VHDL-Overview

Christophe Bobda

VHDL

VHDL: Very high speed integrated circuits **H**ardware **D**escription **L**anguage

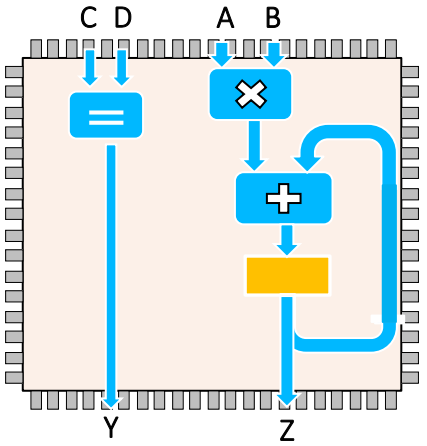
~~Very Hard Design Language~~

Hardware-Description Language with the goals

- Simulation
- IC synthesis

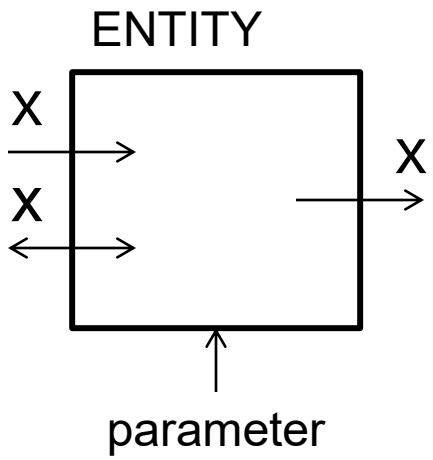
Description is done under the aspects

- Behavior
- Structure
- Hierarchy
- Parallelism



Only synthesizable Subset is considered !

Entity – Wrapper and interface definition



logic encapsulation of a unit Interface

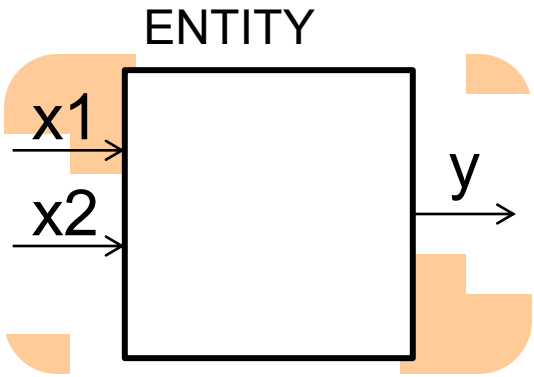
Definition: **Port** → X

Port are defined through **Identifier**,

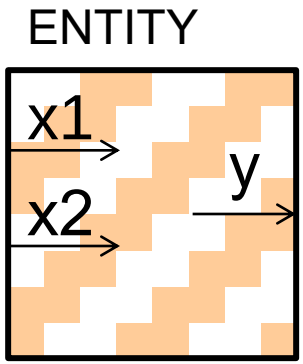
Direction und **Data-type**

Direction**: in, out, inout, buffer**

Data-type: bit, std_logic(_vector), ADT



View from outside



View from inside

Signals can be connected to ports from inside or from outside.

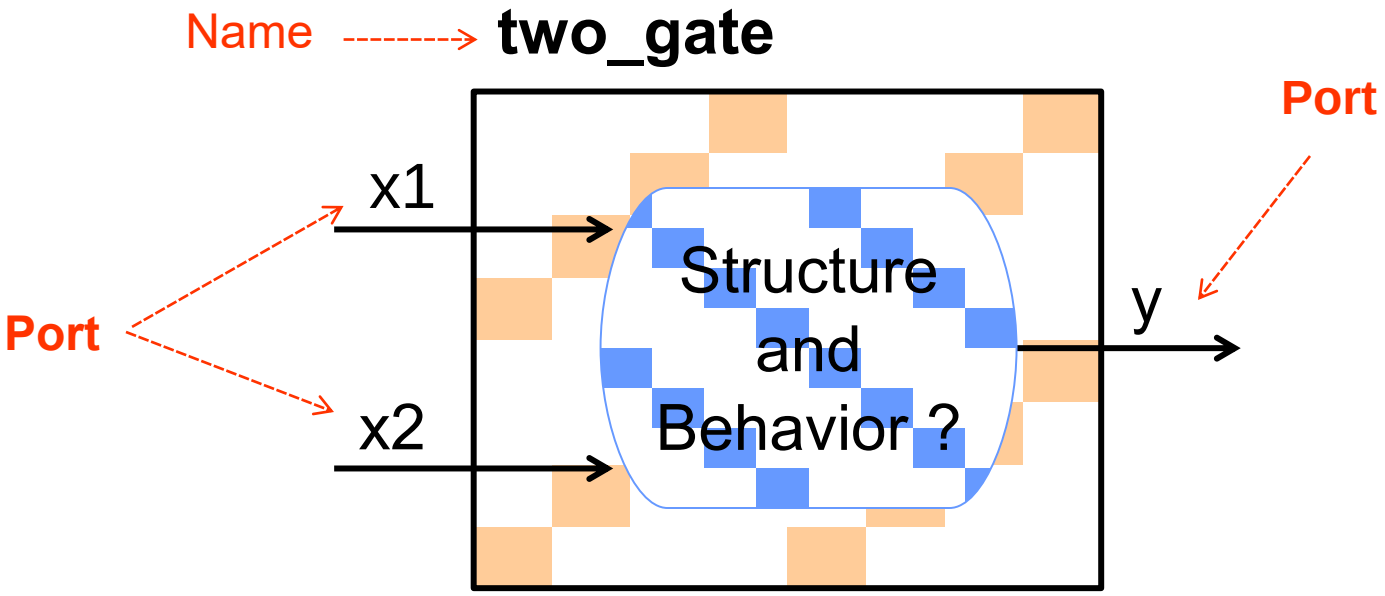
Entity as „Black Box“

Viewed from outside an entity is a „black box“

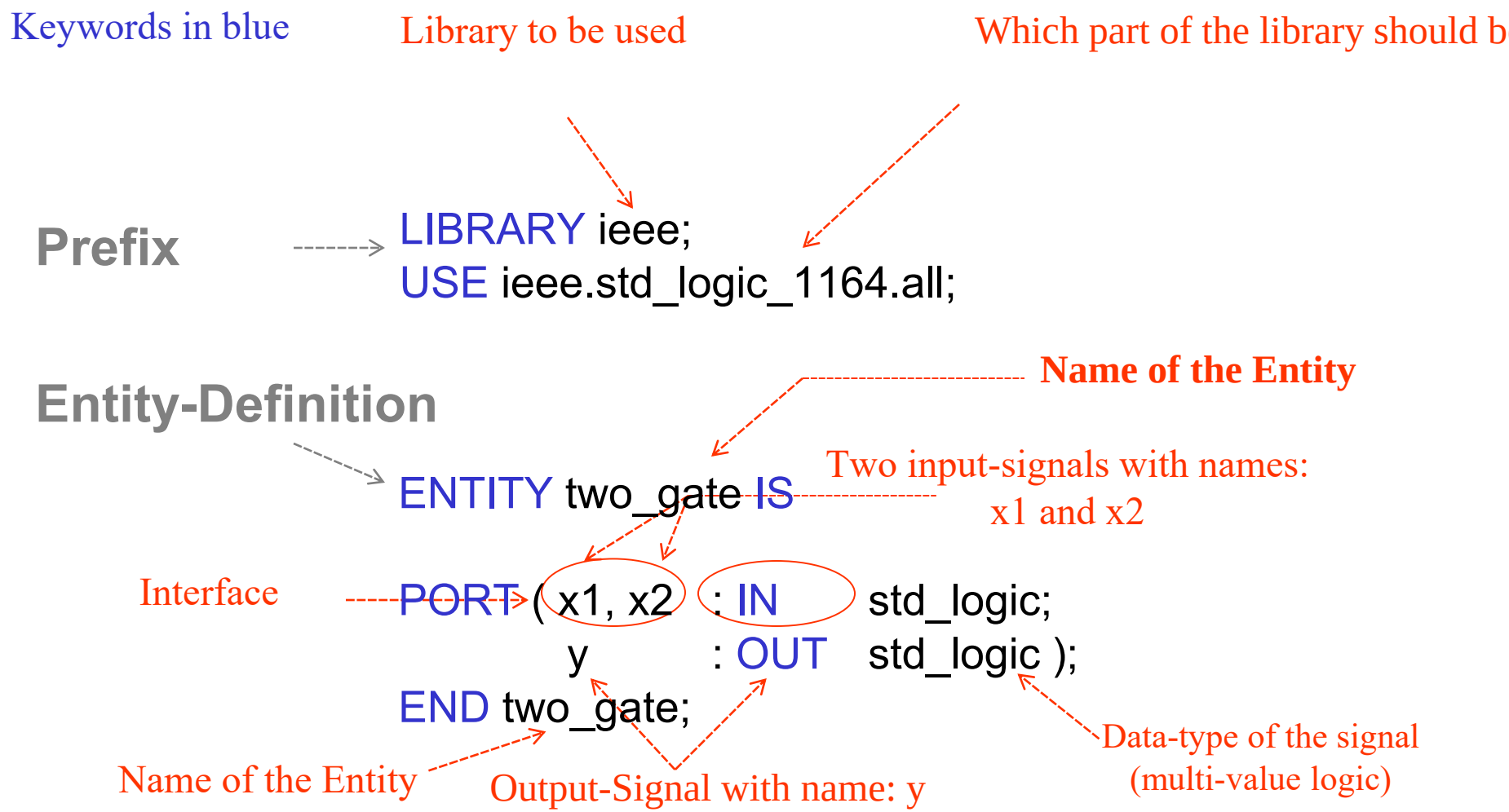
→ The structure and behavior (functionality) is hidden

Example below: Entity with two inputs and one output port

Functionality not defined yet



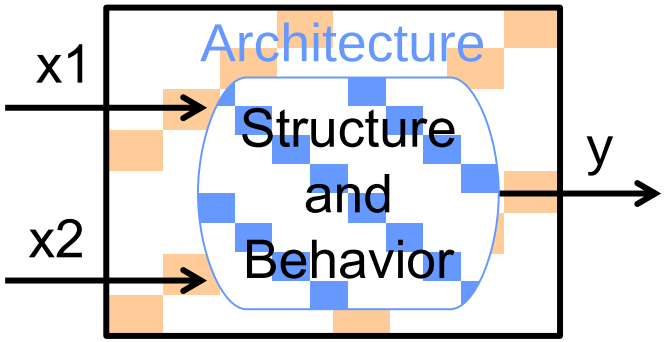
Entity – Definition



VHDL is case insensitive !

Architecture – Behavior of the Entity

The behavior of an Entity is described in the **Architecture**.



The Architecture

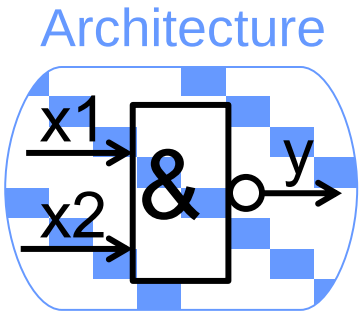
- has a unique name,
- signals and components can be internally declared,
- is assigned to a unique Entity.

```
ARCHITECTURE <Architecture name> OF <Entity-Name> IS
[Declarations]
BEGIN
    [Architecture Statements]
END <Architecture name>;
```

Architecture – Example

NAND-Gate

based on the previous Entity-Definition:



Signal assignment

Architecture name

Entity name

```
ARCHITECTURE two_gate_nand OF two_gate IS
BEGIN

    y <= NOT ( x1 AND x2 );

END two_gate_nand;
```

Behavioral description:

one statement with pre-defined and pre-implemented functions: AND, OR, NOT

Architecture name

XOR-Entity

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY xor_gate IS

PORT( x1, x2    : IN      std_logic;
          y     : OUT     std_logic );
END xor_gate;

ARCHITECTURE behavioral OF xor_gate IS
BEGIN

    y <=  x1 XOR x2;

END behavioral;
```

VHDL Operators

Signal Assignment: y <= x1

Boolean Operators:

- AND/and: logical and
- OR/or: logical or
- NAND/nand: logical complement of and
- NOR/ nor: logical complement
- XOR/xor: logical exclusive
- XNOR/ xnor : logical complement of exclusive

Random Boolean statement can be specified in VHDL.
Use parentheses to enforce precedence

```
y <=  (x1 XOR NOT(x2 XOR X3)) NAND y;
```

Process – Parallel statements

VHDL describes parallel processes

- All signal assignments take place simultaneously

VHDL		JAVA
A <= B; C <= D;	=	C <= D; A <= B;

VHDL		JAVA
A <= B; C <= A;	≠	A = B; C = A;

Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY my_circuit IS

PORT( a, b       : IN      std_logic;
          x, y, z    : OUT   std_logic );
END my_circuit ;

ARCHITECTURE behavioral OF my_circuit IS
BEGIN
    y <=  a XOR (a AND NOT(b));
    z <=  b OR NOT b;
    x <=  b OR NOT b;
END behavioral;
```

Design Flow

Modern design is done using CAD (computer-aided design) or EDA (Electronic Design Automation) tools

- FPGA: Xilinx Vivado, Intel Quartus
- VLSI: Synopsys DC, Cadence,

Design Flow: Lab1 will take you though the steps

- Design Entry
- Functional Simulation
- Synthesis (Pins and Library assignment, compilation)
- Post-synthesis simulation (timing analysis, verification)
- Place and Route (Layout)
- Fabrication/Bitstream Download (FPGA)

Signals

- A signal defines a connection in hardware. Can be used to connect components within an entity
- A signal has a datatype, which is the of possible values + operations on those values
- Signal Declaration

  SIGNAL *signal_name1[, signal_name2, …]* : *signal_type*;

- Declaration can be done in
  - Package
  - Port-section of an entity
  - Declaration section of an entity
- Signal Assignment is performed with symbol <=

  a <= b xor c

Internal Signals

```
ENTITY Position IS
PORT(   A          : in   std_logic_vector( 2 downto 0 );
        B          : out std_logic_vector( 0 to 2 )  );
END Position;

ARCHITECTURE behv OF Position IS
SIGNAL signal1, signal2: std_logic;
SIGNAL signal3 : std_logic_vector(4 downto 0);

Begin
signal1 <= A(0) AND B(2);
B(0) <= signal1 XOR A(1);
...
```

Variables

- A variable defines a memory location. Can be used to store values.
- A variable has a datatype, which is the set of all possible values + operations on those values
- Variable Declaration

  VARIABLE *var_name1[, var_name2, …]* : *variable_type*;

- Declaration can be done exclusively in a process
- Variable assignment is performed with symbol :=

  a := b xor c

```
PROCESS(a, b)
  VARIABLE c: bit;

  c := b and c;
```

Datatypes

Datatype

- Define a set of values a signal or a variable can take, along with operations on those values.
- Predefined datatype
  - bit (std_logic): '0', '1'
  - bit_vector (std_logic_vector): "00101", "001001"
  - integer: 10, 123, 56
  - character: 'a', 'v', 'n'
  - string: "anmggzem"
  - Boolean: FALSE, TRUE
  - real: 1.23, 234.6, -0.34
- User-defined datatype
  - SATES
  - CAN_PARAMETERS
  - NETWORK_PACKET
  - …

Multi-Value Logic

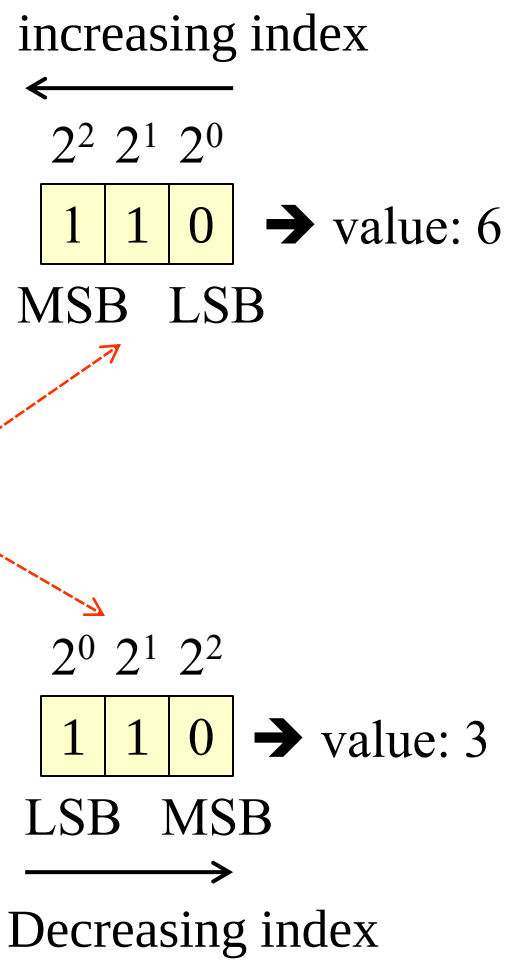
- Standard logic IEEE 1164
  - Declaration in Package „standard_logic_1164“
  - Possible values:
    - ´U´ not initialized
    - ´X´ unknown
    - ´0´ Logic 0
    - ´1´ Logic 1
    - ´Z´ high impedance
    - ´W´ unknown (weak signal)
    - ´L´ logic 0 (weak signal)
    - ´H´ Logic 1 (weak signal)
    - ´-´ Don't care
  - Standard_logic_vector: Vektor von standard_logic values

Bit vectors

Single signals can be grouped in a vector:

std_logic → std_logic_vector( )

Example: 3-bit coded position




```
ENTITY Position IS
PORT(   A       : in   std_logic_vector( 2 downto 0 );
        B       : out std_logic_vector( 0 to 2 )  );
END Position;
```

3-bit wide, binary coded Bit-vector,
write: x **downto** y means decreasing index from the MSB to the LSB
(MSB: most significant bit, LSB: least ...)

Bit Vectors

- Declaration
  - signal Z_Bus: bit_vector(3 downto 0);
  - signal Z_Bus: std_logic_vector(0 to 3);

- Assignment:
  - “By position”
    - Z_Bus(3) <= C_Bus(1);
    - Z_Bus(1) <= C_Bus(1);
    - Z_Bus(3 downto 2) <= “01“
  - “Concatenation”:
    - Z_Bus <= A & B & C & D;
  - “Aggregates”:
    - Z_Bus <= (A, B, C, D);
  - “Specifying by element index“:
    - Z_Bus <= (3 => ‘ 1‘, 1 downto 0 => ‘ 1‘, 2 => ‘ 0‘);
  - “others“:
    - Z_Bus <= (3 => ‘ 1‘, 1 => ‘ 0‘, other => ‘ 0‘);

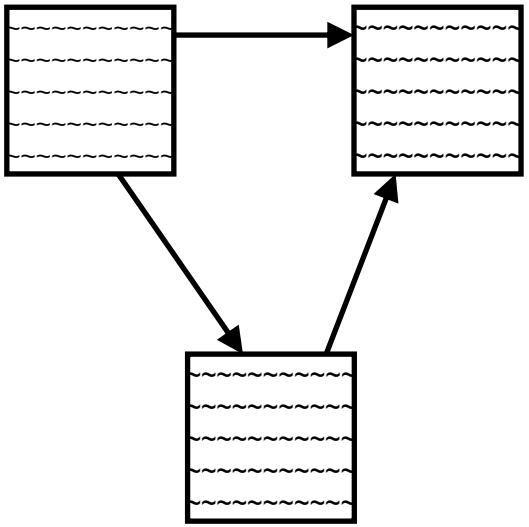
VHDL Operators

- Logic operators
  - AND, OR, NAND, NOR, XOR (equal precedence)
  - NOT (higher precedence)
- comparison
  - < (smaller than), <= (smaller or equal than), = (equal), >= (bigger or equal) , > (bigger than), /= (different)
- Arithmetic operators
  - + (Addition), - (subtraction), * (Multiplication), / (Division), ** (Exponent), abs (Magnitude), mod (Modulo), rem (Remainder)

Process – Parallel statements

Process execution:

- Section (module) in the architecture where instructions execute sequentially
- Multiple processes in same architecture execute in parallel
- Processes communicate with other components using signals
- A process has a sensitivity list. That is the list of all signals that activate the process when their value change (event)



Process – Declaration

Syntax of a process declaration

Sensitivity list:
driven Signals activate the process on change.
Analog to: „wait to see if x1 or x2 has change“
(alternative: internal wait-statement)

Inside an Architecture

Name of the Process (optional)

```
...
BEGIN
[label:] PROCESS ( x1, x2, ... )
BEGIN
    parallel Area;
END PROCESS [label];

END;
```

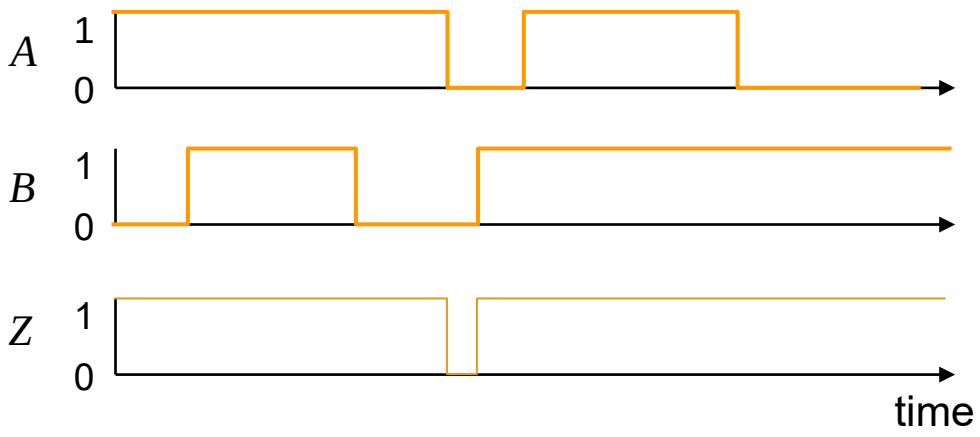
Behavior is evaluated in parallel !
Different to variables in traditional Programming language

Process name (optional)

VHDL		JAVA
A <= B; C <= A;	≠	A = B; C = A;

Process – Example

```
Entity Or is
   port (A, B: in bit;
         Z: out bit);
End or;

Architecture Behave of or is
Begin
-- process declaration
or_func:process (A,B)
-- A, B in the sensitivity list
begin
   if (A = ' 1' or B = ' 1' ) then
      Z <= ' 1' ;
   Else
      Z <= ' 0' ;
   End If;
End Behave;
```



Controll-flow

Branching

If and If - then - else – syntax

```
If condition 1 then
    sequential instructions
End if;
If condition 2 then
    sequential instructions
End if;
If condition 3 then
    sequential instructions
End if;
If condition 4 then
    sequential instructions
Else
    sequential inistructions
End if;
```

If - elsif – syntax

```
If condition 1 then
    sequential instructions
elsif condition 2
    sequentielle instructions
elsif condition 3
    sequential instructions
else
    sequential instructions
End if;
```

Controll-flow

Branching

```
Case Object is
    when value 1 => instructions
    when value 2 => instructions
            .....
    when value n => instructions
End case;
```

```
Interval differentiation
Case X is
    when 0 to 2 => instructions
    when value 1 => instructions
     .....
    when value n => instructions
End case;
```

```
Subset differentiation
Case X is
    when 0 to 2 => instructions
    when value 1 => instructions
    .....
    when value n => instructions
    when others  => instructions
End case;
```

Controll-flow

Branching

IF – THEN – ELSE

CASE

(**y** of type **std_logic:** 1, 0, Z, X, …)

```
LOGIC : PROCESS ( x1, x2, x3 )
BEGIN
    IF ( x1 = '1' ) THEN
        y <= '0' ;
    ELSIF ( x2 = '1' ) THEN
        y <= '1' ;
    ELSE
        y <= 'Z' ;
    END IF;
...
```

Not elseif

```
...
CASE x3 IS
    WHEN '0' =>
        y <= '0' ;
    …
    WHEN others =>
        y <= '1' ;
END CASE;
END PROCESS LOGIC;
```

Default case

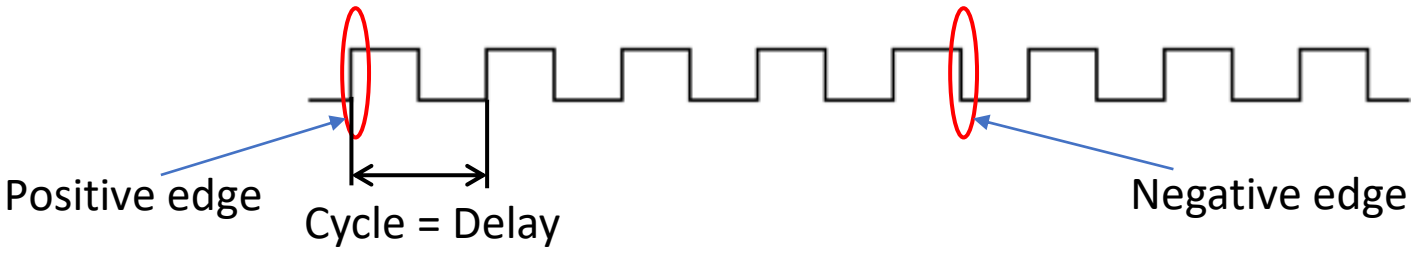
Synchronous Circuits

Synchronous circuits: outputs are correct with the value of the clock

Clock: special signal with alternating 1 and 0 values

Clock Cycle: Distance between two edge

Edges can be positive or negative



VHDL: Positive edge ((clock'event) AND (clock = '1'))

VHDL: negative edge ((clock'event) AND (clock = '0'))

Hierarchy

Hierarchy (structural description)

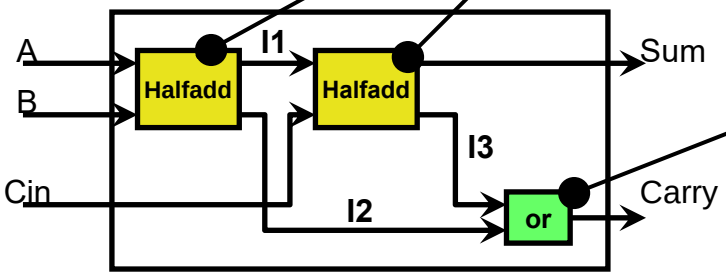
- Components (modules, submodules) are instantiated in a high level (top level) design to build more complex structures
- **Example: Full Adder**
  - Instantiation of 2 half adders and 1 XOR-Gate

In the same or separate files of the same project

Name file after the entity

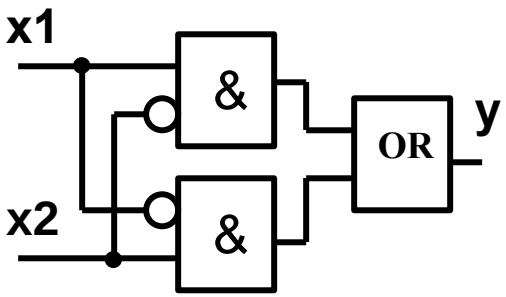
```
entity fulladd is
   port (A, B, Cin: in bit;
         Sum, Carry: out bit);
End fulladd;
```

```
entity Halfadd is
   port (A, B: in bit;
         Sum, Carry: out bit);
End Halfadd;
```




```
Architecture structural of fulladd is
   Component Halfadd is
      port (A, B: in bit;
            Sum, Carry: out bit);
   End component;
   Component or is
      port (A, B: in bit;
            Z : out bit);
   End component;
   Signal I1,I2,I3: bit;
Begin
   H1: Halfadd port map(A, B, I1, I2);
   H2: Halfadd port map(Cin, I1, sum, I3);
   Or1: or port map(I3, I2, carry);
End structural;
```

Use of components

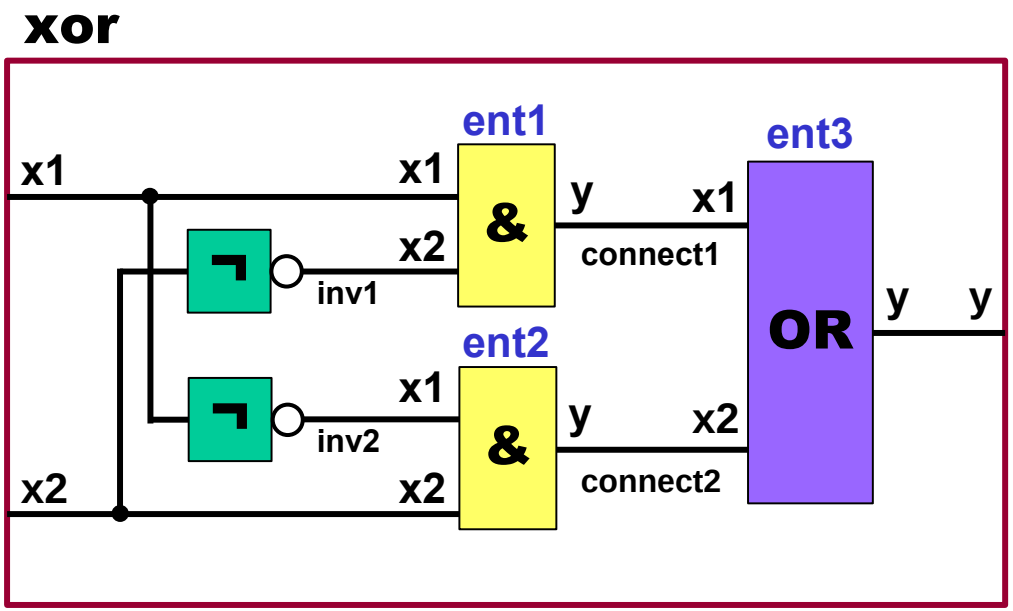


Example: XOR

Structural description

Ports: 2 inputs and 1 output

Reuse of **two_gate**, here with names **ent1, ent2** and **ent3**



Components

ent1, ent2 (AND-Gate)

ent3 (OR-Gat)

inv1, inv2 (Inverter)

XOR-Entity

Entity-Declaration

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY xor_gate IS

PORT( x1, x2    : IN     std_logic;
      y         : OUT    std_logic );

END xor_gate;
```

XOR – Components Declarations

Architecture-Declaration

Use of pre-implemented components

```
ARCHITECTURE xor_behavior OF xor_gate IS

COMPONENT and_gate
    PORT ( x1, x2: in std_logic;  y: out std_logic );
END COMPONENT;

COMPONENT or_gate
  PORT ( x1, x2: in std_logic;  y: out std_logic );
END COMPONENT;

SIGNAL connect1, connect2: std_logic;
SIGNAL inv1, inv2: std_logic;

BEGIN
...
```

XOR – Components Instantiation

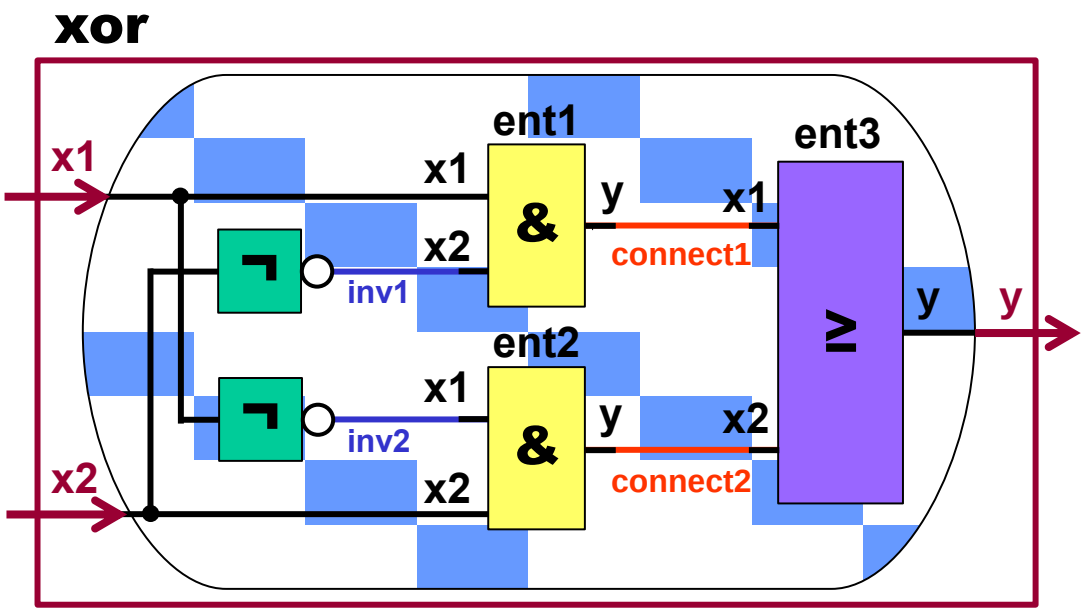
Instantiation and component wiring

Type of the Instance

Wiring: **Internal => External**

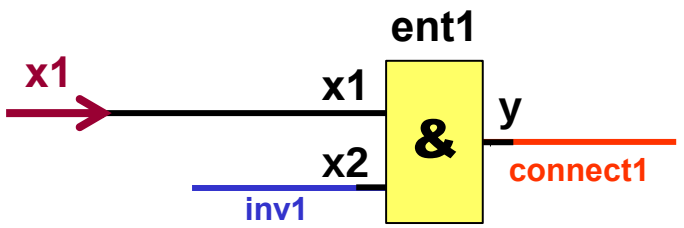
```
...
ent1: and_gate PORT MAP (x1 => x1,       x2 => inv1,     y => connect1 );
ent2: and_gate PORT MAP (x1 => inv2,     x2 => x2,       y => connect2 );
ent3: or_gate PORT MAP (x1 => connect1,  x2 => connect2, y => y );
...
```

Instance

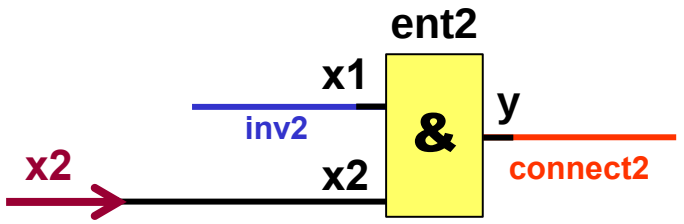


XOR- Component Wiring

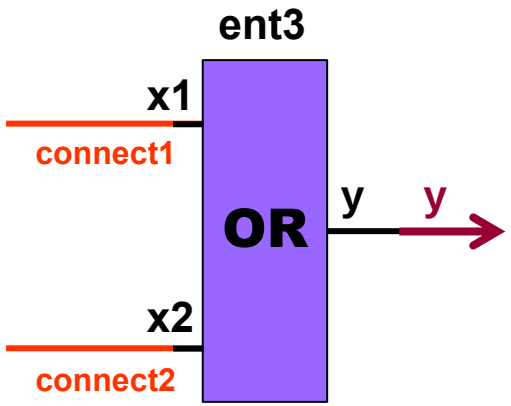
```
ent1: two_gate
  PORT MAP ( x1 => x1,
             x2 => inv1,
             y  => connect1 );
```




```
ent2: two_gate
  PORT MAP ( x1 => inv2,
             x2 => x2,
             y  => connect2 );
```




```
ent3: two_gate
  PORT MAP ( x1 => connect1,
             x2 => connect2,
             y  => y );
```



Line break is not required !!

XOR-Architecture

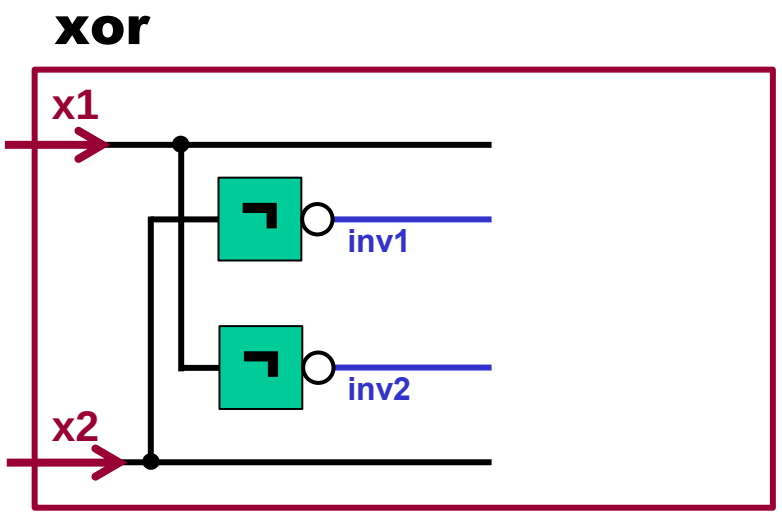
Inversion of the input-signals

Process name

Sensitivity-List

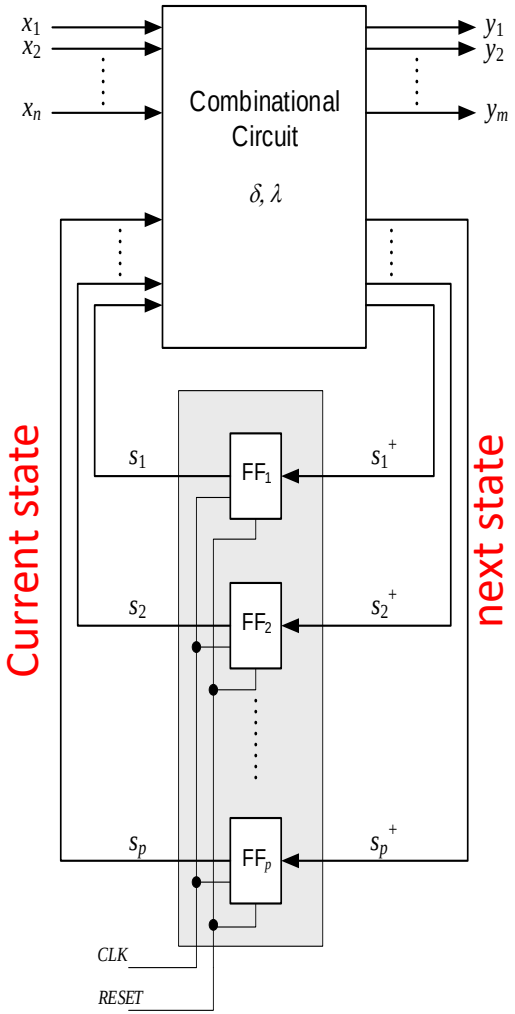
```
...
INV : PROCESS ( x1, x2 )
BEGIN
        inv1 <= NOT x2;
        inv2 <= NOT x1;
END PROCESS INV;

END xor_arch;
```

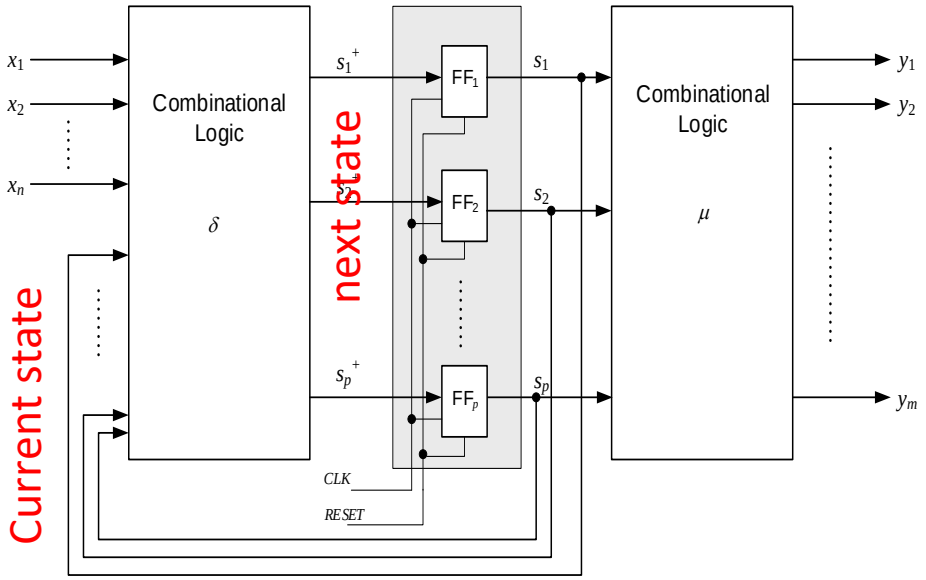
Assignment of the inverted input-signals



Automata - Finite State Machine

Combinational block + Sequential (Mealy or Moore)





Automata - Finite State Machine

- Combinational block + Sequential (Mealy or Moore)
- 2 different processes

```
TYPE state_type IS ( S0, S1 ) ;
SIGNAL current_state, next_state: state_type ;

Comb : PROCESS ( current_state )
BEGIN
    CASE current_state IS
        WHEN S0 =>
            next_state <= S1 ; ...
        WHEN others =>
            next_state <= S0 ; ...
    END CASE;

END PROCESS Comb;
...
```

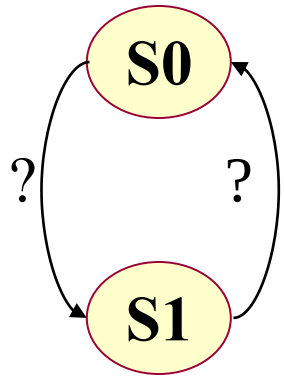
Definition of the state vect

This will connect the input and output of your FF

Set the next state

Here also output !!

Default case



Automata - Finite State Machine

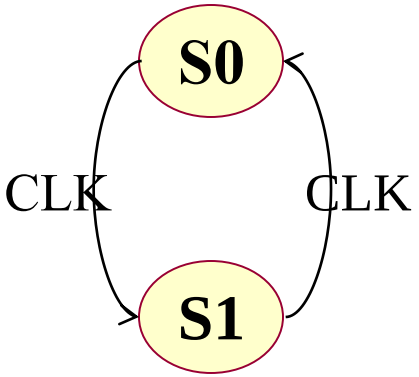
- Next-State-Assignment in a separate process
- Asynchronous Reset → Reset to state „S0“ (intial state)
- Synchronous state transition on rising edge of the Clock-Signal „CLK“

```
...
Synch : PROCESS ( CLK, RESET )
BEGIN

    IF ( RESET = '1' ) THEN -- define an asynchronous reset
        CURRENT_STATE <= S0;
    ELSIF ( CLK'EVENT and CLK = '1' ) THEN
        CURRENT_STATE <= NEXT_STATE;
    END IF;

END PROCESS Synch;
```

rising CLK-edge

State transition

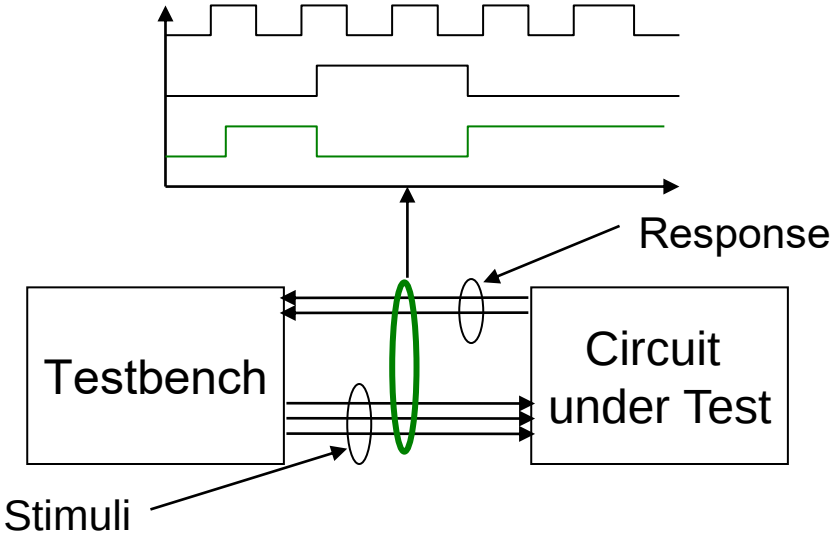
Implicit Instantiation Of FFs



Functional Simulation

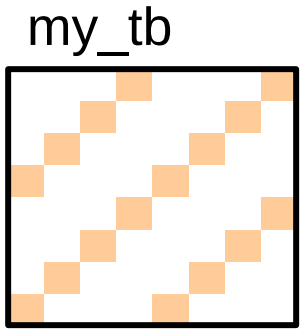
Used to check functional correctness of the circuit

- A testbench consist of the design under test (DUT) and the framework to generate stimuli and verify result
  - The correctness of the circuit is verified in a ”Waveform-Viewer“



Testbench-Entity

Empty wrapper which instantiates the entity under test



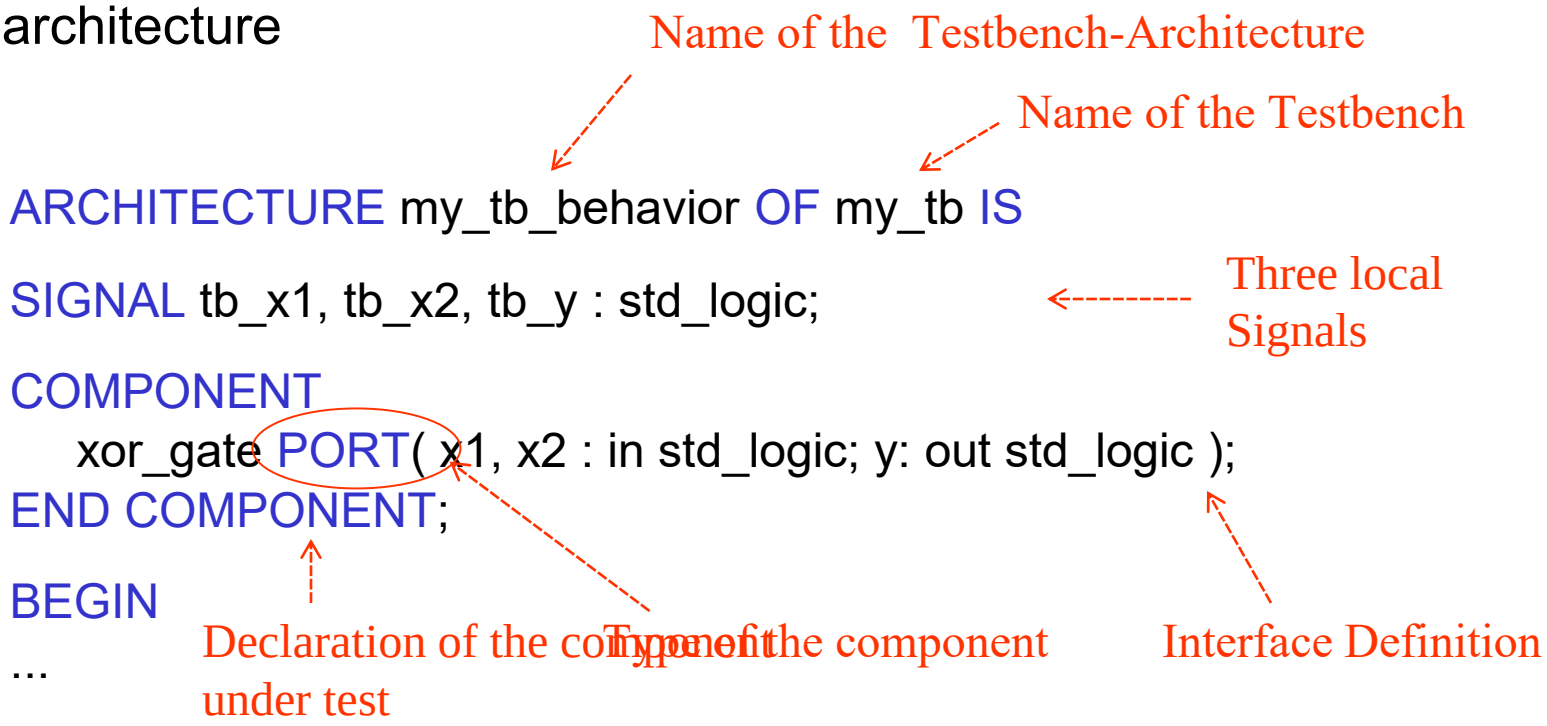

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY my_tb IS

END my_tb ;
```

No Port-Definition

Name of the Testbench

Testbench-Architecture

- Declaration of the component under test
- Local signal to connect the component under test
- Signals and csomponents declaration at the beginning of the architecture



Testbench-Architecture

Instantiation and component wiring

Component instantiation

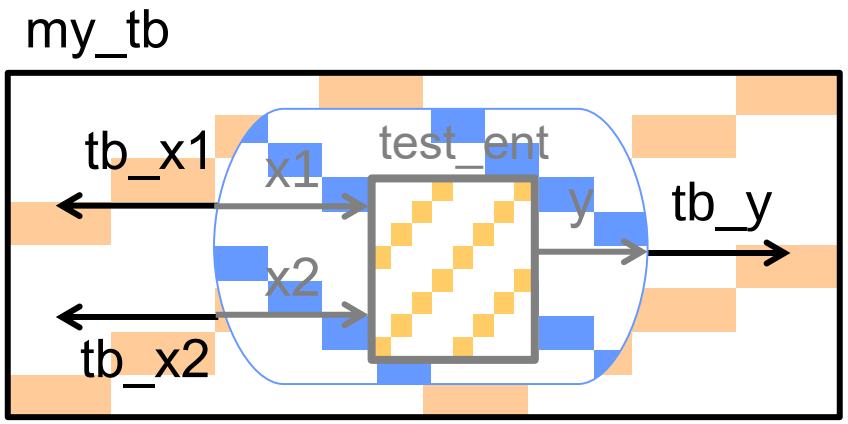
Component's type

Component-Port

```
...
BEGIN
        test_ent : xor_gate
            PORT MAP (x1 => tb_x1, x2 => tb_x2, y => tb_y );
...
```

local signal

wiring:
Component-Port => local signal



Testbench-Architecture

- The component under test is known
- Generate the Stimuli in a process
- Delay statement are **not** synthesizable. Dont worry!!! The testbench will never be synthesized !

```
...
stimuli : PROCESS
BEGIN
        tb_x1 <= '0';
        tb_x2 <= '0';
        WAIT FOR 10 ns;
        tb_x1 <= '1';
        WAIT FOR 10 ns;
        tb_x1 <= '0' AFTER 5 ns ;
....
END PROCESS stimuli;
```

Signals and value assignment

Only tb_x1 contains new values,

tb_x2 maintain old values

Drive the signals for 10 ns

Set this values after 5 ns

Entry in Signal-Schedule List

Timing-Behavior

Timing can be modelled differently

- wait for
- after
- Sensitivity-list

Use of **Generic**-Parameters

Here: delayed assignment (Start-delay: *del_time: time*)

Parameter for time-delay with default value

```
ENTITY two_gate IS
GENERIC( del_time: time := 2 ns );
        PORT( x1, x2: in std_logic; y: out std_logic );
END two_gate;

ARCHITECTURE two_gate_del_nand OF two_gate IS
BEGIN
    y <= NOT ( x1 AND x2 ) AFTER del_time;
END two_gate_del_nand ;
```

delayed assignment after 2ns

Simulation – vhdldbx

vhdldbx as Simulator (Synopsys)

- Selection of the entity under test
- **Hierarchy Browser**
- **Wave-Form-Viewer**

Alternative:

- **Modelsim** (Mentor Graphics- Free version for student)

Stimuli from the testbench (Test-Input)

Simulation result is the waveform of the reaction of component's signals to the stimuli.

The „Mean“ Vending Machine

```
entity G_Automat is
port (clk,reset, E, C,S7, R: in std_logic;
      GC, G7, M : out std_logic);
end G_Automat;

architecture Behavioral of G_Automat is
Type state_type is (I, H);
Signal current_state, next_state: state_type;
Begin
     comp:process (E, C, S7, R, current_state)
       begin
          case current_state is
                when I =>
                  if (E = '1') then next_state <= H; else next_state <= I; end if;
                when  H =>
                   if (C= '1') then next_state <= I; M <= '0' ; GC <= '1' ;  G7 <= '0';
                   elsif (S7 = '1') then next_state <= I; M <= '0' ; G7 <= '1' ;  G7 <= '0';
                   elsif (R = '1') then next_state <= I; M <= '1' ; G7 <= '0'; G7 <= '0';  GC <= '0';
                   elsif (E = '1') then  next_state <= I; M <= '0' ; G7 <= '0';  GC <= '0' ;
                   else next_state <= H; M <= '0' ; G7 <= '0';  GC <= '0'; end if;
        end case;
....
```

The „Mean“ Vending Machine Combinational Block

```
entity G_Automat is
port (clk,reset, E, C, S7, R: in std_logic;
      GC, G7, M : out std_logic);
end G_Automat;

architecture Behavioral of G_Automat is
Type state_type is (I, H);
Signal current_state, next_state: state_type;
Begin
     comp:process (E, C, S7, R, current_state)
       begin
          case current_state is
                when I =>
                  if (E = '1') then next_state <= H; else next_state <= I; end if;
                when  H =>
                   if (C= '1') then next_state <= I; M <= '0' ; GC <= '1' ;  G7 <= '0';
                   elsif (S7 = '1') then next_state <= I; M <= '0' ; G7 <= '1' ;  G7 <= '0';
                   elsif (R = '1') then next_state <= I; M <= '1' ; G7 <= '0'; G7 <= '0';  GC <= '0';
                   elsif (E = '1') then  next_state <= I; M <= '0' ; G7 <= '0';  GC <= '0' ;
                   else next_state <= H; M <= '0' ; G7 <= '0';  GC <= '0'; end if;
        end case;
....
```

The „Mean“ Vending Machine Register Block

```
....

        reg: process(clk, next_state, reset)
        begin
                if (reset = '1') then current_state <= I;
                        elsif (clk'event and clk = '1') then
                                current_state <= next_state;
                        end if;
                end process;
end Behavioral;
```

The „Mean“ Vending Machine Testbench

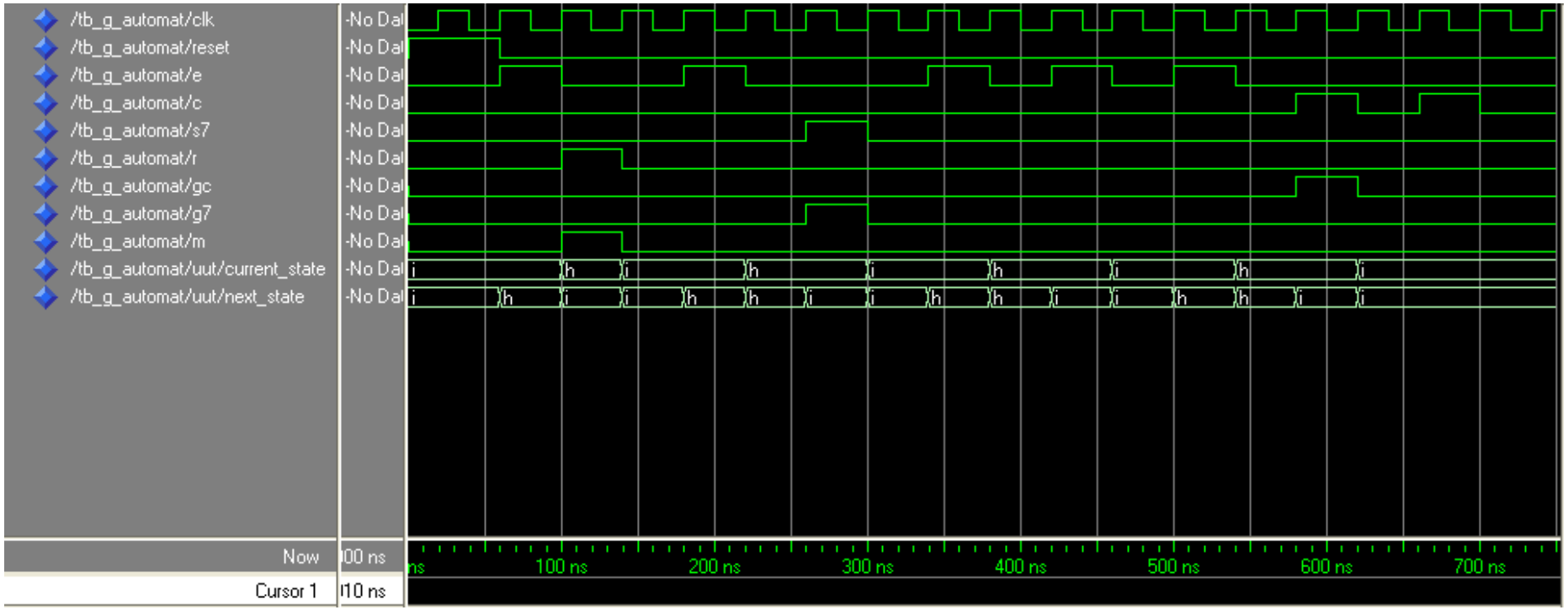
```
ENTITY tb_g_automat IS
END tb_g_automat;

ARCHITECTURE testbench_arch OF tb_g_automat IS
    COMPONENT g_automat
        PORT (
            clk, reset, E, C, S7, R : In std_logic;
            GC, G7, M : Out std_logic
        );
    END COMPONENT;

    SIGNAL clk : std_logic := '0';
    SIGNAL reset : std_logic := '0';
    SIGNAL E : std_logic := '0';
    SIGNAL C : std_logic := '0';
    SIGNAL S7 : std_logic := '0';
    SIGNAL R : std_logic := '0';
    SIGNAL GC : std_logic := '0';
    SIGNAL G7 : std_logic := '0';
    SIGNAL M : std_logic := '0';

    BEGIN
        UUT : g_automat
        PORT MAP ( clk => clk, reset => reset, E => E,
                   C => C, S7 => S7, R => R, GC => GC,
                   G7 => G7, M => M );
...
```

```
...
 PROCESS    -- clock process for clk
       BEGIN
           CLOCK_LOOP : LOOP
               clk <= '0';
               WAIT FOR 20 ns;
               clk <= '1';
               WAIT FOR 20 ns;
           END LOOP CLOCK_LOOP;
       END PROCESS;

       PROCESS
        BEGIN
               reset <= '1';
               WAIT FOR 55 ns;
               reset <= '0';
               E <= '1';
               WAIT FOR 40 ns;
               E <= '0';
               R <= '1';
               WAIT FOR 40 ns;
               R <= '0';
               WAIT FOR 40 ns;
               E <= '1';
               WAIT FOR 40 ns;
               E <= '0';
               WAIT FOR 40 ns;
```

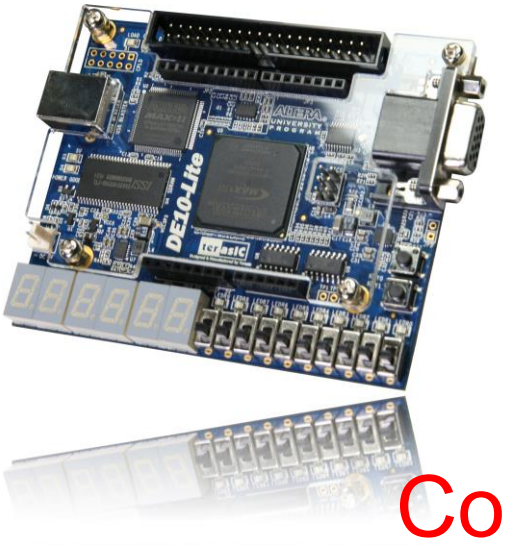
The „Mean“ Vending Machine

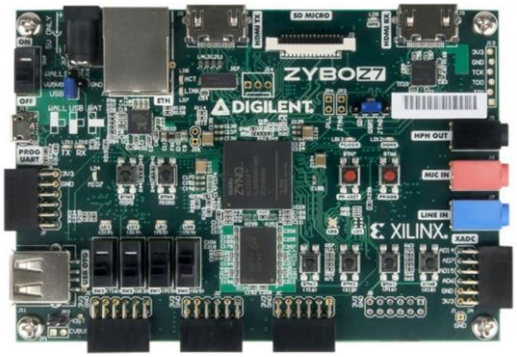
```
...                                                    ...
            S7 <= '1';
             WAIT FOR 40 ns;
             S7 <= '0';
             WAIT FOR 40 ns;
             E <= '1';
             WAIT FOR 40 ns;
             E <= '0';
             WAIT FOR 40 ns;
             E <= '1';
             WAIT FOR 40 ns;
             E <= '0';
             WAIT FOR 40 ns;
             E <= '1';
             WAIT FOR 40 ns;
             E <= '0';
             WAIT FOR 40 ns;
             C <= '1';
             WAIT FOR 40 ns;
             C <= '0';
             WAIT FOR 40 ns;
             C <= '1';
             WAIT FOR 40 ns;
             C <= '0';
             WAIT FOR 345 ns;
        END PROCESS;
    END testbench_arch; ..
```

The „Mean“ Vending Machine Signal Waveform



Want To Learn More ?





Consider Undergraduate Research With Us/Other Labs

https://smartsystems.ece.ufl.edu/