Mälardalen University

DVA494

Lab 1: Designing a 4-Bit Combinational ALU

Spring 2026

1 Introduction & Objectives

Welcome to the first VHDL design lab in the course **Programming of Reliable Embedded Systems**! The goal of this lab is to design, simulate, and test a 4-bit Arithmetic Logic Unit (ALU). An ALU is the core component of a CPU, responsible for all mathematical and logical calculations.

This lab will require you to use the concurrent design principles you've just learned. You will create a **purely combinational circuit** (no clock signal) that performs eight different operations on two 4-bit inputs (`A` and `B`), selected by a 3-bit `OPCODE` .

1.1 Key Learning Objectives

- Write a VHDL entity and architecture from the given specification.
- Implement combinational logic using concurrent statements.
- Correctly use the `IEEE.NUMERIC_STD` library for arithmetic operations.
- Differentiate between `SIGNED` and `UNSIGNED` data types and their implications.
- Implement logic for status flags (Carry and Overflow).
- Create a comprehensive testbench to verify your design.

2 Prerequisites

- **Knowledge** : Digital logic fundamentals (gates, adders, 2's complement), basic VHDL syntax, and concurrent VHDL. - **Check Lectures 1 and 2**
- **Software**: AMD Vivado.

3 Design Specification

Your task is to implement a VHDL module named `alu` that matches the inputs, outputs, and functional behaviour defined in the tables below.

Table 1: Entity Port Definition

Name	Mode	Width	Meaning
A	INPUT	4	Input A
B	INPUT	4	Input B
OPCODE	INPUT	3	Operation Code
X	OUTPUT	4	Output or Least Significant Nibble (LSN) of Output
Y	OUTPUT	4	Most Significant Nibble of Output *(Used only by multiplication)*
C	OUTPUT	1	Carry Out (or Borrow)
V	OUTPUT	1	Overflow

Table 2: Meaning of Opcodes

OPCODE	OPERATION	FORMULA	Y	V	C
”000”	NOR	$X = A$ NOR B	0	0	0
”001”	NAND	$X = A$ NAND B	0	0	0
”010”	XOR	$X = A$ XOR B	0	0	0
”011”	Unsigned Addition	$(C : X) = A + B$	0	0	↑
”100”	Signed Addition	$X = A + B$	0	↑	↑
”101”	Signed Subtraction	$X = A - B$	0	↑	↑
”110”	Unsigned Multiplication	$(Y : X) = A * B$	↑	0	0
”111”	Signed Multiplication	$(Y : X) = A * B$	↑	0	0

Table Key:

- ↑: This output is active and should be calculated for this operation.
- 0: This output should be driven to a logical ’0’ for this operation.
- $(C : X)$: A 5-bit result, where `C` is the most significant bit (MSB) and `X` is the 4-bit result.
- $(Y : X)$: An 8-bit result, where `Y` is the most significant 4 bits (MSN) and `X` is the least significant 4 bits (LSN).
- **Signed** operations treat `A` and `B` as 4-bit 2’s complement numbers.
- **Unsigned** operations treat `A` and `B` as standard 4-bit positive integers.

4 Implementation Guidance

The requirement is that you strictly treat this as **a combinational circuit**. So all outputs must be a direct function of the inputs. A common and efficient way to implement this in VHDL is to calculate **all** possible results concurrently and then use the `OPCODE` to select the correct ones for the final outputs.

4.1 File Setup

Create a file `alu.vhd`. Start with the necessary libraries. You **must** use `NUMERIC_STD` for this lab.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL; -- This is essential!

entity alu is
    port (
        A       : in  std_logic_vector(3 downto 0);
        B       : in  std_logic_vector(3 downto 0);
        OPCODE  : in  std_logic_vector(2 downto 0);
        X       : out std_logic_vector(3 downto 0);
        Y       : out std_logic_vector(3 downto 0);
        C       : out std_logic;
        V       : out std_logic
    );
end entity alu;

architecture behavioral of alu is
    -- TODO: Declare internal signals here
begin
    -- TODO: Add concurrent logic here
end architecture behavioral;
```

4.2 Implement All Operations Concurrently

Write the VHDL code for all eight operations. These statements will ”run” in parallel.

4.3 Calculate Flags (V and C)

The Overflow (`V`) and Carry (`C`) flags are the trickiest part. They *depend* on the operation. Implement the correct logic for determining the Overflow (`V`) and Carry (`C`) flags.
Use concurrent statements for C and V based on the logic.

5 Testbench Requirements

Create a testbench file (`alu_tb.vhd`) to verify your design. Your testbench should be a **linear testbench** that steps through a series of inputs and checks the outputs.

Your testbench MUST:

1. Instantiate your `alu` component (the “Device Under Test” or DUT).
2. Have a single `process` that drives the `A` , `B` , and `OPCODE` signals.
3. Include `wait for <time>` statements (e.g., `wait for 10 ns;`) after each input change to allow the combinational logic to update.
4. Test **all 8 operations**.
5. For each operation, include at least **two test cases**:
   - One ”normal” case (e.g., `A=5` , `B=2`).
   - One ”edge” case that tests the flags (e.g., for unsigned add, `A=12` , `B=5` to test the carry; for signed add, `A=7` , `B=1` to test the overflow).
6. Use `assert` statements to automatically check if the outputs (`X` , `Y` , `C` , `V`) match the expected values.

Example `assert` usage:

```vhdl
        -- Test NOR
        report "Testing NOR (000)";
        A <= "0101"; B <= "1001"; OPCODE <= "000";
        wait for 10 ns;
        assert (X = "0010") report "NOR test 1 failed!" severity failure;

        -- Test Unsigned Add
        report "Testing Unsigned Add (011)";
        A <= "1100"; B <= "0101"; -- 12 + 5
        OPCODE <= "011";
        wait for 10 ns;
        assert (X = "0001" AND C = '1')
            report "Unsigned Add test 1 failed!" severity failure;
```

6 Deliverables

Please submit the following:

1. `alu.vhd` : Your completed VHDL code for the ALU entity and architecture.
2. `alu_tb.vhd` : Your completed testbench file.
3. Simulation Waveforms: Screenshots from your simulator clearly showing:
    - The test for **Unsigned Multiplication**.
    - The test for **Signed Addition** (showing an overflow case).
    - The test for **Signed Subtraction** (showing an overflow case).
    - Make sure to include the `A` , `B` , `OPCODE` , `X` , `Y` , `C` , and `V` signals in your waveforms.
4. Short Report: A few pages summary describing you implementation, simulation and any challenges you faced, particularly in implementing the `C` and `V` flags.