END SEMESTER ASSESSMENT (ESA)

B.TECH. (CSE)

III SEMESTER

UE18CS206 – DIGITAL DESIGN & COMPUTER ORGANIZATION LABORATORY

PROJECT REPORT:

DESIGN AND IMPLEMENT A 16-BIT SHIFT ADDER (SERIAL ADDER)

SUBMITTED BY

SL. NO	NAME	SRN
1.	RAZIK FATIN SHARIFF	PES2UG19CS323
2.	REKHA C	PES2UG19CS324
3.	REYYALA CHETHAN	PES2UG19CS325
4.	RIA SINGH	PES2UG19CS326

AUGUST – DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ELECTRONIC CITY CAMPUS,

BENGALURU – 560100, KARNATAKA, INDIA

TABLE OF CONTENTS

ABSTRACT OF THE PROJECT:

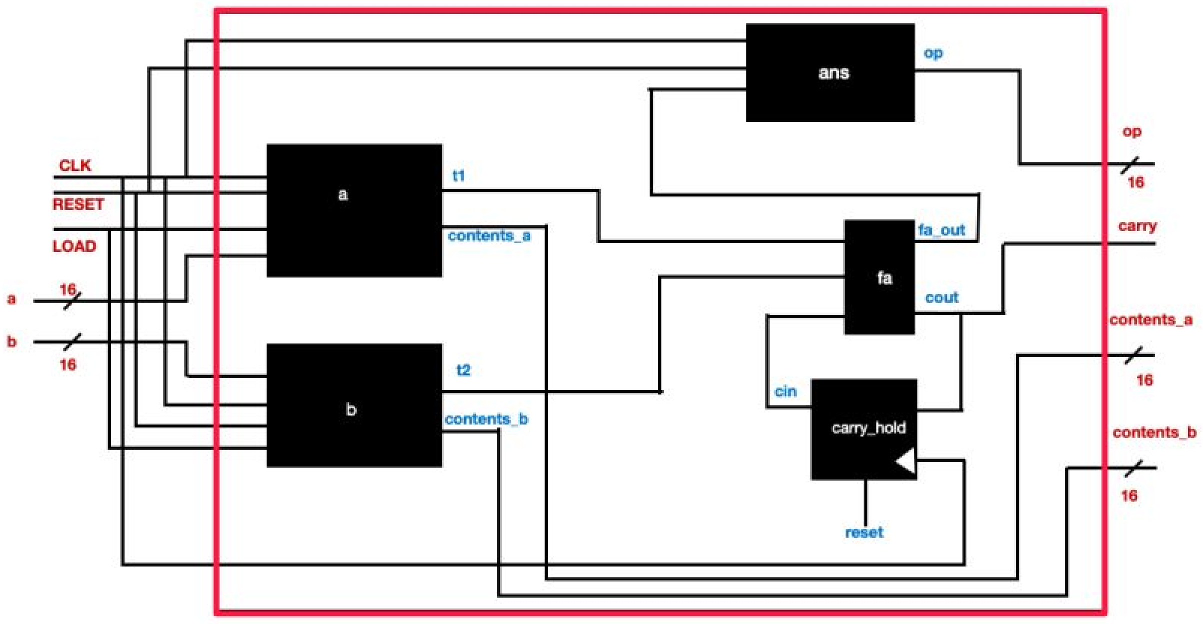
We have designed and implemented a 16-bit Shift Adder (Serial Adder), a combinational logic circuit that performs the addition of two binary numbers in serial form. Serial binary adder performs binary addition bit by bit simultaneously during each clock cycle.

The application of Full Adders, Registers, Multiplexers, and a few other basic modules have been done in the Verilog code to construct the Serial Adder. Two shift registers are used to store the binary numbers that are to be added. The circuit adds one pair at a time with the help of one full adder. The circuit adds one pair at a time with the help of one full adder. However the sum bit from the output of the full adder can be transferred into a third shift register.

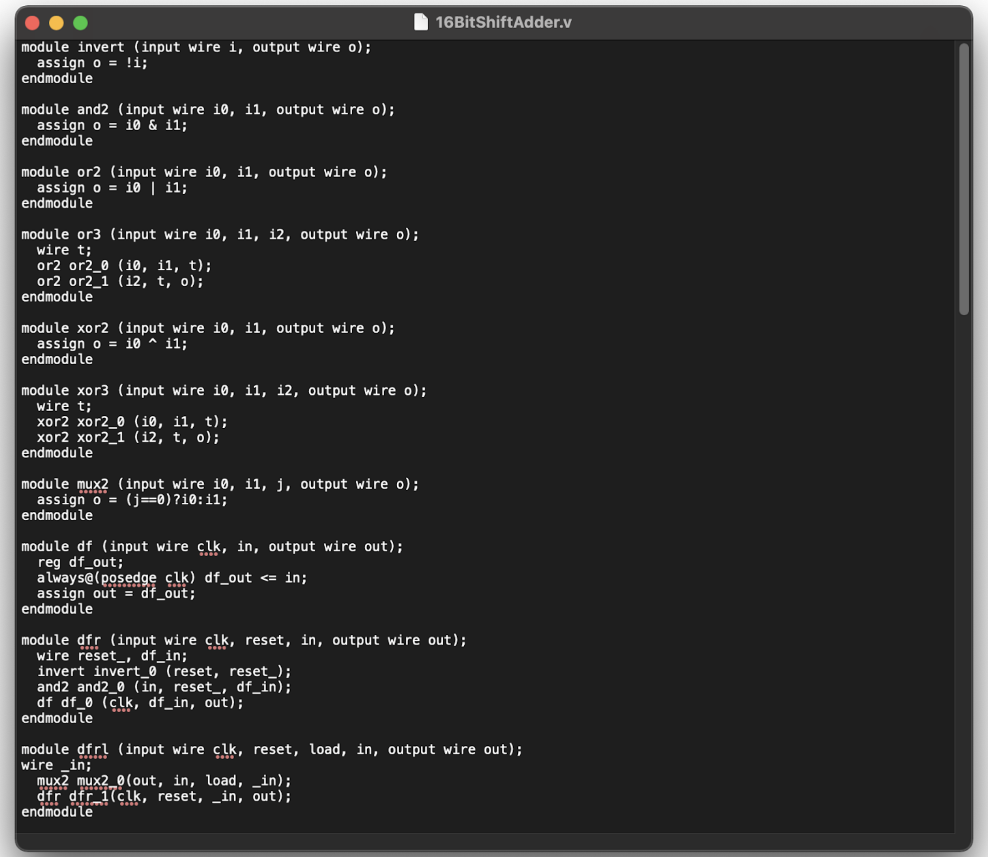
The circuit diagram included gives an overall idea of the electrical circuit, parts and how it has been connected in order to derive the necessary results. Gtkwave is the waveform analyzer and is the primary tool used for the visualization and hence get the output waveform of the Serial Adder. If speed is not a major factor, then serial adders are a much more cost effective way.

CIRCUIT DIAGRAM:

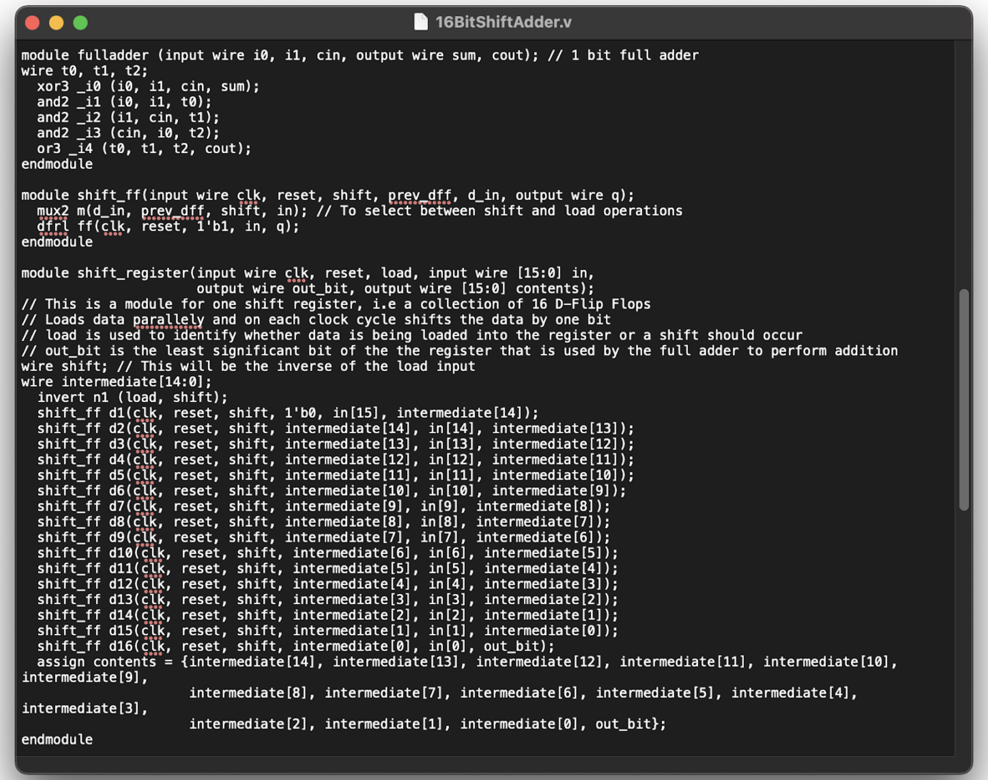
16 BIT-SHIFT ADDER (SERIAL)



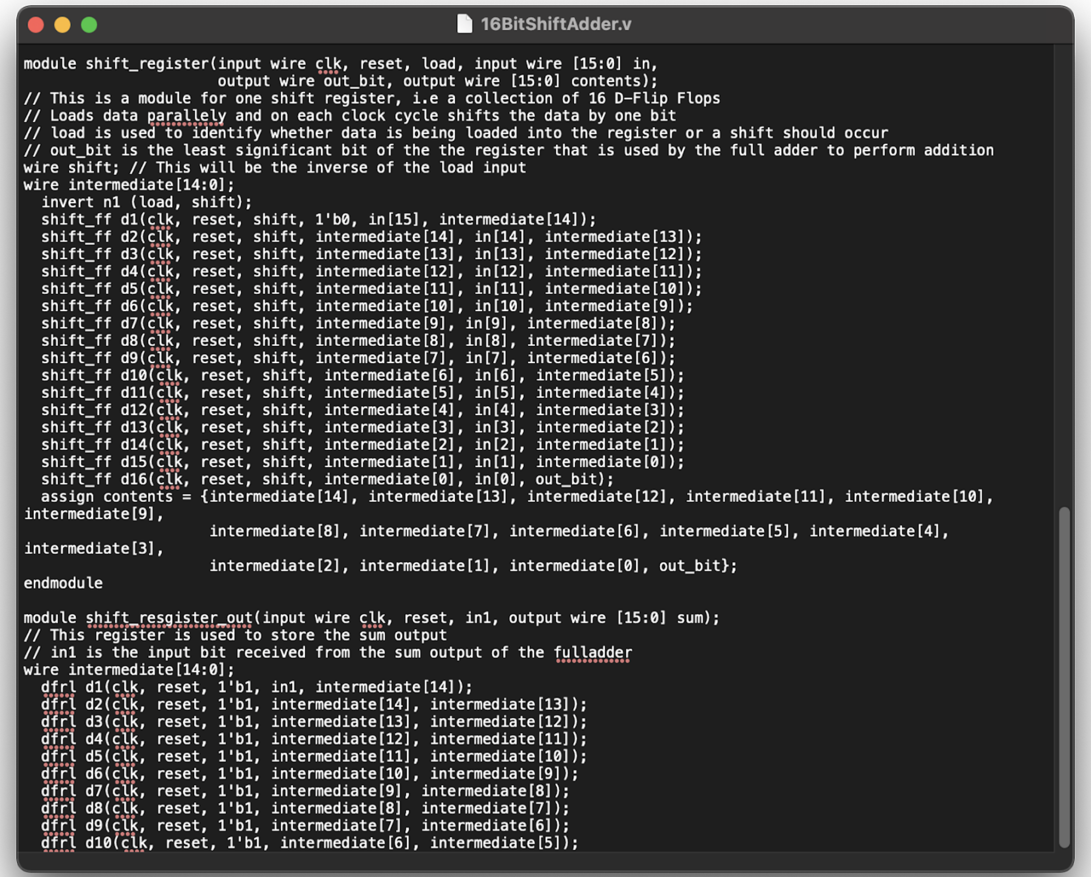
VERILOG CODE:



```verilog
module invert (input wire i, output wire o);
  assign o = !i;
endmodule

module and2 (input wire i0, i1, output wire o);
  assign o = i0 & i1;
endmodule

module or2 (input wire i0, i1, output wire o);
  assign o = i0 | i1;
endmodule

module or3 (input wire i0, i1, i2, output wire o);
  wire t;
  or2 or2_0 (i0, i1, t);
  or2 or2_1 (i2, t, o);
endmodule

module xor2 (input wire i0, i1, output wire o);
  assign o = i0 ^ i1;
endmodule

module xor3 (input wire i0, i1, i2, output wire o);
  wire t;
  xor2 xor2_0 (i0, i1, t);
  xor2 xor2_1 (i2, t, o);
endmodule

module mux2 (input wire i0, i1, j, output wire o);
  assign o = (j==0)?i0:i1;
endmodule

module df (input wire clk, in, output wire out);
  reg df_out;
  always@(posedge clk) df_out <= in;
  assign out = df_out;
endmodule

module dfr (input wire clk, reset, in, output wire out);
  wire reset_, df_in;
  invert invert_0 (reset, reset_);
  and2 and2_0 (in, reset_, df_in);
  df df_0 (clk, df_in, out);
endmodule

module dfrl (input wire clk, reset, load, in, output wire out);
wire _in;
  mux2 mux2_0(out, in, load, _in);
  dfr dfr_1(clk, reset, _in, out);
endmodule
```

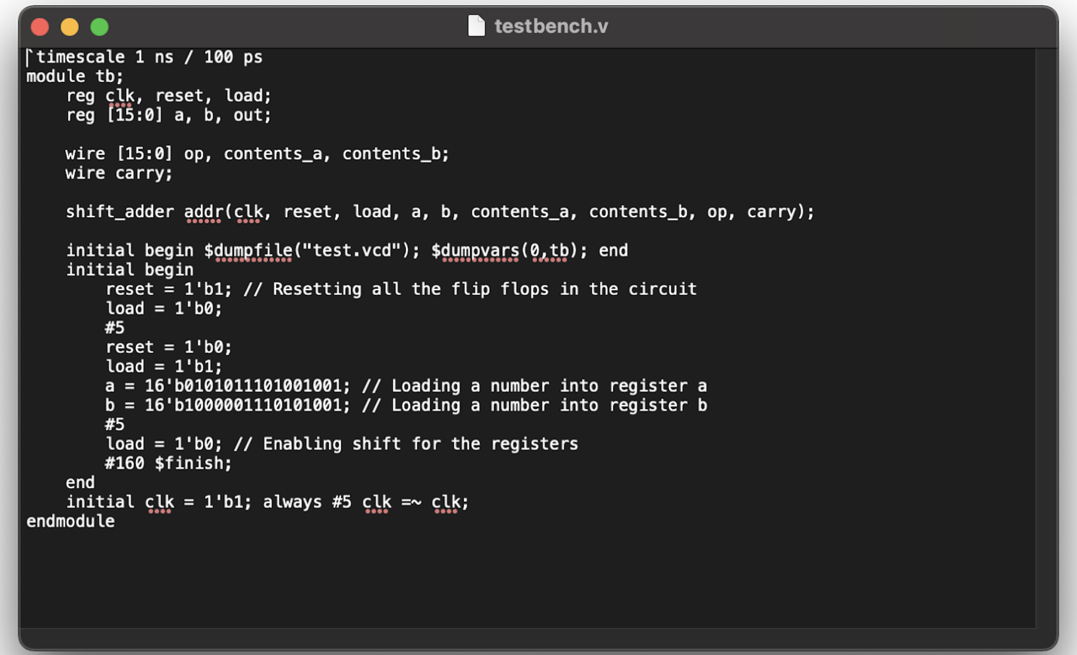


```verilog
module fulladder (input wire i0, i1, cin, output wire sum, cout); // 1 bit full adder
wire t0, t1, t2;
  xor3 _i0 (i0, i1, cin, sum);
  and2 _i1 (i0, i1, t0);
  and2 _i2 (i1, cin, t1);
  and2 _i3 (cin, i0, t2);
  or3 _i4 (t0, t1, t2, cout);
endmodule

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
```


16BitShiftAdder.v

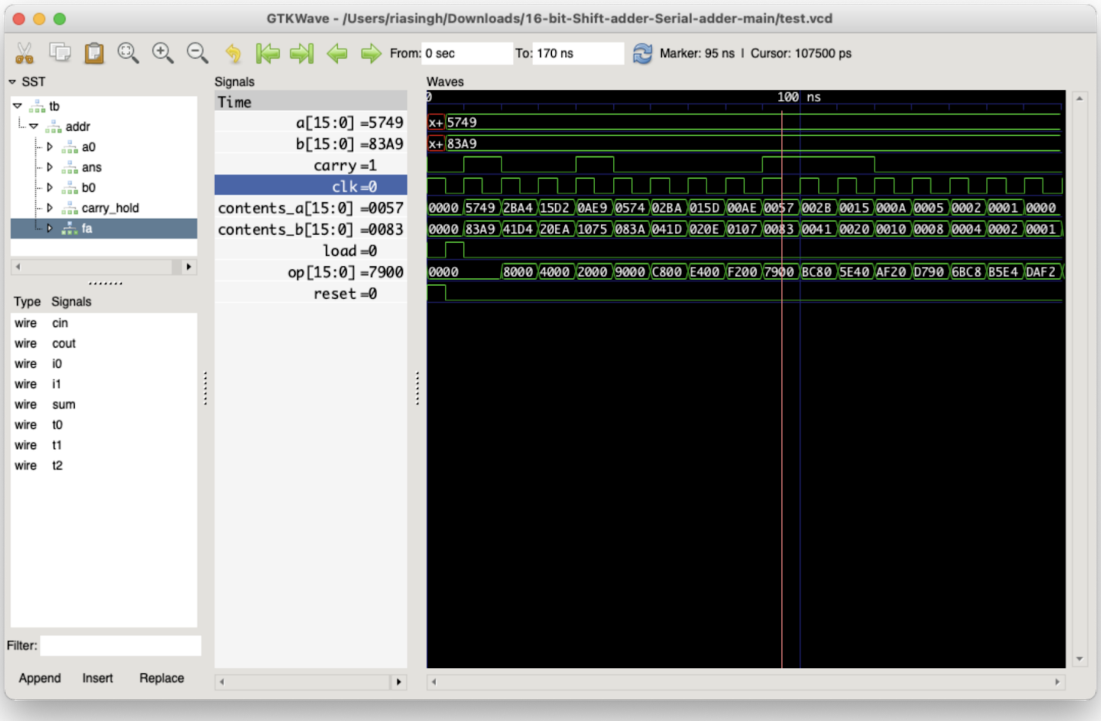
```
module shift_register(input wire clk, reset, load, input wire [15:0] in,
                      output wire out_bit, output wire [15:0] contents);
// This is a module for one shift register, i.e a collection of 16 D-Flip Flops
// Loads data parallely and on each clock cycle shifts the data by one 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]);
```

TEST BENCH FILE:


testbench.v

```
`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
```

SCREENSHOT OF THE OUTPUT:



```
GTKWave - /Users/riasingh/Downloads/16-bit-Shift-adder-Serial-adder-main/test.vcd

From: 0 sec    To: 170 ns    Marker: 95 ns | Cursor: 107500 ps

SST
tb
  addr
    a0
    ans
    b0
    carry_hold
    fa

Type  Signals
wire  cin
wire  cout
wire  i0
wire  i1
wire  sum
wire  t0
wire  t1
wire  t2

Signals                      Waves
Time                         0                                                   100 ns
a[15:0] =5749                5749
b[15:0] =83A9                83A9
carry =1
clk =0
contents_a[15:0] =0057       0000 5749 2BA4 15D2 0AE9 0574 02BA 015D 00AE 0057 002B 0015 000A 0005 0002 0001 0000
contents_b[15:0] =0083       0000 83A9 41D4 20EA 1075 083A 041D 020E 0107 0083 0041 0020 0010 0008 0004 0002 0001
load =0
op[15:0] =7900               0000      8000 4000 2000 9000 C800 E400 F200 7900 BC80 5E40 AF20 D790 6BC8 B5E4 DAF2
reset =0
```