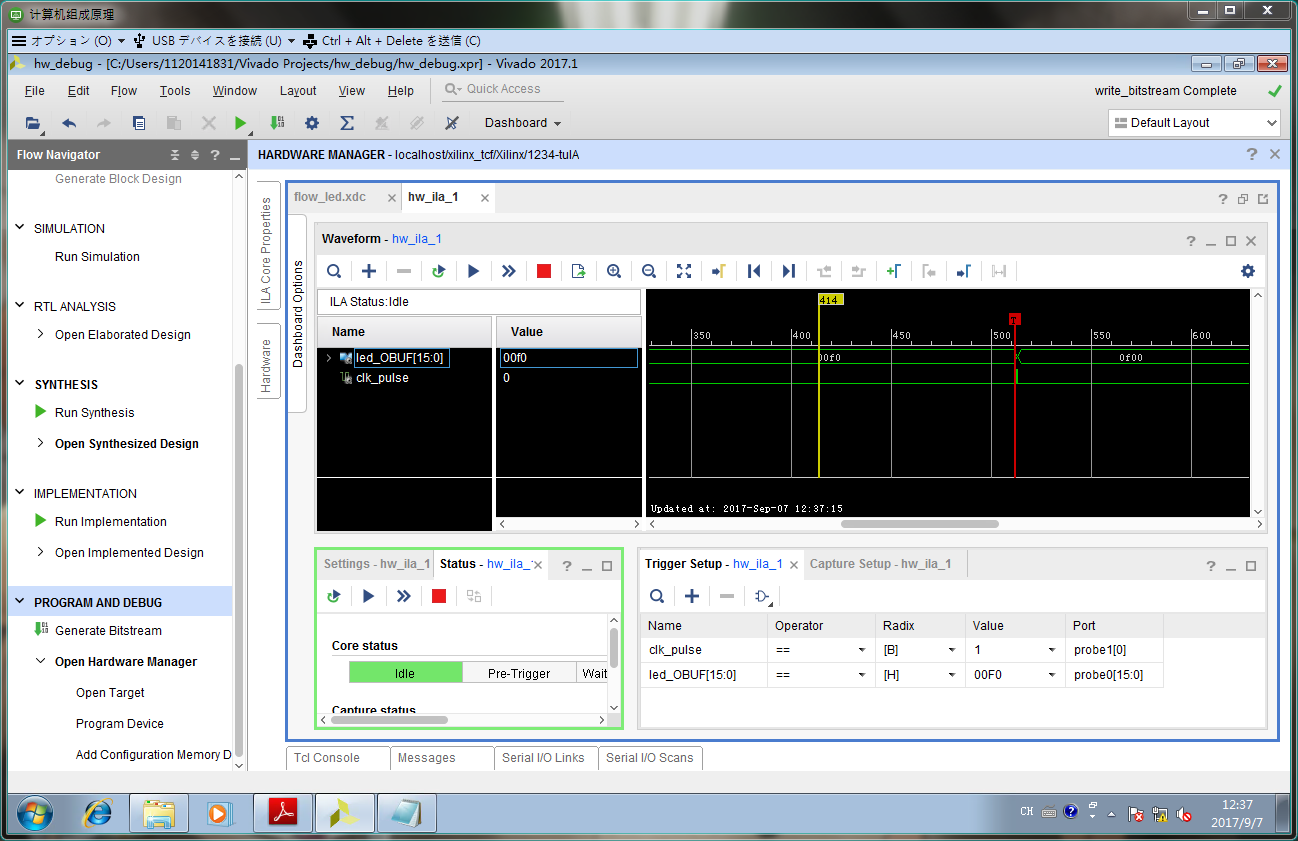
1. 下载
   1. 用Micro USB线连接电脑与板卡上的JTAG端口，打开电源开关。
   2. 生成比特流文件完成后，打开“Hardware Manager”。在“Hardware Manager”界面点击“Open target”，选择“Auto Connect”。

连接成功后，在目标芯片上右击，选择“Program Device”。在弹出的对话框中“Bitstream File”一栏已经自动加载本工程生成的比特流文件，点击“Program”对FPGA芯片进行编程。

* 1. 下载完成后“Hardware Manager”界面如下图所示（已省略）。

1. Hardware Debug
   1. 选中目标芯片，点击上方的“Run Trigger Immediate”，在波形窗口中可看到触发信号。
   2. 在“Trigger Setup”窗口中点击加好，选中“clk\_pulse”，双击添加到窗口中。
   3. 在“Trigger Setup”窗口中将“led\_OBUF”的“Compare Value”设为“[H] 00F0”，将“clk\_pulse”的“Compare Value”设为1。
   4. 在“Waveform”窗口点击加好，在“Add Probes”列表中双击“clk\_pulse”将其添加到波形窗口中。
   5. 在“hw\_ila\_1”的“Settings”窗口中，将“Trigger position in window”一栏设为512。
   6. 在“Hardware”窗口选中“hw\_ila\_1”，然后点击上方的“Run Trigger”按钮。在“Status”窗口可观察到状态由“Idle”跳转到“Waiting for Trigger”。

当状态跳转到“Full”后回到“Idle”状态，此时将波形图中红色竖线标注出了触发的时刻。将波形图放大之后，可以看到出发时刻的信号“led\_OBUF”和“clk\_pulse”与设置的触发条件一致。



1. 心得体会

首次接触FPGA开发板。这块板子与我在项目中接触到的CC3200要高级很多，因此也复杂了很多。但此次实验重在使用Vivado进行调试开发，因此也并没有深入了解FPGA开发板。在编码并使用Vivado的时候，体会了代码 – 硬件逻辑的过程，并添加了Debug信息，进行调试的实验。

此次实验，让我熟悉了Vivado的使用方法，为模块二的VGA实验以及后续实验做了准备。