SYSTEM MODELLING AND SYNTHESIS WITH HDL

DTEK0078
2022 Lecture 2

UNIVERSITY OF TURKU

Moodle

- Enrollment key is

  HDL2022

- Course id is

  14534

- Course name is

  System Modelling and Synthesis with HDL S2022

UNIVERSITY OF TURKU

NEWS

First week (29.8. - 2.9.2022). Lectures every day as scheduled:

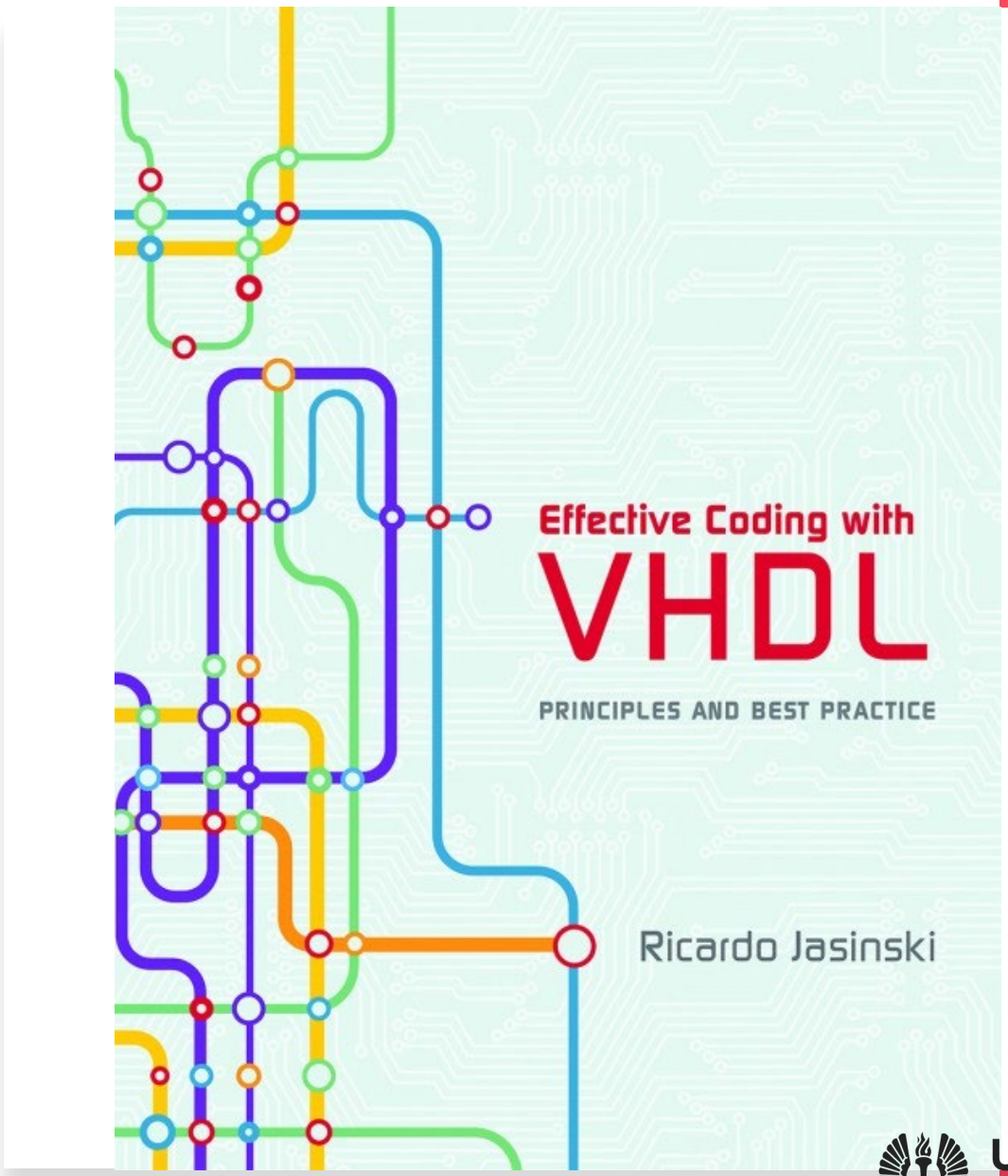
- Tue 30.08.2022 14:15-16:00 110C, Agora 110C
- Wed 31.08.2022 12:15-14:00 110C, Agora 110C
- ~~Thu 01.09.2022 14:15-16:00 110C, Agora 110C~~ CANCELLED BECAUSE OF OVERLAPPIN MANDATORY COURSE

Second week. Lectures are on Tuesday and Wednesday. On Thursday, we have the first exercise.

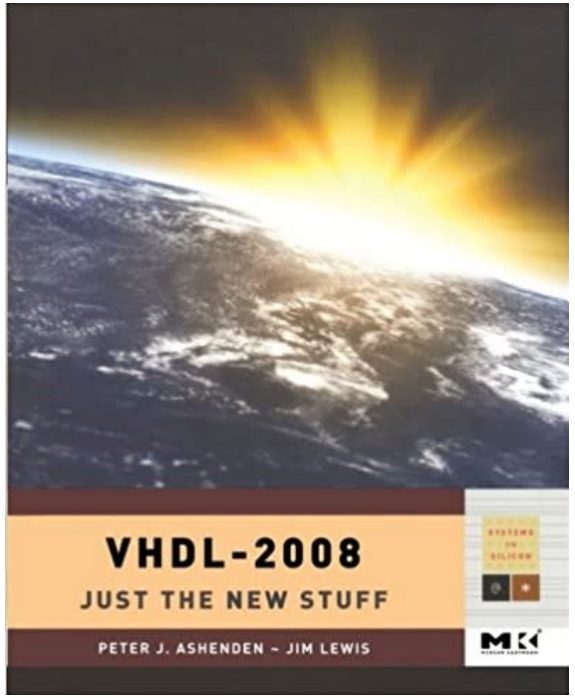
UNIVERSITY OF TURKU

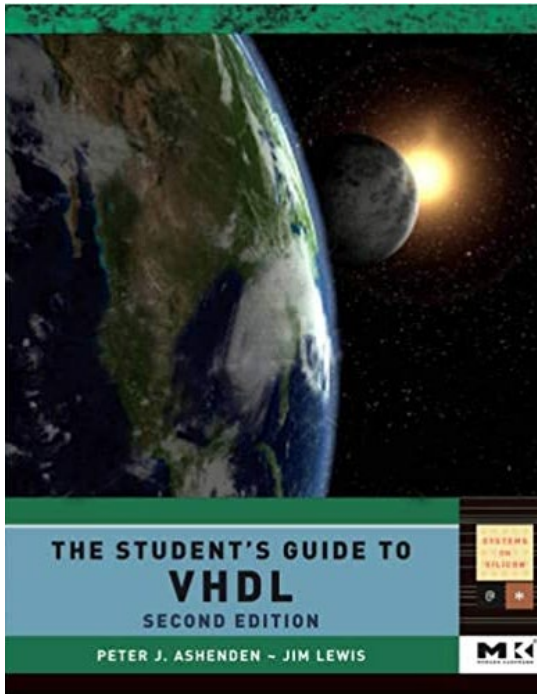
Course Book

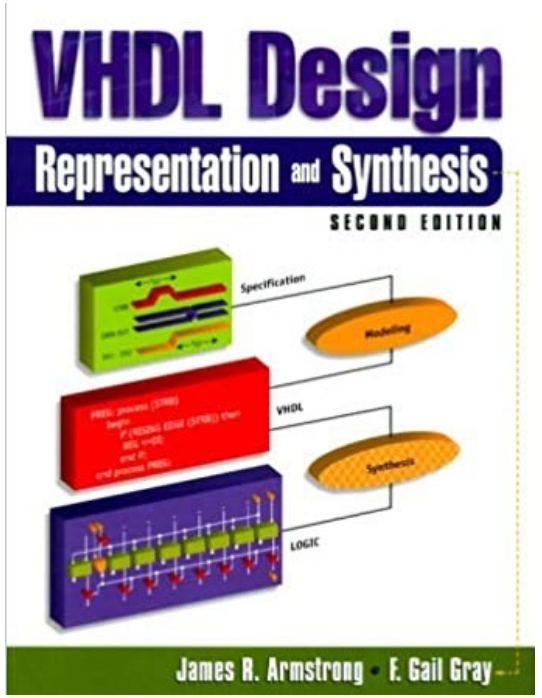
- Effective Coding with VHDL: Principles and Best Practice By Ricardo Jasinski
- Can be found here https://mitpress.mit.edu/books/effective-coding-vhdl
  - On other online bookstores as well

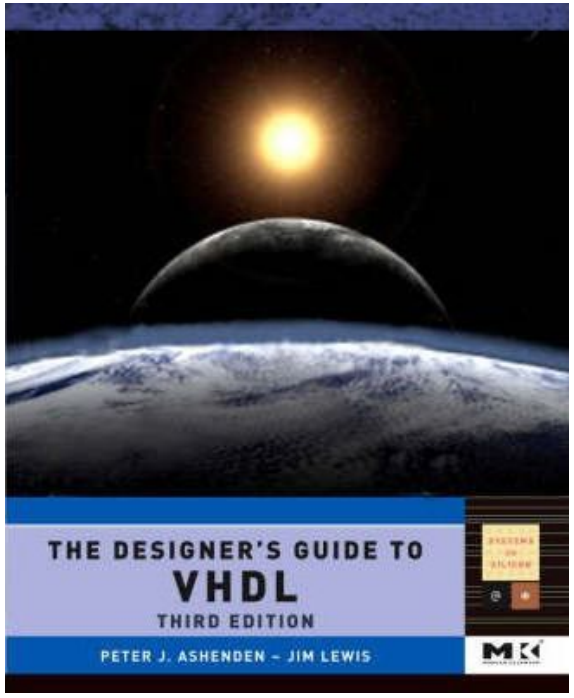


UNIVERSITY OF TURKU









Other Good books

Good to know

UNIVERSITY OF TURKU

Free Format

- VHDL is a “free format” language
- No formatting conventions, such as spacing or indentation imposed by VHDL compilers
  - Space and carriage return treated the same way

  Example:

```
if (x=y) then
```

  or

```
if  (x=y)          then
```

  or

```
if (x = y)
 then
```

  are all equivalent

UNIVERSITY OF TURKU

Naming and Labeling

- VHDL is case insensitive

  Example:

  Names or labels

  `databus`

  `Databus`

  `DataBus`

  `DATABUS`

  are all equivalent

UNIVERSITY OF TURKU

Comments

- Comments are indicated with a “double dash”

  “--”

  - Comment indicator can be placed anywhere in the line
  - Any text that follows in the same line is treated as a comment
  - Carriage return terminates a comment
  - No method for commenting a block extending over a couple of lines

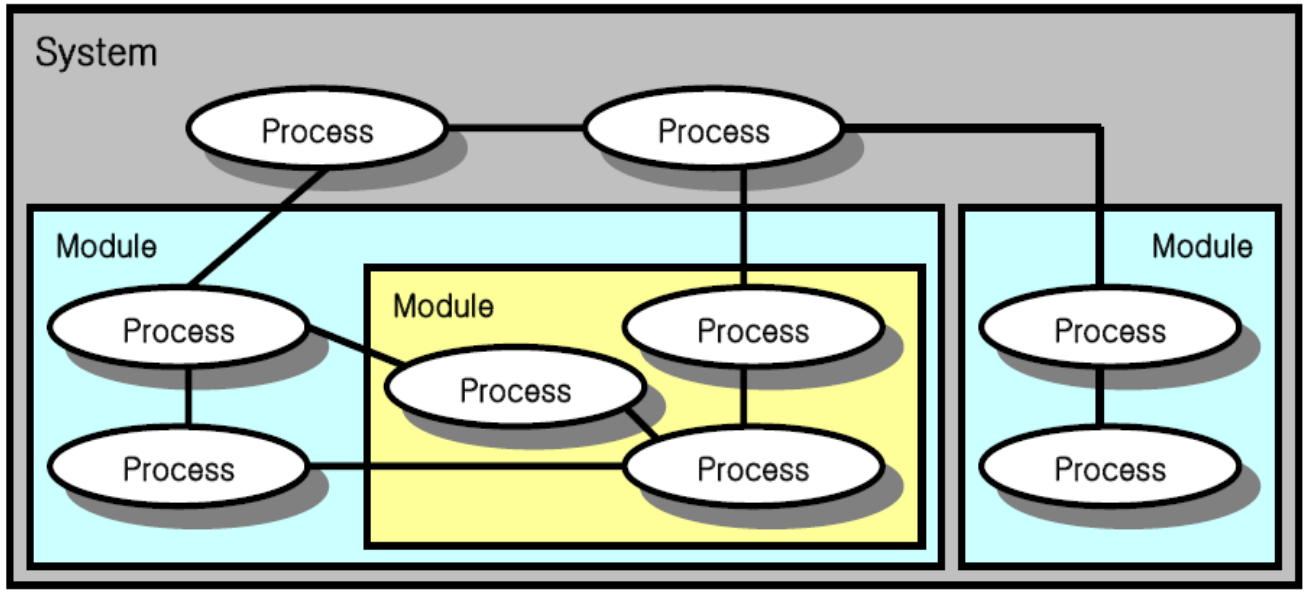
Examples:

```
-- this is a comment
xy <= x * y;     -- comments also work in the end of a line
```

UNIVERSITY OF TURKU

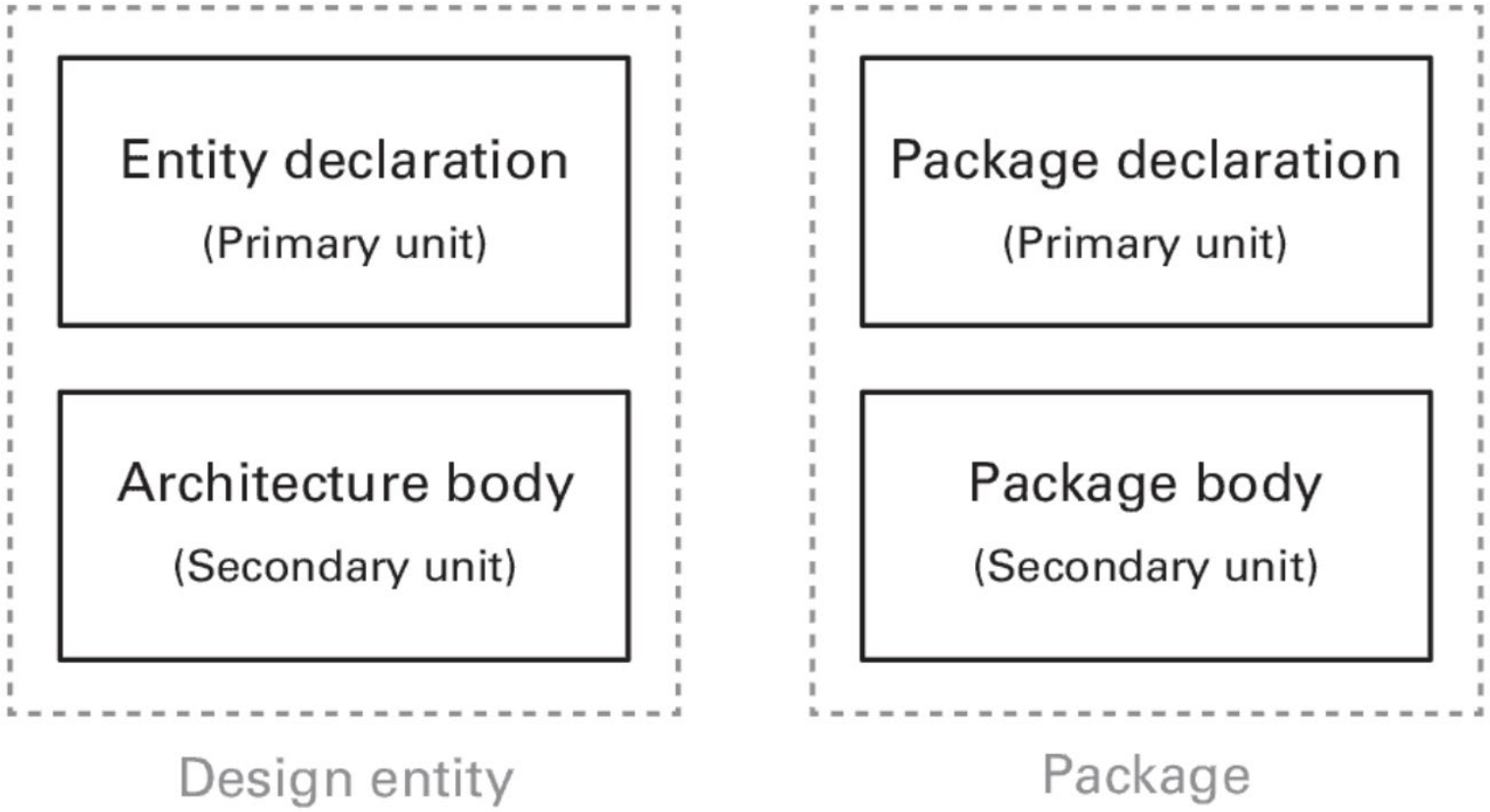
Basic VHDL Concepts

- Entities
- Architectures
- Packages



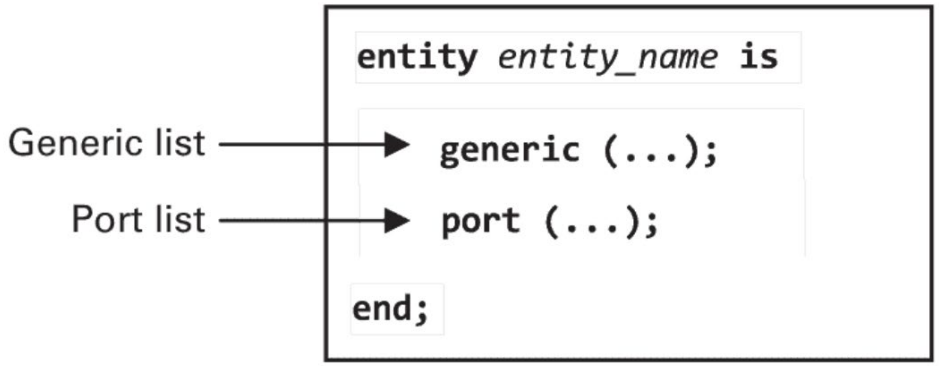
UNIVERSITY OF TURKU

Main Design Units in VHDL



UNIVERSITY OF TURKU

Entity and Architecture



- An entity declaration describes a module from an external viewpoint
- A generic list and a port list. Both lists are optional, but any entity that is intended for synthesis needs ports
- A port can be
  - IN
  - OUT
  - INOUT
  - BUFFER (do not use!, use OUT instead (HDL2008))

UNIVERSITY OF TURKU

Port Modes

The *Port Mode* of the interface describes the direction in which data travels with respect to the *component*

- **In**: Signal can only be read, that is, it can appear **only on the right side** of a signal or variable assignment
- **Inout**: The value of a bi-directional port can be read and written. It can appear on **both sides** of a signal assignment
- **Out**: Depending on the VHDL version **out** can only be written or read as well
- **Buffer:** Output signal, but it can also be read, and therefore a signal can appear on **both sides** of a signal assignment.
  - Not recommended to be used in the synthesisable code OR in general

UNIVERSITY OF TURKU

Port Modes Out and Buffer

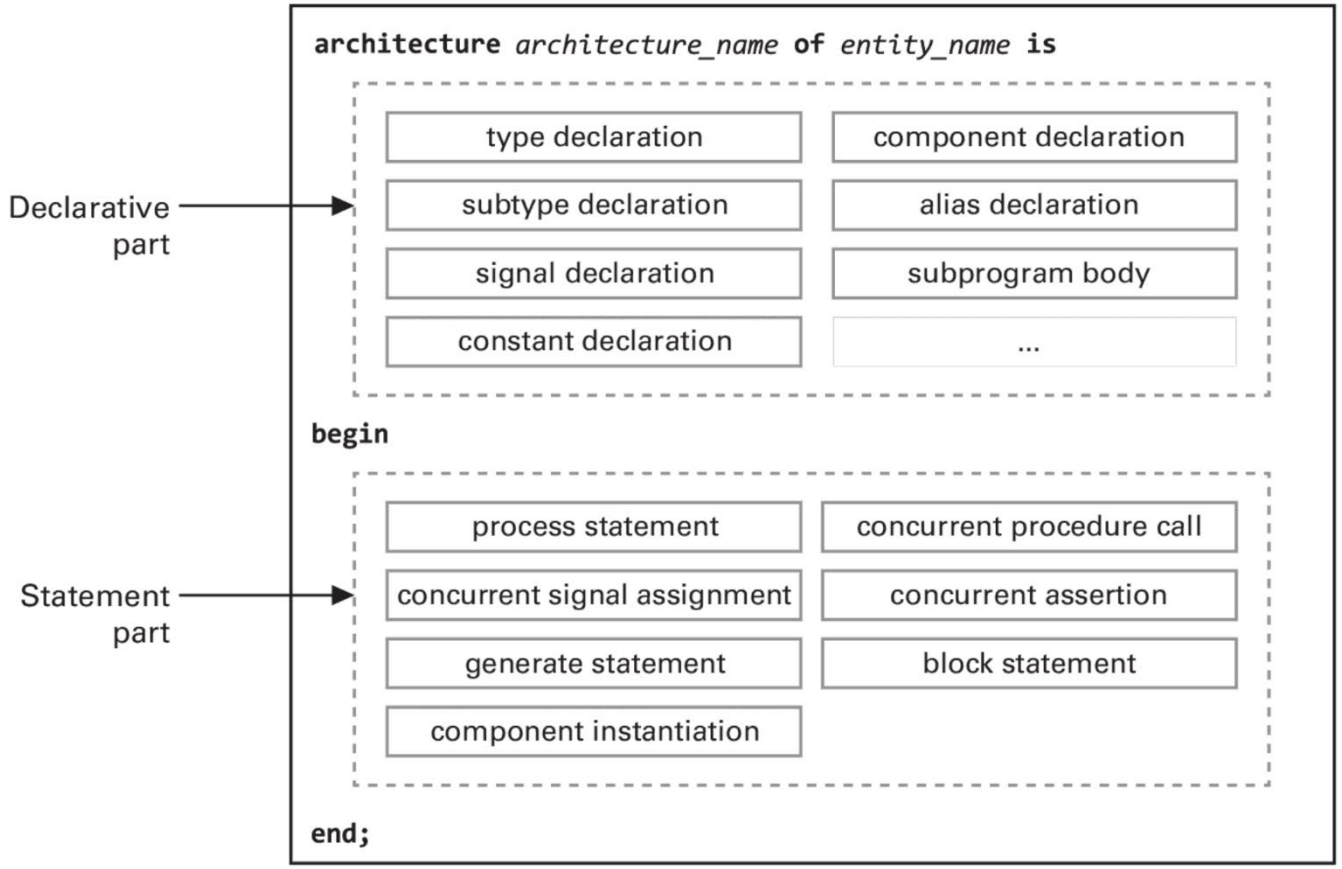
- **Out** port is intended to be used when there is no internal use of the port's value
- **Buffer** port is intended to be used when the port's value is used within the module

- In VHDL-2002, the rules for **buffer** ports were relaxed
  - Allowed to be connected to external **out** and **buffer** ports
- In VHDL-2008, the restriction on reading **out** ports is removed

UNIVERSITY OF TURKU

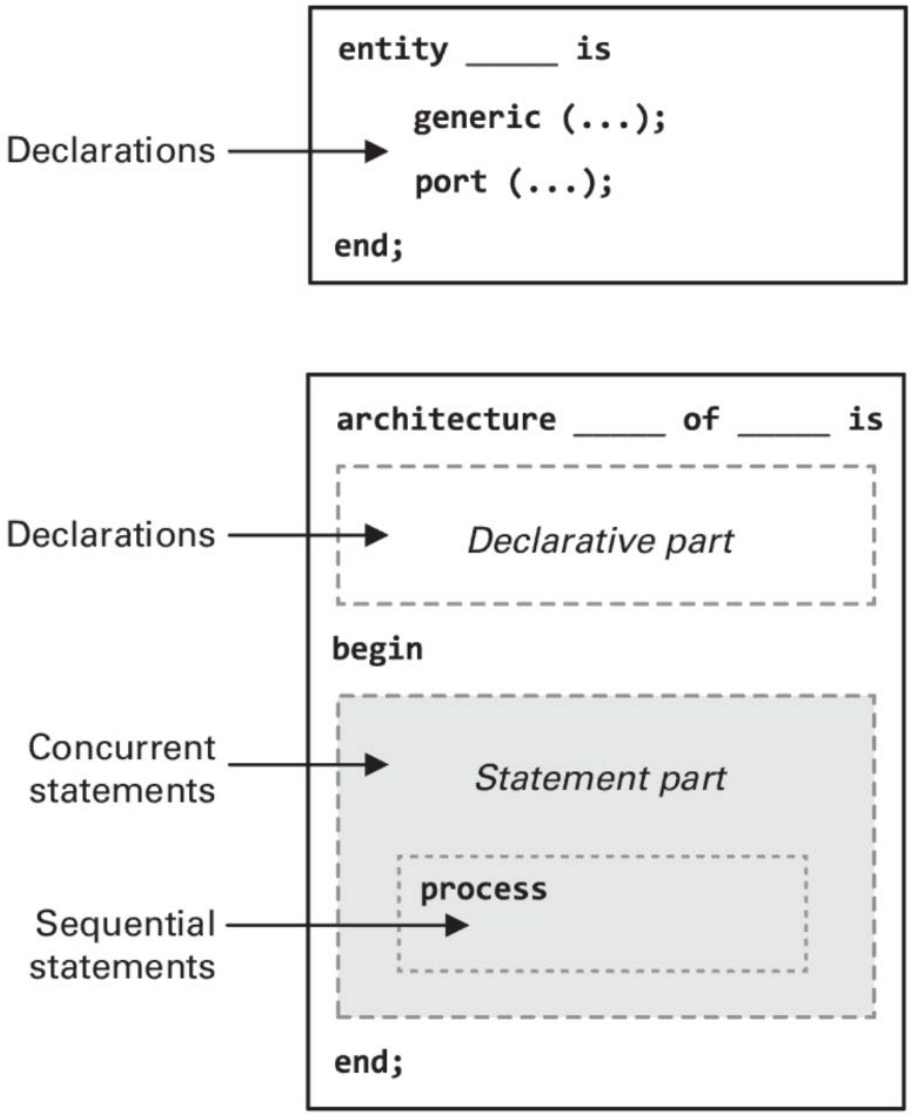
Entity and Architecture

- An architecture body specifies the operation of a module, describing its internal behavior or structure
- An architecture body is always associated with a single entity declaration



UNIVERSITY OF TURKU

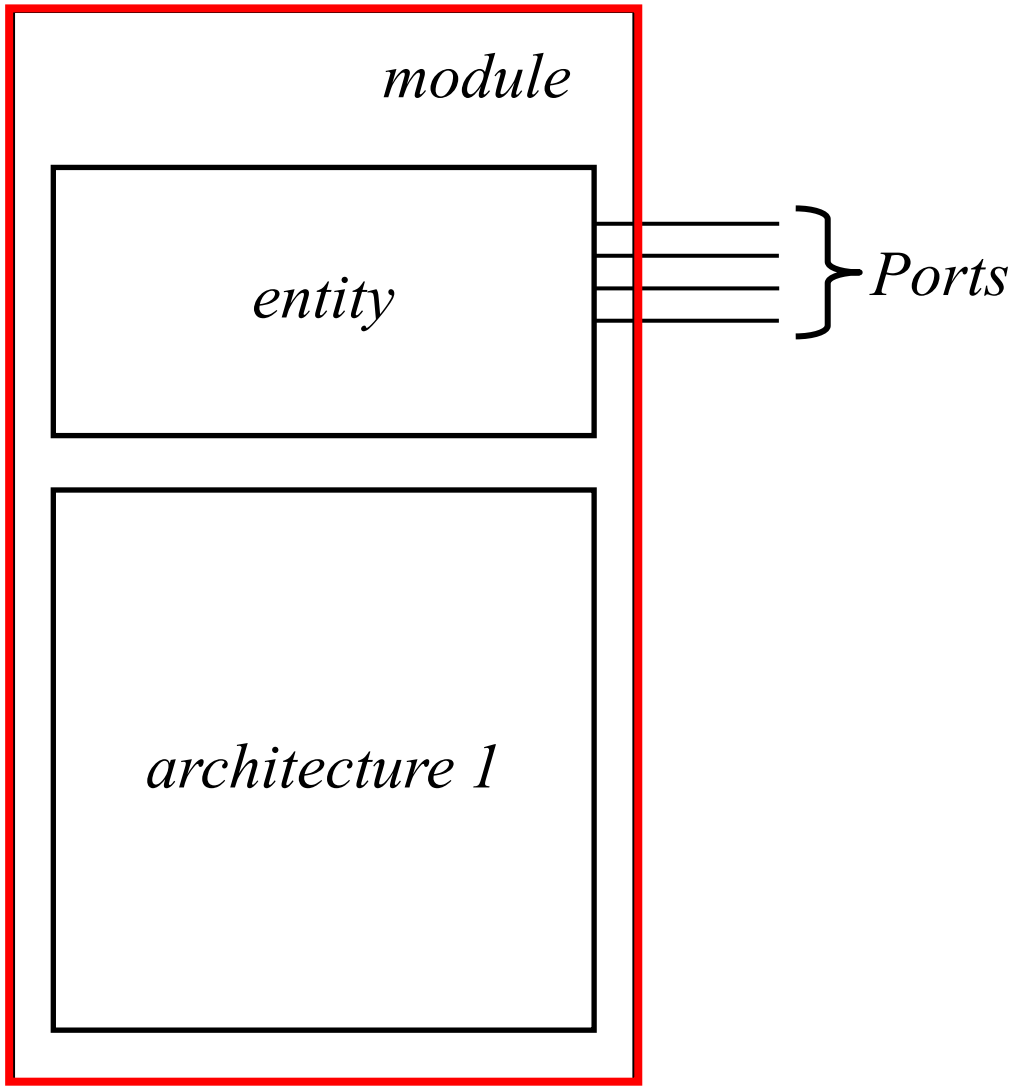
Statements and Declarations in VHDL code



UNIVERSITY OF TURKU

Design Entity

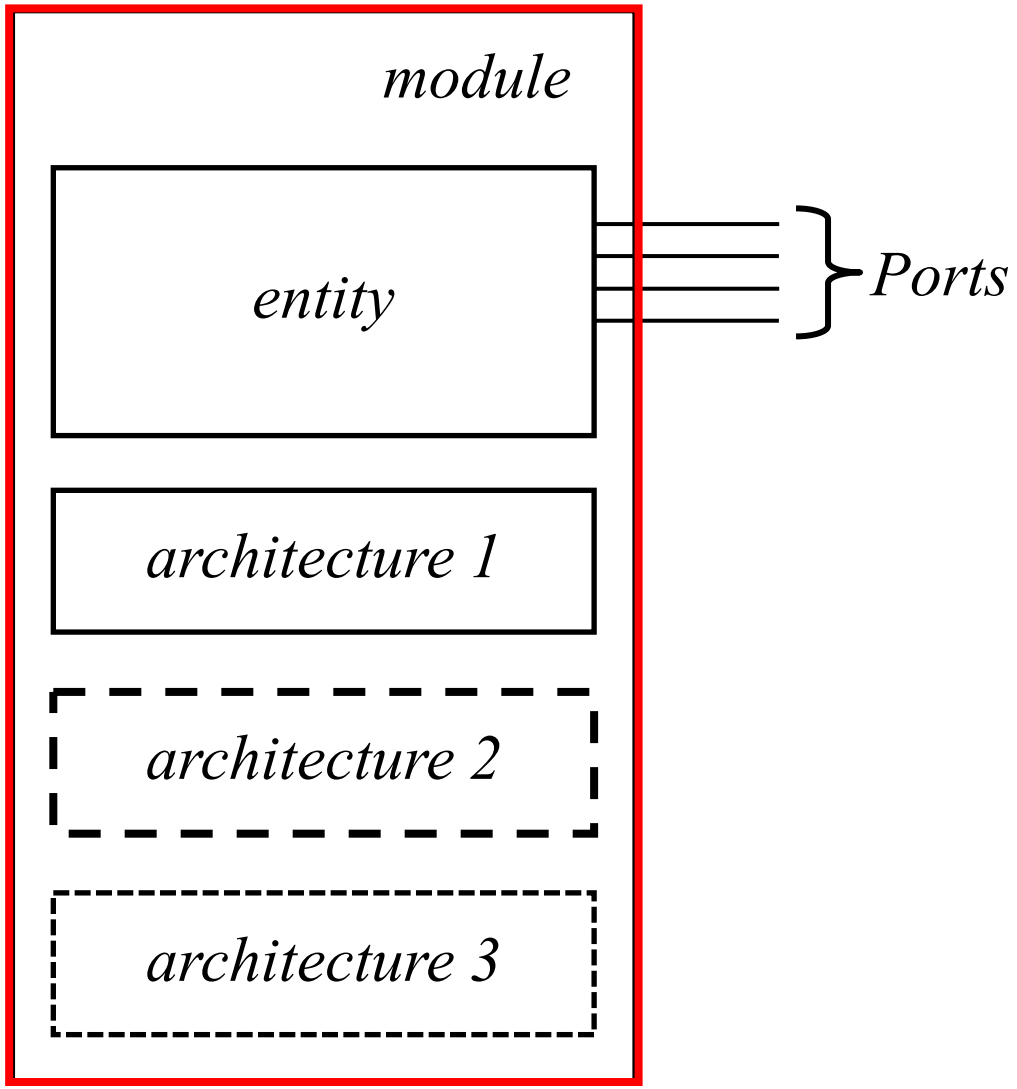
- Entity
  - Defines the interface of the module
- Architecture
  - Defines the behaviour of the module



UNIVERSITY OF TURKU

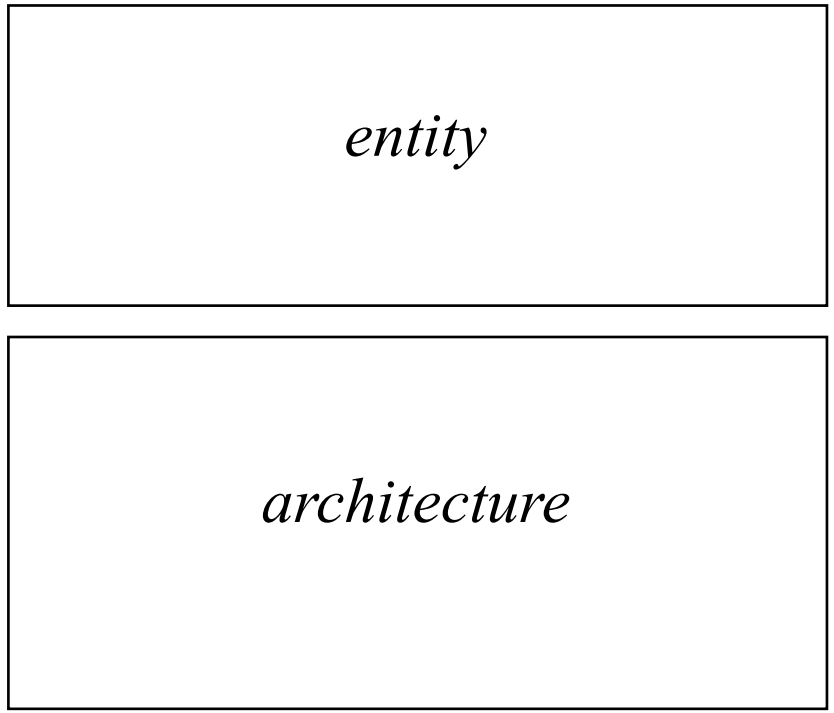
Design Entity

- Entity
  - Defines the interface of the module
- Several Architecture
  - Different implementations of the model
  - Same entity



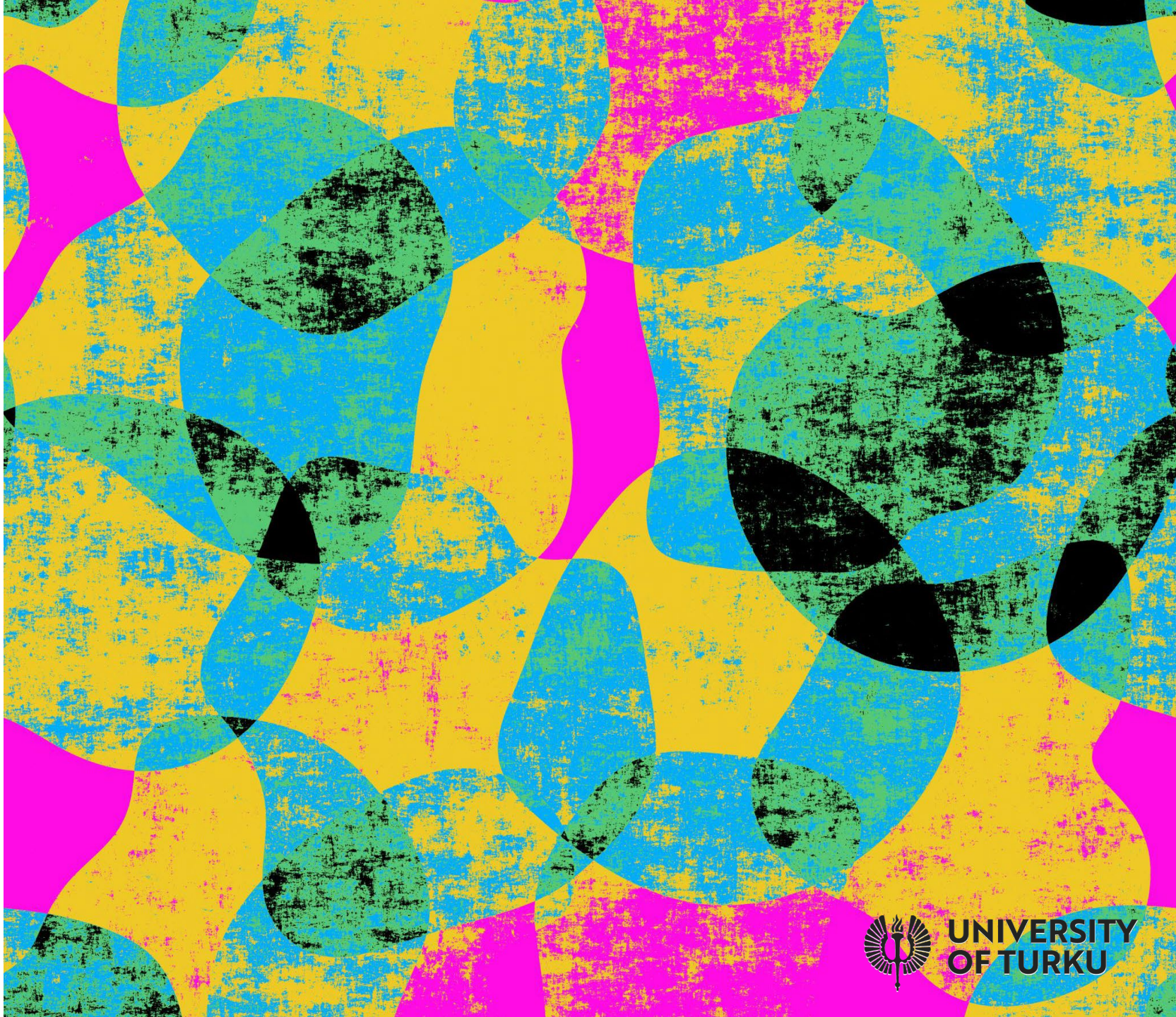
UNIVERSITY OF TURKU

An Example Module



UNIVERSITY OF TURKU

HDL Coding Styles



HDL Coding Styles

Dataflow	Behavioural	Structural
• Concurrents signal assignments	• Process statements	• Component instantiations

- Depending on the system being desinged, one style may be the most intuitive to use

UNIVERSITY OF TURKU

Dataflow Modelling

- Functionality is implemented with concurrent statements
- Statements define the actual flow of data
- In statements, one can use logical operators as well as selected and conditional signal assingments

```
architecture dataflow of SR_flipflop is
begin

  gate_1 : q <= s_n nand q_n;
  gate_2 : q_n <= r_n nand q;

end architecture dataflow;
```

UNIVERSITY OF TURKU

Behavioural Modelling

- Algorithmic presentation
  - No implementation information
  - Defines how the system is to behave

```
architecture checking of SR_flipflop is
begin

    set_reset : process (S, R) is
    begin
      assert S nand R;
      if S then
        Q <= '1';
      end if;
      if R then
        Q <= '0';
      end if;
    end process set_reset;

end architecture checking;
```

UNIVERSITY OF TURKU

Structural Modelling

- System is implemented as a composition of subsystems
- Interconnection of components
  - Signals for internal connections

```
architecture struct of reg4 is
begin

  bit0 : entity work.edge_triggered_Dff(behavioral)
    port map (d0, clk, clr, q0);
  bit1 : entity work.edge_triggered_Dff(behavioral)
    port map (d1, clk, clr, q1);
  bit2 : entity work.edge_triggered_Dff(behavioral)
    port map (d2, clk, clr, q2);
  bit3 : entity work.edge_triggered_Dff(behavioral)
    port map (d3, clk, clr, q3);

end architecture struct;
```

UNIVERSITY OF TURKU

HDL Coding Styles

```
architecture checking of SR_flipflop is
begin

    set_reset : process (S, R) is
    begin
      assert S nand R;
      if S then
        Q <= '1';
      end if;
      if R then
        Q <= '0';
      end if;
    end process set_reset;

end architecture checking;
```

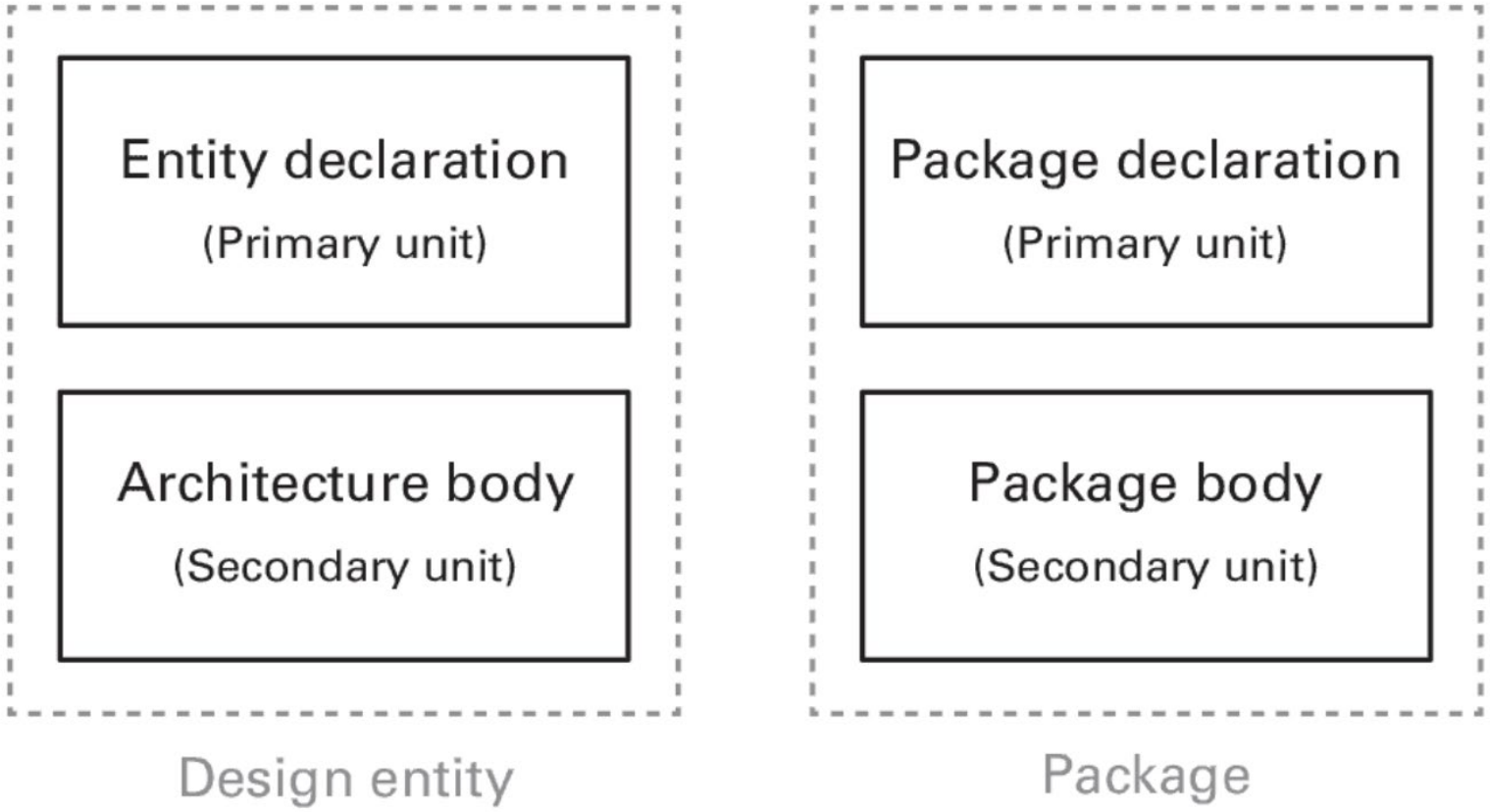
```
architecture struct of reg4 is
begin

  bit0 : entity work.edge_triggered_Dff(behavioral)
    port map (d0, clk, clr, q0);
  bit1 : entity work.edge_triggered_Dff(behavioral)
    port map (d1, clk, clr, q1);
  bit2 : entity work.edge_triggered_Dff(behavioral)
    port map (d2, clk, clr, q2);
  bit3 : entity work.edge_triggered_Dff(behavioral)
    port map (d3, clk, clr, q3);

end architecture struct;
```

```
architecture dataflow of SR_flipflop is
begin

  gate_1 : q <= s_n nand q_n;
  gate_2 : q_n <= r_n nand q;

end architecture dataflow;
```

Packages

UNIVERSITY OF TURKU

Main Design Units in VHDL



UNIVERSITY OF TURKU

Package Declaration and Package Body

- Packages are a mechanism for sharing and reusing code across design units
- The most common items in *a package declaration* are types, constants, subprogram declarations, and component declarations
  - Anything that should not be publicly visible should be in *the package body* instead
- In practice, every design uses packages
  - The standard packages provide types that are commonly needed in a large number of designs, as well as the operations to work with these types

UNIVERSITY OF TURKU

Libraries and Packages

- A *design library* is a logical storage area for compiled design units

Packages	Library	Standard Number	Standard Title
standard, textio	std	IEEE Std 1076	IEEE Standard VHDL Language Reference Manual
std_logic_1164	ieee	IEEE Std 1164	IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)
numeric_bit, numeric_std	ieee	IEEE Std 1076.3	IEEE Standard VHDL Synthesis Packages
std_logic_unsigned, std_logic_signed	ieee	None	None
std_logic_arith	ieee	None	None

- Libraries can be found in Vivado's install directory:

Vivado\2021.2\data\vhdl\src\ieee_2008

UNIVERSITY OF TURKU

Visibility of Declarations

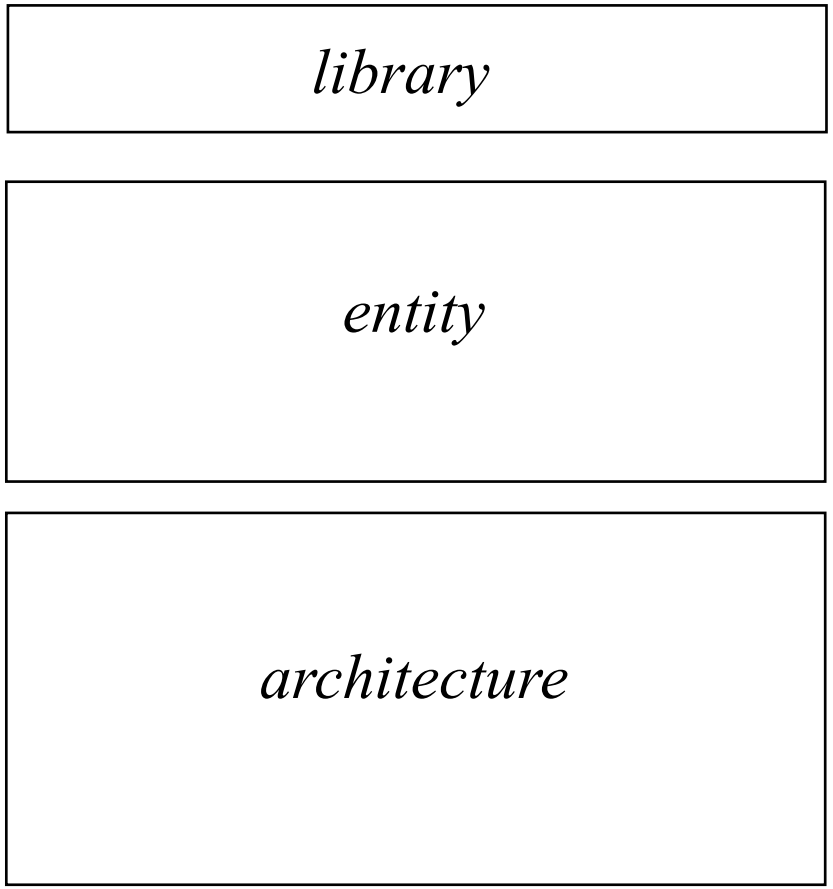
- A use clause turns declarations that were visible only by selection into directly visible declarations

```
-- Declare the name of a library so that we can use it in the code.
library ieee;
-- "Import" declarations from package std_logic_1164 into the current design unit.
use ieee.std_logic_1164.all;
```

UNIVERSITY OF TURKU

Libraries and Packages

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

entity half_adder is
port(
  x,y: in std_logic;
  sum, carry: out std_logic);
end half_adder;

architecture myadder of half_adder is
begin
  sum <= x xor y;
  carry <= x and y;
end myadder;
```



UNIVERSITY OF TURKU

Libraries and Packages

Function	Operand Types	Result Types	Package
+	signed, unsigned integer, natural	signed, unsigned	numeric_std
–	signed, unsigned integer, natural	signed, unsigned	numeric_std
*	signed, unsigned integer, natural	signed, unsigned	numeric_std
/	signed, unsigned integer, natural	signed, unsigned	numeric_std
rem	signed, unsigned integer, natural	signed, unsigned	numeric_std
mod	signed, unsigned integer, natural	signed, unsigned	numeric_std
+	signed, unsigned integer, std_logic	signed, unsigned std_logic_vector	std_logic_arith
–	signed, unsigned integer, std_logic	signed, unsigned std_logic_vector	std_logic_arith
*	signed, unsigned integer, std_logic	signed, unsigned, std_logic_vector	std_logic_arith
+	std_logic, integer, std_logic_vector	std_logic_vector	std_logic_unsigned
–	std_logic, integer, std_logic_vector	std_logic_vector	std_logic_unsigned
*	std_logic, integer, std_logic_vector	std_logic_vector	std_logic_unsigned
>	signed, unsigned integer, natural	boolean	numeric_std
<	signed, unsigned integer, natural	boolean	numeric_std
=	signed, unsigned integer, natural	boolean	numeric_std
>=	signed, unsigned integer, natural	boolean	numeric_std

UNIVERSITY OF TURKU

Libraries and Packages

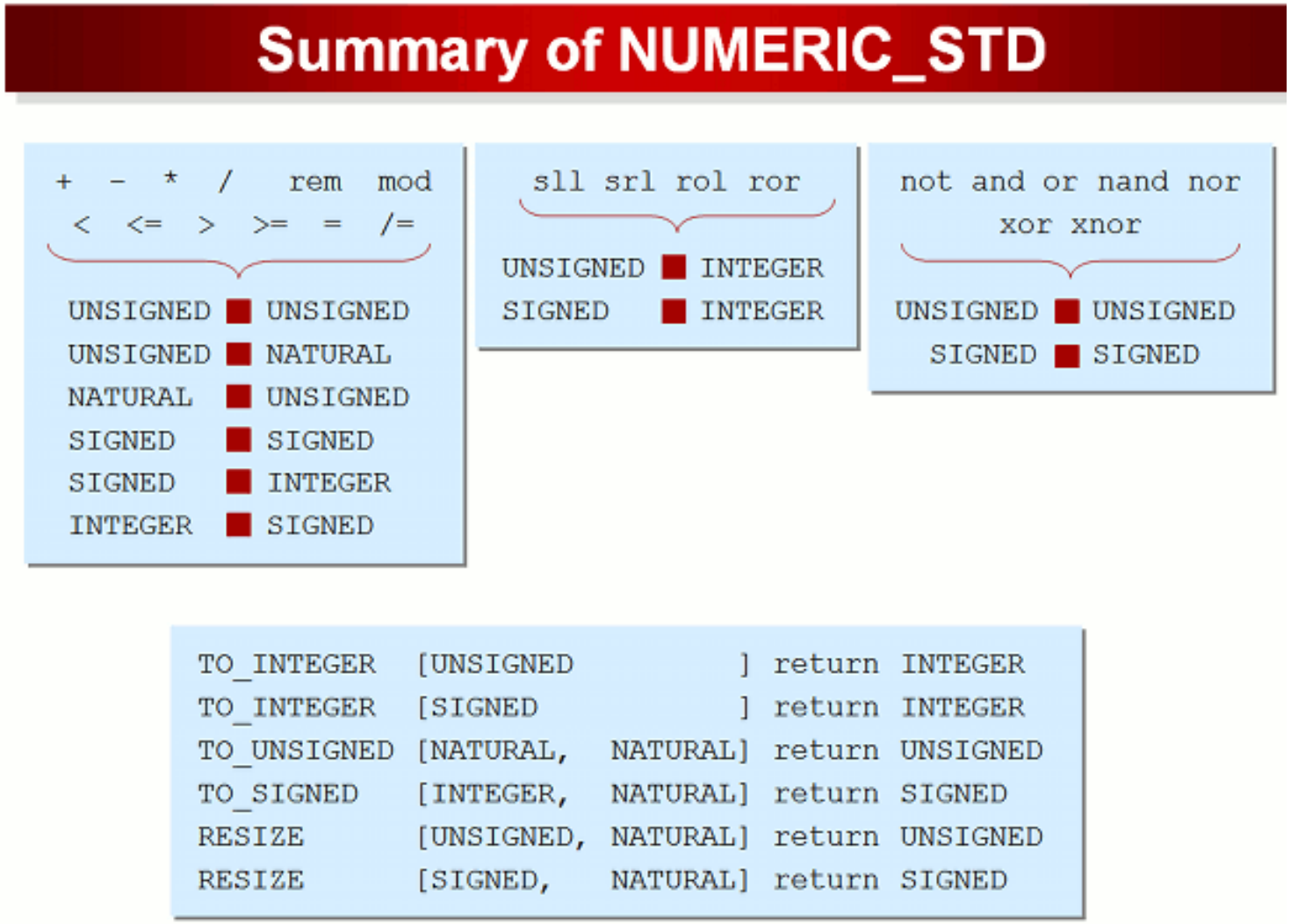
Function	Operand Types	Result Types	Package
<=	signed, unsigned integer, natural	boolean	numeric_std
/=	signed, unsigned integer, natural	boolean	numeric_std
>	signed, unsigned integer	boolean	std_logic_arith
<	signed, unsigned integer	boolean	std_logic_arith
=	signed, unsigned integer	boolean	std_logic_arith
>=	signed, unsigned integer	boolean	std_logic_arith
<=	signed, unsigned integer	boolean	std_logic_arith
/=	signed, unsigned integer	boolean	std_logic_arith
>	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
<	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
=	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
>=	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
<=	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
/=	std_logic_vector, integer	std_logic_vector	std_logic_unsigned

UNIVERSITY OF TURKU

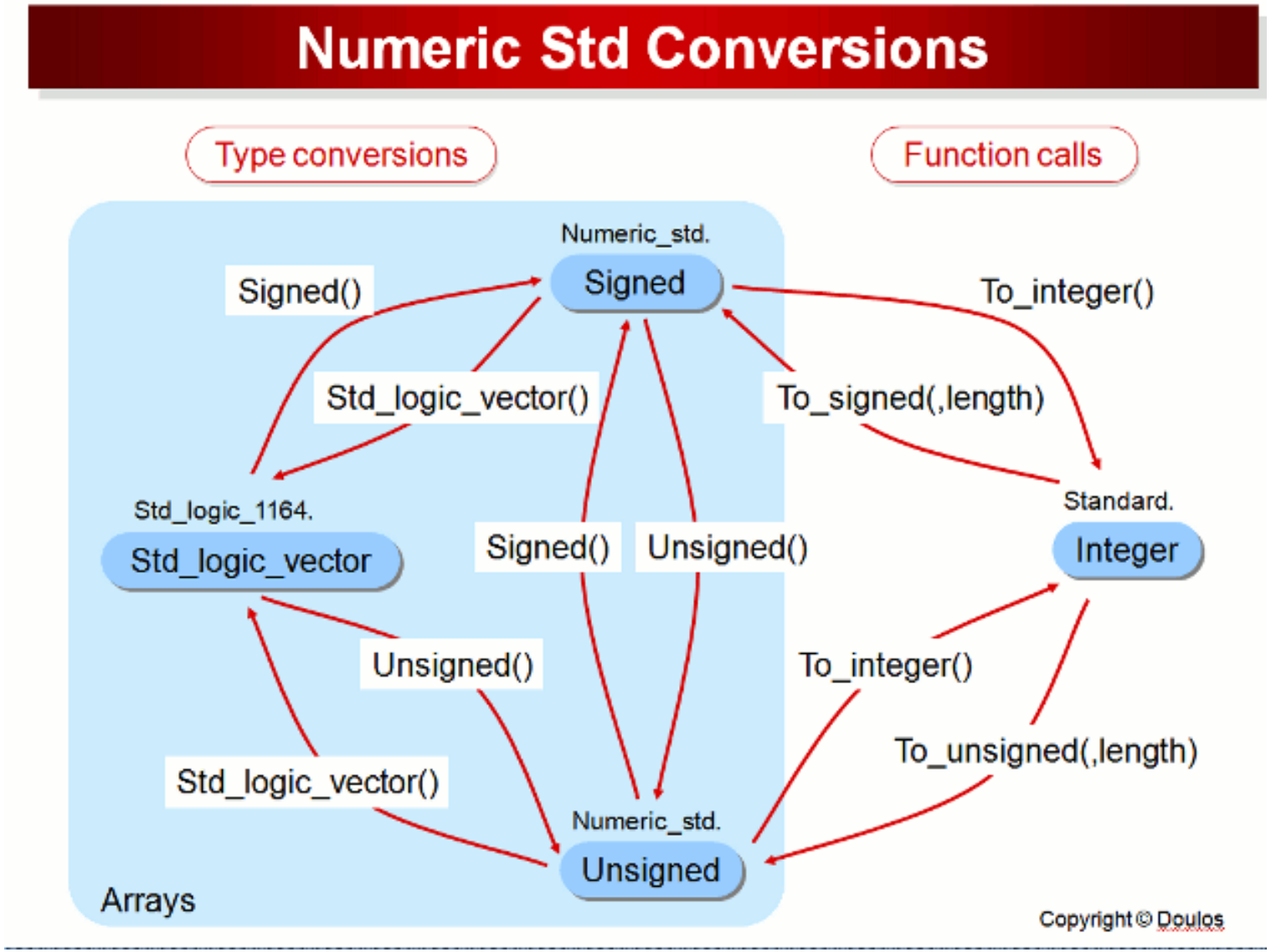
Libraries and Packages

Function	Operand Types	Result Types	Package
shift_left	signed, unsigned, natural	signed, unsigned	numeric_std
shift_right	signed, unsigned, natural	signed, unsigned	numeric_std
rotate_left	signed, unsigned, natural	signed, unsigned	numeric_std
rotate_right	signed, unsigned, natural	signed, unsigned	numeric_std
sll	signed, unsigned, integer	signed, unsigned	numeric_std
srl	signed, unsigned, integer	signed, unsigned	numeric_std
rol	signed, unsigned, integer	signed, unsigned	numeric_std
ror	signed, unsigned, integer	signed, unsigned	numeric_std
shl	signed, unsigned integer	signed, unsigned	std_logic_arith
shr	signed, unsigned integer	signed, unsigned	std_logic_arith
shl	std_logic_vector, integer	std_logic_vector	std_logic_unsigned
shr	std_logic_vector, integer	std_logic_vector	std_logic_unsigned

Numeric STD Summary



Numeric STD Conversions



UNIVERSITY OF TURKU

Some Conversions

- **Convert from Integer to Signed using Numeric_Std**

```
signal input_3  : integer;
signal output_3 : signed(3 downto 0);

output_3 <= to_signed(input_3, output_3'length);
```

- **Convert from Integer to Std_Logic_Vector using Numeric_Std**

```
signal input_1   : integer;
signal output_1a : std_logic_vector(3 downto 0);
signal output_1b : std_logic_vector(3 downto 0);

-- This line demonstrates how to convert positive integers
output_1a <= std_logic_vector(to_unsigned(input_1, output_1a'length));
 -- This line demonstrates how to convert positive or negative integers
output_1b <= std_logic_vector(to_signed(input_1, output_1b'length));
```

UNIVERSITY OF TURKU

Library and Package declarations - syntax

```
LIBRARY  library_name;
USE library_name.package_name.package_parts;
```

UNIVERSITY OF TURKU


UNIVERSITY
OF TURKU