L-Università ta' Malta

DEPARTMENT OF COMMUNICATIONS AND COMPUTER ENGINEERING

**Faculty of Information and Communication Technology**

CCE1013—Computer Logic 1

# Practical 4

Trevor Spiteri

trevor.spiteri@um.edu.mt

http://staff.um.edu.mt/trevor.spiteri

30 November 2018

# ALU

## Objective

The objective of this practical is to design and implement a simple ALU.

## The inputs and outputs

Table 1 lists the ALU inputs and outputs. $A$ and $B$ are input operands, and $Y$ is an output operand. $C_{\text{in}}$ and $Op$ are control inputs, while $C_{\text{out}}$, $V$, $Z$ and $S$ are control outputs.

## The operations

The ALU supports 11 operations.

**TABLE 1:** THE ALU INPUTS AND OUTPUTS.

| Name | Direction | Description |
|---|---|---|
| $A$ | input | eight-bit input operand |
| $B$ | input | eight-bit input operand |
| $C_{\text{in}}$ | input | carry in |
| $Op$ | input | four-bit operation code |
| $Y$ | output | eight-bit output operand |
| $C_{\text{out}}$ | output | carry out |
| $V$ | output | overflow flag |
| $Z$ | output | zero flag |
| $S$ | output | sign flag, or negative flag |

**ADD** Add without carry.

$$Y = A + B$$

$C_{\text{out}}$ is the carry out flag, which should be equal to the adder's $C_{\text{out}}$.

$V$ is the overflow flag, which should be set to $1$ if an overflow occurs.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**ADC** Add with carry, where $C_{\text{in}}$ is the carry.

$$Y = A + B + C_{\text{in}}$$

$C_{\text{out}}$ is the carry out flag, which should be equal to the adder's $C_{\text{out}}$.

$V$ is the overflow flag, which should be set to $1$ if an overflow occurs.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**SUB** Subtract without borrow.

$$Y = A - B = A + \overline{B} + 1$$

$C_{\text{out}}$ is the borrow out flag. The borrow output is specified to be equal to the complement of the adder's carry out.

$$C_{\text{out}} = \overline{C_{\text{out,adder}}}$$

$V$ is the overflow flag, which should be set to $1$ if an overflow occurs.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**SBB** Subtract with borrow, where $C_{\text{in}}$ is the borrow.

$$Y = A - B - C_{\text{in}} = A + \overline{B} + \overline{C_{\text{in}}}$$

$C_{\text{out}}$ is the borrow out flag. The borrow output is specified to be equal to the complement of the adder's carry out.

$$C_{\text{out}} = \overline{C_{\text{out,adder}}}$$

$V$ is the overflow flag, which should be set to $1$ if an overflow occurs.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**INC** Increment.

$$Y = A + 1$$

This operation should be identical to the ADD operation with $B = 1$.

**DEC** Decrement.

$$Y = A - 1$$

This operation should be identical to the ADD operation with $B = -1$.

**NEG** Negate.

$$Y = -A$$

This operation should be identical to the SUB operation with the first operand set to $0$ and the second operand set to $A$.

**NOT** Bitwise complement. The output is the bitwise complement of $A$.

$$Y = \overline{A}$$

$C_{\text{out}}$ is the carry out flag, which should be cleared to $0$.

$V$ is the overflow flag, which should be cleared to $0$.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**AND** Bitwise AND. The output is the bitwise logic AND of $A$ and $B$.

$$Y = A \,\text{AND}\, B$$

$C_{\text{out}}$ is the carry out flag, which should be cleared to $0$.

$V$ is the overflow flag, which should be cleared to $0$.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**Table 2:** Example opcode mapping and adder inputs.

| $Op$ | Operation | $A_{\text{adder}}$ | $+$ | $B_{\text{adder}}$ | $+$ | $C_{\text{in,adder}}$ |
|---|---|---|---|---|---|---|
| 0000 | ADD | $A$ | $+$ | $B$ | $+$ | $0$ |
| 0001 | ADC (with carry) | $A$ | $+$ | $B$ | $+$ | $C$ |
| 0010 | SUB | $A$ | $+$ | $\overline{B}$ | $+$ | $1$ |
| 0011 | SBB (with borrow) | $A$ | $+$ | $\overline{B}$ | $+$ | $\overline{C}$ |
| 0100 | INC | $A$ | $+$ | $1$ | $+$ | $0$ |
| 0101 | DEC | $A$ | $+$ | $(-1)$ | $+$ | $0$ |
| 0110 | NEG | $0$ | $+$ | $\overline{A}$ | $+$ | $1$ |
| 0111 | NOT | | | | | |
| 1000 | AND | | | | | |
| 1001 | OR | | | | | |
| 1010 | XOR | | | | | |

**OR** Bitwise OR. The output is the bitwise logic OR of $A$ and $B$.

$$Y = A\,\text{OR}\,B$$

$C_{\text{out}}$ is the carry out flag, which should be cleared to $0$.

$V$ is the overflow flag, which should be cleared to $0$.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

**XOR** Bitwise XOR. The output is the bitwise logic XOR of $A$ and $B$.

$$Y = A\,\text{XOR}\;B$$

$C_{\text{out}}$ is the carry out flag, which should be cleared to $0$.

$V$ is the overflow flag, which should be cleared to $0$.

$Z$ is the zero flag, which should be set to $1$ if $Y = 0$.

$S$ is the sign flag, which should be set to $1$ if $Y < 0$.

## The operation codes

Table 2 shows an example mapping of the operation code $Op$ to the operations. For the arithmetic operations, a working input combination of $A$, $B$ and $C_{\text{in}}$ to the adder is also provided.

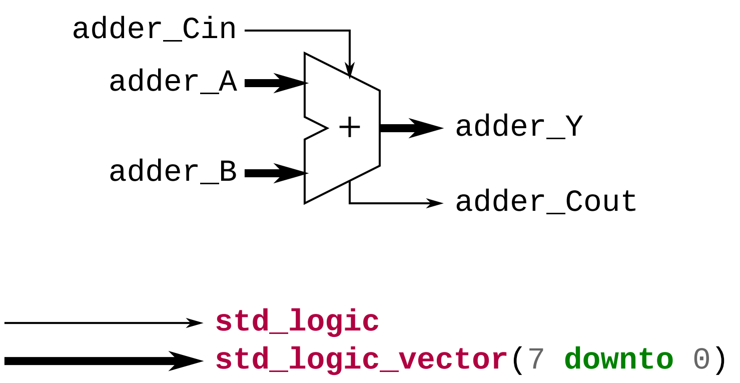

**Figure 1:** The internal eight-bit adder.

## Tasks

1. Draw a sketch of the ALU implementation. The ALU should have **one** eight-bit adder, and can have other combinational logic components such as multiplexers.

2. Create a new project and a new VHDL module. The module should have the inputs and outputs of Table 1.

3. Implement the sketched ALU in VHDL.

   **Hint 1:** To implement a multiplexer in VHDL, use code similar to the following. Note that in this example, the selection input is three bits wide, while for the ALU, the operation code can be wider.

   ```
   mux_output <= mux_input_0 when sel = "000" else
                 mux_input_1 when sel = "001" else
                 mux_input_2 when sel = "010" else
                 mux_input_3 when sel = "011" else
                 mux_input_4 when sel = "100" else
                 mux_input_5 when sel = "101" else
                 mux_input_6 when sel = "110" else
                 mux_input_7;
   ```

   **Hint 2:** To implement the eight-bit adder shown in Figure 1, you can use the `unsigned` vector type in the `ieee.numeric_std` package by following these three steps:

(a) Add the package line at the appropriate place near the top of the VHDL file:

```
use ieee.numeric_std.all;
```

(b) Define signals of type **unsigned** in the architecture declaration part for the values $A$, $B$, $C_{\text{in}}$, $Y$ and $C_{\text{out}}$. The declaration part is between the **architecture** line and the **begin**. To make the operation simpler, $Y$ and $C_{\text{out}}$ are combined into one nine-bit signal. The signal declarations are shown below.

```
signal uns_A: unsigned(7 downto 0); -- 8 bits
signal uns_B: unsigned(7 downto 0); -- 8 bits
signal uns_Cin: unsigned(0 downto 0); -- 1 bit
signal uns_Cout_Y: unsigned(8 downto 0); -- 9 bits
```

(c) In the architecture body, that is after the architecture's **begin**, you can perform the addition like this:

```
uns_A <= unsigned(adder_A);
uns_B <= unsigned(adder_B);
uns_Cin(0) <= adder_Cin;

-- concatenate "0" to uns_A to make it 9 bits wide
uns_Cout_Y <= ("0" & uns_A) + uns_B + uns_Cin;

adder_Cout <= uns_Cout_Y(8);
adder_Y <= std_logic_vector(uns_Cout_Y(7 downto 0));
```

Of course, you have to use your own signal names for the signals `adder_A`, `adder_B`, `adder_Cin`, `adder_Cout` and `adder_Y`.

4. Write a suitable test bench to verify the correct operation of the ALU. You should test all the operations at least once. For some operations you should have more than one test, for example, you should test additions with and without overflow.

## Report

Your report should describe your work concisely. The design of the ALU should be included. For the diagram of Task 1, a scan or a photo of a hand-drawn diagram is sufficient. The report should also include the VHDL code you wrote, including the test bench, and details about how you tested the ALU.