Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022-2023 Fall

Combinational Circuits in Verilog

December 10, 2022

Student name:
Eren Gül

Student Number:
b2200356037

1 Problem Definition

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).

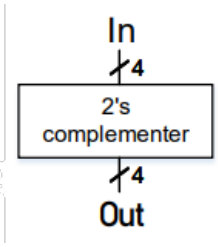

Figure 1: 2's complementer

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. 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.

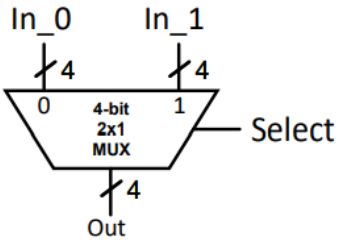

Figure 2: 4-bit 2-to-1 MUX

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. A 4-Bit Ripple Carry Adder can be implemented by instantiating four 1-Bit Full Adder modules.

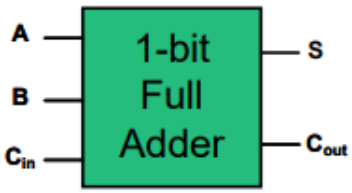

Figure 3: 1-bit full adder

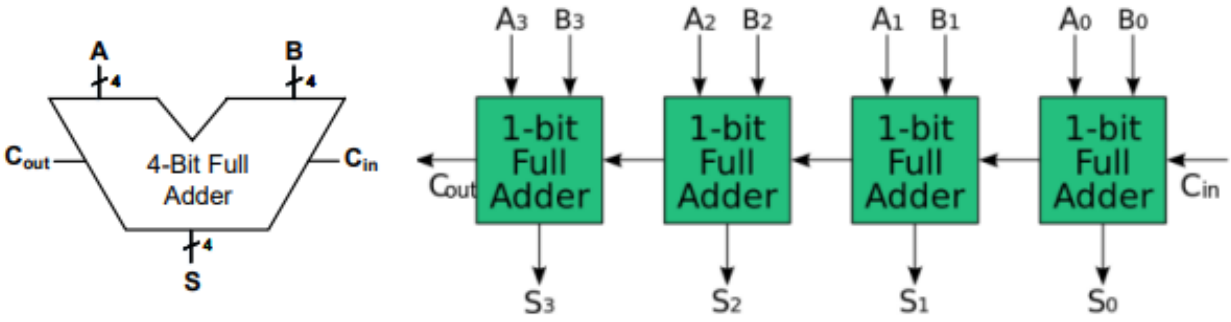

Figure 4: 4-bit ripple carry adder

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. A 4-bit adder/substractor circuit can be implemented with the previously defined modules.

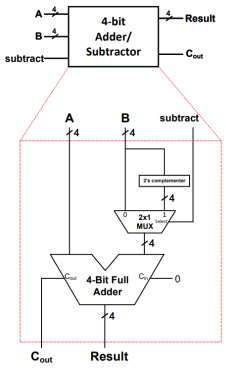

Figure 5: 4-bit adder/substractor

2 Solution Implementation

2's complementer

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

    // Assign values to outputs
    assign Out[3] = In[3] ^ (In[2] | In[1] | In[0]);
    assign Out[2] = In[2] ^ (In[1] | In[0]);
    assign Out[1] = In[1] ^ In[0];
    assign Out[0] = In[0];
endmodule
```

4-bit 2-to-1 MUX

```
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 value to output
    assign Out = Select ? In_1 : In_0;
endmodule
```

1-bit full adder

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

    // Assign values to outputs
    assign S = (A ^ B) ^ Cin;
    assign Cout = (A & B) | (B & Cin) | (Cin & A);

endmodule
```

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[2:0] Carries; // Declaring carries as wire
    // Using full_adder module for structural design with explicit association
    full_adder FA1(.A(A[0]), .B(B[0]), .Cin(Cin), .S(S[0]), .Cout(Carries[0]));
    full_adder FA2(.A(A[1]), .B(B[1]), .Cin(Carries[0]), .S(S[1]), .Cout(Carries[1]))
    full_adder FA3(.A(A[2]), .B(B[2]), .Cin(Carries[1]), .S(S[2]), .Cout(Carries[2]))
    full_adder FA4(.A(A[3]), .B(B[3]), .Cin(Carries[2]), .S(S[3]), .Cout(Cout));
endmodule
```

4-bit adder/substractor

```
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] ComplementB; // Result of TC
    wire [3:0] MUXResult; // Result of MUX

    // Used previous modules for structural design with explicit association
    two_s_complement TC(.In(B), .Out(ComplementB));
    four_bit_2x1_mux MUX(.In_1(B), .In_0(ComplementB), .Select(~subtract), .Out(MUXRe
    four_bit_rca RCA(.A(A), .B(MUXResult), .Cin(1'b0), .S(Result), .Cout(Cout));

endmodule
```

3 Testbench Implementation

2's complementer

```
`timescale 1ns/10ps
module two_s_complement_tb;
   // Declaring inputs as regs
   reg[3:0] In;
   reg[3:0] count = 4'b0000;
   // Declaring output as wire
   wire[3:0] Out;
   integer i; // Integer used in for loop

   two_s_complement UUT(In, Out); // Instantiate UUT

   initial begin // Generating stimuli
      $dumpfile("result0.vcd");
      $dumpvars;
      for (i = 0; i < 16; i++) begin
         {In[3], In[2], In[1], In[0]} = count;
         count += 1;
```

```
            #10;
        end
        $finish;
    end

endmodule
```

4-bit 2-to-1 MUX

```
`timescale 1ns/10ps
module four_bit_2x1_mux_tb;
        // Declaring inputs as regs
        reg [3:0] In_1;
        reg [3:0] In_0;
        reg Select;
        // Declaring output as wire
        wire [3:0] Out;
        // Declaring integers to use in for loops
        integer i;
        integer j;
        integer k;
        // Declaring binary numbers as regs to test all cases of inputs
        reg selectCount = 0;
        reg [3:0] count1 = 4'b0000;
        reg [3:0] count0 = 4'b0000;

        four_bit_2x1_mux UUT(In_1, In_0, Select, Out); // Instantiate UUT

        initial begin // Generating stimuli
                $dumpfile("result1.vcd");
        $dumpvars;
        for (i = 0; i < 2; i++) begin
                        Select = selectCount;
                        selectCount += 1;
                        for (j = 0; j < 16; j++) begin
                                // Assign count1 to In_1
                                {In_1[3], In_1[2], In_1[1], In_1[0]} = count1;
                                count1 += 1;
                                for (k = 0; k < 16; k++) begin
                                        // Assign count0 to In_0
                                        {In_0[3], In_0[2], In_0[1], In_0[0]} = count0
                                        count0 += 1;
                                        #10;
                                end
                                count0 = 0;
                                #10;
                        end
                        count1 = 0;
                        #10;
```

```
                end
        $finish;
        end

endmodule
```

1-bit full adder

```
`timescale 1 ns/10 ps
module full_adder_tb;
    reg A, B, Cin; // Declaring inputs as regs
    wire S, Cout; // Declaring outputs as wires
    integer i; // Declaring integer to use in for loop
    // Declaring binary number as reg to test all cases of inputs
    reg [3:0] count = 4'b0000;

    full_adder UUT(A, B, Cin, S, Cout); // Instantiate UUT

    initial begin // Generating stimuli
        $dumpfile("result2.vcd");
        $dumpvars;
        for (i = 0; i < 8; i++) begin
            // Assign count to all inputs
            {A, B, Cin} = count;
            count += 1;
            #10;
        end
        $finish;
    end

endmodule
```

4-bit ripple carry adder

```
`timescale 1 ns/10 ps
module four_bit_rca_tb;
  // Declaring inputs as regs
  reg [3:0] A, B;
  reg Cin;
  // Declaring outputs as wires
  wire [3:0] S;
  wire Cout;
  // Declaring integers to use in for loops
  integer i;
  integer j;
  integer k;
  // Declaring binary numbers as regs to test all cases of inputs
  reg [3:0] countA = 4'b0000;
  reg [3:0] countB = 4'b0000;

```

```
  four_bit_rca UUT(.A(A), .B(B), .Cin(Cin), .S(S), .Cout(Cout)); // Instantiate UUT

  initial begin // Generating stimuli
    $dumpfile("result3.vcd");
    $dumpvars;
    for (i = 0; i < 2; i++) begin
      Cin = i;
      for (j = 0; j < 16; j++) begin
        // Assign countA to input A
        {A[3], A[2], A[1], A[0]} = countA;
        countA += 1;
        for (k = 0; k < 16; k++) begin
          // Assign countB to input B
          {B[3], B[2], B[1], B[0]} = countB;
          countB += 1;
          #10;
        end
        countB = 0;
        #10;
      end
      countA = 0;
      #10;
    end
    $finish;
  end

endmodule
```

4-bit adder/substractor

```
`timescale 1ns/1ps
module four_bit_adder_subtractor_tb;
    // Declaring inputs as regs
    reg [3:0] A, B;
    reg subtract;
    // Declaring outputs as wires
    wire [3:0] Result;
    wire Cout;
    // Declaring integers to use in for loops
    integer i;
    integer j;
    integer k;
    // Declaring binary numbers as regs to test all cases of inputs
    reg [3:0] countA = 4'b0000;
    reg [3:0] countB = 4'b0000;

    // Instantiate UUT
    four_bit_adder_subtractor UUT(.A(A), .B(B), .subtract(subtract), .Result(Result),

```

```
20      initial begin // Generating stimuli
21          $dumpfile("result4.vcd");
22          $dumpvars;
23          for (i = 0; i < 2; i++) begin
24              subtract = i;
25              for (j = 0; j < 16; j++) begin
26                  // Assign countA to input A
27                  {A[3], A[2], A[1], A[0]} = countA;
28                  countA += 1;
29                  for (k = 0; k < 16; k++) begin
30                      // Assign countB to input B
31                      {B[3], B[2], B[1], B[0]} = countB;
32                      countB += 1;
33                      #10;
34                  end
35                  countB = 0;
36                  #10;
37              end
38              countA = 0;
39              #10;
40          end
41          $finish;
42      end
43
44  endmodule
```

4 Results

2's complementer

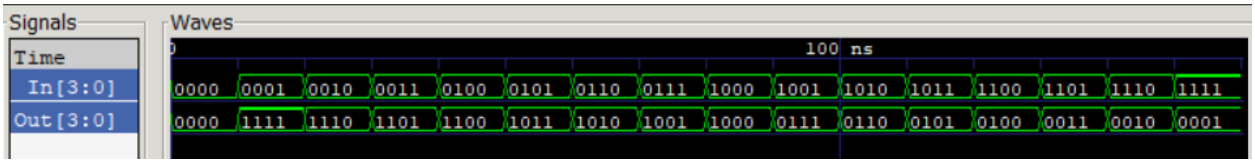

Figure 6: Resulting waveform of 2's complementer

4-bit 2-to-1 MUX

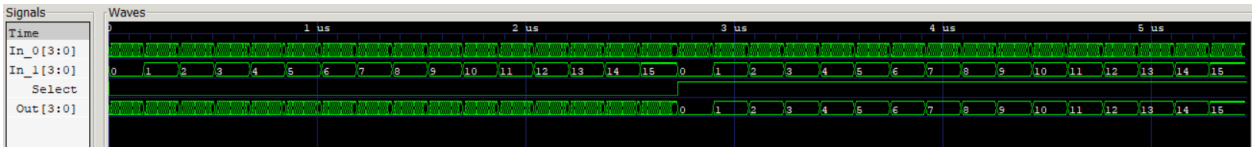

Figure 7: Resulting waveform of 4-bit 2-to-1 MUX

1-bit full adder

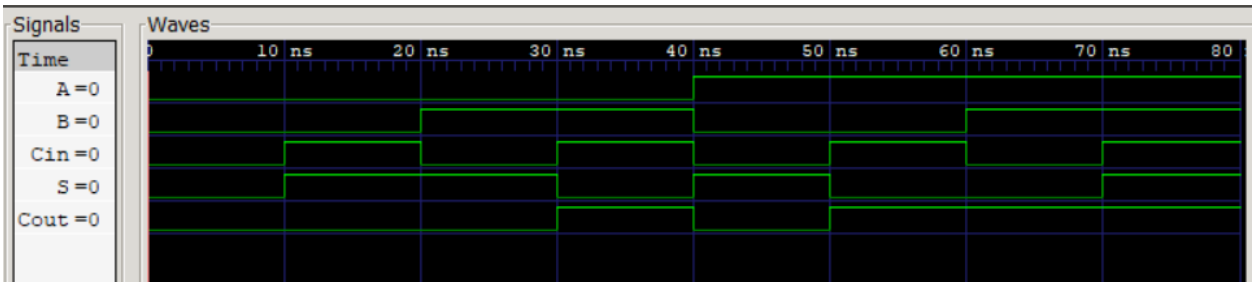

Figure 8: Resulting waveform of 1-bit full adder

4-bit ripple carry adder

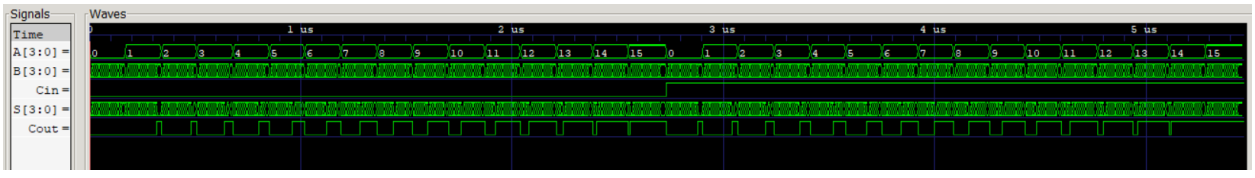

Figure 9: Resulting waveform of 4-bit ripple carry adder

4-bit adder/substractor

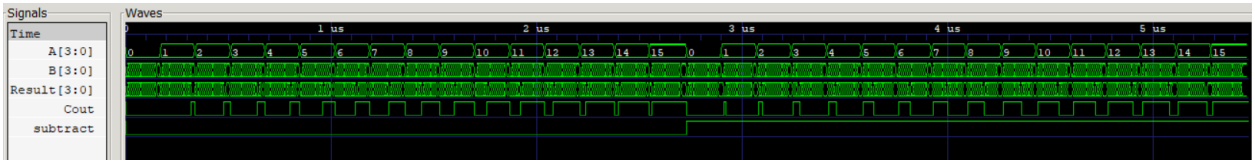

Figure 10: Resulting waveform of 4-bit adder/substractor

References

- https://cdn-uploads.piazza.com/paste/itmvemweb267cd/1e748d391563184df77907a5a81401e0692b VerilogIntro.pdf
- https://cdn-uploads.piazza.com/paste/itmvemweb267cd/20ca9bf7d8564d6df9dde83eddff107faef5 IcarusVerilogGuide2022.pdf