

## Practical 5: An $n$ -bit Arithmetic-Logic Unit

---

### 5.1 About this practical

The objective of this practical is to design and test a 4-bit Arithmetic-Logic Unit (ALU) as discussed in lecture notes.

Possible variants to consider:

- 1-bit component and 4-bit ALU designed using graphical entry
- Design 1-bit component in VHDL, 4-bit ALU graphically
- Design as single-level VHDL component using the “generate” statement.
- Design first a 1-bit VHDL component and then 4-bit ALU using the “component instantiation” statement and “generate” statement.
- Simulate 1-bit design, or the complete ALU only.

I will try first the second option.

### Contents

5.1	About this practical . . . . .	1
5.2	The 1-bit building block . . . . .	1
5.2.1	VHDL specification of the 1-bit ALU component . . . . .	2
5.3	The 4-bit ALU . . . . .	4
5.4	Simulating the ALU . . . . .	5
5.5	The report . . . . .	6

### 5.2 The 1-bit building block

The  $n$ -bit Arithmetic-Logic Unit (ALU) that we are about to design will consist of 1-bit slices connected in a linear array fashion. We design first such a 1-bit component. The operations performed by the  $i$ -th bit of the ALU and its structure are presented in Figure 1.

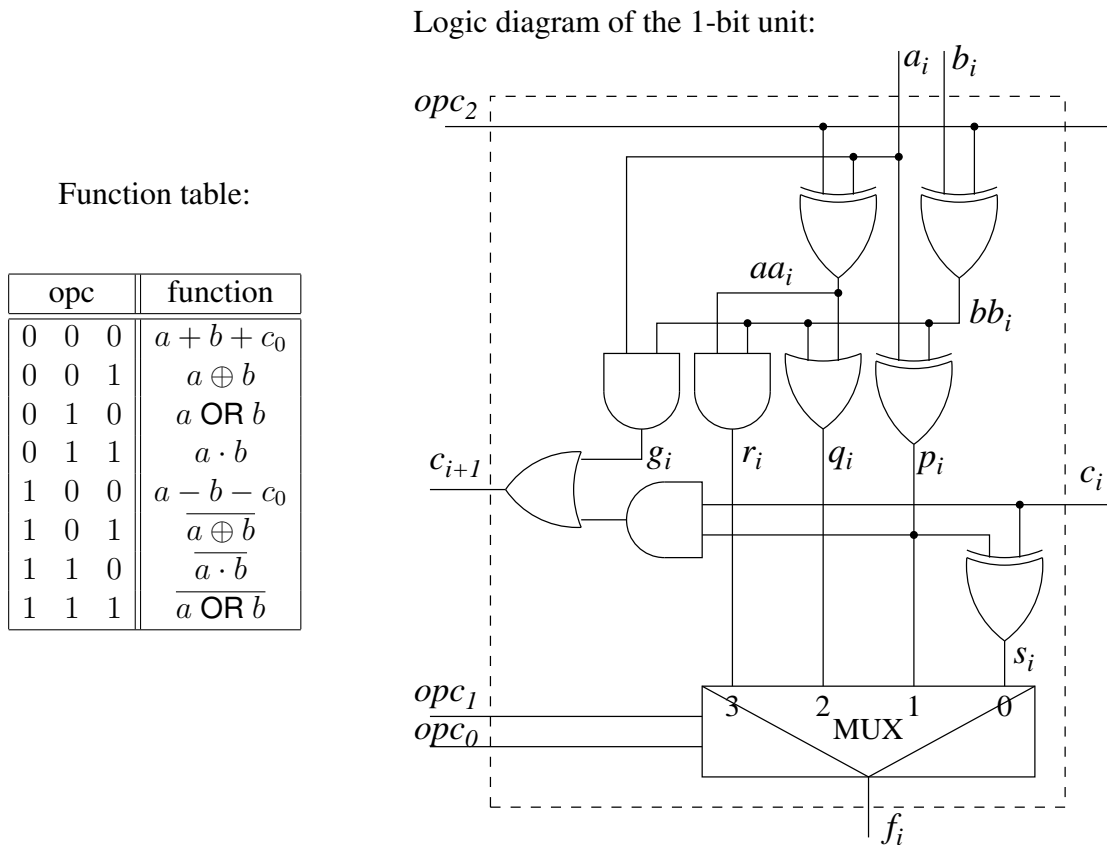


Figure 1: The  $i$ th bit of the ALU: the function table and the logic structure

### 5.2.1 VHDL specification of the 1-bit ALU component

- Start Designer Manager invoking **FPGAdv** tools.
- In the **Getting Started** wizard select **Create a new Project** button and click **OK**
- In a **Creating a New Project** wizard specify:

Name of new project: P5**you**  
 Directory in which your project folder will be created: DigDes

where **you** should be replaced with **your initials**.

- Open a Block Diagram window by selecting in the Design Manager **File → New → VHDL Views → VHDL Combined**
- In the **File Creation Wizard** window specify:
  - Entity name, e.g., **ALU1b**
  - Architecture name, e.g., **dflow**

This will open a **VHDL editor window** with a VHDL template

- Enter the **entity** specification to be similar to the following:

```
ENTITY ALU1b IS
  PORT (
    a, b, c : IN  std_logic ;
    f, d : OUT std_logic ;
    opc : IN  std_logic_vector(2 downto 0)
  );
END ENTITY ALU1b;
```

- In the dataflow architecture we specify all the internal signal and write all logic equations. The architecture part may look as follows:

```
ARCHITECTURE dflow OF ALU1b IS
  SIGNAL aa, bb, g, q, p, r, s : std_logic ;
BEGIN
  aa <= a XOR opc(2) ;
  bb <= b XOR opc(2) ;
  p <= a XOR bb ;
  g <= a AND bb ;
  r <= aa AND bb ;
  q <= aa OR bb ;
  s <= p XOR c ;
  d <= g OR (p AND c) ;

  WITH opc(1 downto 0) SELECT
  f <= s WHEN "00" ,
    p WHEN "01" ,
    q WHEN "10" ,
    r WHEN OTHERS ;

END ARCHITECTURE dflow;
```

- Save the VHDL file.
- Now, we can generate the component and compile the VHDL specification. To do that select in the VHDL window **Tasks → ModelSim flow → Run single**
- Most likely you will have **syntactic errors** reported in the **log window**. Fix the errors save the file and re-run the ModelSim flow.
- At the conclusion of the successful compilation a window **Start ModelSim** appears. Accept its default settings.
- It will open ModelSim simulation window.

### Simulate or not to simulate

Since the time is an issue, I will leave the decision to you.

If not, you can quit the simulator for now.

- We now **convert the VHDL view to graphics**, to create a graphical block that can be used in the graphical entry. One way of doing so is to select in the VHDL window:

**Convert to Graphics** button (icon): 

- If there are no errors (better not) in the **Design Manager** window you will now find an icon **symbol** attached to the **Design Unit** ALU1b
- Double-click on this symbol to open up a graphical view of the symbol.
- Edit (and save) the symbol to look similar to the one in Figure 2.

Created using Mentor Graphics HDL2Graphics(TM) Technology  
 on - 14:08:00 29/03/2006  
 from - C:\app\teach\DigDes\Projects\p5ALU\_lib\hdl\  
 ALU1b\_dflow.vhd

**Package List**  
 LIBRARY ieee;  
 USE ieee.std\_logic\_1164.all;  
 USE ieee.std\_logic\_arith.all;

**Declarations**

**Ports:**

```

a : IN      std_logic ;
b : IN      std_logic ;
c : IN      std_logic ;
f : OUT     std_logic ;
d : OUT     std_logic ;
opc : IN    std_logic_vector (2 downto 0)
        
```

**User:**

<company name>		Project: p5ALU
Title:	1-bit slice of the ALU	Move port around so that it will be easier to create an n-bit ALU
Path:	p5ALU_lib/ALU1b/symbol	
Edited:	by app on 29 Mar 2006	

Generic Declarations

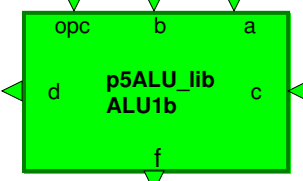


Figure 2: The symbol of the 1-bit ALU component

### 5.3 The 4-bit ALU

We are now ready to create a 4-bit ALU using the 1-bit component specified previously. The symbol for the 4-bit ALU will eventually be similar to the one in Figure 3.

- From the Design Manager open a new Block Diagram window by selecting **File → New → Graphical view → Block Diagram**

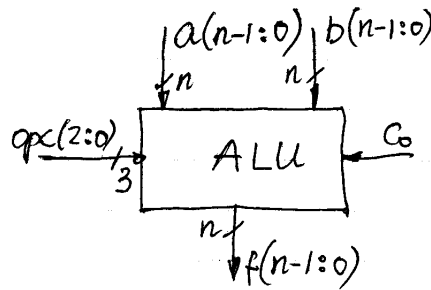
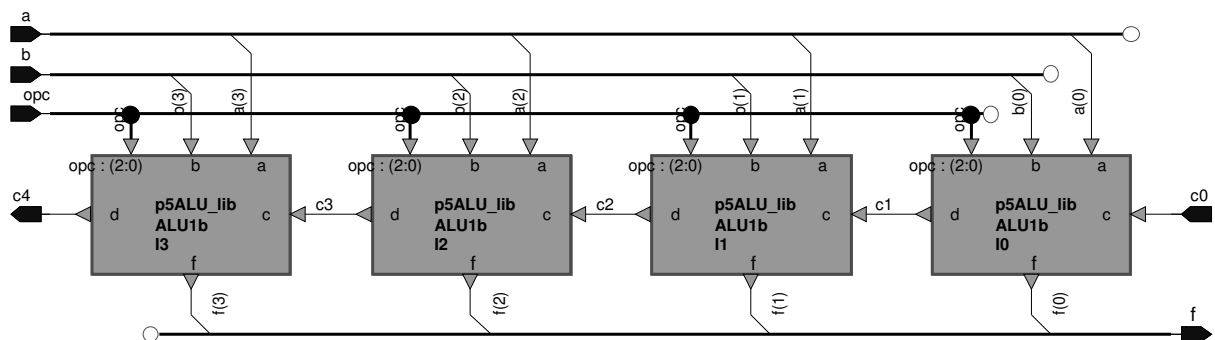


Figure 3: The symbol of the 1-bit ALU component

- Create a 4-bit ALU similar to that in Figure 4.
- Save it under a sensible name, e.g. ALU4bit.

**Package List**  
 LIBRARY ieee;  
 USE ieee.std\_logic\_1164.all;  
 USE ieee.std\_logic\_arith.all;

**Declarations**  
**Ports:**  
 a : std\_logic\_vector(3 DOWNTO 0)  
 b : std\_logic\_vector(3 DOWNTO 0)  
 c0 : std\_logic  
 opc : std\_logic\_vector(3 DOWNTO 0)  
 c4 : std\_logic  
 f : std\_logic\_vector(3 DOWNTO 0)  
**Diagram Signals:**  
 SIGNAL c1 : std\_logic  
 SIGNAL c2 : std\_logic  
 SIGNAL c3 : std\_logic



<company name>	<<-- p5ALU
Title: 4-bit ALU	Note that each 1-bit component has its own ID: I0 ... I3
Path: p5ALU_lib/ALU4b/struct	
Edited: by app on 29 Mar 2006	

Figure 4: The block-diagram of the 4-bit ALU

### 5.4 Simulating the ALU

Simulation is a very difficult task and must be well planned. Our 4-bit ALU has 12 input signals. It means that we might need to test 4096 different cases. It is a rather difficult task to perform it manually. Therefore we will limit testing to the following cases:

1. For positive  $a$  and  $b$  such that there is no overflow in addition perform all 8 possible operations for all  $opc$ .  
 Select  $c_0$  to be zero for addition and one for subtraction operation.

2. As above for negative  $a$  and  $b$ .
3. Two pairs of addition and subtraction operations to show overflow and correct results.

Write an appropriate simulation script to cover all the above cases.

The following graph can assist you with planning generation of the input signals:

$opc$	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	4	0	4
$a$	+	+	+	+	+	+	+	+	-	-	-	-	-	-	-	-	$a1$	$a2$	$a3$	$a4$
$b$	+	+	+	+	+	+	+	+	-	-	-	-	-	-	-	-	$b1$	$b2$	$b3$	$b4$
$c_0$	0	*	*	*	1	*	*	*	0	*	*	*	1	*	*	*	0	0	1	1

You can assume that one time step is, say 10ns, and write the simulation script using the **force** command. For your convenience I attach the full description of the command in the file **force.pdf**.

## 5.5 The report

In your report (due after prac 6) include the results in the form of:

- Logic equations as specified in this manual,
- block/logic diagrams,
- VHDL programs (if available),
- simulation scripts (if available),
- simulation waveforms,
- **short** description of the above.

Wherever possible publish the results selecting in the **Block Diagram** window

**File** → **HTML Export ...**. Specify the export target directory to be `... \DigDes \Reports`.