CS 226/225
MINI PROJECT PART A
LAB REPORT

NAME: ADARSH KUMAR CHAUDHARY
ROLL: 1701CS01

FULL ADDER USING HALF ADDER

PROBLEM STATEMENT

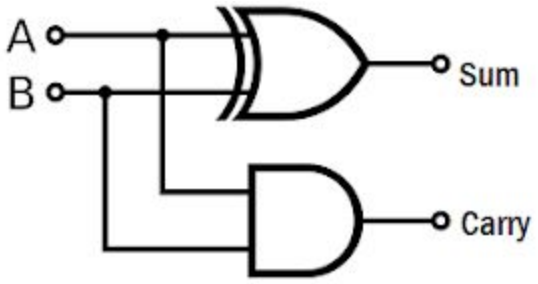
DESIGN FULL ADDER USING TWO HALF ADDERS AND SIMULATE IT USING VERILOG.

SOLUTION

TRUTH TABLES AND DIAGRAMS:

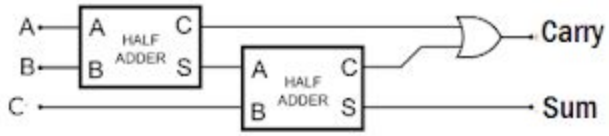
HALF ADDER:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



FULL ADDER:

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



HALF ADDER CODE:

```
//Declare the ports of Half adder module

module half_adder(

    Data_in_A,

    Data_in_B,

    Data_out_Sum,

    Data_out_Carry

    );


    //what are the input ports.

    input Data_in_A;

    input Data_in_B;

    //What are the output ports.

    output Data_out_Sum;

     output Data_out_Carry;


    //Implement the Sum and Carry equations using Verilog Bit operators.

    assign Data_out_Sum = Data_in_A ^ Data_in_B;  //XOR operation

    assign Data_out_Carry = Data_in_A & Data_in_B; //AND operation


Endmodule
```

FULL ADDER CODE:

```
module full_adder(
    Data_in_A,  //input A
    Data_in_B,  //input B
    Data_in_C,  //input C
     Data_out_Sum,
     Data_out_Carry
    );

    //what are the input ports.
    input Data_in_A;
    input Data_in_B;
     input Data_in_C;
    //What are the output ports.
    output Data_out_Sum;
     output Data_out_Carry;
     //Internal variables
     wire ha1_sum;
     wire ha2_sum;
     wire ha1_carry;
     wire ha2_carry;
     wire Data_out_Sum;
     wire Data_out_Carry;
```

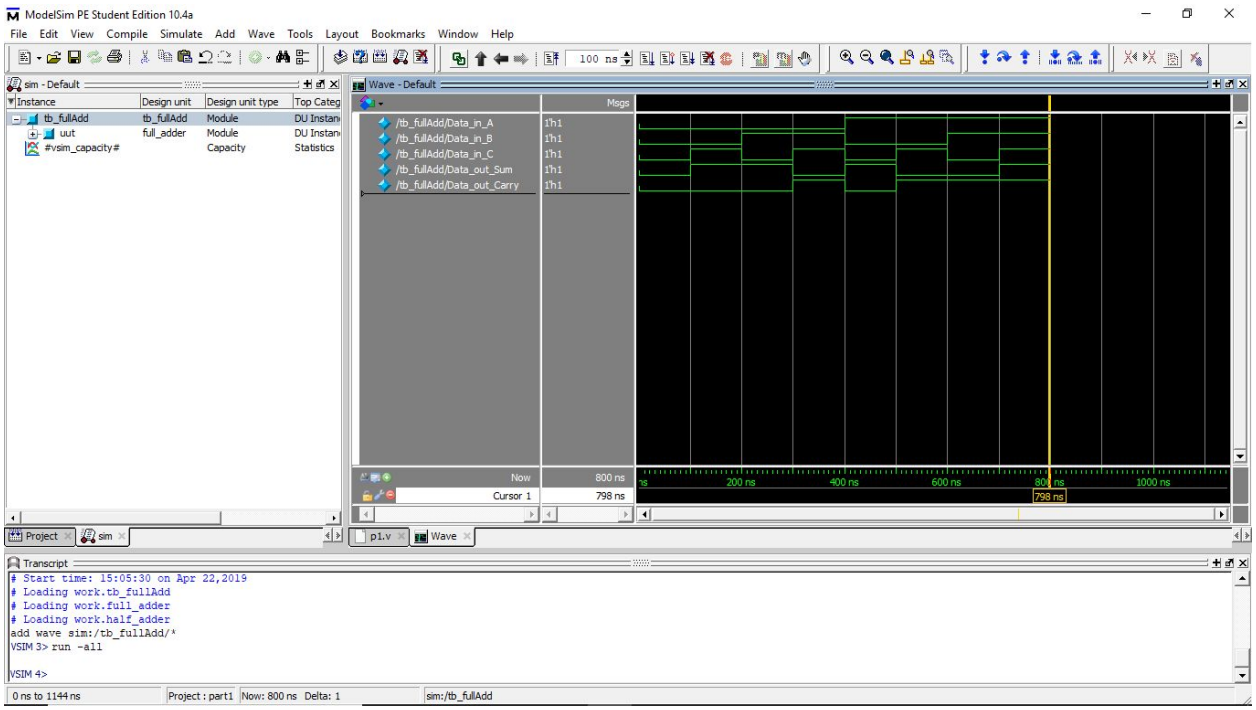
```
    //Instantiate the half adder 1
    half_adder  ha1(
        .Data_in_A(Data_in_A),
        .Data_in_B(Data_in_B),
        .Data_out_Sum(ha1_sum),
        .Data_out_Carry(ha1_carry)
    );

    //Instantiate the half adder 2
    half_adder  ha2(
        .Data_in_A(Data_in_C),
        .Data_in_B(ha1_sum),
        .Data_out_Sum(ha2_sum),
        .Data_out_Carry(ha2_carry)
    );

    //sum output from 2nd half adder is connected to full adder output
    assign Data_out_Sum = ha2_sum;
    //The carry's from both the half adders are OR'ed to get the final carry./
    assign Data_out_Carry = ha1_carry | ha2_carry;

endmodule
```

TEST BENCH CODE:

```
module tb_fullAdd;


    // Inputs
    reg Data_in_A;
    reg Data_in_B;
    reg Data_in_C;

    // Outputs
    wire Data_out_Sum;
    wire Data_out_Carry;

    // Instantiate the Unit Under Test (UUT)
    full_adder uut (
        .Data_in_A(Data_in_A),
        .Data_in_B(Data_in_B),
        .Data_in_C(Data_in_C),
        .Data_out_Sum(Data_out_Sum),
        .Data_out_Carry(Data_out_Carry)
    );

    initial begin
        //Apply inputs. 8 combinations of inputs are possible.
        //They are given below.
```

```
        Data_in_A = 0;  Data_in_B = 0;  Data_in_C = 0;  #100;
        Data_in_A = 0;  Data_in_B = 0;  Data_in_C = 1;  #100;
        Data_in_A = 0;  Data_in_B = 1;  Data_in_C = 0;  #100;
        Data_in_A = 0;  Data_in_B = 1;  Data_in_C = 1;  #100;
        Data_in_A = 1;  Data_in_B = 0;  Data_in_C = 0;  #100;
        Data_in_A = 1;  Data_in_B = 0;  Data_in_C = 1;  #100;
        Data_in_A = 1;  Data_in_B = 1;  Data_in_C = 0;  #100;
        Data_in_A = 1;  Data_in_B = 1;  Data_in_C = 1;  #100;
    end


Endmodule
```

IMAGE OF SIMULATION ON MY PC:



Critical Analysis:

This assignment familiarizes with describing computer architectural blocks in Verilog Hardware Description Language (HDL). This question can be directlty implemented by making full adder only using behavioural or structural approach but implementing using module of half adder teaches using different modules for a single implementation which is necessary for solving more complex problems. Also it reduces the chances of error.