浙江工业大学

《可编程逻辑器件及应用》
课程实验报告

学生姓名 凌智城

指导教师 龚树凤

专业班级 通信工程 1803 班

培养类别 全日制本科

所在学院 信息工程学院

提交日期 2021 年 1 月 4 日

实验二：加法器的设计与仿真

2.1 1 位半加器的设计与仿真

2.1.1 设计任务

完成第十二章 12.3 节半加器设计内容，完成设计输入、编译、时间约束设置，功能仿真，管脚分配。

2.1.2 设计思路与原理

一位半加器有两输入两输出，输入为两个数据位相加，没有进位输入，输出为一个数据位一个进位，实现一位二进制加法。其中 a，b 为输入，cout 为进位输出，sum 为和输出。

一位半加器真值表

a	b	sum	cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.1.3 Verilog 源代码与注释

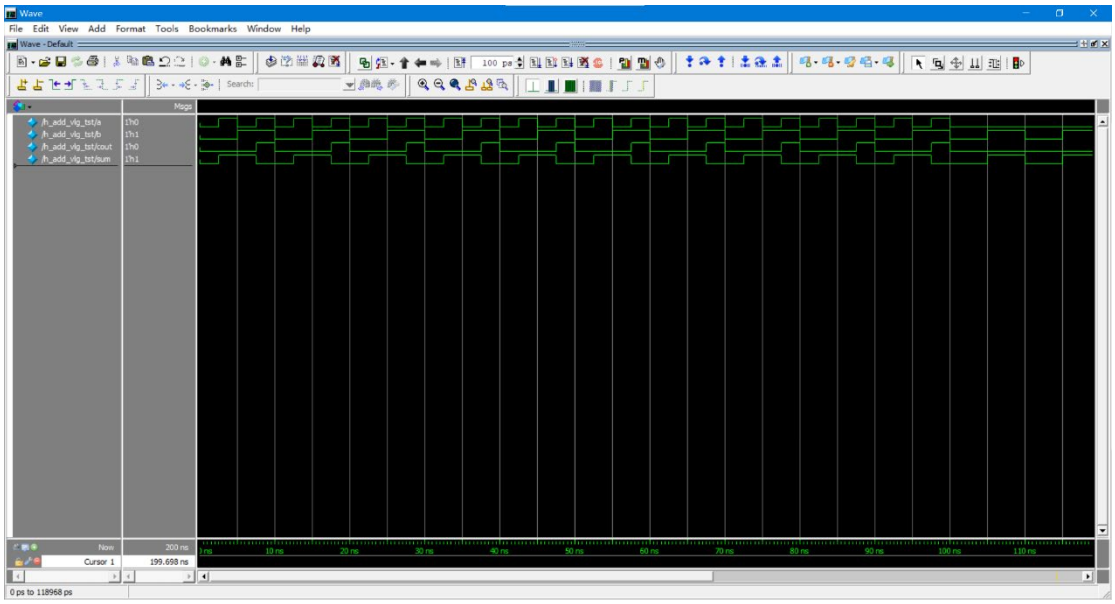
```
module h_add(cout,sum,a,b);
    output cout,sum;
    input a,b;
    wire cout,sum;
    assign{cout,sum}=a+b;          //用 assign 进行线网赋值
endmodule                          //{}为连接符号


`timescale 100ps/10 ps
module h_add_vlg_tst();
reg a,b;
   wire cout,sum;
   h_add i1(cout,sum,a,b);
      parameter period1=50,period2=100;      //period1 为 a 的周期
      parameter pulse=40;                    //period2 为 b 的周期  pluse 重复四十次

initial
    begin
      a=1'b0;
```

```
        repeat(pulse)
            #(period1/2) a=~a;
    end
initial
    begin
      b=1'b0;
     repeat(pulse)
       #(period2/2) b=~b;
    end
endmodule
```

2.1.4 仿真结果与分析



a，b 为输入位，testbench 设置 a 翻转周期 50，b 反转周期为 100，在无时序延迟影响下，可从图中直观看出符合半加器设计要求。

图 2-1　1 位半加器的功能仿真截图

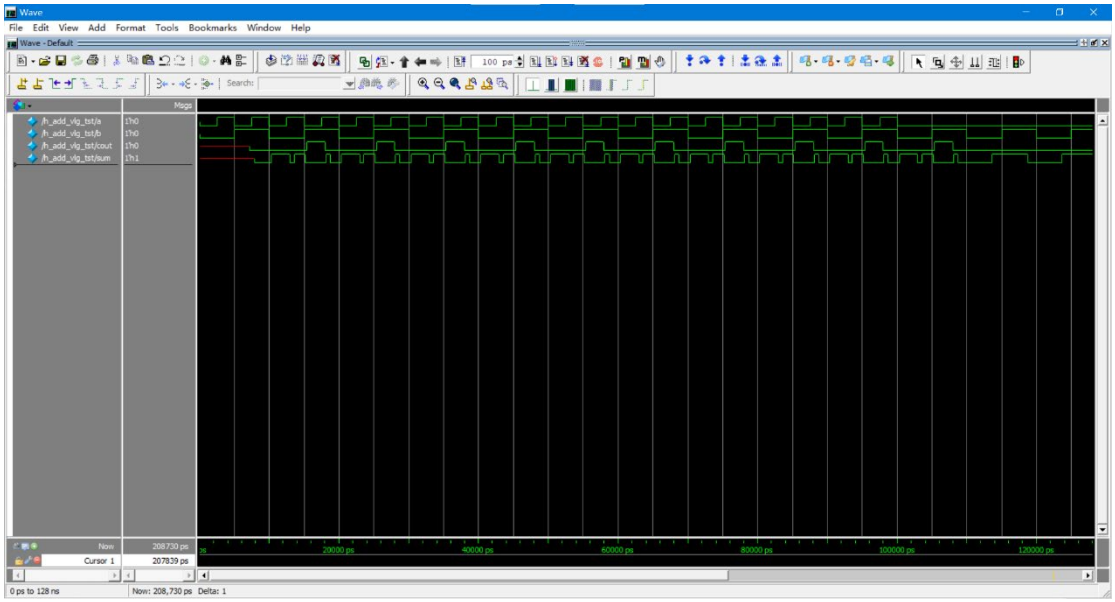


图 2-2　1 位半加器的时序仿真截图

时序仿真使用布局布线后器件给出的模块和连线的延时信息，在最坏的情况下对电路的行为作出实际地估价，时序仿真信号加载了时延，波形产生毛刺现象。

2.2 8 位全加器的设计与仿真

2.2.1 设计任务

利用 1 位半加器设计 8 位全加器，使用 Quartus II 软件进行仿真验证，测试功能。

2.2.2 设计思路与原理

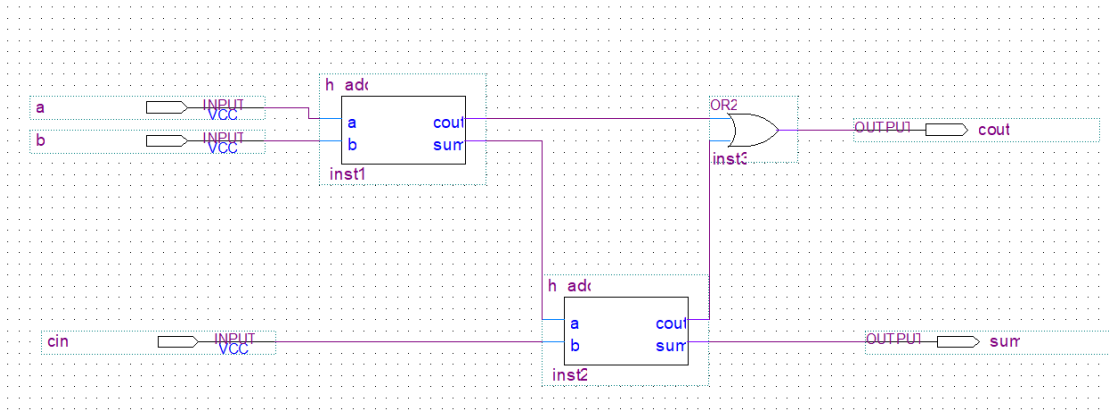
一位半加器与一位全加器的区别在于全加器有从低位进上来的进位位，可以使用两个半加器与一个二输入或门构成一位全加器，有三个输入信号分别问 cin，a，b 和两个输出信号 cout 和 sum。

一位半加器真值表

cin	a	b	cout	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

由八个一位全加器级联即可获得一个八位全加器

2.2.3 Verilog 源代码与注释




```
module add_1(
    a,
    b,
    cin,
    cout,
    sum
```

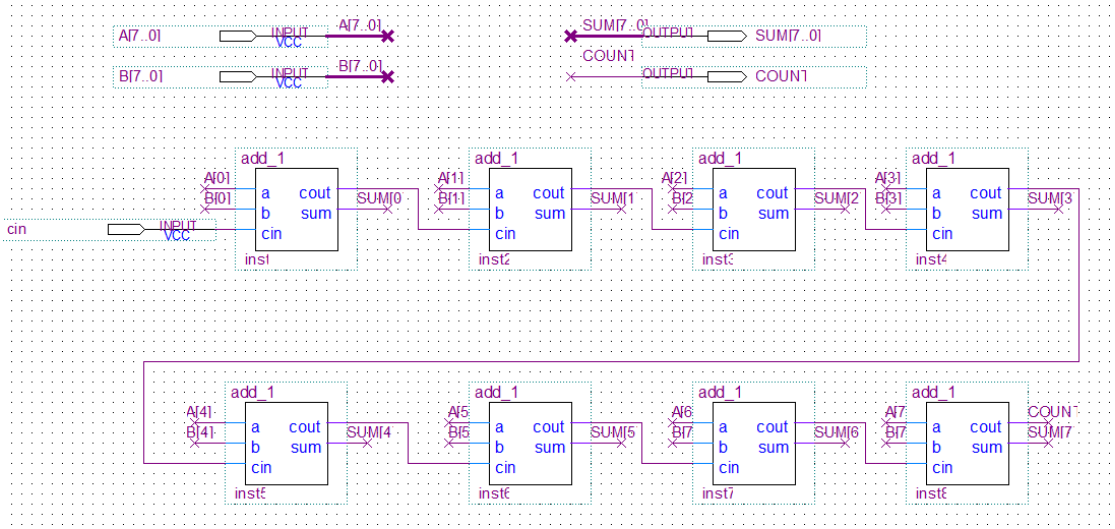
```
);                                          //一位全加器有三个输入信号，两个输出信号

input wire    a;
input wire    b;
input wire    cin;
output wire   cout;
output wire   sum;

wire      SYNTHESIZED_WIRE_0;               //定义三个综合过的线网信号
wire      SYNTHESIZED_WIRE_1;               //用于全加器之间的连接
wire      SYNTHESIZED_WIRE_2;

h_add    b2v_inst1(
      .a(a),
      .b(b),
      .cout(SYNTHESIZED_WIRE_2),
      .sum(SYNTHESIZED_WIRE_0));


h_add    b2v_inst2(
      .a(SYNTHESIZED_WIRE_0),
      .b(cin),
      .cout(SYNTHESIZED_WIRE_1),
      .sum(sum));
assign    cout = SYNTHESIZED_WIRE_1 | SYNTHESIZED_WIRE_2;
//一位全加器是否有进位是看低位是否有进位以及本位是否产生新的进位信号
endmodule
```



```
module add_8(                    //八位全加器也是三个输入信号，两个输出信号
    cin,
    A,
    B,
    COUNT,
    SUM
);


input wire    cin;
input wire    [7:0] A;
input wire    [7:0] B;
output wire  COUNT;
output wire  [7:0] SUM;

wire     [7:0] SUM_ALTERA_SYNTHESIZED;//用于暂时保存八个一位全加器的和
wire     SYNTHESIZED_WIRE_0;
wire     SYNTHESIZED_WIRE_1;
wire     SYNTHESIZED_WIRE_2;
wire     SYNTHESIZED_WIRE_3;
wire     SYNTHESIZED_WIRE_4;
wire     SYNTHESIZED_WIRE_5;
wire     SYNTHESIZED_WIRE_6;




add_1    b2v_inst(
    .a(A[0]),
    .b(B[0]),
    .cin(cin),
    .cout(SYNTHESIZED_WIRE_0),
    .sum(SUM_ALTERA_SYNTHESIZED[0]));


add_1    b2v_inst2(
    .a(A[1]),
    .b(B[1]),
    .cin(SYNTHESIZED_WIRE_0),
    .cout(SYNTHESIZED_WIRE_1),
    .sum(SUM_ALTERA_SYNTHESIZED[1]));
```

```
add_1    b2v_inst3(
      .a(A[2]),
      .b(B[2]),
      .cin(SYNTHESIZED_WIRE_1),
      .cout(SYNTHESIZED_WIRE_2),
      .sum(SUM_ALTERA_SYNTHESIZED[2]));


add_1    b2v_inst4(
      .a(A[3]),
      .b(B[3]),
      .cin(SYNTHESIZED_WIRE_2),
      .cout(SYNTHESIZED_WIRE_3),
      .sum(SUM_ALTERA_SYNTHESIZED[3]));


add_1    b2v_inst5(
      .a(A[4]),
      .b(B[4]),
      .cin(SYNTHESIZED_WIRE_3),
      .cout(SYNTHESIZED_WIRE_4),
      .sum(SUM_ALTERA_SYNTHESIZED[4]));


add_1    b2v_inst6(
      .a(A[5]),
      .b(B[5]),
      .cin(SYNTHESIZED_WIRE_4),
      .cout(SYNTHESIZED_WIRE_5),
      .sum(SUM_ALTERA_SYNTHESIZED[5]));


add_1    b2v_inst7(
      .a(A[6]),
      .b(B[6]),
      .cin(SYNTHESIZED_WIRE_5),
      .cout(SYNTHESIZED_WIRE_6),
      .sum(SUM_ALTERA_SYNTHESIZED[6]));


add_1    b2v_inst8(
      .a(A[7]),
      .b(B[7]),
      .cin(SYNTHESIZED_WIRE_6),
```

```
    .cout(COUNT),
    .sum(SUM_ALTERA_SYNTHESIZED[7]));

assign    SUM = SUM_ALTERA_SYNTHESIZED;

endmodule


`timescale 100 ps/ 10 ps
module add_8_vlg_tst();
// test vector input registers
reg [7:0] A;
reg [7:0] B;
reg cin;
// wires
wire COUNT;
wire [7:0]    SUM;

// assign statements (if any)
add_8 i1 (
// port map - connection between master ports and signals/registers
    .A(A),
    .B(B),
    .cin(cin),
    .COUNT(COUNT),
    .SUM(SUM)
);
always   #5 cin=~cin;          //每隔五个时间 进位信号就翻转一次，更加直观

initial                        //两个 initial 模块，初值 A 和 B 都是 8'b0
begin
A=8'b00000000;
B=8'b00000000;
cin=1'b0;
repeat(200)
    #10 B=B+1;
end

initial
begin
A=8'b00000000;
B=8'b00000000;
cin=1'b1;
repeat(200)
```

```
    #10 A=A+1;
end

endmodule
```

2.2.4 仿真结果与分析

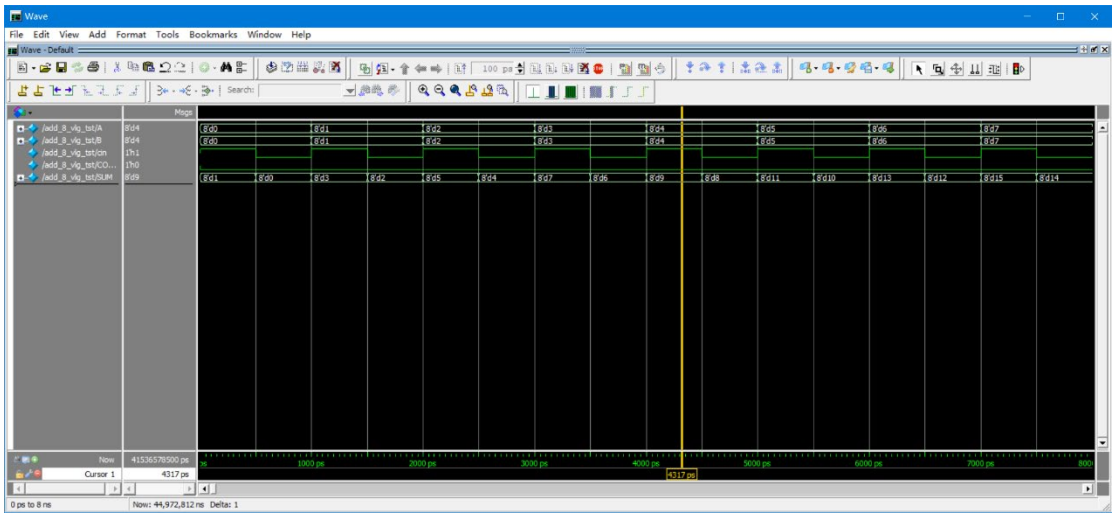


图 1-3 八位全加器功能仿真 1

`timescale 100 ps/ 10 ps，低位进位信号每隔 500ps 翻转一次，A 和 B 位输入信号，每隔 1000ps 则+1 可以发现在 127+127+1 以及之前无向高位的进位信号 COUT 产生

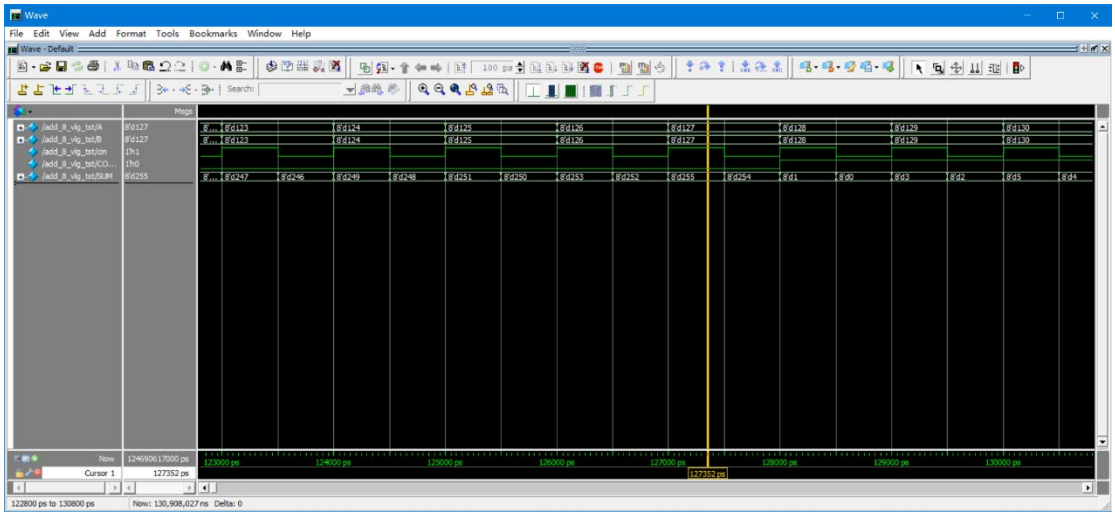


图 1-4 八位全加器功能仿真 2

A+B 大于 255 后，超过八位二进制数范围，产生向高位的进位信号

实验总结

1、 通过仿真和硬件实验的比较，你对哪些概念从不理解到理解了？

第一次进行 FPGA 的仿真和硬件实验，在 verilogHDL 文件中的 wire 和 reg 类型,以及测试模块中的类型,输入需要是 wire,输出可以是 wire/reg,而测试模块输入必须是 reg，输出必须是 wire，initial 和 always 不能互相嵌套，assign 只能对 wire 进行赋值，并且对如何通过 Modelsim 对测试模块进行仿真测试都有了比较详细的了解，加深了代码规范的印象。

2、 相比其它方法（比如数字逻辑设计、单片机等），你认为 FPGA 对数字电路设计有哪些优势？

可以不用考虑具体的数字逻辑实现单元的构成，但传统逻辑电路设计必须考虑这些，并自己搭建；设计灵活，可以不悲标准器件在逻辑功能上限制，并且由于 FPGA 的可编程性，可以大大缩减设计周期

3、 你对设计一个比较复杂的工程项目有何一般性的方法？

模块化设计，分为顶层和底层模块分开设计，将一个复杂的工程项目拆解成多个小模块，可以同时由多个人进行开发，提升设计效率，最后子模块调试完成后由顶层模块调用。

实验改进建议

建议一：

建议二：