数字电路实验 Lab 6 实验报告

奚项正 PB23000020

必做内容

题目 1：加法器（2 分）

设计

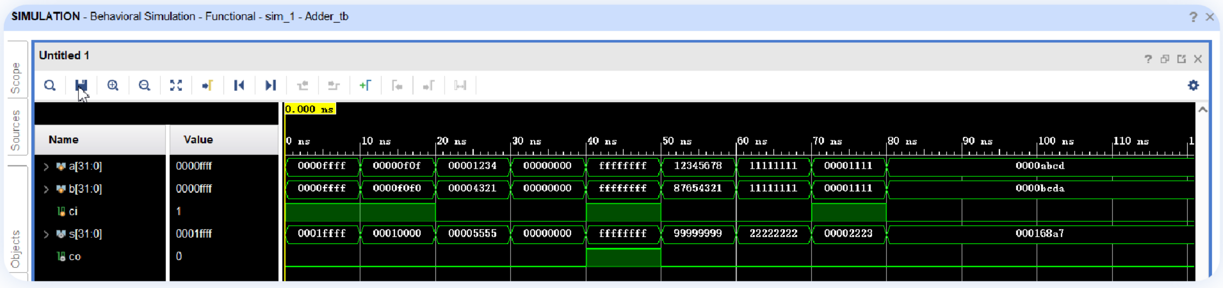
```
module Adder_LookAhead8 (
    input                   [ 7 : 0]            a, b,
    input                   [ 0 : 0]            ci,         // 来自低位的进位
    output                  [ 7 : 0]            s,          // 和
    output                  [ 0 : 0]            co          // 向高位的进位
);

wire    [7:0] C;
wire    [7:0] G;
wire    [7:0] P;

assign  G = a & b;
assign  P = a ^ b;

// 产生逻辑：C_{i}=A_iB_i+(A_i\oplus B_i)C_{i-1}=G_i+P_iC_{i-1}
assign  C[0] = G[0] | ( P[0] & ci );
assign  C[1] = G[1] | ( P[1] & G[0] ) | ( P[1] & P[0] & ci );
assign  C[2] = G[2] | ( P[2] & G[1] ) | ( P[2] & P[1] & G[0] ) | ( P[2] & P[1] & P[0] & ci );
assign  C[3] = G[3] | ( P[3] & G[2] ) | ( P[3] & P[2] & G[1] ) | ( P[3] & P[2] & P[1] & G[0]
) | ( P[3] & P[2] & P[1] & P[0] & ci );
assign  C[4] = G[4] | ( P[4] & G[3] ) | ( P[4] & P[3] & G[2] ) | ( P[4] & P[3] & P[2] & G[1]
) | ( P[4] & P[3] & P[2] & P[1] & G[0] ) | ( P[4] & P[3] & P[2] & P[1] & P[0] & ci );
assign  C[5] = G[5] | ( P[5] & G[4] ) | ( P[5] & P[4] & G[3] ) | ( P[5] & P[4] & P[3] & G[2]
) | ( P[5] & P[4] & P[3] & P[2] & G[1] ) | ( P[5] & P[4] & P[3] & P[2] & P[1] & G[0] ) | (
P[5] & P[4] & P[3] & P[2] & P[1] & P[0] & ci );
assign  C[6] = G[6] | ( P[6] & G[5] ) | ( P[6] & P[5] & G[4] ) | ( P[6] & P[5] & P[4] & G[3]
) | ( P[6] & P[5] & P[4] & P[3] & G[2] ) | ( P[6] & P[5] & P[4] & P[3] & P[2] & G[1] ) | (
P[6] & P[5] & P[4] & P[3] & P[2] & P[1] & G[0] ) | ( P[6] & P[5] & P[4] & P[3] & P[2] & P[1]
& P[0] & ci );
```

```verilog
assign  C[7] = G[7] | ( P[7] & G[6] ) | ( P[7] & P[6] & G[5] ) | ( P[7] & P[6] & P[5] & G[4]
) | ( P[7] & P[6] & P[5] & P[4] & G[3] ) | ( P[7] & P[6] & P[5] & P[4] & P[3] & G[2] ) | (
P[7] & P[6] & P[5] & P[4] & P[3] & P[2] & G[1] ) | ( P[7] & P[6] & P[5] & P[4] & P[3] & P[2]
& P[1] & G[0] ) | ( P[7] & P[6] & P[5] & P[4] & P[3] & P[2] & P[1] & P[0] & ci );

// TODO: 确定 s 和 co 的产生逻辑
// 即S_i=A_i\oplus B_i\oplus C_{i-1}=P_i\oplus C_{i-1}
assign  s[0] = P[0] ^ ci;
assign  s[1] = P[1] ^ C[0];
assign  s[2] = P[2] ^ C[1];
assign  s[3] = P[3] ^ C[2];
assign  s[4] = P[4] ^ C[3];
assign  s[5] = P[5] ^ C[4];
assign  s[6] = P[6] ^ C[5];
assign  s[7] = P[7] ^ C[6];
assign  co   = C[7];

endmodule
```

```verilog
module Adder (
    input                   [31 : 0]        a, b,
    input                   [ 0 : 0]        ci,
    output                  [31 : 0]        s,
    output                  [ 0 : 0]        co
);
wire    [2:0] cmid;

// TODO: 继续例化 Adder_LookAhead8 模块并正确连接
Adder_LookAhead8 adder0(
    .a(a[7:0]),
    .b(b[7:0]),
    .ci(ci),
    .s(s[7:0]),
    .co(cmid[0])
);

Adder_LookAhead8 adder1(
    .a(a[15:8]),
    .b(b[15:8]),
    .ci(cmid[0]),
    .s(s[15:8]),
    .co(cmid[1])
);

Adder_LookAhead8 adder2(
    .a(a[23:16]),
    .b(b[23:16]),
    .ci(cmid[1]),
    .s(s[23:16]),
    .co(cmid[2])
);

Adder_LookAhead8 adder3(
    .a(a[31:24]),
    .b(b[31:24]),
    .ci(cmid[2]),
    .s(s[31:24]),
    .co(co)
);

endmodule
```

仿真

```
module Adder_tb();
reg     [31:0]  a,b;
reg             ci;
wire    [31:0]  s;
wire            co;

initial begin
    a=32'hffff; b=32'hffff; ci=1'b1;
    #10;
    a=32'h0f0f; b=32'hf0f0; ci=1'b1;
    #10;
    a=32'h1234; b=32'h4321; ci=1'b0;
    #10;
    a=32'h0000; b=32'h0000; ci=1'b0;
    #10;
    a=32'hffffffff; b=32'hffffffff; ci=1'b1;
    #10;
    a=32'h12345678; b=32'h87654321; ci=1'b0;
    #10;
    a=32'h11111111; b=32'h11111111; ci=1'b0;
    #10;
    a=32'h1111; b=32'h1111; ci=1'b1;
    #10;
    a=32'habcd; b=32'hbcda; ci=1'b0;
end

Adder adder(
    .a(a),
    .b(b),
    .ci(ci),
    .s(s),
    .co(co)
);

endmodule
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - Adder_tb
Untitled 1
Scope
Sources
Objects
0.000 ns
Name
Value
a[31:0] 0000ffff
b[31:0] 0000ffff
ci 1
s[31:0] 0001ffff
co 0
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
0000ffff 00000f0f 00001234 00000000 ffffffff 12345678 11111111 00001111 0000abcd
0000ffff 0000f0f0 00004321 00000000 ffffffff 87654321 11111111 00001111 0000bcda
0001ffff 00010000 00005555 00000000 ffffffff 99999999 22222222 00002223 000168a7

题目 2：ALU（5 分）

设计

```
module ALU(
    input                   [31 : 0]        src0, src1,
    input                   [11 : 0]        sel,
    output                  [31 : 0]        res
);
// Write your code here
wire [31:0] adder_out;
wire [31:0] sub_out;
wire [0 :0] slt_out;
wire [0 :0] sltu_out;
wire [31:0] and_out;
wire [31:0] or_out;
wire [31:0] nor_out;
wire [31:0] xor_out;
wire [31:0] sll_out;
wire [31:0] srl_out;
wire [31:0] sra_out;
wire [31:0] src1_out;

Adder adder(
    .a(src0),
    .b(src1),
    .ci(0),
    .s(adder_out),
    .co()
);

Sub sub(
    .a(src0),
    .b(src1),
    .out(sub_out),
    .co()
);

Comp comp(
    .a(src0),
    .b(src1),
```

```verilog
    .ul(sltu_out),
    .sl(slt_out)
);

// 其他的运算使用 Verilog 自带的运算符实现
assign and_out = src0 & src1;
assign or_out = src0 | src1;
assign nor_out = ~(src0 | src1);
assign xor_out = src0 ^ src1;
assign sll_out = src0 << src1[4:0];
assign srl_out = src0 >> src1[4:0];
assign sra_out = src0 >>> src1[4:0];
assign src1_out = src1;

// TODO: 完成 res 信号的选择
// 若sel[i] = 1, 则{32{sel[i]}} & xxx_out = 32'hffffffff & xxx_out = xxx_out
// 若sel[i] = 0, 则{32{sel[i]}} & xxx_out = 32'h00000000 & xxx_out = 0
assign res = ({32{sel[0]}} & adder_out) |
             ({32{sel[1]}} & sub_out) |
             ({32{sel[2]}} & slt_out) |
             ({32{sel[3]}} & sltu_out) |
             ({32{sel[4]}} & and_out) |
             ({32{sel[5]}} & or_out) |
             ({32{sel[6]}} & nor_out) |
             ({32{sel[7]}} & xor_out) |
             ({32{sel[8]}} & sll_out) |
             ({32{sel[9]}} & srl_out) |
             ({32{sel[10]}} & sra_out) |
             ({32{sel[11]}} & src1_out);

endmodule
```

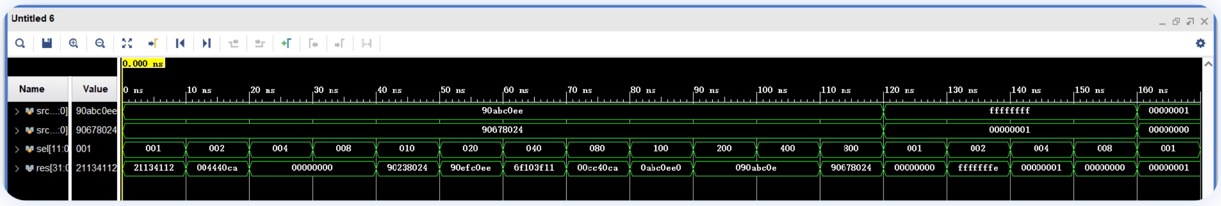
```verilog
module Sub(
    input                   [31 : 0]        a, b,
    output                  [31 : 0]        out,
    output                  [ 0 : 0]        co
);

// 使用加法器实现减法, 即a+(-b)=a+(~b)+1
Adder adder_sub(
    .a(a),
    .b(~b),
    .ci(1),
    .s(out),
    .co(co)
);
endmodule
```

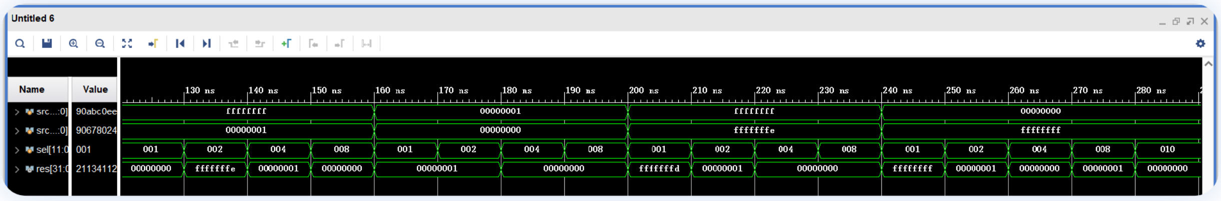
```verilog
module Comp(
    input                   [31 : 0]        a, b,
    output                  [ 0 : 0]        ul,
    output                  [ 0 : 0]        sl
);

wire flag;  //存储a-b的溢出信号
assign ul = ~flag;  //无符号比较时，溢出信号为0等价于a<b
assign sl = (a[31] == b[31]) ? ~flag : flag;    //有符号比较时，同号则溢出信号为0等价于a<b，异号则溢出信号为1等价于a<b

Sub sub_comp(
    .a(a),
    .b(b),
    .out(),
    .co(flag)
);

endmodule
```

> 注：Adder 模块代码同上题

仿真

```
module ALU_tb();
reg     [31:0]  src0;
reg     [31:0]  src1;
reg     [11:0]  sel;
wire    [31:0]  res;

initial begin
    src0=32'h90abc0ee; src1=32'h90678024; sel=12'h001;
    repeat(12) begin
        #10 sel = sel << 1;
    end
    src0=32'hffffffff; src1=32'h1; sel=12'h001;
    repeat(4) begin
        #10 sel = sel << 1;
    end
    src0=32'h1; src1=32'h0; sel=12'h001;
    repeat(4) begin
        #10 sel = sel << 1;
    end
    src0=32'hffffffff; src1=32'hfffffffe; sel=12'h001;
    repeat(4) begin
        #10 sel = sel << 1;
    end
    src0=32'h0; src1=32'hffffffff; sel=12'h001;
    repeat(4) begin
        #10 sel = sel << 1;
    end
end

ALU alu(
    .src0(src0),
        .src1(src1),
        .sel(sel),
        .res(res)
    );

    endmodule
```





选择性必做内容

题目 2：可视化 ALU-plus（3 分）

设计

```
module Adder (       //将上上题中的 8 位超前进位加法器修改为 5 位
    input               [ 4 : 0]        a, b,
    input               [ 0 : 0]        ci,
    output              [ 4 : 0]        s,
    output              [ 0 : 0]        co
);
wire    [4:0] C;
wire    [4:0] G;
wire    [4:0] P;

assign  G = a & b;
assign  P = a ^ b;

assign  C[0] = G[0] | ( P[0] & ci );
assign  C[1] = G[1] | ( P[1] & G[0] ) | ( P[1] & P[0] & ci );
assign  C[2] = G[2] | ( P[2] & G[1] ) | ( P[2] & P[1] & G[0] ) | ( P[2] & P[1] & P[0] & ci );
assign  C[3] = G[3] | ( P[3] & G[2] ) | ( P[3] & P[2] & G[1] ) | ( P[3] & P[2] & P[1] & G[0]
) | ( P[3] & P[2] & P[1] & P[0] & ci );
assign  C[4] = G[4] | ( P[4] & G[3] ) | ( P[4] & P[3] & G[2] ) | ( P[4] & P[3] & P[2] & G[1]
) | ( P[4] & P[3] & P[2] & P[1] & G[0] ) | ( P[4] & P[3] & P[2] & P[1] & P[0] & ci );
```

```
assign  s[0] = P[0] ^ ci;
assign  s[1] = P[1] ^ C[0];
assign  s[2] = P[2] ^ C[1];
assign  s[3] = P[3] ^ C[2];
assign  s[4] = P[4] ^ C[3];
assign  co   = C[4];

endmodule
```

```
module Segment(         //Lab3 中的七段数码管显示模块
    input                       clk,
    input                       rst,
    input       [31:0]          output_data,
    input       [ 7:0]          output_valid,
    output reg  [ 3:0]          seg_data,
    output reg  [ 2:0]          seg_an
);

reg [31:0]  counter;
reg [2:0]   seg_id;

always @(posedge clk) begin
    if (rst) begin
        counter <= 0;
    end
    else if (counter >= 250000) begin
        counter <= 0;
    end
    else begin
        counter <= counter + 1;
    end
end

always @(posedge clk) begin
    if (rst) begin
        seg_id <= 0;
    end
    else if (counter == 1) begin
        if (seg_id >= 8) begin
            seg_id <= 0;
        end
        else begin
            seg_id <= seg_id + 1;
        end
    end
    else begin
        seg_id <= seg_id;
    end
end

always @(*) begin
    seg_data = 0;
    if (output_valid[seg_id] == 1) begin
```

```
        seg_an = seg_id;
    end
    else begin
        seg_an = 0;
    end

    case (seg_an)
        0: seg_data = output_data[3:0];
        1: seg_data = output_data[7:4];
        2: seg_data = output_data[11:8];
        3: seg_data = output_data[15:12];
        4: seg_data = output_data[19:16];
        5: seg_data = output_data[23:20];
        6: seg_data = output_data[27:24];
        7: seg_data = output_data[31:28];
        default: seg_data = 0;
    endcase
end
endmodule
```

```
module Top(
    input                               clk,        //时钟信号
    input                               rst,        //复位信号
    input                               enable,     //使能信号

    input                   [ 4 : 0]    in,         //输入信号
    input                   [ 1 : 0]    ctrl,       //控制信号

    output                  [ 3 : 0]    seg_data,
    output                  [ 2 : 0]    seg_an
);

reg     [3:0]   decoder1_temp;      //译码器1的输出，将2位的ctrl控制信号转为4位
reg     [11:0]  decoder2_temp;      //译码器2的输出，将4位的sel_reg运算选择信号转为12位
wire    [4:0]   alu_temp;           //ALU的输出
wire            sel_en;             //更新运算选择的使能信号
wire            src0_en;            //更新数据源src0的使能信号
wire            src1_en;            //更新数据源src1的使能信号
wire            out_en;             //更新数码管输出的使能信号
reg     [3:0]   sel_reg;            //以下四个信号与上述使能信号一一对应
reg     [4:0]   src0_reg;
reg     [4:0]   src1_reg;
reg     [4:0]   out_reg;

assign sel_en = decoder1_temp[0];   //译码器1的各位输出直接对应上述使能信号
assign src0_en = decoder1_temp[1];
assign src1_en = decoder1_temp[2];
assign out_en = decoder1_temp[3];

always @(*) begin       //译码器1，将2位的ctrl控制信号转为4位
    if (rst) begin
        decoder1_temp = 4'b0000;
    end
    else begin
        if (enable) begin
            case (ctrl)
                2'b00:      decoder1_temp = 4'b0001;
                2'b01:      decoder1_temp = 4'b0010;
                2'b10:      decoder1_temp = 4'b0100;
                2'b11:      decoder1_temp = 4'b1000;
            endcase
        end
        else begin
            decoder1_temp = 4'b0000;
        end
    end
end

always @(*) begin       //译码器2，将4位的sel_reg运算选择信号转为12位
    if (rst) begin
        decoder2_temp = 12'b0000_0000_0000;
    end
    else begin
        case (sel_reg)
            4'b0001:      decoder2_temp = 12'b0000_0000_0001;
            4'b0010:      decoder2_temp = 12'b0000_0000_0010;
            4'b0011:      decoder2_temp = 12'b0000_0000_0100;
            4'b0100:      decoder2_temp = 12'b0000_0000_1000;
            4'b0101:      decoder2_temp = 12'b0000_0001_0000;
            4'b0110:      decoder2_temp = 12'b0000_0010_0000;
            4'b0111:      decoder2_temp = 12'b0000_0100_0000;
            4'b1000:      decoder2_temp = 12'b0000_1000_0000;
            4'b1001:      decoder2_temp = 12'b0001_0000_0000;
            4'b1010:      decoder2_temp = 12'b0010_0000_0000;
            4'b1011:      decoder2_temp = 12'b0100_0000_0000;
            4'b1100:      decoder2_temp = 12'b1000_0000_0000;
            default:      decoder2_temp = 12'b0000_0000_0000;
        endcase
    end
end

ALU alu(        //调用ALU模块
    .src0(src0_reg),
    .src1(src1_reg),
    .sel(decoder2_temp),
    .res(alu_temp)
);

Segment segment(        //调用Segment模块
    .clk(clk),
    .rst(rst),
    .output_data({27'b0, out_reg}),     //前面补0
    .output_valid(8'b11111111),         //不需要用到掩码
    .seg_data(seg_data),
    .seg_an(seg_an)
);
```

```
always @(*) begin       //运算选择sel_reg的寄存器，只在其对应使能信号有效时进行更新
    if(rst) begin
        sel_reg = 0;
    end
    else begin
        if(sel_en) begin
            sel_reg = in[3:0];
        end
        else begin
            sel_reg = sel_reg;
        end
    end
end

always @(*) begin       //数据源src0_reg的寄存器，只在其对应使能信号有效时进行更新
    if(rst) begin
        src0_reg = 0;
    end
    else begin
        if(src0_en) begin
            src0_reg = in[4:0];
        end
        else begin
            src0_reg = src0_reg;
        end
    end
end

always @(*) begin       //数码管输出out_reg的寄存器，只在其对应使能信号有效时进行更新
    if(rst) begin
        src1_reg = 0;
    end
    else begin
        if(src1_en) begin
            src1_reg = in[4:0];
        end
        else begin
            src1_reg = src1_reg;
        end
    end
end

always @(*) begin       //数据源src0_reg的寄存器，只在其对应使能信号有效时进行更新
    if(rst) begin
        out_reg = 0;
    end
    else begin
        if(out_en) begin
            out_reg = alu_temp;
        end
        else begin
            out_reg = out_reg;
        end
    end
```

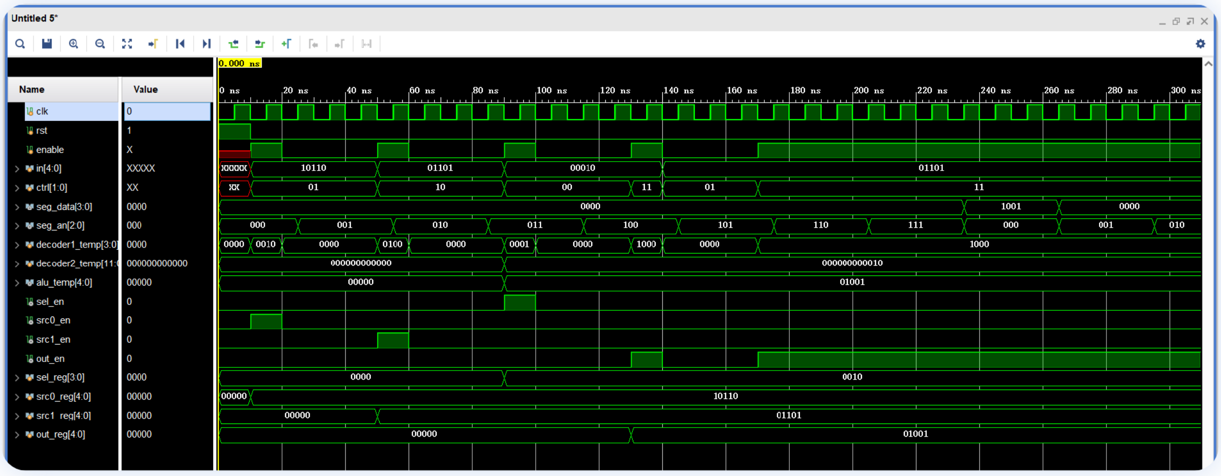
```
end

endmodule
```

> 注：ALU, Sub, Comp 模块代码同上题，仅对所有表示位数的值进行修改 (31 修改为 4)

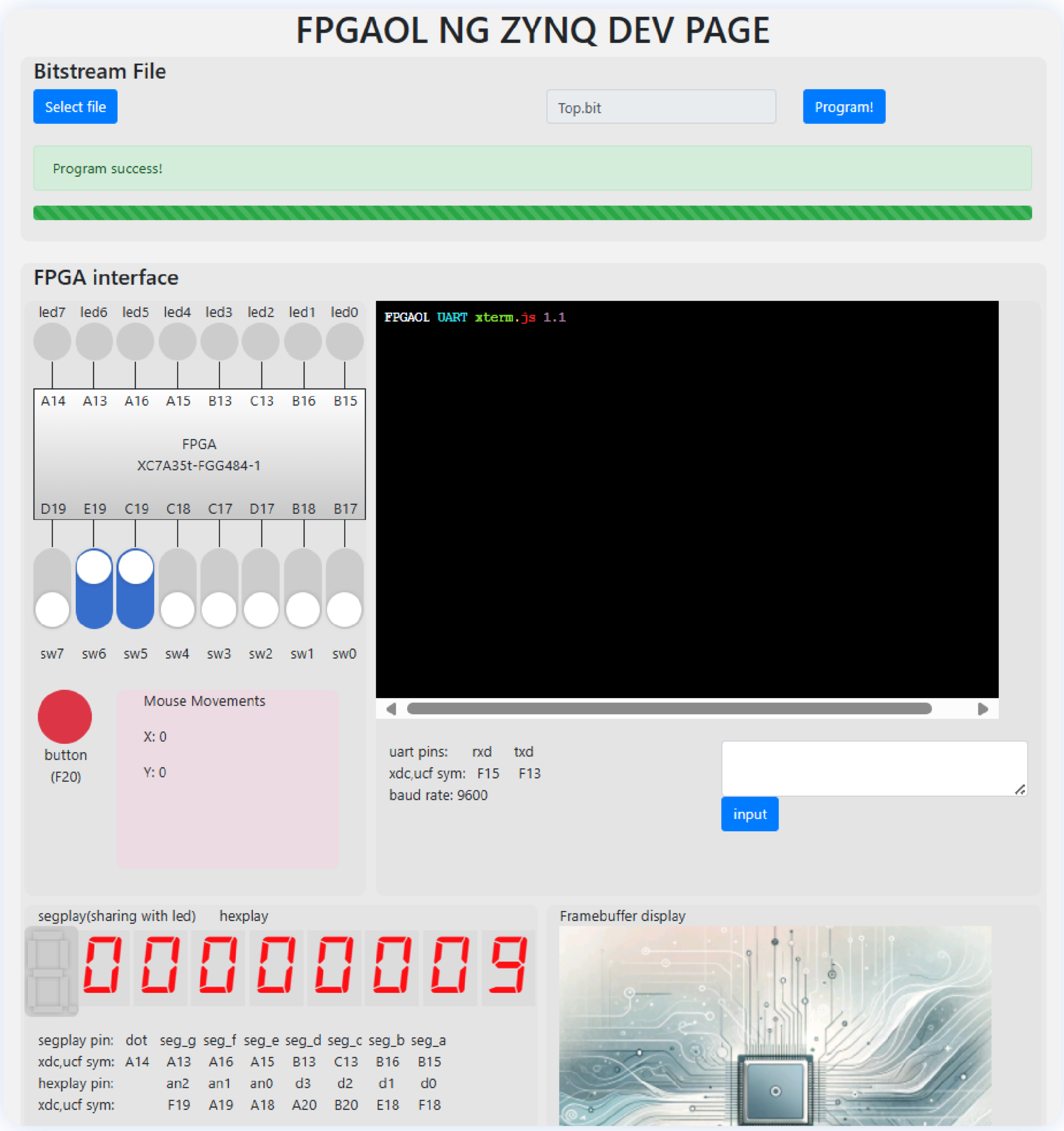
仿真

```
module Top_tb();
reg                                     clk;
reg                                     rst;
reg                                     enable;
reg                     [ 4 : 0]        in;
reg                     [ 1 : 0]        ctrl;
wire                    [ 3 : 0]        seg_data;
wire                    [ 2 : 0]        seg_an;

initial begin
    clk=0;
    rst=1;
    #10 rst=0;

    in=5'b10110; ctrl=2'b01; enable=1'b1;
    #10;
    enable=1'b0;
    #30;
    in=5'b01101; ctrl=2'b10; enable=1'b1;
    #10;
    enable=1'b0;
    #30;
    in=5'b00010; ctrl=2'b00; enable=1'b1;
    #10;
    enable=1'b0;
    #30;
    ctrl=2'b11; enable=1'b1;
    #10;
    in=5'b01101; ctrl=2'b01; enable=1'b0;
    #30 ctrl=2'b11; enable=1'b1;
end

always #5 clk = ~clk;

Top top(
    .clk(clk),
    .rst(rst),
    .enable(enable),
    .in(in),
    .ctrl(ctrl),
    .seg_data(seg_data),
    .seg_an(seg_an)
);
```

```
endmodule
```



实现



FPGAOL NG ZYNQ DEV PAGE

Bitstream File

Select file   Top.bit   Program!

Program success!

FPGA interface

led7	led6	led5	led4	led3	led2	led1	led0
A14	A13	A16	A15	B13	C13	B16	B15

FPGA
XC7A35t-FGG484-1

D19	E19	C19	C18	C17	D17	B18	B17
sw7	sw6	sw5	sw4	sw3	sw2	sw1	sw0

button
(F20)

Mouse Movements

X: 0

Y: 0

```
FPGAOL UART xterm.js 1.1
```

uart pins:	rxd	txd
xdc,ucf sym:	F15	F13

baud rate: 9600

input

segplay(sharing with led)   hexplay

00000009

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	A14	A13	A16	A15	B13	C13	B16	B15
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		F19	A19	A18	A20	B20	E18	F18

Framebuffer display