

# 浙江大学

## 本科实验报告

课程名称:	数字逻辑设计
姓 名:	
学 院:	计算机科学与技术学院
专 业:	计算机科学与技术
邮 箱:	
QQ 号:	
电 话:	
指导教师:	蔡铭
报告日期:	2023 年 11 月 29 日

# 浙江大学实验报告

课程名称： 数字逻辑设计 实验类型： 综合

实验项目名称： 全加器的设计实现

学生姓名： 学号： 同组学生姓名：

实验地点： 紫金港东四 509 室 实验日期： 2023 年 11 月 15 日

## 一、操作方法和实验步骤

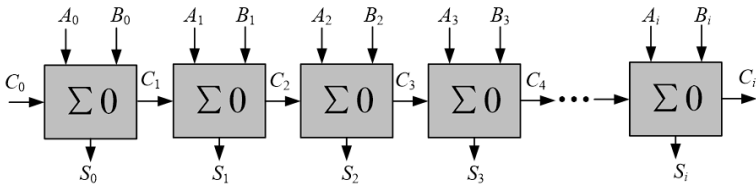
### 实验背景

- 全加器：将两个二进制数  $A$   $B$  和一个进位输入  $C_{in}$  相加，得到一个和输出  $S$  和一个进位输出  $C_{out}$
- 一位全加器：根据一位全加器的功能，我们很容易得到它的真值表并在化简之后得到其化简表达式。

A	B	Cin	S	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

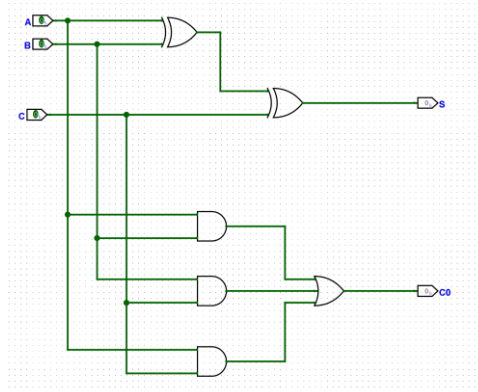
$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + BC_{in} + AC_{in}$$

- 行波加法器：将  $N$  个一位全加器依次相连，我们就可以得到  $N$  位行波加法器。



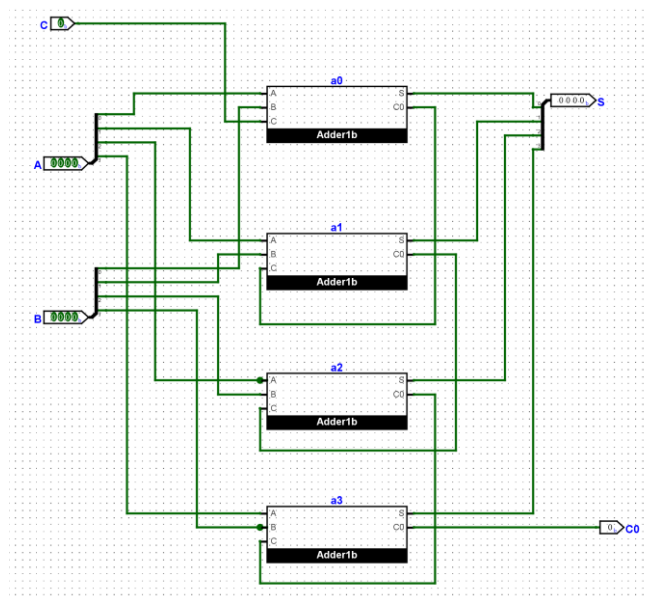
## 一位全加器的实现

- 根据上面我们得到的布尔表达式，我用电路图实现了一位全加器。



## 四位加法器的实现

- 利用行波加法器的原理，我用电路图实现了四位全加器。



## 防抖动模块

- 我们注意到，由于按钮在按下的过程中会有信号扰动，因此一次按下按钮时数字可能会多次闪烁，因此我们在本次实验中加入了防抖动模块，当 **信号稳定一段时间** 后才改变信号值。
- 以下为 Verilog 代码实现

```

module pbdebounce (
    input wire clk,
    input wire button,
    output reg pbreg
);

    reg [7:0] pbshift;

    always@(posedge clk) begin
        pbshift = pbshift<<1;
        pbshift[0] = button;
        if (pbshift==8'b0)
            pbreg=0;
        if (pbshift==8'hFF)
            pbreg=1;
    end

endmodule

```

- 可以看到,每次时钟上升沿时,我们将 **pbshift** 左移一位并在最低位赋值 **button** 的按下状态,只有在 8 个时钟周期中按钮均为按下,即 **pbshift** 为 8'hFF,我们才视其为有效按钮。这样的操作等效地滤过了按钮信号的抖动。

## CreateNumber 模块

- 我们将上一次实验使用的 **CreateNumber** 模块进行更新,加入四位加法器和防抖动模块,实现四位显示为  $C_0CBA$ , 其中  $C_0$  和  $C$  分别为  $A+B$  的进位和加法。  
按钮按下的自增操作也从原来的 **Verilog** 代码替换为全加器的实现。

```

module CreateNumber (
    input wire clk,
    input wire [1:0] btn,
    output reg [7:0] num,
    output C1,
    output [3:0] sum
);

    wire [31:0] div;
    clkdiv cd0(
        .clk(clk),
        .rst(1'b0),
        .div_res(div)
    );

    wire [3:0] A, B;
    wire A1, B1;
    wire [1:0] temp_btn;

    initial num <= 0;

    Adder4b a0 (
        .A(num[3:0]),
        .B(1'b1),
        .C(1'b0),
        .S(A),

```

```

        .C0(A1)
    );

    Adder4b a1 (
        .A(num[7:4]),
        .B(1'b1),
        .C(1'b0),
        .S(B),
        .C0(B1)
    );

    Adder4b a2 (
        .A(num[3:0]),
        .B(num[7:4]),
        .C(1'b0),
        .S(sum),
        .C0(C1)
    );

    pbdebounce pb0 (
        .clk(div[17]),
        .button(btn[0]),
        .pbreg(temp_btn[0])
    );

    pbdebounce pb1 (
        .clk(div[17]),
        .button(btn[1]),
        .pbreg(temp_btn[1])
    );

    always@(posedge temp_btn[0])    num[3:0] <= A;
    always@(posedge temp_btn[1])    num[7:4] <= B;

endmodule

```

## 顶层模块

- 最后，我们将 CreateNumber 和 disp\_num 模块组合实现 top 模块。

```

module top(
    input clk,
    input [7:0] SW,
    input [1:0] btn,
    output [3:0] AN,
    output [7:0] SEGMENT,
    output BTN_X
);

    wire [7:0] num;
    wire [3:0] sum;
    wire C1;

    assign BTN_X = 1'b0;

    CreateNumber c0 (
        .btn(btn),
        .num(num),
        .clk(clk),

```

```

        .sum(sum),
        .C1(C1)
    );

    DisplayNumber d0(
        .clk(clk),
        .rst(1'b0),
        .hexs({3'b0,C1,sum,num}),
        .points(SW[7:4]),
        .LEs(SW[3:0]),
        .AN(AN),
        .SEGMENT(SEGMENT)
    );
endmodule

```

## 二、实验结果与分析

### 四位加法器的仿真激励

- 在仿真过程中，我们遍历 输入 A 和 B 的所有情况，并观察 Sum 和 Cout 的结果。

```

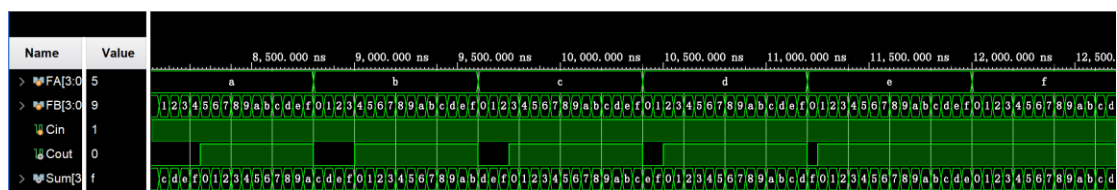
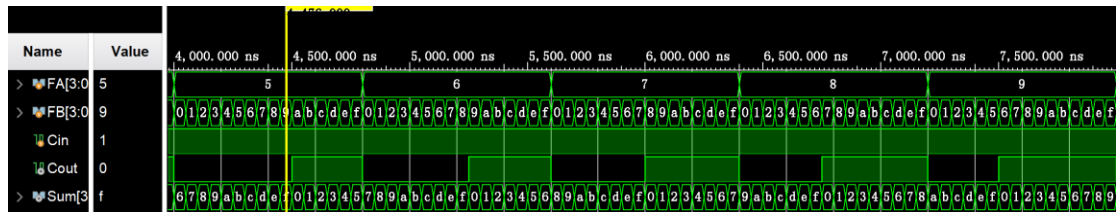
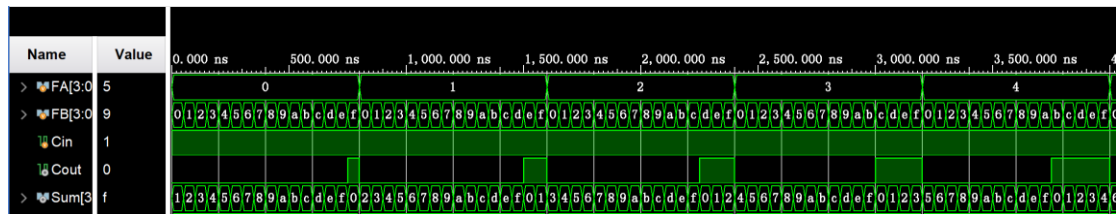
`timescale 1ns / 1ps
module Adder_tb();
    reg [3:0] FA;
    reg [3:0] FB;
    reg Cin;
    Adder4b a0(
        .A(FA),
        .B(FB),
        .C(Cin),
        .C0(Cout),
        .S(Sum)
    );

    wire Cout;
    wire [3:0] Sum;
    integer i, j;

    initial begin
        Cin = 1'b1;
        for (i = 0; i < 16; i = i + 1) begin
            for (j = 0; j < 16; j = j + 1) begin
                FA = i;
                FB = j;
                #50;
            end
        end
    end
endmodule

```

- 为了同时验证 进位 Cin 的情况，我们在实验中恒将 Cin 置一，对激励结果的验证并不会产生影响。



- 对激励结果可以举例说明：

$Cin = 1, FA = 0, FB = 1, Sum = 2, Cout = 0$

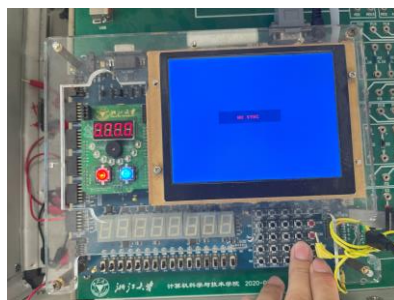
$Cin = 1, FA = A, FB = 5, Sum = 0, Cout = 1$

- 其他情况也类似，可以验证全加器功能的正确性。

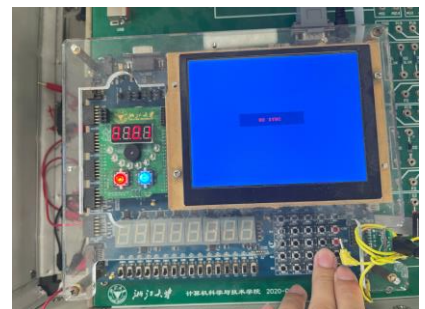
## MyAdder 下板验证



初始状态：0211


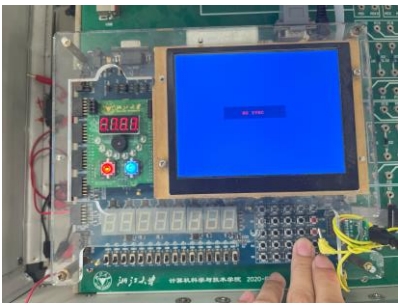
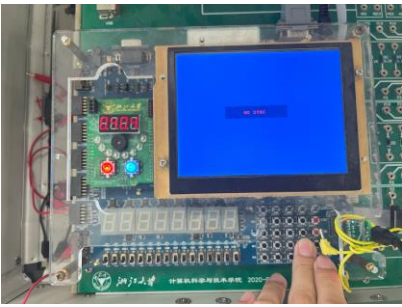







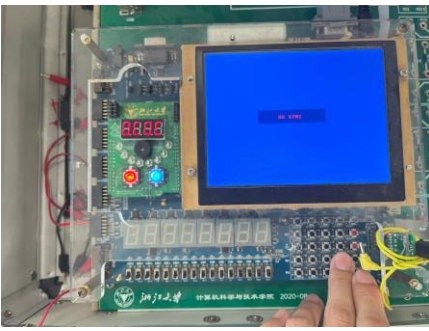
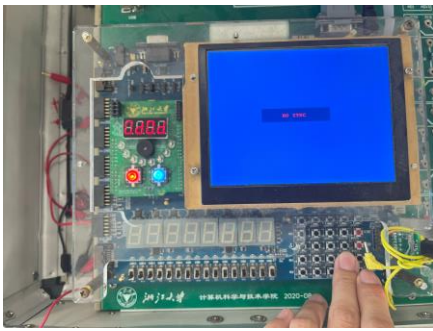


接下来从 0000 开始验证

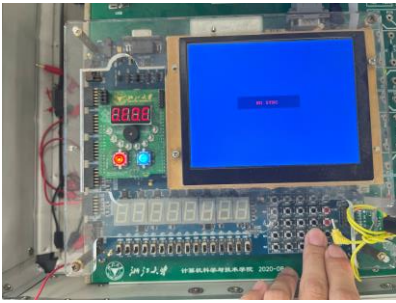
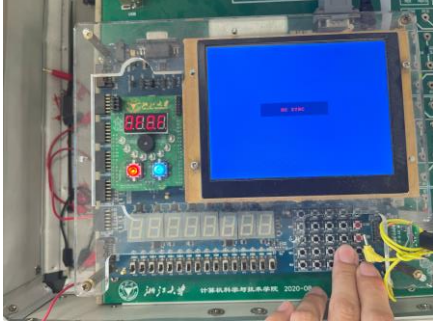

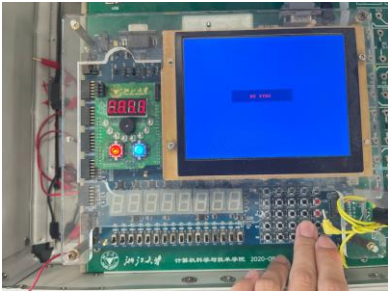

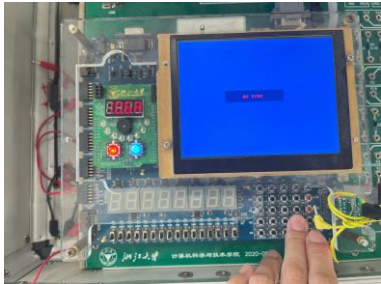

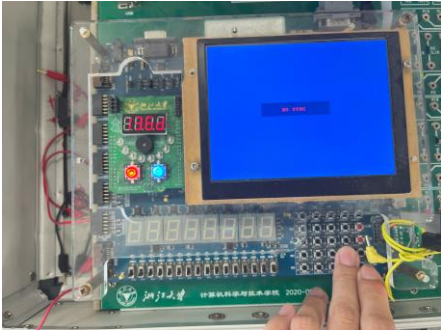
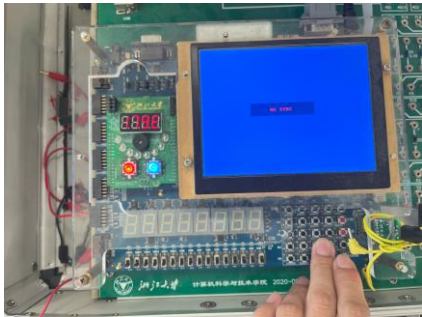





$0 + 1 = 1$



		
$0 + 2 = 2$	$0 + 3 = 3$	$0 + 4 = 4$
		
$0 + 5 = 5$	$0 + 6 = 6$	$0 + 7 = 7$
		
$0 + 8 = 8$	$0 + 9 = 9$	$0 + a = a$
		
$0 + b = b$	$0 + c = c$	$0 + d = d$



		
$0 + e = e$	$0 + f = f$	$1 + 2 = 3$
		
$5 + 3 = 8$	$6 + 7 = d$	$6 + a = (1) 0$ 验证进位
		
$6 + b = (1) 1$	$6 + d = (1) 3$	$9 + e = (1) 7$
		
$d + e = (1) b$	$f + e = (1) d$	$f + f = (1) e$

### 三、讨论、心得

本次实验我实现了全加器，这次理论课和实验课也为我揭示了计算机实现加法计算的本质，事实上就是对加法真值表的电路实现，并没有什么神秘。

本次实验中我也总算有点上手 Verilog 书写了，开始能够看懂基本的 Verilog 代码并自己进行书写。

事后来看来，其实本次实验要求完成的 Verilog 代码并不复杂，由于加法器的实现我依然使用原理图实现，因此代码实现的事实上就是连线的工作，只需要理解了整个加法器从实现到显示的逻辑，就能够比较轻松地把对应的线和接口连好。