بنام خدا

تمرین دوم طراحی سیستم های دیجیتال

مدرس: **دکتر مدرسی**

اعضای گروه: **محمد محجل صادقی ۸۱۰۱۹۹۴۸۳**

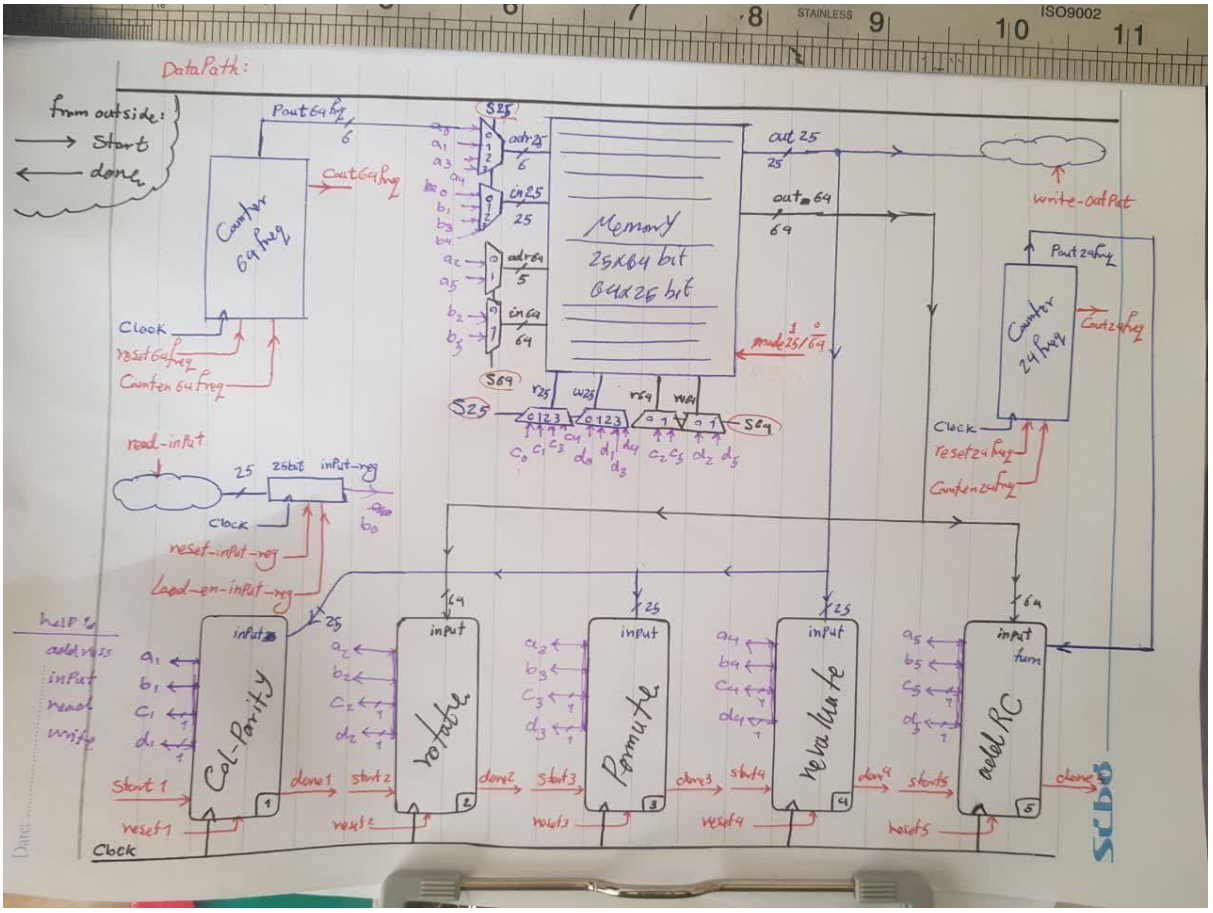
علی عطاءاللهی ۸۱۰۱۹۹۴۶۱

مقدمه

در این تمرین، ما یک encoder رمز گذاری را پیاده سازی می‌کنیم. برای این کار ۵ ماژول را به صورت ترتیبی روی ورودی اجرا می‌کنیم و این کار را ۲۵ بار تکرار می‌کنیم.

۳ تا از توابع قبلا پیاده سازی شده و دو تای دیگر را نیز پیاده سازی خواهیم کرد.

مسیر داده:



همانگونه که مشاهده می‌کنید مسیر داده از یک مموری تشکیل شده است که در دو حالت ۲۵ و ۶۴ بیتی می‌توان از آن استفاده کرد و ۵ ماژول با توجه به نیازشان از آنها استفاده می کنند و مسیر داده توسط mux هایی در هر مرحله، آن ماژول مورد نظر را به مموری وصل می‌کند.

از یک counter 64 freq برای شمردن ۶۴ ورودی و از یک کانتر ۲۴ تایی برای شمردن ۲۴ بار اجرای فرآیند استفاده می‌شود.

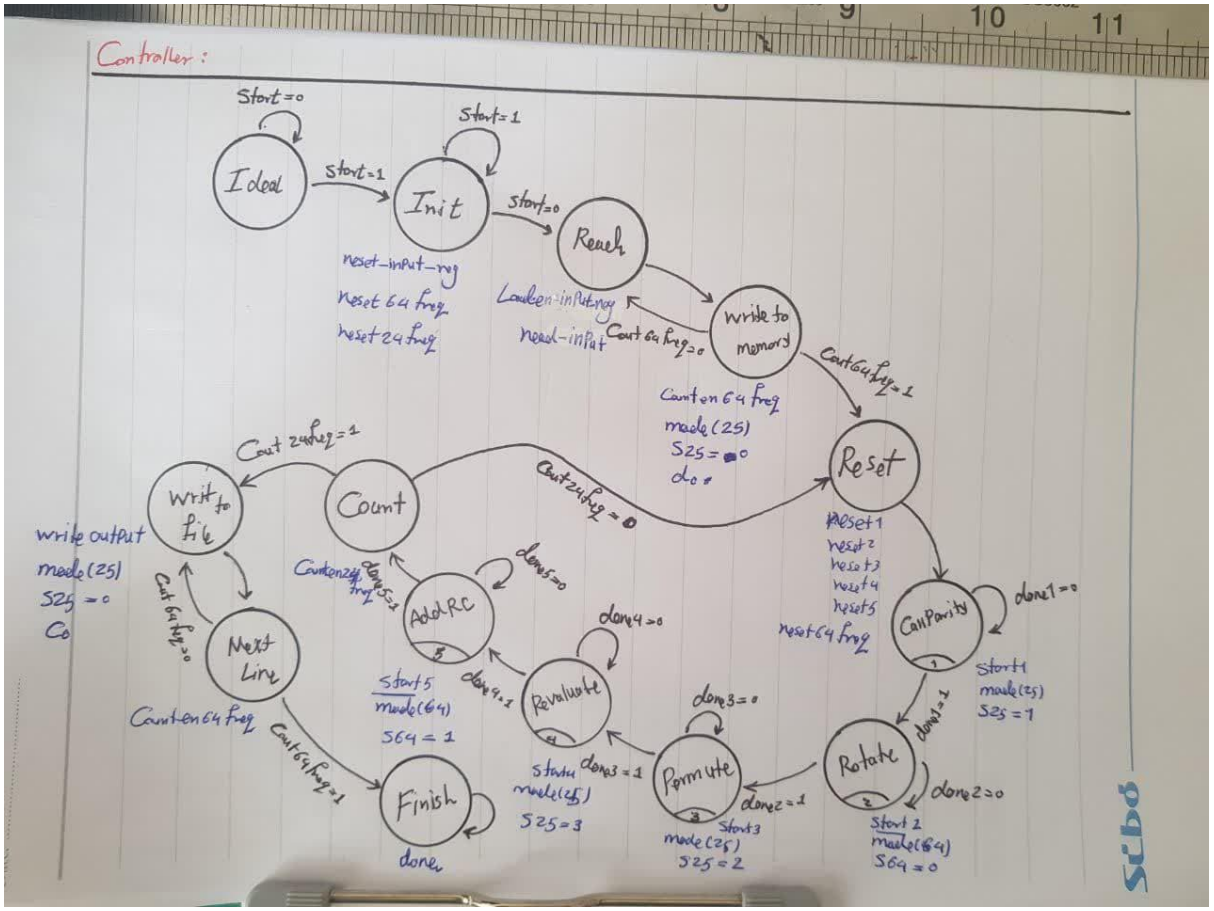
تمام ماژول ها یک بیت start ورودی و یک بیت done دارند که با دادن start شروع به کار می‌کنند و در پایان done داده می‌شود.

تابع addrc به اینکه در کدام سری اجرا نیاز هست هم نیاز دارد که با بیت turn به آن می‌دهیم.

کد مربوط به مسیر داده که دارای ۵ ماژول، یه مموری و mux ها مطابق شکل است را می‌توانید در زیر مشاهده نمایید.

```
 4
 5  reg clock = 0, start = 0;
 6  wire start_addrc, start_colparity, start_permute, start_revaluate, start_rotate;
 7  wire done, done_addrc, done_colparity, done_permute, done_revaluate, done_rotate;
 8  wire reset_addrc, reset_colparity, reset_permute, reset_revaluate, reset_rotate;
 9  wire mode; //remember to change this to 0 for 64 bit mode
10  wire sel64;
11  wire [1 : 0] sel25;
12  wire [4 : 0] turn;
13  wire wr_file;
14  wire count_en, reset_counter, cout;
15
16  wire [5 : 0] a0, a1, a3, a4, out_mux1;
17  wire [24 : 0] b0, b1, b3, b4, out_mux2;
18  wire [4 : 0] a2, a5, out_mux3;
19  wire [63 : 0] b2, b5, out_mux4;
20  wire c0, c1, c2, c3, c4, c5, d1, d2, d3, d4, d5, out_mux5, out_mux6, out_mux7, out_mux8;
21
22  wire [0:24]out25;
23  wire [0:63]out64;
24
25  mux_4 #(.bits(6)) m1(a0, a1, a3, a4, out_mux1, sel25);
26  mux_4 #(.bits(25)) m2(b0, b1, b3, b4, out_mux2, sel25);
27
28  mux_2 #(.bits(5)) m3(a2, a5, out_mux3, sel64);
29  mux_2 #(.bits(64)) m4(b2, b5, out_mux4, sel64);
30
31  mux_4 #(.bits(1)) m5(c0, c1, c3, c4, out_mux5, sel25);
32  mux_4 #(.bits(1)) m6(d0, d1, d3, d4, out_mux6, sel25);
33
34  mux_2 #(.bits(1)) m7(c2, c5, out_mux7, sel64);
35  mux_2 #(.bits(1)) m8(d2, d5, out_mux8, sel64);
36
37
38  Memory #(.input_file("input_1.txt")) Memory_ (…
53  );
54
55  Colparity Colparity_ (…
65  );
66
67  Rotate Rotate_ (…
77  );
78
79  Permute Permute_ (…
89  );
90
91  Revaluate Revaluate_ (…
101 );
102
103 Addrc Addrc_ (…
114 );
115
116 counter24 counter6bit_(…
122 );
123
124 cu cu_
125 (…
138 );
139
140
141
```

کنترلر:



نحوه ی کار کنترلر بدین گونه است که پس از گرفتن یک پالس استارت، ابتدا ورودی را سطر به سطر و ۲۵ بیت، ۲۵ بیت از فایل ورودی می‌گیرد و در مموری می‌نویسد.

برای این کار از کانتر ۶۴ تایی استفاده می‌کند.

پس از آن، داخل یک لوپ می‌افتد:

- یکی یکی سیگنال های start پنج ماژول را فعال می‌کنیم و منتظر می‌مانیم تا تمام شوند
- سپس این کار را ۲۴ بار ادامه می‌دهیم.
- برای این کار از کانتر ۲۴ بیت استفاده می‌کنم.

پس از اتمام، باز هم ۲۵ بیت ۲۵ بیت و سطر به سطر از مموری در فایل خروجی می‌نویسیم و تمام!

به استیت پایان می رویم و سیگنال پایان را فعال می‌کنیم.

کد مربوط به کنترلر را در زیر مشاهده می‌کنید که مدل هافمن و جدا کردن مدل ترتیبی و ترکیبی رعایت شده و از سه حلقه ی always استفاده شده.

```
// In the Name of God

module cu
(
    input clock, start,
    input done_addrc, done_colparity, done_permute, done_revaluate, done_rotate,
    input [4 : 0] turn,
    input cout,
    output reg start_addrc, start_colparity, start_permute, start_revaluate, start_rotate,
    output reg reset_addrc, reset_colparity, reset_permute, reset_revaluate, reset_rotate,
    output reg mode, sel64,
    output reg [1 : 0] sel25,
    output reg wr_file,
    output reg done,
    output reg count_en,
    output reg reset_counter
);
    reg [4:0] ps, ns;
    parameter Idle = 0, Init = 1, ResetColParity = 2, StartColParity = 3, DoneColParity = 4, ResetRotate = 5,
    StartRotate = 6, DoneRotate = 7, ResetPermute = 8, StartPermute = 9, DonePermute = 10, ResetRevaluate = 11, StartRevaluate = 12, DoneRevaluate = 13,
    ResetAddRc = 14, StartAddRc = 15, DoneAddRc = 16, Finish = 17;

    always @(posedge clock) begin
        ps <= ns;
    end

    always @(ps, start, cout, done_addrc, done_colparity, done_permute, done_revaluate, done_rotate) begin
        ns = Idle;
        case (ps)
            Idle: ns = start ? Init : Idle;
            Init: ns = start ? Init : ResetColParity;
            ResetColParity: ns = StartColParity;
            StartColParity: ns = done_colparity ? DoneColParity : StartColParity;
            DoneColParity: ns = ResetRotate;

            ResetRotate: ns = StartRotate;
            StartRotate: ns = done_rotate ? DoneRotate : StartRotate;
            DoneRotate: ns = ResetPermute;

            ResetPermute: ns = StartPermute;
            StartPermute: ns = done_permute ? DonePermute : StartPermute;
            DonePermute: ns = ResetRevaluate;

            ResetRevaluate: ns = StartRevaluate;
            StartRevaluate: ns = done_revaluate ? DoneRevaluate : StartRevaluate;
            DoneRevaluate: ns = ResetAddRc;

            ResetAddRc: ns = StartAddRc;
            StartAddRc: ns = done_addrc ? DoneAddRc : StartAddRc;
            DoneAddRc: ns = cout ? Finish : ResetColParity;

            Finish: ns = Finish;
            default: ns = Idle;
        endcase
    end

    always @(ps,start,cout) begin

        case (ps)
        endcase
    end
    assign state = ps;

endmodule
```

توضیح ماژول ها:

memory:

همانگونه که قبلا گفته شد، مسیر داده از یک مموری تشکیل شده است که در دو حالت ۲۵ و ۶۴ بیتی می‌توان از آن استفاده کرد و ۵ ماژول با توجه به نیازشان از آنها استفاده می کنند و مسیر داده توسط mux هایی در هر مرحله، آن ماژول مورد نظر را به مموری وصل می‌کند.

همچنین یک تست برای اطمینان از درستی اش نوشته شد.

در زیر می‌توانید کد مربوط به مموری و تست را مشاهده نمایید:

```
module Memory #(parameter input_file)
(
    input [5:0]adr25,
    input [0:24]in25,
    input r25,
    input w25,
    output reg [0:24]out25,

    input [4:0]adr64,
    input [0:63]in64,
    input r64,
    input w64,
    output reg [0:63]out64,

    input mode,
    input wr_file
);
    // craete 64x25 bit memory
    reg [0:24] memory[0:63];



    //for testing
    initial begin
        $readmemb(input_file, memory);
    end

    integer i;

    parameter Mode64 = 0;
    parameter Mode25 = 1;

    always @(adr25 or in25 or r25 or w25 or adr64 or in64 or r64 or w64 or mode)begin
        case (mode)
            Mode64: begin
                if (r64) begin
                    for (i = 0; i < 64; i = i + 1) begin
                        out64[i] <= memory[i][adr64];
                    end
                end
                if (w64) begin
                    for (i = 0; i < 64; i = i + 1) begin
                        memory[i][adr64] <= in64[i];
                    end
                end
            end
            Mode25: begin
                if (r25) begin
                    out25 <= memory[adr25];
                end
                if (w25) begin
                    memory[adr25] <= in25;
                end
            end
        endcase
    end
    always @(wr_file) begin
        if(wr_file) begin
            $writememb("output.txt", memory);
        end
    end
endmodule
```

```
timescale 1ns/1ns

module tb_memory ();

reg [5:0]adr25;
reg [0:24]in25;
reg r25;
reg w25;
wire [0:24]out25;

reg [4:0]adr64;
reg [0:63]in64;
reg r64;
reg w64;
wire [0:63]out64;

reg mode;

Memory uut (
    .adr25(adr25),
    .in25(in25),
    .r25(r25),
    .w25(w25),
    .out25(out25),

    .adr64(adr64),
    .in64(in64),
    .r64(r64),
    .w64(w64),
    .out64(out64),

    .mode(mode)
);

initial begin
    adr25 = 0;
    in25 = 0;
    r25 = 0;
    w25 = 0;

    adr64 = 0;
    in64 = 0;
    r64 = 0;
    w64 = 0;

    mode = 0;
end

initial begin
    //write with 25
    adr25 = 5'd3;
    in25 = 25'b00000_11101_00000_00000_00000;
    r25 = 0;
    w25 = 1;
    mode = 1;
    #100
    //write with 64
    adr64 = 4'd3;
    in64 = 64'b1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_1111_0101_0101;
    r64 = 0;
    w64 = 1;
    mode = 0;
    #100
    //read with 64
```

Rotate:

با فعال کردن سیگنال start، تابع از یک کانتر ۲۵ تایی استفاده می‌کند و در هر کلاک، ۶۴ بیت یک لاین را میگیرد.

سپس با استفاده از بیت مورد استفاده، مقدار x ,y این لاین را حساب می‌کند.

y/x	x = 0	x = 1	x = 2	x = 3	x = 4
y = 4	18	2	61	56	14
y = 3	41	45	15	21	8
y = 2	3	10	43	25	39
y = 1	36	44	6	55	20
y = 0	0	1	62	28	27

پس از آن توسط جدول زیر، که به ازای هر y, x مقدار شیفت را مشخص می‌کند، شروع به شیفت دادن آن لاین می‌کند.

در آخر نیز نتیجه را در همان لاین می‌نویسد.

در زیر می‌توانید کد مربوط به Rotate و تست را مشاهده نمایید:

```
1  `timescale 1ns/1ns
2
3  module Rotate
4  (
5      input [0:63] in,
6      input start,
7      input clock,
8      input reset,
9      output reg done,
10     output reg [4:0] mem_adr,
11     output reg [0:63] mem_in,
12     output reg mem_r,
13     output reg mem_w
14 );
15
16 integer i, x, y, final_index, lane = 0;
17
18 reg [0:63] in_temp;
19
20 reg [5:0] f [0:4] [0:4];
21
22 initial begin
23     f[0][0] = 0;
24     f[0][1] = 36;
25     f[0][2] = 3;
26     f[0][3] = 41;
27     f[0][4] = 18;
28     f[1][0] = 1;
29     f[1][1] = 44;
30     f[1][2] = 10;
31     f[1][3] = 45;
32     f[1][4] = 2;
33     f[2][0] = 62;
34     f[2][1] = 6;
35     f[2][2] = 43;
36     f[2][3] = 15;
37     f[2][4] = 61;
38     f[3][0] = 28;
39     f[3][1] = 55;
40     f[3][2] = 25;
41     f[3][3] = 21;
42     f[3][4] = 56;
43     f[4][0] = 27;
44     f[4][1] = 20;
45     f[4][2] = 39;
46     f[4][3] = 8;
47     f[4][4] = 14;
48 end
49
50 reg [5:0] rotate_f;
51
52 always @(posedge clock, posedge reset) begin
53     if (reset) begin
54         lane = 0;
55     end
56     else if (start && lane < 25) begin
57         mem_r = 1;
58         mem_adr = lane;
59         mem_w = 0;
60
61         x = ((lane % 5) + 3) % 5;
62         y = ((lane / 5) + 3) % 5;
63
64         rotate_f = f[x][y];
```

```
`timescale 1ns/1ns

module tb_rotate ();

reg clock = 0;
reg start = 0;
reg reset = 0;
reg mode = 0; //remember to change this to 0 for 64 bit mode


wire [5:0]adr25;
wire [0:24]in25;
wire r25;
wire w25;
wire [0:24]out25;

wire [4:0]adr64;
wire [0:63]in64;
wire r64;
wire w64;
wire [0:63]out64;


Memory m1 (
    .adr25(adr25),
    .in25(in25),
    .r25(r25),
    .w25(w25),
    .out25(out25),

    .adr64(adr64),
    .in64(in64),
    .r64(r64),
    .w64(w64),
    .out64(out64),

    .mode(mode)
);

Rotate r1 (
    .in(out64),
    .start(start),
    .clock(clock),
    .done(done),
    .reset(reset),
    .mem_adr(adr64),
    .mem_in(in64),
    .mem_r(r64),
    .mem_w(w64)
);

always #200 clock = ~clock;

initial begin
    #100
    start = 1;
    #50000
    $stop;
end


endmodule
```

addRc:

این تابع، لاین $x = 0, y = 0$ را با جدول زیر XOR می‌کند.

RC[0] = 0x0000000000000001	RC[12] = 0x000000008000808B
RC[1] = 0x0000000000008082	RC[13] = 0x800000000000008B
RC[2] = 0x800000000000808A	RC[14] = 0x8000000000008089
RC[3] = 0x8000000080008000	RC[15] = 0x8000000000008003
RC[4] = 0x000000000000808B	RC[16] = 0x8000000000008002
RC[5] = 0x0000000080000001	RC[17] = 0x8000000000000080
RC[6] = 0x8000000080008081	RC[18] = 0x000000000000800A
RC[7] = 0x8000000000008009	RC[19] = 0x800000008000000A
RC[8] = 0x000000000000008A	RC[20] = 0x8000000080008081
RC[9] = 0x0000000000000088	RC[21] = 0x8000000000008080
RC[10] = 0x0000000080008009	RC[22] = 0x0000000080000001
RC[11] = 0x000000008000000A	RC[23] = 0x8000000080008008

این لاین در ۲۵ لاین ورودی برابر با لاین ۱۲ می‌باشد. (لاین ۱۳ ام)

این تابع به عنوان ورودی، turn را می‌گیرد و بر حسب آن تصمیم می‌گیرد که از کدام یک از مقادیر جدول استفاده کند.

پس از XOR کردن، نتیجه را در همان لاین ورودی می‌نویسیم.

کد مربوط به addrc و تست آن را در زیر مشاهده می‌کنید.

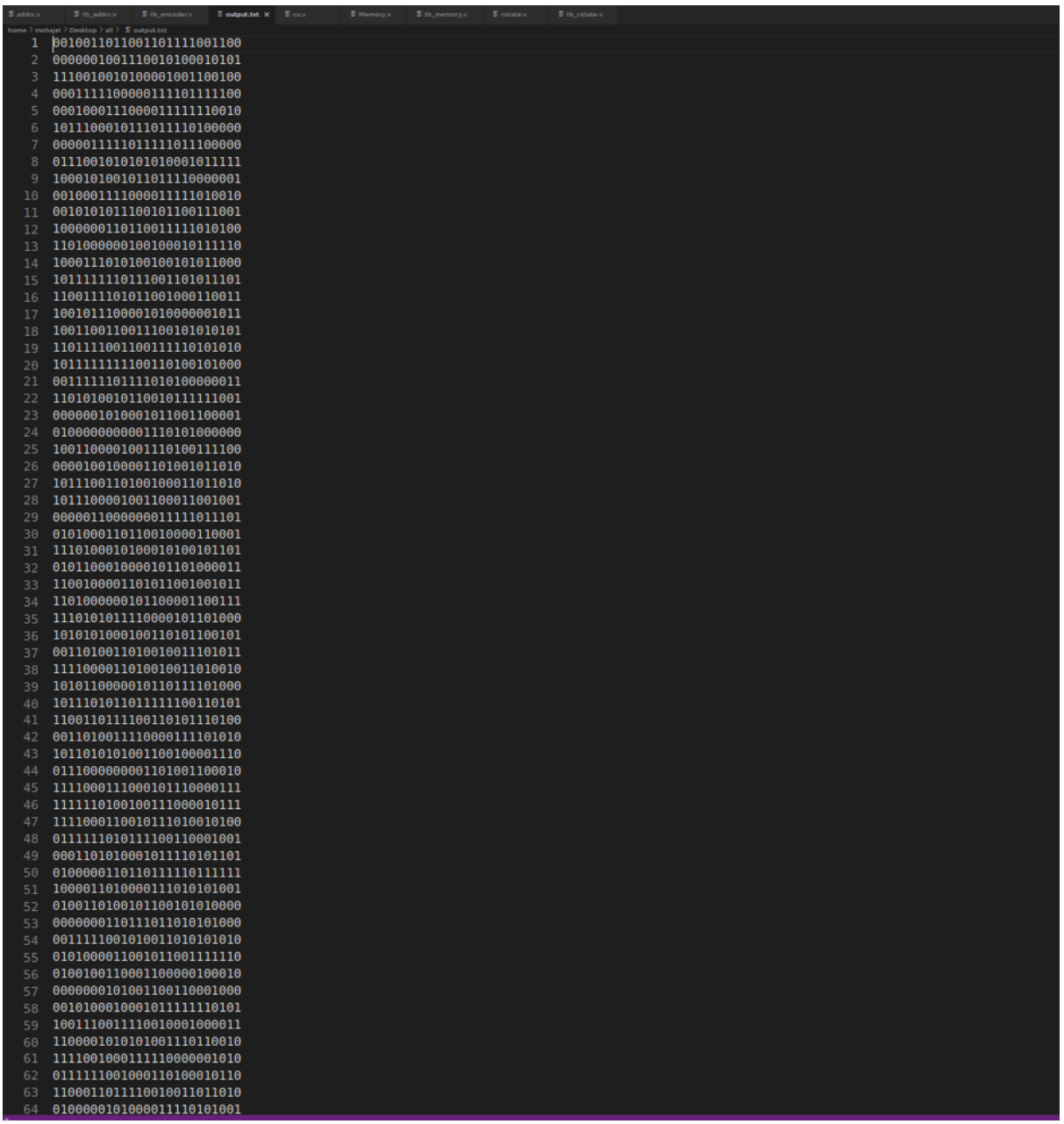
```
`timescale 1ns/1ns

module Addrc
(
    input [0:63] in,
    input start,
    input clock,
    input reset,
    input [4:0] turn,
    output reg done,
    output reg [4:0] mem_adr,
    output reg [0:63] mem_in,
    output reg mem_r,
    output reg mem_w
);

integer i, x, y, final_index, lane = 0;

reg [0:63] in_temp;

reg [0:63] rc [0:23];

initial begin
    done = 0;
    rc[0]  = 64'h0000000000000001;
    rc[1]  = 64'h0000000000008082;
    rc[2]  = 64'h800000000000808A;
    rc[3]  = 64'h8000000080008000;
    rc[4]  = 64'h000000000000808B;
    rc[5]  = 64'h0000000080000001;
    rc[6]  = 64'h8000000080008081;
    rc[7]  = 64'h8000000000008009;
    rc[8]  = 64'h000000000000008A;
    rc[9]  = 64'h0000000000000088;
    rc[10] = 64'h0000000080008009;
    rc[11] = 64'h000000008000000A;
    rc[12] = 64'h000000008000808B;
    rc[13] = 64'h800000000000008B;
    rc[14] = 64'h8000000000008089;
    rc[15] = 64'h8000000000008003;
    rc[16] = 64'h8000000000008002;
    rc[17] = 64'h8000000000000080;
    rc[18] = 64'h000000000000800A;
    rc[19] = 64'h800000008000000A;
    rc[20] = 64'h8000000080008081;
    rc[21] = 64'h8000000000008080;
    rc[22] = 64'h0000000080000001;
    rc[23] = 64'h8000000080008008;
end

always @(posedge clock, posedge reset) begin
    if (reset) begin
        done = 0;
    end
    else if (start && ~done) begin
        mem_r = 1;
        mem_adr = 5'd12;
        mem_w = 0;
        #3;
        in_temp = in;
        #2;
        in_temp = in_temp ^ rc[turn];
        #2
        mem_in = in_temp;
```

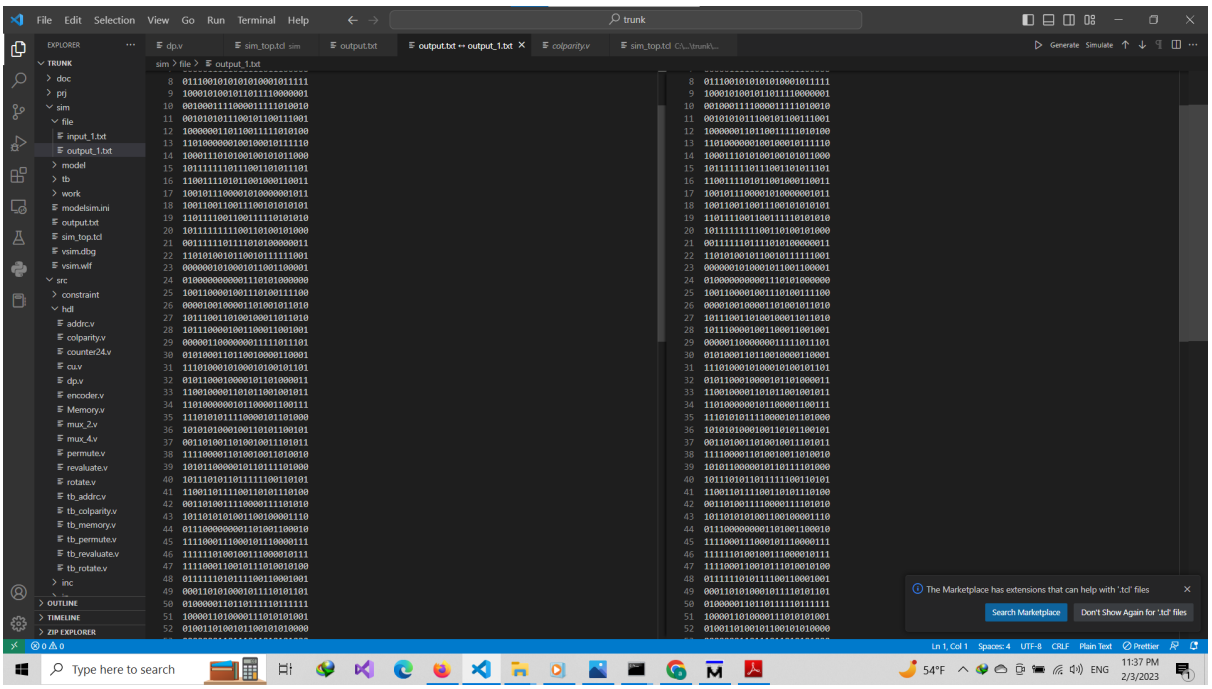
```
`timescale 1ns/1ns

module tb_addrc ();

reg clock = 0;
reg start = 0;
reg reset = 0;
reg mode = 0; //remember to change this to 0 for 64 bit mode
reg [4:0] turn = 5'd0;


wire [5:0]adr25;
wire [0:24]in25;
wire r25;
wire w25;
wire [0:24]out25;

wire [4:0]adr64;
wire [0:63]in64;
wire r64;
wire w64;
wire [0:63]out64;


Memory m1 (
    .adr25(adr25),
    .in25(in25),
    .r25(r25),
    .w25(w25),
    .out25(out25),

    .adr64(adr64),
    .in64(in64),
    .r64(r64),
    .w64(w64),
    .out64(out64),

    .mode(mode)
);

Addrc a1 (
    .in(out64),
    .start(start),
    .clock(clock),
    .done(done),
    .reset(reset),
    .turn(turn),
    .mem_adr(adr64),
    .mem_in(in64),
    .mem_r(r64),
    .mem_w(w64)
);

always #100 clock = ~clock;

initial begin
    #400
    start = 1;
    #5000
    $stop;
end
```

نتایج:

فقط کافیست تست بنچ بنویسیم و در آن سیگنال های کلاک و start را به مدل اصلی بدهیم.

در زیر می‌توانید تست بنچ و نتیجه ی نهایی پس از ۲۴ بار و اتمام پروسه و نوشته شدن نتیجه در فایل خروجی مشاهده می‌کنید که با نتیجه ی داده شده یکسان است.


addrc.v
tb_addrc.v
tb_encoder.v
output.txt
su.v
Memory.v
tb_memory.v
rotate.v
tb_rotate.v
home > mohajel > Desktop > ali > output.txt
1 0010011011001101111001100
2 0000001001110010100010101
3 1110010010100001001100100
4 0001111100000111101111100
5 0001000111000011111110010
6 1011100010111011110100000
7 0000011111011111011100000
8 0111001010101010001011111
9 1000101001011011110000001
10 0010001111000011111010010
11 0010101011100101100111001
12 1000000110110011111010100
13 1101000000100100010111110
14 1000111010100100101011000
15 1011111110111001101011101
16 1100111101011001000110011
17 1001011100001010000001011
18 1001100110011100101010101
19 1101111001100111110101010
20 1011111111100110100101000
21 0011111101111010100000011
22 1101010010110010111111001
23 0000001010001011001100001
24 0100000000001110101000000
25 1001100001001110100111100
26 0000100100001101001011010
27 1011100110100100011011010
28 1011100001001100011001001
29 0000011000000001111011101
30 0101000110110010000110001
31 1110100010100010100101101
32 0101100010000101101000011
33 1100100001101011001001011
34 1101000000101100001100111
35 1110101011110000101101000
36 1010101000100110101100101
37 0011010011010010011101011
38 1111000011010010011010010
39 1010110000010110111101000
40 1011101011011111100110101
41 1100110111100110101110100
42 0011010011110000111101010
43 1011010101001100100001110
44 0111000000001101001100010
45 1111000111000101110000111
46 1111110100100111000010111
47 1111000110010111010010100
48 0111111010111100110001001
49 0001101010001011110101101
50 0100000110110111110111111
51 1000011010000111010101001
52 0100110100101100101010000
53 0000000110111011010101000
54 0011111001010011010101010
55 0101000011001011001111110
56 0100100110001100000100010
57 0000000101001100110001000
58 0010100010001011111110101
59 1001110011110010001000011
60 1100001010101001110110010
61 1111001000111110000001010
62 0111111001000110100010110
63 1100011011110010011011010
64 0100000101000011110101001


File Edit Selection View Go Run Terminal Help
trunk
output.txt ↔ output_1.txt
The Marketplace has extensions that can help with '.tcl' files
Search Marketplace
Don't Show Again for '.tcl' files
Type here to search
54°F
ENG
11:37 PM
2/3/2023

پایان