PROJECT STEP 1

GROUP TOMATO

Ulises Espinoza-Gonzalez and Jack Stolpman

CS.3339 Computer Architecture

Texas State University

September 26, 2024

1 Introduction

Digital logic and circuits are an essential part of understanding the hardware components of any electronic device. Hardware descriptive languages such as Verilog, introduce us to the process of designing a new computer circuit from scratch. In this first section of our project, we will be utilizing 1-bit Not, Nand, Nor, and 1x4-bit input / 1x4-bit output shift circuits to generate simulation waveforms. Some goals that we as a group share is to improve our understanding of the methodology of designing new computer circuits from scratch. We also wish to familiarize ourselves with the Verilog programming language, which is new to us.

2 Verilog Code

In this section, we are going to go over the circuits and then the Verilog code for each one with the test modules.

3 Not Circuit

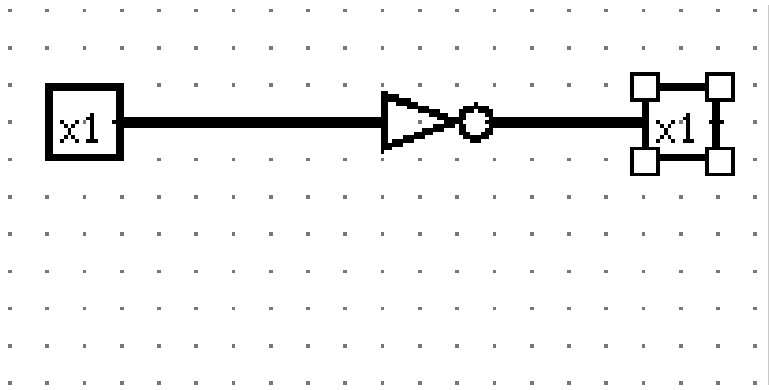
The Not circuit takes two inputs: A and B.The Not gate will revert the value inputted. If the value of A is 1, then the value of B will be 0 and vice versa.

```
1  `include "CompArchFirst.v"
2
3  module CompArchFirst_tb;
4
5      reg a, b;
6
7      wire not_out;
8
9      not_gate G1 (.a(a), .not_out(not_out));
10
11     initial begin
12
13         $dumpfile("CompArchFirst.vcd");
14         $dumpvars(0, CompArchFirst_tb);
15
16          // Display headers for better clarity in the console
17         $display("a b | not_out");
18         $display("--------------");
19
20         // Apply test cases
```

```
        a = 0; b = 0; #10;
        $display("%b %b |    %b", a, b, not_out);

        a = 0; b = 1; #10;
        $display("%b %b |    %b", a, b, not_out);

        a = 1; b = 0; #10;
        $display("%b %b |    %b", a, b, not_out);

        a = 1; b = 1; #10;
        $display("%b %b |    %b", a, b, not_out);

        $finish;  // End simulation

    end

endmodule
```

To test the Not circuit, we have created two registers, A and B. This way we can take two inputs at a time and test each possible input for the circuit. If it's working correctly, A should be equal to 1 whenever B is equal to 0.

```
// 1-bit Not Gate
module not_gate
(
    input a,         // Input
    output not_out   // Output - NOT result of 'a'
);

    // Assign the inverted value of 'a' to 'not_out'
    assign not_out = ~a;

endmodule
```



a	b
0	0
0	1
1	0
1	1

Figure 1: Not truth table and gate

3.1 Waveform Tests

The last section we will showcase the waveforms created using our testbenches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

4 Nand Circuit

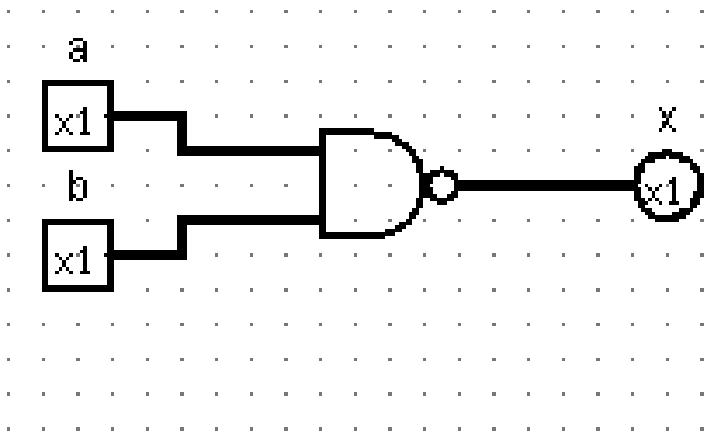
The Nand circuit takes two inputs, A and B, with an output X. This output will only shoot back a 1 assuming that both A and B are not equal to 1, hence the Nand (Not and) logic gate

```
1  `include "CompArchFirst.v"
2
3  module CompArchFirst_tb;
4
5      reg a, b;
6
7      wire nand_out;
8
9      nand_gate G2 (.a(a), .b(b), .nand_out(nand_out));
10
11     initial begin
12
13         $dumpfile("CompArchFirst.vcd");
14         $dumpvars(0, CompArchFirst_tb);
15
16          // Display headers for better clarity in the console
17         $display("a b | nand_out");
18         $display("--------------");
```

```
19
20          // Apply test cases
21          a = 0; b = 0; #10;
22          $display("%b %b |    %b", a, b, nand_out);
23
24          a = 0; b = 1; #10;
25          $display("%b %b |    %b", a, b, nand_out);
26
27          a = 1; b = 0; #10;
28          $display("%b %b |    %b", a, b, nand_out);
29
30          a = 1; b = 1; #10;
31          $display("%b %b |    %b", a, b, nand_out);
32
33          $finish;  // End simulation
34
35      end
36
37  endmodule
```

To test the Nand circuit, we have created two registers, A and B, as well as a wire X. This way we are able to take two inputs at a time and test each possible input for the circuit. If it's working correctly, X should be equal to 1 for all inputs except when both A and B are equal to 1

```
1  // 1-bit NAND Gate
2  module nand_gate
3  (
4      input a,        // First input
5      input b,        // Second input
6      output nand_out // Output - NAND result of 'a' and 'b'
7  );
8
9      // Assign the NAND result of 'a' and 'b' to 'nand_out'
10     assign nand_out = ~(a & b);
11
12 endmodule
```



a	b	x
0	0	1
0	1	1
1	0	1
1	1	0

Figure 2: Nand truth table and gate

4.1 Waveform Tests

The last section we will showcase the waveforms created using our testbenches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

5 Nor Circuit

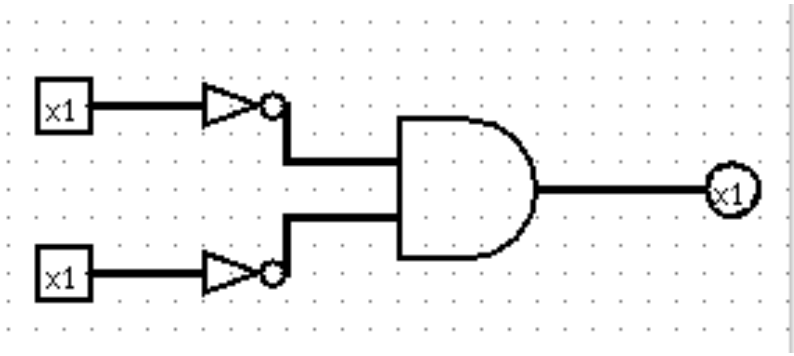
The Nor circuit takes two inputs, A and B, with an output X. This output will only shoot back a 1 assuming that both A and B are equal to 0, hence the Nor logic gate. For the output to return a high value, both inputs need to be low.

```
1  `include "CompArchFirst.v"
2
3  module CompArchFirst_tb;
4
5      reg a, b;
6
7      wire nor_out;
8
9      nor_gate G3 (.a(a), .b(b), .nor_out(nor_out));
10
11     initial begin
12
13         $dumpfile("CompArchFirst.vcd");
14         $dumpvars(0, CompArchFirst_tb);
15
16          // Display headers for better clarity in the console
17         $display("a b | nor_out");
```

```
        $display("--------------");

        // Apply test cases
        a = 0; b = 0; #10;
        $display("%b %b |    %b", a, b, nor_out);

        a = 0; b = 1; #10;
        $display("%b %b |    %b", a, b, nor_out);

        a = 1; b = 0; #10;
        $display("%b %b |    %b", a, b, nor_out);

        a = 1; b = 1; #10;
        $display("%b %b |    %b", a, b, nor_out);

        $finish;  // End simulation

    end

endmodule
```

To test the Nor circuit, we have created two registers, A and B, as well as a wire X. This way we are able to take two inputs at a time and test each possible input for the circuit. If it's working correctly, X should be equal to 1 for all inputs except when both A and B are equal to 0.

```
// 1-bit NOR Gate
module nor_gate
(
    input a,       // First input
    input b,       // Second input
    output nor_out // Output - NOR result of 'a' and 'b'
);

    // Assign the NOR result of 'a' and 'b' to 'nor_out'
    assign nor_out = ~(a | b);

endmodule
```



a	b	x
0	0	1
0	1	0
1	0	0
1	1	0

Figure 3: Nor truth table and gate

5.1 Waveform Tests

The last section we will showcase the waveforms created using our testbenches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

6 1x4-bit input Circuit

The 4 bit input circuit takes two inputs, fourin and fourout. We create a shift input

```
`include "CompArchFirst.v"

// Testbench module
module CompArchFirst_tb;

    // Registers for inputs and wires for outputs
    reg clock;
    reg [3:0] four_in;

    wire [3:0] four_out; // Output wires for input and shift circuit

    // Instantiate 4-bit input module
    four_bit_input G4
    (
        .clock(clock),
        .four_in(four_in),
        .four_out(four_out)
    );

    // 10 sec toggle for clock signal
    always begin
```

```
        #10 clock = ~clock;
    end

    initial begin

        $dumpfile("CompArch.vcd"); // Waveform Sim's
        $dumpvars(0, CompArchFirst_tb);

        // Initialize signals
        clock = 0;

        // Test inputs to logic gates and input module
        four_in = 4'b0000; #10; // Test case 1
        four_in = 4'b1010; #10; // Test case 2
        four_in = 4'b1100; #10; // Test case 3
        four_in = 4'b1111; #10; // Test case 4

        $finish;
    end

endmodule
```

To test this 4 bit circuit, we create a register that takes in a 4 bit value. This value works alongside the clock which when executed using the @posedge and the clock signal goes from high to low, the input will be transfered into the output.

```
// 1x4-bit Input Module
module four_bit_input
(
    input clock, // clock
    input [3:0] four_in, // 4-bit input signal ([3:0])
    output [3:0] four_out = 4'b0000 // 4-bit output signal ([3:0])
);

    always @(posedge clock) begin

            four_out <= four_in;

    end

endmodule
```

6.1 Waveform Tests

The last section we will showcase the waveforms created using our testbenches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

7 1x4-bit output Shift Circuit

The 4 bit input circuit takes two inputs, datain and dataout. We use a clock/ reset and create a shift input in order to determine the shift direction (high = 1 = right, low = 0 = left). On the rising edge of the reset, we set the output to 4'b0000 (0 in verilog) which clears the shift register. While this process is happening, if the shift is high, the data is shifted right by 1 bit and when the shift is low, the data is shifted left by 1 bit.

```
1  `include "CompArchFirst.v"
2
3  // Testbench module
4  module CompArchFirst_tb;
5
6      // Registers for inputs and wires for outputs
7      reg clock, reset, right_shift;
8      reg [3:0] four_in;
9
10     wire [3:0] four_shifted; // Output wires for input and shift circuit
11
12     // Instantiate 4-bit shift circuit
13     four_bit_shift_circuit G1
14     (
15         .clock(clock),
16         .reset(reset),
17         .right_shift(right_shift),
18         .data_in(four_in),
19         .data_out(four_shifted)
20     );
21
22     // 10 sec toggle for clock signal
23     always begin
24         #10 clock = ~clock;
25     end
26
27     initial begin
28
29         $dumpfile("CompArch.vcd"); // Waveform Sim's
```

```
        $dumpvars(0, CompArchFirst_tb);

        // Initialize signals
        clock = 0;
        reset = 1;
        right_shift = 0;

        // Test inputs to logic gates and input module
        four_in = 4'b0000; #10; // Test case 1
        four_in = 4'b1010; #10; // Test case 2
        four_in = 4'b1100; #10; // Test case 3
        four_in = 4'b1111; #10; // Test case 4

        // Testing left shift
        right_shift = 0; // Set to left shift
        four_in = 4'b0101; #10; // Input 0101
        four_in = 4'b0011; #10; // Input 0011

        // Testing right shift
        right_shift = 1; // Set to right shift
        four_in = 4'b1010; #10; // Input 1010
        four_in = 4'b1100; #10; // Input 1100

        // Test reset
        reset = 1; #10; // reset to 0000
        reset = 0; #10; // Release reset

        // Final input tests
        four_in = 4'b1111; #10; // Test input 1111
        right_shift = 0; // Set to left shift
        four_in = 4'b0001; #10; // Shift input 0001 left

        $finish;
    end

endmodule
```

To test the 4bit shift circuit, we have created two registers, A and B, as well as a wire X. We also create a

```
// 1x4-bit Shift Circuit
module four_bit_shift_circuit
(
    input clock,
```

```
    input reset,
    input right_shift,
    input [3:0] data_in,
    output reg [3:0] data_out
);

    always @(posedge clock or posedge reset) begin

        if (reset) begin

            data_out <= 4'b0000; // Reset output to 0

        end

        else begin

            if (right_shift) begin

                data_out <= data_in >> 1; // Shift right

            end

            else begin

                data_out <= data_in << 1; // Shift left

            end

        end

    end

endmodule
```

7.1 Waveform Tests

The last section we will showcase the waveforms created using our testbenches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

8 Conclusion

In conclusion, we successfully created and tested various circuits using Verilog and GTK-Wave. Learning how to code in Verilog, at least for our purposes, was relatively simple with few difficulties. Using GTKWave was also fairly intuitive. The most difficulty our group has had so far was most likely creating this report in LaTeX, as we have some trouble with coding certain formatting as well as issues with figures floating around the document. Creating these logic circuits from scratch allowed us to work on the basic logic gates (NOT, NAND, NOR) that are so critical to all electronics. The use of Verilog and GTKWave works as a stepping stone to furthering our engineering careers and building on our knowledge of both the software and hardware components of electronics.

9 Waveforms

9.1 Not Circuit Waveform

At 0 ns, we can see that both A and B are 0, so the output X is 1 which is consistent with the NOT gate.

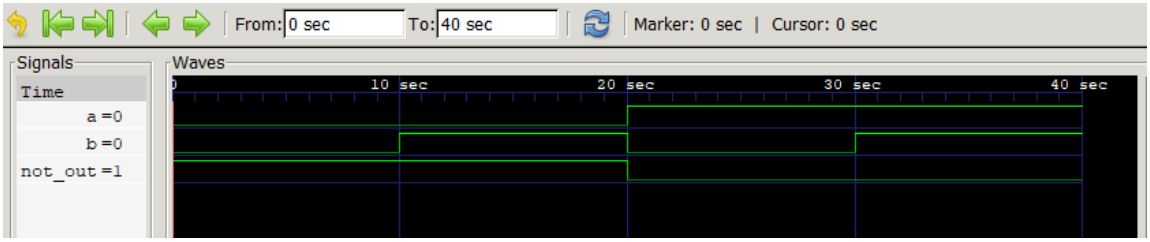


Figure 4: Not Circuit with marker at 0ns

At 20 ns, A is 1 and B is 0, so X is 0

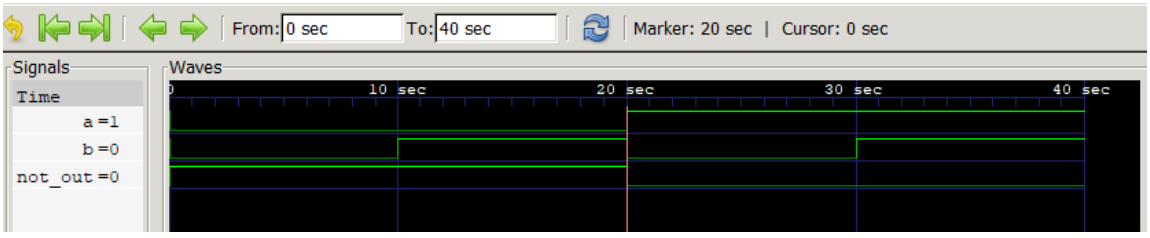


Figure 5: Not Circuit with marker at 20ns

At 40 ns, both A and B are 1, so X is now 0 because the not condition has been upheld

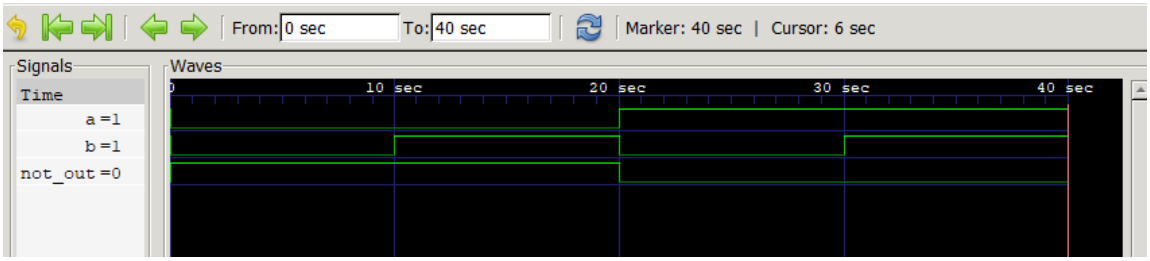


Figure 6: Not Circuit with marker at 40ns

9.2 Nand Circuit Waveform

At 0 ns, we can see that both A and B are 0, so the output X is 1 which is consistent with Nand gate behavior where both inputs have to be high(1) for the output to become low(0).

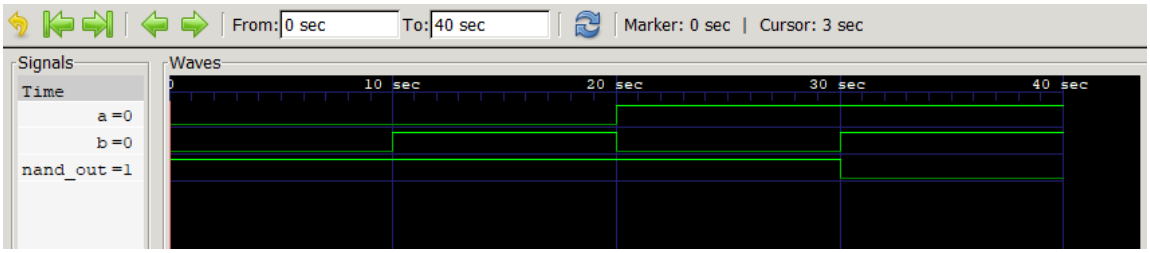


Figure 7: Nand Circuit with marker at 0ns

At 10 ns, A is 0 and B is 1, so X still remains at 1

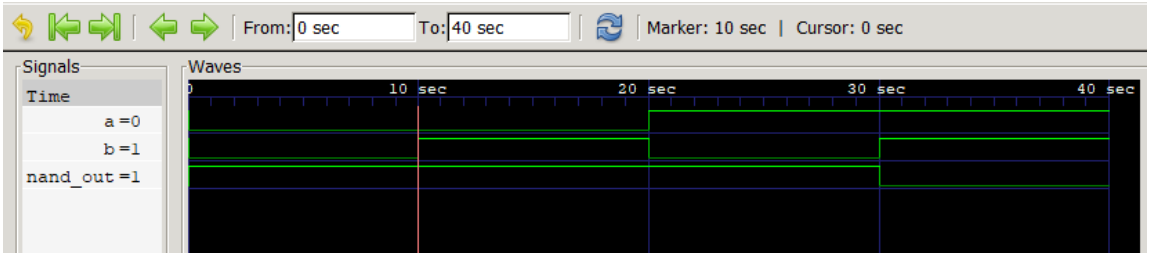


Figure 8: Nand Circuit with marker at 10ns

At 20 ns, A is 1 and B is 0, so X still remains at 1

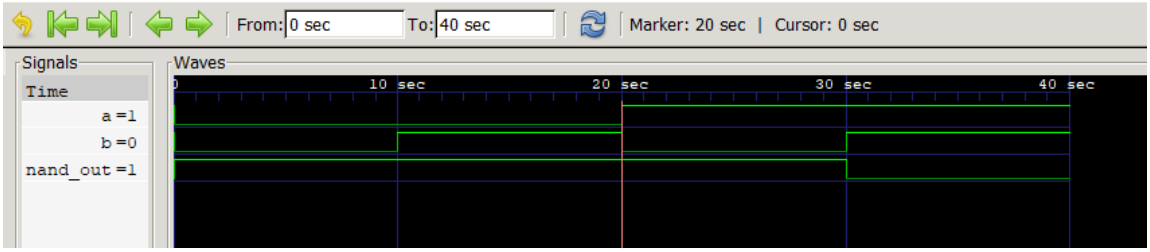


Figure 9: Nand Circuit with marker at 20ns

At 30 ns, A is 1 and B is 1, so X is 0

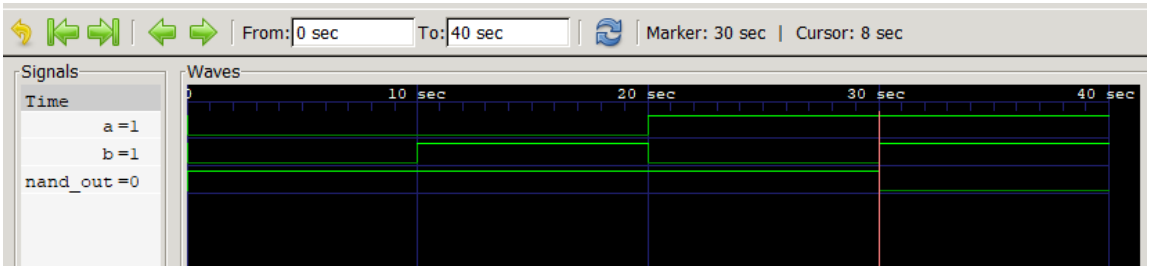


Figure 10: Nand Circuit with marker at 30ns

At 40 ns, both A and B are 1, so X is now 0 because the Nand gate requires both inputs to be high(1) in order for the output to become low (0).

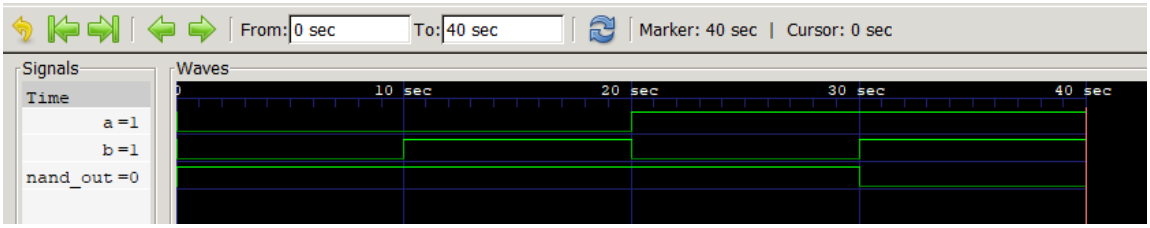


Figure 11: Nand Circuit with marker at 40ns

9.3 Nor Circuit Waveform

At 0 ns, we can see that both A and B are 0, so the output X is 1

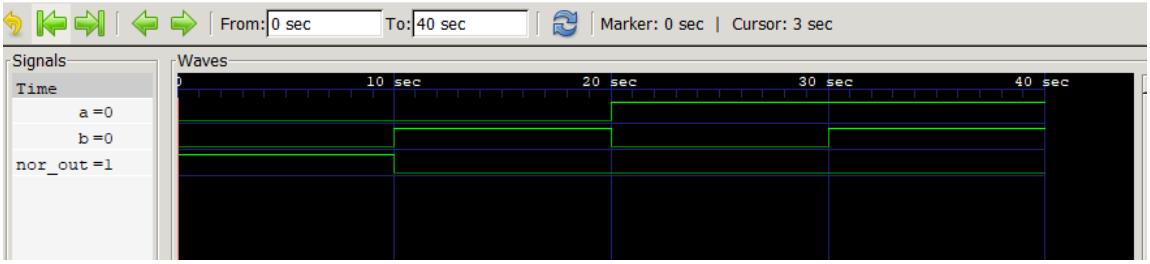


Figure 12: Nor Circuit with marker at 0ns

At 10 ns, we can see that A is 0 and B is 1, so the output X is 0

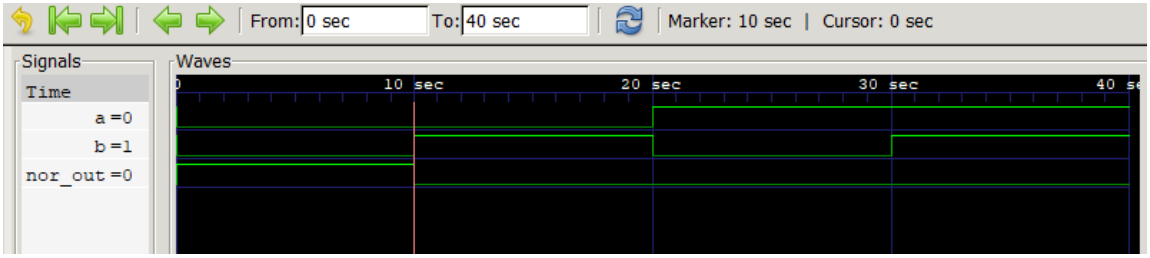


Figure 13: Nor Circuit with marker at 10ns

At 20 ns, A is 1 and B is 0, so X is 1

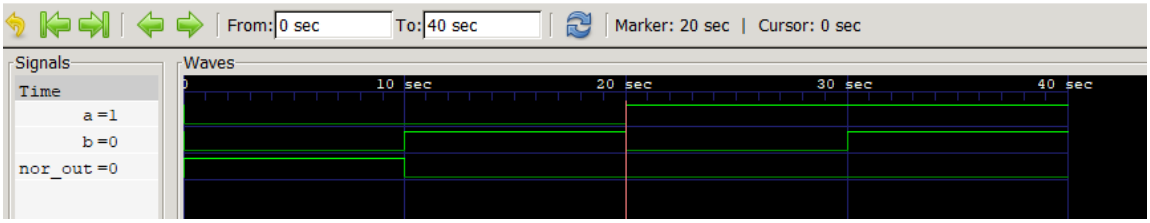


Figure 14: Nor Circuit with marker at 20ns

At 30 ns, A is 1 and B is 1, so X is 0

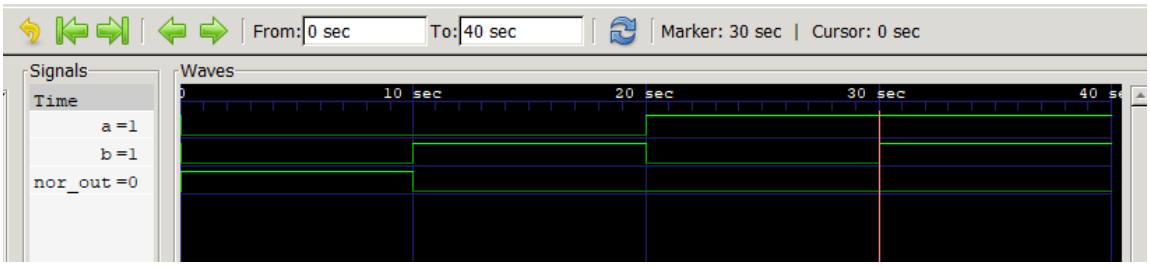


Figure 15: Nor Circuit with marker at 30ns

At 40 ns, both A and B are 1, so X is now 0

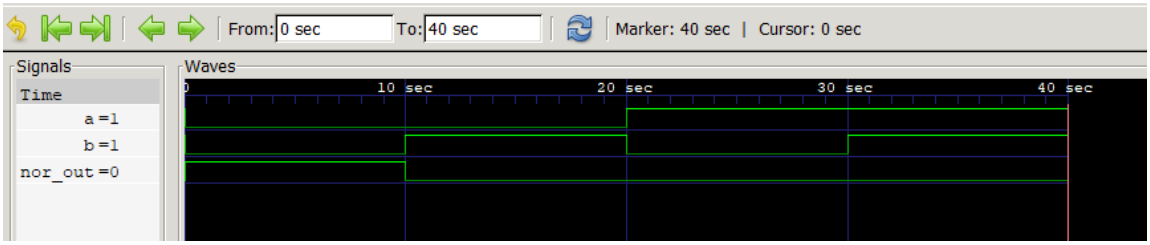


Figure 16: Nor Circuit with marker at 40ns

9.4 1x4-bit input Circuit Waveform

At 0 s, we can see that four in is 0, so the output four out is 0

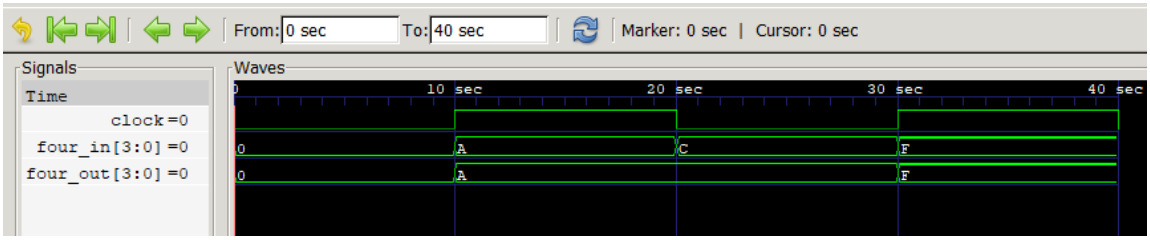


Figure 17: 1x4-bit input Circuit with marker at 0ns

At 10 s, we can see that four in is A, so the output four out is A

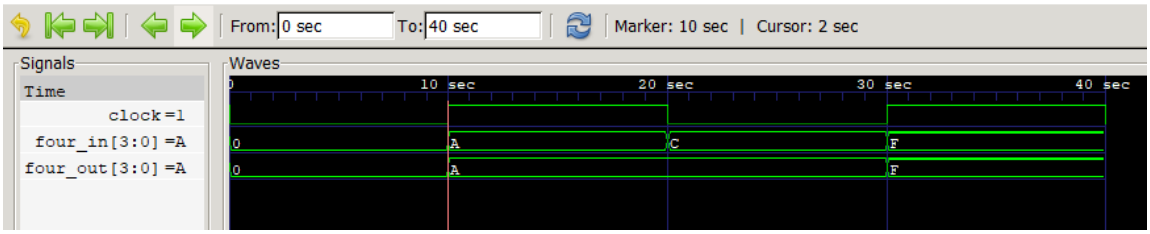


Figure 18: 1x4-bit input Circuit with marker at 10ns

At 20 s, four in is C, so four out is A

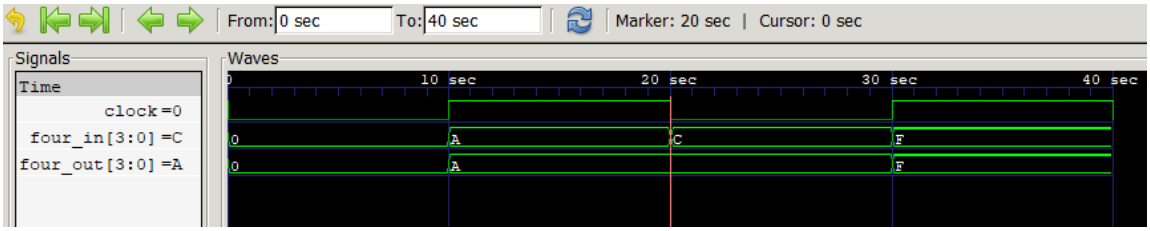


Figure 19: 1x4-bit input Circuit with marker at 20ns

At 30 s, four in is F, so four out is F

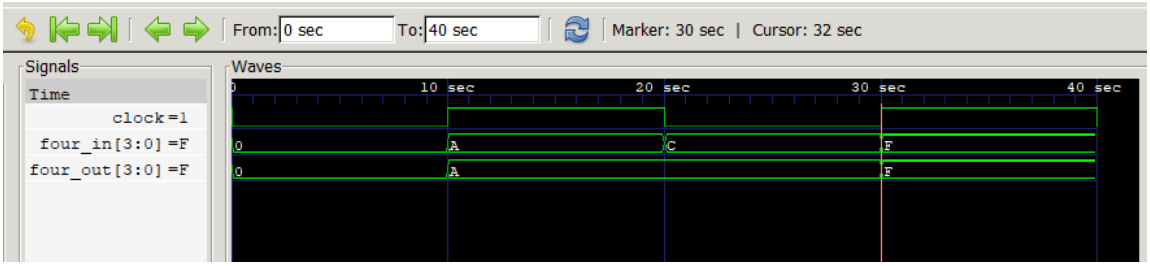


Figure 20: 1x4-bit input Circuit with marker at 30ns

At 40 s, four in is F and four out becomes F

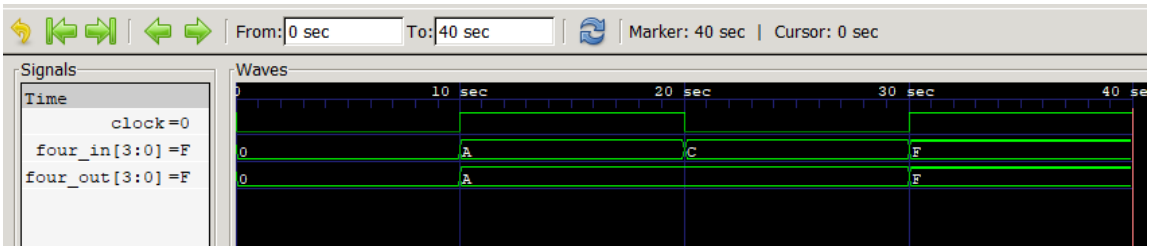


Figure 21: 1x4-bit input Circuit with marker at 40ns

9.5 1x4-bit output Shift Circuit Waveform

At 0 s, we can see that four in is 0, four shifted is 0 and right shift is 0

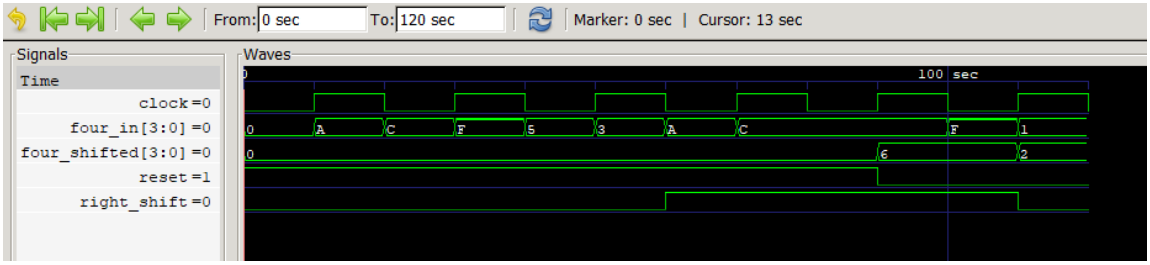


Figure 22: 1x4-bit output Shift Circuit with marker at 0s

At 10 s, we can see that four in is A, four shifted is 0 and right shift is 0

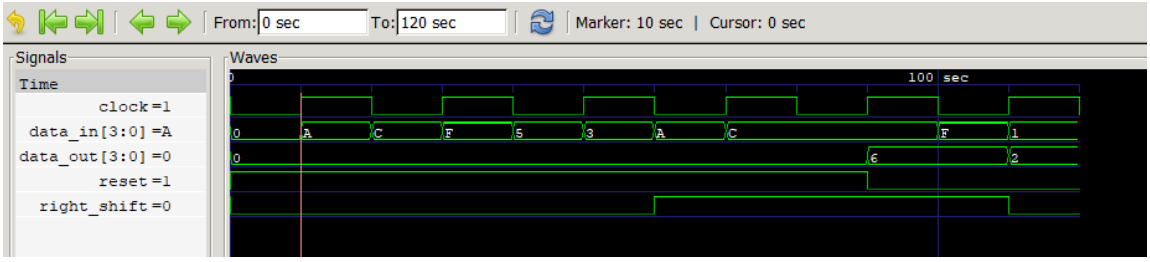


Figure 23: 1x4-bit output Shift Circuit with marker at 10s

At 20 s, we can see that four in is C, four shifted is 0 and right shift is 0

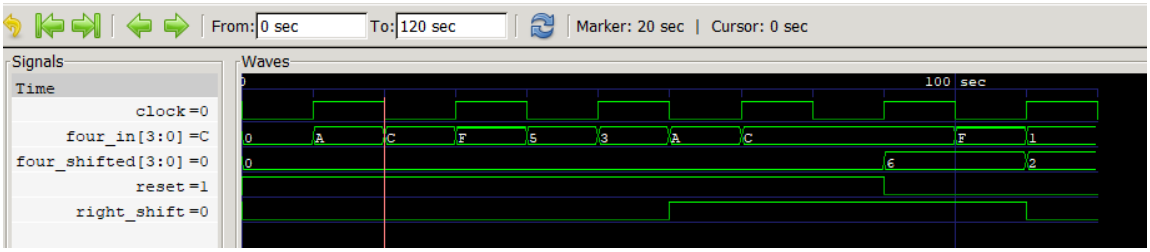


Figure 24: 1x4-bit output Shift Circuit with marker at 20s

At 30 s, we can see that four in is F, four shifted is 0 and right shift is 0

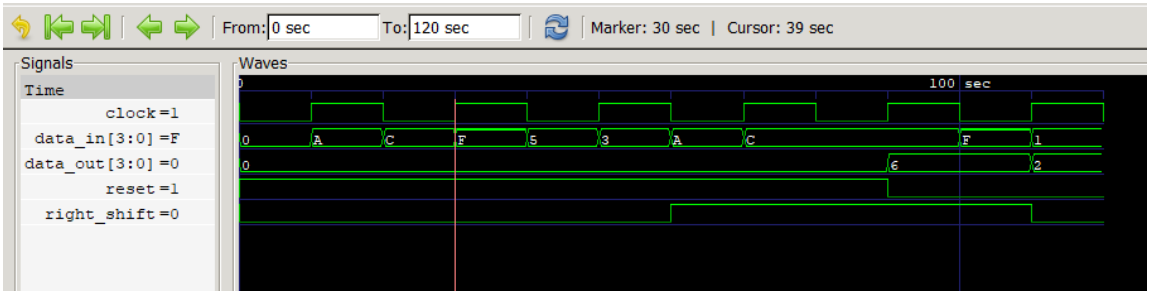


Figure 25: 1x4-bit output Shift Circuit with marker at 30s

At 40 s, four in is 5, four shifted is 0 and right shift is 0

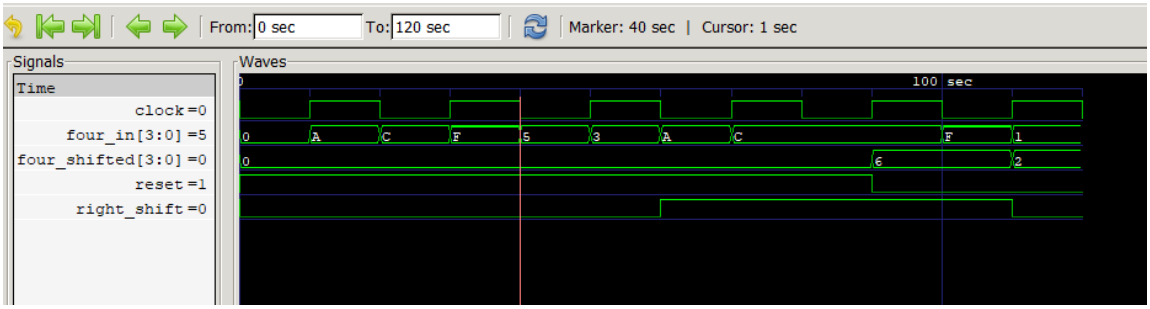


Figure 26: 1x4-bit output Shift Circuit with marker at 40s

At 50 s, we can see that four in is 3, four shifted is 0 and right shift is 0

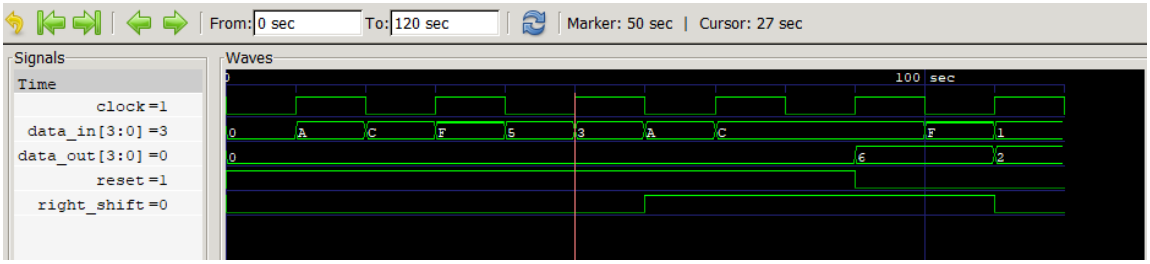


Figure 27: 1x4-bit output Shift Circuit with marker at 50s

At 60 s, we can see that four in is A, four shifted is 0 and right shift is 1

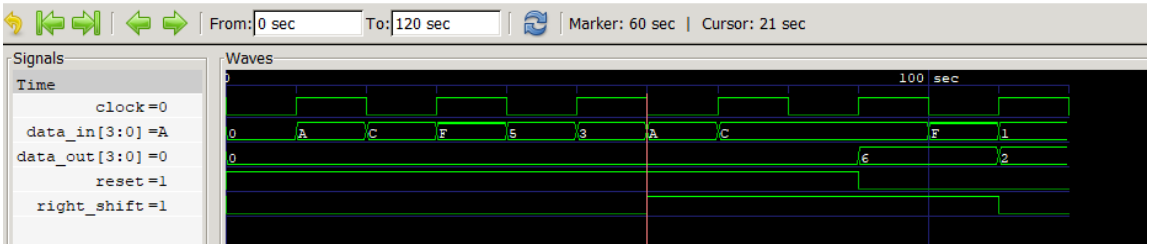


Figure 28: 1x4-bit output Shift Circuit with marker at 60s

At 70 s, we can see that four in is C, four shifted is 0 and right shift is 1

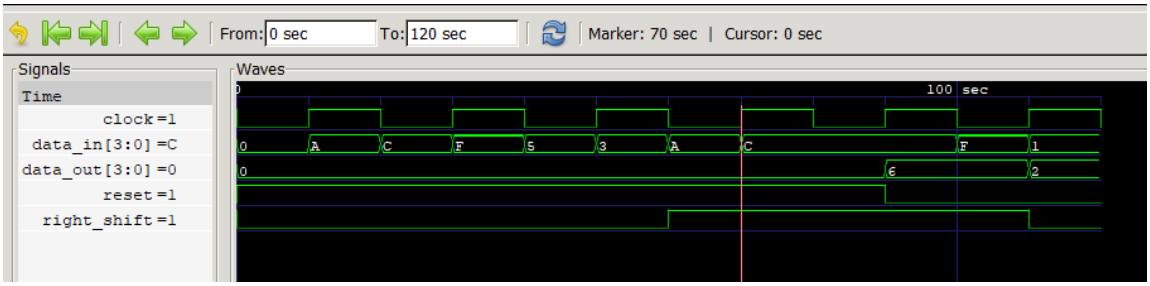


Figure 29: 1x4-bit output Shift Circuit with marker at 70s

At 80 s, we can see that four in is C, four shifted is 0 and right shift is 1

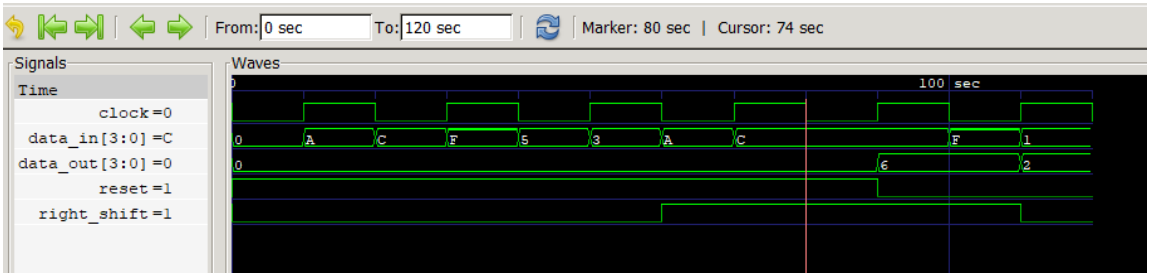


Figure 30: 1x4-bit output Shift Circuit with marker at 80s

At 90 s, we can see that four in is C, four shifted is 6 and right shift is 1

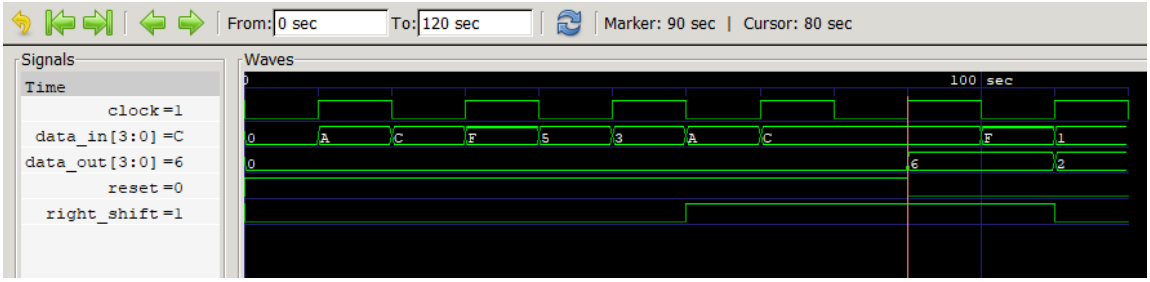


Figure 31: 1x4-bit output Shift Circuit with marker at 90s

At 100 s, we can see that four in is F, four shifted is 6 and right shift is 1

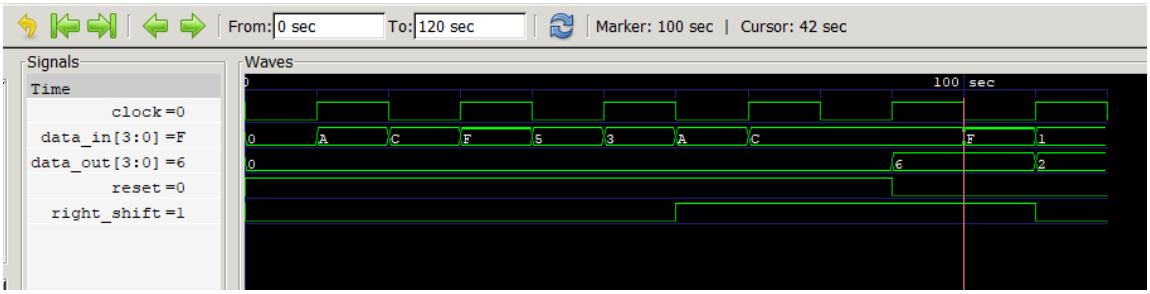


Figure 32: 1x4-bit output Shift Circuit with marker at 100s

At 110 s, we can see that four in is 1, four shifted is 2 and right shift is 0

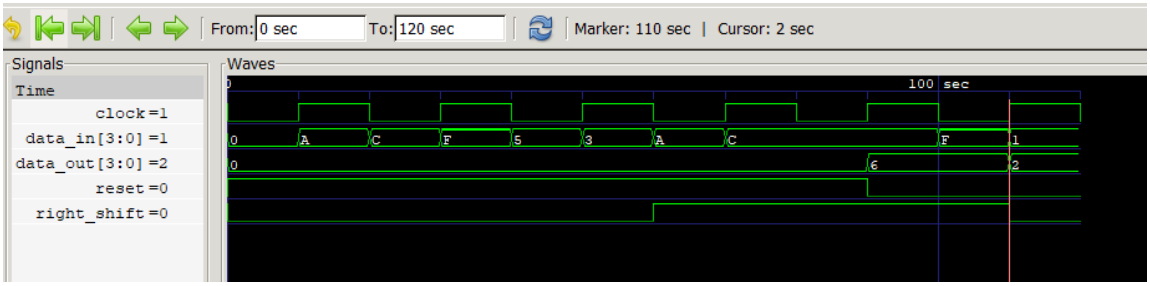


Figure 33: 1x4-bit output Shift Circuit with marker at 110s

At 120 s, we can see that four in is 1, four shifted is 2 and right shift is 0

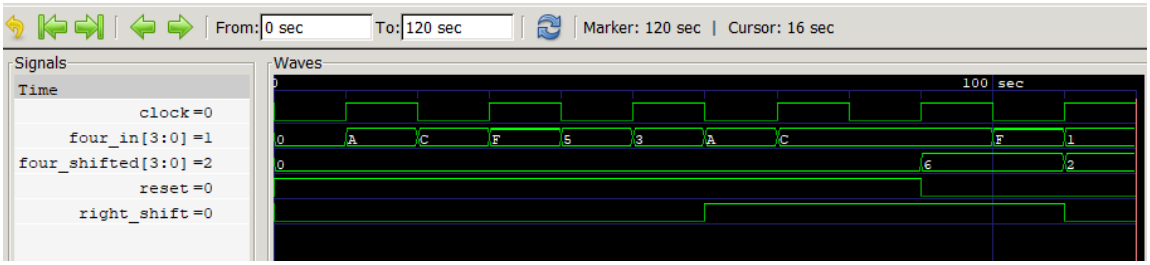


Figure 34: 1x4-bit output Shift Circuit with marker at 120s