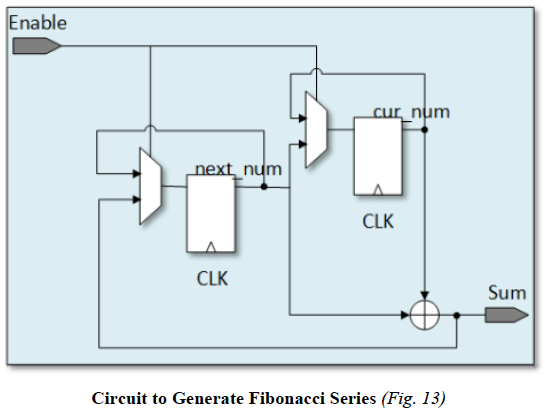
***ASSIGNMENT-04***

1.Write a Verilog code to print Fibonacci series of n-terms

**BLOCK DIAGRAM:**



**TEST BENCH:**

timescale ins / ipa

module sin fibonacci;

reg clk;

reg reset;

wire [31:0] fib\_out;

fibonacci DUT(.clk(clk), reset(reset), fib\_out(fib\_out));

always #2 clk=~ clk

initial begin

clk = 0;

reset = 0;

#1 reset = 1;

1500 $finish;

end

endmodule

**DESIGN UNDER TEST:**

module fibonacci(

input clk;

input reset;

output [31:0] fib\_out

);

reg (3110) num1, num2;

always @(posedge clk or negedge reset) begin

if (~reset) begin

num1<= 0;

num2<=1;

end

else begin

num1 <= num2;

num2 <= fib\_out;

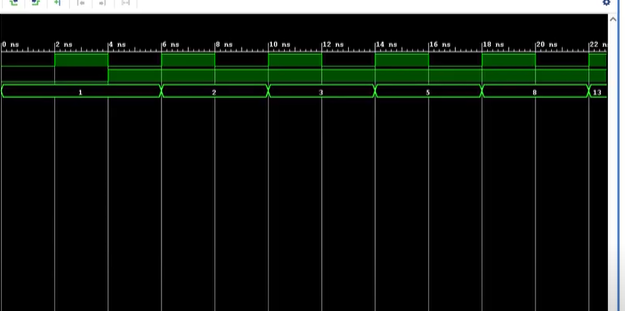
end

end

assign fib \_out = num1 + num2;

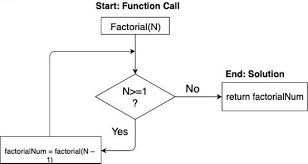
endmodule

**WAVE FORM:**



2. Write a Verilog code to calculate and print the factorial of a number using function.

**BLOCK DIAGRAM:**

****

**TEST BENCH:**

module factorial\_test\_bench();

reg [3:0] inp1;

wire [31:0] out1;

factorial a1(inp1,out1**);**

initial begin

inp1=4

#5 inp1=5;

#6 inp1=8;

end

endmodule  
**DESIGN UNDER TEST:**

module factorial(inp1,out1);

input [3:0] inp1;

output [31:0] out1;

reg [31:0] out\_temp;

reg [3:0] index;

always@\*

begin

out\_temp=inp1;

for(index=inp1-1;index=1;index-index-1)

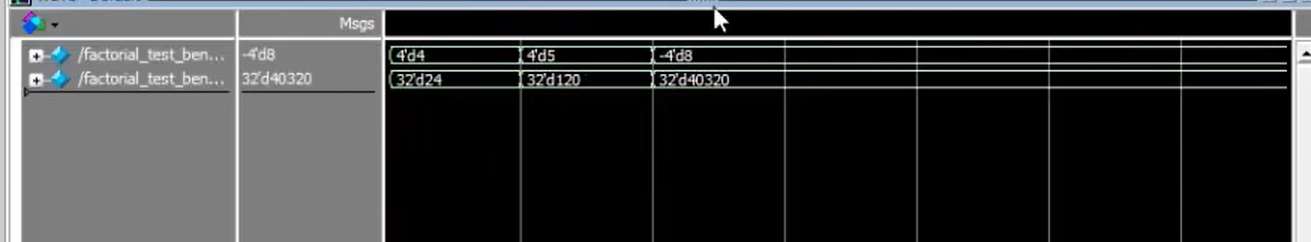
out\_temp=out\_temp\*index)

end

assign out1=out\_temp;

endmodule

**WAVE FORM:**



4.Write a Verilog code for sequence detectors

A.101 using mealy FSM with Overlapping

B.101 using Mealy FSM with non-overlapping

C.101 using Moore FSM with overlapping

D.101 using Moore FSM with non-overlapping

3.Write a Verilog code for patterns

**1**

**1 2**

**1 2 3**

**1 2 3 4**

B.**TEST BENCH:**

module testbench;

reg clk;

wire [3:0] pattern;

number\_pattern uut (clk, pattern);

initial begin

$dumfile("dump.vcd");

$dumpvars(1);

end

initial begin

clk = 0;

#5 clk = ~clk;

end

endmodule

**DESIGN UNDER TEST:**

module number\_pattern (input clk, output reg [3:0] pattern);

always @ (posedge clk)

begin

case (pattern)

4'b0001: pattern <= 4'b0010;

4'b0010: pattern <= 4'b0100;

4'b0100: pattern <= 4'b1000**;**

4'b1000: pattern <= 4'b0001;

default: pattern <= 4'b0001;

endcase

end

endmodule

D. 1

2 1

3 2 1

4 3 2 1

module pascal\_triangle(output reg [7:0] pattern);

always@(\*) begin

pattern = 8'b0;

case (pattern[3:0])

4'b0000: pattern = 8'b00010001;

4'b0001: pattern = 8'b00100010;

4'b0010: pattern = 8'b01000100;

4'b0011: pattern = 8'b10001000;

4'b0100: pattern = 8'b00010010;

4'b0101: pattern = 8'b00100100;

4'b0110: pattern = 8'b01001000;

4'b0111: pattern = 8'b10010000;

4'b1000: pattern = 8'b00010100;

4'b1001: pattern = 8'b00101000;

4'b1010: pattern = 8'b01010000;

4'b1011: pattern = 8'b10100000;

4'b1100: pattern = 8'b00100011;

4'b1101: pattern = 8'b01000101;

4'b1110: pattern = 8'b10001001;

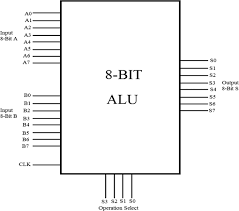
4'b1111: pattern = 8'b00010101;

default: pattern = 8'b00000000;

endcase

end

5.Write a Verilog code to implement a 8-bit ALU to perform at least 8 different operations

BLOCK DIAGRAM:  


TEST BENCH:  
module ALU\_tb;

reg [7:0] a, b;

reg [3:0] op;

wire [7:0] result;

wire zero, negative, overflow;

ALU alu (a, b, op, result, zero, negative, overflow);

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

end

initial begin

// test case 1

a = 8'b00010001;

b = 8'b00110010;

op = 4'b0011;

#10;

// test case 2

a = 8'b01100100;

b = 8'b10010010;

op = 4'b0100;

#10;

// test case 3

a = 8'b00000001;

b = 8'b00000010;

op = 4'b0110;

#10;

// add more test cases as needed

End

endmodule

**DESIGN UNDER TEST:**

module ALU (input [7:0] a, b,

input [3:0] op,

output reg [7:0] result,

output reg zero,

output reg negative,

output reg overflow);

always @(\*) begin

case (op)

4'b0000: result = a & b;

4'b0001: result = a | b;

4'b0010: result = a ^ b;

4'b0011: result = a + b;

4'b0100: result = a - b;

4'b0101: result = b - a;

4'b0110: result = a \* b;

4'b0111: result = a / b;

default: result = 8'b0;

endcase

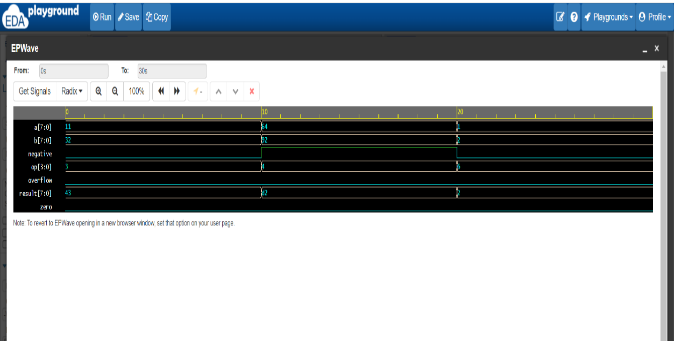
zero = (result == 8'b0);

negative = (result[7]);

overflow = ((a[7] == b[7]) && (a[7] != result[7]));

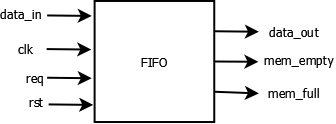
end

endmodule

**WAVE FORM:** 

6.Write a Verilog code for FIFO and Verify it’s working

**BLOCK DIAGRAM:**



TEST BENCH:

module fifo (

input clk,

input rst,

input push,

input pop,

input [7:0] data\_in,

output reg [7:0] data\_out,

output full,

output empty

);

reg [7:0] memory [15:0];

reg [4:0] head, tail;

always @(posedge clk) begin

if (rst) begin

head <= 0;

tail <= 0;

end else if (push) begin

if (!full) begin

memory[head] <= data\_in;

head <= head + 1;

end

end else if (pop) begin

if (!empty) begin

data\_out <= memory[tail];

tail <= tail + 1;

end

end

end

assign full = (head == tail + 16);

assign empty = (head == tail);

endmodule

**DESIGN UNDER TEST:**

module fifo\_tb;

reg clk, rst, push, pop;

reg [7:0] data\_in;

wire [7:0] data\_out;

wire full, empty;

fifo uut (

.clk(clk),

.rst(rst),

.push(push),

.pop(pop),

.data\_in(data\_in),

.data\_out(data\_out),

.full(full),

.empty(empty)

);

initial begin

clk = 0;

forever #5 clk = !clk;

end

initial begin

rst = 1;

#10 rst = 0;

pop = 0;

push = 0;

#10 push = 1;

data\_in = 8'h01;

#10 push = 0;

#10 pop = 1;

#10 pop = 0;

#10 push = 1;

data\_in = 8'h02;

#10 push = 0;

#10 pop = 1;

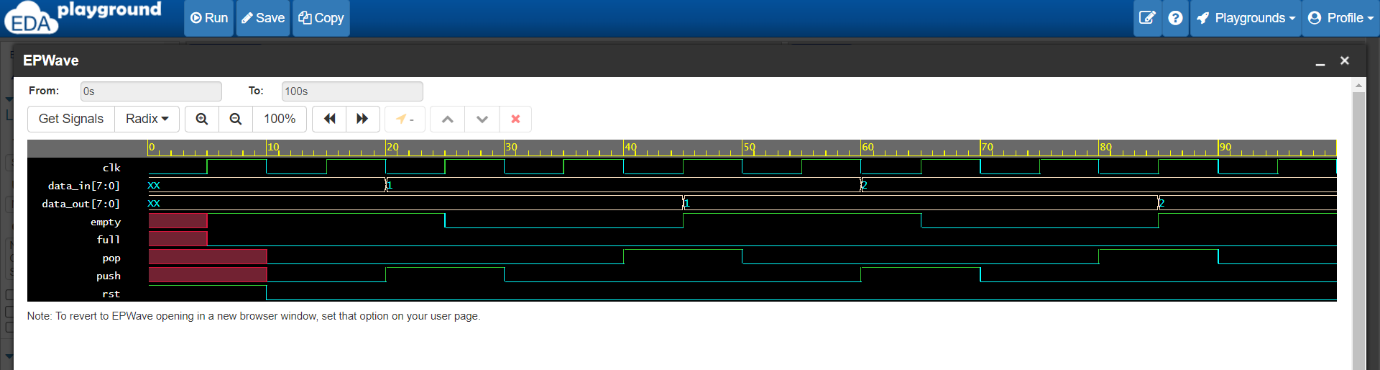
#10 pop = 0;

#10 $finish;

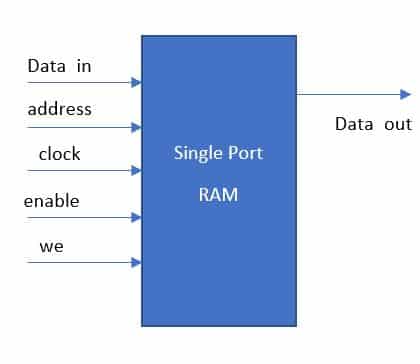
end

endmodule

**WAVE FORM:**



7.Write a Verilog code for RAM and verify its working

**BLOCK DIAGRAM:** 

**TEST BENCH:**

module single\_port\_ram\_testbench;

reg [7:0]data\_in;

reg [5:0] ram\_address;

reg write\_enable;

reg clk;

wire [7:0]data\_out;

single\_port\_ram ram1(data\_in , ram\_address,write\_enable,clk,data\_out);

initial begin // clock initialization

clk =1'b1;

forever #10 clk=~clk;

end

initial

begin

// writing data into the memory

write\_enable =1'b1;

#20;

ram\_address=5'd0;

data\_in = 8'h10;

#20;

ram\_address=5'd2;

data\_in = 8'h11;

#20;

ram\_address=5'd7;

data\_in = 8'haf;

#20;

//reading data from the memory

write\_enable = 1'b0;

ram\_address=5'd0;

#20;

ram\_address=5'd2;

#20;

ram\_address=5'd7;

#20;

$finish;

end

endmodule

**DESIGN UNDER TEST:**

timescale 1ns / 1ps

module single\_port\_ram(data\_in , ram\_address,write\_enable,clk,data\_out);

input [7:0]data\_in;

input [5:0] ram\_address;

input write\_enable;

input clk;

output [7:0]data\_out;

reg [7:0] ram\_memory[31:0]; // a 32 byte ( 32\*8 bit) RAM

reg [5:0] address\_register;

always @(posedge clk)

begin

if (write\_enable) // write operation

ram\_memory[ram\_address] <= data\_in;

else

address\_register <= ram\_address;

end

assign data\_out = ram\_memory[address\_register];

endmodule

**WAVE FORM:**

