《数字电路》实验报告

实验名称： 熟悉 vivado 环境 指导教师：王珏，范志华

姓名：韩初晓 学号：2023K8009908002 专业：计算机科学与技术 班级：2306

实验日期：2024.10.17 实验地点：教学楼 221 是否调课/补课：否 成绩：

目录

第 1 部分 实验目的

1. 熟悉 Vivado 设计流程（设计 → 仿真 → 综合 → 优化 → 门级仿真 → 电路制造工艺文件）
2. 掌握 Verilog 的基本程序结构，掌握利用 Vivado 创建设计的方法（以实现 4 位加法器为例）
3. 掌握编写 Testbench 的方法，以及行为仿真方法

1.1 实验环境

- Vivado 2017.4 开发工具
- FPGA 开发平台（根据手册中的默认设置进行选择）

第 2 部分 实验内容

2.1 实验一：实现四位加法器

2.1.1 原理说明

四位加法器接受两个四位的二进制输入和一个进位输入，输出四位的和和一个进位输出。我的代码中，加法器可以表示为：

$$\{cout, out\} = in\_0 + in\_1 + cin$$

其中，`cout` 是加法运算中的进位输出，`out` 是四位加法的结果。

2.1.2 接口定义

- 输入信号：
  - `in_0[3:0]`：四位二进制数的第一个加数。
  - `in_1[3:0]`：四位二进制数的第二个加数。
  - `cin`：输入进位信号，用于处理来自前一级的进位。
- 输出信号：
  - `out[3:0]`：四位加法运算的结果。
  - `cout`：加法运算的进位输出信号，表示最高位的进位。

2.1.3 调试过程及结果

‘add_4’ 模块，用于实现四位加法器，通过编写 Testbench 模块 ‘test_add_4’ 对 ‘add_4’ 进行仿真测试，仿真波形展示了输入信号的变化以及对应的输出结果如下：

2.2 实验二：实现一位加法器

2.2.1 原理说明

一位加法器接受两个单比特二进制输入和一个进位输入，产生一个二进制和和一个进位输出。它可以用如下的逻辑表达式写出：

$$\text{sum} = in\_a \oplus in\_b \oplus cin$$

$$\text{cout} = (in\_a \wedge in\_b) \vee ((in\_a \oplus in\_b) \wedge cin)$$

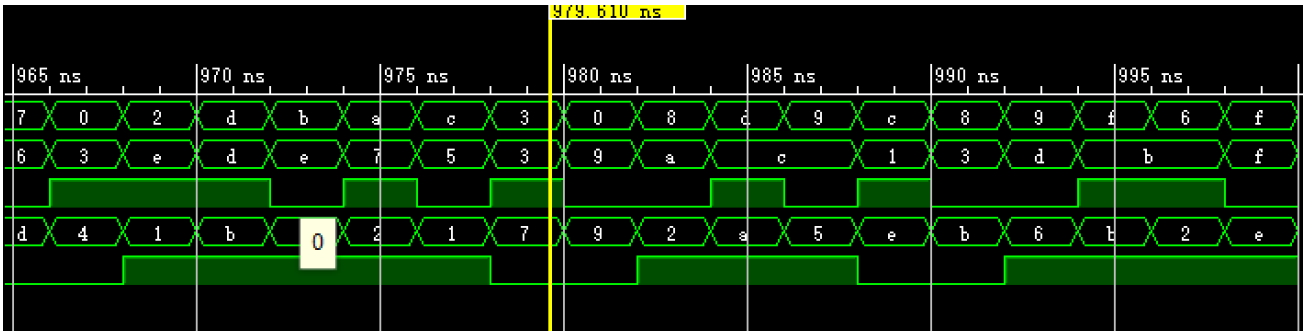

图 1: add_4 模块的测试结果

其中：*in_a* 和 *in_b* 是两个单比特输入。*cin* 是进位输入。*sum* 是和输出。*cout* 是进位输出。

2.2.2 接口定义

- 输入信号：
  - `in_a`：第一个加数。
  - `in_b`：第二个加数。
  - `cin`：进位输入。
- 输出信号：
  - `sum`：加法运算的和。
  - `cout`：进位输出。

2.2.3 调试过程及结果

‘add_1’ 模块实现了一位加法器，Testbench 模块 ‘test_add_1’ 对 ‘add_1’ 进行仿真测试，结果波形图如下：

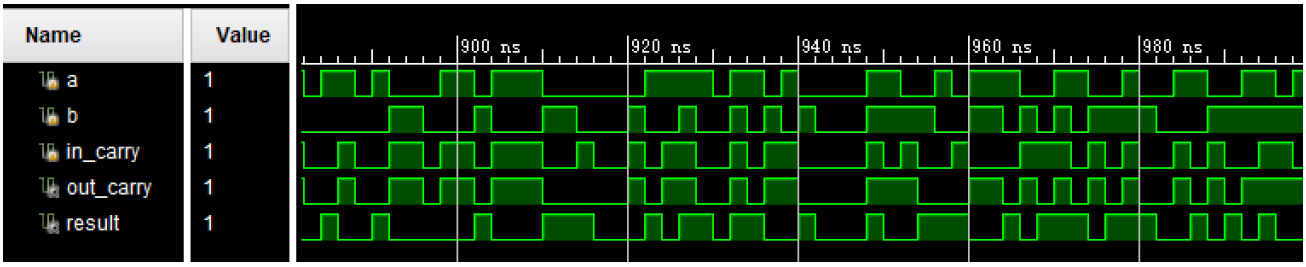

图 2: add_1 模块的测试结果

2.3 实验三：实现三八译码器

2.3.1 原理说明

三八译码器将输入的三位二进制数（0-7）转换为八个输出线其中一个的高电平，其他输入线保持低电平。每个输入值只激活一个输出信号。同时，在本实验中实现的三八译码器具有只有控制信号在高电平的情况下译码器才工作的动能。

三八译码器的真值表如下：

表 1: 三八译码器真值表

输入	输出
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

2.3.2 接口定义

- 输入信号：
  - `in[2:0]`：三位二进制输入信号。
- 输出信号：
  - `out[7:0]`：八位输出信号，输出时只有一个为 1，其余都为 0。
- 控制信号：
  - `signal`：一位控制信号，控制信号为高电平时，译码器工作。

2.3.3 调试过程及结果

‘decoder’ 模块实现了三八译码器，Testbench 模块 ‘test_decoder’ 对 ‘decoder’ 进行仿真，仿真结果如下：

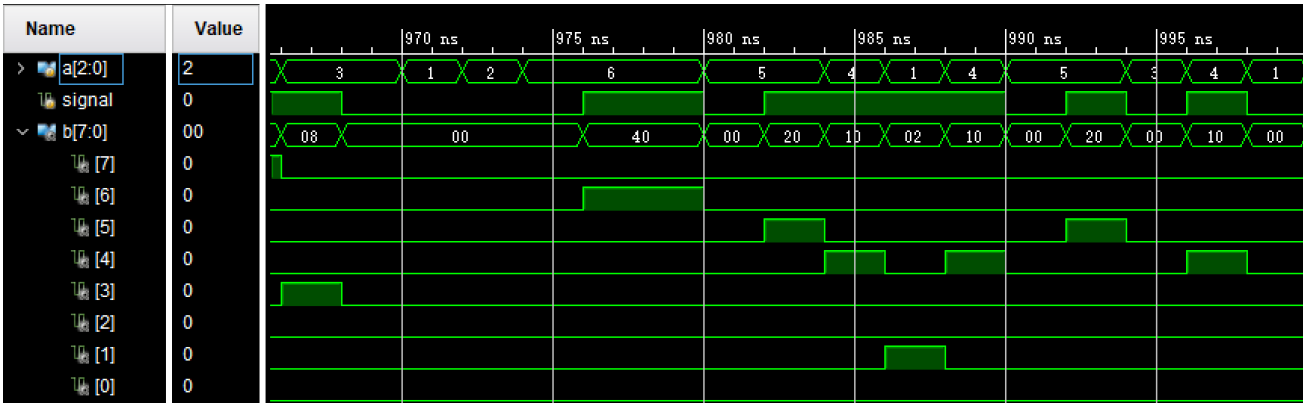

图 3: decoder 模块的测试结果

第 3 部分 实验总结

在本实验中我通过 Verilog 实现了数字电路中三个重要的模块——四位加法器、一位加法器和三八译码器。在亲手进行代码编写的时候，我更深入的理解了“行为级”和“结构级”的等价性，同时，我也更清楚的感受到了一些容易犯错的代码规范问题，提高了我使用 Verilog 语言进行硬件描述的能力。同时，我也学会了编写测试模块，这对于仿真验证的过程是十分重要的。

第4部分 源代码

4.1 实验一：实现四位加法器

```
// this is the add_4 module
module add_4(
    input [3:0] in_0,
    input [3:0] in_1,
    input cin,
    output cout,
    output [3:0] out
    );
    assign {cout, out} = in_0 + in_1 + cin;
endmodule

// this is the test_add_4 module
module test_add_4()(
    .in_0(a),
    .in_1(b),
    .cin(in_carry),
    .out(sum),
    .cout(out_carry)
  );

  initial begin
    a = 4'h1;
    b = 4'h0;
    in_carry = 1'b0;
  end

  always begin
    #2;
    a = $random() % 16;
    b = $random() % 16;
    in_carry = $random() % 2;
  end
endmodule
```

4.2 实验二：实现一位加法器

```
// this is the add_1 module
module add_1(
    input in_a,
```

```
    input in_b,
    input cin,
    output cout,
    output sum
    );
    assign sum = in_a ^ in_b ^ cin;
    assign cout = (in_a & in_b) | ((in_a ^ in_b) & cin);
endmodule

// this is the test_add_1 module
module test_add_1();
    reg a;
    reg b;
    reg in_carry;
    wire out_carry;
    wire result;

    add_1 instance_add1(
      .in_a(a),
      .in_b(b),
      .cin(in_carry),
      .cout(out_carry),
      .sum(result)
    );
    initial begin
      a = 1'b1;
      b = 1'b0;
      in_carry = 1'b0;
    end
    always begin
      #2;
      a = $random() % 2;
      b = $random() % 2;
      in_carry = $random() % 2;
    end
endmodule
```

4.3 实验三：实现三八译码器

```
//this is the decoder module
module decoder(
    input [2:0] in,
    input signal,
    output reg [7:0] out
    );
```

```
    always @(*) begin
      out = 8'b00000000;
      if (signal) begin
      case (in)
        3'b001: out[1] = 1;
        3'b010: out[2] = 1;
        3'b011: out[3] = 1;
        3'b100: out[4] = 1;
        3'b101: out[5] = 1;
        3'b110: out[6] = 1;
        3'b111: out[7] = 1;
        3'b000: out[0] = 1;
      endcase
    end
    end
endmodule

//this is the test_decoder module
module test_decoder();
    reg [2:0] a;
    reg signal;
    wire [7:0] b;
    decoder instance_decoder(
      .in(a), .out(b), .signal(signal)
    );
initial begin
  a = 3'b000;
end
always begin
  #2;
  signal = $random() % 2;
  a = $random() % 8;
end
endmodule
```