25A, No.1369,Dongfang Road, Shanghai
Tel:+86-021-58893151 Fax: +86-021-58894598

Lab05: MIPS 多周期流水化处理器实现

Targeting on Digilent Anvyl

Lab 05: MIPS 多周期流水化处理器实现

实验简介

本实验旨在使读者实现一个简单的类 MIPS 多周期流水化处理器。

实验目标

在完成本实验后，您将学会：

- 理解 CPU 的流水化设计
- 初步认识 Data Hazard

实验过程

本实验旨在使读者理解和掌握 5 级流水线 CPU 的设计。本实验建立在前几个实验的基础上，在实验中主要的工作就是重新设计 Control 模块，以及修改模块间互联的定义。

实验由以下步骤组成：

1. 创建工程
2. 编写 Verilog 代码
3. 仿真测试
4. 下载验证

创建工程 Step 1

打开 ISE 工具进行数字逻辑设计。

- 打开 ISE 工具，新建工程(**注：本实验使用 ISE 13.4**)

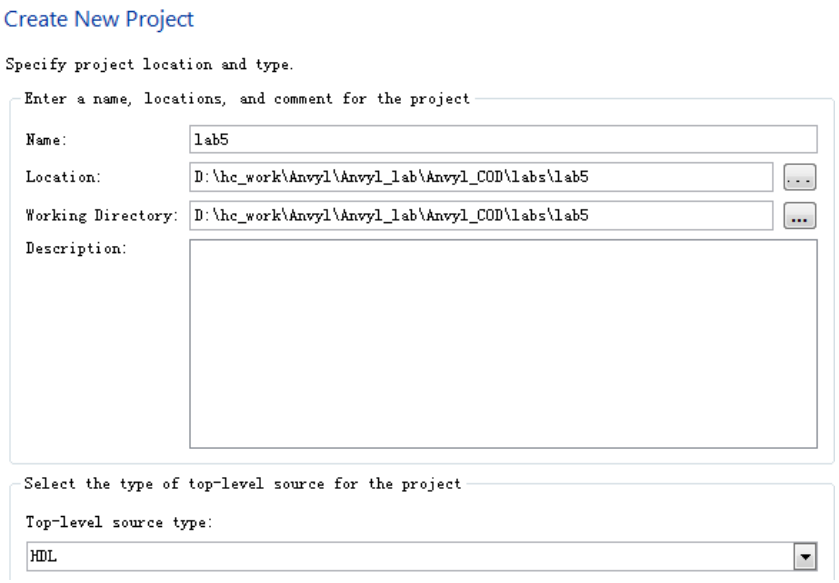

图 1. 创建新工程

- 选择 FPGA 型号、综合和仿真工具、推荐描述语言等配置

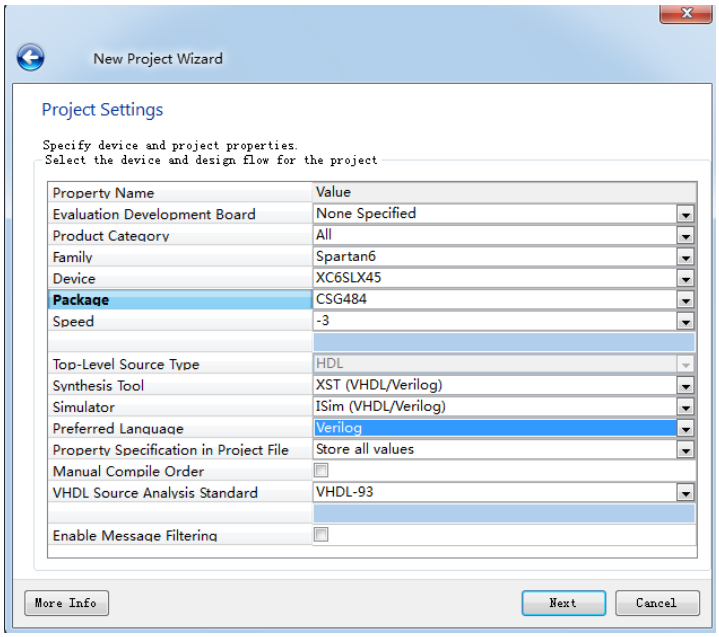

图 2. 新工程设置

- 右键点击 Hierarchy 窗口，选择 Add Copy of Source ，添加已有的模块，例如 Register file，Sign Extend，Data Memory，Instruction Memory，ALU 模块。Top 层模块和 Control 需要重新定义。

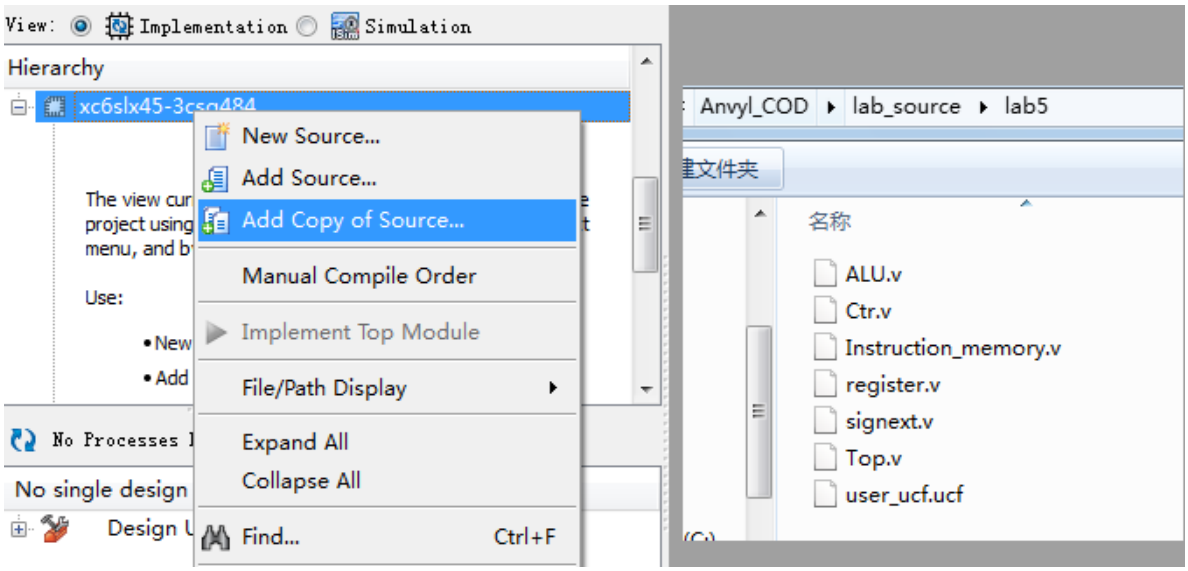

图 3.添加已有的 Verilog 模块

注：由于在流水线处理下，Data memory 的读和写全部用时序逻辑实现，即读写都需要时钟控制。所以采用使用 BRAM 来完成设计，使用 Core Generator 来生成。

- 添加 DmemB coregen 文件，右键选择 Source 窗口，选择 New source，输入模块名字，然后选择 IP（CORE Generator & Architecture Wizard）。

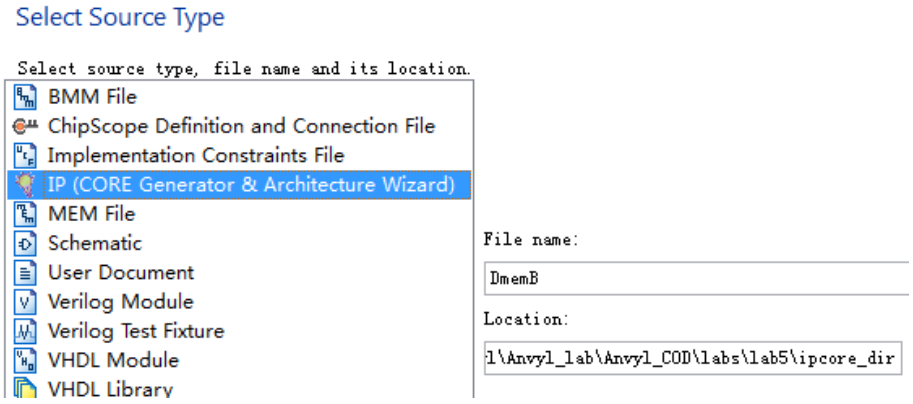

图 4. 使用 Core Generator 生成 Data memory

- IP 中选择 Block RAM Generator。

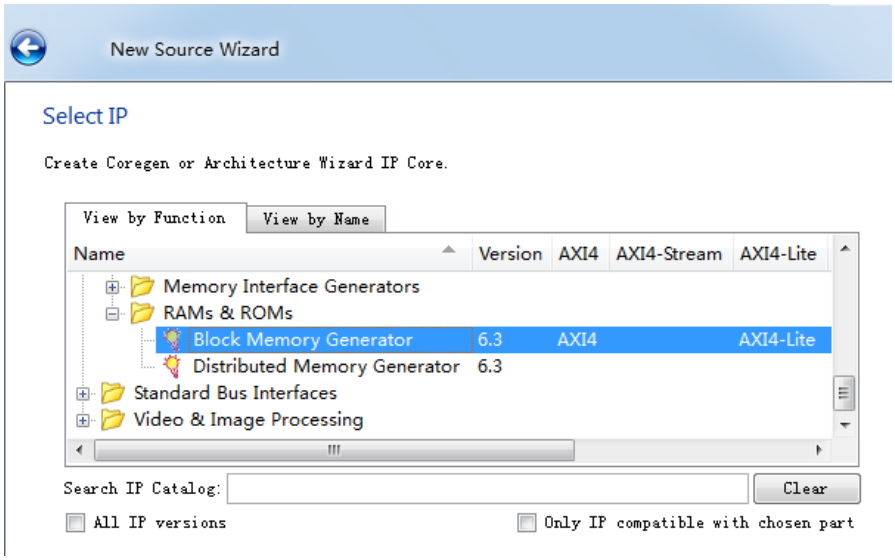

图 5. 调用 Block Memory Generator

- 配置 Block RAM 的参数。选择…\Anvyl_COD\lab_source\lab5 下的 data.coe 文件来初始化该 Bram 模块

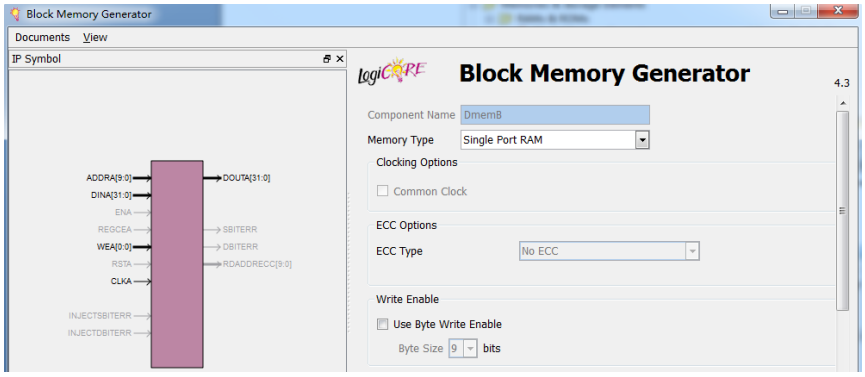

图 6. 选择单端口的 RAM

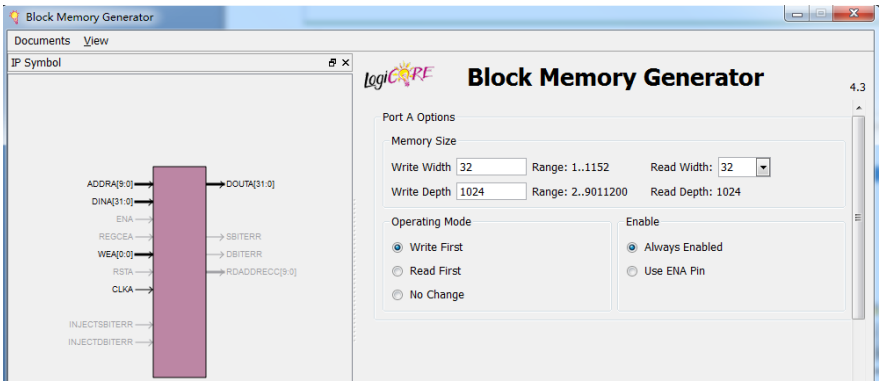

图 7. 配置 Block Memory 宽度和深度

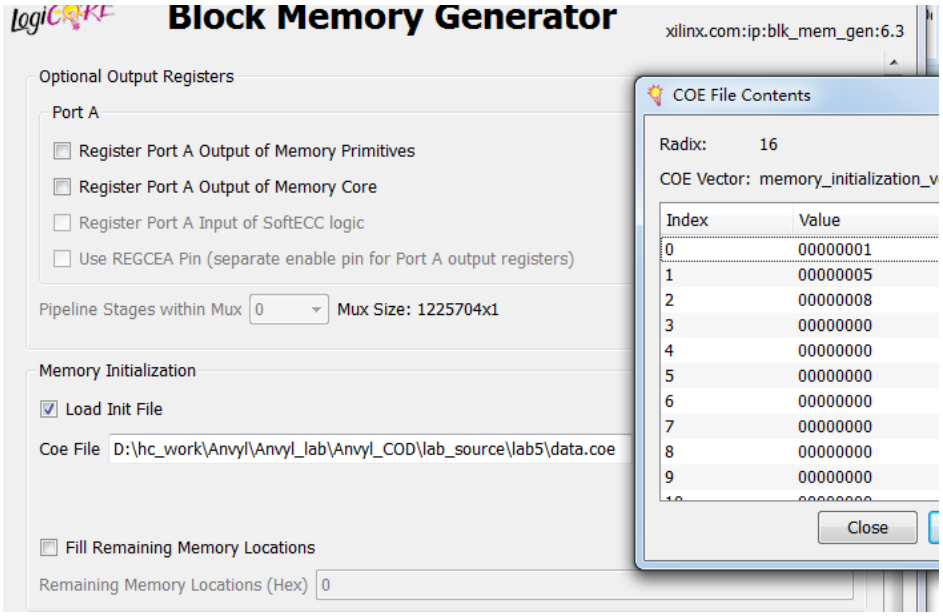

图 8. 装载 Block RAM 初始化文件以及 coe 文件格式

- 在 Top 模块中添加语句调用 Block RAM。

```
103     DmemB Dmem(
104       .clka(Clk),
105       .addra(ALUOutM[11:2]),
106       .douta(ReadDataW),
107       .wea(MemWriteM),
108       .dina(WriteDataM)
109     );
```

图 9. 在 Top 模块中调用 Block memory

编写 Verilog 代码 Step 2

模块之间插入了 4 个寄存器，将单周期的处理器操作分成 5 个周期完成，添加用于控制数据流的控制器，以实现 5 级流水线，包括 Fetch，Decode，Execute，Memory 和 Writeback。见图 10

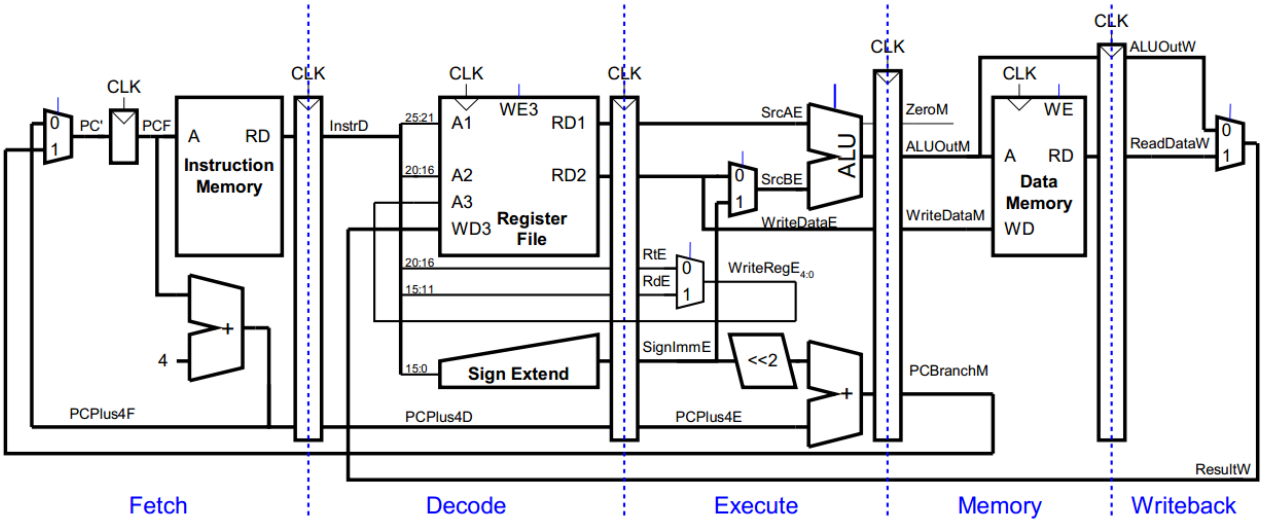

图 10. MIPS 5 级流水线划分

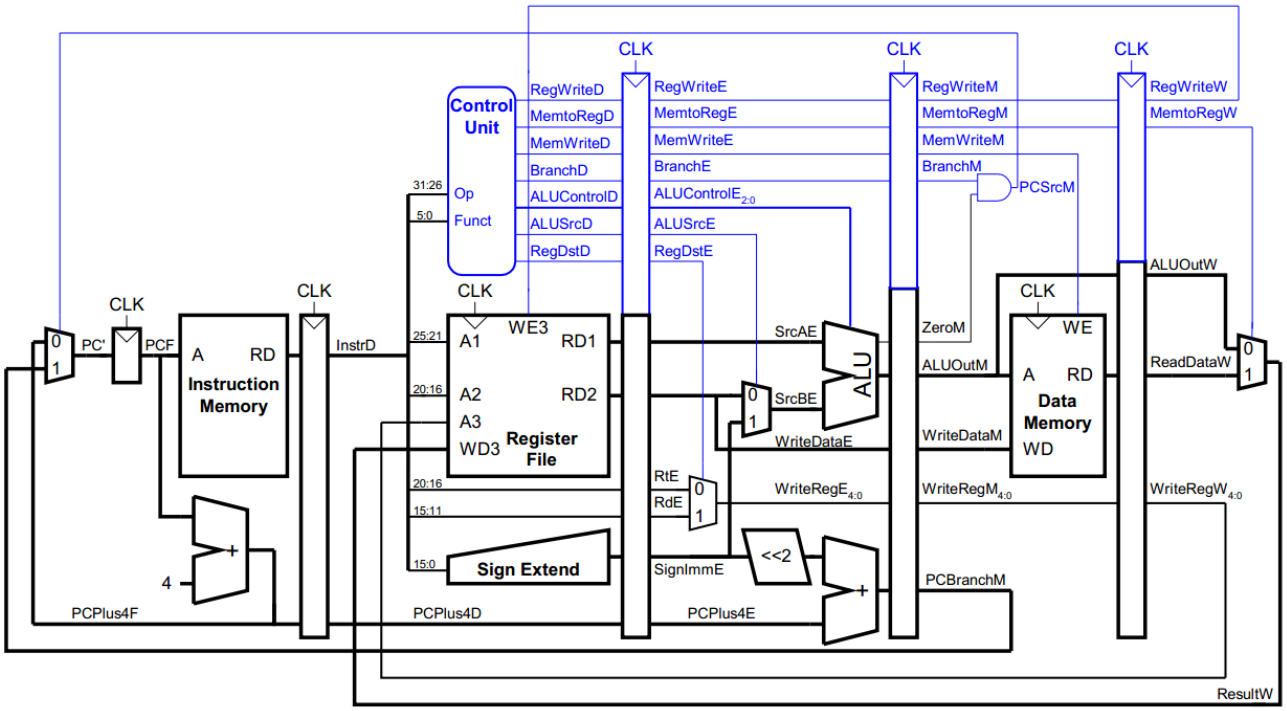

图 11. MIPS 5 级流水线原理图

由于各种变量名比较复杂，需要创建一套命名规则，方便代码的编写和阅读。图 11 中表示了一个套命名规则，可做参考，F 表示 Fetch 级，D 表示 Decode 级，E 表示 Execute 级，M 表示 Memory 级，W 表示 Writeback 级。

为了实现流水线设计，控制器所输出的控制信号也需要插入寄存器，以控制数据路径的同步。

- 编辑 control 模块

```
20  //////////////////////////////////////////////////////////////////
21  module Ctr(
22      input [5:0] OpCode,
23      input [5:0] Funct,
24      output reg RegDstD,
25      output reg ALUSrcD,
26      output reg MemToRegD,
27      output reg RegWriteD,
28      output reg MemWriteD,
29      output reg BranchD,
30      output reg [3:0] ALUControlD
31      );
32
33  reg [1:0] ALUOp;
34
35  always @(OpCode)
36  begin
37      case(OpCode)
38
39          //R type
40          6'b000000:
41          begin
42              RegDstD = 1;
43              ALUSrcD = 0;
44              MemToRegD = 0;
45              RegWriteD = 1;
46              MemWriteD = 0;
47              BranchD = 0;
48              ALUOp = 2'b10;
49          end
```

图 12.控制器端口重新定义

- 根据图 11，编辑 Top 层模块，将各个模块互联起来。

```
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////
21
22  module Top(
23      input Clk,
24      input Rst,
25      input [7:0] Switch,
26      output [7:0] Led
27      );
28
29  wire Zero;
30  wire [31:0] SignExtOut;
31  wire [31:0] ALURes;
32
33  //fetch internal signal
34  wire [31:0] PCPlus4F;
35  reg [31:0] PCF;
36
37  //Decode internal signal
38  wire RegWriteD;
39  wire MemToRegD;
40  wire MemWriteD;
41  wire BranchD;
42  wire [3:0]ALUControlD;
43  wire ALUSrcD;
44  wire RegDstD;
45  wire [31:0] InstructionD;
46  reg [31:0] PCPlus4D;
```

图 13. Top 层模块端口定义

仿真测试 Step 3

整个处理器设计完成，接下来编辑 testbench 文件，进行行为级的仿真。

- 初始化 data memory、instruction memory 和 register 三大存储模块，这里仅以初始化 instruction memory 为例说明，其他类推即可。该 memory 用于存储二进制代码。如图 14 显示，Verilog 中调用了系统任务$readmemh 将 Instruction 文件中的数据读入到 InstMem 数组中。

  注意：在单周期处理器设计时，为了使得所有操作在单个周期内完成，我们将几个存储器的读逻辑用组合逻辑实现。但在多周期处理器中，为了实现流水线操作，读和写操作同样都用时序逻辑实现。

```
20 //////////////////////////////////////////////////////////////////
21 module Instruction_memory(
22     input Clk,
23     input [31:0] ImemRdAddr,
24     output reg [31:0] Instruction
25     );
26
27 reg [31:0] InstMem [0:255];   //memory space for storing instructions
28
29 //initial the instruction and data memory
30 initial
31 begin
32    $readmemh("Instruction",InstMem,8'h0);
33 end
34
35 always @(posedge Clk)
36 begin
37    Instruction <= InstMem[ImemRdAddr];
38 end
39 endmodule
40
```

图 14. 初始化 memory

表 1. MIPS 测试指令

指令地址	二进制代码	寄存器翻译	MIPS 汇编指令	指令解释
[0x00000000]	0x8c020000	lw $2, 0($0)	1: lw $v0, 0x0($zero)	$2 <= 1
[0x00000004]	0x8c030004	lw $3, 4($0)	2: lw $v1, 0x4($zero)	$3 <= 5
[0x00000008]	0x8c040008	lw $4, 8($0)	3: lw $a0, ox8($zero)	$4 <= 8
[0x0000000c]	0x00432820	add $5, $2, $3	4: add $a1, $v0, $v1	$5 <= $2 + $3 = 6
[0x00000010]	0x00823022	sub $6, $4, $2	5: sub $a2, $a0, $v0	$6 <= $4 - $2 = 7
[0x00000014]	0x00623824	and $7, $3, $2	6: and $a3, $v1, $v0	$7 <= $3 and $2 = 1
[0x00000018]	0x8c0b0000	lw $11, 0($0)	7: lw $t3, 0x0($zero)	$11 <= 1
[0x0000001c]	0x8c0b0000	lw $11, 0($0)	8: lw $t3, 0x0($zero)	$11 <= 1
[0x00000020]	0x8c0b0000	lw $11, 0($0)	9: lw $t3, 0x0($zero)	$11 <= 1
[0x00000024]	0x00824025	or $8, $4, $2	10: or $t0,$a0, $v0	$8 <= $4 or $2 = 9
[0x00000028]	0x0082482a	slt $9, $4, $2	11: slt $t1,$a0, $v0	$9 <= 0
[0x0000002c]	0x1000000a	beq $0, $0, 40 [start-0x0000002c]	12: beq $zero, $zero, start	
[0x00000030]	0x01095020	add $10, $8, $9	13: add $t2,$t0, $t1	$10 <= $8 + $9 =9
[0x00000034]	0x8c0b0000	lw $11, 0($0)	14: lw $t3, 0x0($zero)	$11 <= 1
[0x00000038]	0x8c0b0000	lw $11, 0($0)	15: lw $t3, 0x0($zero)	$11 <= 1
[0x0000003c]	0x8c0b0000	lw $11, 0($0)	16: lw $t3, 0x0($zero)	$11 <= 1
[0x00000040]	0x8c0b0000	lw $11, 0($0)	17: lw $t3, 0x0($zero)	$11 <= 1
[0x00000044]	0x8c0b0000	lw $11, 0($0)	18: lw $t3, 0x0($zero)	$11 <= 1
[0x00000048]	0x8c0b0000	lw $11, 0($0)	19: lw $t3, 0x0($zero)	$11 <= 1
[0x0000004c]	0x8c0b0000	lw $11, 0($0)	20: lw $t3, 0x0($zero)	$11 <= 1
[0x00000050]	0x8c0b0000	lw $11, 0($0)	21: lw $t3, 0x0($zero)	$11 <= 1
[0x00000054]	0x8c0c0000	lw $12, 0($0)	23: lw $t4, 0x0($zero)	$12 <= 1
[0x00000058]	0x8c0c0000	lw $12, 0($0)	24: lw $t4, 0x0($zero)	$12 <= 1
[0x0000005c]	0x8c0c0000	lw $12, 0($0)	25: lw $t4, 0x0($zero)	$12 <= 1

[0x00000060]	0x8c0c0000	lw $12, 0($0)	26: lw $t4, 0x0($zero)	$12 <= 1
[0x00000064]	0x8c0c0000	lw $12, 0($0)	27: lw $t4, 0x0($zero)	$12 <= 1
[0x00000068]	0x8c0c0000	lw $12, 0($0)	28: lw $t4, 0x0($zero)	$12 <= 1
[0x0000006c]	0x8c0c0000	lw $12, 0($0)	29: lw $t4, 0x0($zero)	$12 <= 1
[0x00000070]	0x8c0c0000	lw $12, 0($0)	30: lw $t4, 0x0($zero)	$12 <= 1
[0x00000074]	0x8c0c0000	lw $12, 0($0)	31: lw $t4, 0x0($zero)	$12 <= 1
[0x00000078]	0x8c0c0000	lw $12, 0($0)	32: lw $t4, 0x0($zero)	$12 <= 1

表 2. Data memory 部分数据

数据地址	数据
[0x00000000]	**00000001**
[0x00000004]	**00000005**
[0x00000008]	**00000008**
[0x0000000c]	ffff0000
[0x00000010]	ffff0000
[0x00000014]	ffff0000
[0x00000018]	ffff0000
[0x0000001c]	ffff0000
[0x00000020]	ffff0000
[0x00000024]	ffff0000
[0x00000028]	ffff0000
[0x0000002c]	ffff0000
[0x00000030]	ffff0000
[0x00000034]	ffff0000
[0x00000038]	ffff0000
[0x0000003c]	ffff0000
…	…

- 编写 Top 层的 testbench 文件，右键选中 Hierachy 窗口，选择 new source。如图 15 所示，定义 file name 为 Top_tb，在左侧栏中选择 Verilog Test Fixture，点击 Next，选择 Top 模块。自动生成 Top_tb 测试文件。

  注意：...\Anvyl_COD\lab_source\lab4 下有提供 Top_tb 文件，可以直接添加。

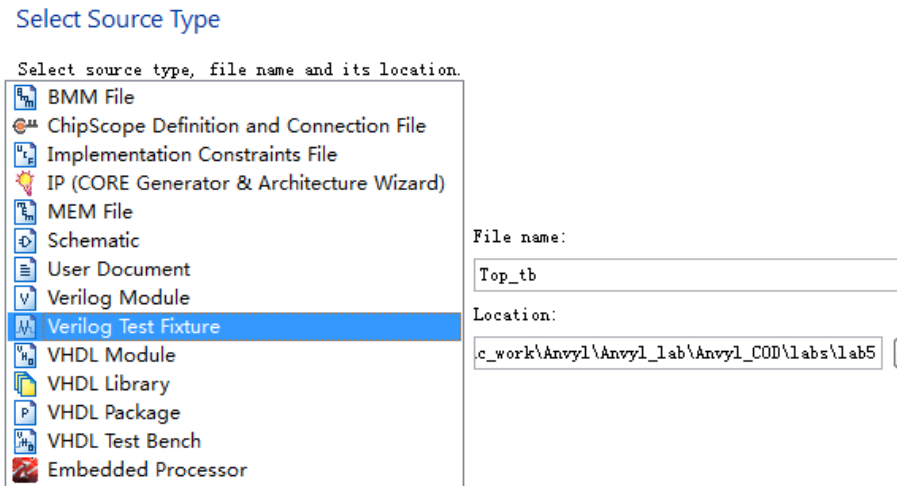

图 15. 添加 Top_tb 仿真测试文件

- 添加时钟激励和其他输入信号的初始化。

```
35    // Instantiate the Unit Under Test (UUT)
36    Top uut (
37       .Clk(Clk),
38       .Rst(Rst),
39       .Switch(Switch),
40       .Led(Led)
41    );
42
43    initial begin
44       // Initialize Inputs
45       Clk = 0;
46       Rst = 1;
47       Switch = 0;
48       // Wait 100 ns for global reset to finish
49       #10;
50       Rst = 0;
51    end
52
53    //Clock generate
54    always #2 Clk = !Clk;
55
```

图 16. 编辑 TestBench 文件

- 调用 ISE 自带 Isim 仿真工具进行仿真，双击 Simulate Behavioral Model

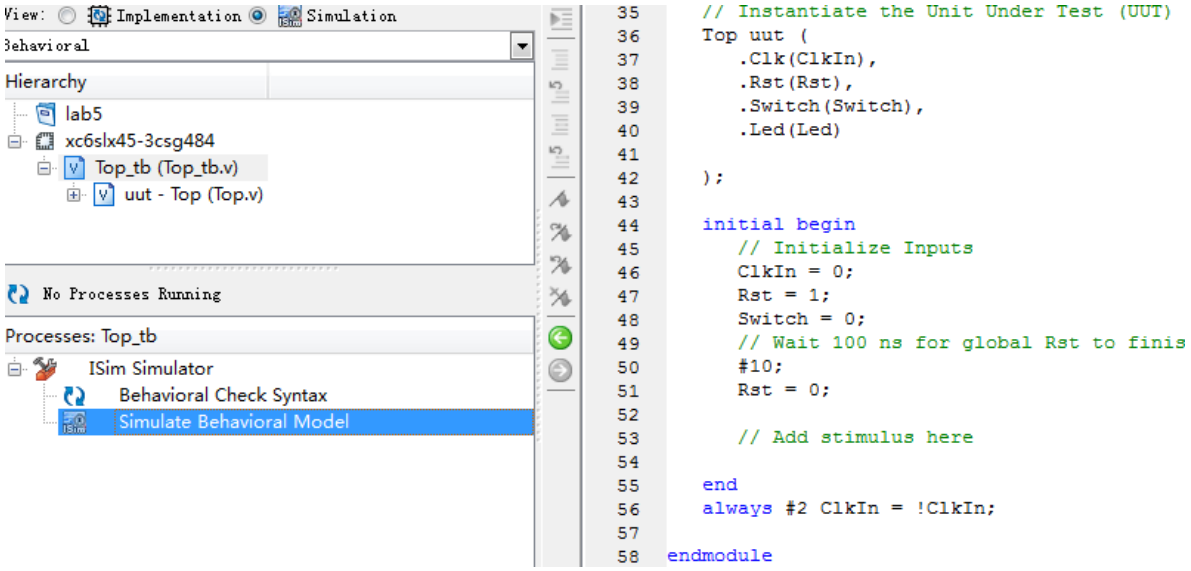

图 17. 调用 ISim 仿真工具

- 添加 register 模块中的 regfile 寄存器数组到波形窗口,观察各个寄存器的变化情况，如图 12 所示
- 再添加 uut 模块的 InstructionD[31:0]和 PCF[31:0]信号到波形窗口。

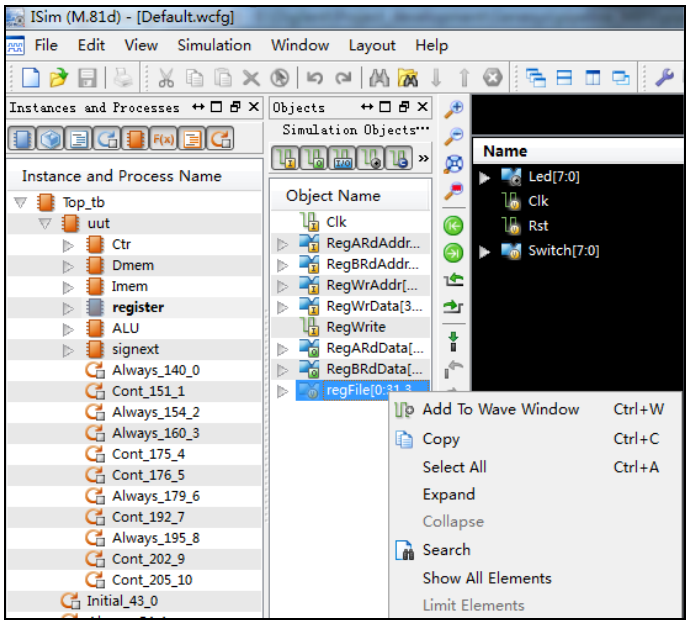

图 18. 添加 regfile 到波形窗口

- 在 Console 窗口中输入 restart；run 200ns，重新进行仿真。观察如图 19 的波形可知，$5 寄存器的值是 0x0000000 和$6 寄存器的值是 0xffffffff；但是通过汇编指令可知，$5 寄存器的值应该是 0x00000006，$6 寄存器的值应该是 0x00000007。这是由于流水化处理后，当前指令用到了前面指令尚未计算完成的寄存器值，导致计算结果出错，这就是 Data Hazard。

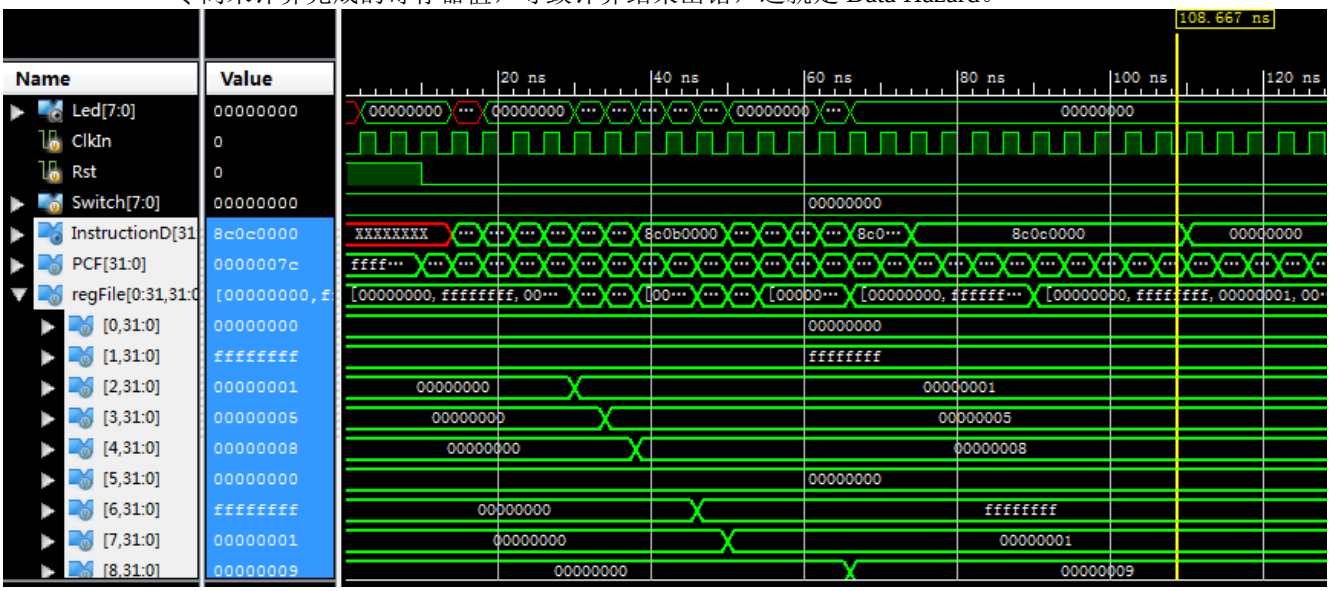

图 19. Data Hazard

- 消除 Data Hazard 的简单方法是：在指令中添加空转指令，以保证指令所需要的寄存器值都是已经计算完成的，例如

1: lw $v0, 0x0($zero)
2: lw $v1, 0x4($zero)
3: lw $a0, ox8($zero)
4: **nop**
5: **nop**
6: add $a1, $v0, $v1
7: sub $a2, $a0, $v0

- 修改指令后，重新仿真，可看到如图 20 的波形

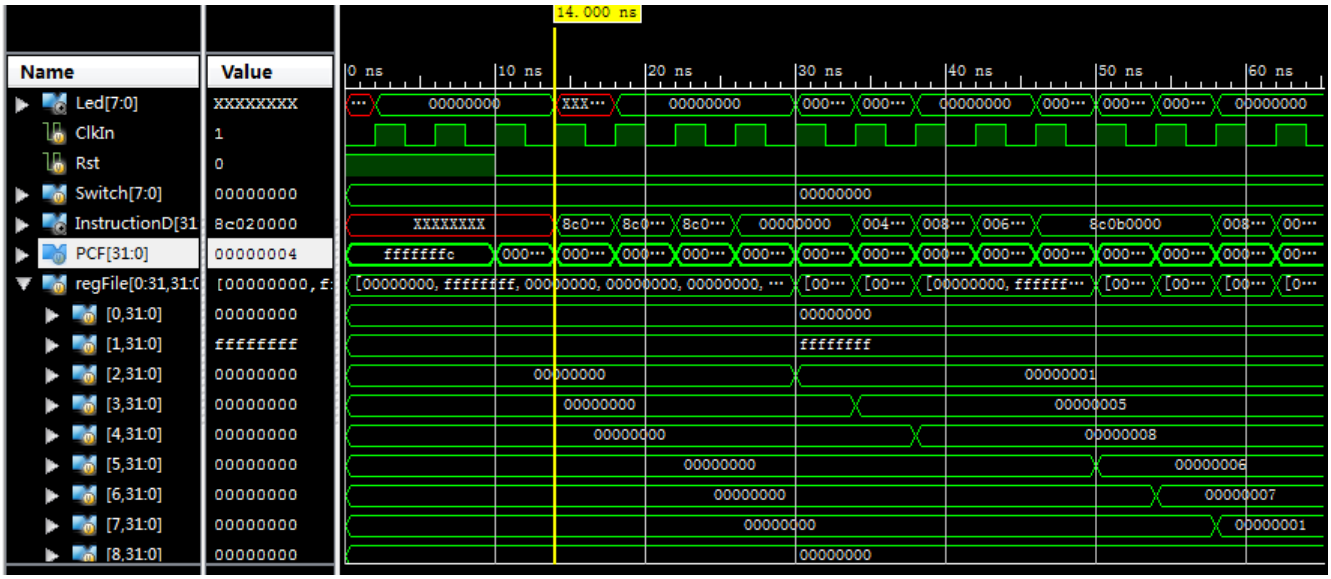

图 20.消除 Data Hazard 后的波形

下载验证 Step 4

行为级仿真已经通过，那接下来下载到板上进行验证。方法是利用 ChipScope 工具，抓取 FPGA 运行过程产生的波形，进行观察分析。

- 添加 UCF 文件，包括物理管脚约束和时钟约束，具体请参考实验 1。

- 配置 FPGA 综合的参数，右键选择 Processes 栏中的 Synthesize-XTS，点击 Process Properties，在 Keep Hierarchy 栏中选择 Yes。这样做的目的为了综合之后保持原来的信号命名，便于识别。

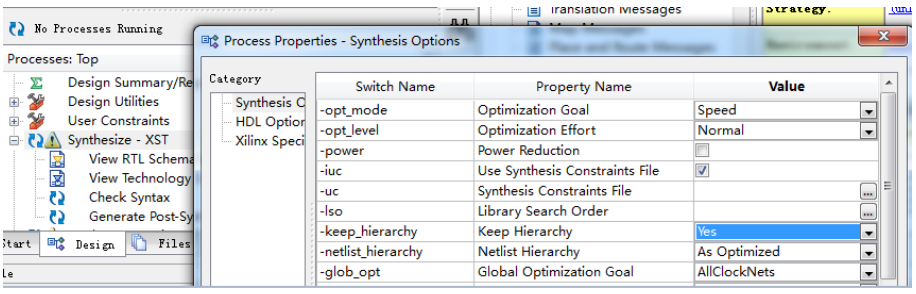

图 21. 配置 FPGA 综合参数

- 添加用于抓取波形的 ChipScope IP Core，右键选中 Hierarchy 窗口，点击 New source。

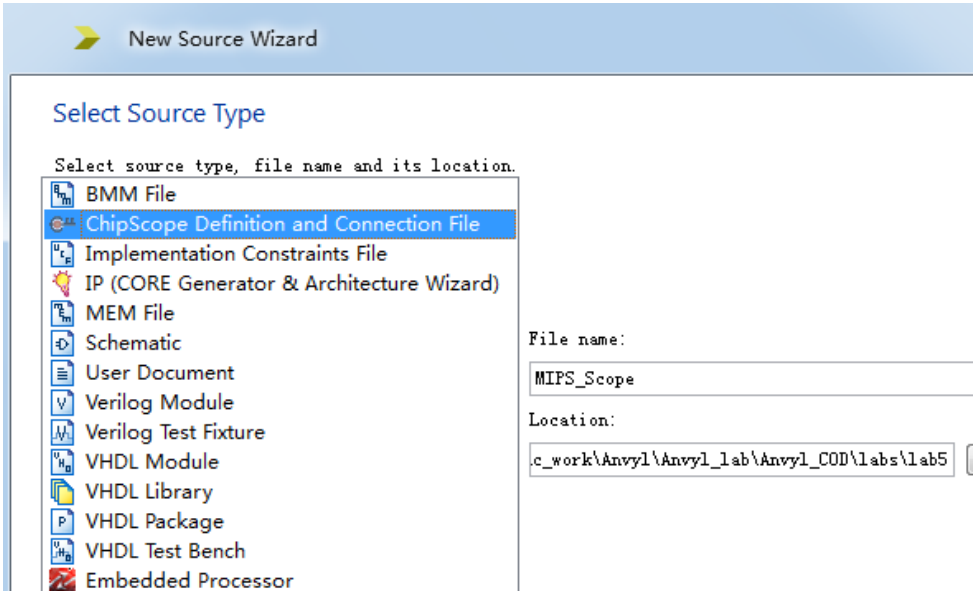

图 22.添加 ChipScope IP Core

- 双击 Hierarchy 窗口中新建的 MIPS_Scope.cdc，配置需要抓取的信号，配置触发的端口数、每个端口的位宽、采样深度和采样方式等。

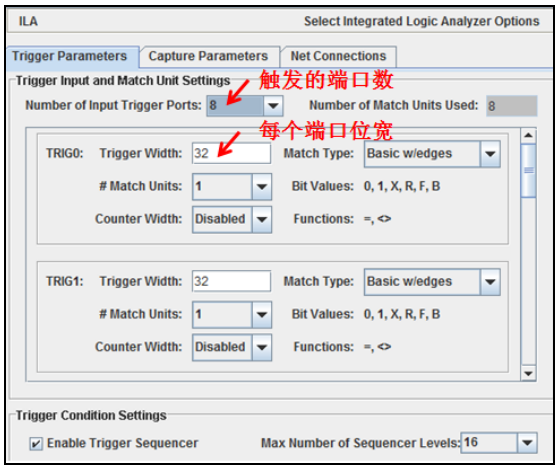

图 23. 配置触发参数

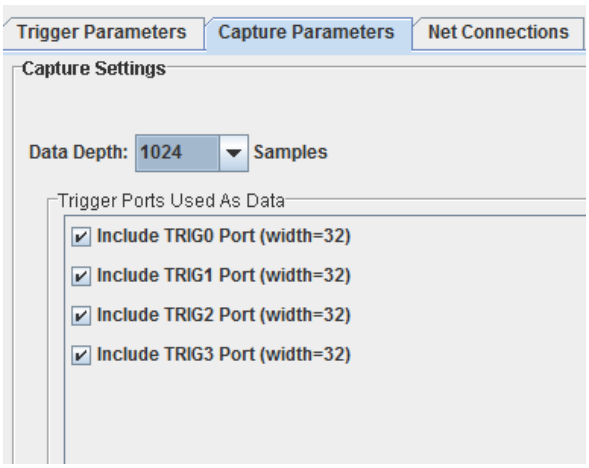

图 24.配置采样参数

- 连接需要抓取的采样信号，以及采样的时钟信号。图 25 所示，点击 Modify Connections，进入图 26 所示的窗口，在左下方窗口选择信号，右侧窗口选择需要连接的位置，然后点击 Make Connections。这里主要采样了 InstructionD[31:0]，ReadDataW[31:0]，ALUOutW[31:0]，ResultW[31:0]。

  技巧：在 Pattern 中输入 ResultW*，点击 Filter 可以搜索到前缀是 ResultW 的信号，按照 Net Name 排序之后，全部选中，在右侧窗口选择 CH:0，点击 Make Connections，32 位就同时连上了。

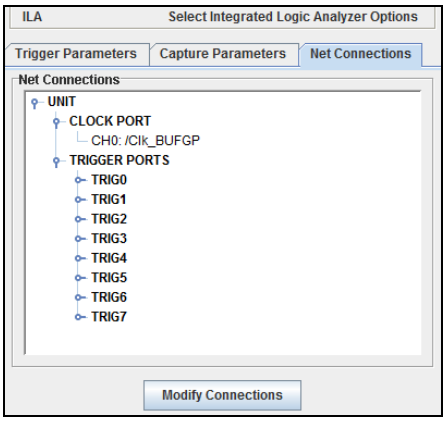

图 25. 配置连接

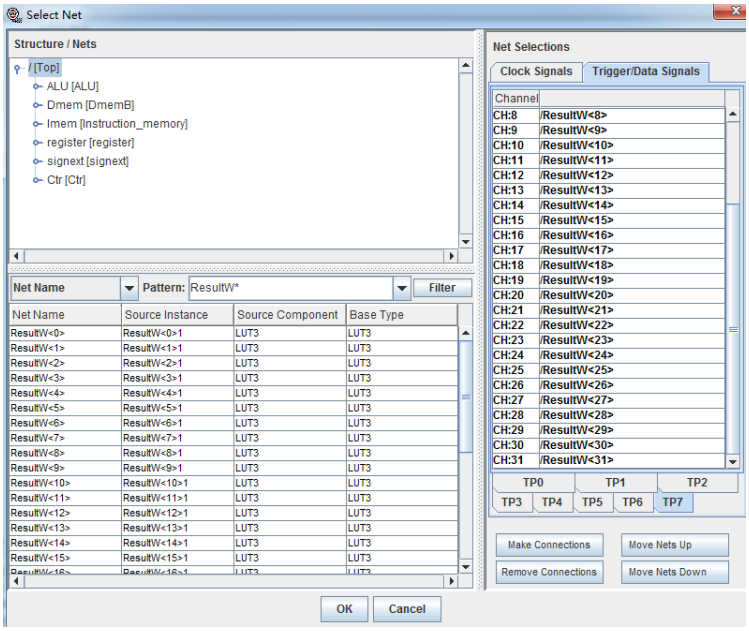

图 26. 选择网络

- 配置完之后，双击 Process 窗口中的 Generate Programming File，完成综合和实现，生成 FPGA 配置文件
- Anvyl（燧石）开发板与 5V 直流电源连接
- PC 机通过 USB 下载线与 USB PROG 端口连接，打开 Anvyl 电源，电源指示灯亮起
- 打开 Digilent Adept 工具，下载 top.bit 文件。

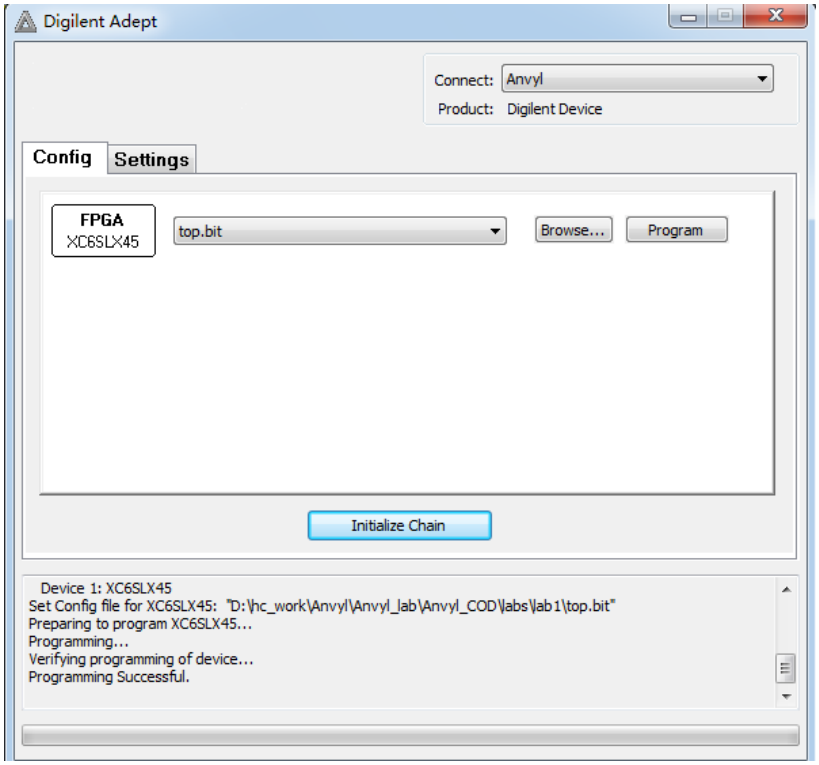

图 27. Digilent Adept 下载程序

➢ 调用 ChipScope 工具中的 Analyze，双击 Process 窗口中 Analyze Design Using ChipScope

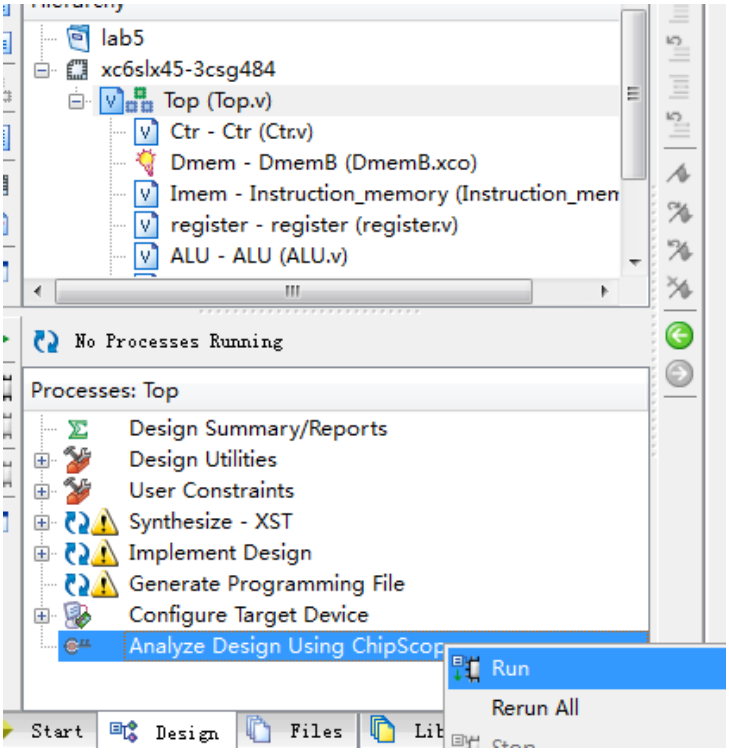

图 28 调用 ChipScope Analyzer

- 进入 Analyzer 窗口后，点击 File 下面的 JTAG-Chain 图标，以打开 JTAG 链。

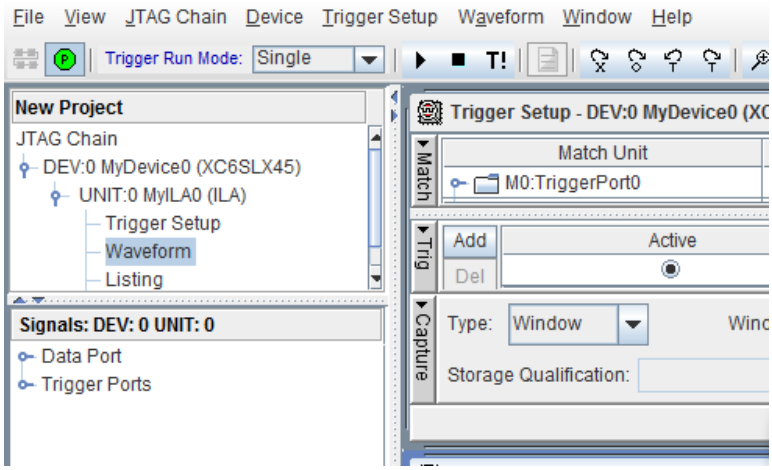

图 29. 配置 FPGA

- 点击 File-> Import，选择 CDC 文件，确认 Auto-create Buses 已勾选，点击 OK。导入 CDC 文件的好处是：所观察的信号自动分组命名，方便信号观察。

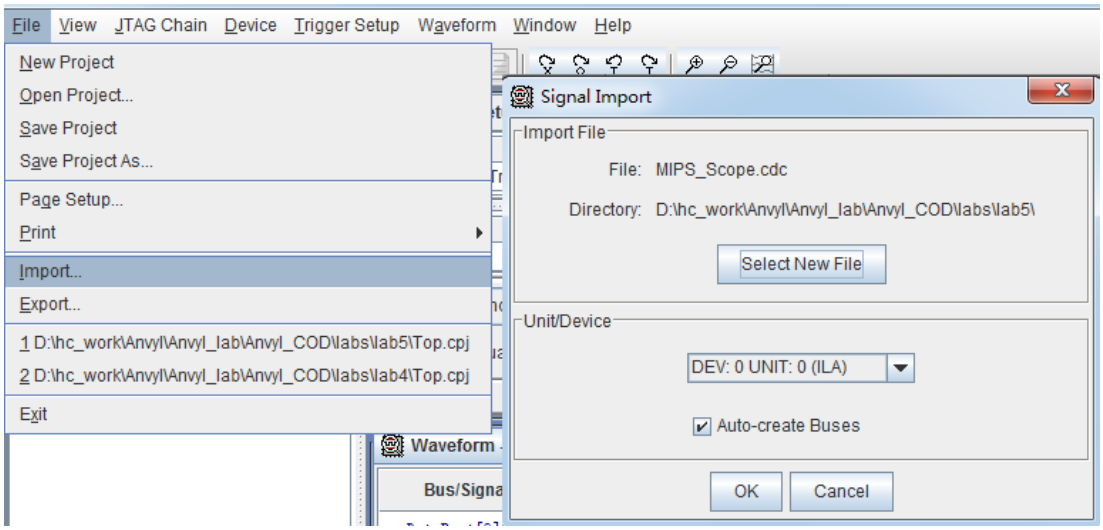

图 30. 导入 CDC 文件

- 双击 Trigger Setup 和 Waveform，得到两个窗口。按 F5 键，开始进行信号采样，采样完毕后可得到如图 30 的波形。观察分析波形

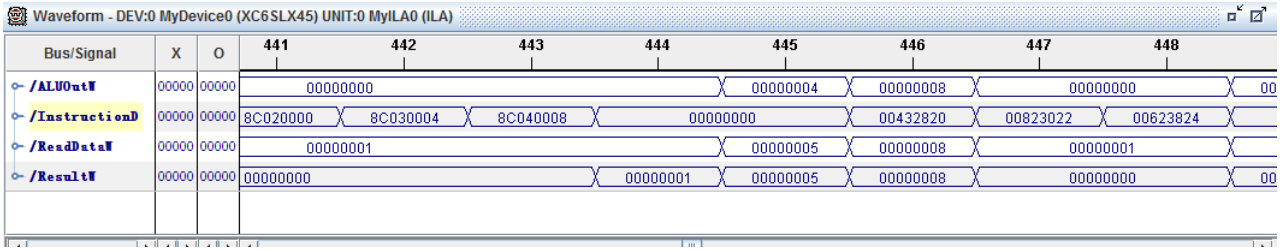

图 31. 观察运行的波形