

数字电路实验 Lab 7 实验报告

奚项正 PB23000020

数字电路实验 Lab 7 实验报告

必做内容

题目 1: 乘法器基础版 (3 分)

题目 2: 乘法器优化版 (2 分)

题目 3: 性能比较 (2 分)

题目 4: 可选有无符号乘法 (2 分)

选择性必做内容

题目 1: Logisim 搭建乘法器 (2 分)

必做内容

题目 1: 乘法器基础版 (3 分)

设计

```
module MUL #(
    parameter WIDTH = 32 // 仿真时修改为 8, 以加快运行速度
) (
    input [0 : 0] clk,
    input [0 : 0] rst,
    input [0 : 0] start,
    input [WIDTH-1 : 0] a,
    input [WIDTH-1 : 0] b,
    output reg [2*WIDTH-1:0] res,
    output reg [0 : 0] finish
);

reg [1 : 0] current_state, next_state; // 乘法器状态
reg [WIDTH-1 : 0] current_bit, next_bit; // 已移位的次数
reg [2*WIDTH-1 : 0] current_multiplicand, next_multiplicand; // 被乘数寄存器
reg [WIDTH-1 : 0] current_multiplier, next_multiplier; // 乘数寄存器
reg [2*WIDTH-1 : 0] current_product, next_product; // 乘积寄存器

localparam IDLE = 2'b00; // 空闲状态。这个周期寄存器保持原值不变。当 start 为 1 时跳转到 INIT。
localparam INIT = 2'b01; // 初始化。下个周期跳转到 CALC
localparam CALC = 2'b10; // 计算中。计算完成时跳转到 DONE
localparam DONE = 2'b11; // 计算完成。下个周期跳转到 IDLE

// Part 1: 使用同步时序进行状态更新, 即更新 current_state 的内容。
always @(posedge clk) begin
    if(rst) begin // 检测复位信号
        current_state <= IDLE;
        current_bit <= 0;
        current_multiplicand <= 0;
        current_multiplier <= 0;
```

```

        current_product ≤ 0;
    end
    else begin        // 进行状态更新
        current_state ≤ next_state;
        current_bit ≤ next_bit;
        current_multiplicand ≤ next_multiplicand;
        current_multiplier ≤ next_multiplier;
        current_product ≤ next_product;
    end
end

// Part 2: 使用组合逻辑判断状态跳转逻辑, 即根据 current_state 与其他信号确定 next_state.
always @(*) begin
    next_state = current_state;    // 先对 next_state 进行默认赋值, 防止出现遗漏
    next_bit = current_bit;
    next_multiplicand = current_multiplicand;
    next_multiplier = current_multiplier;
    next_product = current_product;
    case(current_state)           // 根据当前的state分情况讨论
        IDLE: begin              // 空闲状态, 当 start 为 1 时跳转到 INIT
            if (start) begin
                next_state = INIT;
                next_bit = 0;
                next_multiplicand = a;
                next_multiplier = b;
                next_product = 0;
            end
        end
        INIT: begin              // 初始化, 下个周期跳转到 CALC
            next_state = CALC;
            next_bit = current_bit + 1;
            next_multiplicand = {current_multiplicand[2*WIDTH-2:0], 1'b0};
            next_multiplier = {1'b0, current_multiplier[WIDTH-1:1]};
            if(current_multiplier[0] == 1) begin
                next_product = current_multiplicand + current_product;    //注, 由于参数WIDTH
可変, 此处并未使用Lab6中超前进位加法器的例化, 而直接使用Verilog中自带的加法, 而本实验必做第4题的实现中则采用固
定位数, 并例化使用了Lab6中超前进位加法器
            end
        end
        CALC: begin              // 计算中, 计算完成时跳转到 DONE
            if (current_bit == WIDTH) begin    // current_bit == WIDTH 时即计算完成
                next_state = DONE;
                next_bit = 0;
            end
            else begin            // 否则进行移位, 且如果当前乘数末位为 0, 则乘积与被乘数相加
                next_bit = current_bit + 1;
                next_multiplicand = {current_multiplicand[2*WIDTH-2:0], 1'b0};
                next_multiplier = {1'b0, current_multiplier[WIDTH-1:1]};
                if(current_multiplier[0] == 1) begin
                    next_product = current_multiplicand + current_product;
                end
            end
        end
        DONE: begin

```

```

        next_state = IDLE;
    end
endcase
end

// Part 3: 使用组合逻辑描述状态机的输出。
always @(*)begin
    if (rst) begin
        finish = 0;
        res = 0;
    end
    else begin
        finish = (current_state == DONE) ? 1 : 0;    //仅在 DONE 态时 finish 为 1
        res = finish ? current_product : res;        //finish 为 1 时更新 res, 其他时候保持
    end
end

endmodule

```

仿真

```

module MUL_tb #(
    parameter WIDTH = 8
) ();
reg [WIDTH-1:0] a, b;
reg rst, clk, start;
wire [2*WIDTH-1:0] res;
wire finish;
integer seed;

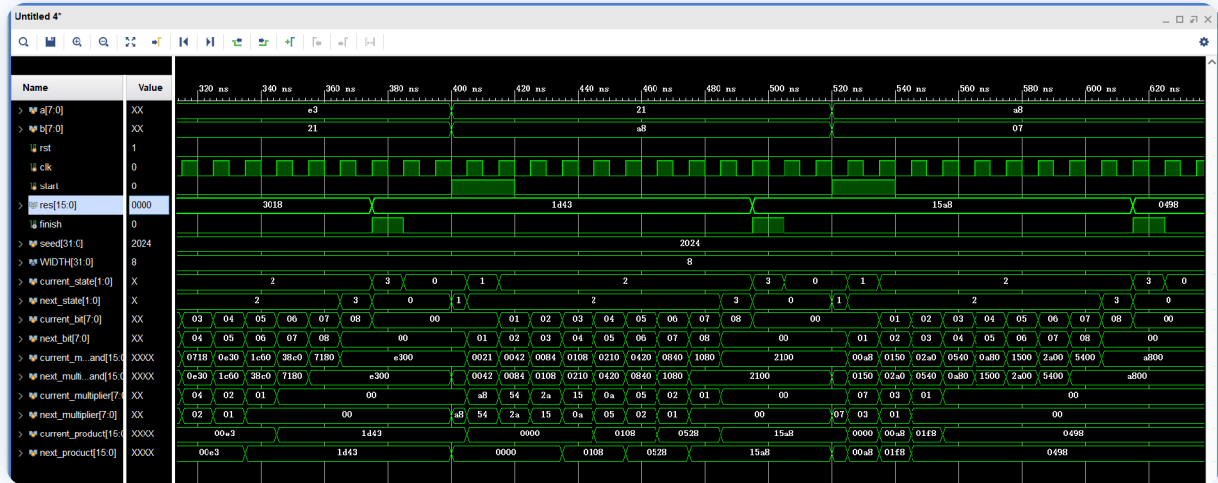
initial begin
    clk = 0;
    seed = 2024; // 种子值
    forever begin
        #5 clk = ~clk;
    end
end

initial begin
    rst = 1;
    start = 0;
    #20;
    rst = 0;
    #20;
    repeat (5) begin
        a = $random(seed);    // $random 返回的是 32 位随机数, 如果你需要得到少于 32 位的随机
数, 可以通过 % 运算得到
        b = $random(seed + 1); // 你可以通过设置种子值改变随机数的序列
        start = 1;
        #20 start = 0;
        #100;
    end
end

```

```
MUL mul(
    .clk          (clk),
    .rst          (rst),
    .start        (start),
    .a            (a),
    .b            (b),
    .res          (res),
    .finish       (finish)
);

endmodule
```



设计

```
module Top(  
    input      clk,  
    input      btn,  
    input [7:0] sw,
```

```

        output [7:0] led,
        output [3:0] d,
        output [2:0] an
    );

    wire [7:0] res;
    assign led = res[7:0];

    MUL mul(
        .clk(clk),
        .rst(sw[7]),
        .start(btn),
        .a({1'b0}, {sw[6:4]}),
        .b(sw[3:0]),
        .res(res),
        .finish()
    );

    Segment segment(
        .clk(clk),
        .rst(rst),
        .output_data({24'b0}, {res[7:0]}),
        .output_valid(8'b00000011),
        .seg_data(d),
        .seg_an(an)
    );
endmodule

```

```

module MUL #(
    parameter WIDTH = 4 // 上板测试需要令WIDTH=4
) (
    input clk,
    input rst,
    input start,
    input [WIDTH-1 : 0] a,
    input [WIDTH-1 : 0] b,
    output reg [2*WIDTH-1:0] res,
    output reg finish
);

reg [1 : 0] current_state, next_state;
reg [WIDTH-1 : 0] current_bit, next_bit;
reg [WIDTH-1 : 0] current_multiplicand, next_multiplicand; // 被乘数寄存器
reg [WIDTH-1 : 0] current_product_low, next_product_low; // 乘积寄存器低位
reg [WIDTH : 0] current_product_high, next_product_high; // 乘积寄存器高位, 即乘积
// 寄存器共计2*WIDTH+1位

localparam IDLE = 2'b00; // 空闲状态。这个周期寄存器保持原值不变。当 start 为 1 时跳转到 INIT。
localparam INIT = 2'b01; // 初始化。下个周期跳转到 CALC
localparam CALC = 2'b10; // 计算中。计算完成时跳转到 DONE
localparam DONE = 2'b11; // 计算完成。下个周期跳转到 IDLE

```

```

always @(posedge clk) begin
    if(rst) begin
        current_state ≤ IDLE;
        current_bit ≤ 0;
        current_multiplicand ≤ 0;
        current_product_low ≤ 0;
        current_product_high ≤ 0;
    end
    else begin
        current_state ≤ next_state;
        current_bit ≤ next_bit;
        current_multiplicand ≤ next_multiplicand;
        current_product_low ≤ next_product_low;
        current_product_high ≤ next_product_high;
    end
end

always @(*) begin
    next_state = current_state;
    next_bit = current_bit;
    next_multiplicand = current_multiplicand;
    next_product_low = current_product_low;
    next_product_high = current_product_high;
    case(current_state)
        IDLE: begin
            if (start) begin
                next_state = INIT;
                next_bit = 0;
                next_multiplicand = a;
                next_product_low = b;          // 初始时乘积寄存器的低WIDTH位存放乘数
                next_product_high = 0;
            end
        end
        INIT: begin
            next_state = CALC;
            next_bit = current_bit + 1;
            next_product_low = {current_product_high[0], current_product_low[WIDTH-1:1]};
            // 乘积低位右移一位，其中最高位继承乘积高位的最低位
            if(current_product_low[0] == 1) begin
                next_product_high = current_multiplicand + current_product_high[WIDTH:1];
                // 乘积高位右移一位，如果乘积低位的最低位为1，即表示需要累加被乘数，则用将被乘数与高位对齐进行相加
            end
            else begin
                next_product_high = current_product_high[WIDTH:1];          // 否则只进行乘积高位右移一位
            end
        end
        CALC: begin
            if (current_bit == WIDTH) begin
                next_state = DONE;
                next_bit = 0;
            end
            else begin
                next_bit = current_bit + 1;
            end
        end
    endcase
end

```

```

        next_product_low = {current_product_high[0], current_product_low[WIDTH-1:1]};
//此处逻辑同上
        if(current_product_low[0] == 1) begin
            next_product_high = current_multiplicand + current_product_high[WIDTH:1];
//注，由于参数WIDTH可变，此处并未使用Lab6中超前进位加法器的例化，而直接使用Verilog中自带的加法，而本实验必做
//第4题的实现中则采用固定位数，并例化使用了Lab6中超前进位加法器
        end
    end
    else begin
        next_product_high = current_product_high[WIDTH:1];
    end
end
end
DONE: begin
    next_state = IDLE;
end
endcase
end

always @(*)begin
    if (rst) begin
        finish = 0;
        res = 0;
    end
    else begin
        finish = (current_state == DONE) ? 1 : 0;
        res = finish ? {current_product_high[WIDTH:0], current_product_low[WIDTH-1:1]} : res;
//由于最后一步未进行右移，故res需要在拼接乘积高位与乘积低位后，截去乘积低位的最低位
    end
end

endmodule

```

```

module Segment(           // Lab3中的数码管显示模块
    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

```

仿真

```

module MUL_tb #(           // 仿真同上题，仅针对MUL模块进行
    parameter WIDTH = 4
) ();
    reg  [WIDTH-1:0]  a, b;
    reg              rst, clk, start;
    wire [2*WIDTH-1:0] res;
    wire              finish;

```



```

integer          seed;

initial begin
    clk = 0;
    seed = 2024; // 种子值
    forever begin
        #5 clk = ~clk;
    end
end

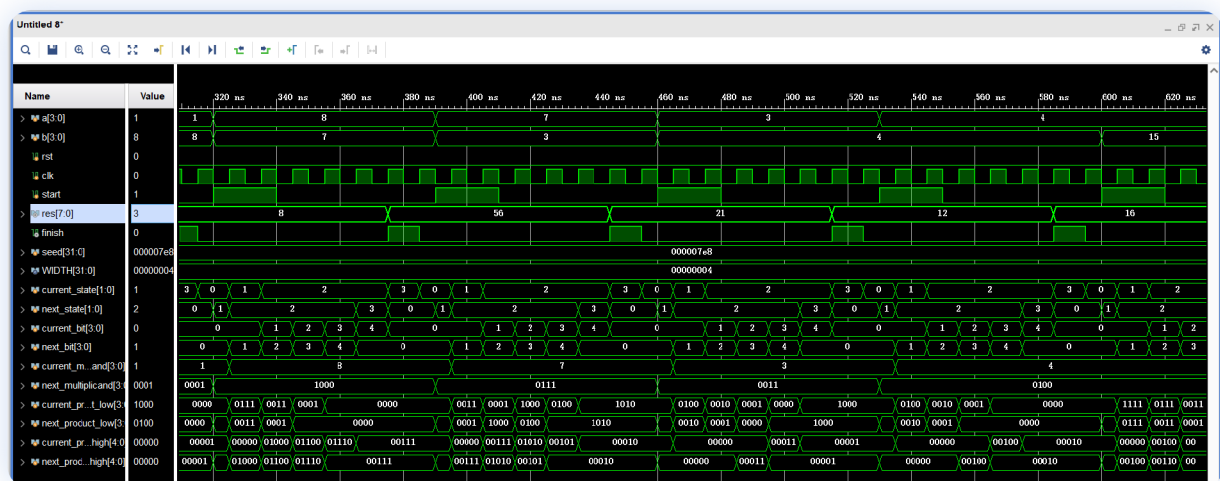
initial begin
    rst = 1;
    start = 0;
    #20;
    rst = 0;
    #20;
    repeat (10) begin
        a = $random(seed);          // $random 返回的是 32 位随机数, 如果你需要得到少于 32 位的随机
数, 可以通过 % 运算得到
        b = $random(seed + 1);      // 你可以通过设置种子值改变随机数的序列
        start = 1;
        #20 start = 0;
        #50;
    end

end

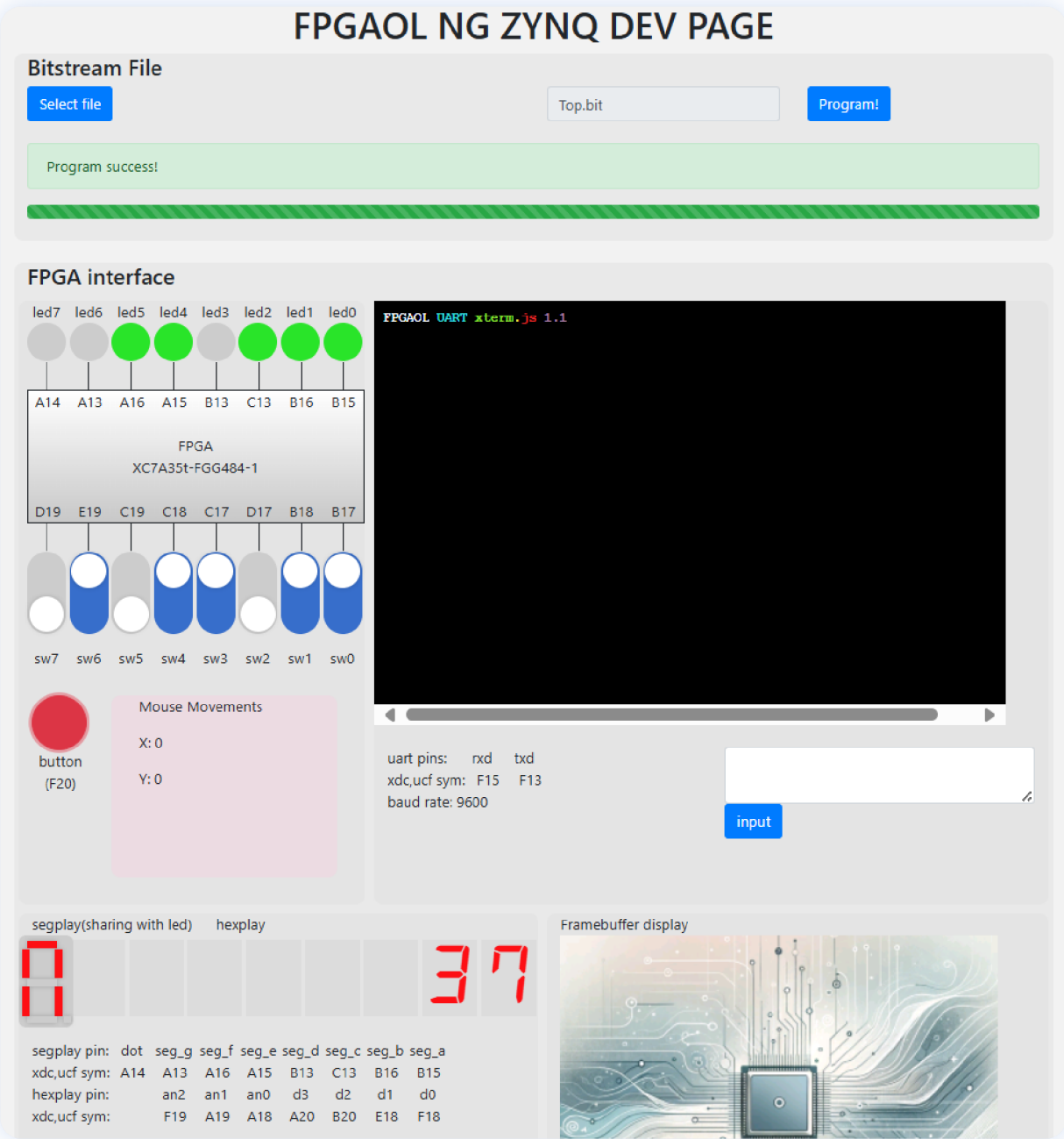
end

MUL mul(
    .clk      (clk),
    .rst      (rst),
    .start    (start),
    .a        (a),
    .b        (b),
    .res      (res),
    .finish   (finish)
);
endmodule

```



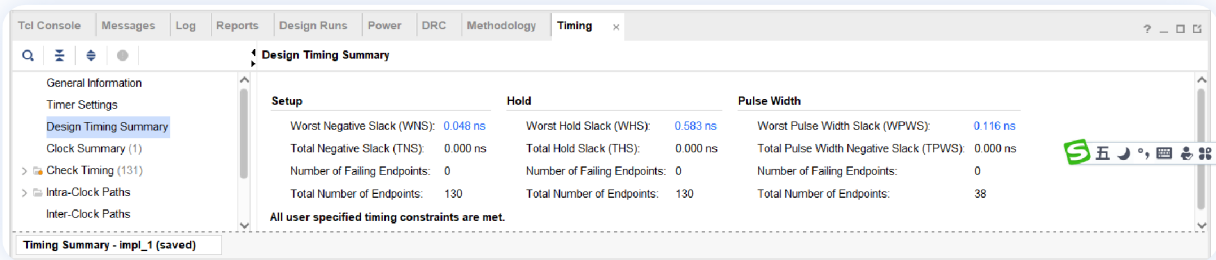
实现



题目 3：性能比较（2 分）

Verilog 中自带的乘法

当 $f = 277.8 \text{ MHz}$ 时，得到如图 1 所示的性能测试结果，即运行时间 $t = \frac{1}{f} = 4 \text{ ns}$



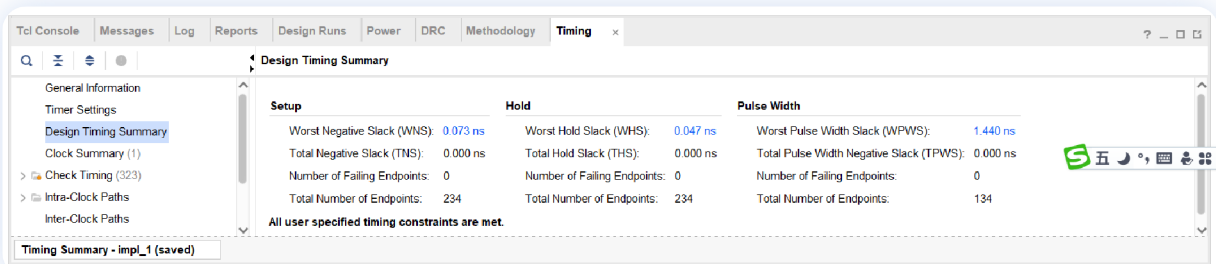
The screenshot shows the 'Timing' tab in a design tool. The 'Design Timing Summary' window is open, displaying a table of timing metrics. The table is divided into three columns: Setup, Hold, and Pulse Width. The 'Setup' column shows Worst Negative Slack (WNS) at 0.048 ns, Total Negative Slack (TNS) at 0.000 ns, and 0 failing endpoints. The 'Hold' column shows Worst Hold Slack (WHS) at 0.583 ns, Total Hold Slack (THS) at 0.000 ns, and 0 failing endpoints. The 'Pulse Width' column shows Worst Pulse Width Slack (WPWS) at 0.116 ns, Total Pulse Width Negative Slack (TPWS) at 0.000 ns, and 0 failing endpoints. The total number of endpoints is 130 for Setup and Hold, and 38 for Pulse Width. A green status bar at the bottom indicates 'All user specified timing constraints are met.'

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.048 ns	Worst Hold Slack (WHS): 0.583 ns	Worst Pulse Width Slack (WPWS): 0.116 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 130	Total Number of Endpoints: 130	Total Number of Endpoints: 38

All user specified timing constraints are met.

自己实现的优化版乘法器

当 $f = 257.7 \text{ MHz}$ 时，得到如图 2 所示的性能测试结果，即运行时间 $t = \frac{WIDTH}{f} = 15.52 \text{ ns}$



The screenshot shows the 'Timing' tab in a design tool. The 'Design Timing Summary' window is open, displaying a table of timing metrics. The table is divided into three columns: Setup, Hold, and Pulse Width. The 'Setup' column shows Worst Negative Slack (WNS) at 0.073 ns, Total Negative Slack (TNS) at 0.000 ns, and 0 failing endpoints. The 'Hold' column shows Worst Hold Slack (WHS) at 0.047 ns, Total Hold Slack (THS) at 0.000 ns, and 0 failing endpoints. The 'Pulse Width' column shows Worst Pulse Width Slack (WPWS) at 1.440 ns, Total Pulse Width Negative Slack (TPWS) at 0.000 ns, and 0 failing endpoints. The total number of endpoints is 234 for Setup and Hold, and 134 for Pulse Width. A green status bar at the bottom indicates 'All user specified timing constraints are met.'

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.073 ns	Worst Hold Slack (WHS): 0.047 ns	Worst Pulse Width Slack (WPWS): 1.440 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 234	Total Number of Endpoints: 234	Total Number of Endpoints: 134

All user specified timing constraints are met.

题目 4：可选有无符号乘法（2 分）

设计

```
module Top (
    input                clk,
    input                rst,
    input                start,
    input                [31 : 0] a,
    input                [31 : 0] b,
    input                mul_signed,
    output reg           [62 : 0] res,
    output               finish
);

wire [31:0] a_sub, a_mul, b_sub, b_mul;
wire [61:0] res_sub, res_mul, res_unsigned;
wire       res_mid;
assign a_mul = a[31] ? a_sub : a;           // 有符号乘法器中使用a的绝对值进行运算
assign b_mul = b[31] ? b_sub : b;

always @(*) begin
    if (rst) begin
        res = 0;
    end
    else begin
        if (finish) begin
            if (mul_signed) begin           // 将有符号乘法器所得的乘积绝对值，根据a,b的符号，再添回符号

```

```

        res = (a[31] == b[31]) ? {{1'b0}, {res_mul}} : {{1'b1}, {res_sub}};
    end
    else begin
        // mul_signed=0时, 无符号乘法器
        res = res_unsigned;
    end

    end

    else begin
        res = res;
    end

    end

end

Adder adder0(    //计算a的相反数, 即取反加一
    .a(0),
    .b(~a),
    .ci(1),
    .s(a_sub),
    .co()
);

Adder adder1(    //计算b的相反数
    .a(0),
    .b(~b),
    .ci(1),
    .s(b_sub),
    .co()
);

Adder adder2(    //计算res低位的相反数
    .a(0),
    .b(~(res_mul[31:0])),
    .ci(1),
    .s(res_sub[31:0]),
    .co(res_mid)
);

Adder adder3(    //计算res高位的相反数, 其是否加一, 取决于低位中有无进位res_mid
    .a(0),
    .b(~(res_mul[61:32])),
    .ci(res_mid),
    .s(res_sub[61:32]),
    .co()
);

MUL mul(
    //有符号乘法器部分, 例化使用(无符号)乘法器
    .clk      (clk),
    .rst      (rst),
    .start    (start),
    .a        (a_mul),
    .b        (b_mul),
    .res      (res_mul),
    .finish   (finish)
);

```

```

    MUL mul_unsigned(          // 无符号乘法器部分，结果存至res_unsigned
        .clk      (clk),
        .rst      (rst),
        .start    (start),
        .a        (a),
        .b        (b),
        .res      (res_unsigned),
        .finish   (finish)
    );
endmodule

```

```

module MUL (                // 与本实验必做第2题基本相同，但采用固定位数，并例化使用了超前进位加法器
    input                    clk,
    input                    rst,
    input                    start,
    input [31 : 0]          a,
    input [31 : 0]          b,
    output reg [61 : 0]     res,
    output reg              finish
);

reg [ 1 : 0]    current_state, next_state;
reg [31 : 0]    current_bit, next_bit;
reg [31 : 0]    current_multiplicand, next_multiplicand;    // 被乘数寄存器
reg [31 : 0]    current_product_low, next_product_low;    // 乘积寄存器低位
reg [32 : 0]    current_product_high, next_product_high;    // 乘积寄存器高位
wire [32 : 0]    adder_temp;

localparam IDLE = 2'b00;    // 空闲状态。这个周期寄存器保持原值不变。当 start 为 1 时跳转到 INIT。
localparam INIT = 2'b01;    // 初始化。下个周期跳转到 CALC
localparam CALC = 2'b10;    // 计算中。计算完成时跳转到 DONE
localparam DONE = 2'b11;    // 计算完成。下个周期跳转到 IDLE

always @(posedge clk) begin
    if(rst) begin
        current_state ≤ IDLE;
        current_bit ≤ 0;
        current_multiplicand ≤ 0;
        current_product_low ≤ 0;
        current_product_high ≤ 0;
    end
    else begin
        current_state ≤ next_state;
        current_bit ≤ next_bit;
        current_multiplicand ≤ next_multiplicand;
        current_product_low ≤ next_product_low;
        current_product_high ≤ next_product_high;
    end
end
end

```

```

always @(*) begin
    next_state = current_state;
    next_bit = current_bit;
    next_multiplicand = current_multiplicand;
    next_product_low = current_product_low;
    next_product_high = current_product_high;
    case(current_state)
        IDLE: begin
            if (start) begin
                next_state = INIT;
                next_bit = 0;
                next_multiplicand = a;
                next_product_low = b;
                next_product_high = 0;
            end
        end
        INIT: begin
            next_state = CALC;
            next_bit = current_bit + 1;
            next_product_low = {current_product_high[0], current_product_low[31:1]};
            if(current_product_low[0] == 1) begin
                next_product_high = adder_temp;    //不再使用Verilog中自带的加法
            end
        end
        else begin
            next_product_high = current_product_high[32:1];
        end
    end
    CALC: begin
        if (current_bit == 32) begin
            next_state = DONE;
            next_bit = 0;
        end
        else begin
            next_bit = current_bit + 1;
            next_product_low = {current_product_high[0], current_product_low[31:1]};
            if(current_product_low[0] == 1) begin
                next_product_high = adder_temp;
            end
            else begin
                next_product_high = current_product_high[32:1];
            end
        end
    end
    DONE: begin
        next_state = IDLE;
    end
endcase
end

always @(*)begin
    if (rst) begin
        finish = 0;
        res = 0;
    end
end

```

```

    else begin
        finish = (current_state == DONE) ? 1 : 0;
        res = finish ? {current_product_high[32:0], current_product_low[31:1]} : res;
    end
end

Adder adder_mul(          // 例化使用了超前进位加法器
    .a(current_multiplicand),
    .b(current_product_high[32:1]),
    .ci(0),
    .s(adder_temp[31:0]),
    .co(adder_temp[32])
);
endmodule

```

注：Adder, LookAhead8 模块与 Lab 6 中的相同

仿真

```

module Top_tb();
    reg  [31:0]      a, b;
    reg             rst, clk, start, mul_signed;
    wire [62:0]      res;
    wire            finish;
    integer          seed;

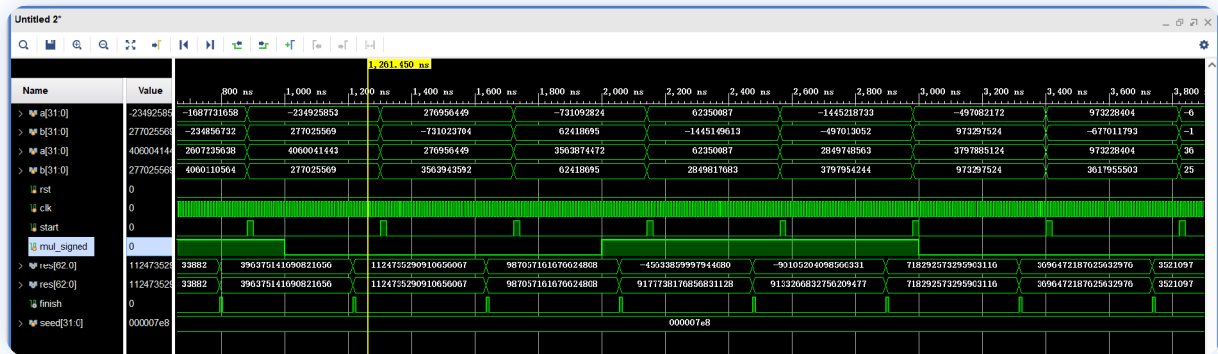
    initial begin
        clk = 0;
        seed = 2024; // 种子值
        forever begin
            #5 clk = ~clk;
        end
    end

    initial begin
        rst = 1;
        mul_signed = 1;
        start = 0;
        #20;
        rst = 0;
        #20;
        repeat (10) begin
            a = $random(seed);          // $random 返回的是 32 位随机数，如果你需要得到少于 32 位的随机
            b = $random(seed + 1);      // 你可以通过设置种子值改变随机数的序列
            start = 1;
            #20 start = 0;
            #400;
        end
    end
end

```

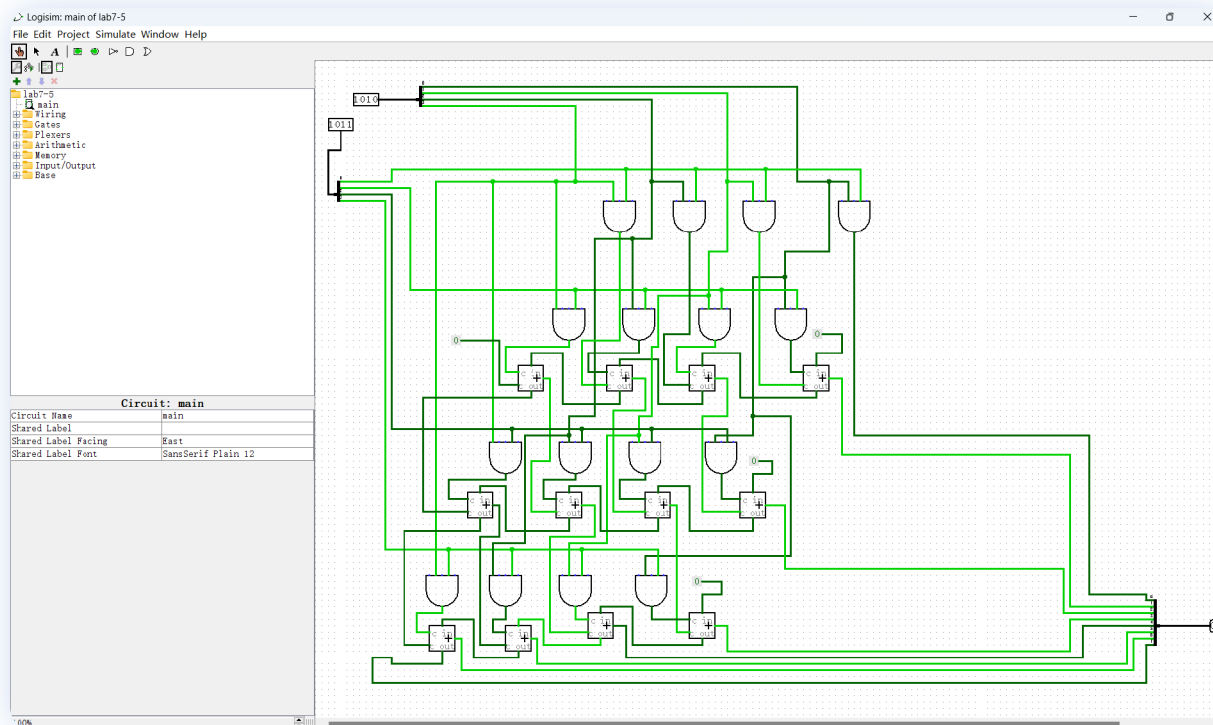
```
always #1000 mul_signed = ~mul_signed; //切换mul_signed进行选择有无符号
```

```
Top top(  
    .clk      (clk),  
    .rst      (rst),  
    .start    (start),  
    .a        (a),  
    .b        (b),  
    .mul_signed (mul_signed),  
    .res      (res),  
    .finish    (finish)  
);  
endmodule
```



选择性必做内容

题目 1: Logisim 搭建乘法器 (2 分)



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project source="2.7.1" version="1.0">
This file is intended to be loaded by Logisim (http://www.cburch.com/logisim/).
<lib desc="#Wiring" name="0"/>
  <lib desc="#Gates" name="1"/>
  <lib desc="#Plexers" name="2"/>
  <lib desc="#Arithmetic" name="3">
    <tool name="Adder">
      <a name="width" val="1"/>
    </tool>
  </lib>
  <lib desc="#Memory" name="4"/>
  <lib desc="#I/O" name="5"/>
  <lib desc="#Base" name="6">
    <tool name="Text Tool">
      <a name="text" val=""/>
      <a name="font" val="SansSerif plain 12"/>
      <a name="halign" val="center"/>
      <a name="valign" val="base"/>
    </tool>
  </lib>
  <main name="main"/>
  <options>
    <a name="gateUndefined" val="ignore"/>
    <a name="simlimit" val="1000"/>
    <a name="simrand" val="0"/>
  </options>
  <mappings>
    <tool lib="6" map="Button2" name="Menu Tool"/>
  </mappings>
</project>
```

```

<tool lib="6" map="Button3" name="Menu Tool"/>
<tool lib="6" map="Ctrl Button1" name="Menu Tool"/>
</mappings>
<toolbar>
  <tool lib="6" name="Poke Tool"/>
  <tool lib="6" name="Edit Tool"/>
  <tool lib="6" name="Text Tool">
    <a name="text" val=""/>
    <a name="font" val="SansSerif plain 12"/>
    <a name="halign" val="center"/>
    <a name="valign" val="base"/>
  </tool>
  <sep/>
  <tool lib="0" name="Pin">
    <a name="tristate" val="false"/>
  </tool>
  <tool lib="0" name="Pin">
    <a name="facing" val="west"/>
    <a name="output" val="true"/>
    <a name="labelloc" val="east"/>
  </tool>
  <tool lib="1" name="NOT Gate"/>
  <tool lib="1" name="AND Gate"/>
  <tool lib="1" name="OR Gate"/>
</toolbar>
<circuit name="main">
  <a name="circuit" val="main"/>
  <a name="clabel" val=""/>
  <a name="clabelup" val="east"/>
  <a name="clabelfont" val="SansSerif plain 12"/>
  <wire from="(510,590)" to="(510,600)" />
  <wire from="(480,470)" to="(540,470)" />
  <wire from="(470,550)" to="(520,550)" />
  <wire from="(470,710)" to="(520,710)" />
  <wire from="(280,760)" to="(330,760)" />
  <wire from="(670,460)" to="(670,530)" />
  <wire from="(410,650)" to="(410,660)" />
  <wire from="(1210,890)" to="(1310,890)" />
  <wire from="(320,880)" to="(320,890)" />
  <wire from="(300,860)" to="(300,880)" />
  <wire from="(420,580)" to="(420,600)" />
  <wire from="(400,720)" to="(400,740)" />
  <wire from="(830,430)" to="(830,460)" />
  <wire from="(370,750)" to="(370,900)" />
  <wire from="(760,360)" to="(760,390)" />
  <wire from="(520,360)" to="(520,390)" />
  <wire from="(400,560)" to="(400,590)" />
  <wire from="(180,70)" to="(410,70)" />
  <wire from="(260,670)" to="(300,670)" />
  <wire from="(420,510)" to="(460,510)" />
  <wire from="(210,770)" to="(310,770)" />
  <wire from="(1210,890)" to="(1210,940)" />
  <wire from="(650,890)" to="(650,930)" />
  <wire from="(630,500)" to="(650,500)" />

```

```
<wire from="(500,280)" to="(530,280)" />
<wire from="(300,490)" to="(320,490)" />
<wire from="(1220,590)" to="(1220,840)" />
<wire from="(210,770)" to="(210,810)" />
<wire from="(350,910)" to="(350,950)" />
<wire from="(810,190)" to="(840,190)" />
<wire from="(1280,900)" to="(1310,900)" />
<wire from="(490,170)" to="(490,220)" />
<wire from="(490,890)" to="(490,940)" />
<wire from="(280,760)" to="(280,810)" />
<wire from="(740,340)" to="(740,390)" />
<wire from="(520,650)" to="(520,660)" />
<wire from="(540,670)" to="(540,680)" />
<wire from="(400,760)" to="(460,760)" />
<wire from="(630,440)" to="(630,450)" />
<wire from="(850,590)" to="(1220,590)" />
<wire from="(690,340)" to="(740,340)" />
<wire from="(100,60)" to="(160,60)" />
<wire from="(790,520)" to="(790,530)" />
<wire from="(540,670)" to="(590,670)" />
<wire from="(460,590)" to="(510,590)" />
<wire from="(570,450)" to="(630,450)" />
<wire from="(630,590)" to="(690,590)" />
<wire from="(420,770)" to="(540,770)" />
<wire from="(590,670)" to="(590,740)" />
<wire from="(320,930)" to="(320,940)" />
<wire from="(620,300)" to="(620,370)" />
<wire from="(410,860)" to="(410,880)" />
<wire from="(520,780)" to="(750,780)" />
<wire from="(120,170)" to="(120,190)" />
<wire from="(510,560)" to="(510,590)" />
<wire from="(140,920)" to="(180,920)" />
<wire from="(530,580)" to="(530,600)" />
<wire from="(210,660)" to="(210,690)" />
<wire from="(280,190)" to="(280,600)" />
<wire from="(610,560)" to="(610,710)" />
<wire from="(570,490)" to="(590,490)" />
<wire from="(530,450)" to="(530,560)" />
<wire from="(710,700)" to="(740,700)" />
<wire from="(90,940)" to="(90,980)" />
<wire from="(500,500)" to="(520,500)" />
<wire from="(610,840)" to="(640,840)" />
<wire from="(480,270)" to="(480,440)" />
<wire from="(190,190)" to="(190,810)" />
<wire from="(650,930)" to="(1190,930)" />
<wire from="(540,540)" to="(610,540)" />
<wire from="(340,910)" to="(350,910)" />
<wire from="(540,370)" to="(620,370)" />
<wire from="(50,210)" to="(120,210)" />
<wire from="(600,170)" to="(600,220)" />
<wire from="(520,360)" to="(640,360)" />
<wire from="(120,580)" to="(310,580)" />
<wire from="(300,670)" to="(300,740)" />
<wire from="(50,220)" to="(110,220)" />
```

```
<wire from="(120,940)" to="(120,950)" />
<wire from="(200,860)" to="(200,870)" />
<wire from="(610,460)" to="(670,460)" />
<wire from="(340,520)" to="(340,530)" />
<wire from="(610,560)" to="(650,560)" />
<wire from="(340,630)" to="(340,710)" />
<wire from="(520,780)" to="(520,810)" />
<wire from="(480,660)" to="(480,690)" />
<wire from="(540,720)" to="(540,740)" />
<wire from="(760,40)" to="(760,190)" />
<wire from="(110,770)" to="(210,770)" />
<wire from="(640,820)" to="(640,840)" />
<wire from="(710,170)" to="(860,170)" />
<wire from="(1280,900)" to="(1280,950)" />
<wire from="(1290,910)" to="(1290,960)" />
<wire from="(510,560)" to="(530,560)" />
<wire from="(1330,890)" to="(1410,890)" />
<wire from="(470,550)" to="(470,710)" />
<wire from="(180,50)" to="(650,50)" />
<wire from="(690,650)" to="(720,650)" />
<wire from="(590,270)" to="(590,430)" />
<wire from="(700,510)" to="(770,510)" />
<wire from="(360,500)" to="(370,500)" />
<wire from="(120,210)" to="(120,580)" />
<wire from="(440,700)" to="(440,750)" />
<wire from="(710,170)" to="(710,220)" />
<wire from="(560,700)" to="(570,700)" />
<wire from="(250,940)" to="(320,940)" />
<wire from="(180,40)" to="(760,40)" />
<wire from="(420,440)" to="(420,510)" />
<wire from="(530,580)" to="(650,580)" />
<wire from="(740,800)" to="(1180,800)" />
<wire from="(760,190)" to="(810,190)" />
<wire from="(370,500)" to="(370,630)" />
<wire from="(480,470)" to="(480,480)" />
<wire from="(850,270)" to="(850,590)" />
<wire from="(840,190)" to="(840,220)" />
<wire from="(1180,800)" to="(1180,870)" />
<wire from="(590,740)" to="(690,740)" />
<wire from="(310,770)" to="(420,770)" />
<wire from="(610,840)" to="(610,870)" />
<wire from="(300,450)" to="(400,450)" />
<wire from="(280,190)" to="(380,190)" />
<wire from="(790,460)" to="(790,480)" />
<wire from="(160,870)" to="(160,900)" />
<wire from="(50,200)" to="(150,200)" />
<wire from="(610,520)" to="(610,540)" />
<wire from="(320,880)" to="(360,880)" />
<wire from="(620,370)" to="(620,390)" />
<wire from="(400,560)" to="(440,560)" />
<wire from="(40,110)" to="(40,140)" />
<wire from="(720,630)" to="(720,650)" />
<wire from="(540,770)" to="(540,810)" />
<wire from="(810,190)" to="(810,300)" />
```

```
<wire from="(150,200)" to="(150,360)" />
<wire from="(360,920)" to="(450,920)" />
<wire from="(630,890)" to="(650,890)" />
<wire from="(190,190)" to="(280,190)" />
<wire from="(260,750)" to="(290,750)" />
<wire from="(650,190)" to="(650,300)" />
<wire from="(500,280)" to="(500,390)" />
<wire from="(470,890)" to="(490,890)" />
<wire from="(300,450)" to="(300,490)" />
<wire from="(1180,870)" to="(1310,870)" />
<wire from="(460,590)" to="(460,760)" />
<wire from="(160,900)" to="(180,900)" />
<wire from="(20,210)" to="(30,210)" />
<wire from="(250,880)" to="(250,940)" />
<wire from="(90,980)" to="(1310,980)" />
<wire from="(330,590)" to="(400,590)" />
<wire from="(1290,910)" to="(1310,910)" />
<wire from="(570,700)" to="(570,900)" />
<wire from="(670,530)" to="(790,530)" />
<wire from="(290,440)" to="(290,510)" />
<wire from="(630,590)" to="(630,600)" />
<wire from="(200,880)" to="(200,890)" />
<wire from="(340,460)" to="(390,460)" />
<wire from="(120,170)" to="(490,170)" />
<wire from="(530,880)" to="(590,880)" />
<wire from="(400,440)" to="(400,450)" />
<wire from="(200,880)" to="(250,880)" />
<wire from="(380,190)" to="(380,390)" />
<wire from="(690,560)" to="(750,560)" />
<wire from="(280,880)" to="(280,900)" />
<wire from="(340,460)" to="(340,480)" />
<wire from="(480,520)" to="(480,540)" />
<wire from="(410,360)" to="(520,360)" />
<wire from="(690,650)" to="(690,680)" />
<wire from="(390,460)" to="(390,540)" />
<wire from="(690,340)" to="(690,560)" />
<wire from="(640,360)" to="(640,390)" />
<wire from="(310,580)" to="(420,580)" />
<wire from="(750,560)" to="(750,780)" />
<wire from="(300,740)" to="(400,740)" />
<wire from="(540,370)" to="(540,450)" />
<wire from="(470,190)" to="(470,220)" />
<wire from="(340,710)" to="(380,710)" />
<wire from="(440,280)" to="(440,560)" />
<wire from="(480,690)" to="(520,690)" />
<wire from="(570,450)" to="(570,490)" />
<wire from="(450,740)" to="(540,740)" />
<wire from="(740,300)" to="(740,340)" />
<wire from="(430,490)" to="(460,490)" />
<wire from="(210,660)" to="(300,660)" />
<wire from="(290,510)" to="(320,510)" />
<wire from="(410,880)" to="(430,880)" />
<wire from="(360,880)" to="(360,920)" />
<wire from="(330,590)" to="(330,760)" />
```

```
<wire from="(230,910)" to="(230,960)" />
<wire from="(400,760)" to="(400,810)" />
<wire from="(830,500)" to="(830,620)" />
<wire from="(700,270)" to="(700,510)" />
<wire from="(500,860)" to="(500,920)" />
<wire from="(530,450)" to="(540,450)" />
<wire from="(440,280)" to="(500,280)" />
<wire from="(20,140)" to="(20,210)" />
<wire from="(450,860)" to="(500,860)" />
<wire from="(450,860)" to="(450,870)" />
<wire from="(260,670)" to="(260,680)" />
<wire from="(650,190)" to="(690,190)" />
<wire from="(510,440)" to="(510,460)" />
<wire from="(420,580)" to="(530,580)" />
<wire from="(490,170)" to="(600,170)" />
<wire from="(830,620)" to="(1190,620)" />
<wire from="(580,190)" to="(580,220)" />
<wire from="(200,930)" to="(200,950)" />
<wire from="(550,430)" to="(550,510)" />
<wire from="(1190,860)" to="(1310,860)" />
<wire from="(530,860)" to="(530,880)" />
<wire from="(260,920)" to="(300,920)" />
<wire from="(610,460)" to="(610,480)" />
<wire from="(650,580)" to="(650,600)" />
<wire from="(1310,840)" to="(1310,850)" />
<wire from="(480,660)" to="(520,660)" />
<wire from="(180,60)" to="(530,60)" />
<wire from="(1190,880)" to="(1190,930)" />
<wire from="(640,650)" to="(640,690)" />
<wire from="(280,880)" to="(300,880)" />
<wire from="(1310,920)" to="(1310,980)" />
<wire from="(420,700)" to="(440,700)" />
<wire from="(20,140)" to="(40,140)" />
<wire from="(800,430)" to="(830,430)" />
<wire from="(740,700)" to="(740,800)" />
<wire from="(740,300)" to="(810,300)" />
<wire from="(120,950)" to="(200,950)" />
<wire from="(1220,840)" to="(1310,840)" />
<wire from="(220,910)" to="(230,910)" />
<wire from="(430,460)" to="(510,460)" />
<wire from="(650,500)" to="(650,560)" />
<wire from="(290,700)" to="(290,750)" />
<wire from="(490,940)" to="(1210,940)" />
<wire from="(750,440)" to="(750,490)" />
<wire from="(640,360)" to="(760,360)" />
<wire from="(370,900)" to="(430,900)" />
<wire from="(610,910)" to="(610,920)" />
<wire from="(650,50)" to="(650,190)" />
<wire from="(530,190)" to="(580,190)" />
<wire from="(230,440)" to="(290,440)" />
<wire from="(140,740)" to="(260,740)" />
<wire from="(360,660)" to="(410,660)" />
<wire from="(450,910)" to="(450,920)" />
<wire from="(540,470)" to="(540,540)" />
```

```
<wire from="(260,720)" to="(260,740)" />
<wire from="(690,190)" to="(690,220)" />
<wire from="(170,530)" to="(340,530)" />
<wire from="(790,460)" to="(830,460)" />
<wire from="(350,950)" to="(1280,950)" />
<wire from="(230,960)" to="(1290,960)" />
<wire from="(160,870)" to="(200,870)" />
<wire from="(600,170)" to="(710,170)" />
<wire from="(360,660)" to="(360,690)" />
<wire from="(810,500)" to="(830,500)" />
<wire from="(390,540)" to="(480,540)" />
<wire from="(210,690)" to="(240,690)" />
<wire from="(1190,620)" to="(1190,860)" />
<wire from="(570,900)" to="(590,900)" />
<wire from="(700,630)" to="(720,630)" />
<wire from="(360,690)" to="(380,690)" />
<wire from="(620,300)" to="(650,300)" />
<wire from="(310,770)" to="(310,810)" />
<wire from="(280,700)" to="(290,700)" />
<wire from="(170,530)" to="(170,710)" />
<wire from="(140,740)" to="(140,920)" />
<wire from="(520,500)" to="(520,550)" />
<wire from="(150,360)" to="(410,360)" />
<wire from="(450,670)" to="(450,740)" />
<wire from="(410,190)" to="(470,190)" />
<wire from="(420,440)" to="(480,440)" />
<wire from="(400,670)" to="(450,670)" />
<wire from="(530,60)" to="(530,190)" />
<wire from="(400,670)" to="(400,680)" />
<wire from="(400,590)" to="(400,600)" />
<wire from="(610,710)" to="(670,710)" />
<wire from="(300,650)" to="(300,660)" />
<wire from="(550,430)" to="(590,430)" />
<wire from="(550,510)" to="(590,510)" />
<wire from="(310,580)" to="(310,600)" />
<wire from="(530,190)" to="(530,280)" />
<wire from="(690,560)" to="(690,590)" />
<wire from="(410,360)" to="(410,390)" />
<wire from="(430,460)" to="(430,490)" />
<wire from="(500,920)" to="(610,920)" />
<wire from="(1190,880)" to="(1310,880)" />
<wire from="(690,720)" to="(690,740)" />
<wire from="(90,940)" to="(120,940)" />
<wire from="(750,490)" to="(770,490)" />
<wire from="(340,630)" to="(370,630)" />
<wire from="(380,190)" to="(410,190)" />
<wire from="(110,220)" to="(110,770)" />
<wire from="(640,690)" to="(670,690)" />
<wire from="(280,900)" to="(300,900)" />
<wire from="(610,820)" to="(640,820)" />
<wire from="(420,770)" to="(420,810)" />
<wire from="(260,750)" to="(260,920)" />
<wire from="(170,710)" to="(240,710)" />
<wire from="(50,190)" to="(120,190)" />
```

```

<wire from="(410,70)" to="(410,190)" />
<wire from="(860,170)" to="(860,220)" />
<wire from="(370,750)" to="(440,750)" />
<comp lib="3" loc="(360,500)" name="Adder">
  <a name="width" val="1" />
</comp>
<comp lib="0" loc="(1330,890)" name="Splitter">
  <a name="facing" val="west" />
  <a name="fanout" val="8" />
  <a name="incoming" val="8" />
  <a name="appear" val="center" />
</comp>
<comp lib="3" loc="(340,910)" name="Adder">
  <a name="width" val="1" />
</comp>
<comp lib="0" loc="(800,430)" name="Constant">
  <a name="value" val="0x0" />
</comp>
<comp lib="0" loc="(30,210)" name="Splitter">
  <a name="fanout" val="4" />
  <a name="incoming" val="4" />
  <a name="appear" val="center" />
</comp>
<comp lib="3" loc="(500,500)" name="Adder">
  <a name="width" val="1" />
</comp>
<comp lib="1" loc="(590,270)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="1" loc="(410,860)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="1" loc="(200,860)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="1" loc="(480,270)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="3" loc="(280,700)" name="Adder">
  <a name="width" val="1" />
</comp>
<comp lib="1" loc="(520,650)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="0" loc="(700,630)" name="Constant">
  <a name="value" val="0x0" />
</comp>
<comp lib="3" loc="(810,500)" name="Adder">
  <a name="width" val="1" />
</comp>
<comp lib="1" loc="(700,270)" name="AND Gate">
  <a name="facing" val="south" />
</comp>
<comp lib="0" loc="(610,820)" name="Constant">

```



```

    <a name="value" val="0x0"/>
  </comp>
  <comp lib="1" loc="(630,440)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(410,650)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="0" loc="(160,60)" name="Splitter">
    <a name="fanout" val="4"/>
    <a name="incoming" val="4"/>
    <a name="appear" val="center"/>
  </comp>
  <comp lib="1" loc="(300,860)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="0" loc="(1410,890)" name="Pin">
    <a name="facing" val="west"/>
    <a name="output" val="true"/>
    <a name="width" val="8"/>
    <a name="labelloc" val="east"/>
  </comp>
  <comp lib="3" loc="(630,890)" name="Adder">
    <a name="width" val="1"/>
  </comp>
  <comp lib="1" loc="(510,440)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(640,650)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(300,650)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(850,270)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(530,860)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="1" loc="(400,440)" name="AND Gate">
    <a name="facing" val="south"/>
  </comp>
  <comp lib="3" loc="(470,890)" name="Adder">
    <a name="width" val="1"/>
  </comp>
  <comp lib="0" loc="(40,110)" name="Pin">
    <a name="facing" val="south"/>
    <a name="width" val="4"/>
    <a name="tristate" val="false"/>
  </comp>
  <comp lib="3" loc="(220,910)" name="Adder">
    <a name="width" val="1"/>
  </comp>

```

```
<comp lib="0" loc="(100,60)" name="Pin">
  <a name="width" val="4"/>
  <a name="tristate" val="false"/>
</comp>
<comp lib="3" loc="(710,700)" name="Adder">
  <a name="width" val="1"/>
</comp>
<comp lib="0" loc="(230,440)" name="Constant">
  <a name="value" val="0x0"/>
</comp>
<comp lib="3" loc="(420,700)" name="Adder">
  <a name="width" val="1"/>
</comp>
<comp lib="1" loc="(750,440)" name="AND Gate">
  <a name="facing" val="south"/>
</comp>
<comp lib="3" loc="(630,500)" name="Adder">
  <a name="width" val="1"/>
</comp>
<comp lib="3" loc="(560,700)" name="Adder">
  <a name="width" val="1"/>
</comp>
</circuit>
</project>
```