Computer Architecture Homework of Lmxyy*

李沐阳 516021910346
Shanghai Jiao Tong University

2017 年 11 月 4 日

*https://github.com/lmxyy/Computer-Architecture-Task-1

目录

1 Task 1

1.1 Subtask 1

```
module adder1(x,y,sum,cin,cout);
   input x,y,cin;
   output sum,cout;

   assign sum = x^y^cin;
   assign cout = (x&y)^(cin&(x^y));
endmodule // adder1

module adder(x,y,sum);
   parameter maxn = 16;

   input [maxn-1:0] x,y;
   output [maxn-1:0] sum;

   wire [maxn-1:0]      c;

   adder1 obj0(.x(x[0]),.y(y[0]),.sum(sum[0]),.cin(z),.cout(c[0]));
   adder1 obj1(.x(x[1]),.y(y[1]),.sum(sum[1]),.cin(c[0]),.cout(c[1]));
   adder1 obj2(.x(x[2]),.y(y[2]),.sum(sum[2]),.cin(c[1]),.cout(c[2]));
   adder1 obj3(.x(x[3]),.y(y[3]),.sum(sum[3]),.cin(c[2]),.cout(c[3]));
   adder1 obj4(.x(x[4]),.y(y[4]),.sum(sum[4]),.cin(c[3]),.cout(c[4]));
   adder1 obj5(.x(x[5]),.y(y[5]),.sum(sum[5]),.cin(c[4]),.cout(c[5]));
   adder1 obj6(.x(x[6]),.y(y[6]),.sum(sum[6]),.cin(c[5]),.cout(c[6]));
   adder1 obj7(.x(x[7]),.y(y[7]),.sum(sum[7]),.cin(c[6]),.cout(c[7]));
   adder1 obj8(.x(x[8]),.y(y[8]),.sum(sum[8]),.cin(c[7]),.cout(c[8]));
   adder1 obj9(.x(x[9]),.y(y[9]),.sum(sum[9]),.cin(c[8]),.cout(c[9]));
   adder1 obj10(.x(x[10]),.y(y[10]),.sum(sum[10]),.cin(c[9]),.cout(c[10]));
   adder1 obj11(.x(x[11]),.y(y[11]),.sum(sum[11]),.cin(c[10]),.cout(c[11]));
   adder1 obj12(.x(x[12]),.y(y[12]),.sum(sum[12]),.cin(c[11]),.cout(c[12]));
   adder1 obj13(.x(x[13]),.y(y[13]),.sum(sum[13]),.cin(c[12]),.cout(c[13]));
   adder1 obj14(.x(x[14]),.y(y[14]),.sum(sum[14]),.cin(c[13]),.cout(c[14]));
   adder1 obj15(.x(x[15]),.y(y[15]),.sum(sum[15]),.cin(c[14]),.cout(z));

endmodule // adder
```

1.2 Subtask 2

```
module adder4(x,y,sum,cin,cout);
   parameter maxn = 4;

   input [maxn-1:0] x,y;
   input            cin;
   output [maxn-1:0] sum;
   output             cout;

   assign sum[0] = x[0]^y[0]^cin;
   assign sum[1] = x[1]^y[1]^((x[0]&y[0])^((x[0]^y[0])&cin));
   assign sum[2] =
    x[2]^y[2]^((x[1]&y[1])^((x[1]^y[1])&(x[0]&y[0]))^((x[1]^y[1])&(x[0]^y[0])&cin));
   assign sum[3] = x[3]^y[3]^((x[2]&y[2])^((x[2]^y[2])&(x[1]&y[1]))
                              ^((x[2]^y[2])&(x[1]^y[1])&(x[0]&y[0]))
                              ^((x[2]^y[2])&(x[1]^y[1])&(x[0]^y[0])&cin));
   assign cout = (x[3]&y[3])^((x[3]^y[3])&(x[2]&y[2]))
     ^((x[3]^y[3])&(x[2]^y[2])&(x[1]&y[1]))
     ^((x[3]^y[3])&(x[2]^y[2])&(x[1]^y[1])&(x[0]&y[0]))
       ^((x[3]^y[3])&(x[2]^y[2])&(x[1]^y[1])&(x[0]^y[0])&cin);
endmodule // adder4

module adder(x,y,sum);
   parameter maxn = 16;

   input [maxn-1:0] x,y;
   output [maxn-1:0] sum;

   wire              c0,c1,c2;

   adder4 obj1(.x(x[3:0]),.y(y[3:0]),.sum(sum[3:0]),.cin(z),.cout(c0));
   adder4 obj2(.x(x[7:4]),.y(y[7:4]),.sum(sum[7:4]),.cin(c0),.cout(c1));
   adder4 obj3(.x(x[11:8]),.y(y[11:8]),.sum(sum[11:8]),.cin(c1),.cout(c2));
   adder4 obj4(.x(x[15:12]),.y(y[15:12]),.sum(sum[15:12]),.cin(c2),.cout(z));
endmodule // adder
```

1.3 Subtask 3

答：由于超前 16 位进位加法器中，进位不存在数据依赖性，即进位不需要依赖前一位的结果，可以从输入信号中直接得出，从而可以进行并行计算。他优化了暴力加法器中，数据转运所需要的时间。

2 Task 2

2.1 Subtask 1

```
/* ACM Class System (I) 2017 Fall Homework 1
 *
 * PART II: Correct the following program
 *
 * GUIDE:
 *   1. Create a RTL project in Vivado
 *   2. Put this file into `Simulation Sources'
 *   3. Run Behavioral Simulation
 *   4. You can see the results in `Tcl console'
 *
 */

module testfault;
    reg[15:0] a,b;
    wire[15:0] answer;
    fault f(a,b,answer);
    initial begin
        a = 10;
        b = 20;

        #1;
        if(answer != 20) begin
            $display("Expected 20, got %d", answer);
            $fatal("Wrong Answer");
        end

        #1;

        a = 40;
        b = 30;

        #1;

        if(answer != 40) begin
            $display("Expected 40, got %d", answer);
            $fatal("Wrong Answer");
        end

```

```
        $display("Congratulations! You have passed this test.");
        $finish;
    end
endmodule

module fault(a,b,answer);
   input wire[15:0] a,b;
   output wire [15:0] answer;
   assign answer = a>b?a:b;
endmodule
```

2.2 Subtask 2

```
/* ACM Class System (I) 2017 Fall Homework 1
 *
 * PART II: Correct the following program
 *
 * GUIDE:
 *   1. Create a RTL project in Vivado
 *   2. Put this file into `Simulation Sources'
 *   3. Run Behavioral Simulation
 *   4. You can see the results in `Tcl console'
 *
 */

module testfault;
    reg[15:0] a,b;
    wire[15:0] answer;
    fault f(a,b,answer);
    initial begin
        a = 10;
        b = 20;

        #1;
        if(answer != 20) begin
            $display("Expected 20, got %d", answer);
            $fatal("Wrong Answer");
        end

        #1;

        a = 40;
```

```
30          b = 30;
31
32          #1;
33
34          if(answer != 40) begin
35              $display("Expected 40, got %d", answer);
36              $fatal("Wrong Answer");
37          end
38
39          $display("Congratulations! You have passed this test.");
40          $finish;
41      end
42  endmodule
43
44  module fault(a,b,answer);
45      input wire[15:0] a,b;
46      output reg[15:0] answer;
47
48     always@(a,b)
49       begin
50          if(a>b) answer=a;
51          else answer=b;
52       end
53  endmodule
```

2.3 Subtask 3

答：从软件思路讲，这时：

- wire 对应于连续赋值，如 assign;
- reg 对应于过程赋值，如 always，initial。

从电路角度考虑，这时：

- wire 型的变量综合出来一般是一根导线；
- reg 变量在 always 块中有两种情况：
  1. always 后的敏感表中是（a or b or c）形式的，也就是不带时钟边沿的，综合出来还是组合逻辑；
  2. always 后的敏感表中是（posedge clk）形式的，也就是带边沿的，综合出来一般是时序逻辑，会包含触发器（Flip - Flop）。

2.4 Subtask 4

答：搭配 assign 时，或要变量连续赋值时，reg 不能够当成 wire；否则可以将 reg 当成 wire 用。