# 实验7—全加器的设计分析实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 赵冰骞 | | 学号： | 3090103420 | | | 专业： | 理科试验班 | | |
| 课程名称： | | 逻辑与计算机设计基础实验 | | | 同组学生姓名： | 朱里 | | | |
| 实验时间： | | 2010-11-19 | 实验地点： | | 紫金港东4-509 | | 指导老师： | | 王总辉 | |

# 一、实验目的和要求

1. 掌握一位全加器的工作原理和逻辑功能
2. 掌握串行进位加法器的工作原理和进位延迟
3. 掌握超前进位的工作原理
4. 掌握减法器的实现原理
5. 了解加法器在CPU中的地位
6. 掌握FPGA开发平台进行简单的I/O数据交互

# 二、实验内容和原理

## 2.1 一位全加器

* 三个输入位：数据位Ai 和Bi，低位进位输入Ci
* 二个输出位：全加和Si，进位输出Ci+1

图表1 一位全加器真值表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ai | Bi | Ci | Si | Ci+1 |
| 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 |



图表 2 一位全加器电路图

## 2.2 多位串行进位加法器

* 多位全加器可由一位全加器将进位串接构成
* 高位进位生成速度慢，位数越多时间越长



图表 3 多位串行进位加法器

## 2.3 多位串行进位全减器

* 用负数补码加法实现，减数当作负数求补码
* 共用加法器
* 用“异或”门控制求反，最低进位位加一



图表 4 多位串行进位全减器

## 2.4 超前进位加法器

* 超前进位加法器用当前输入直接产生进位输出
* 对于给定的第i 位全加器，其进位有如下规律：
  + 当Ai=Bi=1时，无论是否有进位输入，都有进位产生，即Ci+1=1
    - 定义进位生成函数：Gi = Ai ∧Bi
  + 当半加器和为1，且进位Ci = 1时，进位被传递至输出，即Ci+1=1
    - 定义进位传递函数：Pi = Ai⊕Bi
* 输出进位通过二个门（与门、或门）即可得到



图表 5 四位超前进位加法器

* 当加法器位数较多时，进位输出函数式很长，电路很复杂
* 为了在速度和复杂性之间权衡，可采用分组超前进位方式
* 16位超前进位加法器可分4 组4位加法器，每4位加法有一个向上的进位输出
  + 每组加法部件产生组进位函数
  + C16表达式形式上与C4相同，可以使用4位超前进位电路



图表6 16位分组超前进位加法器

# 三、主要仪器设备

1. 装有ISE的计算机系统 1台
2. Spartan-III 开发板 1套

# 四、操作方法与实验步骤

## 4.1 实现16位加减器的调试仿真

1. 用Verilog结构化描述语言实现16位加减器

add\_sub\_16bits

|  |
| --- |
| `timescale 1ns / 1ps  module add\_sub\_16bits(  input wire [16:1] A,B,  input wire Ctr,  output [16:1] S,  output wire Co  );  wire [16:1] Bo;  wire [15:1] Ctemp;    assign Bo = {16{Ctr}} ^ B;  adder\_1bit A1\_1(A[1], Bo[1], Ctr, S[1], Ctemp[1]),  A1\_2(A[2], Bo[2], Ctemp[1], S[2], Ctemp[2]),  A1\_3(A[3], Bo[3], Ctemp[2], S[3], Ctemp[3]),  A1\_4(A[4], Bo[4], Ctemp[3], S[4], Ctemp[4]),  A1\_5(A[5], Bo[5], Ctemp[4], S[5], Ctemp[5]),  A1\_6(A[6], Bo[6], Ctemp[5], S[6], Ctemp[6]),  A1\_7(A[7], Bo[7], Ctemp[6], S[7], Ctemp[7]),  A1\_8(A[8], Bo[8], Ctemp[7], S[8], Ctemp[8]),  A1\_9(A[9], Bo[9], Ctemp[8], S[9], Ctemp[9]),  A1\_10(A[10], Bo[10], Ctemp[9], S[10], Ctemp[10]),  A1\_11(A[11], Bo[11], Ctemp[10], S[11], Ctemp[11]),  A1\_12(A[12], Bo[12], Ctemp[11], S[12], Ctemp[12]),  A1\_13(A[13], Bo[13], Ctemp[12], S[13], Ctemp[13]),  A1\_14(A[14], Bo[14], Ctemp[13], S[14], Ctemp[14]),  A1\_15(A[15], Bo[15], Ctemp[14], S[15], Ctemp[15]),  A1\_16(A[16], Bo[16], Ctemp[15], S[16], Co);  endmodule  module adder\_1bit(a, b, ci, s, co);  // port declaration  input wire a,b,ci;  output wire s,co;  wire 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 |

1. 令A[15:0] = AAAA, B[15:0] = BBBB，对add\_sub\_16bits模块进行仿真
2. 记录并分析仿真结果，若有错误应返回修改代码

## 4.2 16位计算器设计

1. 功能：
   1. 两个16位操作数支持按键修改
   2. 进入计算状态后，可进行加/减/与/或操作，并显示结果
2. 步骤：
   1. 编写Top模块

|  |
| --- |
| `timescale 1ns / 1ps  module top(  input wire clk,  input wire [3:0]btn\_in,  input wire[1:0]switch,  output wire [3:0]anode,  output wire [7:0]segment  );    //variable definition: op1, op2,disp\_num…  reg [15:0] display\_num;  wire [15:0] op1,op2;  wire [3:0] btn\_out;  wire [15:0] result;  pbdebounce b0(clk, btn\_in[0], btn\_out[0]);  pbdebounce b1(clk, btn\_in[1], btn\_out[1]);  pbdebounce b2(clk, btn\_in[2], btn\_out[2]);  pbdebounce b3(clk, btn\_in[3], btn\_out[3]);  display m0(clk, 0, display\_num, 4'b1111, anode, segment); //display module  calculate\_result m1(btn\_out[3:0], op1, op2, result); //calculate result  create\_oprands m2(switch[1:0], btn\_out[3:0], op1, op2); //generate op1/op2  //pbdebouce…  always @\* begin  case (switch)  2'b01: display\_num = op1; //operand 1  2'b10: display\_num = op2; //operand 2  2'b00: display\_num = result; //result  endcase  end  endmodule |

* 1. 调用前面16为全加器和显示模块
  2. 增加创建操作数模块/计算结果模块

|  |
| --- |
| `timescale 1ns / 1ps  module create\_oprands(  input wire [1:0] switch,  input wire [3:0] btn,  output reg [15:0] op1,op2  );  initial op1 <= 16'b1010\_1011\_1100\_1101; //op1,op2初始化  initial op2 <= 16'b1010\_1011\_1100\_1101;  always @(posedge btn[0])begin  if(switch == 2'b01) op1[ 3: 0]<= op1[ 3: 0] + 1'd1;  else if(switch == 2'b10) op2[ 3: 0]<= op2[ 3: 0] + 1'd1;  end  always @(posedge btn[1])begin  if(switch == 2'b01) op1[ 7: 4]<= op1[ 7: 4] + 1'd1;  else if(switch == 2'b10) op2[ 7: 4]<= op2[ 7: 4] + 1'd1;  end  always @(posedge btn[2])begin  if(switch == 2'b01) op1[ 11: 8]<= op1[ 11: 8] + 1'd1;  else if(switch == 2'b10) op2[ 11: 8]<= op2[ 11: 8] + 1'd1;  end  always @(posedge btn[3])begin  if(switch == 2'b01) op1[ 15: 12]<= op1[ 15: 12] + 1'd1;  else if(switch == 2'b10) op2[ 15: 12]<= op2[ 15: 12] + 1'd1;  end  endmodule |

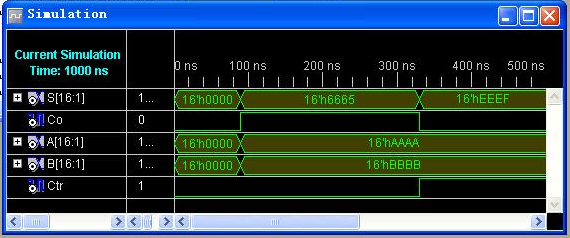
|  |
| --- |
| `timescale 1ns / 1ps  module calculate\_result(  input wire [3:0] btn,  input wire [15:0] op1,  input wire [15:0] op2,  output reg [15:0] result  );    wire [15:0] add\_sub\_result;  wire co;  add\_sub\_16bits m0(op1, op2, btn[1], add\_sub\_result, co);    always @(btn) begin  if(btn[0] == 1 || btn[1] == 1)  result = add\_sub\_result;  if(btn[2] == 1)  result = op1 & op2;  if(btn[3] == 1)  result = op1 | op2;  end  endmodule |

* 1. 编写UCF文件。

|  |
| --- |
| NET "clk" LOC = "t9" ;  NET "btn\_in[0]" LOC = "m13" ;  NET "btn\_in[1]" LOC = "m14" ;  NET "btn\_in[2]" LOC = "l13" ;  NET "btn\_in[3]" LOC = "l14" ;  NET "switch[0]" LOC = "k13" ;  NET "switch[1]" LOC = "k14" ;  NET "segment[0]" LOC = "e14" ;  NET "segment[1]" LOC = "g13" ;  NET "segment[2]" LOC = "n15" ;  NET "segment[3]" LOC = "p15" ;  NET "segment[4]" LOC = "r16" ;  NET "segment[5]" LOC = "f13" ;  NET "segment[6]" LOC = "n16" ;  NET "segment[7]" LOC = "p16" ;  NET "anode[0]" LOC = "D14" ;  NET "anode[1]" LOC = "G14" ;  NET "anode[2]" LOC = "F14" ;  NET "anode[3]" LOC = "E13" ; |

* 1. 下载到板子上验证。

# 五、实验结果与分析



图表7 16位加减器的调试仿真结果

* 16位加减器的调试仿真结果正常

# 六、讨论、心得

这次实验是较麻烦的一次实验，需要调用上一次实验的显示模块和去抖模块，另外加上创建操作数模块和计算结果模块，再在top模块中集成调用。这需要我们对各模块的变量非常清楚，定义正确，调用无误。同时，实验中进行了波形仿真，再次练习让我们熟练掌握。

总体来说，只要理清楚实验的思路，理解各变量的意义，这次实验进行的还是顺利的。