https://www.pes.edu/wp-content/uploads/2019/09/pes_logo.png

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

**III SEMESTER**

**UE20CS206 – DIGITAL DESIGN & COMPUTER ORGANIZATION LABORATORY**

**PROJECT REPORT**

**ON**

“DESIGN AND IMPLEMENT 16 BIT SHIFT ADDER(SERIAL ADDER)”

SUBMITTED BY

**NAME SRN**

**AUGUST – DECEMBER 2021**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

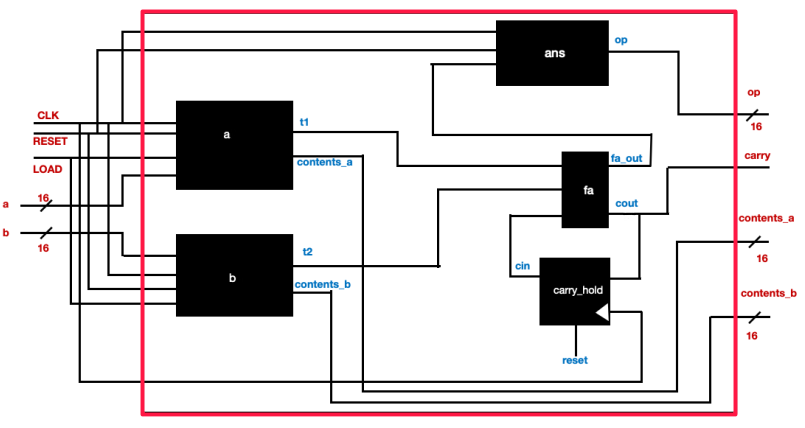
**ELECTRONIC CITY CAMPUS,**

**BENGALURU – 560100, KARNATAKA, INDIA**

|  |  |  |
| --- | --- | --- |
| TABLE OF CONTENTS | | |
| Sl.No | TOPIC | PAGE No |
|  | ABSTRACT OF THE PROJECT | 1 |
|  | CIRCUIT DIAGRAM | 2 |
|  | MAIN VERILOG CODE | 4 |
|  | TEST BENCH FILE | 5 |
|  | SCREEN SHOTS OF THE OUTPUT | 6 |

**ABSTRACT OF THE PROJECT:**

**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);

// This is a module for one shift register, i.e a collection of 16 D-Flip Flops

// Loads data parallely and on each clock cycle shifts the data by one bit

// load is used to identify whether data is being loaded into the register or a shift should occur

// out\_bit is the least significant bit of the the register that is used by the full adder to perform addition

wire shift; // This will be the inverse of the load input

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);

// This register is used to store the sum output

// in1 is the input bit received from the sum output of the fulladder

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);

// This is the serial shift adder module

// It takes two 16 bit numbers a and b as input and loads them to the shift registers a and b

// Their content is monitored using the output wires contents\_a and contents\_b respectively

// op is the output shift register where the output gets stored

// The full adder adds the LSBs of a and b at every clock cycle and pushes it onto the output shift register

// A DFF is used to hold the carry generated by the fulladder and this is fed back to the fulladder in the next clock cycle

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:**