Group: Leftovers ALU Functions Putting It All Together

Luis Herrera, Philip Ervin
CS.3339 Computer Architecture
Texas State University

November 2023

1 Introduction

This step of the project combines the efforts described in steps 1 and 2. The goal of this step is to combine previously-created modules onto one virtual circuit and create a control unit to integrate them. For this control unit, operation codes to indicate which operation should be done for each input provided must be defined. With these operation codes and ALU, we have created. This report describes the control unit, operation codes, and overall circuit that comprise our ALU.

2 Verilog Code For ALU Functions

In this section, we will go over every Verilog module which was utilized in the final control circuit.

2.1 4 Bit AND Gate

The AND module takes two 4-bit inputs, A and B, and performs a bitwise AND operation between the corresponding bits of these inputs. It also produces a 4-bit output, Y

```verilog
module and_4bit(

    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = A & B;

endmodule
```

2.2 4 Bit NAND Gate

The NAND module takes two 4-bit inputs, A and B, and performs a bitwise NAND operation between the corresponding bits of these inputs. It also produces a 4-bit output, Y.

```verilog
module nand_4bit(
    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = ~(A & B); // 4-bit NAND operation

endmodule
```

2.3 4 Bit OR Gate

The OR module takes two 4-bit inputs, A and B, and performs a bitwise OR operation between the corresponding bits of these inputs. It also produces a 4-bit output, Y.

```
module or_4bit(
    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = A | B; // 4-bit OR operation

endmodule
```

2.4 4 Bit NOR Gate

The NOR module takes two 4-bit inputs, A and B, and performs a bitwise NOR operation between the corresponding bits of these inputs. It also produces a 4-bit output, Y.

```
module nor_4bit(
    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = ~(A | B); // 4-bit NOR operation

endmodule
```

2.5 4 Bit XOR Gate

The XOR module takes two 4-bit inputs, A and B, and performs a bitwise XOR operation between the corresponding bits of these inputs. It also produces a 4-bit output, Y.

```
module xor_4bit (
    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = A ^ B;

endmodule
```

2.6 4 Bit XNOR Gate

The XNOR module takes two 4-bit inputs, A and B, and performs a bitwise XNOR operation between the corresponding bits of these inputs.It also produces a 4-bit output, Y.

```
module xnor_4bit (
    input [3:0] A,
    input [3:0] B,
    output [3:0] Y
);

assign Y = ~(A ^ B);

endmodule
```

2.7 4 Bit NOT Gate

The NOT module takes a 4-bit input, A, and produces an inverted 4-bit output, Y.

```
module not_4bit (
    input [3:0] A,
    output [3:0] Y
);

assign Y = ~A;

```

```
endmodule
```

2.8 4 Bit Shifter Gate

The Shifter module takes two 4-bit inputs A and B, where B[3] determines the shift direction, B[2:1] specify the shift amount, and B[0] indicates whether the shift should be filled or emptied. The output Y is the result.

```
module newShifter (
    input wire [3:0] A,
    input wire [3:0] B,
    output wire [3:0] Y
);

  wire shift = B[3];                  // 1sh bit in B dictates direction
  wire [2:1] shift_amount = B[2:1]; // Bits 1 and 2 in B tell the amount
  wire fill = B[0];                   // 1ast bit in B is fill or full

  assign Y = (shift) ?
             ((fill) ? {A[3], A[3:1]} << shift_amount : A << shift_amount) :
             ((fill) ? {A[2:0], A[0]} >> shift_amount : A >> shift_amount);

endmodule
```

2.9 4 Bit Addition Circuit

The 4-bit addition operation It takes two 4-bit inputs, A and B, along with a carry-in signal Cin, and produces a 4-bit output Sum and a 4-bit output Cout which take the overflow.

```
module add_4bit(
    input wire [3:0] A,
    input wire [3:0] B,
    input wire Cin,
    output wire [3:0] Sum,
    output wire [3:0] Cout
);

```

```
assign {Cout, Sum} = A + B + Cin;

endmodule
```

2.10 4 Bit Subtraction Circuit

The 4-bit Subtraction operation accepts two 4-bit inputs, A and B, in addition to a carry-in signal Cin, and generates a 4-bit output Result 4-bit output Cout which take the overflow.

```
module sub_4bit (
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] Result,
    output [3:0] Cout
);

assign {Cout, Result} = A - B - Cin;

endmodule
```

2.11 4 Bit Multiplication Circuit

The 4-Bit Multiplication Circuit is designed to take two 4-bit inputs,, and generate a 4-bit product. It does this through a for loop, utilizing the AND operator to calculate partial products for each bit. The final product is then outputted into a 8 bit form named product.

```
module multi_4bit (
    input [3:0] A,
    input [3:0] B,
    output [7:0] Product
);

wire [7:0] partial_products[0:3];
wire [7:0] sum;

generate
    genvar i; // Declare 'i' for generate loop
    for (i = 0; i < 4; i = i + 1) begin : gen_mult
        // Generate the partial products by AND operation
   then shifts the result left by i bits.
```

```
        assign partial_products[i] = (A & (B[i] ? 4'b1111 :
    4'b0000)) << i;
    end
endgenerate

// Calculate the sum of the partial products
assign sum = partial_products[0] + partial_products[1] +
    partial_products[2] + partial_products[3];

assign Product = sum; // Product is the sum of the partial
    products

endmodule
```

3 Control Circuit

3.1 Control Circuit Verilog Code

The Verilog module, named "integratedModule," is the Control Circuit which performs various 4-bit operations based on the opcode input. It includes modules for AND, NAND, OR, XOR, XNOR, NOR, NOT, SHIFTER, ADD, SUBTRACT, and MULTIPLY. The output Y return a 4-bit result. To accommodate the 8-bit multiplication results we created a secondary output Y-8bit. The Cin (carry-in) input and Cout (carry out) output is for all modules but only effects the addition, subtraction operations.

The opcode function as such:

- Input 0000 (Bit 0) is responsible for the AND operation.
- Input 0001 (Bit 1) is responsible for the NAND operation.
- input 0010 (Bit 2) is responsible for the OR operation.
- input 0011 (Bit 3) is responsible for the XOR operation.
- input 0100 (Bit 4) is responsible for the XNOR operation.
- input 0101 (Bit 5) is responsible for the NOR operation.
- input 0110 (Bit 6) is responsible for the NOT operation.

- input 0111 (Bit 7) is responsible for the ADD operation.
- input 1000 (Bit 8) is responsible for the SUBTRACTION operation.
- input 1001 (Bit 9) is responsible for the MULTIPLICATION operation.
- input 1010 (Bit 10) is responsible for the SHIFTER operation.

```
module integratedModule (
    input [3:0] A,
    input [3:0] B,
    input Cin,
    input [3:0] opcode, // 4-bit opcode to select the
   operation
    output [3:0] Cout,
    output [3:0] Y,
    output [7:0] Y_8bit
);

  // Internal signals
  wire [3:0] Y_and, Y_nand, Y_or, Y_xor, Y_xnor, Y_nor, Y_not
   , Y_shifter,Y_add,Y_sub;

  wire [7:0] Y_multi; // Separate wire for add, sub and
   multiplication result

  // Module instantiations
  and_4bit and_inst (
    .A(A),
    .B(B),
    .Y(Y_and)
  );

  nand_4bit nand_inst (
    .A(A),
    .B(B),
    .Y(Y_nand)
  );

  or_4bit or_inst (
    .A(A),
    .B(B),
    .Y(Y_or)
  );

  xor_4bit xor_inst (
    .A(A),
    .B(B),
    .Y(Y_xor)
  );

  xnor_4bit xnor_inst (
    .A(A),
    .B(B),
    .Y(Y_xnor)
  );

  nor_4bit nor_inst (
    .A(A),
    .B(B),
    .Y(Y_nor)
  );

  not_4bit not_inst (
    .A(A),
    .Y(Y_not)
  );

  newShifter shifter_inst(
    .A(A),
    .B(B),
    .Y(Y_shifter)
  );

  add_4bit add_inst (
    .A(A),
    .B(B),
    .Cin(Cin),
    .Cout(Cout),
    .Sum(Y_add)

  );

  sub_4bit sub_inst (
    .A(A),
    .B(B),
    .Cin(Cin),
    .Cout(Cout),
    .Result(Y_sub)
  );

  multi_4bit multi_inst (
    .A(A),
    .B(B),
    .Product(Y_multi)
  );

 assign Y = (opcode == 4'b0000) ? Y_and :
            (opcode == 4'b0001) ? Y_nand :
            (opcode == 4'b0010) ? Y_or :
            (opcode == 4'b0011) ? Y_xor :
            (opcode == 4'b0100) ? Y_xnor :
            (opcode == 4'b0101) ? Y_nor :
            (opcode == 4'b0110) ? Y_not :
            (opcode == 4'b0111) ? Y_add :
            (opcode == 4'b1000) ? Y_sub :
            (opcode == 4'b1010) ? Y_shifter :
                                  4'b0;
assign Y_8bit = (opcode == 4'b1001) ? Y_multi:
                                  8'b0;


endmodule
```

3.2 Control Circuit Test Case Verilog Code

The integrated module Test Case which takes several inputs A, B, Cin, and opcode and produces and 8-bit output Y. The test bench initializes the inputs with specific values, and executes a series of tests in 5 second increments, and monitors the outputs. This test bench tests out all the opcode to verify if they work.

```
module benchtest_IntegratedModule;

  reg [3:0] A;
  reg [3:0] B;
  reg cin;
  reg [3:0] opcode;

  // Outputs
  wire [3:0] Y;
  wire [7:0] Y_8bit;
  wire [3:0] Cout;

  // Instantiate the integrated module
  integratedModule uut (
   .A(A),
   .B(B),
   .Cin(cin),
   .opcode(opcode),
   .Y(Y),
   .Y_8bit(Y_8bit),
   .Cout(Cout)
);


  // Initialize test vectors
  initial begin
    $dumpfile("integratedModule_test.vcd");
    $dumpvars(0, benchtest_IntegratedModule);

    // Test 1: AND operation (A=12, B=11)
    A = 4'b1100;
    B = 4'b1011;
    cin = 1;
    opcode = 4'b0000;
    #5;

    // Test 2: XOR operation (A=11, B=6)
    A = 4'b1011;
    B = 4'b0110;
    cin = 0;
    opcode = 4'b0011;
    #5;

    // Test 3: Addition (A=14, B=5)
    A = 4'b1110;
    B = 4'b0101;
    cin = 0;
    opcode = 4'b0111;
    #5;

    // Test 4: Right shift by 1 with fill (A=13, B=9)
    A = 4'b1101;
    B = 4'b1001;
    opcode = 4'b1010;
    #5;

    // Test 5: NAND operation (A=8, B=10)
    A = 4'b1000;
    B = 4'b1010;
    cin = 1;
    opcode = 4'b0001;
    #5;

    // Test 6: OR operation (A=5, B=14)
    A = 4'b0101;
    B = 4'b1110;
    cin = 0;
    opcode = 4'b0010;
    #5;

    // Test 7: XNOR operation (A=13, B=6)
    A = 4'b1101;
    B = 4'b0110;
    cin = 0;
    opcode = 4'b0100;
    #5;

    // Test 8: NOR operation (A=10, B=15)
    A = 4'b1010;
    B = 4'b1111;
    cin = 0;
    opcode = 4'b0101;
    #5;

    // Test 9: NOT operation (A=2, B=0)
    A = 4'b0010;
    B = 4'b0000;
    cin = 0;
    opcode = 4'b0110;
    #5;

    // Test 10: Subtraction (A=11, B=5)
    A = 4'b1011;
    B = 4'b0101;
    cin = 0;
    opcode = 4'b1000;
    #5;

    // Test 11: Multiplication (A=12, B=3)
    A = 4'b1100;
    B = 4'b0011;
    cin = 0;
    opcode = 4'b1001;
    #5;

    $finish;
  end

endmodule
```

4 Integrated Module Waveform Tests

In this section we will showcase the waveforms created using our test benches for each circuit we coded in Verilog. We used GTKWave to create these waveforms.

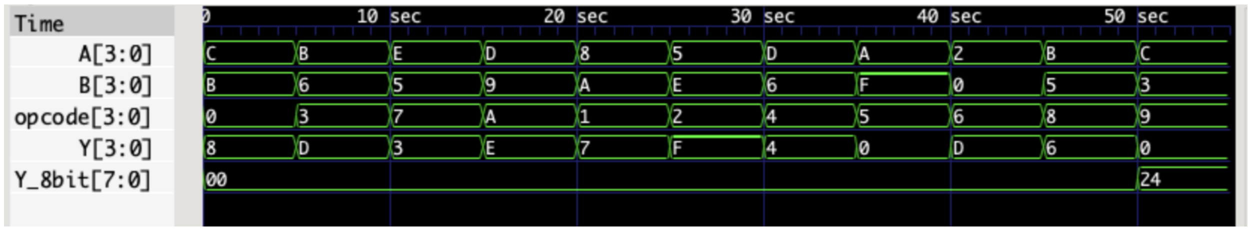

Figure 1: Integrated Module with markers at 5s

Results:

- Test 1: From 0 to 5 seconds, the input A is 1010, input B is 1100 and opcode 0000 which performs the AND Operation outputs Y as 1000.
- Test 2: From 5 to 10 seconds, the input A is 1011, input B is 0110 and opcode 0011 which performs the XOR Operation outputs Y as 1011.
- Test 3: From 10 to 15 seconds, the input A is 1110, input B is 0101, input Cin is 1, and opcode 1010 which performs the ADDITION Operation outputs Cout is 1 and Y as 0011 .
- Test 4 From 15 to 20 seconds, the input A is 1101, input B is 1001 and opcode 0111 which performs the SHIFT Operation outputs Y as 1110.
- Test 5: From 20 to 25 seconds, the input A is 1000, input B is 1010 and opcode 0001 which performs the NAND Operation outputs Y as 0111.

- Test 6: From 25 to 30 seconds, the input A is 0101, input B is 1110 and opcode 0010 which performs the OR Operation outputs Y as 1111.
- Test 7: From 30 to 35 seconds, the input A is 1101, input B is 0110 and opcode 0100 which performs the XNOR Operation outputs Y as 0100.
- Test 8: From 35 to 40 seconds, input B is 1111 and opcode 0101 which performs the NOR Operation outputs Y as 1111.
- Test 9: From 40 to 45 seconds, the input A is 0010, input B is 0000 and opcode 0110 which performs the NOT Operation outputs Y as 1101.
- Test 10: From 45 to 50 seconds, the input A is 1011, input B is 0101 and opcode 1000 which performs the SUBTRACTION Operation outputs Y as 0110.
- Test 11: From 50 to 55 seconds, the input A is 1100, input B is 0011 and opcode 1001 which performs the MULTIPLICATION Operation outputs Y-8bit as 100100 .

5 Conclusion

We are proud to have defined an ALU from first principles in software. For this step, our most significant challenge was defining the formatting of the output and the shift behavior. Defining the operation codes was a methodical process. As a consequence of this project, our team will walk away from this course with improved skills in communication, development, and writing. We also have a new appreciation for the hardware that we are now capable of defining.