`

UNIVERSITY OF TEHRAN

Report for Computer Assignment 6

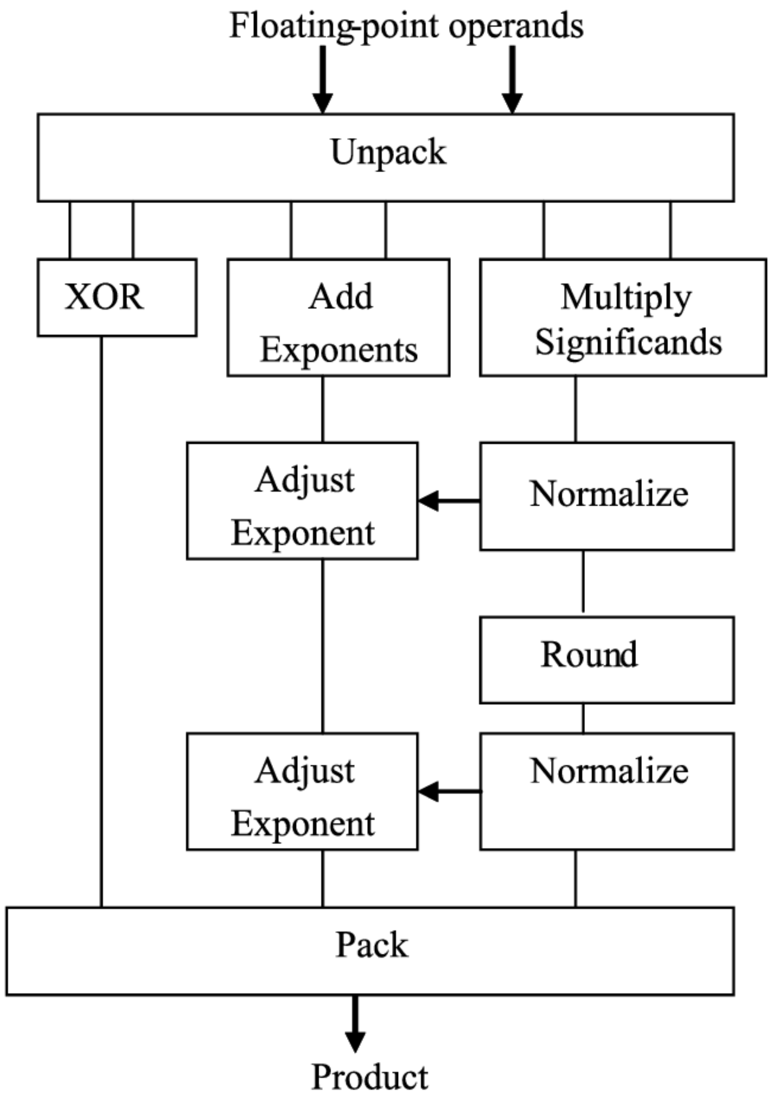
Hierarchical RTL Design

Instructor : Dr. Navabi

Danial Saeedi

`

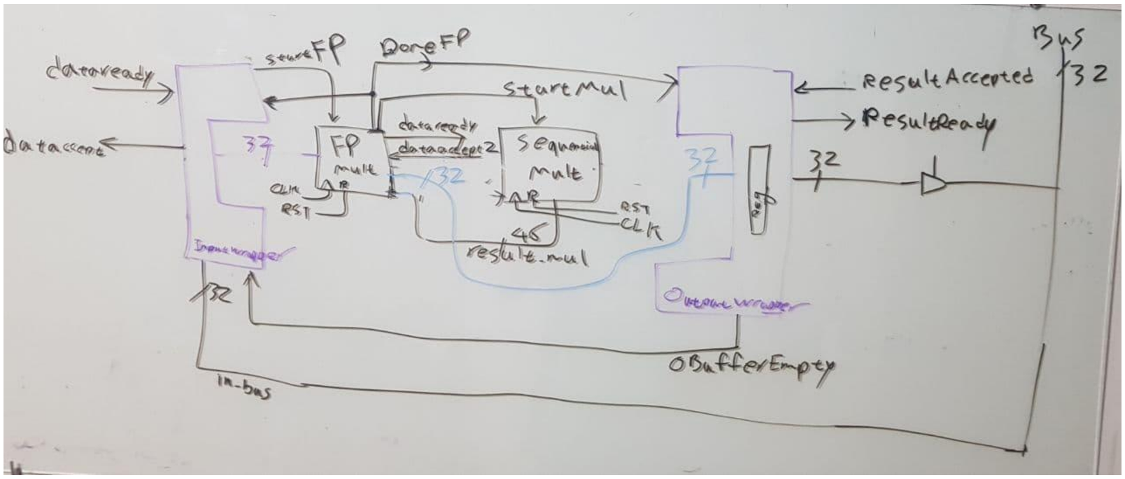
Algorithm of Floating-point multiplication



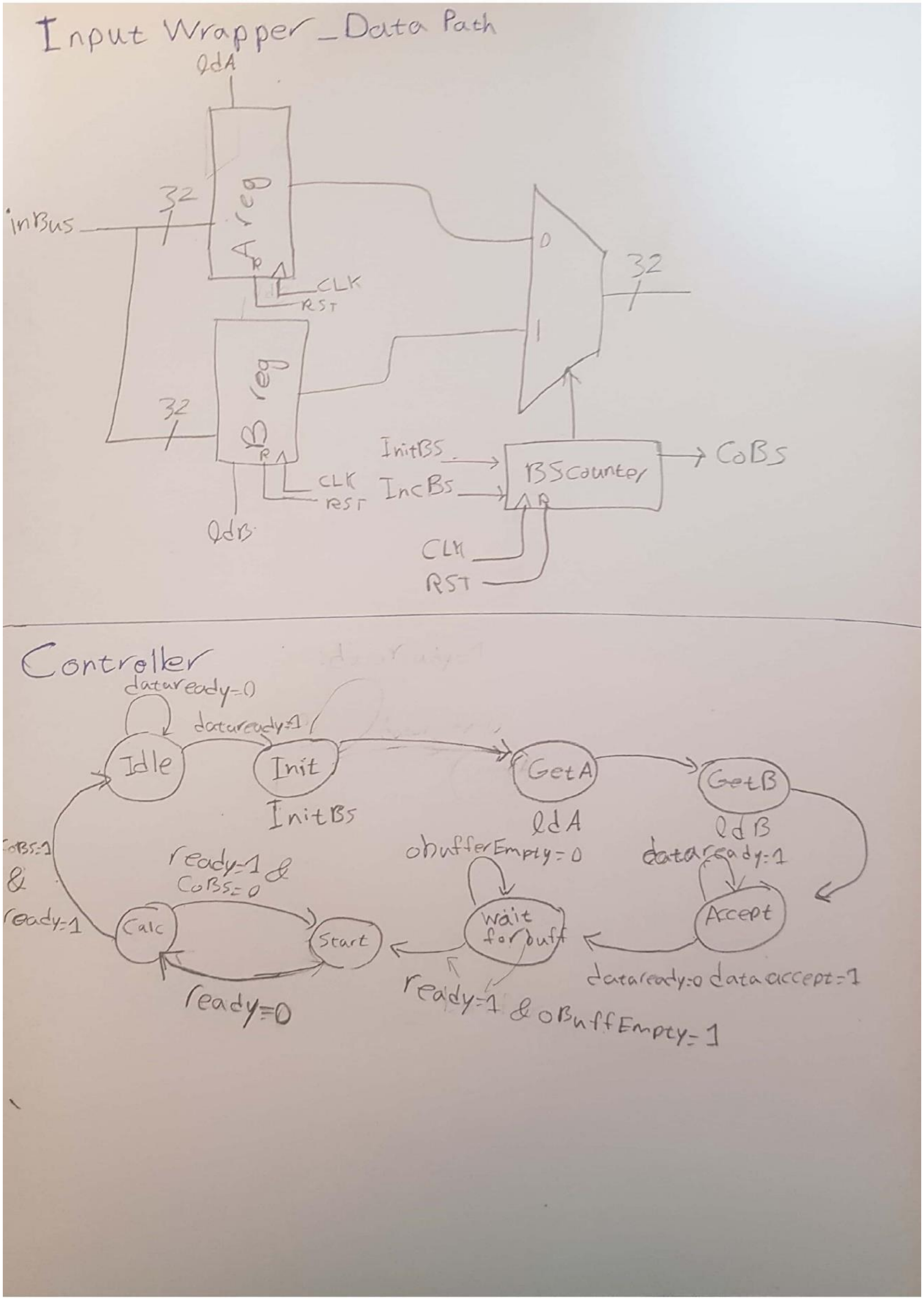
+

`

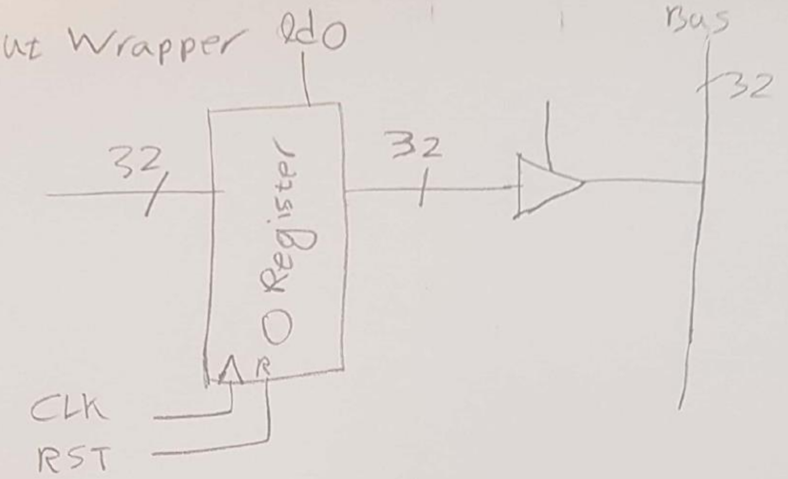
A. Show the block diagram of the entire circuit including the three parts, Wrappers, Floating Point, and the Sequential Multiplier.



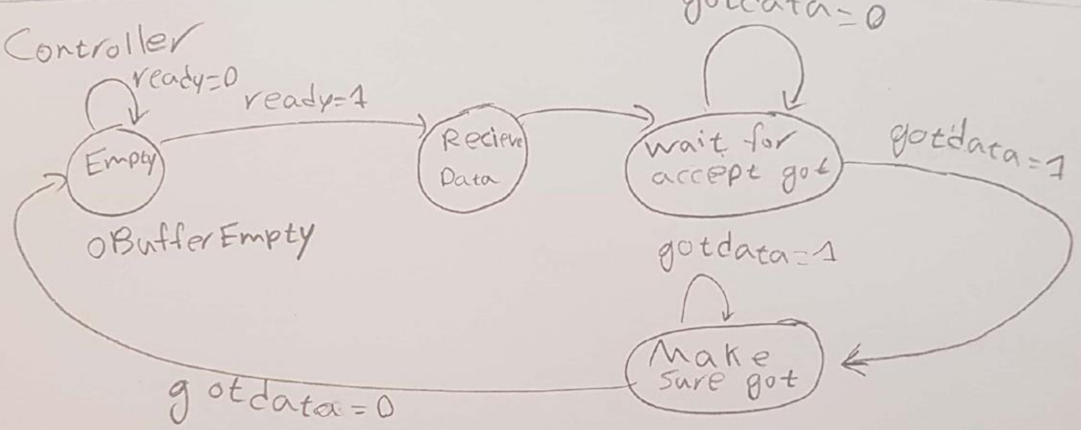
C. Show datapath and controller of the



Output Wrapper

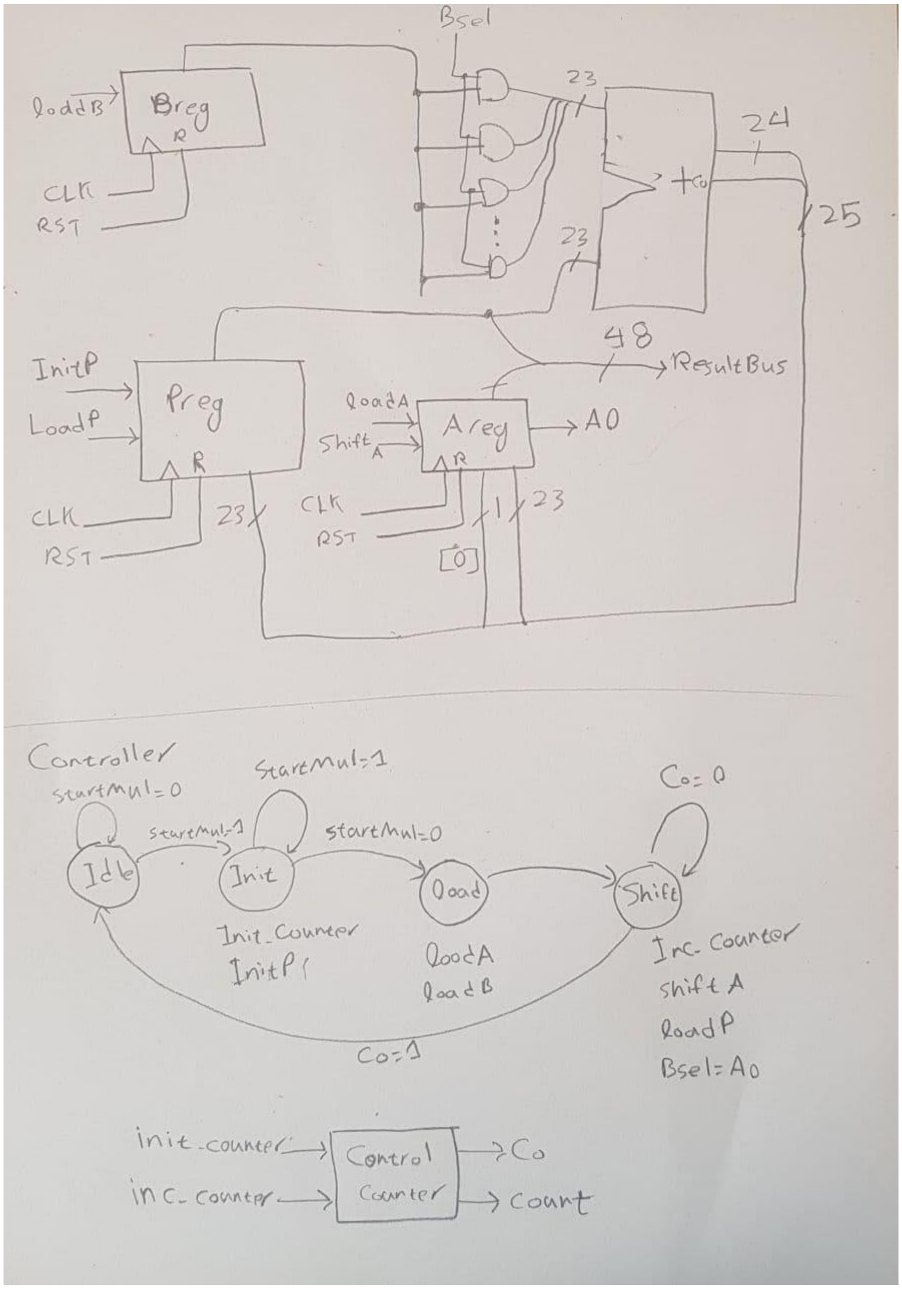


Controller



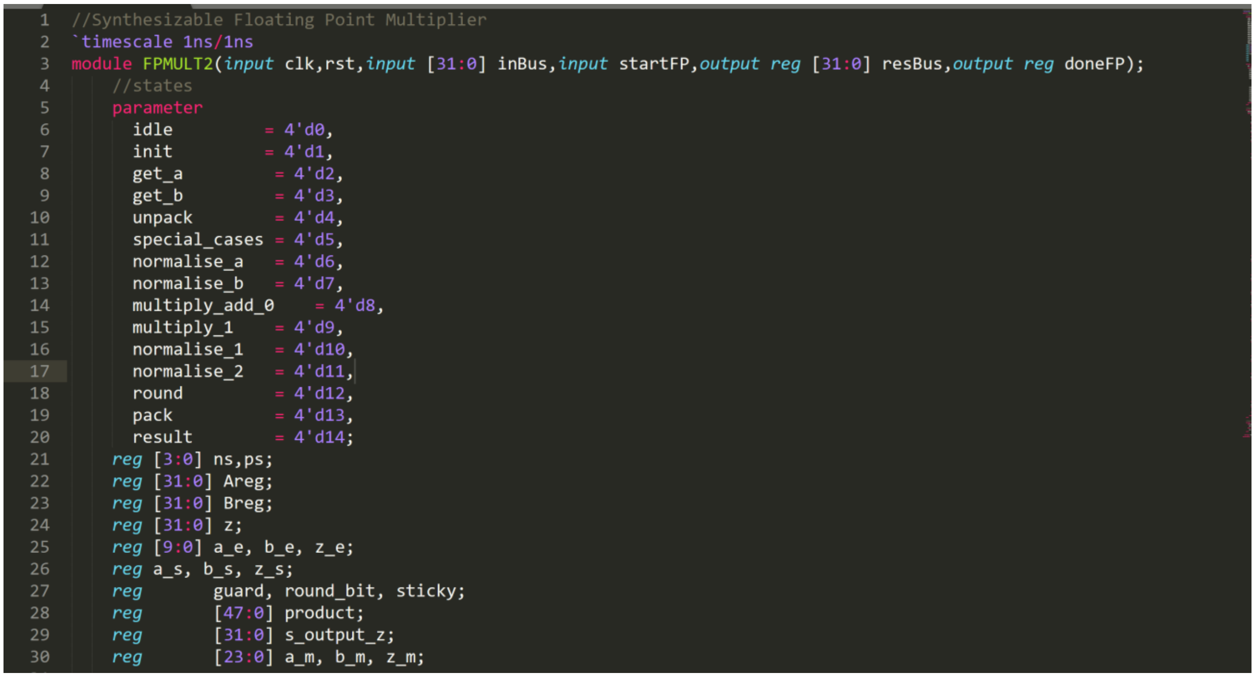
`

D. Show datapath and controller of the Sequential multiplication part

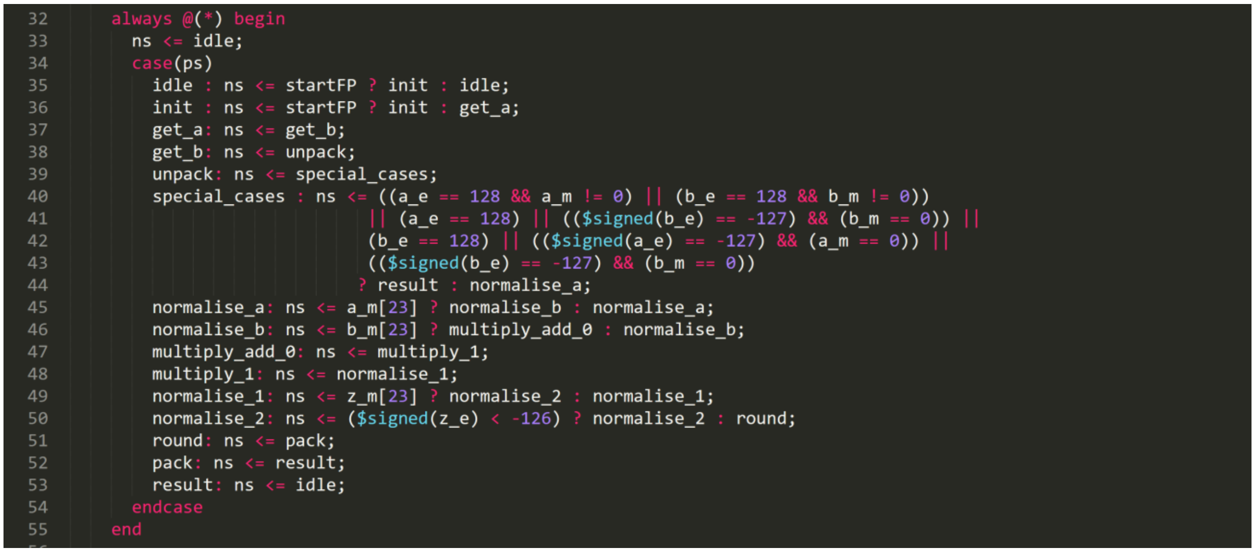


`

E. Show datapath and controller of the Sequential multiplication part

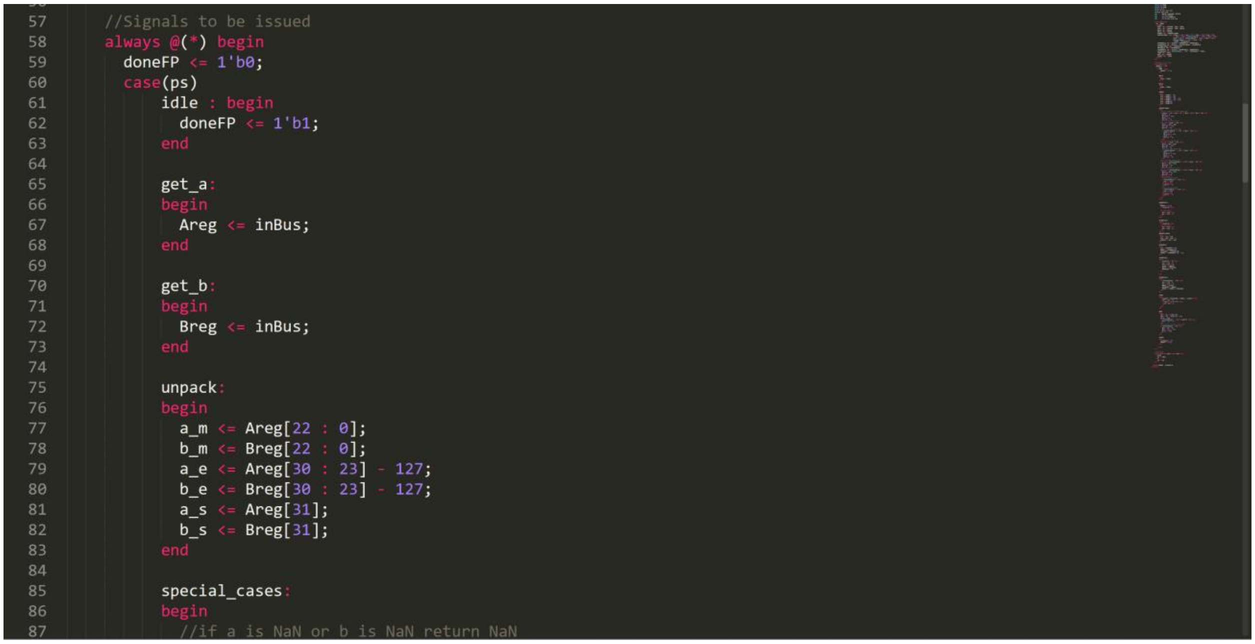


```
//Synthesizable Floating Point Multiplier
`timescale 1ns/1ns
module FPMULT2(input clk,rst,input [31:0] inBus,input startFP,output reg [31:0] resBus,output reg doneFP);
    //states
    parameter
      idle         = 4'd0,
      init         = 4'd1,
      get_a         = 4'd2,
      get_b         = 4'd3,
      unpack        = 4'd4,
      special_cases = 4'd5,
      normalise_a   = 4'd6,
      normalise_b   = 4'd7,
      multiply_add_0    = 4'd8,
      multiply_1    = 4'd9,
      normalise_1   = 4'd10,
      normalise_2   = 4'd11,
      round         = 4'd12,
      pack          = 4'd13,
      result        = 4'd14;
    reg [3:0] ns,ps;
    reg [31:0] Areg;
    reg [31:0] Breg;
    reg [31:0] z;
    reg [9:0] a_e, b_e, z_e;
    reg a_s, b_s, z_s;
    reg       guard, round_bit, sticky;
    reg       [47:0] product;
    reg       [31:0] s_output_z;
    reg       [23:0] a_m, b_m, z_m;
```

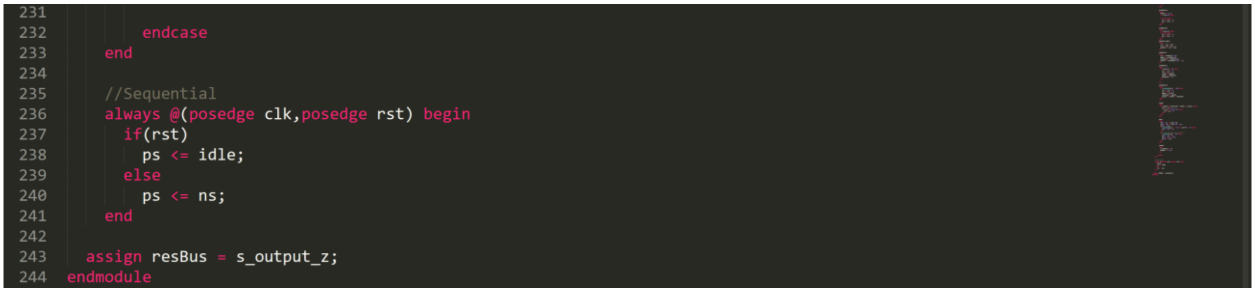


```
    always @(*) begin
      ns <= idle;
      case(ps)
        idle : ns <= startFP ? init : idle;
        init : ns <= startFP ? init : get_a;
        get_a: ns <= get_b;
        get_b: ns <= unpack;
        unpack: ns <= special_cases;
        special_cases : ns <= ((a_e == 128 && a_m != 0) || (b_e == 128 && b_m != 0))
                              || (a_e == 128) || (($signed(b_e) == -127) && (b_m == 0)) ||
                             (b_e == 128) || (($signed(a_e) == -127) && (a_m == 0)) ||
                             (($signed(b_e) == -127) && (b_m == 0))
                            ? result : normalise_a;
        normalise_a: ns <= a_m[23] ? normalise_b : normalise_a;
        normalise_b: ns <= b_m[23] ? multiply_add_0 : normalise_b;
        multiply_add_0: ns <= multiply_1;
        multiply_1: ns <= normalise_1;
        normalise_1: ns <= z_m[23] ? normalise_2 : normalise_1;
        normalise_2: ns <= ($signed(z_e) < -126) ? normalise_2 : round;
        round: ns <= pack;
        pack: ns <= result;
        result: ns <= idle;
      endcase
    end
```

`



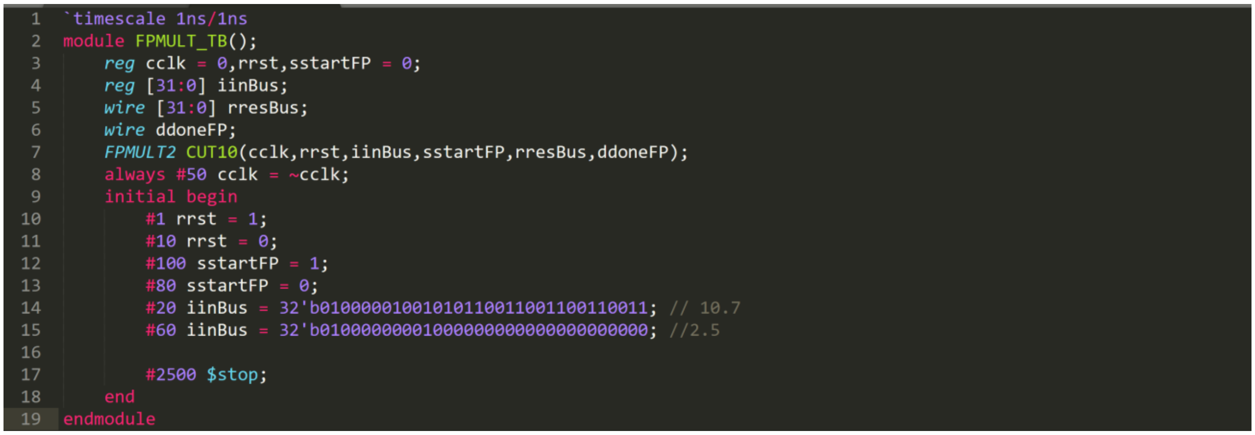
```
    //Signals to be issued
    always @(*) begin
      doneFP <= 1'b0;
      case(ps)
          idle : begin
            doneFP <= 1'b1;
          end

          get_a:
          begin
            Areg <= inBus;
          end

          get_b:
          begin
            Breg <= inBus;
          end

          unpack:
          begin
            a_m <= Areg[22 : 0];
            b_m <= Breg[22 : 0];
            a_e <= Areg[30 : 23] - 127;
            b_e <= Breg[30 : 23] - 127;
            a_s <= Areg[31];
            b_s <= Breg[31];
          end

          special_cases:
          begin
            //if a is NaN or b is NaN return NaN
```



```

        endcase
    end

    //Sequential
    always @(posedge clk,posedge rst) begin
      if(rst)
        ps <= idle;
      else
        ps <= ns;
    end

  assign resBus = s_output_z;
endmodule
```

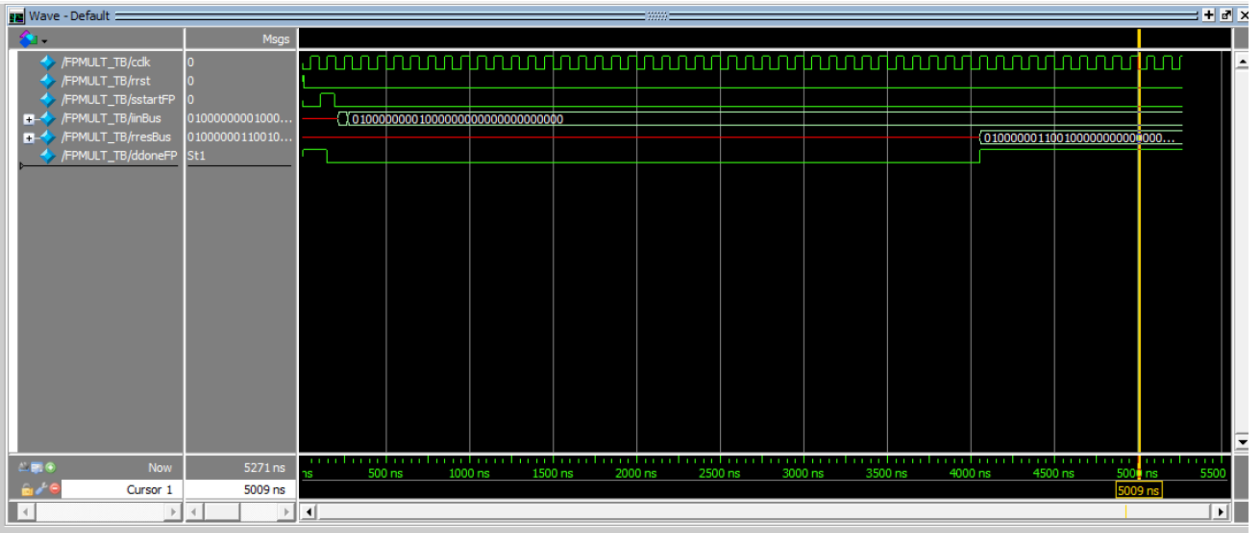
`

Testbench for FP Multiplier



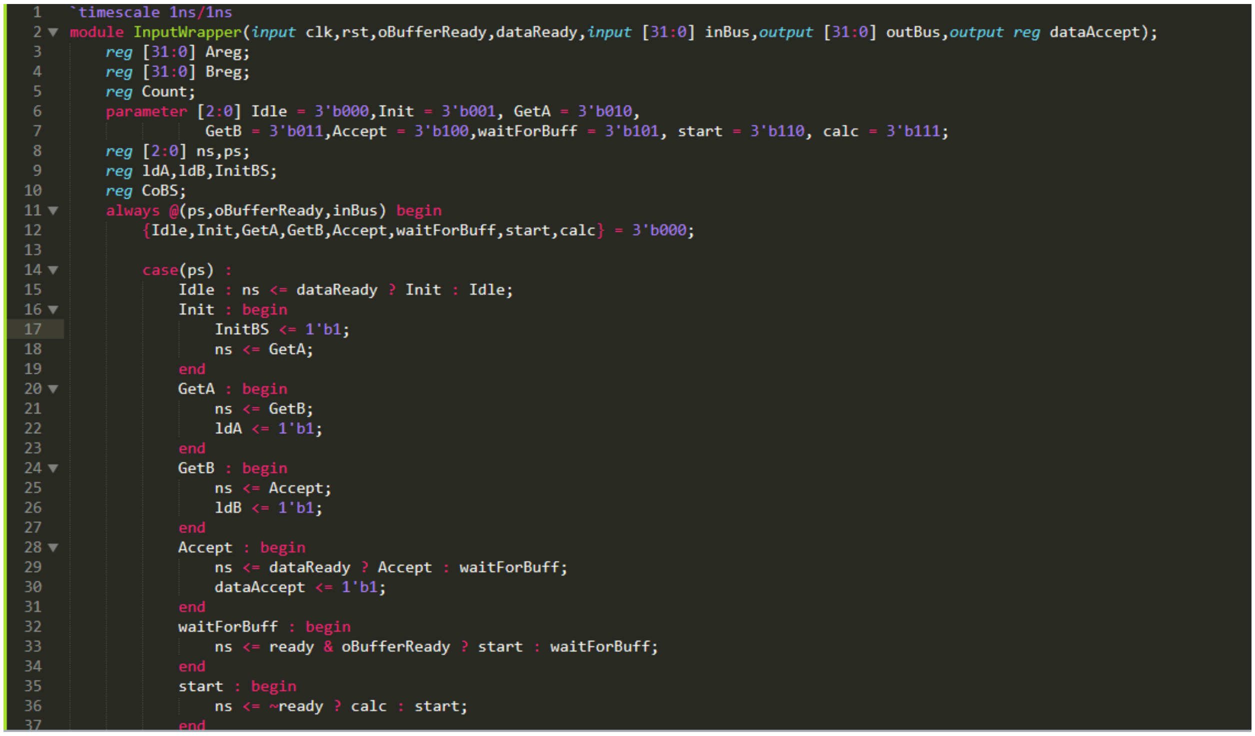
```verilog
`timescale 1ns/1ns
module FPMULT_TB();
    reg cclk = 0,rrst,sstartFP = 0;
    reg [31:0] iinBus;
    wire [31:0] rresBus;
    wire ddoneFP;
    FPMULT2 CUT10(cclk,rrst,iinBus,sstartFP,rresBus,ddoneFP);
    always #50 cclk = ~cclk;
    initial begin
        #1 rrst = 1;
        #10 rrst = 0;
        #100 sstartFP = 1;
        #80 sstartFP = 0;
        #20 iinBus = 32'b01000001001010110011001100110011; // 10.7
        #60 iinBus = 32'b01000000001000000000000000000000; //2.5

        #2500 $stop;
    end
endmodule
```

Simulation Result

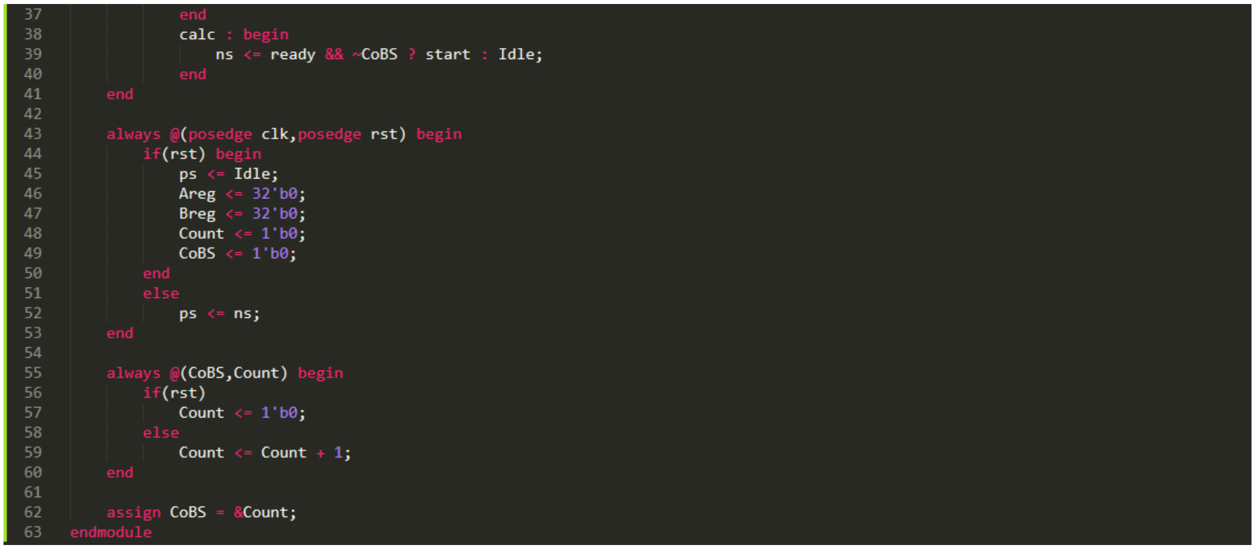


`

G. Write complete SystemVerilog description of the Wrapper circuits.

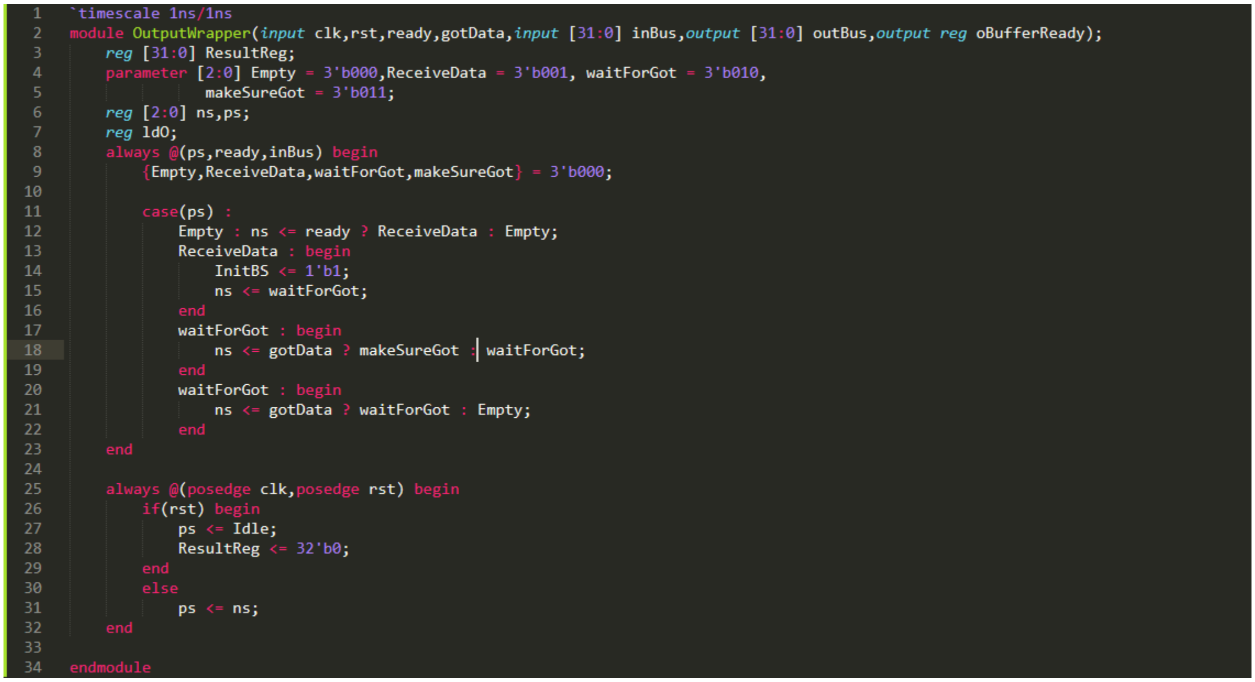


```verilog
`timescale 1ns/1ns
module InputWrapper(input clk,rst,oBufferReady,dataReady,input [31:0] inBus,output [31:0] outBus,output reg dataAccept);
    reg [31:0] Areg;
    reg [31:0] Breg;
    reg Count;
    parameter [2:0] Idle = 3'b000,Init = 3'b001, GetA = 3'b010,
                GetB = 3'b011,Accept = 3'b100,waitForBuff = 3'b101, start = 3'b110, calc = 3'b111;
    reg [2:0] ns,ps;
    reg ldA,ldB,InitBS;
    reg CoBS;
    always @(ps,oBufferReady,inBus) begin
        {Idle,Init,GetA,GetB,Accept,waitForBuff,start,calc} = 3'b000;

        case(ps) :
            Idle : ns <= dataReady ? Init : Idle;
            Init : begin
                InitBS <= 1'b1;
                ns <= GetA;
            end
            GetA : begin
                ns <= GetB;
                ldA <= 1'b1;
            end
            GetB : begin
                ns <= Accept;
                ldB <= 1'b1;
            end
            Accept : begin
                ns <= dataReady ? Accept : waitForBuff;
                dataAccept <= 1'b1;
            end
            waitForBuff : begin
                ns <= ready & oBufferReady ? start : waitForBuff;
            end
            start : begin
                ns <= ~ready ? calc : start;
```



```verilog
            end
            calc : begin
                ns <= ready && ~CoBS ? start : Idle;
            end
    end

    always @(posedge clk,posedge rst) begin
        if(rst) begin
            ps <= Idle;
            Areg <= 32'b0;
            Breg <= 32'b0;
            Count <= 1'b0;
            CoBS <= 1'b0;
        end
        else
            ps <= ns;
    end

    always @(CoBS,Count) begin
        if(rst)
            Count <= 1'b0;
        else
            Count <= Count + 1;
    end

    assign CoBS = &Count;
endmodule
```

`

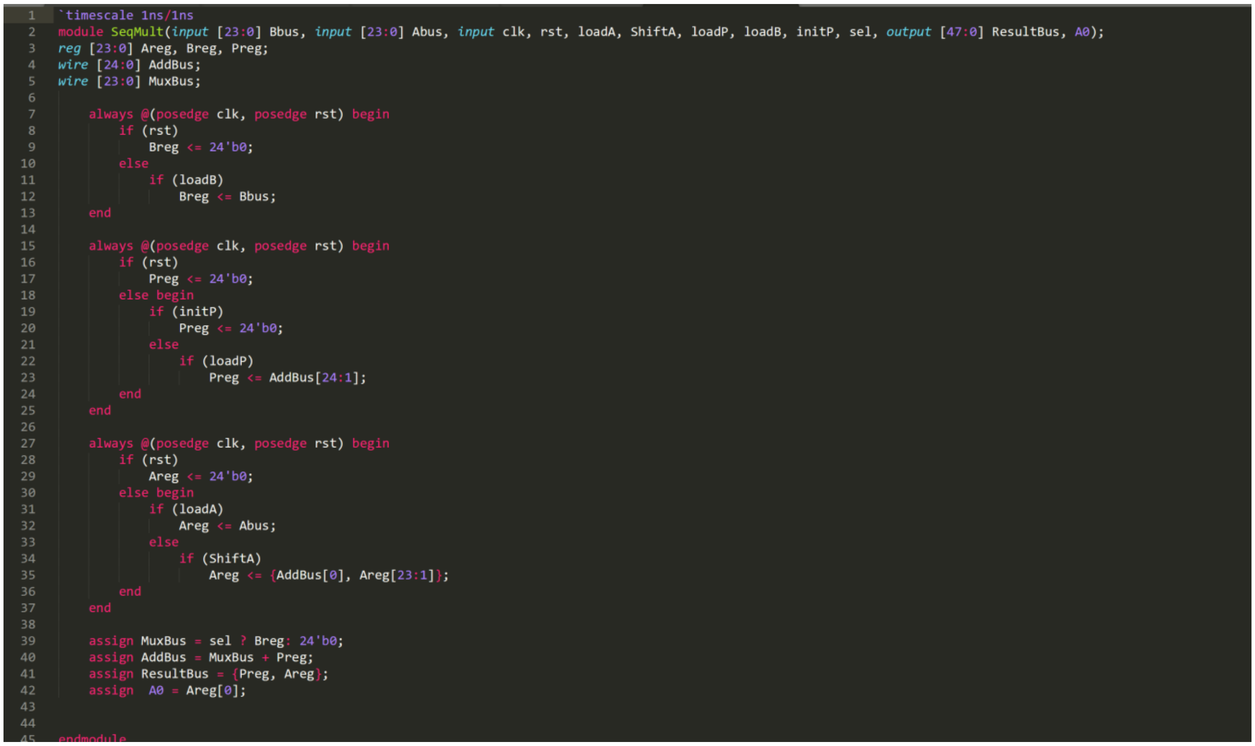


```
`timescale 1ns/1ns
module OutputWrapper(input clk,rst,ready,gotData,input [31:0] inBus,output [31:0] outBus,output reg oBufferReady);
    reg [31:0] ResultReg;
    parameter [2:0] Empty = 3'b000,ReceiveData = 3'b001, waitForGot = 3'b010,
                makeSureGot = 3'b011;
    reg [2:0] ns,ps;
    reg ldO;
    always @(ps,ready,inBus) begin
        {Empty,ReceiveData,waitForGot,makeSureGot} = 3'b000;

        case(ps) :
            Empty : ns <= ready ? ReceiveData : Empty;
            ReceiveData : begin
                InitBS <= 1'b1;
                ns <= waitForGot;
            end
            waitForGot : begin
                ns <= gotData ? makeSureGot : waitForGot;
            end
            waitForGot : begin
                ns <= gotData ? waitForGot : Empty;
            end
    end

    always @(posedge clk,posedge rst) begin
        if(rst) begin
            ps <= Idle;
            ResultReg <= 32'b0;
        end
        else
            ps <= ns;
    end

endmodule
```

`

G. Write complete SystemVerilog description of the Sequential multiplier part. Write a testbench and test this part.

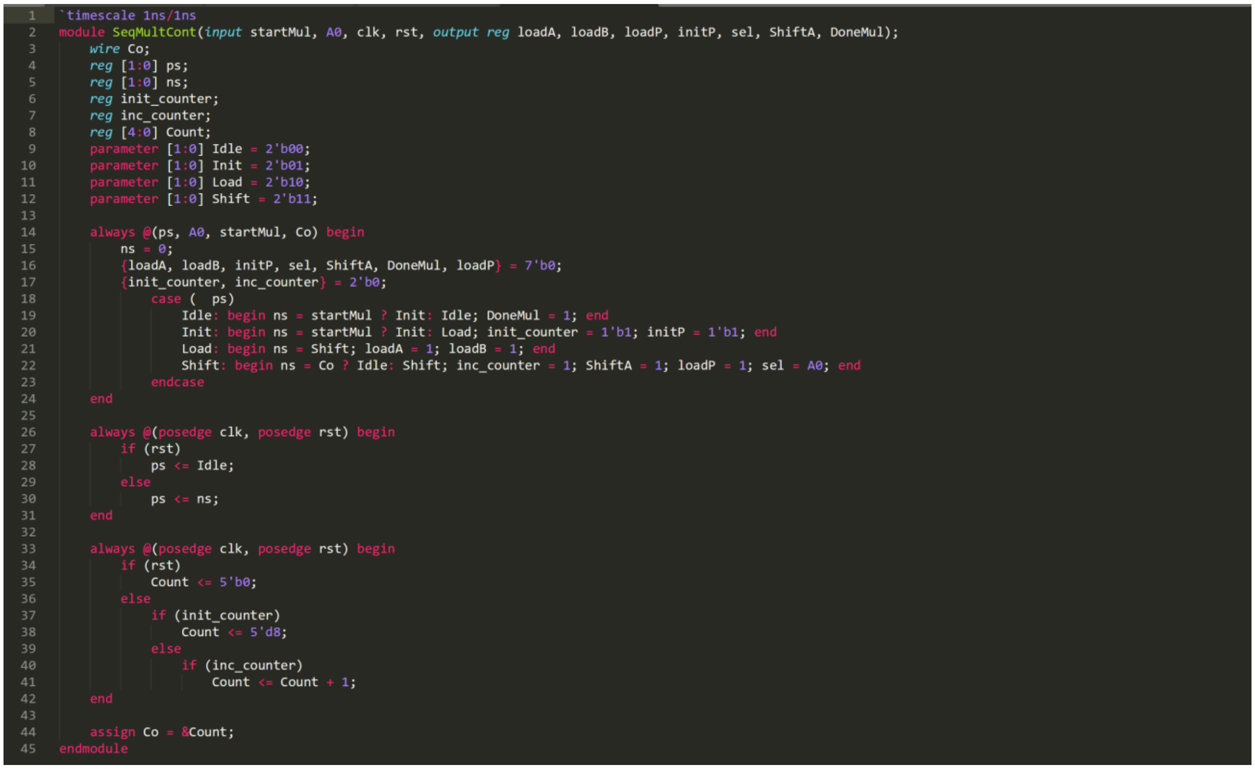
Datapath Verilog



```
`timescale 1ns/1ns
module SeqMult(input [23:0] Bbus, input [23:0] Abus, input clk, rst, loadA, ShiftA, loadP, loadB, initP, sel, output [47:0] ResultBus, A0);
reg [23:0] Areg, Breg, Preg;
wire [24:0] AddBus;
wire [23:0] MuxBus;

    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[24:1];
        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 MuxBus = sel ? Breg: 24'b0;
    assign AddBus = MuxBus + Preg;
    assign ResultBus = {Preg, Areg};
    assign  A0 = Areg[0];


endmodule
```

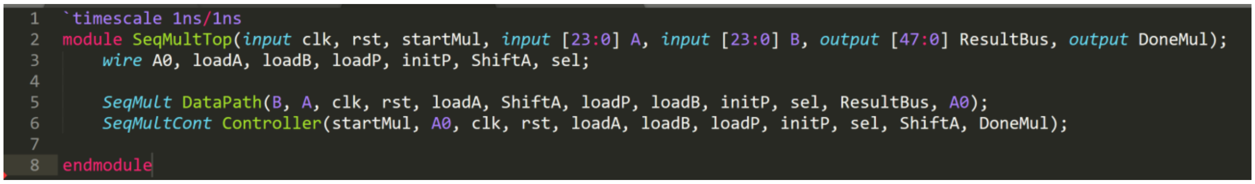
`

Controller



```verilog
`timescale 1ns/1ns
module SeqMultCont(input startMul, A0, clk, rst, output reg loadA, loadB, loadP, initP, sel, ShiftA, DoneMul);
    wire Co;
    reg [1:0] ps;
    reg [1:0] ns;
    reg init_counter;
    reg inc_counter;
    reg [4:0] Count;
    parameter [1:0] Idle = 2'b00;
    parameter [1:0] Init = 2'b01;
    parameter [1:0] Load = 2'b10;
    parameter [1:0] Shift = 2'b11;

    always @(ps, A0, startMul, Co) begin
        ns = 0;
        {loadA, loadB, initP, sel, ShiftA, DoneMul, loadP} = 7'b0;
        {init_counter, inc_counter} = 2'b0;
            case (  ps)
                Idle: begin ns = startMul ? Init: Idle; DoneMul = 1; end
                Init: begin ns = startMul ? Init: Load; init_counter = 1'b1; initP = 1'b1; end
                Load: begin ns = Shift; loadA = 1; loadB = 1; end
                Shift: begin ns = Co ? Idle: Shift; inc_counter = 1; ShiftA = 1; loadP = 1; sel = A0; end
            endcase
    end

    always @(posedge clk, posedge rst) begin
        if (rst)
            ps <= Idle;
        else
            ps <= ns;
    end

    always @(posedge clk, posedge rst) begin
        if (rst)
            Count <= 5'b0;
        else
            if (init_counter)
                Count <= 5'd8;
            else
                if (inc_counter)
                    Count <= Count + 1;
    end

    assign Co = &Count;
endmodule
```

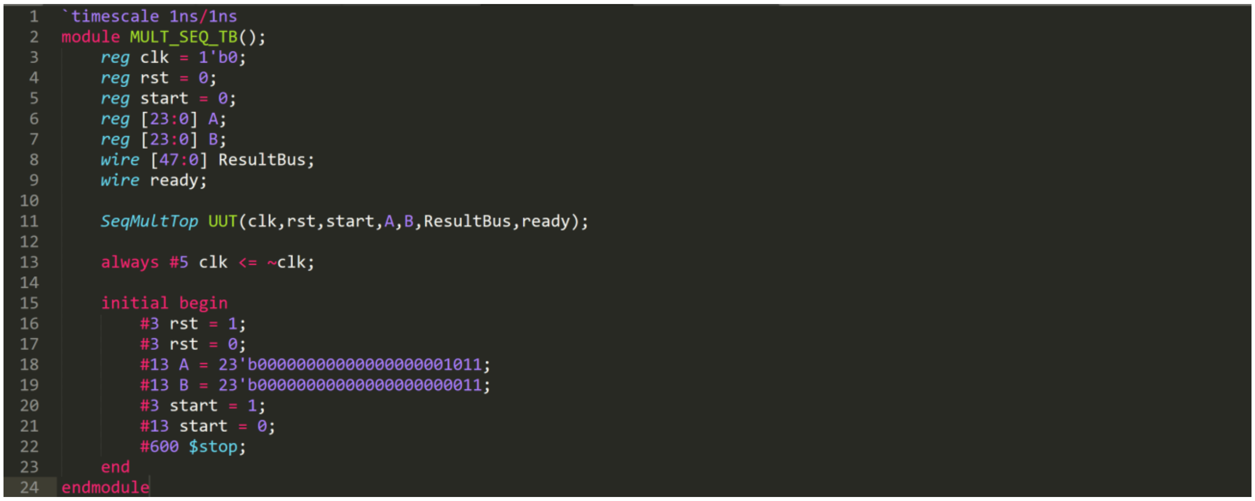
Top Level Module



```verilog
`timescale 1ns/1ns
module SeqMultTop(input clk, rst, startMul, input [23:0] A, input [23:0] B, output [47:0] ResultBus, output DoneMul);
    wire A0, loadA, loadB, loadP, initP, ShiftA, sel;

    SeqMult DataPath(B, A, clk, rst, loadA, ShiftA, loadP, loadB, initP, sel, ResultBus, A0);
    SeqMultCont Controller(startMul, A0, clk, rst, loadA, loadB, loadP, initP, sel, ShiftA, DoneMul);

endmodule
```

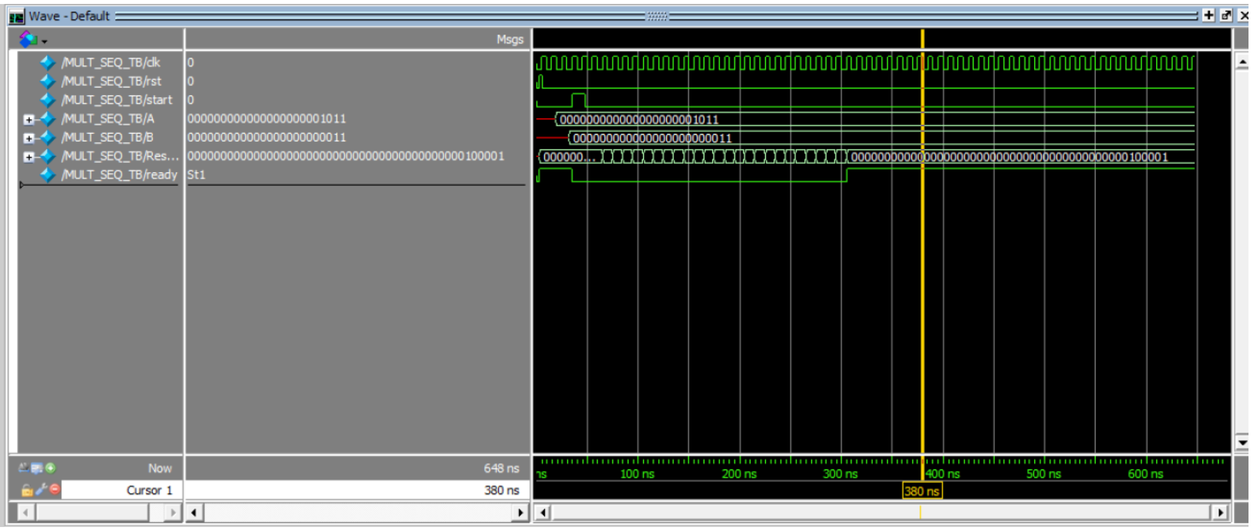
`

Testbench



```
`timescale 1ns/1ns
module MULT_SEQ_TB();
    reg clk = 1'b0;
    reg rst = 0;
    reg start = 0;
    reg [23:0] A;
    reg [23:0] B;
    wire [47:0] ResultBus;
    wire ready;

    SeqMultTop UUT(clk,rst,start,A,B,ResultBus,ready);

    always #5 clk <= ~clk;

    initial begin
        #3 rst = 1;
        #3 rst = 0;
        #13 A = 23'b00000000000000000001011;
        #13 B = 23'b00000000000000000000011;
        #3 start = 1;
        #13 start = 0;
        #600 $stop;
    end
endmodule
```

Simulation Result



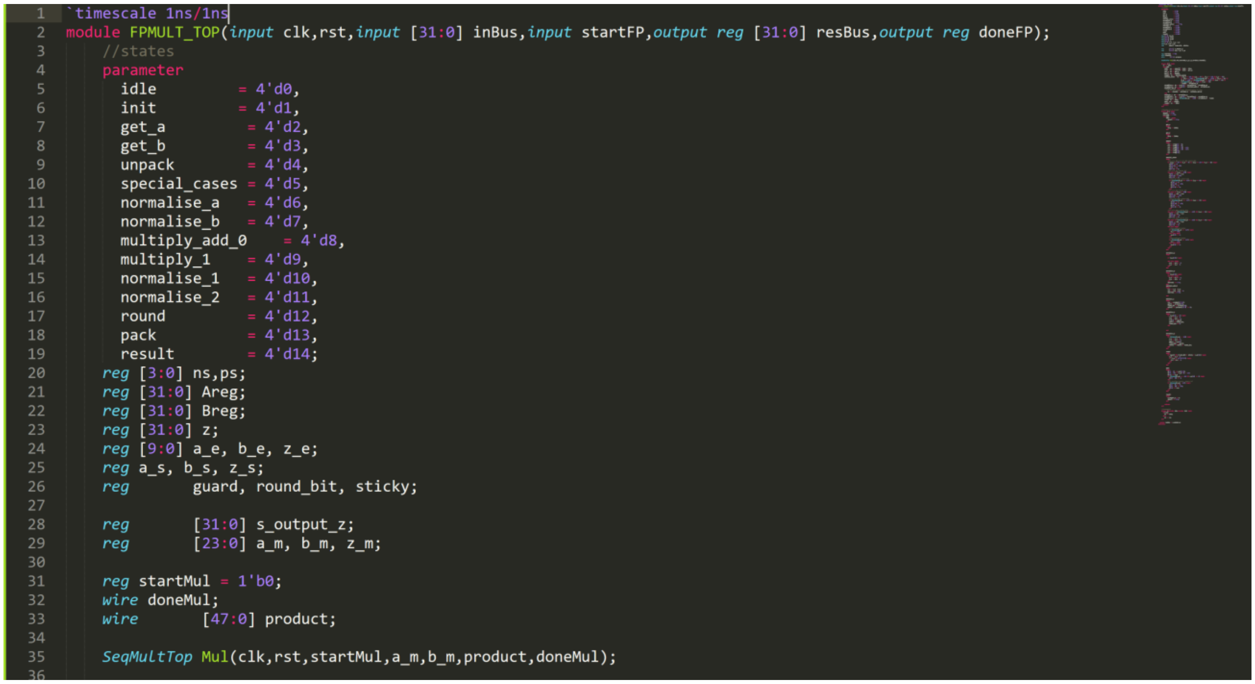
Wave - Default

```
                        Msgs
/MULT_SEQ_TB/clk        0
/MULT_SEQ_TB/rst        0
/MULT_SEQ_TB/start      0
/MULT_SEQ_TB/A          000000000000000000001011
/MULT_SEQ_TB/B          000000000000000000000011
/MULT_SEQ_TB/Res...     000000000000000000000000000000000000000000100001
/MULT_SEQ_TB/ready      St1
```

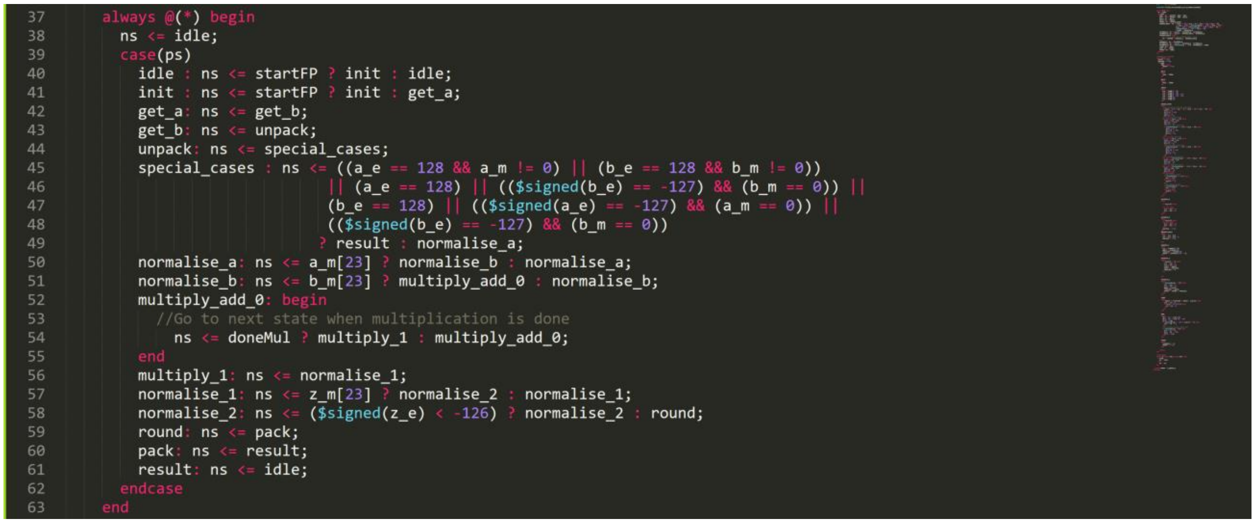
```
Now         648 ns
Cursor 1    380 ns
```

`

H. Put the above three parts together and create the complete design. Simulate this circuit.

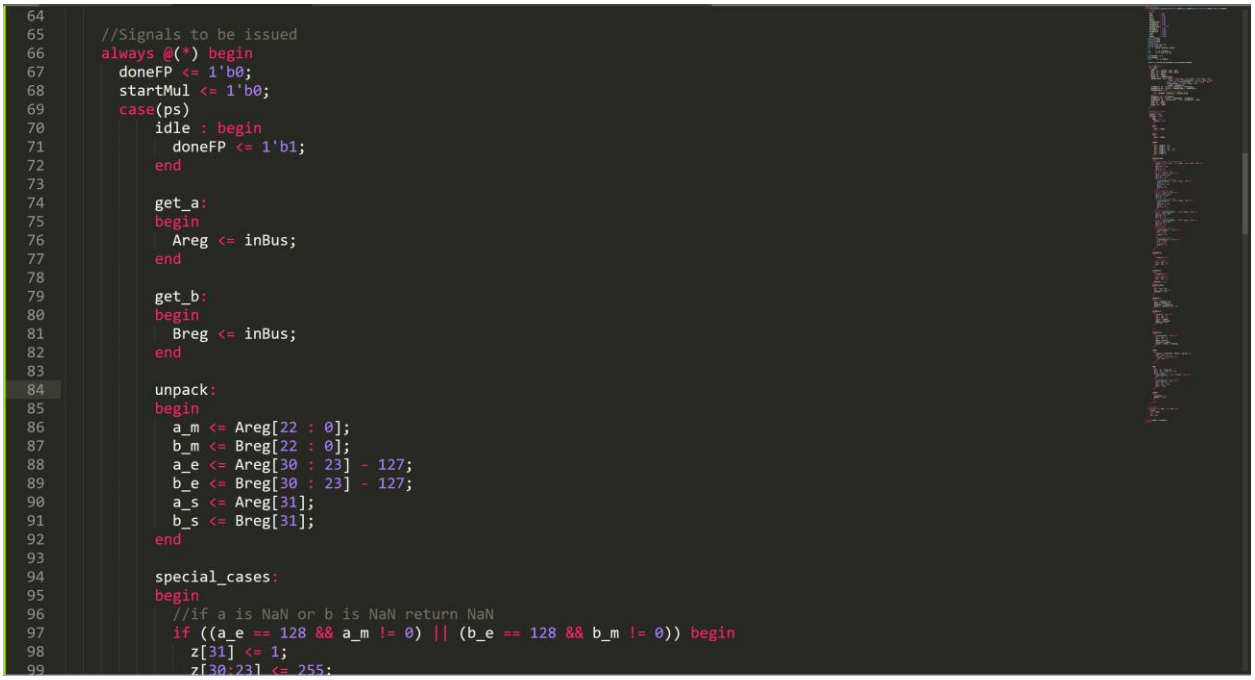


```
`timescale 1ns/1ns
module FPMULT_TOP(input clk,rst,input [31:0] inBus,input startFP,output reg [31:0] resBus,output reg doneFP);
    //states
    parameter
      idle         = 4'd0,
      init         = 4'd1,
      get_a         = 4'd2,
      get_b         = 4'd3,
      unpack        = 4'd4,
      special_cases = 4'd5,
      normalise_a   = 4'd6,
      normalise_b   = 4'd7,
      multiply_add_0     = 4'd8,
      multiply_1    = 4'd9,
      normalise_1   = 4'd10,
      normalise_2   = 4'd11,
      round         = 4'd12,
      pack          = 4'd13,
      result        = 4'd14;
    reg [3:0] ns,ps;
    reg [31:0] Areg;
    reg [31:0] Breg;
    reg [31:0] z;
    reg [9:0] a_e, b_e, z_e;
    reg a_s, b_s, z_s;
    reg       guard, round_bit, sticky;

    reg       [31:0] s_output_z;
    reg       [23:0] a_m, b_m, z_m;

    reg startMul = 1'b0;
    wire doneMul;
    wire       [47:0] product;

    SeqMultTop Mul(clk,rst,startMul,a_m,b_m,product,doneMul);
```

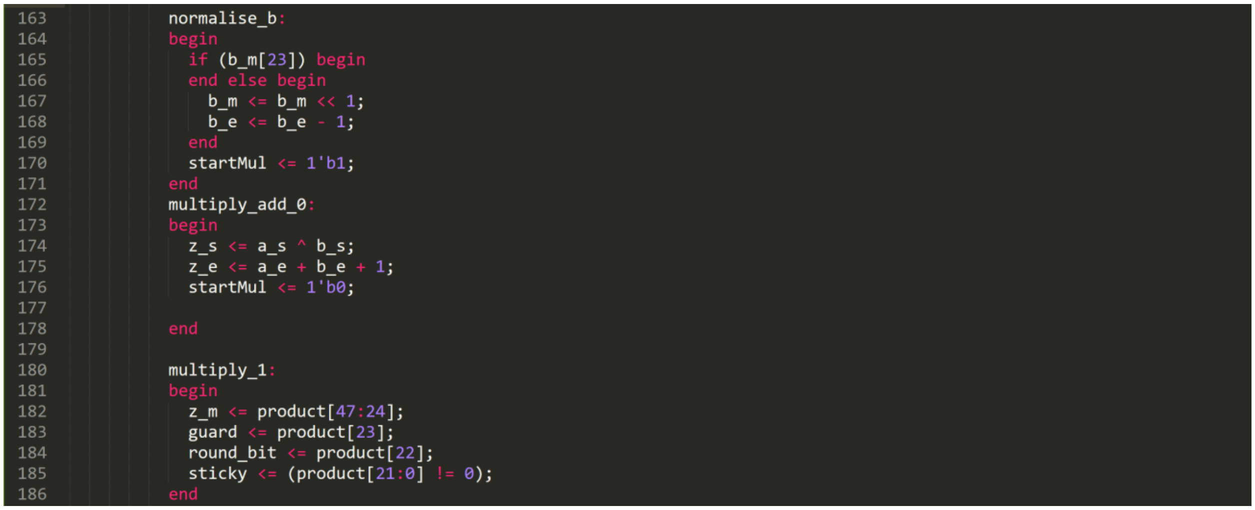


```
    always @(*) begin
      ns <= idle;
      case(ps)
        idle : ns <= startFP ? init : idle;
        init : ns <= startFP ? init : get_a;
        get_a: ns <= get_b;
        get_b: ns <= unpack;
        unpack: ns <= special_cases;
        special_cases : ns <= ((a_e == 128 && a_m != 0) || (b_e == 128 && b_m != 0))
                             || (a_e == 128) || (($signed(b_e) == -127) && (b_m == 0)) ||
                             (b_e == 128) || (($signed(a_e) == -127) && (a_m == 0)) ||
                             (($signed(b_e) == -127) && (b_m == 0))
                            ? result : normalise_a;
        normalise_a: ns <= a_m[23] ? normalise_b : normalise_a;
        normalise_b: ns <= b_m[23] ? multiply_add_0 : normalise_b;
        multiply_add_0: begin
          //Go to next state when multiplication is done
            ns <= doneMul ? multiply_1 : multiply_add_0;
        end
        multiply_1: ns <= normalise_1;
        normalise_1: ns <= z_m[23] ? normalise_2 : normalise_1;
        normalise_2: ns <= ($signed(z_e) < -126) ? normalise_2 : round;
        round: ns <= pack;
        pack: ns <= result;
        result: ns <= idle;
      endcase
    end
```

`

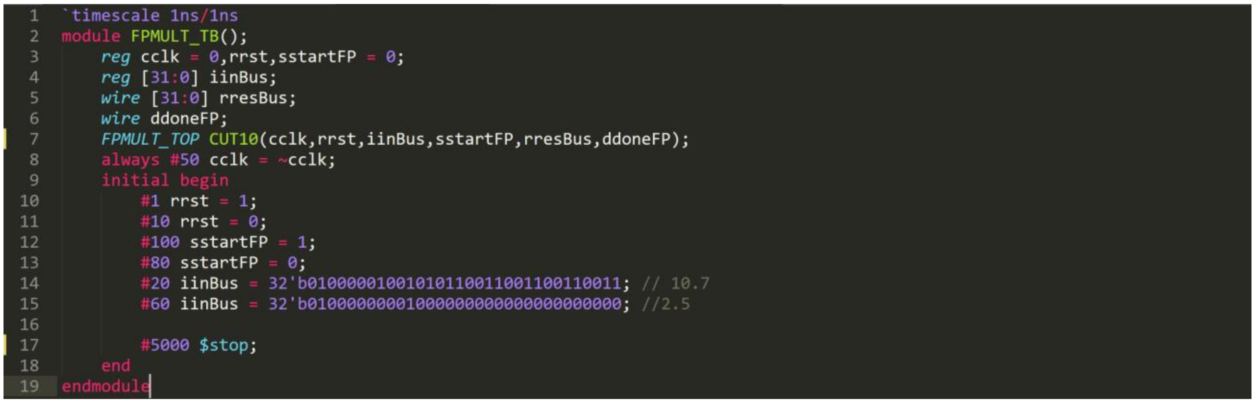


```
    //Signals to be issued
    always @(*) begin
      doneFP <= 1'b0;
      startMul <= 1'b0;
      case(ps)
          idle : begin
            doneFP <= 1'b1;
          end

          get_a:
          begin
            Areg <= inBus;
          end

          get_b:
          begin
            Breg <= inBus;
          end

          unpack:
          begin
            a_m <= Areg[22 : 0];
            b_m <= Breg[22 : 0];
            a_e <= Areg[30 : 23] - 127;
            b_e <= Breg[30 : 23] - 127;
            a_s <= Areg[31];
            b_s <= Breg[31];
          end

          special_cases:
          begin
            //if a is NaN or b is NaN return NaN
            if ((a_e == 128 && a_m != 0) || (b_e == 128 && b_m != 0)) begin
              z[31] <= 1;
              z[30:23] <= 255;
```



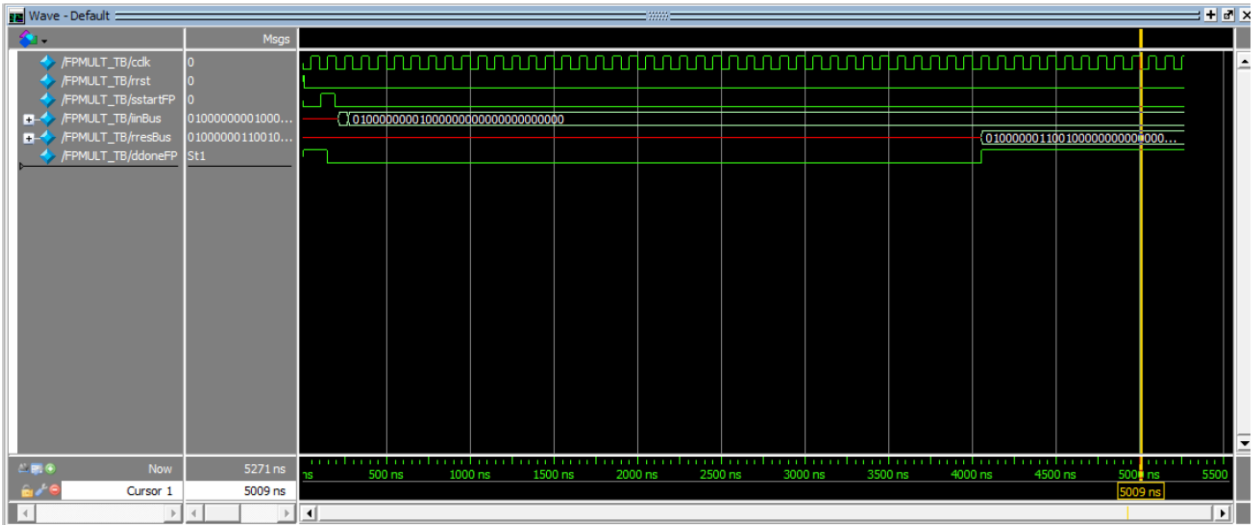
```
          normalise_b:
          begin
            if (b_m[23]) begin
            end else begin
              b_m <= b_m << 1;
              b_e <= b_e - 1;
            end
            startMul <= 1'b1;
          end
          multiply_add_0:
          begin
            z_s <= a_s ^ b_s;
            z_e <= a_e + b_e + 1;
            startMul <= 1'b0;

          end

          multiply_1:
          begin
            z_m <= product[47:24];
            guard <= product[23];
            round_bit <= product[22];
            sticky <= (product[21:0] != 0);
          end
```

Testbench



```
`timescale 1ns/1ns
module FPMULT_TB();
    reg cclk = 0,rrst,sstartFP = 0;
    reg [31:0] iinBus;
    wire [31:0] rresBus;
    wire ddoneFP;
    FPMULT_TOP CUT10(cclk,rrst,iinBus,sstartFP,rresBus,ddoneFP);
    always #50 cclk = ~cclk;
    initial begin
        #1 rrst = 1;
        #10 rrst = 0;
        #100 sstartFP = 1;
        #80 sstartFP = 0;
        #20 iinBus = 32'b01000001001010110011001100110011; // 10.7
        #60 iinBus = 32'b01000000001000000000000000000000; //2.5

        #5000 $stop;
    end
endmodule
```

`

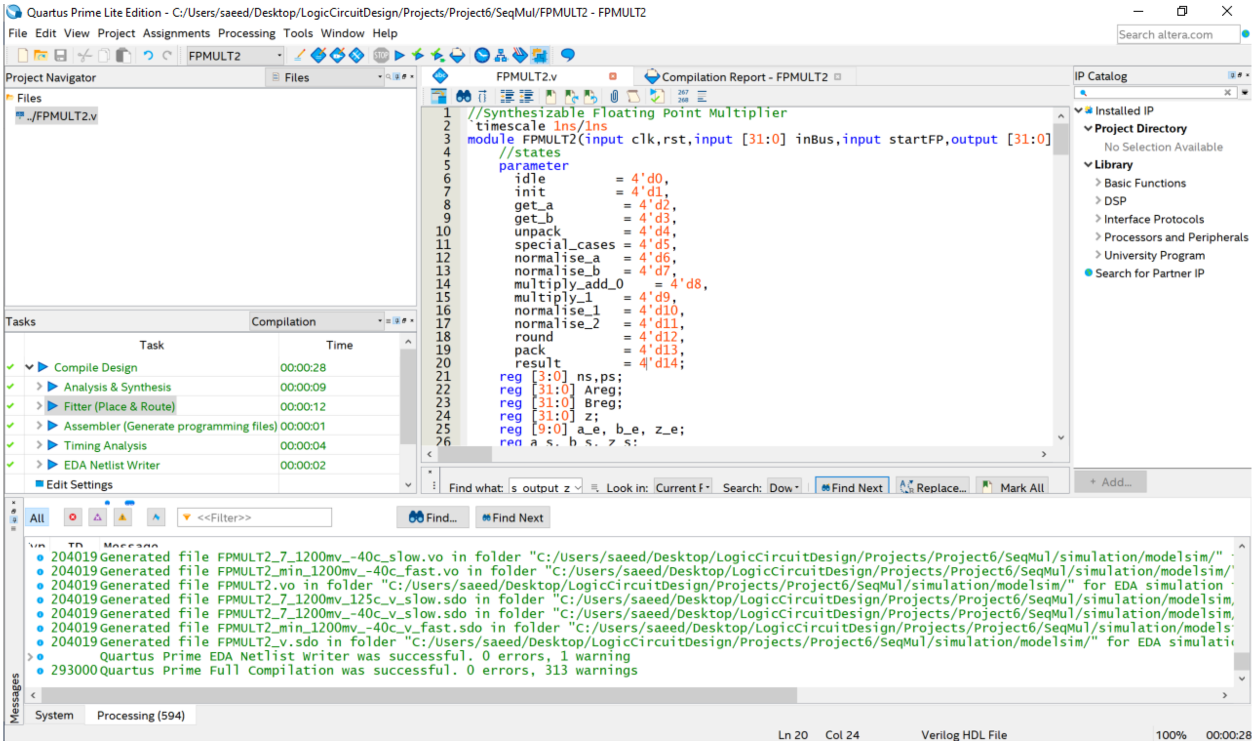


Wave - Default

	Msgs
/FPMULT_TB/cclk	0
/FPMULT_TB/rrst	0
/FPMULT_TB/sstartFP	0
/FPMULT_TB/iinBus	01000000001000...
/FPMULT_TB/rresBus	01000000110010...
/FPMULT_TB/ddoneFP	St1
Now	5271 ns
Cursor 1	5009 ns

`

K. Synthesize the FP multiplier part (excluding the Wrappers and the Sequential multiplier part) and create a symbol for it. Show synthesis reports.


Quartus Prime Lite Edition - C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/FPMULT2 - FPMULT2

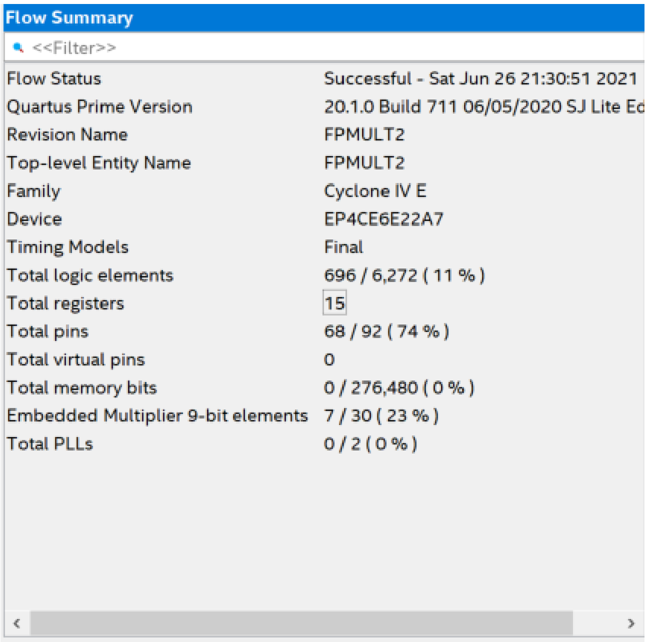
Task	Time
✓ Compile Design	00:00:28
✓ Analysis & Synthesis	00:00:09
✓ Fitter (Place & Route)	00:00:12
✓ Assembler (Generate programming files)	00:00:01
✓ Timing Analysis	00:00:04
✓ EDA Netlist Writer	00:00:02

```
//Synthesizable Floating Point Multiplier
`timescale 1ns/1ns
module FPMULT2(input clk,rst,input [31:0] inBus,input startFP,output [31:0]
    //states
    parameter
      idle          = 4'd0,
      init          = 4'd1,
      get_a          = 4'd2,
      get_b          = 4'd3,
      unpack         = 4'd4,
      special_cases = 4'd5,
      normalise_a   = 4'd6,
      normalise_b   = 4'd7,
      multiply_add_0     = 4'd8,
      multiply_1     = 4'd9,
      normalise_1   = 4'd10,
      normalise_2   = 4'd11,
      round          = 4'd12,
      pack           = 4'd13,
      result         = 4'd14;
    reg [3:0] ns,ps;
    reg [31:0] Areg;
    reg [31:0] Breg;
    reg [31:0] z;
    reg [9:0] a_e, b_e, z_e;
    reg a_s, b_s, z_s;
```

```
ID     Message
204019 Generated file FPMULT2_7_1200mv_-40c_slow.vo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim/"
204019 Generated file FPMULT2_min_1200mv_-40c_fast.vo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim/"
204019 Generated file FPMULT2.vo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim/" for EDA simulation
204019 Generated file FPMULT2_7_1200mv_125c_v_slow.sdo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim,
204019 Generated file FPMULT2_7_1200mv_-40c_v_slow.sdo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim,
204019 Generated file FPMULT2_min_1200mv_-40c_v_fast.sdo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/models
204019 Generated file FPMULT2_v.sdo in folder "C:/Users/saeed/Desktop/LogicCircuitDesign/Projects/Project6/SeqMul/simulation/modelsim/" for EDA simulati
       Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
293000 Quartus Prime Full Compilation was successful. 0 errors, 313 warnings
```

Ln 20 Col 24 Verilog HDL File 100% 00:00:28

Flow Summary



Flow Status	Successful - Sat Jun 26 21:30:51 2021
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Ed
Revision Name	FPMULT2
Top-level Entity Name	FPMULT2
Family	Cyclone IV E
Device	EP4CE6E22A7
Timing Models	Final
Total logic elements	696 / 6,272 (11 %)
Total registers	15
Total pins	68 / 92 (74 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	7 / 30 (23 %)
Total PLLs	0 / 2 (0 %)

`

RTL Viewer