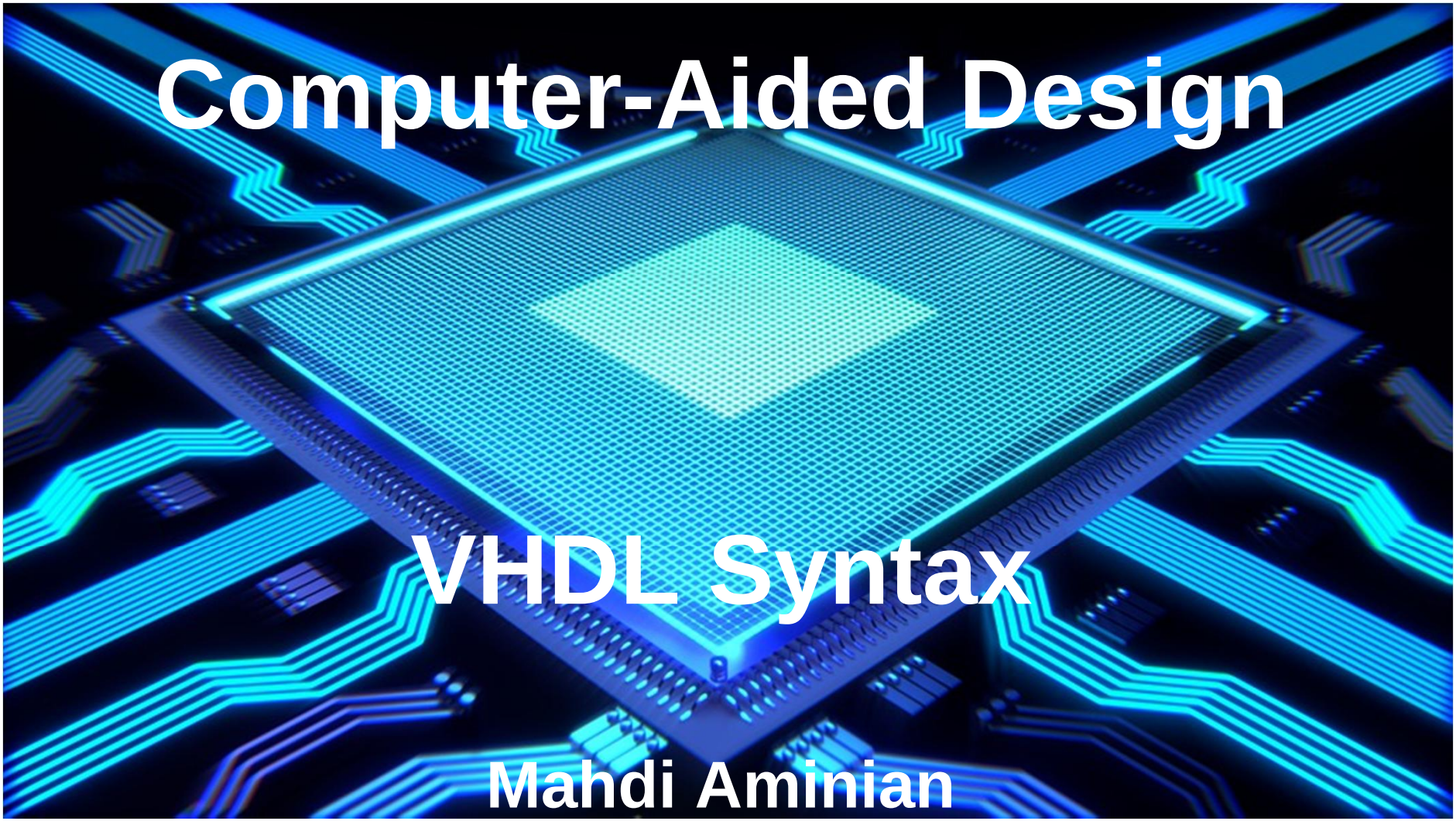


Computer-Aided Design

VHDL Syntax

Mahdi Aminian

The University Of Guilan

Rasht - Iran

Some slides courtesy of:

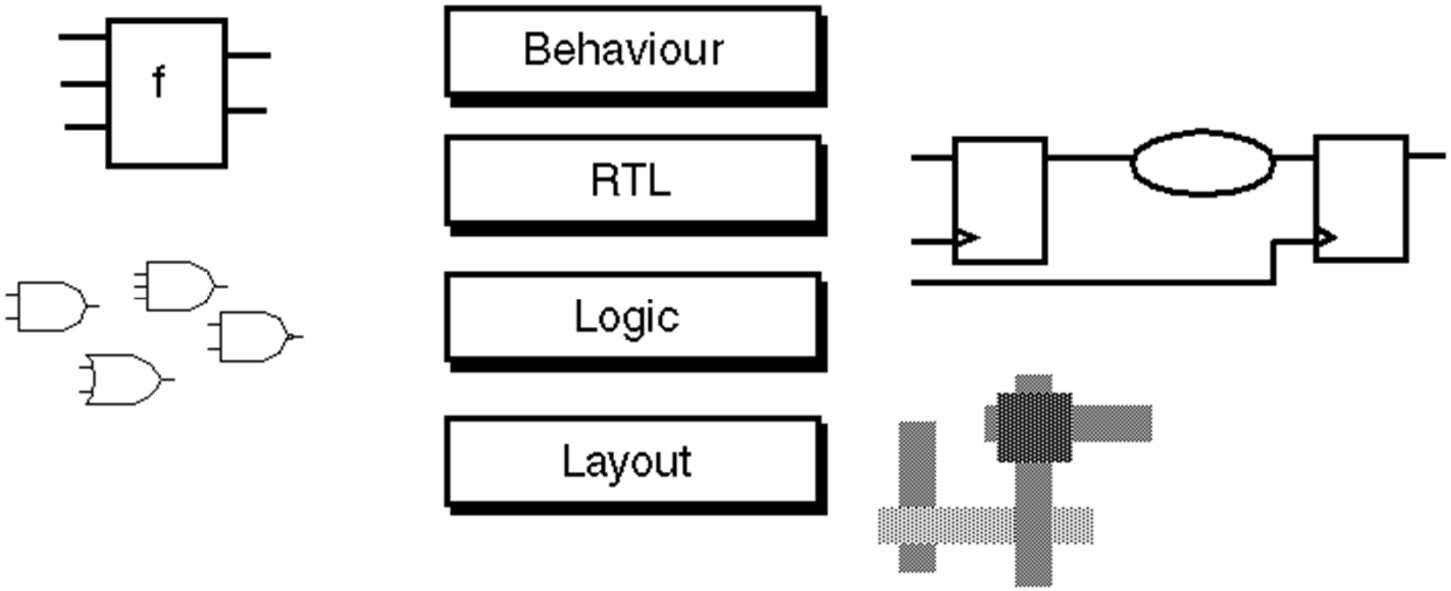
- “Hardware Systems Modeling”, A. Vachoux, EPFL
- “VHDL”, Dan Solarek, University of Toledo
- CAD slides from Dr. Saheb Zamani

VHDL - Overview

- Very High Speed Integrated Circuit Hardware Description Language
  - Modeling of digital systems
  - Concurrent and sequential statements
- International Standards
  - IEEE Std 1076-1987
  - IEEE Std 1076-1993, 2000, 2002, …

- **علاوه بر استانداردهای خالص, تلاشهایی برای استاندارد کردن عوامل مربوط به VHDL انجام گرفته است:**
  - **پکیج های**
    - **Std_logic_1164**
    - **Numeric_bit**
    - **Numeric_std**
  - **زیرمجموعهٔ قابل سنتز: استاندارد 1076.6**
  - **VHDL-AMS 1076.1**

Abstraction Levels in IC Design



- **VHDLبرای سطح layout مناسب نیست.**

VHDL Structural Elements

- **Entity**: Interface
- **Architecture**: Implementation, behavior
- **Configuration**: Structure, hierarchy
- **Process**: Sequential Execution
- **Package**: Components (Modular design), Utilities (data types, constants, subprograms)
- **Library**: Group of compiled units, object code

Design Library

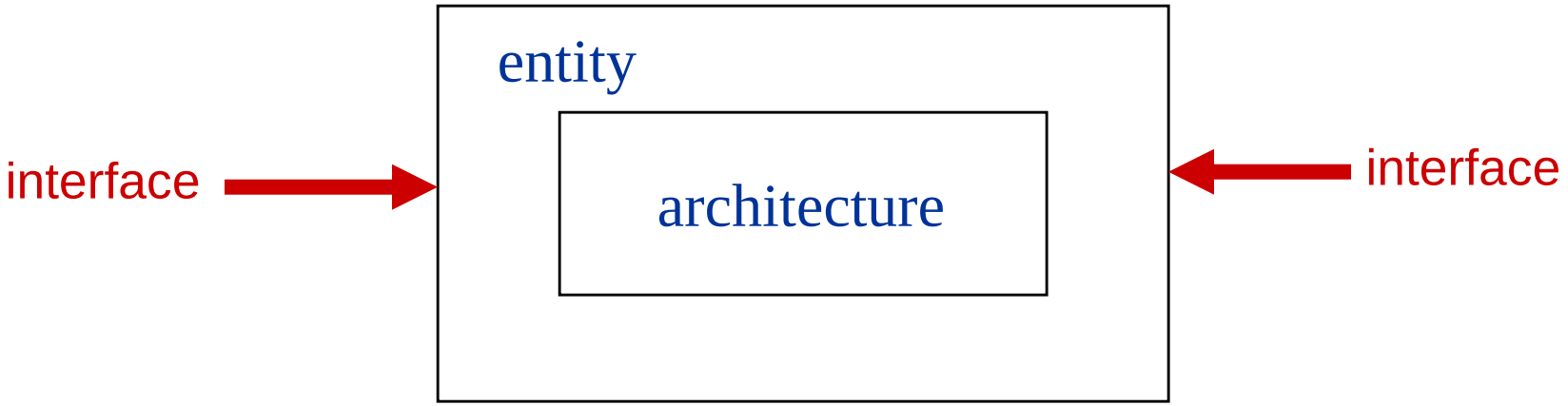
- **Storage location for analyzed design units**
- **VHDL models only deal with logical library names**
  - Mapping to physical locations is up to VHDL tools
- **Two classes of design libraries**
  - **Working** library (read-write, only one active)
  - **Resource** library (read-only, possibly more than one active)
- **Predefined logical design libraries**
  - **WORK** Current active working library
  - **STD** Resource library containing only three packages (STANDARD, TEXTIO)
  - **IEEE** Resource library containing only standard VHDL packages

Design File

- **Text file containing one or more design units**
  - Recommended file extension: **.vhd**
- **Order of design units is important**
  - Primary units must appear before related secondary units
- **Successful analysis of a design file generates separate binary design units in the working library**

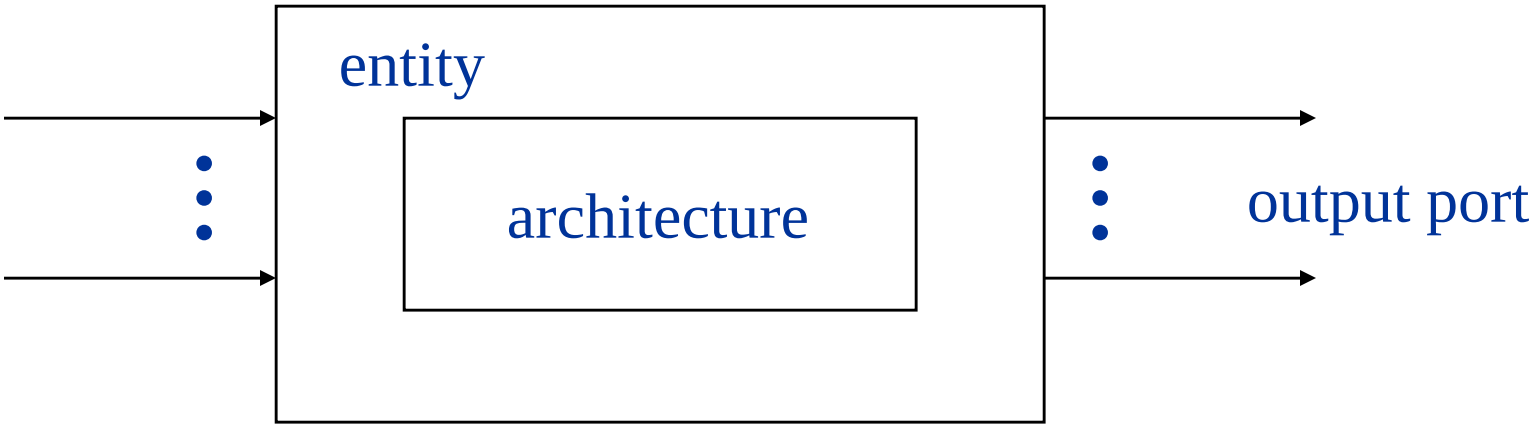
VHDL Key Idea

- A key idea in VHDL is to define the interface of a hardware module while hiding its internal details.
  - A VHDL **entity** is simply a declaration of a module's inputs and outputs, i.e., its external interface signals or ports.
  - A VHDL **architecture** is a detailed description of the module's internal structure or behavior.



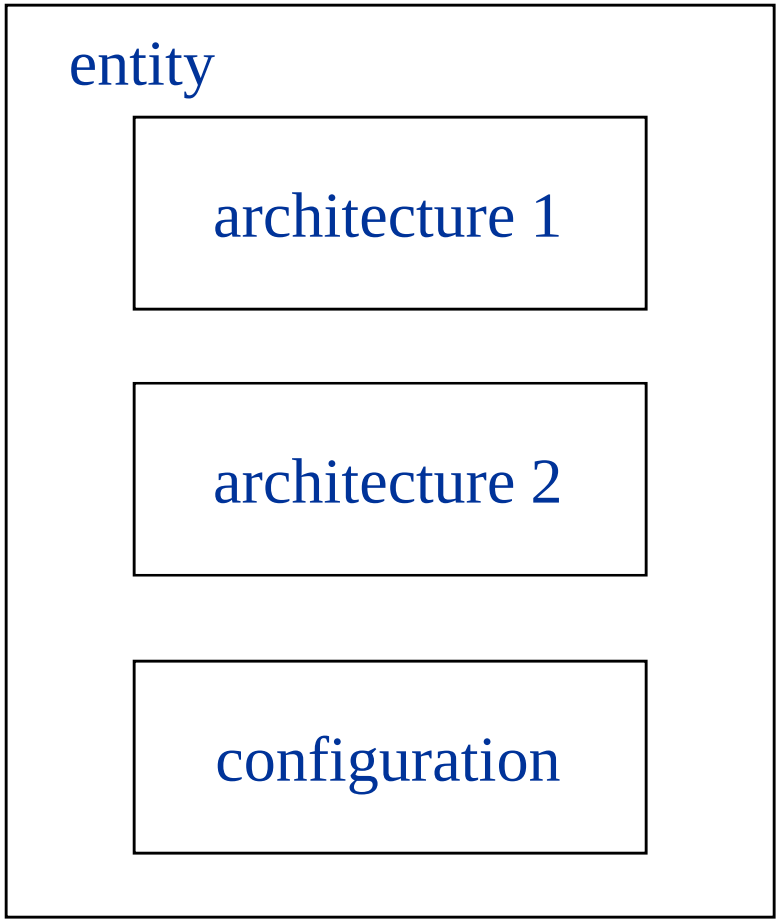
VHDL Interface - Ports

- You can think of the entity as a “wrapper” for the architecture
  - hiding the details of what’s inside
  - providing “ports” to other modules



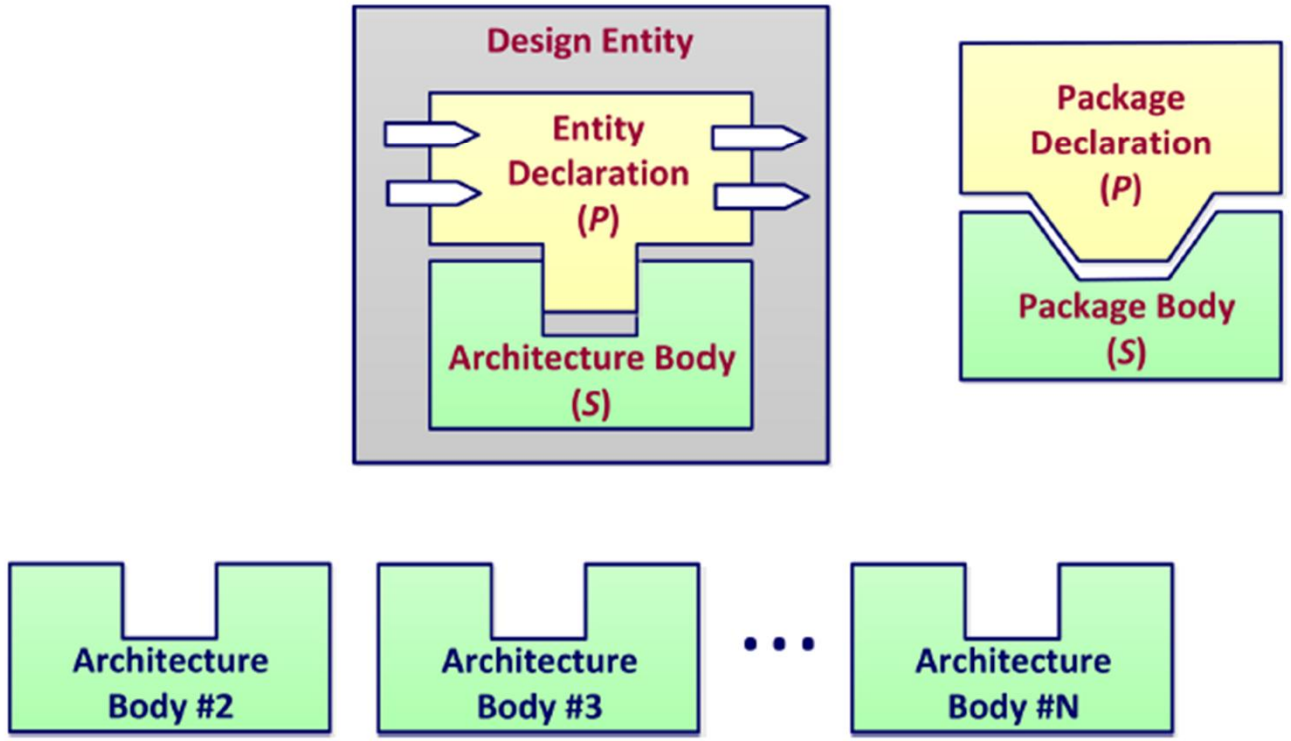
VHDL Conceptual Model

- VHDL actually allows you to define multiple architectures for a single entity
- it also provides a configuration management facility that allows you to specify which architecture to use during a particular synthesis run



VHDL Design Units

- **Smallest model elements that can be separately analyzed**
  - P: Primary units
  - S: Secondary units

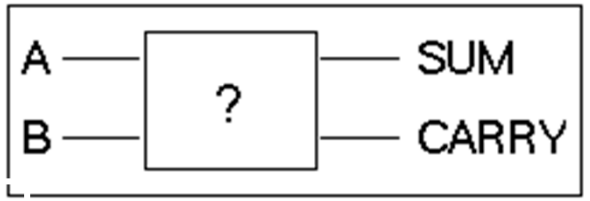


- **Design *entity*: Primary hardware abstraction**

Entity

```
entity HALFADDER is
   port(
      A, B:                in   bit;
      SUM, CARRY: out bit);
end entity HALFADDER;
```

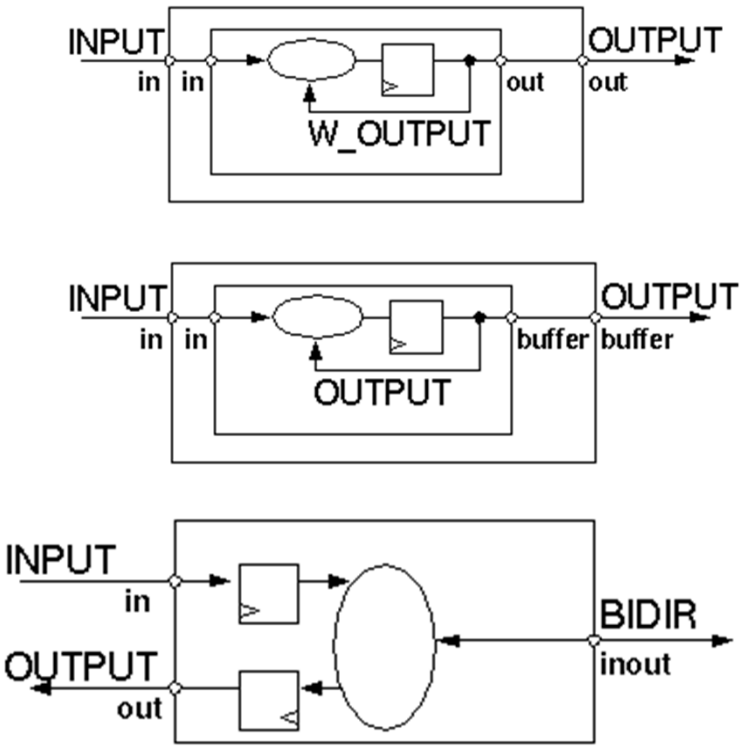
- **Interface description**
- **No behavior/implementation definition**




```
entity ADDER is                    in    bit_vector (3 downto 0);
   port(                           out   bit_vector (3 downto 0);
     A, B:            in    integer range 15 to 0;
     SUM:             out   integer range 15 to 0;
     CARRY:           out   bit );
end ADDER;
```

- **Linking via port signals**
  - data types
  - signal width
  - signal direction

Entity Port Modes



- **in:**
  - signal values are read-only
- **out:**
  - signal values are write-only
  - multiple drivers
- **buffer:**
  - similar to out
  - signal values may be read, as well
  - only 1 driver
- **inout:**
  - bidirectional port

General

```
---------------------------------
-- Example VHDL Code --
---------------------------------
signal mySignal: bit;       -- an example signal
MYsignal <= '0',                    -- start with '0'
                '1' AFTER 10 ns,    -- and toggle after
                '0' after 20 ns,    -- every 10 ns
                '1' afTer 30 ns;
```

- **Statements are terminated by ';'**
  (may span multiple lines)
- **List delimiter: ','**
- **Signal assignment:** '<='
- **User defined names:**
  - letters, numbers, underscores
  - start with a letter
- **Comments: '--' until end of line**

VHDL Identifiers (Names)

- **Basic identifier**
  - starts with a letter
  - made up of letters, numbers, and underscore "_" character
  - cannot end with an underscore
  - case-insensitive: `MY_Signal_Name = my_signal_name`

- **Extended Identifier** ← not recommended
  - any text within 2 backslashes
  - e.g., `\2FOR$\` `\-23signal\` etc.
  - case is significant: `\COUNT\` not equal to `\count\`, and `\FRAMUS\` not equal to basic identifier `FRAMUS`
  - **Not often used – *not necessarily supported in synthesis !!***

Identifiers

```
mySignal_23      -- normal identifier
rdy, RDY, Rdy    -- identical identifiers
vector_&_vector  --  X : special character
last of Zout     --  X : white spaces
idle__state      --  X : consecutive underscores
24th_signal      --  X : begins with a numeral
open, register   --  X : VHDL keywords
```

- **(Normal) Identifier**
  - Letters, numerals, underscores

```
\mySignal_23\        -- extended identifier
\rdy\, \RDY\, \Rdy\  -- different identifiers
\vector_&_vector\    -- legal
\last of Zout\       -- legal
\idle__state\        -- legal
\24th_signal\        -- legal
\open\, \register\   -- legal
```

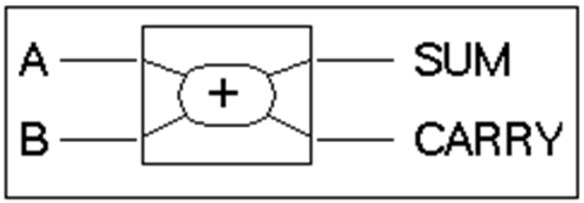
- **Extended Identifier:**
  - Enclosed in back slashes
  - Case sensitive
  - Graphical characters allowed
  - May contain spaces and consecutive underscores
  - VHDL keywords allowed

Architecture

```
entity HALFADDER is
   port(
      A, B:              in   bit;
      SUM, CARRY: out bit);
end HALFADDER;

architecture RTL of HALFADDER is
begin
   SUM     <= A xor B;
   CARRY <= A and B;
end architecture RTL;
```

- **Implementation of the design**
- **Always connected with a specific entity**
  - one entity can have several architectures
  - entity ports are available as signals within the architecture
- **Contains concurrent statements**



Architecture Structure

```
architecture EXAMPLE of STRUCTURE is
  subtype DIGIT is integer range 0 to 9;
  constant BASE: integer := 10;
  signal DIGIT_A, DIGIT_B: DIGIT;
  signal CARRY:            DIGIT;
begin
  DIGIT_A <= 3;
  SUM <= DIGIT_A + DIGIT_B;
  DIGIT_B <= 7;
  CARRY <= 0 when SUM < BASE else
           1;
end EXAMPLE ;
```

- **Declarative part:**
  - data types
  - constants
  - intermediate signals
  - component declarations
  - ...
- **Statement part (after 'begin'):**
  - signal assignments
  - processes
  - component instantiations
  - concurrent statements: order not important

    à Simulator recalculates **sum** any time an operand changes

- **Port is a kind of signal.**

Four Classes of Data Objects

- **Constant**
  - Holds a single value of a given type – cannot change.
- **Signal** ← analogous to a "wire"
  - Holds a LIST of values of a given type.
  - Present value + a set of possible future values
  - New values can be assigned **at some future time – not now!**
  - Signals have ATTRIBUTES: [signal'attribute]
- **Variable**
  - Holds a single value of a given type.
  - New value of same type can be "assigned" – (instantly)
- **File**
  - Contains a sequence of values of one or more types.
  - Usually read or written to using procedures
  - For simulation – not synthesis

Data Types

- Scalar Types
  - Enumerated : a list of values
  - Integer
  - Floating Point
  - Physical : with units, for physical quantities
- Composite Types
  - Array (all of same type)
  - Record (can be different types)
- Access Type
- File Type

Enumerated Types

- **CHARACTER –** one of the ASCII set
- **BOOLEAN** – can be FALSE or TRUE
- **BIT** – can be '0' or '1' (note single quotes)
- **STD_LOGIC** ⟵ IEEE std_logic_1164 package
  - Has NINE legal values:
  - 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'
- **Example Declarations**
  - **Type** CAR_STATE **is** (back, stop, slow, medium, fast);
  - **Subtype** GO_KART **is** CAR_STATE **range** stop to medium;
  - **Type** DIGIT **is** ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');

Value	Interpretation
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

Arrays

- An Array of type **BIT** is called a **BIT_VECTOR**

```
signal MYSIG : IN     BIT_VECTOR( 0 to 3);
```

- An Array of type **STD_LOGIC** is called a **STD_LOGIC_VECTOR**

```
signal yourSIG : OUT    BIT_VECTOR( 31 downto 0);
```

- A **STRING** is a character array

```
csonstant GREETING: STRING := “Hello!”;
```

Signal

- **VHDL object representing a time-domain waveform**
  - Stores time-value pair(s)
  - Stored values depend on the signal's type
  - Stored times depend on when the signal's value change
- **Signal declaration**
  - May appear in entity (signal port) or in architecture (local signal)
  - Optional expression defines the initial value (only for simulation)
  - Default initial value depends on the signal's type
  - Examples:

```
                                                    -- initial values:
signal clk : std_logic;                             -- 'U' (default)
signal clk : std_logic := '0';                      -- '0'
signal count : integer;                             -- -(2^31-1) (default)
signal count : integer := 0;                        -- 0
signal reset, strobe, enable : boolean;             -- FALSE (default)

signal data_bus : std_logic_vector(31 downto 0);    -- 32-bit "UUU...U"
signal addr_bus : std_logic_vector(7 downto 0) := (others => '0');
                                                    -- 8-bit "0000_0000"
```

Signal Assignment Statement

- **Zero delay assignment**

```
sum  <= opa xor opb xor cin;
```

- **Non-zero delay asignment**

```
sum  <= opa xor opb xor cin after 2 ns;
```

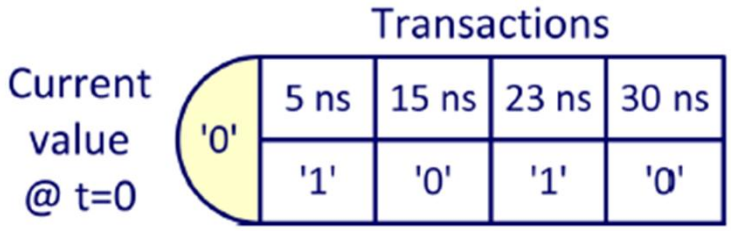
  - Signal sum will react to a change of value with a delay of 2 ns
  - Delay is relative to the simulation time at which the statement is executed
  - Inertial delay mode
  - Ignored in synthesis

- **Multiple waveform elements**

```
signal s : std_logic;
s <= '0', '1' after 5 ns, '0' after 15 ns, '1' after 23 ns, '0' after 30 ns;
```

  - Rejected in synthesis

- **A signal stores its information in a specific data structure called a driver**
  - Current value and future **transactions**
  - Ex.: driver of signal s at time 0



Logic Data Representation

9-state Logic Value System

- **Types std_logic and std_logic_vector**

```
type std_logic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
                   |    |    |    |    |    |    |    `-> don't care
                   |    |    |    |    |    |    `-> weak high/true
                   |    |    |    |    |    `-> weak low/false
                   |    |    |    |    `-> weak unknown
                   |    |    |    `-> high-impedance
                   |    |    `-> strong high/true
                   |    `-> strong low/false
                   `-> strong uknown
                  `-> uninitialized/unassigned
type std_logic_vector is array (natural range <>) of std_logic;
```

- **Supported operators:**

Relational:	= /= < <= > >=
Logical:	**and or nand nor not xor xnor**
Shift and rotate[1]:	**sll srl rol ror** *-- VHDL-2008 only*
Concatenation[1]:	&

[1] for type std_logic_vector

Time Representation: Type time

```
type time is range integer-range
   units
      fs;            -- femtosecond, base unit, minimum time resolution
      ps = 1000 fs;  -- picosecond
      ns = 1000 ps;  -- nanosecond
      …
      hr = 60 min;   -- hour
   end units;

subtype delay_length is time range 0 to integer'high;  -- positive delay
```

- **Supported operators:**

```
Relational:   =   /=   <   <=   >   >=
Arithmetic:   unary/binary + and -   *   /   abs
```

- **Time literals:**

```
10 ns    ms (= 1 ms)   46.9 us (= 46900 ns)
```

  - One space character required between value and unit

Process Statement

- **Fundamental concurrent statement**
- **General form**
  - Optional: label, sensitivity list, keyword **is**

optional sensitivity signal list

```
label : process ( signal-list ) is
   declarative-part
begin
   sequential-statement-part
end process label ;
```

```
entity AND_OR_XOR is
  port (A,B :    in    bit;
         Z_OR, Z_AND, Z_XOR : out bit);
end AND_OR_XOR;

architecture RTL of AND_OR_XOR is
begin

   A_O_X:  process  (A, B)
    begin
     Z_OR   <= A or    B;
     Z_AND <= A and B;
     Z_XOR <= A xor  B;
    end process  A_O_X ;

end RTL;
```

- **Contains sequentially executed statements**
- **Only within an architecture**
- **Several processes run concurrently**
- **Execution is suspended either via**
  - sensitivity list (contains trigger signals), or
  - wait-statements
- **Signal values don't change until suspension**

Process Activation Control

- **On signal events**

```
process (signal-name-list)
begin
   sequential-statements
end process;
```

↔

```
process
begin
   sequential-statements
   wait on signal-name-list;
end process;
```

  - Both forms are mutually exclusive and strictly equivalent

- **On timeout:**

```
wait for time-expression;
wait for 10 ns;
```

- **On condition:**

```
wait until condition;
```

```
signal clk, enable : std_logic;
wait until clk = '1';              -- wait on rising edge
wait on clk until clk = '1';       -- equivalent to the previous one
wait on enable until clk = '1';    -- no more sensitive to clk!
wait until enable = '1' for 1 ms; -- whichever comes first
```

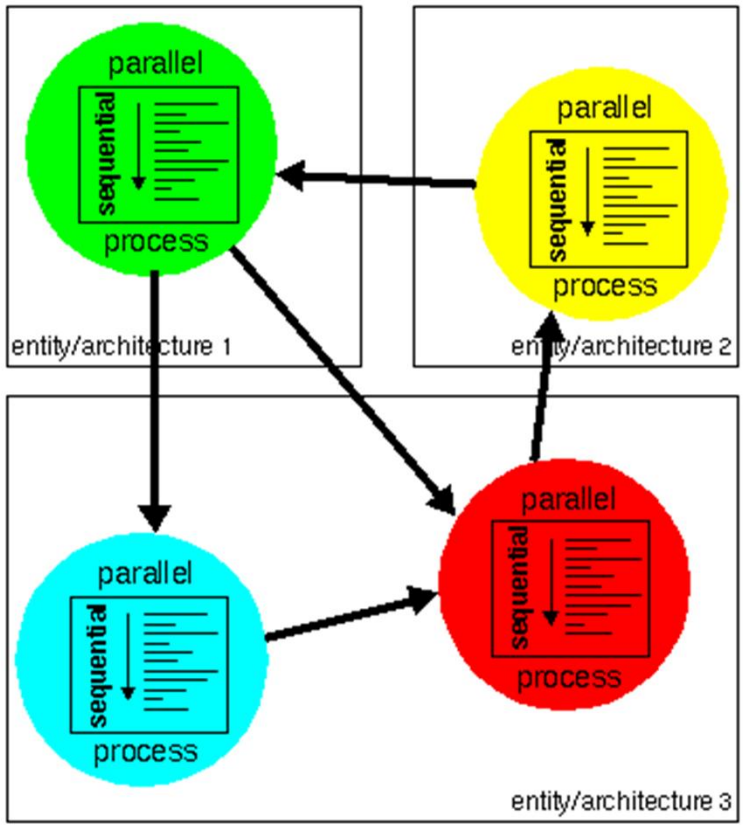
- **Forever stop:**

```
wait;
```

Process

- ابزارهای سنتز معمولاً لیست حساسیت را نادیده می گیرند
- برای مدارهای ترکیبی همهٔ ورودیها باید در لیست قرار گیرند تا نتیجهٔ شبیه سازی قبل از سنتز با سخت افزار سنتز شده مطابقت داشته باشد.
- برای مدارهای ترتیبی، فقط **clock** و ورودیهای آسنکرون (**reset, preset**) در لیست قرار گیرند.

VHDL Communication Model

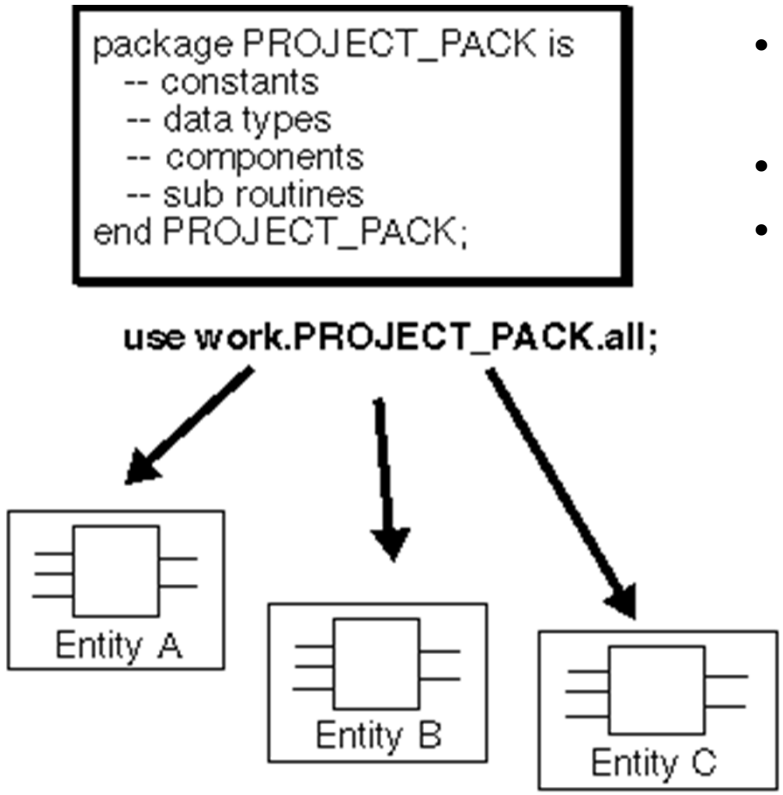


Processes are concurrent statements

- Several processes
- run in parallel
- linked by signals in the sensitivity list

Link to processes of other entity/architecture pairs via entity interface

Packages



- **Collection of definitions, datatypes, subprograms**
- **Reference made by the design team**
- **Any changes are known to the team immediately**
  - same data types ("downto vs. to")
  - extended functions for all
  - clearing errors for all

Packages

- پکیجها در دو بخش نوشته می شوند:
  - **Package Header**: شامل اعلان **subprogram**ها، **data type**ها، ...
  - **Package Body**: شامل پیاده سازی **subprogram**ها
- کامپایل را آسان می کند: فقط **header** باید خوانده شود تا تطبیق کد **VHDL** جاری با تعاریف اعلانهای قبلی مشخص شود.

Package Declaration & Body

- **Design units**
- **Package declaration**
  - Declarations of constants, types, signals, subprograms, files, components
- **Package body**
  - Definition of deferred constants
  - Subprogram bodies
- **Used with a context clause**

```
use work.example_pkg.all;
-- assuming that the package units
-- have been analyzed in the
-- logical library WORK
```

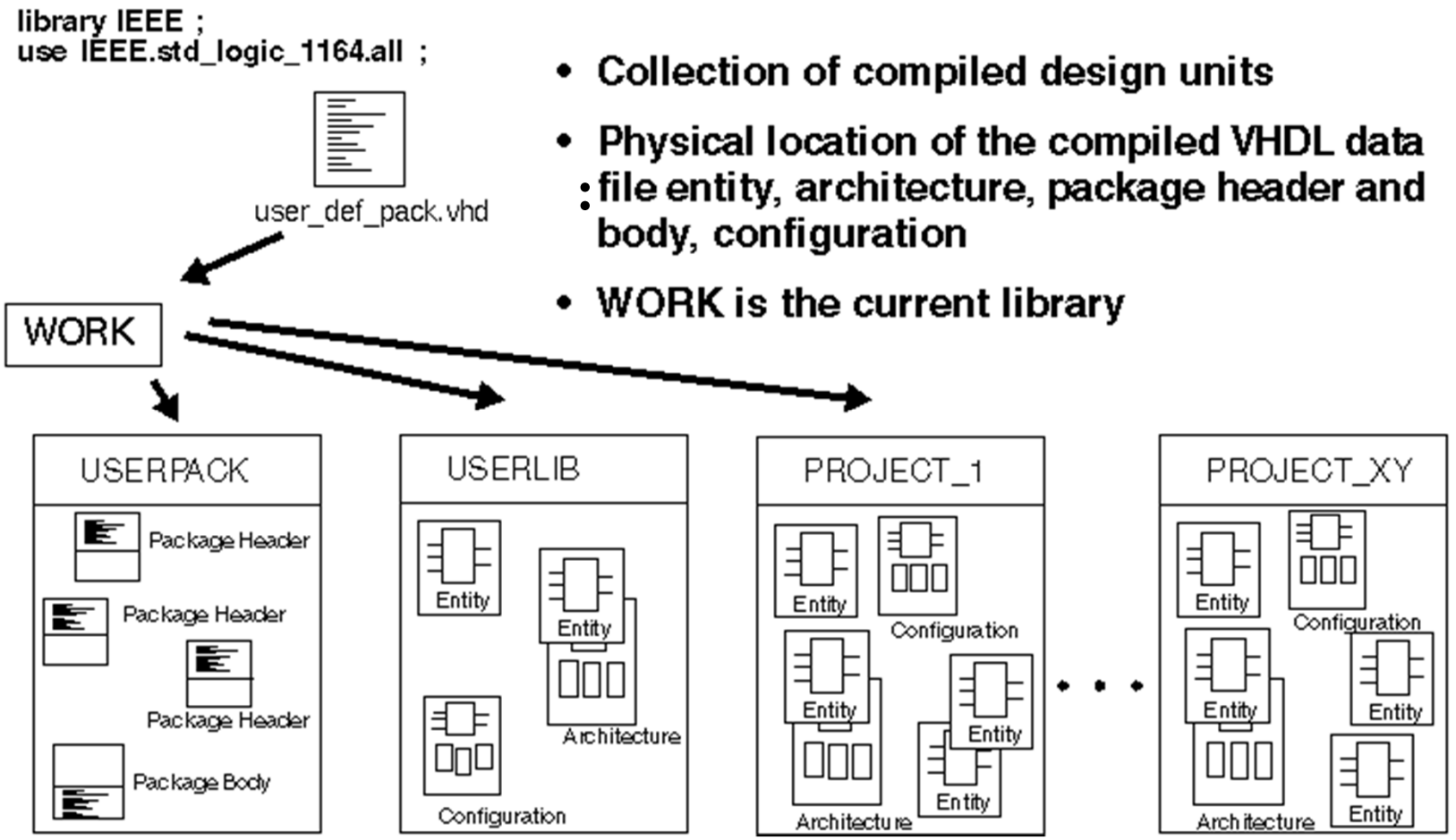
```
package example_pkg is
   constant MAX : integer := 10;
   constant MAX_SIZE : natural;  -- deferred constant
   subtype bv10 is bit_vector(MAX-1 downto 0);
   procedure proc (A : in bv10; B : out bv10);
   function func (A, B : in bv10) return bv10;
end package example_pkg;
```

```
package body example_pkg is
   constant MAX_SIZE : natural := 200;

   procedure proc (A : in bv10; B : out bv10) is
   begin
      B := abs(A);
   end procedure proc;

   function func (A, B : in bv10) return bv10 is
      variable V : bv10;
   begin
      V := A and B;
      return (not(V));
   end function func;
end package body example_pkg;
```

Library



- Collection of compiled design units
- Physical location of the compiled VHDL data file entity, architecture, package header and body, configuration
- WORK is the current library

Modeling Combinational Logic: 1-Bit Full Adder

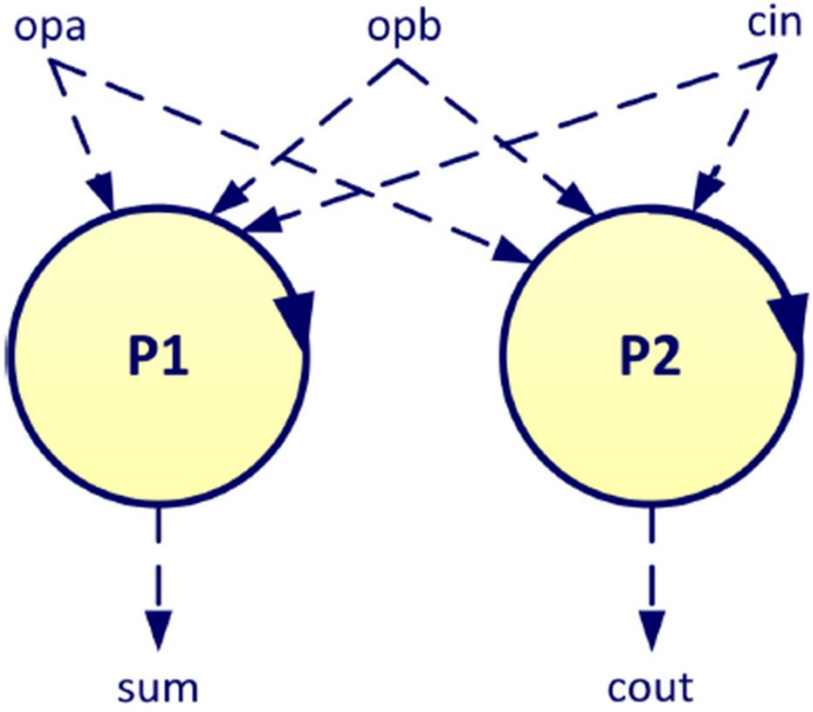
- **External (symbol) view**

- **Logic function**

$$sum = opa \oplus opb \oplus cin$$

$$cout = (opa \bullet opb) + (opa \bullet cin) + (opb \bullet cin)$$

- **Process (hardware) view**



- **Three equivalent models or modeling styles**
  - Dataflow
  - Algorithmic
  - Structural

1-Bit Full Adder: Dataflow Models

```
library ieee;                    -- context clause
use ieee.std_logic_1164.all;  -- for using type std_logic
entity fadd1 is
   port (  -- port declaration
      signal opa, opb, cin : in  std_logic;   -- input signal ports
      signal sum, cout     : out std_logic);  -- output signal ports
end entity fadd1;
```

```
architecture dfl of fadd1 is
begin
   P1 : sum  <= opa xor opb xor cin;
   P2 : cout <= (opa and opb) or (opa and cin) or (opb and cin);
end architecture dfl;
```

- **Concurrent signal assignment statements**

- **Concurrent process statements**

```
architecture dfl2 of fadd1 is
begin
   P1 : process (opa, opb, cin)   -- signal sensitivity list
   begin
      sum <= opa xor opb xor cin;
   end process P1;
   P2 : process (opa, opb, cin)   -- signal sensitivity list
   begin
      cout <= (opa and opb) or (opa and cin) or (opb and cin);
   end process P2;
end architecture dfl2;
```