به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم‌های دیجیتال 1

استاد: دکتر نوابی

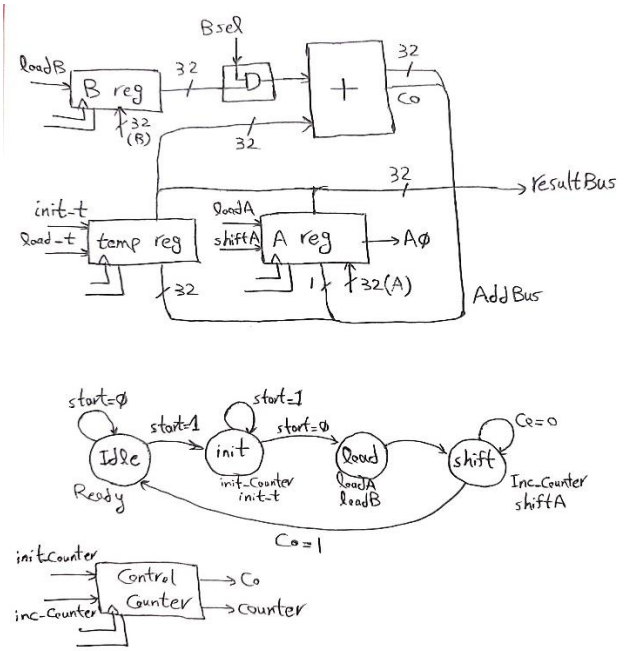
تمرین کامپیوتری شماره 6

فربد سیاه‌کلی

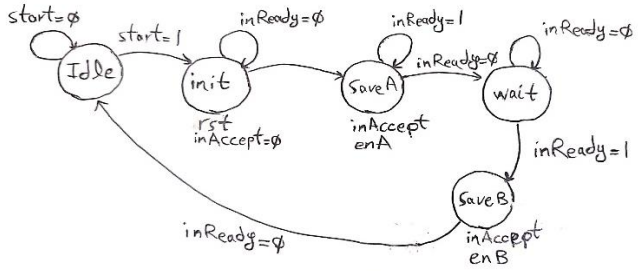
شماره دانشجویی: 810198510

فروردین 1400

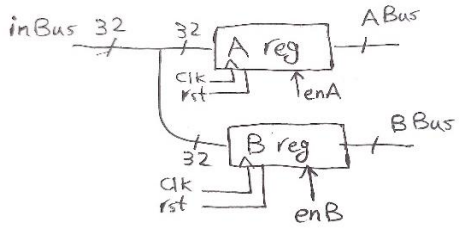
شکل Datapath و Controller برای Sequential Multiplier:



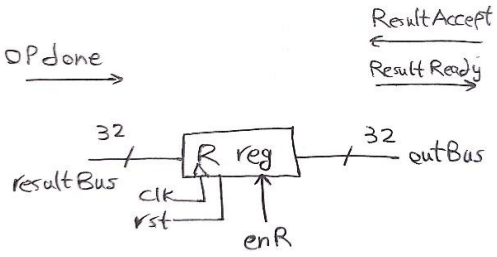
شکل Controller برای Input Wrapper:



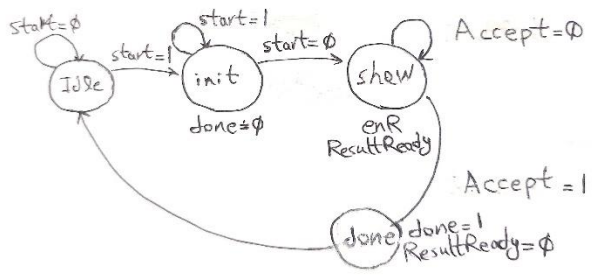
شکل Datapath برای Input Wrapper:



شکل Datapath برای Output Wrapper:



شکل Controller برای Output Wrapper:



کد وریلاگ Sequential Multiplier:

```
`timescale 1ns/1ns
module MULTDP(input clk, rst, loadA, loadB, loadP, shiftA, InitP, Bsel,
        input [23:0] Abus, BBus, output [47:0] ResultBus, output A0);
    reg [23:0] Areg, Breg, Preg;
    wire [23:0] B_AND;
    wire [25:0] AddBus;
    always @(posedge clk, posedge rst) begin
        if (rst) Breg <= 24'b0;
        else if(loadB) Breg <= BBus;
    end
    always @(posedge clk, posedge rst) begin
        if (rst) Preg <= 24'b0;
        else begin
            if(InitP) Preg <= 24'b0;
            else if (loadP) Preg <= AddBus [25:1];
        end
    end
    always @(posedge clk, posedge rst) begin
        if (rst) Areg <= 24'b0;
        else begin
            if(loadA) Areg <= Abus;
            else if (shiftA) Areg <= {AddBus[0], Areg[23:1]};
        end
    end
    assign B_AND = Bsel ? Breg : 24'b0;
    assign AddBus = B_AND + Preg;
    assign ResultBus = {Preg, Areg};
    assign A0 = Areg[0];
endmodule
```

```
module MULTCU(input clk, rst, start, A0, output reg loadA, shiftA, loadB, loadP, InitP, Bsel, ready);
    wire Co;
    reg Init_counter, Inc_counter;
    reg [1:0] pstate, nstate;
    reg [4:0] Count;
    parameter [1:0] Idle = 0 , Init = 1, load = 2, shift = 3;
    always @(pstate,start, A0, Co) begin
        nstate=0;
        {loadA, shiftA, loadB, loadP, InitP, Bsel, ready} =  7'b0;
        {Init_counter, Inc_counter} = 2'b0;
        case(pstate)
            Idle: begin nstate = start ? Init : Idle; ready = 1'b1; end
            Init: begin nstate = start ? Init : load; Init_counter = 1'b1; InitP = 1'b1; end
            load: begin nstate = shift ; loadA = 1'b1; loadB = 1'b1; end
            shift: begin nstate = Co ? Idle : shift; loadP = 1'b1; shiftA = 1'b1; Inc_counter = 1'b1; Bsel = A0; end
            default: nstate = Idle;
        endcase
    end
    always @(posedge clk, posedge rst) begin
        if (rst) pstate <= Idle;
        else pstate <= nstate;
    end
    always@(posedge clk, posedge rst) begin
        if (rst) Count <= 3'b0;
        else if(Init_counter) Count <= 4'b1000;
        else if (Inc_counter) Count <= Count +1;
    end
    assign Co = &Count;
endmodule
```

کد وریلاگ Input Wrapper:

```
module Input_wrapper(input clk, rst, inReady, inAccept, output reg enA, enB);
    integer counter = 0;
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            enA <= 1'b1;
            enB <= 1'b0;
            counter = 0;
        end
        else if(inReady & ~inAccept & ~counter) begin
            enA <= ~enA;
            enB <= ~enB;
            counter = 1;
        end
    end

endmodule

module register32(input [31:0] Parallel_in, input clk, rst, enable, output reg [31:0] parallel_out, input Ready, output reg Accepted);
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            parallel_out <= 32'd0;
            Accepted <= 0;
        end
        else if(Ready & enable) begin
            parallel_out <= Parallel_in;
            Accepted = 1;
        end
        else Accepted <= 0;
    end
endmodule
```

کد وریلاگ Output Wrapper:

```
module Output_Wrapper(input clk, rst, input [9:0] Resultexpfinal, input [22:0] product_mantissa, input Sign, doneMul,
                      output reg [31:0] ResultBus, input resultaccepted, output reg resultready, Operationdone, enR);
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            resultready <= 0;
            Operationdone <= 0;
        end
        else if(resultaccepted) begin
            resultready <= 0;
            Operationdone = 1;
        end
        else if(doneMul & ~Operationdone) begin
            enR = 1;
            resultready <= 1;
            ResultBus <= {Sign, Resultexpfinal[7:0], product_mantissa};
        end
        else resultready <= 0;
    end
endmodule
```

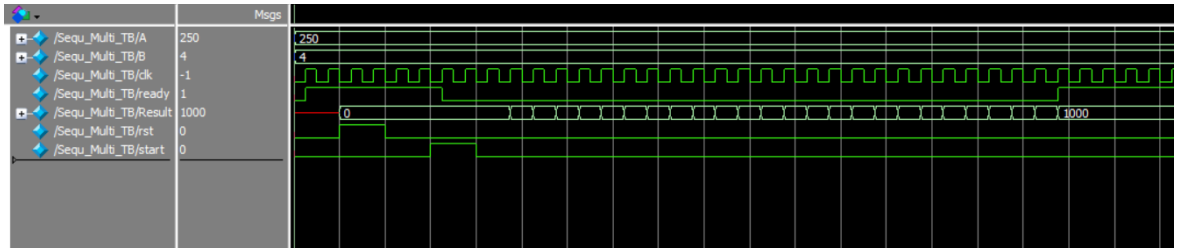
کد وریلاگ Top Level:

```
module Multiplier_32bit_TOP_PRE(input clk, rst, startMul, input [31:0] IN, output [31:0] ResultBusOut,
                                input inReady, output Accept, input resultaccepted, output resultready);
    wire A0, loadA, shiftA, loadB, loadP, InitP, Bsel;
    wire enA, enB, Accept1, Accept2, Co, Co1, Sign, Operationdone, enR;
    wire [47:0] ResultMul, product_normalised;
    wire [22:0] product_mantissa;
    wire [31:0] A, B, ResultBus;
    wire [9:0] Resultexp, Resultadder, Resultexpfinal;
    Input_wrapper IW(clk, rst, inReady, Accept, enA, enB);
    register32 Areg(IN, clk, rst, enA, A, inReady, Accept1);
    register32 Breg(IN, clk, rst, enB, B, inReady, Accept2);
    assign Accept = Accept1 | Accept2;
    MULTDP dp(clk, rst, loadA, loadB, loadP, shiftA, InitP, Bsel, {1'b1,A[22:0]}, {1'b1,B[22:0]}, ResultMul, A0);
    MULTCU cu(clk, rst, startMul, A0, loadA, shiftA, loadB, loadP, InitP, Bsel, doneMul);
    adder8bit add(A[30:23], B[30:23], 1'b0, Resultadder, Co);
    adder10bit subtract(Resultadder, {10'b1000000010}, 1'b0, Resultexp, Co1);
    SignDetector signdetect(A[31], B[31], Sign);
    Normalizer Norm(ResultMul, Resultexp, product_mantissa, Resultexpfinal);
    Output_Wrapper OW(clk, rst, Resultexpfinal, product_mantissa, Sign, doneMul, ResultBus, resultaccepted, resultready, Operationdone, enR);
    register32out Rreg(ResultBus, clk, rst, enR, ResultBusOut);
endmodule
```

تست بنچ مربوطه:

```
`timescale 1ns/1ns
module Multiplier_32bit_TB();
    reg clk=1'b0, rst = 0, start = 0, inReady = 0, resultAccepted = 0, inReady2 = 0, resultAccepted2 = 0;
    wire inAccept, resultReady, inAccept2, resultReady2;
    reg [31:0] inBus, outBus_pre, outBus_post;
    wire [31:0] out1, out2;
    Multiplier_32bit_TOP_PRE UUT1(clk, rst, start, inBus, out1, inReady, inAccept, resultAccepted, resultReady);
    Multiplier_32bit_TOP UUT2(clk, rst, start, inBus, out2, inReady2, inAccept2, resultAccepted2, resultReady2);
    always #5 clk <= ~clk;
    initial begin
        #20  rst=1;
        #20  rst=0;
        #20  inBus = 32'b10111110100110011001100110011010;
        #20  inReady = 1; inReady2 = 1;
             #20  if(inAccept) begin inReady = 0; end
            if(inAccept2) begin inReady2 = 0; end
        #20  inBus = 32'b01000011111110100010000000000000;
        #20  inReady = 1; inReady2 = 1;
             #20  if(inAccept) inReady = 0;
            if(inAccept2) inReady2 = 0;
        #20  start=1;
        #20  start=0;
        #300;
             #20 if(resultReady) begin
            outBus_pre = out1;
            resultAccepted = 1;
        end
            if(resultReady2) begin
            outBus_post = out2;
            resultAccepted2 = 1;
        end
             #30 if(~resultReady) resultAccepted = 0;
            if(~resultReady2) resultAccepted2 = 0;
        #30
        #0 $stop;
    end
endmodule
```

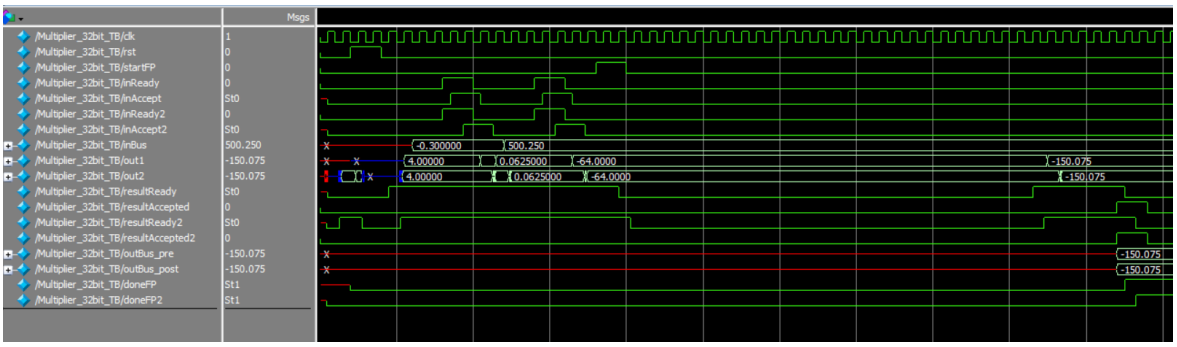
برای تست کردن ماژول ضرب Sequential، تست بنچی درست کردیم که درستی خروجی تایید شود:



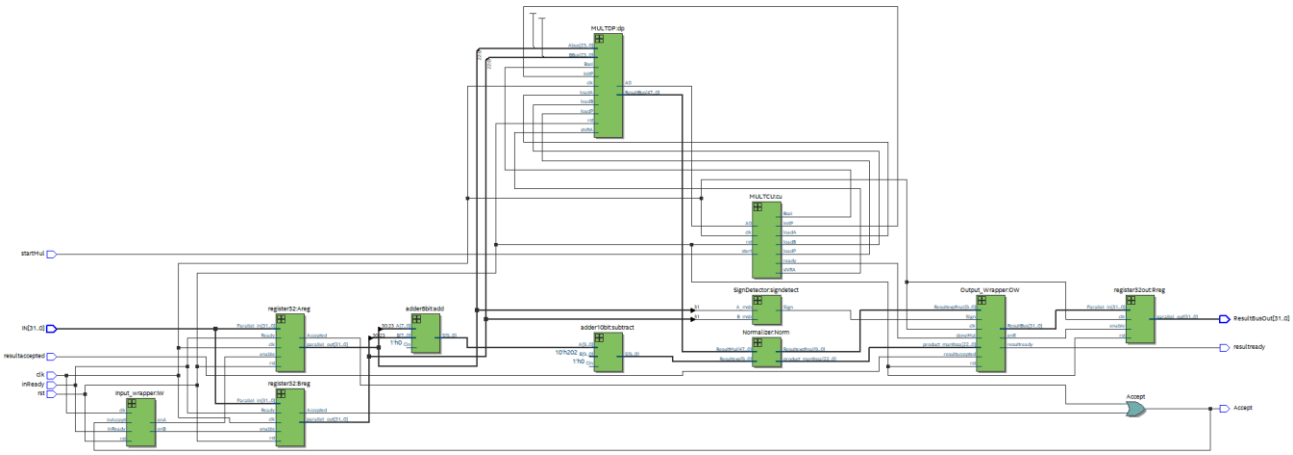
حال با استفاده از کوارتز، کد سنتز شده و خروجی‌های vo. و sdo. که شامل دیلی‌های ماژول است دریافت شده و در پایان کد اولیه و کد سنتز شده در کنار هم اجرا شده‌اند تا تفاوت قابل مشاهده باشد.

(از آنجا که در این شبیه‌سازی، مقادیر ورودی و خروجی رپرهای input و output همگی مشخص هستند، دیگر تست بنچی جدا برایشان نوشته نشد و همینجا کارکرد آنها را می‌توان بررسی کرد)

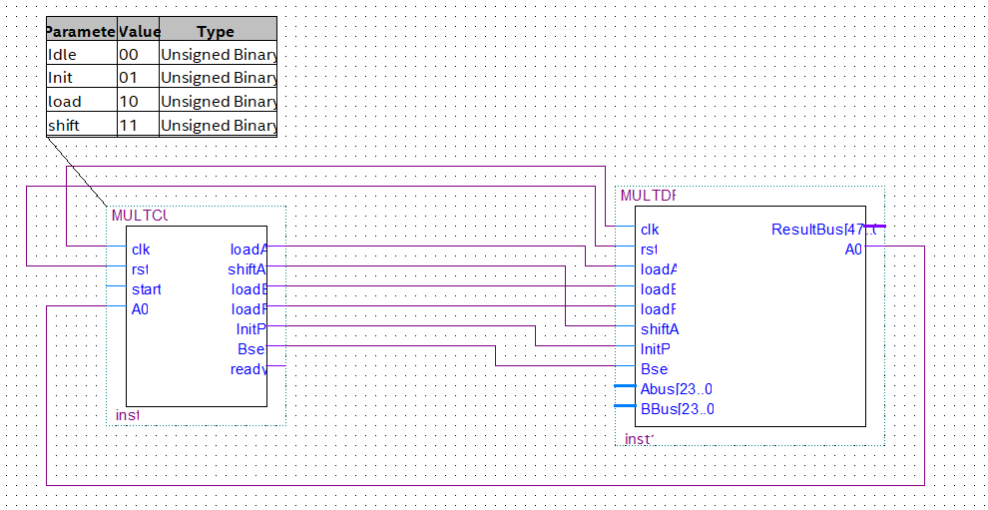
شکل موج خروجی:



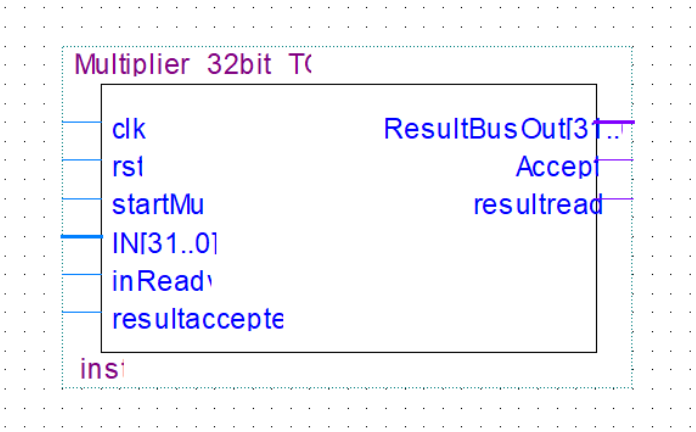
رسم RTL view ماژول:



بلاک دیاگرام Sequential Multiplier:



بلاک دیاگرام Top Level:



بلاک دیاگرام Wrappers: