

**数学计算机科学 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 计算机组成原理实验 | | | | |
| 实验编号： | Project 1 | | | | |
| 实验名称： | Verilog HDL模块化设计 | | | | |
| 实验人员： | 学号 | 15111206033 | | | |
| 姓名 | 孙浩 | | | |
| 班级 | 2015级物联网工程 | | | |
| 实验日期： | 2017/2/27 | | | | |
| 实验室： | 学苑南楼1幢305 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# Verilog HDL模块化设计

# 一、实验目的

## 1. 掌握Verilog HDL的模块化设计方法。

## 2. 掌握Verilog HDL的模块端口关联方法。

## 3. 掌握Verilog HDL的模块在的仿真和综合方法。

# 二、实验工具

## 1.Xilinx Vivado 2014.2软件。

## 2.Windows系统PC机。

# 三、实验要求

## 1．利用Verilog HDL的模块化设计方法建模。

## 2．仿真和综合Verilog HDL模块。

# 四、实验内容

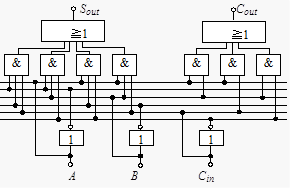
## 1. 全加器建模与仿真。

### (1) 全加器真值表和逻辑表达式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | cin | sout | cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



### (2) 全加器逻辑电路（与门和或门）



### (3) 创建工程

工程名：P2

创建模块文件fulladder.v

module fulladder(a,b,cin,sout,cout);

output sout,cout;

input a,b,cin;

wire a,b,cout,sout;

wire a\_bar,b\_bar,cin\_bar;

not not1(a\_bar,a);

not not2(b\_bar,b);

not not3(cin\_bar,cin);

and and4(and4\_out,a\_bar,b\_bar,cin);

and and5(and5\_out,a\_bar,b,cin\_bar);

and and6(and6\_out,a,b\_bar,cin\_bar);

and and7(and7\_out,a,b,cin);

or or8(sout,and4\_out,and5\_out,and6\_out,and7\_out);

and and9(and9\_out,a,b);

and and10(and10\_out,a,cin);

and and11(and11\_out,b,cin);

or or12(cout,and9\_out,and10\_out,and11\_out);

endmodule

创建test bench文件test\_fulladder.v

`timescale 1ns/100ps

module test\_fulladder();

reg my\_a,my\_b,my\_cin;

wire my\_sum,my\_cout;

reg clk;

fulladder u0(.a(my\_a),.b(my\_b),.cin(my\_cin),.sout(my\_sum),.cout(my\_cout));

always #10 clk=~clk;

initial

begin

clk=1'b0;

my\_a=1'b0;my\_b=1'b0;my\_cin=1'b0;

#100;

my\_a=1'b0;my\_b=1'b0;my\_cin=1'b1;

#100;

my\_a=1'b0;my\_b=1'b1;my\_cin=1'b0;

#100;

my\_a=1'b0;my\_b=1'b1;my\_cin=1'b1;

#100;

my\_a=1'b1;my\_b=1'b0;my\_cin=1'b0;

#100;

my\_a=1'b1;my\_b=1'b0;my\_cin=1'b1;

#100;

my\_a=1'b1;my\_b=1'b1;my\_cin=1'b0;

#100;

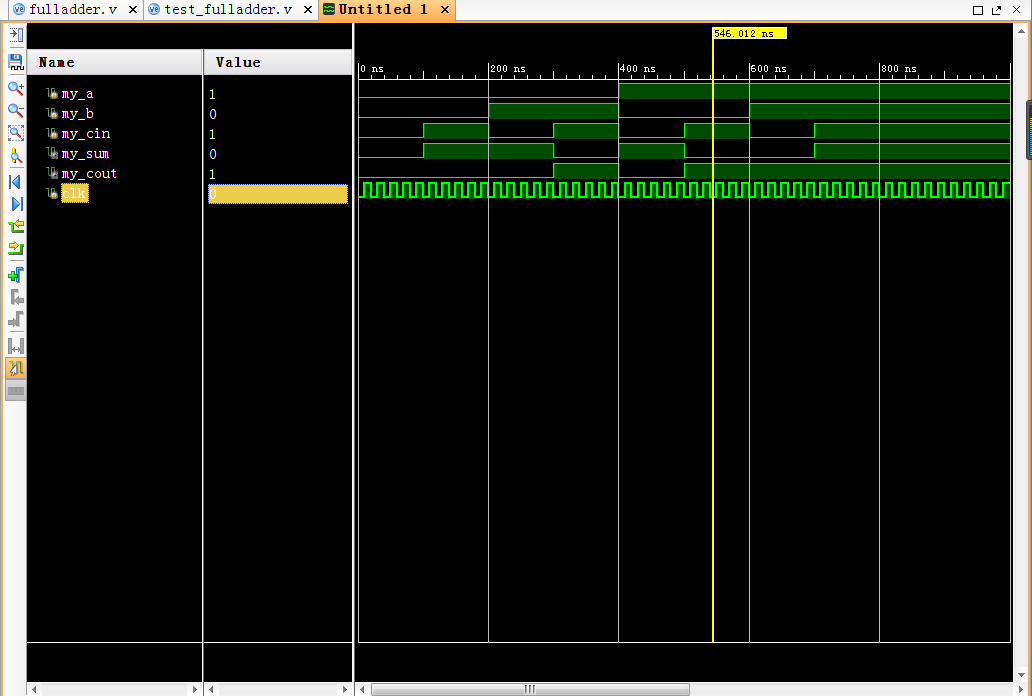
my\_a=1'b1;my\_b=1'b1;my\_cin=1'b1;

#100;

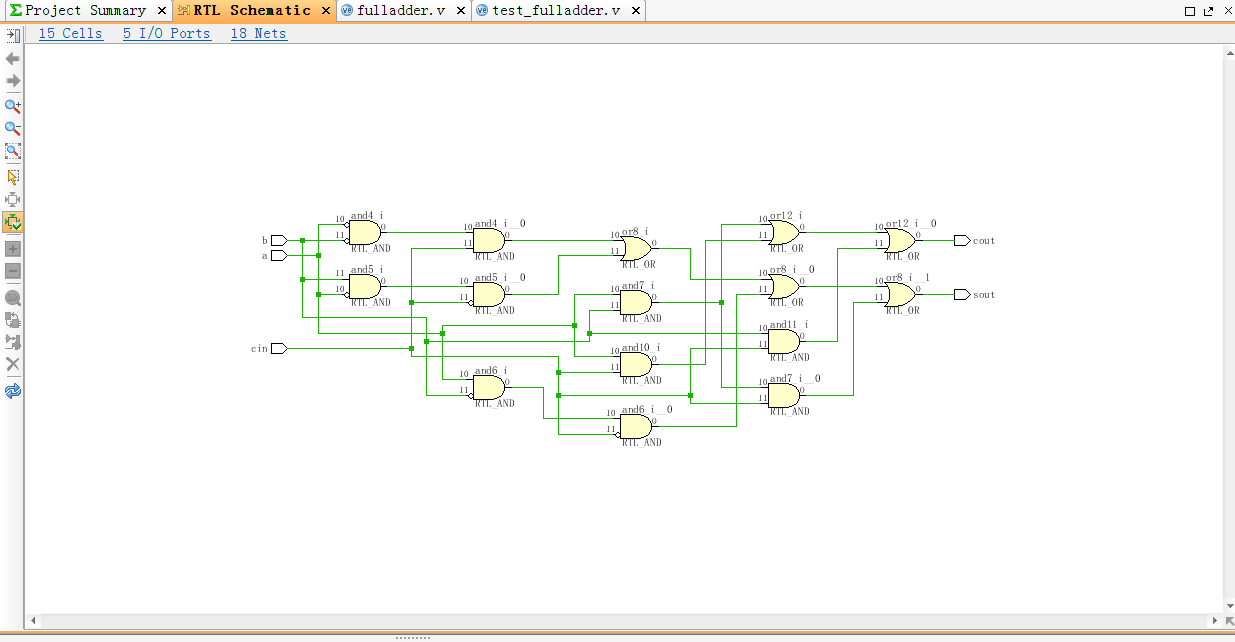
end

endmodule

### (4)利用Xilinx Vivado获取仿真波形图

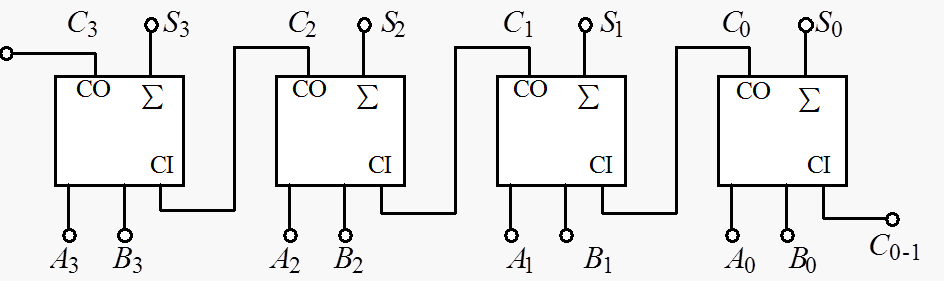


### (5)综合，查看RTL视图



## 2. 4位串行进位加法器建模与仿真

### (1) 利用全加器构造4位串行进位加法器



### (2)创建工程

在P2工程中新建模块文件adder.v。

方案一：位置关联法

module adder(a,b,cin,s,cout);

parameter N=4;

output wire [N-1:0] s;

output wire cout;

input wire [N-1:0]a;

input wire [N-1:0]b;

input wire cin;

wire [N-2:0]c;

fulladder FA1(a[0],b[0],cin,s[0],c[0]);

fulladder FA2(a[1],b[1],c[0],s[1],c[1]);

fulladder FA3(a[2],b[2],c[1],s[2],c[2]);

fulladder FA4(a[3],b[3],c[2],s[3],cout);

endmodule

方案二：名称关联法

module adder(a,b,ci,s,co);

parameter N=4;

output wire [N-1:0]s;

output wire co;

input wire [N-1:0]a;

input wire [N-1:0]b;

input wire ci;

wire [N-2:0]c;

fulladder FA1(.a(a[0]), .b(b[0]), .cin(ci),.sout(s[0]),.cout(c[0]));

fulladder FA2(.a(a[1]), .b(b[1]), .cin(c[0]), .sout(s[1]), .cout(c[1]));

fulladder FA3(.a(a[2]), .b(b[2]), .cin(c[1]), .sout(s[2]), .cout(c[2]));

fulladder FA4(.a(a[3]), .b(b[3]), .cin(c[2]), .sout(s[3]), .cout(co));

endmodule

创建test bench文件test\_adder.v

`timescale 1ns/100ps

module test\_adder();

reg [3:0] my\_a;

reg [3:0] my\_b;

reg my\_cin;

wire [3:0] my\_sum;

wire my\_cout;

reg clk;

adder u0(my\_a,my\_b,my\_cin,my\_sum,my\_cout);

always #10 clk=~clk;

initial

begin

clk=1'b0;

my\_a=4'b0000;my\_b=4'b0000;my\_cin=1'b0;

#100;

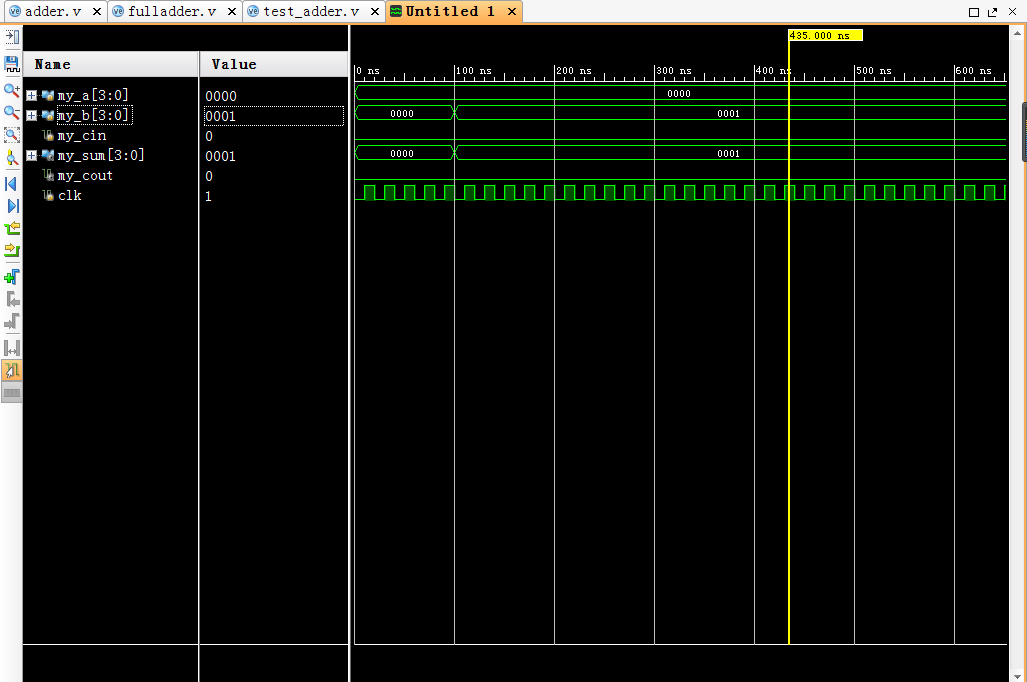
my\_a=4'b0000;my\_b=4'b0001;my\_cin=1'b0;

#100;//一共有2^9种情况 ，可以添加下其他激励信号

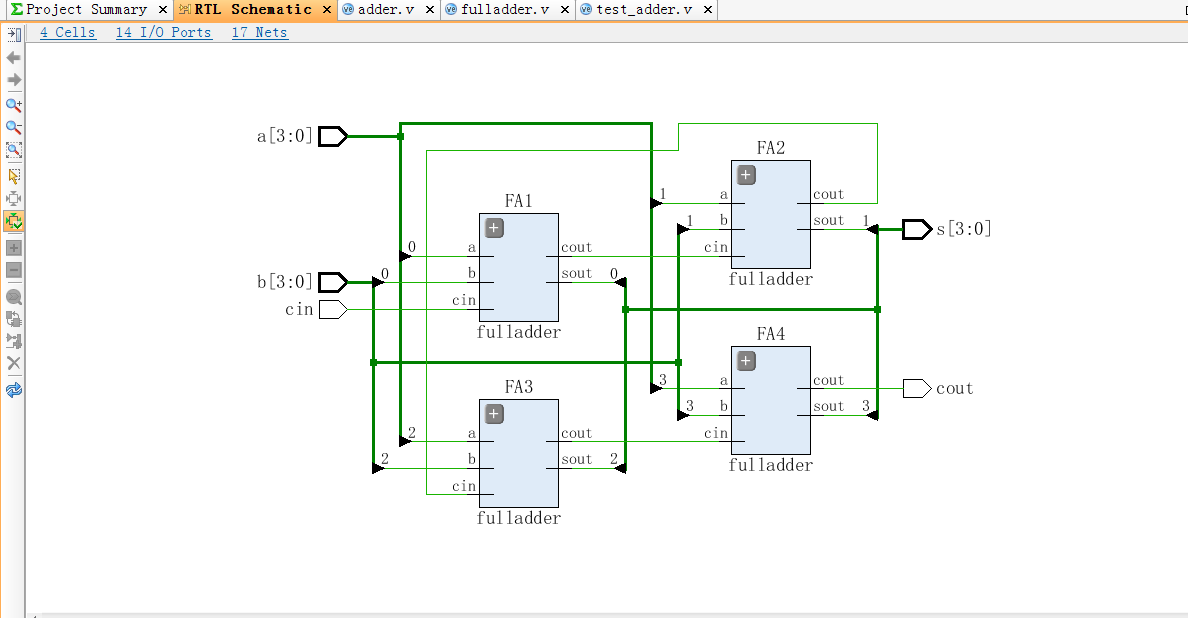
end

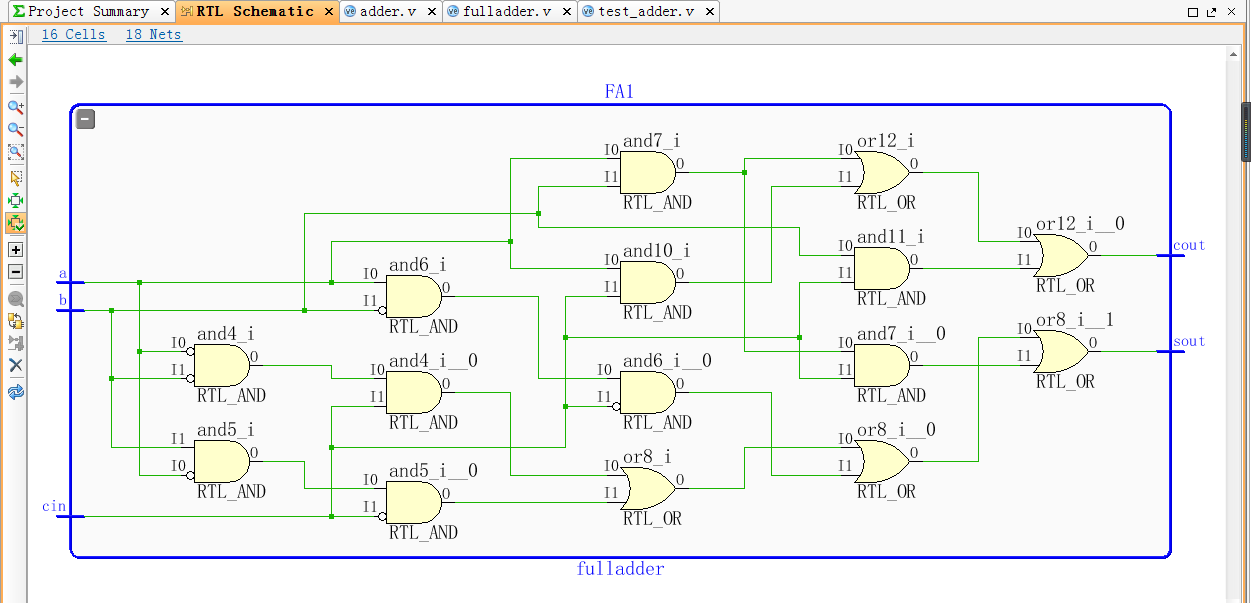
endmodule

### (4)利用Xilinx Vivado获取仿真波形图



### (5)综合，查看RTL视图





# 五、实验思考

## 1.Verilog HDL的模块和C语言的函数有何区别与联系？

区别：

C语言是一种大众化的计算机语言体系，C语言具有一定的普遍性，不具备什么对象性；Verilog HDL可以说是一种面向对象的计算机语言，它主要针对于硬件这个部分。

联系：

两者目的是相通的，都是为了提取公共模式，简化编程

2.为何要为所设计模块编写test bench文件？一般test bench的结构是什么？

编写test bench的目的是用来验证设计的正确性。通过模拟各种可能的情况，查看输入输出结果是否符合设计的要求。简单的test bench向要设计提供向量，人工验证输出。复杂的test bench是自检测的，其结果是自动验证。

一个完整的testbench包含下列几个部分：

（1）module的定义，一般无输入输出端口；

（2）信号的定义，定义哪些是你要输入，输入的定义；

（3）实例化待测试的模块；

（4）提供测试激励；