END SEMESTER ASSESSMENT (ESA)
B.TECH. (CSE)
III SEMESTER

UE18CS206 – DIGITAL DESIGN & COMPUTER ORGANIZATION LABORATORY

PROJECT REPORT

ON

"16-bit Shift adder (Serial adder)"

SUBMITTED BY

NAME	SRN
1) Gurram Balaji	PES2UG20CS805
2) Chandan Kumar S	PES2UG20CS804
3) Upendra	PES2UG20CS920
4) Vijay. J	PES2UG20CS815

AUGUST – DECEMBER 2021

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ELECTRONIC CITY CAMPUS,

BENGALURU – 560100, KARNATAKA, INDIA

TABLE OF CONTENTS		
Sl.No	TOPIC	PAGE No

ABSTRACT OF THE PROJECT:

16-bit Shift adder (Serial adder)

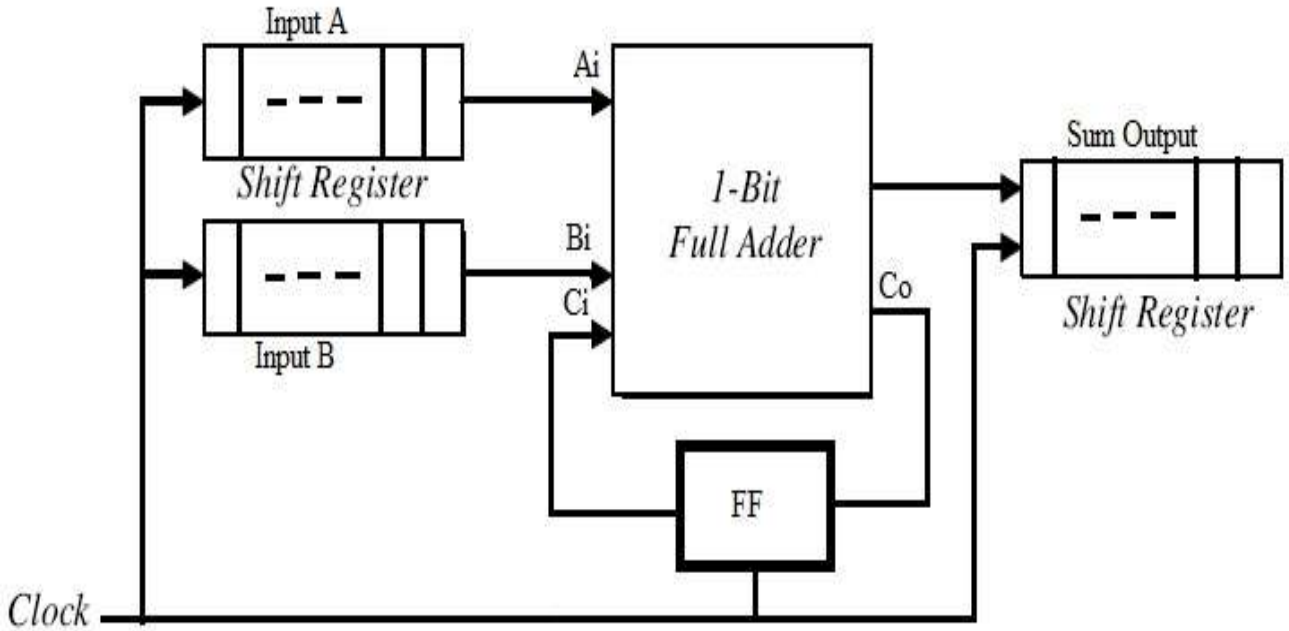
This is an implementation of a 16 bit serial shift adder using verilog.

A serial shift adder consists of three Shift Registers (two input registers and one output register), a D flip flop and a Full Adder. The least significant bits of the two input registers are fed to the Full Adder whose sum output is fed to the most significant bit location of the output shift register. The carry generated is stored in a D Flip Flop attached to the carry output of the Full Adder. The output of this D Flip Flop is fed as carry input to the Full Adder. All the registers and the D Flip Flop share the same clock input and at the positive edge of each clock cycle, all the registers are shifted right by one bit.

The input registers can be loaded parallelly while the output register can only take in serial data. Each D Flip Flop of the input registers has a corresponding 2:1 multiplexer used to choose between loading data parallelly to them or shifting right by 1 bit. The output sum register however does not need any of this and hence, the outputs of each D Flip Flop is directly fed as input to the next one

Since addition takes place in a serial fashion, this adder takes n-cycles to add two n-bit numbers. In this implementation, n is 16 and hence this adder takes 16 cycles to compute the sum of two 16 bit numbers

CIRCUIT DIAGRAM:



MAIN VERILOG CODE:

```
module shift_ff(input wire clk, reset, shift, prev_dff, d_in, output wire q);
  mux2 m(d_in, prev_dff, shift, in); // To select between shift and load
operations
  dfrl ff(clk, reset, 1'b1, in, q);
endmodule
module shift_register(input wire clk, reset, load, input wire [15:0] in,
                      output wire out_bit, output wire [15:0] contents);
wire shift;
wire intermediate[14:0];
  invert n1 (load, shift);
  shift_ff d1(clk, reset, shift, 1'b0, in[15], intermediate[14]);
  shift_ff d2(clk, reset, shift, intermediate[14], in[14], intermediate[13]);
  shift_ff d3(clk, reset, shift, intermediate[13], in[13], intermediate[12]);
  shift_ff d4(clk, reset, shift, intermediate[12], in[12], intermediate[11]);
  shift_ff d5(clk, reset, shift, intermediate[11], in[11], intermediate[10]);
  shift_ff d6(clk, reset, shift, intermediate[10], in[10], intermediate[9]);
  shift_ff d7(clk, reset, shift, intermediate[9], in[9], intermediate[8]);
  shift_ff d8(clk, reset, shift, intermediate[8], in[8], intermediate[7]);
  shift_ff d9(clk, reset, shift, intermediate[7], in[7], intermediate[6]);
  shift_ff d10(clk, reset, shift, intermediate[6], in[6], intermediate[5]);
  shift_ff d11(clk, reset, shift, intermediate[5], in[5], intermediate[4]);
  shift_ff d12(clk, reset, shift, intermediate[4], in[4], intermediate[3]);
  shift_ff d13(clk, reset, shift, intermediate[3], in[3], intermediate[2]);
  shift_ff d14(clk, reset, shift, intermediate[2], in[2], intermediate[1]);
  shift_ff d15(clk, reset, shift, intermediate[1], in[1], intermediate[0]);
  shift_ff d16(clk, reset, shift, intermediate[0], in[0], out_bit);
  assign contents = {intermediate[14], intermediate[13], intermediate[12],
intermediate[11], intermediate[10], intermediate[9],
                   intermediate[8], intermediate[7], intermediate[6],
intermediate[5], intermediate[4], intermediate[3],
```

```
    intermediate[2], intermediate[1], intermediate[0], out_bit};
endmodule
module shift_resgister_out(input wire clk, reset, in1, output wire [15:0] sum);
wire intermediate[14:0];
 dfrl d1(clk, reset, 1'b1, in1, intermediate[14]);
 dfrl d2(clk, reset, 1'b1, intermediate[14], intermediate[13]);
 dfrl d3(clk, reset, 1'b1, intermediate[13], intermediate[12]);
 dfrl d4(clk, reset, 1'b1, intermediate[12], intermediate[11]);
 dfrl d5(clk, reset, 1'b1, intermediate[11], intermediate[10]);
 dfrl d6(clk, reset, 1'b1, intermediate[10], intermediate[9]);
 dfrl d7(clk, reset, 1'b1, intermediate[9], intermediate[8]);
 dfrl d8(clk, reset, 1'b1, intermediate[8], intermediate[7]);
 dfrl d9(clk, reset, 1'b1, intermediate[7], intermediate[6]);
 dfrl d10(clk, reset, 1'b1, intermediate[6], intermediate[5]);
 dfrl d11(clk, reset, 1'b1, intermediate[5], intermediate[4]);
 dfrl d12(clk, reset, 1'b1, intermediate[4], intermediate[3]);
 dfrl d13(clk, reset, 1'b1, intermediate[3], intermediate[2]);
 dfrl d14(clk, reset, 1'b1, intermediate[2], intermediate[1]);
 dfrl d15(clk, reset, 1'b1, intermediate[1], intermediate[0]);
 assign sum = {in1, intermediate[14], intermediate[13], intermediate[12],
intermediate[11], intermediate[10],
          intermediate[9], intermediate[8], intermediate[7], intermediate[6],
intermediate[5],
          intermediate[4], intermediate[3], intermediate[2], intermediate[1],
intermediate[0]};
endmodule
module shift_adder (input wire clk, reset, load, input wire [15:0] a, b,
            output wire [15:0] contents_a, contents_b, op, output wire carry);
registers a and b
 wire t1, t2, fa_out, cin, cout;
 shift_register a0(clk, reset, load, a, t1, contents_a);
 shift_register b0(clk, reset, load, b, t2, contents_b);
 fulladder fa(t1, t2, cin, fa_out, cout);
 dfrl carry_hold(clk, reset, 1'b1, cout, cin); // This DFF will hold the carry to
be used by the full adder in the next clock cycle
 shift_resgister_out ans(clk, reset, fa_out, op);
 assign carry = cout;
endmodule
```

TEST BENCH FILE:

```
`timescale 1 ns / 100 ps
module tb;
   reg clk, reset, load;
   reg [15:0] a, b, out;

   wire [15:0] op, contents_a, contents_b;
   wire carry;

   shift_adder addr(clk, reset, load, a, b, contents_a, contents_b,
op, carry);

   initial begin $dumpfile("test.vcd"); $dumpvars(0,tb); end
   initial begin
      reset = 1'b1; // Resetting all the flip flops in the circuit
      load = 1'b0;
      #5
      reset = 1'b0;
      load = 1'b1;
      a = 16'b0101011101001001; // Loading a number into
register a
      b = 16'b1000001110101001; // Loading a number into
register b
      #5
      load = 1'b0; // Enabling shift for the registers
      #160 $finish;
   end
   initial clk = 1'b1; always #5 clk =~ clk;
endmodule
```

SCREEN SHOT OF THE OUTPUT: