به نام خدا

محمد محجل صادقی 810199483

علی عطاءاللهی 810199461

تمرین اول طراحی سیستم های دیجیتال (CAD)

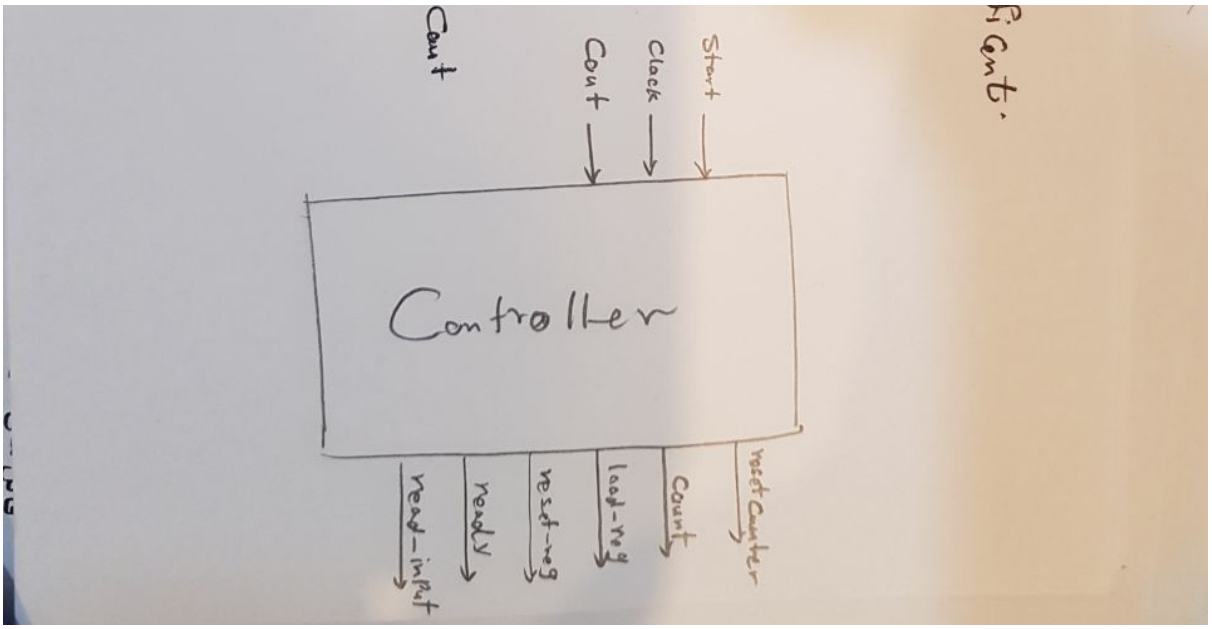
استاد مدرسی

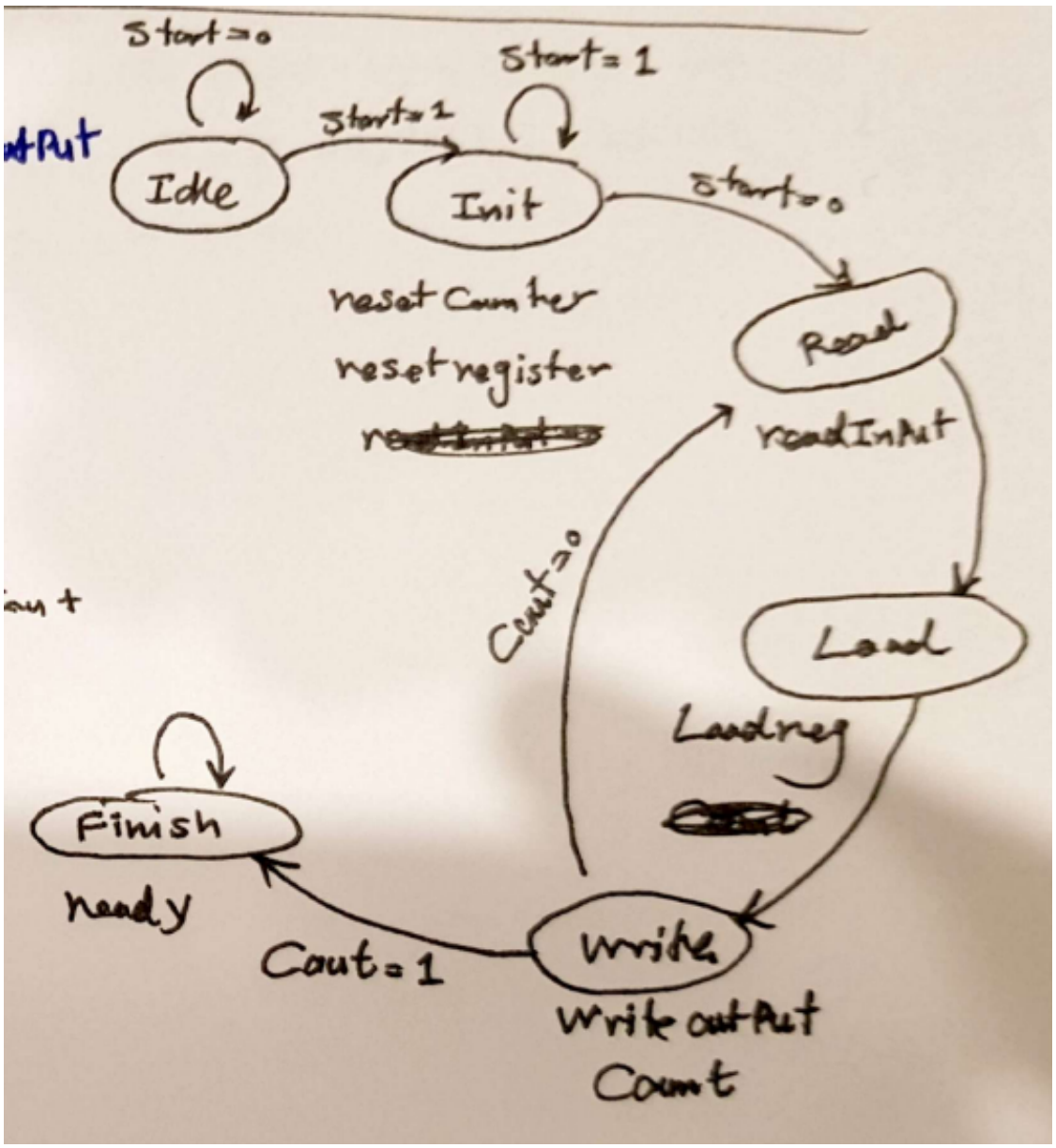
توضیح controller:

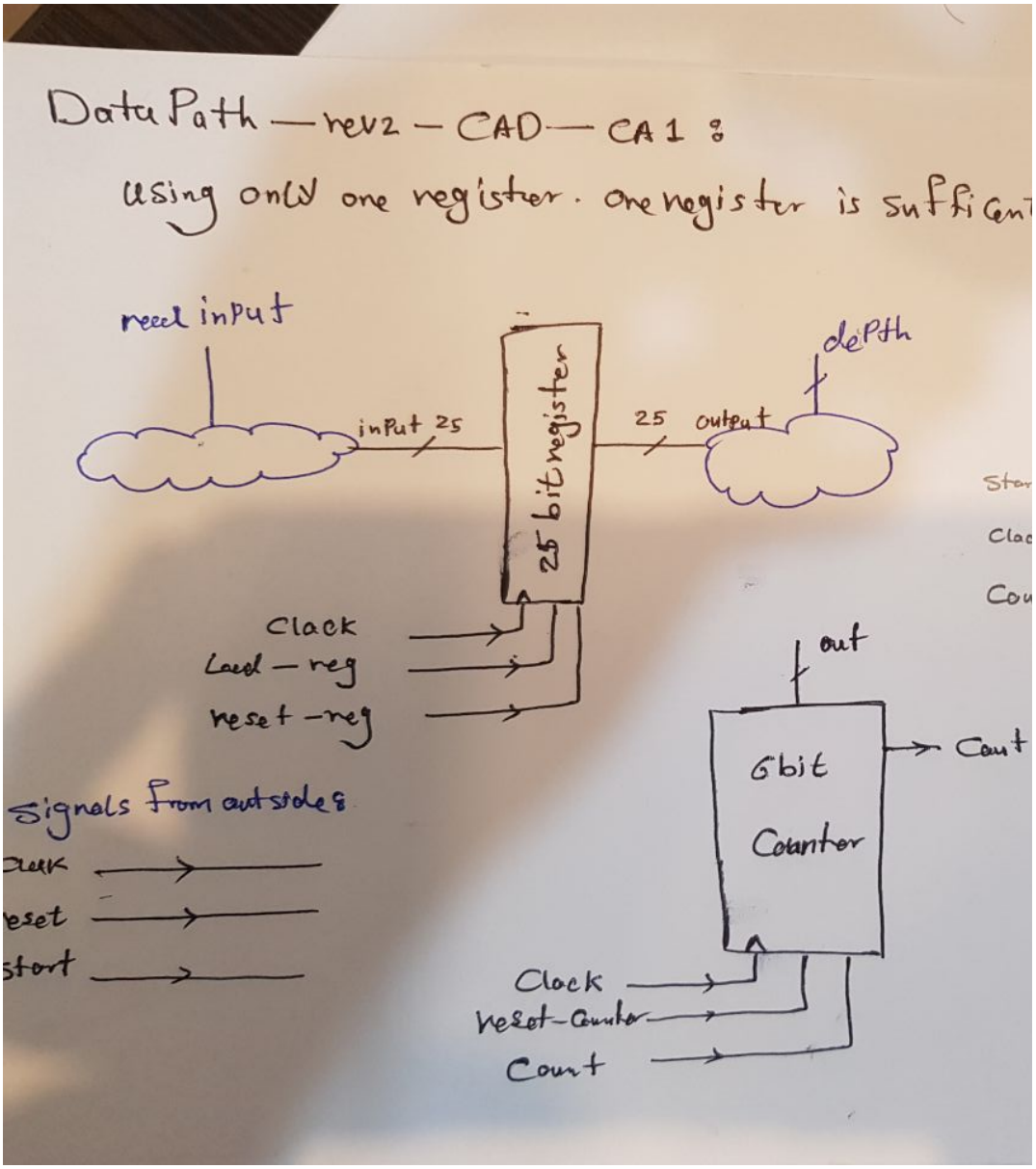
همانطور که مشاهده می‌کنید، پس از دریافت یک پالس در سیگنال start ، کنترل در یک لوپ می افتد و به اندازه ی تعداد سطر های ورودی ابتدا از فایل ورودی با فعال کردن سیگنال read_input ورودی را می خواند.

پس از آن، خروجی مپ شده را در رجیستر وسط ذخیره می کند و در آخر با فعال کردن سیگنال write_output ، خروجی را در فایل خروجی ذخیره می‌کند.

پس از پایان عملیات در استیت Finish باقی می‌مانیم و سیگنال ready را issue می‌کنیم.




Start=0
Start=1
Start=1
Idle
Init
Start=0
reset Counter
reset register
Read
readInPut
Cout=0
Load
Loadreg
Finish
ready
Write
Cout=1
Write outPut
Count



کنترلر در فایل cu.v نوشته شده است که می توانید مشاهده نمایید.

دقت کنید طبق توصیه های درسی، در نوشتن کنترلر بخش های دارای حافظه و بدون حافظه را از هم تفکیک کرده ایم و از سه حلقه ی always استفاده شده است.

تفکیک combinational part و sequential part را می‌توانید در زیر مشاهده کنید.

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

always @(ps,start,cout) begin
    ns = Idle;
    case (ps)
        Idle: ns = start ? Init : Idle;
        Init: ns = start ? Init : Read;
        Read: ns = Load;
        Load: ns = Write;
        Write: ns = cout ? Finish : Read;
        Finish: ns = Finish;
        default: ns = Idle;
    endcase
end

always @(ps,start,cout) begin

    case (ps)
        Idle:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b0000000;
        end
        Init:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b1001000;
        end
        Read:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b0000010;
        end
        Load:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b0010000;
        end
        Write:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b0100001;
        end
        Finish:begin
            {reset_counter, count, load_reg, reset_reg, ready, read_input, write_output} = 7'b0000100;
        end
    endcase
end
```

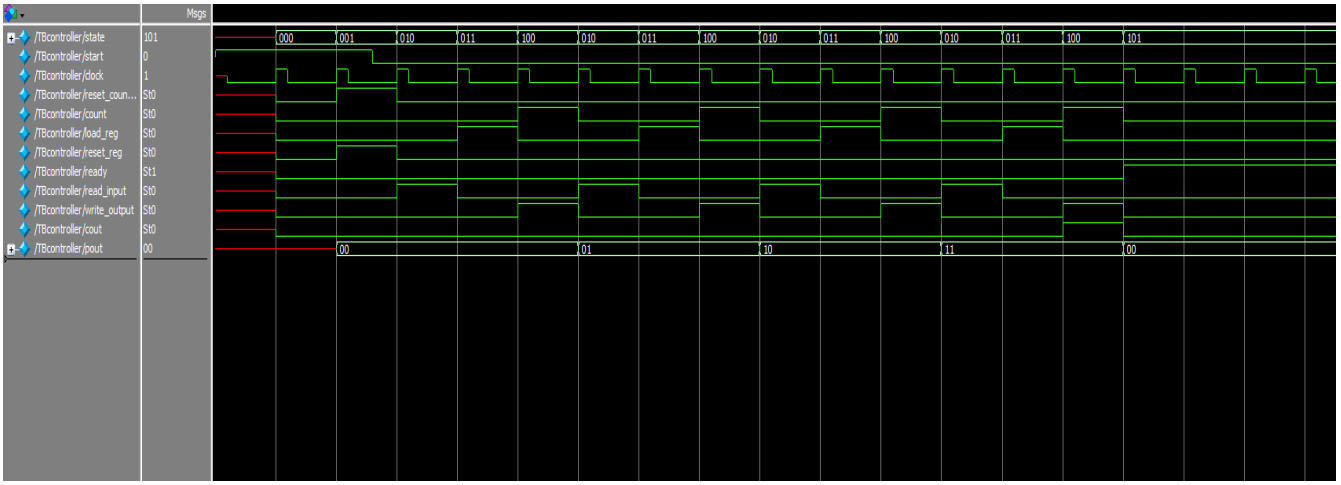
برای تست کردن‌کنترلر یک تست بنچ نوشته شد درفایلTBcontroller.v

و از درست کار کردن آن اطمینان حاصل شد.

```
//in the name of God

`timescale 1ns/1ns

module TBcontroller ();


    wire [2:0] state;
    reg start = 1 , clock;
    wire reset_counter, count, load_reg, reset_reg,ready, read_input, write_output, cout;

    Controller c1
    (
        .start(start),
        .clock(clock),
        .cout(cout),
        .reset_counter(reset_counter),
        .count(count),
        .load_reg(load_reg),
        .reset_reg(reset_reg),
        .ready(ready),
        .read_input(read_input),
        .write_output(write_output),
        .state(state)
    );

    wire [1:0] pout;

    Counter conter1
    (
        .count_en(count),
        .clock(clock),
        .reset(reset_counter),
        .pout(pout),
        .cout(cout)
    );

    always begin
        #10 clock = 0;
        #40 clock = 1;
    end

    initial begin
        #130 start = 0;
        #10000 $stop;
    end


endmodule
```

نتیجه را می توانید در زیر ببینید:

(از counter 2bit استفاده شده تا بتوان راحت تر دنبال کرد.)



توضیحات بخش datapath:

در این قسمت از یک رجیستر استفاده شده که در آن با هر بار زده شدن read_input (در واقع همان خواندن از رجیستر و ریختن داخل فایل است.) یک لاین از فایل ورودی گرفته شده داخل فایل output نوشته می‌شود. روش بدین صورت است که ابتدا از فایل input در یک آرایه ذخیره و سپس آن را در یک رجیستر نگه‌داری می‌کنیم. حال با استفاده از فرمول هایی که برای جایگشت استفاده کرده ایم خروجی را ایجاد می‌کنیم و در فایل نهایی میریزیم.

همچنین نام فایل ورودی و نام فایل خروجی هر دو در تست بنچ داده می‌شود.

```
`timescale 1ns/1ns

module  dp #(parameter read_file_name, write_file_name)(
    clk,
    rst,
    ld,
    read_input,
    out
);

    input clk, rst, read_input;
    input ld;
    output [24:0] out;
    reg [24:0] in_arr_temp, in;
    integer i, x, y, final_index, depth = 0;
    integer read_file_buffer, scan_file, write_file_buffer;

    initial begin
        depth = 0;
        read_file_buffer = $fopen(read_file_name, "r");
        write_file_buffer = $fopen(write_file_name, "w");
        $fclose(read_file_name);
        $fclose(write_file_name);
    end

    always @(depth) begin
        if (!(^in === 1'bx))
            $fdisplay(write_file_buffer, "%b", in);
        scan_file = $fscanf(read_file_buffer, "%b\n", in_arr_temp);
    end

    always @(read_input) begin : computation
        for (i = 0 ; i < 25 ; i = i + 1) begin
            x = ((i % 5) + 3) % 5;
            y = ((i / 5) + 3) % 5;
            final_index = ((y + 2) % 5) + ((((((2 * x) + (3 * y)) % 5) + 2) % 5 ) * 5);
            in[final_index] = in_arr_temp[i];
        end
        if (depth < 64)
            depth = depth + 1;
    end
```

رجیستر استفاده شده (25 بیت) :

```verilog
module reg25bit(
    in,
    out,
    clk,
    rst,
    ld
);
    input [24 : 0] in;
    output [24 : 0] out;
    reg [24 : 0] out_;
    input clk, rst;
    input ld;

    always @(posedge clk, posedge rst) begin
        if (rst)
            out_ <= 25'd0;
        else
            if (ld)
                out_ <= in;
    end
    assign out = out_;

endmodule
```

تغییرات فایل sim_top.tcl:

در این قسمت نام فایل های وریلاگ خود و همچنین نام تست بنچ را مشخص کرده‌ایم.

```
    set TB                  "testbench"
    set hdl_path            "../src/hdl"
    set inc_path            "../src/inc"

    set run_time            "1 us"
#   set run_time            "-all"

#========================== Add verilog files  ================================
# Pleas add other module here
    vlog    +acc -incr -source  +define+SIM     $hdl_path/cu.v
    vlog    +acc -incr -source  +define+SIM     $hdl_path/dp.v
    vlog    +acc -incr -source  +define+SIM     $hdl_path/counter6bit.v
    vlog    +acc -incr -source  +define+SIM     $hdl_path/permutation_func.v
    vlog    +acc -incr -source  +define+SIM     $hdl_path/reg25bit.v

    vlog    +acc -incr -source  +incdir+$inc_path +define+SIM   ./tb/$TB.v
    onerror {break}
```

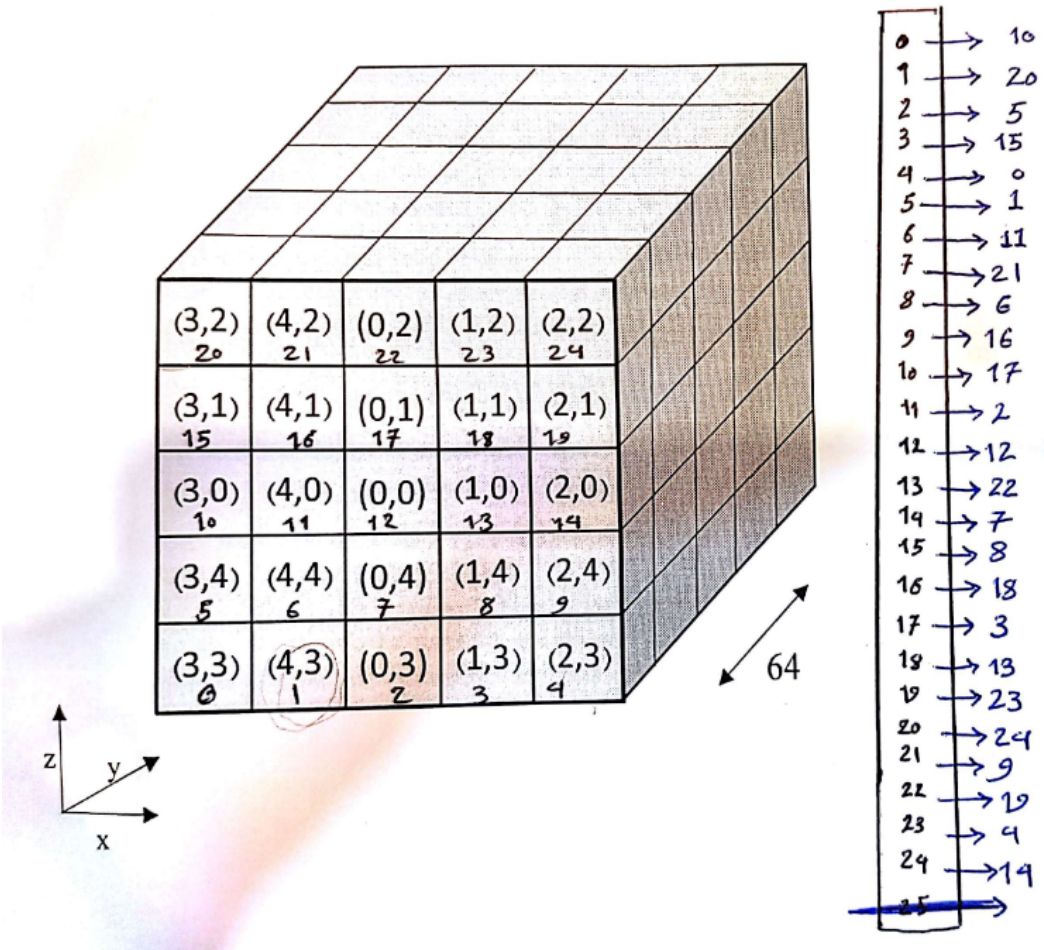
برای ساده تر شدن اجرای کد ، می‌توانیم از دستور do sim_top.tcl استفاده کنیم.

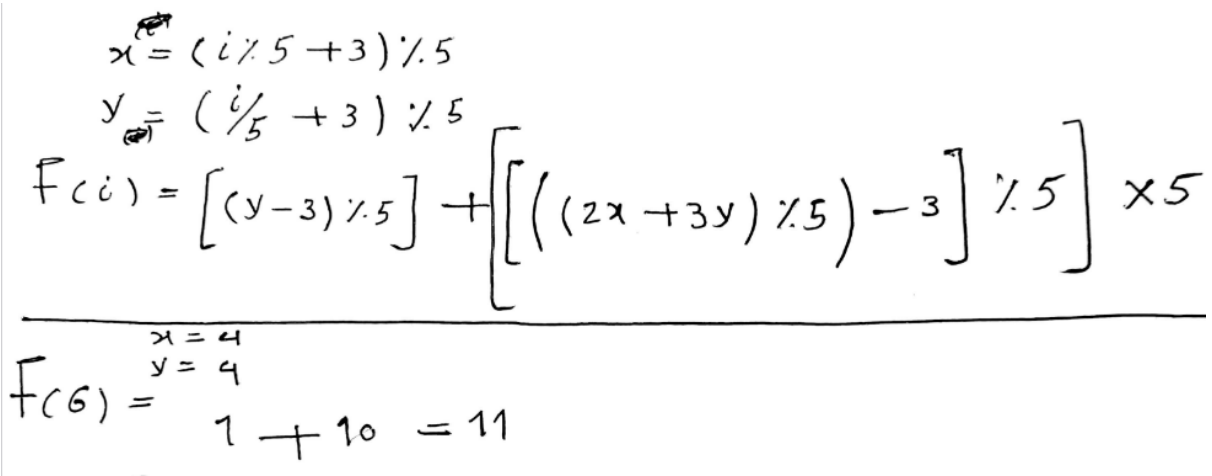
بخش امتیازی:

همانگونه که قبلا اشاره و توضیح داده شد، برای مپ کردن بیت های رجیستر ها بجای hard-wire ، از فرمول اولیه و نتیجه آن استفاده شد.

```
x = ((i % 5) + 3) % 5;
y = ((i / 5) + 3) % 5;
final_index = ((y + 2) % 5) + ((((((2 * x) + (3 * y)) % 5) + 2) % 5 ) * 5);
in2[final_index] = out1[i];
```

بدست آوردن فرمول:





نتایج تست:

از test-bench زیر برای اجرا کد استفاده می کنیم:

```
1   `timescale 1ns/1ns
2
3   module tb();
4       reg clk = 1'b1, rst = 1'b0, start = 1'b0;
5       permutation_func #("input_3.txt", "output_3.txt") pf(clk, rst, start);
6       initial begin
7           repeat(1000) #50 clk = ~clk;
8       end
9       initial begin
10          #110 start = 1'b1;
11          #110 start = 1'b0;
12          #10000 $stop;
13      end
14  endmodule
```

همانطور که می بیند، نام فایل ورودی و خروجی را به عنوان پارامتر به permutatton_func می‌دهیم.

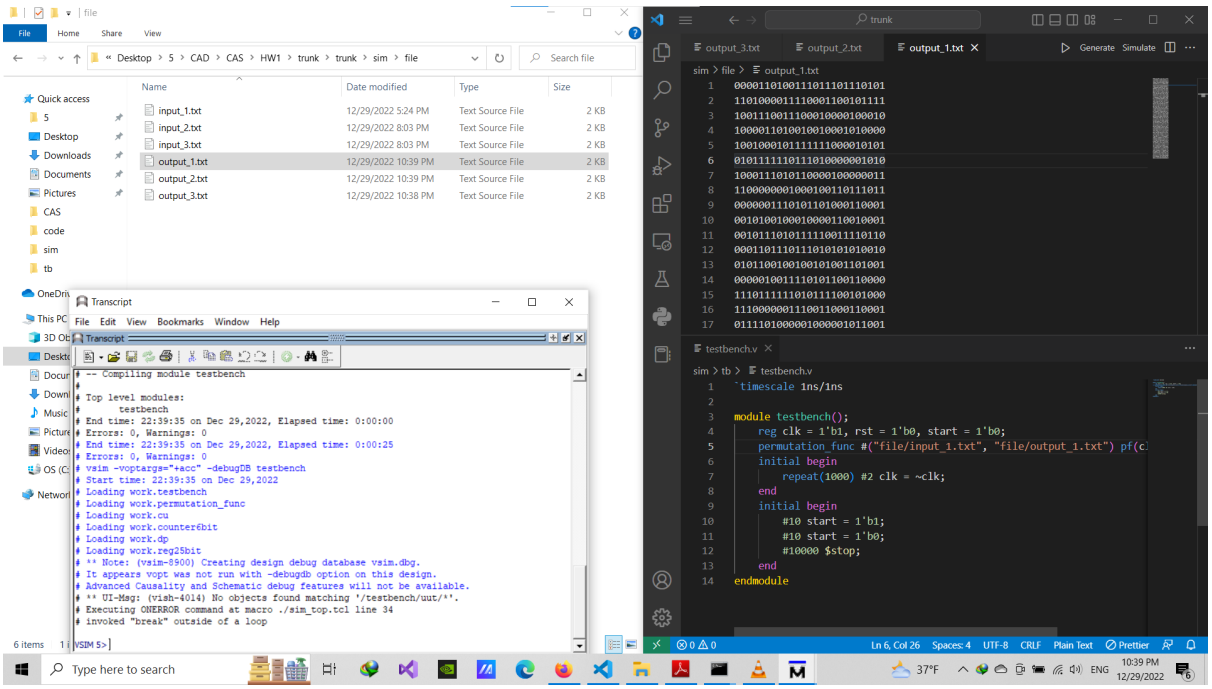
این فایل (permutatton_func) وایر کننده‌ی دیتاپث و کنترلر ما است.

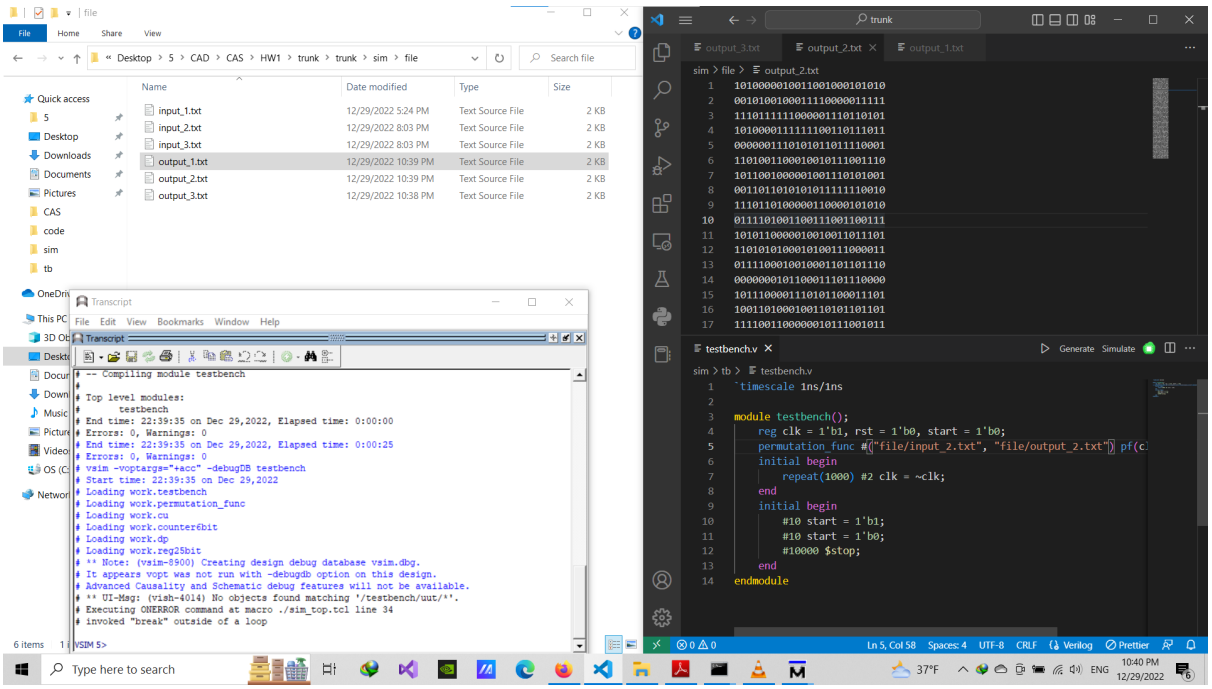
```
`timescale 1ns/1ns

module permutation_func #(parameter read_file_name, write_file_name)(
    clk,
    rst,
    start
);
    input clk, rst, start;
    reg ld = 1'b1;
    wire [24:0] out;
    wire [5:0] pout;
    wire reset_counter, count, load_reg, reset_reg,ready, read_input, write_output, cout;
    cu c1 (start, clk, cout, reset_counter, count, load_reg, reset_reg, ready, read_input, write_output);
    counter6bit conter1(count, clk, reset_counter, pout, cout);
    dp #(read_file_name, write_file_name) dp_(clk, rst, ld, read_input, out);
endmodule
```

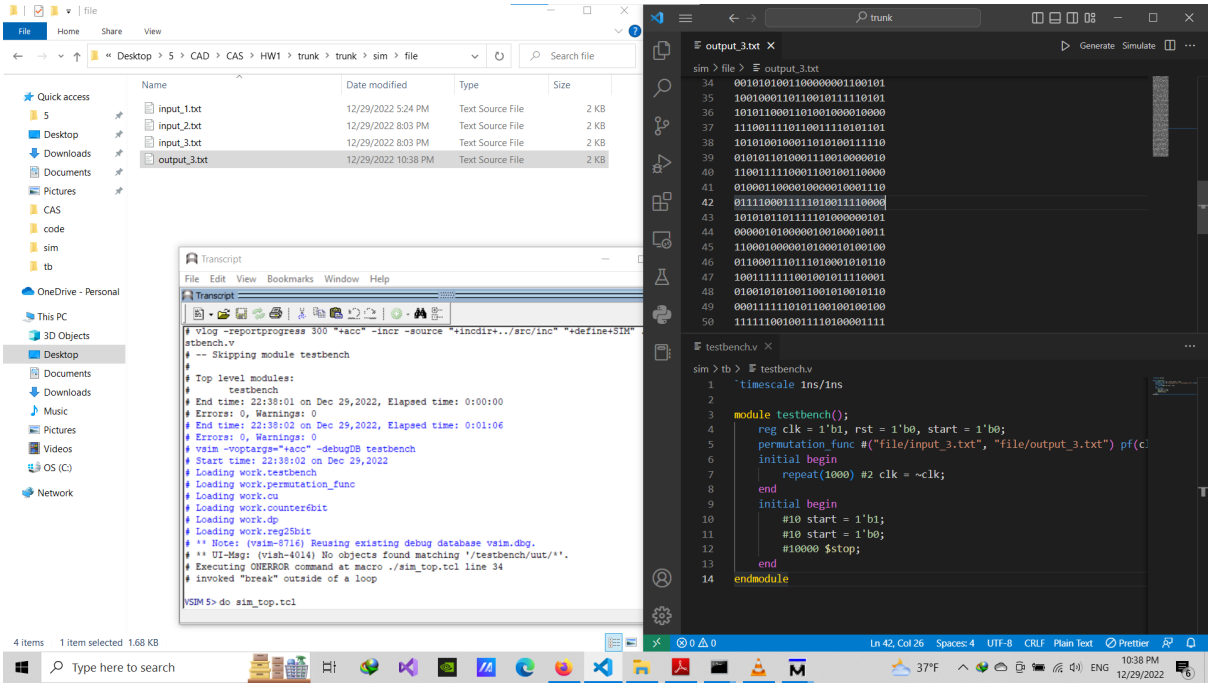
نکته : برای دیدن **out** و **ready** باید وارد سیگنال‌های این بخش بشویم و از **testbench** نمی‌توان آنها را دید.

می توانید نتیجه تست ها را در زیر مشاهده نمایید.

نتیجه برای هر سه تست درست و مشابه خروجی داده شده می باشد.




Transcript
-- Compiling module testbench
Top level modules:
testbench
End time: 22:39:35 on Dec 29,2022, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
End time: 22:39:35 on Dec 29,2022, Elapsed time: 0:00:25
Errors: 0, Warnings: 0
vsim -voptargs="+acc" -debugDB testbench
Start time: 22:39:35 on Dec 29,2022
Loading work.testbench
Loading work.permutation_func
Loading work.cu
Loading work.counter6bit
Loading work.dp
Loading work.reg25bit
** Note: (vsim-8900) Creating design debug database vsim.dbg.
It appears vopt was not run with -debugdb option on this design.
Advanced Causality and Schematic debug features will not be available.
** UI-Msg: (vish-4014) No objects found matching '/testbench/uut/*'.
Executing ONERROR command at macro ./sim_top.tcl line 34
invoked "break" outside of a loop
VSIM 5>
testbench.v
`timescale 1ns/1ns
module testbench();
reg clk = 1'b1, rst = 1'b0, start = 1'b0;
permutation_func #("file/input_2.txt", "file/output_2.txt") pf(c
initial begin
repeat(1000) #2 clk = ~clk;
end
initial begin
#10 start = 1'b1;
#10 start = 1'b0;
#10000 $stop;
end
endmodule


Transcript
vlog -reportprogress 300 "+acc" -incr -source "+incdir+../src/inc" "+define+SIM"
stbench.v
-- Skipping module testbench
Top level modules:
testbench
End time: 22:38:01 on Dec 29,2022, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
End time: 22:38:02 on Dec 29,2022, Elapsed time: 0:01:06
Errors: 0, Warnings: 0
vsim -voptargs="+acc" -debugDB testbench
Start time: 22:38:02 on Dec 29,2022
Loading work.testbench
Loading work.permutation_func
Loading work.cu
Loading work.counter6bit
Loading work.dp
Loading work.reg25bit
** Note: (vsim-8716) Reusing existing debug database vsim.dbg.
** UI-Msg: (vish-4014) No objects found matching '/testbench/uut/*'.
Executing ONERROR command at macro ./sim_top.tcl line 34
invoked "break" outside of a loop
VSIM 5> do sim_top.tcl
testbench.v
`timescale 1ns/1ns
module testbench();
reg clk = 1'b1, rst = 1'b0, start = 1'b0;
permutation_func #("file/input_3.txt", "file/output_3.txt") pf(c
initial begin
repeat(1000) #2 clk = ~clk;
end
initial begin
#10 start = 1'b1;
#10 start = 1'b0;
#10000 $stop;
end
endmodule