计算机系统结构实验

lab2: FPGA 基础实验-4-bit Adder

周旭东 521021910829

2023 April

摘要

学习使用 Xilinx 逻辑设计工具 Vivado 的基本操作，Verilog HDL 进行简单的逻辑设计与功能仿真并使用 1/0 Planing 添加管脚约束生成 Bitstream 文件。在此基础上实现了一个四位全加器。

目录

1 实验目的

- 掌握 Xilinx 逻辑设计工具 Vivado 的基本操作
- 掌握使用 Verilog HDL 进行简单的逻辑设计
- 掌握功能仿真
- 使用 I/O Planing 添加管脚约束
- 生成 Bitstream 文件
- 上板验证

2 实验原理

2.1 一位全加器功能原理

FPGA 一位全加器是一个数字电路组件，用于将两个二进制位的数字相加，同时考虑任何先前的进位（carry）。

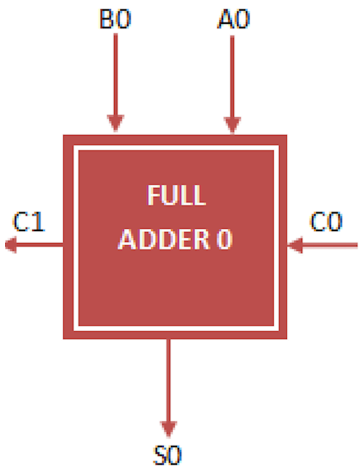

图 1: 一位全加器结构

一个一位全加器通常由三个输入和两个输出组成。三个输入是两个被加数的二进制位和前一位的进位。两个输出是当前位的和和向下一位的进位。当加数和进位位一起输入时，一位全加器会输出两个二进制数字，即当前位的和和下一位的进位。这些输出可以馈入其他一位全加器，以实现更复杂的数字计算。

2.2 四位全加器功能原理

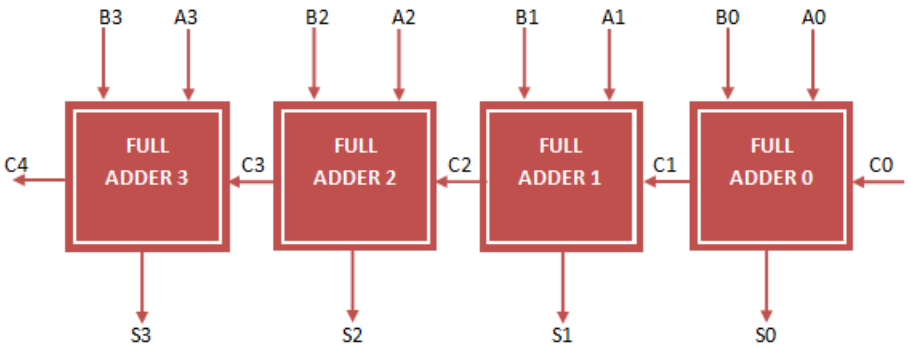

图 2: 四位全加器结构

FPGA 四位全加器是一个数字电路组件，用于将四个二进制位的数字相加，同时考虑任何先前的进位（carry）。2展示了这一结构。

一个四位全加器通常由三个输入和两个输出组成。三个输入是两个被加数的二进制位和前一位的进位。两个输出是当前位的和和向下一位的进位。当加数和进位位一起输入时，四位全加器会输出两个二进制数字，即当前位的和和下一位的进位。这些输出可以馈入其他四位全加器，以实现更复杂的数字计算。

3 功能实现

3.1 一位全加器功能实现

模块的内部，使用了四个 wire 变量 s1, c1, c2 和 c3 分别代表加法器中的中间变量。核心部分是由三个 and 门和一个 xor 门构成的加法器电路。其中 c1 = a b，c2 = b ci，c3 = a ci 分别代表了进位位的计算。s1 = a + b 表示了和值的计算。最终的进位位 co = c1 | c2 | c3 表示了三种情况下的进位情况。

```
module adder_1bit(
    input a,
    input b,
    input ci,
    output s,
    output co
    );
    wire s1, c1, c2, c3;
    and (c1, a,b),
```

```
        (c2, b, ci),
        (c3, a, ci);

    xor (s1, a, b),
        (s, s1, ci);

    or  (co, c1, c2, c3);

endmodule
```

3.2 四位全加器功能实现

利用四个 1 位全加器构建四位全加器如下。

```
module adder_4bits(
    input [3:0] a,
    input [3:0] b,
    input ci,
    output [3:0] s,
    output co
    );
    wire [2:0] ct;

    adder_1bit  a1(.a (a[0]), .b (b [0]), .ci (ci), .s (s [0]), .co(ct [0])),
                a2 (.a (a[1]), .b (b [1]), .ci (ct [0]), .s (s [1]), .co (ct [1])),
                a3 (.a (a[2]), .b (b [2]), .ci (ct [1]), .s (s [2]), .co (ct [2])),
                a4 (.a (a[3]), .b (b [3]), .ci (ct [2]), .s (s [3]), .co (co) );
endmodule
```

4 仿真测试

建立激励文件。首先对模块实例化，之后每隔 100 个时钟周期更改输入内容。

```
module adder_4bits_tb(  );
    reg [3:0] a;
    reg [3:0] b;
    reg ci;

    wire [3:0] s;
    wire co;
```

```
8
9       adder_4bits u0 (
10      .a (a),
11      .b (b),
12      .ci (ci),
13      .s (s),
14      .co (co)
15      );
16
17  initial begin
18      a = 0;
19      b = 0;
20      ci = 0;
21
22      #100;
23      a = 4'b0001;
24      b = 4'b0010;
25      #100;
26      a = 4'b0010;
27      b = 4'b0100;
28
29      #100;
30      a = 4'b1111;
31      b = 4'b0001;
32      #100;
33      ci = 1'b1;
34  end
35  endmodule
```

得到仿真结果如图 3所示。

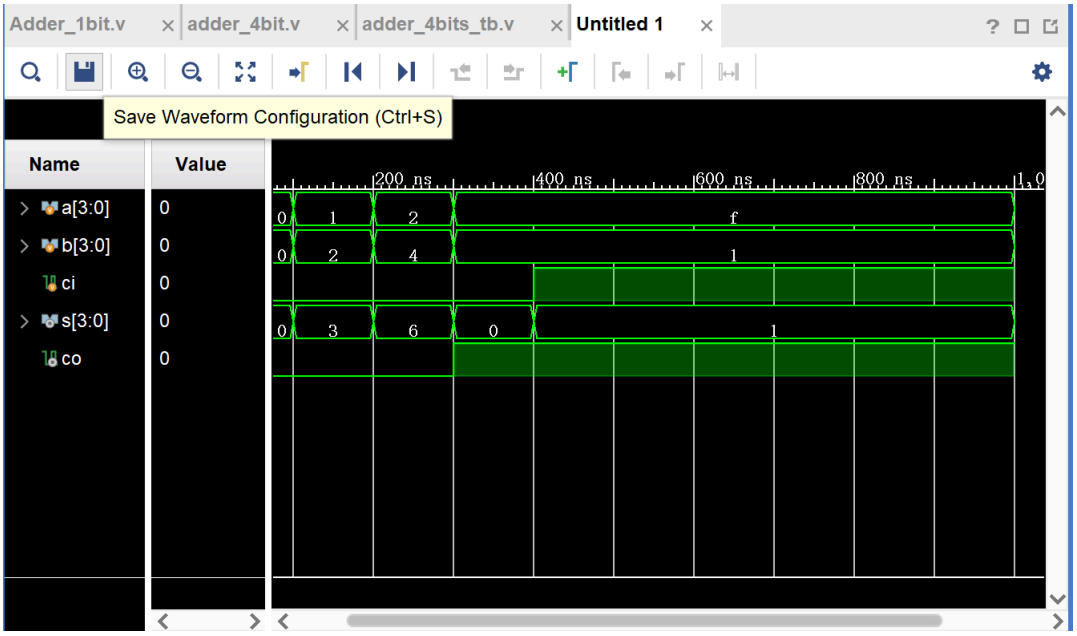

图 3: 四位全加器仿真测试

由结果可以看出全加器的功能实现正确。

5 管脚约束

构建管脚约束文件

```
set_property PACKAGE_PIN AC18 [get_ports clk_p]
set_property IOSTANDARD LVDS [get_ports clk_p]

set_property PACKAGE_PIN AA12 [get_ports {a[3]}]
set_property PACKAGE_PIN AA13 [get_ports {a[2]}]
set_property PACKAGE_PIN AB10 [get_ports {a[1]}]
set_property PACKAGE_PIN AA10 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {a[3]}]

set_property PACKAGE_PIN AD10 [get_ports {b[3]}]
set_property PACKAGE_PIN AD11 [get_ports {b[2]}]
set_property PACKAGE_PIN Y12 [get_ports {b[1]}]
set_property PACKAGE_PIN Y13 [get_ports {b[0]}]
```

```
set_property IOSTANDARD LVCMOS15 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {b[3]}]

set_property PACKAGE_PIN N26 [get_ports led_clk]
set_property PACKAGE_PIN M26 [get_ports led_do]
set_property PACKAGE_PIN P18 [get_ports led_en]
set_property IOSTANDARD LVCMOS33 [get_ports led_clk]
set_property IOSTANDARD LVCMOS33 [get_ports led_do]
set_property IOSTANDARD LVCMOS33 [get_ports led_en]

set_property PACKAGE_PIN M24 [get_ports seg_clk]
set_property PACKAGE_PIN L24 [get_ports seg_do]
set_property PACKAGE_PIN R18 [get_ports seg_en]
set_property IOSTANDARD LVCMOS33 [get_ports seg_clk]
set_property IOSTANDARD LVCMOS33 [get_ports seg_do]
set_property IOSTANDARD LVCMOS33 [get_ports seg_en]
```

6 上板验证

构建顶层文件进行 LED 显示

```
module Top(
    input clk_p,
    input clk_n,
    input [3:0] a,
    input [3:0] b,
    input reset,

    output led_clk,
    output led_do,
    output led_en,

    output wire seg_clk,
    output wire seg_en,
    output wire seg_do
    );
    wire CLK_i;
    wire Clk_25M;

```

```
19      IBUFGDS IBUFGDS_inst(
20          .O(CLK_i),
21          .I(clk_p),
22          .IB(clk_n)
23      );
24
25      wire [3:0] s;
26      wire co;
27      wire [4:0] sum;
28      assign sum = {co,s};
29
30      adder_4bits U1(
31          .a(a),
32          .b(b),
33          .ci(1'b0),
34          .s(s),
35          .co(co)
36      );
37      reg [1:0] clkdiv;
38      always @ (posedge CLK_i)
39          clkdiv<=clkdiv+1;
40      assign Clk_25M=clkdiv[1];
41
42      display DISPLAY(
43          .clk(Clk_25M),
44          .rst(1'b0),
45          .en(8'b00000011),
46          .data({27'b0,sum}),
47          .dot(8'b00000000),
48          .led(~{11'b0,sum}),
49          .led_clk(led_clk),
50          .led_en(led_en),
51          .led_do(led_do),
52          .seg_clk(seg_clk),
53          .seg_en(seg_en),
54          .seg_do(seg_do)
55      );
56  endmodule
```

图 4: 上板测试结果

如图 4所示，进行了 2+3=5 的测试，结果正确。

7 总结与反思

实验 2 深入学习使用了 Xilinx 逻辑设计工具 Vivado 的基本操作，Verilog HDL 进行简单的逻辑设计与功能仿真并使用 1/0 Planing 添加管脚约束生成 Bitstream 文件。在此基础上实现了一个四位全加器。

本次实验后，对 Vertilog 的语言有了更好的了解，例如 always 语段如何使用及其含义，begin 和 end 相当于括号，reg 表示寄存器，input 和 output 的含义，以及实例化模块的方法等等。

对 Vivado 的操作也更加熟悉，例如熟悉了如何添加各种文件，进行仿真和上板操作等等。为之后的实验打下基础。

8 致谢

感谢刘雨桐老师为实验提供了良好的讲解。
感谢助教蔡明昕、黄正翔老师帮助我解决在实验中遇到的问题。
感谢黄小平老师及实验室提供的资源和硬件支持。
感谢在实验过程中给予我帮助的同学。