CE3001 Lab1. Arithmetic Logic Unit(ALU)

Lu Shengliang

SLU001

School of Computer Engineering
Nanyang Technological University
SLU001@e.ntu.edu.sg

September 18, 2013

Abstract

An Arithmetic Logic Unit(ALU) is a circuit that does arithmetic, such as addition, subtraction, bitwise AND, bitwise OR, etc. The ALU is entirely combinational logic implemented in Verilog, with no storage or sequential operations.

1 Introduction

1.1 Description of ALU *Operations*

The eight computation types of ALU in this lab report are: *Addition*, *Subtraction*, *Logical AND*, *Logical OR*, *Shift left logical*, *Shift right logical*, *Shift right arithmetic* and *Rotate left*. The operations of ALU instructions are Described in the table below.

1.2 Structure of the rest of the paper

The rest of the paper first describes the Verilog implementation of ALU in *Section 2*. *Section 3* presents the experimental results using testbench, which valid the functions of our ALU. *Section 4* presents our conclusions and discussions.

2 Implementation

2.1 Verilog Code alu.v

The basic operations for Verilog implementing ALU are *arithmetic operators*, *logical operators*, *shift operators* and *concatenation*.

Operation	value	Equation	Description
ADD	000	A + B	Addition A + B in 2's Complement format
SUB	001	A - B	Subtraction A - B in 2's Complement format
AND	010	A & B	Logical(bitwise) AND of A, B
OR	011	A | B	Logical(bit-wise) AND of A,B
SLL	100	A << Imm	Shift left logical
SRL	101	A >> Imm	Shift right logical
SRA	110	A >> Imm(MSB shifted in)	shift right arithmetic
RL	111	A rot Imm	Rotate left

Note: The interface to the ALU consists of *A, B, op, Imm and out*. The information is listed in the table below.

Table 1: Description of ALU Operations

```
module alu(A, B, op, out, Imm);

  input signed [15:0] A, B;
  input [2:0] op;
  input [3:0] Imm;
  output [15:0] out;

  wire [3:0] i;
  reg [15:0] out;
  reg [31:0] tmp;

  always @(A or B or op)
    begin
      case (op)
      3'b000: out = A + B;         //ADD
      3'b001: out = A - B;         //SUB
      3'b010: out = A && B;        //AND
      3'b011: out = A || B;        //OR
      3'b100: out = A << Imm;      //SLL
      3'b101: out = A >> Imm;      //SRL
      3'b110: out = A >>> Imm;     //SRA
      3'b111:                      //RL
        begin
          tmp = {A, A} << Imm;
          out = tmp[31:16];
        end
      default: out = 16'd0;
    endcase
  end

endmodule
```

The implementation results will be listed in *Section 3*.

Port Name	Port Direction	size	Description
A	Input	16-bit	First operand
B	Input	16-bit	Second operand
op	Input	3-bit	Specify operation to be performed
Imm	Input	4-bit	Second amount for SLL, SRL, SRA and RL
Out	Output	16-bit	Output of the operation

Table 2: Port List Specification

3 Evaluation

3.1 Testbench Code 1 alu_tb.v

```
module alu_tb;
  reg [15:0] A, B;
  reg [2:0] op;
  reg [3:0] imm;

  wire [15:0] out;

  alu alu0(.A(A), .B(B), .op(op), .out(out), .imm(imm));
  initial
    begin
      A = 16'd0; B = 16'd0; op = 3'b000; imm = 4'd0;
      #10 A = 16'h130f; B = 16'h5701; op = 3'b000;
      #10 A = 16'hfedc; B = 16'hab98; op = 3'b001;
      #10 A = 16'hcdef; B = 16'h89ab; op = 3'b010;
      #10 A = 16'hcdef; B = 16'h89ab; op = 3'b011;
      #10 A = 16'hb042; imm = 4'd1; op = 3'b100;
      #10 A = 16'hb042; imm = 4'd1; op = 3'b101;
      #10 A = 16'hb742; imm = 4'd4; op = 3'b110;
      #10 A = 16'hb742; imm = 4'd4; op = 3'b111;
      #10 finish;
    end
endmodule
```

A testbench has been designed to test the *alu.v* file. A simulation function is provided by *ModelSim* software, which is used to generate visualized test procedure for checking and debugging. We also use the simulation results to test the correctness of the output of *alu.v*.

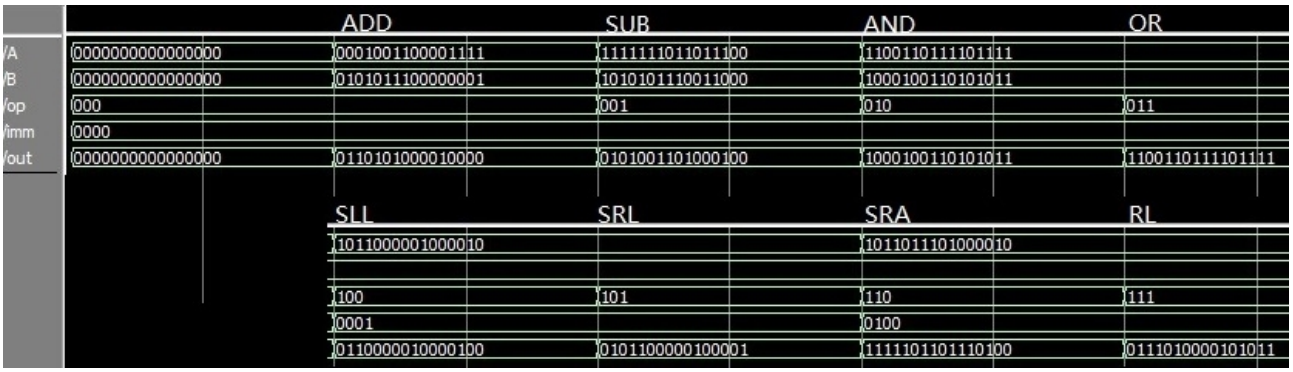

Figure 1: alu Testbench Simulation Result

Simulations results are intuitive and clear for checking the current status of each inputs and outputs. Figure 1 is a simulation result shown on ModelSim software. The first testbench is using to test the functionality of alu design. According to figure 1, basic functions of alu is implemented.

3.2 Testbench Code 2 alu_tb.v

An additional testbench is assigned to test the validity of alu design, which contains more conditions for testing the implementation. The Verilog code is shown below.

```
1       #10 A = 16'hfff9; B = 16'h0007; op = 3'b000;
2       #10 A = 16'h0007; B = 16'hfff9; op = 3'b001;
3       #10 A = 16'h89ab; B = 16'hfedc; op = 3'b010;
4       #10 A = 16'h89ab; B = 16'hfedc; op = 3'b011;
5       #10 A = 16'h789a; imm = 4'd15; op = 3'b100;
6       #10 A = 16'h789a; imm = 4'd15; op = 3'b101;
7       #10 A = 16'h8054; imm = 4'd15; op = 3'b110;
8       #10 A = 16'h8754; imm = 4'd15; op = 3'b111;
```

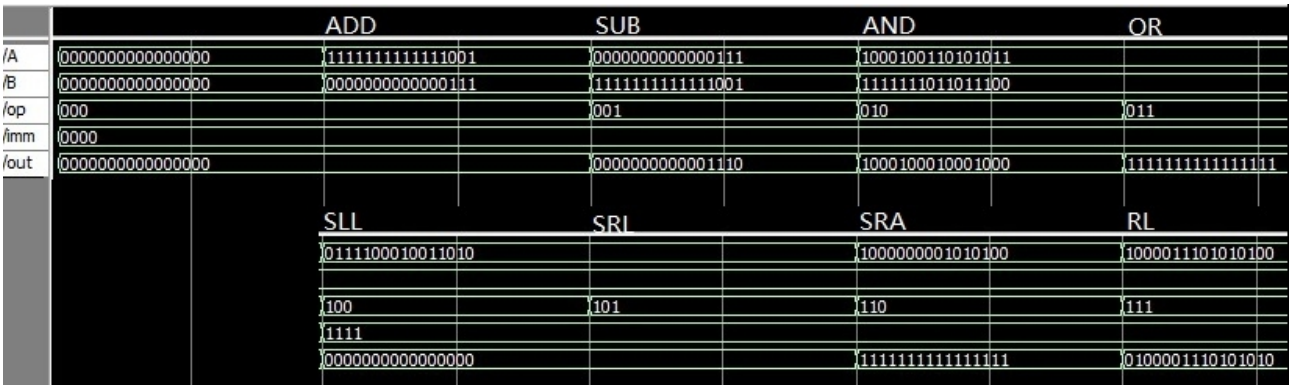

Figure 2: alu Testbench Simulations Result

For ADD operation, an overflow condition is calculated correctly.
For SUB operation, an underflow condition is correct.
Three 15-bit shift operations and one rotation is display properly.

4 Conclusions and Future Work

The ALU design works properly based on two testbenchs testing results.
During the Verilog implementation section, input A and B must be declared as *'signed'*. Otherwise, A and B will be performed as unsigned integer by default, which will result a wrong answer after SRA operation.
For this report, we use concatenation operation to implement the rotation. RL implementation methods are diverse. For instance, a for loop combined with concatenation operation can reduce the memory usage of rotation.
One improvement, which can be implemented, is the approaching a harder versions of ALU, by applying more efficient optimization and operations. Another one is the combination with register files and later laboratory implementations.