جواب سؤال ۱:

Syntehsis: ورودی این مرحله کد HDL است و خروجی آن Gate-level Netlist یا Circuit Netlist است (0.5).

این مرحله شامل سه عمل اصلی Translation، Logic Optimization و Technology Mapping یا Technology Mapping & Optimization است (0.375).

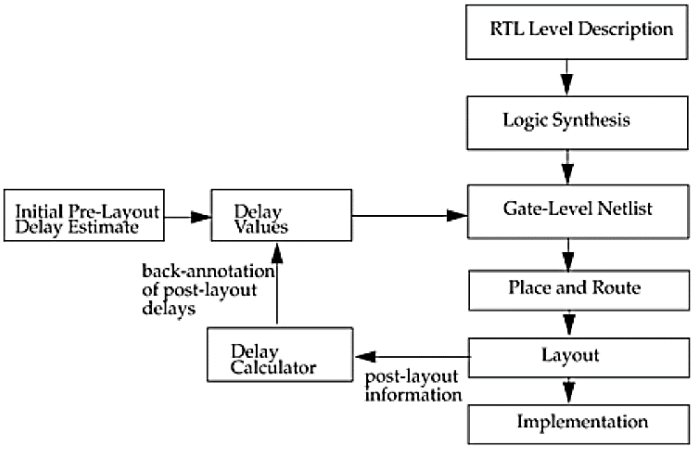
I/O Planning: در این مرحله ورودی و خروجی‌های مدار به پایه‌های تراشه متصل می‌شوند (0.25).

Place & Route: ورودی این مرحله Gate-level Netlist یا Circuit Netlist و خروجی آن Layout است (0.375).

این مرحله شمال عملیات جانمایی (Placement) هر واحد پایه تکنولوژی در جای مناسب است. سپس مرحله Routing (مسیریابی): اتصال این واحدهای پایه تکنولوژی با سیم کشی مناسب بین آن‌ها (0.25).

در پایان این دو مرحله تاخیر دقیق قابل محاسبه است (0.25).

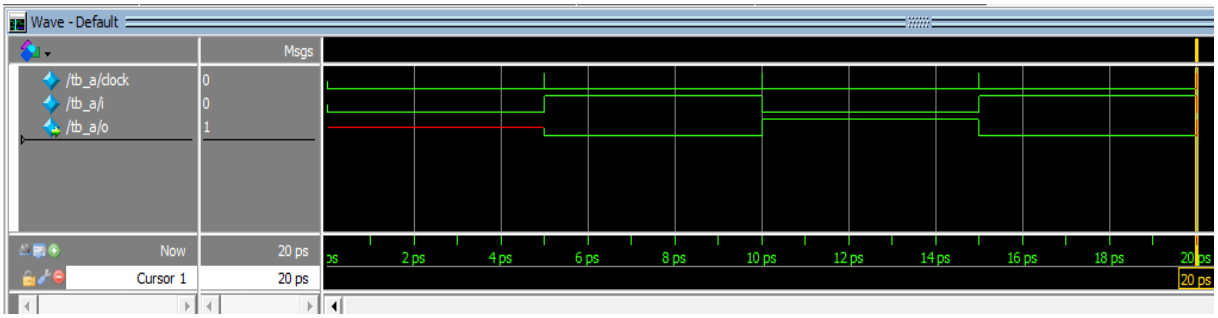
این شکل را شاید یک سری از افراد کشیده باشند:



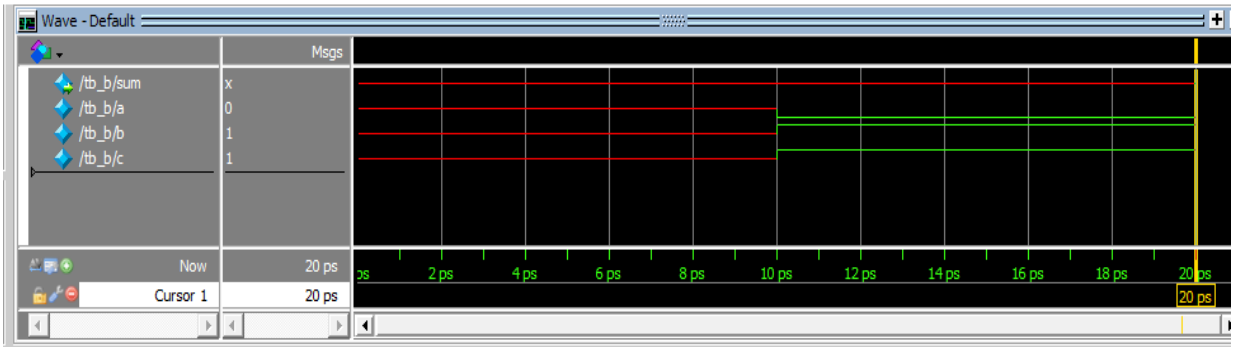
جواب سوال ۲:

رنگ‌های قرمز برابر با مقدار x است.

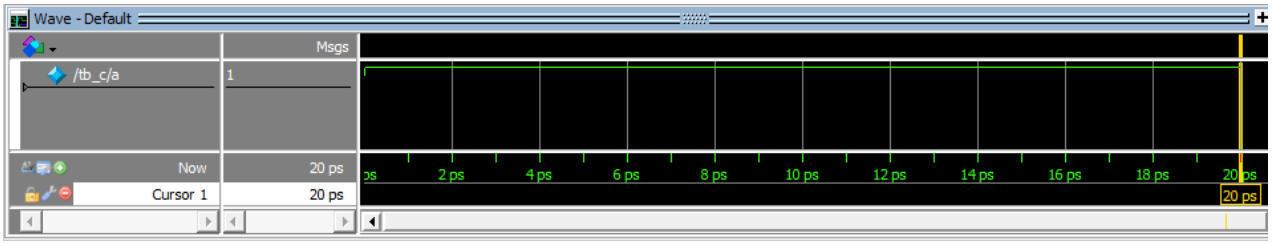
الف) نیم نمره



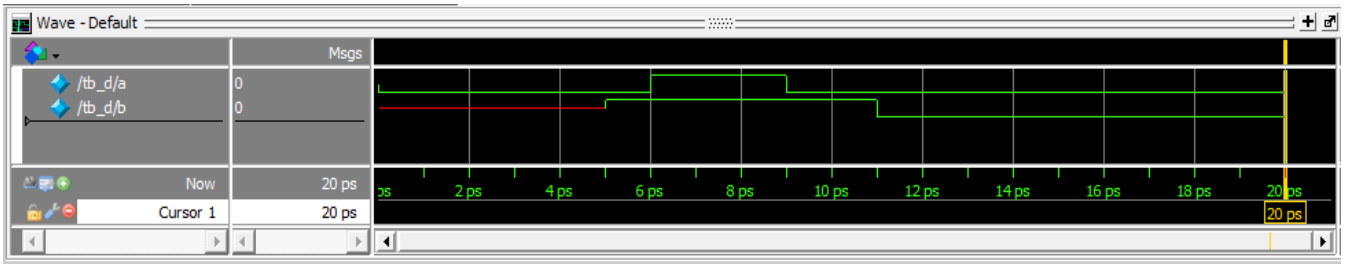
ب) نیم نمره



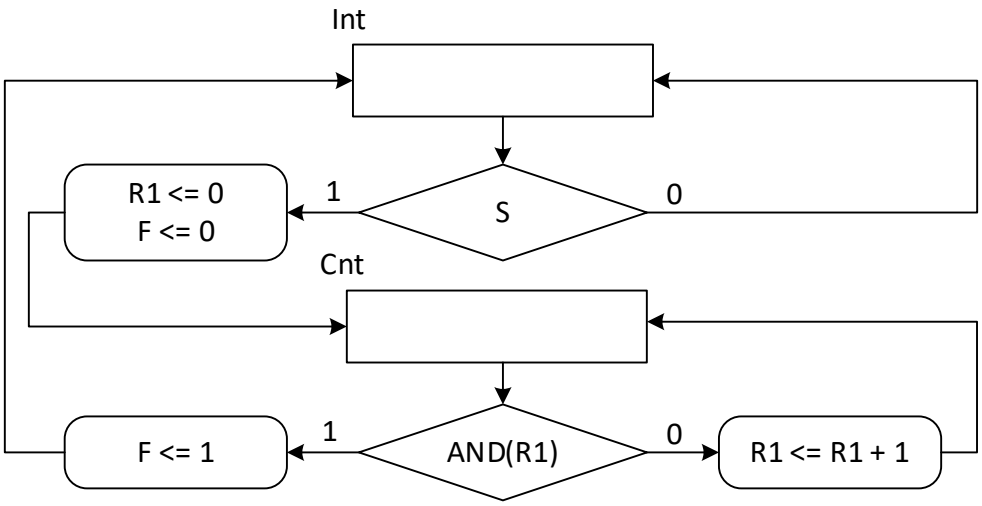
پ) نیم نمره



ت) نیم نمره

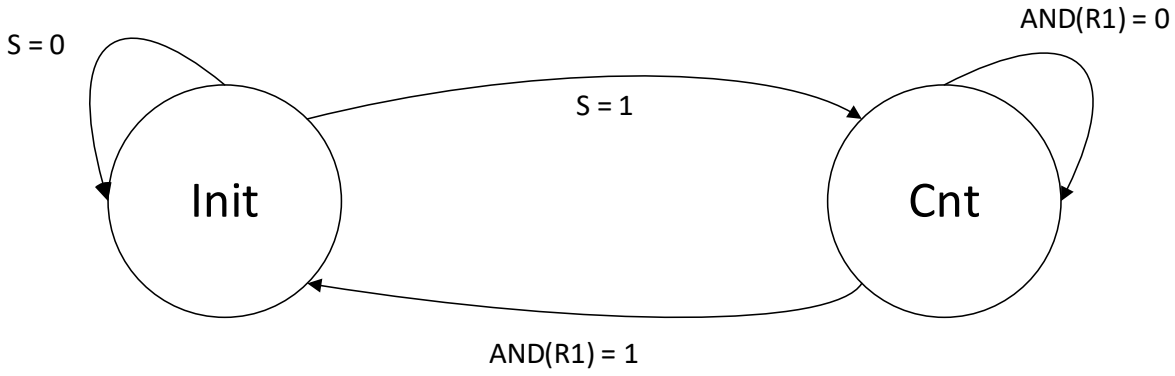


جواب سوال ۳:



کشیدن نمودار زیر نمره ندارد، اما اگر کسی کشیده باشد، نشان از این است که احتمالا درس را مطالعه کرده است.

این کد دارای دو ASM Block است که نمودار حالت آن به فرم زیر است.



برای طراحی DataPath لازم است سخت‌افزارهای مورد نیاز را تشخیص دهیم. در این طراحی ما به یک رجیستر ۸ بیتی به نام R1، یک AND ۸ ورودی و یک فلیپ فلاپ با نام F نیاز داریم. بنابراین ماژول‌های زیر را تعریف می‌کنیم.

طراحی ماژول رجیستر: 0.25

در ماژول رجیستر باید عملیات زیر پیاده‌سازی شده باشد.

S0	S1	Register Operation
0	0	No Change
0	1	Parallel Load
1	0	Increment
1	1	Clear

برای انجام این عملیات ورودی‌های پالس ساعت (clk) و ریست (n_rst) به همراه ورودی‌های کنترلی (S)، داده (in_data) و خروجی داده (out_data) مورد نیاز است.

```
module universal_register(input clk, n_rst, [1:0] s, [7:0] in_data, output [7:0]
out_data);
reg [7:0] data;
assign out_data = data;
always @(posedge clk, negedge n_rst) begin
            if ( !n_rst )
               data <= 0;
            else begin
                case(s)
                    2'b00: data  <= data;
                    2'b01: data  <= in_data;
                    2'b10: data  <= data + 1;
                    2'b11: data  <= 0;
                endcase
            end

end
endmodule
```

همچنین می‌توان قسمت case را با if-then-else هم پیاده‌سازی شود. یا می‌توان ورودی‌های کنترلی را به ازای هر عملیات در نظر گرفت.

clear	increment	load	Register Operation
0	0	0	No Change
x	x	1	Parallel Load
x	1	x	Increment
1	x	x	Clear

کد if-then-else با ورودی‌های کنترلی جداگانه در زیر آورده شده است.

```
module universal_register(input clk, n_rst, clear, increment, load, [7:0] in_data, output
[7:0] out_data);
reg  [7:0] data;
assign out_data = data;
always @(posedge clk, negedge n_rst) begin
            if ( !n_rst )
               data <= 0;
            else begin
                if ( load && !increment && !clear )
                    data  <= in_data;
                if ( !load && increment && !clear )
                    data <= data + 1;
                if ( !load && !increment && clear  )
```

```
                    data <= 0;
                else
                    data <= data;
            end
end
endmodule
```

در صورتی که خط assign حذف شود باید خروجی out_data از نوع reg تعریف شود (`output reg [7:0] out_data`).
امکان دارد در طراحی رجیستر ورودی و خروجی‌های دیگری هم در نظر گرفته شود، که آن هم درست است. ولی طراحی حتما باید شامل ورودی و خروجی‌های کد بالا باشد.

نکته: اگر درون کد رجیستر از دستور assign یا گیت‌های پایه استفاده شده باشد، آن‌ها هم امکان دارد جواب درست باشد، آن برگه‌ها را به خود من نشان دهید.

می‌توان به جای negedge n_rst از posedge rst استفاده شود، یا اصلا مقدار rst را فقط درون بلاک always مانند کد زیر استفاده کرده باشند (ریست همزمان).

```
always @(posedge clk) begin
            if ( rst )
               data <= 0;
            else begin
                case(s)
                    2'b00: data  <= data;
                    2'b01: data  <= in_data;
                    2'b10: data  <= data + 1;
                    2'b11: data  <= 0;
                endcase
            end

end
```

نکته: اگر برای رجیستر ماژول جداگانه در نظر گرفته نشده باشد، حتما باید درون ماژول DataPath کدشان دارای یک always حاوی کدهای بالا باشد.
هرچند این روش رویکرد خوبی نیست و در این موارد به خود من کد را نشان دهید.

ماژول AND، ۸ ورودی را می‌توان به حالات زیر نوشت: 0.25

۱-

```
module AND_8(input [7:0] in, output reg out);
always @(in) begin
    if ( in == 8'b1111_1111 )
        out = 1;
    else
        out = 0;
end
endmodule
```

۲-

```
module AND_8(input [7:0] in, output reg out);
always @(in) begin
    out = in[0] & in[1] & in[2] & in[3] & in[4] & in[5] & in[6] & in[7];
end
endmodule
```

۳-

```
module AND_8(input [7:0] in, output out);
    assign out = in[0] & in[1] & in[2] & in[3] & in[4] & in[5] & in[6] & in[7];
endmodule
```

۴-

```
module AND_8(input [7:0] in, output out);
    assign out = (in ==  8'b1111_1111) ? 1 : 0;
endmodule
```

۵-

```
module AND_8(input [7:0] in, output out);
    and(out, in[0], in[1], in[2], in[3], in[4], in[5], in[6], in[7]);
endmodule
```

نکته: اگر برای AND ۸ ورودی ماژول مستقلی در نظر نگرفته باشند، باید آن را در DataPath توصیف کرده باشند.

طراحی ماژول فیلیپ فلاپ: 0.25

```
module DFF (input d,
            input n_rst,
            input clk,
            output reg q);

    always @ (posedge clk or negedge n_rst)
       if (!n_rst)
          q <= 0;
       else
          q <= d;
endmodule
```

می‌توان به جای negedge n_rst از posedge rst استفاده شود، یا اصلا مقدار rst را فقط درون بلاک always مانند کد زیر استفاده کرده باشند (ریست همزمان)

```
module DFF (input d,
            input rst,
            input clk,
            output reg q);

    always @ (posedge clk)
       if (rst)
          q <= 0;
       else
          q <= d;
endmodule
```

در انتها لازم است یک ماژول با نام DataPath و به صورت زیر تعریف شود. امکان دارد در ماژول DataPath کدهای درون هر کدام از ماژول‌ها آورده شود. آن‌ها را هم صحیح بگیرید، هر چند رویکرد مناسبی نیست.

```
module DataPath(input clk, input n_rst, output [7:0] r1_data, output f, output and_status,
input clear_r1, input increment_r1, input load_r1, input f_val);
```

```
    universal_register R1(clk, n_rst, clear_r1, increment_r1, load_r1, r1_data, r1_data);
    AND_8 and_8(r1_data, and_status);
    DFF F_FF(clk, n_rst, f_val, F);
endmodule
```

0.25

ماژول Control Unit نیز به صورت زیر تعریف می‌شود.

```
module ControlUnit(input clk, n_rst, output reg clear_r1, increment_r1, load_r1, f_value,
input and_status, s);

reg p_state, n_state;

localparam init=1'b0, cnt=1'b1;

always @( p_state or and_status or s )
begin:combi
    // Set all of Control Signals to Zero
    increment_r1 = 0;
    load_r1 = 0;
    clear_r1 = 0;
    case (p_state)
        init:
        begin
            if ( s == 1 )
            begin
                n_state = cnt;
                f_value = 0;
                clear_r1 = 1;
            end else
                n_state = init;
        end
        cnt:
        begin
            if ( and_status == 1 )
            begin
                f_value = 1;
                n_state = init;
            end else begin
                n_state = cnt;
                increment_r1 = 1;
            end
        end
    endcase
end

always @(posedge clk, negedge n_rst)
begin:sequential
    if ( !n_rst ) p_state = init;
    else p_state = n_state;
end

endmodule
```

0.25

0.25

0.25

به جای بخش sequential می‌توان از کد زیر نیز استفاده کرد.

```
always @(posedge clk)
begin:sequential
    if ( rst ) p_state = init;
    else p_state = n_state;
end
```

در پایان یک ماژول system به صورت زیر توصیف می‌کنیم.

0.25

```
module system(input S, output reg [7:0] R1, output F);

    DataPath dp(clk, n_rst, R1, F, and_status, clear_r1, increment_r1, load_r1, f_val);
    ControlUnit cu(clk, n_rst, clear_r1, increment_r1, load_r1, f_value, and_status, S);

endmodule
```

جواب سوال ۴:

برای حل این سوال نیاز به طراحی یک ماژول LUT داریم که مهمترین بخش این سوال است. این ماژول را می‌توان به یکی از حالت‌های زیر طراحی کرد.

۱- بهترین پیاده‌سازی

```
module LUT_1(input A, B, C, D, output out);
    reg [15:0] sram;
    wire [3:0] index;

    assign index = {D, C, B, A};
    assign out = sram[index];
endmodule
```

۲- بهترین پیاده‌سازی

```
module LUT_2(input A, B, C, D, output reg out);
    reg [15:0] sram;

    always @(A or B or C or D) begin
        out = sram[{D, C, B, A}];
    end
endmodule
```

۳-

```
module LUT_3(input A, B, C, D, output reg out);
    reg [15:0] sram;
    reg [3:0] index;

    always @(A or B or C or D) begin
        index = {D, C, B, A};
        case(index)
            0: out = sram[0];
            1: out = sram[1];
            2: out = sram[2];
            3: out = sram[3];
            4: out = sram[4];
            5: out = sram[5];
            6: out = sram[6];
            7: out = sram[7];
            8: out = sram[8];
            9: out = sram[9];
            10: out = sram[10];
            11: out = sram[11];
            12: out = sram[12];
            13: out = sram[13];
            14: out = sram[14];
            15: out = sram[15];
        endcase
    end
endmodule
```

۴-

```
module LUT_4(input A, B, C, D, output reg out);
    reg [15:0] sram;
    wire [3:0] index;

    assign index = {D, C, B, A};
    always @(index) begin
        case(index)
            0: out = sram[0];
            1: out = sram[1];
            2: out = sram[2];
            3: out = sram[3];
            4: out = sram[4];
            5: out = sram[5];
            6: out = sram[6];
            7: out = sram[7];
            8: out = sram[8];
            9: out = sram[9];
            10: out = sram[10];
            11: out = sram[11];
            12: out = sram[12];
            13: out = sram[13];
            14: out = sram[14];
            15: out = sram[15];
        endcase
    end
endmodule
```

امکان دارد یک نفر این ۱۶ حالت را با if-then-else هم پیاده‌سازی کرده باشد که کار صحیحی نیست ولی مورد قبول است.
در این ماژول وکتور sram همان واحدهای قابل برنامه‌ریزی هستند.

ماژول DFF:

```
module DFF (input d,
            input rst,
            input clk,
            output reg q);

    always @ (posedge clk)
       if (rst)
          q <= 0;
       else
          q <= d;
endmodule
```

ماژول مالتی پلکسر:

برای این ماژول یکی از ۳ حالت زیر قابل پیاده‌سازی است:
۱-

```
module multiplexer(input [1:0]in, input sel, output out);
    assign out = sel ? in[0] : in[1];
endmodule
```

=۲

```
module multiplexer(input [1:0]in, input sel, output reg out);
    always @(sel) begin
        if (sel == 1)
            out = in[0];
        else
            out = in[1];
    end
endmodule
```

-۳

```
module multiplexer(input [1:0]in, input sel, output reg out);
    always @(sel) begin
        case(sel)
            0: out = in[1];
            1: out = in[0];
        endcase
    end
endmodule
```

دراینجا امکان دارد عملیات مربوط به حالت صفر یا حالت یک با هم متفاوت باشد. مثل حالت زیر:

```
module multiplexer(input [1:0]in, input sel, output reg out);
    always @(sel) begin
        case(sel)
            0: out = in[0];
            1: out = in[1];
        endcase
    end
endmodule
```

در پایان باید یک ماژول CLB طراحی شود که می‌توان آن را به حالت‌های زیر طراحی کرد.

۱- اگر هر سه ماژول تعریف شده باشد، ماژول CLB به صورت زیر تعریف می‌شود.

```
module CLB(input A, B, C, D, input rst, clk, output out);
    reg sel;
    LUT lut(A, B, C, D, out_lut);
    DFF dff(out_lut, rst, clk, out_dff);
    multiplexer mux({out_lut, out_dff}, sel, out);
endmodule
```

۲- اگر فقط یک ماژول برای LUT تعریف شده باشد، می‌توان ماژول CLB را مانند زیر توصیف کرد.

```
module CLB(input A, B, C, D, input rst, clk, output out);
    reg sel;
    reg out_dff;

    LUT   lut(A, B, C, D, out_lut);

    assign out = sel ? out_lut : out_dff;

    always @(clk) begin
        if ( rst )
            out_dff = 0;
        else
            out_dff = out_lut;
    end

endmodule
```

۳-

```
module CLB(input A, B, C, D, input rst, clk, output reg out);
    reg sel;
    reg out_dff;

    LUT_4 lut(A, B, C, D, out_lut);

    always @(clk) begin
        if ( rst )
            out_dff = 0;
        else
            out_dff = out_lut;
    end

    always @(sel) begin
        if ( sel )
            out = out_lut;
        else
            out = out_dff;
    end

endmodule
```

امکان دارد LUT را هم در همین ماژول CLB توصیف کرده باشند که البته روش جالبی نیست.

واحدهای قابل پیکربندی در این طراحی sram در LUT و sel در CLB است (0.25 نمره).

توصیف LUT (۰.۷۵ نمره)

توصیف multiplexer (۰.۲۵ نمره)

توصیف DFF (نیم نمره)

توصیف کل CLB (۰.۲۵ نمره)