ZC - University of Science and Technology
Communications & Information Engineering Program
CIE 239: Digital Design and Computer Architecture

Fall 2023

Digital Arithmetic and Logic Unit

Project Overview

Team Members:

202200341	Abdulrahman Magdy	s-abdulrahman.abdulrahman@zewailcity.edu.eg
202201079	SalahDin Rezk	s-salahdin.rezk@zewailcity.edu.eg
202200285	Shehab Mohamed	s-shehab.mohamed@zewailcity.edu.eg

Contents

List of Figures

List of Tables

Abstract

The Digital Arithmetic Logic Unit (ALU) is a crucial component in digital systems, responsible for performing arithmetic and logical operations. This project focuses on designing and implementing a 4-bit ALU using System Verilog. The ALU will be capable of handling basic arithmetic operations such as addition and subtraction, as well as logical operations like AND, OR, and XOR.

1 Design

1.1 Requirements

Table 1: ALU Operations

Operation	Output (E)
OR	$E = A \vee B$
AND	$E = A \wedge B$
CMP	$E = \overline{B}$
ADD	$E = A + B$
SUB	$E = A - B$
XOR	$E = A \oplus B$
ASR	$E_{3:0} = A_3 A_2 A_1 A_0$
INC	$E = B + 1$

Table 2: ALU Inputs

Inputs	Description
$A_{3:0}$	First operand
$B_{3:0}$	Second operand
C_{in}	Carry-in
$S_{2:0}$	Operation selection

Table 3: ALU Outputs

Output	Description
$E_{3:0}$	Result
C_{out}	Carry-out
Z	Zero output
V	Overflow

1.2 Structure

The project is structured into two main components: the Arithmetic Unit and the Logic Unit. Inputs A and B both are 4-bit signals while selection S is 3-bit and C_{in} is one-bit. The output E is 4-bit while C_{out}, Z, and V are one-bit. Both units are connected to the same input and output signals. Figure 1 shows the flowchart of the ALU structure, while Figure 2 shows the schematic of the structure. Finally, Table 4 shows the Mux configuration for the ALU selection.

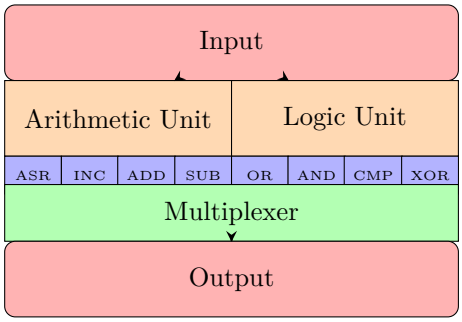

Figure 1: ALU Structure Flowchart

Table 4: MUX Configurations for ALU Selection

s_2	Operation
0	Arithmetic Unit
1	Logic Unit

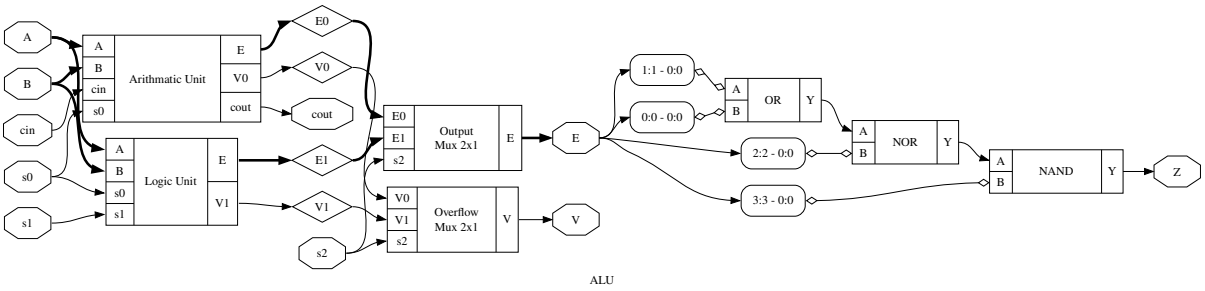

Figure 2: ALU Structure Diagram

1.3 Arithmetic Unit

1. Input Signals
   - The Arithmetic Unit receives two 4-bit inputs, A and B in the form of two's complement numbers.
   - The reason behind using two's complement is that it allows for addition and subtraction to be performed using the same circuitry.
2. 4-Bit Operations
   - The Arithmetic Unit comprises four Full Adders and 4x1 MUXs for versatile 4-bit operations.
   - The first set of MUXs facilitates selection among operations, including ASR, Increment, addition, or subtraction.
3. MUX Configurations
   - Four inputs to the first MUXs correspond to:

Table 5: MUX Configurations for Arithmetic Unit

S_0	C_{in}	Operation
0	0	ASR
0	1	$A - B$
1	0	$A + B$
1	1	$B + 1$

   - Another set of MUXs manage the shift of A or set A to 0 for incrementing B.
4. Bitwise Calculations
   - Each bit of the output is computed using a Full Adder and two MUXs.
   - The Full Adder generates two outputs: the corresponding bit of E and the carry-out, which serves as the carry-in for the subsequent Full Adder.
5. Overflow Detection
   - Overflow is detected by comparing the sign bits of A and B, and comparing the sign bit of E.

$$V = \overline{A_3}\,\overline{B_3}E_3 + A_3B_3\overline{E_3}.$$

   Generating the following truth table:

A_3	B_3	E_3	V
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Which is equivalent to XOR of (A, B) and E.

- The rationale behind this decision is that overflow occurs when the sign bits of A and B are the same, but the sign bit of E is different. That is a result of adding two positive numbers and getting a negative number, or adding two negative numbers and getting a positive number in the two's complement system.

6. Zero Output Determination

   - Output Z is determined using a NOR gate, signaling 1 only if all inputs are zeros.

$$Z = \overline{E_3}\,\overline{E_2}\,\overline{E_1}\,\overline{E_0}.$$

1.4 Logic Unit

1. Operation Selection:

   - The Logic Unit utilizes the least significant bits of S to decide between AND, OR, inversion, or XOR operations on A and B. Multiplexers are used to select the appropriate operation.

Table 6: MUX Configurations for Logic Unit

s_1	s_0	Operation
0	0	$A \wedge B$
0	1	$A \vee B$
1	0	$A \oplus B$
1	1	$\overline{B}$

2. Basic Gates:

   - The unit employs basic gates (AND, OR, NOT, XOR) to perform the selected logical operation.

3. Overflow detection:

   - Since overflow is only relevant for arithmetic operations, it is always set to zero inside the logic unit.

4. Zero Output Validation:

   - The output of the Logic Unit undergoes validation through a NOR gate to ascertain whether it is all zeros, triggering output Z accordingly.

$$Z = \overline{E_3}\,\overline{E_2}\,\overline{E_1}\,\overline{E_0}.$$

1.5 Testing

The functionality of the ALU is rigorously tested through the creation of a comprehensive test bench, focusing on a specific case where A = 0101 and B = 1101.

```
module tb_ALU();
    logic s2, s1, s0, cin;
    logic [3:0] A, B;
    logic cout, V, Z;
    logic [3:0] E;

    ALU test(s2, s1, s0, cin, A, B, cout, V, Z, E);

    initial begin
        A=4'b0101; B=4'b1101;                //Default values of A and B
        s2=0; s1=0; s0=0; cin=0; #100;       //Arithematic -> ASR -> A
        s2=0; s1=0; s0=0; cin=1; #100;       //Arithematic -> SUB -> A - B
        s2=0; s1=0; s0=1; cin=0; #100;       //Arithematic -> ADD -> A + B
        s2=0; s1=0; s0=1; cin=1; #100;       //Arithematic -> INC -> B + 1

        s2=1; s1=0; s0=0; cin=0; #100;       //Logic -> AND -> A & B
        s2=1; s1=0; s0=1; cin=0; #100;       //Logic -> OR -> A | B
        s2=1; s1=1; s0=0; cin=0; #100;       //Logic -> XOR -> A ^ B
        s2=1; s1=1; s0=1; cin=0; #100;       //Logic -> CMP -> ~B
    end
endmodule
```

2 Schematics

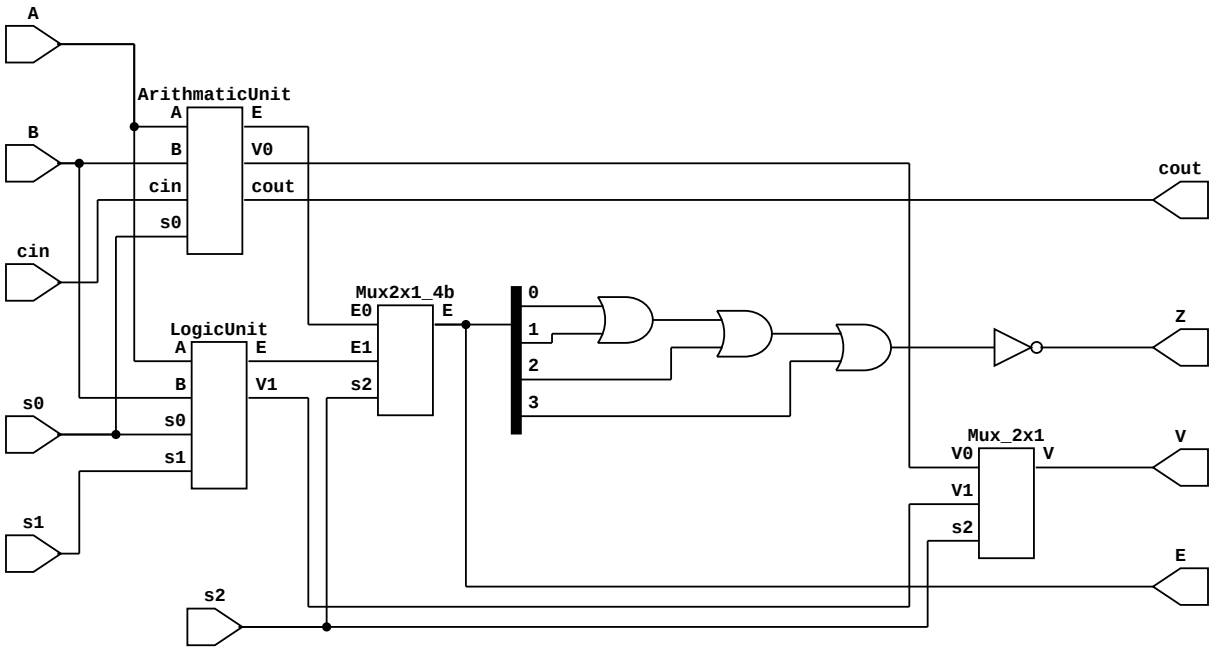

Figure 3: ALU Schematic

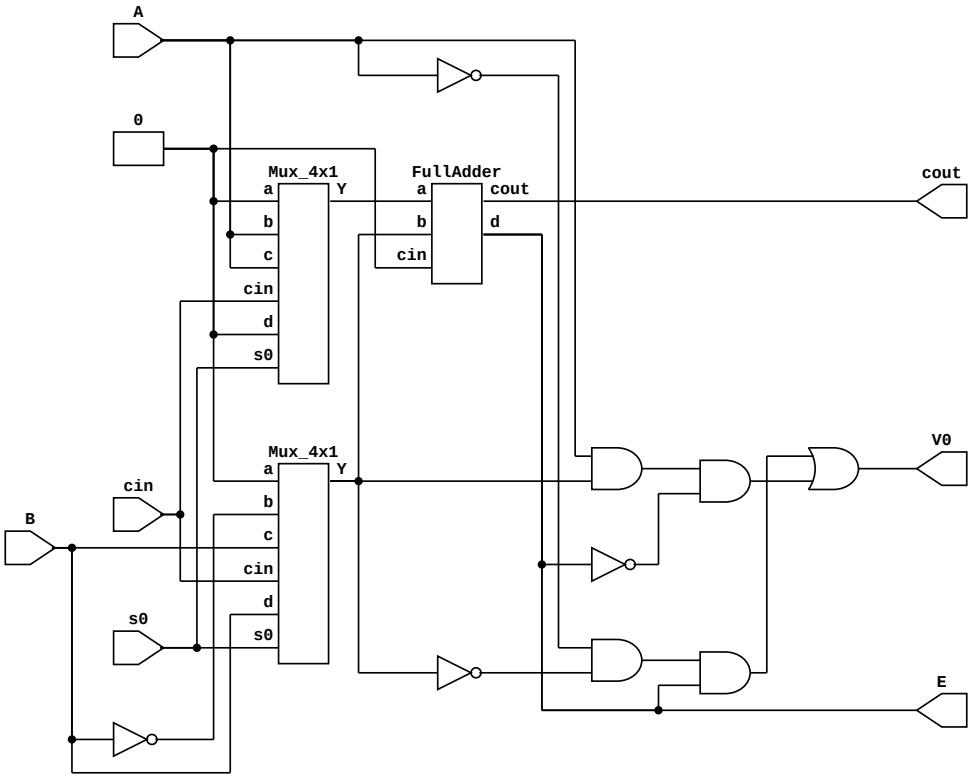

Figure 4: Arithmetic Unit Schematic

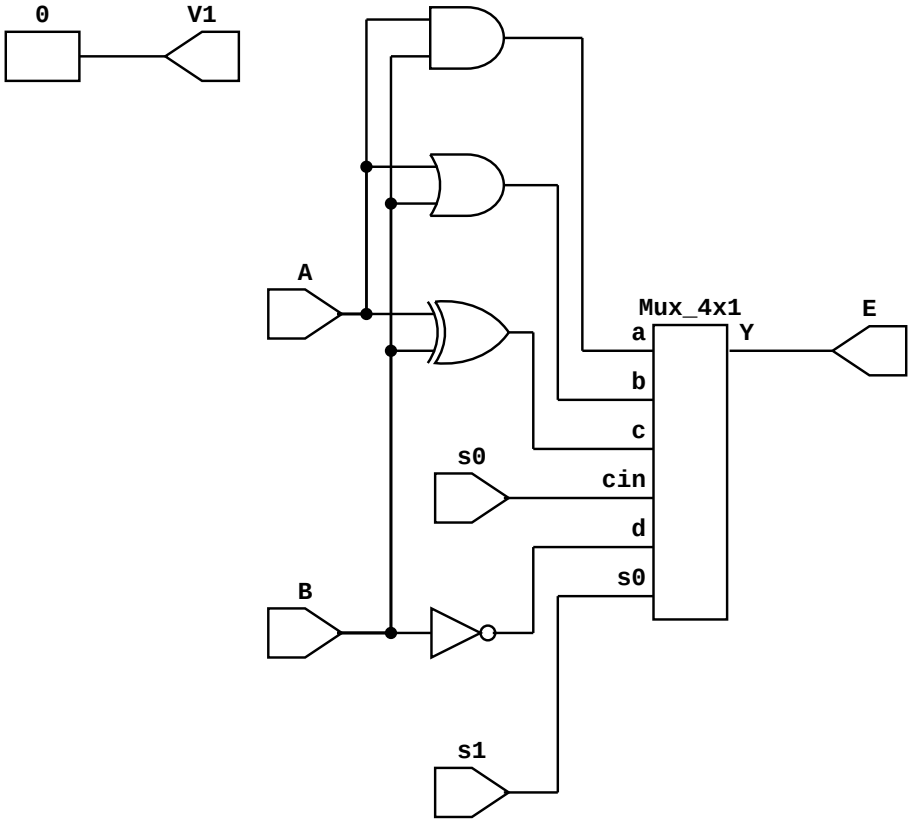

Figure 5: Logic Unit Schematic

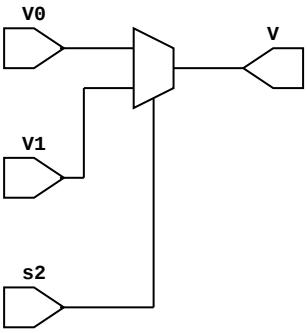

Figure 6: 2x1 MUX Schematic

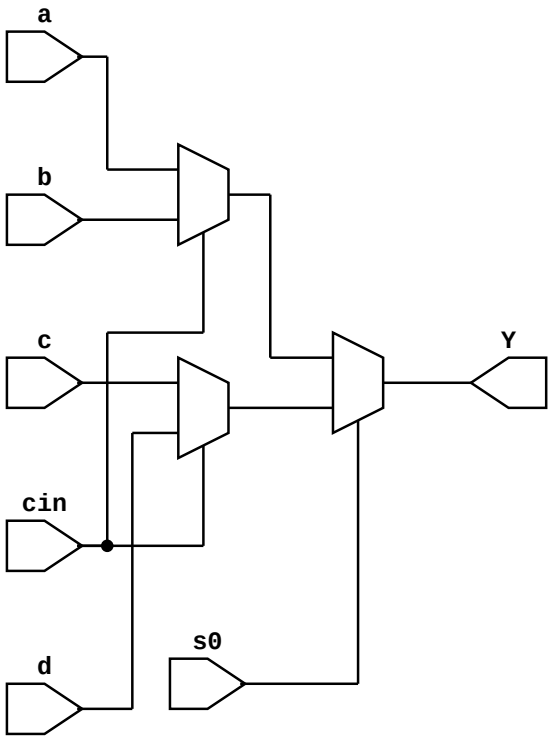

Figure 7: 4x1 MUX Schematic

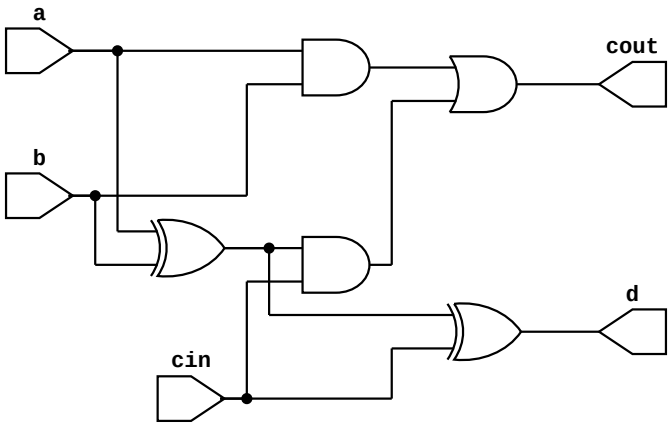

Figure 8: Full Adder Schematic

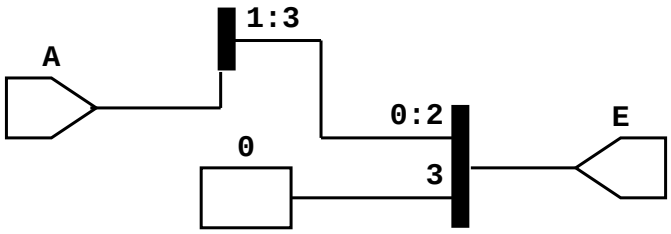

Figure 9: Arithmetic Shift Right Schematic

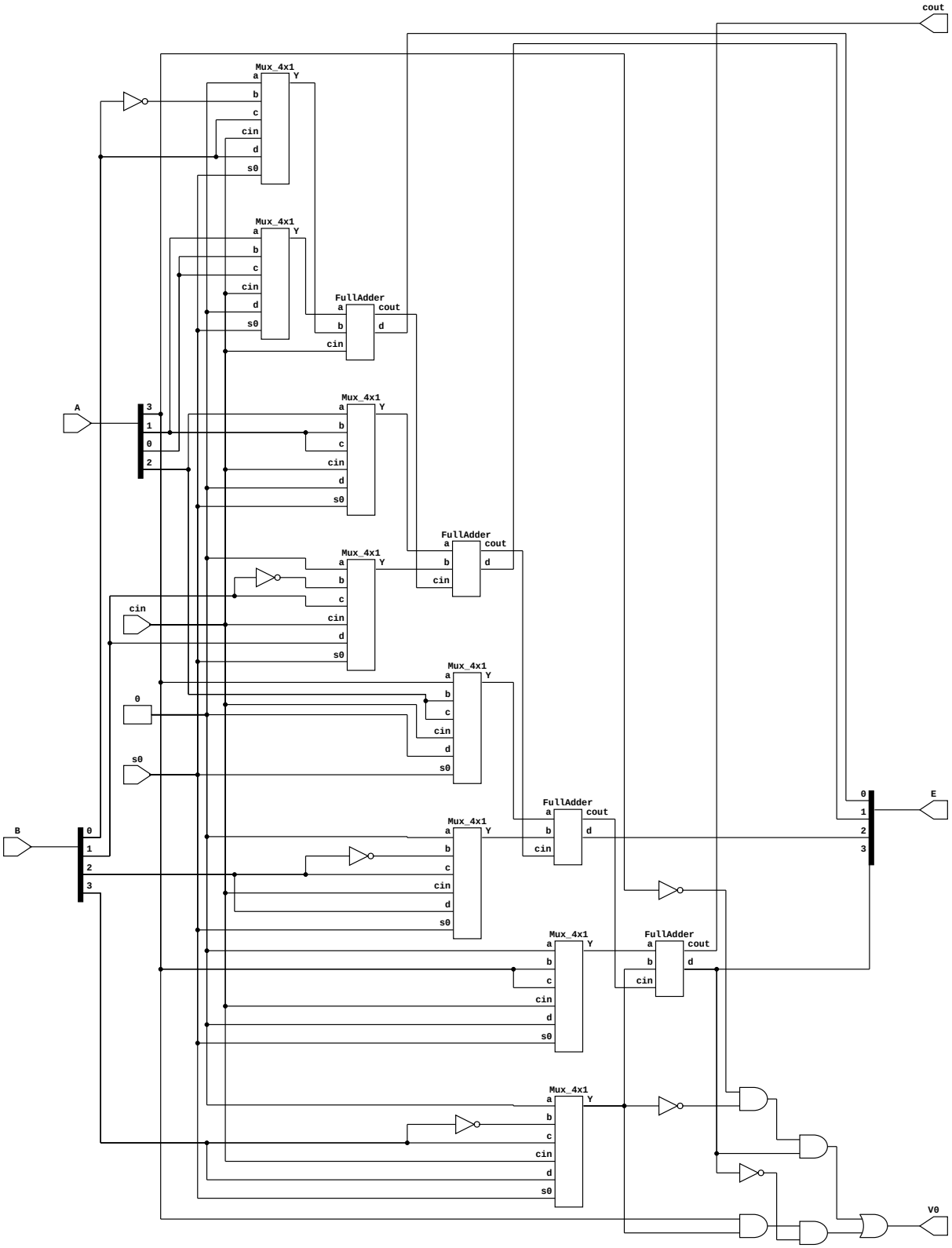

Figure 10: Complete Arithmetic Unit Schematic

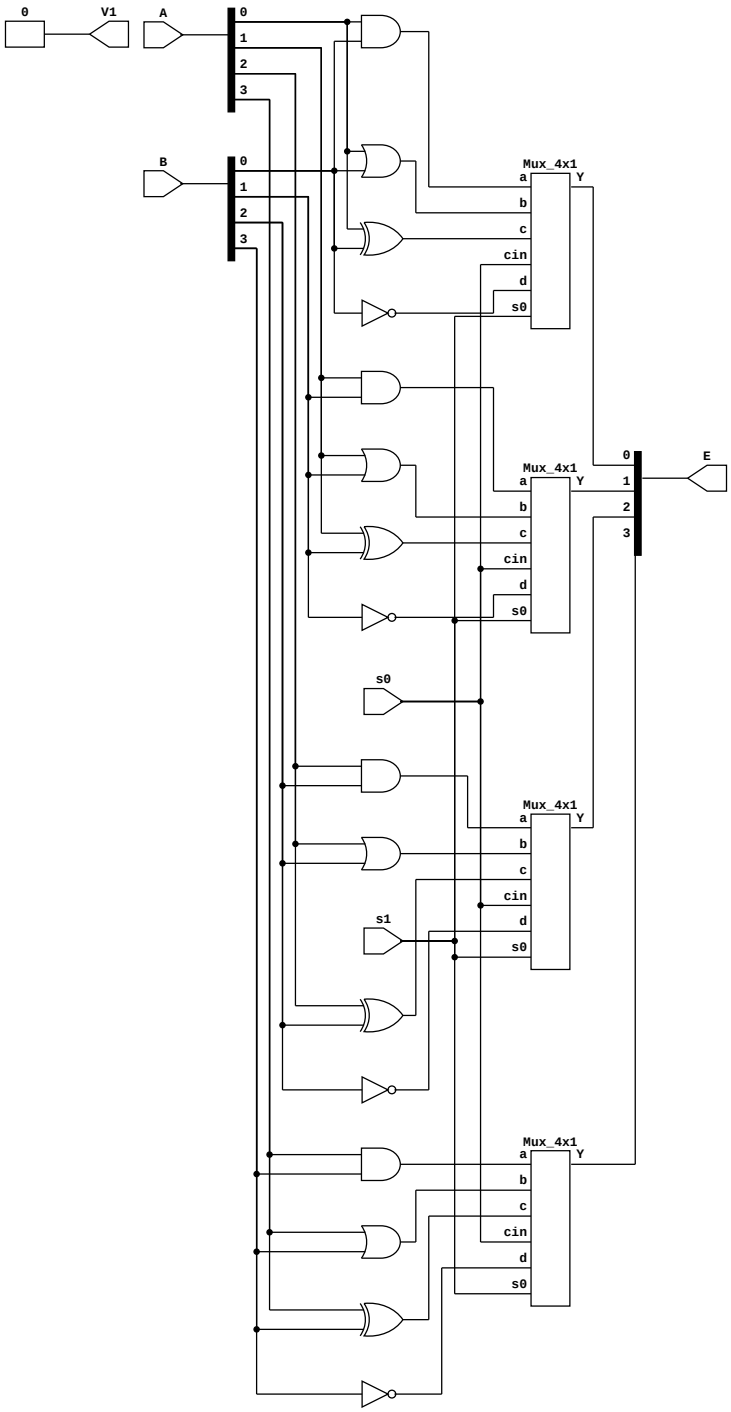

Figure 11: Complete Logic Unit Schematic

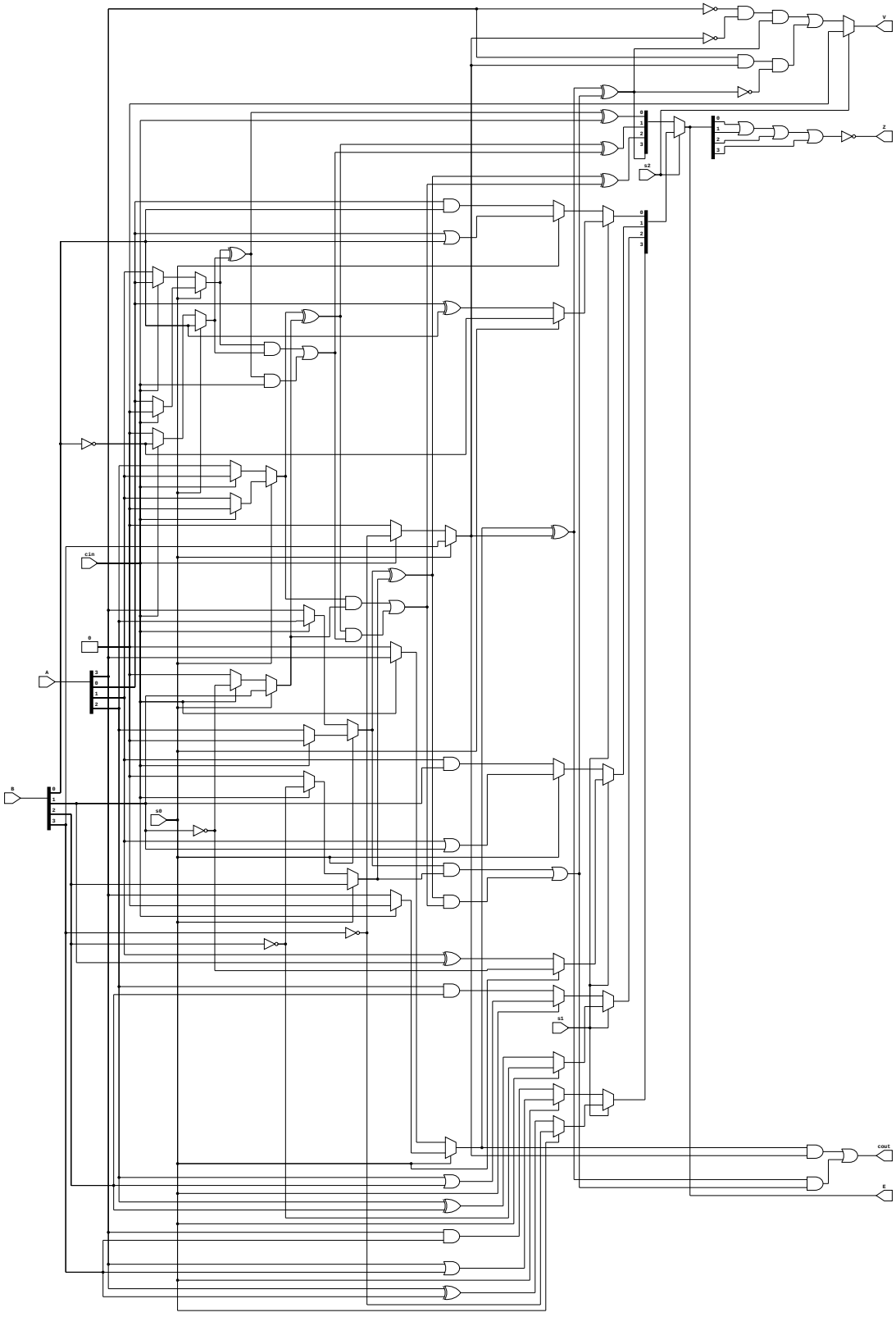

Figure 12: ALU Schematic (Flattened)

3 Code Implementation

```
module FullAdder(
    input logic a, b, cin,
    output logic d, cout
);
    // Full Adder logic
    assign d = a ^ b ^ cin;
    assign cout = (a & b) | ((a ^ b) & cin);
endmodule

module Mux_2x1(
    input logic s2,
    input logic V1, V0,
    output logic V
);
    // 2x1 Multiplexer for 1-bit values
    assign V = s2 ? V1 : V0;
endmodule

module Mux2x1_4b(
    input logic s2,
    input logic [3:0] E1, E0,
    output logic [3:0] E
);
    // 2x1 Multiplexer for 4-bit values
    assign E = s2 ? E1 : E0;
endmodule

module Mux_4x1(
    input logic s0, cin, a, b, c, d,
    output logic Y
);
    // 4x1 Multiplexer logic
    assign Y = s0 ? (cin ? d : c) : (cin ? b : a);
endmodule

module ArithmaticUnit(
    input logic s0, cin,
    input logic [3:0] A, B,
    output logic cout, V0,
    output logic [3:0] E
);
    // Arithmatic Unit
    logic [3:0] F;
    assign F = A >>> 1;

    // Bit-wise operations using Mux_4x1 and FullAdder
    logic b0, a0, c0;
    Mux_4x1 m0(s0, cin, 0, ~B[0], B[0], B[0], b0);
    Mux_4x1 m00(s0, cin, F[0], A[0], A[0], 0, a0);
    FullAdder f0(a0, b0, cin, E[0], c0);

    logic b1, a1, c1;
    Mux_4x1 m1(s0, cin, 0, ~B[1], B[1], B[1], b1);
    Mux_4x1 m11(s0, cin, F[1], A[1], A[1], 0, a1);
    FullAdder f1(a1, b1, c0, E[1], c1);

    logic b2, a2, c2;
    Mux_4x1 m2(s0, cin, 0, ~B[2], B[2], B[2], b2);
    Mux_4x1 m22(s0, cin, F[2], A[2], A[2], 0, a2);
    FullAdder f2(a2, b2, c1, E[2], c2);

```

```
    logic b3, a3;
    Mux_4x1 m3(s0, cin, 0, ~B[3], B[3], B[3], b3);
    Mux_4x1 m33(s0, cin, F[3], A[3], A[3], 0, a3);
    FullAdder f3(a3, b3, c2, E[3], cout);

    // Overflow condition
    assign V0 = ((~A[3]) & (~b3) & E[3]) | (A[3] & b3 & (~E[3]));
endmodule

module LogicUnit(
    input logic s1, s0,
    input logic [3:0] A, B,
    output logic V1,
    output logic [3:0] E
);
    // Logic Unit
    // 4x1 Multiplexers for bit-wise AND, OR, XOR, and NOT(B) operations
    Mux_4x1 M11(s1, s0, (A[0] & B[0]), (A[0] | B[0]), (A[0] ^ B[0]), ~B[0], E[0]);
    Mux_4x1 M22(s1, s0, (A[1] & B[1]), (A[1] | B[1]), (A[1] ^ B[1]), ~B[1], E[1]);
    Mux_4x1 M33(s1, s0, (A[2] & B[2]), (A[2] | B[2]), (A[2] ^ B[2]), ~B[2], E[2]);
    Mux_4x1 M44(s1, s0, (A[3] & B[3]), (A[3] | B[3]), (A[3] ^ B[3]), ~B[3], E[3]);

    // V1 is always assigned 0 in this module
    assign V1 = 0;
endmodule

module ALU(
    input logic s2, s1, s0, cin,
    input logic [3:0] A, B,
    output logic cout, V, Z,
    output logic [3:0] E
);
    // Arithmetic and Logic Unit (ALU)
    logic V0, V1;
    logic [3:0] E0, E1;

    // ArithmaticUnit and LogicUnit instances
    ArithmaticUnit Au(s0, cin, A, B, cout, V0, E0);
    LogicUnit Lu(s1, s0, A, B, V1, E1);

    // Z (Zero) flag is true if all bits in E are zero
    assign Z = ~(E[0] | E[1] | E[2] | E[3]);

    // Multiplexer for Overflow flag
    Mux_2x1 isOverflow(s2, V1, V0, V);

    // Multiplexer for 4-bit result E
    Mux2x1_4b out(s2, E1, E0, E);
endmodule
```

4 Simulation Outputs

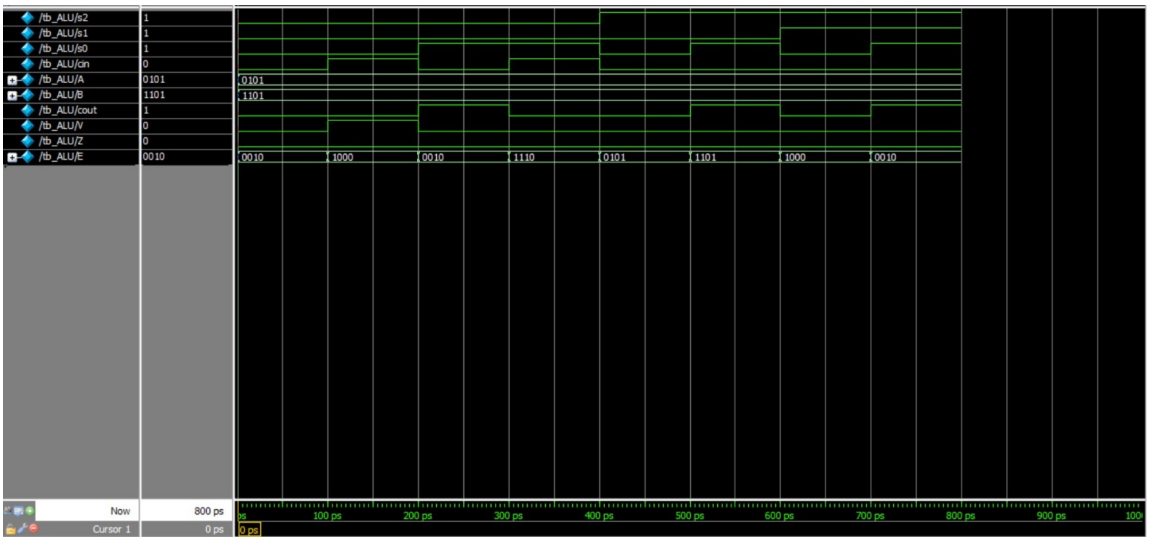

Figure 13: ALU Test Bench Simulation Output (ModelSim)