COD_LAB1 运算器的实现

黄业琦 PB17000144

March 22, 2019

1 实验目的

1.1 实现运算器

Verilog 实现算术逻辑单元 ALU
s：功能选择。加、减、与、或、非、异或等运算
a, b：两操作数。对于减运算，a 是被减数；对于非运算，操作数是 a
y：运算结果。和、差……
f：标志。进位/借位、溢出、零标志

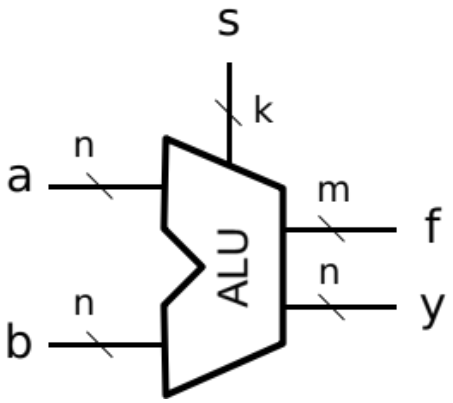

Figure 1: ALU

1.2 Fibonacci 数列

求给定两个初始数的斐波拉契数列（结果从同一端口分时输出）。

1.3 求多个数的累加和

求多个数的累加和（来自同一端口分时输入）。

2 实验环境

Linux 下编程调试和仿真，使用 IVerilog，GtkWave 系列工具。
Windows 下用于生成比特流文件，使用 Vivado 2018.2，Verilog HDL
所有下载均在 Nexsy4-DDR 实验板完成

3 逻辑设计

3.1 ALU 设计

目前 ALU 只实现了 6 个功能：4 个逻辑运算和 2 个算术运算。
对于逻辑运算，不可能产生进位、溢出等问题。
算术运算考虑方法为：溢出位为直接加考虑，进位/借位考虑需要用异或实现，详细见附录代码。

3.2 Fibonacci 设计和累计求和

我们需要额外多加 register 用于保存中间结果（老师毙掉了我想使用 *inout* 接口 + 外设输入的想法，但最后一周我还是会这么玩的）

4 仿真截图

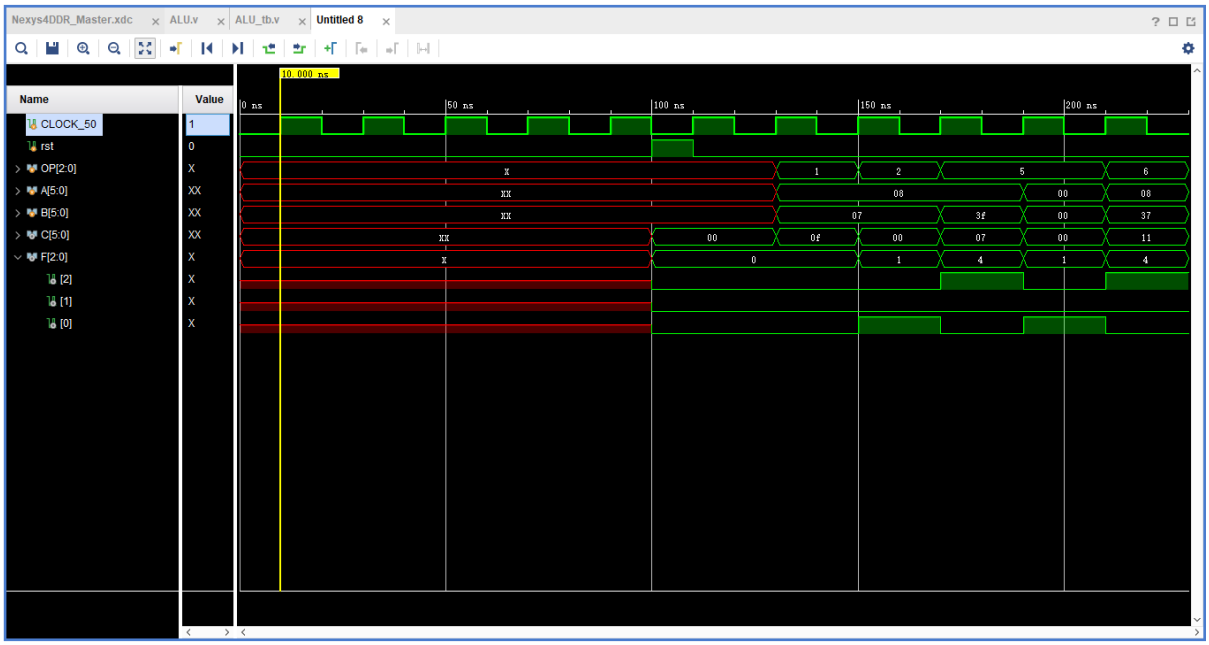

Figure 2: ALU_sim

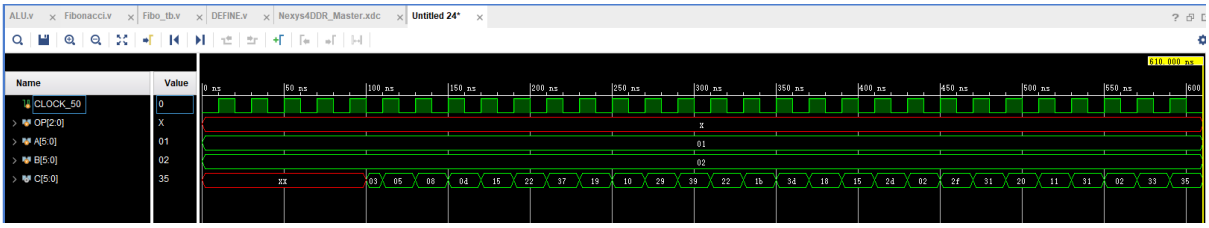

Figure 3: Fibonacci_sim

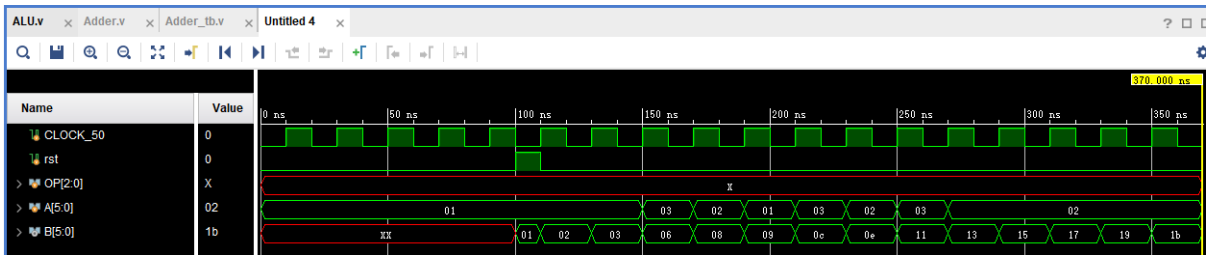

Figure 4: Adder_sim

5 性能评测截图

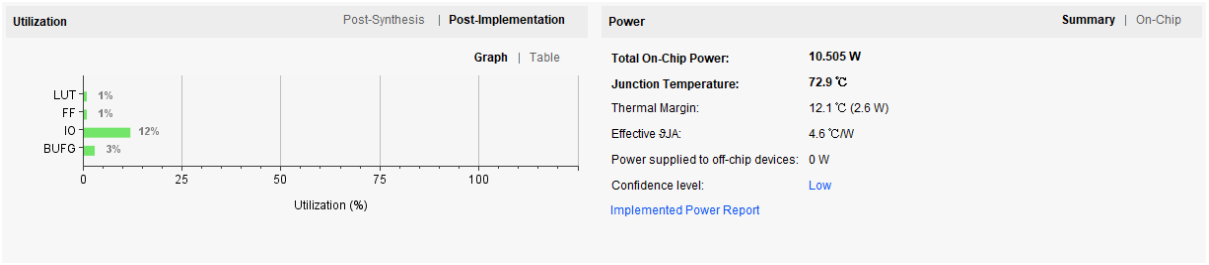

Figure 5: ALU_Perfoemance1

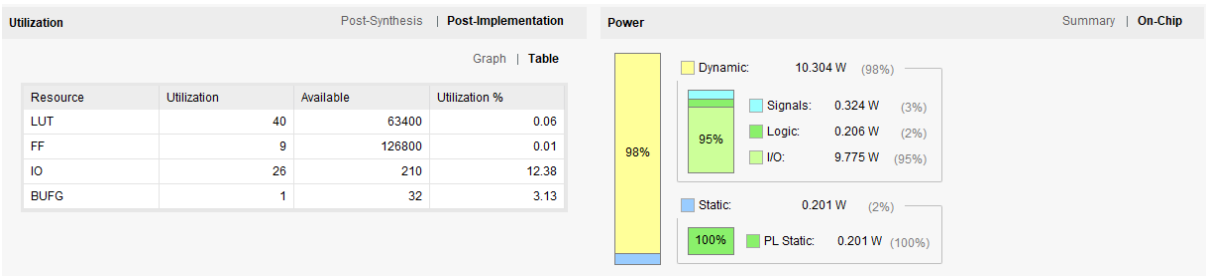

Figure 6: Adder_sim

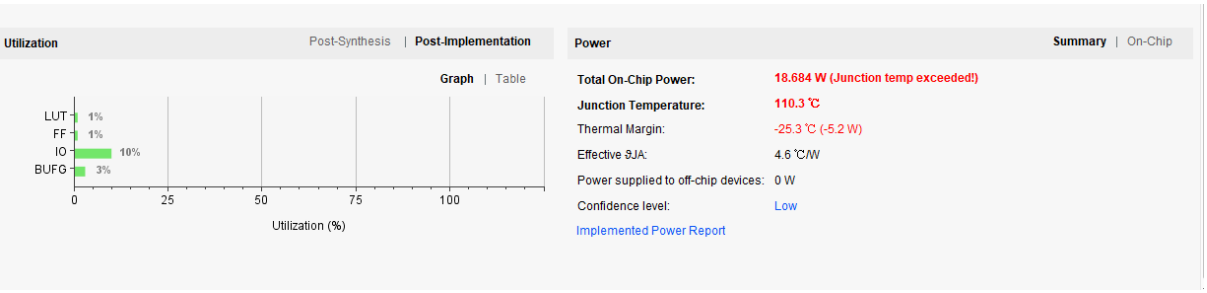

Figure 7: Fibo_Perfoemance1

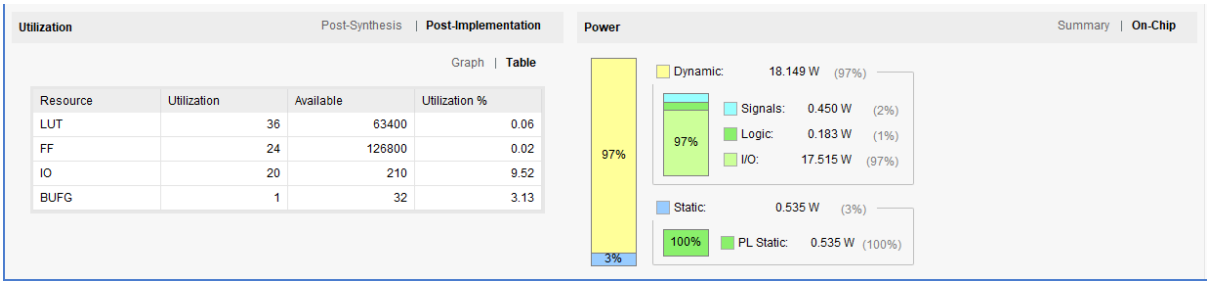

Figure 8: Fibo_Perfoemance2

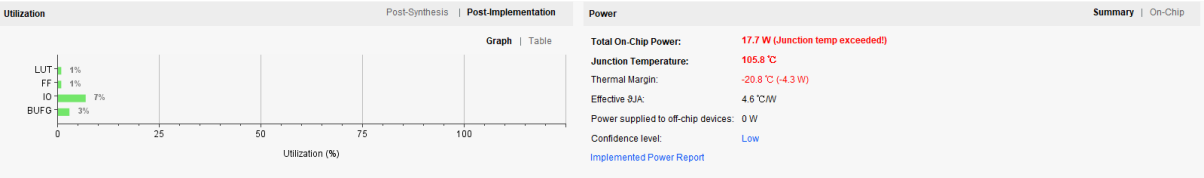

Figure 9: Adder_Perfoemance1

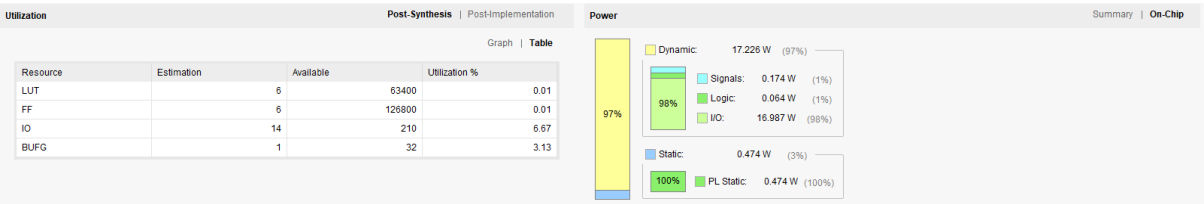

Figure 10: Adder_Perfoemance2

6 实验代码

6.1 ALU 实现代码

Listing 1: DEFINE.v

```
1 `define EXE_ORI 3'b001 //OR
2 `define EXE_AND 3'b010 //AND
3 `define EXE_NOT 3'b011 //NOT
```

```
`define EXE_XOR 3'b100 //XOR

`define EXE_ADD 3'b101 //ADD
`define EXE_SUB 3'b110 //SUB

`define EXE_NOP 3'b000 //EMPTY

`define OPERAND_LIM 5
`define OPERAND_BUS 5:0
`define OPECODE_BUS 2:0
```

Listing 2: ALU.v

```
`timescale 1ns / 1ps

`include "DEFINE.v"

module ALU (
    input  clk,
    input  rst,
    input  [`OPECODE_BUS] OP,
    input  [`OPERAND_BUS] A,
    input  [`OPERAND_BUS] B,
    output reg [`OPERAND_BUS] C,
    output reg [`OPECODE_BUS] F
    );

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            C = 0;
            F = 0;
        end
        else
        begin
            case (OP)
                F = 0;
                `EXE_ORI: begin C = A | B; end
                `EXE_AND: begin C = A & B; end
                `EXE_NOT: begin C = ~ A  ; end
                `EXE_XOR: begin C = A ^ B; end
                `EXE_ADD: begin {F[2],C} = A + B; end
                `EXE_SUB: begin {F[2],C} = A - B; end
            endcase

            F[0] = (C == 0);
            if (OP==`EXE_ADD || OP==`EXE_SUB)
            F[1] = A[`OPERAND_LIM] ^ B[`OPERAND_LIM] ^ C[`OPERAND_LIM] ^ F[2];
            else
            F[1] = 0;
        end
    end
endmodule
```

Listing 3: ALU_tb.v

```
`timescale 1ns / 1ps

`include "DEFINE.v"

module ALU_tb (
    );

   reg CLOCK_50;
   reg rst;

    reg  [`OPECODE_BUS] OP;
    reg  [`OPERAND_BUS] A;
    reg  [`OPERAND_BUS] B;
    wire  [`OPERAND_BUS] C;
    wire  [`OPECODE_BUS] F;

   initial begin
    CLOCK_50 = 1'b0;
    forever #10 CLOCK_50 = ~CLOCK_50;
   end

   initial begin
    rst = 0;
    #100 rst = 1;
    #10  rst = 0;
    #20 OP = `EXE_ORI; A = 6'b01000; B = 6'b00111;
    #20 OP = `EXE_AND; A = 6'b01000; B = 6'b00111;
    #20 OP = `EXE_ADD; A = 6'b01000; B = 6'b111111;
    #20 OP = `EXE_ADD; A = 6'b000000; B = 6'b000000;
    #20 OP = `EXE_SUB; A = 6'b01000; B = 6'b110111;
    #20 $stop;
   end

   ALU aluer(
    .clk(CLOCK_50),
    .rst(rst),
    .OP(OP),
    .A(A),
    .B(B),
    .C(C),
    .F(F)
   );
endmodule
```

6.2 Fibonacci 实现代码

Listing 4: DEFINE.v

```
`define EXE_ORI 3'b001 //OR
`define EXE_AND 3'b010 //AND
`define EXE_NOT 3'b011 //NOT
`define EXE_XOR 3'b100 //XOR

`define EXE_ADD 3'b101 //ADD
`define EXE_SUB 3'b110 //SUB

`define EXE_NOP 3'b000 //EMPTY

`define OPERAND_LIM 5
`define OPERAND_BUS 5:0
`define OPECODE_BUS 2:0
```

Listing 5: ALU.v

```
`timescale 1ns / 1ps

`include "DEFINE.v"

module ALU (
    input  [`OPECODE_BUS] OP,
    input  [`OPERAND_BUS] A,
    input  [`OPERAND_BUS] B,
    output reg [`OPERAND_BUS] C
    );

    always @* begin
            case (OP)
                `EXE_ORI: begin C <= A | B;  end
                `EXE_AND: begin C <= A & B;  end
                `EXE_NOT: begin C <= ~ A  ;  end
                `EXE_XOR: begin C <= A ^ B;  end
                `EXE_ADD: begin C <= A + B; end
                `EXE_SUB: begin C <= A - B; end
            endcase
    end
endmodule
```

Listing 6: Fibonacci.v

```
`timescale 1ns / 1ps

module Fibonacci (
    input clk,
    input rst,
    input [`OPERAND_BUS] A,
    input [`OPERAND_BUS] B,
    output reg [`OPERAND_BUS] C
```

```
    );
    reg [`OPERAND_BUS] TMP1,TMP2;
    reg [`OPECODE_BUS] ADD;
    reg [`OPERAND_BUS] AT;
    reg [`OPERAND_BUS] BT;
    wire [`OPERAND_BUS] CT;


    always @* AT <= TMP1;
    always @* BT <= TMP2;
    initial ADD = `EXE_ADD;

    ALU ALUER(ADD,AT,BT,CT);
    always @(posedge clk or posedge rst) begin
       if (rst) begin
              TMP1 <= A;
              TMP2 <= B;
           end
           else
           begin
              TMP1 <= TMP2;
              TMP2 <= CT;
           end
    end

    always @*
    C <= CT;
endmodule
```

Listing 7: Fibo_tb.v

```
    `timescale 1ns / 1ps

`include "DEFINE.v"

module Fibo_tb (
    );

   reg CLOCK_50;
   reg rst;

    reg  [`OPECODE_BUS] OP;
    reg  [`OPERAND_BUS] A;
    reg  [`OPERAND_BUS] B;
    wire  [`OPERAND_BUS] C;

   initial begin
    CLOCK_50 = 1'b0;
    forever #10 CLOCK_50 = ~CLOCK_50;
   end

   initial begin
```

```
22          A = 6'b01;
23          B = 6'b10;
24       rst = 0;
25       #100 rst = 1;
26       #10  rst = 0;
27
28       #500 $stop;
29      end
30
31      Fibonacci F(
32       .clk(CLOCK_50),
33       .rst(rst),
34       .A(A),
35       .B(B),
36       .C(C)
37      );
38  endmodule
```

6.3 累加器实现代码

Listing 8: DEFINE.v

```
`define EXE_ORI 3'b001 //OR
`define EXE_AND 3'b010 //AND
`define EXE_NOT 3'b011 //NOT
`define EXE_XOR 3'b100 //XOR

`define EXE_ADD 3'b101 //ADD
`define EXE_SUB 3'b110 //SUB

`define EXE_NOP 3'b000 //EMPTY

`define OPERAND_LIM 5
`define OPERAND_BUS 5:0
`define OPECODE_BUS 2:0
```

Listing 9: ALU.v

```
`timescale 1ns / 1ps

`include "DEFINE.v"

module ALU (
    input  [`OPECODE_BUS] OP,
    input  [`OPERAND_BUS] A,
    input  [`OPERAND_BUS] B,
    output reg [`OPERAND_BUS] C
    );

    always @* begin
            case (OP)
                `EXE_ORI: begin C <= A | B;  end
                `EXE_AND: begin C <= A & B;  end
                `EXE_NOT: begin C <= ~ A  ;  end
                `EXE_XOR: begin C <= A ^ B;  end
                `EXE_ADD: begin C <= A + B; end
                `EXE_SUB: begin C <= A - B; end
            endcase
    end
endmodule
```

Listing 10: Adder.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2019/03/23 21:47:04
// Design Name:
// Module Name: Adder
```

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Adder(
    input clk,
    input rst,
    input [5:0] A,
    output reg [5:0] B
    );

    reg [`OPERAND_BUS] TMP;
    reg [`OPECODE_BUS] ADD;
    reg [`OPERAND_BUS] T;
    wire [`OPERAND_BUS] CT;

    always @* T <= TMP;
    initial ADD = `EXE_ADD;

    ALU ALUER(ADD,A,TMP,CT);
    always @(posedge clk or posedge rst) begin
       if (rst) begin
               TMP = 0;
            end
            else
            begin
               TMP = CT;
            end
    end

    always @*
    B <= CT;
endmodule
```

Listing 11: Adder_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2019/03/23 21:53:28
```

```
7  // Design Name:
8  // Module Name: Adder_tb
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22 `include "DEFINE.v"
23
24 module Adder_tb (
25     );
26
27    reg CLOCK_50;
28    reg rst;
29
30     reg  [`OPECODE_BUS] OP;
31     reg  [`OPERAND_BUS] A;
32     wire  [`OPERAND_BUS] B;
33
34    initial begin
35     CLOCK_50 = 1'b0;
36     forever #10 CLOCK_50 = ~CLOCK_50;
37    end
38
39    initial begin
40     A = 6'b01;
41
42     rst = 0;
43     #100 rst = 1;
44     #10  rst = 0;
45     #20  A = 6'b01;
46     #20  A = 6'b11;
47     #20  A = 6'b10;
48     #20  A = 6'b01;
49     #20  A = 6'b11;
50     #20  A = 6'b10;
51     #20  A = 6'b11;
52     #20  A = 6'b10;
53     #100 $stop;
54    end
55
56    Adder Add(
57     .clk(CLOCK_50),
```

```
58      .rst(rst),
59      .A(A),
60      .B(B)
61    );
62 endmodule
```

7 实验总结

本次试验总体算比较顺利的，中途有一处问题，再 Fibonacci 数列试验中，直接用了前一个实验的 ALU 部件，倒置 clk 和 rst 没有删去，倒置在加法时多跑了一个时间周期，出现了问题，以后需要注意。最终是重写了简单版的 ALU 模块替换解决了问题。