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

Lab04: MIPS 单周期流处理器实现

Targeting on Digilent Anvyl

Contains material © Digilent, Inc.

Lab 04: MIPS 单周期处理器实现

实验简介

本实验旨在使读者实现一个简单的类 MIPS 单周期处理器。

实验目标

在完成本实验后，您将学会：

- 完成单周期的类 MIPS 处理器

实验过程

本实验旨在使读者理解和掌握单周期 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**)

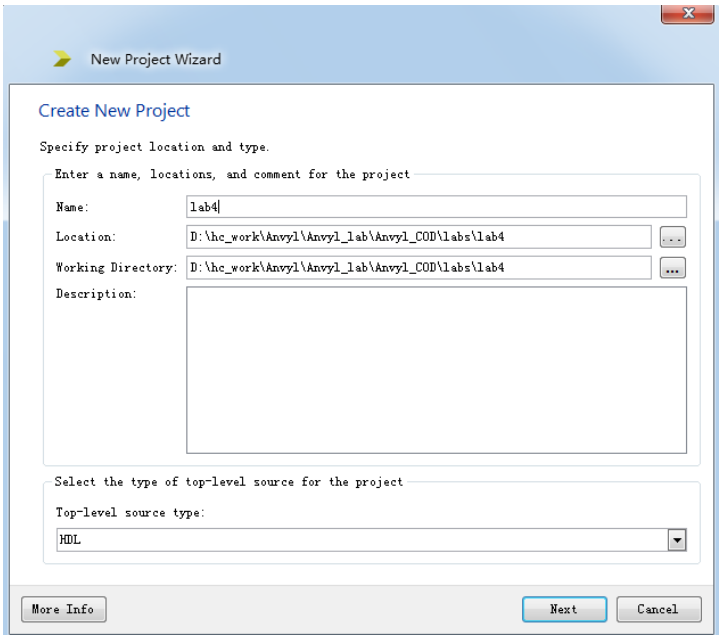

图 1. 创建新工程

- 选择 FPGA 型号、综合和仿真工具、推荐描述语言等配置

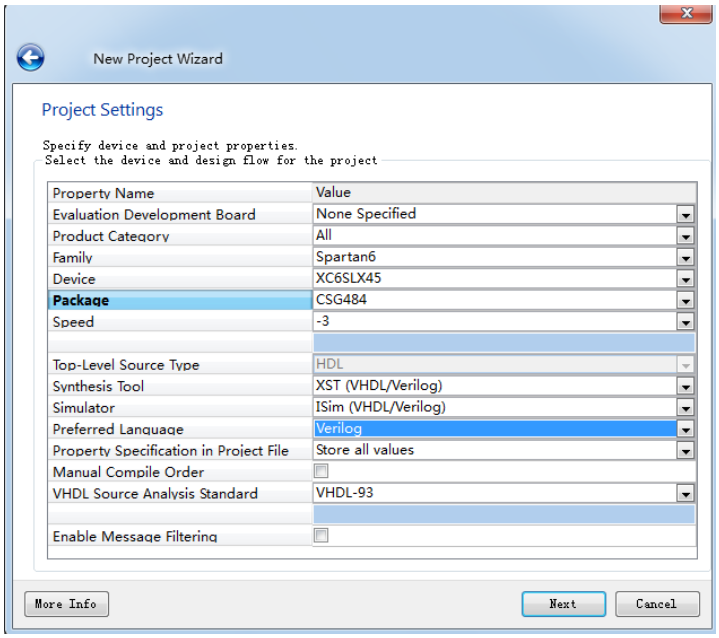

图 2. 新工程设置

- 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。

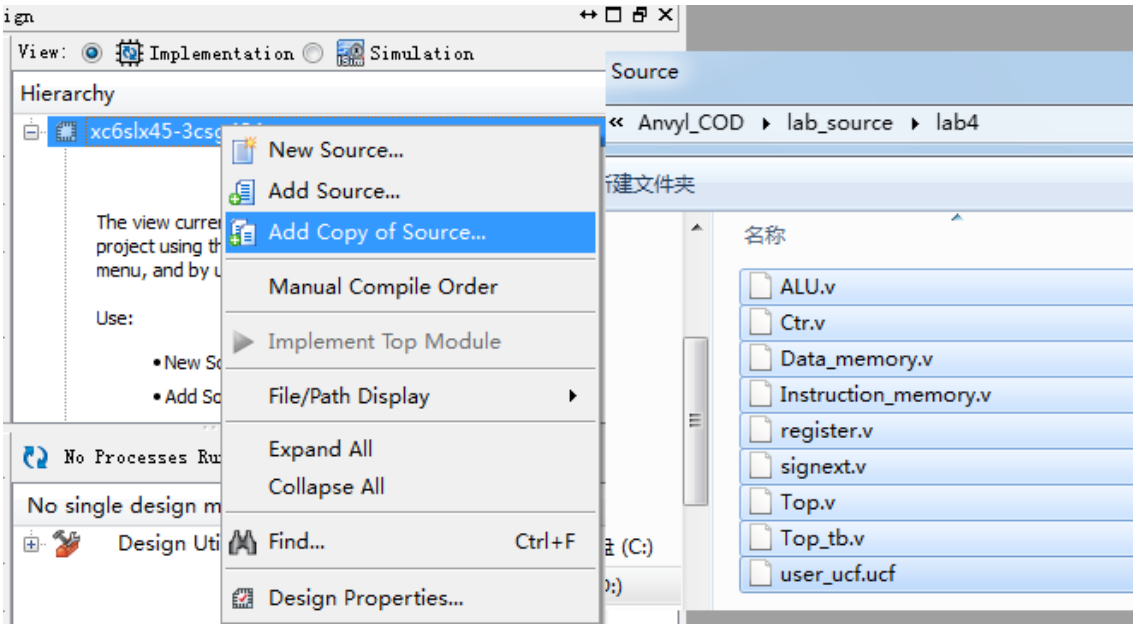

图 3.添加已有的 Verilog 模块

编写 Verilog 代码 Step 2

单周期处理器的设计，关键是确定数据路径以及确定哪些操作需要时钟，哪些不需要时钟。例如对于 Data Memory 读和写就需要区别对待。将计算好的数据写到 Data Memory 需要一个时钟周期来完成。而读 Memory 的操作，意味着将读到的数据写入到 Register file，如果读 Memory 和写 Register 都需要一个时钟周期，那就无法实现单周期的处理器。所以 Data Memory 写操作时序逻辑来完成，而读操作用组合逻辑来完成，本实验采用的就是这种方式。当然，还有其他方式来解决多周期修改成单周期，例如快慢两个时钟，这里不做细述。

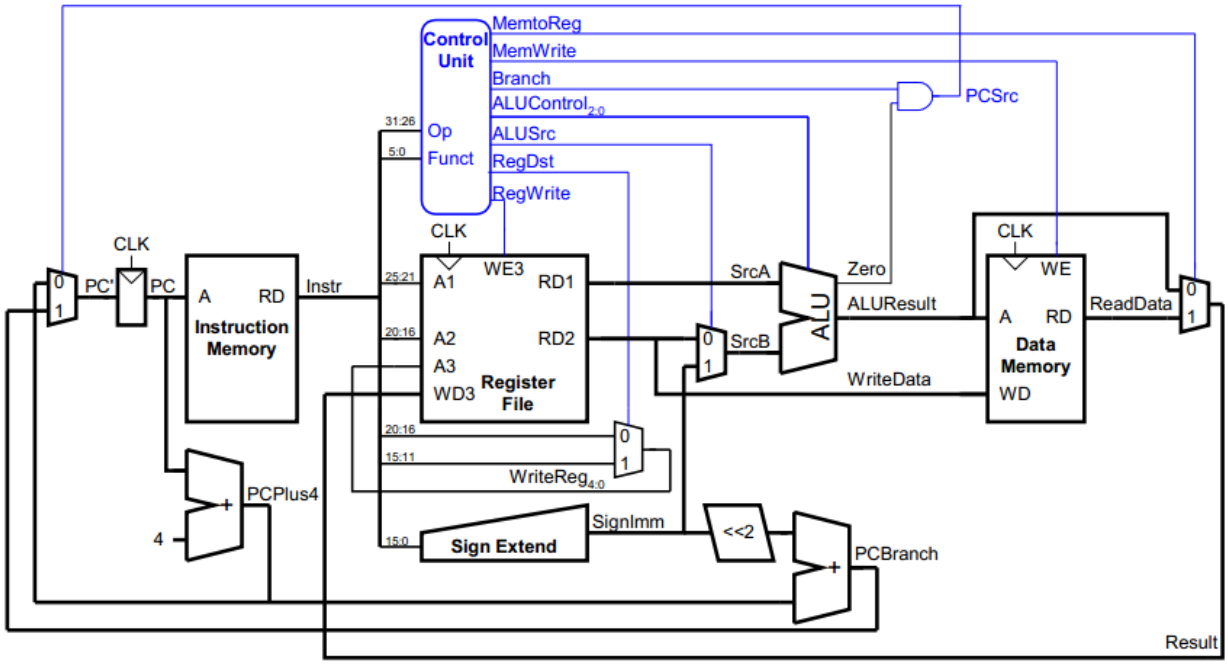

图 4. MIPS 单周期处理器原理图

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 5 中表示了一个套命名规则，可做参考。

- 根据图 4，编辑 control 模块

```
20  ////////////////////////////////
21  module Ctr(
22      input [5:0] OpCode,
23      input [5:0] Funct,
24      output reg RegDst,
25      output reg ALUSrc,
26      output reg MemToReg,
27      output reg RegWrite,
28      output reg MemWrite,
29      output reg Branch,
30      output reg [3:0] ALUControl,
31      output reg Jump
32      );
33
34  reg [1:0] ALUOp;
35
36  always @(OpCode)
37  begin
38      case(OpCode)
39
40          //R type
41          6'b000000:
```

图 5. 控制器端口定义

- 根据图 4，编辑 Top 层模块，将各个模块互联起来。

```
20  //////////////////////////////////////////////////
21
22  module Top(
23      input ClkIn,
24      input Rst,
25      input [7:0] Switch,
26     output [7:0] Led
27      );
28
29  wire Clk;
30  wire RegDst;
31  wire ALUSrc;
32  wire MemToReg;
33  wire RegWrite;
34  wire MemRead;
35  wire MemWrite;
36  wire Branch;
37  wire  Jump;
38
39  wire [5:0] Funct;
40  wire [3:0] ALUCtr;
41  wire Zero;
42
43  wire [4:0] RegARdAddr;
44  wire [4:0] RegBRdAddr;
45  wire [4:0] RegWrAddr;
46  wire [31:0] RegWrData;
47  wire [31:0] RegARdData;
48  wire [31:0] RegBRdData;
```

图 7. Top 层模块端口定义

- 设计的系统时钟暂定为 32MHz，外部时钟是 100MHz，可以添加一个 Clocking Wizard 时钟管理 IP 进行处理，得到 32MHz。（布线布局后，请阅读 P&R static timing report，查看最大时钟，要求是输入的时钟一定要低于最大时钟，否则无法正常运行。请思考如何提升时钟频率）

- 利用 Core Generator 来生成时钟，右键选中 Hierarchy 窗口，选择 New Source，左侧框中选择 IP，右侧输入文件名。如图 8 所示。

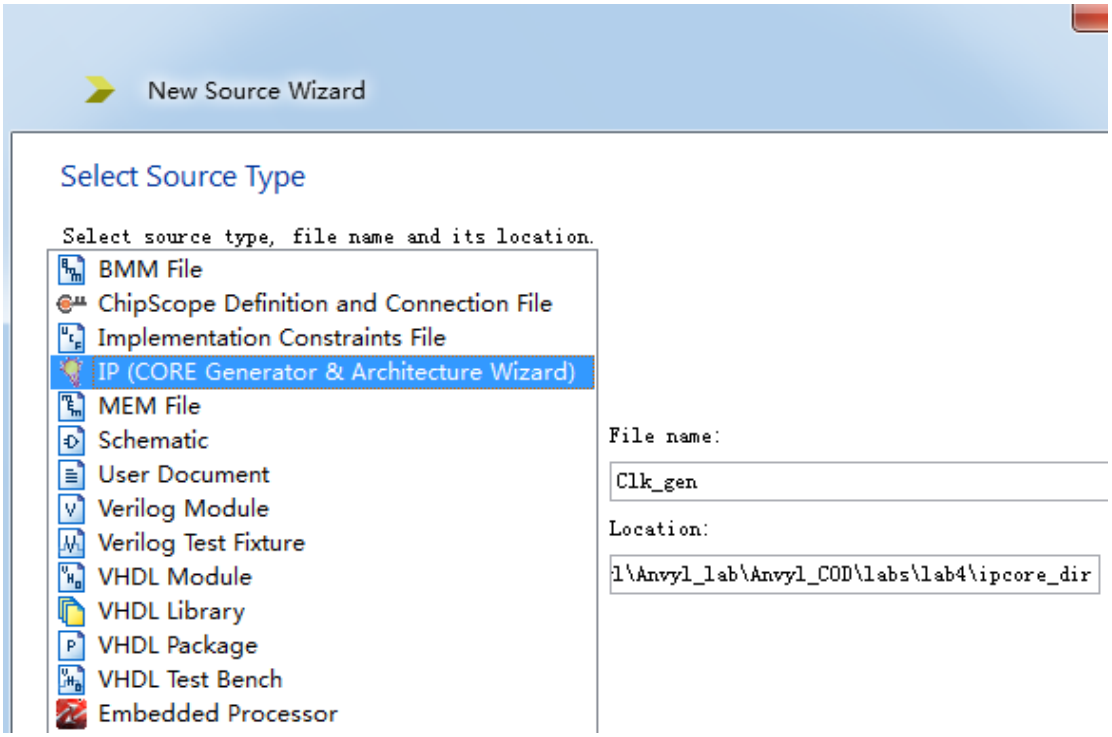

图 8. 创建 Clocking wizard 时钟管理 IP Core

➢ 选择 IP Core 类型，这里使用 Clocking Wizard。如图 8 所示。

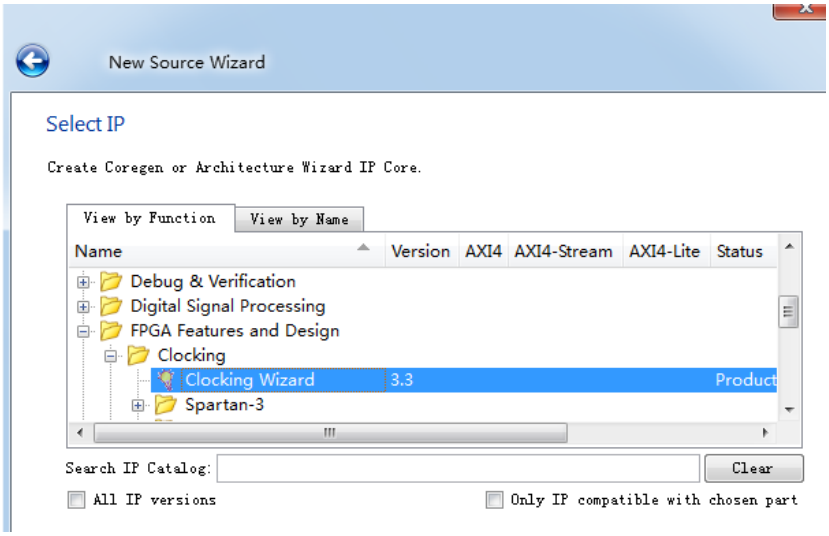

图 9. 选择 IP Core 类型

➢ 如图 10 和 11 所示，配置 Clocking Wizard 时钟参数。

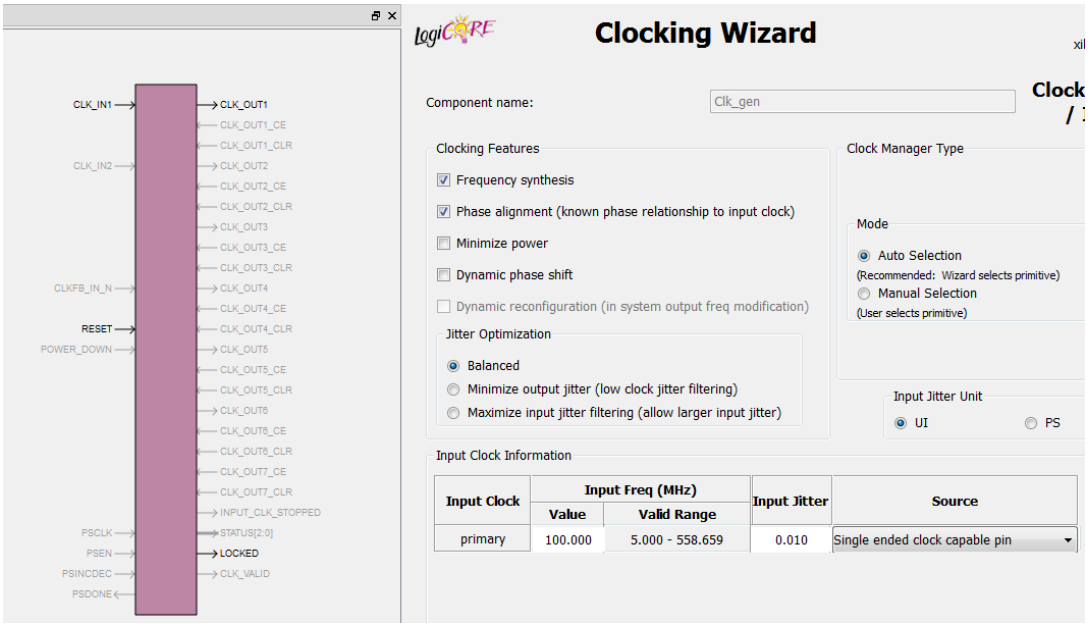

图 10，100MHz 时钟输入

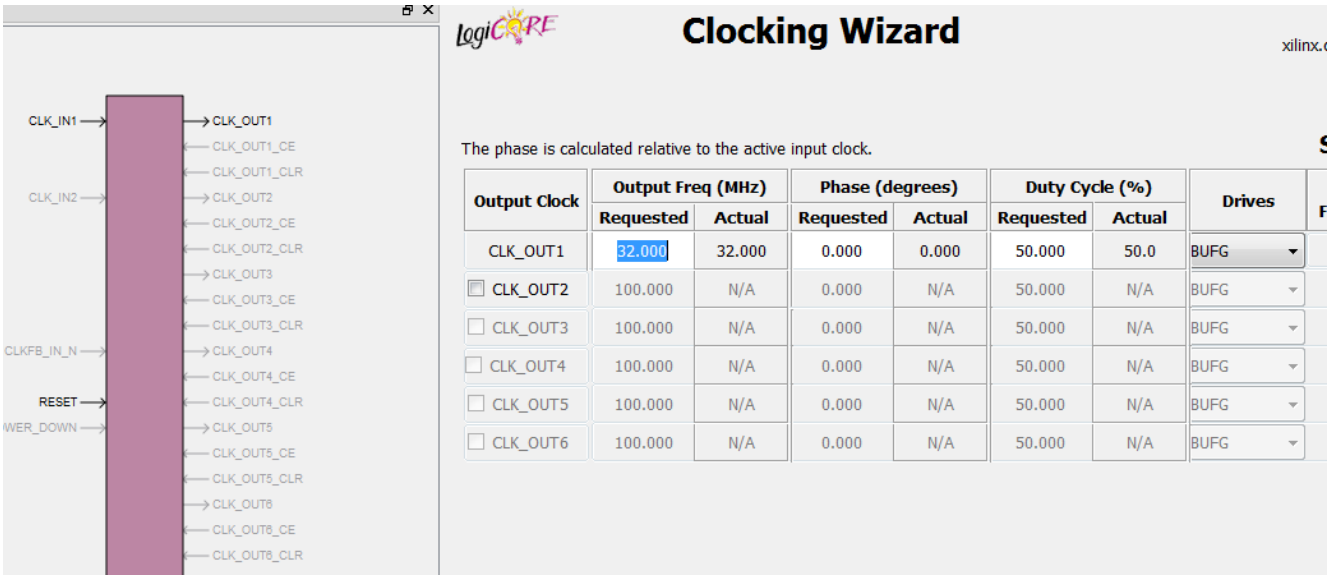

图 11，32MHz 时钟输入

- 在 Top 模块中调用 Clk_gen DCM 模块。

```
Clk_gen  Clock(
    .CLK_IN1(ClkIn),
    .RESET(Rst),
    .CLK_OUT1(Clk),
    .LOCKED()
);
```

图 12，调用 Clk_gen 模块

仿真测试 Step 3

整个处理器设计基本完成，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 13 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。

```
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 13. 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]	00000000
[0x00000010]	00000000
[0x00000014]	00000000
[0x00000018]	00000000
[0x0000001c]	00000000
[0x00000020]	00000000
[0x00000024]	00000000
[0x00000028]	00000000
[0x0000002c]	00000000
[0x00000030]	00000000
[0x00000034]	00000000
[0x00000038]	00000000
[0x0000003c]	00000000
…	…

- 编写 Top 层的 testbench 文件，右键选中 Hierachy 窗口，选择 new source。如图 14 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。

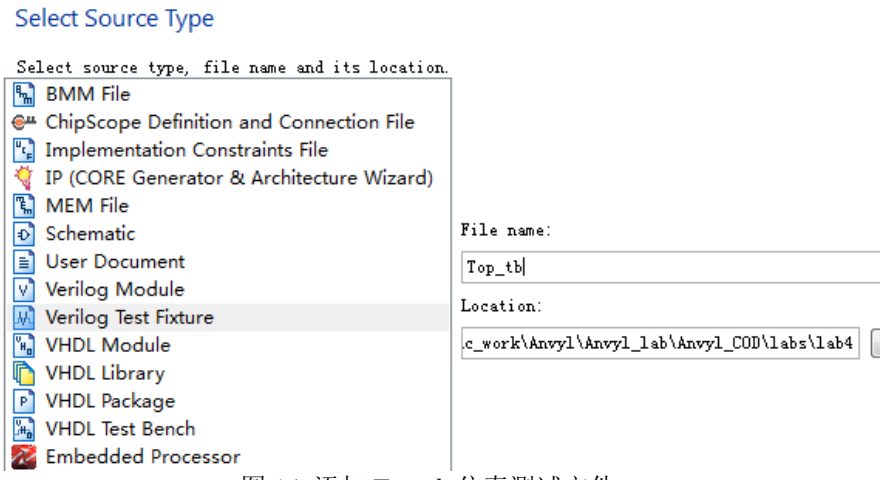

图 14. 添加 Top_tb 仿真测试文件

- 添加时钟激励和其他输入信号的初始化。

```
    // Instantiate the Unit Under Test (UUT)
    Top uut (
        .ClkIn(ClkIn),
        .Rst(Rst),
        .Led(Led)
    );

    initial begin
        // Initialize Inputs
        ClkIn = 0;
        Rst = 0;
        Switch = 0;
        #20
        Rst = 1;
        // Wait 100 ns for global Rst to finish
        #80;
         Rst = 0;

        // Add stimulus here

    end
    always #5 ClkIn = !ClkIn;

endmodule
```

图 15. 编辑 TestBench 文件

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

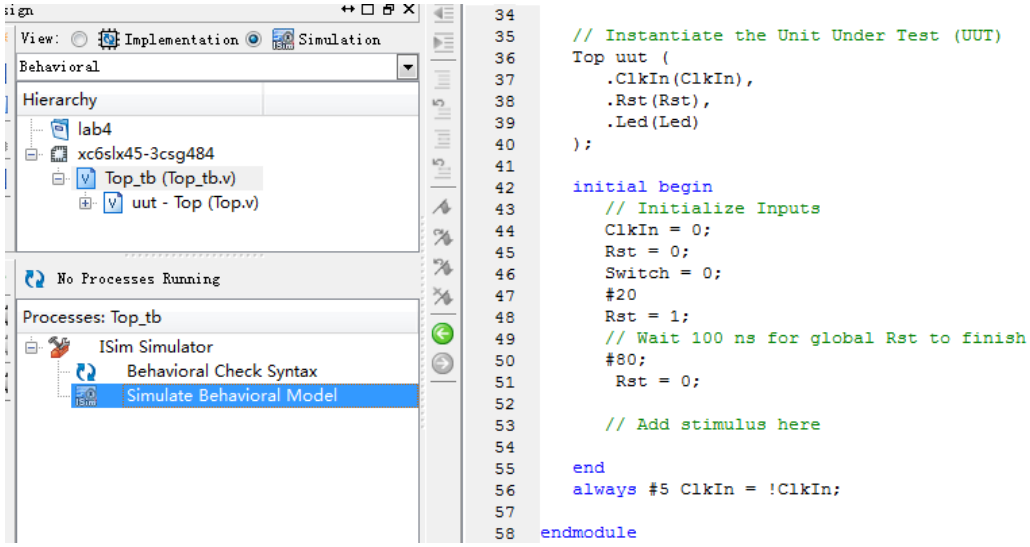

图 16. 调用 ISim 仿真工具

- 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 Instruction[31:0]、PC[31:0]以及 Clk 信号到波形窗口。

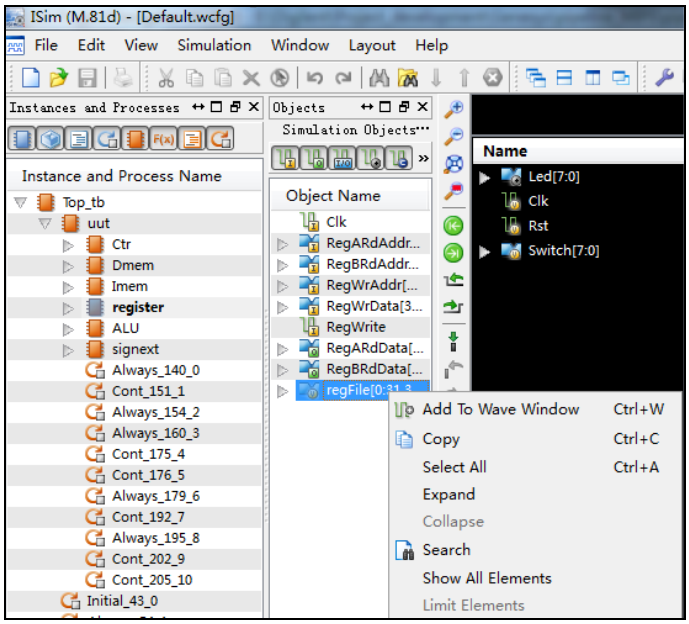

图 17. 添加 regfile 到波形窗口

- 在 Console 窗口中输入 restart；run 2000ns，重新进行仿真。观察如图 18 的波形与表 1 所提供的 MIPS 指令结果对比。

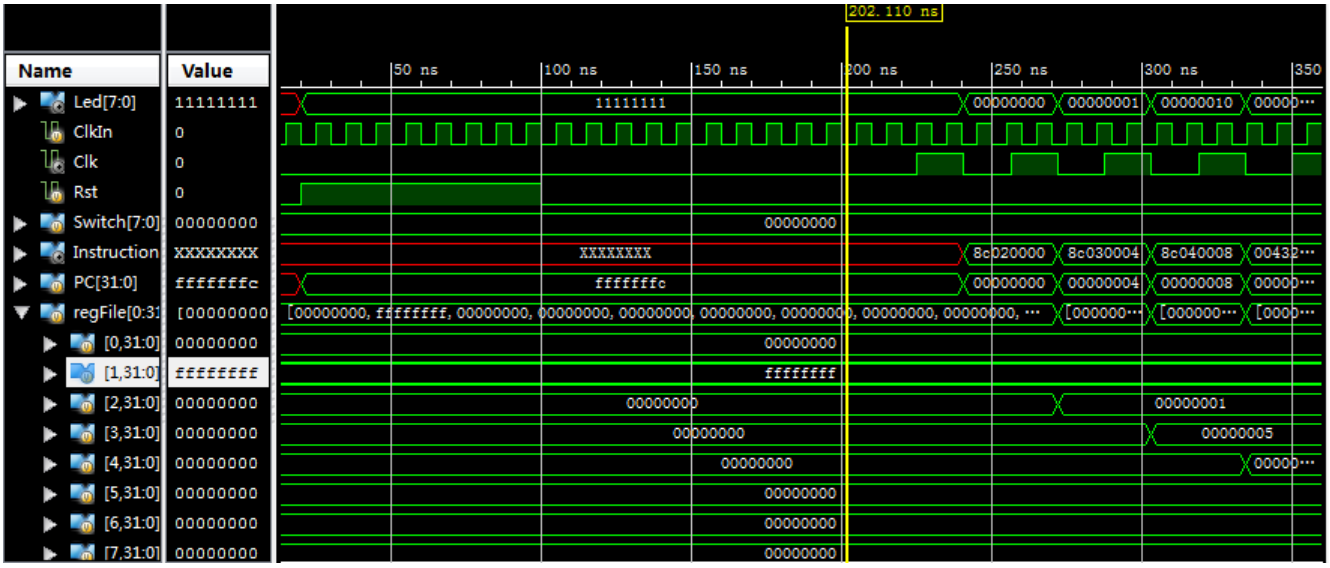

图 18. 仿真波形

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。

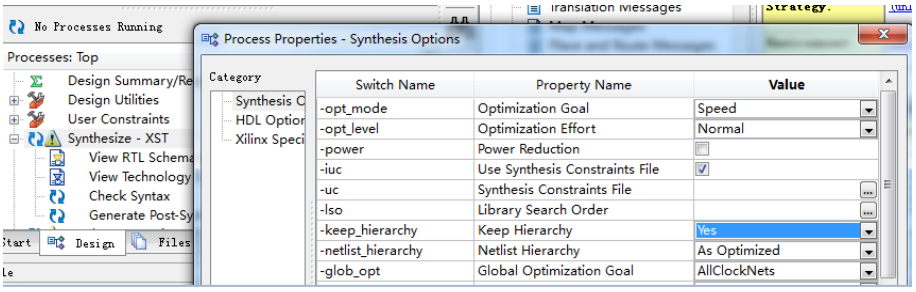

图 19. 配置 FPGA 综合参数

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

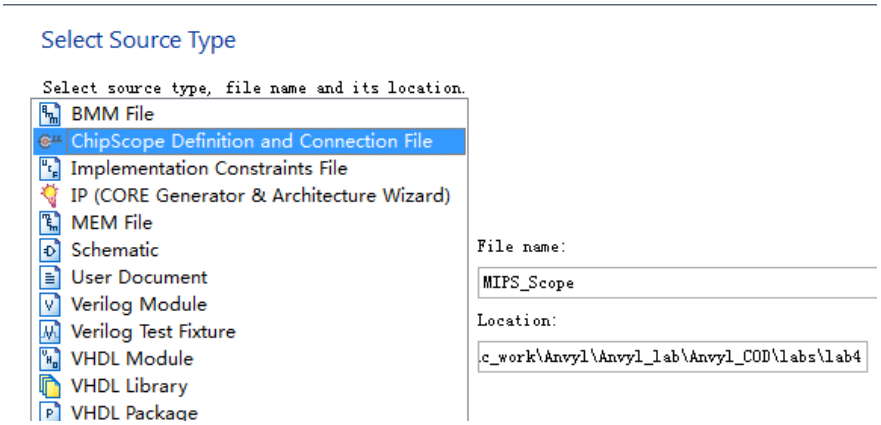

图 20.添加 ChipScope IP Core

- 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。

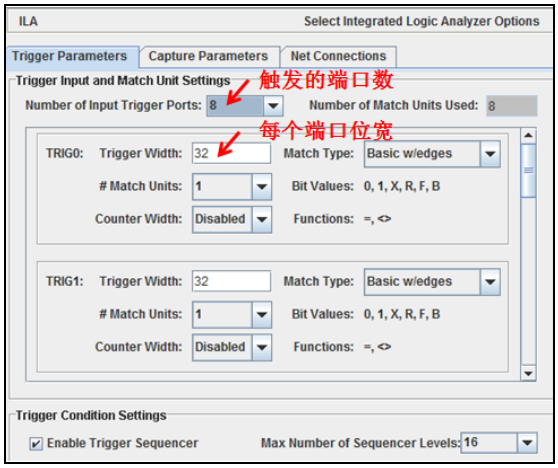

图 21. 配置触发参数

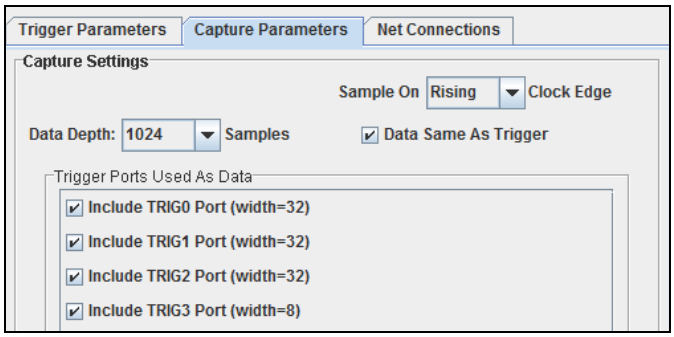

图 22.配置采样参数

- 连接需要抓取的采样信号，以及采样的时钟信号。图 23 所示，点击 Modify Connections，进入图 24 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 Instruction[31:0]，PC[9:2]，ALURes[31:0]，DmemRdData[31:0]。

  技巧：在 Pattern 中输入 Instruction*，点击 Filter 可以搜索到前缀是 Instruction 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。

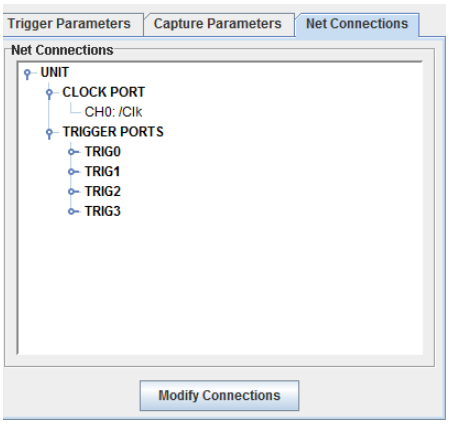

图 23. 配置连接

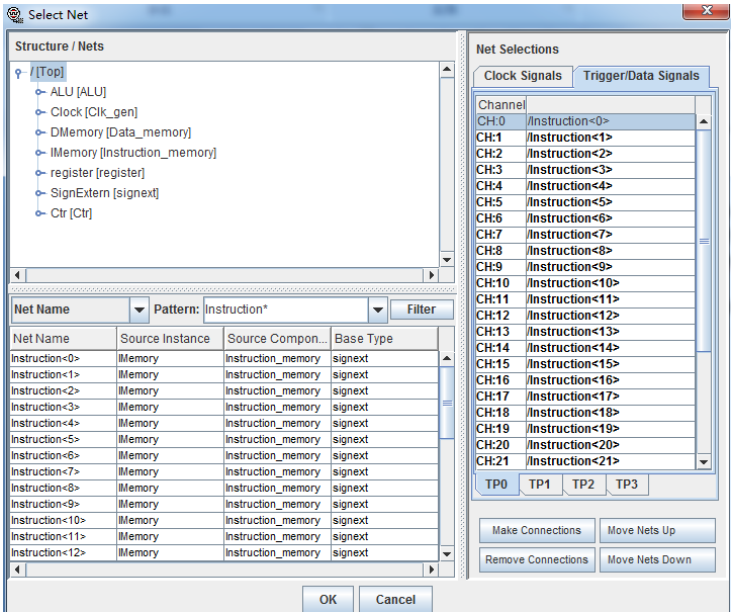

图 24. 选择网络

- 配置完之后，开始综合和实现，在 process 栏中双击 Generate Programming File，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。

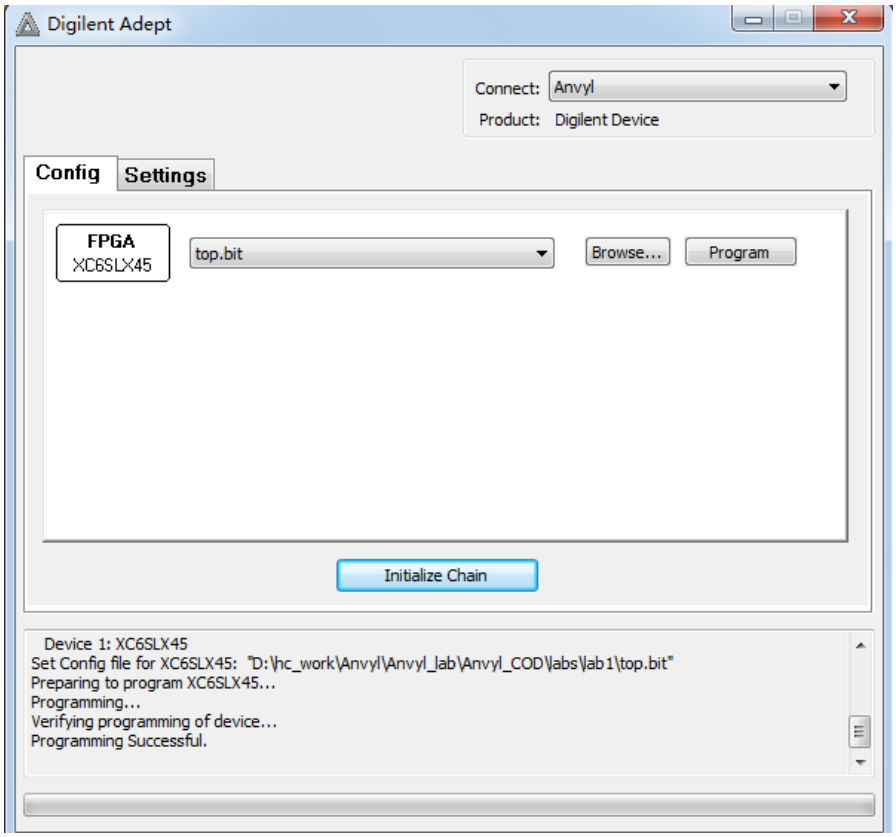

图 25. Digilent Adept 下载程序

- 调用 ChipScope 工具中的 Analyze，双击 Process 窗口中 Analyze Design Using ChipScope。也可以右键点击 Run。

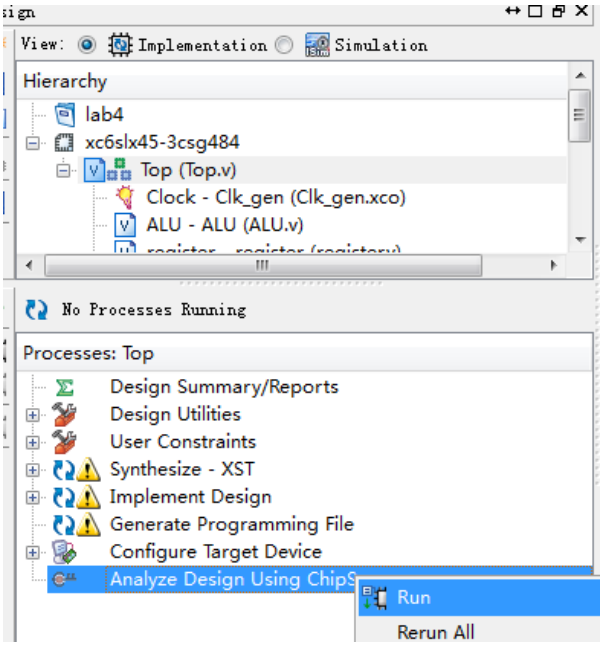

图 26. 调用 ChipScope Analyzer

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

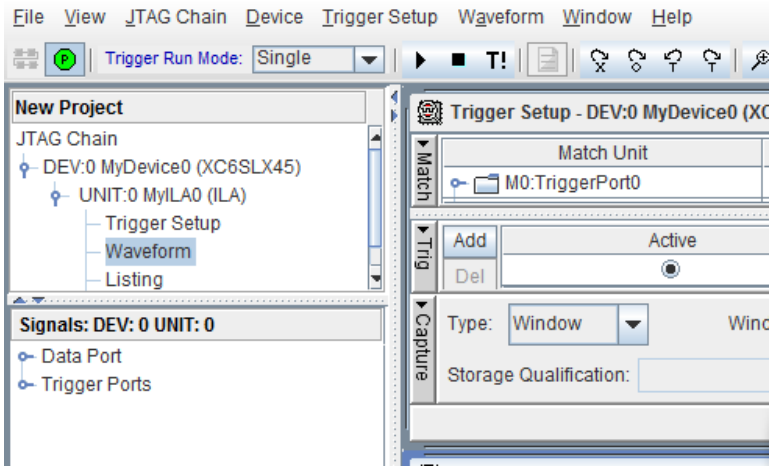

图 27. 配置 FPGA

- 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。

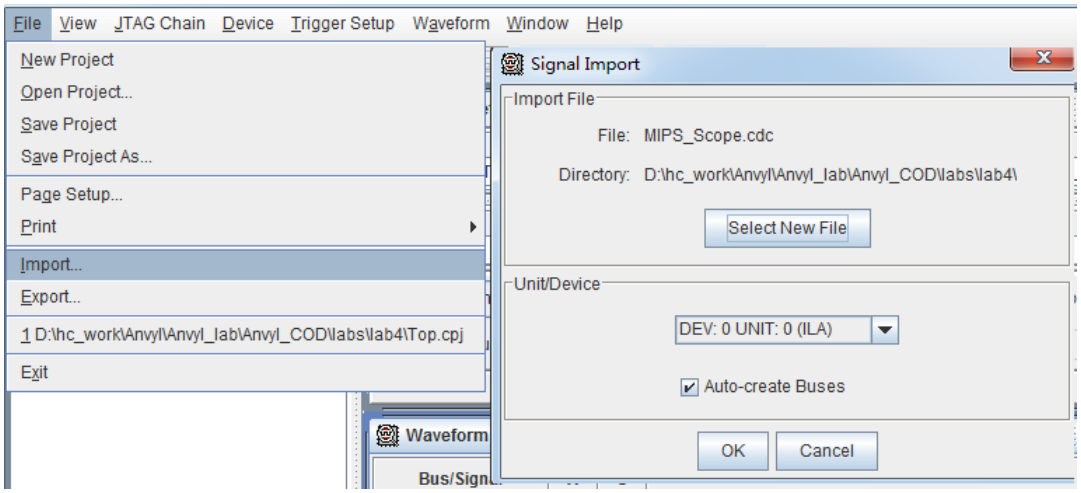

图 28. 导入 CDC 文件

- 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 28 的波形。观察分析波形

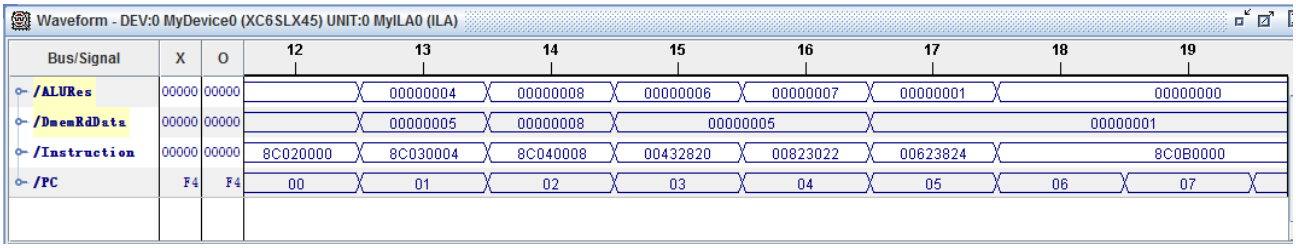

图 29 观察运行的波形