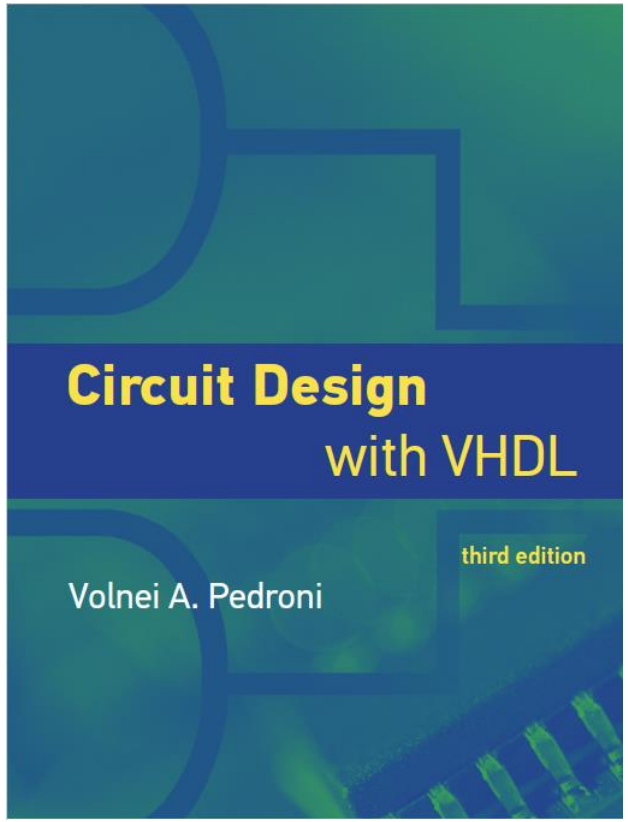


Circuit Design with VHDL

3rd Edition

Volnei A. Pedroni

MIT Press, 2020

Slides Chapter 5

Introduction to VHDL

Revision 2

Book Contents

Part I: Digital Circuits Review

1. Review of Combinational Circuits
2. Review of Sequential Circuits
3. Review of State Machines
4. Review of FPGAs

Part II: VHDL

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
15. The Case of State Machines
16. The Case of State Machines – Practice
17. Additional Design Examples
18. Intr. to Simulation with Testbenches

Appendices

A. Vivado Tutorial
B. Quartus Prime Tutorial
C. ModelSim Tutorial
D. Simulation Analysis and Recommendations
E. Using Seven-Segment Displays with VHDL
F. Serial Peripheral Interface
G. I2C (Inter Integrated Circuits) Interface
H. Alphanumeric LCD
I. VGA Video Interface
J. DVI Video Interface
K. TMDS Link
L. Using Phase-Locked Loops with VHDL
M. List of Enumerated Examples and Exercises

Book Contents

Part I: Digital Circuits Review

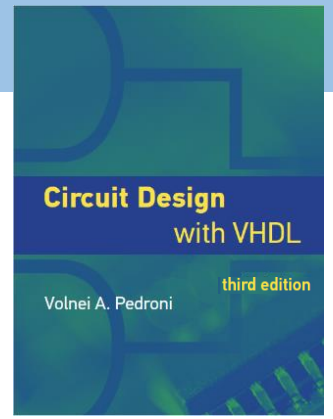
1. Review of Combinational Circuits
2. Review of Sequential Circuits
3. Review of State Machines
4. Review of FPGAs

Part II: VHDL

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
15. The Case of State Machines
16. The Case of State Machines – Practice
17. Additional Design Examples
18. Intr. to Simulation with Testbenches

Appendices

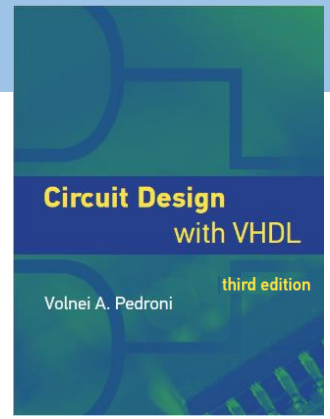
A. Vivado Tutorial
B. Quartus Prime Tutorial
C. ModelSim Tutorial
D. Simulation Analysis and Recommendations
E. Using Seven-Segment Displays with VHDL
F. Serial Peripheral Interface
G. I2C (Inter Integrated Circuits) Interface
H. Alphanumeric LCD
I. VGA Video Interface
J. DVI Video Interface
K. TMDS Link
L. Using Phase-Locked Loops with VHDL
M. List of Enumerated Examples and Exercises



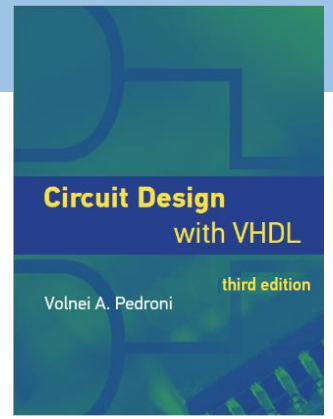
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design



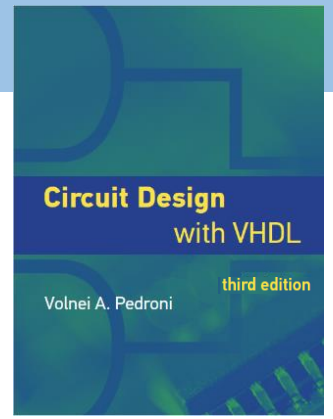
1. Versions and purposes



1. Versions and purposes

VHDL = VHSIC Hardware Description Language

(VHSIC = Very High Speed ICs, USA Dept. of Defense, 1980s)



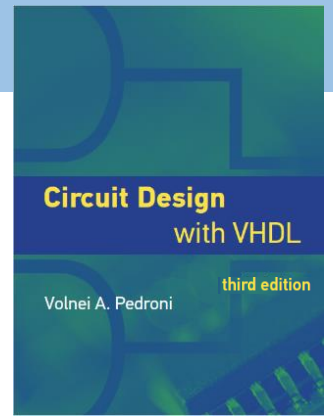
1. Versions and purposes

VHDL = VHSIC Hardware Description Language

(VHSIC = Very High Speed ICs, USA Dept. of Defense, 1980s)

Versions:

- VHDL-87
- VHDL-93
- (VHDL-2000)
- VHDL-2002
- VHDL-2008



1. Versions and purposes

VHDL = VHSIC Hardware Description Language

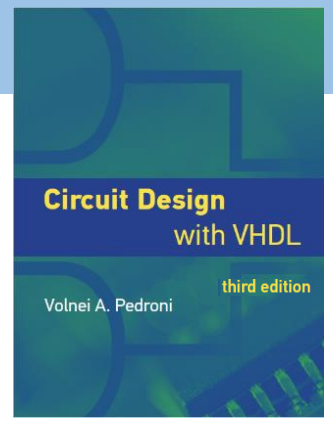
(VHSIC = Very High Speed ICs, USA Dept. of Defense, 1980s)

Versions:

- VHDL-87
- VHDL-93
- (VHDL-2000)
- VHDL-2002
- VHDL-2008

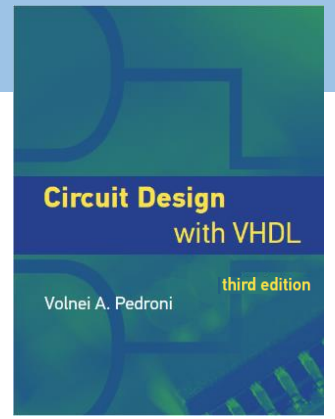
Purposes:

- FPGA programming
- Digital ASIC design/fabrication (masks generation)



1. Versions and purposes

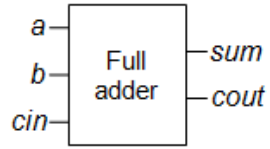
General concept



1. Versions and purposes

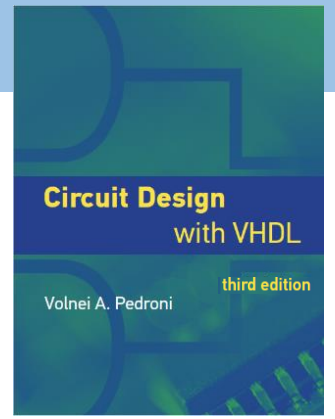
General concept

Desired functionality



a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

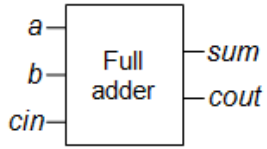
Output	Equation
sum	$a \oplus b \oplus cin$
cout	$a \cdot b + a \cdot cin + b \cdot cin$



1. Versions and purposes

General concept

Desired functionality →

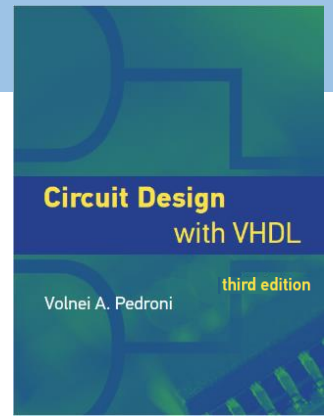


a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Output	Equation
sum	$a \oplus b \oplus cin$
cout	$a \cdot b + a \cdot cin + b \cdot cin$

VHDL code →

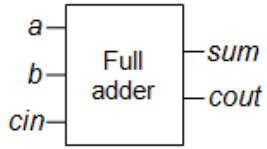
```
entity full_adder_unit is
  port (
    a, b, cin: in bit;
    sum, cout: out bit);
end entity full_adder_unit;
-------------------------------------------------
architecture boolean_equations of full_adder_unit is
begin
  sum <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end architecture boolean_equations;
```



1. Versions and purposes

General concept

Desired functionality →



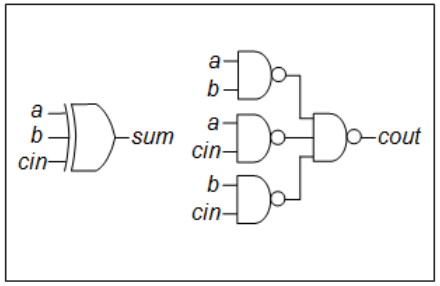
a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

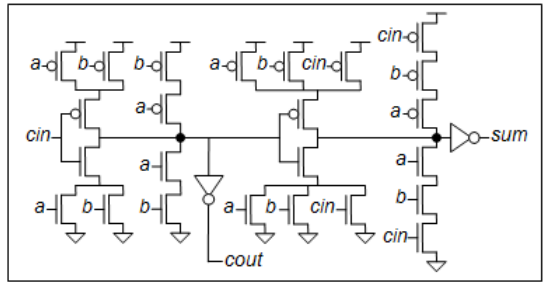
Output	Equation
sum	$a \oplus b \oplus cin$
cout	$a \cdot b + a \cdot cin + b \cdot cin$

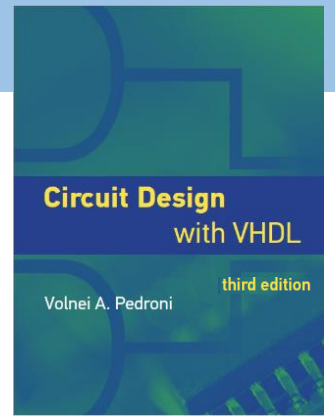
VHDL code →

```
entity full_adder_unit is
  port (
    a, b, cin: in bit;
    sum, cout: out bit);
end entity full_adder_unit;
-------------------------------------------------
architecture boolean_equations of full_adder_unit is
begin
  sum <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end architecture boolean_equations;
```

Compliant circuit →



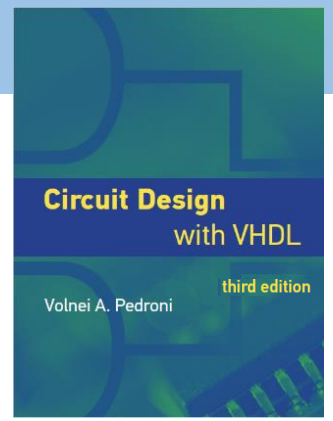




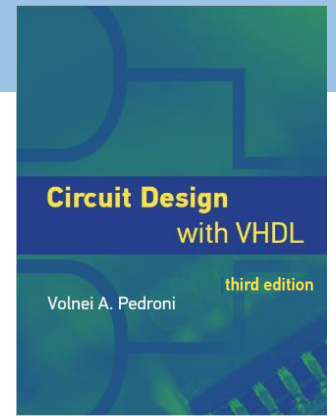
Chapter 5

Introduction to VHDL

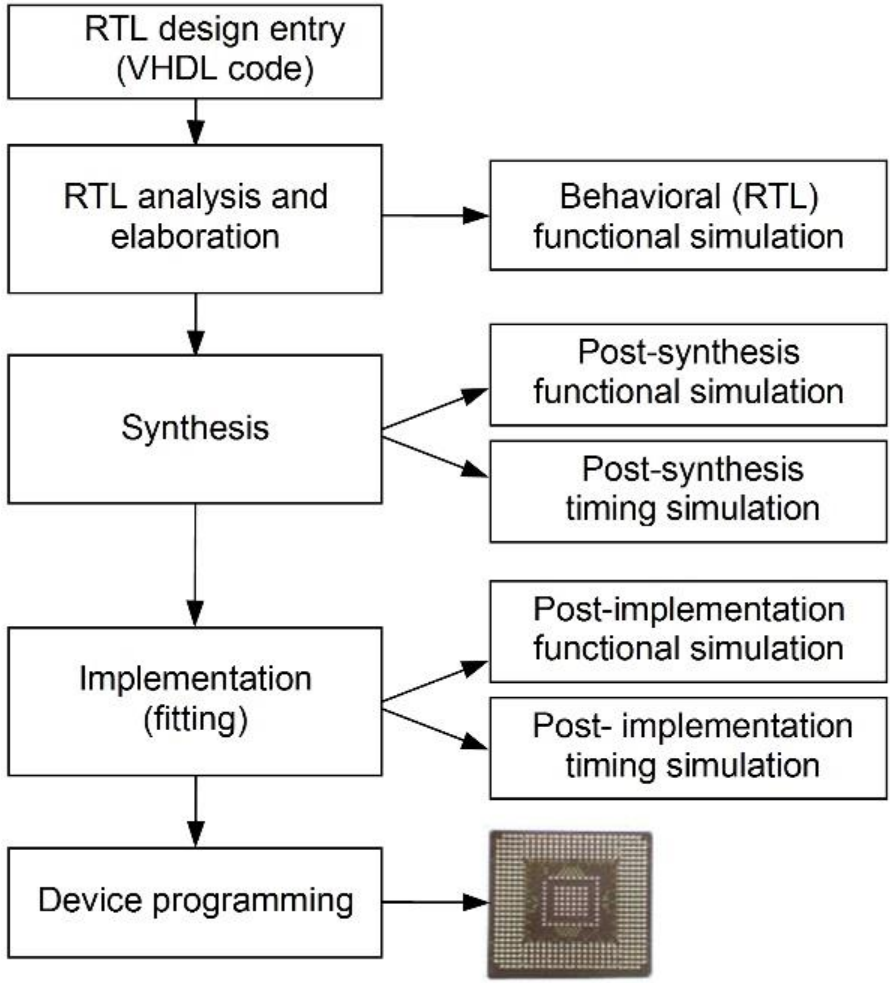
1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design

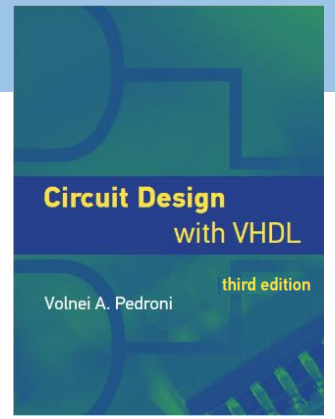


2. Simplified design flow



2. Simplified design flow

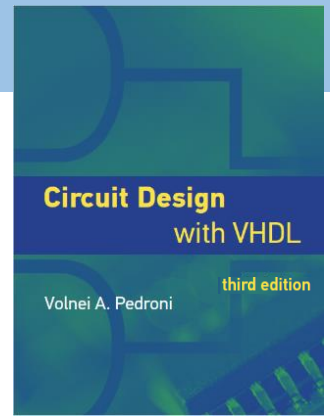




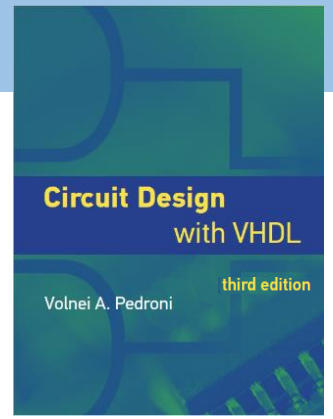
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design

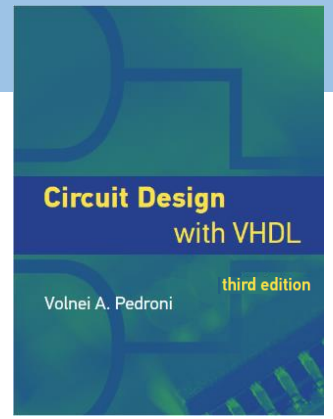


3. Simulation types



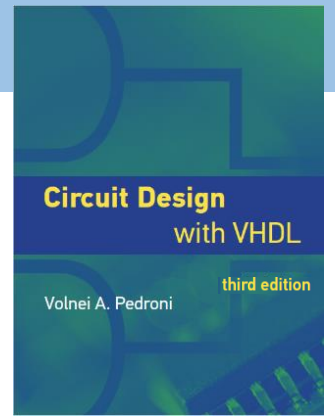
3. Simulation types

- Functional simulation: ?
- Timing simulation: ?



3. Simulation types

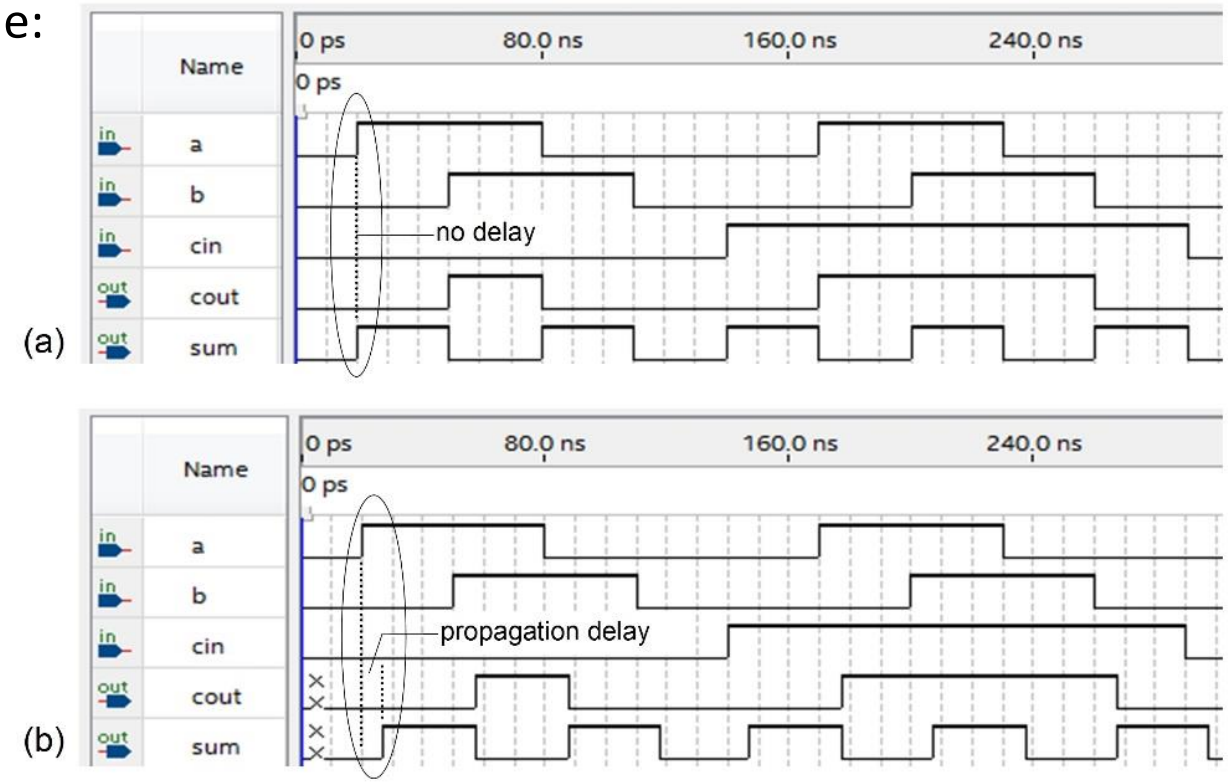
- Functional simulation: Logic behavior only
- Timing simulation: Propagation delays included

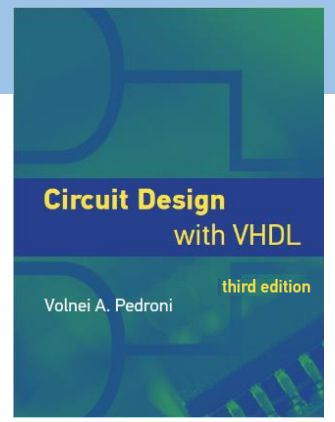


3. Simulation types

- Functional simulation: Logic behavior only
- Timing simulation: Propagation delays included

Example:

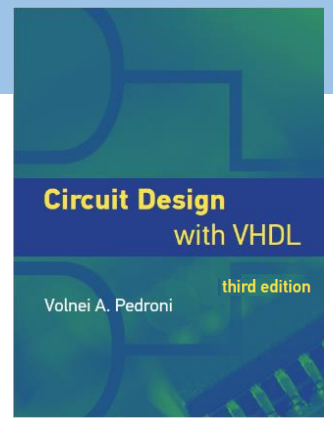




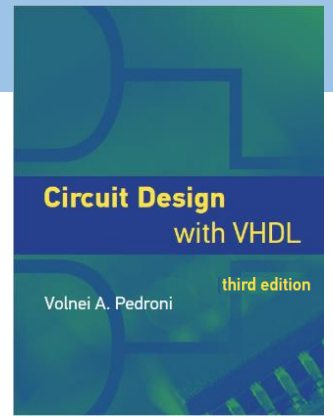
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design

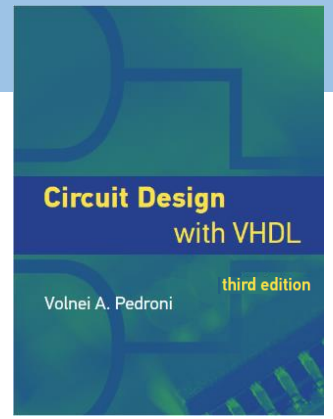


4. Concurrent versus sequential statements



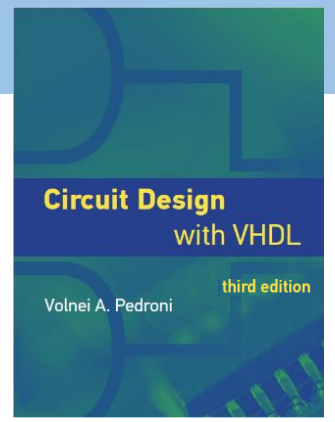
4. Concurrent versus sequential statements

- Concurrency is a major feature of VHDL (or any other HDL)
- So VHDL is a code, not a program
- And it has statements, not “instructions”



4. Concurrent versus sequential statements

- Concurrency is a major feature of VHDL (or any other HDL)
- So VHDL is a code, not a program
- And it has statements, not “instructions”
- Only code placed inside a process (or subprogram) is interpreted as in a program
- But a process, as a whole, is still concurrent with respect to all other statements



4. Concurrent versus sequential statements

- Concurrency is a major feature of VHDL (or any other HDL)
- So VHDL is a code, not a program
- And it has statements, not "instructions"
- Only code placed inside a process (or subprogram) is interpreted as in a program
- But a process, as a whole, is still concurrent with respect to all other statements

Examples:

These 3 pairs of statements are equivalent:

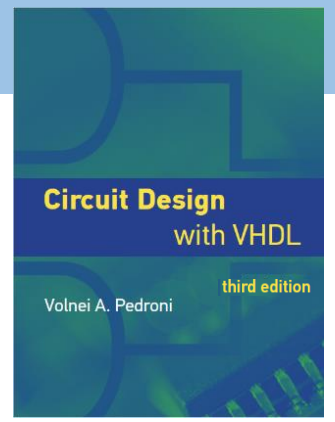
```
total <= a1 + a2*a3;
flag <= '1' when total > LIMIT else '0';

flag <= '1' when total > LIMIT else '0';
total <= a1 + a2*a3;

total <= a1 + a2*a3;
flag <= '1' when a1 + a2*a3 > LIMIT else '0';
```

Illegal:

```
sum <= 0;
...
sum <= a + b;
```

OK (last value prevails):

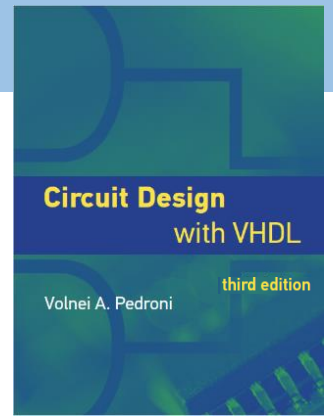
```
process (all)
begin
    sum <= 0;
    ...
    sum <= a + b;
end process;
```



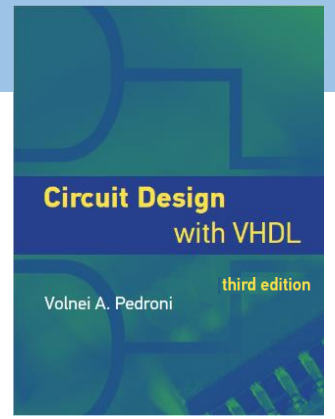
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design



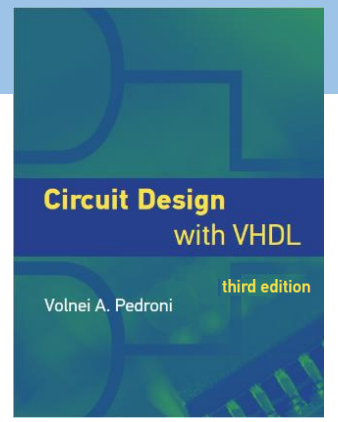
5. A special data type: *std_ulogic*



5. A special data type: *std_ulogic*

Before (1st VHDL version):

Type bit	
'0'	Logic 0
'1'	Logic 1



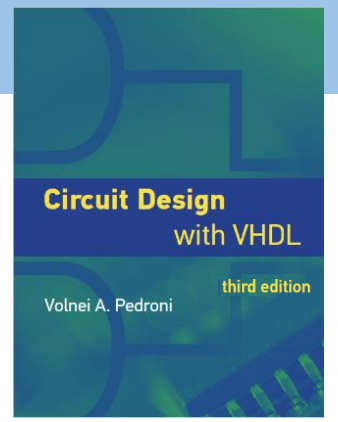
5. A special data type: *std_ulogic*

Before (1st VHDL version):

Type bit	
'0'	Logic 0
'1'	Logic 1

After (2nd VHDL version on):

Type std_ulogic	
'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



5. A special data type: *std_ulogic*

Before (1st VHDL version):

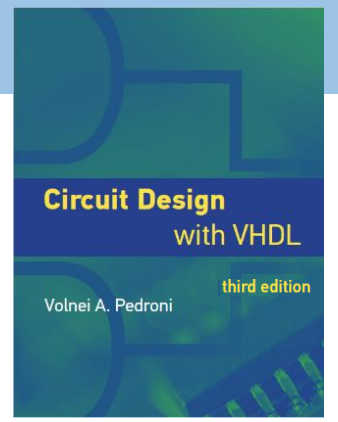
Type bit	
'0'	Logic 0
'1'	Logic 1

After (2nd VHDL version on):

Type std_ulogic	
'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

Synthesis tools:

'L' = ?
'H' = ?
'X' = ?



5. A special data type: *std_ulogic*

Before (1st VHDL version):

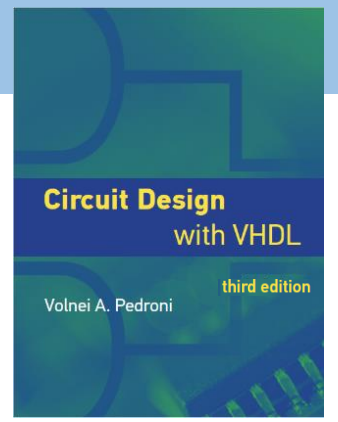
Type bit	
'0'	Logic 0
'1'	Logic 1

After (2nd VHDL version on):

Type std_ulogic	
'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

Synthesis tools:

```
'L' = '0'
'H' = '1'
'X' = '-'
```



5. A special data type: *std_ulogic*

Before (1st VHDL version):

Type bit	
'0'	Logic 0
'1'	Logic 1

After (2nd VHDL version on):

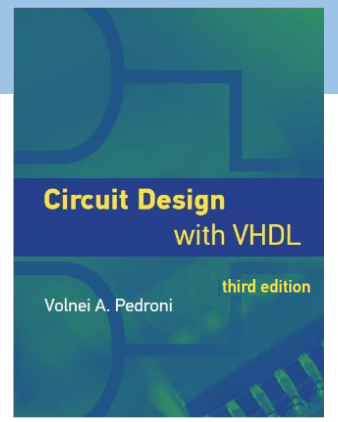
Type std_ulogic	
'U'	Uninitialized
'X'	Forcing unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High impedance
'W'	Weak unknown
'L'	Weak 0
'H'	Weak 1
'_'	Don't care

Synthesis tools:

```
'L' = '0'
'H' = '1'
'X' = '-'
```

Good design practice:

1) Use only `...???...` for inputs
2) Use only `...???...` for outputs
3) Use only `...???...` for arithmetic circuits (inputs and outputs)



5. A special data type: *std_ulogic*

Before (1st VHDL version):

Type bit	
'0'	Logic 0
'1'	Logic 1

After (2nd VHDL version on):

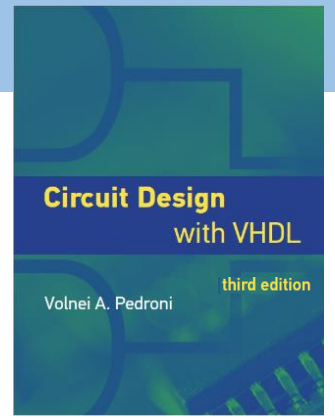
Type std_ulogic	
'U'	Uninitialized
'X'	Forcing unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High impedance
'W'	Weak unknown
'L'	Weak 0
'H'	Weak 1
'_'	Don't care

Synthesis tools:

```
'L' = '0'
'H' = '1'
'X' = '-'
```

Good design practice:

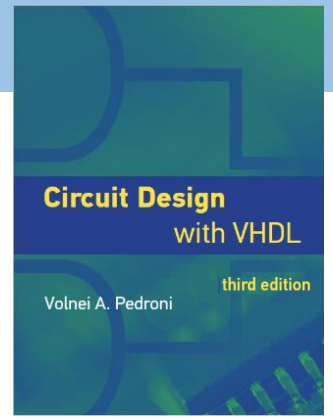
1) Use only '0', '1', '-' for inputs
2) Use only '0', '1', '-', 'Z' for outputs
3) Use only '0', '1' for arithmetic circuits (inputs and outputs)



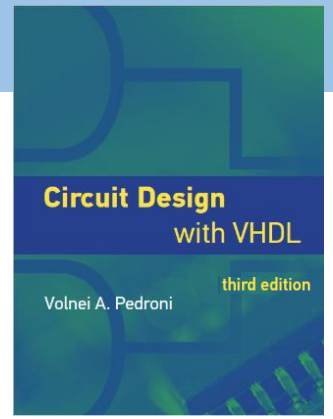
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design



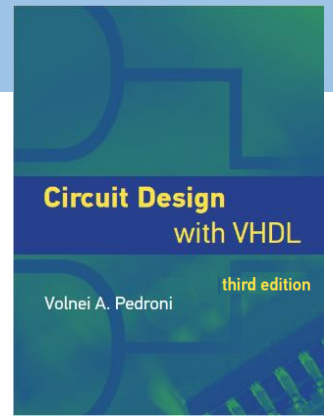
6. Lexical elements of VHDL



6. Lexical elements of VHDL

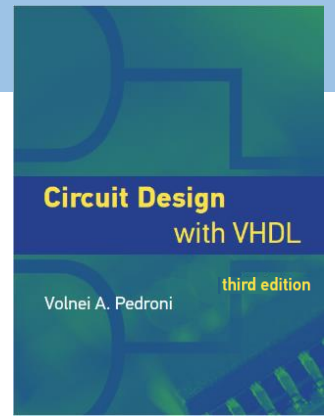
How to write:

- Bit and bit strings
- Integers
- Character and character string
- Assignment symbols
- Comments
- Identifiers
- Delimiters
- Reserved VHDL words



6. Lexical elements of VHDL

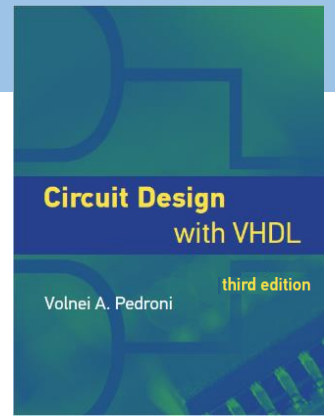
a) Bit and bit string



6. Lexical elements of VHDL

a) Bit and bit string

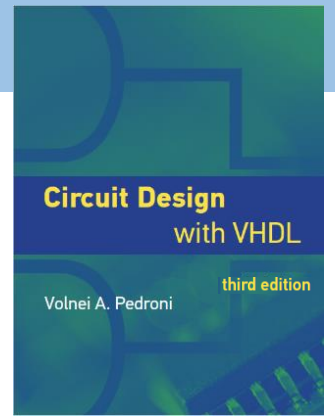
- Single quotes for single bit
  Examples: `'0', '1'`



6. Lexical elements of VHDL

a) Bit and bit string

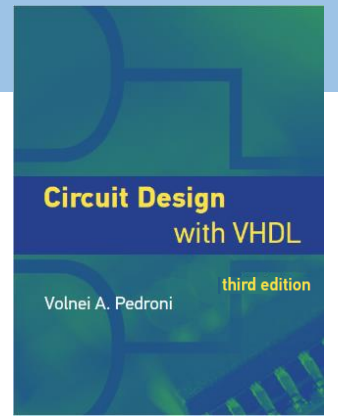
- Single quotes for single bit
  Examples: `'0', '1'`
- Double quotes for multiple bits
  Examples: `"0111", b"0111", B"0111"` (prefix optional for binary)
  `x"C2F", X"C2F"` (prefix mandatory for hexadecimal and other bases)



6. Lexical elements of VHDL

a) Bit and bit string

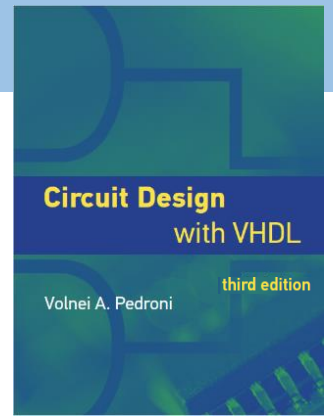
- Single quotes for single bit
  Examples: `'0', '1'`
- Double quotes for multiple bits
  Examples: `"0111", b"0111", B"0111"` (prefix optional for binary)
  `x"C2F", X"C2F"` (prefix mandatory for hexadecimal and other bases)
- Underline character requires prefix, except for integers
  Examples: `"10_0010"` (illegal), `b"10_0010"` (legal), `10_0010` (legal; an integer)



6. Lexical elements of VHDL

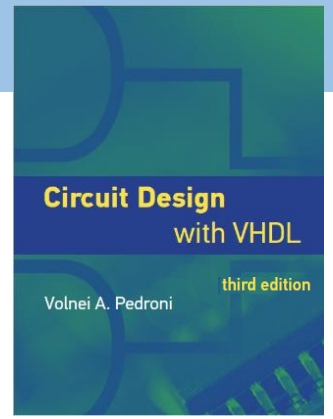
a) Bit and bit string

- Single quotes for single bit
  Examples: `'0', '1'`
- Double quotes for multiple bits
  Examples: `"0111", b"0111", B"0111"` (prefix optional for binary)
  `x"C2F", X"C2F"` (prefix mandatory for hexadecimal and other bases)
- Underline character requires prefix, except for integers
  Examples: `"10_0010"` (illegal), `b"10_0010"` (legal), `10_0010` (legal; an integer)
- Negative values are represented in two's complement format (except floating-point)
  Examples:
  Unsigned system: `"0111"` (7), `"1000"` (8), `"00000000"` (= 0), `"11111111"` (= 255)
  Signed system: `"0111"` (7), `"1000"` (–8), `"00000000"` (= 0), `"11111111"` (= –1)



6. Lexical elements of VHDL

b) Integers

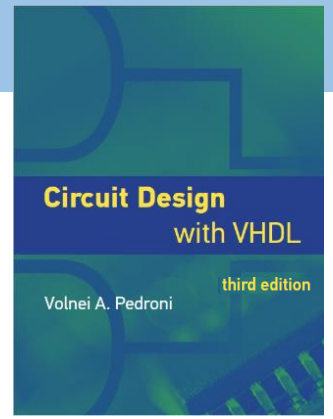


6. Lexical elements of VHDL

b) Integers

- Default = 32 bits (so don't forget to specify range)
- Max range = -2^{31} to $2^{31}-1$

Examples: `5`, `–32`, `3250`, `3_250`, `3E2` (=300), `52e3` (=52_000)



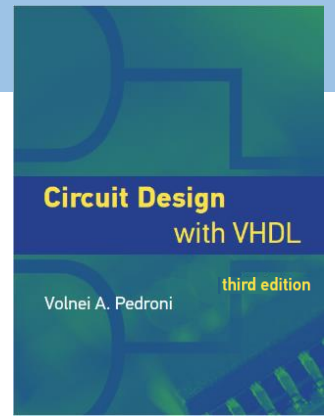
6. Lexical elements of VHDL

b) Integers

- Default = 32 bits (so don't forget to specify range)
- Max range = -2^{31} to $2^{31}-1$

Examples: `5`, `–32`, `3250`, `3_250`, `3E2` (=300), `52e3` (=52_000)

c) Character and character string



6. Lexical elements of VHDL

b) Integers

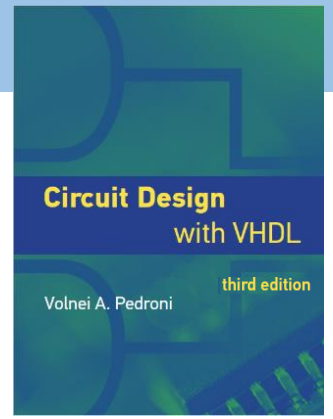
- Default = 32 bits (so don't forget to specify range)
- Max range = -2^{31} to $2^{31}-1$

Examples: `5`, `–32`, `3250`, `3_250`, `3E2` (=300), `52e3` (=52_000)

c) Character and character string

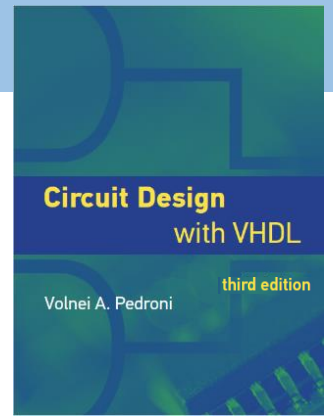
- 256-character ISO 8859-1 standard
- Single quotes for single character
- Double quotes for multiple characters

Examples: `'a'`, `'A'`, `"timed_out"`



6. Lexical elements of VHDL

d) Assignment symbols

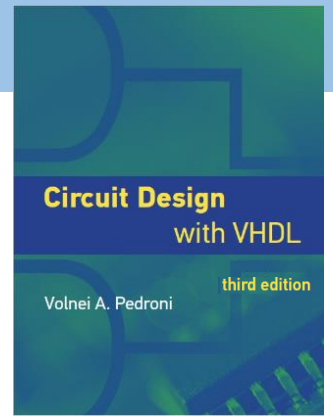


6. Lexical elements of VHDL

d) Assignment symbols

- For signals: **<=**
- For variables, constants, and initial/default values: **:=**

Examples:
```
sig <= "00000000";
var := 255;
```



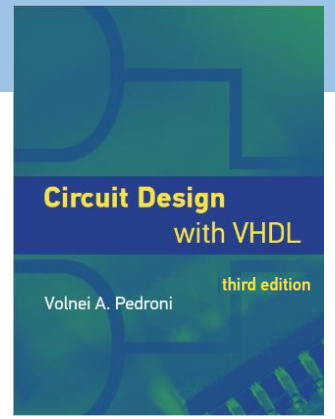
6. Lexical elements of VHDL

d) Assignment symbols

- For signals: **<=**
- For variables, constants, and initial/default values: **:=**

Examples:
```
sig <= "00000000";
var := 255;
```

e) Comments



6. Lexical elements of VHDL

d) Assignment symbols

- For signals: **<=**
- For variables, constants, and initial/default values: **:=**

Examples:
```
sig <= "00000000";
var := 255;
```

e) Comments

- Up to the end of the line: **--**
- Enclosed region (in one or multiple lines): /* */

Examples:
```
data_out <= data_in when ena else 'Z'; --tri-state buffer

/*
  ... (commented-out region)
*/
```

6. Lexical elements of VHDL

f) Identifiers (names of VHDL entities)

6. Lexical elements of VHDL

f) Identifiers (names of VHDL entities)

- Can contain only letters (a to z, A to Z), decimal digits (0 to 9), and underline character
- Must start with a letter
- Must not have two adjacent underline characters or end with an underline character
- Must not be a reserved VHDL word
- VHDL is not case sensitive

6. Lexical elements of VHDL

f) Identifiers (names of VHDL entities)

- Can contain only letters (a to z, A to Z), decimal digits (0 to 9), and underline character
- Must start with a letter
- Must not have two adjacent underline characters or end with an underline character
- Must not be a reserved VHDL word
- VHDL is not case sensitive

Examples:

Legal names: `clk, clk_5MHz, data_ready, NUMBER_OF_BITS`

Illegal names: `!rst, ena_, 4input_nand, return`

6. Lexical elements of VHDL

f) Identifiers (names of VHDL entities)

- Can contain only letters (a to z, A to Z), decimal digits (0 to 9), and underline character
- Must start with a letter
- Must not have two adjacent underline characters or end with an underline character
- Must not be a reserved VHDL word
- VHDL is not case sensitive

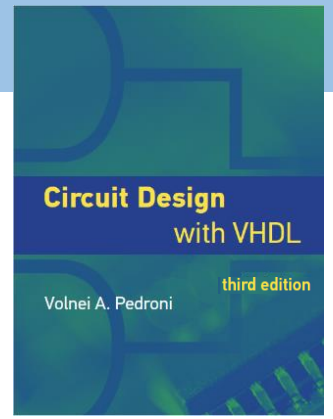
Examples:

Legal names: `clk, clk_5MHz, data_ready, NUMBER_OF_BITS`

Illegal names: `!rst, ena_, 4input_nand, return`

g) Delimiters and reserved words

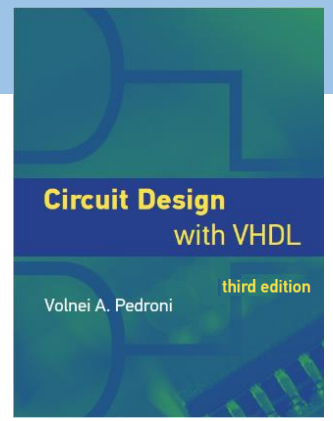
See sections 5.7.7 and 5.7.8



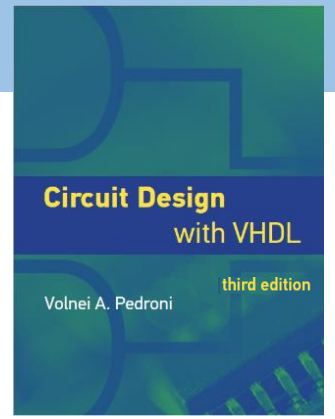
Chapter 5

Introduction to VHDL

1. Versions and purposes
2. Simplified design flow
3. Simulation types
4. Concurrent versus sequential statements
5. A special data type: *std_ulogic*
6. Lexical elements of VHDL
7. Choosing good names for your design



7. Choosing good names for your design



7. Choosing good names for your design

Read section 5.8:

5.8.1 Naming an Entity Declaration (“The design”)

5.8.2 Naming an Architecture Body

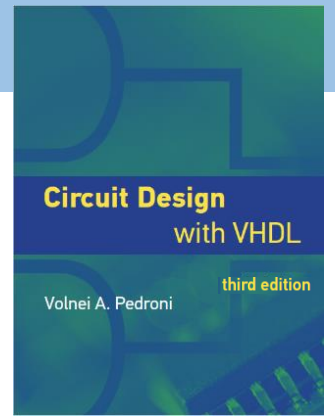
5.8.3 Naming Constants

5.8.4 Naming Signals and Variables

5.8.5 Naming Functions and Procedures

5.8.6 Naming Types

5.8.7 Naming Files



7. Choosing good names for your design

Read section 5.8:

5.8.1 Naming an Entity Declaration ("The design")

5.8.2 Naming an Architecture Body

5.8.3 Naming Constants

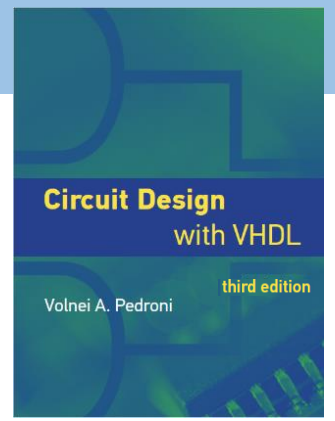
5.8.4 Naming Signals and Variables

5.8.5 Naming Functions and Procedures

5.8.6 Naming Types

5.8.7 Naming Files

This is very important and will affect your grade!



End of Chapter 5