Remi de Ferrieres
X664611
Yann Abou Jaoude
X663581

Mini Project report

The purpose of this project was to make us use several aspects and different components all into one project. In other words, make a multifunctional project. We had to create something using what we have learned in the previous labs.
With this knowledge we can create all sorts of project.

For this project we decided to work on the assembly of an ALU, which enables you to choose an operation to apply to two 4bit number entered as inputs.

For this project we decided to go from creating the schematics of every components and to assemble them together to finally having a practical ALU.

Verilog code:

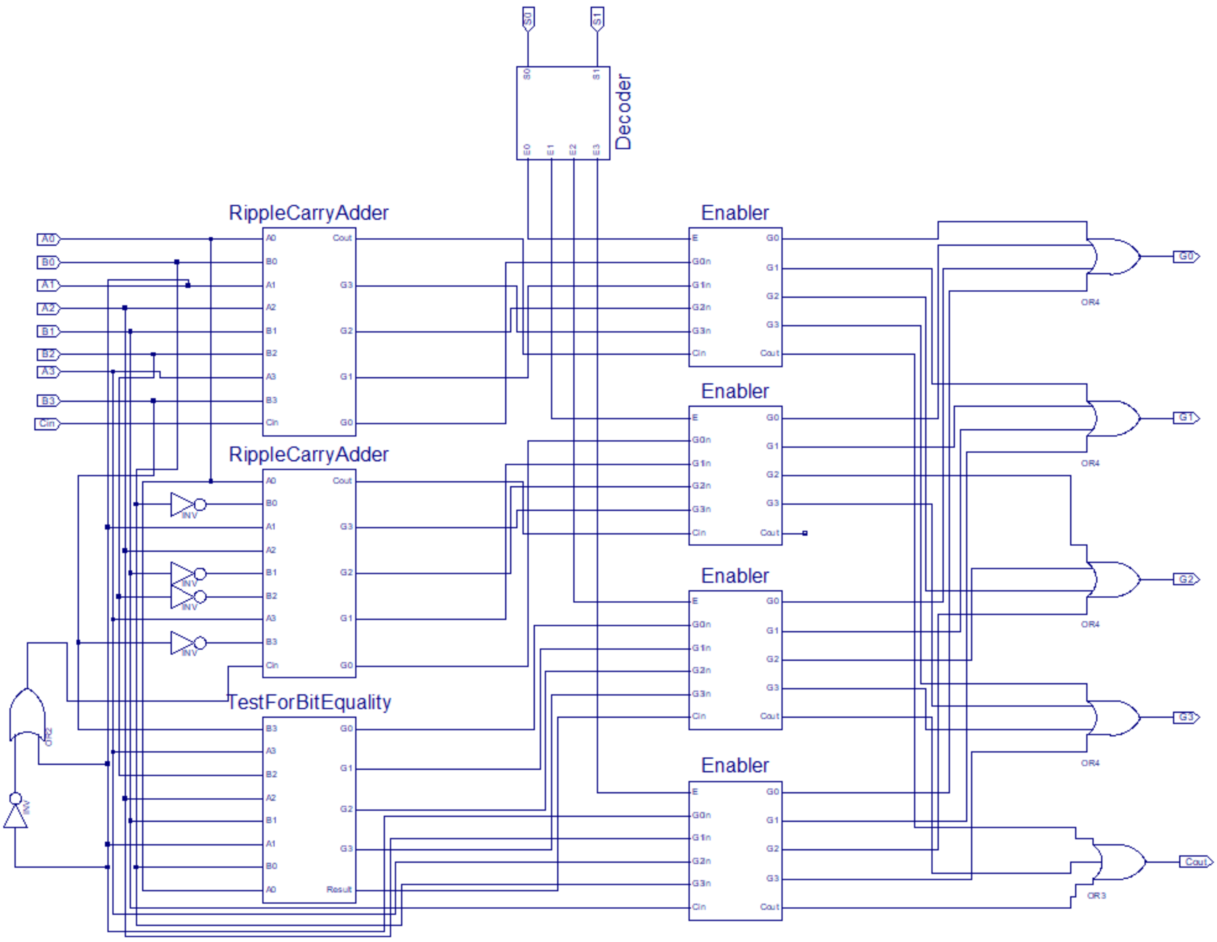
```
module FullAdder_MUSER_ALU(A,B,Cin,Cout,S);

   input A;
   input B;
   input Cin;
   output Cout;
   output S;

   wire XLXN_1;
   wire XLXN_4;
   wire XLXN_9;

   XOR2  XLXI_1 (.I0(B),.I1(A),.O(XLXN_1));
   AND2  XLXI_2 (.I0(B),.I1(A),.O(XLXN_9));
   AND2  XLXI_3 (.I0(Cin),.I1(XLXN_1),.O(XLXN_4));
   XOR2  XLXI_4 (.I0(Cin),.I1(XLXN_1),.O(S));
   OR2  XLXI_6 (.I0(XLXN_9),.I1(XLXN_4),.O(Cout));
endmodule


module RippleCarryAdder_MUSER_ALU(A0,A1,A2,A3,B0,B1,B2,B3,Cin,Cout,G0,G1,G2,G3);

    input A0;
    input A1;
    input A2;
    input A3;
    input B0;
    input B1;
    input B2;
    input B3;
    input Cin;
   output Cout;
   output G0;
   output G1;
   output G2;
   output G3;

   wire XLXN_1;
   wire XLXN_2;
   wire XLXN_3;

   FullAdder_MUSER_ALU  XLXI_5 (.A(A0),.B(B0),.Cin(Cin),.Cout(XLXN_1),.S(G0));
   FullAdder_MUSER_ALU  XLXI_6 (.A(A1),.B(B1),.Cin(XLXN_1),.Cout(XLXN_2),.S(G1));
   FullAdder_MUSER_ALU  XLXI_7 (.A(A2),.B(B2),.Cin(XLXN_2),.Cout(XLXN_3),.S(G2));
   FullAdder_MUSER_ALU  XLXI_8 (.A(A3),.B(B3),.Cin(XLXN_3),.Cout(Cout),.S(G3));
endmodule
```

```
module Enabler_MUSER_ALU(Cin,E,G0in,G1in,G2in,G3in,Cout,G0,G1,G2,G3);

    input Cin;
    input E;
    input G0in;
    input G1in;
    input G2in;
    input G3in;
   output Cout;
   output G0;
   output G1;
   output G2;
   output G3;


   AND2  XLXI_1 (.I0(E),.I1(G0in),.O(G0));
   AND2  XLXI_2 (.I0(E),.I1(G1in),.O(G1));
   AND2  XLXI_3 (.I0(E),.I1(G2in),.O(G2));
   AND2  XLXI_4 (.I0(E),.I1(G3in),.O(G3));
   AND2  XLXI_5 (.I0(E),.I1(Cin),.O(Cout));
endmodule


module TestForBitEquality_MUSER_ALU(A0,A1,A2,A3,B0,B1,B2,B3,G0,G1,G2,G3,Result);

    input A0;
    input A1;
    input A2;
    input A3;
    input B0;
    input B1;
    input B2;
    input B3;
   output G0;
   output G1;
   output G2;
   output G3;
   output Result;

   wire G0_DUMMY;
   wire G1_DUMMY;
   wire G2_DUMMY;
   wire G3_DUMMY;

   assign G0 = G0_DUMMY;
   assign G1 = G1_DUMMY;
   assign G2 = G2_DUMMY;
   assign G3 = G3_DUMMY;
   AND4  XLXI_5 (.I0(G3_DUMMY),.I1(G2_DUMMY),.I2(G1_DUMMY),.I3(G0_DUMMY),.O(Result));
   XNOR2  XLXI_7 (.I0(B0),.I1(A0),.O(G0_DUMMY));
   XNOR2  XLXI_8 (.I0(B1),.I1(A1),.O(G1_DUMMY));
   XNOR2  XLXI_9 (.I0(B2),.I1(A2).O(G2_DUMMY));
   XNOR2  XLXI_10 (.I0(B3),.I1(A3),.O(G3_DUMMY));
endmodule

module Decoder_MUSER_ALU(S0,S1,E0,E1,E2,E3);

    input S0;
    input S1;
   output E0;
   output E1;
   output E2;
   output E3;

   wire XLXN_6;
   wire XLXN_7;

   INV  XLXI_1 (.I(S1),.O(XLXN_6));
   INV  XLXI_2 (.I(S0),.O(XLXN_7));
   AND2  XLXI_3 (.I0(S1), .I1(S0),.O(E3));
   AND2  XLXI_4 (.I0(S1),.I1(XLXN_7),.O(E2));
   AND2  XLXI_5 (.I0(XLXN_6),.I1(S0),.O(E1));
   AND2  XLXI_6 (.I0(XLXN_6),.I1(XLXN_7),.O(E0));
endmodule
```

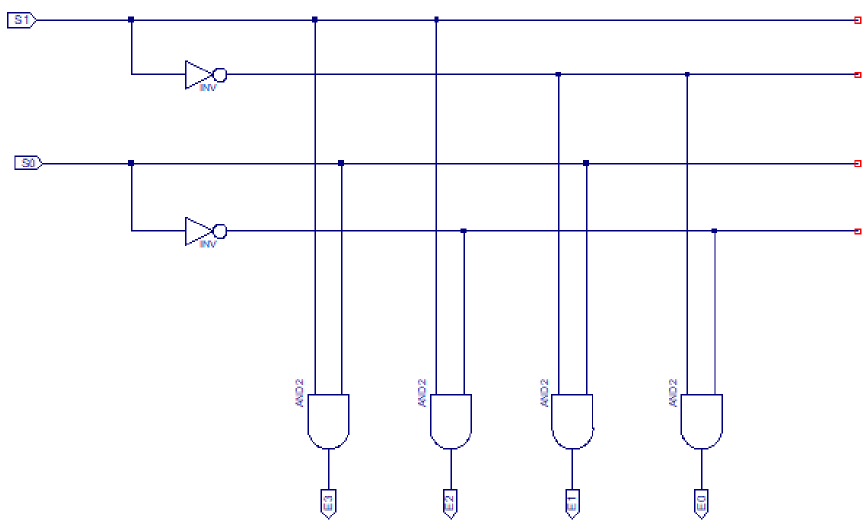
```
module ALU(A0,A1,A2,A3,B0,B1,B2,B3,Cin,S0,S1,Cout,G0,G1,G2,G3);
    input A0,A1,A2,A3,B0,B1,B2,B3,Cin,S0,S1;
   output Cout;
   output G0;
   output G1;
   output G2;
   output G3;
   wire One;
   wire XLXN_4;
   wire XLXN_5;
   wire XLXN_6;
   wire XLXN_7;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire XLXN_77;
   wire XLXN_78;
   wire XLXN_80;
   wire XLXN_81;
   wire XLXN_82;
   wire XLXN_83;
   wire XLXN_84;
   wire XLXN_85;
   wire XLXN_86;
   wire XLXN_90;
   wire XLXN_92;
   wire XLXN_93;
   wire XLXN_94;
   wire XLXN_95;
   wire XLXN_99;
   wire XLXN_100;
   wire XLXN_101;
   wire XLXN_102;
   wire XLXN_103;
   wire XLXN_105;
   wire XLXN_106;
   wire XLXN_107;
   wire XLXN_108;
   wire XLXN_109;
   wire XLXN_110;
   wire XLXN_111;
   wire XLXN_112;
   wire XLXN_113;
   wire XLXN_114;
   wire XLXN_115;
   wire XLXN_116;
   wire XLXN_126;
   wire XLXN_128;
   wire XLXN_129;

   Decoder_MUSER_ALU  XLXI_1 (.S0(S0),.S1(S1),.E0(XLXN_72),.E1(XLXN_73),.E2(XLXN_74),.E3(XLXN_75));
   Enabler_MUSER_ALU  XLXI_2 (.Cin(XLXN_76),.E(XLXN_72), .G0in(XLXN_81),.G1in(XLXN_80),.G2in(XLXN_78),.G3in(XLXN_77),.Cout(XLXN_116),.G0(XLXN_99),.G1(XLXN_103),.G2(XLXN_111),.G3(XLXN_112));
   Enabler_MUSER_ALU  XLXI_3 (.Cin(XLXN_82),.E(XLXN_73),.G0in(XLXN_86),.G1in(XLXN_85),.G2in(XLXN_84),.G3in(XLXN_83),.Cout(),.G0(XLXN_100),.G1(XLXN_105),.G2(XLXN_108),.G3(XLXN_113));
   Enabler_MUSER_ALU  XLXI_4 (.Cin(XLXN_95),.E(XLXN_74),.G0in(XLXN_90),.G1in(XLXN_92),.G2in(XLXN_93),.G3in(XLXN_94),.Cout(XLXN_129),.G0(XLXN_101),.G1(XLXN_106),.G2(XLXN_109),.G3(XLXN_114));
   Enabler_MUSER_ALU  XLXI_5 (.Cin(B1),.E(XLXN_75),.G0in(A1),.G1in(A2),.G2in(A3),.G3in(B0),.Cout(XLXN_128),.G0(XLXN_102),.G1(XLXN_107),.G2(XLXN_110),.G3(XLXN_115));
   RippleCarryAdder_MUSER_ALU  XLXI_8 (.A0(A0),.A1(A1),.A2(A2),.A3(A3), .B0(B0),.B1(B1),.B2(B2),.B3(B3),.Cin(Cin),.Cout(XLXN_76),.G0(XLXN_81),.G1(XLXN_80),.G2(XLXN_78),.G3(XLXN_77));
   RippleCarryAdder_MUSER_ALU  XLXI_15 (.A0(A0),.A1(A1),.A2(A2),.A3(A3),.B0(XLXN_4),.B1(XLXN_5),.B2(XLXN_6),.B3(XLXN_7),.Cin(One),.Cout(XLXN_82),.G0(XLXN_86),.G1(XLXN_85),.G2(XLXN_84),.G3(XLXN_83));
   INV  XLXI_16 (.I(B0),.O(XLXN_4));
   INV  XLXI_17 (.I(B1),.O(XLXN_5));
   INV  XLXI_18 (.I(B2),.O(XLXN_6));
   INV  XLXI_20 (.I(B3),.O(XLXN_7));
   TestForBitEquality_MUSER_ALU  XLXI_21 (.A0(A0),.A1(A1),.A2(A2),.A3(A3),.B0(B0),.B1(B1),.B2(B2),.B3(B3),.G0(XLXN_90),.G1(XLXN_92),.G2(XLXN_93),.G3(XLXN_94),.Result(XLXN_95));
   OR4  XLXI_53 (.I0(XLXN_102),.I1(XLXN_101),.I2(XLXN_100),.I3(XLXN_99),.O(G0));
   OR4  XLXI_54 (.I0(XLXN_107),.I1(XLXN_106),.I2(XLXN_105),.I3(XLXN_103),.O(G1));
   OR4  XLXI_55 (.I0(XLXN_110),.I1(XLXN_111),.I2(XLXN_109),.I3(XLXN_108),.O(G2));
   OR4  XLXI_56 (.I0(XLXN_115),.I1(XLXN_114),.I2(XLXN_113),.I3(XLXN_112),.O(G3));
   OR2  XLXI_60 (.I0(A1),.I1(XLXN_126),.O(One));
   INV  XLXI_61 (.I(A1),.O(XLXN_126));
   OR3  XLXI_62 (.I0(XLXN_128),.I1(XLXN_129),.I2(XLXN_116),.O(Cout));
endmodule
```

Schematics :

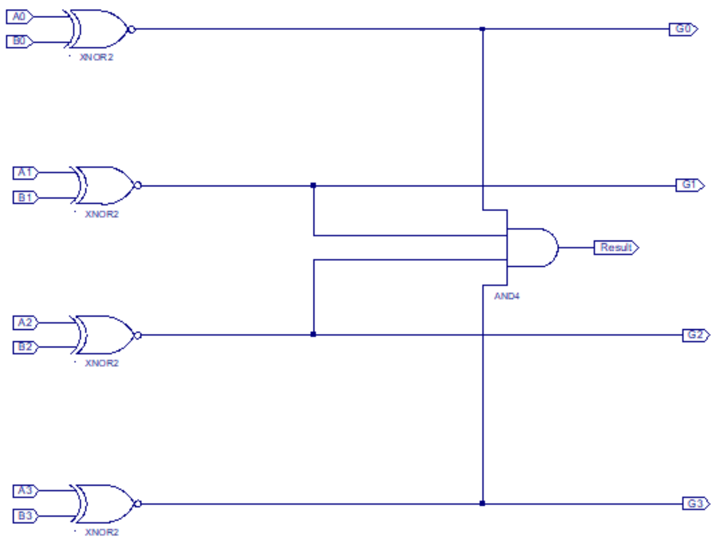
General Schematic :



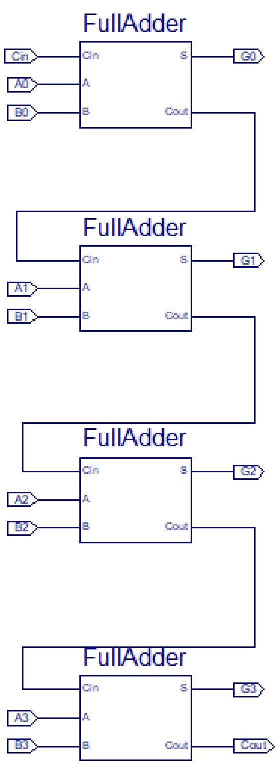
Decoder :



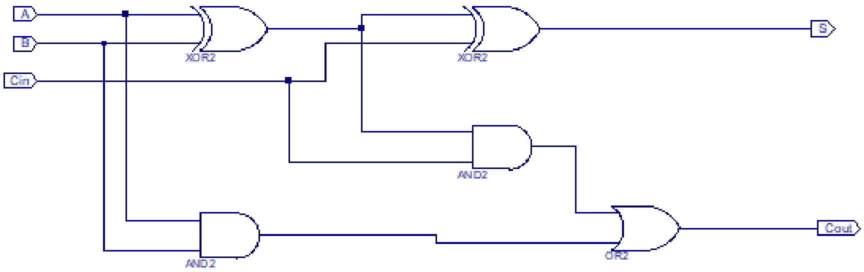
Test for bit equality :



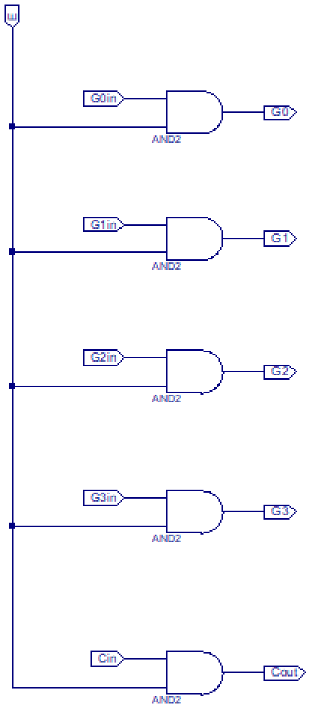
Ripple carry adder :



Full adder :



Enabler :



Detailed description

We choose to do the ALU.
An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers.
The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed.
We have 3 operands : A and B are four bits each, and Cin is one bit.
Our ALU can do four different operations, so we need a two bit code to know which operation to do. These input are s0 and s1.
The operations are :
00 Unsigned add
01 Unsigned subtraction
10 Test for bit equality
11 Divide by Two

We made our project by realizing 6 schematics.
The main schematic is the ALU itself.
It contains: -2 ripple carry adder.
-A decoder
-A module to compare bits
- 4 enabler

Each ripple carry adder is made using 4 full adder .
A full adder is a logical circuit that performs an addition operation on three one-bit binary numbers.
The full adder produces a sum of the three inputs and carry value.

The add function is done by using a simple carry adder.

The subtraction function uses the complement method. We invert B and use the ripple carry adder to do A+Band we add one. We add one using the Cin input of the rripple carry adder module.

The Test for bit equality has its own schematic, which is just a 2 Level circuit. We use a XNOR gates for each element of A to compare it with his B homologue. Then we use a 4AND gate to see if A completely equal B.

The Divide by Two function hasn't got any module, it is a simple shift left of the input. It goes directly in the enabler, the entry are just shifted.

The ALU calculates everything simultaneously. Then we use an enabler to select the result. Each function has its own enabler and All the enablers are the same. An enabler just passes the inputs into outputs when the enable line is on.

To turn on the right enable line according to s0 and s1, we use a decoder.

Conclusion :

Our ALU works according to the specified certification.

We made a gate level system taking care to do as few levels as possible.

In addition to work, our solution is optimized.

However, you have to pay attention to some details:

- It is not planned to be able to subtract a large number from a small one. The displayed result is not valid.

- We can divide only a binary number of 6 digits. Additional digits will be ignored.

There are many ways to improve our system.

-On the circuit, we did not know how to access the positive wire.When it was necessary to add 1 during the complement method for the subtraction, we used a stratagem thanks to an inverter. This is certainly not optimized.

- We can check which is the largest number for subtraction, or turn off the LEDs if A <B.

- The divide by 2 function could divide larger numbers.

- We used the 25mhz clock. Our system could be 4 times faster if we use the 100mhz clock.

- On this FPGA, we can use the four 7-segment LED display to show our result instead of LED.

-The point of an ALU is to be multifunctional. But to add functions, we need more input and output on our FPGA. Or we can use the serial ports to add many switches and LED. So we can add almost as many functions as we want.