Lab #2: Partial Arithmetic and Logic Unit

EECE 2323 – Prof. Xiaolin Xu

Date: 02/08/2022

Name: Fengyi Quan && Yichen Sun

Northeastern University

1. **Background & Purpose**

The object of this lab is to create a combination circuit, Arithmetic and logic Unit (ALU) which can perform certain calculations on binary inputs. In this lab, it is required to implement a partial ALU which responsible for addition, bitwise NOT, bitwise AND, and bitwise OR. Then generate a bitstream to be tested on the TUL PYNQ board and display the results on the LEDs. Each LED represents one single bit of outputs. It has a select signal to indicate which operation should perform and two 8-bit inputs. It outputs the result based on the select signal and one overflow bit, which only applicable when doing addition. Overflow bit should always be 0 when doing other operations.

1. **Prelab**
   1. **Design the Partial ALU**

It uses a case statement to check if select signal is to indicate which operation to perform, default case should never reach since all possible cases are specified above. The ovf bit is calculated after result f computed, thus, assignment is outside of the case statement.

Graphical user interface, text, application, email

Description automatically generated

* 1. **Create Test Vector**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | binary\_a | b | binary\_b | s | f | binary\_f | ovf |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 8’d0 | 8’d0 | 0000 0000 | 1’d0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 8’d0 | 8’d46 | 0010 1110 | 1’d0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 8’d0 | -8’d46 | 1101 0010 | 1’d0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 8’d0 | 8’d50 | 0011 0010 | 1’d0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 8’d0 | -8’d50 | 1100 1110 | 1’d0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 8’d0 | -8’d56 | 1100 1000 | 1’d1 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 8’d0 | 8’d56 | 0011 1000 | 1’d1 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 8’d0 | 8’d3 | 0000 0011 | 1’d0 |
| -8’d3 | 1111 1101 | 8’d0 | 0000 0000 | 8’d0 | -8’d3 | 1111 1101 | 1’d0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 8’d0 | 8’d2 | 0000 0010 | 1’d0 |
| 8’d75 | 0100 1011 | 8’d100 | 0110 0100 | 8’d0 | - 8’d81 | 1010 1111 | 1’d1 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 8’d1 | -8’d1 | 1111 1111 | 1’d0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 8’d1 | -8’d35 | 1101 1101 | 1’d0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 8’d1 | 8’d33 | 0010 0001 | 1’d0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 8’d1 | 8’d49 | 0011 0001 | 1’d0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 8’d1 | -8’d51 | 1100 1101 | 1’d0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 8’d1 | -8’d101 | 1001 1011 | 1’d0 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 8’d1 | 8’d99 | 0110 0011 | 1’d0 |
| 8’d2 | 0000 0010 | 8’d3 | 0000 0011 | 8’d1 | -8’d4 | 1111 1100 | 1’d0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 8’d1 | -8’d2 | 1111 1110 | 1’d0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 8’d2 | 8’d0 | 0000 0000 | 1’d0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 8’d2 | -8’d1 | 1111 1111 | 1’d0 |
| 8’d15 | 0000 1111 | 8’d35 | 0010 0011 | 8’d2 | 8’d3 | 0000 0011 | 1’d0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 8’d2 | -8’d44 | 1101 0100 | 1’d0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 8’d2 | 8’d68 | 0100 0100 | 1’d0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 8’d2 | 8’d100 | 0110 0100 | 1’d0 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 8’d2 | -8’d100 | 1001 1100 | 1’d0 |
| 8’d127 | 0111 1111 | 8’d1 | 0000 0001 | 8’d2 | 8’d1 | 0000 0001 | 1’d0 |
| 8’d75 | 0100 1011 | 8’d102 | 0110 0110 | 8’d2 | 8’d66 | 0100 0010 | 1’d0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 8’d3 | 8’d0 | 0000 0000 | 1’d0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 8’d3 | 8’d46 | 0010 1110 | 1’d0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 8’d3 | -8’d2 | 1111 1110 | 1’d0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 8’d3 | -8’d66 | 1011 1110 | 1’d0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 8’d3 | 8’d100 | 0110 0100 | 1’d0 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 8’d3 | 8’d3 | 0000 0011 | 1’d0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 8’d3 | 8’d1 | 0000 0001 | 1’d0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 8’d3 | -8’d1 | 1111 1111 | 1’d0 |

* 1. **Testbench**

Graphical user interface, application

Description automatically generated

Graphical user interface, application

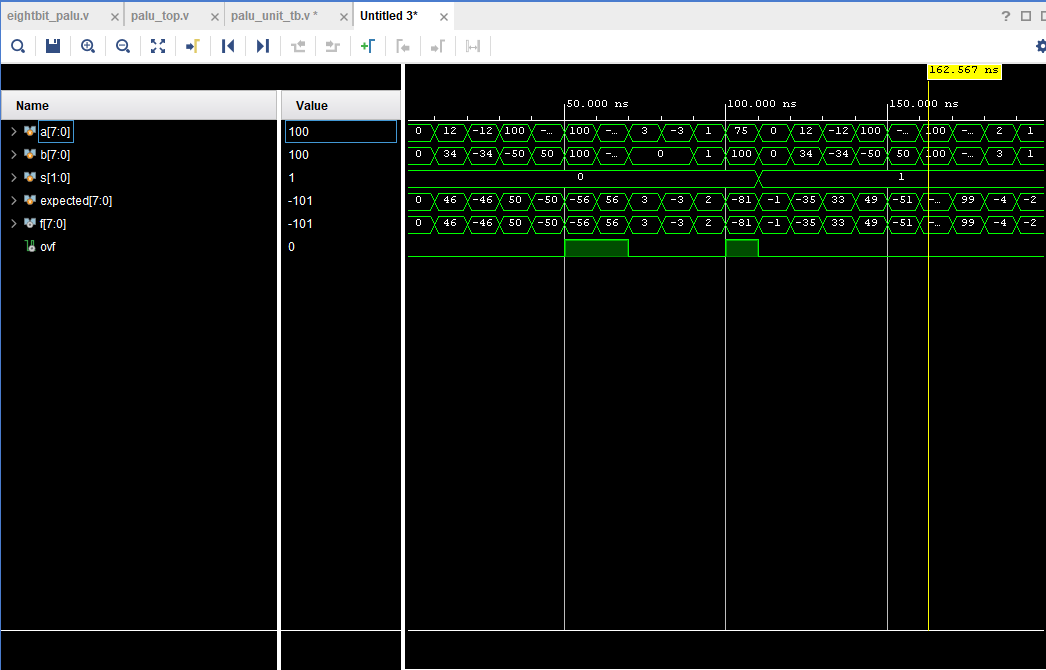
Description automatically generated

* 1. **Simulation**

Please see Appendix C: Partial ALU testbench simulation waveform.

1. **Results and Analysis**

The designed partial ALU module includes two 8-bit inputs, a and b, a 2-bit input s as operator with one 8-bit output f, and an overflow signal output. According to the testbench simulation, the results correctly demonstrate partial ALU module operations. Two 8-bit inputs represent two digits in the simulation waveform and complete the calculation with corresponding operator s since the partial ALU maintains some utilities the typical ALU does, so the calculated output largely depends on the choice of operators and input values.



Appendix C: Partial ALU testbench simulation waveform

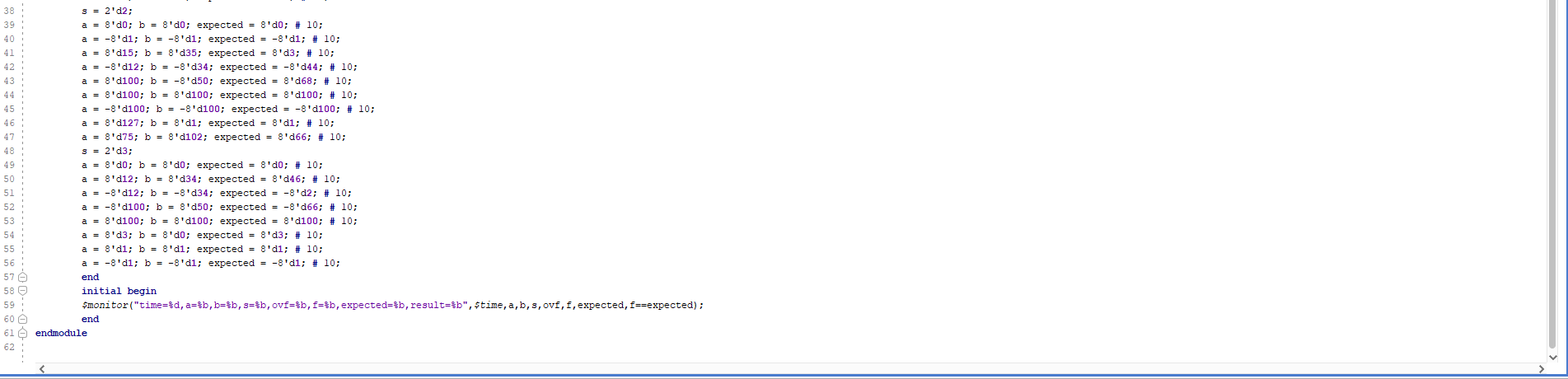
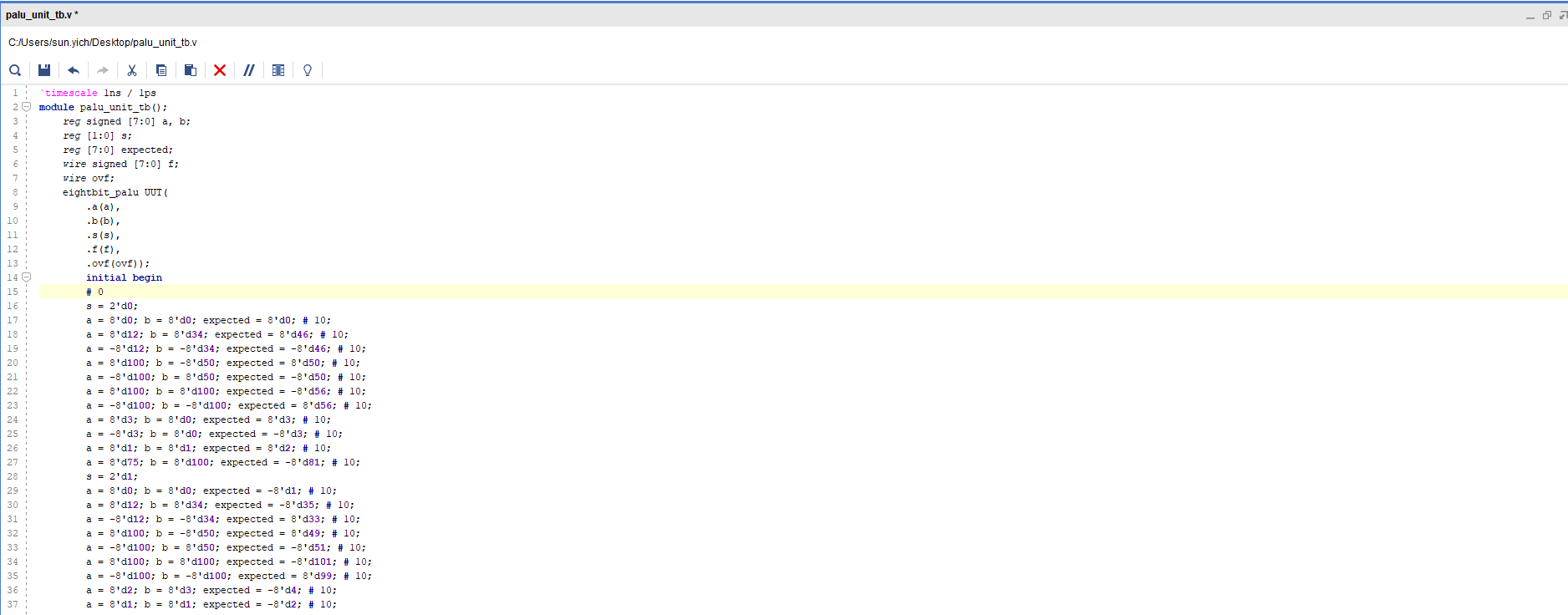
The partial ALU design represents the first flow of the data channel of the calculator. The data channel contains circuits that store, move, and manipulate data. Implementing the partial ALU circuit provides a practical analysis of ALU on the computer. The gates in the partial ALU circuit are controlled by a sequence logic unit that uses a particular operator or algorithm for each input operation. Examples of arithmetic operations in partial arithmetic are addition. Examples of logic operations consist of comparing values such as INV, AND, and OR. More importantly, the addition of partial ALU is said to overflow if the most significant number that an 8-bit output can hold is exceeded. This fact leads to further analysis of subtraction and other arithmetic operations occurring later in the complete design. Consequently, the overflow signal ovf indicates integer overflow of addition. On the other hand, overflow outputs have no meaning when partial ALU is doing logic operations (INV, AND, OR).

1. **Conclusion & Recommendations**

The object of this lab is to implement a partial ALU module in Verilog and simulated logic circuit with testbench, then checked working behavior on the TUL PYNQ board. The result came out successfully. Based on the results of the simulation, it is necessary to experiment with more than 7 test values including the pre-lab's expected input and output table to comprehensively certify accurate behavior of the design. The design closely resembles the complete ALU interface, except less-bit select signal and no take branch output. It is helpful to get a start to implement a ALU for the next lab.

1. **Appendices**

Appendix A: Partial ALU Module 

Appendix B: Partial ALU testbench 

Appendix C: Partial ALU testbench simulation waveform 