

**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 10, 2018  
Lab Session: Tuesday, 1:00PM  
Lab Location: 9 Hayden Hall, Northeastern University,  
Boston, MA 02115

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Design Approach</b>	<b>3</b>
<b>3</b>	<b>Results and Analysis</b>	<b>4</b>
3.1	Design Simulation . . . . .	4
3.1.1	8-Bit Adder . . . . .	4
3.1.2	Partial Arithmetic and Logic Unit . . . . .	4
3.1.3	Arithmetic and Logic Unit . . . . .	4
3.2	Hardware Testing . . . . .	5
3.2.1	8-Bit Adder . . . . .	5
3.2.2	Partial Arithmetic and Logic Unit . . . . .	5
3.2.3	Arithmetic and Logic Unit . . . . .	5
<b>4</b>	<b>Conclusions</b>	<b>6</b>
<b>Appendices</b>		<b>7</b>
<b>A</b>	<b>Lab 1 8-Bit Adder Module</b>	<b>7</b>
<b>B</b>	<b>Lab 2 Partial ALU Verilog Module</b>	<b>8</b>
<b>C</b>	<b>Lab 3 ALU Verilog Module</b>	<b>9</b>
<b>D</b>	<b>Hardware Test Results for Lab 1 - 8-Bit Adder</b>	<b>10</b>
<b>E</b>	<b>Hardware Test Results for Lab 2 - Partial Arithmetic Logic Unit</b>	<b>11</b>
<b>F</b>	<b>Hardware Test Results for Lab 3 - Arithmetic Logic Unit</b>	<b>13</b>

# **1 Introduction**

## **2 Design Approach**

### 3 Results and Analysis

#### 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

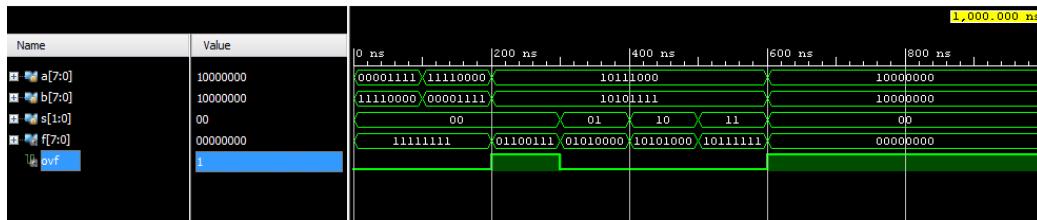


Figure 2: Simulation results for lab 2, **Partial Arithmetic and Logic Unit**.

##### 3.1.3 Arithmetic and Logic Unit

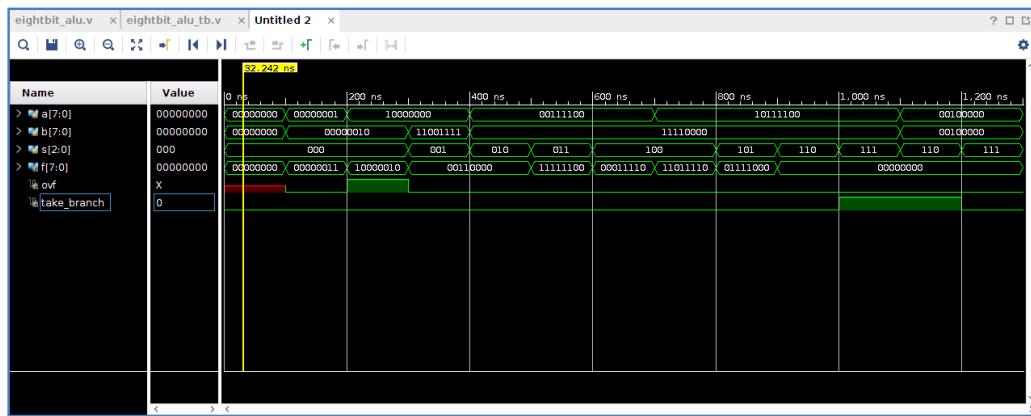


Figure 3: Simulation results for lab 3, Arithmetic Logic Unit.

## 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 [7:0] f,
    output ovf
);
// module-local registers
reg [7:0] f_reg, ovf_reg;
// continuous assignment to outputs from module-local registers
assign f = f_reg;
assign ovf = ovf_reg;
always @(a, b) begin
    f_reg = a + b;
    // overflow check for addition
    ovf_reg = (s == 0) && (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
);
// module-local registers
reg [7:0] f_reg, ovf_reg;
// continuous assignment to outputs from module-local registers
assign f = f_reg;
assign ovf = ovf_reg;
always @(a, b, s) begin
    // operations on 'f'
    case(s)
        0: f_reg = a + b;
        1: f_reg = ~b;
        2: f_reg = a & b;
        3: f_reg = a | b;
    endcase
    // overflow check for addition
    ovf_reg = (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 [7:0] f,
    output ovf,
    output take_branch
);
// module-local registers
reg [7:0] f_reg, ovf_reg, take_branch_reg;
// continuous assignment to outputs from module-local registers
assign f = f_reg;
assign ovf = ovf_reg;
assign take_branch = take_branch_reg;
always @(a, b, s) begin
    // operations on 'f'
    case(s)
        0: f_reg = a + b;
        1: f_reg = ~b;
        2: f_reg = a & b;
        3: f_reg = a | b;
        4: f_reg = a >>> 1;
        5: f_reg = a << 1;
        6: f_reg = 0;
        7: f_reg = 0;
    endcase
    // overflow check for addition
    ovf_reg = (s == 0) && (f[7] != a[7]) && (f[7] != b[7]);
    // 'take_branch' special cases
    take_branch_reg = ((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

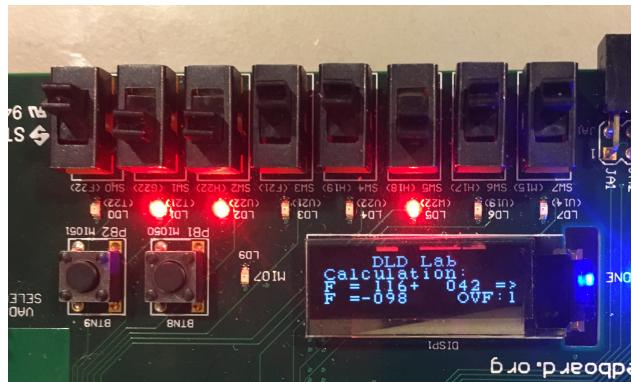


Figure 7: First test case.

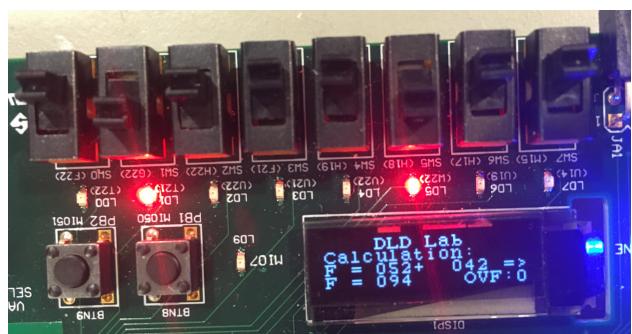


Figure 8: Second test case.

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

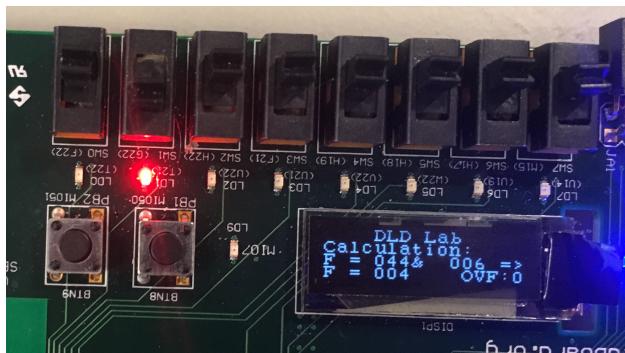


Figure 9: First test case.

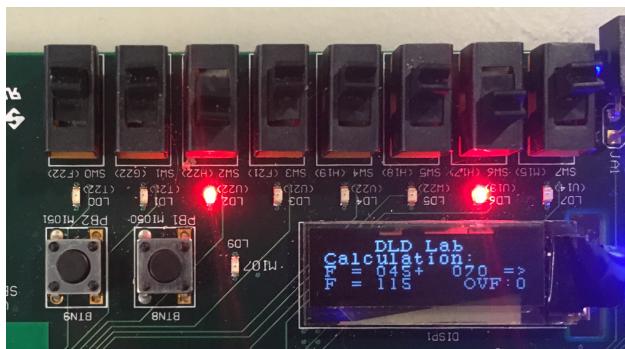


Figure 10: Second test case.



Figure 11: Third test case.

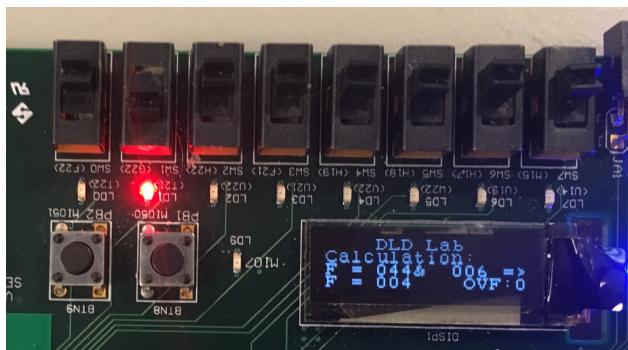


Figure 12: Fourth test case.

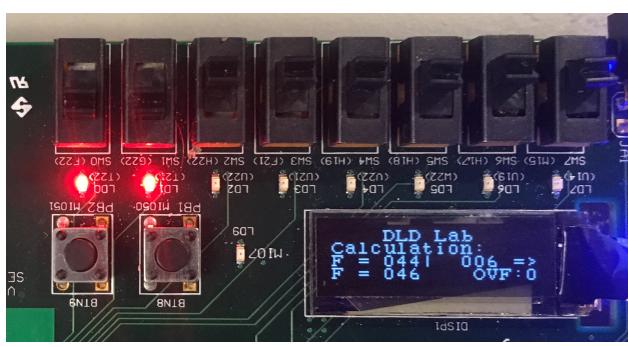


Figure 13: Fifth test case.

## F Hardware Test Results for Lab 3: Arithmetic Logic Unit

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 000		Output	hw_vio_1
alu_output[7:0]	[B] 0110_1000		Input	hw_vio_1
alu_ovf_flag	[B] 1		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

Figure 14: First test case –  $a + b$ .

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 001		Output	hw_vio_1
alu_output[7:0]	[B] 0100_1011		Input	hw_vio_1
alu_ovf_flag	[B] 0		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

Figure 15: Second test case –  $\sim b$ .

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1000_0000		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 010		Output	hw_vio_1
alu_output[7:0]	[B] 1000_0000		Input	hw_vio_1
alu_ovf_flag	[B] 0		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

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

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1000_0000		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 011		Output	hw_vio_1
alu_output[7:0]	[B] 1011_0100		Input	hw_vio_1
alu_ovf_flag	[B] 0		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

Figure 17: Fourth test case –  $a|b$ .

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1000_0000		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 100		Output	hw_vio_1
alu_output[7:0]	[B] 1101_1010		Input	hw_vio_1
alu_ovf_flag	[B] 0		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

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

Name	Value	Activity	Direction	VIO
alu_1st_input[7:0]	[B] 1011_0100		Output	hw_vio_1
alu_2nd_input[7:0]	[B] 1000_0000		Output	hw_vio_1
alu_operation_sel[2:0]	[B] 101		Output	hw_vio_1
alu_output[7:0]	[B] 0110_1000		Input	hw_vio_1
alu_ovf_flag	[B] 0		Input	hw_vio_1
alu_take_branch_output	[B] 0		Input	hw_vio_1

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

hw_vio_1					
	Name	Value	Activity	Direction	VIO
[+]	alu_1st_input[7:0]	[B] 1011_0100	▼	Output	hw_vio_1
[+]	alu_2nd_input[7:0]	[B] 1000_0000	▼	Output	hw_vio_1
[+]	alu_operation_sel[2:0]	[B] 110	▼	Output	hw_vio_1
[+]	alu_output[7:0]	[B] 0000_0000	▼	Input	hw_vio_1
-	alu_ovf_flag	[B] 0		Input	hw_vio_1
-	alu_take_branch_output	[B] 0		Input	hw_vio_1

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

hw_vio_1					
	Name	Value	Activity	Direction	VIO
[+]	alu_1st_input[7:0]	[B] 1011_0100	▼	Output	hw_vio_1
[+]	alu_2nd_input[7:0]	[B] 1011_0100	▼	Output	hw_vio_1
[+]	alu_operation_sel[2:0]	[B] 110	▼	Output	hw_vio_1
[+]	alu_output[7:0]	[B] 0000_0000		Input	hw_vio_1
-	alu_ovf_flag	[B] 0		Input	hw_vio_1
-	alu_take_branch_output	[B] 1	↑	Input	hw_vio_1

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

hw_vio_1					
	Name	Value	Activity	Direction	VIO
[+]	alu_1st_input[7:0]	[B] 1011_0100	▼	Output	hw_vio_1
[+]	alu_2nd_input[7:0]	[B] 1011_0100	▼	Output	hw_vio_1
[+]	alu_operation_sel[2:0]	[B] 111	▼	Output	hw_vio_1
[+]	alu_output[7:0]	[B] 0000_0000		Input	hw_vio_1
-	alu_ovf_flag	[B] 0		Input	hw_vio_1
-	alu_take_branch_output	[B] 0	▼	Input	hw_vio_1

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