جامعة بيرزيت
BIRZEIT UNIVERSITY

Verilog HDL project

Leen alqazaqi
1220380
Section 8

—

ALU

—

overview

In this project we made a design of a simple arithmetic logic unit (ALU)-which is like the brain for the computer which chooses operations based on an input (OpCode here)- using HDL Verilog language . our ALU is capable of performing four basic arithmetic and logic operations which are : addition, subtraction, bitwise AND, and bitwise OR.(for 2 numbers of 4 bits)

THE PROCESS

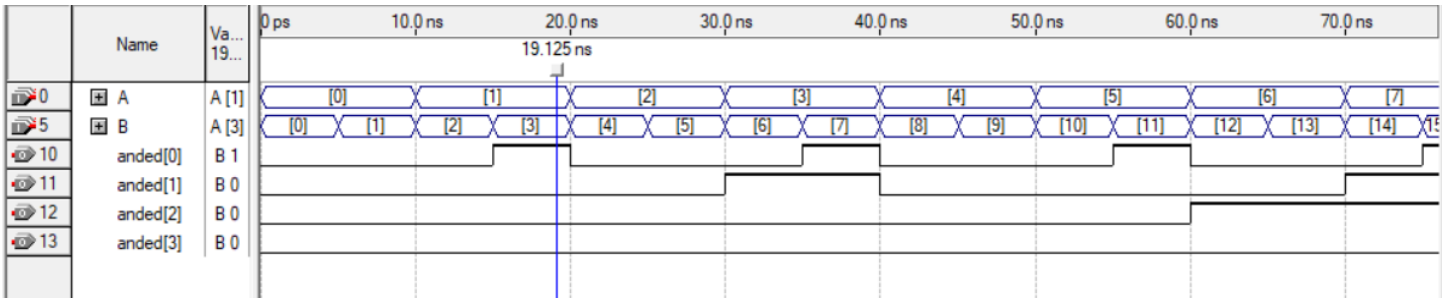
To do this (ALU four operations) we need to design every single process individually as a module (in structural modeling) and then connect them inside the ALU (in this project we used two ways ,dataflow and behavioral modeling)

First structural modeling :

Bitwise And gate :

And bitwise gate is a module that does the and operation between the corresponding bits in two numbers resulting a number with the same number of bits . here is the code and the waveform for it

```
// Bitwise AND compares corresponding bits of two numbers
module AndGate (anded,A,B) ;
//inputs and outputs
input [3:0] A,B;
output[3:0] anded;
// and between the corresponding bits of the two numbers
and first (anded[0],A[0],B[0]);
and second (anded[1],A[1],B[1]);
and third(anded[2],A[2],B[2]);
and fourth(anded[3],A[3],B[3]);
endmodule
```



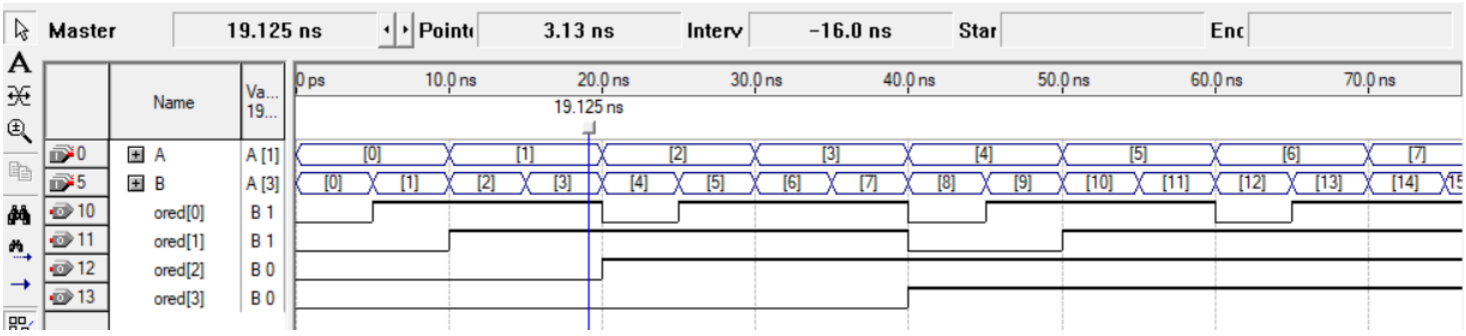
The picture above is from the simulation (wave form) and it contains many possible cases for example A = 0000 , B = 0001 then the result is 0000

Bitwise or gate :

or bitwise gate is a module that does the or operation between the corresponding bits in two numbers and the result will have the same number of bits (here 4). here is the code and the waveform for it

```
//Bitwise OR compares corresponding bits of two numbers
module OrGate (ored,A,B) ;
//inputs and outputs
input [3:0] A,B;
output[3:0] ored;
//OR between the corresponding bits
or first (ored[0],A[0],B[0]);
or second (ored[1],A[1],B[1]);
or third(ored[2],A[2],B[2]);
or fourth(ored[3],A[3],B[3]);
endmodule
```

Next is the wave form (includes all the possible cases)



Here one of the examples if A = 0001 and B = 0011 then the result will be 0011

Adder :

To design the adder we need three steps or two . three if we designed the half adder then the full from the half then the whole adder which is what I did
And two if we designed the full adder directly
Here is the codes for the half adder,full adder and the 4 bit Adder

Half adder : each half adder sums 2 bits

```
// a half adder that sums up two bits
module HA (sum,c,A,B);
output sum,c;
input A,B;
xor (sum,A,B);
and (c,A,B);
 endmodule
```

Full adder

To design a full adder we need two half adders and an or gate each full adder sums three bits
Code :

```
//ful adder made of two half adders ,it sums up three bits
module FA (sum,carryout,A,B,c);
//inputs and outputs and wires , inputs for a gate output from another
input A,B,c ;
output sum,carryout;
wire w1,w2,w3;
 //instantiate 2 half adders
HA num1(w1,w2,A,B);
HA num2(sum,w3,w1,c);
or (carryout,w2,w3);

endmodule
```

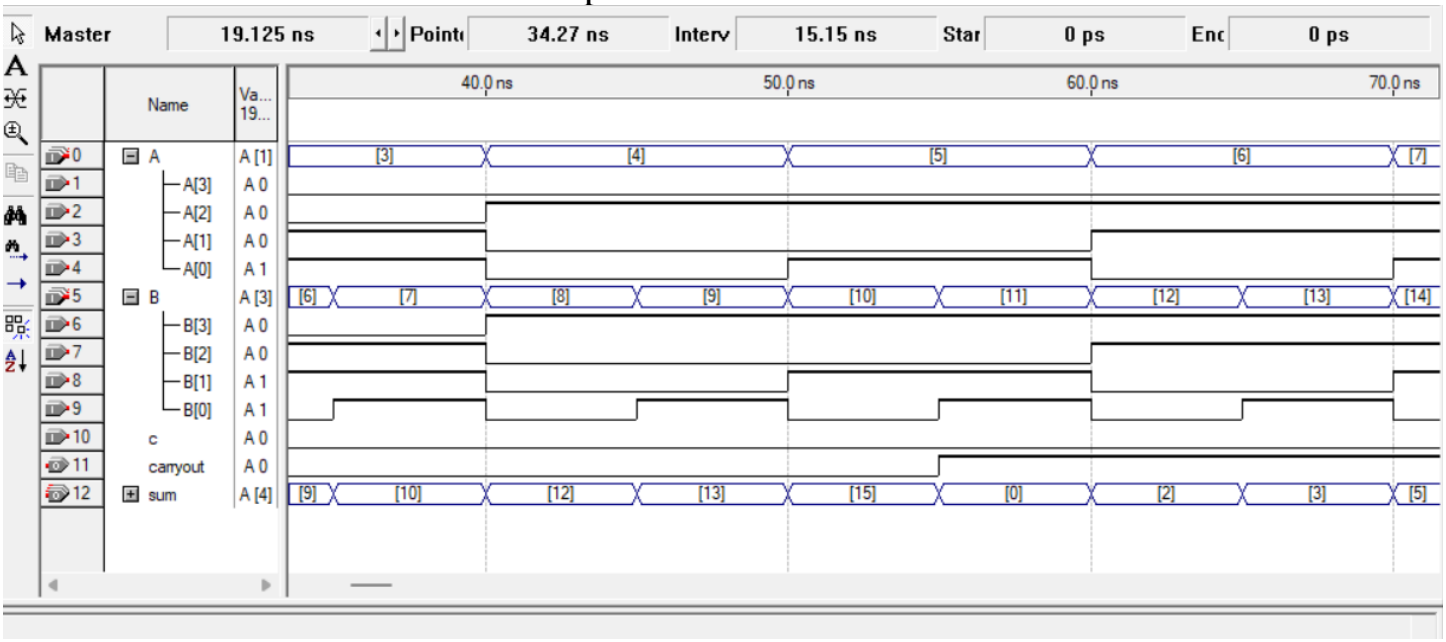
Adder :

To design an adder that sums 2 numbers each with 4 bits we will need 4 ful adders one for every corresponding bits in the two numbers

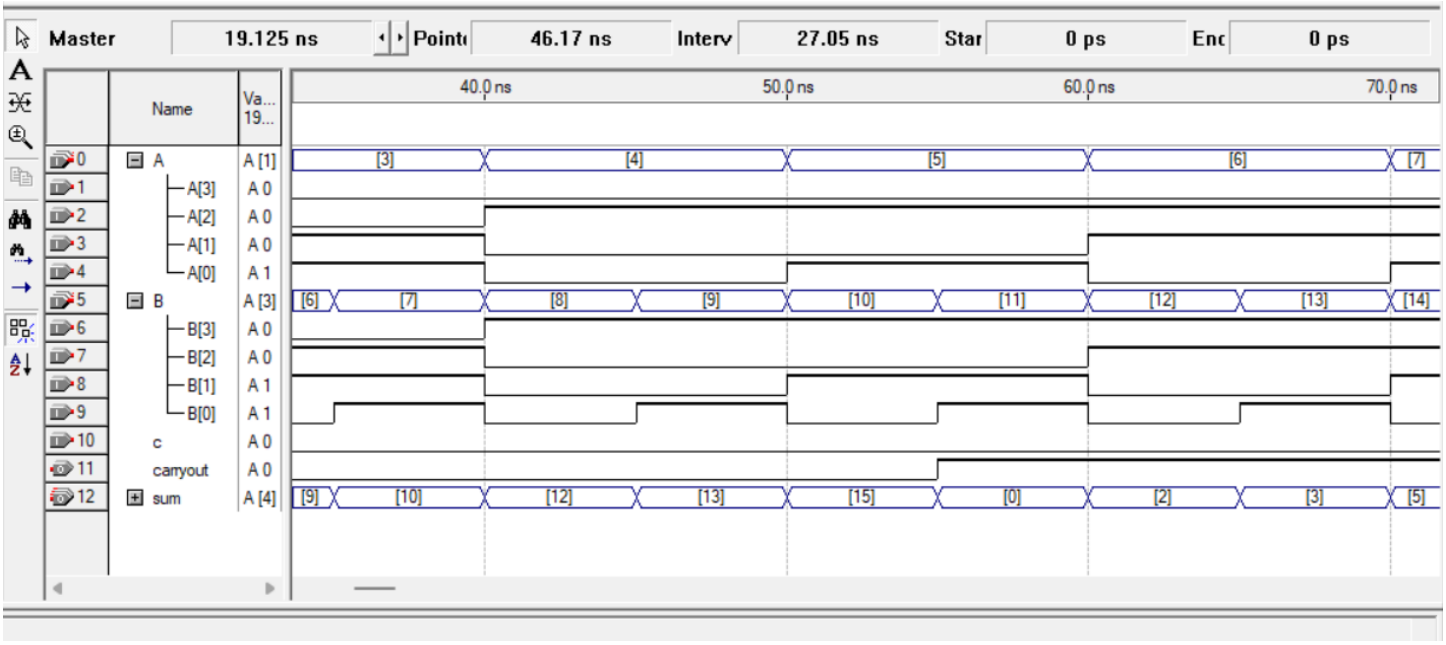
```
// adder for four bit number made of four full adders , sums two numbers each made of 4 bits
module Adder (sum,carryout,A,B,c);
//inputs and outputs
input c ;
input[3:0] A,B ;
output carryout;
output [3:0]sum;
wire c1,c2,c3;
 //instantiate four full adders
FA num1 (sum[0],c1,A[0],B[0],c);
FA num2 (sum[1],c2,A[1],B[1],c1);
FA num3 (sum[2],c3,A[2],B[2],c2);
FA num4 (sum[3],carryout,A[3],B[3],c3);
endmodule
```

note that if the numbers added cannot be more than 15 other wise the result (sum) will be out of the register's range and the result provided will be wrong

Here are the waveForms for all the possible cases for Adder



Note that after 10 + 5 (15) the answer became wrong



Subtractor :

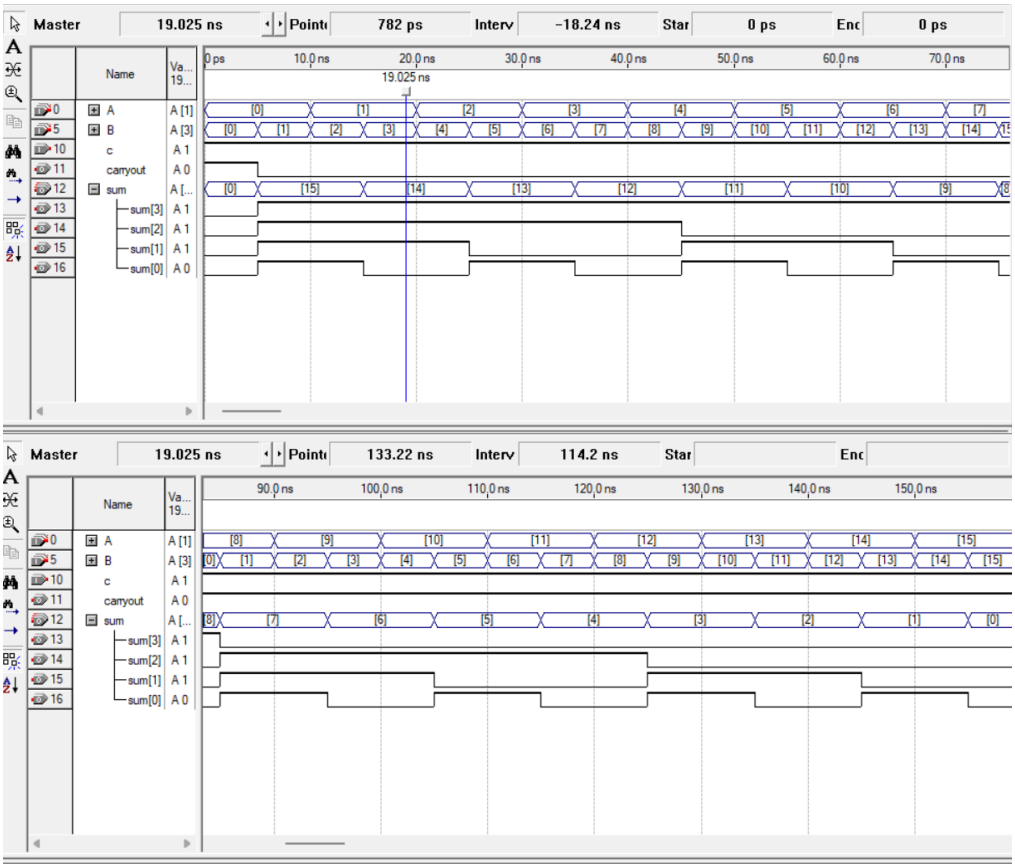
Designing 4 bit subtractor is similar to design the adder, the only difference is that we need B 2's complement so we add B inverted with not with a and with an initial carry 1 (in summation its zero)

Here is the code and the wave form for it

```
// subtractor for four bit number made of four full adders its similar to the adder but the B is negated and the carry should be 1
//subtracts two numbers each has 4 bits
module Subtractor (sum,carryout,A,B,c);
//inputs and outputs
input c  ;
input[3:0] A,B ;
output carryout;
output [3:0]sum;
wire c1,c2,c3;
//here i will be using not gate instead of simply saying ~B to keep it structural

wire [3:0] notB;
not (notB[0],B[0]);
not (notB[1],B[1]);
not (notB[2],B[2]);
not (notB[3],B[3]);
//instantiate four full adders
FA num1 (sum[0],c1,A[0],notB[0],c);
FA num2 (sum[1],c2,A[1],notB[1],c1);
FA num3 (sum[2],c3,A[2],notB[2],c2);
FA num4 (sum[3],carryout,A[3],notB[3],c3);
endmodule
```

The wave form after simulation : note that it will not work probably unless A is bigger than B and the answer is not out of the range we can fix the first problem by a simple if statement (if A>B) and if it the answer is true then we should put a minus and take the 2's complement* but that will not be structural as asked in the instructions*



ALU (top level module) :

ALU is basically modeling multi functions in one module and choose one based on an input named operation code or OpCode (similar to the MUX)
I made it in two modeling ways dataflow that focuses on how the data flows from inputs to outputs and behavioral that just explains the behavior of the module
Here is the code for data flow :

```
//this module will select one of the four operations we made depending on OpCode
   module dataflowALU (Result,carrysum,carrysub,A,B,c,OpCode);
//the process of selecting the operation will work perfectly as 4 to 1 MUX since the MSB is always zero
//inputs and outputs
input [3:0] A,B;
input c ;
input [2:0] OpCode;
output carrysum,carrysub;
output [3:0] Result ;
//every wire represent the output of one of the four operations
wire [3:0] wireand,wireor,wireadd,wiresub ;
//calling or instantiate the four modules we made
AndGate (wireand,A,B);
OrGate (wireor,A,B);
Adder (wireadd,carrysum,A,B,c);
Subtractor (wiresub,carrysub,A,B,c);
//result using data flow modeling
assign Result = (OpCode==3'b000)? wireadd :
(OpCode==3'b001)? wiresub :
(OpCode==3'b010)? wireand :
(OpCode==3'b011)? wireor :4'bz ; /*the conditional operator does not include all possible cases so we say that the
other cases are z which means its lika an open circuit (no signal) */

endmodule
```

And here is the code for the behavioral

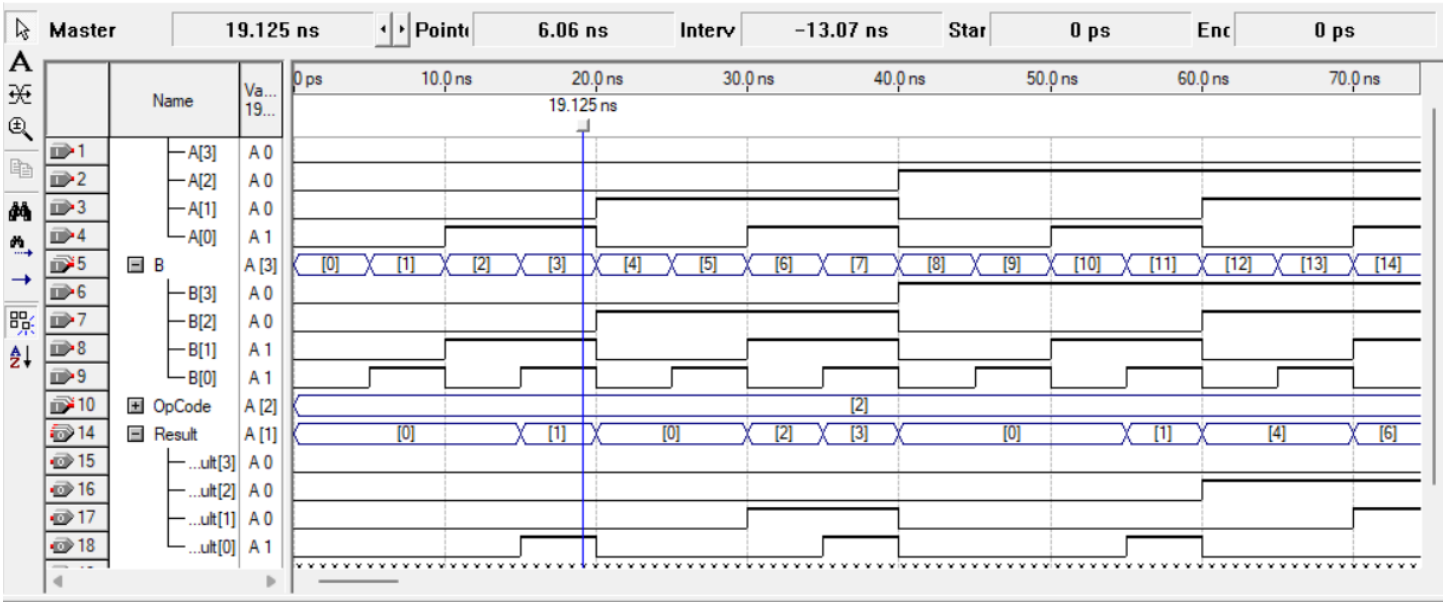
```
//this module will select one of the four operations we made depending on OpCode
//it gives the same result as the data flow one but here in behavioral modeling
module behavioralALU (Result,carrysum,carrysub,A,B,c,OpCode);
//inputs and outputs
input c ;
input [3:0] A,B;
input [2:0] OpCode;
output reg [3:0] Result ;
output  carrysum,carrysub;
wire [3:0] wireand,wireor,wireadd,wiresub ;
//instantiate the four operations
AndGate (wireand,A,B);
OrGate (wireor,A,B);
Adder (wireadd,carrysum,A,B,c);
Subtractor (wiresub,carrysub,A,B,c);
//sensitivity list = @* which means all since this design is combinational
// default case is for any other cases and in this design we do not have more than 4 operations so its z high impedance
always @(*) begin
case(OpCode)
3'b000:Result = wireadd;
3'b001:Result = wiresub;
3'b010: Result = wireand;
3'b011: Result = wireor;
default:Result = 4'bz;
endcase
end
endmodule
```

both of them will give the same results it is just different ways to model the code

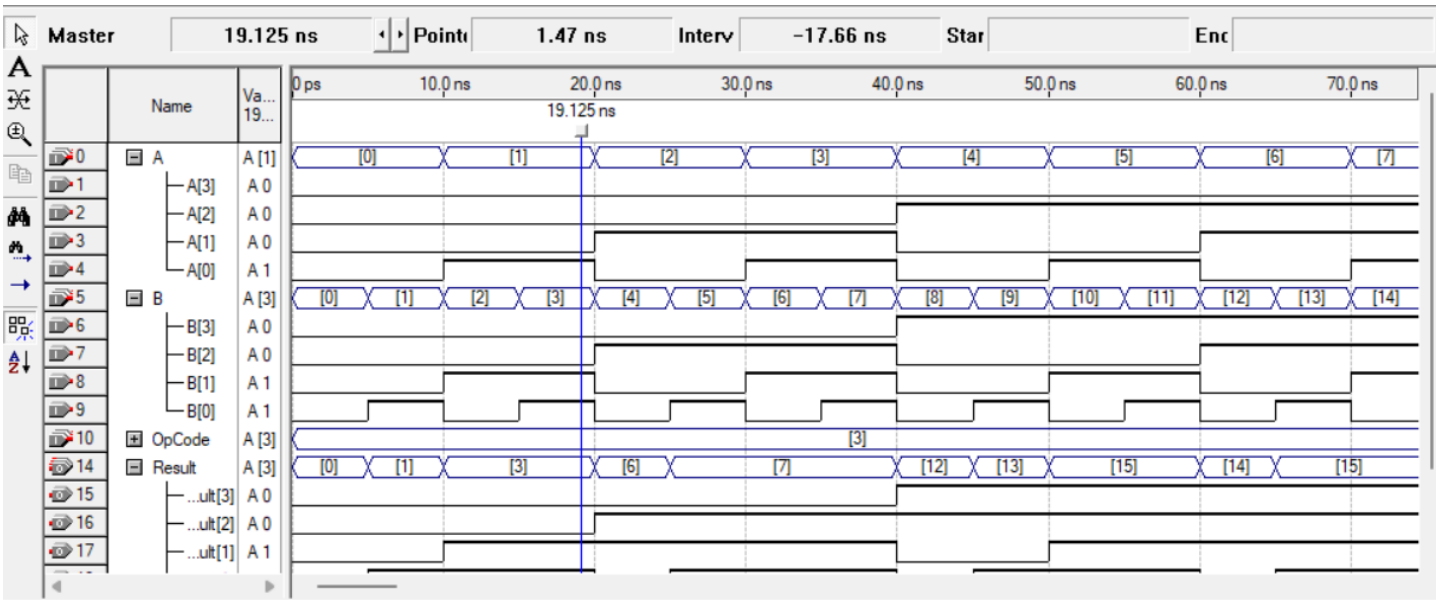
Here is some wave forms for the data flow one trying different OpCodes :

Operation name and its code

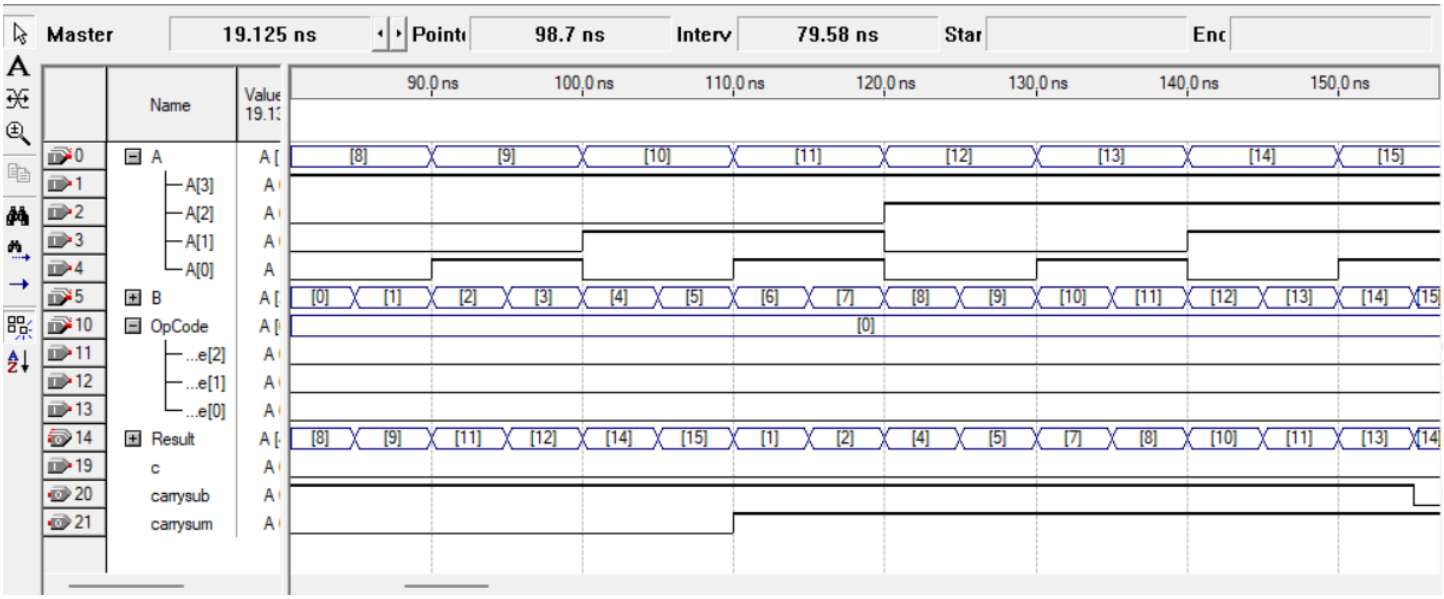
And 010



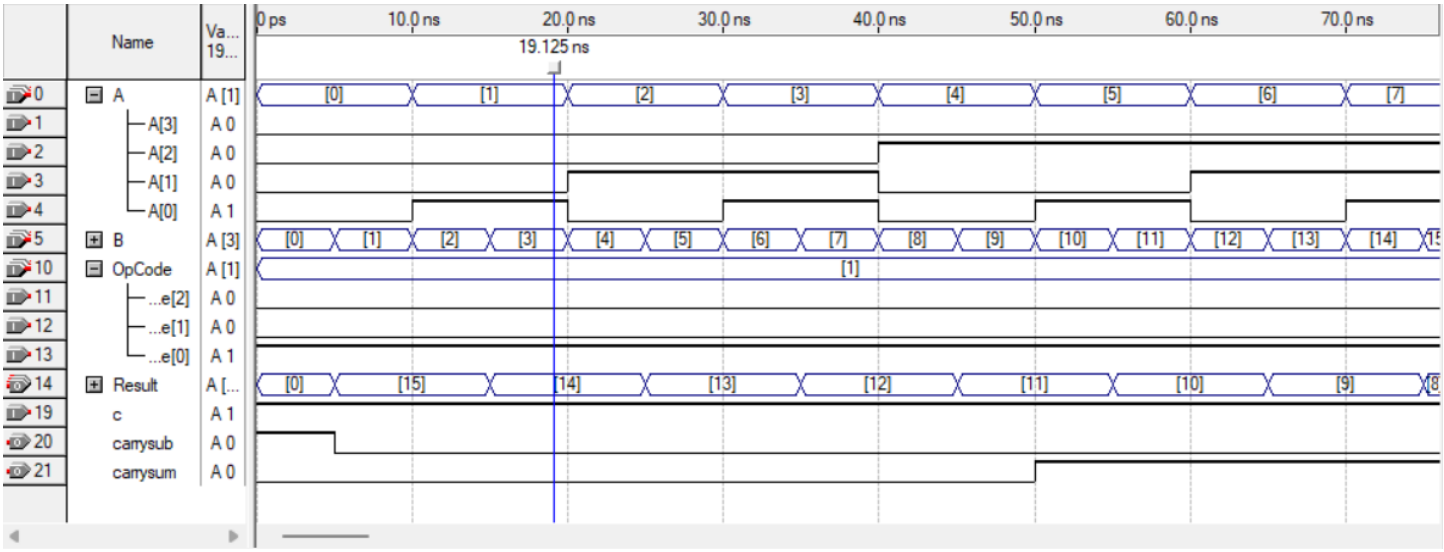
Or 011



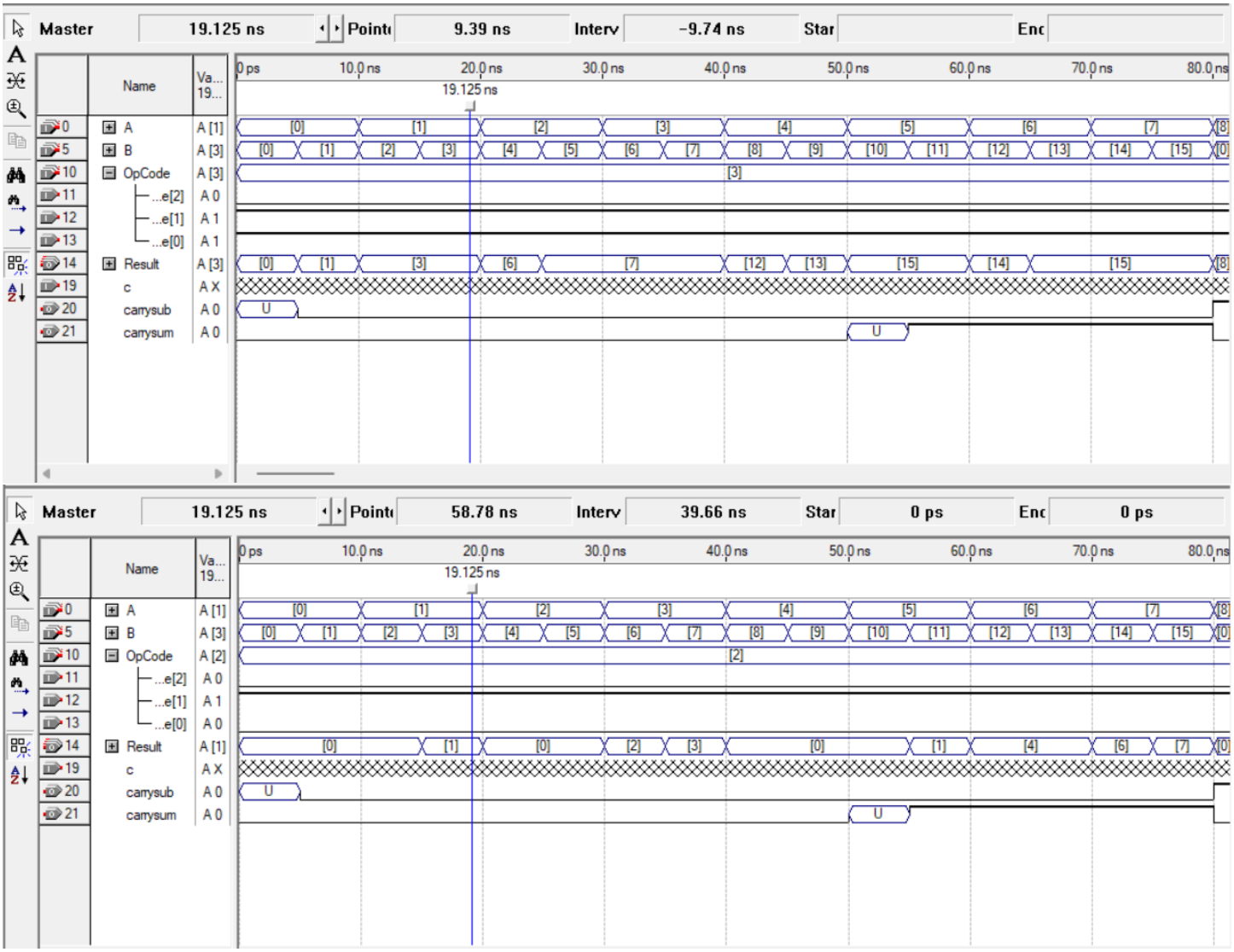
Adder 000



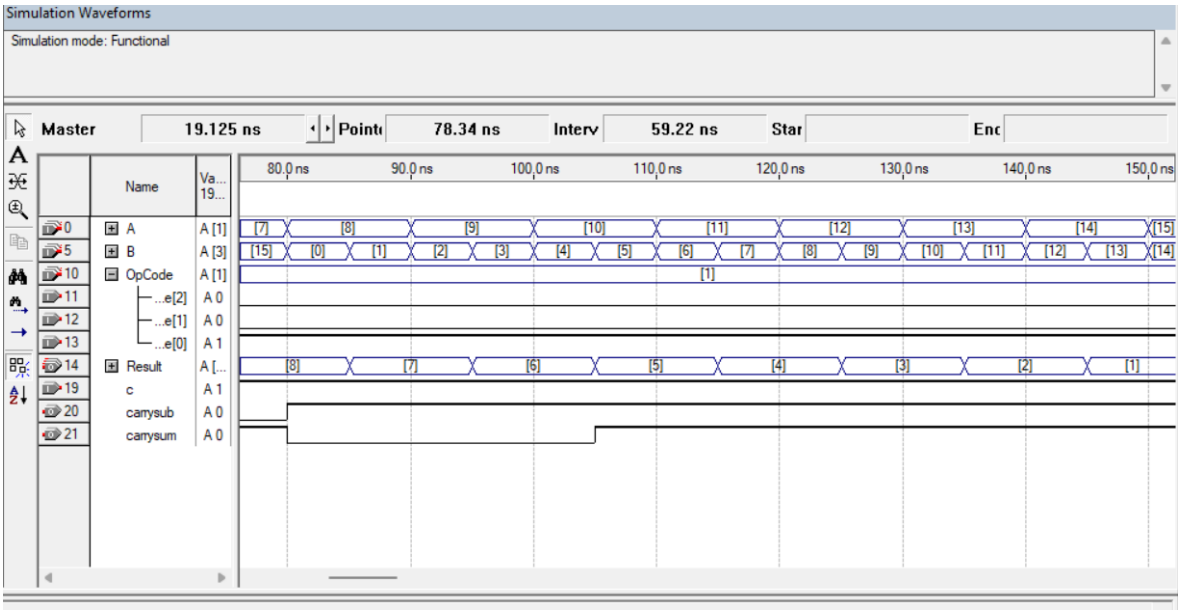
Subtractor 001



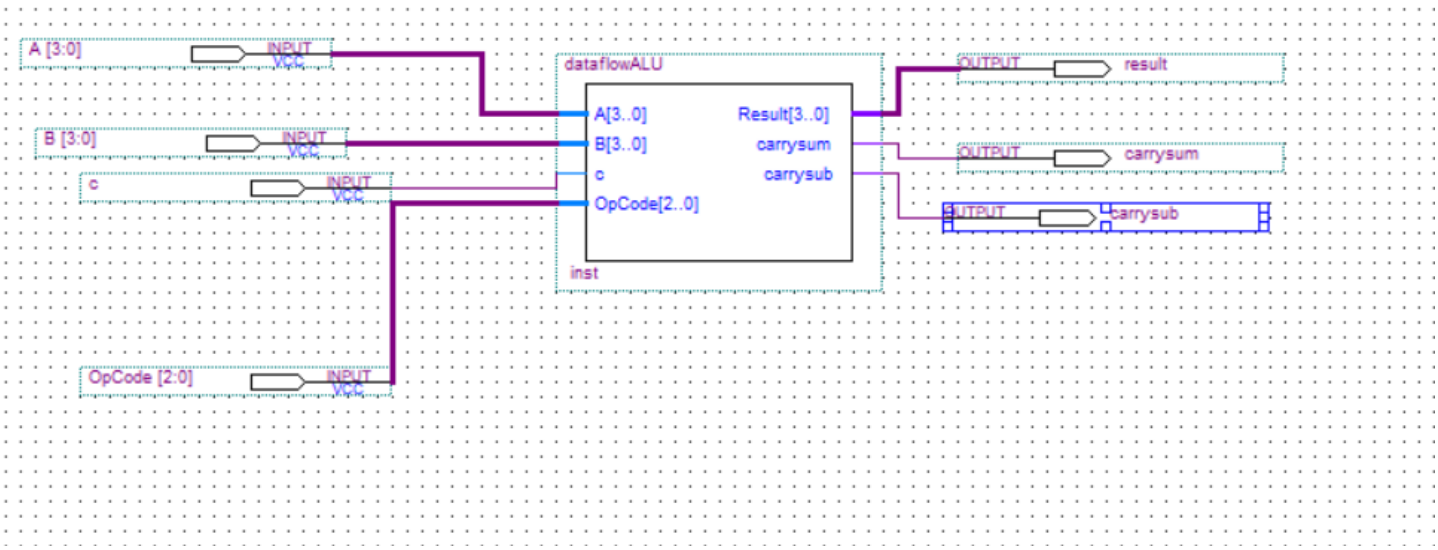
And now the wave form for the behavioral :



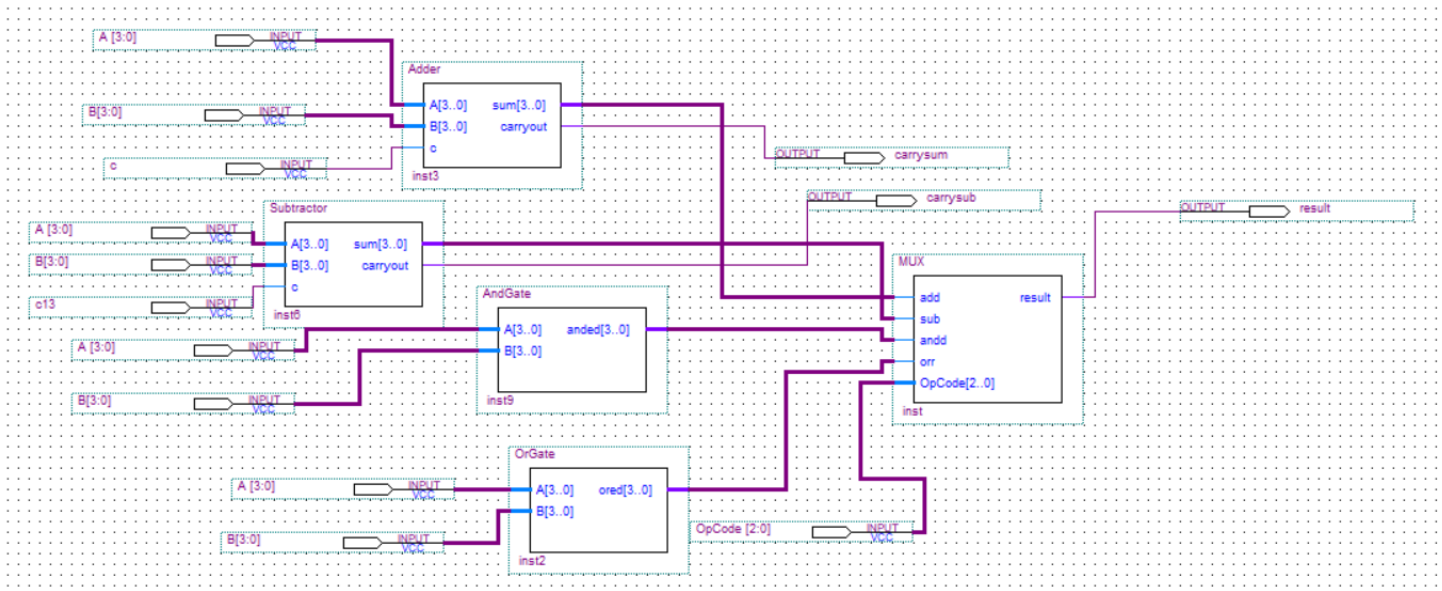
c in the above pictures is set to be don't care because there is no carry in the AndGate and OrGate operations



In this section I will add the design from the schematic :



Here is the design of our ALU as a one piece next picture will show the structure inside



In this design it shows the structure from inside as I said before the ALU in this case will work like a mux
Have inputs and chooses one to go to the output
Here is the code for the MUX I added only to have the symbol of the MUX and use it

```
//this module was made only to help in the schematic file / symbol design since the idea works like a 4 to 1 MUX
module MUX
//the outputs and inputs
  (output reg result,input add,sub,andd,orr,input [2:0] OpCode);
always @* begin
if (OpCode ==3'b000) result = add ;
else if (OpCode ==3'b001) result = sub ;
else if (OpCode ==3'b010) result = andd ;
else if (OpCode ==3'b011) result = orr ;
else result = 4'bz;
end
endmodule
```

In conclusion we will have the same result in the two ways of modeling each has its benefits for example the behavioral is the easiest but we cannot download it directly into the hard ware pieces