实验报告

实验目的：

1. 熟悉对时序逻辑电路的设计与调试，体会时序逻辑电路设计与软件设计的区别。
2. 2. 熟悉 CPU 的组成部分。

PC

实现逻辑

我通过时序逻辑电路设计PC寄存器。当处于clk上升沿时，判断此时reset和jumpEnabled状态，当reset不为0时，将pcValue设置为初始值32'h00003000，当jumpEnabled不为0时，将当前PC值设为jumpInput输入值，否则PC值自增4。

代码

```
module ProgramCounter(
    input reset,
    input clock,
    input jumpEnabled,
    input [31:0] jumpInput,
    output reg [31:0] pcValue
);

    always @(posedge clock or posedge reset) begin
        if (reset) begin
            pcValue <= 32'h00003000;
        end else if (jumpEnabled) begin
            pcValue <= jumpInput;
        end else begin
            pcValue <= pcValue + 4;
        end
    end

endmodule
```

测试文件

测试逻辑

我用相同逻辑设计测试文件，并用correct变量来判断结果是否正确，并对测试结果进行仿真。

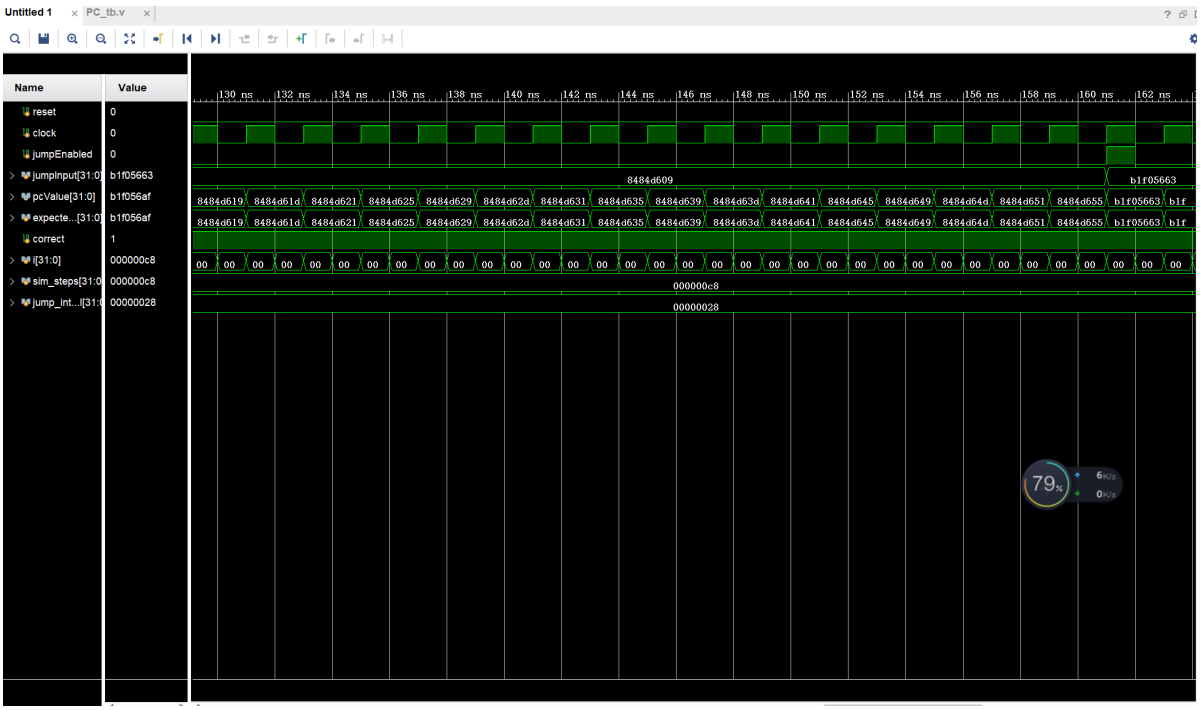
测试代码

```
module PC_tb;

    reg reset, clock, jumpEnabled;
    reg [31:0] jumpInput;
    wire [31:0] pcValue;
    reg [31:0] expected_pc;
    reg correct;
```

```
    // Instantiate the ProgramCounter module
    ProgramCounter PC_inst (
        .reset(reset),
        .clock(clock),
        .jumpEnabled(jumpEnabled),
        .jumpInput(jumpInput),
        .pcValue(pcValue)
    );

    // Test duration and simulation steps
    integer i;
    integer sim_steps = 200;  // Number of simulation steps
    integer jump_interval = 40;  // Jump every 40 cycles

    // Clock generation
    initial begin
        clock = 0;
        reset = 1;
        jumpEnabled = 0;
        jumpInput = 0;
        expected_pc = 32'h00003000;
        correct <= 1;

        // Apply reset for one clock cycle
        #1 reset = 0;

        // Simulation steps
        for (i = 0; i < sim_steps; i = i + 1) begin
            // Toggle clock every cycle
            clock = ~clock;

            // Check if it's time for a jump or regular increment
            if (jumpEnabled) begin
                expected_pc = jumpInput;
                jumpEnabled = 0;
            end else if (i % jump_interval == 0 && i > 0) begin
                // Set jump input to a random value
                jumpInput = $random;
                jumpEnabled = 1;
                // Expected PC value after jump
                expected_pc = jumpInput;
            end else if (clock) begin
                // Expected PC value after regular increment on rising edge of
clock
                expected_pc = expected_pc + 4;
            end

            // On negative edge of clock, check correctness
            if (~clock) begin
                correct <= (expected_pc == pcValue);
            end

            // Wait for a half clock cycle
            #0.5;
        end
```

```
        // Display test result
        if (correct)
            $display("Test passed!");
        else
            $display("Test failed!");

        // Finish simulation
        $finish;
    end

endmodule
```

仿真结果



DM

实现逻辑

首先定义一个数组作为数据块，之后设计一个组合逻辑电路用于数据的读操作，并设计一个时序逻辑电路用于数据的写操作。当处于时钟下降沿时，判断reset信号，若reset信号为1，则将该数据块清零，否则判断writeEnabled信号，若为1，写入writeInput的值到address地址上。

代码

```
module DataMemory(
    input reset,
    input clock,
    input [31:0] address,
    input writeEnabled,
    input [31:0] writeInput,
    output [31:0] readResult
);

    reg [31:0] data [0:1023];  // 32位×1024的存储器
```

```
    integer i;

    always @(posedge clock or posedge reset) begin
        if (reset) begin
            // 在复位时将存储器清零
            for (i = 0; i < 1024; i = i + 1) begin
                data[i] <= 32'h00000000;
            end
        end else if (writeEnabled) begin
            // 在写入使能时，写入数据到指定地址
            data[address[31:2]] <= writeInput;
        end
    end

    // 组合逻辑，读取指定地址的数据
    assign readResult = data[address[31:2]];

endmodule
```

测试文件

测试逻辑

我在测试文件中通过循环，在每个仿真步骤中执行以下操作：

- 判断是否需要进行读写操作，若需要则根据随机数决定是读操作还是写操作。
- 对读操作：
  - 设置写使能信号为 0，随机生成地址进行读取。
  - 检查读取的实际结果与期望值是否一致。
- 对写操作：
  - 设置写使能信号为 1，随机生成数据和地址进行写入。
  - 更新 DataMemory 中的数据以便后续的正确性检查。
- 更新期望值以反映当前的 DataMemory 内容（读操作和写操作都需要更新期望值）。
- 等待半个时钟周期，以模拟时序操作。

代码

```
module DM_tb;

    reg reset, clock, writeEnabled;
    reg [31:0] address, writeInput, expectValue;
    wire [31:0] readResult;
    reg correct;

    // 实例化 DataMemory 模块
    DataMemory DM_inst (
        .reset(reset),
        .clock(clock),
        .address(address),
        .writeEnabled(writeEnabled),
        .writeInput(writeInput),
```

```
        .readResult(readResult)
    );

    // 测试持续时间和仿真步数
    integer i;
    integer sim_steps = 200;  // 仿真步数
    integer read_write_interval = 10;  // 每 10 个周期读或写一次

    // 时钟生成
    initial begin
        clock = 0;
        reset = 1;
        writeEnabled = 0;
        address = 0;
        writeInput = 0;
        correct = 1;
        expectValue = 0;

        // 应用复位信号一个时钟周期
        #1 reset = 0;

        // 仿真步骤
        for (i = 0; i < sim_steps; i = i + 1) begin
            // 每个周期切换时钟
            clock = ~clock;

            // 根据间隔执行读或写操作
            if (i % read_write_interval == 0 && i > 0) begin
                // 随机选择读或写
                if ($random % 2 == 0) begin
                    // 读操作
                    writeEnabled = 0;
                    address = $random & 1023;  // 在范围内随机选择地址

                    // 检查实际读取的结果是否与期望值匹配
                    if (readResult !== expectValue) begin
                        $display("在仿真步骤 %0d 处发生不匹配。期望数据: %h, 实际数据:
%h", i, expectValue, readResult);
                        correct = 0;
                    end
                end else begin
                    // 写操作
                    writeEnabled = 1;
                    writeInput = $random;
                    address = $random & 1023;  // 在范围内随机选择地址

                    // 更新 DataMemory 中的期望数据以进行正确性检查
                    DM_inst.data[address] = writeInput;
                end
            end

            // 为读操作更新期望值
            expectValue = DM_inst.data[address];

            // 等待半个时钟周期
            #0.5;
```

```
        end

        // 显示测试结果
        if (correct)
            $display("测试通过!");
        else
            $display("测试失败!");

        // 结束仿真
        $finish;
    end

endmodule
```

仿真结果