**中国科学技术大学计算机学院**

**《计算机组成原理实验》报告**



实验题目：Lab1运算器及其应用

学生姓名：\_\_\_ 吴毅龙\_\_\_\_\_\_

学生学号：\_\_\_PB19111749\_\_

完成日期： 2021/4/7

计算机实验教学中心制

2020年09月

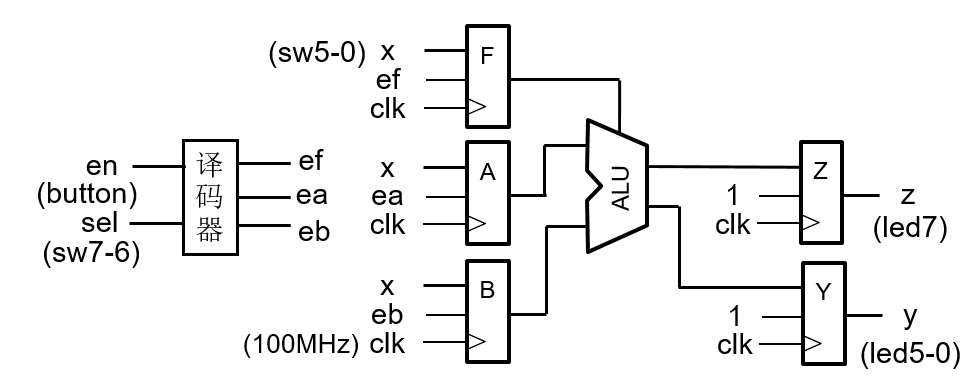
1. 算术逻辑单元（ALU）
2. 逻辑设计

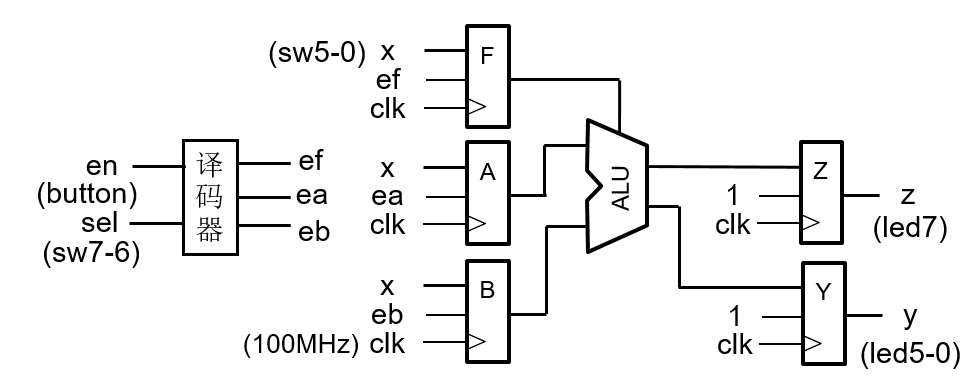
最基本的ALU模块比较容易实现，只需要根据输入的运算模式信号对操作数进行加、减、与、或、异或操作即可，输出信号z为零标志，当运算结果为0时其值为1，因此只需对结果进行一次或非运算，当结果为0时得到1，不为0时得到0。

需要引起注意的地方是实现ALU时，由于硬件的限制需要对操作数、运算模式信号进行分时输入，这里就需要一个2-4译码器模块来区分各个输入。具体而言就是：

sel=00时，ef=1，输入x选择操作类型F：000为加，001为减，010为与，011为或，100为异或

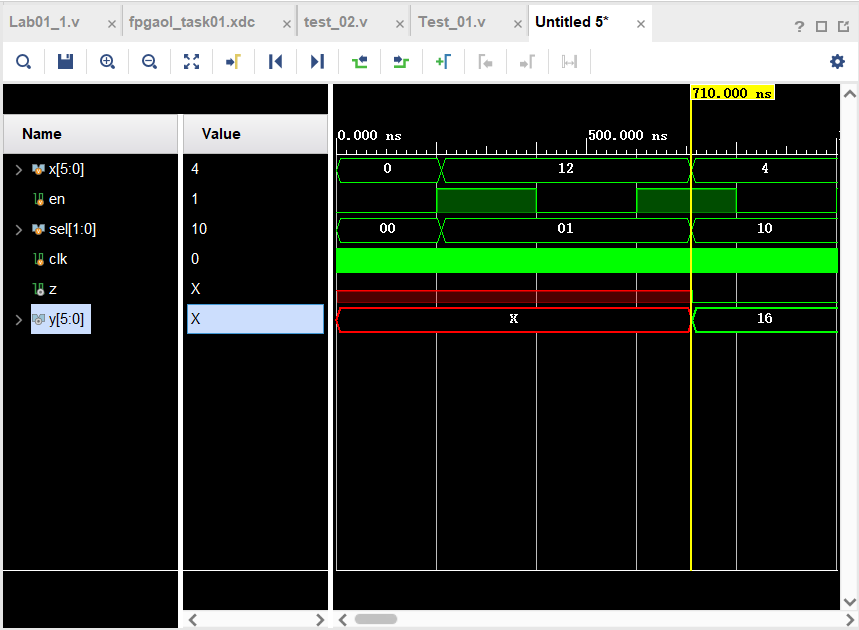
sel=01时，ea=1，输入x赋值给操作数A

sel=10时，eb=1，输入x赋值给操作数B

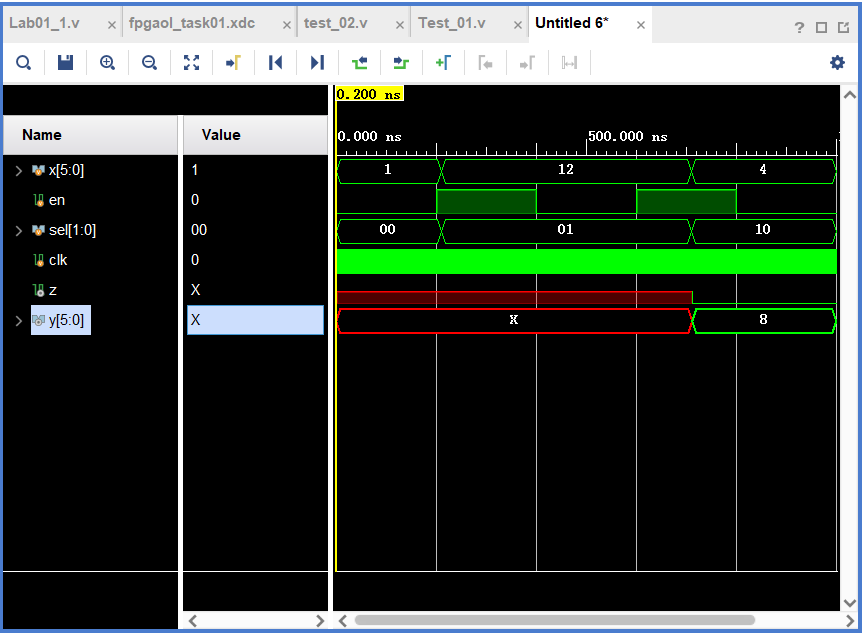


1. 核心代码
2. Decoder Decoder(.code(sel), .out(eout));
3. ALU #(6)ALU(.a(A), .b(B), .f(F), .z(Z), .y(Y));
4. assign ef = eout[0] & en;
5. assign ea = eout[1] & en;
6. assign eb = eout[2] & en;
7. always@(posedge clk)
8. begin
9. **if**(ef) F <= x[2:0];
10. **if**(ea) A <= x;
11. **if**(eb) B <= x;
12. end
13. always@(posedge clk)
14. begin
15. y <= Y;
16. z <= Z;
17. end
18. 仿真结果

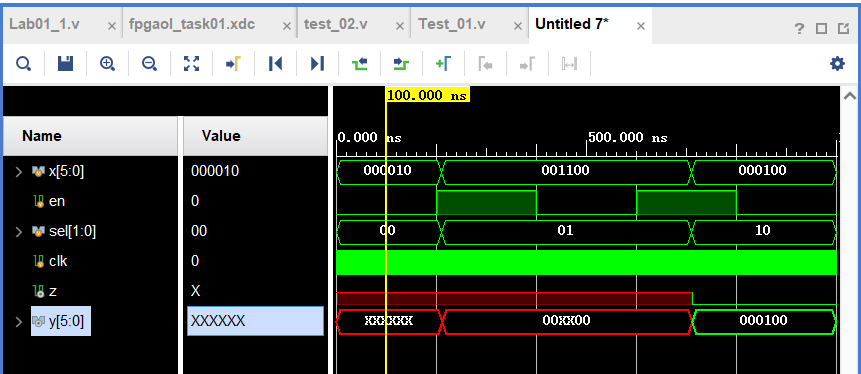
加法运算



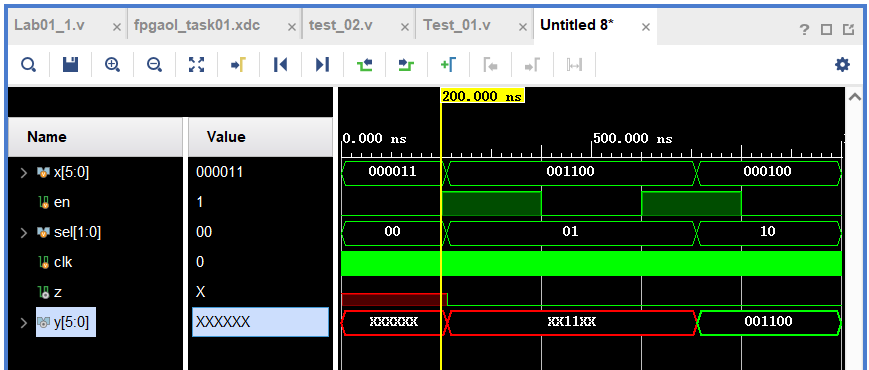
减法运算



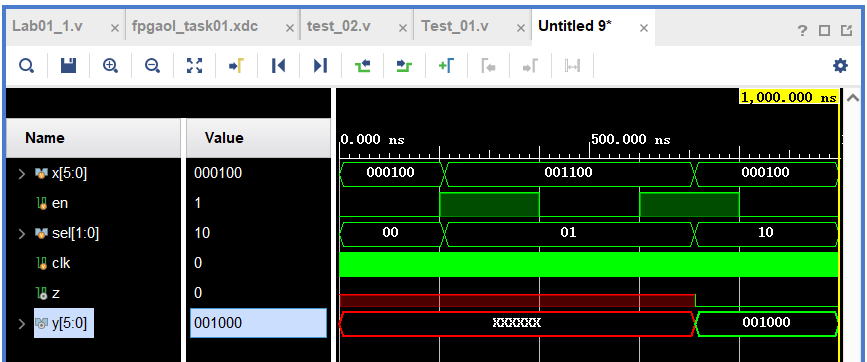
与运算



或运算

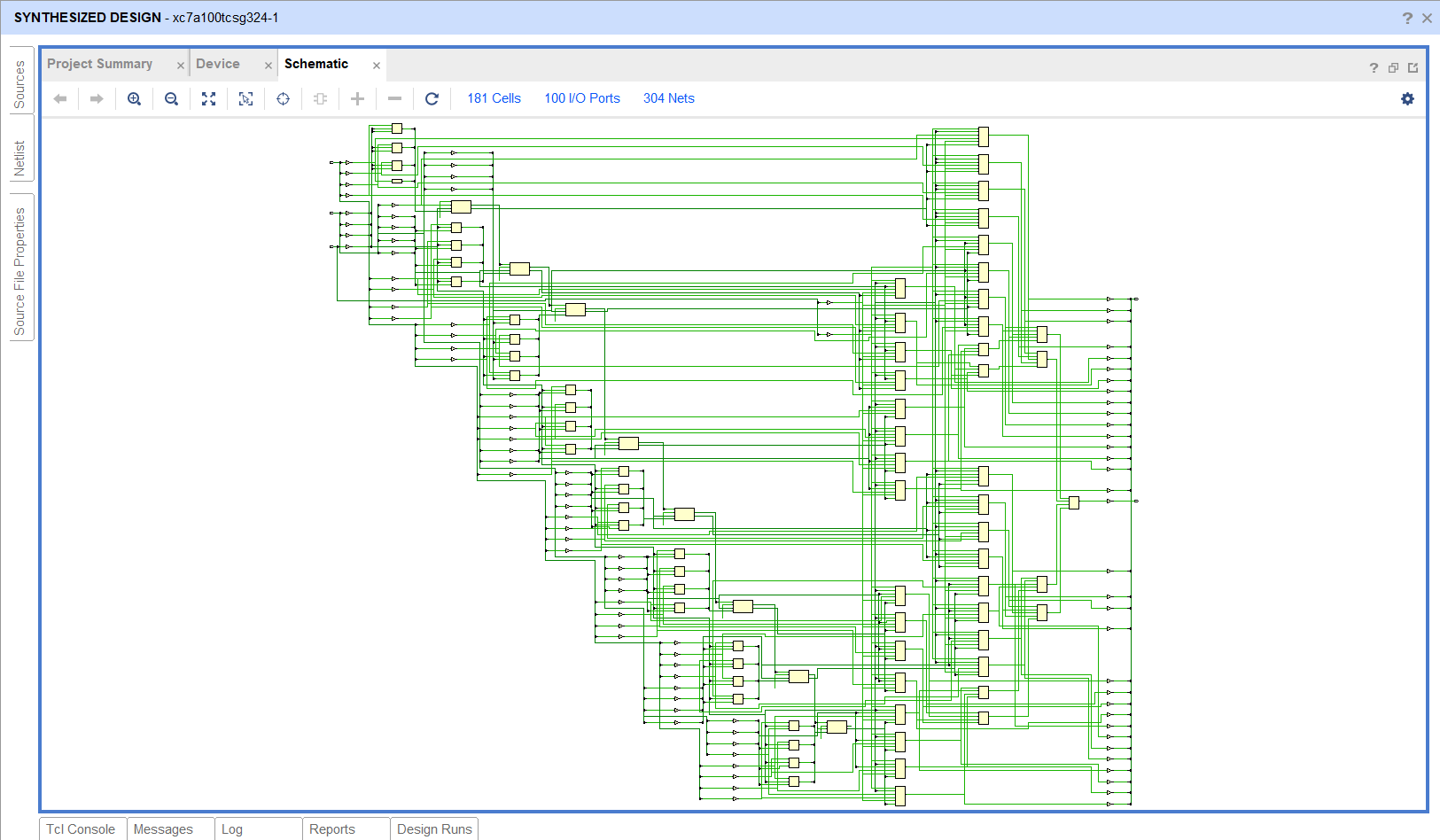


异或运算

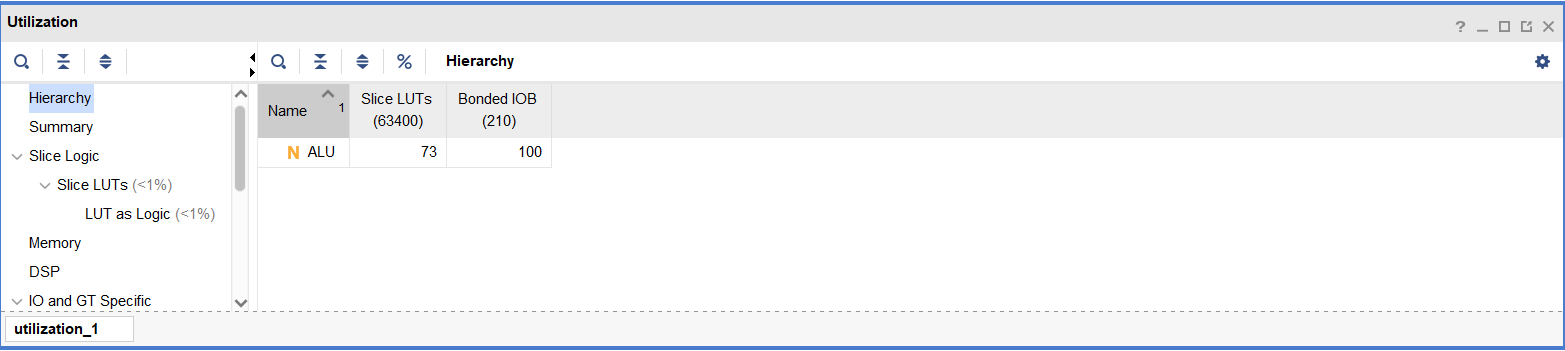


1. 实现后的电路图，电路资源和时间性能报告

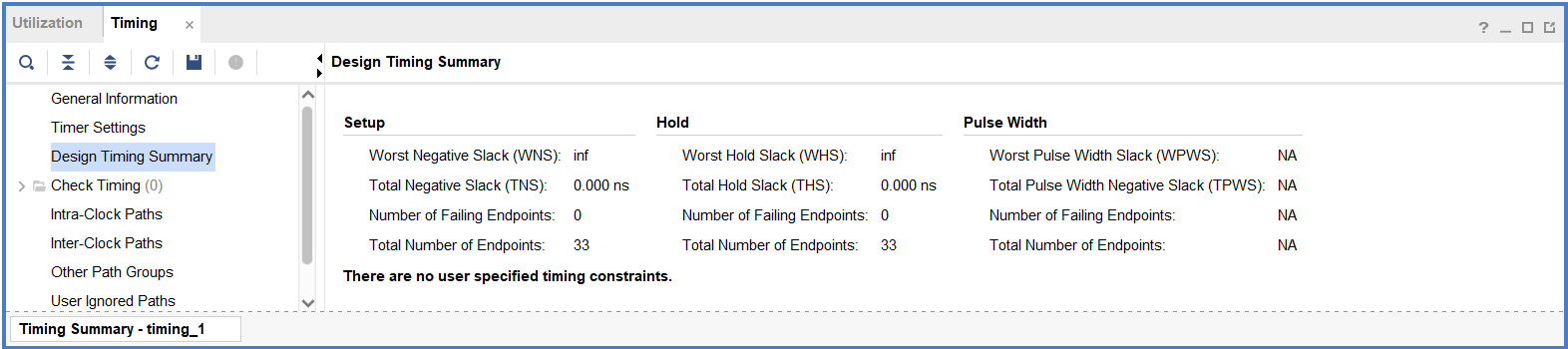
32位ALU综合电路



电路资源使用情况



时间性能报告



1. ALU的应用：计算斐波那契-卢卡斯数列
2. 逻辑设计

该实验主要考察有限状态机的使用和ALU模块的应用。设置7个状态，分别为S0、S1、S2、S3、S4、S5、S6，其中S0、S1、S2、S3为赋值状态，S4、S5、S6为数列计算状态

S0：初始状态，该状态为赋值的初始状态，rst信号有效进入此状态，f=0,ff=0；en信号为0时维持此状态，en信号为1时进入S1；

S1：赋值状态同时防抖动，将输入信号d赋给f和ff，en信号为1时维持此状态，en信号为0时进入S2；

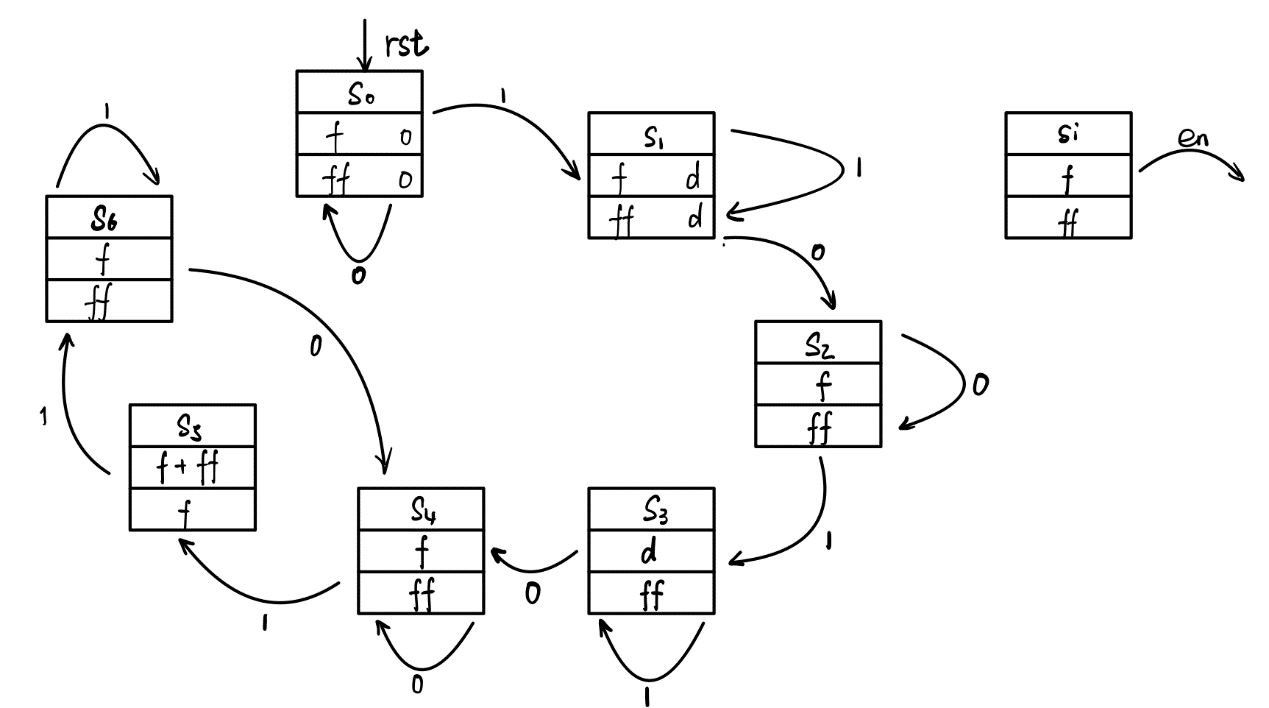
S2：赋值中间状态，en信号为0时维持此状态，en信号为1时进入S3；

S3：赋值状态同时防抖动，将输入信号d赋给f，en信号为1时维持此状态，en信号为0时进入S4；

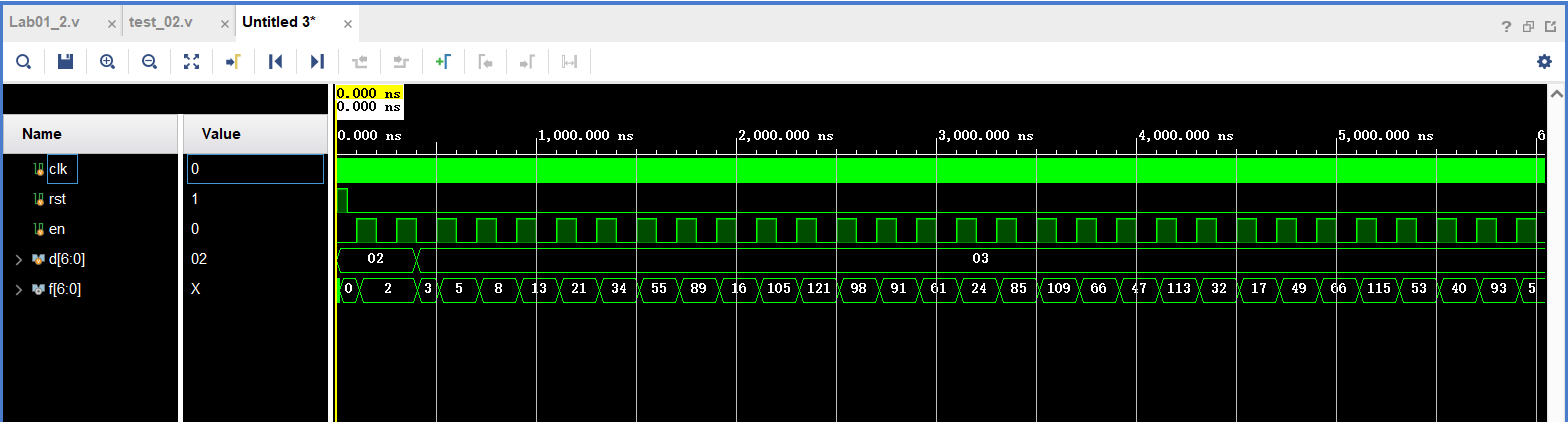
S4：计算中间状态，en信号为0时维持此状态，en信号为1时进入S5；

S5：计算状态，调用ALU模块计算f+ff，将f赋值给ff，进入S6;

S6：计算中间状态，en信号为1时维持此状态，en信号为0时进入S4；

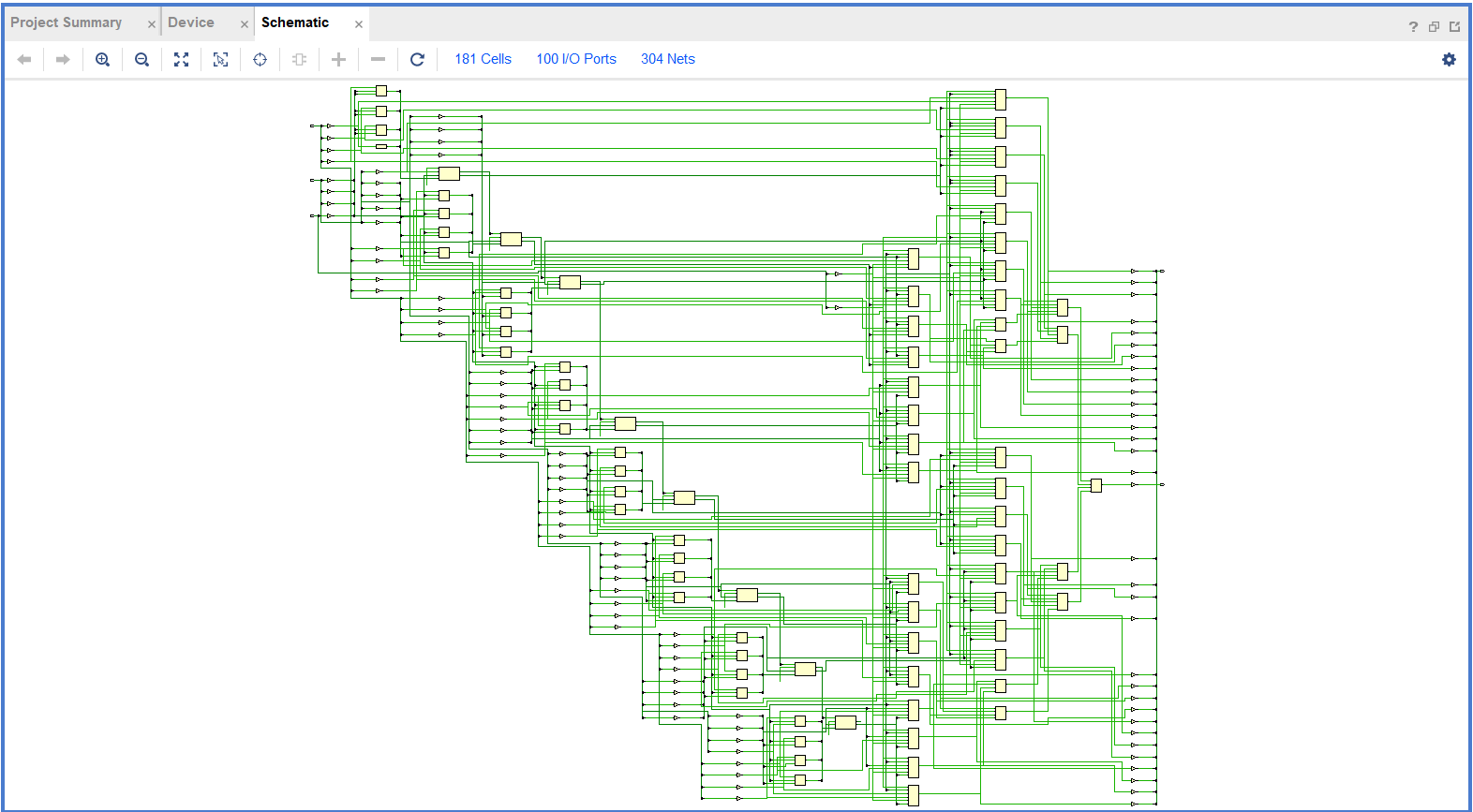


1. 核心代码
2. always @(posedge clk) begin
3. **if**(rst)state <= s0;
4. **else** begin
5. **case** (state)
6. s0: begin
7. **if**(en)state <= s1;
8. **else** state <= s0;
9. end
10. s1: begin
11. **if**(en)state <= s1;
12. **else** state <= s2;
13. end
14. s2: begin
15. **if**(en)state <= s3;
16. **else** state <= s2;
17. end
18. s3: begin
19. **if**(en)state <= s3;
20. **else** state <= s4;
21. end
22. s4: begin
23. **if**(en)state <= s5;
24. **else** state <= s4;
25. end
26. s5: state <= s6;
27. s6: begin
28. **if**(en)state <= s6;
29. **else** state <= s4;
30. end
31. endcase
32. end
33. end
35. always @(posedge clk)
36. begin
37. **case** (state)
38. s0: ff <= 0;
39. s1: ff <= d;
40. s5: ff <= f;
41. **default**: ff <= ff;
42. endcase
43. end
45. always @(posedge clk)
46. begin
47. **case** (state)
48. s0: f <= 0;
49. s1: f <= d;
50. s3: f <= d;
51. s5: f <= temp;
52. **default**: f <= f;
53. endcase
54. end
55. 仿真结果

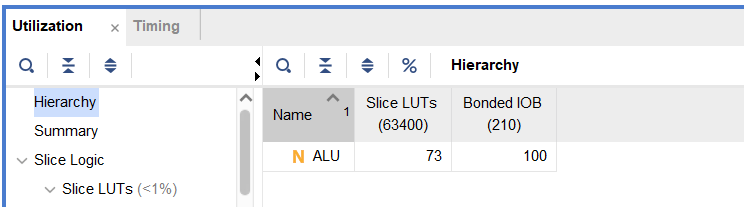


1. 实现后的电路图，电路资源和时间性能报告

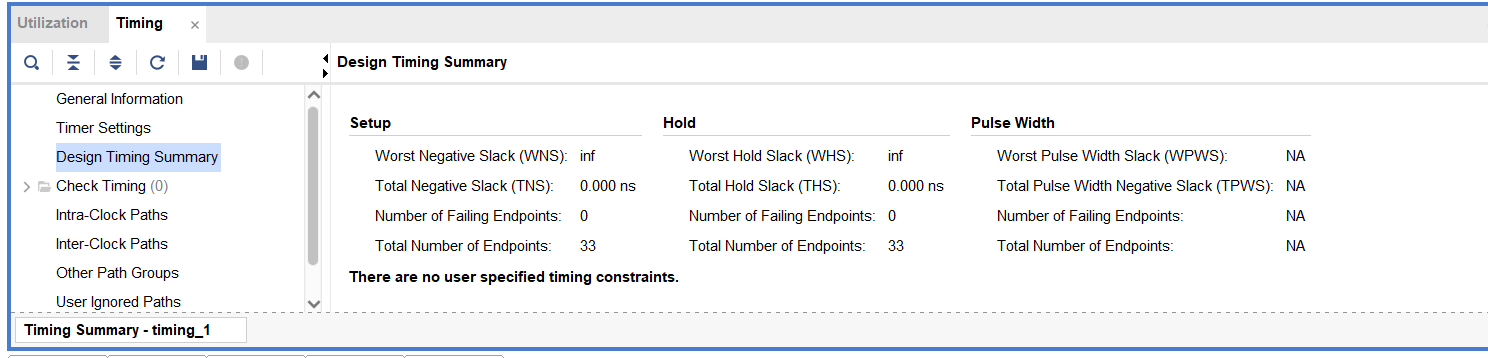
综合电路



电路资源使用情况



综合电路性能



1. 意见和建议

通过本次实验，我对verilog的理解和掌握又更近了一步，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得以运用，加深了我对逻辑电路和时序电路知识的理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。

1. 设计和测试文件源代码

ALU模块

1. module ALU #(parameter WIDTH = 6)
2. (input [WIDTH - 1:0]a, b,
3. input [2:0]f,
4. output reg[WIDTH - 1:0]y,
5. output z);
6. reg[WIDTH:0]result;
7. always@(\*)
8. begin
9. **case**(f)
10. 3'b000:result = {0,a} + {0,b};
11. 3'b001:result = {0,a} - {0,b};
12. 3'b010:result = {0,a} & {0,b};
13. 3'b011:result = {0,a} | {0,b};
14. 3'b100:result = {0,a} ^ {0,b};
15. **default**:result = 0;
16. endcase
17. y[WIDTH - 1:0] = result[WIDTH - 1:0];
18. end
19. assign z = ~| y;
20. endmodule

2-4译码器模块

1. module Decoder
2. (input [1:0]code,
3. output reg[3:0]out);
4. always@(\*)
5. begin
6. **case**(code)
7. 2'b00:out = 4'b0001;
8. 2'b01:out = 4'b0010;
9. 2'b10:out = 4'b0100;
10. 2'b11:out = 4'b1000;
11. endcase
12. end
13. endmodule

任务一

1. module Lab01\_1(
2. input [5:0]x,
3. input en,
4. input [1:0]sel,     //sel=00时，ef=1，输入x选择操作类型：000为加，001为减，
5. 010为与，011为或，100为异或
6. //sel=01时，ea=1，输入x赋值给操作数a
7. //sel=10时，eb=1，输入x赋值给操作数b
8. input clk,
9. output reg z,
10. output reg [5:0]y);
11. wire [3:0] eout;
12. wire ea, eb, ef, Z;
13. wire [5:0]Y;
14. reg [5:0]A, B;
15. reg [2:0]F;
16. Decoder Decoder(.code(sel), .out(eout));
17. ALU #(6)ALU(.a(A), .b(B), .f(F), .z(Z), .y(Y));
18. assign ef = eout[0] & en;
19. assign ea = eout[1] & en;
20. assign eb = eout[2] & en;
21. always@(posedge clk)
22. begin
23. **if**(ef) F <= x[2:0];
24. **if**(ea) A <= x;
25. **if**(eb) B <= x;
26. end
27. always@(posedge clk)
28. begin
29. y <= Y;
30. z <= Z;
31. end
33. endmodule

任务二

1. module Lab01\_2(
2. input clk, rst, en,
3. input [6:0]d,
4. output reg [6:0]f);
5. reg[6:0] ff;
6. parameter s0 = 3'h0, s1 = 3'h1, s2 = 3'h2, s3 = 3'h3, s4 = 3'h4, s5 = 3'h5, s6 = 3'h6;
7. reg[2:0] state;
8. wire[6:0]temp;
9. ALU #(7)ALU(.a(f), .b(ff), .f(3'b000), .y(temp), .z());
10. always @(posedge clk) begin
11. **if**(rst)state <= s0;
12. **else** begin
13. **case** (state)
14. s0: begin
15. **if**(en)state <= s1;
16. **else** state <= s0;
17. end
18. s1: begin
19. **if**(en)state <= s1;
20. **else** state <= s2;
21. end
22. s2: begin
23. **if**(en)state <= s3;
24. **else** state <= s2;
25. end
26. s3: begin
27. **if**(en)state <= s3;
28. **else** state <= s4;
29. end
30. s4: begin
31. **if**(en)state <= s5;
32. **else** state <= s4;
33. end
34. s5: state <= s6;
35. s6: begin
36. **if**(en)state <= s6;
37. **else** state <= s4;
38. end
39. endcase
40. end
41. end
43. always @(posedge clk)
44. begin
45. **case** (state)
46. s0:
47. ff <= 0;
48. s1: ff <= d;
49. s5: ff <= f;
50. **default**: ff <= ff;
51. endcase
52. end
54. always @(posedge clk)
55. begin
56. **case** (state)
57. s0: f <= 0;
58. s1: f <= d;
59. s3: f <= d;
60. s5: f <= temp;
61. **default**: f <= f;
62. endcase
63. end
64. endmodule