Object Oriented Modeling of Electronic Circuits
Computer Assignment 1 – C++ Logic Modeling

Amir Hosein Samoudi – 810100108

• Introduction

In this assignment we are going to translate a combinational hardware design described by Verilog in a gate-level (by Verilog gate primitives "and", "or"," not") to a functional description of that using assign statements for each of the primary output. By doing this we are guarantying the same functionality of the gate-level design and the simulation time could improve massively.

• Methodology

To simplify the given netlist to number of assign statements first we should extract the primary inputs and outputs from the given file. Since the given file is line-oriented, we read it line by line and process it by extracting the first word and check it with Verilog key-words that is used in code.

Our goal is to start from a primary output and backtrack on its inputs to finally get to the inputs same as primary inputs and by doing this we omit all the in-between wires.

We created a data structure in our code to read each of the logic gates in input file and extract the type of the gate, its inputs and its output. After that we have the necessary information of all the gates.

We have solved this problem recursively and the approach is as below:

- For the first primary-output find the gate which derives this signal and extract its inputs
- Repeat this process for the extracted inputs (as the new output)
- Stop when we reach to the inputs same as the primary-inputs and complete the assign statement
- Repeat it for other primary-outputs

For testing the result, we have also generated a test-bench called test-data to simulate both designs (structural and behavioral) alongside each other and make sure outputs are exactly like each other and the functionality of the design is maintained.

In test-bench like the given sample, first we've declared separate wires for two circuits and then have got instances from both of them. For inputs, we have generated random values and assigned them to the identical ports of two designs and then compared the output waveforms.

- ## Test_1

First we have tested the result for the given sample file.

```
module Structural(
    input A,
    input B,
    input C,
    input D,
    output W
);

    wire w1;
    wire w2;
    wire w3;

    and i1(w1, A, B);
    and i2(w2, C, D);
    or i3(w3, w1, w2);
    not i4(W, w3);

endmodule
```

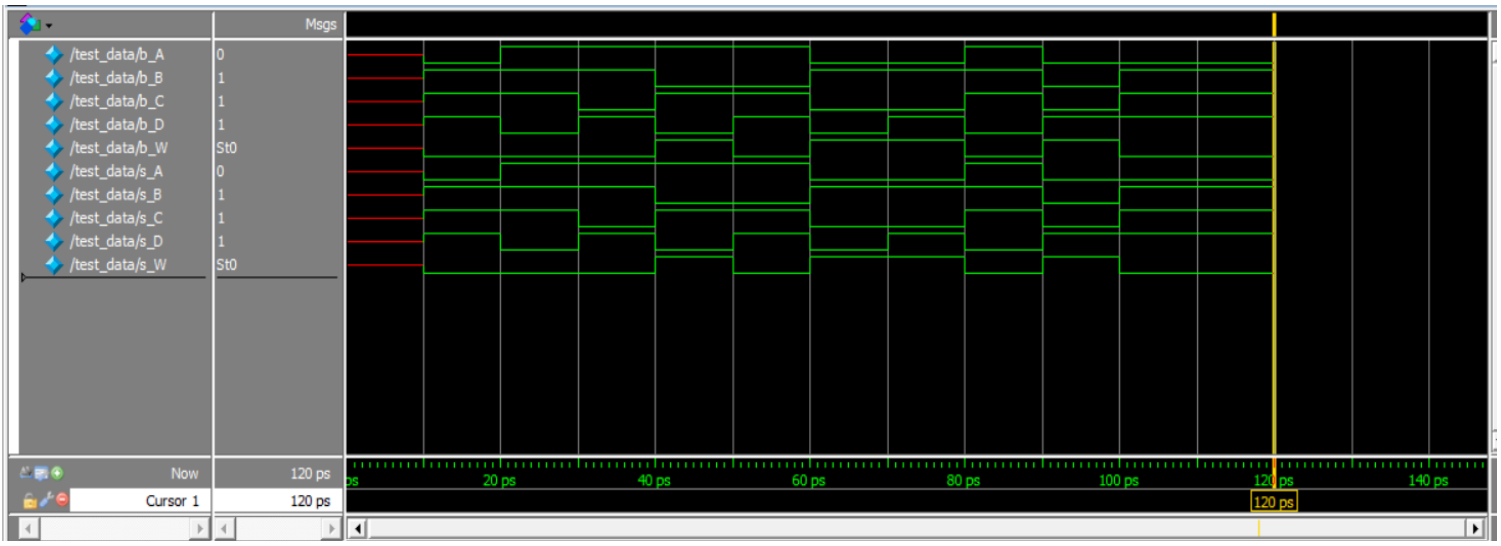
The generated functional circuit is as below:

```
module Behavioral(
    input A,
    input B,
    input C,
    input D,
    output W
);
    assign W = ~(((A & B) | (C & D)));

endmodule
```

The generated test-bench to examine the both circuits is as below:

```
module test_data();
reg b_A;
reg b_B;
reg b_C;
reg b_D;
wire b_W;
reg s_A;
reg s_B;
reg s_C;
reg s_D;
wire s_W;

Behavioral B_UUT(
    b_A,
    b_B,
    b_C,
    b_D,
    b_W
);

Structural S_UUT(
    s_A,
    s_B,
    s_C,
    s_D,
    s_W
);

initial begin
    repeat(10) begin
        #10
        s_A = $random();
        b_A = s_A;
        s_B = $random();
        b_B = s_B;
        s_C = $random();
        b_C = s_C;
        s_D = $random();
        b_D = s_D;
    end
    #20 $stop();
end
endmodule
```

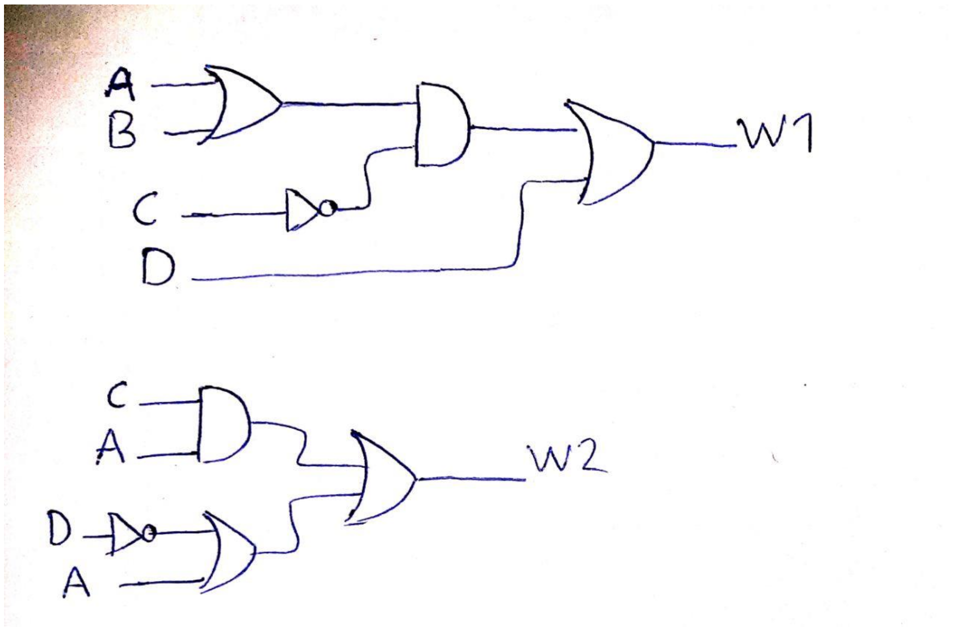
The waveform result of the above test-bench is as below:



Which validates the behavioral circuit.

- Test_2

For further testing the code we have wrote the netlist of the below circuit which includes two primary-outputs.



Here we have the netlist description:

```verilog
module Structural(
    input A,
    input B,
    input C,
    input D,
    output W1,
    output W2
);

    wire w1;
    wire w2;
    wire w3;
    wire w4;
    wire w5;
    wire w6;

    or i1(w1, A, B);
    not i2(w2, C);
    and i3(w3, w1, w2);
    or i4(W1, w3, D);

    and i5(w4, C, A);
    not i6(w6, D);
    or i7(w5, w6, A);
    or i8(W2, w4, w5);

endmodule
```

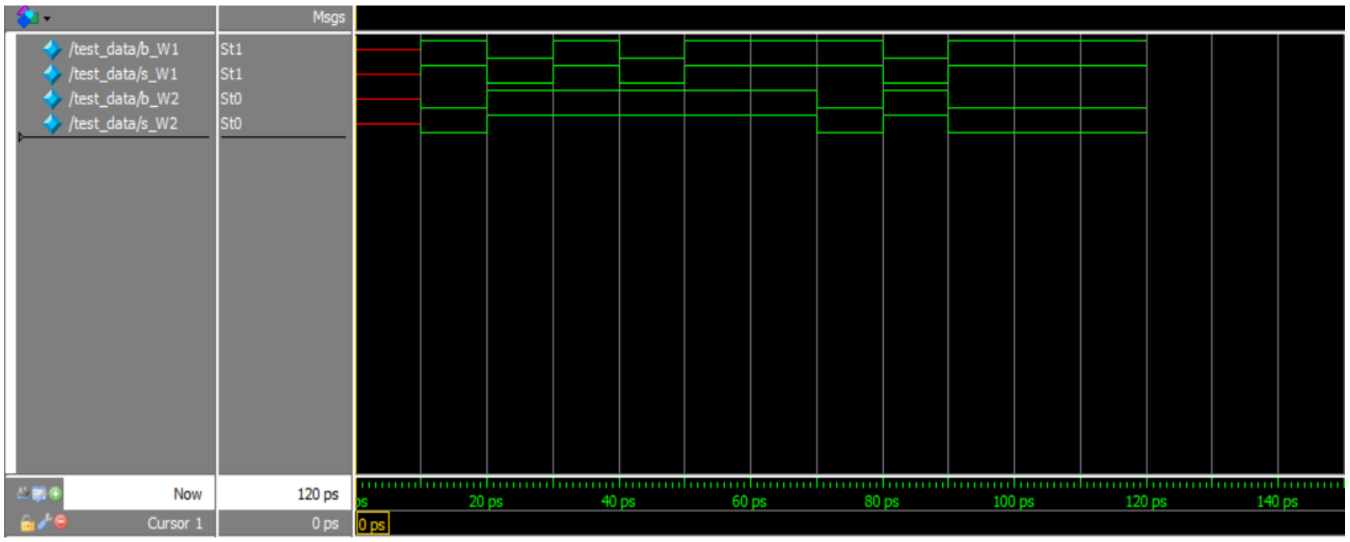
The functional code created is as below:

```verilog
module Behavioral(
    input A,
    input B,
    input C,
    input D,
    output W1,
    output W2
);
    assign W1 = (((A | B) & ~(C)) | D);
    assign W2 = ((C & A) | (~(D) | A));

endmodule
```

The generated test-bench as below:

```
module test_data();

reg b_A;
reg b_B;
reg b_C;
reg b_D;
wire b_W1;
wire b_W2;
reg s_A;
reg s_B;
reg s_C;
reg s_D;
wire s_W1;
wire s_W2;

Behavioral B_UUT(
    b_A,
    b_B,
    b_C,
    b_D,
    b_W1,
    b_W2
);

Structural S_UUT(
    s_A,
    s_B,
    s_C,
    s_D,
    s_W1,
    s_W2
);

initial begin
    repeat(10) begin
        #10
        s_A = $random();
        b_A = s_A;
        s_B = $random();
        b_B = s_B;
        s_C = $random();
        b_C = s_C;
        s_D = $random();
        b_D = s_D;
    end
    #20 $stop();
end

endmodule
```

then we have simulated it on ModelSim. The outputs waveforms are as below:



As it is clear, both outputs of our design are identical to the design described by the netlist.