# 实验报告 1

## 1 实验目的

### 1.1 熟悉 vivado 环境

数是 vivado 设计流程, 掌握利用 vivado 创建设计的方法, 掌握编写 testbench 的方法 (利用 testbench 来测试设计), 掌握行为仿真法

### 1.2 掌握基本的 vivado 设计编写

以四位全加加器以及 3-8 译码器为例, 熟悉并掌握基本的设计编写思路

## 2 实验环境

AMD Vivado2022.2

## 3 原理说明

### 3.1 四位全加器 (利用门电路实现一位加法器)

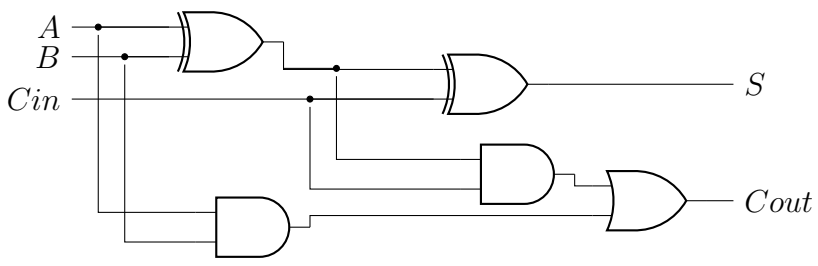
考虑全加器, 输入为 A,B,Cin, 输出为本位输出 S 和进位输出 Cout

$$S = A \oplus B \oplus Cin$$

经过测试, 在 verilog 语言中无法进行三个异或, 因此需要声明一个 wire w1 在存前两个异或, 以实现三个异或

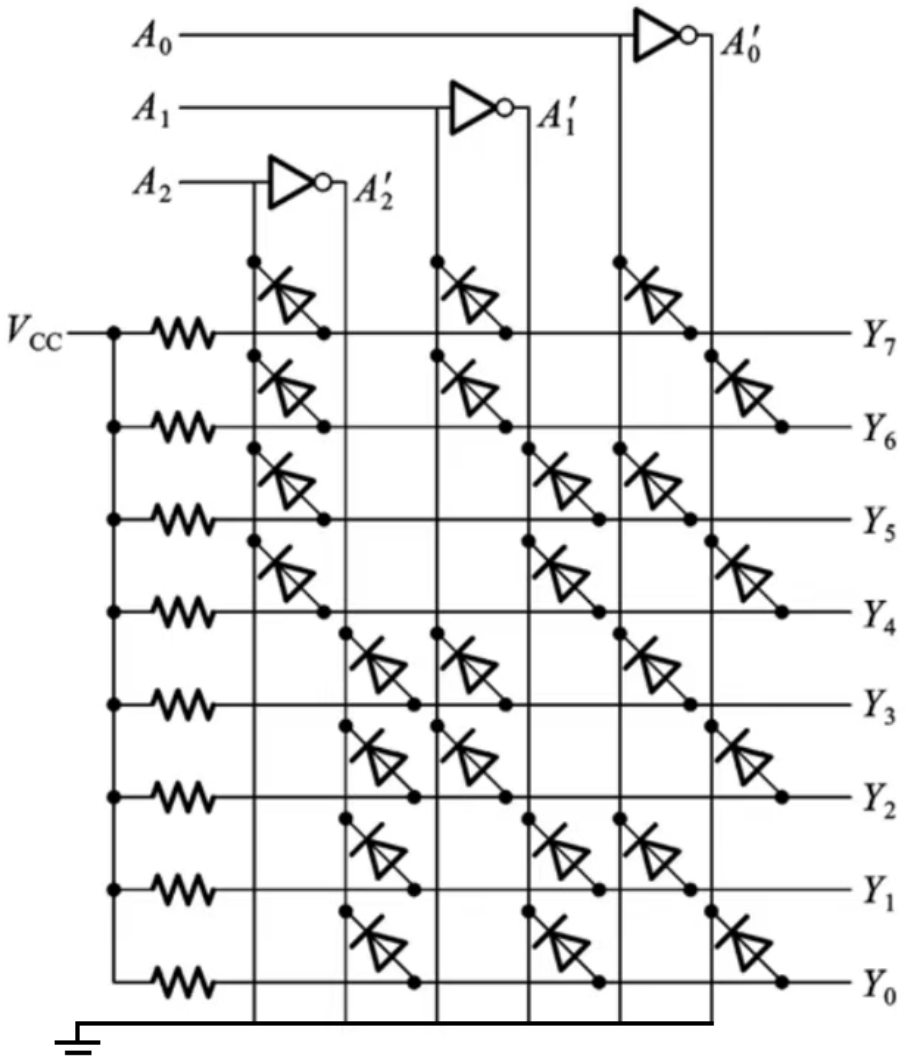
$$Cout = (A \oplus B) \cdot Cin + A \cdot B \cdot Cin$$

门电路图如下



## 3.2 3-8 译码器

输入的是一个三位的二进制数, 而这个电路是高电平有效的, 因此在 state=1 时才执行译码, 否则全部都是未驱动. 直接采取一一对应的电路, 电路图如下 (采取 0 输出有效)




在这里提出一种想法, 每次读取输入的最低位, 然后根据 0 和 1 的不同来选取可能的结果, 之后再将输入右移 1, 重复此过程, 例如输入是 101, 最低位是 1, 则有可能是 1,3,5,7, 右移变为 010, 最低为是 0, 则有可能是 1,5, 以此类推 (这样可能比较符合程序的逻辑, 但在电路中的实现貌似更麻烦)

# 4 接口定义

## 4.1 4 位全加器

```
输入：
    [3:0]in_0   (四位输入1)
    [3:0]in_1   (四位输入2)
    c_in    (进位输入)
输出：
    [3:0]out    (四位输出)
    c_out   (进位输出)
```

## 4.2 3-8 译码器

```
输入：
    [2:0]in   (三位输入)
    state   (有效电平)
输出：
    [7:0]out    (输出)
```

# 5 调试过程及结果

## 5.1 4 位加法器

在四位加法器中并未遇到什么困难, 在这里直接给出结果波形图

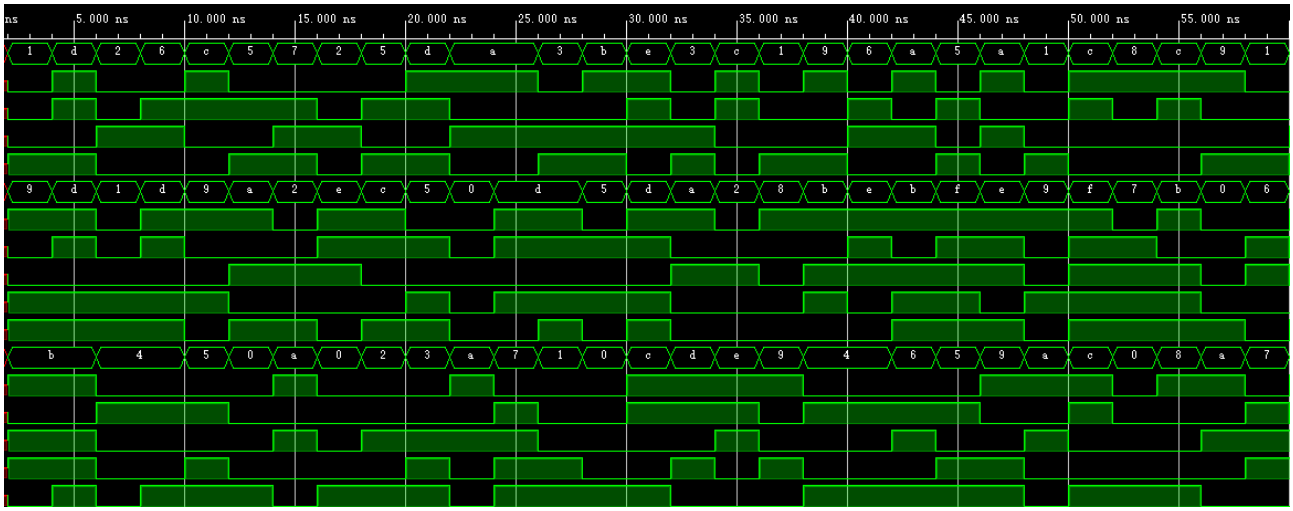



图 1: 从上到下分别是输入 1, 输入 2, 进位输入, 输出, 进位输出

## 5.2 1 位全加器

在一位全加器中, 我注意到在 verilog 语言中不可以利用门进行两个以上的逻辑运算操作, 需要引入线来作为中间变量, 例如存下前两个异或的结果再将这个结果与第三个变量做异或得到三个变量异或的结果

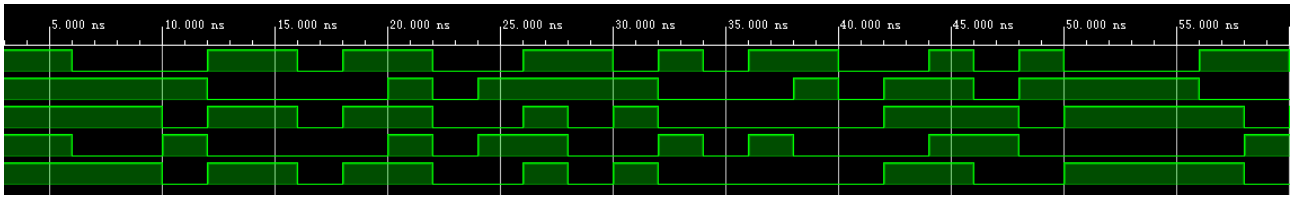



图 2: 从上到下分别是输入 1, 输入 2, 进位输入, 本位输出, 进位输出

## 5.3 3-8 译码器

在 3=8 译码器中, 注意到进行 reg 时, 不可以在声明端口外进行 reg 声明, 这样会造成不期望的 Z 驱动

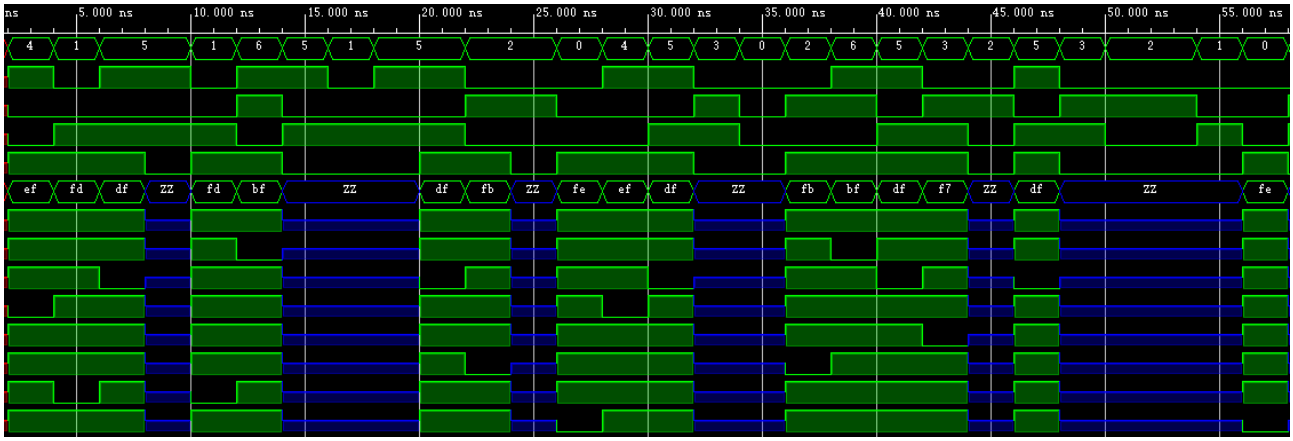



图 3: 从上到下分别是 3 位输入, 有效电平 state, 输出

# 6 实验总结

## 6.1 Vivado 使用方面

熟悉了如何创建、编辑、设计工程, 并学会使用了激励文件来测试工程

## 6.2 Verilog 语言方面

进一步了解了 reg 的用法、随机数的生成以及 always, if, cases 的使用

# 7 源代码

## 7.1 四位加法器

```
module add_4(
    input [3:0] in_0,
    input [3:0] in_1,
    input c_in,
    output [3:0] out,
    output cout);

    assign{cout, out}=in_0+in_1+c_in;

endmodule
```

## 7.2 1 位全加器

```
module add_1(
    input in_0,
    input in_1,
    input cin,
    output cout,
    output s
    );
    wire x1;
    xor(w1,in_0,in_1);
    xor(s,w1,cin);
    wire y1,y2,y3;
    and(y1,w1,cin);
    and(y2,in_0,in_1);
    and(y3,y2,cin);
    or(cout,y3,y1);

endmodule
```

## 7.3 3-8 译码器

```
module decoder1(
    input [2:0]in,
    input state,
    output reg[7:0]out
    );

    always @(state or in)begin
    if(state)begin
        case(in)
            3'b000: out=8'b11111110;
            3'b001: out=8'b11111101;
            3'b010: out=8'b11111011;
            3'b011: out=8'b11110111;
            3'b100: out=8'b11101111;
            3'b101: out=8'b11011111;
            3'b110: out=8'b10111111;
            3'b111: out=8'b01111111;
            default out=8'b11111111;
        endcase
    end
    else
        out=8'bZZZZZZZZ;
end
endmodule
```