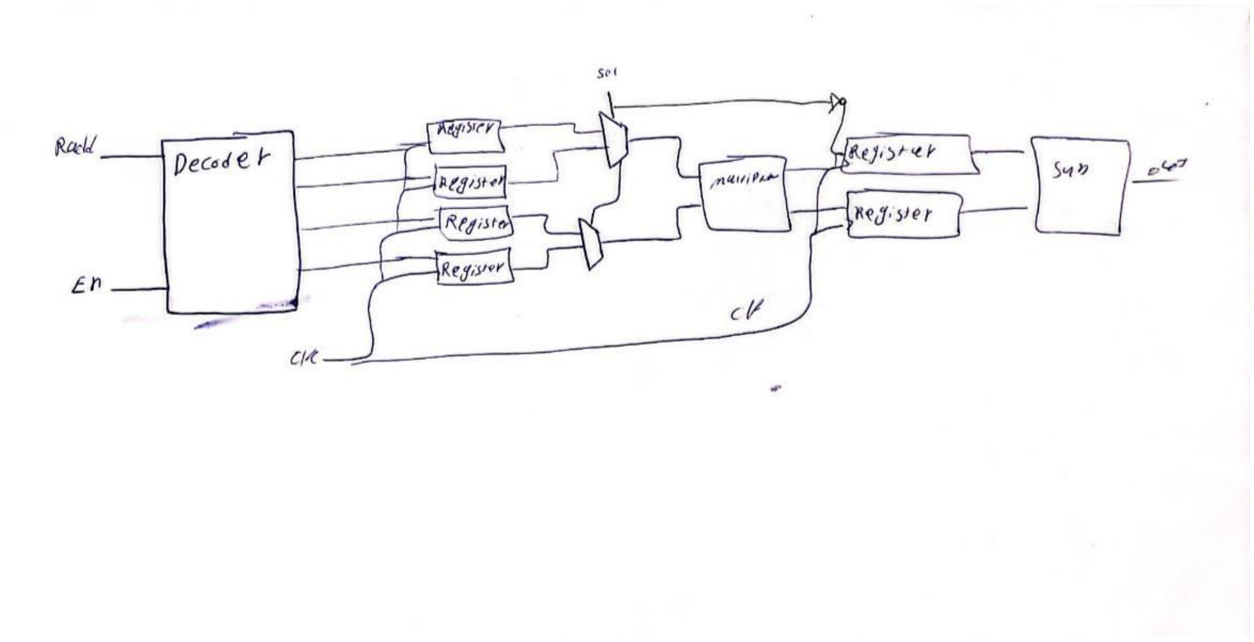
Sadra Madayeni
810102564
CA 4

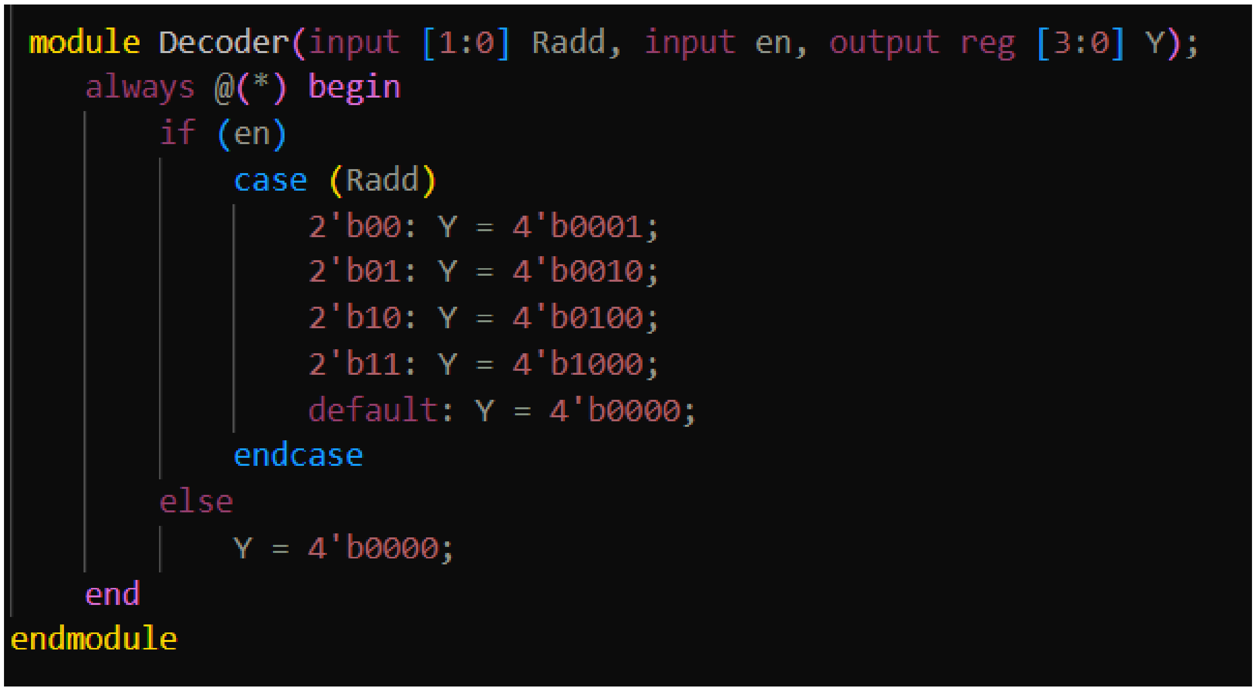
In this computer assignment we wanted to calculate the Determinant Of 2 * 2 Matrix

Here Is How Our Data Path Should Work In 2 * 2 module



In Our Second Register Enable Should be The inverted Mux Selector

Here is My Decoder First Of all

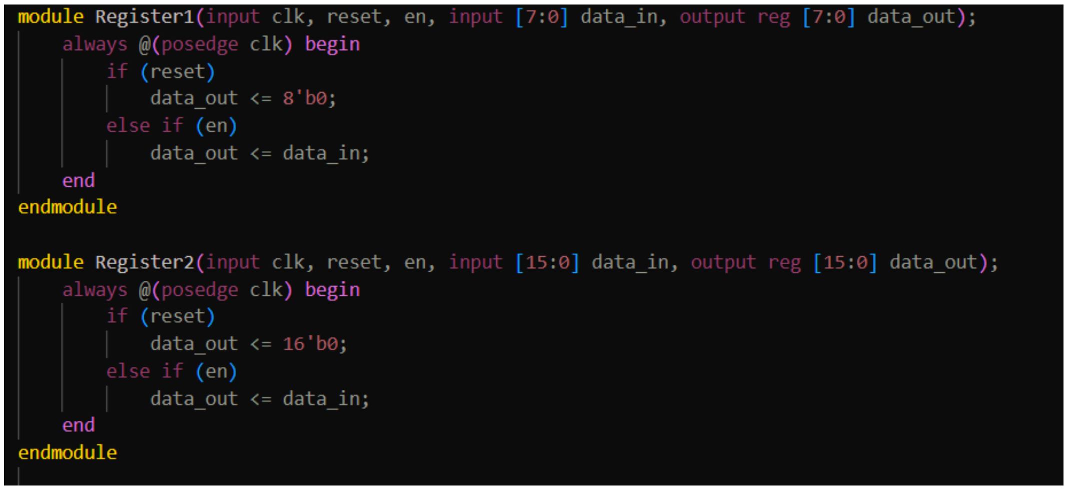


```
module Decoder(input [1:0] Radd, input en, output reg [3:0] Y);
   always @(*) begin
      if (en)
         case (Radd)
            2'b00: Y = 4'b0001;
            2'b01: Y = 4'b0010;
            2'b10: Y = 4'b0100;
            2'b11: Y = 4'b1000;
            default: Y = 4'b0000;
         endcase
      else
         Y = 4'b0000;
   end
endmodule
```

As You Can See Its 2 To 4 Decoder Implemented Behavioral

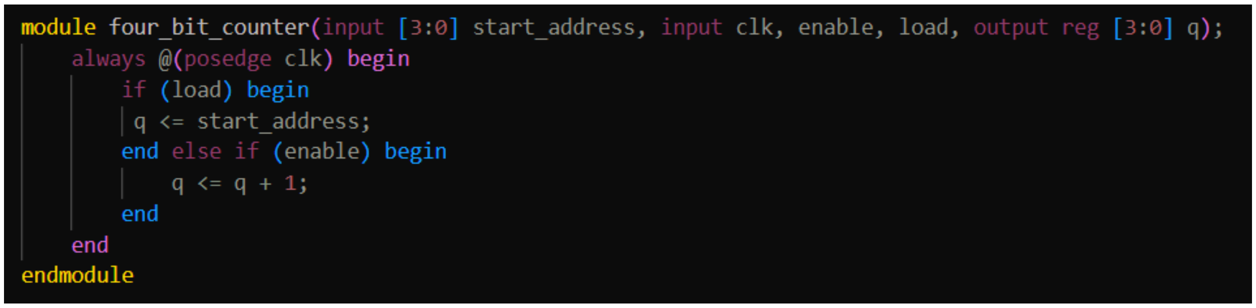
Next We Need A Register
I Made Two Different Types Of Register First 8 bit And second one 16 bit

Here Is How I Implemented Registers



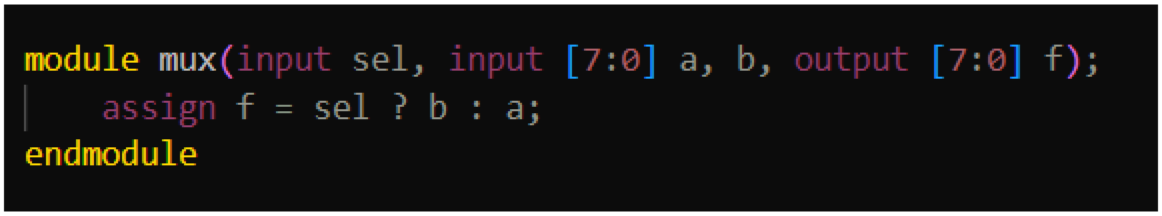
```
module Register1(input clk, reset, en, input [7:0] data_in, output reg [7:0] data_out);
    always @(posedge clk) begin
        if (reset)
            data_out <= 8'b0;
        else if (en)
            data_out <= data_in;
    end
endmodule

module Register2(input clk, reset, en, input [15:0] data_in, output reg [15:0] data_out);
    always @(posedge clk) begin
        if (reset)
            data_out <= 16'b0;
        else if (en)
            data_out <= data_in;
    end
endmodule
```

Then We Needed a Counter To Calculate The Cycles



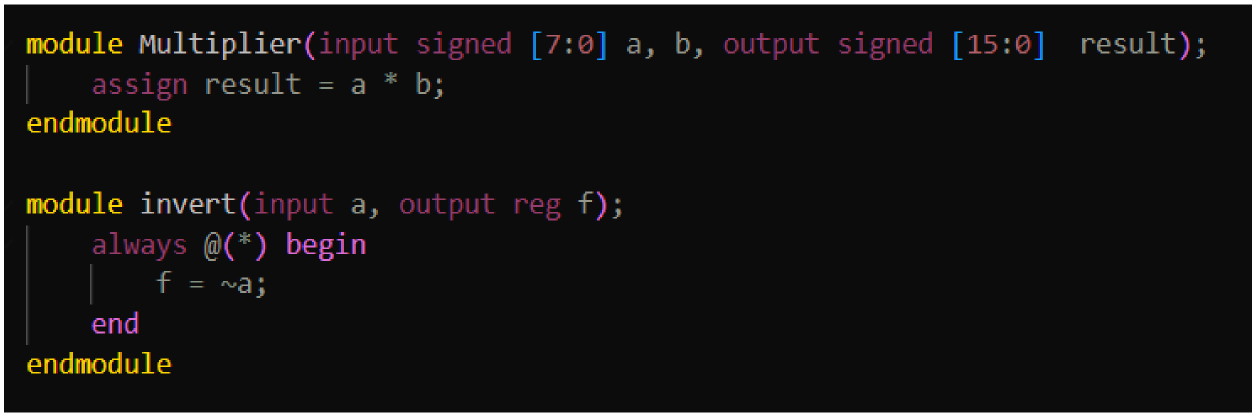
```
module four_bit_counter(input [3:0] start_address, input clk, enable, load, output reg [3:0] q);
    always @(posedge clk) begin
        if (load) begin
            q <= start_address;
        end else if (enable) begin
            q <= q + 1;
        end
    end
endmodule
```

Then We Need a Multiplexer To Decide What Output Goes Into Multiplier



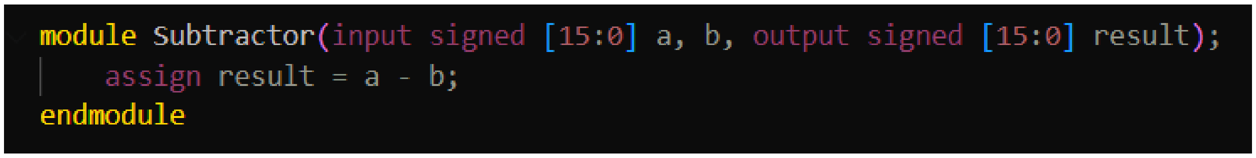
```
module mux(input sel, input [7:0] a, b, output [7:0] f);
    assign f = sel ? b : a;
endmodule
```

And Next Here Is How I Implemented My Multiplier



```
module Multiplier(input signed [7:0] a, b, output signed [15:0]  result);
    assign result = a * b;
endmodule

module invert(input a, output reg f);
    always @(*) begin
        f = ~a;
    end
endmodule
```

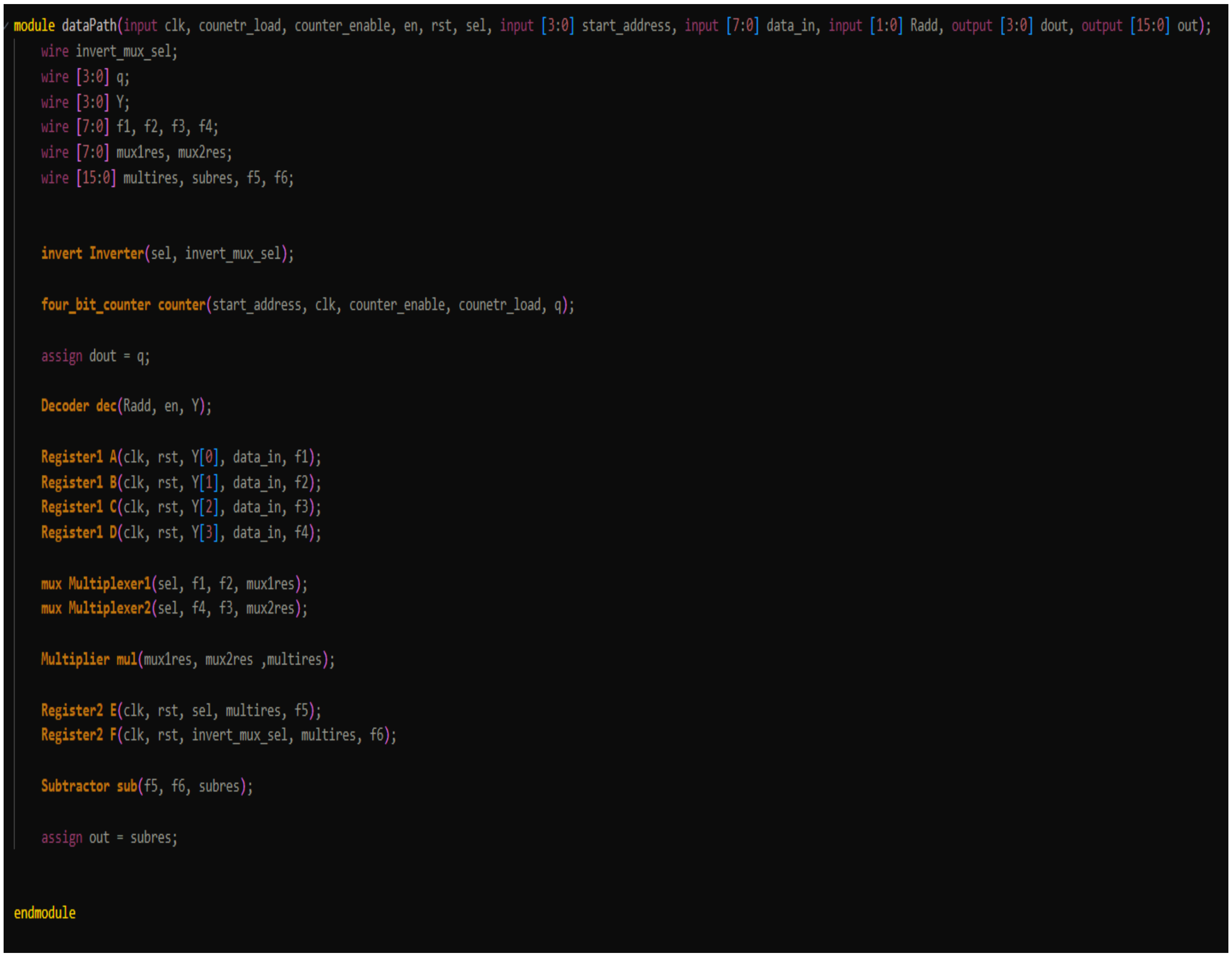
Next We Needed Another Register That I Called It Register2 (Module In The Register Pictures)



```
module Subtractor(input signed [15:0] a, b, output signed [15:0] result);
    assign result = a - b;
endmodule
```

At Least We Need a Subtractor To Calculate Result

Then We Need To Implement Our Data Path

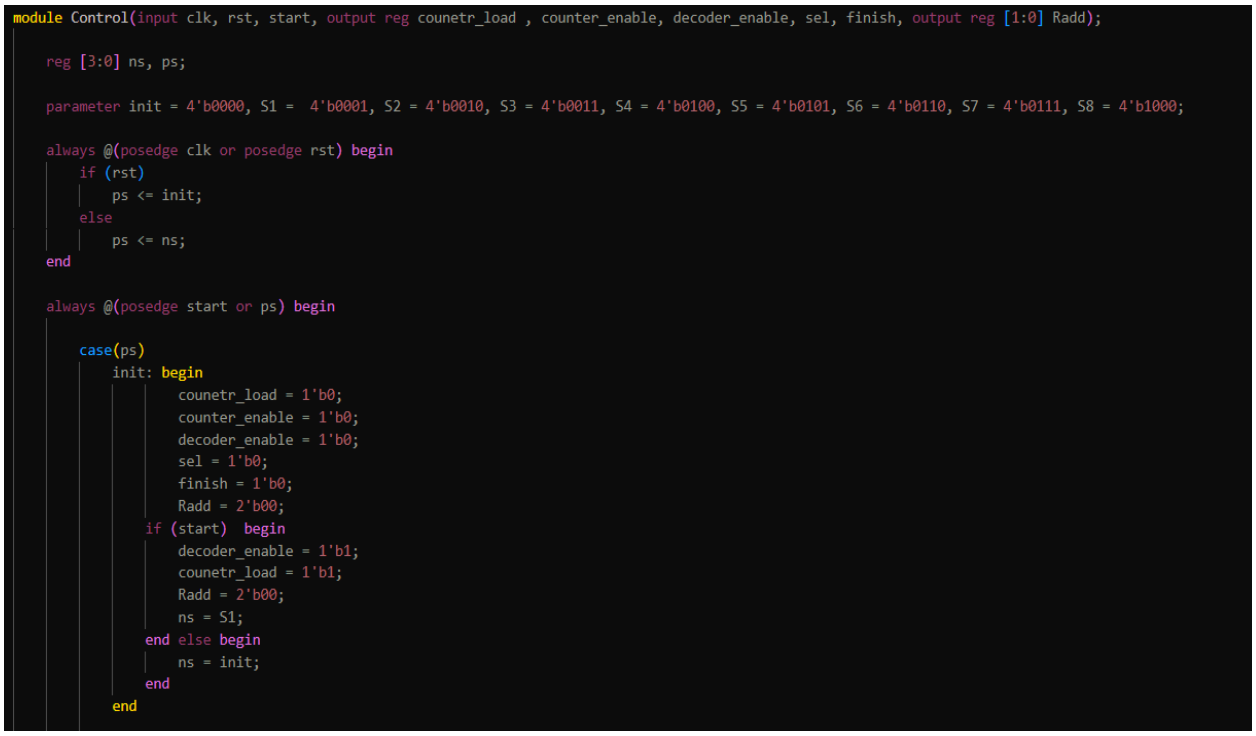


```
module dataPath(input clk, counetr_load, counter_enable, en, rst, sel, input [3:0] start_address, input [7:0] data_in, input [1:0] Radd, output [3:0] dout, output [15:0] out);
    wire invert_mux_sel;
    wire [3:0] q;
    wire [3:0] Y;
    wire [7:0] f1, f2, f3, f4;
    wire [7:0] mux1res, mux2res;
    wire [15:0] multires, subres, f5, f6;


    invert Inverter(sel, invert_mux_sel);

    four_bit_counter counter(start_address, clk, counter_enable, counetr_load, q);

    assign dout = q;

    Decoder dec(Radd, en, Y);

    Register1 A(clk, rst, Y[0], data_in, f1);
    Register1 B(clk, rst, Y[1], data_in, f2);
    Register1 C(clk, rst, Y[2], data_in, f3);
    Register1 D(clk, rst, Y[3], data_in, f4);

    mux Multiplexer1(sel, f1, f2, mux1res);
    mux Multiplexer2(sel, f4, f3, mux2res);

    Multiplier mul(mux1res, mux2res ,multires);

    Register2 E(clk, rst, sel, multires, f5);
    Register2 F(clk, rst, invert_mux_sel, multires, f6);

    Subtractor sub(f5, f6, subres);

    assign out = subres;


endmodule
```

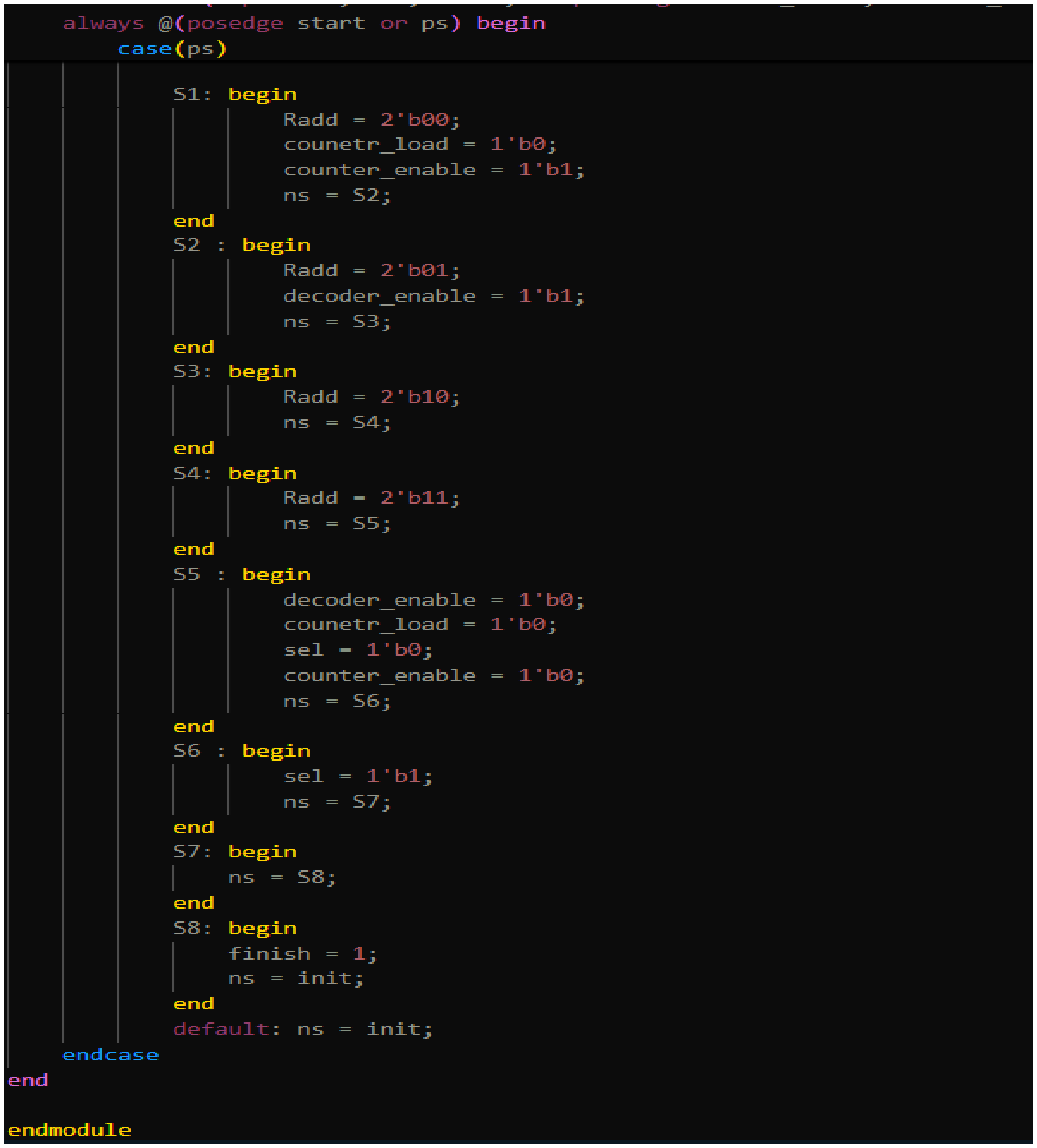
And At Least I Assigned Out = Subresult

Here Was How My Data - Path Works

Then We Needed A Controller To Control The States OF Our Determinant Calculator

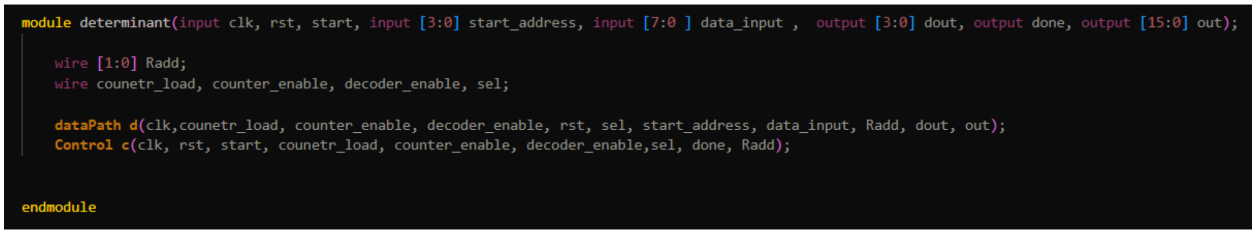


```
module Control(input clk, rst, start, output reg counetr_load , counter_enable, decoder_enable, sel, finish, output reg [1:0] Radd);

    reg [3:0] ns, ps;

    parameter init = 4'b0000, S1 =  4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 = 4'b0100, S5 = 4'b0101, S6 = 4'b0110, S7 = 4'b0111, S8 = 4'b1000;

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

    always @(posedge start or ps) begin

        case(ps)
            init: begin
                    counetr_load = 1'b0;
                    counter_enable = 1'b0;
                    decoder_enable = 1'b0;
                    sel = 1'b0;
                    finish = 1'b0;
                    Radd = 2'b00;
                if (start)  begin
                    decoder_enable = 1'b1;
                    counetr_load = 1'b1;
                    Radd = 2'b00;
                    ns = S1;
                end else begin
                    ns = init;
                end
            end
```



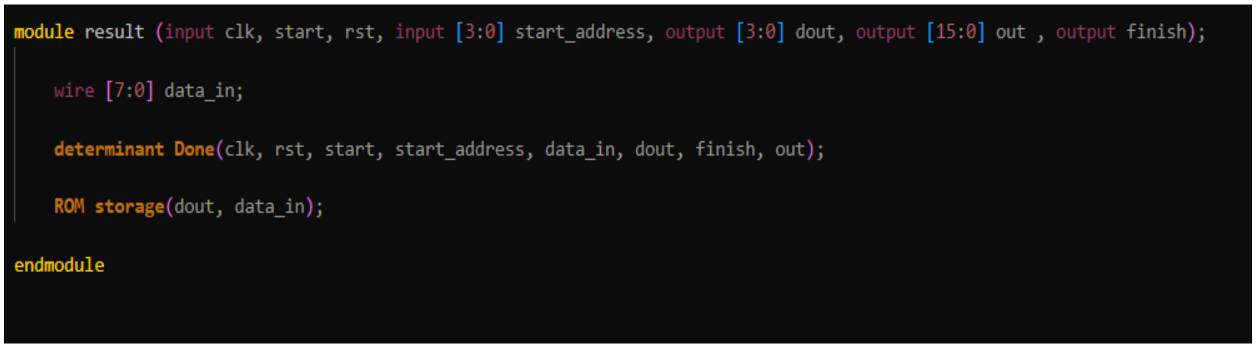
```
    always @(posedge start or ps) begin
        case(ps)

            S1: begin
                    Radd = 2'b00;
                    counetr_load = 1'b0;
                    counter_enable = 1'b1;
                    ns = S2;
            end
            S2 : begin
                    Radd = 2'b01;
                    decoder_enable = 1'b1;
                    ns = S3;
            end
            S3: begin
                    Radd = 2'b10;
                    ns = S4;
            end
            S4: begin
                    Radd = 2'b11;
                    ns = S5;
            end
            S5 : begin
                    decoder_enable = 1'b0;
                    counetr_load = 1'b0;
                    sel = 1'b0;
                    counter_enable = 1'b0;
                    ns = S6;
            end
            S6 : begin
                    sel = 1'b1;
                    ns = S7;
            end
            S7: begin
                ns = S8;
            end
            S8: begin
                finish = 1;
                ns = init;
            end
            default: ns = init;
    endcase
end

endmodule
```

After That I’ve made a Determinant Module That Has Everything In It



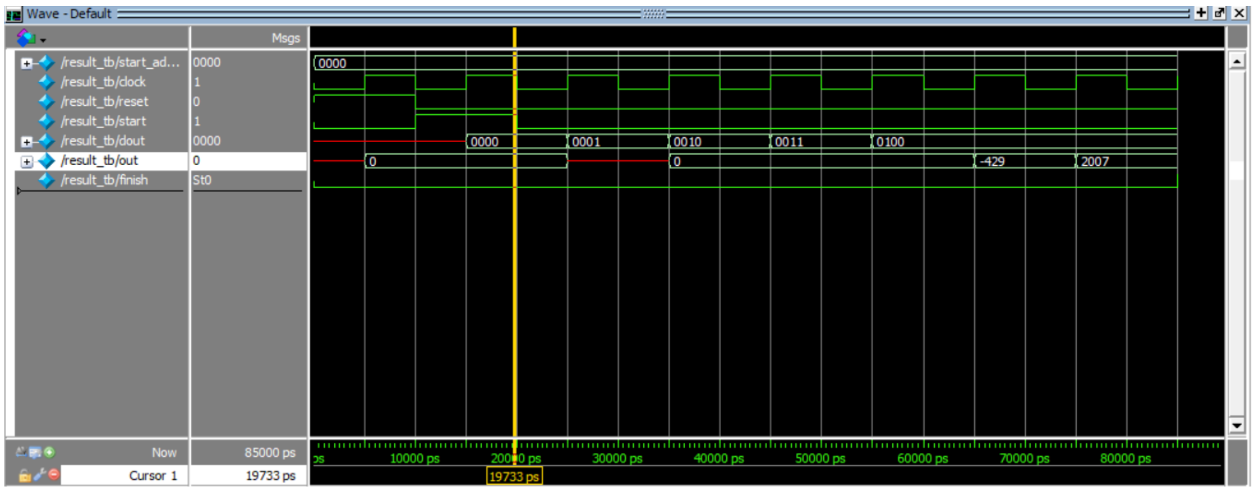
```
module determinant(input clk, rst, start, input [3:0] start_address, input [7:0 ] data_input ,  output [3:0] dout, output done, output [15:0] out);

    wire [1:0] Radd;
    wire counetr_load, counter_enable, decoder_enable, sel;

    dataPath d(clk,counetr_load, counter_enable, decoder_enable, rst, sel, start_address, data_input, Radd, dout, out);
    Control c(clk, rst, start, counetr_load, counter_enable, decoder_enable,sel, done, Radd);


endmodule
```

Then At least I have My Result That contains Calculates 2 * 2 determinant



```
module result (input clk, start, rst, input [3:0] start_address, output [3:0] dout, output [15:0] out , output finish);

    wire [7:0] data_in;

    determinant Done(clk, rst, start, start_address, data_in, dout, finish, out);

    ROM storage(dout, data_in);

endmodule
```

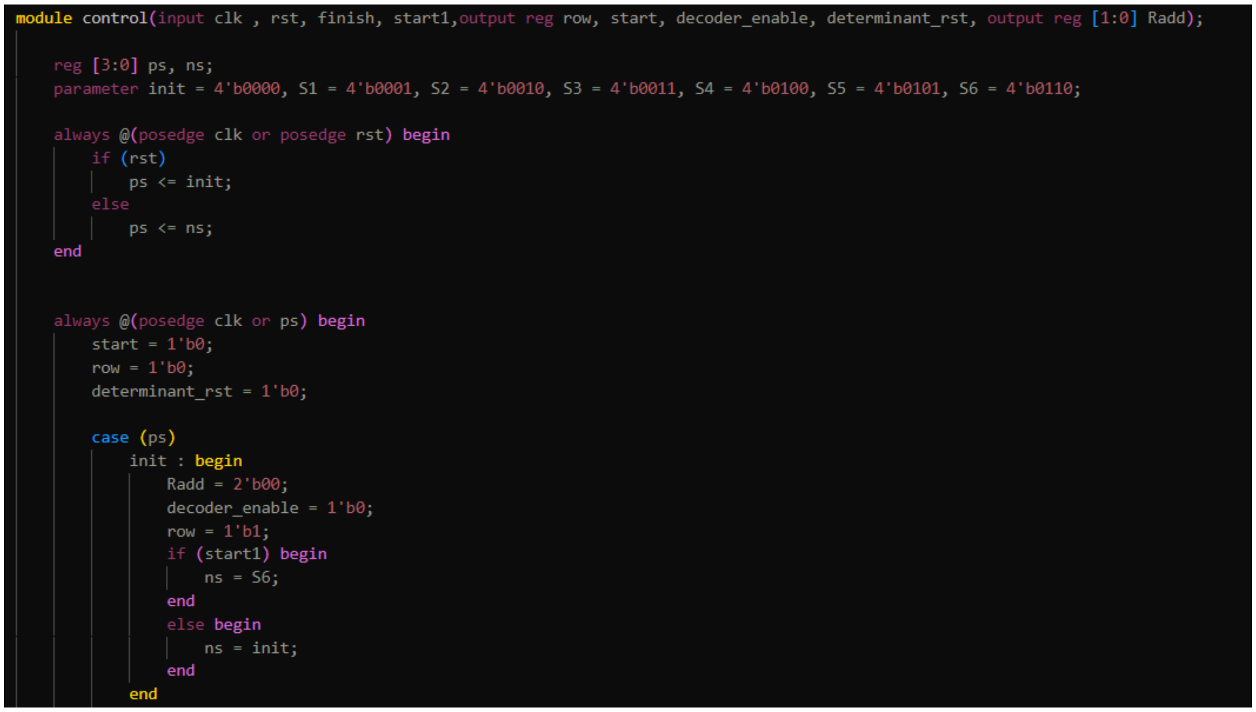
Here is The Result of 2 * 2 calculator for Start Address of 0 Which is correct

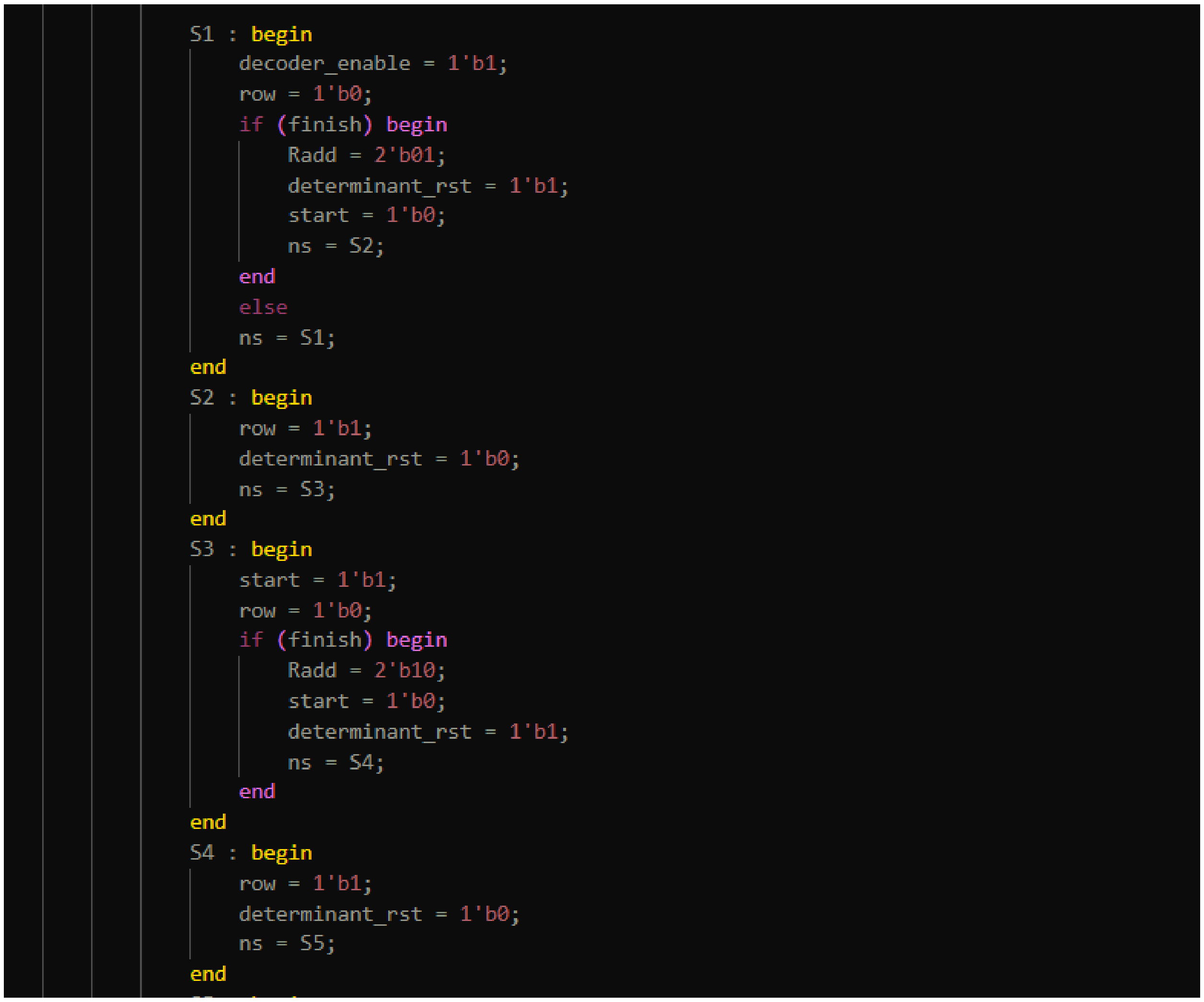


We got 2007 And Its Correct With the Start Address Of 0

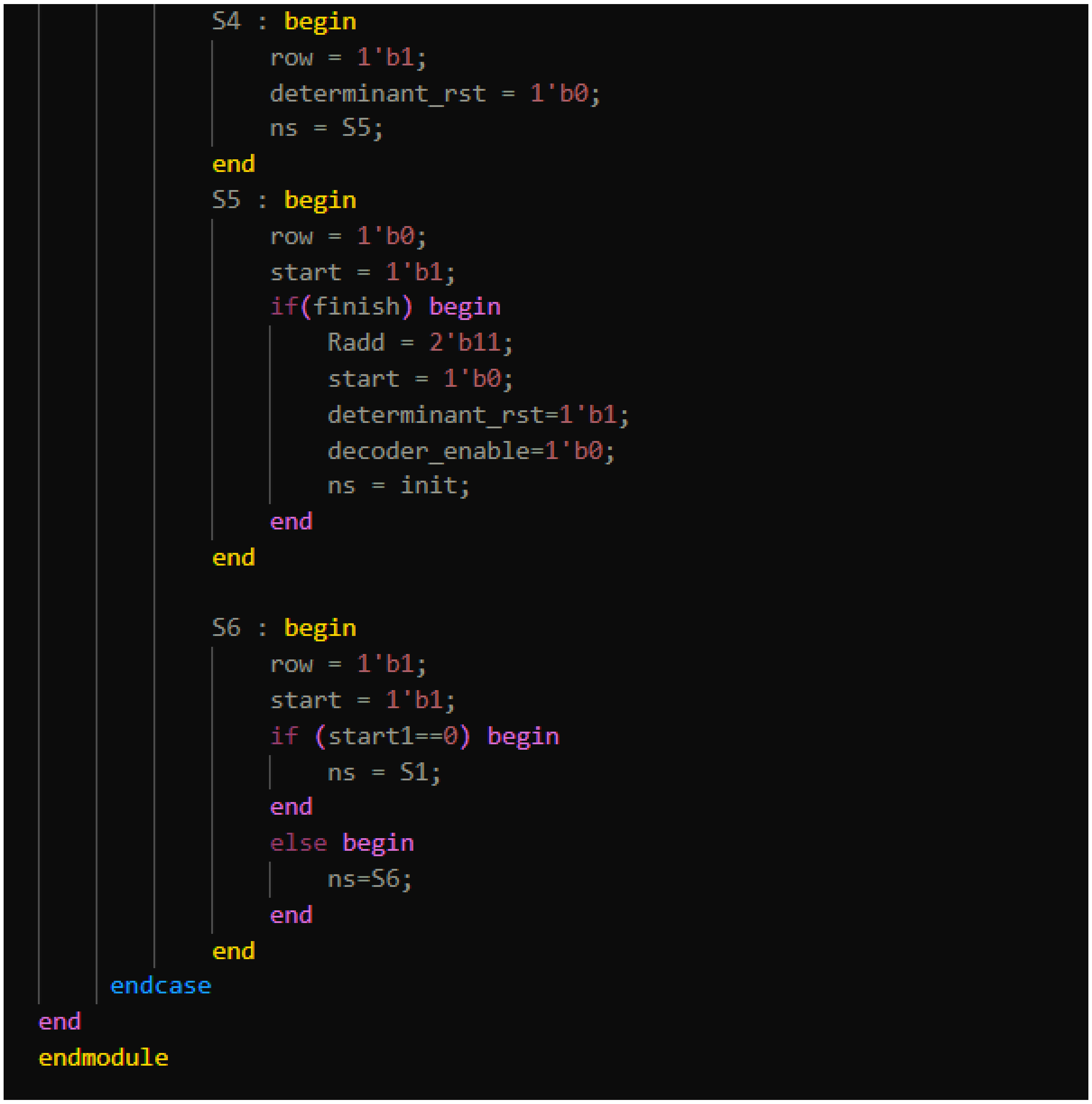
For Calculating 3 * 3 Determinant We Needed Another Controller To Read The Row And Calculate it!

Here is Another Controller


```
module control(input clk , rst, finish, start1,output reg row, start, decoder_enable, determinant_rst, output reg [1:0] Radd);

    reg [3:0] ps, ns;
    parameter init = 4'b0000, S1 = 4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 = 4'b0100, S5 = 4'b0101, S6 = 4'b0110;

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


    always @(posedge clk or ps) begin
        start = 1'b0;
        row = 1'b0;
        determinant_rst = 1'b0;

        case (ps)
            init : begin
                Radd = 2'b00;
                decoder_enable = 1'b0;
                row = 1'b1;
                if (start1) begin
                    ns = S6;
                end
                else begin
                    ns = init;
                end
            end
```

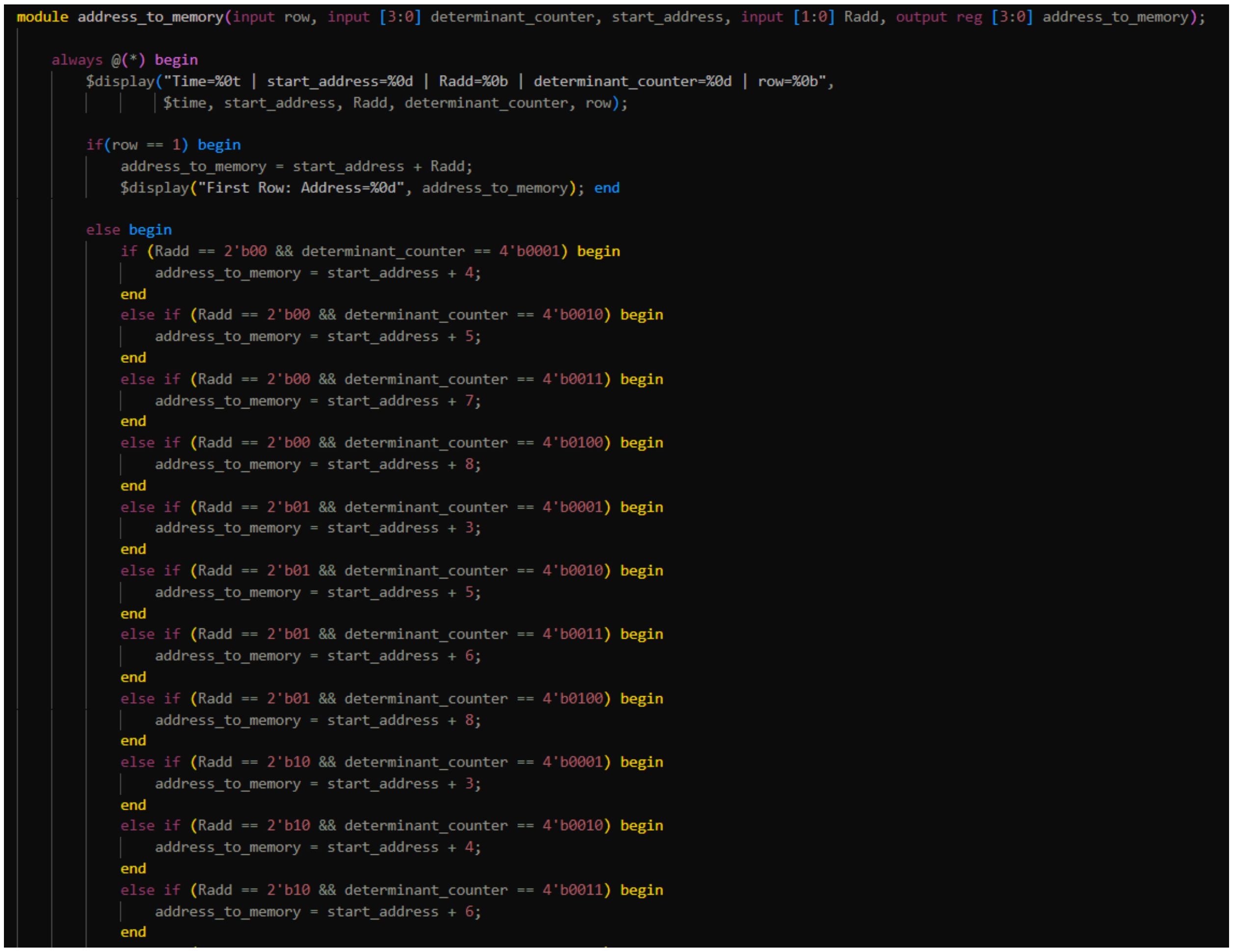


```verilog
            S1 : begin
                decoder_enable = 1'b1;
                row = 1'b0;
                if (finish) begin
                    Radd = 2'b01;
                    determinant_rst = 1'b1;
                    start = 1'b0;
                    ns = S2;
                end
                else
                ns = S1;
            end
            S2 : begin
                row = 1'b1;
                determinant_rst = 1'b0;
                ns = S3;
            end
            S3 : begin
                start = 1'b1;
                row = 1'b0;
                if (finish) begin
                    Radd = 2'b10;
                    start = 1'b0;
                    determinant_rst = 1'b1;
                    ns = S4;
                end
            end
            S4 : begin
                row = 1'b1;
                determinant_rst = 1'b0;
                ns = S5;
            end
```



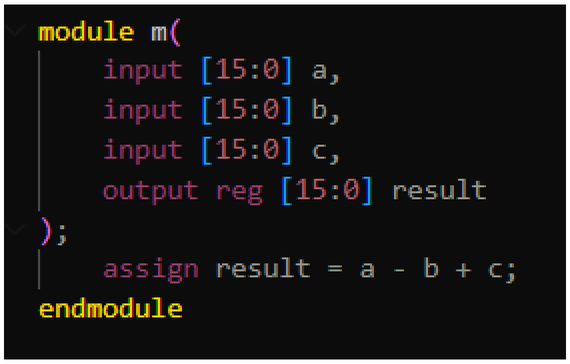
```
                S4 : begin
                    row = 1'b1;
                    determinant_rst = 1'b0;
                    ns = S5;
                end
                S5 : begin
                    row = 1'b0;
                    start = 1'b1;
                    if(finish) begin
                        Radd = 2'b11;
                        start = 1'b0;
                        determinant_rst=1'b1;
                        decoder_enable=1'b0;
                        ns = init;
                    end
                end

                S6 : begin
                    row = 1'b1;
                    start = 1'b1;
                    if (start1==0) begin
                        ns = S1;
                    end
                    else begin
                        ns=S6;
                    end
                end
        endcase
end
endmodule
```

Then We Needed A Address_to_memory module



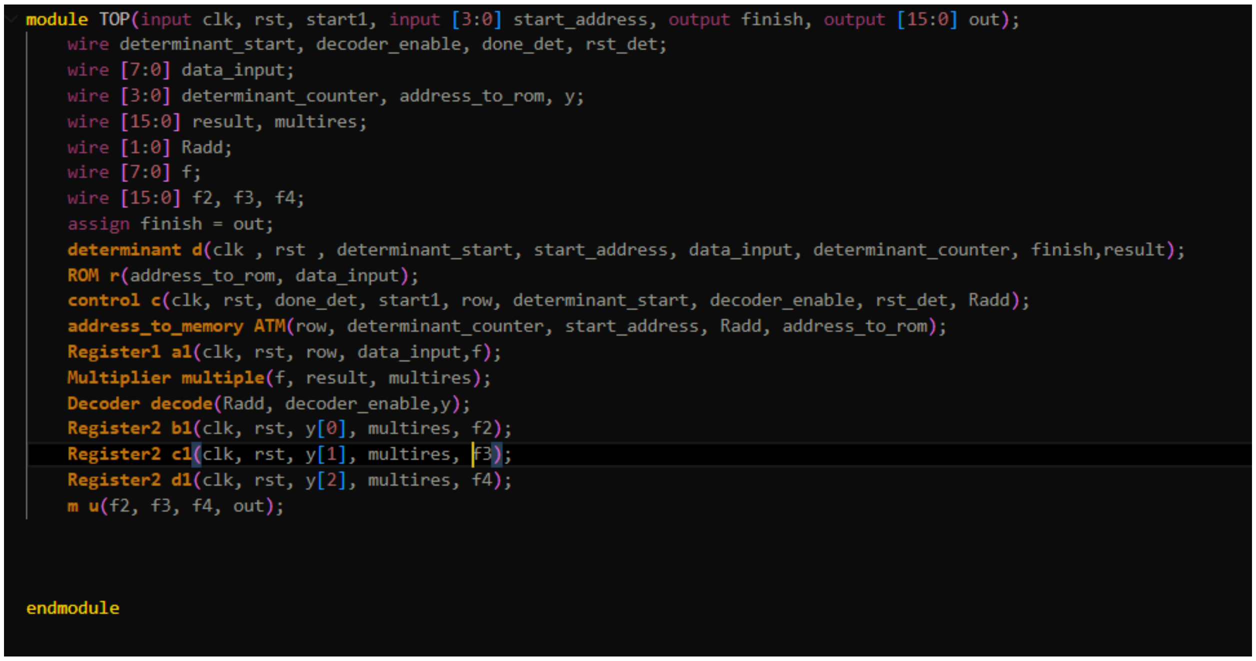
```
module address_to_memory(input row, input [3:0] determinant_counter, start_address, input [1:0] Radd, output reg [3:0] address_to_memory);

    always @(*) begin
        $display("Time=%0t | start_address=%0d | Radd=%0b | determinant_counter=%0d | row=%0b",
                $time, start_address, Radd, determinant_counter, row);

        if(row == 1) begin
            address_to_memory = start_address + Radd;
            $display("First Row: Address=%0d", address_to_memory); end

        else begin
            if (Radd == 2'b00 && determinant_counter == 4'b0001) begin
                address_to_memory = start_address + 4;
            end
            else if (Radd == 2'b00 && determinant_counter == 4'b0010) begin
                address_to_memory = start_address + 5;
            end
            else if (Radd == 2'b00 && determinant_counter == 4'b0011) begin
                address_to_memory = start_address + 7;
            end
            else if (Radd == 2'b00 && determinant_counter == 4'b0100) begin
                address_to_memory = start_address + 8;
            end
            else if (Radd == 2'b01 && determinant_counter == 4'b0001) begin
                address_to_memory = start_address + 3;
            end
            else if (Radd == 2'b01 && determinant_counter == 4'b0010) begin
                address_to_memory = start_address + 5;
            end
            else if (Radd == 2'b01 && determinant_counter == 4'b0011) begin
                address_to_memory = start_address + 6;
            end
            else if (Radd == 2'b01 && determinant_counter == 4'b0100) begin
                address_to_memory = start_address + 8;
            end
            else if (Radd == 2'b10 && determinant_counter == 4'b0001) begin
                address_to_memory = start_address + 3;
            end
            else if (Radd == 2'b10 && determinant_counter == 4'b0010) begin
                address_to_memory = start_address + 4;
            end
            else if (Radd == 2'b10 && determinant_counter == 4'b0011) begin
                address_to_memory = start_address + 6;
            end
```

And Also Also another module to calculate the last opperations



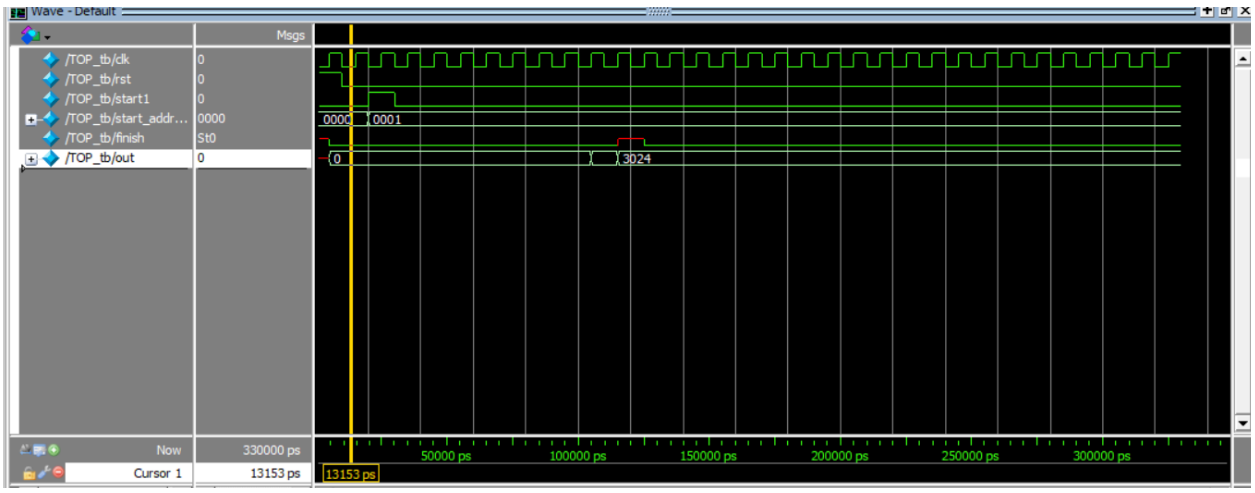
```
module m(
    input [15:0] a,
    input [15:0] b,
    input [15:0] c,
    output reg [15:0] result
);
    assign result = a - b + c;
endmodule
```

Then we Got all Our Module In The TOP module Which Calculates 3 * 3 Determinant



```
module TOP(input clk, rst, start1, input [3:0] start_address, output finish, output [15:0] out);
    wire determinant_start, decoder_enable, done_det, rst_det;
    wire [7:0] data_input;
    wire [3:0] determinant_counter, address_to_rom, y;
    wire [15:0] result, multires;
    wire [1:0] Radd;
    wire [7:0] f;
    wire [15:0] f2, f3, f4;
    assign finish = out;
    determinant d(clk , rst , determinant_start, start_address, data_input, determinant_counter, finish,result);
    ROM r(address_to_rom, data_input);
    control c(clk, rst, done_det, start1, row, determinant_start, decoder_enable, rst_det, Radd);
    address_to_memory ATM(row, determinant_counter, start_address, Radd, address_to_rom);
    Register1 a1(clk, rst, row, data_input,f);
    Multiplier multiple(f, result, multires);
    Decoder decode(Radd, decoder_enable,y);
    Register2 b1(clk, rst, y[0], multires, f2);
    Register2 c1(clk, rst, y[1], multires, f3);
    Register2 d1(clk, rst, y[2], multires, f4);
    m u(f2, f3, f4, out);


endmodule
```

And here is Final Output Which I think its incorrect!


Wave - Default
Msgs
/TOP_tb/clk 0
/TOP_tb/rst 0
/TOP_tb/start1 0
/TOP_tb/start_addr... 0000
/TOP_tb/finish St0
/TOP_tb/out 0
0000
0001
0
3024
Now 330000 ps
Cursor 1 13153 ps
13153 ps
50000 ps
100000 ps
150000 ps
200000 ps
250000 ps
300000 ps