The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

VHDL basics: multiplexers

Cluster: Cluster1
Module: Module1d

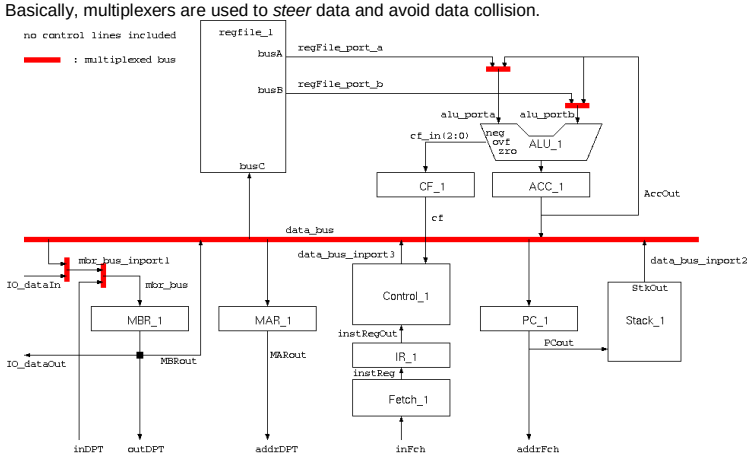
Target group: Students

Version: 1.0
Date: 21/03/06
Author: Osman Allam
Modified by: Geert Vanwijnsberghe
History : 4/12/06 : testbench added

This material was developed with support of the European Social Fund.
ESF: Prevent and combat unemployment by promoting employability, entrepreneurship, adaptability and equal opportunities between women and men, and by investment in people.
http://www.esf-agentschap.be

Introduction

Basically, multiplexers are used to *steer* data and avoid data collision.



Objectives

After completing this module, you will be able to design elementary combinational circuits.

Knowledge background

Basic VHDL knowledge

Classification

- Level: 1
- Duration: 30 minutes

Input

- VHDL template of 2-input multiplexer (`mux2bus.vhd`).
- VHDL template of 4-inputer multiplexer (`mux4bus.vhd`).

The lab

Multiplexers are combinational logic circuits. It is necessary to carefully cover all possible cases and include all relevant signals in the process sensitivity list (if you use a process). Deviation from these 2 rules may result in latches being generated and undesirable or unpredictable behavior.

Multiplexers can be modelled VHDL in one of the following two ways:

1. **Concurrent statements:**
   A multiplexer can be a signal concurrent conditional signal assignment statement.
   Example: 2-input multiplexer

```
-- ports
A, B : in bit_vector (7 downto 0);
Outp : out bit_vector (7 downto 0);
Sel : in bit;
..
-- concurrent statement
outp <= A when sel = '0', B when sel = '1';
```

2. **Sequential statements within a process:**
   A multiplexer can be a case statement or an if-statement inside a process.
   Example: 2-input multiplexer

```
-- The same ports as in the previous example
..
-- concurrent statement
process (A, B, sel) -- remember that processes themselves are
                    -- concurrent statement
begin
  if (sel = '0') then
    outp <= A;
  else
    outp <= B;
  end if;
end process;
```

Selecting bit type for the selection line (sel) is to demonstrate the concept simply. However, if you use std_logic type (which should always be done), you have to take into account that a std_logic signal may take any of 9 possible values.
Thus the first example has to be re-writen to cover all possibilites as follows

```
-- ports
..
sel : in std_logic;
..
-- concurrent statement
outp <= A when sel = '0', B when OTHERS = '1';
```

The second example doesn't have to be re-written (except changing the ports declaration) since "else" effectively covers all possibilites other than '0' in the if-condition. The same effect is

achieved by using 'OTHERS' in the concurrent assignment statement.

Exercise 1: 2-input multiplexer

Design a 2-input multiplexer. The inputs and output are unconstrained std_logic_vector. Use suitable width for the selection line(s). Implement your multiplexer as a case statement.
Use the template provided in the file `mux2bus.vhd`.

Exercise 2: 4-input multiplexer

Design a 4-input multiplexer. The inputs and output are unconstrained std_logic_vector. Use a suitable width for the selection line(s). Implement your multiplexer as a case statement.
Use the template provided in the file `mux4bus.vhd`.
Use the testbench tb_mux to verify the multiplexers. This testbench does not contain automatic output comparison. This means that you have to verify the output manually.