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 goal of the following experiments is implement a simple 8-bit Arithmetic Logic Unit (ALU) using the Verilog Hardware Description Language (HDL). This implementation will be tested both in a software simulation use Xilinx Vivado, and in hardware on a ZedBoard with an integrated Field-Programmable Gate Array (FPGA)[1]. In our experiments, we bootstrap our design up from a simple adder circuit to an ALU capable of working with 8-bit integers, providing an instruction set of size 8.

By exploring this process, we wish to explore the basics of digital design by providing a minimal example of an ALU.

Expand on this.

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

2 Design Approach

2.1 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.

2.2 Logic Design in Verilog Code

Since both the partial partial ALU and full ALU have an overflow (`ovf`) output bit that depends on the computed result of the ALU's `ADD` (`s`=0) operation, blocking assignment is used to compute `f` and `ovf`. A non-blocking assignment would allow incorrect behavior by not obeying the dependency relationship between `f` and `ovf`. It is worth noting that the `take_branch` output bit in the full ALU could be set via a continuous assignment like so:

```
// requires 'take_branch' to be declared as a 'wire'
assign take_branch = ((s == 6) && (a == b)) || ((s == 7) && (a != b));
```

However, for the sake of consistency, we opt for a blocking assignment in the same `always` block as `f` and `ovf`.

The Verilog module definitions for the eight-bit adder, eight-bit partial ALU, and the eightbit ALU can respectively be found in the appendix sections A, B, and C.

Opcode dispatch is implemented in Verilog using a `case` statement which handles each instruction individually. The `take_branch` and `ovf` output bits in the eight-bit ALU are handled outside of the `case` statement, since they have consistent behavior in most cases. For example, the `ovf` bit is set using the following statement:

```
ovf = (s == 0) && (f[7] != a[7]) && (f[7] != b[7]);
```

This simplifies the `case` structure immensely; only `f` is mutated within it.

3 Results and Analysis

All

3.1 Design Simulation

3.1.1 8-Bit Adder

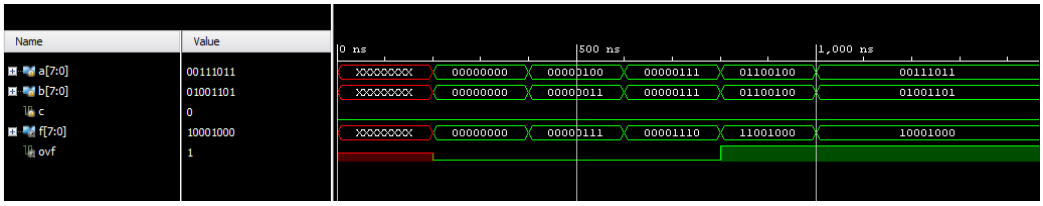

Figure 1: Simulation results for lab 1, **8-Bit Adder**.

3.1.2 Partial Arithmetic and Logic Unit

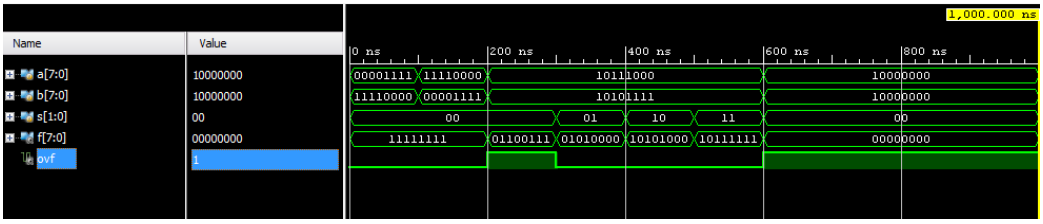

Figure 2: Simulation results for lab 2, **Partial Arithmetic and Logic Unit**.

3.1.3 Arithmetic and Logic Unit

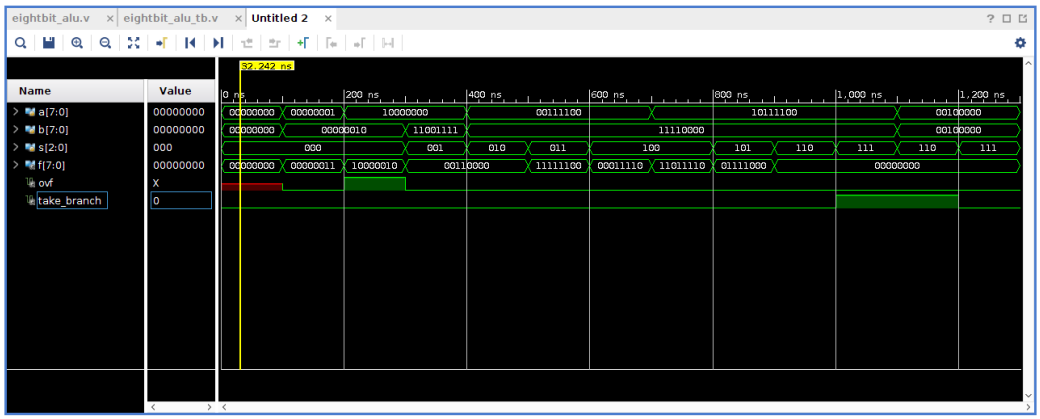

Figure 3: Simulation results for lab 3, **Arithmetic Logic Unit**.

3.2 Hardware Testing

3.2.1 8-Bit Adder

3.2.2 Partial Arithmetic and Logic Unit

3.2.3 Arithmetic and Logic Unit

4 Conclusions

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

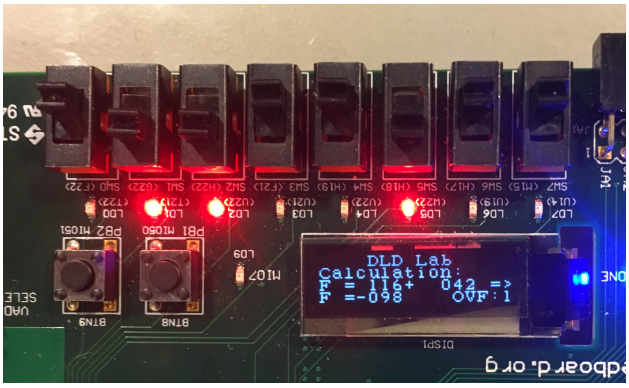

Figure 7: First test case.

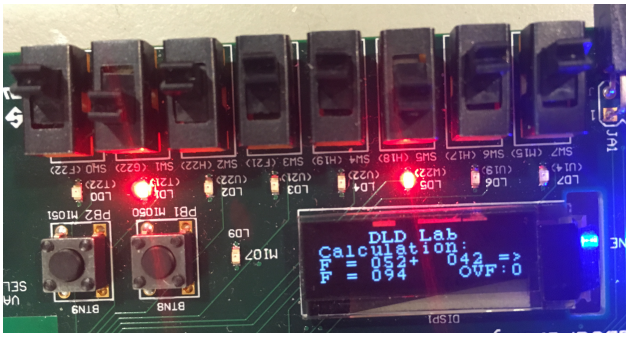

Figure 8: Second test case.

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

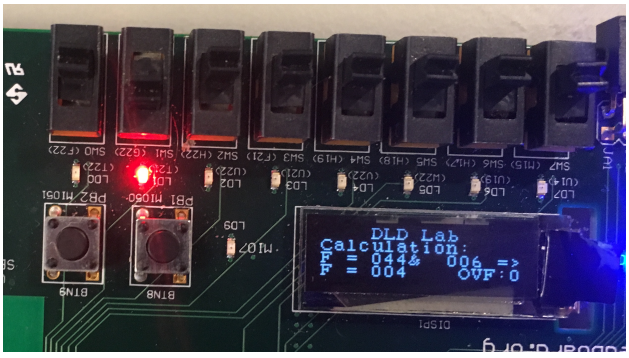

Figure 9: First test case.

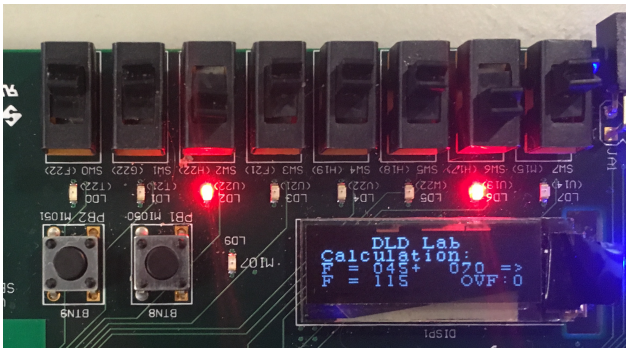

Figure 10: Second test case.

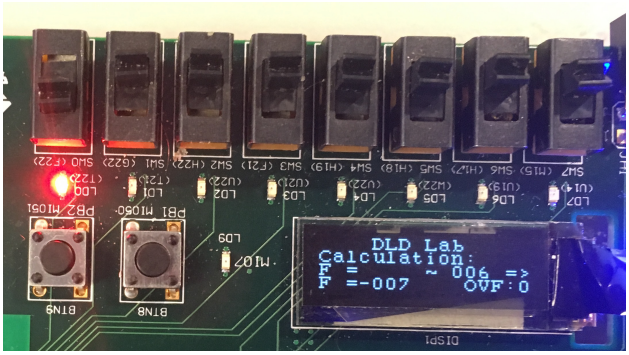

Figure 11: Third test case.

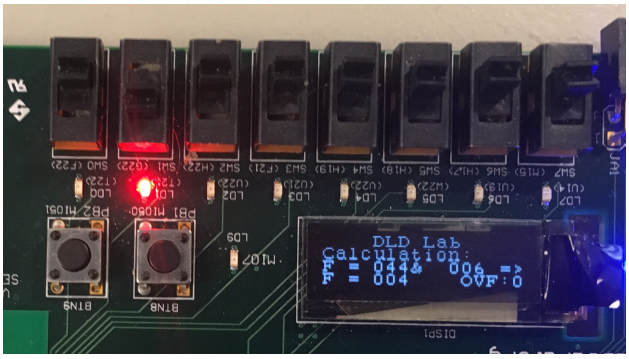

Figure 12: Fourth test case.

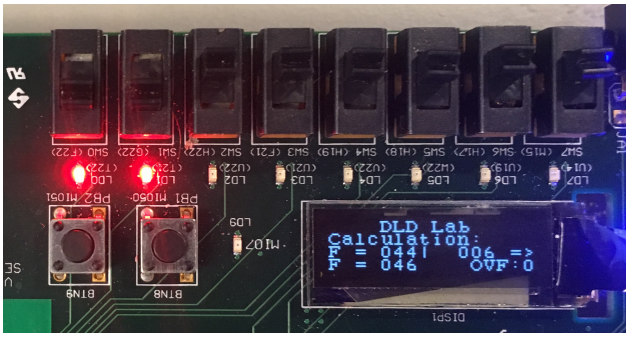

Figure 13: Fifth test case.

F Hardware Test Results for Lab 3: Arithmetic Logic Unit

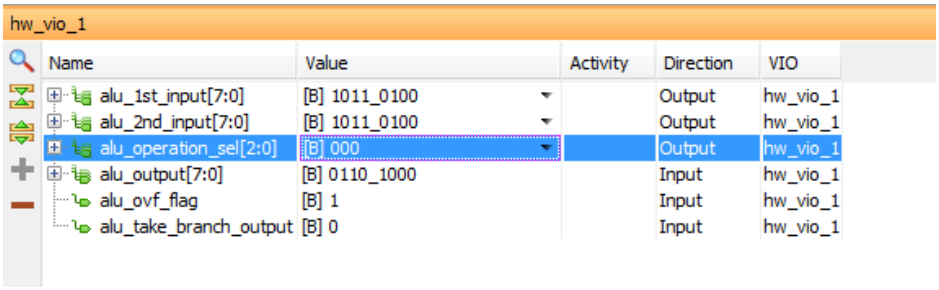

Figure 14: First test case – $a + b$.

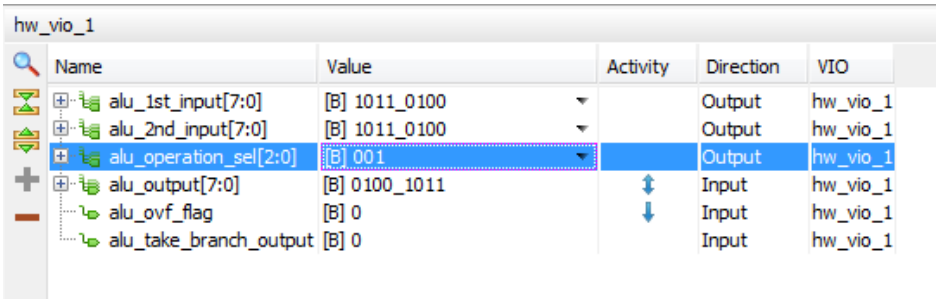

Figure 15: Second test case – $\tilde{}b$.

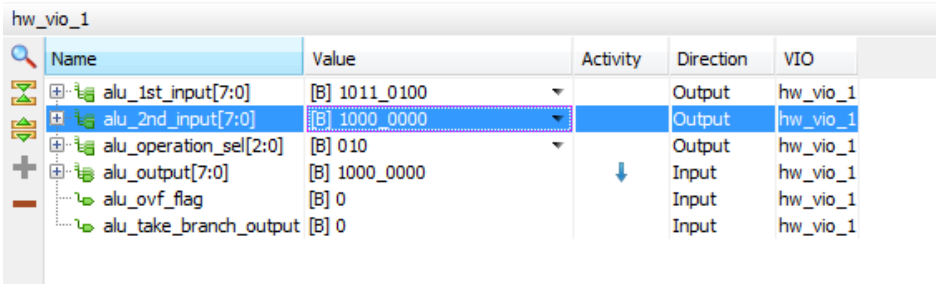

Figure 16: Third test case – $a \cdot b$.

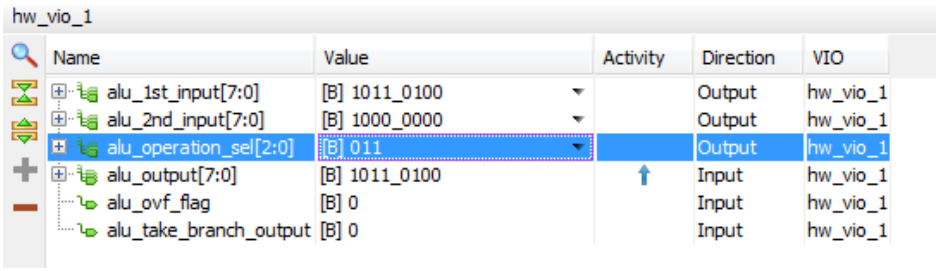

Figure 17: Fourth test case – $a|b$.

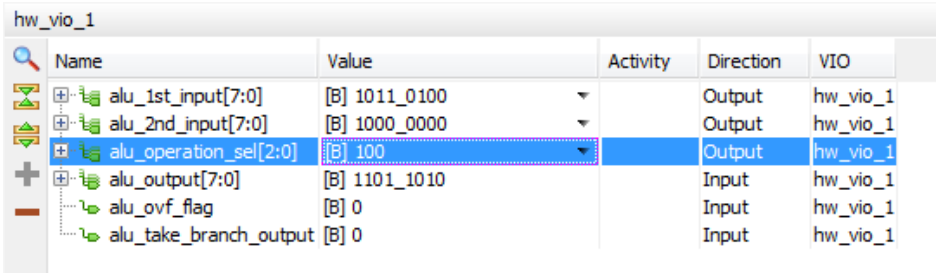

Figure 18: Fifth test case – $a >>> 1$.

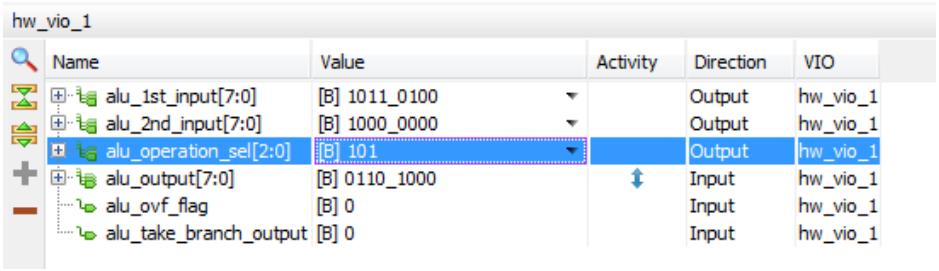

Figure 19: Sixth test case – $a << 1$.

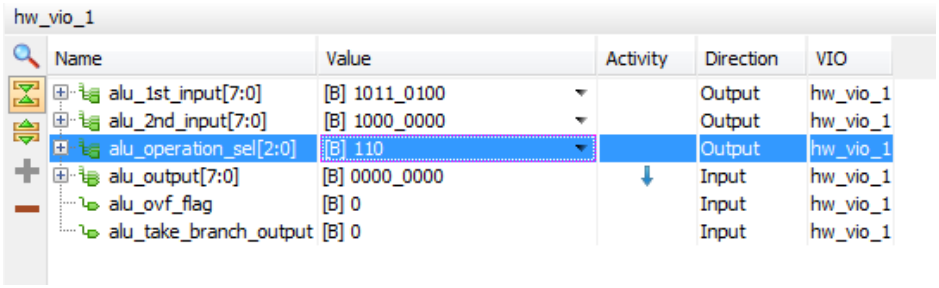

Figure 20: Seventh test case – $beq(a, b) \quad (a \neq b)$.

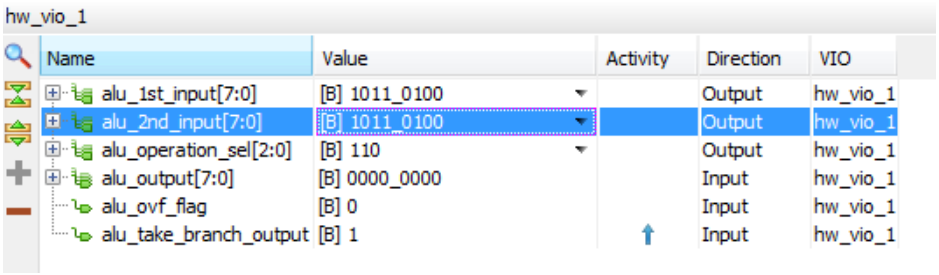

Figure 21: Eighth test case – $beq(a, b) \quad (a = b)$.

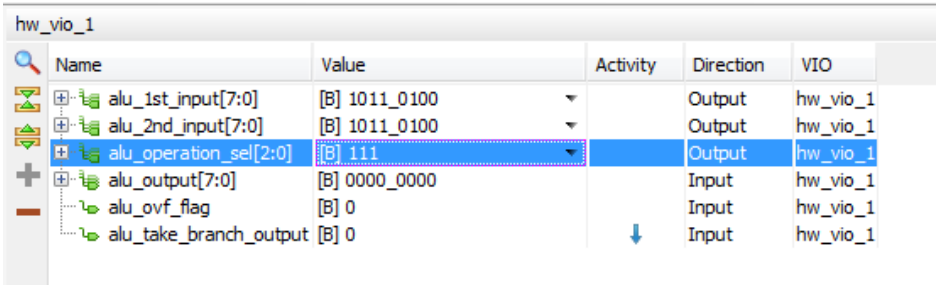

Figure 22: Ninth test case – $bne(a, b) \quad (a = b)$.