Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS2340**: Digital Systems**
First Semester, 2023/2024

Student Name : Aya Abed Al-Rahman Fares

Student Id : 1222654

Instructor : Dr. ismael

Section.no : 1

Date : 20/12/2023

Abstract:

The project focuses on the development of a basic Arithmetic Logic Unit (ALU) using Verilog Hardware Description Language (HDL). The ALU is designed to perform four fundamental arithmetic and logic operations: addition, subtraction, bitwise AND, and bitwise OR.

Theory:

These truth tables illustrate the output for each operation based on the given inputs. The ALU's design utilizes these tables to determine the output Result based on the selected OpCode.

1. Addition :

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Explanation: The addition operation produces a sum bit and a carry bit. In this case, the XOR gate generates the sum bit, and the AND gate produces the carry bit.

2. Subtraction :

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Explanation: Subtraction can be implemented using two XOR gates for the difference and borrow bits. The borrow bit indicates whether a borrow is required during subtraction.

3. AND :

A	B	Result
0	0	0
0	1	0
1	0	0
1	1	1

Explanation: Bitwise AND operation performs the AND operation on each corresponding bit of the operands, producing a result with a 1 only when both input bits are 1.

4. OR:

A	B	Result
0	0	0
0	1	1
1	0	1
1	1	1

Explanation: Bitwise OR operation performs the OR operation on each corresponding bit of the operands, producing a result with a 1 when at least one of the input bits is 1.

5.ALU :

The truth table for the Arithmetic Logic Unit (ALU) summarizes the output based on the combination of inputs (A and B) and the control input (OpCode) :

| A | B | OpCode | Result (ALU) |

| 0 | 0 | 3'b000 | 0 |

| 0 | 1 | 3'b000 | 1 |

| 1 | 0 | 3'b000 | 1 |

| 1 | 1 | 3'b000 | 0 |

| 0 | 0 | 3'b001 | 0 |

| 0 | 1 | 3'b001 | 1 |

| 1 | 0 | 3'b001 | 1 |

| 1 | 1 | 3'b001 | 0 |

| 0 | 0 | 3'b010 | 0 |

| 0 | 1 | 3'b010 | 0 |

| 1 | 0 | 3'b010 | 0 |

| 1 | 1 | 3'b010 | 1 |

| 0 | 0 | 3'b011 | 0 |

| 0 | 1 | 3'b011 | 1 |

| 1 | 0 | 3'b011 | 1 |

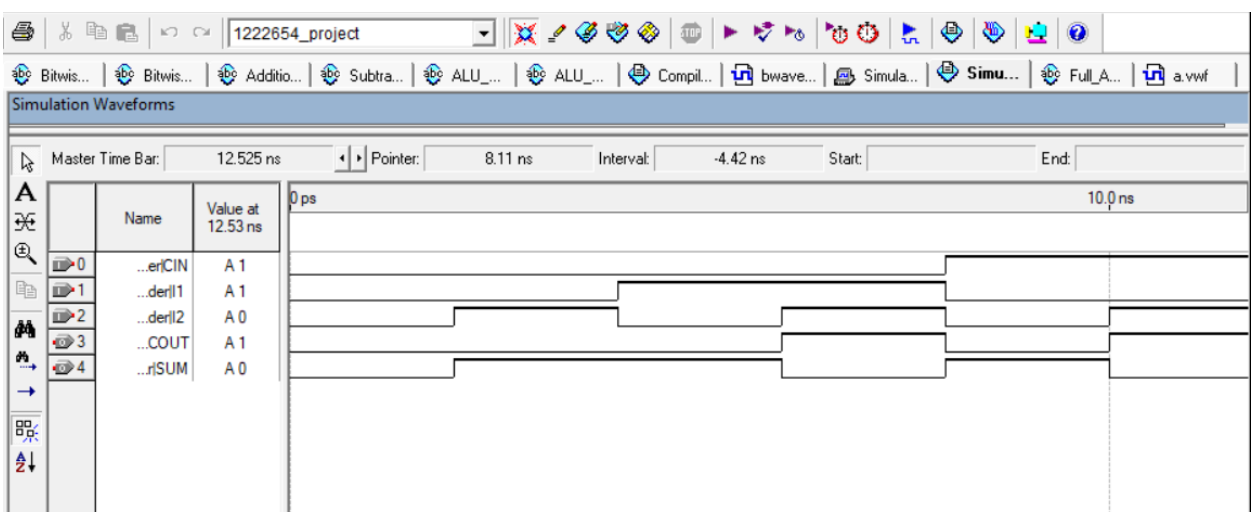
| 1 | 1 | 3'b011 | 1 |

Implement the modules using structure, dataflow, and behavioral modeling as below:

Note : before I implemented the addition and subtraction modules I created an extra module for the full adder to use in these modules

```
module Full_Adder(input I1,I2,CIN,output SUM,COUT);

wire w1, w2, w3;

and (w1, I1,I2);
xor (w2,I1,I2);
and (w3, w2, CIN);
xor (SUM, w2, CIN);
or (COUT, w1, w3);
endmodule
```

And here its waveform

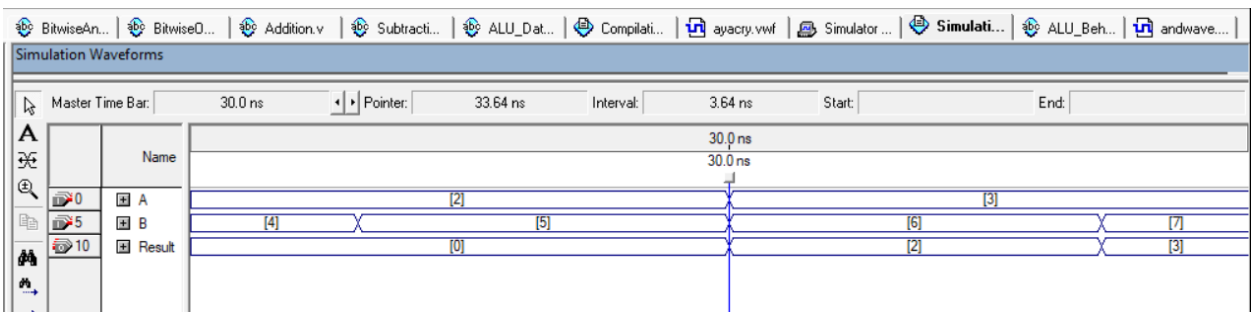


1. Utilize structural modeling for Adder, Subtractor, and logic gates.

Bitwise AND module :

```
module BitwiseAnd(input[3:0]A,B,output [3:0]Result);
//i created bitwise and by calling it 4 times using structural
// to operate the anding logic for each digit in A with each digit in B
and(Result[0],A[0],B[0]);
and(Result[1],A[1],B[1]);
and(Result[2],A[2],B[2]);
and(Result[3],A[3],B[3]);
endmodule
```

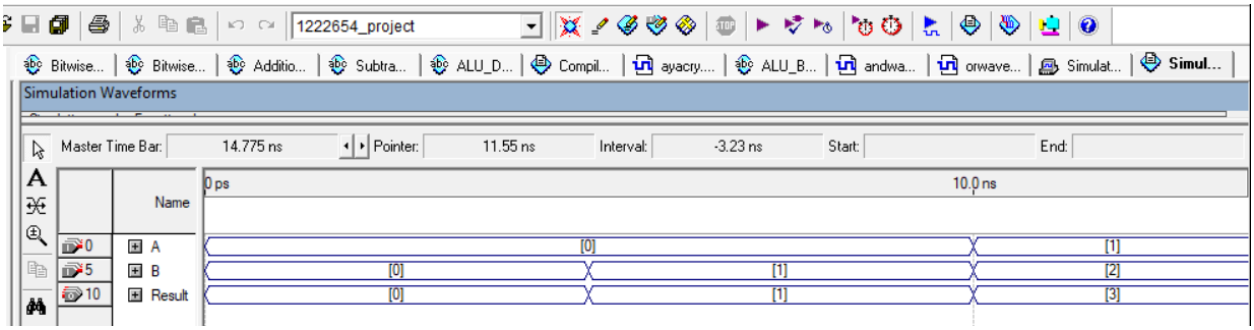
Bitwise AND waveform :



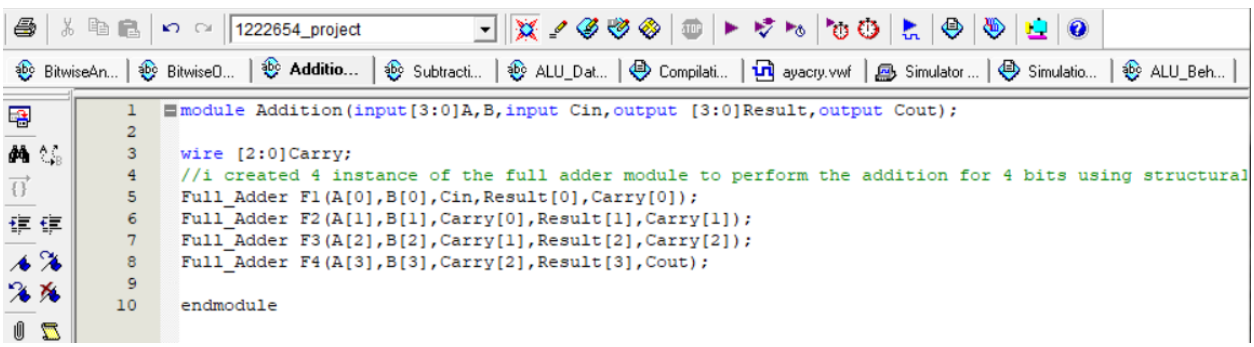
Bitwise OR module :

```
module BitwiseOR(input[3:0]A,B,output [3:0]Result);
//i created bitwise or by calling it 4 times using structural
// to operate the oring logic for each digit in A with each digit in B
or(Result[0],A[0],B[0]);
or(Result[1],A[1],B[1]);
or(Result[2],A[2],B[2]);
or(Result[3],A[3],B[3]);
endmodule
```

Bitwise OR waveform :

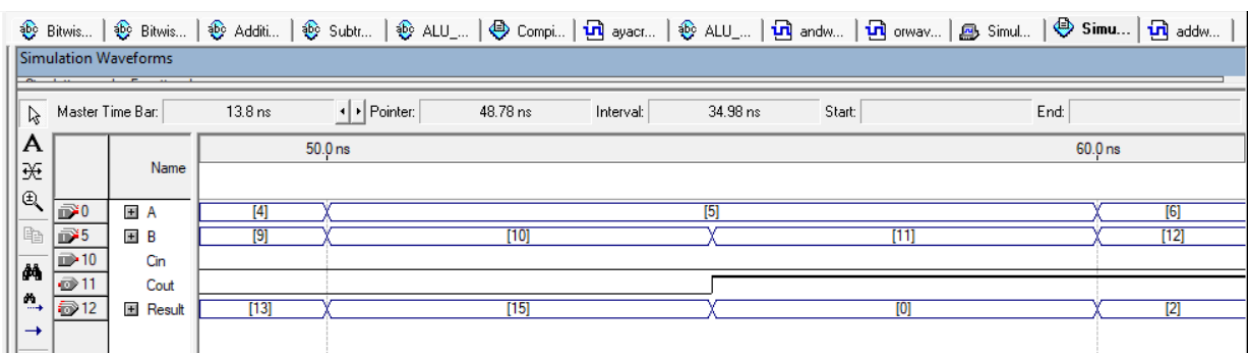


Addition module :



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

wire [2:0]Carry;
//i created 4 instance of the full adder module to perform the addition for 4 bits using structural
Full_Adder F1(A[0],B[0],Cin,Result[0],Carry[0]);
Full_Adder F2(A[1],B[1],Carry[0],Result[1],Carry[1]);
Full_Adder F3(A[2],B[2],Carry[1],Result[2],Carry[2]);
Full_Adder F4(A[3],B[3],Carry[2],Result[3],Cout);

endmodule
```

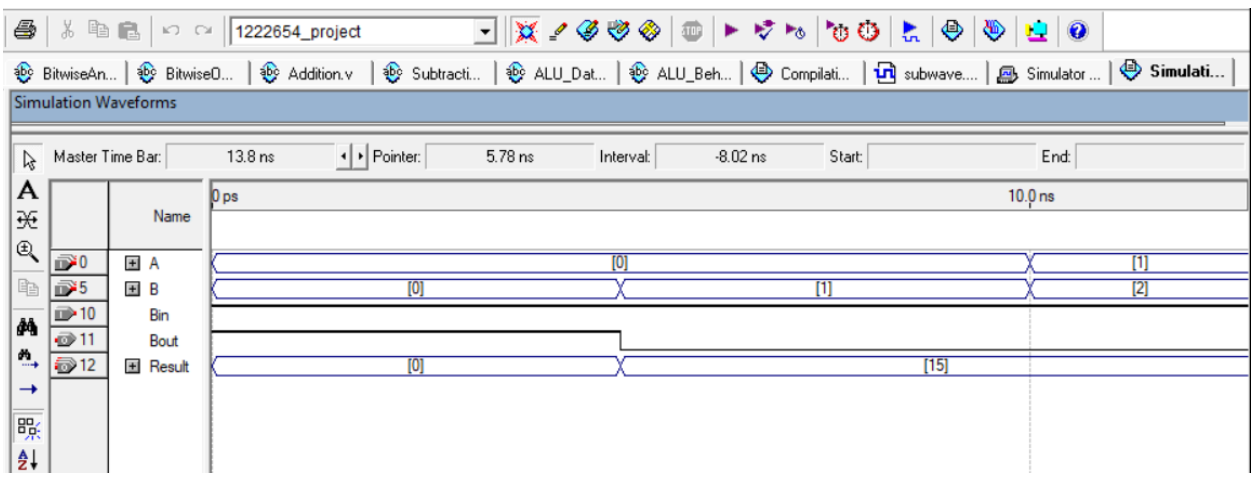
Addition waveform :



Subtraction module :

```
module Subtraction(input[3:0]A,B,input Bin,output [3:0]Result,output Bout);

wire [2:0]Borrow;
//i created 4 instance of the full adder module to perform the subtraction for 4 bits using structu
//and negated the B to do ones complement and in the waveform
//i give to Bin 1 so it do 2s complement
Full_Adder f1(A[0],~B[0],Bin,Result[0],Borrow[0]);
Full_Adder f2(A[1],~B[1],Borrow[0],Result[1],Borrow[1]);
Full_Adder f3(A[2],~B[2],Borrow[1],Result[2],Borrow[2]);
Full_Adder f4(A[3],~B[3],Borrow[2],Result[3],Bout);

endmodule
```

Subtraction waveform :

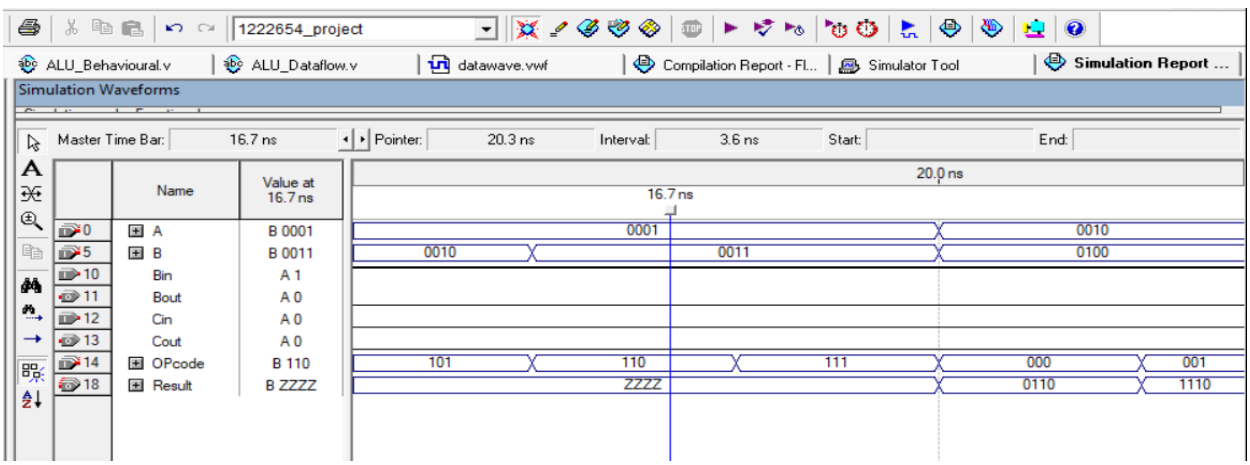


2. Implement dataflow modeling for connecting the modules.

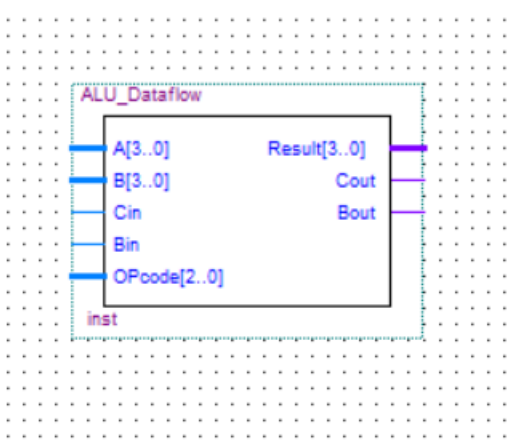
ALU_Dataflow.v

```
module ALU_Dataflow(input[3:0]A,B,input Cin,Bin,input[2:0]OPcode,output [3:0]Result,output Cout,Bou

wire [3:0]AND_OUT,OR_OUT,ADD_OUT,SUB_OUT;

//i called instance of the user gates that i previously created to take there output
Addition inst1(A,B,Cin,ADD_OUT,Cout);
Subtraction inst2(A,B,Bin,SUB_OUT,Bout);
BitwiseAnd inst3(A,B,AND_OUT);
BitwiseOR inst4(A,B,OR_OUT);

//i assigned the addittion , subtraction ,logic gates output to the result output of the alu
//by checking the opcode for each
assign Result=(OPcode==3'b000)?ADD_OUT:// if opcode is 3'b000 set Result to the output of the ADD c
(OPcode==3'b001)?SUB_OUT: // if opcode is 3'b001 set Result to the output of the SUB operation
(OPcode==3'b010)?AND_OUT:// if opcode is 3'b010 set Result to the output of the AND operation
(OPcode==3'b011)?OR_OUT:3'bz;// if opcode is 3'b011 set Result to the output of the OR operation
//the 3'bz is a dont care default case
endmodule
```

Dataflow ALU waveform :



Dataflow ALU Block Design:

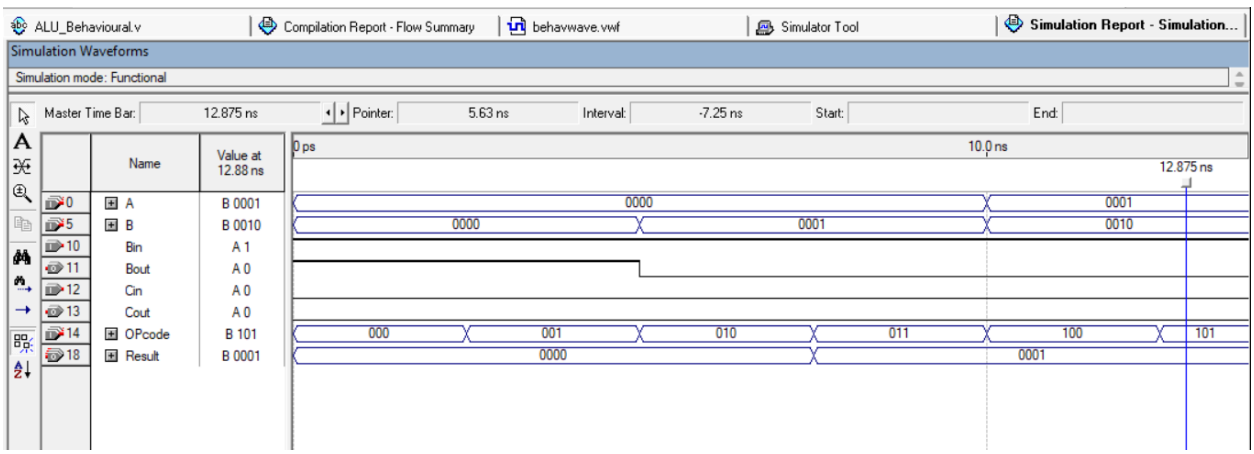


3. Utilize behavioral modeling for the top-level ALU module.

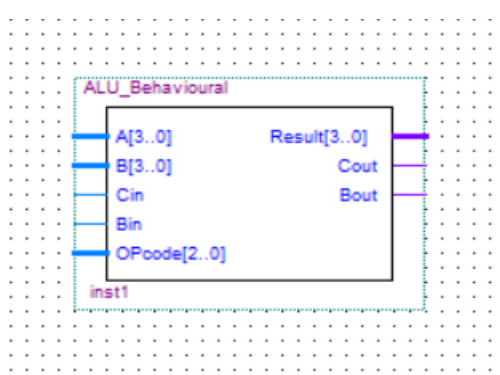
ALU_Behavioural.v* | ALU_Dataflow.v*

```
module ALU_Dataflow(input[3:0]A,B,input Cin,Bin,input[2:0]OPcode,output [3:0]Result,output Cout,Bout);

 wire [3:0]AND_OUT,OR_OUT,ADD_OUT,SUB_OUT;

 //i called instance of the user gates that i previously created to take there output
 Addition inst1(A,B,Cin,ADD_OUT,Cout);
 Subtraction inst2(A,B,Bin,SUB_OUT,Bout);
 BitwiseAnd inst3(A,B,AND_OUT);
 BitwiseOR inst4(A,B,OR_OUT);

 //i assigned the addittion , subtraction ,logic gates output to the result output of the alu
 //by checking the opcode for each
 assign Result=(OPcode==3'b000)?ADD_OUT:// if opcode is 3'b000 set Result to the output of the ADD operation
 (OPcode==3'b001)?SUB_OUT: // if opcode is 3'b001 set Result to the output of the SUB operation
 (OPcode==3'b010)?AND_OUT:// if opcode is 3'b010 set Result to the output of the AND operation
 (OPcode==3'b011)?OR_OUT:3'bz;// if opcode is 3'b011 set Result to the output of the OR operation
 //the 3'bz is a dont care default case
 endmodule
```

Behavioral ALU waveform :



Behavioral ALU Block Design :



Conclusion :

The Verilog modules, including Adder, Subtractor, AndGate, OrGate, and the top-level ALU, were developed using a combination of structural, dataflow, and behavioral modeling approaches.Testing and simulation were carried out to ensure the correct functionality of the ALU, covering various input combinations and control codes. The Verilog simulator was utilized to analyze the results of individual components and the overall ALU.