Project 2

Jacob Abel

November 1, 2017

Contents

List of Tables

List of Figures

Purpose

The purpose of this project was to design and implement a 8-bit Arithmetic Logic Unit (ALU) in verilog. This requires application of combinational logic design skills, knowledge of verilog syntax and style, an understanding of multiplexers, and circuit analysis such as comparing waveforms and measuring propagation delay and total gate count.

DIP[3:0]	Display Mode
1000	Current Address
1001	Opcode with leading 0s
1010	Operand A
1011	Operand B
1100	Result
1101	Status Bits (VZCN) with leading 0s
1110	First 2 of the last 4 Student ID digits in BCD
1111	Last 2 of the last 4 Student ID digits in BCD

Table 1: LED Display Modes

Problem Specification

This project requires that an ALU be implemented as combinational circuit taking two 8-bit operands and outputting an 8-bit resultant and 4 status bits representing integer overflow(V), an empty(zero) resultant(Z), the carry out of an addition or subtraction(C), and whether the resultant is negative(N). The required operations and their corresponding opcodes are provided in table 2. Additionally the project requires that a multiplexer based combinational circuit be implemented that processes inputs to feed into the ALU as well as format the output so that it can be displayed on a seven LED array. The overall system will take in two push-buttons for reading instructions from ROM and 4 DIP switches for toggling between display modes for the seven LEDs. The display modes are provided in table 1.

OPCODE	Operation	Inputs	Status Bits	Description
0x0	NOT	A	0, Z, 0, N	Bitwise NOT
0x1	AND	A, B	0, Z, 0, N	Bitwise AND
0x2	XNOR	A, B	0, Z, 0, N	Bitwise XNOR
0x3	LSL	B	0, 0, 0, 0	Logical Shift Left « 1
0x4	ASR	B	0, 0, 0, 0	Artihmetic Shift Right » 1
0x5	ADD	A, B	V, Z, C, N	Addition
0x6	SUB	A, B	V, Z, C, N	Subtraction
0x7	INC	A	V, Z, C, N	Increment by 1
0x8	DEC	A	V, Z, C, N	Decrement by 1
0x9	NEG	B	V, Z, C, N	Invert Sign
0xA	MULTI8	B	0, Z, 0, N	Multiply by 8
0xB	MOD4	A	0, Z, 0, N	Modulus by 4 (Remainder)

Table 2: ALU Operations

Design Process

The first stage of the project was grouping by their commonalities. Adding, Subtracting, Incrementing, Decrementing and Inverting sign all require the use of an adder and as such they were grouped together. Similarly the shift operators logical shift left and arithmetic shift right were grouped together. Multiply by 8 is grouped with the shift operators as multiplication by a power of 2 is equivalent to a shift operation. Additionally, all the bitwise primitive operations were grouped together as they all only required primitive gates. The Modulus by 4 operator was unique in that it was a shift based operator that was dependent on sign. This makes it a hybrid between the shift and adder groups.

The second stage was parsing the status bits. The overflow bit was just an XOR of the last two C_{out} bits of the adder. The zero bit was handled by simply comparing the result to 0. The carry bit was a direct wire from the adder. The negative bit was a direct wire from the sign bit of the result. The disabled bits are disabled on certain operations by wrapping them into a conditional.

The final step of designing the ALU was retrieving the result from the operators. The result was selected from the ALU by wrapping all the operators into a 4x12 multiplexer.

Designing the ALU Mux wrapper module was fairly straight-forward as all it required was updating the ID wire, instantiating the ALU, and feeding all the outputs necessary for the various LED display modes into the 8-bit wide 4x16 multiplexer and mapping the multiplexer output to the module output. While there was an option of adding additional display modes to the LEDs, the decision was made that only the standard display modes would be implemented. As such, display modes 000-111 are all disabled.

Implementation

Following classification, each group of operators was implemented with each operator in the group being implemented with only minor variations compared to the rest of the group. Primitives were implemented with basic structural verilog requiring only one gate per bit. The shift operators were implemented using the replication and concatenation operators. Adder based operators were implemented with only one ripple carry adder and several multiplexers to introduce the variations necessary to implement the various operators. The Modulus by 4 operator was implemented by rendering the input into a positive value with a multiplexer and the inverted sign operator (NEG) and then concatenating down to the first 2 bits and padding the rest with 0s.

The implementation makes every effort to be efficient, maintaining only one instance of a ripple carry adder and avoiding duplication wherever possible. This is evident with a gate count coming out to 358 AND gates, 53 OR gates, and 97 inverters as shown in Appendix B: Gate Count Calculations. Given additional time, gate count could likely be further reduced but is currently for all intents and purposes sufficient. In respect to efficiency with regard to propagation delay, performance is fairly decent despite focus being primarily placed on minimising gate count. As shown in Appendix C: Propagation Delay Calculations, the propagation delay for the final circuit comes out to $40t_{pdAND} + 20t_{pdNOT} + 8t_{pdOR}$ from the opcode input to the overflow status bit.

Validation

To validate the modules, the operators were tested and compared against predicted values. The ALU simulation waveform figure 1 and ALU multiplexer simulation waveform figure 2 are located in Appendix A: Simulation Waveforms. The values returned by the ALU and the expected results provided in table 3 demonstrate that the ALU does function correctly.

OP	Name	A	B	Result(ALU)	Result(Real)	Status(ALU)	Status(Real)
0x6	SUB	17	-87	104	104	0000	0000
0x5	ADD	119	-11	108	108	0010	0010
0x1	AND	-57	118	70	70	0000	0000
0x2	XNOR	63	-64	0	0	0100	0100
0x0	NOT	73	-57	-74	-74	0001	0001
0xA	MULTI8	118	-9	-72	-72	0001	0001
0x7	INC	-82	63	-81	-81	0001	0001
0x9	NEG	-47	17	47	47	0000	0000
0x8	DEC	-107	-57	-108	-108	0011	0011
0xB	MOD4	-47	-11	3	3	0000	0000
0x4	ASR	-9	-103	-52	-52	0000	0000
0x3	LSL	63	92	-72	-72	0000	0000

Table 3: Validation Results

Conclusion

This project was a fantastic exercise in designing and implementing an ALU. It provides a wide breadth of tasks as well as a bit of depth without being too overwhelming. The assignment documentation is clear and precise. There was nothing that couldn't be understood with a few reads of the documentation.

Given sufficient time to redo the project, gate count and propagation delay could be reduced in a number of ways. Replacing the ripple carry adder with a carry lookahead adder could reduce propagation delay. Additionally, further analysis could likely allow duplicate circuits to be further reduced and eliminated as the focus was largely on eliminating duplicates with large signatures. Something that would have made testing significantly simpler would have been implementing test fixtures rather than just comparing simulation results. Next time a project occurs, this will likely be integrated into the design process.

Appendix A: Simulation Waveforms

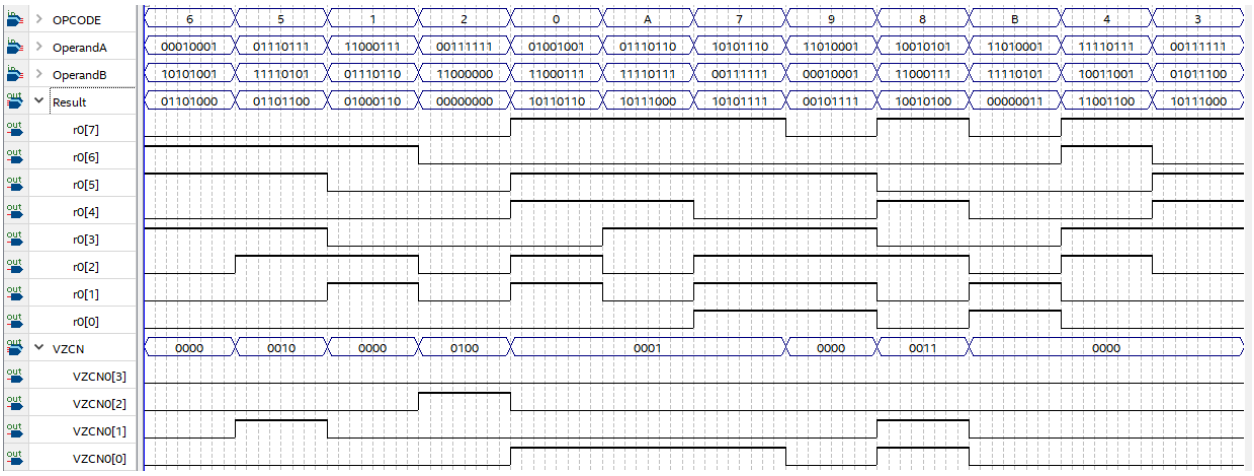

Figure 1: ALU Simulation Waveform

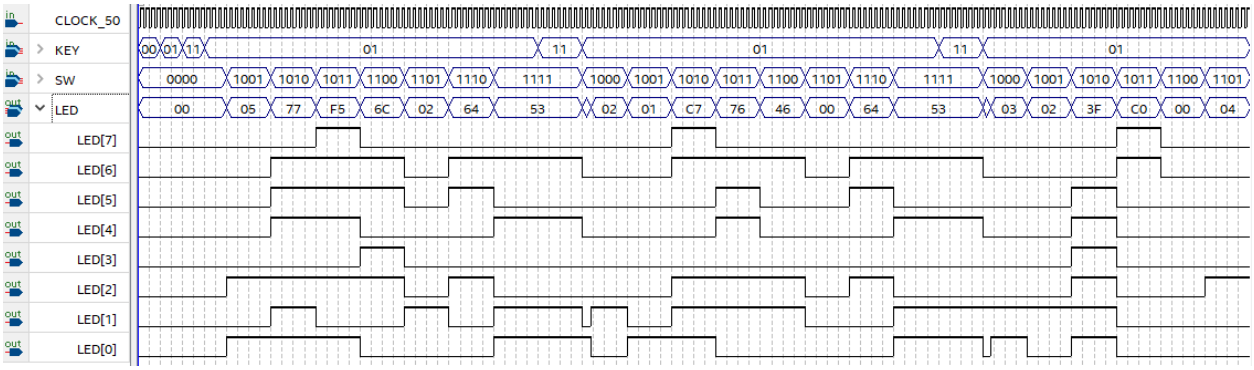

Figure 2: ALU Multiplexer Simulation Waveform

Appendix B: Gate Count Calculations

$$
\begin{aligned}
HA &= XOR + AND = 3AND + OR + NOT \\
FA &= 2HA + OR = 6AND + 3OR + 2NOT \\
RC8 &= 8FA = 48AND + 24OR + 16NOT \\
OP_{LSL} &= 0 \\
OP_{ASR} &= 0 \\
OP_{MULTI8} &= 0 \\
OP_{NOT} &= 8NOT \\
OP_{AND} &= 8AND \\
OP_{XNOR} &= 8XOR + 8NOT = 16AND + 8OR + 16NOT \\
MUX_{1x2} &= NOT + 2AND \\
EQ_4 &= 4XNOR + 3AND = 11AND + 4OR + 8NOT \\
EQ_8 &= 8XNOR + 7AND = 23AND + 8OR + 16NOT \\
DEC_{4x2} &= 4AND + 2NOT \\
MUX_{4x12} &= 2DEC_{4x2Shared} + 12AND \\
MUX_{4x6} &= 2DEC_{4x2Shared} + 6AND \\
MUX_{4x4} &= 2DEC_{4x2Shared} + 4AND \\
MUX_{4x3} &= 2DEC_{4x2Shared} + 3AND \\
MUX_{4x2} &= 2DEC_{4x2Shared} + 2AND \\
OP_{ADD} &= OP_{SUB} = OP_{INC} = OP_{DEC} = OP_{NEG} \\
&= 8MUX_{4x2} + 8MUX_{4x4} + 8MUX_{4x3} + RC8 \\
&= 2DEC_{4x2Shared} + 120AND + 24OR + 16NOT \\
OP_{MOD4} &= 8MUX_{1x2} + OP_{NEGSHARED} \\
ALU &= OP_{NOT} + OP_{AND} + OP_{XNOR} + 8MUX_{4x12} + OP_{ADD} \\
&+ OP_{MOD4} + 2MUX_{4x6} + 3EQ_4 + 3NOT + EQ_8 + XOR \\
&= 2DEC_{4x2} + OP_{NOT} + OP_{AND} + OP_{XNOR} \\
&+ 8MUX_{4x12} + 2MUX_{4x6} + OP_{ADD} + 2AND + OR + NOT \\
&+ OP_{MOD4} + 3EQ_4 + 3NOT + EQ_8 \\
&= 8AND + 4NOT + 8NOT + 8AND + 16AND + 8OR + 16NOT \\
&+ 96AND + 12AND + 120AND + 24OR + 16NOT \\
&+ 2AND + OR + NOT + 8NOT + 16AND + 33AND \\
&+ 12OR + 24NOT + 23AND + 8OR + 16NOT \\
&= 93NOT + 334AND + 53OR \\
MYMUX &= 2DEC_{4x2} + 16AND = 24AND + 4NOT \\
yourALUmux &= ALU + MYMUX = 93NOT + 334AND + 53OR + 24AND + 4NOT \\
&= 97NOT + 358AND + 53OR
\end{aligned}
$$

Appendix C: Propagation Delay Calculations

$$
\begin{aligned}
t_{pdXNOR} &= 2t_{pdAND} + 2t_{pdNOT} \\
t_{pdEQ4} &= t_{pdXNOR} + 2t_{pdAND} \\
&= 4t_{pdAND} + 2t_{pdNOT} \\
t_{pdHA} &= t_{pdXOR} = 2t_{pdAND} + t_{pdNOT} \\
t_{pdFAs} &= 2t_{pdHA} = 4t_{pdAND} + 2t_{pdNOT} \\
t_{pdFAc} &= 2t_{pdHA} + t_{pdOR} = 4t_{pdAND} + 2t_{pdNOT} + t_{pdOR} \\
t_{pdRC8s} &= 7t_{pdFAc} + t_{pdFAs} = 32t_{pdAND} + 16t_{pdNOT} + 7t_{pdOR} \\
t_{pdRC8c} &= 8t_{pdFAc} = 32t_{pdAND} + 16t_{pdNOT} + 8t_{pdOR} \\
t_{pdDEC} &= t_{pdNOT} + t_{pdAND} \\
t_{pdmux4xA} &= t_{pdDEC} + t_{pdAND} = t_{pdNOT} + 2t_{pdAND} \\
t_{pdOverflow} &= 2t_{pdmux4xA} + t_{pdRC8c} + t_{pdXOR} \\
&= 38t_{pdAND} + 19t_{pdNOT} + 8t_{pdOR} \\
t_{pdALU} &= t_{pdOverflow} = 38t_{pdAND} + 19t_{pdNOT} + 8t_{pdOR} \\
t_{pdMYMUX} &= t_{pdDEC} + tpdAND \\
&= t_{pdNOT} + 2t_{pdAND} \\
t_{pdyourALUmux} &= t_{pdALU} + t_{pdMYMUX} \\
&= 40t_{pdAND} + 20t_{pdNOT} + 8t_{pdOR}
\end{aligned}
$$

Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board
Validation Sheet – Page 1

The Validation Sheet for Project 2 is two pages long. Make sure that you include both pages as the last two pages of your project report. You should complete page 1 and leave page 2 blank for the TA to complete.

No *GTA* or *student* should discuss any aspect of a student's design with another student. Among other things, this includes the manner in which a student implements specific operations.

Student Name: Jacob Abel
Last four Digits of your student ID: 6453

GTA Validation Instructions

1. Program the FPGA on the DE0 Nano board using the Start button on the programmer window.
2. When the programming has successfully completed, reset the design by pressing and releasing the KEY0 pushbutton.
3. Set the DIP switches to 1110 and record the value of the LEDs as two hex digits in Table 1 on the next page. (The DIP switches are 0 on the side labeled "ON".)
4. Set the DIP switches to 1111 and record the value of the LEDs as two hex digits in Table 1.
5. Compare the four digits from steps 3 and 4 to the last four digits of the student's ID number. **If the four digits do not match the last four digits of the student's ID number on their ID card, STOP THE VALIDATION. DO NOT CONTINUE.**

For the remaining steps of the validation, the values of the switch settings in Gray code order will allow the requested items to be checked in the order in the table more quickly. **All values should be recorded in the table as two hexadecimal digits.** For each address, verify that the opcode, operand A and operand B digits match the value in the corresponding address in the rom.txt file on the student's computer.

6. Fill in the row of Table 2 for the "sub" operation. Verify that the address is 0.
7. Press and release KEY1.
8. Fill in the row of Table 2 for the "add" operation. Verify that the address is 1.
9. Press and release KEY1.
10. Fill in the row of Table 2 for the "and" operation. Verify that the address is 2.
11. Press and release KEY1.
12. Fill in the row of Table 2 for the "xnor" operation. Verify that the address is 3.
13. Press and release KEY1.
14. Fill in the row of Table 2 for the "not" operation. Verify that the address is 4.
15. Press and release KEY1.
16. Fill in the row of Table 2 for the "mult8" operation. Verify that the address is 5.
17. Press and release KEY1.
18. Fill in the row of Table 2 for the "inc" operation. Verify that the address is 6.
19. Press and release KEY1.
20. Fill in the row of Table 2 for the "neg" operation. Verify that the address is 7.
21. Press and release KEY1.
22. Fill in the row of Table 2 for the "dec" operation. Verify that the address is 8.
23. Press and release KEY1.
24. Fill in the row of Table 2 for the "mod4" operation. Verify that the address is 9.
25. Press and release KEY1.
26. Fill in the row of Table 2 for the "asr" operation. Verify that the address is A.
27. Press and release KEY1.
28. Fill in the row of Table 2 for the "lsl" operation. Verify that the address is B.

Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board
Validation Sheet – Page 2

Table 1: Checking BCD equivalent to student ID

	Switch setting, SW[3:0]	
	1110	**1111**
LED value in hexadecimal	____ ____	____ ____

Table 2. Checking the operation of the ALU. All values of the LEDs should be recorded as two digit hexadecimal numbers.

SW[3:0] →	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**
operation	**address**	**opcode**	**operandA**	**operandB**	**result**	**status**
sub	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
add	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
and	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
xnor	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
not	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
mult8	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
inc	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
neg	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
dec	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
mod4	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
asr	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
lsl	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___

Comments (only required if something is unusual or wrong):