FYS4220/9220

UNIVERSITY OF OSLO

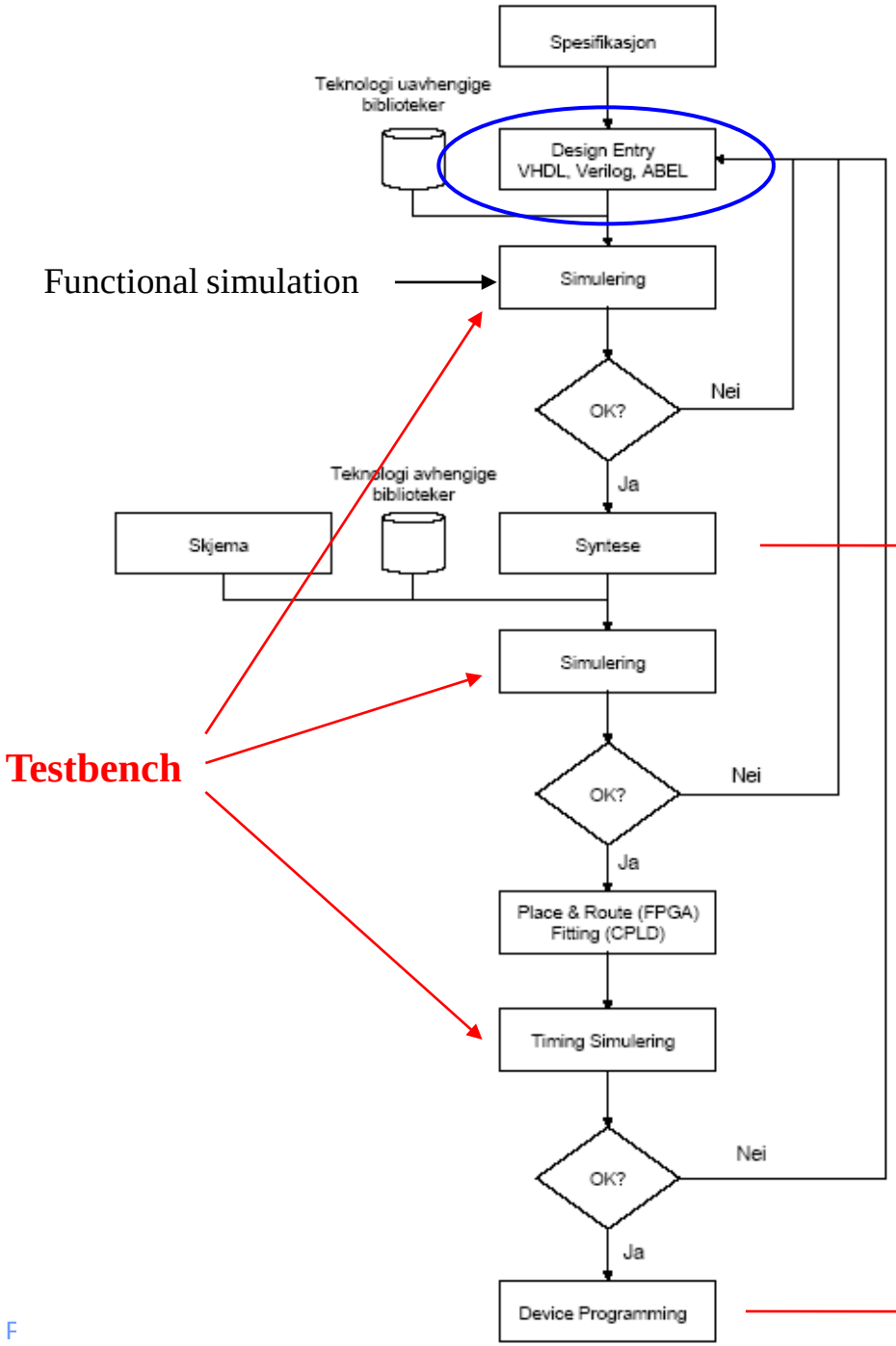
Introduction to VHDL

Lecture #2

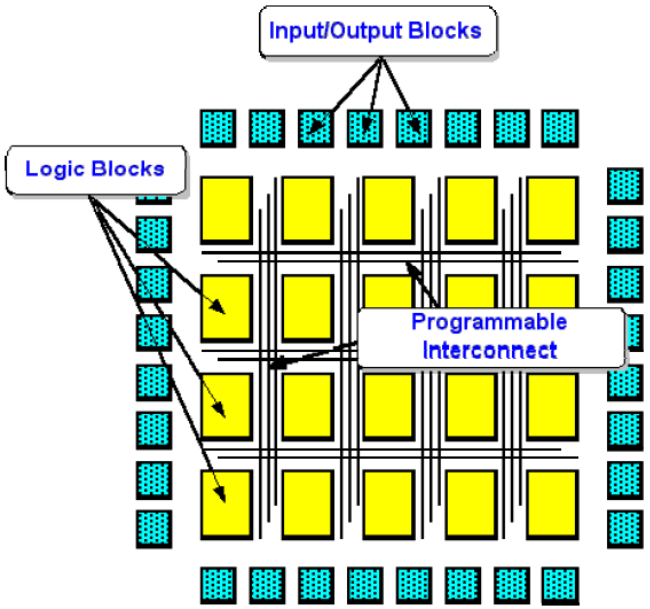
Reading: 3.1 – 3.4, 3.6, 3.7, 5.6, 4.1, 4.2.1 and 4.7 in Zwolinski

J. K. Bekkeng, 3.08.2011

Design flow

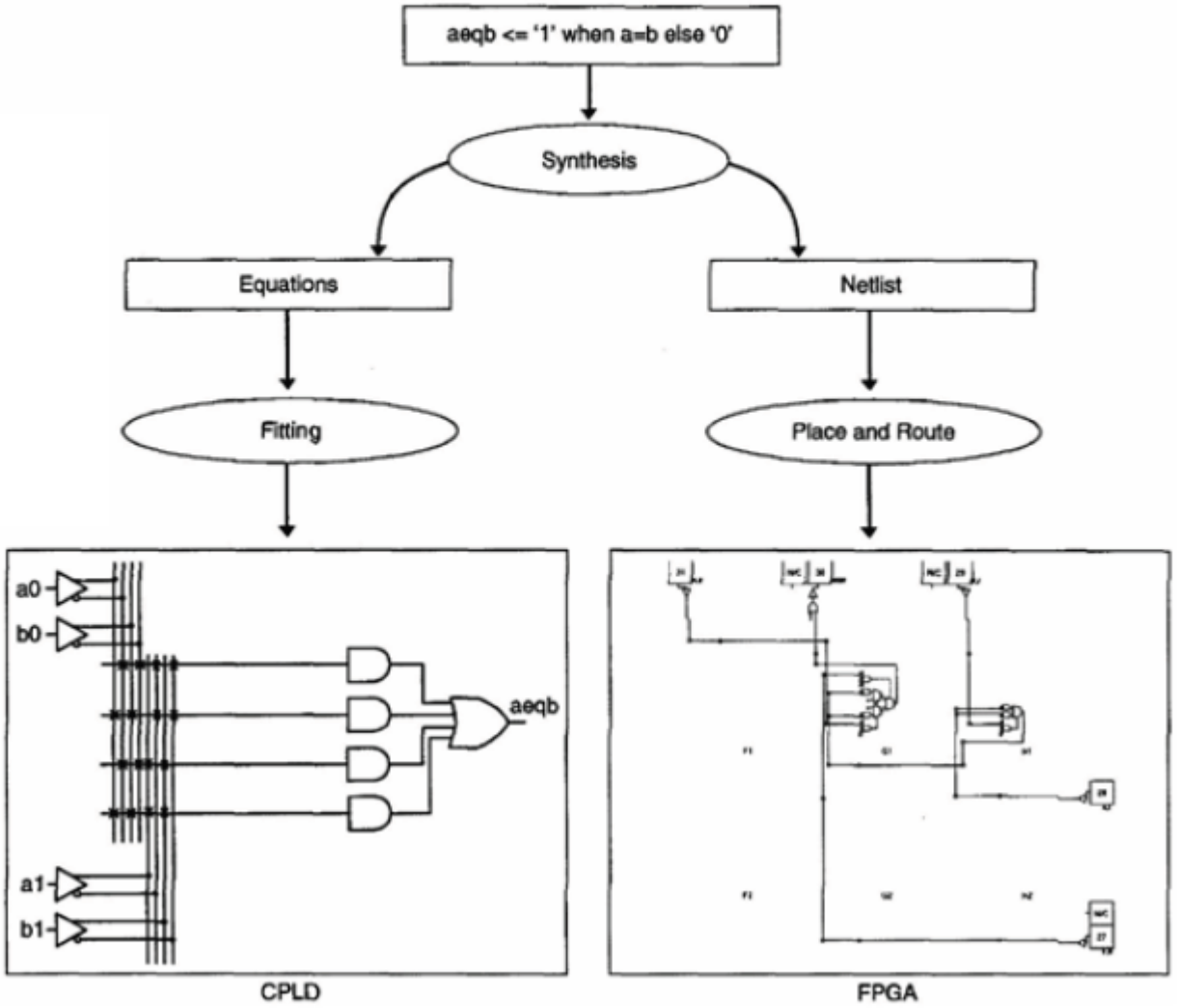


Boolean equations/Netlist



”101001110001”

Syntese til design



VHDL

UNIVERSITY OF OSLO

- VHDL = Very high-speed integrated circuit Hardware Description Language
- VHDL is an industry standard for description, modeling and synthesis of digital circuits and systems
- Introduced in 1981 for the Department of Defence (DoD)
- Became an IEEE standard in 1987

- **We will look at VHDL for synthesis of logic**

- **VHDL standards: VHDL 93,** 2000, 2002, 2007, 200x

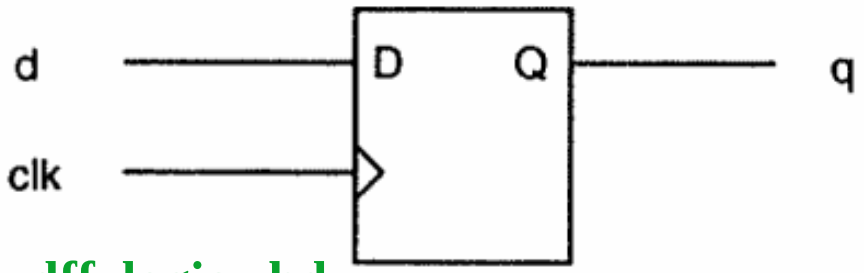
UNIVERSITY OF OSLO

Recommended free VHDL editors

- Notepad ++
  - http://notepad-plus.sourceforge.net/uk/site.htm
- EmacsW32
  - http://ourcomments.org/Emacs/EmacsW32.html

First VHDL example
D-flip-flop

UNIVERSITY OF OSLO




```
library ieee;
use ieee.std_logic_1164.all;
```

File name: **dff_logic.vhd**

```
entity dff_logic is port (
        d, clk : in std_logic;
        q      : out std_logic);
end dff_logic;
```

entity

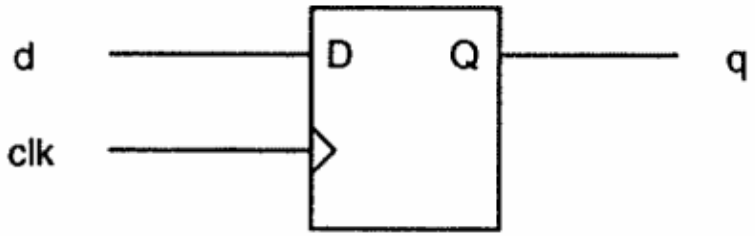
Note: the file name must be the same as the name of the entity!

```
architecture example of dff_logic is
begin
        process (clk) begin
                if (clk'event and clk = '1') then
                        q <= d;
                end if;
        end process;
end example;
```

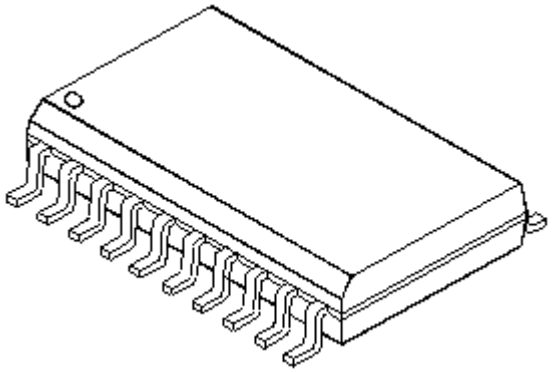
architecture

Entities and architectures

- Entity declaration and architecture body



- Compared with an IC:
  - **The entity describes the interface** (the connection pins of the package)
  - **The architecture describes the functionality of the entity** (the functionality of the circuit)



UNIVERSITY OF OSLO

Template - Entity/Architecture

entity *model_name* **is**
port
(
list of inputs and outputs
);
end *model_name*;

Same name as the file, e.g. **test.vhd**

architecture *architecture_name* **of** *model_name* **is**
begin
...
VHDL concurrent statements
....

end *architecture_name* ;

concurrent = samtidig

Comparator

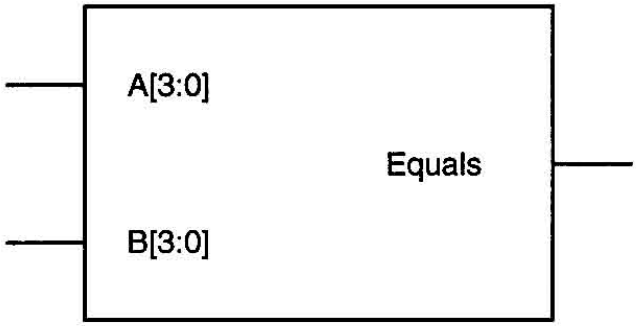


Figure 3-1 Schematic symbol equivalent of eqcomp4 entity

```
-- eqcomp4 is a four bit equality comparator
entity eqcomp4 is
port (a, b     :     in std_logic_vector(3 downto 0);
            equals:     out std_logic);
end eqcomp4;

architecture dataflow of eqcomp4 is
begin
     equals <= '1' when (a = b) else '0';
end dataflow;
```

MSB

[a(3) a(2) a(1) a(0)]
[b(3) b(2) b(1) b(0)]

<= ”settes lik”

Ports

UNIVERSITY OF OSLO

- Each port must have a name, direction (mode) and data type

```
entity add4 is port(
        a, b:  in std_logic_vector(3 downto 0);
        ci:    in std_logic;
        sum:   out std_logic_vector(3 downto 0);
        co:    out std_logic);
end add4;
```

name

mode

data type

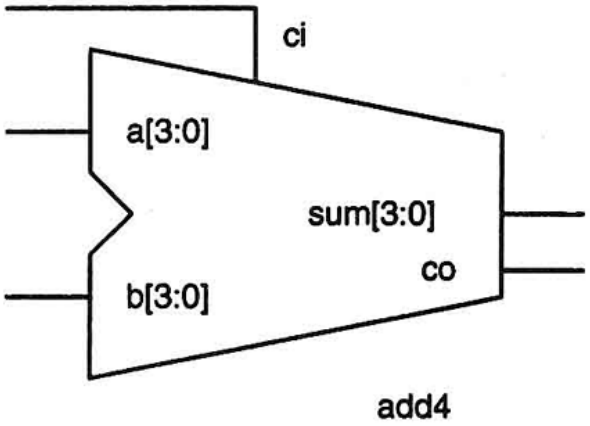


Figure 3-3 Symbol equivalent of entity add4

Name

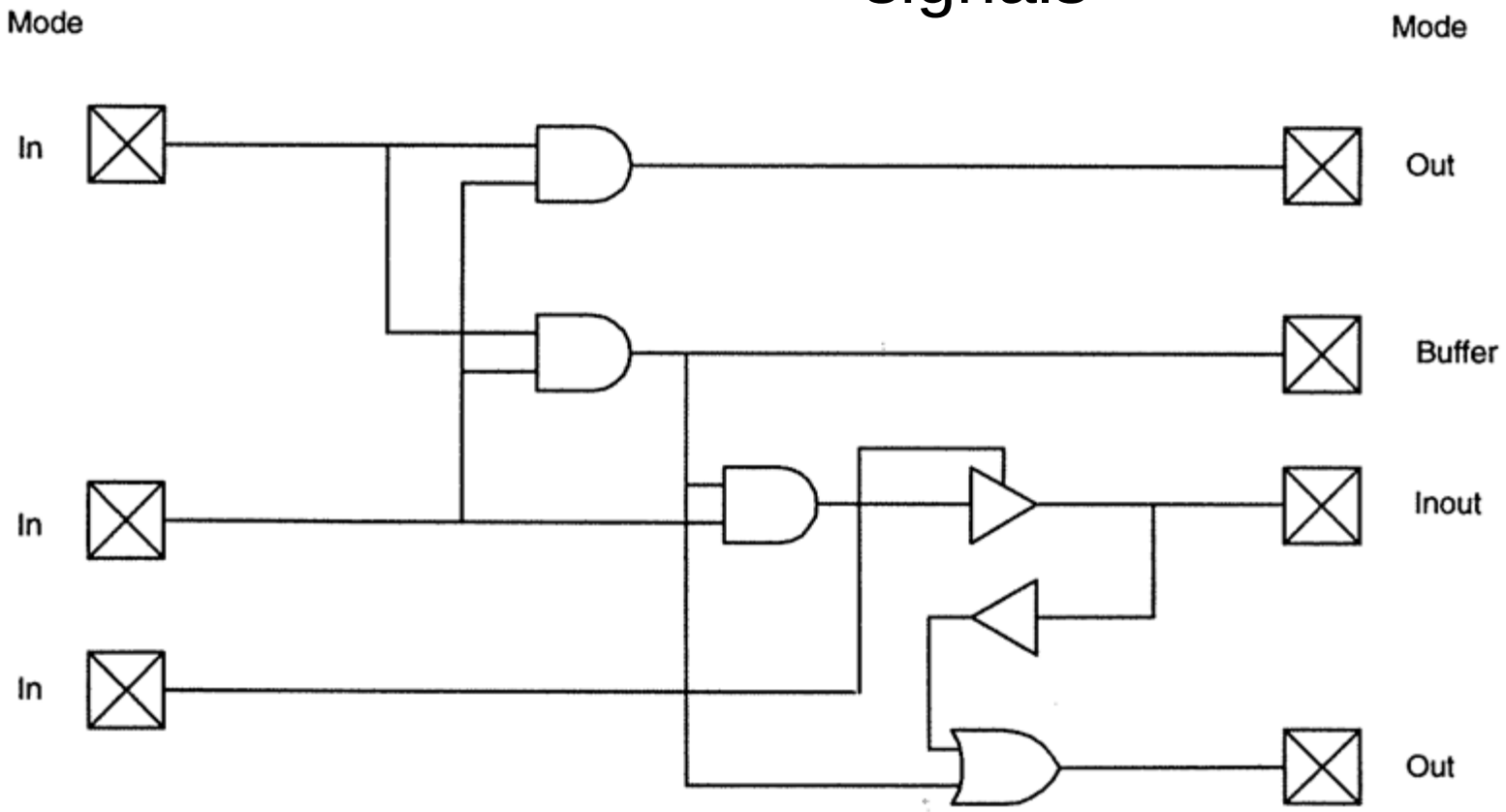
UNIVERSITY OF OSLO

- Names can be constructed using:
  - a b c….z (letters)
  - 0 1..9 (numbers)
  - _ (underscore)
- With the following reservations:
  - Always start with a letter
  - Can not use VHDL reserved words
  - Last character must be a letter or a number
  - Two following underscores are not allowed
  - Not case sensitive
    - TcK = tck

Direction (mode)

UNIVERSITY OF OSLO

- **In** – flow into the entity
- **Out** – flow out of the entity, ***no*** feedback
- **Buffer** - flow out of the entity, feedback allowed
- **Inout** - for bi-directional signals



Important Data types

UNIVERSITY OF OSLO

- bit, bit_vector (’1’ or ’0’)
- ieee.std_logic_1164:
  - **std_logic** (’U’, ’X’, ’0’, ’1’, ’Z’, ’W’, ’L’, ’H’,’-’)
  - **std_logic_vector** (e.g. ”010101”)
- The IEEE library must be made visible by *library* and *use*

for synthesis of logic

```
library ieee;
use ieee.std_logic_1164.all;
entity add4 is port(
        a, b:   in std_logic_vector(3 downto 0);
        ci:     in std_logic;
        sum:    out std_logic_vector(3 downto 0);
        co:     out std_logic);
end add4;
```

UNIVERSITY OF OSLO

The data type std_logic 1164

```
type std_ulogic is ( 'U',   -- Uninitialized
                     'X',   -- Forcing  Unknown
                     '0',   -- Forcing  0
                     '1',   -- Forcing  1
                     'Z',   -- High Impedance
                     'W',   -- Weak     Unknown
                     'L',   -- Weak     0
                     'H',   -- Weak     1
                     '-'    -- Don't care
                   );
```

← Three-state

9 different values!

UNIVERSITY OF OSLO

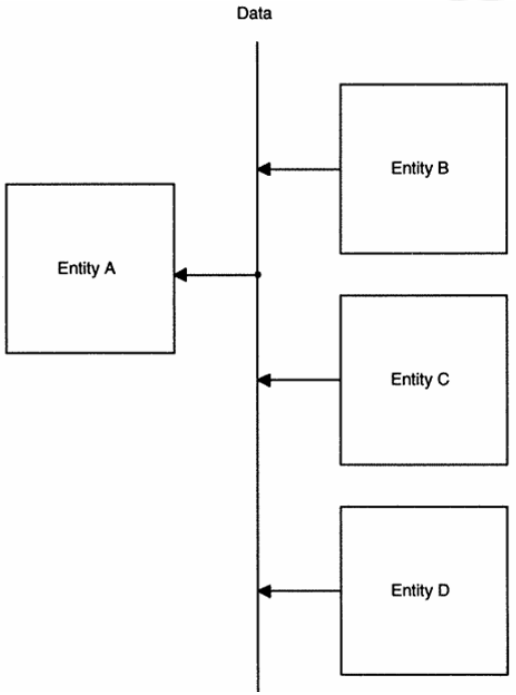
Std_logic 1164 resolution function

The sub type **std_logic** is ”resolved” **std_ulogic**. When two or more drivers are connected together the value is determined by a ”resolution table”

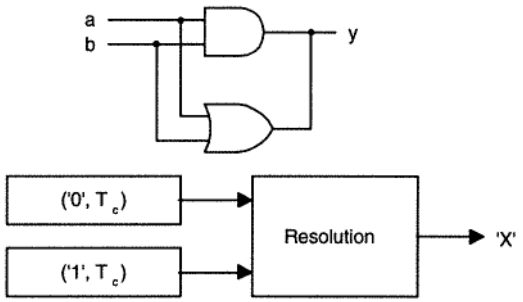
```
-- ---------------------------------------------------------
-- |  U    X    0    1    Z    W    L    H    -          |   |
-- ---------------------------------------------------------

   ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
   ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
   ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
   ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
   ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
   ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
   ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
   ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
   ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
```

Bus

- The resolution function is used to simulate a data bus
- Useful that the simulator can indicate an unknown value if two or more entities write to the same bus line at the same time with opposite logic values.
- **Two entities can not write to a bus line at the same time!**
- **If an entity write to the bus the other entities must be in three-state (high impedance) on their outputs**
- **The unknown value (‘X’) has no meaning for synthesis!**




```
architecture multiply_driven_signal of my_design is
begin
   y <= a and b;
   y <= a or b;
end multiply_driven_signal;
```



~~y <= ’X’~~

Arithmetic and logical operators

UNIVERSITY OF OSLO

Arithmetic operators:

- \+ Addition
- \- Subtraction
- * Multiplication
- / Division

Use with care, creates much logic

Logical operators:

- and, nand, or, nor, not, xor, xnor

Example of Hex-number:

X”FA” = ”11111010”

Logical operators

UNIVERSITY OF OSLO

- **and, or, not, nand, nor, xor og xnor** are predefined for **bit** and **boolean**
- IEEE 1164 uses these operators in **std_logic**
- Logical operators do not have precedence in VHDL, therefore parenthesis is demanded in multi level logic:
  - A + B • C is ok in Boolean algebra due to precedence
  - X <= A **or** B **and** C gives an error in VHDL
  - A **or** (B **and** C)
  - (A **or** B) **and** C

  } Correct for VHDL

Precedence in Boolean algebra:

()
not
and ·
or +

Relational operators

- equality =
- inequality /=
- Size operators < , <= , > , >=

- The operands must both be of the same type, and the result is a Boolean value (true/false)

Example:

```
signal a : std_logic_vector(7 downto 0);

      ..........

if a = 3 then
```

Gives an error, becasue ***a*** is ***std_logic***, while 3 is an ***integer***

Coding style (Architecture)

UNIVERSITY OF OSLO

Structural description

```
u1: xor2 port map(a(0), b(0), x(0));
u2: xor2 port map(a(1), b(1), x(1));
u3: nor2 port map(x(0), x(1), aeqb);
```

Netlists

Dataflow

```
aeqb <= (a(0) XOR b(0)) NOR
        (a(1) XOR b(1));
```

Boolean equations

```
aeqb <= '1' when a = b else '0';
```

Concurrent statements

Dataflow

```
if a = b then aeqb <= '1';
   else aeqb <= '0';
end if;
```

Sequential statements

Behavioral

”Process”

UNIVERSITY OF OSLO

- The process is executed when one of the signals in the sensitivity list has a change (an event)
- Then, the sequential signal assignments are executed
- The process continue to the last signal assignment, and terminates
- **The signals are updated just before the process terminates!**
- The process is not executed again before one of the signals in the sensitivity list has a new event (change)

```
proc1: process (a, b, c)
   begin
      x <= a and b and c;
   end process;
```

```
process (<sens list>)
< declaration>
begin
    <signal assignment1>
    .
    .
    <signal assignment n>
end process;
```

```
clk: process is  -- without sensitivity list
begin
clock <= ’0’;
wait for 50 ns;
clock <= ’1’;
wait for 50 ns;  -- wait needed!
end process;
```

Three-state buffers

UNIVERSITY OF OSLO

- The output buffer can be put into a high impedance (’Z’) state, such that only one entity writes to the bus
  - Three possible signal levels: ’0’, ’1’, ’Z’
- FPGAs and CPLDs have three-state buffers on the outputs (the signals defined as ***port*** in the entity)
- However, many programmable logic devices can not have three-state buffers internally on the circuit (on internal signals)

Three-state buffer

```
oes: process (oe, cnt)
  begin
    if oe = '0' then
        cnt_out <= (others => 'Z');
    else
        cnt_out <= cnt;
    end if;
  end process oes;

end archcnt8;
```

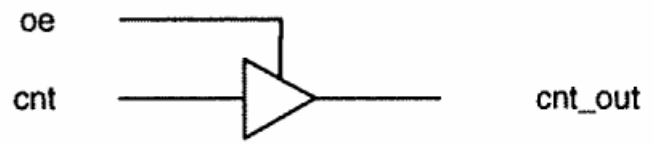
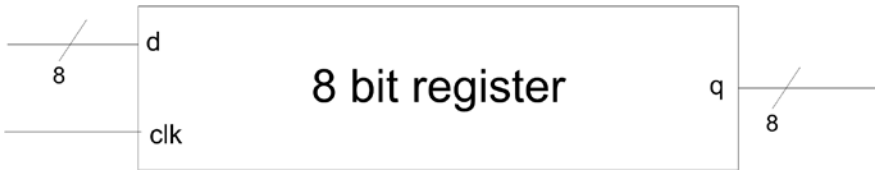


Figure 4-17 A three-state buffer

Or:

```
y <= output_signal when output_enable = '1' else 'Z';
```

8-bits register

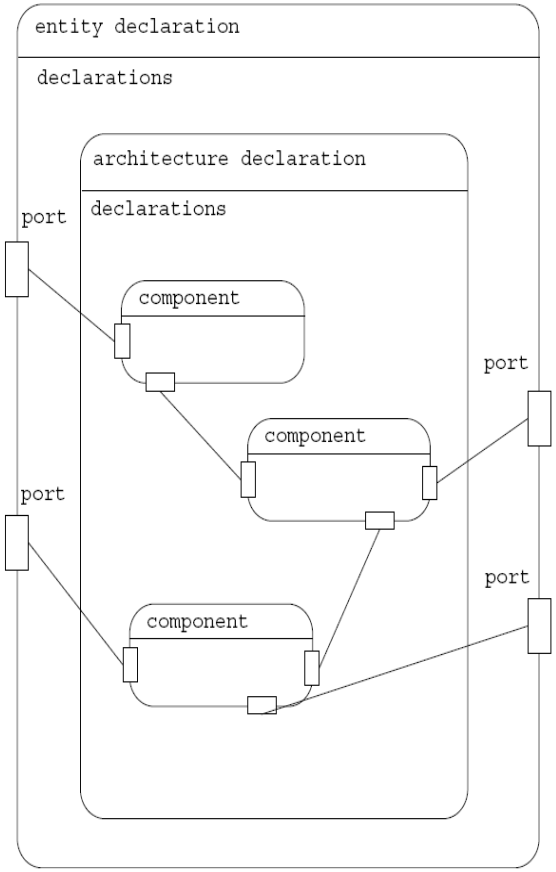



```
library ieee;
use ieee.std_logic_1164.all;
entity reg_logic is port (
        d   : in std_logic_vector(7 downto 0);
        clk : in std_logic;
        q   : out std_logic_vector(7 downto 0)
);
end reg_logic;
architecture r_example of reg_logic is
begin
        process (clk) begin
                if (clk'event and clk = '1') then
                        q <= d;
                end if;
        end process;
end r_example;
```

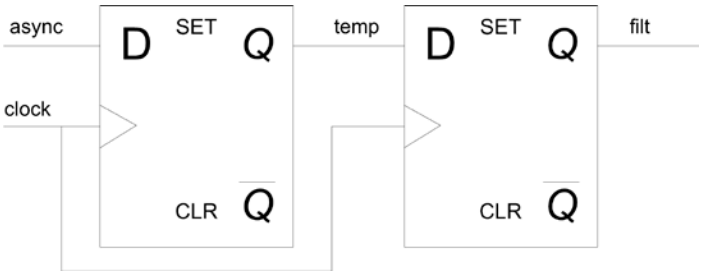
A new value is transferred to the q output on the rising clock edge

Component

- A component is an entity that is used in another entity



Use of components

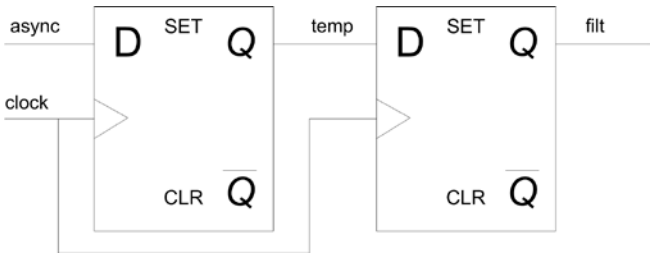


Fil: dflop.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity dflop i s port (
            d, clk : in std_logic;
            q      : out std_logic);
end dflop;

architecture arch_dflop of dflop is
begin
   process (clk) begin
      if (clk'event and clk = '1') then
         q <= d;
      end if;
   end process;
end arch_dflop;
```

```
entity test_dff is port (
            async, clock: in std_logic;
            filt                : out std_logic);
end test_dff;

architecture arch_test_dff of test_dff is
-- Component declaration
component dflop
port(
 d, clk  : in   std_logic
 q        :out std_logic);
end component;

-- Declaration of internal signals
signal temp : std_logic;

begin
-- Component instantiation
u1: dflop port map (async, clock, temp);
u2: dflop port map (temp, clock, filt);
end arch_test_dff ;
```

”Port map”




```
-- Position based
u2: dflop port map (temp, clock, filt);
```

```
-- Name based (component name to the left of the arrow)
u2: dflop port map (d => temp, clk => clock, q => filt);
```

Can not directly connect together the input/output of a component to another component’s output/input! Must use an internal signal (such as temp in this example), unless a connection to a port is made

input output

```
U3: navn
port map (a, b, c, open, d);
```

All *inputs* to a component must be connected! If an *output* is not needed, the reserved word **open** can be used

Direct Instantiation

- An alternative coding style
- Used in Zwolinski, see e.g. page 42-43, and page 49
- WORK = Current working directory
- No explicit component declaration before they are used (port map).

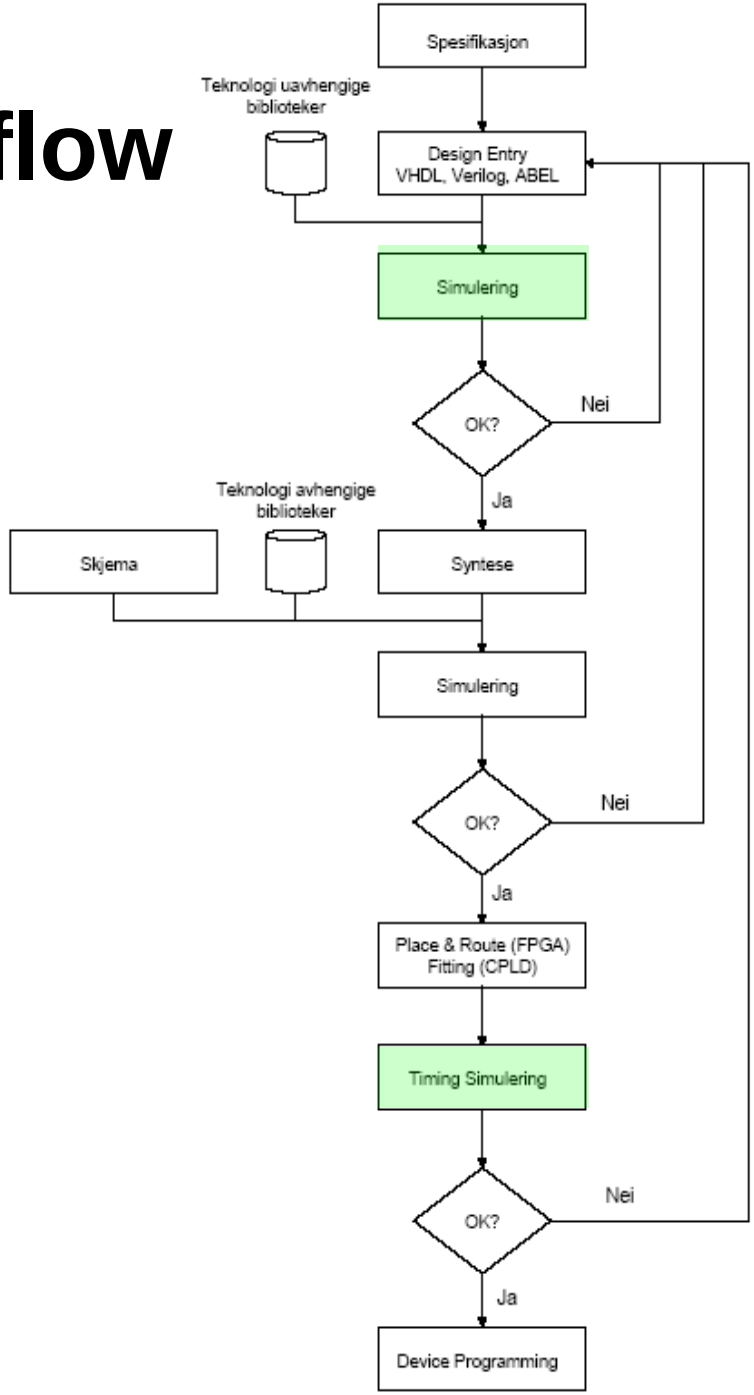
```
architecture netlist of comb_function is
  signal p, q, r : BIT;
begin
  g1: entity WORK.Not1(ex1) port map (a, p);
  g2: entity WORK.And2(ex1) port map (p, b, q);
  g3: entity WORK.And2(ex1) port map (a, c, r);
  g4: entity WORK.Or2(ex1) port map (q, r, z);
end architecture netlist;
```

Where the model is located

The name of the model's entity

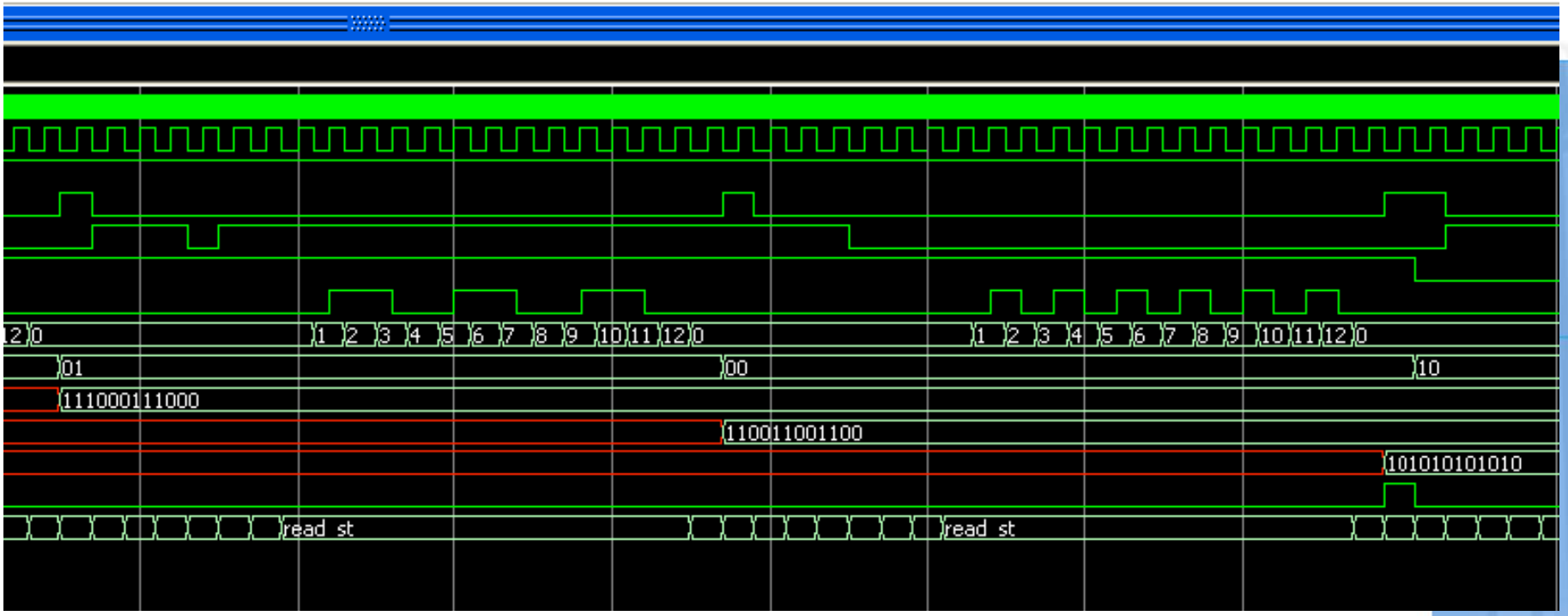
The name of the architecture; not need if only one architecture is related to this entity

Design flow

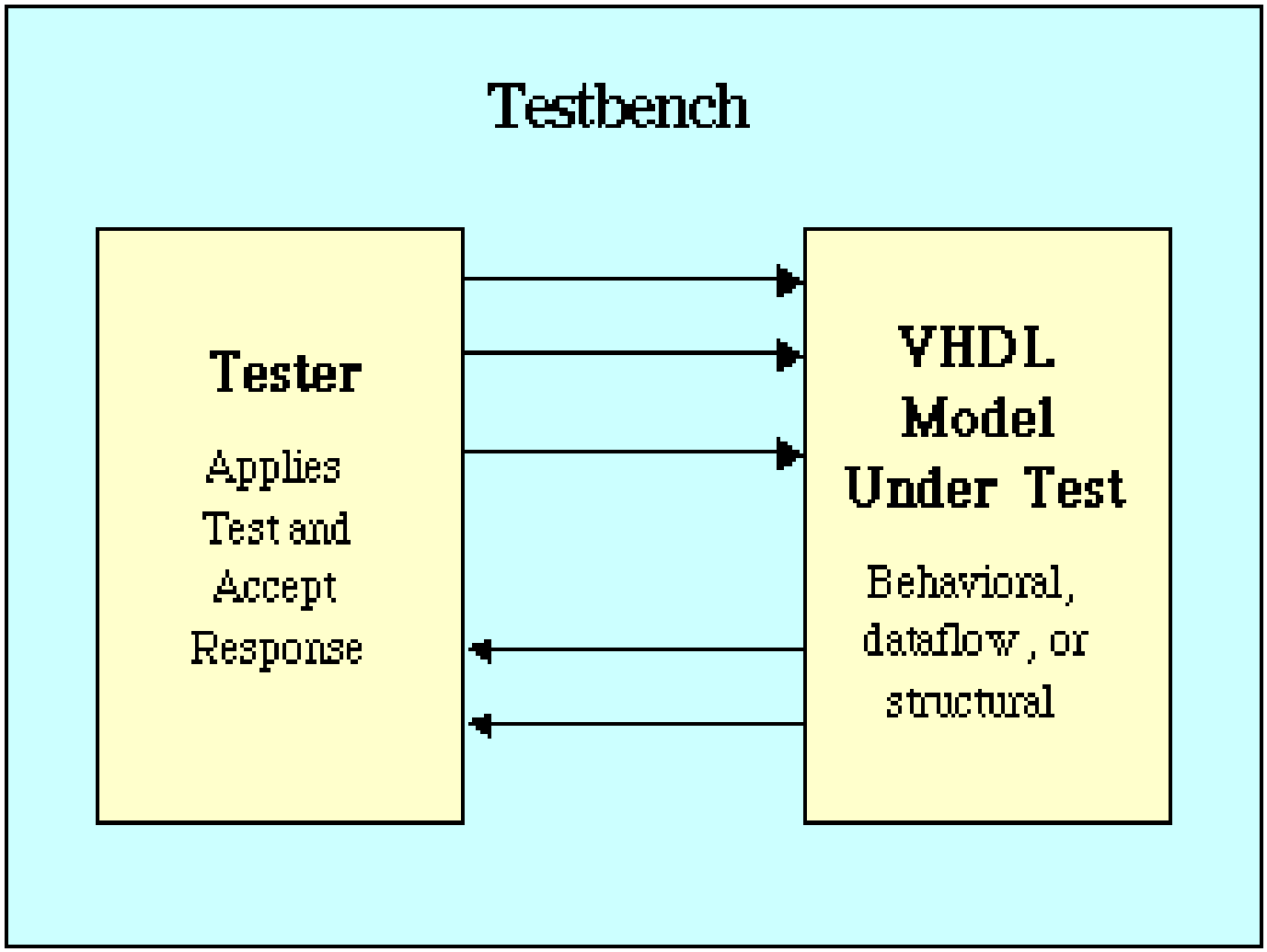

Spesifikasjon
Teknologi uavhengige biblioteker
Design Entry VHDL, Verilog, ABEL
Simulering
OK?
Nei
Ja
Teknologi avhengige biblioteker
Skjema
Syntese
Simulering
OK?
Nei
Ja
Place & Route (FPGA) Fitting (CPLD)
Timing Simulering
OK?
Nei
Ja
Device Programming

UNIVERSITY OF OSLO

Test vectors



Test bench


Testbench
Tester
Applies
Test and
Accept
Response
VHDL
Model
Under Test
Behavioral,
dataflow, or
structural

Test benches

UNIVERSITY OF OSLO

- **Add a stimuli (input) to the circuit under test, and observe the outputs to verify correct behavior/functionality**
- When a test bench has been made, a functional test can be repeated quickly after a design change
- The same test bench can be used to verify the VHDL-code functionality (RTL level), and to verify the functionality and timing after synthesis and fitting (simulation on post-fit VHDL-model generated by the design tool)
- Test benches are not to be synthesized, and can therefore use the entire VHDL language (e.g. **after**)

```
x <= '1' after 4 ns;
```

Testbench ”template”

UNIVERSITY OF OSLO

```
library ieee;
use ieee.std_logic_1164.all;

entity test_UUT is     -- empty entity
end test_UUT

architecture testbenk_arch of test_UUT is
component UUT:
port
(
                ……
);
end component;
signal  …….
signal  …….   :=’0’; -- start value for inputs
begin
U1: UUT
port map (……..);

STIMULI:
process
begin
…….
wait;
end process;
```

(UUT = Unit Under Test)

Component declaration

Defines a signal for each port in the UUT

Component instantiation

Add stimuli

Generating a Test Bench Template from Quartus II

UNIVERSITY OF OSLO

1. If you have not already done so, open an existing project
2. If you have not already done so, perform a full compilation
3. Specify Modelsim-Altera as the simulation tool under **Assignments – EDA tool settings - Simulation**
4. In the Processing menu, point to **Start**, then click **Start Test Bench Template Writer**. The test bench file is written to the location specified as the output directory for the tool you selected. The default is ***/<project directory>/*simulation*/<EDA simulation tool>***.

Testbench clock generation

Clock generation

The most important signal in any design is the clock. In the simplest case, a clock can be generated by inverting its value at a regular interval:

```
clock <= not clock after 10 NS;
```

If clock is of type `BIT`, the initial value is automatically `'0'`. If clock is of type `std_logic`, an initial value must be assigned at the time of declaration:

```
signal clock : std_logic := '0';
```

If this initialization is not performed, the initial value is `'U'`, and '**not** `'U'`' is also `'U'`.

The same effect can be achieved by explicitly assigning values to the clock within a process:

```
clk: process is
  begin
    clock <= '0';
    wait for 10 NS;
    clock <= '1';
    wait for 10 NS;
  end process clk;
```

Testbench example

UNIVERSITY OF OSLO

```
signal clk   : std_logic :='0';

begin



clk <= not(clk) after 50 ns;      -- gives a clock period of  100 ns

STIMULI:
process
begin
……......
reset <= '0', '1' after 100 ns;
cnt    <= "0000",  "1010" after 600 ns;
…….....
wait;
end process;
```

A process without a sensitivity list must have a **wait** at the end

A better way to write the testbench stimuli

```
process is
begin
  Cin <= '0';
  A <= "0000";
  B <= "0000";
  wait for 5 NS;
  A <= "1111";
  wait for 5 NS;
  Cin <= '1';
  wait for 5 NS;
  A <= "0111";
  wait for 5 NS;
  B <= "1111";
  wait for 5 NS;
  Cin <= '0';
  wait;
end process;
```

Test benches

UNIVERSITY OF OSLO

- Add a stimuli (input) to the circuit under test, using VHDL, and observe the outputs to verify correct behavior/functionality
- Can have a table with test vectors integrated into the test bench or in a separate file
- Test benches are not to be synthesized, and can therefore use the entire VHDL language (e.g. **after**)
- File I/O Package defined in IEEE 1076: **textio**
  - Read test patterns from file
  - Write results to file and compare manually with an answer file
  - The test bench can also read the answer file such that the test bench can compare the results and the correct answers

- Can build in models for external circuits on the PCB
  - demands correct modeling of the external circuits

Self-testing test benches

- In a self-testing test bench all outputs are checked against an answer, and the result of the simulation is reported as ”Ok” or ”Not Ok”.
- The advantage is that search in timing diagrams are not needed (saves time)
- Other people can more easily maintain the code
- However, it is a demanding task to make a self-testing test bench!