ETH

# Hardware Description Language VHDL

Revision 1.0 / Oct. 1997

Michael H. Eisenring
Computer Engineering and Networks Lab (TIK)
Swiss Federal Institut of Technology (ETH)
Gloriastr. 35, 8092 Zurich, Switzerland
http://www.tik.ee.ethz.ch/~eisenri/
eisenring@tik.ee.ethz.ch



ETH

## Why a course about VHDL?

- Short and fast introduction to VHDL
  → efficient and productive
- usage of common terms and design rules
  → quality of code
- fun to give a course

Goal: A running JAVA processor at the end of the semester

ETH

# Course outline

Chapter 1: Overview
Chapter 2: Design structure
Chapter 3: Basics
Chapter 4: Concurrent statements
Chapter 5: Sequential statements
Chapter 6: Testbenches
Chapter 7: VHDL for synthesis



ETH

# Chapter 1: Overview

VHDL is:

1. a hardware description language (compiler)
2. a programming language (types, funct., ctrl....)
3. useful for documentation (structure ... )
4. useful for simulation (time concept)
5. useful for synthesis (restricted subset only)

Defined as IEEE standard!

ETH

# Advantages for digital design

1. Industry support (EDA)
2. Portability
3. Modeling capability (several levels)
4. Technology and foundry independence
5. Reusability
6. Early verification/simulation of functionality



ETH

# levels of description

- system level
- board level
- behavioral level
- register transfer level
- gate level
- transistor level

ETH

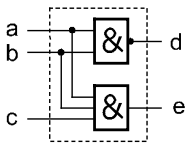
# Evolution of the language

- 1980: US government developed VHSIC project
  → enhance the electronic design process
  VHSIC = Very High Speed Integrated Circuit
  → initially used for modeling, simulation not synthesis
- 1987: IEEE 1076.1 standard for VHDL (VHDL'87)
  → foundation for most simulators, synthesis tools
  VHDL = VHSIC Hardware Description Language
- 1993: update of VHDL (VHDL'93)



ETH

# simple example



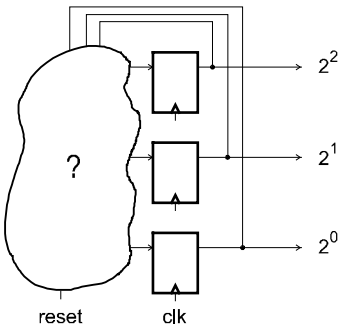

| a b c | d e |
|---|---|
| 0 0 0 | 1 0 |
| 0 0 1 | 1 0 |
| 0 1 0 | 1 0 |
| 0 1 1 | 1 0 |
| 1 0 0 | 1 0 |
| 1 0 1 | 1 0 |
| 1 1 0 | 0 0 |
| 1 1 1 | 0 1 |

```
d = not (a and b)
e = a and b and c
```

```
entity BSP is
  port (a,b,c : in  bit;
        d,e   : out bit);
end Bsp;

architecture GATE of BSP is
begin
  d <= not (a and b);

  AND3: process(a,b,c)
  begin
    e <= a and b and c;
  end process AND3;
end GATE;
```

ETH

# more complex example: 3 Bit counter




counts 0 ... 7

**earlier days:**

→ error prone and time consuming task

→ done by hand with discrete components

**today:**

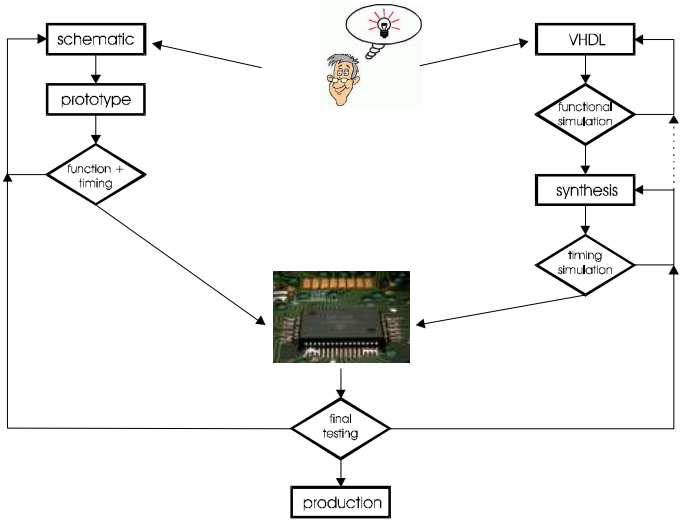
→ done by using a HDL programming language, e.g. VHDL

→ simulators, larger design space exploration



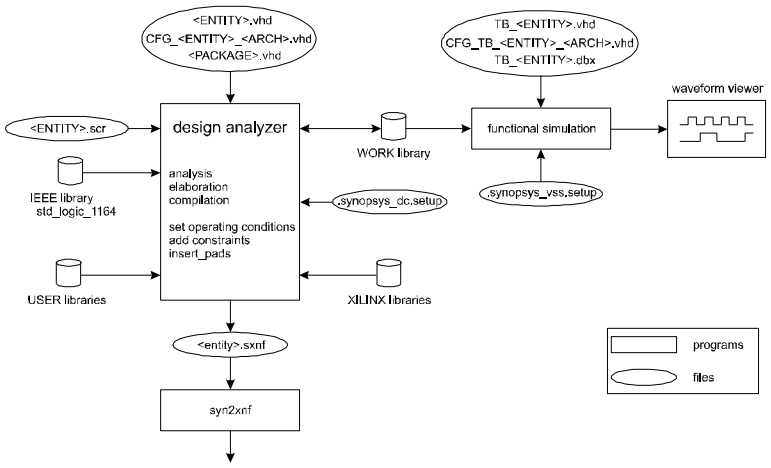
ETH

# design methodology



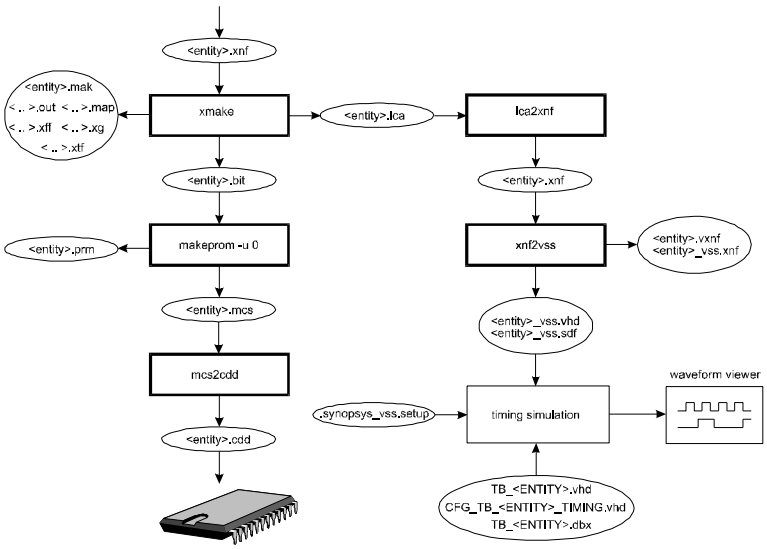
ETH

# SYNOPSYS/XILINX design flow






ETH



ETH

# C versus VHDL

- software language
- sequential
- structure, behavioral
- no time concept
- commercial libraries

- hardware language
- sequential and parallel
- several layers
  transistor, gate, RTL, block, ASIC, board, system, behavioral
- time concept
- signals
- commercial libraries



ETH

# JAVA versus VHDL

- software language
- sequential and parallel
- object oriented
- networking
- mechanism for synchronisation
- no time concept
- commercial libraries

- hardware language
- sequential and parallel
- several layers
  transistor, gate, RTL, block, ASIC, board, system, behavioral
- time concept
- signals
- commercial libraries

ETH

# recommended books

- Ben Cohen, *VHDL Coding Styles and Methodologies*, Kluwer Academic Publishers, 1995
  → practise oriented VHDL, a lot of examples
- K.C. Chang, *Digital Design and Modeling with VHDL and Synthesis*, IEEE Computer Society Press, 1997
  → outlined to synthesis
- Z. Navabi, *Analysis and Modeling of Digital Systems*, McGraw-Hill, 1993
  → good overview, no synthesis



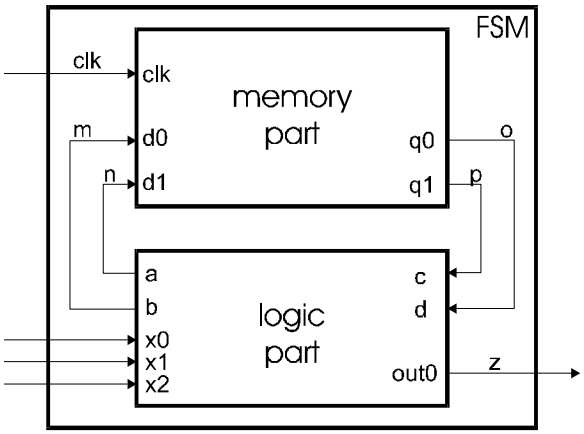
ETH

# Chapter 2: Design structure

- digital circuit
- main parts of a VHDL design
  - entity
  - architecture
  - configuration
  - package

ETH

# digital circuit






ETH

VHDL Design

VHDL Files

**Entity specifications**
external view

**Architectures**
implementations of entities

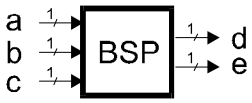
**Configurations**
binding of architectures to entities

**Packages**
types, constants, components, subprogramms

ETH

# entity

```
entity BSP is
  port (a,b,c : in  bit;
        d,e   : out bit);
end Bsp;
```




- An entity defines the external view (interface) of a component.
- Defines one level of design hierarchy
- The behavior is determined by the associated architecture.
- An entity may have several architectures.
- Architectures are assigned by the configuration.

Synthesis: Do not use generics, use packages with constants instead!

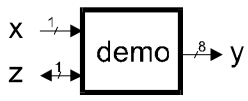


ETH

# entity

```
entity name is
  port(port_list);
end name;
```

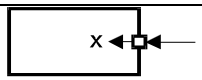
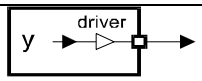
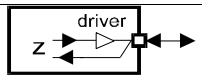
A port is a SIGNAL with a specified direction.




```
entity demo is
  port(x : in std_logic;
       y : out std_logic_vector(7 downto 0);
       z : inout std_logic);
end demo;
```

ETH

# interpretation of port direction

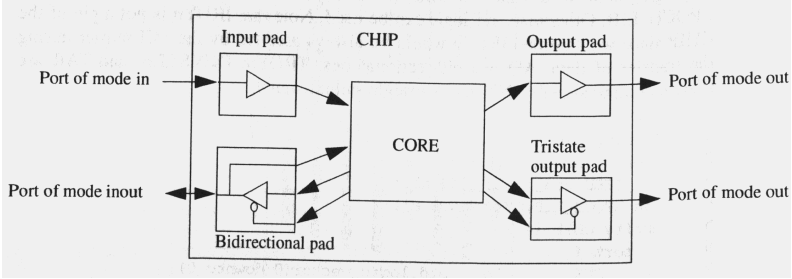
| | |
|---|---|
| **in** (input)<br>A variable or signal can read from an in port |  <br>var := x; sig <= x; |
| **out** (output)<br>A variable or signal can write to an out port |  <br>y <= var; y <= sig; |
| **inout** (bi-directional)<br>Read and write possible |  <br>z <= var; z <= sig;<br>var := z; sig <= z; |

Remark: The kind of driver depends on assigned signals and the target technology



ETH

# Physical chip, core and pads




K. C. Chang, Digital Design and Modeling width VHDL and Synthesis, IEEE Computer Society Press, 1997

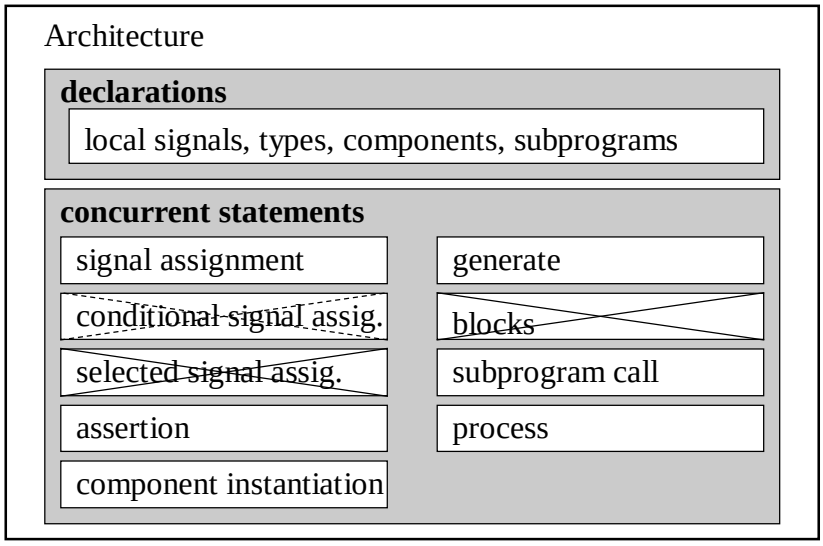
ETH

# architecture

- defines one possible implementation of an entity
- different styles: RTL, structural, behavioral, dataflow
- entity + architecture represents a component
- contain concurrent statements



ETH



ETH

# architecture

```
architecture name of entity_name is
  {types, constants, signals, components, no
   variables}
begin
  {concurrent statement}
end name;
```

```
architecture my_arch of entity_name is
  signal a,b : bit;
begin
  a <= '1';                -- concurrent signal assi.
  b <= '1' after 50 ns,    -- conditional signal assi.
       '0' after 60 ns,
       '1' after 10 ns;
end my_arch;
```

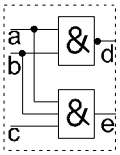


ETH

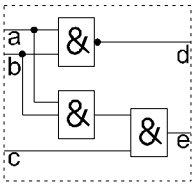
# example architecture

```
architecture GATE of BSP is
begin
  d <= not (a and b);

  AND3: process(a,b,c)
  begin
    e <= a and b and c;
  end process AND3;
end GATE;
```

Case 1




Case 2



ETH

# configuration

- bind a particular architecture to an entity
- bind entity/architecture to a component

```
configuration name of entity_name is
  for architecture_name
  end for;
end name;
```

```
-- This is the configuration for our simple example
configuration CFG_BSP of BSP is
  for Gate
  end for;
end CFG_BSP;
```



ETH

# package

- collection of constants, declarations, subprograms
- important for a structured design flow
- system / user packages, e.g.

```
library IEEE;
library MyLib;

use IEEE.std_logic_1164.all;
use MyLib.all;
```

→ readable and maintainable code

ETH

# example package

```
package BSP_OWN is
  type color is (blue, green, yellow, red);
  function add(i : integer) return integer;
end BSP_OWN;

package body BSP_OWN is
  function add(i : integer) return integer is
    variable j : integer;
  begin
    j := i;
    j := j + 1;
    return j;
  end add;
end BSP_OWN;
```



ETH

# Chapter 3: Basics

- operators
- data types
- object classes
- signals and variables
- resolved signals

ETH

# operators

| Precedence | Operator Class | Operators |
|---|---|---|
| LOW | Logical operator | **and** \| **or** \| **nand** \| **nor** \| **xor** \| **xnor**[9] |
| | relational operator | = \| /= \| < \| <= \| > \| >= |
| | shift operator[6] | **sll** \| **srl** \| **sla** \| **sra** \| **rol** \| **ror** |
| | adding operator | + \| - \| **&** |
| | sign | + \| - |
| | Multiplying operator | * \| / \| **mod** \| **rem** |
| HI | Miscellaneous operator | ** \| **abs** \| **not** |

Ben Cohen, VHDL Coding Styles and Methodologies, Kluwer Academic Publishers, 1995



ETH

# data types

***VHDL is a strongly typed language***

- Predefined types are declared in the *standard package*.

```
boolean, bit, bit_vector, character, integer ....
```

- Enumeration Types

```
type color is (blue, green, yellow, red);
type bit is ('0', '1');
```

- Integer Types

```
type integer is range -2147483648 to 2147483647;
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
```

ETH

# data types

- Constrained Array

```
type byte is array (7 downto 0) of bit;
type vector is array (3 downto 0) of byte;
```

- Unconstrained Array

```
type bit_vector is array (integer range <>) of bit;
```

- Real (implementation dependent range)

```
real
```

- Physical

```
time with units fs, ps, ns, us, ms, min
```



ETH

# data types

- Record

```
type user_rec is
  record
    by : byte;
    x  : integer range 0 to 7;
  end record;
```

ETH

# object classes

constant : value may not change

```
constant DataWidth : integer := 16;
```

signal : object with past history (equivalent to wire)

```
signal b : bit;
```

variable : single current value (no history)

```
variable j : integer;
```

file : files in the host environment

```
file DataIn : TextIO.text is in "dat.txt";
```



ETH

| signals | variables |
|---|---|
| properties:<br>1. type and type attributes<br>2. value (current, past)<br>3. time | properties:<br>1. type and type attributes<br>2. value |
| used to communicate between processes and entities | exist only in processes and subprograms |
| not local to a process | only local |

| signals | variables |
|---|---|
| do not use for memories (bad for simulation! see next slide) | description of memories |
| not local to a process | only local |
| assignments at the end of a process | assignments are immediately done |
| `a <= '1';` | `a := '1';` |





# storage size for declared elements

| ELEMENT TYPE | VARIABLE | SIGNAL |
|---|---|---|
| Enumeration type < 256 states | **1 byte -- 8 bits** | **1 + ~ 100 byte** |
| Enumeration type > 256 states | **4 bytes -- 32 bits** | **4 + ~ 100 byte** |
| Array(1 to n) of EnumType (<256 states) | **n * 1 byte** | **n * (1 + ~ 100 byte)** |
| integer | **4 bytes -- 32 bits** | **4 + ~ 100 bytes** |
| real | **8 bytes -- 64 bits** | **8 + ~ 100 bytes** |
| time | **8 bytes -- 64 bits** | **8 + ~ 100 bytes** |
| array 1 to 64,000 of Std_LogicVector(31 downto 0) -- 64K X 32 bits Memory | 64K * 1 * 32 = 2,048 Kbytes **~ 2 MBytes** | 64K * 101 * 32 = 206,848 Kbytes **~ 207 Mbytes** |
| array 1 to 64,000 of integer -- 64K integer Memory | 64K * 4 = 256 K **~ 1/4 Mbyte** | 64K * 104 = 6,656 Kbytes **~ 7 Mbytes** |

Ben Cohen, VHDL Coding Styles and Methodologies, Kluwer Academic Publishers, 1995

ETH

examples:

```
variable col      : color;
signal     i      : integer;
variable   j      : integer range 0 to 18;
signal   abc      : byte;
variable my_vect  : bit_vector(3 downto 0);
variable r        : real;
variable rec      : user_rec;
signal   sig_time : bit;

col      := red;
i        <= 5000;
j        := 12;
abc      <= "00101110";
my_vect  <= "0110";
r        := 3.0;
rec.x    := 3;
sig_time <= '1' after 2 ns;  -- only for simulation
```



ETH

# value update

## signal

```
process(a, ...)

begin
     :              a = 0
  a <= '1';         a = 0
  a <= '1' and a;   a = 0
     :
  a <= not a;       a = 0
     :
end process;        a = 1
```

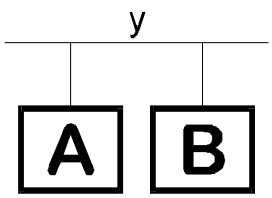
## variable

```
process(...)
  variable a : bit;
begin
     :              a = 0
  a := '1';         a = 1
  a := '1' and a;   a = 1
     :
  a := not a;       a = 0
     :
end process;        a = 0
```

ETH

# common bus

problem: two processes write to one signal






ETH

# resolved signal

IEEE defines std_logic_1164 package

```
type std_ulogic is ('U','X','0','1','Z','W','L','H','-');
```

| | |
|---|---|
| U: unititialized | X : forcing unknown |
| 0 : forcing 0 | 1 : forcing 1 |
| Z : high impedance | W: weak unknown |
| L : weak low | H : weak high |
| - : don't care | |

**resolution function** : determine the result value for signals that are driven by more than one driver

ETH

# resolution table

```
TYPE stdlogic_table is array(Std_uLogic, Std_uLogic) of Std_uLogic;
    ----------------------------------------------------------------
    -- resolution function
    ----------------------------------------------------------------
    constant resolution_table : Stdlogic_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' )  -- | - |
        );
```

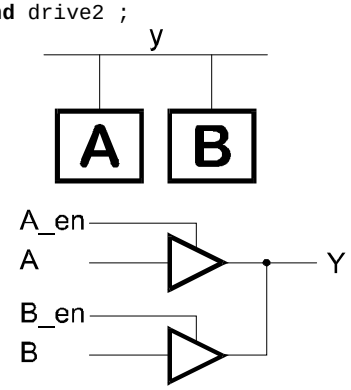


ETH

# resolved type

**recommended for synthesis!**
**std_logic**
**std_logic_vector**

```
library IEEE;
use IEEE.std_logic_1164.all;

entity drive2 is
  port(...);
end drive2 ;
```




```
architecture RTL of drive2 is
  signal Y,A_en,A, B_en,B:std_logic;
begin
  process A(...)
  begin
        :
    if (A_en = '1') then
      Y <= A;
    else
      Y <= 'Z';
    end if;
  end process A;

  process B(...)
  begin
    Y <= 'Z';
        :
    if (B_en = '1') then
      Y <= B;
    end if;
  end process B;
end RTL;
```





# Chapter 4: Concurrent statements

- assertion
- component instantiation
- generate
- process

ETH

# Concurrent statements

- order of their execution is not affected by their order of appearance
- activated by events on signals



ETH

Architecture

**declarations**

local signals, types, components, subprograms

**concurrent statements**

| | |
|---|---|
| signal assignment | generate |
| conditional signal assig. | ~~blocks~~ |
| ~~selected signal assig.~~ | subprogram call |
| assertion | process |
| component instantiation | |

ETH

# assertion

```
assert condition report string severity level;
```

- checks if condition is TRUE and reports an error if condition is FALSE
- useful in testbenches

***level***:

| | |
|---|---|
| FAILURE | : error in the model itself |
| ERROR | : timing violation, invalid data affecting the state of the model |
| WARNING | : timing violation, invalid data not affecting the state of the model |
| NOTE | : message for debugging use |



ETH

# assertion

```
architecture my_arch of entity_name is
  signal a : bit;
begin
  process(a, ...)
  begin
     :
    assert not a report "a is one" severity ERROR;
     :
  end process;
end my_arch;
```

ETH

# component instantiation

- introduces hierarchy
- allows structural composition
- each component has to be declared in an architecture or package before instantiation

declaration:

```
component component_name
  port(port_list);
end component;
```

instantiation:

```
instance_name: component_name
  port map(port_name => signal {, port_name => signal});
```



ETH

```
entity BSP is
  port (a,b,c : in bit;
        d,e   : out bit);
end Bsp;

architecture GATE of BSP is
begin
  d <= not (a and b);

  AND3: process(a,b,c)
  begin
    e <= a and b and c;
  end process AND3;
end GATE;
```

```
architecture XYZ of BSP2 is

  component BSP
  port (a,b,c : in bit;
        d,e   : out bit);
  end component;

  signal i,k,l,m,n : bit
begin

  A: Bsp
  port map
  (a => i, b => k, c => l,
   d => m, e => n);

end XYZ;
```

ETH

# generate

Creates zero or more copies of an enclosed set of concurrent statements

```
[label:]for identifier in range generate
  {concurrent statement}
end generate [label];
```

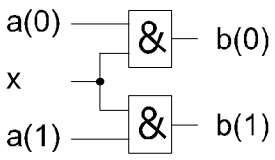
synthesis: useful for input/output pads insertion



ETH

# generate example

```
architecture bsp_gen of bsp is
  signal a,b :std_logic_vector(1 downto 0);
  signal x   :std_logic;
begin
           :
  A: for I in 1 downto 0 generate
    b(I) <= a(I) and x;
  end generate A;
           :
end bsp_gen;
```



ETH

# process

- contain **sequential** statements
- are activated by signals in the sensitivity list or by wait statements
- restart when they have finished (i.e. automatic loop back to the first statement)
- only processes define variables for temporary values
- behave like **concurrent** statements to the rest of the design

```
signal x,y,z : bit;

process(x,y)
  variable p : bit;
begin
  z <= '0';
  if (x = '1') then
    p := x and y;
  end if;
  z <= p;
end process;
```



ETH

# process

```
[label:]process[(sensitivity_list)]is
  {declarations, e.g. variables, types but no signals}
begin
  {sequential statement}
end process [label];
```

```
signal clk,a,b,c:bit;

X: process
begin
  c <= a and b;
  d <= c or (a and b);
end process X;
```

Process

**local declarations**

variables, types, constants, components, subprograms

**sequential statements**

| | |
|---|---|
| signal assignments | loop statements |
| variable assignments | next statements |
| subprogram calls | exit statements |
| if statements | wait statements |
| case statements | null statements |
| return statement | assertion statement |





Roughly spoken:
A VHDL description consists of a set of processes communicating by using global signals

ETH

# Chapter 5: Sequential statements

- if
- case
- loop
- for loop
- wait
- if and clock
- loop
- while loop
- subprogram
- automatic latch inferencing



ETH

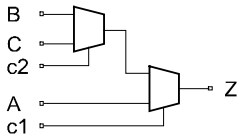
# Sequential statements

- are interpreted one after another
- exist only in processes and subprograms

# if

```
if condition then
  {sequential statement}
[elsif condition then
  {sequential statement}]
[else
  {sequential statement}]
end if;
```

```
signal A,B,C,Z,c1,c2:bit;

if (c1='1') then
  Z <= A;
elsif (c2 ='1') then
  Z <= B;
else
  Z <= C;
end if;
```








# case

```
case expression is
  when choices =>
    {sequential statement}
 [when choices =>
    {sequential statement}]
 [when others =>
    {sequential statement}]
end case;
```

```
signal val is integer
   range 0 to 8;
signal A,B,C,Z : bit;

case val is
  when 0 =>
    Z <= A;
  when 1 =>
    Z <= B;
  when others =>
    Z <= C;
end case;
```

ETH

# loop

repeatedly executes a sequence of statements

**warning**:

1. Computable loops (`for ... loop`) must not contain `wait` statements. Otherwise, a race condition may result.

2. Non computable loops (`loop, while ... loop`) must have at least one `wait` statement. Otherwise, a combinational feedback is created.



ETH

# for loop

```
[label:]for identifier in discrete_range loop
  {sequential statement}
end loop [label];
```

```
signal val is integer range 0 to 7;
signal B : std_logic_vector(0 downto 7);
signal C : std_logic_vector(7 downto 0);
```

```
TO_loop: for i in 0 to 7 loop
  B(i) <= C(i);
end loop TO_loop2;
```

```
TO_loop2: for i in val loop
  B(i) <= C(i);
end loop TO_loop2;
```

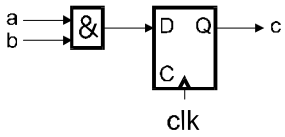
ETH

# wait

```
wait until signal'event
  and signal = value;
```

- A wait statement suspends a process until a positiv going or negativ going edge is detected on a signal.
- implies synchronous logic

```
signal clk,a,b,c:bit;

process
begin
  wait until clk'event and
  clk = '1';

  c <= a and b;
end process;
```

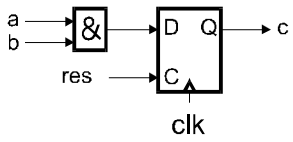





ETH

# if and clock

equal to wait and clock




```
signal clk,a,b,c:bit;

process(clk, a, b)
begin
  if res = '0' then
    c <= '0';
  elsif (clk'event and clk = '1') then
    c <= a and b;
  end if;
end process;
```

ETH

# loop

```
[label:]loop
  {sequential statement}
end loop [label];
```

```
variable val is INTEGER range 0 to 7;
signal i : std_logic;

loop_loop: loop
  if val < val'high then
    val := val + 1;
  else
    exit;
  end if;
  wait until clk'event and clk = '1';
end loop loop_loop;
```



ETH

# while loop

```
[label:]while condition loop
  {sequential statement}
end loop [label];
```

```
variable val is INTEGER range 0 to 7;
signal i : std_logic;

while_loop: while i = '1' loop
  if val < val'high then
    val := val + 1;
  else
    val := 0;
  end if;
  wait until clk'event and clk = '1';
end loop while_loop;
```

ETH

# subprogram

- procedures and functions
- define sequential algorithm
- sequential or concurrent call
- **no** side effects
- no wait statement allowed
  → only combinational logic
- needs declaration and body (like entity) if defined in a package, only body necessary if defined in architecture
- advantage: declaration can be declared in public packages → useful for teamwork



ETH

Declaration:

```
procedure name [(parameter declaration)];
function name [(parameter declaration)] return type;
```

Body:

```
procedure name [(parameter declaration)] is
  {local declaration}
begin
  {sequential statement}
end [name];

function name [(parameter declaration)] return type is
  {local declaration}
begin
  {sequential statement}
end [name];
```

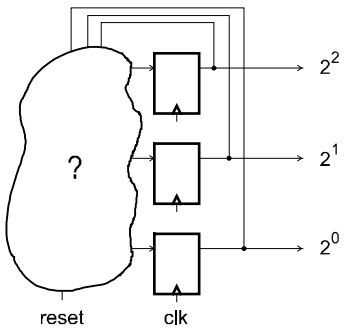
```
package example is
  procedure ADD(A: in integer, B: inout integer);
  function INVERT(i:bit) return bit;
end example;

package body example is
  procedure ADD(A: in integer, B: inout integer)
  begin
    B := A + B;
  end;

  function INVERT(i:bit) return bit is
  begin
    return (not i);
  end;
end example;
```





# our more complex example: 3 Bit counter

counts 0 ... 7



ETH

# 3 Bit counter

```
entity counter3 is
  port(clk  :in bit;
       reset:in bit;
       count:out integer
             range 0 to 7);
end counter3;
```

```
architecture my_arch of counter3 is
begin
  count:process
    variable tmp : integer range 0 to 7;
  begin
    wait until clk'event and clk='1';
    if (reset='1') or (tmp=7)
    then
      tmp := 0;
    else
      tmp := tmp + 1;
    end if;
    count <= tmp;
  end process count;
end my_arch;
```

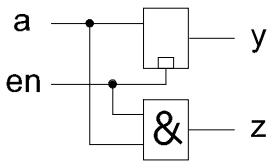


ETH

# problem: automatic latch inferencing

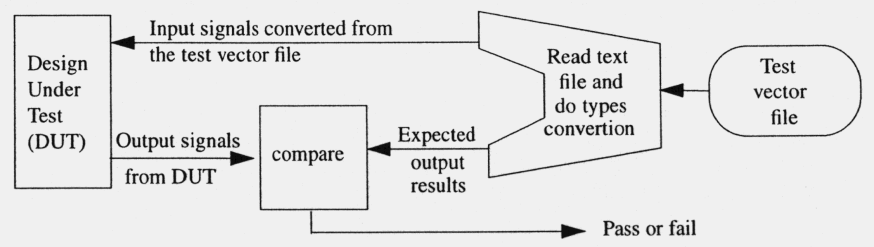
a signal or variable that is not driven under all conditions becomes a latched value

```
signal a,y,z,en:bit;

process
begin
  if (en='1') then
    y <= a;
  end if;

  if (en='1') then
    z <= a;
  else
    z <= '0';
  end if;
end process count;
```



ETH

# Chapter 6: Testbenches




K. C. Chang, Digital Design and Modeling width VHDL and Synthesis, IEEE Computer Society Press, 1997



ETH

# Chapter 7: VHDL for synthesis

Synthesis is the process of translating a design from a hardware description into a circuit.

Synthesis is not defined by IEEE, so tools differ in their capabilities, guidelines and coding styles!