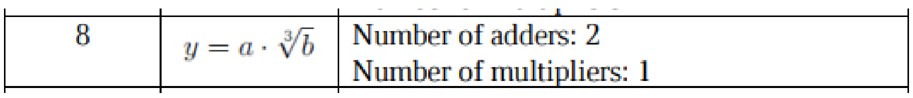
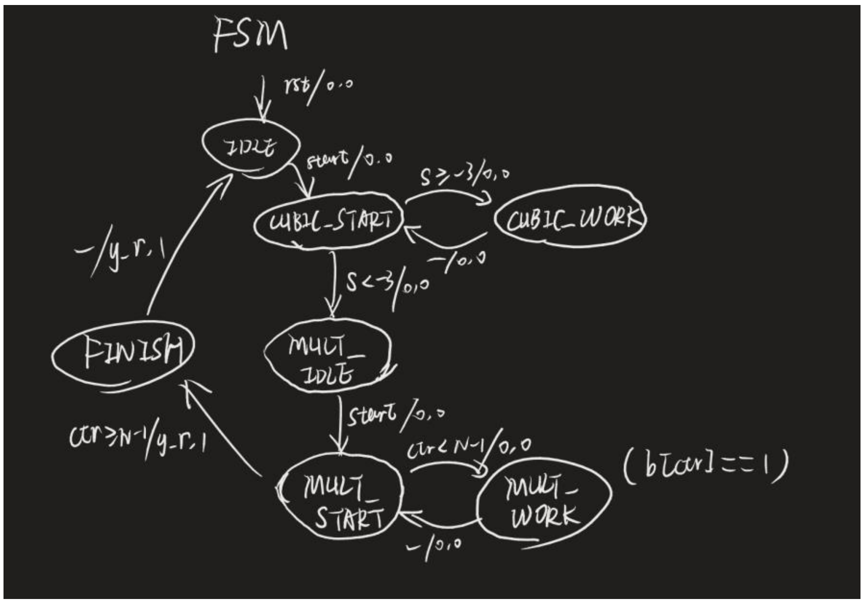
Functional Electronic Circuits Lab3

Student Name: CAO Xinyang
Student ID: 20321308
Variant: 8



8	$y = a \cdot \sqrt[3]{b}$	Number of adders: 2 Number of multipliers: 1

1. The picture with FSM

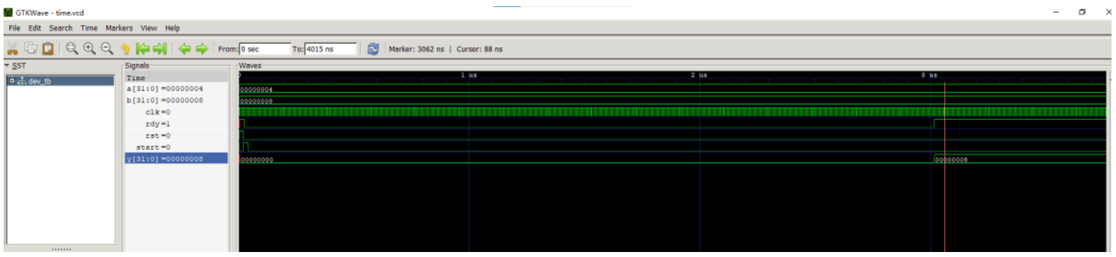


2. The table of transition

rst_x / S	rst	x0:start	x1:s>=3	x2:s<-3	x3:x>=b	x4:ctr<N-1	x5:ctr>=N-1	x6:(b[ctr]==1)
IDLE =S0	S0/0	S1/0	-	-	-	-	-	-
CUBIC_START =S1	S0/0	-	S2/0	S3/0	-	-	-	-
CUBIC_WORK =S2	S0/0	-	-	-	S1/0	-	-	-
MULT_IDLE =S3	S0/0	S4/0	-	-	-	-	-	-
MULT_START =S4	S0/0	-	-	-	-	S5/0	S6/y_r	-
MULT_WORK =S5	S0/0	-	-	-	-	-	-	S4/0
FINISH =S6	S0/0	S0/y_r	S0/y_r	S0/y_r	S0/y_r	S0/y_r	S0/y_r	S0/y_r

3. The timing diagram with simulation results

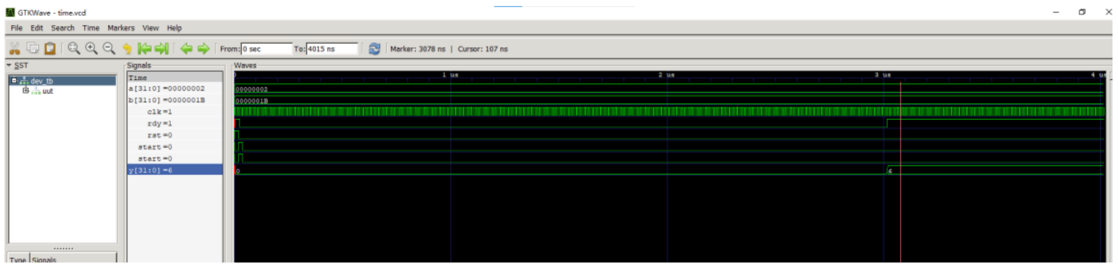
a = 4, b = 8 ➔ y = 8



From: 0 sec To: 4015 ns Marker: 3062 ns | Cursor: 88 ns

```
Time
a[31:0] =00000004
b[31:0] =00000008
clk =0
rdy =1
rst =0
start =0
y[31:0] =00000008
```

a = 2, b = 27 ➔ y = 6



```
Time
a[31:0] =00000002
b[31:0] =0000001B
clk =1
rdy =1
rst =0
start =0
start =0
y[31:0] =6
```

4. Code of the testbench and the device.

dev.v

```
`include "adder.v"
`include "mult.v"
module dev (
    input clk_i,
    input rst_i,

    input [31:0] x_bi,
    input [31:0] y_bi,
    input start_i,

    output reg [31:0] y_bo,
    output reg rdy_o
);

reg signed [31:0] a1_op1, a1_op2;
reg signed [31:0] a2_op1, a2_op2;
wire signed [31:0] a1_res, a2_res;

adder a1(
    .a_bi(a1_op1),
    .b_bi(a1_op2),
    .c_bo(a1_res)
);

adder a2(
    .a_bi(a2_op1),
    .b_bi(a2_op2),
    .c_bo(a2_res)
);

reg [31:0] m1_op1, m1_op2, y1;
reg m1_start;
```

```
wire [15:0] m1_res16;
wire [31:0] m1_res = {16'h0, m1_res16};

mult m1 (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .a_bi(m1_op1[7:0]),
    .b_bi(m1_op2[7:0]),
    .start_i(m1_start),
    .busy_o(m1_busy),
    .y_bo(m1_res16)
);

// FSM control unit code

localparam N = 32;

localparam IDLE              = 0;
localparam CMP_S             = 1;
localparam CALC_MULT_S1_START = 2;
localparam CALC_MULT_S1       = 3;
localparam CALC_MULT_S2_START = 4;
localparam CALC_MULT_S2       = 5;
localparam CALC_B            = 6;
localparam CMP_B             = 7;
localparam CALC_S            = 8;
localparam CALC_MULT_S3      = 9;
localparam FINISH_START       = 10;
localparam FINISH            = 11;

reg [3:0] state_r;
reg [31:0] x_r;
reg signed [31:0] s_r;
reg [31:0] y_r;
reg [31:0] b_r;

always@(posedge clk_i)
    if(rst_i) begin
        x_r <= 0;
        m1_op1 <= 0;
        m1_op2 <= 0;
        m1_start <= 0;
        a1_op1 <= 0;
        a1_op2 <= 0;
        a2_op1 <= 0;
        a2_op2 <= 0;
        y_bo   <= 0;
        y_r    <= 0;
        rdy_o  <= 1;
        s_r    <= 0;
        y1     <= 0;

        state_r <= IDLE;

    end else begin
        case(state_r)
            IDLE:
                if(start_i) begin
                    x_r <= x_bi;
                    y_r <= 0;
                    rdy_o <= 0;
                    b_r <= 0;
                    s_r <= 0;

                    // s = N-2
                    a1_op1 <= N;
                    a1_op2 <= -2;

                    state_r <= CMP_S;
                end
            CMP_S:
                begin
                    s_r <= a1_res;
                    if(a1_res > -3) begin
                        y_r = y_r << 1;

                        // 3*y
                        m1_op1 <= 3;
                        m1_op2 <= y_r;
                        m1_start <= 1;

                        // y+1
                        a1_op1 <= y_r;
                        a1_op2 <= 1;

                        state_r <= CALC_MULT_S1_START;

                    end else begin
```

```
                        y1 <= y_r;
                        m1_start <= 0;
                        state_r <= CALC_MULT_S3;
                    end
                end
            CALC_MULT_S3:
                if(m1_busy == 0) begin
                    m1_op1 <= y1;
                    m1_op2 <= y_bi;
                    m1_start <= 1;
                    state_r <= FINISH_START;
                end
            FINISH_START:
                begin
                    m1_start <= 0;
                    state_r <= FINISH;
                end
            FINISH:
                if(m1_busy == 0) begin
                    y_bo <= m1_res;
                    rdy_o <= 1;

                    state_r <= IDLE;
                end
            CALC_MULT_S1_START: //waiting 1 clk for starting of the mult
module
                begin
                    m1_start <= 0;
                    state_r <= CALC_MULT_S1;
                end
            CALC_MULT_S1:
                if(m1_busy == 0) begin
                    // m1_res = 3*y
                    // a1_res = y+1
                    // m1_res * a1_res = 3*y*(y+1)
                    m1_op1 <= m1_res;
                    m1_op2 <= a1_res;
                    m1_start <= 1;

                    state_r <= CALC_MULT_S2_START;
                end
            CALC_MULT_S2_START: //waiting 1 clk for starting of the mult
module
                begin
```

```verilog
                m1_start <= 0;
                state_r <= CALC_MULT_S2;
            end
        CALC_MULT_S2:
            if(m1_busy == 0) begin
                // 3*y(y+1)+1
                a1_op1 <= m1_res;
                a1_op2 <= 1;

                state_r <= CALC_B;
            end
        CALC_B:
            begin
                b_r <= a1_res << s_r;
                state_r <= CMP_B;
            end
        CMP_B:
            if(x_r >= b_r) begin
                // x-b
                a1_op1 <= x_r;
                a1_op2 <= -b_r;

                // y+1
                a2_op1 <= y_r;
                a2_op2 <= 1;

                state_r <= CALC_S;

            end else begin
                state_r <= CALC_S;
            end
        CALC_S:
            begin
                if(x_r >= b_r) begin
                    //x = x-b
                    x_r <= a1_res;

                    //y = y+1
                    y_r <= a2_res;
                end

                // s-3
                a1_op1 <= s_r;
                a1_op2 <= -3;
```

```
                    state_r <= CMP_S;
                end
            endcase
        end

endmodule
```

dev_tb.v

```
`include "dev.v"
`timescale 1ns/1ps

module dev_tb;


reg [31:0] a,b;
reg clk, rst;
reg start;

wire rdy;
wire [31:0] y;

dev uut(
    .clk_i(clk),
    .rst_i(rst),

    .x_bi(b),
    .y_bi(a),
    .start_i(start),

    .y_bo(y),
    .rdy_o(rdy)
);

always #5 clk = ~clk;

initial begin

    $dumpfile("time.vcd");
    $dumpvars(0, dev_tb);

    clk   = 0;
    rst   = 1;
    a     = 4;
    b     = 8;
    start = 0;

    #20
    rst   = 0;
    start = 1;
    #20
    start = 0;

    @(posedge rdy);
    #1000
    $finish;

end



endmodule
```

adder.v

```
module adder (
    input [31:0] a_bi,
    input [31:0] b_bi,

    output [31:0] c_bo
);

assign c_bo = a_bi + b_bi;

endmodule
```

mult.v

```
module mult(
    input clk_i,
    input rst_i,

    input [7:0] a_bi,
    input [7:0] b_bi,
    input start_i,

    output reg busy_o,
    output reg [15:0] y_bo
);

    localparam IDLE = 2'b00;
    localparam WORK = 2'b01;
    localparam END  = 2'b10;

    reg   [2:0] ctr;
    wire  [2:0] end_step;
    wire  [7:0] part_sum;
    wire [15:0] shifted_part_sum;
    reg   [7:0] a, b;
    reg  [15:0] part_res;
    reg   [1:0] state;

    assign part_sum = a & {8{b[ctr]}};
    assign shifted_part_sum = part_sum << ctr;

    always @(posedge clk_i)
        if (rst_i) begin
           ctr      <= 0;
           part_res <= 0;
           y_bo     <= 0;
           busy_o   <= 0;

           state <= IDLE;
        end else begin


            case (state)
                IDLE:
                    if (start_i) begin
                        state  <= WORK;

                        a        <= a_bi;
                        b        <= b_bi;
                        ctr      <= 0;
                        part_res <= 0;
                        busy_o   <= 1;
                    end
                WORK:
                    begin
                        if (ctr == 3'h7) begin
                           state  <= END;
                        end

                        part_res <= part_res + shifted_part_sum;
                        ctr <= ctr + 1;
                    end
                END:
```

```verilog
                    begin
                        y_bo   <= part_res;
                        busy_o <= 0;
                        state  <= IDLE;
                    end
                default: state <= IDLE;
            endcase
        end

endmodule
```