电子科技大学计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 计算机系统结构综合实验

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名：王涛　　学 号：2020080902001　　指导教师：王华

实验地点：清水河校区主楼 A2 区 413-1　　实验时间：2023.4.20

一、实验室名称：国家级计算机实验教学示范中心

二、实验项目名称：单周期 CPU 代码分析

三、实验学时：4

四、实验原理：

一）单周期 CPU 的特点是：在一条指令的所有操作全部完成后，才开始执行下一条指令。它执行一条指令需要的硬件部件有：

1. 与取指令有关的电路：指令存储器、程序计数器 PC、修改 PC 值的加法器、选择不同 PC 值的多路选择器等；

2. 与数据处理有关的电路，具体说来包括下述电路：

1）算术逻辑运算部件，如 ALU；

2）与寄存器有关的部件，如寄存器堆；

3）与存储器有关的部件，如数据存储器；

4）与处理方式有关的部件，如多路选择器（选择器的具体规格视具体情况而定）；

5）与立即数处理有关的部件，如数据扩展器、移位器等。

二）硬件描述语言（Hardware Description Languages），是设计硬件时使用的语言，常用的有 Verilog HDL 和 VHDL。两者的异同点包括：

1. 两者均为常用硬件描述语言，都有各自的 IEEE 标准；

2. 两者是不同的硬件描述语言；

3. Verilog HDL 带有 C 语言的风格，是工业界常用的 HDL；

4. VHDL 带有 C++的风格；

【说明】本课程只使用 Verilog HDL 语言，因此报告中均采用 Verilog HDL 语言进行器件设计。

三）本次课程的软件环境介绍：

1. 操作系统：Windows 7

2. 开发平台：Xilinx ISE Design Suite 14.7 集成开发系统

四）本实验课所有设计的 CPU 支持的指令集（32 位）如表 1 所示，其中：

- Op 和 func 为操作码；

- shift 保存要移位的位数；
- rd、rs、rt 表示寄存器号；
- immediate 保存立即数的低 16 位；
- offset 为偏移量；
- address 为转移地址的一部分。

表 1

指令	指令意义	Op[31:26]	func[25:20]	[19:15]	[14:10]	[9:5]	[4:0]
add	寄存器加法	000000	000001	00000	rd	rs	rt
and	寄存器与	000001	000001	00000	rd	rs	rt
or	寄存器或	000001	000010	00000	rd	rs	rt
xor	寄存器异或	000001	000100	00000	rd	rs	rt
srl	逻辑右移	000010	000010	shift	rd	00000	rt
sll	逻辑左移	000010	000011	shift	rd	00000	rt
addi	立即数加法	000101	16 位 immediate			rs	rt
andi	立即数与	001001	16 位 immediate			rs	rt
ori	立即数或	001010	16 位 immediate			rs	rt
xori	立即数异或	001100	16 位 immediate			rs	rt
load	取整数数据字	001101	16 位 offset			rs	rt
store	存整数数据字	001110	16 位 offset			rs	rt
beq	相等则跳转	001111	16 位 offset			rs	rt
bne	不相等则跳转	010000	16 位 offset			rs	rt
jump	无条件跳转	010010	26 位 address				

指令功能说明如下：

1、对于 add/and/or/xor　rd,rs,rt 指令　//rd←rs op rt　其中 rs 和 rt 是两个源操作数的寄存器号，rd 是目的寄存器号。

2、对于 sll/srl　rd,rt,shift 指令　//rd←rt 移动 shift 位

3、对于 addi　rt,rs,imm 指令　//rt←rs+imm(符号拓展)　rt 是目的寄存器号，立即数要做符号拓展到 32 位。

4、对于 andi/ori/xori　rt,rs,imm 指令　//rt←rs op imm(零拓展) 因为是逻辑指令，所以是零拓展（32 位）。

5、对于 load　rt,offset(rs)　指令　//rt←memory[rs+offset]　load 是一条取存储器字的指令。寄存器 rs 的内容与符号拓展的 offset 相加得到存储器地址。从存储器取来的数据存入 rt 寄存器。

6、对于 store　rt,offset(rs)　指令　// memory[rs+offset]←rt　store 是一条存字指令。存储器地址的计算方法与 load 相同。

7、对于 beq　rs,rt,label 指令　//if(rs==rt)　PC←label

beq 是一条条件转移指令。当寄存器 rs 内容与 rt 相等时，转移到 label。如果程序计数器 PC 是 beq 的指令地址，则 label=PC+4+offset<<2。offset 左移两位导致 PC 的最低两位永远是 0，这是因为 PC 是字节地址，而一条指令要占 4 个字节。offset 要进行符号拓展，因为 beq 能实现向前和向后两种转移。

8、bne 指令与 beq 类似，当寄存器 rs 内容与 rt 不相等时，转移到 label。
9、对于 jump　target 指令　//PC←target　jump 是一条跳转指令。target 是转移的目标地址，32 位，由 3 部分组成：最高 4 位来自于 PC+4 的高 4 位，中间 26 位是指令中的 address，最低两位为 0。

五、实验目的：

1. 掌握单周期 CPU 的特点；
2. 熟悉 Verilog HDL 硬件设计语言设计方法及其相关语法等；
3. 熟悉 Xilinx ISE Design Suite 14.7 集成开发环境的操作方法和仿真方法。

六、实验内容：

1. 认真阅读并分析所给的单周期 CPU 代码，掌握单周期 CPU 电路结构中各模块的工作原理；
2. 对单周期 CPU 中两个模块进行仿真，分析并理解仿真结果，验证模块逻辑功能：
1）IF_STAGE（取指阶段）；
2）Control_Unit（控制单元）；
3. 自行设计一个指令序列，要求尽可能涵盖 CPU 指令集中不同类型的指令。将指令序列写入指令存储器 inst_mem 中，使用该指令序列对指定的 SCCPU.v（单周期 CPU 完整电路模块）进行仿真，记录仿真结果并进行分析，直至仿真结果与预期相符。

七、实验器材：

PC 机
操作系统：Windows 7
开发平台：Xilinx ISE Design Suite 14.7 集成开发系统

八、实验步骤：

step1：创建工程（Project）
1）启动 ISE 软件，然后选择菜单 File→New Project，弹出 New Project Wizard 对话框，在对话框中输入，并指定工作路径，如图 1 所示。

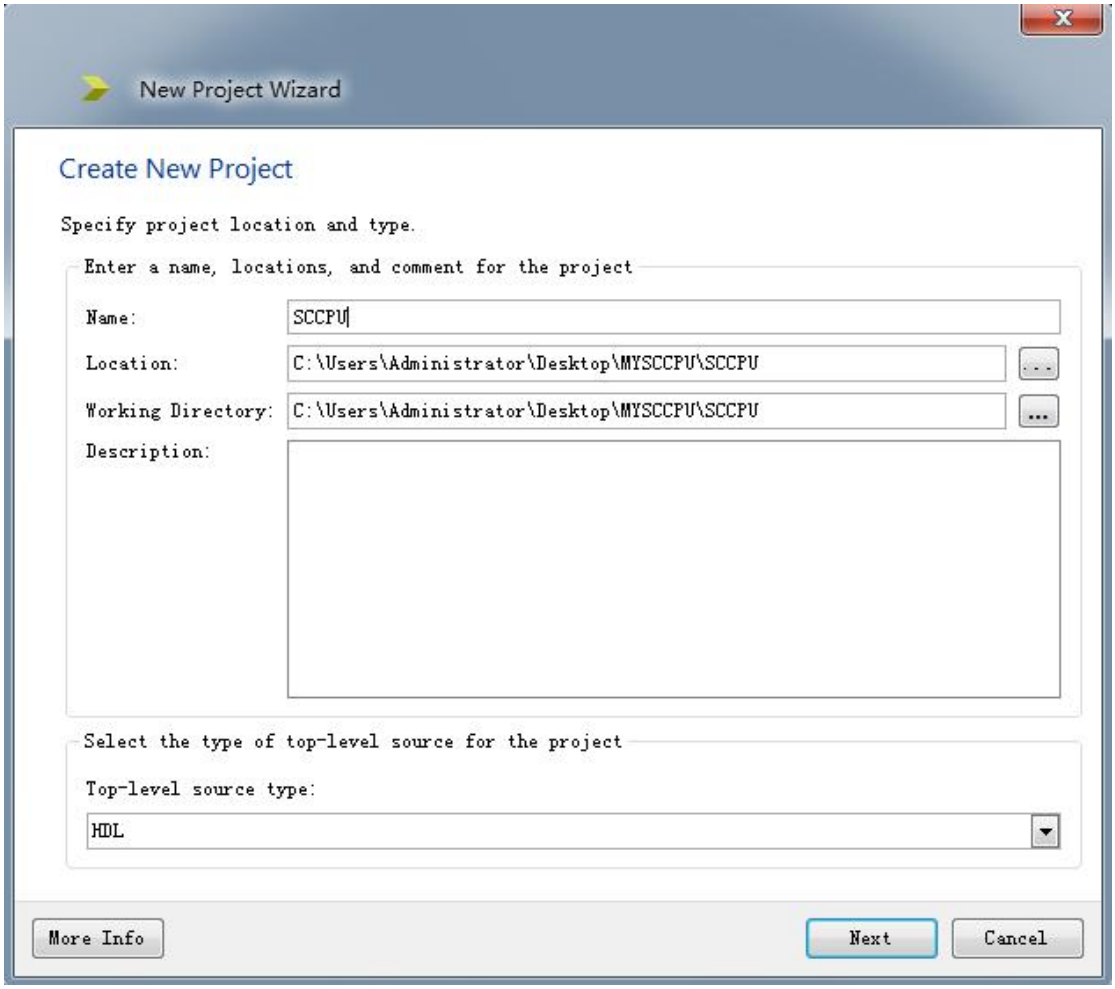


图 1

图 1 中指定的工程名为 SCCPU，指定的工作路径为：C:\Users\Administrator\Desktop\MYSCCPU\SCCPU

2）在上图中输入完工程名和工作路径后，点击 Next 进入下一页：Project Settings。如图 2 所示：

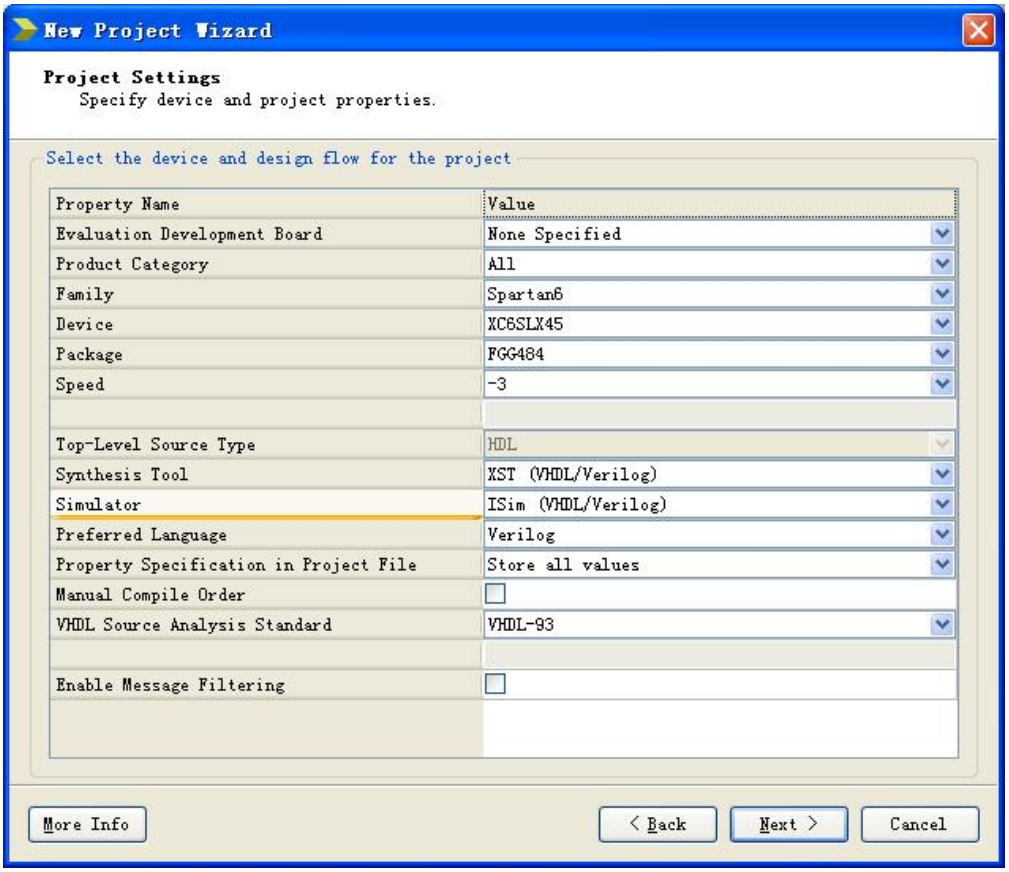


图 2

3）设置好后，在上图中点击 Next 进入下一页：Project Summary，如图 3 所示，这里显示了新建工程的信息，确认无误后，点击 Finish 按钮就可以建立一个新的工程了。

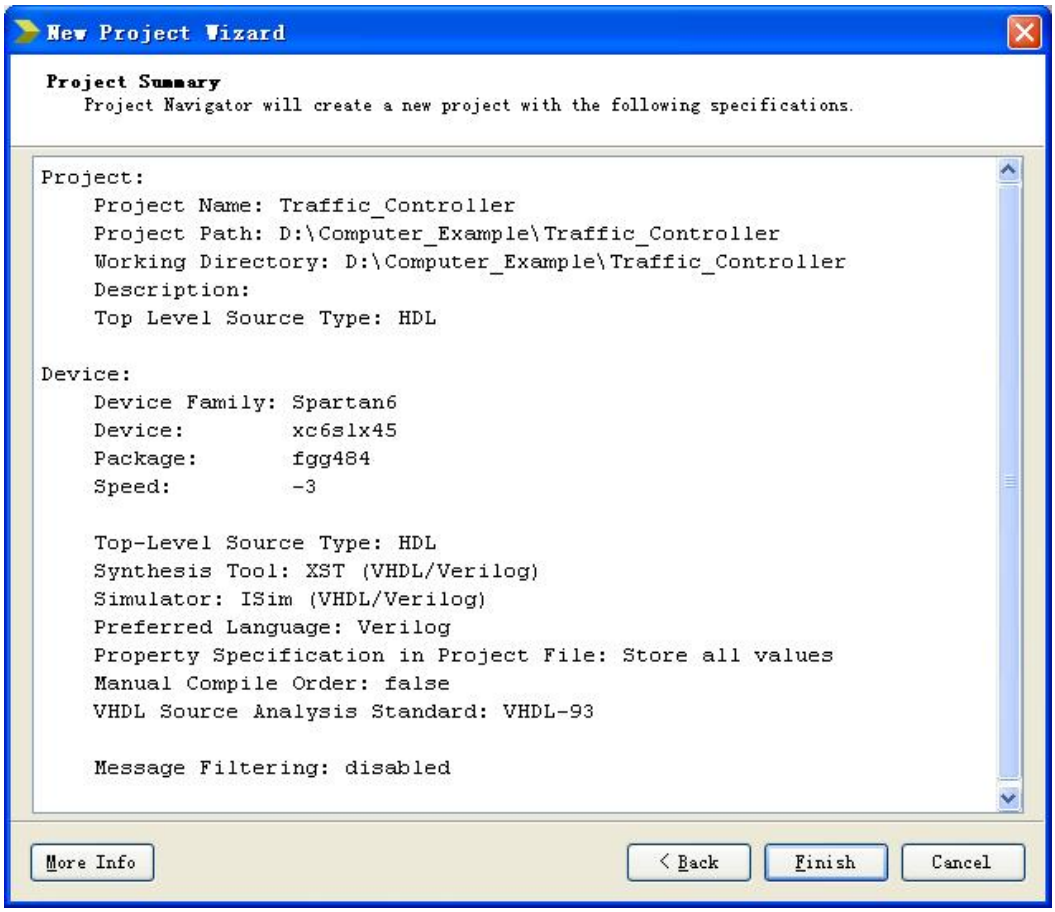


图 3

step2：创建模块（Module）

1）ISE 集成开发环境如图 4 所示，主要分为 4 个区：工程管理区、过程管理区、源代码编辑区和信息显示区。

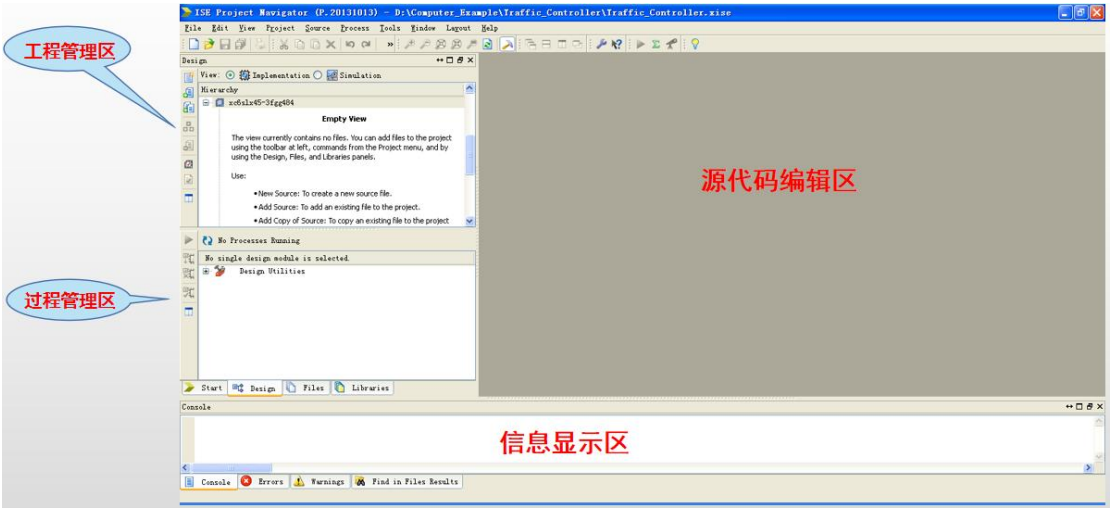


图 4

2）在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source，会弹出如图 5 所示的 New Source Wizard 对话框：Select Source Type。

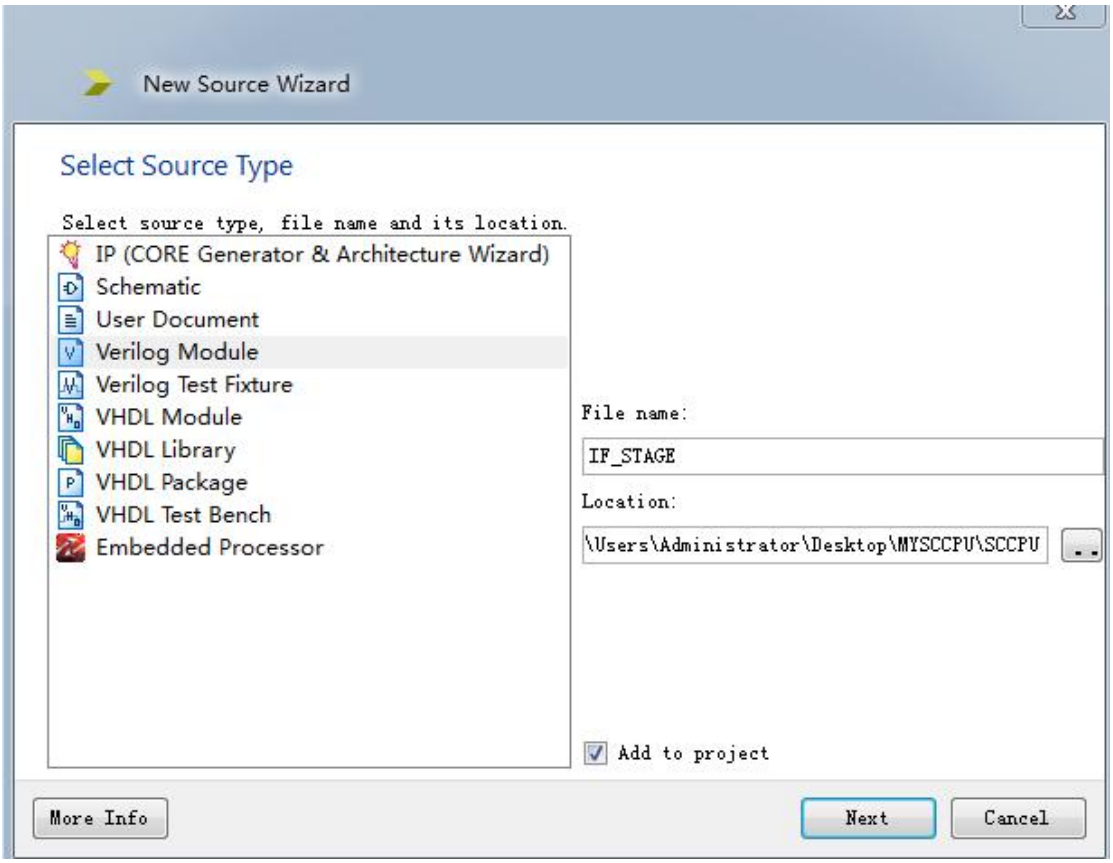


图 5

① 在图中选择 Verilog Module；

② 输入 Verilog 文件名，如： IF_STAGE；

③ 输入代码存放位置（Location），如 C:\Users\Administrator\Desktop\MYSCCPU\SCCPU；

④ 点击 Next；

3）接下来进入端口定义对话框：Define Module，如图 6 所示。

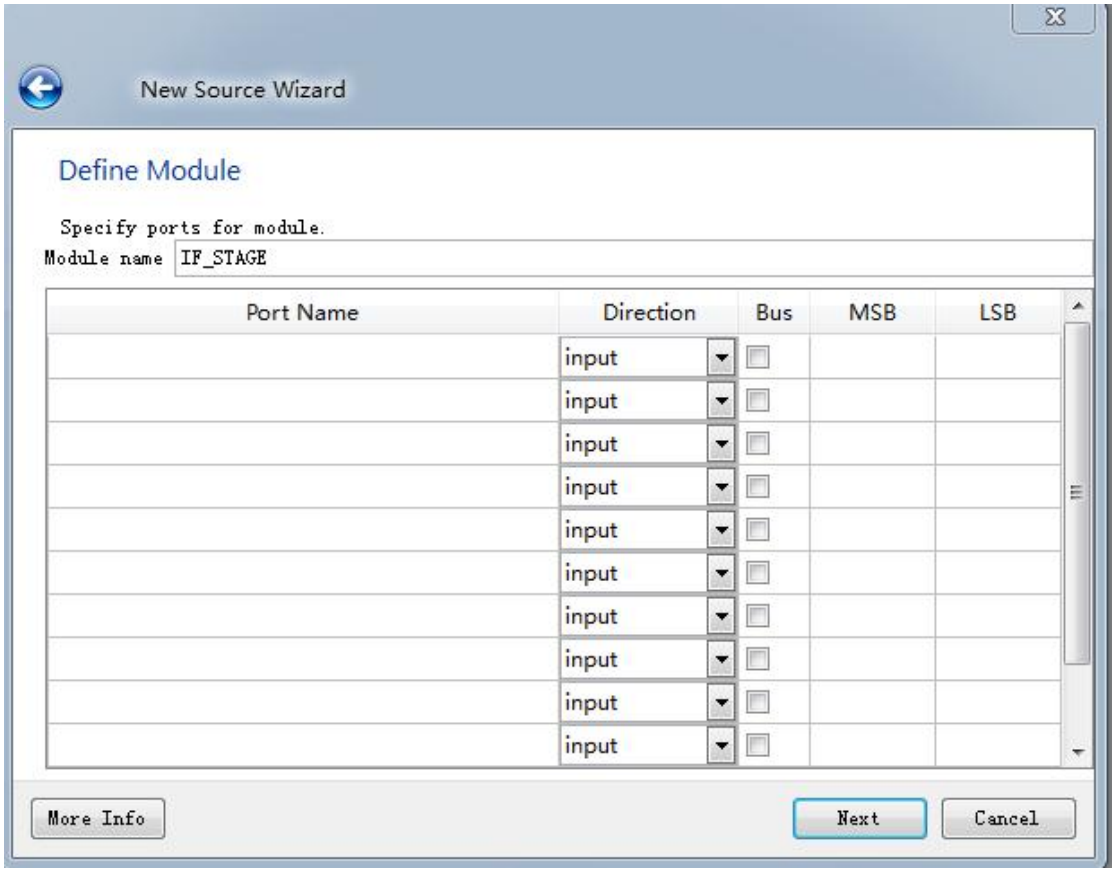


图 6

① Module name 栏用于输入模块名，比如 IF_STAGE.

② 下面的列表框用于端口的定义，端口定义这一步略过，在源程序中自行添加。其中：

Port Name 表示端口名称；

Direction 表示端口方向(可选择为 input、output 或 inout)；

MSB 表示信号最高位；

LSB 表示信号最低位；

③ 定义完端口后，点击 Next 按钮进入下一步，然后点击 Finish 按钮完成模块创建。

4）开始进行模块设计，代码输入完成后，首先检查 Verilog HDL 语法：在工程管理区选中要检查的模块，在过程管理区双击 Synthesize － XST→Check Syntax。

① 如果有语法错误，会在信息显示区给出指示，请检查调试。

② 如果没有语法错误，在模拟仿真前要进行 Verilog HDL 代码综合。

5）所谓综合，就是将 Verilog HDL 语言、原理图等设计输入翻译成由与、或、非门和 RAM、触发器等基本逻辑单元的逻辑连接，并根据目标和要求（约束条件）优化生成的 RTL（Register-Transfer-Level）层连接。在工程管理区的 View 中选择 Implementation，并选中要综合的模块 Controller，然后在过程管理区双击 Synthesize-XST，就开始综合过程，如图 7 所示。

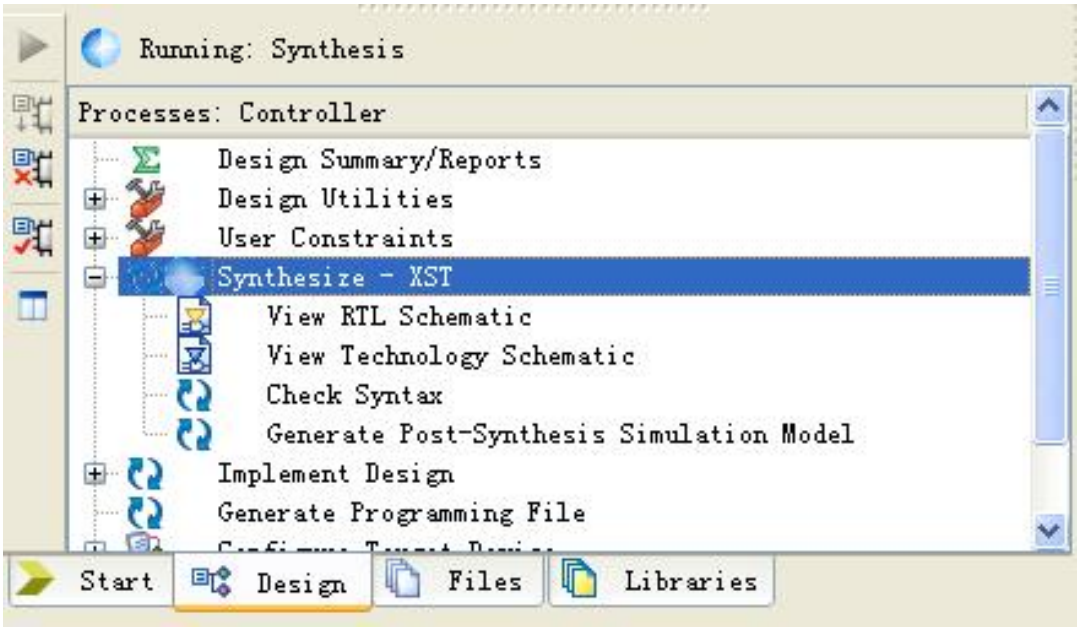


图 7

step3：仿真（Simulation）

仿真并不是设计过程必须的步骤。但是为了尽量减少设计中的错误，在将所做设计下载到开发板上进行板级验证之前，对所做设计进行仿真是必要的。

在工程管理区任意位置单击鼠标右键，在弹出的菜单中选择 New Source，会弹出如图 8 所示的 New Source Wizard 对话框：Select Source Type。

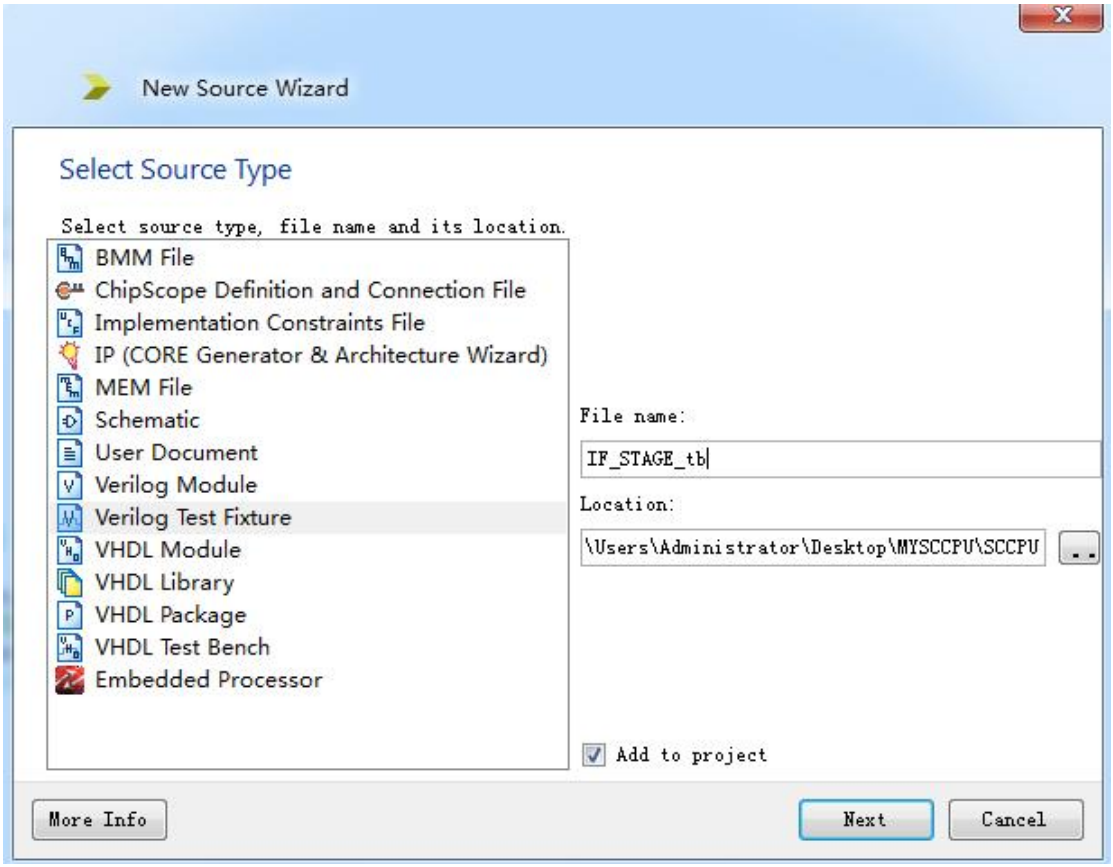


图 8

在图 8 中选择 Verilog Test Fixture，输入测试文件名：IF_STAGE_tb，单击

Next 按钮，进入下一个对话框，如图 9 所示。

在图 9 中会显示工程中所有的模块名，选择 IF_STAGE。

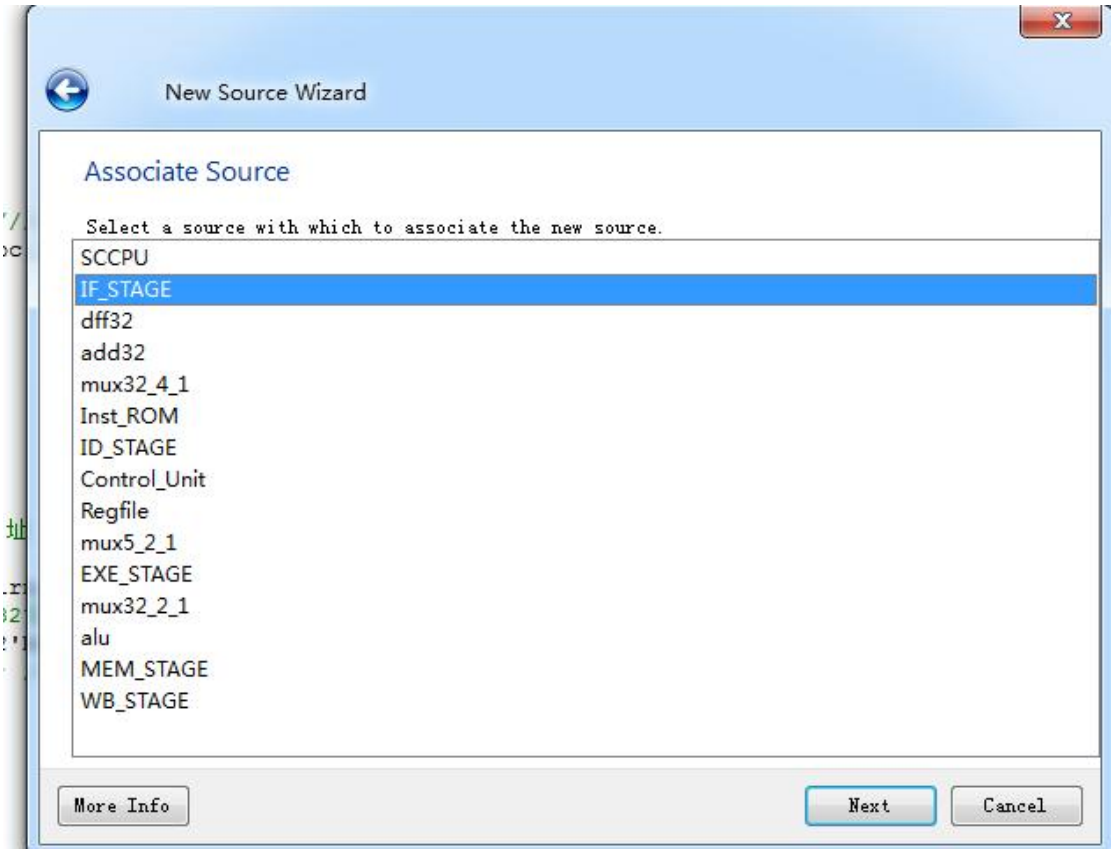


图 9

选择要测试的模块后，点击 Next 按钮，再点击 Finish 按钮，ISE 会在源代码编辑区自动生成测试模块的代码，如图 10 所示。

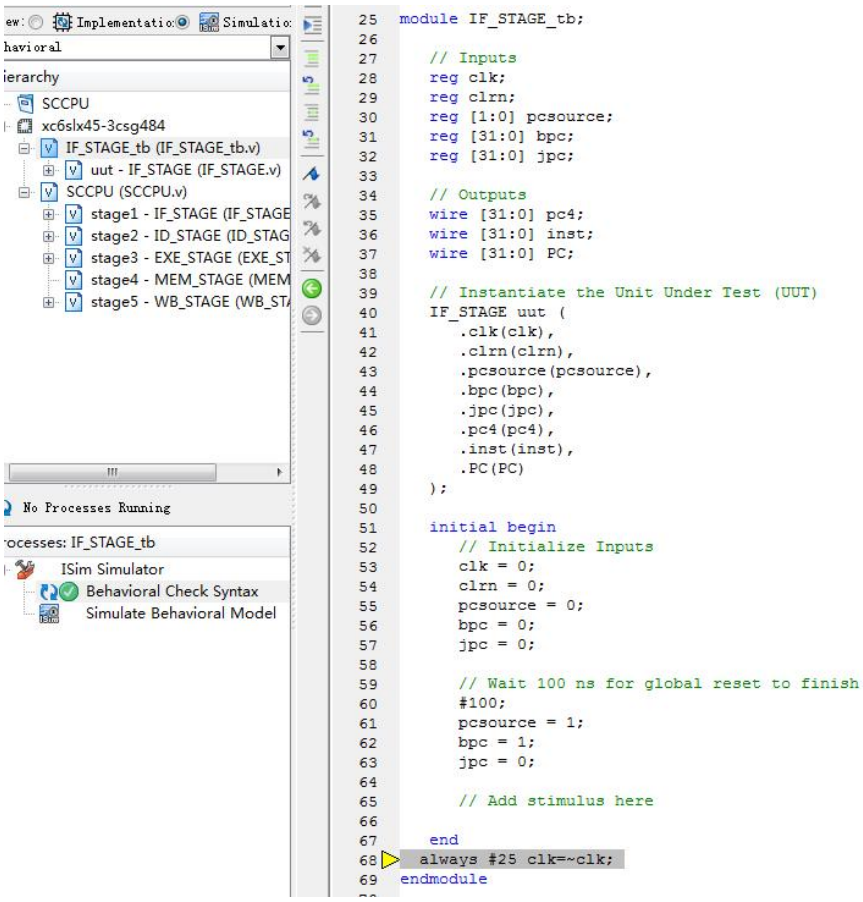


图 10

完成测试文件编辑后，确认工程管理区中 View 选项设置为 Simulation，并选中 IF_STAGE_tb 模块。 这时在过程管理区会显示与仿真有关的进程，如图 11 所示。

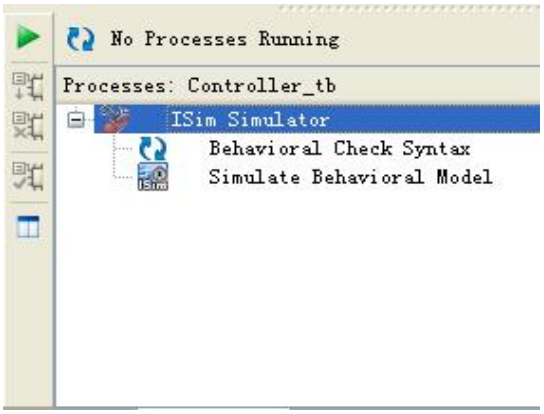


图 11

单击其中的 Simulate Behavioral Model 项，选择弹出菜单中的 Process Properties 项，会弹出如图 12 所示的属性设置对话框。

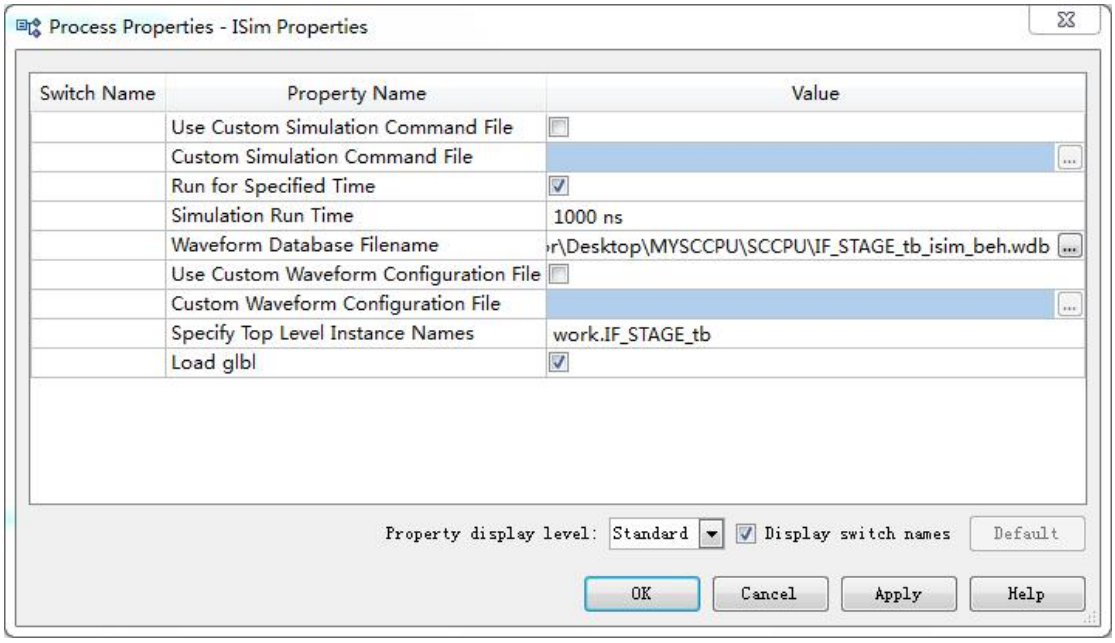


图 12

其中 Simulation Run Time 是仿真时间的设置，可将其修改为任意时长，这里设置为 1000ns。

仿真参数设置完后，就可以进行仿真。首先在工程管理区选中测试代码，然后在过程管理区双击 Simulate Behavioral Model，ISE 将启动 ISE Simulator，可以得到仿真结果，如图 13 所示。

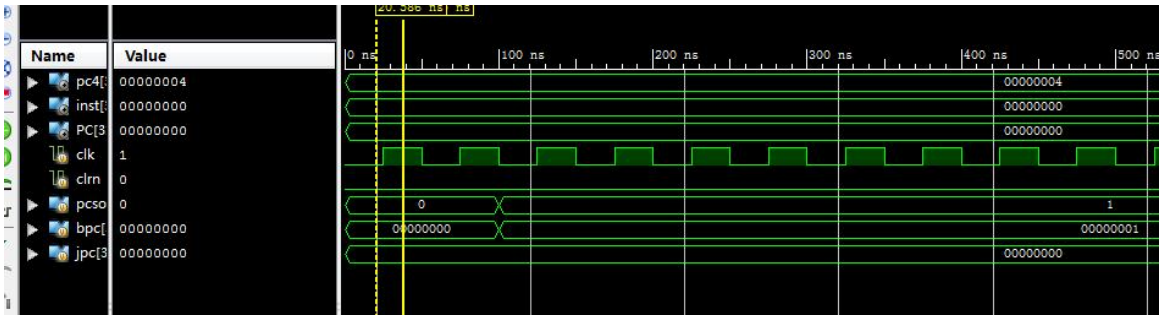
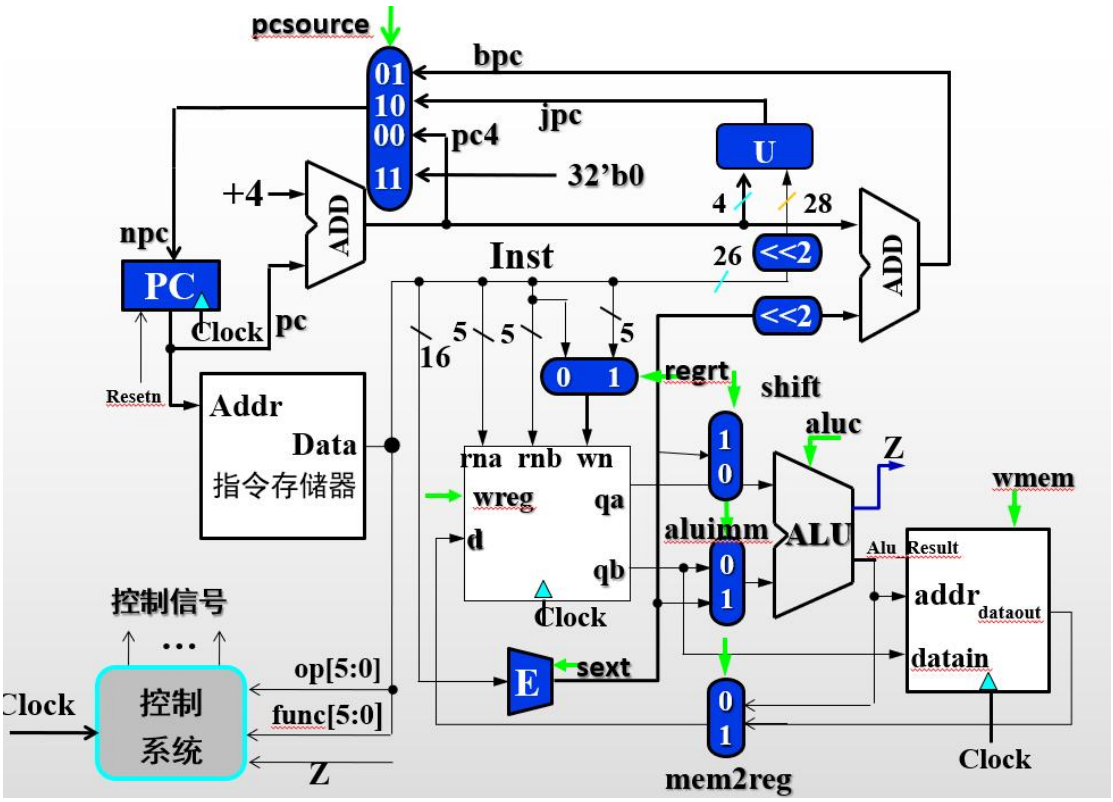


图 13

九、实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）

1. 补充完整单周期 CPU 微架构图（包括统一信号名），并进行必要的文字说明。补充完整的单周期 CPU 微架构图如下：



相应的文字说明：

1. 将信号与代码中的信号进行了统一，如使用了顶层模块中 Clock 替换原来的 clk，使用 Resetn 替换 clrn 信号等。
2. 在 Alu 的 a 端加入一个多路器，增加 shift 信号用于选择 a 端选择立即数还是 ra；
3. 将 rsrtequ 信号使用 Z 代替，储存器的输入使用 datain 信号，输出使用 dataout 信号；

2. 分析 IF_STAGE、Control_Unit 两个模块的代码

1）IF_STAGE

输入信号	信号说明
clk	时钟信号（1 bit）
clrn	复位信号，为 0 生效（1 bit）
pcsource	PC 值来源 4 选 1 选择器控制信号（2 bit）
bpc	Branch 指令选择的地址 （32 bit）
jpc	Jump 指令选择的地址（32 bit）
输出信号	信号说明
pc4	当前 pc+4 后的 pc 值（32 bit）
inst	在指令储存器去除的指令(32bit)
PC	当前的 PC 值（32 bit）

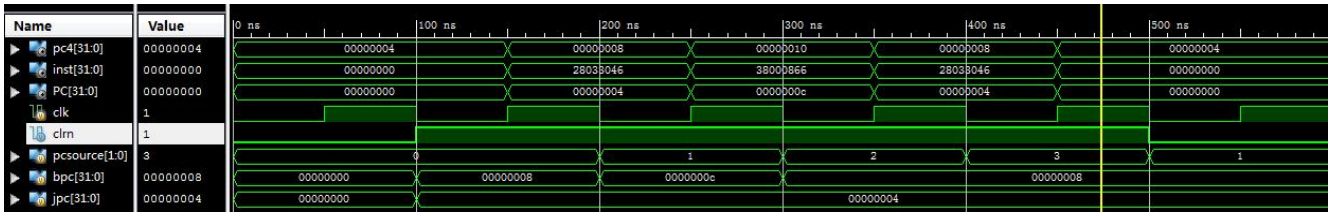
仿真测试代码如下：

initial begin

```
// Initialize Inputs
clk = 0;
clrn = 0;
pcsource = 0;
bpc = 0;
jpc = 0;

//选择 pc4
#100;
clrn=1;
pcsource = 0;
bpc = 32'h00000008;
jpc = 32'h00000004;

//选择 bpc
#100;
clrn=1;
pcsource = 1;
bpc = 32'h0000000C;
jpc = 32'h00000004;

//选择 jpc
#100;
clrn=1;
pcsource = 2;
bpc = 32'h00000008;
jpc = 32'h00000004;

//选择 0
#100;
clrn=1;
pcsource = 3;
bpc = 32'h00000008;
jpc = 32'h00000004;

//置位 0
#100;
clrn=0;
pcsource = 1;
bpc = 32'h00000008;
jpc = 32'h00000004;
```

```
    end
  always#50 clk=~clk;
```

仿真结果（截图）：



对结果进行简要的说明（**上升沿到来时改变 PC**）：

时钟周期	clrn	Pcsource	Npc	PC	Inst
1	0			X	
2	1	0	=pc4	32’h00000004	32’h28033046
3	1	1	=bpc	32’h0000000C	32’h38000866
4	1	2	=jpc	32’h00000004	32’h28033046
5	1	3	=0	32’h00000000	32’h00000000
6	0	X	X	32’h00000000	32’h00000000

结论：**结论符合预期**

2）Control_Unit

输入信号	信号说明
rsrtequ	ALU 运算结果 0 标志位，为 1 表示运算结果为 0（1 bit）
func	R 型指令的第二个操作码（6 bit）
op	操作码（6 bit）
输出信号	信号说明
wreg	控制写寄存器堆的信号（1bit）
m2reg	控制写回到寄存器堆的数据是来自 ALU 运算结果还是 DM 中取得的（1 bit）
wmem	写 DM 的控制信号（1 bit）
aluc	ALU 的运算操作码（3 bit）
regrt	选择目的寄存器（1 bit）
aluimm	控制 ALU B 端来自立即数还是 qb（1 bit）
sext	立即数扩展控制信号（1 bit）
pcsource	Npc 的值来源控制信号（2 bit）
shift	ALUA 端来源立即数还是 qa 的控制信号（1 bit）

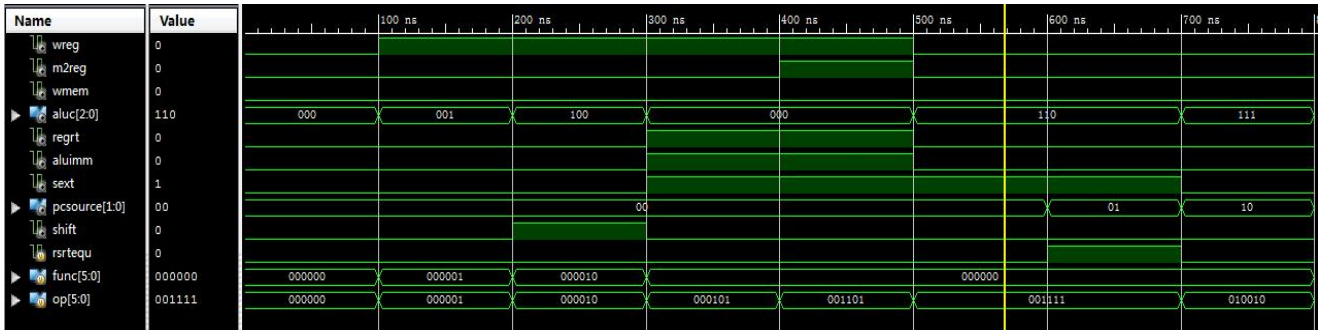
仿真测试代码如下：

```
initial begin
        // Initialize Inputs
        rsrtequ = 0;
```

```
        func = 0;
        op = 0;

        //寄存器与
        #100;
      rsrtequ = 0;
        func = 6'b000001;
        op = 6'b000001;

        //逻辑右移
        #100;
      rsrtequ = 0;
        func = 6'b000010;
        op = 6'b000010;

        //立即数加法
        #100;
      rsrtequ = 0;
        func = 6'b000000;
        op = 6'b000101;

        //load
        #100;
      rsrtequ = 0;
        func = 6'b000000;
        op = 6'b001101;

        //beq（不跳转）
        #100;
      rsrtequ = 0;
        func = 6'b000000;
        op = 6'b001111;

        //beq（跳转）
        #100;
      rsrtequ = 1;
        func = 6'b000000;
        op = 6'b001111;

        //jump
        #100;
      rsrtequ = 0;
        func = 6'b000000;
        op = 6'b010010;
```

```
end
```

仿真结果（截图）：



对结果进行简要的说明：

时钟周期	指令类型	rsrtequ	wreg	m2reg	wmem	aluc	regrt	aluimm	sext	pcsource	shift
1	X										
2	and	0	1	0	0	001	0	0	0	00	0
3	srl	0	1	0	0	100	0	0	0	00	1
4	addi	0	1	0	0	000	1	1	1	00	0
5	load	0	1	1	0	000	1	1	1	00	0
6	beq	0	0	0	0	110	0	0	1	00	0
7	beq	1	0	0	0	110	0	0	1	01	0
8	jump	0	0	0	0	111	0	0	0	10	0

结论：**结果符合预期**

3. 分析 SCCPU（单周期 CPU）代码

1）测试的指令序列如下所示，对应的初始化 inst_mem 部分 Verilog 代码如下：

```
assign rom[6'h00]=32'h00000000;      //指令地址从 1 开始；
assign rom[6'h01]=32'h28033046;      //ori r6,r2,0x00cc   r6=0x000000ce   i 型指令
assign rom[6'h02]=32'h00101464;      //add r5,r3,r4   r5=0x00000007   r 型指令
assign rom[6'h03]=32'h38000866;      //store r6,0x0002(r3)   m5=0x000000ce
assign rom[6'h04]=32'h34000489;      //load r9,0x0001(r4)   r9=0x000000ce
assign rom[6'h05]=32'h3c000c21;      //beq r1,r1,6'h0a   相等转移到 PC+0aH 处
assign rom[6'h06]=32'h00100421;          //add r1,r1,r1
assign rom[6'h07]=32'h0821a408;          //sr1 r9,r8,5'h03
assign rom[6'h08]=32'h00100421;          //add r1,r1,r1
assign rom[6'h09]=32'h48000001;         //jump 0x0000001   无条件转移到 01H 处
```

2）顶层模块的信号如下表所示：

输入信号	信号说明
Clock	时钟信号（1 bit）
Resetn	复位信号（1 bit）
输出信号	信号说明
PC	当前 PC 值 （32 bit）
Inst	当前执行的指令 （32 bit）
Alu_Result	ALU 运算结果 （32 bit）

3）regfile 初始化部分数据：

```
register[5'h01]<=32'h00000001;
register[5'h02]<=32'h00000002;
register[5'h03]<=32'h00000003;
register[5'h04]<=32'h00000004;
register[5'h05]<=32'h00000005;
register[5'h06]<=32'h00000006;
register[5'h07]<=32'h00000007;
register[5'h08]<=32'h00000008;
register[5'h09]<=32'h00000009;
```

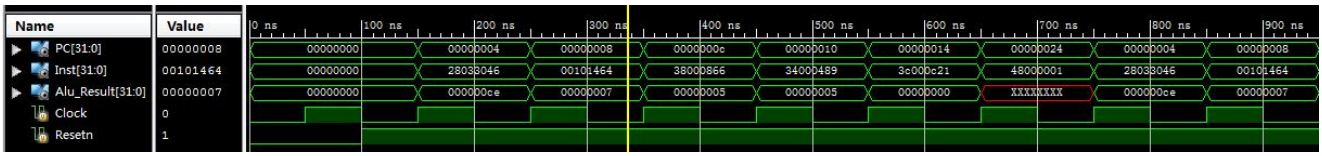
4）数据存储器初始化数据：

```
for(i=0;i<32;i=i+1)              //存储器清零
        ram[i]=0;
    ram[5'h01]=32'h0000000a;        //存储器对应地址初始化赋值
    ram[5'h02]=32'h0000000b;
    ram[5'h03]=32'h0000000c;
    ram[5'h04]=32'h0000000d;
```

5）SCCPU 模块仿真测试代码如下：

```
initial begin
        // Initialize Inputs
        Clock = 0;
        Resetn = 0;

        // Wait 100 ns for global reset to finish
        #100;
        Resetn = 1;

        // Add stimulus here

    end
   always #50 Clock=~Clock;
```

6）仿真结果：



【对结果进行说明】（上升沿触发）

时钟周期	Resetn	PC	Inst	Alu_Result	指令类型
1	0	X			
2	1	32’h00000004	32’h28033046	32’h000000ce	ori
3	1	32’h00000008	32’h00101464	32’h00000007	add
4	1	32’h0000000c	32’h38000866	32’h00000005	store
5	1	32’h00000010	32’h34000489	32’h00000005	load
6	1	32’h00000014	32’h3c000c21	32’h00000000	beq
7	1	32’h00000024	32’h48000001	X	jump
8	1	32’h00000004	32’h28033046	32’h000000ce	ori
9	1	32’h00000008	32’h00101464	32’h00000007	add

【结论】结果符合预期

十、实验结论：（联系理论知识进行说明）

仿真时可以通过设置仿真的总时间来得到理性的效果，如果需要仿真的时钟周期大于设置的仿真总时间的时钟周期，那么会导致后面的时钟周期仿真结果不可见，因此可以根据需要的时钟周期来设置总时间。同时在设置指令序列时，应该尽可能的覆盖到每种指令，这样才能保证 CPU 的逻辑是正确的，同时仿真也是对代码逻辑正确性的检验。

十一、总结及心得体会：

在本次实验中，对取指阶段，控制单元，顶层模块分别进行了测试仿真，在仿真过程中也发现了一些代码的编写问题，在写测试文件中就能很好的理解代码的逻辑，同时在对每个模块进行分析后，也明白了各信号的具体含义以及 CPU 的工作原理，更进一步的理解了单周期 CPU 的结构。

十二、对本实验过程及方法、手段的改进建议：

可以对 CPU 的其他部分也进行仿真分析，可以使用更多的指令进行仿真，观察仿真的结果以更深层次的理解单周期 CPU 的原理以及更加熟练使用 Verilog 语言。

报告评分：

指导教师签字：