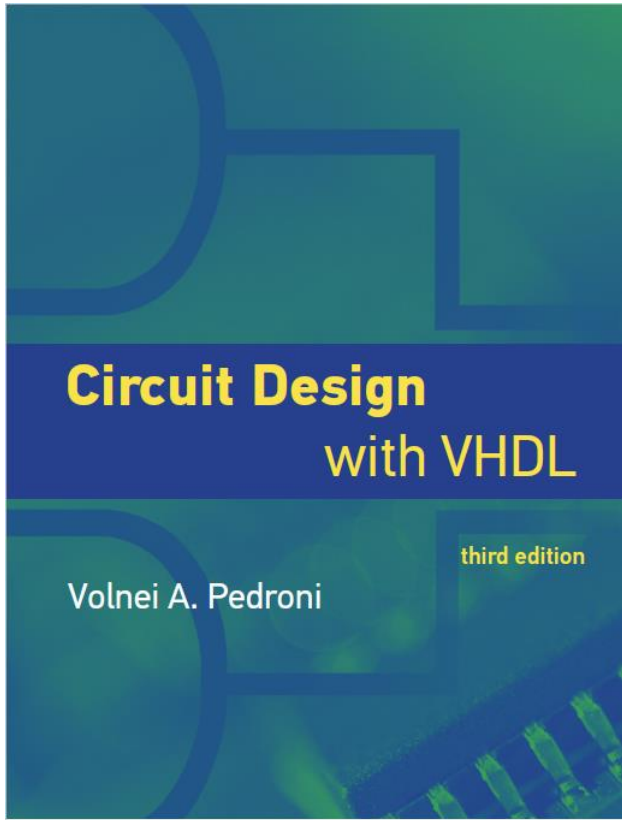


Circuit Design with VHDL

3rd Edition

Volnei A. Pedroni

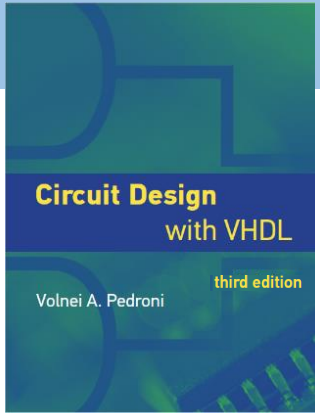
MIT Press, 2020

Slides Chapter 6

Code Structure and Composition

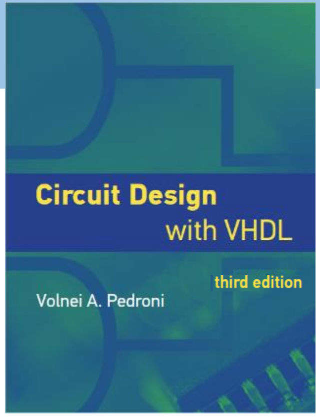
Revision 1

Book Contents



VHDL for Synthesis Slides

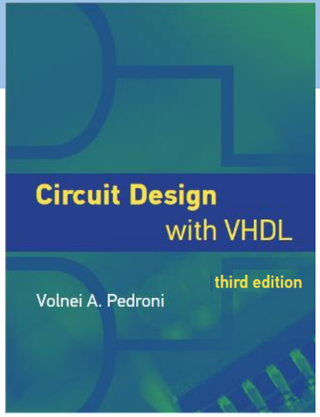
Chapter	Title
5	Introduction to VHDL
6	Code Structure and Composition
7	Predefined Data Types
8	User-Defined Data Types
9	Operators and Attributes
10	Concurrent Code
11	Concurrent Code – Practice
12	Sequential Code
13	Sequential Code – Practice
14	Packages and Subprograms



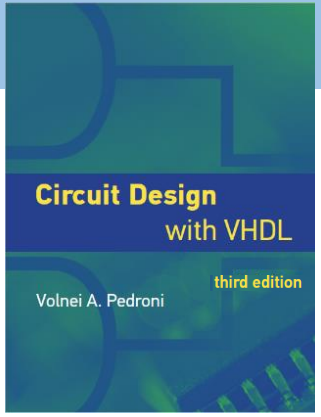
Chapter 6

Code Structure and Composition

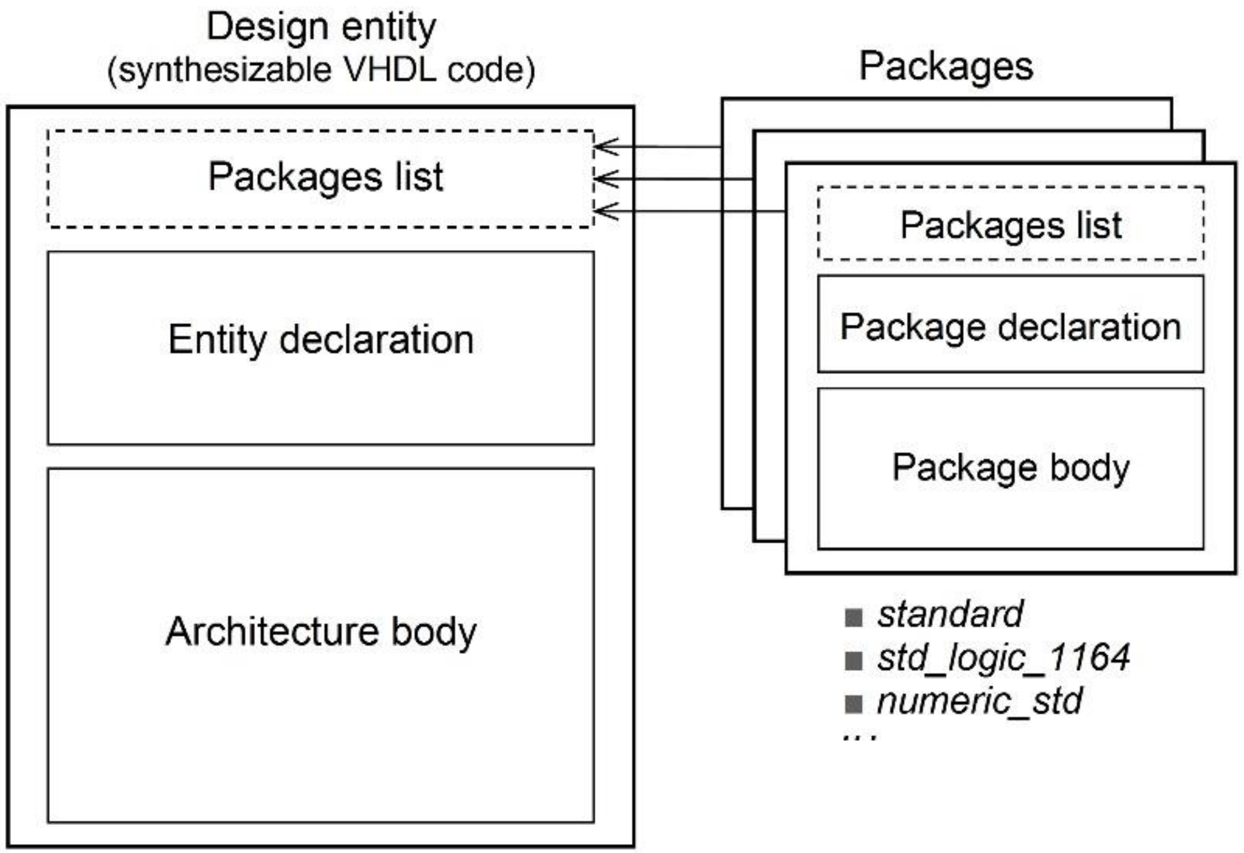
1. Structure of VHDL code
2. Official packages
3. Package declaration
4. Entity declaration
5. Architecture body
6. Object classes
7. Introductory examples

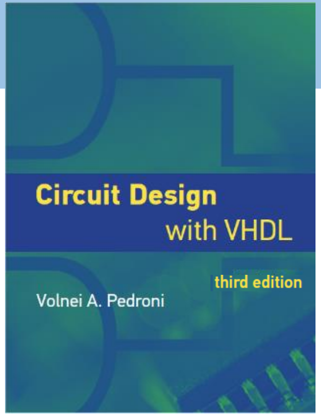


1. Structure of VHDL code

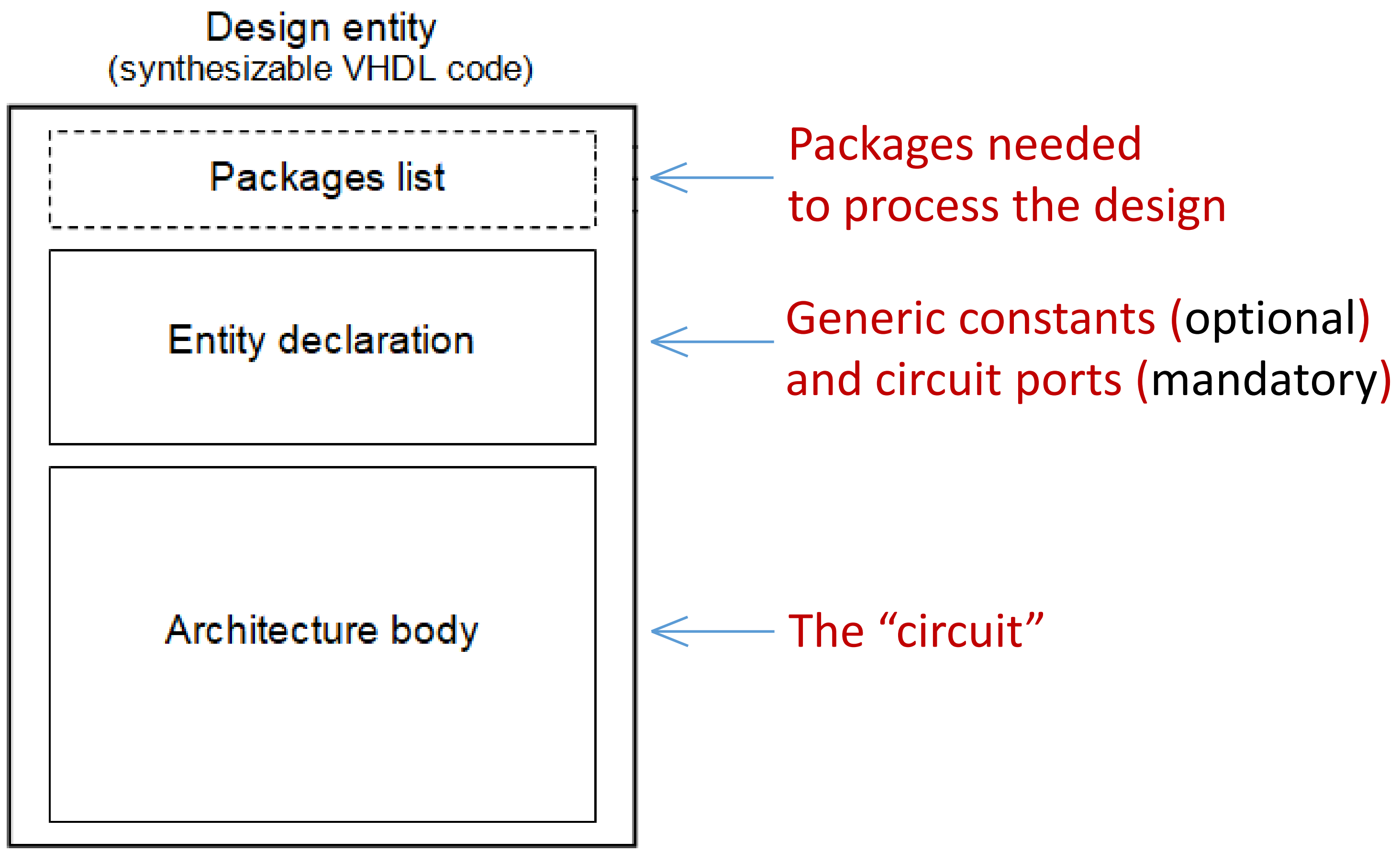


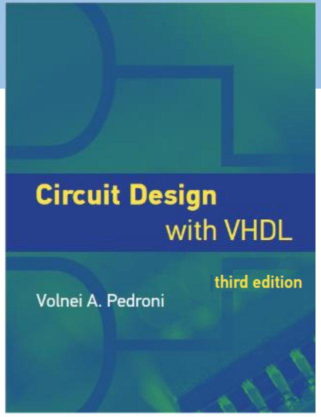
1. Structure of VHDL code





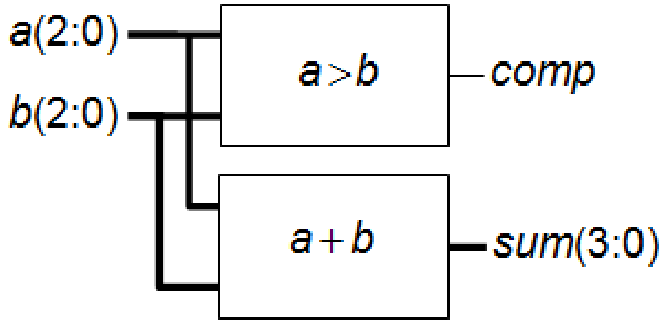
1. Structure of VHDL code

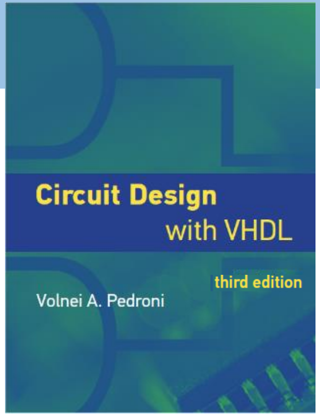

Design entity
(synthesizable VHDL code)
Packages list
Entity declaration
Architecture body
Packages needed to process the design
Generic constants (optional) and circuit ports (mandatory)
The "circuit"



1. Structure of VHDL code

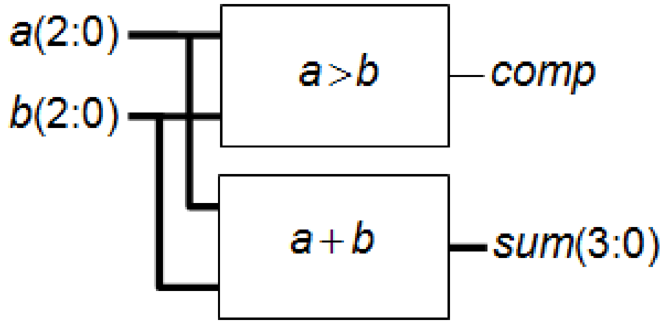
Example:





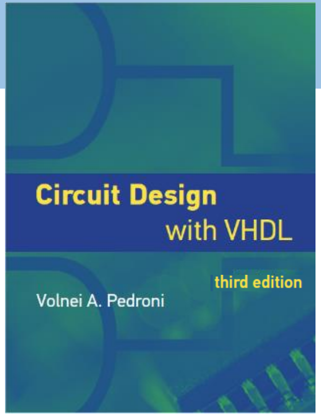
1. Structure of VHDL code

Example:



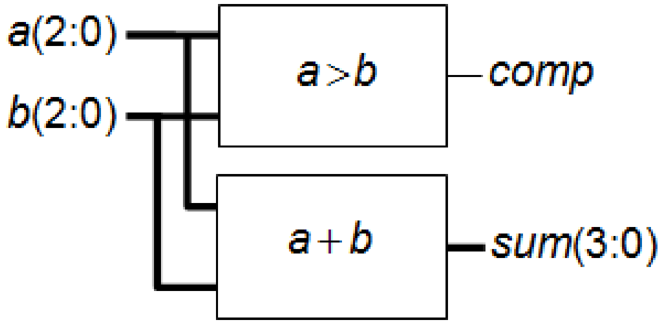
Solution 1:

```
-----------------------------------------
entity add_compare_cell is
  port (
    a, b: in integer range 0 to 7;
    comp: out bit;
    sum: out integer range 0 to 15);
end entity;

architecture dataflow of add_compare_cell is
begin
  comp <= '1' when a>b else '0';
  sum <= a + b;
end architecture;
-----------------------------------------
```



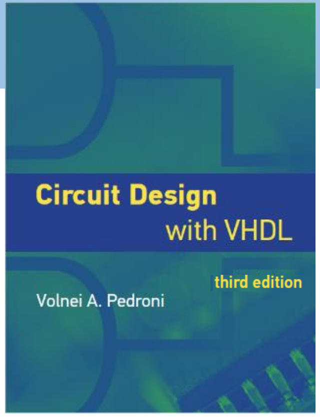
1. Structure of VHDL code

Example:



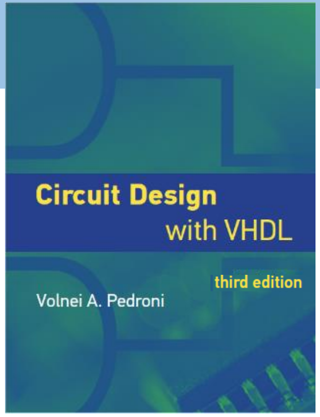
Solution 2:

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

entity add_compare_cell is
  port (
    a, b: in std_logic_vector(2 downto 0);
    comp: out std_logic;
    sum: out std_logic_vector(3 downto 0));
end entity;

architecture dataflow of add_compare_cell is
  signal a_uns, b_uns: unsigned(3 downto 0);
begin
  a_uns <= unsigned('0' & a);
  b_uns <= unsigned('0' & b);
  comp <= '1' when a_uns > b_uns else '0';
  sum <= std_logic_vector(a_uns + b_uns);
end architecture;
---------------------------------------------
```

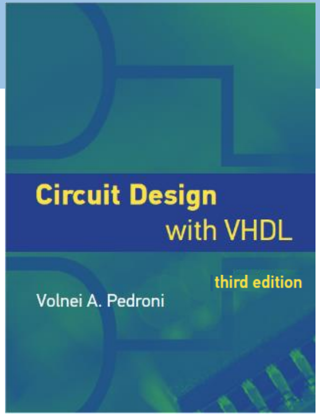


Chapter 6

Code Structure and Composition



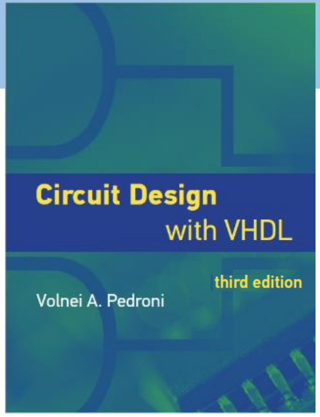
2. Official packages



2. Official packages

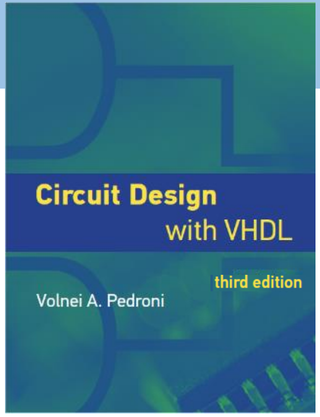
Table 6.1. VHDL libraries and packages.

Library	Package		Typical usage
std	1	*standard*	For implementing all sorts of logic and integer-based arithmetic circuits, but with major limitations; not recommended for arithmetic circuits and circuit ports
	2	*textio*	For dealing with text and files
	3	*env*	For communication with the simulation environment
ieee	4	*std_logic_1164*	For implementing any logic or arithmetic circuit (for the latter, must associate with another package, like #7, #10, or #13)
	5	*std_logic_textio*	Complement to package #2
	6	*numeric_bit*	For implementing integer arithmetic circuits with type unsigned or signed, having bit as base type
	7	*numeric_std*	Same as above, but with std_ulogic as base type
	8	*numeric_bit_unsigned*	For doing unsigned operations with type bit_vector
	9	*numeric_std_unsigned*	Same as above, for type std_ulogic_vector
	10	*fixed_pkg*	For implementing fixed-point arithmetic circuits
	11	*fixed_generic_pkg*	
	12	*fixed_float_types*	
	13	*float_pkg*	For implementing floating-point arithmetic circuits (associated with package #12)
	14	*float_generic_pkg*	
	15	*math_real*	For determining generic parameters (support not required for synthesis, but might exist for real values that are static)
	16	*math_complex*	

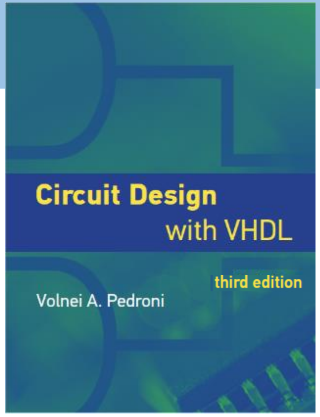


Chapter 6

Code Structure and Composition



3. Package declaration

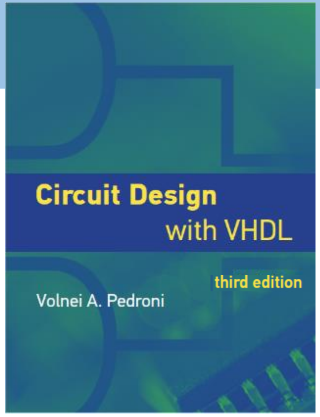


3. Package declaration

```
library library_name;
use library_name.package_name.all;
```

Included by default:

```
library std, work;
use std.standard.all;
```



3. Package declaration

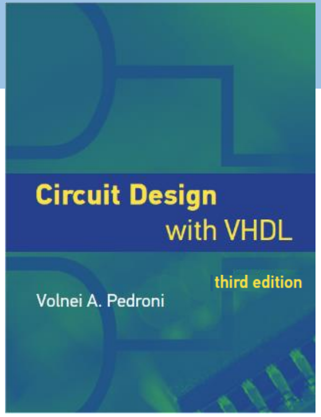
```
library library_name;
use library_name.package_name.all;
```

Included by default:

```
library std, work;
use std.standard.all;
```

Examples of needed (non-default) declarations:

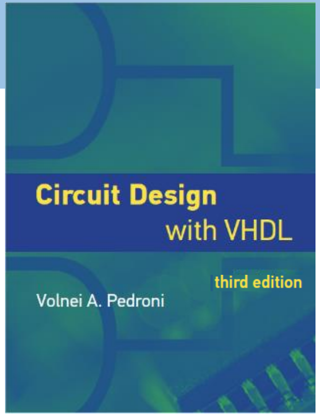
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.functions_pkg.all;
use work.procedures_pkg.sort;
```



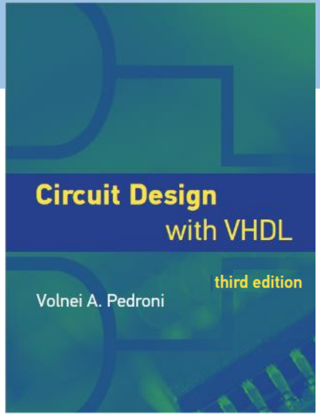
Chapter 6

Code Structure and Composition

1. Structure of VHDL code
2. Official packages
3. Package declaration
4. Entity declaration
5. Architecture body
6. Object classes
7. Introductory examples



4. Entity declaration

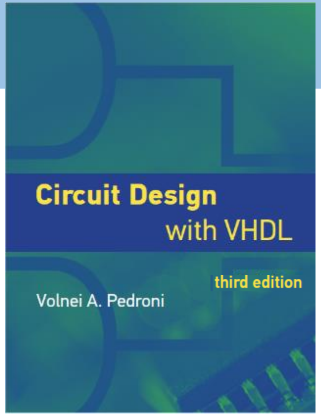


4. Entity declaration

```
entity entity_name is

   [generic (
      constant_name: constant_type [:= constant_value];
      constant_name: constant_type [:= constant_value];
      ...);]

   port (
      port_name: port_mode port_type;
      port_name: port_mode port_type;
      ...);

end [entity] [entity_name];
```

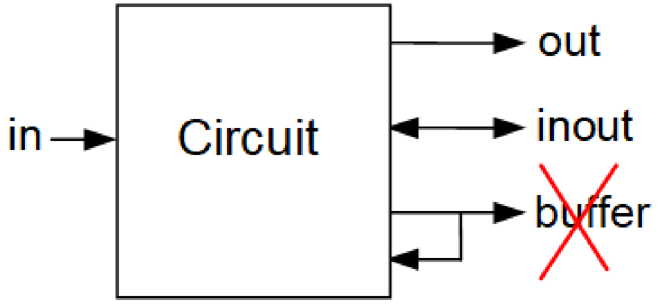
Generic constants (optional)

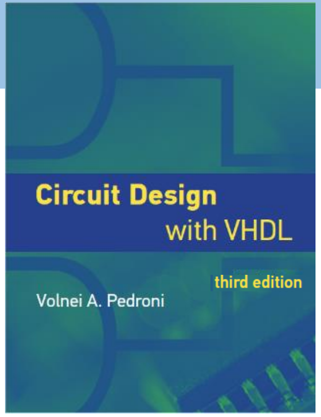
Circuit ports (mandatory for synthesis)



4. Entity declaration

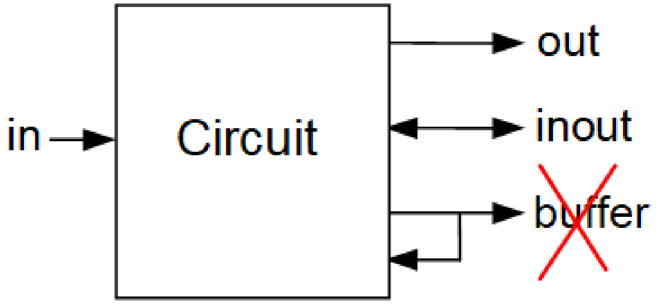
Port modes: *in*, *out*, *inout*



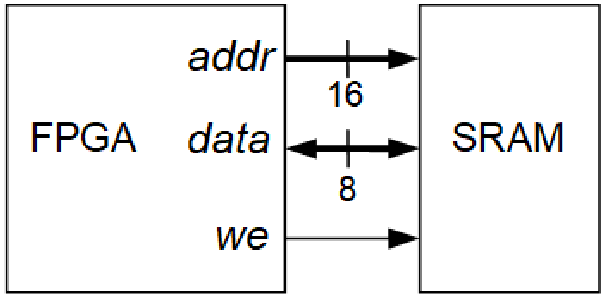


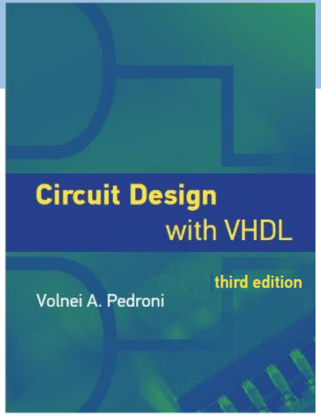
4. Entity declaration

Port modes: *in*, *out*, *inout*



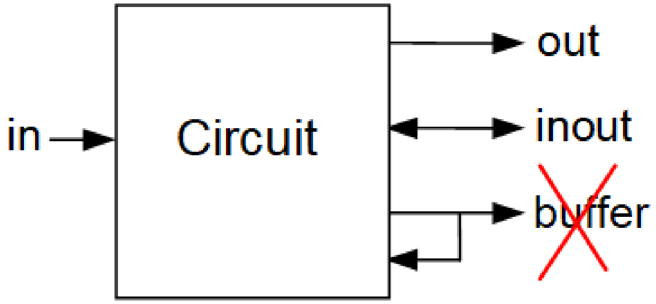
Example:



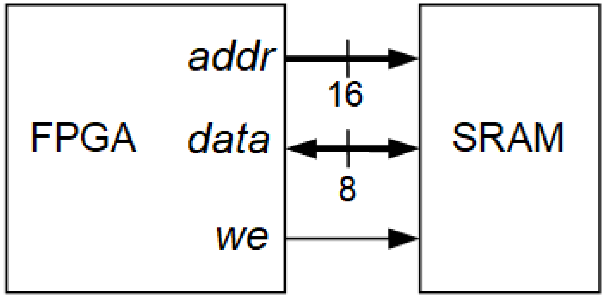


4. Entity declaration

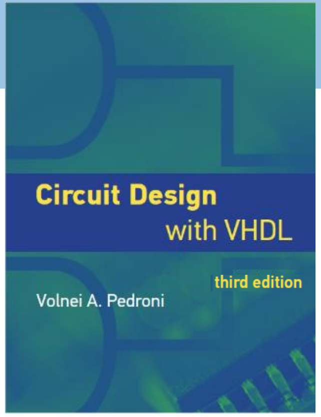
Port modes: *in*, *out*, *inout*



Example:

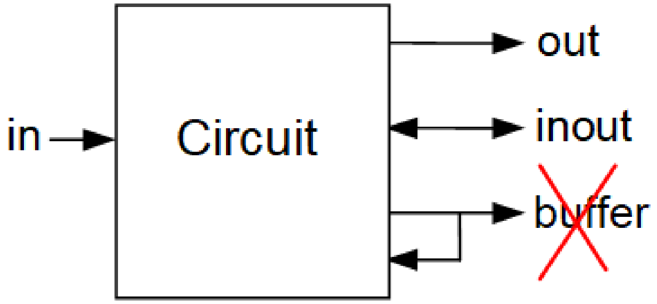


```
entity SRAM_interface is
   port (
      we: out std_logic;
      addr: out std_logic_vector(15 downto 0);
      data: inout std_logic_vector(7 downto 0));
 end entity;
```

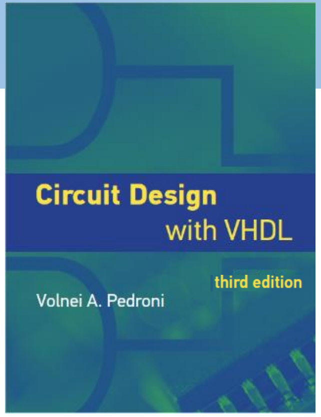


4. Entity declaration

Port modes: *in*, *out*, *inout*

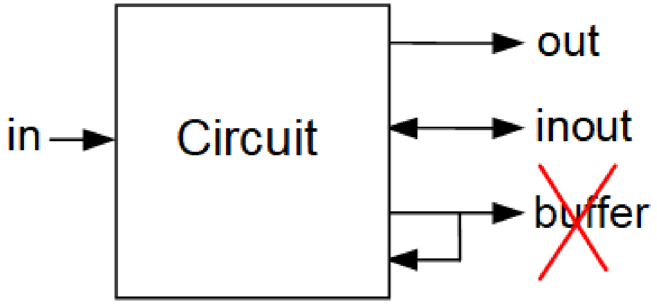


How is *inout* constructed?

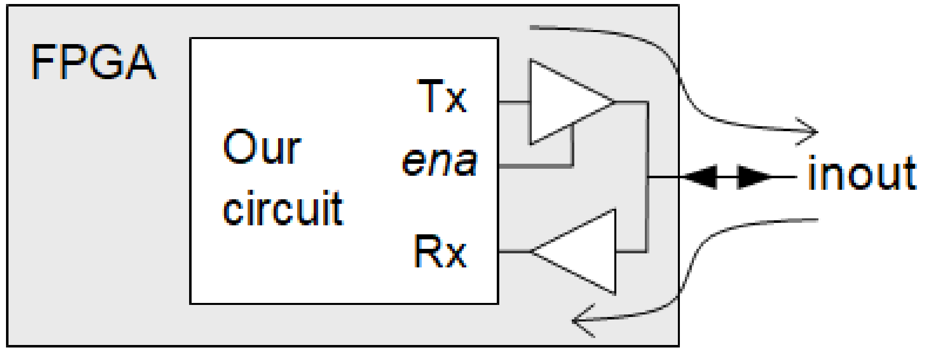


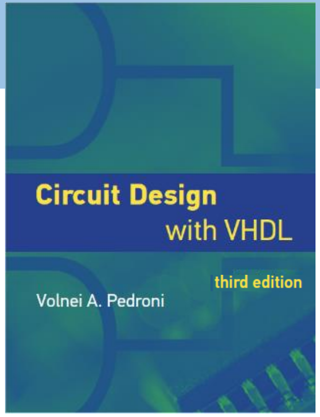
4. Entity declaration

Port modes: *in*, *out*, *inout*



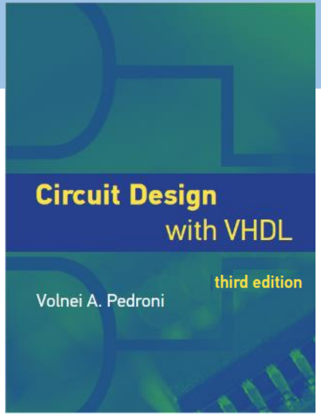
How is *inout* constructed?





4. Entity declaration

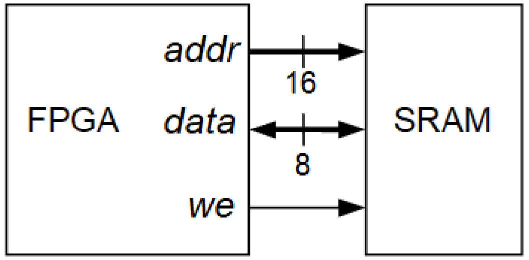
Generic constants

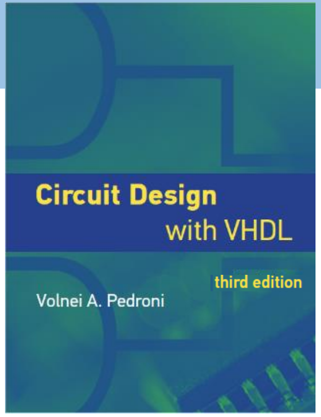


4. Entity declaration

Generic constants

Example:

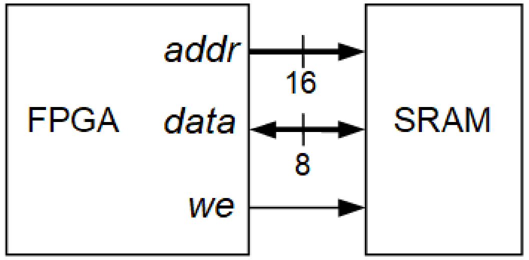




4. Entity declaration

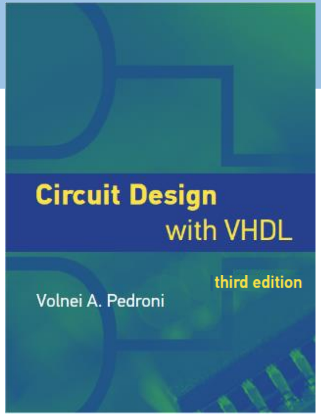
Generic constants

Example:



Without generics:

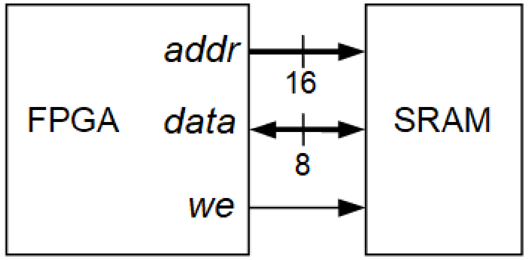
```
entity SRAM_interface is
   port (
      we: out std_logic;
      addr: out std_logic_vector(15 downto 0);
      data: inout std_logic_vector(7 downto 0));
 end entity;
```



4. Entity declaration

Generic constants

Example:

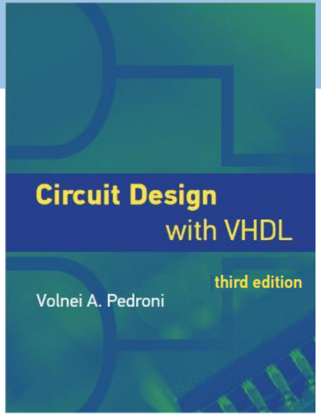


Without generics:

```
entity SRAM_interface is
   port (
      we: out std_logic;
      addr: out std_logic_vector(15 downto 0);
      data: inout std_logic_vector(7 downto 0));
 end entity;
```

With generics:

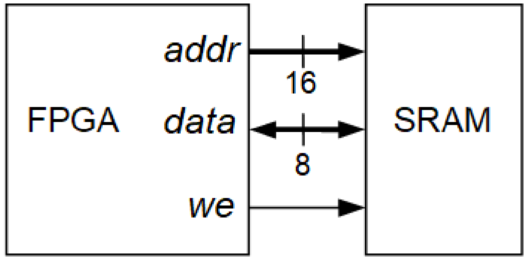
```
entity SRAM_interface is
   generic (
      ADDR_WIDTH: natural := 16;
      DATA_WIDTH: natural := 8);
   port (
      we: out std_logic;
      addr: out std_logic_vector(...???...);
      data: inout std_logic_vector(...???...));
 end entity;
```



4. Entity declaration

Generic constants

Example:

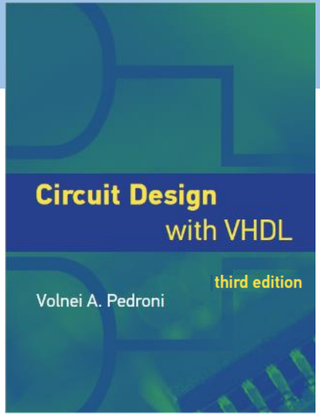


Without generics:

```vhdl
entity SRAM_interface is
   port (
      we: out std_logic;
      addr: out std_logic_vector(15 downto 0);
      data: inout std_logic_vector(7 downto 0));
 end entity;
```

With generics:

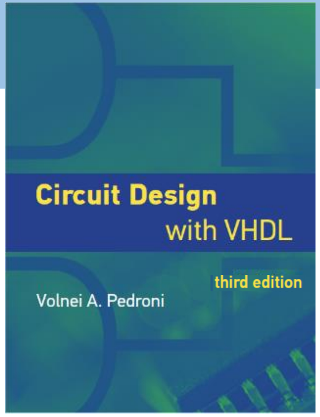
```vhdl
entity SRAM_interface is
   generic (
      ADDR_WIDTH: natural := 16;
      DATA_WIDTH: natural := 8);
   port (
      we: out std_logic;
      addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
      data: inout std_logic_vector(DATA_WIDTH-1 downto 0));
 end entity;
```



4. Entity declaration

Generic constants

VHDL-2008:
- Expressions allowed in generic declarations (helpful)
- Type declarations allowed but require instantiation (not helpful in general)



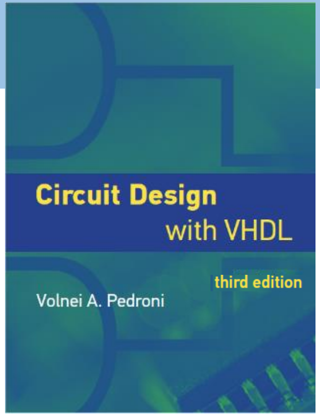
4. Entity declaration

Generic constants

VHDL-2008:
- Expressions allowed in generic declarations (helpful)
- Type declarations allowed but require instantiation (not helpful in general)

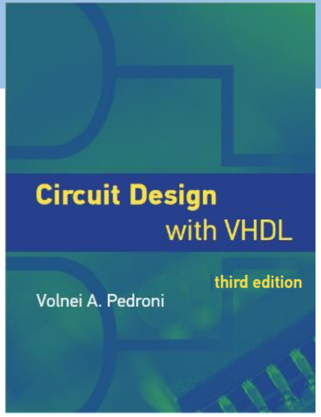
Example:

```
generic (
   NUM_BITS: natural := 16;
   DEPTH: natural := 2**NUM_BITS
);
```

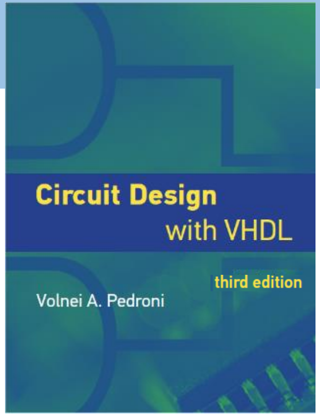


Chapter 6

Code Structure and Composition



5. Architecture body

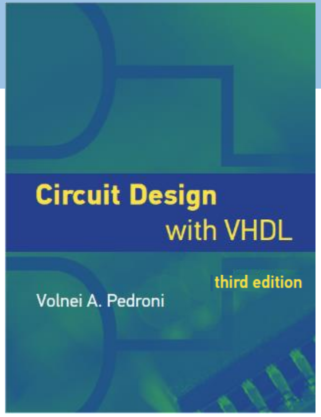


5. Architecture body

```
architecture architecture_name of entity_name is
   [architecture_declarative_part]
begin
   architecture_statement_part
end [architecture] [architecture_name];
```

Declarative region (optional)

Statements region (mandatory)



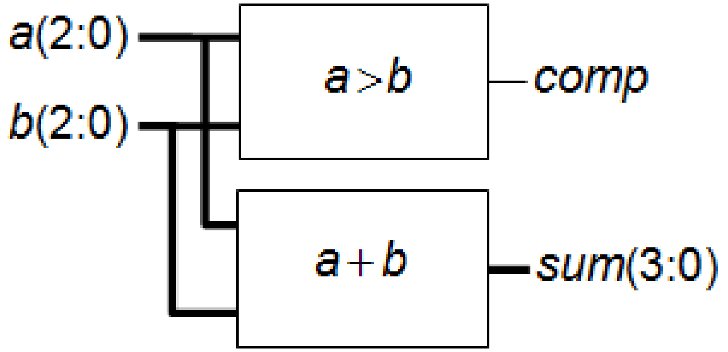
5. Architecture body

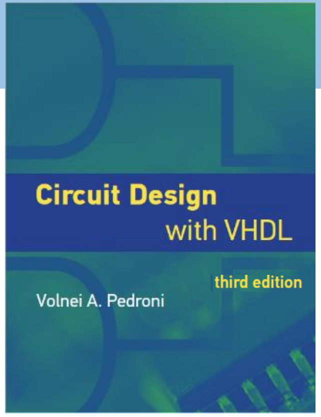
```
architecture architecture_name of entity_name is
   [architecture_declarative_part]
begin
   architecture_statement_part
end [architecture] [architecture_name];
```

← Declarative region (optional)

← Statements region (mandatory)

Example (seen earlier):





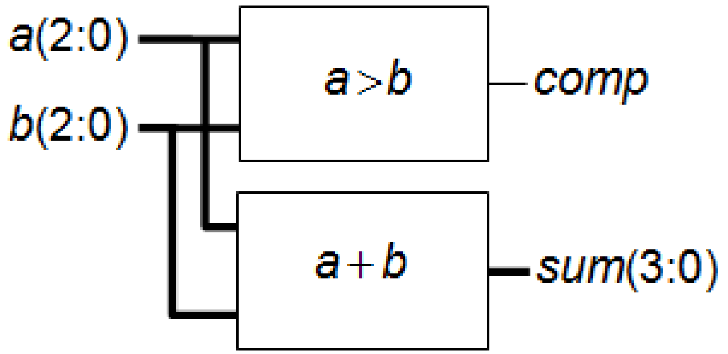
5. Architecture body

```
architecture architecture_name of entity_name is
   [architecture_declarative_part]
begin
   architecture_statement_part
end [architecture] [architecture_name];
```

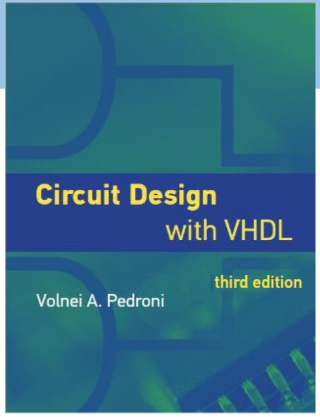
Declarative region (optional)

Statements region (mandatory)

Example (seen earlier):

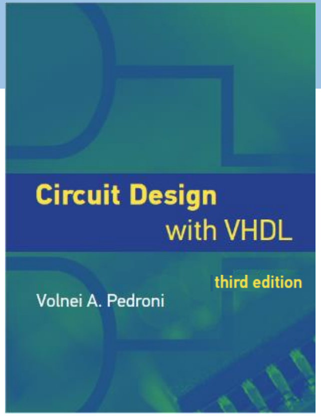


```
---------------------------------------------
architecture dataflow of add_compare_cell is
begin
  comp <= '1' when a>b else '0';
  sum <= a + b;
end architecture;
---------------------------------------------
```

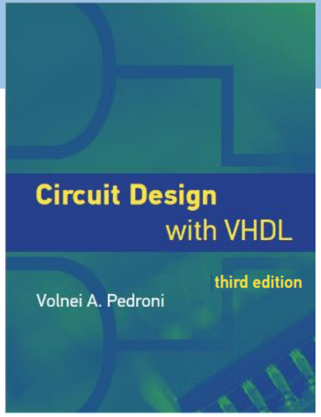


Chapter 6

Code Structure and Composition



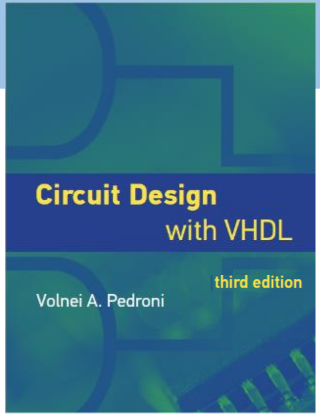
6. Object classes



6. Object classes

- Object: a named item that has a value of a type

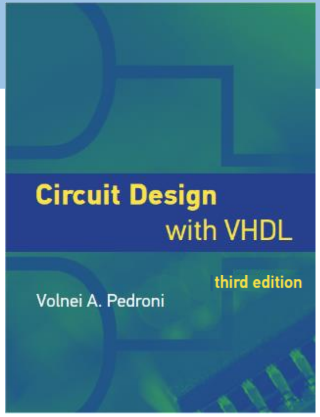
- There are four kinds of objects (or object classes) in VHDL:
  - Constant
  - Signal
  - Variable
  - File (for simulation)



6. Object classes

Constant:

```
constant constant_name: constant_type := constant_value;
```



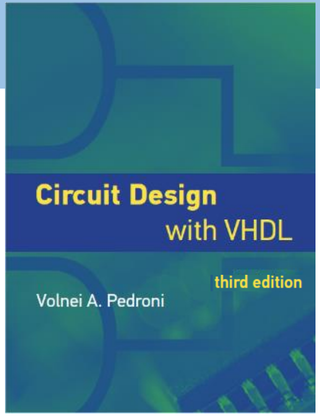
6. Object classes

Constant:

```
constant constant_name: constant_type := constant_value;
```

Examples:

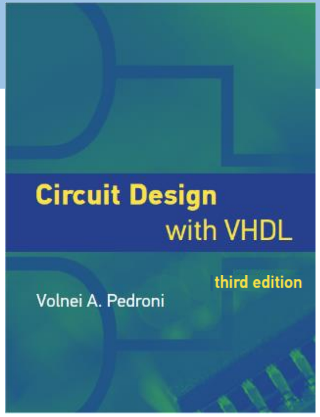
```
constant CLK_FREQ_MHZ: natural := 150;

constant MASK: std_logic_vector(7 downto 0) := "00001111";

constant BIT_MATRIX: bit_matrix_type := (('1','1','0'),
                                          ('0','1','1'),
                                          ('1','0','0'));
```



6. Object classes

Signal:

```
signal signal_name: signal_type [:= default_value];
```



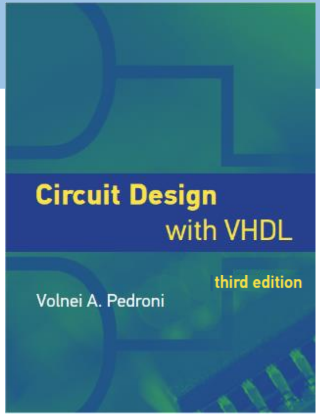
6. Object classes

Signal:

```
signal signal_name: signal_type [:= default_value];
```

Examples:

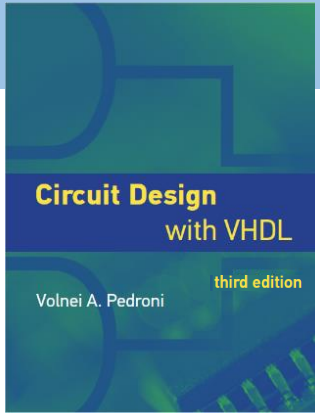
```
--Signal declarations:
signal rst, ena: std_logic;
signal count: natural range 0 to 255;

--Signal assignments:
if rst then reg <= (others => '0');
outp <= count when ena else 0;
```



6. Object classes

Variable:

```
variable variable_name: variable_type [:= initial_value];
```



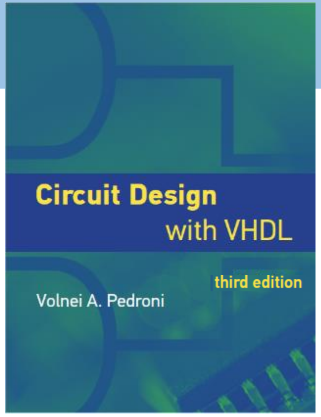
6. Object classes

Variable:

```
variable variable_name: variable_type [:= initial_value];
```

Example:

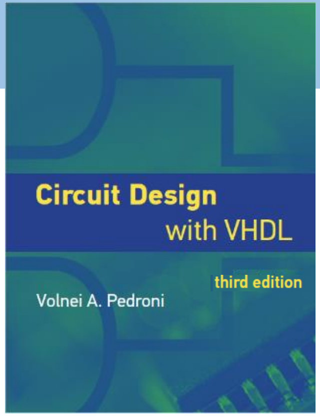
```
variable count: natural range 0 to 2**DEPTH–1;
...
if rising_edge(clk) then
   count := count + 1;
end if;
```



6. Object classes

Additional notes:

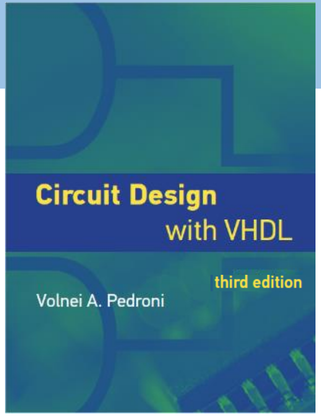
- Shared variables:
- Signal x Variable:
- Initial/default values:



6. Object classes

Additional notes:

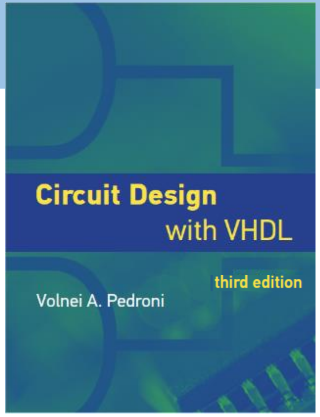
- Shared variables: Avoid them for synthesis (and even for simulation)
- Signal x Variable:
- Initial/default values:



6. Object classes

Additional notes:

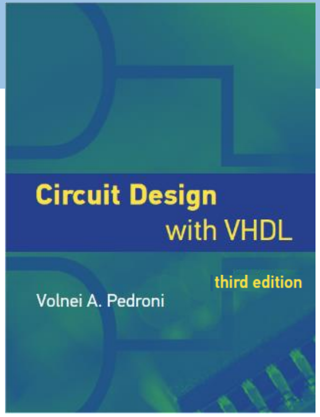
- Shared variables: Avoid them for synthesis (and even for simulation)
- Signal x Variable: Will be seen in chapter 12
- Initial/default values:



6. Object classes

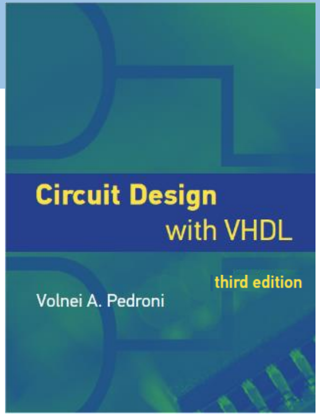
Additional notes:

- Shared variables: Avoid them for synthesis (and even for simulation)
- Signal x Variable: Will be seen in chapter 12
- Initial/default values: Why are they not synthesizable? (see sec. 6.6.2)



Chapter 6

Code Structure and Composition



7. Introductory examples

Example 6.1. Parity detector
Example 6.2. D-type flip-flop (DFF)
Example 6.3. Registered add-and-compare circuit

7. Introductory examples

Example 6.1. Parity detector

Example 6.2. D-type flip-flop (DFF) ⟶

Example 6.3. Registered add-and-com

```vhdl
------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity flip_flop is
   port (
      d, clk, rst: in std_logic;
      q: out std_logic);
end entity;

architecture flip_flop of flip_flop is
begin
   process (clk, rst)
   begin
      if rst then
         q <= '0';
      elsif rising_edge(clk) then
         q <= d;
      end if;
   end process;
end architecture;
------------------------------------------
```

End of Chapter 6