



LUND
UNIVERSITY

EITF35: Introduction to Structured VLSI Design

Part 3.1.2: VHDL-4

Liang Liu
liang.liu@eit.lth.se

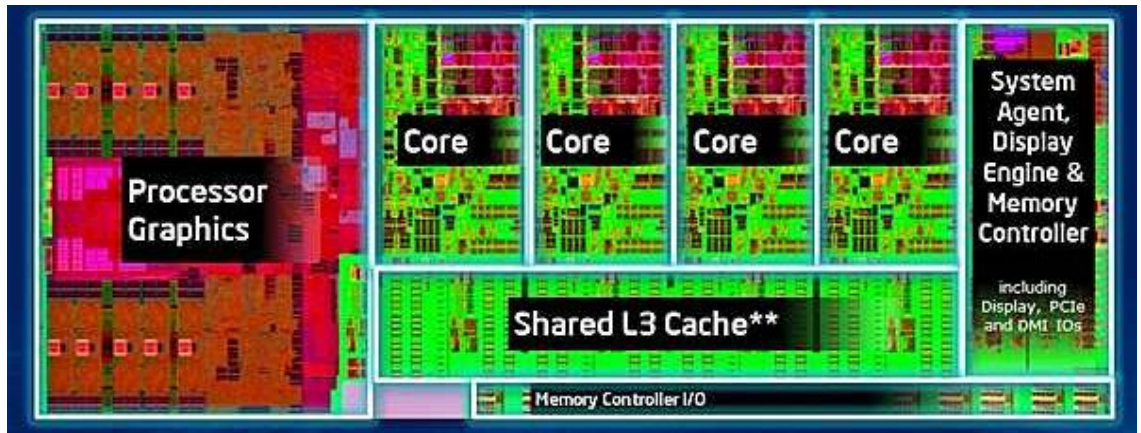
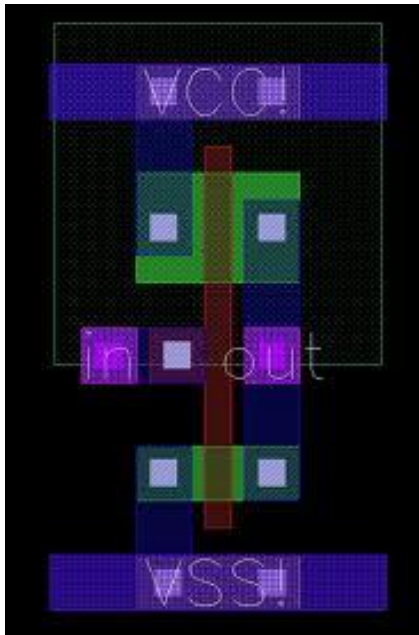


Outline

- **Handling Large Designs: Hierarchical**
- **Component**
- **Generics**
- **Configurations**
- **Library and Package**



Large Scale Design?



Hierarchical Design

□ Hierarchical design

- **Divided-and-conquer** strategy
- Divide a system into *smaller parts*
- Constructs each module *independently*
- Recursively: division process can be applied *repeatedly* and the modules can be further decomposed

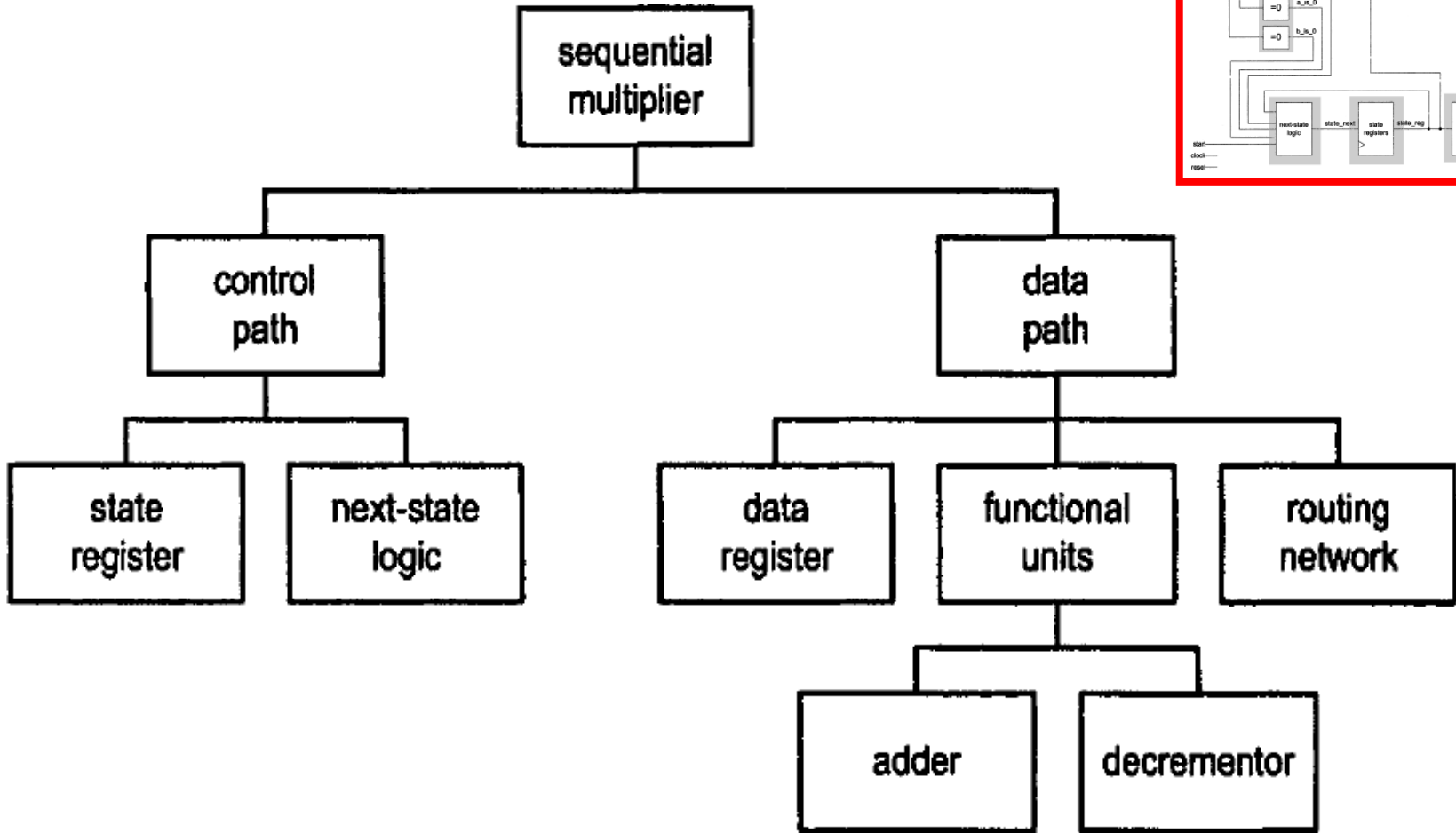
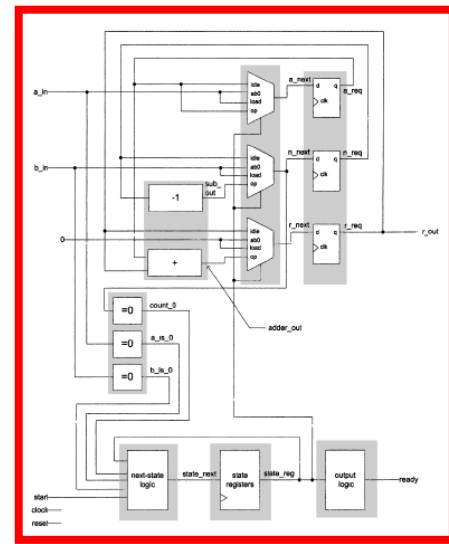


***Conquer one
problem each time***



Hierarchical Design

Example: repetitive-addition multiplier



Hierarchical Design: Advantage

□ Complexity management

- Focus on a **manageable portion** of the system, and analyze, design and verify each module in isolation.
- Construct the system concurrently by **a team of designers**.
- Allows for more **readable** design files: top-level design file as a simple integration of smaller building blocks

□ Design reuse

- Use predesigned modules or third-party cores (e.g., IP cores)
- Use the same module in different design or your future design



VHDL Supporting Hierarchical Design

□ Relevant VHDL constructs

- Component
- Generic
- Configuration
- Library
- Package
- Subprogram
- The **component**, **generic** and **configuration** constructs help to **describe** a hierarchical design.
- The **library**, **package**, and **subprogram** help the **management** of complicated code



Outline

- Handling Large Designs: Hierarchical
- **Component**
- Generics
- Configurations
- Library and Package



Component

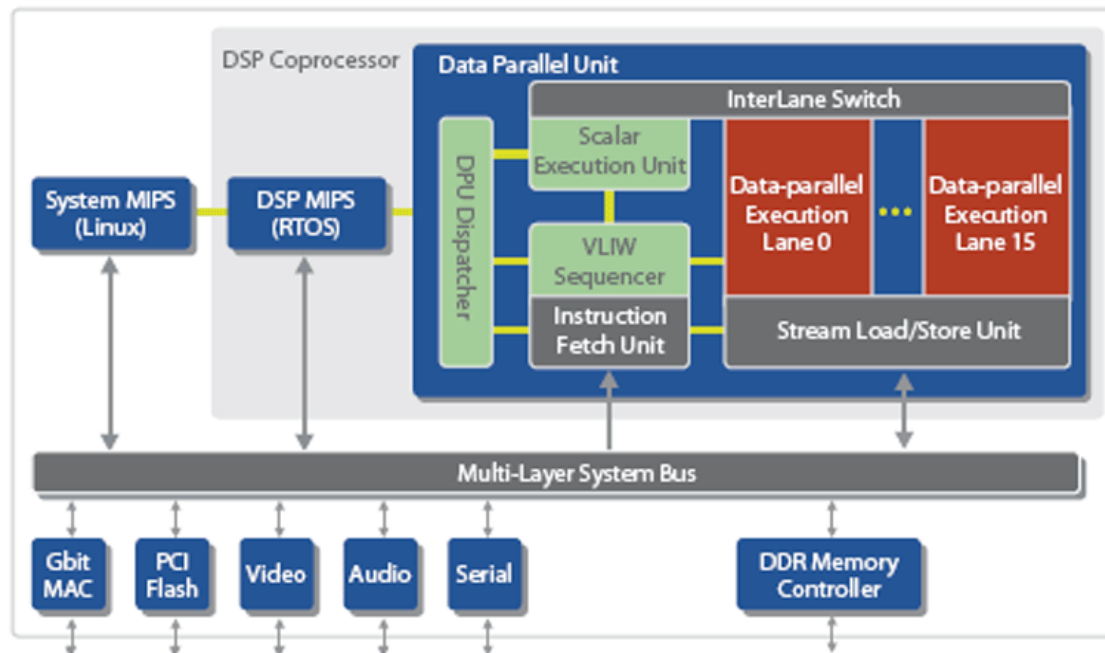
□ Hierarchical design usually shown as a block diagram

- Specify the **module** used
- The **interconnections** among these parts

□ VHDL component describes structural description in text

□ How to use a component?

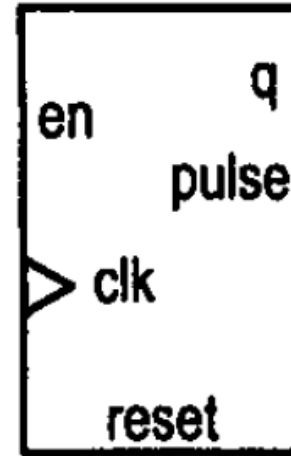
- Component **declaration** (make known)
- Component **instantiation** (create an instance)



Component Declaration

- Component declaration provides information about the external *interface* of a component
 - The *input and output* ports
 - Relevant *parameters*
- The information is similar to that provided in an entity declaration

```
component component_name is
  generic (
    generic_declaration;
    generic_declaration;
    ...
  );
  port (
    port_declaration;
    port_declaration;
    ...
  );
end component
```



Component Initialization

□ Instantiate an instance of a component

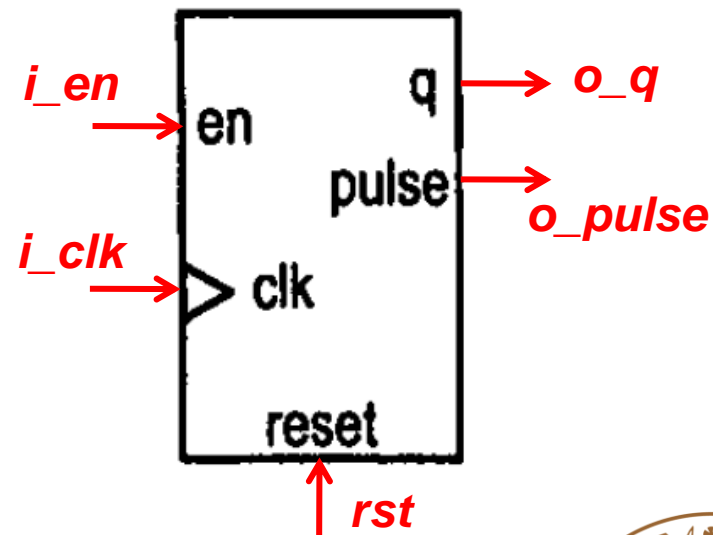
- Provide a generic value
- Map formal signals to actual signals

□ Syntax

```
instance_label: component_name  
  generic map(  
    generic_association;  
    generic_association;  
  )  
  port map(  
    port_association;  
    port_association;  
  );
```

□ Port Map

```
port_name => signal_name
```

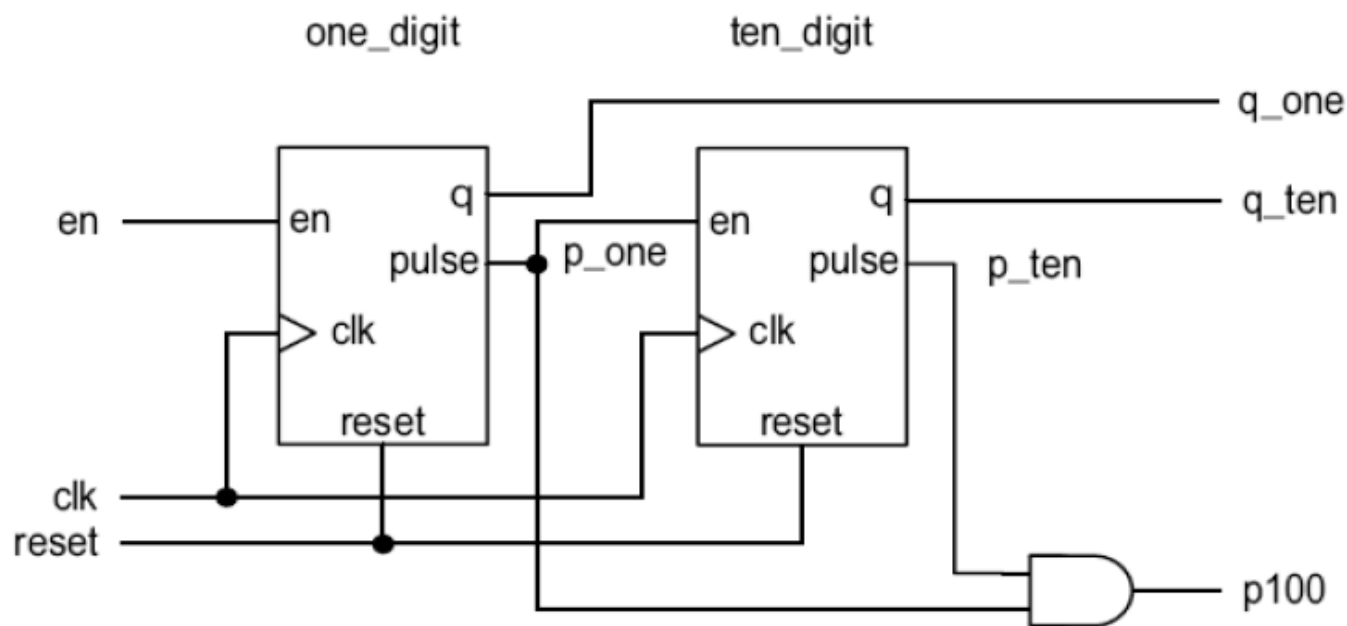


Component: Design Example

□ Mod-100 counter: 0,1,2, ... 98,99,0,1,2, ... 98,99,0

□ Step1: **block diagram** design

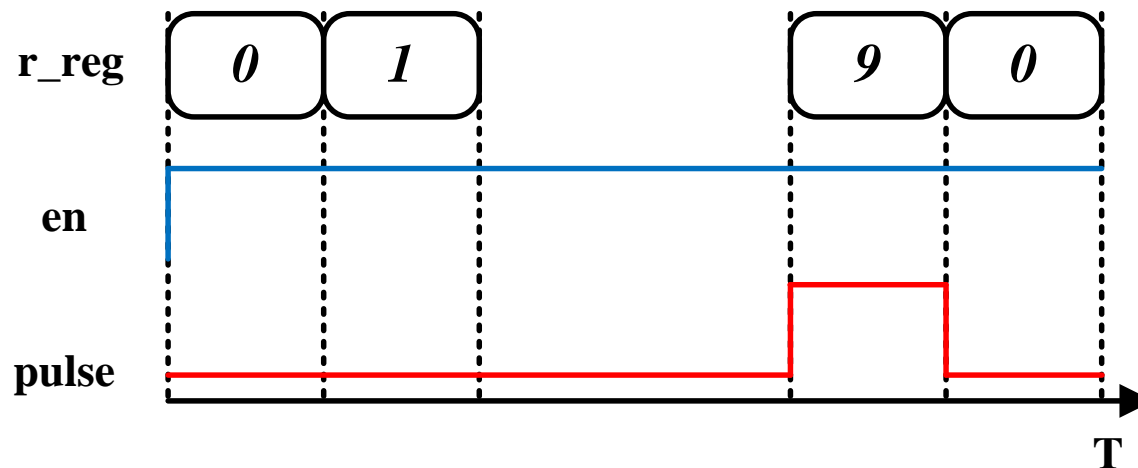
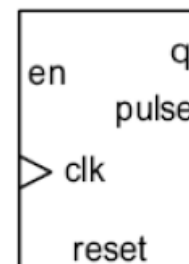
- Design two mod-10 counter
- One for one-digit, one for ten-digit
- Controlled by enable signal: one-digit counter increment every clock, ten-digit counter increment every ten clocks



Component: Design Example

Step2: component design

```
entity dec_counter is
  port(
    clk, reset: in std_logic;
    en: in std_logic;
    q: out std_logic_vector(3 downto 0);
    pulse: out std_logic
  );
end dec_counter;
architecture up_arch of dec_counter is
  signal r_reg: unsigned(3 downto 0);
  signal r_next: unsigned(3 downto 0);
  constant TEN: integer := 10;
begin
```



Component: Design Example

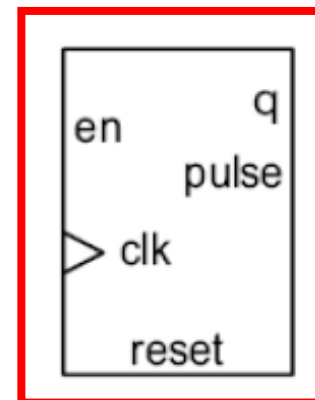
□ Step3: component declaration

```
library ieee;  
use ieee.std_logic_1164.all;  
entity hundred_counter is  
  port(  
    clk, reset: in std_logic;  
    en: in std_logic;  
    q_ten, q_one: out std_logic_vector(3 downto 0);  
    p100: out std_logic  
  );  
end hundred_counter;
```

```
architecture vhd1_87_arch of hundred_counter is
```

```
  component dec_counter  
    port(  
      clk, reset: in std_logic;  
      en: in std_logic;  
      q: out std_logic_vector(3 downto 0);  
      pulse: out std_logic  
    );
```

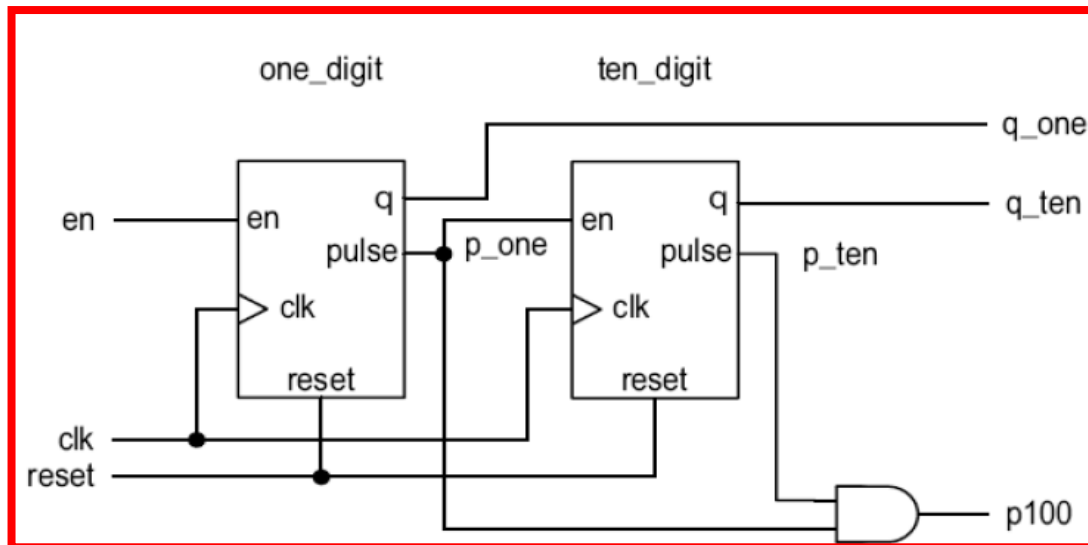
```
  end component;  
  signal p_one, p_ten: std_logic;
```



Component: Design Example

□ Step4: Instantiate and connect

```
begin
  one_digit: dec_counter
    port map (clk=>clk, reset=>reset, en=>en,
              pulse=>p_one, q=>q_one);
  ten_digit: dec_counter
    port map (clk=>clk, reset=>reset, en=>p_one,
              pulse=>p_ten, q=>q_ten);
  p100 <= p_one and p_ten;
end vhdl_87_arch;
```



Component: Attention

□ Port Mapping

- **Recommend one-to-one mapping (name association)**

```
one_digit: dec_counter
  port map (clk=>clk, reset=>reset, en=>en,
           pulse=>p_one, q=>q_one);
```

- **Do NOT recommend direct association**

```
one_digit: dec_counter
  port map (clk, reset, en, q_one, p_one);
```

- **For unused port, keyword “open”**

```
port_name => open
```

- **Do NOT OPEN inputs**
- Synthesis software should be able to optimize OPEN output



Outline

- Handling Large Designs: Hierarchical
- Component
- **Generics**
- Configurations
- Library and Package



Generic

- ❑ Mechanism to *pass info* into an entity/component
- ❑ Declared in *entity declaration* and then can be used as a **constant** in port declaration and architecture body
- ❑ Assigned a value when the component is *instantiated*
- ❑ Like a parameter, but HAS TO BE a **CONSTANT**
- ❑ Example: *step1 declaration*

```
entity entity_name is
  generic (
    generic_names: data_type;
    generic_names: data_type;
    . . .
  );
  port (
    port_names: mode data_type;
    ...
  );
end entity_name;
```



Generic

- ❑ Mechanism to *pass info* into an entity/component
- ❑ Declared in *entity declaration* and then can be used as a **constant** in port declaration and architecture body
- ❑ Assigned a value when the component is *instantiated*
- ❑ Like a parameter, but HAS TO BE a **CONSTANT**
- ❑ Example: *step1 declaration*

```
entity dec_counter is
  port (
    clk, reset: in std_logic;
    en: in std_logic;
    q: out std_logic_vector(3 downto 0);
    pulse: out std_logic
  );
end dec_counter;
```

Declare before port

Can be used in port declaration



Generic

- ❑ Mechanism to *pass info* into an entity/component
- ❑ Declared in *entity declaration* and then can be used as a **constant** in port declaration and architecture body
- ❑ Assigned a value when the component is *instantiated*
- ❑ Like a parameter, but HAS TO BE a **CONSTANT**
- ❑ Example: *step1 declaration*

```
entity para_binary_counter is
  generic(WIDTH: natural);
  port(
    clk, reset: in std_logic;
    q: out std_logic_vector(WIDTH-1 downto 0)
  );
end para_binary_counter;
```

Declare before port

Can be used in port declaration



Generic

□ Example: *step 2 utilization*

```
architecture arch of para_binary_counter is
  signal r_reg, r_next: unsigned(WIDTH-1 downto 0);
begin
  process (clk, reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;
  r_next <= r_reg + 1;
  q <= std_logic_vector(r_reg);
end arch;
```

Can also be used to parameterize signals within an architecture



Generic

□ Example: *step3 instantiation*

```
architecture vhdl_87_arch of generic_demo is
  component para_binary_counter
    generic(WIDTH: natural);
    port(
      clk, reset: in std_logic;
      q: out std_logic_vector(WIDTH-1 downto 0)
    );
  end component;
begin
  four_bit: para_binary_counter
    generic map (WIDTH=>4)
    port map (clk=>clk, reset=>reset, q=>q_4);
  twe_bit: para_binary_counter
    generic map (WIDTH=>12)
    port map (clk=>clk, reset=>reset, q=>q_12);
end vhdl_87_arch;
```

Note the
semicolon “;”



Outline

- Handling Large Designs: Hierarchical
- Component
- Generics
- **Configurations**
- Function
- Library and Package



Configuration

❑ Bind a component with an entity and an architecture

- Bind a component with a design entity
- Bind the design entity with a body architecture
- **Default binding**

❑ **Not supported by all synthesis software**

❑ **Suggestion: Use only in testbench**

- Testbench is reused by declaring a different configuration
- Examples:
 - ❑ ***Behaviorial model***
 - ❑ ***Gate-level model***

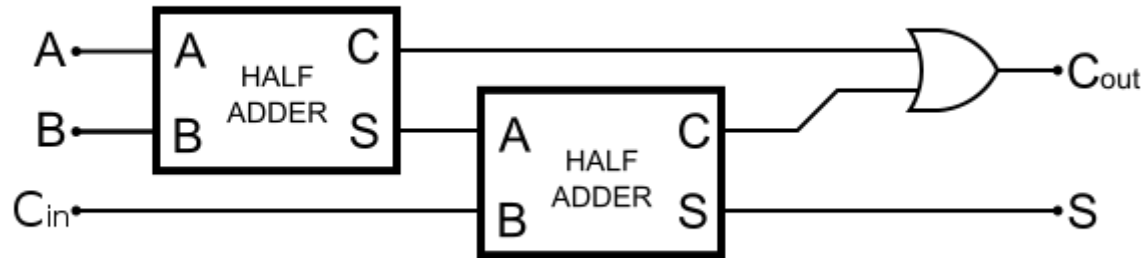


Configuration Declaration

```
configuration conf_name of entity_name is
  for architecture_name
    for instance_label: component_name
      use entity lib_name.bound_entity_name(bound_arch_name);
    end for;
    for instance_label: component_name
      use entity lib_name.bound_entity_name(bound_arch_name);
    end for;
    .
    .
  end for;
end;
```



Configuration-Example



entity name and
component name differs

```
configuration THREE of FULLADDER is
  for STRUCTURAL
    for INST_HA1, INST_HA2: HA
      use entity WORK.HALFADDER (CONCURRENT);
    end for;
    for INST_XOR: XOR
      use entity WORK.XOR2D1 (CONCURRENT);
    end for;
  end for;
end THREE;
```



Suggestion:

- ❑ One entity per file, file name the same with entity name
- ❑ Do NOT put critical path between component



Outline

- Handling Large Designs: Hierarchical
- Component
- Generics
- Configurations
- **Library and Package**



Libraries and Packages

□ Used to declare and store:

- Components
- Type declarations
- Functions
- Procedures

□ Packages and libraries provide the ability to reuse constructs in multiple entities and architectures



Libraries

- ❑ Library is a place to which design units may be compiled
- ❑ Two predefined libraries are the *IEEE* and *WORK* libraries
- ❑ *WORK* is the default library
- ❑ *IEEE* standard library contains the IEEE standard design units.
 - std_logic_1164
 - numeric_std
- ❑ IEEE is non-default library, must be declared:

```
library ieee;
```

- ❑ Design units within the library must also be made visible via the use clause.

```
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;
```



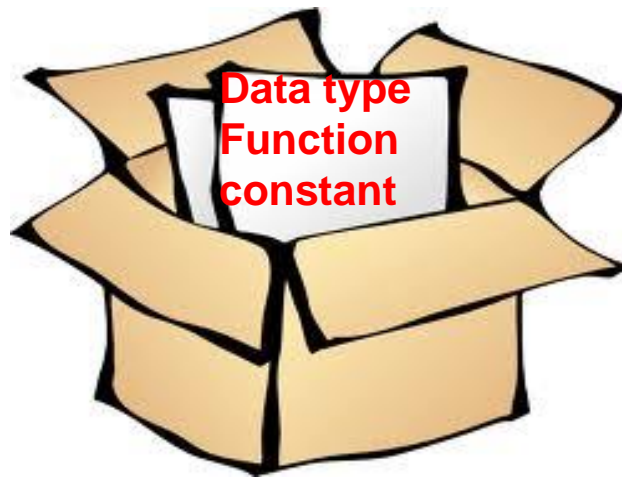
Packages

□ Declarations in an architecture

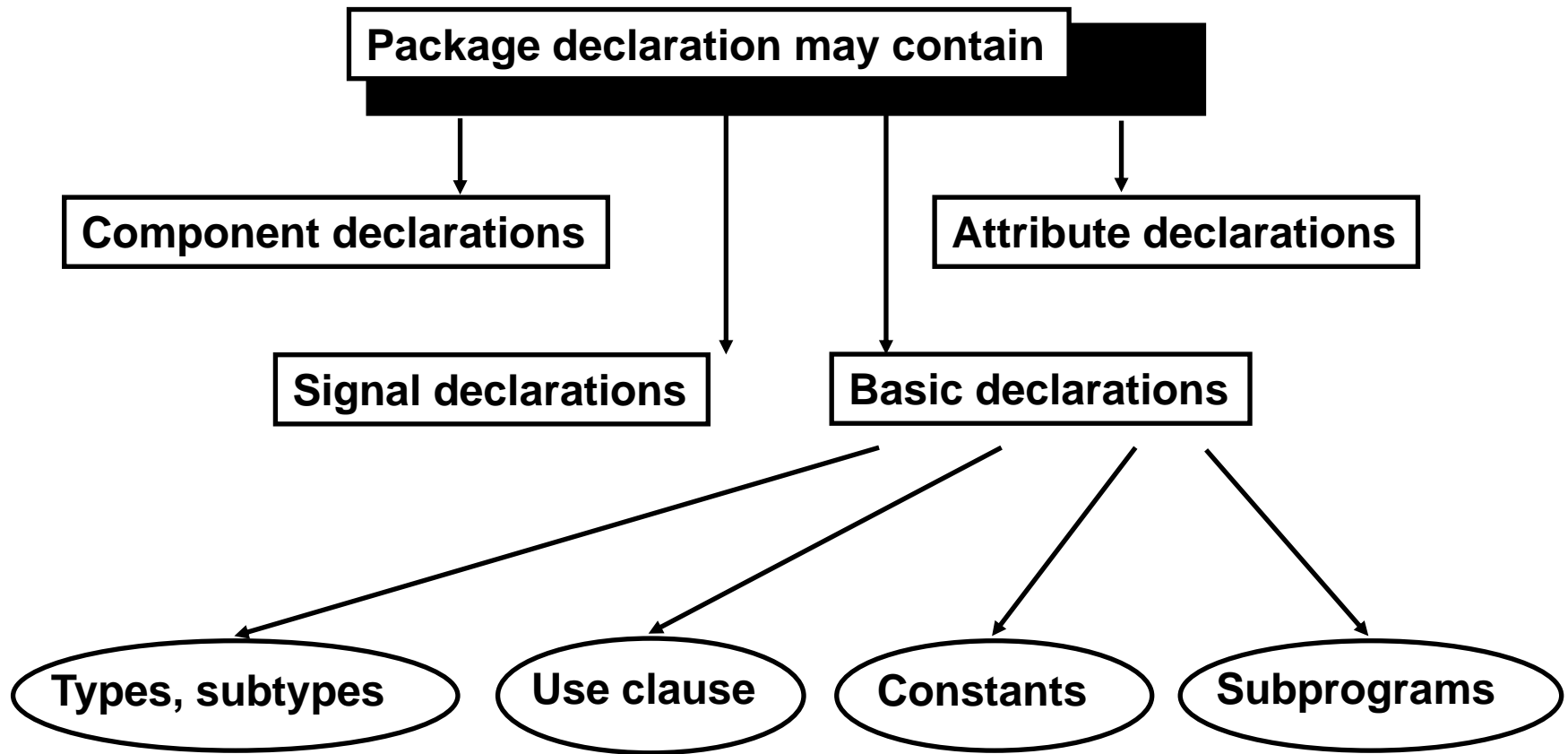
- Consist of the declarations of *constants*, *data types*, *components*, *functions* and so on
- Must be *duplicated* in many different design units, for hierarchical design

□ Packages

- Organize and store declaration information



Packages Declaration

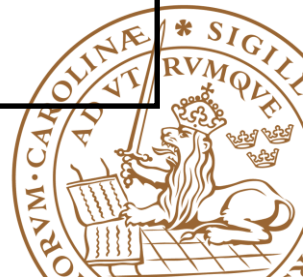


```
package package_name is
  declaration item;
  declaration item;
  . . .
end package_name;
```



Packages Declaration: Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package my_package is
    type binary is (on, off);
    constant C_ROUTING_ID_BITS: integer := 3;
    component counter_dec is
        generic (constant WIDTH: integer);
        port (
            clk_in, rst_n: in std_logic;
            en: in std_logic;
            q: out std_logic_vector (WIDTH-1 downto 0);
            puls: out std_logic
        );
    end component;
end my_package;
```



Package: How to use?

- A package is made visible using the **use** clause

```
use library_name.package_name.item
```

```
use work.my_package.binary;  
use work.my_package.counter_dec;  
... entity declaration ...  
... architecture declaration ...
```

use the *binary* and *counter_dec* declarations

```
use work.my_package.all;  
... entity declaration ...  
... architecture declaration ...
```

use *all* of the declarations in package my_package



Reading advice

FSMD: *RTL Hardware Design Using VHDL*, Chapter 11,
P373-P420

Hierarchical VHDL: *RTL Hardware Design Using VHDL*,
Chapter 13, P473-P498

