Experiment 5. Combinational Logic Design

Masud ul Hasan · Mohammad Elrabaa · Ahmad Khayyat – Version 151, 8 September 2015

Table of Contents

1. Objectives

- Design a simple arithmetic logic unit (ALU) as an example of a combinational logic circuit.
- Use a multiplexer, an adder, and simple logic gates to implement the arithmetic unit.
- Implement an ALU using Verilog behavioral description.

2. Materials Required

- An FPGA prototyping board.
- Design and simulation software tools.

3. Background

In this experiment, you will design a combinational logic circuit that exists in every computer: the arithmetic logic unit (ALU). You will implement the design using Verilog.

Verilog supports two different ways of describing a design: *structural* and *behavioral* descriptions. *Structural description* of a circuit involves describing the components that make up the circuit, and the way they are interconnected, i.e. the structure of the circuit. Such description is also known as a netlist. *Behavioral description*, on the other hand, focuses on what the circuit does, i.e. its behavior. Behavioral description is more powerful and allows us to describe very complex designs easily using familiar sequential statements that are similar to statements of software programming language.

First, you will implement a simple 4-bit arithmetic unit structurally with Verilog using a multiplexer, an adder and other gates.

What is a Multiplexer?

A *multiplexer* is a combinational circuit that accepts multiple inputs, and selects one of them to form its single output. The selection is determined using a separate set of inputs known as the *select lines*.

A multiplexer is also referred to as a MUX.

After implementing the arithmetic unit, you will implement a 4-bit ALU using behavioral Verilog description.

3.1. Building the Arithmetic Unit Using Structural Description

The functionality of the 4-bit arithmetic unit is described in the Functionality of the 4-bit Arithmetic Unit table. `A` and `B` in the table are 4-bit inputs, and `Y` is the 5-bit output.

Table 1. Functionality of the 4-bit Arithmetic Unit

Control Word (S_1 S_0)	Function	Description
0 0	Y = A + B	Add A and B
0 1	Y = A × 2	Multiply A by 2 (A + A)
1 0	Y = A + 1	Increment A by 1
1 1	Y = A + 2	Increment A by 2

Note that all the required operations can be performed using a single 4-bit binary adder. The trick is to carefully manipulate its inputs for each of the desired operations.

The Required Adder Inputs for Each Arithmetic Operation table shows the inputs that should be applied to the adder to implement each of the required arithmetic operations. Note that, throughout the table:

- The `A` input is always used as is.
- C_{in} = 1 only when S_1 = 1, or, equivalently, C_{in} is equal to S_1.
- There are four different possibilities for the `B` input of the adder. This is where a 4-to-1 multiplexer can become useful. It allows us to select one of the four possibilities depending on the operation being performed.

The Structure of the Arithmetic Unit figure illustrates these connections.

Table 2. Required Adder Inputs for Each Arithmetic Operation

Control Word (S_1 S_0)	Required input connections	Function
0 0	A → A, B → B, 0 → C_{in}	Y = A + B + C_{in} = A + B + 0 = A + B
0 1	A → A, A → B, 0 → C_{in}	Y = A + B + C_{in} = A + A + 0 = A + A = 2A
1 0	A → A, 0000 → B, 1 → C_{in}	Y = A + B + C_{in} = A + 0 + 1 = A + 1
1 1	A → A, 0001 → B, 1 → C_{in}	Y = A + B + C_{in} = A + 1 + 1 = A + 2

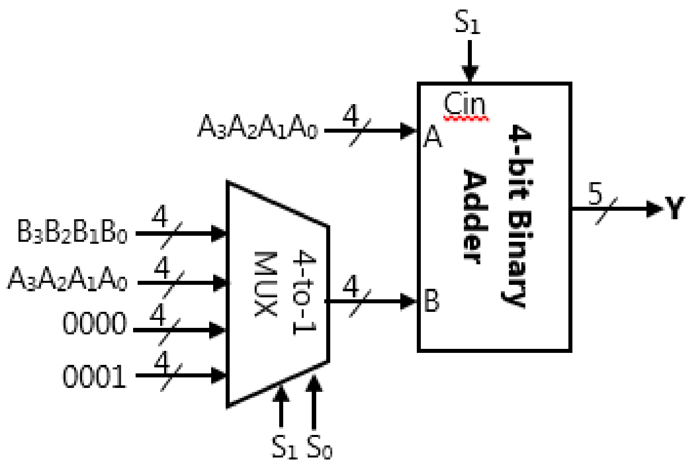

Figure 1. Structure of the Arithmetic Unit

The Verilog description of a 2-to-1 MUX, and a 4-to-1 MUX using the *select* operator `?` are listed below.

2-to-1 Multiplexer

```
assign  Y = S  ?  D1 : D0;
// If S is 1, Y <- D1, else Y <- D0. Y, D0, and D1 are wires of any width
```

4-to-1 Multiplexer

```
module mux_4bit (
    input  wire [3:0]  D0, D1, D2, D3,
    input  wire        S1, S0,
    output wire [3:0]  Y);

    assign  Y = S1 ? (S0 ? D3 : D2) : (S0 ? D1 : D0 );
    // if S1 is 1 then select between D3 and D2
    // else select between D1 and D0
endmodule
```

To build the arithmetic unit shown in the Structure of the Arithmetic Unit figure, we need to modify the above 4-to-1 MUX description so that it selects one of the B, A, 0, or 1 inputs, as shown in the Structure of the Arithmetic Unit.

Replace the `Dx` inputs in the 4-to-1 MUX description with the required inputs.

3.2. Building the Arithmetic Unit Using Behavioral Description

In the previous section, we saw how we can build the arithmetic unit by putting a few different components together, structurally. In this section, we will implement the same unit by describing its behavior instead of describing its structure or components.

The Verilog behavioral description of the 4-bit arithmetic unit is shown below.

Verilog Behavioral Descriptions of the 4-bit Arithmetic Unit

```
module arithmetic_unit_4bit_behavioral (
    input      [3:0]  A, B,
    input             S1, S0,
    output reg [4:0]  Y);

    always @*
    begin
        case ({S1,S0})
            2'b00 : Y = A + B;          // ADD A and B
            2'b01 : Y = A * 2;          // 2A
            2'b10 : Y = A + 1;          // Increment A by 1
            2'b11 : Y = A + 2;          // Increment A by 2
        endcase
    end
endmodule
```

The above Verilog description will result in a different structure than the one depicted in the Structure of the Arithmetic Unit figure. It results in the structure shown in the figure: Structure of the Arithmetic Unit Resulting from Behavioral Description. Behaviorally, the `case` statement in Verilog describes a multiplexer, and the expression in each branch of the `case` statement results in its own circuit, which generates one of the multiplexer's inputs.

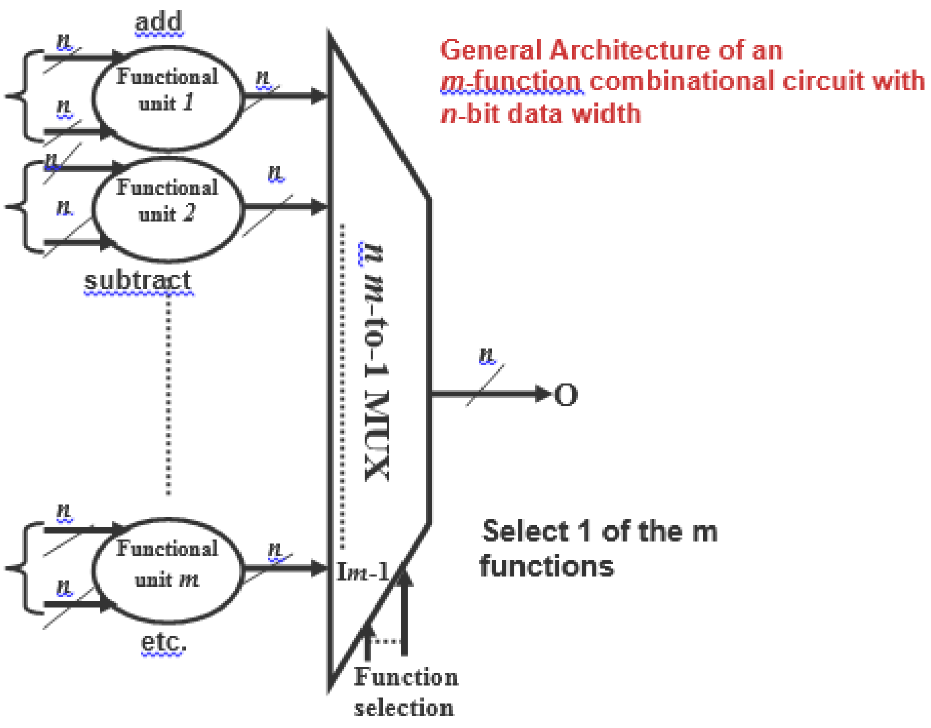
add
Functional unit 1
Functional unit 2
subtract
Functional unit m
etc.
n
n m-to-1 MUX
Im-1
O
Function selection
General Architecture of an m-function combinational circuit with n-bit data width
Select 1 of the m functions

Figure 2. Structure of the Arithmetic Unit Resulting from Behavioral Description

3.3. Extending the Arithmetic Unit Functionality

Lastly, we would like to extend the arithmetic unit to become a full *arithmetic logic unit* (ALU), whose functionality is described in the Required Functionality of the 4-bit ALU table. `A` and `B` in the table are 4-bit inputs, and `Y` is the 5-bit output.

Table 3. Required Functionality of the 4-bit ALU

Control Word (S_2 S_1 S_0)	Function	Description
0 0 0	Y = A + B	Add A and B
0 0 1	Y = A * 2	Multiply A by 2
0 1 0	Y = A + 1	Increment A by 1
0 1 1	Y = A + 2	Increment A by 2
1 0 0	Y = ~A	Bit-wise invert A
1 0 1	Y = A & B	Bit-wise AND A and B

Control Word (S S S)	Function	Description
1 1 0	Y = A | B	Bit-wise OR A and B
1 1 1	Y = A ^ B	Bit-wise XOR A and B

4. Tasks

4.1. Build the 4-Bit Arithmetic Unit Structurally

In this task, you will implement the circuit shown in the Structure of the Arithmetic Unit figure.

1. Build a Verilog module for the 4-bit binary adder.
2. Build a Verilog module for the 4-to-1 multiplexer.
3. Instantiate the two modules in a top-level module, and apply the appropriate inputs to the multiplexer.
4. Verify the correctness of your design using simulation.

   A *test bench* module is provided to you to help you verify your design through simulation. To use the test bench, instantiate your top-level module in the provided test bench, and use the test bench as the top-level module in the simulator.
5. Implement your design, download it to the FPGA, and verify its operation.

4.2. Build the 4-Bit Arithmetic Unit Behaviorally

In this task, you will use the Verilog Behavioral Descriptions of the 4-bit Arithmetic Unit.

1. Enter the Verilog description.
2. Simulate the design and verify its correctness using the provided test bench.
3. Implement the design, download it to the FPGA, and verify its operation.

Use the same UCF file of your structural design in the previous task (Build the 4-Bit Arithmetic Unit Structurally).

4.3. Extend the Arithmetic Unit Functionality

In this task, you will extend the behavioral Verilog description of the arithmetic unit to implement an arithmetic logic unit (ALU) with the functionality described in the Required Functionality of the 4-bit ALU table.

1. Design your ALU in a module named `alu_4bit`.
2. Simulate your design using the provided test bench.
3. Verify its operation on the FPGA.

4.4. Discussion

Answer the following questions:

1. Considering the structural and the behavioral descriptions of the arithmetic unit, which style of

design did you find is easier to describe?

2. Why were you able to reuse the same UCF file for the first two tasks (structural and behavioral implementations of the arithmetic unit)?

   Can you reuse the same UCF file for the third task (building the ALU)?

   Explain your answer.

3. Would you be able to add the additional ALU operations in the third task if you were using the structural design of the first task?

   If no, why? If yes, how?

4. How would you extend the functionality of the ALU in the third task to have three operands (i.e. `A`, `B`, and `C` instead of `A` and `B` only)? What would be the number of possible operations on these operands?

---

5. Grading Sheet

Task	**Points**
Build the arithmetic unit structurally	20
Build the arithmetic unit behaviorally	20
Build the ALU	35
Lab notebook and discussion	25

Version 151
Last updated 2016-02-18 00:10:51 AST