EECS/CSE31L Midterm Project Report

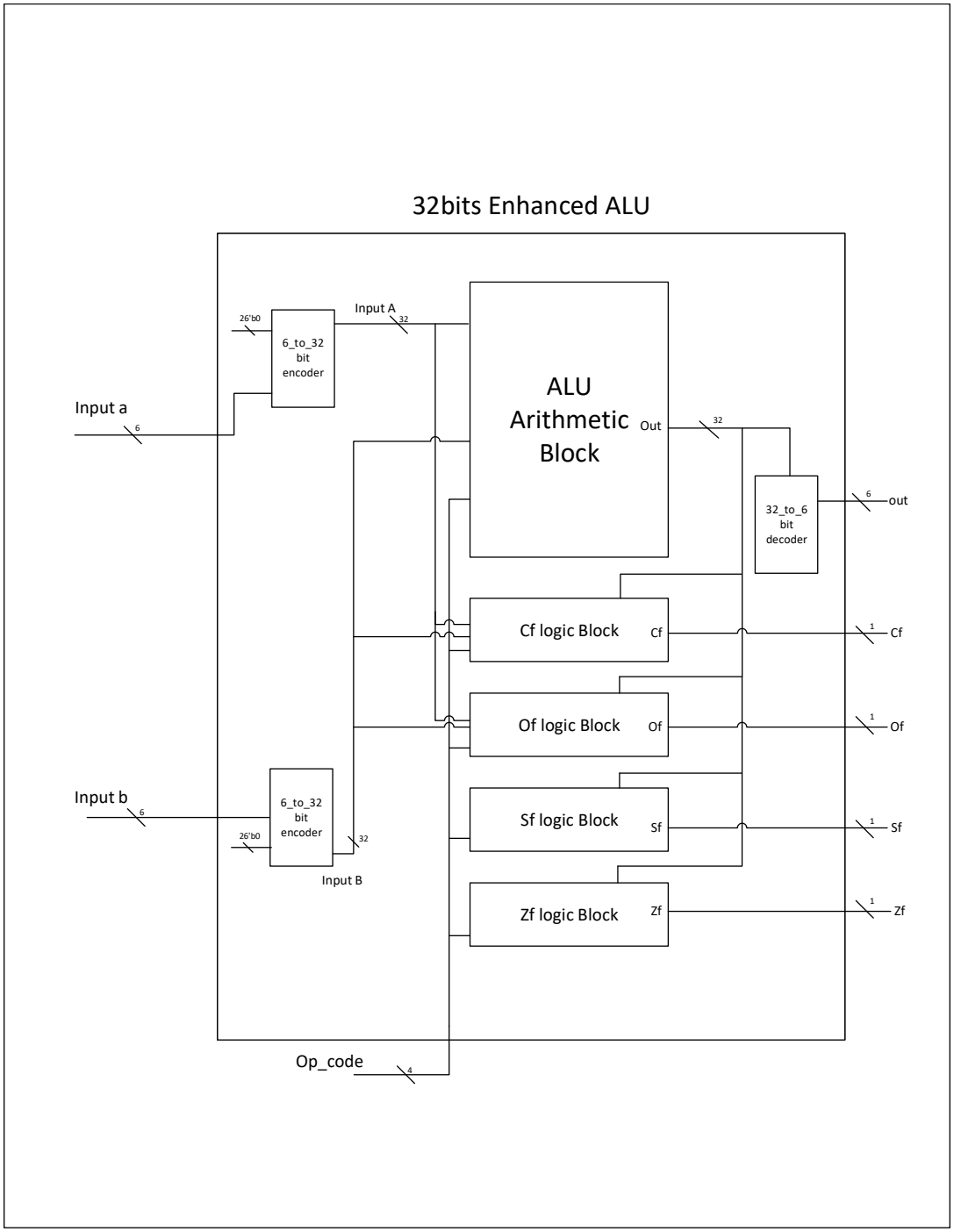
Github repo: 32bits_Enhanced_ALU

Tianyi Yang 69386993, Mina Bedwany 69543570, Toan Tran 36086435,
Takumi Okamoto 86483191, Ivan Jaime 94683522

October 29, 2017

For this project, we designed a 32-bit enhanced ALU using SystemVerilog that performs various operations. The operations were each encoded as 4-bits ranging from `0000` to `1011` for addition, subtraction, increment, decrement, move, 1-bit logical left shift, `SIMD` add, `AND`, `OR`, `XOR`, complement, and 2's complement, respectively. The ALU takes in two 32-bit inputs, A and B, and one 4-bit operation code called `Op_Code`. Then, it gives out the result as a 32-bit output called Out, and 1- bit output for four flags, Overflow flag, Zero flag, Sign flag, and Carry flag. The Carry flag would be set to 1 whenever the operation is Add and output has a carryout, the Overflow Flag would be set to 1 whenever the operation is Add or Subtract and output has an overflow, the Sign flag would be set to 1 whenever the operation is Add or Subtract and the sign bit of the bit was switched, and the Zero flag would be set to 1 whenever the output was 0 for all operations. The operations were done using the SystemVerilog operators, which made it pretty straightforward in terms of code. The only operator that wasn't as straightforward and required some understanding was the `SIMD` Add operator. We were able to write the code for that by slicing each input, A and B, into four 8-bits and slicing the output into four 8-bits as well. Then, each sliced group from A and B were added separately and assigned to the corresponding sliced group of the output, Out. This way, the addition of each 8-bit group will not interfere with the rest of the bits especially if there is a carry out, which in this case will be discarded.

The whole design was heavily dependent on conditional statements (`? :`) for each operation, and basically, the main idea was similarly resembling that of a multiplexer. This is how the block diagram for our design would look like:


32bits Enhanced ALU
Input a
6
26'b0
6_to_32 bit encoder
Input A
32
ALU Arithmetic Block
Out
32
32_to_6 bit decoder
6
out
Cf logic Block
Cf
1
Cf
Of logic Block
Of
1
Of
Sf logic Block
Sf
1
Sf
Zf logic Block
Zf
1
Zf
Input b
6
26'b0
6_to_32 bit encoder
32
Input B
Op_code
4

After the code was implemented, we simulated and tested the design. For testing, we did testing on Vivado by forcing constants and checking the results, and also hardware testing by programming our Basys3 FPGA board to our code and testing various scenarios. On Vivado, we forced constants for each operation to make sure every operation works as expected. For example, for `SIMD` Add operation, we forced constants for the inputs, A and B, using the inputs given in the midterm assignment pdf, `10110111 00111010 11100010 01101100` for A, and `10010010 10010010 00101001 10100010` for B. We got the expected result `01001001 11001100 00001011 00001110`, which shows that indeed the carry bits after every 8-bits were discarded and the code was implemented correctly. All the other operators were tested using multiple examples to ensure everything works correctly, and screenshots of these tests are attached in the Simulation file of this project. The flags also were tested and the showed correct behavior, for example, I forced A to be `10000000000000000000000000000000` and B to be `10000000 00000000 00000000 00000000` and used the Add operator (`Op_Code = 0000`). The result was `00000000 00000000 00000000 00000000` with the Carry flag, Overflow flag and Zero flag all set to 1, which is the expected result. The next step in testing was using the FPGA board, we modified the code to have 6-bit inputs and 6-bit output. The board has 16 switches, the first four were used for the 4-bit `Op_Code`, and the next 6 switched were for the 6-bit input A and the last 6 were for the 6-bit input B. Each output bit was pinned to an LED and several operations were tested and the corresponding LEDs lit up as expected. We didn't face any errors in the results or any bugs, and the code seems to be functioning per the specs.