Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Digital Systems ENCS2340

Supervisor:Dr. Bilal Karaki

Verilog Project Report

Ahmed Hamdan
1210241

February, 1st, 2023

Contents

What are Digital Systems

Digital systems are designed to store, process, and communicate information in digital form. They are found in a wide range of applications, including process control, communication systems, digital instruments, and consumer products. The digital computer, more commonly called the *computer*, is an example of a typical digital system.

A computer manipulates information in digital, or more precisely, binary form. A binary number has only two discrete values — zero or one. Each of these discrete values is represented by the OFF and ON status of an electronic switch called a *transistor*. All computers, therefore, only understand binary numbers.

What is an arithmetic- logic-unit (ALU)

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words.

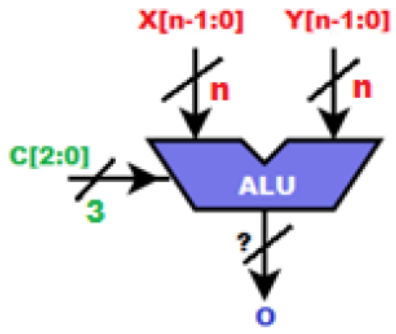
In some processors, the ALU is divided into two units: an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU (e.g. one for fixed-point operations and another for floating-point operations).

Talking about the given ALU in this project

As illustrated, our ALU is divided into two units. When C [3] is 0, the logic unit operates and when C [3] is 1, the arithmetic unit operates. Our main problem which is the number of bits in the output, since it differs from one operation to another, but this problem can be solved by extending the number based on the most significant bit.

P.S. Every .V file can be found in this folder along with the report.

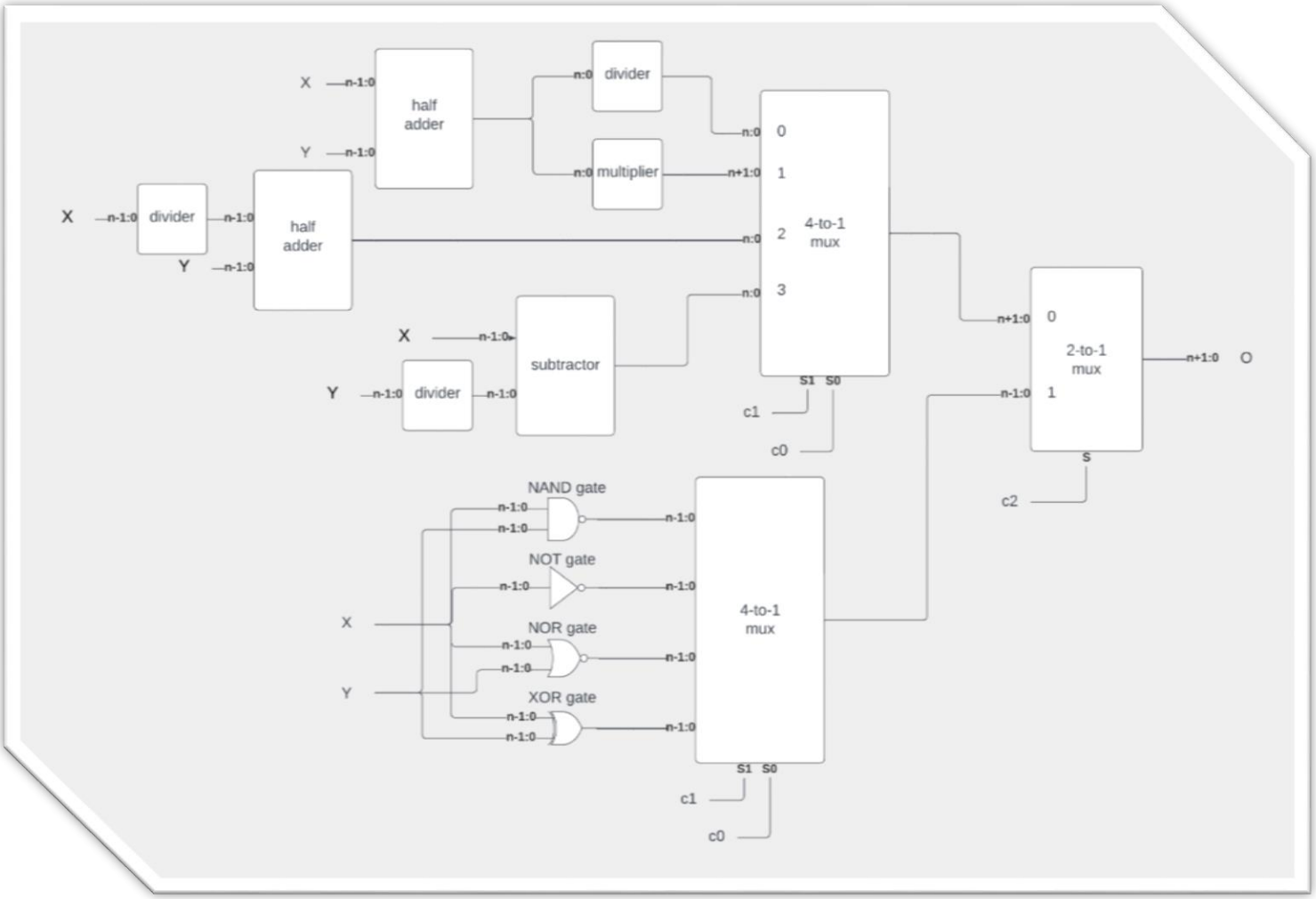
The ALU symbol and supported functions are represented as follow:

ALU Function Code (C)	ALU Output (O)	ALU Symbol
000	(**X**+**Y**)/2	
001	2*(**X**+**Y**)	
010	(**X**/2)+**Y**	
011	**X**-(**Y**/2)	
100	**X** NAND **Y**	
101	NOT(**X**)	
110	**X** NOR **Y**	
111	**X** XOR **Y**	

Part A - Specify the size of the output (**O**) in bits so the overflow can never occur.

- The maximum value of O is when X and Y have the maximum positive value ($2^{n-1} - 1$) and the minimum value of O is when X and Y have the minimum negative value (-2^{n-1}).
- The logical operations which function when $C_3 = 1$ do not affect the number of bits. An n-bits input requires an n-bits to represent the value.
- Division in binary is equivalent to shifting the number to the right. An n-bits requires n-1 bits to represent the value.
- Multiplication in binary is equivalent to shifting the number to the left. An n-bits input requires n+1 bits to represent the value.
- When C = 001, the maximum value of O is $2 * ((2^{n-1} - 1) + (2^{n-1} - 1)) = 2^{n+1} - 4$ which requires n+2 bits to represent this value, and the minimum value of O is $2 * ((-2^{n-1}) + (-2^{n-1})) = -2^{n+1}$ which requires n+2 bits to represent this value.
- Answer: To avoid overflow, O must have at least n+2 bits.

Part B - Show the ALU implementation using medium-scale integration components and minimum number of gates.



Additional explanation:

- Since we are handling signed numbers the extensions are added automatically when assigning an n-1 bit number to an n bit output.
- The division and multiplication have been done in modules.

Part C - Write behavioral Verilog modules for your elements you defined in part (b).

Multiplexers implementation

2-to-1 Multiplexer

```
module mux2to1_1210241 #(parameter n=4) (input signed [n+1:0] I0,input signed [n+1:0] I1, input Selection, output reg signed [n+1:0] O);
    always@(*)
    begin
        if(Selection)
            O = I1;
        else
            O = I0;
    end
endmodule
```

4-to-1 Multiplexer

```
module mux4to1_1210241 #(parameter n=4)( input signed [n:0] I0,input signed [n+1:0] I1,input signed [n:0] I2,I3,input [1:0] Selection, output reg signed [n+1:0] O);
  always @(*)
    begin
      case (Selection)
        0: O=I0;
        1: O=I1;
        2: O=I2;
        3: O=I3;
        default: O=0;
      endcase
    end
endmodule
```

Logical gates implementation

NAND gate

```
module nand_1210241 #(parameter n=4) (input [n-1:0] X, Y, output reg [n-1:0] O);
    always@(*)
    begin
        O = ~(X & Y);
    end
endmodule
```

NOT gate

```
module not_1210241 #(parameter n=4) (input [n-1:0] X, output reg [n-1:0] O);
    always@(*)
    begin
        O = !X;
    end
endmodule
```

NOR gate

```
module nor_1210241 #(parameter n=4) (input [n-1:0] X,Y, output reg [n-1:0] O);
    always@(*)
    begin
        O = ~(X | Y);
    end
endmodule
```

XOR gate

```
module xor_1210241 #(parameter n=4) (input [n-1:0] X,Y, output reg [n-1:0] O);
    always@(*)
    begin
        O = X ^ Y;
    end
endmodule
```

Athematic operators' implementation

Multiplier

```
module Multiplication_1210241 #(parameter n = 4) (input signed [n:0] X, output reg [n+1:0] O);
always@(*)
    begin
        O = X * 2;
    end
endmodule
```

Divider for n bits

```
module Division_1210241 #(parameter n = 4) (input signed [n-1:0] X, output reg signed [n-1:0] O);
always@(*)
    begin
        O = X / 2;
    end
endmodule
```

Divider for n+1 bits

```
module DivisionHigher_1210241 #(parameter n = 4) (input signed [n:0] X, output reg signed [n:0] O);
always@(*)
    begin
        O = X / 2;
    end
endmodule
```

Adder

```
module Adder_1210241 #(parameter n=4) (input signed [n-1:0] X,Y, output reg signed  [n:0] S);
  always @(*)
    begin
     S = X + Y;
    end
endmodule
```

Subtractor

```
module Subtractor_1210241 #(parameter n=4) (input signed [n-1:0] X,Y, output reg signed  [n-1:0] S);
  always @(*)
    begin
     S = X - Y;
    end
endmodule
```

Part D - Write a structural Verilog model for your ALU designed in Part (b) using the elements you defined in Part (c).

What is Structural Model in Verilog

The structural model describes a system using basic components such as digital gates and adders. In structural modeling, the programmer or the designer thinks bout the circuit as a box or a module. It is encapsulated from the outer environment. In other words, it communicates with the outer environment through inputs and outputs.

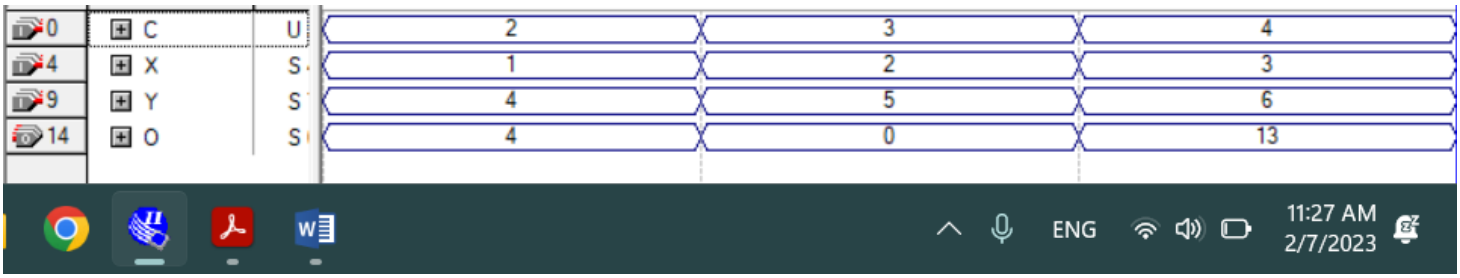
Moreover, it is possible to describe the structure inside a module using gates and submodules. Also, it defines how these modules are connected to each other and to the module ports. Furthermore, the structural model helps to draw a schematic diagram for the circuit

```
module ALU_1210241 #(parameter n=4) (input [2:0] C, input signed [n-1:0] X,Y, output signed [n+1:0] O);
// parameter set to 4 for the test cases
// creating wires
wire signed [n-1:0] XHalf, YHalf;
wire signed [n:0] XplusY, SECOND, THIRD, plusHalf;
wire signed [n+1:0] plusDouble,mux_arth;
wire signed [n+1:0] nand_XY, not_X, nor_XY, xor_XY, mux_logic;

Adder_1210241 add (X, Y,XplusY); // Adds X and Y and saves the value in XplusY
DivisionHigher_1210241 DivideXplusY (XplusY, plusHalf); // Divides X + Y by 2 (OPERATION 000)
Multiplication_1210241 MultiplyXplusY (XplusY, plusDouble); // Multiplies X + Y by 2 (OPERATION 001)

Division_1210241 DivideX (X,XHalf); // Divides X and saves the value in XHalf
Adder_1210241 add2 (XHalf, Y, SECOND); // Adds X/2 and Y and saves the value in SECOND (OPERATION 010)

Division_1210241 DivideY (Y,YHalf); // Divides Y and saves the value in YHalf
Subtractor_1210241 add3 (X, YHalf, THIRD); // Subtracts X and Y/2 and saves the value in THIRD (OPERATION 011)

mux4to1_1210241 MUXLOL (plusHalf, plusDouble, SECOND, THIRD, C[1:0], mux_arth);

// logical operations using their designed functions
nand_1210241 NANDXY (X,Y,nand_XY);
not_1210241 NOTX (X, not_X);
nor_1210241 NORX (X,Y,nor_XY);
xor_1210241 XORX (X,Y,xor_XY);

// muxLogic decides whether the output is nand or not or nor or xor
mux4to1_1210241 muxLogic (nand_XY, not_X, nor_XY, xor_XY, C[1:0], mux_logic);
// MUXO decides whether the output
mux2to1_1210241 MUXO (mux_arth, mux_logic, C[2], O);

endmodule
```

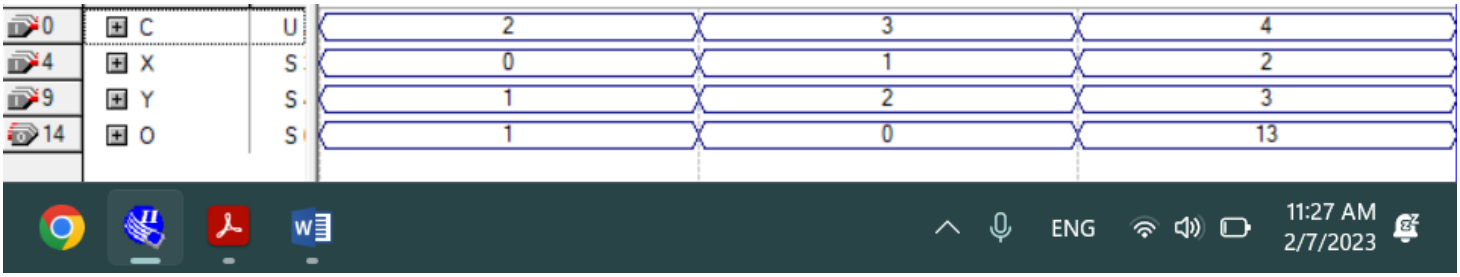
The structural model for my ALU

Part E - Generate the waveforms of the ALU in part (D)

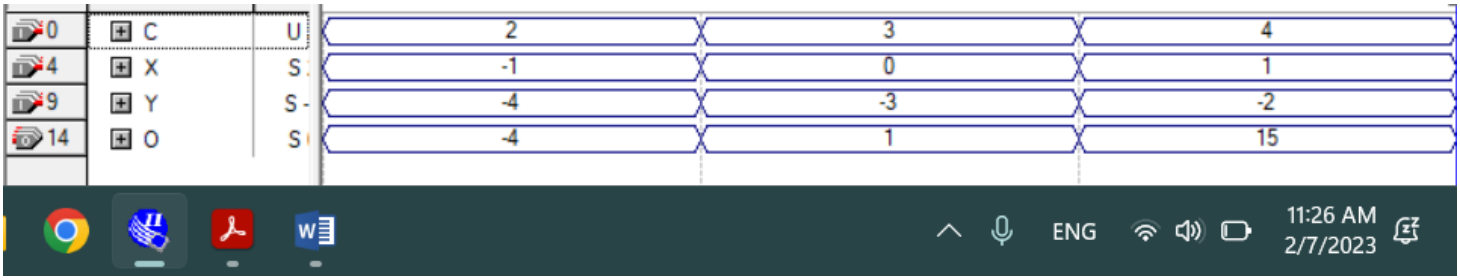
First Case (X = 1, Y = 4, C = 2, O = X/2 + Y = 4)



Second Test Case (X = 0, Y = 1, C = 2, O = X/2 + Y = 1)



Third Test Case (X = -1, Y = -4, C = 2, O = X/2 + Y = -4)



Part F - Write a single behavioral Verilog module that models the designed ALU.

What is behavioral Model in Verilog

Behavioral models in Verilog contain procedural statements, which control the simulation and manipulate variables of the data types. These all statements are contained within the procedures. Each of the procedure has an activity flow associated with it.

During simulation of behavioral model, all the flows defined by the 'always' and 'initial' statements start together at simulation time 'zero'. The initial statements are executed once, and the always statements are executed repetitively. The initial statement is then completed and is not executed again during that simulation run. This initial statement is containing a begin-end block (also called a sequential block) of statements.

```
module ALU_1210241 #(parameter n=4) (input [2:0] C,
input signed [n-1:0] X,Y, output reg signed  [n+1:0] O);
always@ (*) // start sequentially and build the sensitivity list for me
begin // start the sequence
case(C) // using switch case
0: O = (X + Y) / 2; // (X+Y)/2
1: O = 2 * (X + Y); // 2*(X+Y)
2: O = X / 2 + Y; //(X/2)+Y
3: O = X - Y / 2; // X-(Y/2)
4: O = {2'b00,~(X & Y)}; // X NAND Y
5: O = {2'b00,~X}; // NOT(X)
6: O = {2'b00,~(X | Y)}; // X NOR Y
7: O = {2'b00,X ^ Y}; // X XOR Y
default: O = 0; // default value
endcase // end the switch case
end // end the always@(*)
endmodule // end the module
```

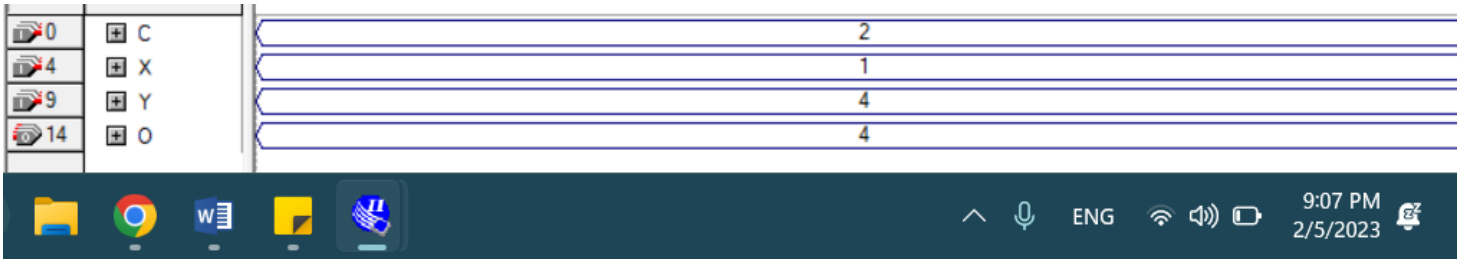
The behavioral model for my ALU

Additional explanation:

- This was done by using switch case conditional method
- {2'b00} was added for the zero extension

Part G - Generate the waveforms of the behavioral ALU in part (F).

First Test Case (X = 1, Y = 4, C = 2, O = X/2 + Y = 4)



Second Test Case (X = 0, Y = 1, C = 2, O = X/2 + Y = 1)

Third Test Case (X = -1, Y = -4, C = 2, O = X/2 + Y = -4)