Northeastern University

Department of Electrical and Computer Engineering

EECE2323: **Digital Systems Design Lab**

Lecturer: **Dr. Emad Aboelela**

TAs:
Ke Chen
Linbin Chen

Lab # 1-3: **8-Bit Adder, Partial Arithmetic Logic Unit, and Arithmetic Logic Unit**

Group # **???**:
Jin Hyeong Kim,
Timothy VanSlyke

Semester: Spring 2018
Date: February 12, 2018
Lab Session: Tuesday, 1:00PM
Lab Location: 9 Hayden Hall, Northeastern University,
Boston, MA 02115

Contents

1 Introduction

The objective of the series of Lab 1, 2, and 3 were to become familiarized with Verilog hardware descriptions (Xilinx) and use Verilog code to implmenet small features on Zedboard hardware. The goals of the following experiments were to implement a simple 8-bit adder, a partial arithmetic logic unit (ALU), and a full ALU by extending partial ALU. The Verilog code was first written and tested in the Xilinix platform, and then a bitstream was be generated for the hardware testing on the Zedboard, and functionality was verified on the board.

2 Design Approach

2.1 Logic Design in Verilog Code

The purpose of an 8-bit adder was to have two inputs of signed 8-bit and produce a signed 8-bit output and an overflow. Overflow was calculated using:

$$ovf = (f[7]! = a[7]) \quad \&\& \quad (f[7]! = b[7]) \tag{1}$$

Appendix A shows details of the code used.

The partial ALU (PALU) had a 2-bit selector and two 8-bit inputs, and the 2-bit selector determined how the output is deteremind. If the selector is 00, the result would be the sum of two 8-bit inputs. If it is 01, the output will be a complement of input b. If it is 10, AND operator logic will apply to the inputs. Finally if it is 11, then OR operator logic will apply. Appendix B shows details of the code used.

Finally the full ALU had a 3-bit selector. Full ALU is an extension of partial ALU, as the selector now has 8 different states. In addition, the full ALU has one more output called *take_branch*. The selector from 000 to 011 is same as PALU. If the selector is 100, the output is an arithmetic shift right of an input a. If it is 101, the output is a logic shift left of an input a. If it is 110, the output f is 0, and the *take_branch* will be the result of $a == b$. If the selector is 111, the output f will also be 0, but the *take_branch* will be the result of $a \neq b$. The full code is shown in Appendix C.

2.2 Software Used

- Xilinx Vivado - Xilinx Vivado was used to perform all model simulations, including testbenches. Additionally, Xilinx Vivado provided the means with which our models were uploaded onto the ZedBoard FPGAs.
- Icarus Verilog - Icarus Verilog was used for rapid development and verification of several of the modules used in our experiments.

3 Results and Analysis

3.1 Design Simulation

All simulation tests were done using Xilinx Vivado's simulation environment. All Verilog modules that were tested were shown to satisfy their respective requirements.

3.1.1 8-Bit Adder

Shown here is the software simulation of the 8-Bit Adder module testbench. A 200ns delay is observed, followed by two trivial test cases where inputs a and b share no common set bits. Subsequent tests show that the adder module successfully handles cases where there are common set bits in both a and b. The final case shows the overflow condition being obeyed.

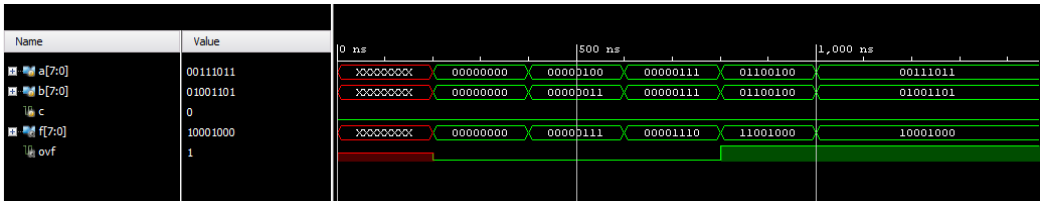

Figure 1: Simulation results for lab 1, **8-Bit Adder**.

3.1.2 Partial Arithmetic and Logic Unit

Shown here is the software simulation of the 8-Bit Partial ALU module testbench. The simulation begins with a series of test cases which demonstrate that the `ADD` ($s = 0$) test case behaves correctly, followed by tests for other operations. Subsequent tests show that the partial ALU module successfully handles other operations. The final case shows the overflow condition being obeyed for the `ADD` ($s = 0$) instruction.

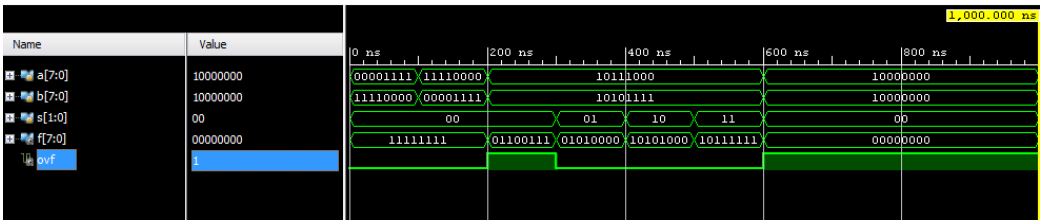

Figure 2: Simulation results for lab 2, **Partial Arithmetic and Logic Unit**.

3.1.3 Arithmetic and Logic Unit

Shown here is the software simulation of the 8-Bit ALU module testbench. The simulation begins with a series of test cases which demonstrate that the `ADD` ($s = 0$) test case behaves correctly, including a test for the `ovf` bit. Subsequent tests show that the ALU module successfully handles operations 1 through 5. The final four cases show the behavior of the `BEQ` and `BNE` instructions. In order, the four cases show that:

- `take_branch` is not set when the instruction is `BEQ` and $a \neq b$.
- `take_branch` is set when the instruction is `BNE` and $a \neq b$.
- `take_branch` is set when the instruction is `BEQ` and $a = b$.
- `take_branch` is not set when the instruction is `BNE` and $a = b$.

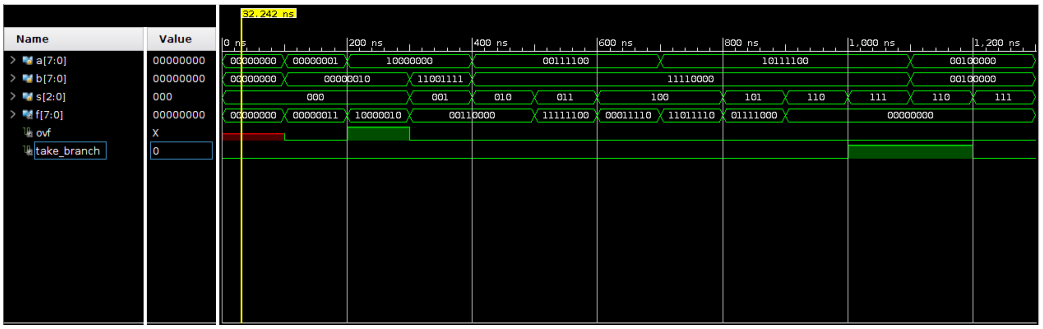

Figure 3: Simulation results for lab 3, **Arithmetic Logic Unit**.

3.2 Hardware Testing

Here we discuss our methods for testing our models in hardware. All hardware testing was done on a ZedBoard with an integrated Field-Programmable Gate Array (FPGA)[1].

3.2.1 8-Bit Adder

The eight-bit adder was tested using the ZedBoard's on-board switches to partially manipulate the inputs. Switches 0 through 3 control bits 0, 5, 6, and 7 of the first operand while switches 4 through 7 control bits 0, 5, 6, and 7 of the second operand. The hardware testing results can be seen in the 8-Bit Adder section of the appendix.

3.2.2 Partial Arithmetic and Logic Unit

The eight-bit partial ALU was tested using the ZedBoard's on-board switches to partially manipulate the inputs. Switches 0 and 1 control bits bits 0 and 1 of the instruction (`s`) input. Switches 2 through 4 control bits 0, 6, and 7 of the first operand while 5 through 7 control bits 0, 6, and 7 of the second operand. The hardware testing results can be seen in the 8-Bit Partial ALU section of the appendix.

3.2.3 Arithmetic and Logic Unit

The eight-bit partial ALU was tested using the Virtual Input/Ouput (VIO) functionality provided by Xilinx Vivado. This proved to be a more effecient way of testing the hardware behavior of our modules. A comprehensive suite of test cases was performed to ensure that all input and output bits were either set or unset during at least one test case. The The hardware testing results can be seen in the 8-Bit ALU section of the appendix.

[1] The particular FPGA used is the Xilinx XC7Z020CLG484-1.

4 Conclusions

The first three Lab sessions introduced the basics of Verilog and the functionalities of the hardware Zedboard. Implementation of logics with Verilog and verification of the logics through the FPGA on the ZedBoard were explored. The results came out successful. In future lab, the knowledge of thorough testing and effectively utilizing the software obtained from this lab would be useful.

Appendices

A 8-Bit Adder Module: `eightbit_adder.v`

```
module eightbit_palu(
    input signed [7:0] a,
    input signed [7:0] b,
    output reg [7:0] f,
    output reg ovf
  );
  always @(a, b) begin
    f = a + b;
    // overflow check for addition
    ovf = (f[7] != a[7]) && (f[7] != b[7]);
  end
endmodule
```

Figure 4: `eightbit_adder.v` implementing an 8-bit adder in Verilog.

B Partual ALU Verilog Module: `eightbit_palu.v`

```
module eightbit_palu(
    input signed [7:0] a,
    input signed [7:0] b,
    input [1:0] s,
    output [7:0] f,
    output ovf
  );
  always @(a, b, s) begin
    // operations on 'f'
    case(s)
      0: f = a + b;
      1: f = ~b;
      2: f = a & b;
      3: f = a | b;
    endcase
    // overflow check for addition
    ovf = (s == 0) && (f[7] != a[7]) && (f[7] != b[7]);
  end
endmodule
```

Figure 5: `eightbit_palu.v` implementing an 8-bit partial ALU in Verilog.

C ALU Verilog Module: `eightbit_alu.v`

```
module eightbit_alu(
    input signed [7:0] a,
    input signed [7:0] b,
    input [2:0] s,
    output reg [7:0] f,
    output reg ovf,
    output reg take_branch
  );
  always @(a, b, s) begin
    // operations on 'f'
    case(s)
      0: f = a + b;
      1: f = ~b;
      2: f = a & b;
      3: f = a | b;
      4: f = a >>> 1;
      5: f = a << 1;
      6: f = 0;
      7: f = 0;
    endcase
    // overflow check for addition
    ovf = (s == 0) && (f[7] != a[7]) && (f[7] != b[7]);
    // 'take_branch' special cases
    take_branch = ((s == 6) && (a == b)) || ((s == 7) && (a != b));
  end
endmodule
```

Figure 6: `eightbit_alu.v` implementing an 8-bit ALU in Verilog.

D Hardware Test Results for Lab 1: 8-Bit Adder

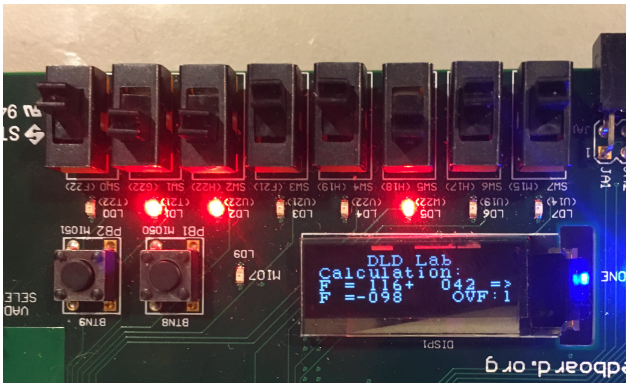

Figure 7: First test case.

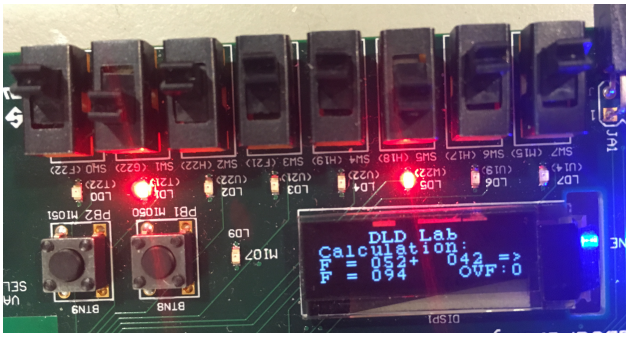

Figure 8: Second test case.

E Hardware Test Results for Lab 2: Partial Arithmetic Logic Unit

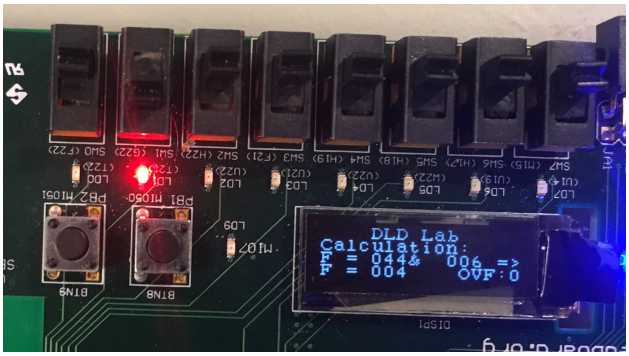

Figure 9: First test case.

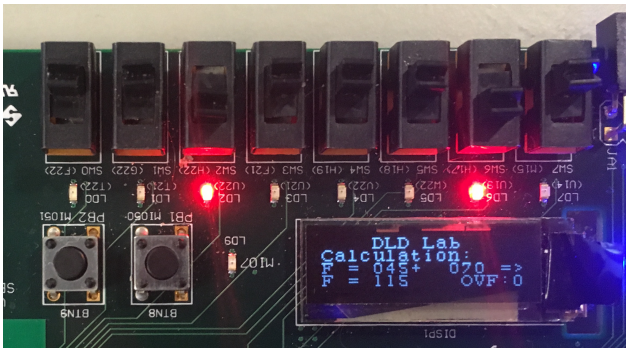

Figure 10: Second test case.

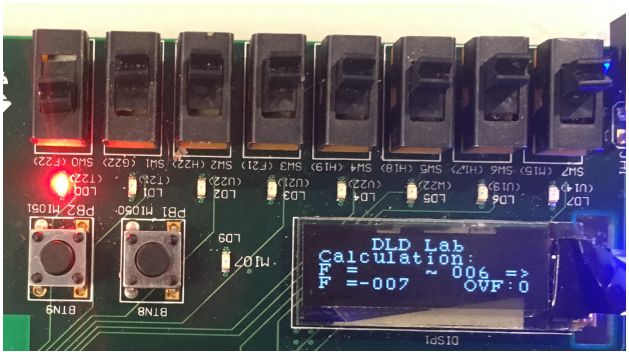

Figure 11: Third test case.

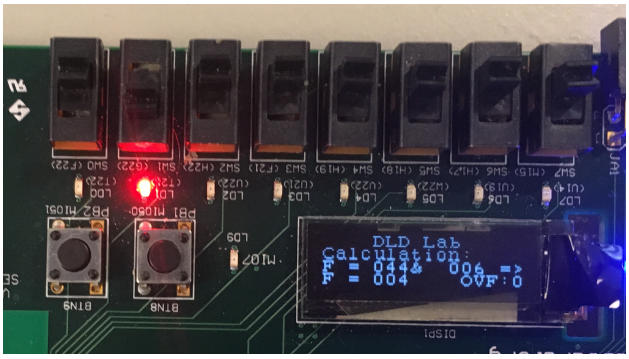

Figure 12: Fourth test case.

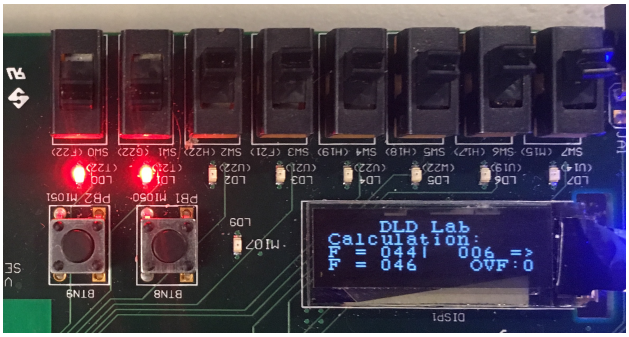

Figure 13: Fifth test case.

F Hardware Test Results for Lab 3: Arithmetic Logic Unit

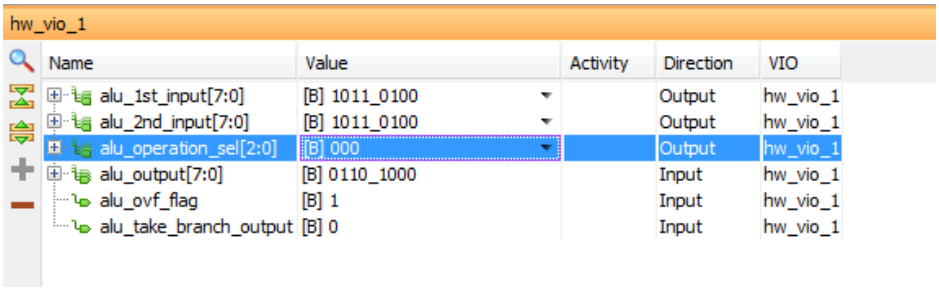

Figure 14: First test case – $a + b$.

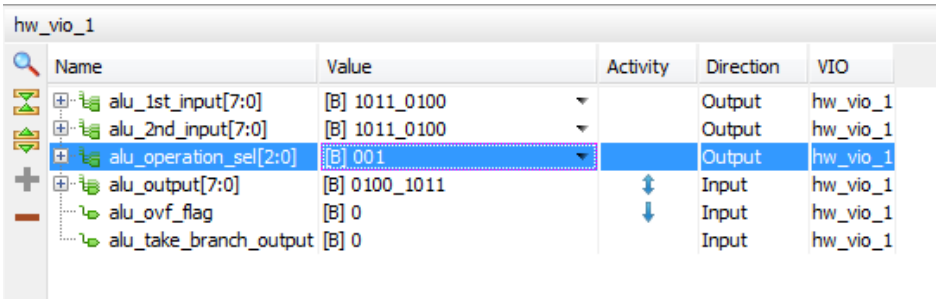

Figure 15: Second test case – $\tilde{}b$.

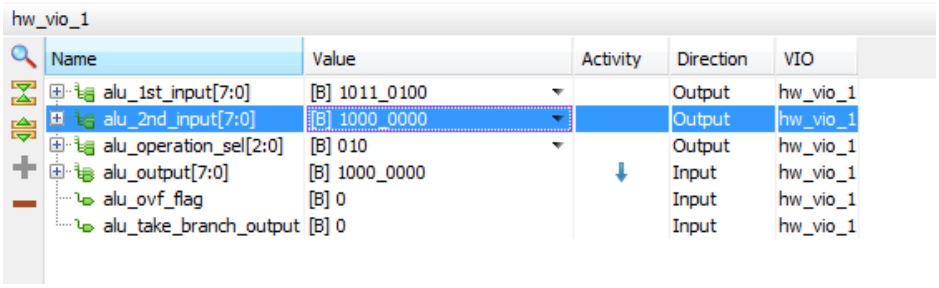

Figure 16: Third test case – $a \cdot b$.

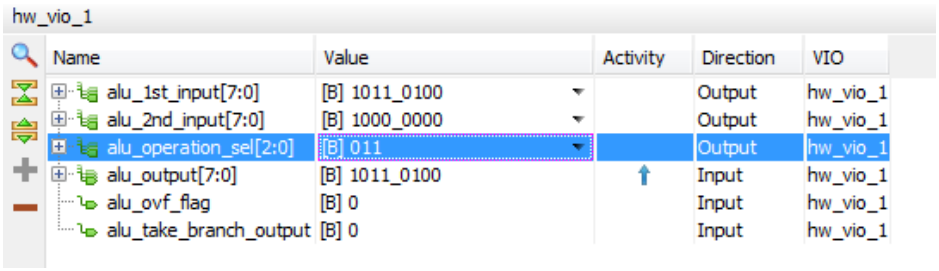

Figure 17: Fourth test case – $a|b$.

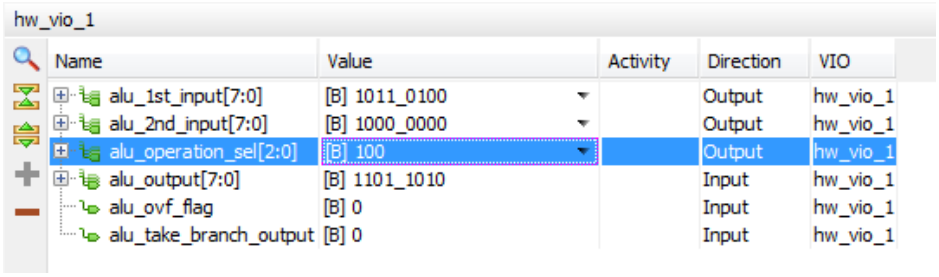

Figure 18: Fifth test case – $a >>> 1$.

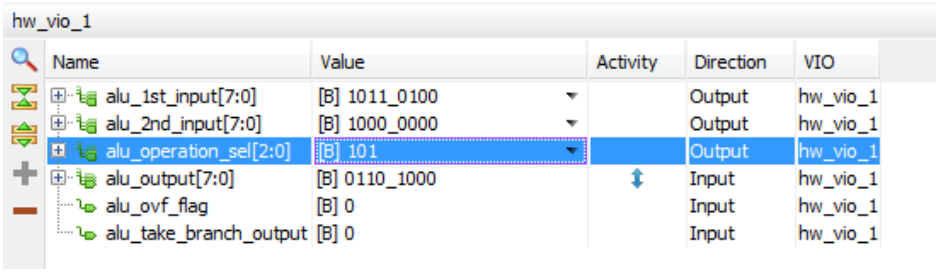

Figure 19: Sixth test case – $a << 1$.

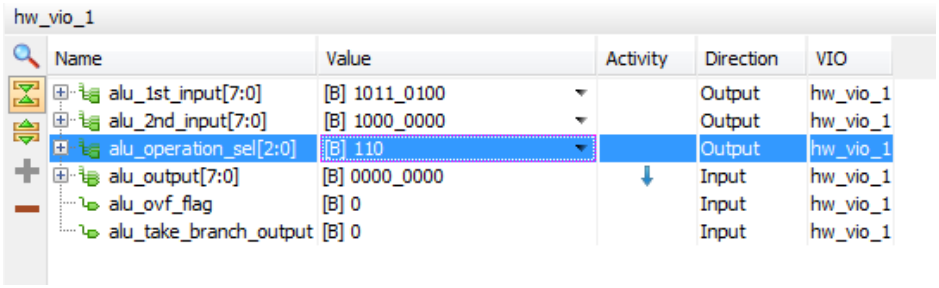

Figure 20: Seventh test case – $beq(a, b) \quad (a \neq b)$.

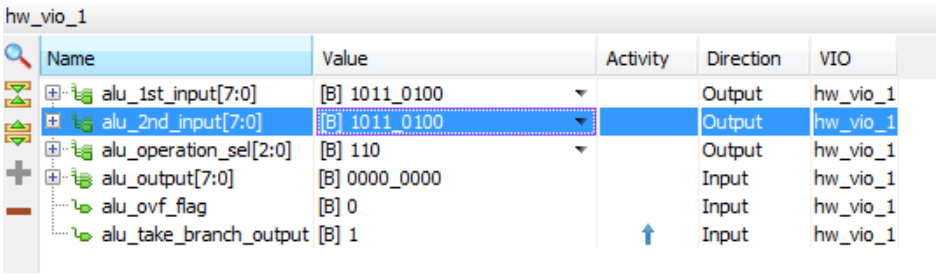

Figure 21: Eighth test case – $beq(a, b) \quad (a = b)$.

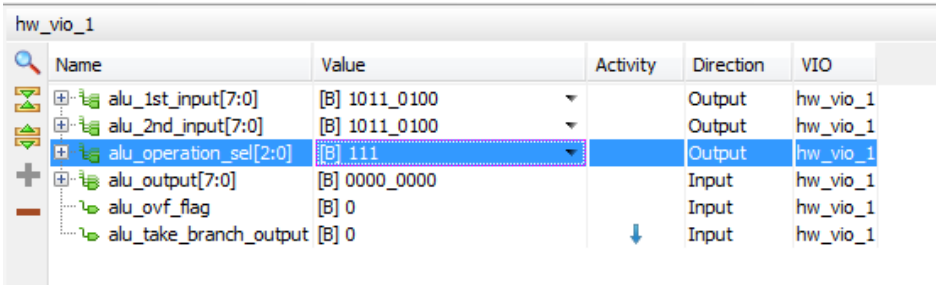

Figure 22: Ninth test case – $bne(a, b) \quad (a = b)$.