Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022 Fall

Experiment 4 - Combinational Circuits in Verilog

December 10, 2022

Student name:
Muhammed Tahir Akdeniz

Student Number:
b2200356027

1 Problem Definition

1.1 Background

1.1.1 Combinational Circuits

Combinational circuits are circuits whose outputs, at any instant of time, depend only on the present inputs (the combinational circuits do not use any memory elements). That is, the previous inputs or state of the circuit do not have any effect on its present state.

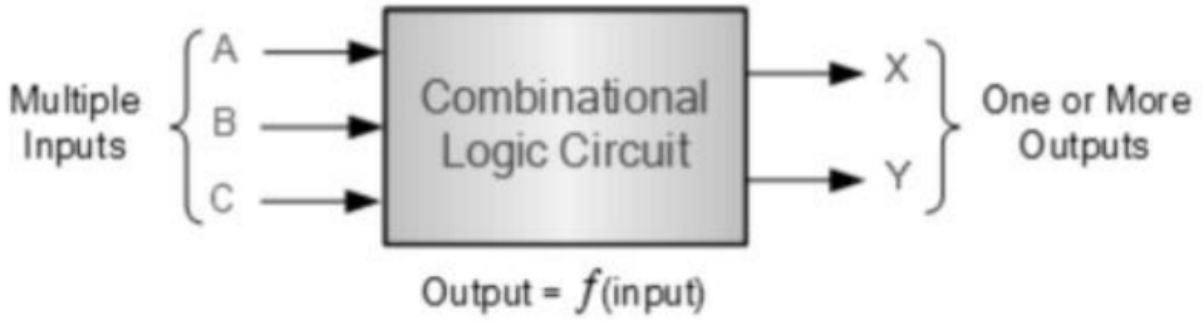

Figure 1: Combinational Logic Circuit

A combinational circuit performs a specific information-processing operation fully specified logically by a set of Boolean functions. The 'n' input variables come from an external source whereas the 'm' output variables go to an external destination. In many applications, the source or destination are storage registers.

1.1.2 Combinational Circuit Design Steps

To design a combinational circuit, start with the problem definition and use the following steps:

1. Identify the number of inputs and outputs of the circuit from the given specifications and assign them letter symbols.
2. Derive the truth table for each of the outputs based on their relationships to the inputs.
3. Simplify the Boolean function for each output (e.g. using Karnaugh Maps or Boolean algebra).
4. Construct the logic circuit diagram using Boolean functions obtained from the Step-3.

1.2 2's Complementer

The binary number system is one of the most popular number representation techniques used in digital systems, in which there are only two digits: 0 (off) and 1 (on). Two's complement is the way

a computer uses to represent signed (positive, negative, and zero) integers. It is a mathematical operation to reversibly convert a positive binary number into a negative binary number with an equivalent (but negative) value. When the most significant bit is a one, the number is signed as negative. Two's complement is obtained by inverting (i.e. flipping) all bits, then adding a 1 to the inverted number. A 4-bit 2's complementer circuit takes a 4- bit wide binary number as input, and produces a single 4-bit wide output that corresponds to its 2's complement. E.g., If the input is binary coded 3 (0011), the output is binary coded -3 (1101).

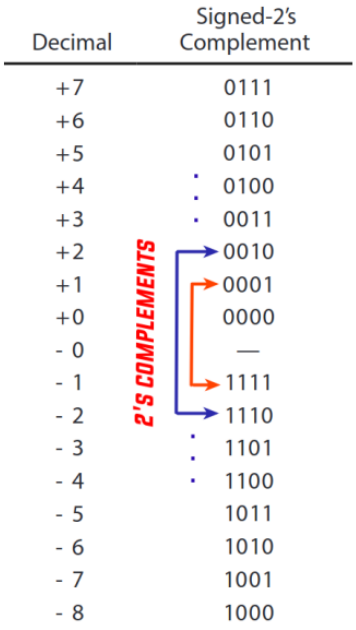

Figure 2: More Examples Of Two's Complement

1.3 Multiplexer

A multiplexer (MUX) is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. A MUX has a maximum of 2 n data inputs. One of the inputs is connected to the output based on the value of the selection line(s). There will be 2 n possible combinations of 1s and 0s since there are 'n' selection lines

1.3.1 4-bit 2-to-1 MUX

In a 4-bit 2-to-1 MUX, only one out of two 4-bit wide inputs is selected as the output based on a 1-bit select signal.

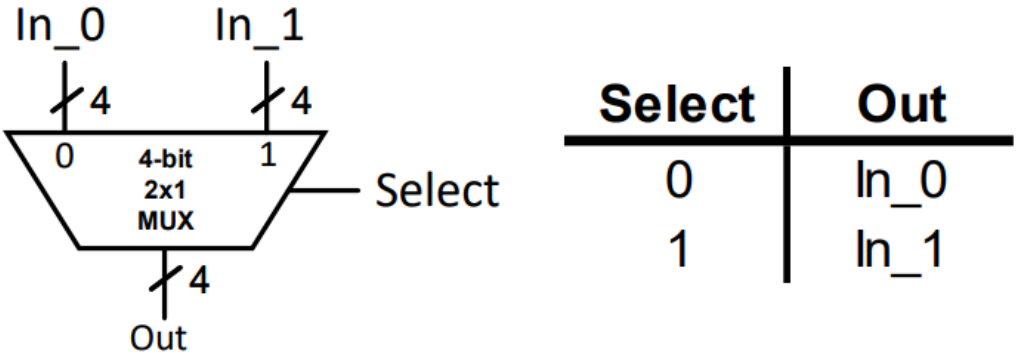

Figure 3: 4-bit 2-to-1 MUX

1.4 4-Bit Full Adder

A full adder is a combinational logic circuit that forms the arithmetic sum of three binary numbers. A full adder consists of three inputs and two outputs. Two of the input variables denoted by A and B represent the two numbers to be added. The remaining input variable Cin represents the carry from the previous summation. The two outputs of the logic circuit consist of the summation result and output carry Cout .

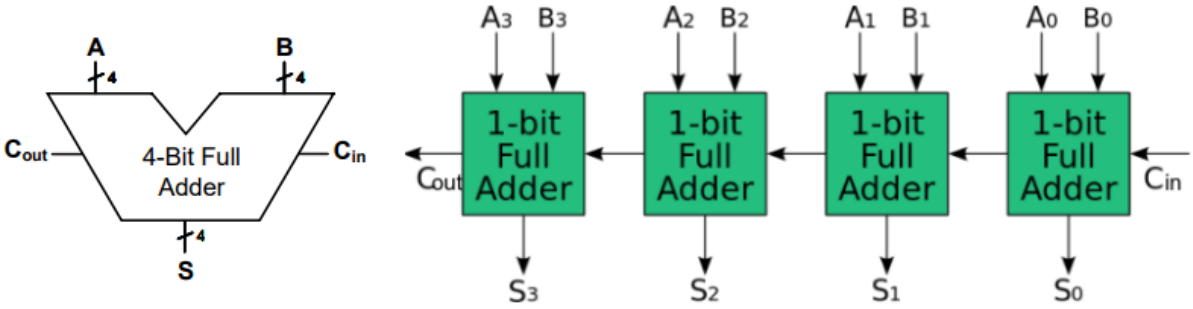

Figure 4: 4-Bit Ripple Carry Adder Implementation

1.5 Adder/Subtractor

Combinational logic circuits can be connected together to form larger circuits. This is done by taking outputs from one circuit and using them as inputs to another. In the figure below, a circuit diagram of a 4-bit adder/substractor circuit implemented with the previously defined modules is given.

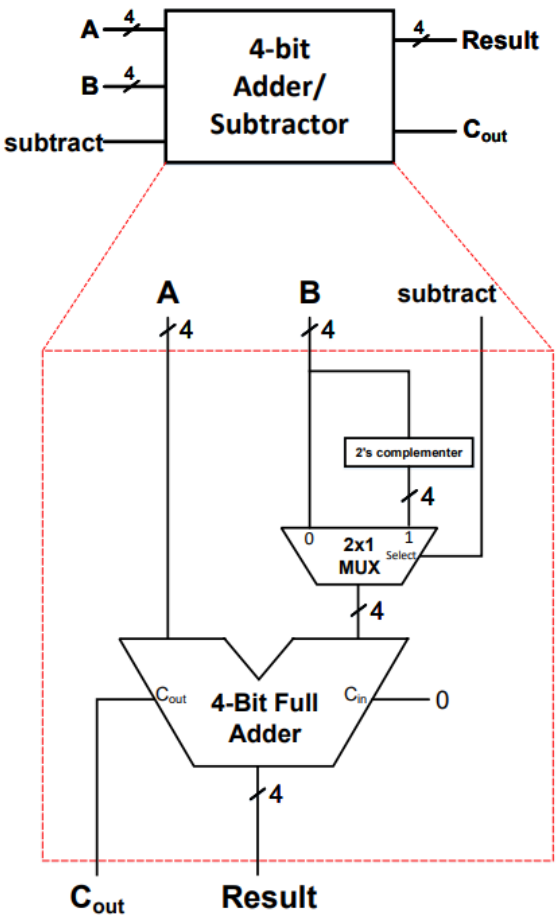

Figure 5: Adder/Subtractor Implementation

A 4-bit adder/substractor circuit has two 4-bit inputs A and B, and a 1-bit input subtract. A and B are the numbers that will be either added or subtracted, while subtract is the mode signal: when subtract is 1 (HIGH), subtraction should be performed (Result = A - B), whereas when subtract is 0 (LOW), addition should be performed (Result = A + B). This circuit has two outputs: one 4-bit output Result and one 1-bit output Cout. Result will output the result of the desired computation on the input numbers A and B (either their sum or difference), whereas Cout will output any carry-out resulting from the calculation. This circuit shows us that it is possible to implement a subtractor using an adder. The idea is to obtain A - B by actually calculating A + (-B), where -B is obtained by 2's complementing B. The multiplexer serves as the selector circuit to pass either B or -B to the adder based on the selected mode of operation via the signal subtract.

2 Solution Implementation

2.1 2's Complementer

	Binary Code				2's Complement			
	In[3]	In[2]	In[1]	In[0]	Out[3]	Out[2]	In[1]	In[0]
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1
2	0	0	1	0	1	1	1	0
3	0	0	1	1	1	1	0	1
4	0	1	0	0	1	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	1	1
10	1	0	1	0	1	1	1	0
11	1	0	1	1	1	0	1	0
12	1	1	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	0	1	0
15	1	1	1	1	0	0	0	1

Figure 6: Truth Table of 2's Complementer

```
module two_s_complement(In,Out);
    input [3:0] In;
    output [3:0] Out;

    assign Out[0] = In[0];
    assign Out[1] = In[1] ^ In[0];
    assign Out[2] = In[2] ^ (In[1] | In[0]);
    assign Out[3] = In[3] ^ (In[1] | In[2] | In[0]);
endmodule
```

2.2 Multiplexer

```
module four_bit_2x1_mux(In_1, In_0, Select, Out);
        input [3:0] In_1;
```

```
        input [3:0] In_0;
        input Select;
        output [3:0] Out;

        assign Out = Select ? In_1 : In_0;

endmodule
```

2.3 4-Bit Full Adder

2.3.1 1-Bit Full Adder

```
module full_adder(
    input A,
    input B,
    input Cin,
    output S,
    output Cout
);

    assign {Cout, S} = A + B + Cin;

endmodule
```

2.3.2 4-Bit Ripple Carry Adder

```
module four_bit_rca(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] S,
    output Cout
);

    wire w1, w2, w3;

    full_adder fa1(A[0], B[0], Cin, S[0], w1);
    full_adder fa2(A[1], B[1], w1, S[1], w2);
    full_adder fa3(A[2], B[2], w2, S[2], w3);
    full_adder fa4(A[3], B[3], w3, S[3], Cout);

endmodule
```

2.4 Adder/Subtractor

```
module four_bit_adder_subtractor(A, B, subtract, Result, Cout);
    input [3:0] A;
    input [3:0] B;
    input subtract;

    output [3:0] Result;
    output Cout;

    wire [3:0] bcom;
    wire [3:0] bMux;

    reg cin = 0;

    two_s_complement tc(B, bcom);
    four_bit_2x1_mux mux(bcom, B, subtract, bMux);
    four_bit_rca rca(A, bMux, cin, Result, Cout);

endmodule
```

3 Testbench Implementation

3.1 2's Complementer

```
`timescale 1ns/10ps
module two_s_complement_tb;

   reg [3:0] In;
   wire [3:0] Out;
   reg [3:0] num = 4'b0000;

   two_s_complement test(In, Out);

   initial begin
      $dumpfile("two_s_complement.vcd");
      $dumpvars;

      for(integer i = 0; i < 16; i++) begin
         In = num;
         #10
         num += 1;
      end
   end

endmodule
```

3.2 Multiplexer

```
`timescale 1ns/10ps
module four_bit_2x1_mux_tb;
        reg [3:0] In_1;
        reg [3:0] In_0;
        reg Select;
        wire [3:0] Out;
        reg [8:0] num = 9'b000000000;

        four_bit_2x1_mux test(In_1, In_0, Select, Out);

        initial begin
                $dumpfile("result2.vcd");
                $dumpvars;

                for(integer i = 0; i < 512; i++) begin
                        {Select, In_1, In_0} = num;
                        #10
                        num += 1;
                end
        end

endmodule
```

3.3 4-Bit Full Adder

3.3.1 1-Bit Full Adder

```
`timescale 1 ns/10 ps
module full_adder_tb;
    reg A;
    reg B;
    reg Cin;
    wire S;
    wire Cout;

    reg [2:0] num = 3'b000;
    full_adder test(A, B, Cin, S, Cout);

    initial begin
        $dumpfile("full_adder.vcd");
        $dumpvars;

        for(integer i = 0; i < 8; i++) begin
            {Cin, A, B} = num;
            #10
```

```
            num += 1;
        end
    end

endmodule
```

3.3.2 4-Bit Ripple Carry Adder

```
`timescale 1 ns/10 ps
module four_bit_rca_tb
;
  reg [3:0] A;
  reg [3:0] B;
  reg Cin;

  wire Cout;
  wire [3:0] S;


  reg [8:0] num = 9'b000000000;


  four_bit_rca test(A, B, Cin, S, Cout);

  initial begin
        $dumpfile("four_bit_rca.vcd");
        $dumpvars;

        for(integer i = 0; i < 512; i++) begin
            {Cin, A, B} = num;
            #10
            num += 1;
        end
    end
endmodule
```

3.4 Adder/Subtractor

```
`timescale 1ns/1ps
module four_bit_adder_subtractor_tb;
    reg [3:0] A;
    reg [3:0] B;
    reg subtract;

    wire [3:0] Result;
    wire Cout;
```

```
9
10      reg [8:0] num = 9'b000000000;
11
12      four_bit_adder_subtractor test(A, B, subtract, Result, Cout);
13
14      initial begin
15          $dumpfile("four_bit_adder_subtractor.vcd");
16          $dumpvars;
17
18          for(integer i = 0; i < 512; i++) begin
19              {subtract, A, B} = num;
20              #10
21              num += 1;
22          end
23      end
24  endmodule
```

4 Results

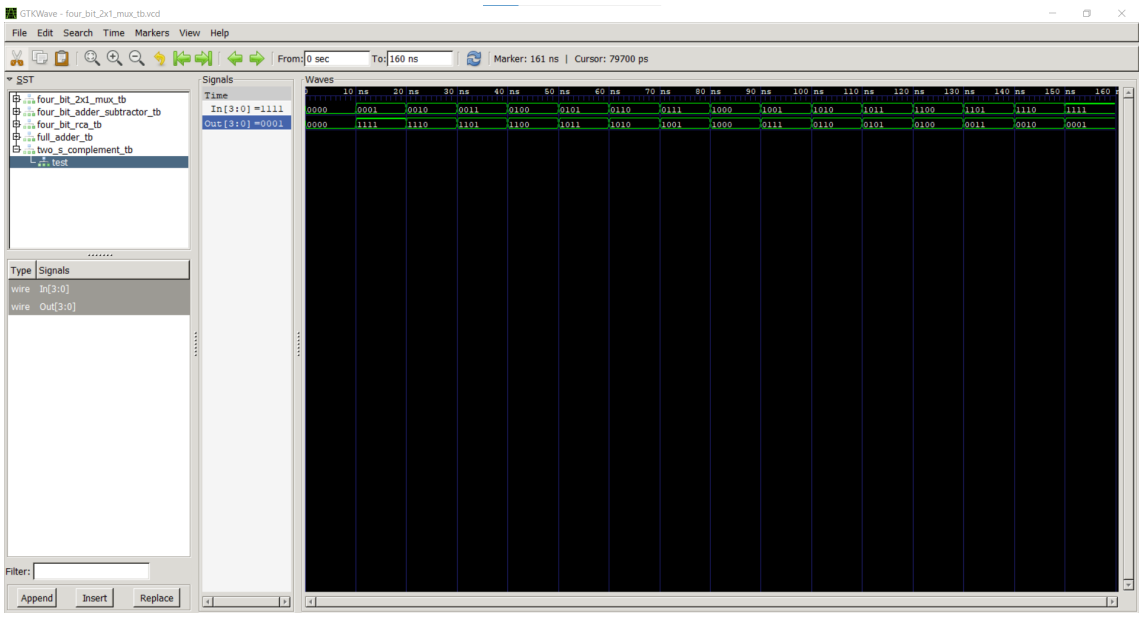

Figure 7: Two's complement module test

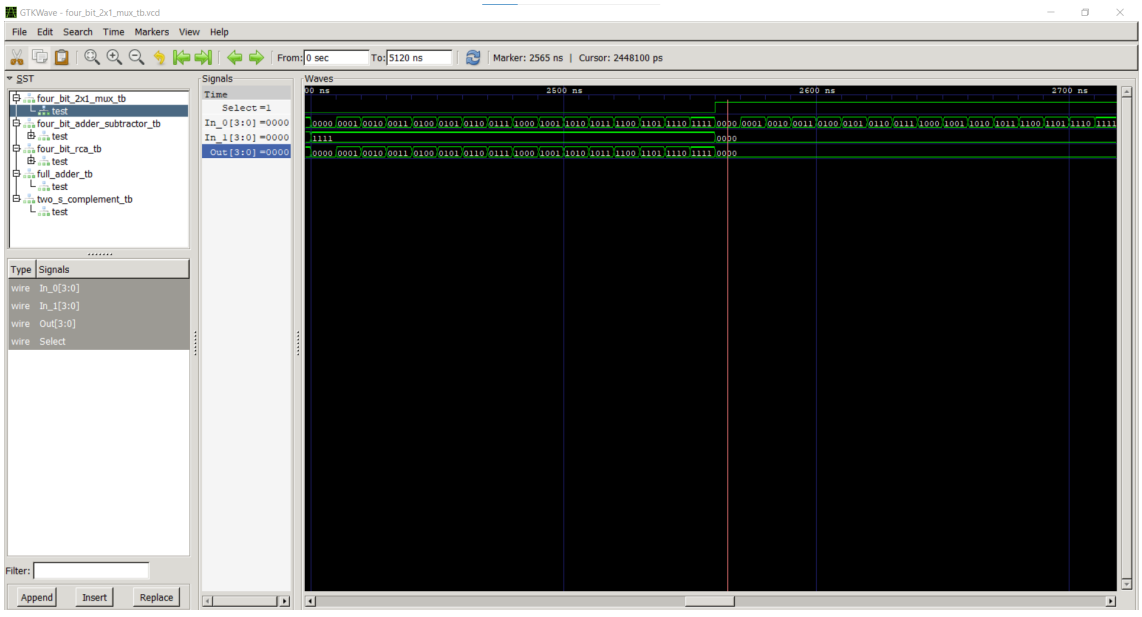

Figure 8: 4-bit 2-to-1 MUX module test

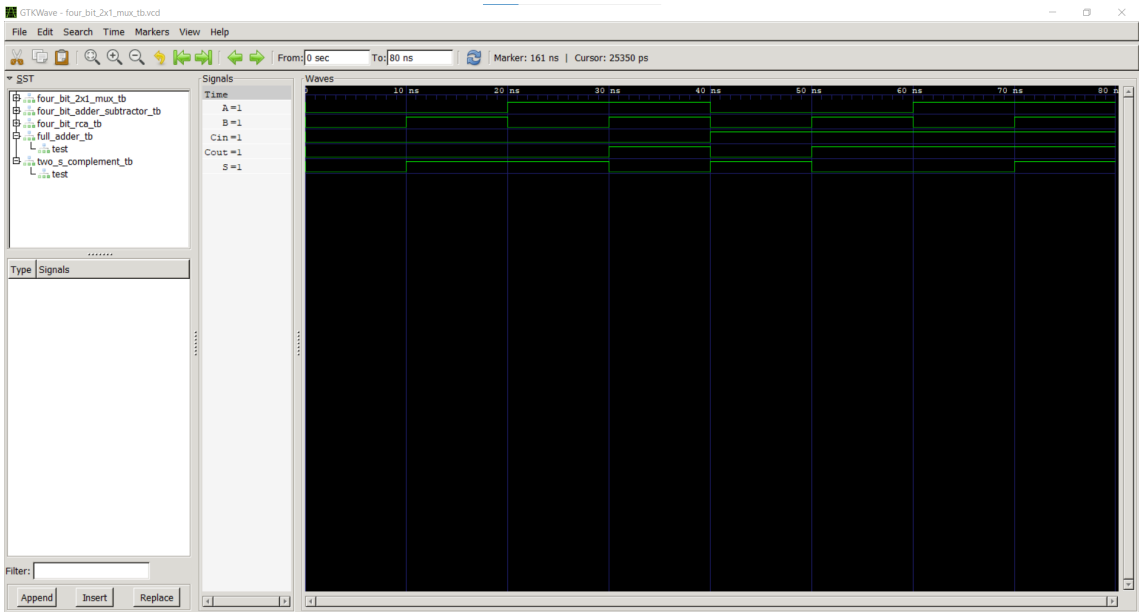

Figure 9: FullAdder module test

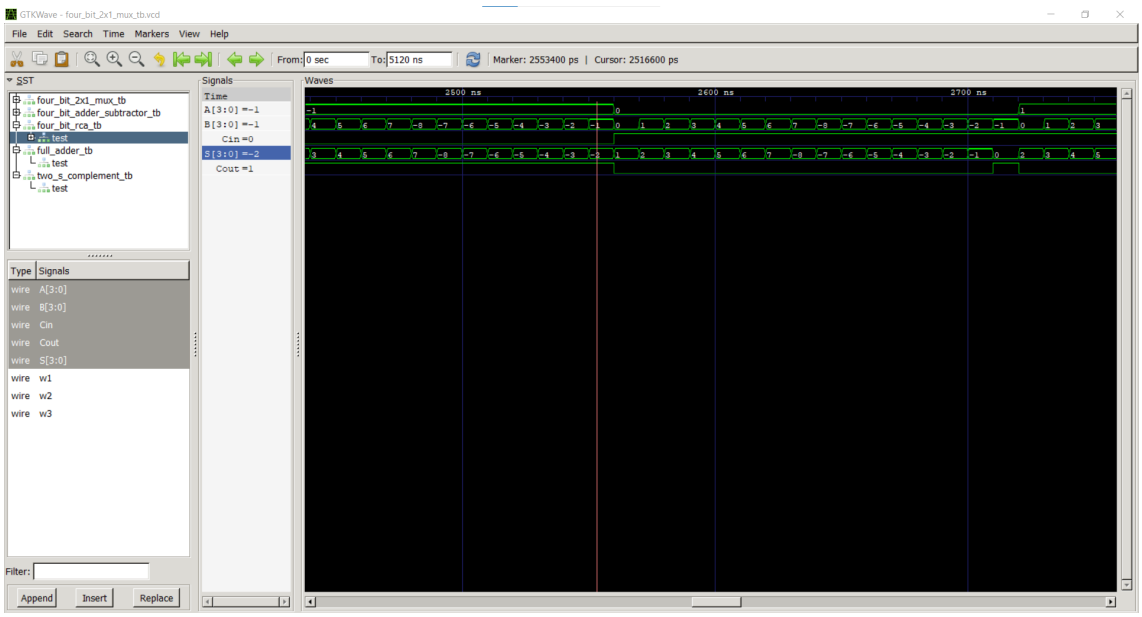

Figure 10: Four Bit Rca module test

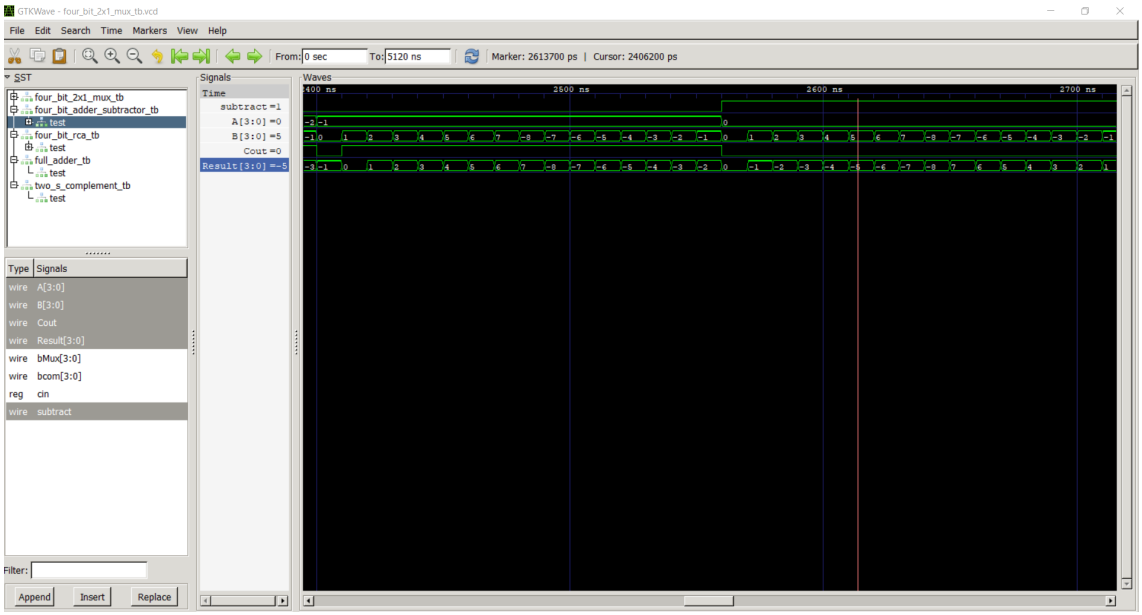

Figure 11: Adder/Substractor module test

References

- Verilog HDL A Brief Introduction, BBM203 Lecture Notes
- Part 5 – Midterm Review, BBM231 Lecture Notes