Assignment 1 – ALU Design

Student Name:	**Aaron Dinesh**
Student ID Number:	**20332661**
Board Number	**8**
Assessment Title:	**Assignment 1**
Lecturer (s):	**Shreejith Shanker**
Date Submitted	**18/03/23**

Signed: [illegible]

Date: 18/03/23

Introduction

The assignment required us to implement a 6-bit ALU using the 6-bit Carry Ripple Adder (CRA) from lab C, the 8-bit greater than or equal to (GTHE) module from lab B and the XNOR module from lab A. The top level ALU module could not have any logical gates in it, but only instantiations of the modules mentioned above. The ALU needed to perform different tasks based on the value of then “fxn” input, these are summarized in the table below.

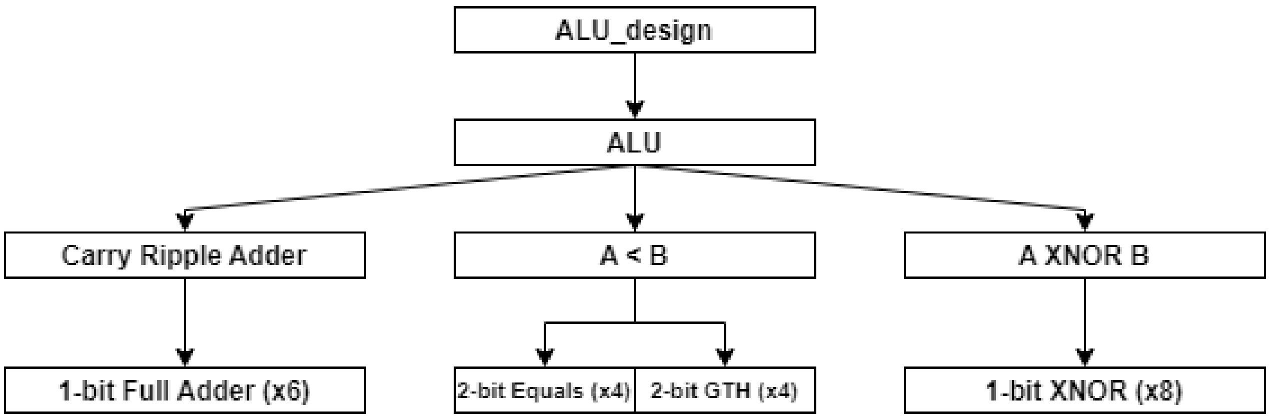
Fxn	X[5:0]
000	A
001	B
010	-A
011	-B
100	A<B
101	A XNOR B (Bitwise)
110	A+B
111	A-B

Modules

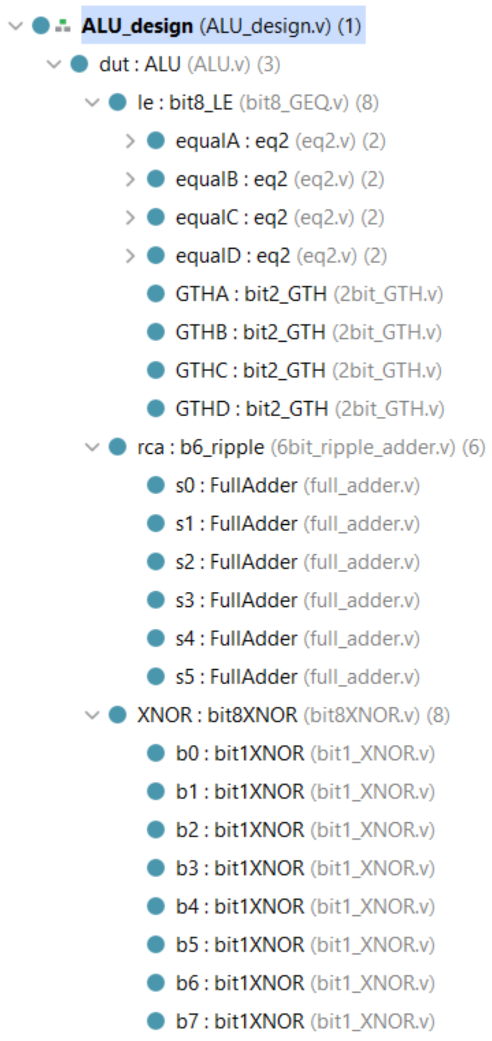
As stated, before only modules designed in previous labs could be used in this assignment. However, some minimal modifications needed to be made to the files in order for them to work. For example, the GTHE module was designed without 2s compliment numbers in mind. So, in order for it to work as a 2s compliment less than module, I needed to implement some logic to check if the numbers were negative and then in response either return the result of the greater than module or return the logical not of the result. A wrapper module also needed to be created for the XNOR module from the first lab, since that was a 1-bit XNOR module and the ALU required it to be a 6-bit bitwise XNOR. Aside from these changes, nothing else about the modules were changed. The 6-bit CRA module was used as is, without any changes. For the modules I decided to use registers for the top-level ALU module, as this allowed me to use always blocks and if statements to switch the inputs A and B into the input registers for the various modules based on what the values for the fxn input was. This simplifies the coding for the top-level module while also allowing the module to be scalable to any length input. However, an argument could be made this this would not be the most energy efficient design, as there needs to be N-bit input and output register for each module.

Module Hierarchy

The top-level module was the ALU_design module, this was the file that instantiated the ALU module and defined the input switches and the output LEDs so that the Basys3 board could be used to interact with the running program. A module hierarchy diagram is shown below:

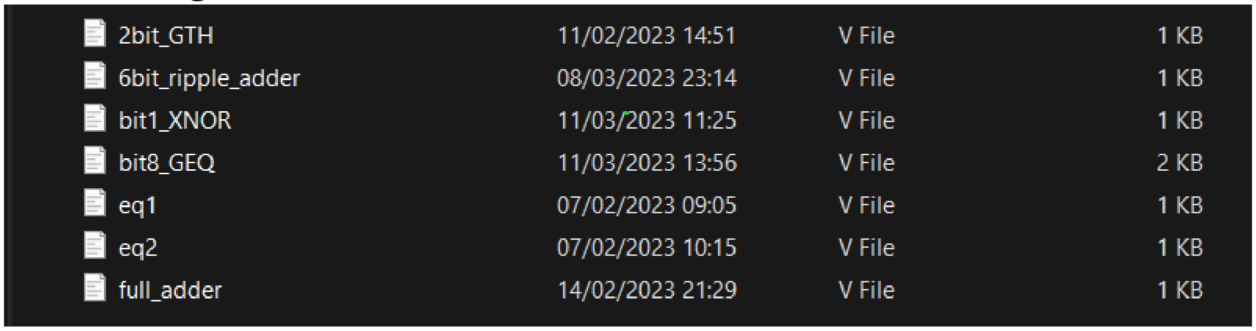


All the modules used were ones that were developed in the previous lab with slight modifications so that they would work for the task at hand. Shown below is the module hierarchy shown in vivado:



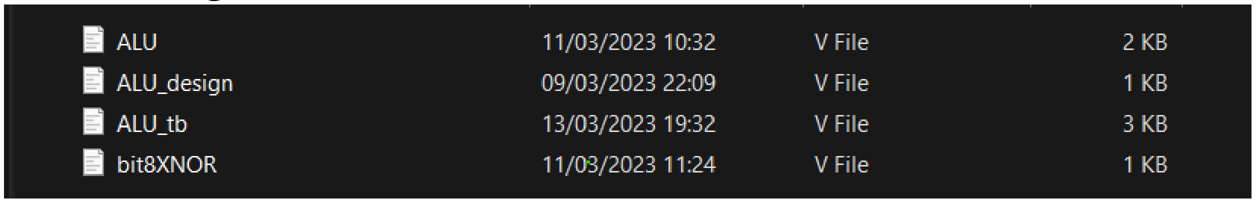
```
ALU_design (ALU_design.v) (1)
  dut : ALU (ALU.v) (3)
    le : bit8_LE (bit8_GEQ.v) (8)
      equalA : eq2 (eq2.v) (2)
      equalB : eq2 (eq2.v) (2)
      equalC : eq2 (eq2.v) (2)
      equalD : eq2 (eq2.v) (2)
      GTHA : bit2_GTH (2bit_GTH.v)
      GTHB : bit2_GTH (2bit_GTH.v)
      GTHC : bit2_GTH (2bit_GTH.v)
      GTHD : bit2_GTH (2bit_GTH.v)
    rca : b6_ripple (6bit_ripple_adder.v) (6)
      s0 : FullAdder (full_adder.v)
      s1 : FullAdder (full_adder.v)
      s2 : FullAdder (full_adder.v)
      s3 : FullAdder (full_adder.v)
      s4 : FullAdder (full_adder.v)
      s5 : FullAdder (full_adder.v)
    XNOR : bit8XNOR (bit8XNOR.v) (8)
      b0 : bit1XNOR (bit1_XNOR.v)
      b1 : bit1XNOR (bit1_XNOR.v)
      b2 : bit1XNOR (bit1_XNOR.v)
      b3 : bit1XNOR (bit1_XNOR.v)
      b4 : bit1XNOR (bit1_XNOR.v)
      b5 : bit1XNOR (bit1_XNOR.v)
      b6 : bit1XNOR (bit1_XNOR.v)
      b7 : bit1XNOR (bit1_XNOR.v)
```

Old Verilog Modules



2bit_GTH	11/02/2023 14:51	V File	1 KB
6bit_ripple_adder	08/03/2023 23:14	V File	1 KB
bit1_XNOR	11/03/2023 11:25	V File	1 KB
bit8_GEQ	11/03/2023 13:56	V File	2 KB
eq1	07/02/2023 09:05	V File	1 KB
eq2	07/02/2023 10:15	V File	1 KB
full_adder	14/02/2023 21:29	V File	1 KB

New Verilog Modules



ALU	11/03/2023 10:32	V File	2 KB
ALU_design	09/03/2023 22:09	V File	1 KB
ALU_tb	13/03/2023 19:32	V File	3 KB
bit8XNOR	11/03/2023 11:24	V File	1 KB

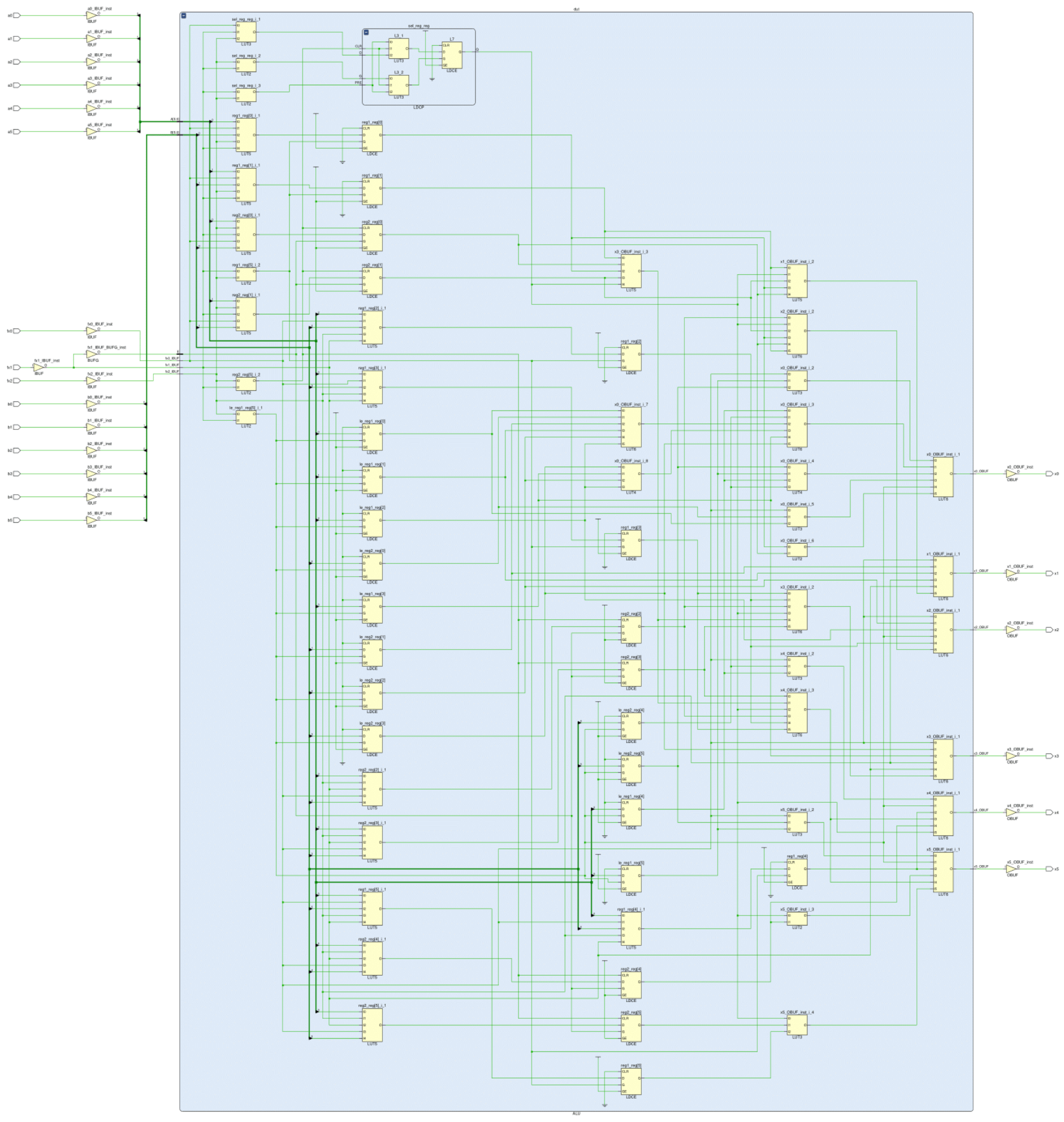
Testbench Used

I developed a testbench to verify that all the functions of the ALU were working as designed. For this, 14 test cases were created to test both expected cases as well as edges cases that could potentially break the module.

A [5:0]	B [5:0]	Fxn [3:0]	Expected Out [5:0]	Function
001000	000000	000	001000	Ret A for +ve A
010010	001010	001	001010	Ret B for +ve B
001010	001010	010	110110	Ret -A for +ve A
001010	001110	011	110010	Ret -B for +ve B
101010	001010	010	010110	Ret -A for -ve A
101010	101110	011	010010	Ret -B for -ve B
000010	001010	100	000001	Ret A< B for +ve A and +ve B with A<B
000100	000010	100	000000	Ret A<B for +ve A and +ve B with A>B
110101	001001	100	000001	Ret A<B for -ve A and +ve B with A<B
001010	001010	101	111111	Ret A~^B (Bitwise XNOR)
001010	001010	110	010100	Ret A+B for +ve A and +ve B
001010	110110	110	000000	Ret A+B for +ve A and -ve B
001010	001010	111	000000	Ret A-B for +ve A and +ve B
001010	111011	111	001111	Ret A-B for +ve A and -ve B

Testing all 32,768 possible permutations on the ALU would be unfeasible so I believe these 14 testcases showcase enough of the module to trust that all the functions are implemented properly.

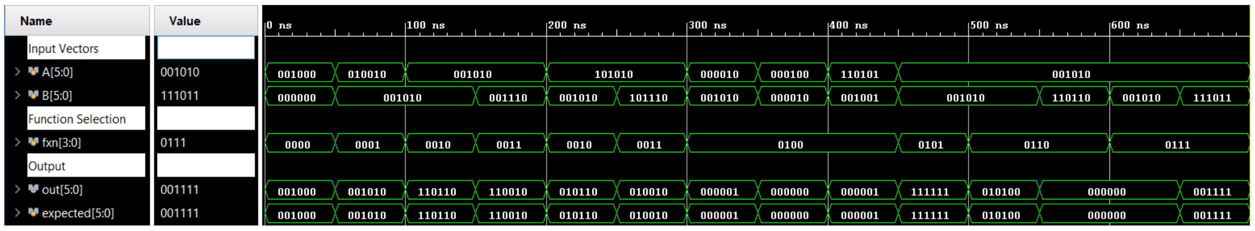
ALU Schematic



Aaron Dinesh 20332661

Simulation Waveforms

The above test cases were simulated using the FPGA simulator in vivado with the output wave form listed below.



The waveforms are split into 3 distinct sections. A and B show the 2 input test vectors, fxn is the 3-bit number that selects what function the ALU should perform based on the table shown in the introductory section. Finally, the output of the ALU is listed as well as the expected output. The expected output was computed by hand based on the test vectors and the current ALU function selected. As evident in the screenshot all the outputs match what the expected output should be for that function, so we can conclude that the ALU is working according to the specification outlined in the assignment brief.

Demonstration

In order to get the code working on the Basys3 board you need to make sure that the ALU_desgin.v module is the top level module before you perform the synthesis. Once the board is flashed, the switches are allocated as follows. Starting from the right of the board, the first 6 switches (V17, V16, W17, W15, V15) control bits 0 to 5 of the A input. The next 6 switches (W14, W13, V2, T3, T2, R3) control bits 0 to 5 of the B input and then the next 3 switches (W2, U1, T1) control bits 0 to 3 of the fxn input. The output from the module is mapped to the first 6 LEDs starting from the right, i.e. U16, E19, U19, V19, W18, U15 are mapped to the 0-5 bits of the output respectively.

Conclusions

In conclusion, I believe that the above test cases accurately test the most common use cases of this module as well as any edge cases that may arise. During simulation, the board passed all the test cases, so I can confidently say that the board meets all the specifications listed in the brief.