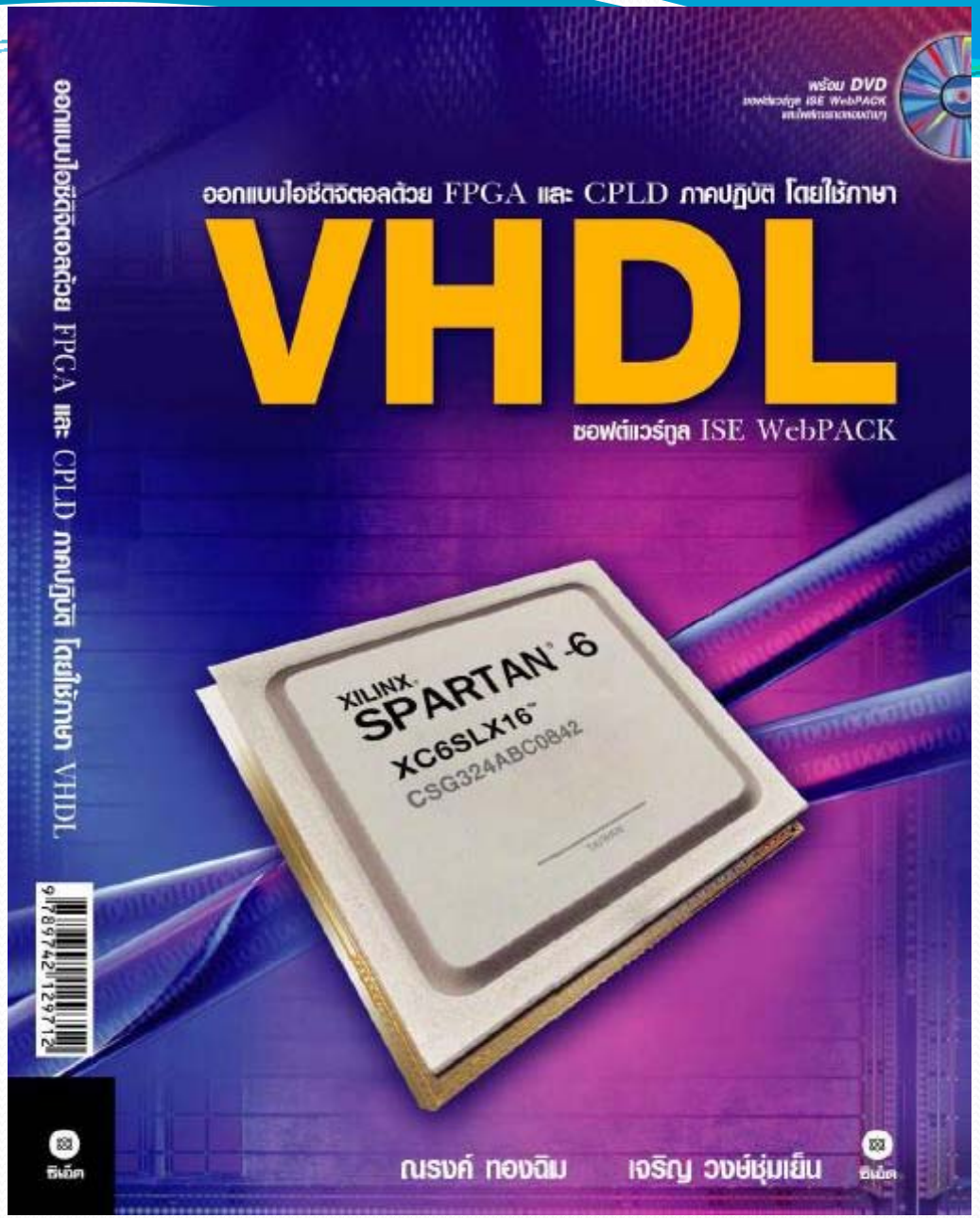
Advance Digital System Design [VHDL]

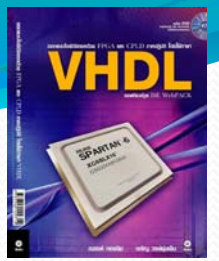
Charoen Vongchumyen

Computer engineering
King Mongkut's Institute of Technology Ladkrabang, Thailand

Jan 2010

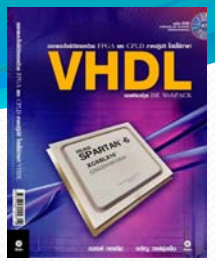

ออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ โดยใช้ภาษา VHDL
พร้อม DVD
ออกแบบไอซีดิจิตอลด้วย FPGA และ CPLD ภาคปฏิบัติ โดยใช้ภาษา
VHDL
ซอฟต์แวร์ทูล ISE WebPACK
XILINX
SPARTAN -6
XC6SLX16
CSG324ABC0842
ณรงค์ ทองฉิม
เจริญ วงษ์ชุ่มเย็น
ซีเอ็ด

VHDL



- What is VHDL?
- Why we have to learn VHDL?
- How do we learn VHDL?
- When do we learn VHDL?
- Where do we learn VHDL?

Digital design



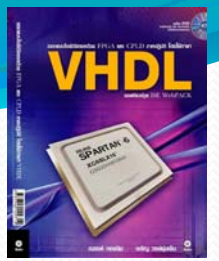
What is Digital design
&
Why ?

Introduction

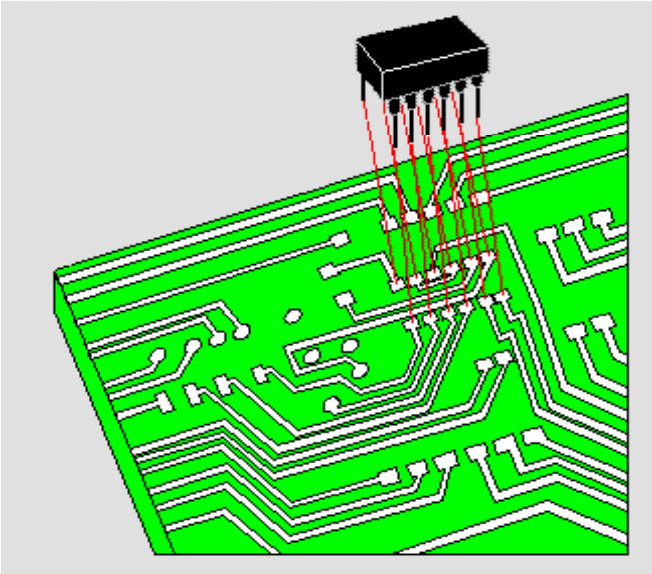
&

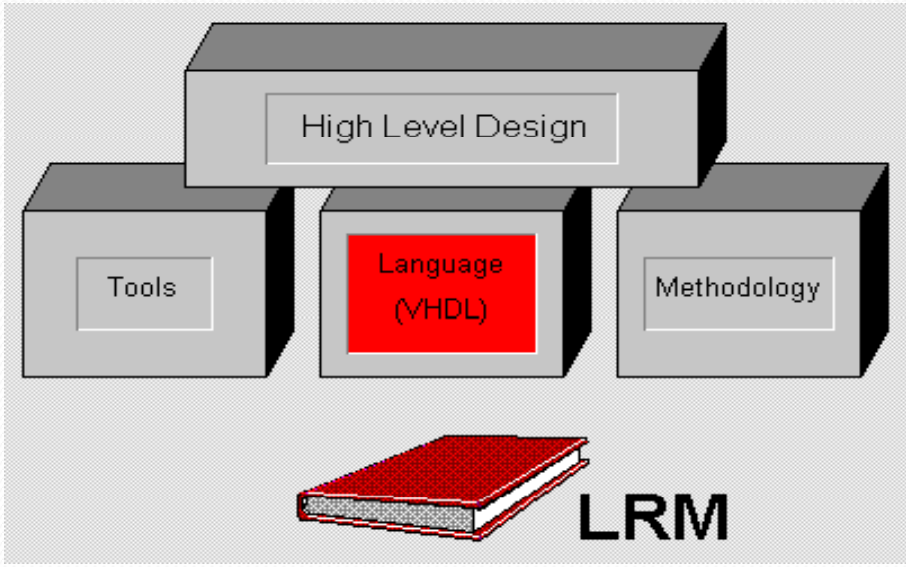
Overview

What is VHDL ?

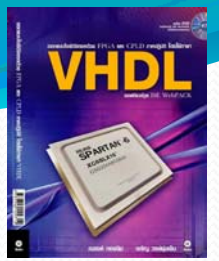


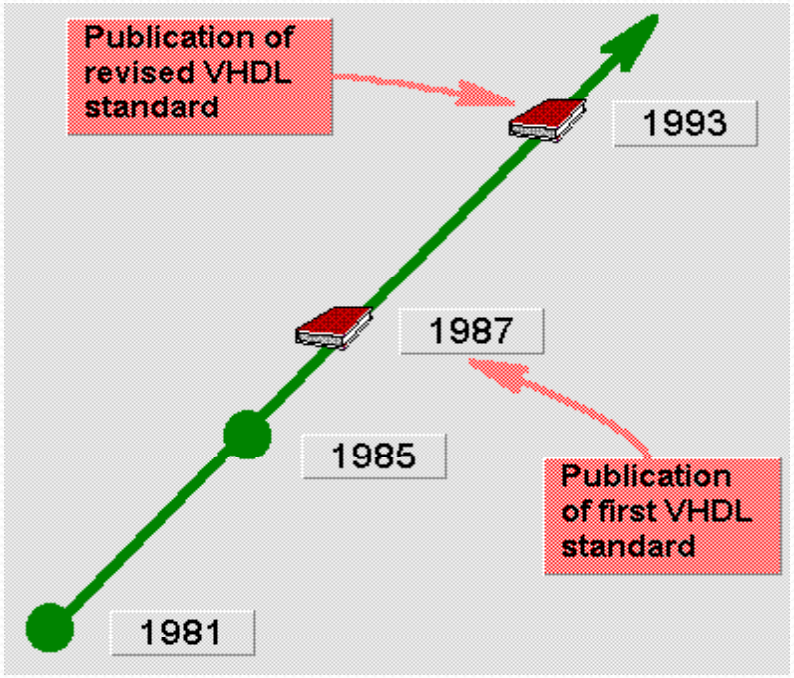
- VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- Modeling DIGITAL Electronic Systems
- Both Concurrent and Sequential statements





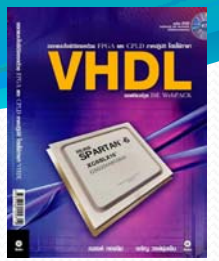
VHDL History

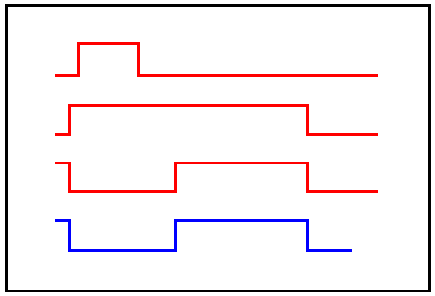




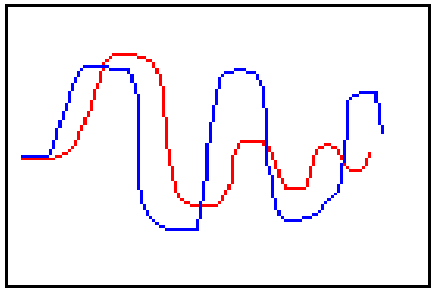
- Department of Defense (DoD) developed in 1981
- IEEE 1076-1987 Standard (VHDL'87)
- IEEE 1076-1993 Standard (VHDL'93)

VHDL Limitation





Digital ✓

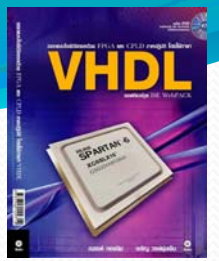


Analog

Now: ✗

Soon: ✓

Hardware Description Language (HDL)

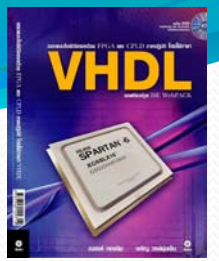


"The VHDL is a software programming language used to model the intended operation of a piece of hardware, similar to Verilog-HDL."

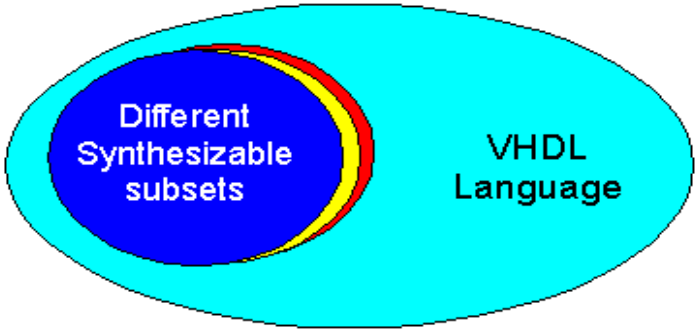
Use of the VHDL Language

1. **Documentation Language**: To provide human and machine readable documentation
2. **Design Language**: To provide a structure reflecting hardware design and hierarchy and provide methods to handle complexity by partitioning the design.

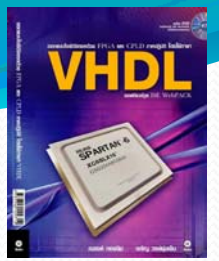
Hardware Description Language (HDL)

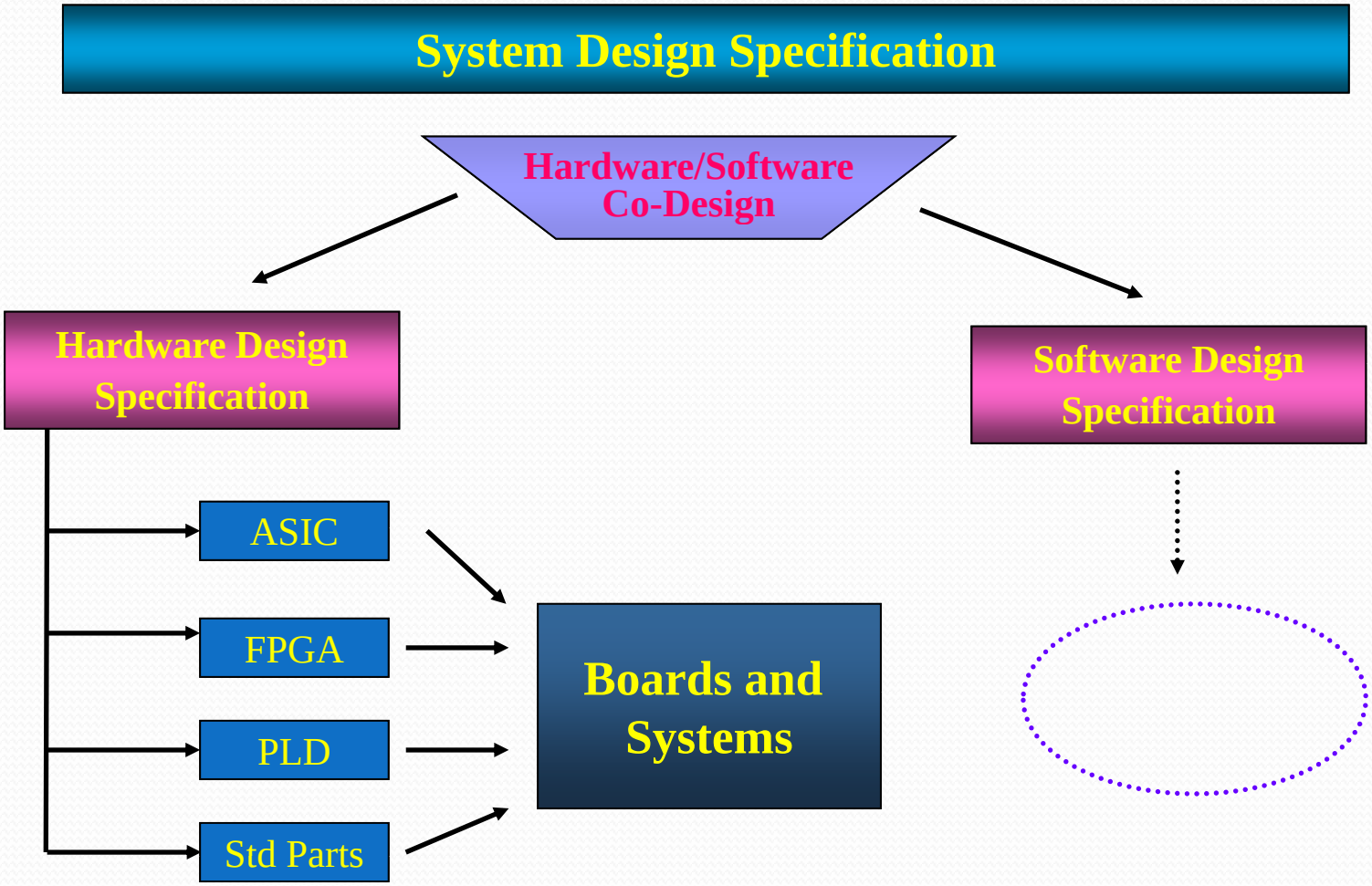


3. **Verification Language** : To provide a concurrent method verifying hardware interaction and provide constructs for stimulating a design.
4. **Test Language** : To provide ability to generate test vectors, multiple testbench strategies and method to write self checking code.
5. **Synthesis Language** : To provide high-level constructs that can be translated to Boolean equations and then translate them to gate as well as to provide constructs that can be optimized.

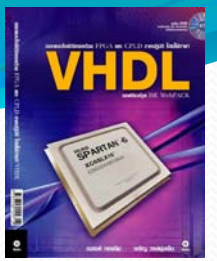


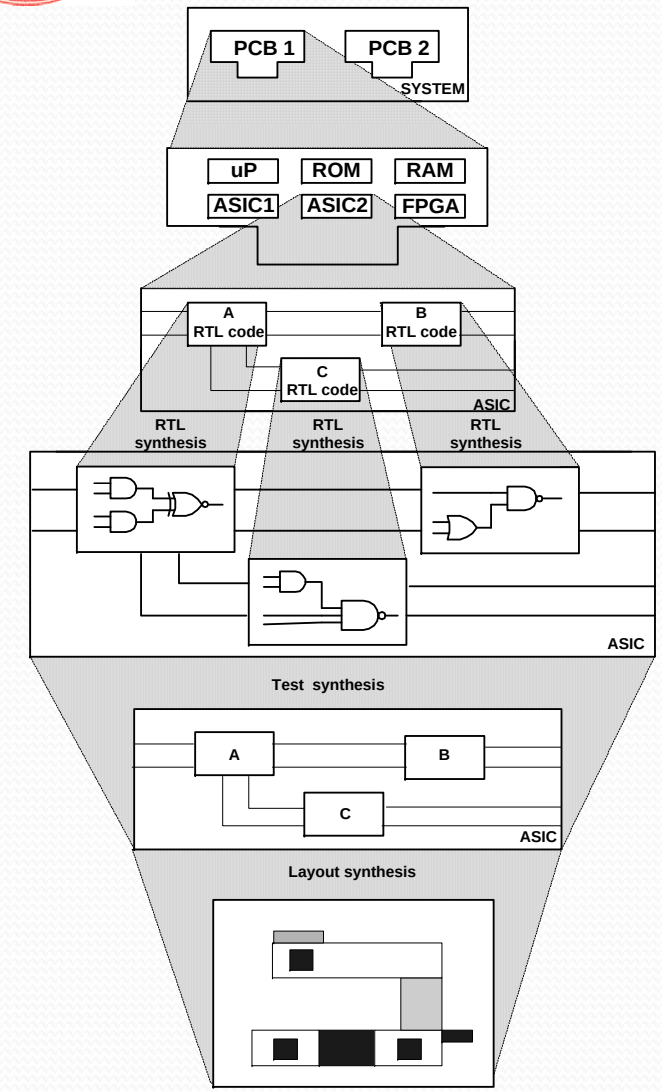
Electronic System Design





Top-Down Design



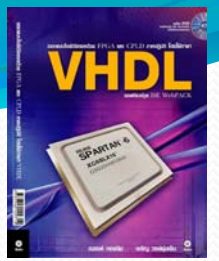


The top-level system is modeled for functionality and performance using a high-level behavioral description.

Each major component is modeled at the behavioral level and the design is simulated again for functionality and performance.

Each major component is modeled at the gate level and the design is simulated again for timing, functionality and performance.

Levels of Abstraction : Capture

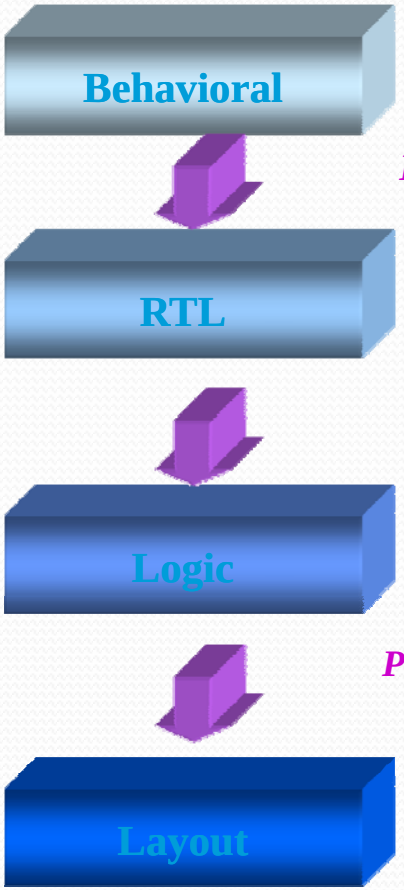


V
H
D
L

Editing VHDL Code
Block Capture
System Level Tools

Schematic Capture

Layout Tools

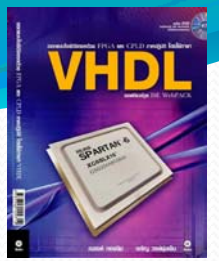


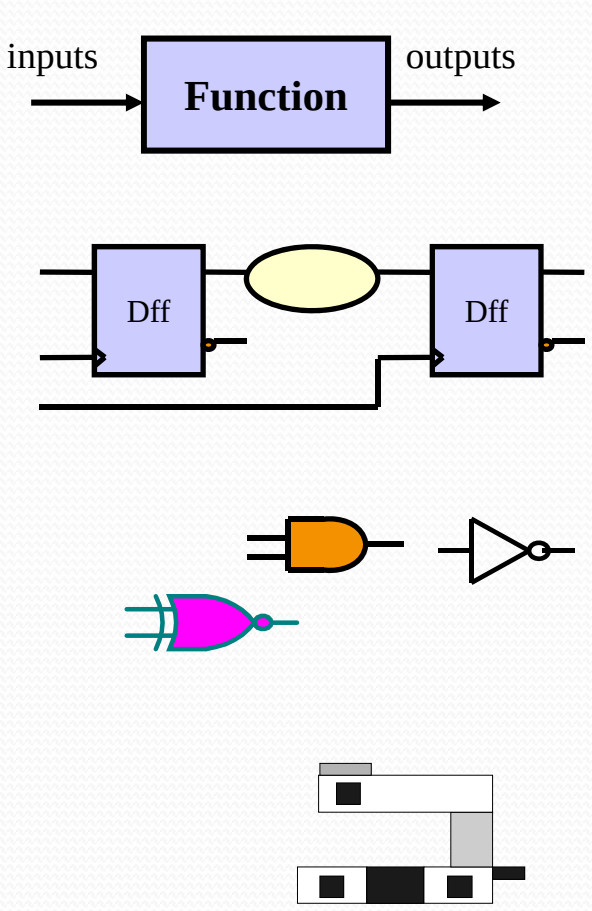
Behavioral Synthesis

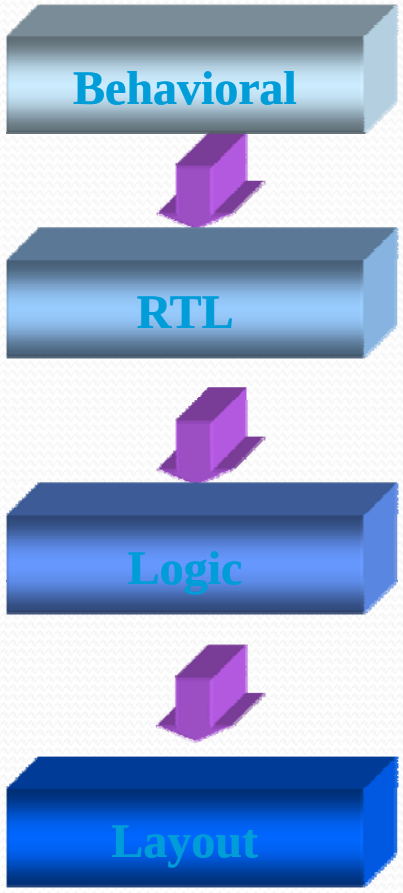
Logic Synthesis

Place and Route

Levels of Abstraction : Definition







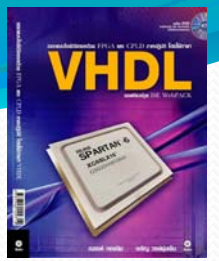
Hardware system specification
Standard parts models

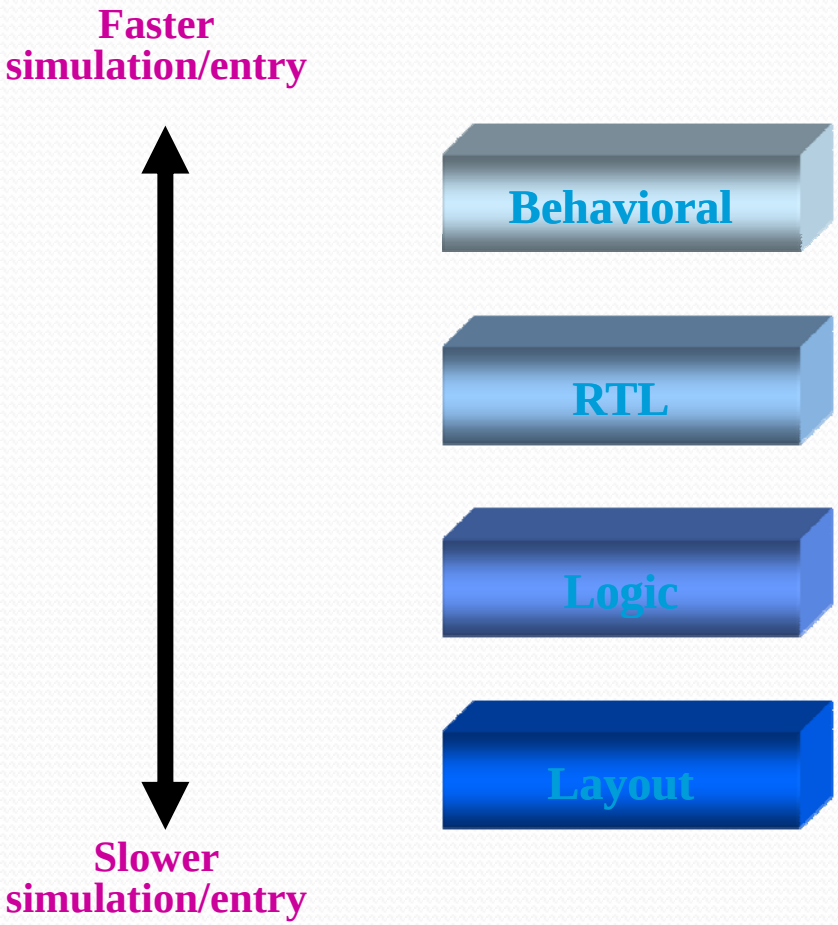
ASIC/FPGA design for synthesis
Synthesizable models

Gate level design
PLD design

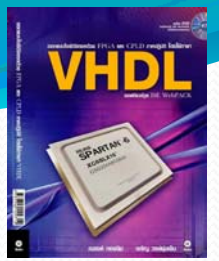
Full custom design

Tradeoffs Between Abstraction Levels



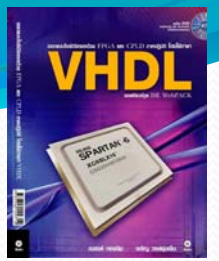


The Benefits of Using VHDL



- **Design at a higher level**
  - ➢Find problems earlier
- **Implementation independent**
  - ➢Last minute changes
  - ➢Delay implementation decisions
- **Flexibility**
  - ➢Re-use
  - ➢Choice of tools, vendors
- **Language based**
  - ➢Faster design capture
  - ➢Easier to manage

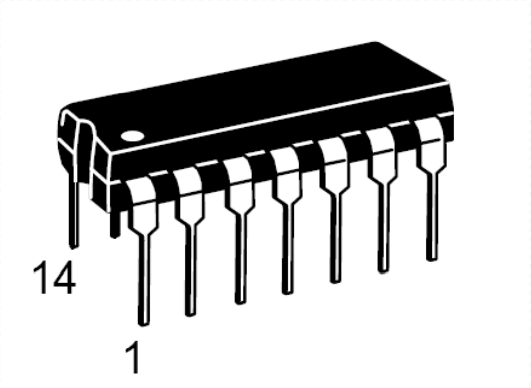
VHDL Organizations



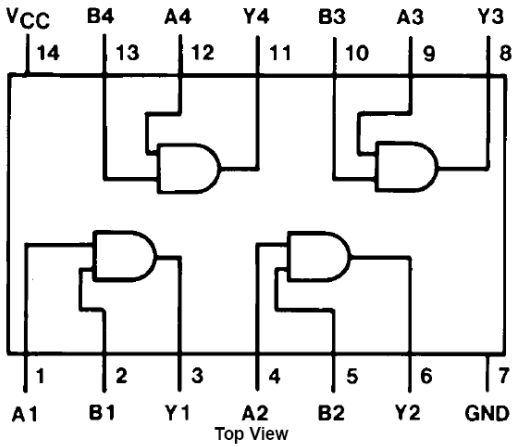
- **VHDL International**
  - Promoting use of VHDL
  - Quarterly publication, “Resource Book”
  - “Simulator Certification Test Suite”
- **IEEE Committees**
  - VASG : defining the language : various others too (VASG : VHDL Analysis and Standardization Group)
- **Europe**
  - European chapter of VASG
  - VHDL Forum for CAD in Europe
  - VHDL Newsletter, published by Esprit project
  - Regional user groups : VHDL-UK, etc . . .
- **Newsgroups**
  - comp.lang.vhdl

Digital design
with
FPGA & CPLD

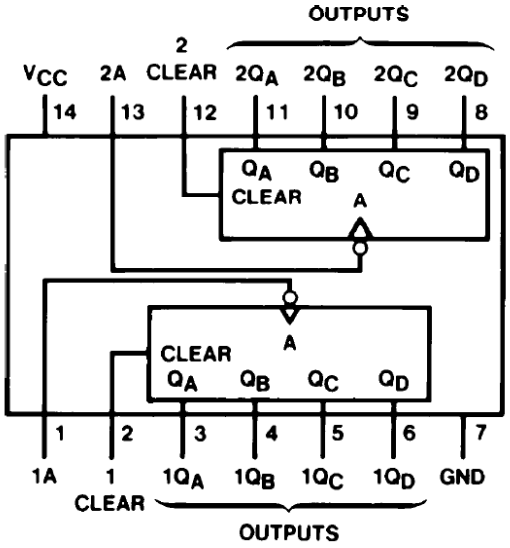
Standard IC



DIP Package

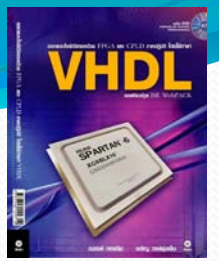


4 Nand gate



Dual binary counter

Evolution on digital design



TTL = Transistor Transistor Logic

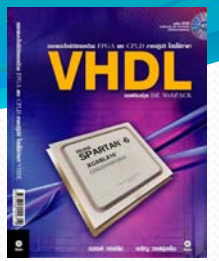
CMOS = Complementary Metal Oxide Silicon

PLD = Programmable Logic Device

CPLD = Complex Programmable Logic Device

FPGA = Field Programmable Gate Array

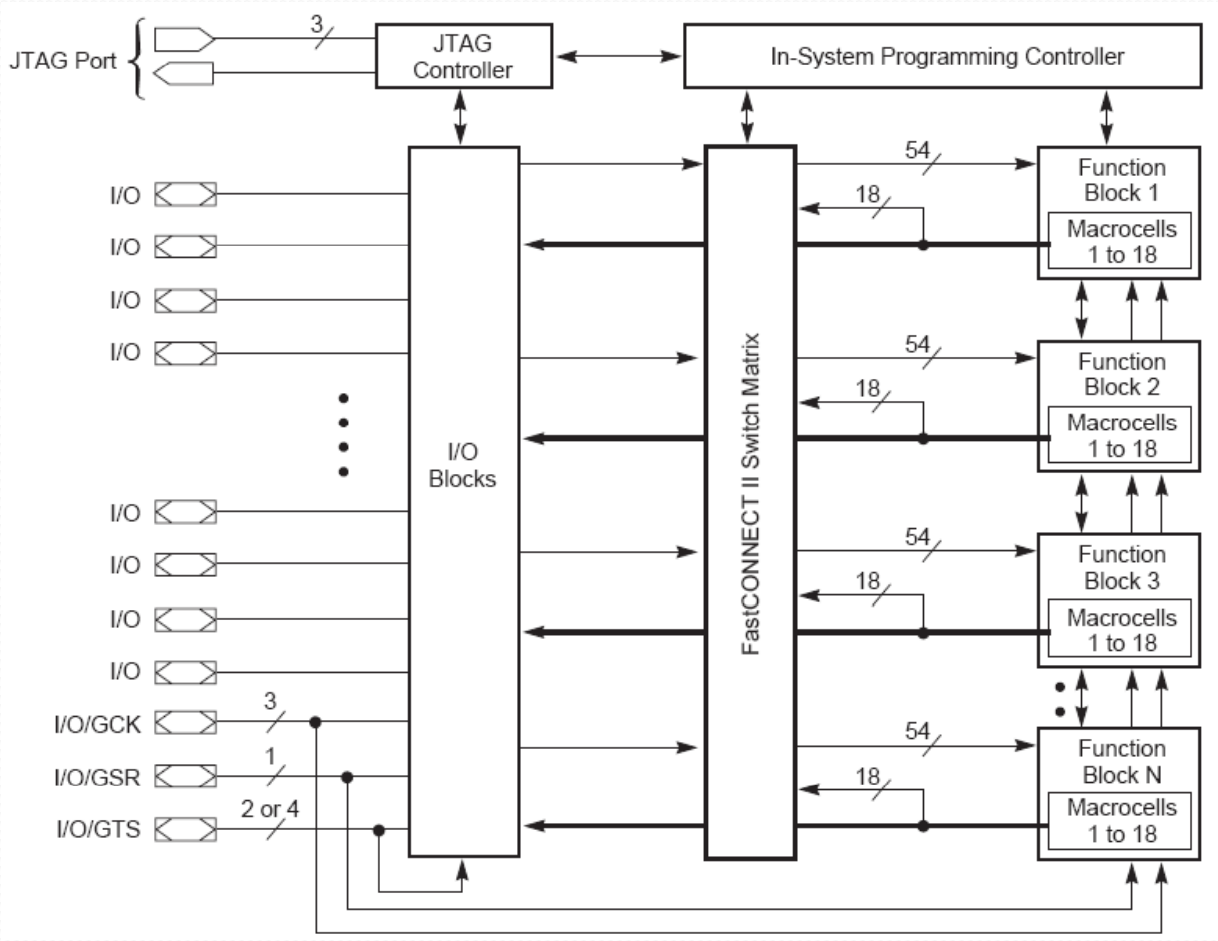
CPLD



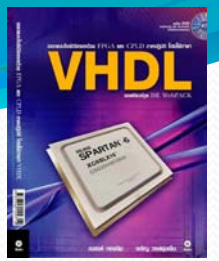
Xilinx

XC9536XL

800 Gates



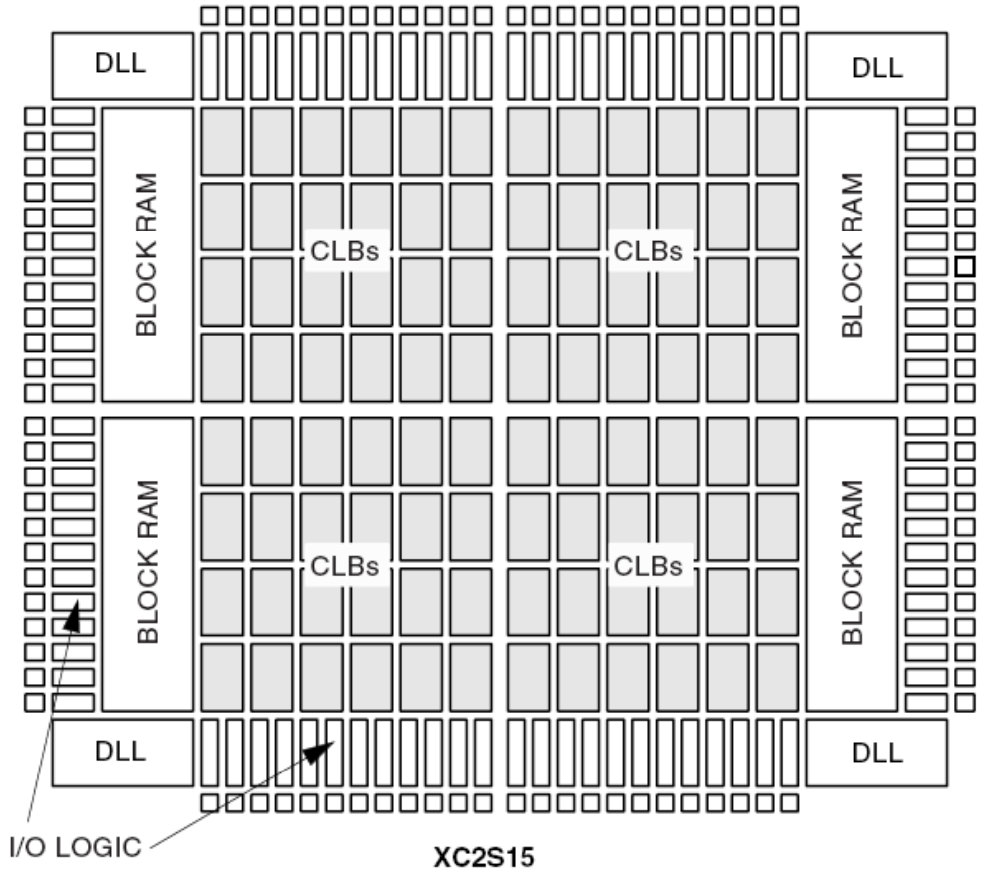
FPGA



Xilinx

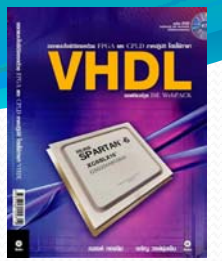
Spartan-II XC2S15

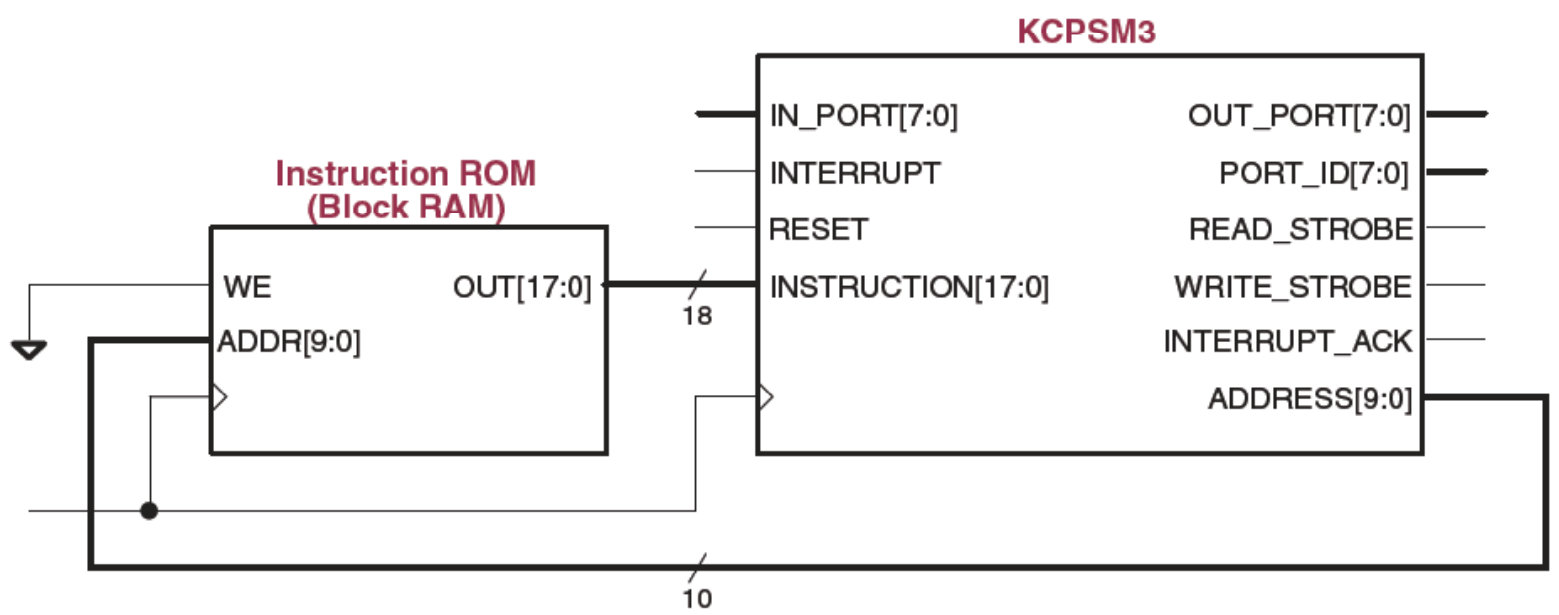
15,00 Gates



- **Delay-Locked Loop (DLL)**
- **Configurable Logic Block (CLB)**

FPGA, CPLD & MCU

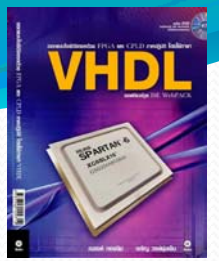


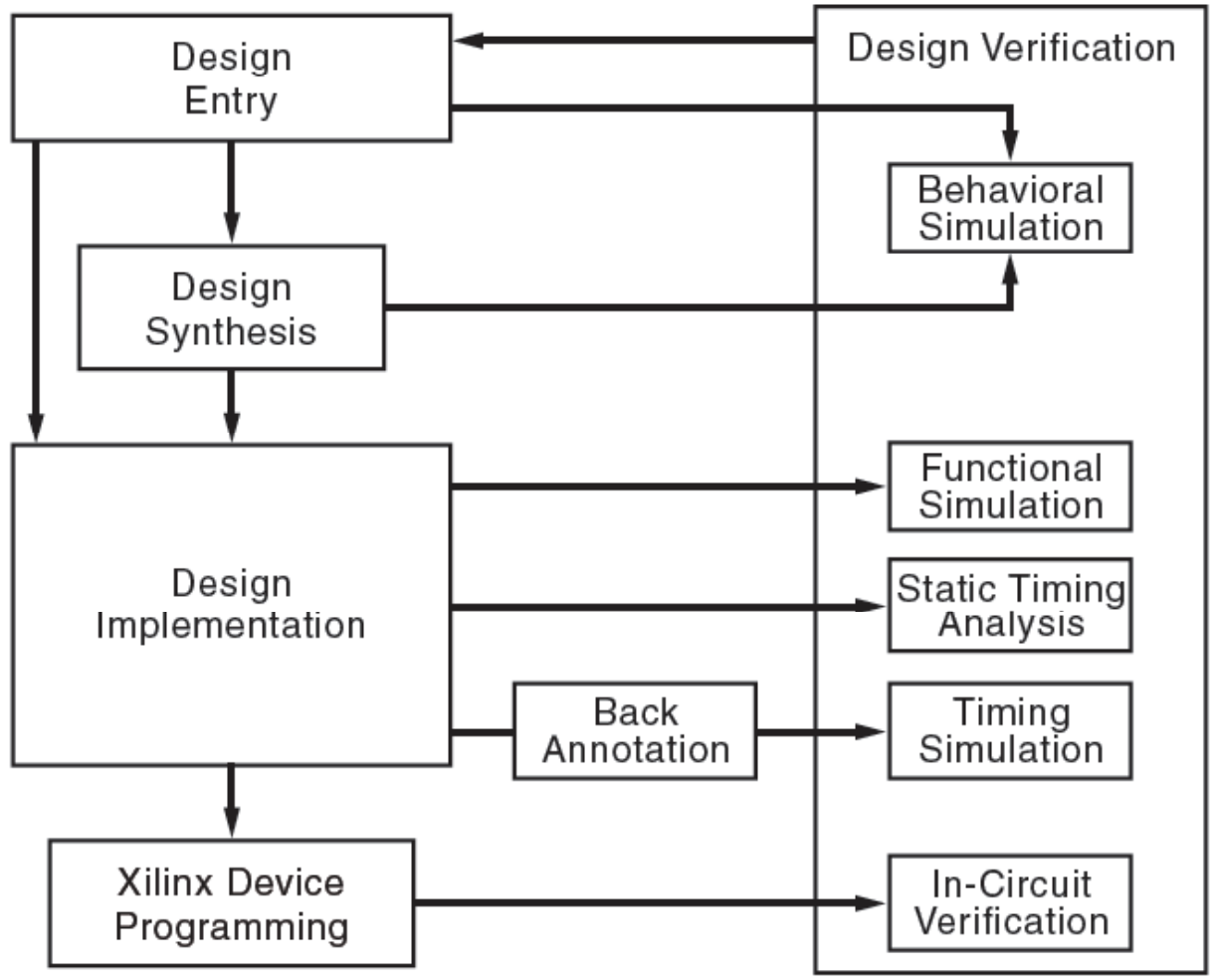


Embedded MCU in FPGA : PicoBlaze

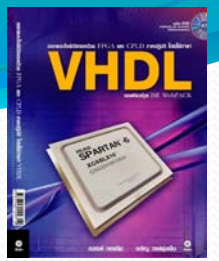
System On a Chip(SOC)

FPGA & CPLD Design flow



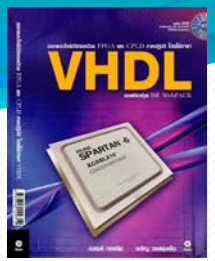


Design process



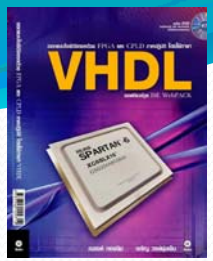
- **Design entry**
  - Schematic
  - HDL : VHDL, Verilog
  - State diagram
- **Design synthesis**
  - Netlist files
  - XST
  - EDIF

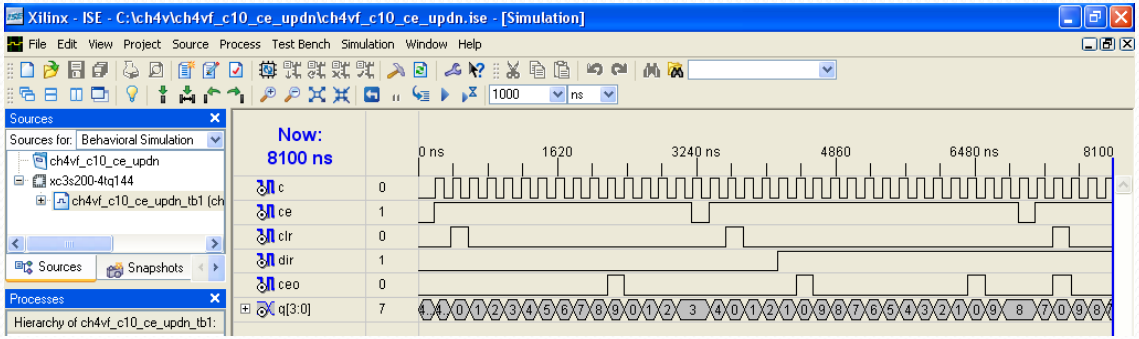
Design process

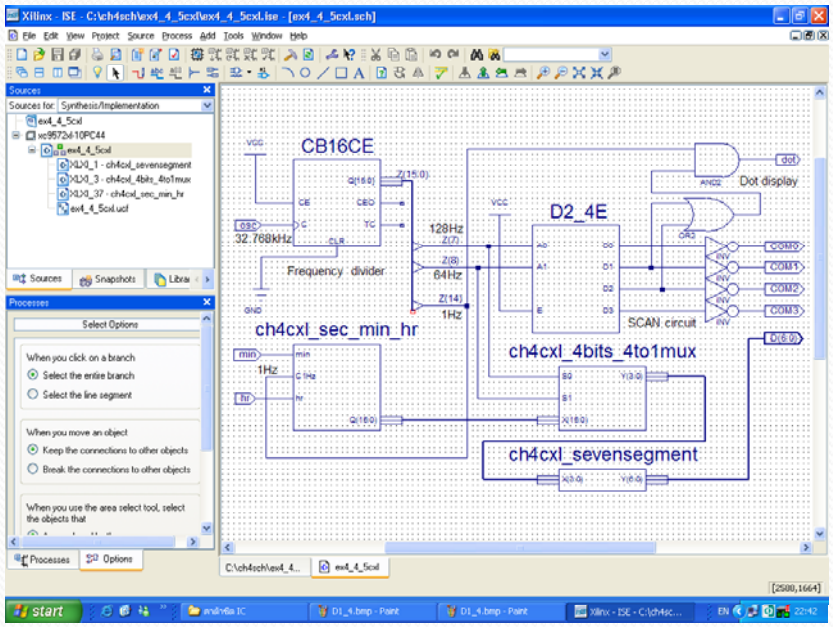


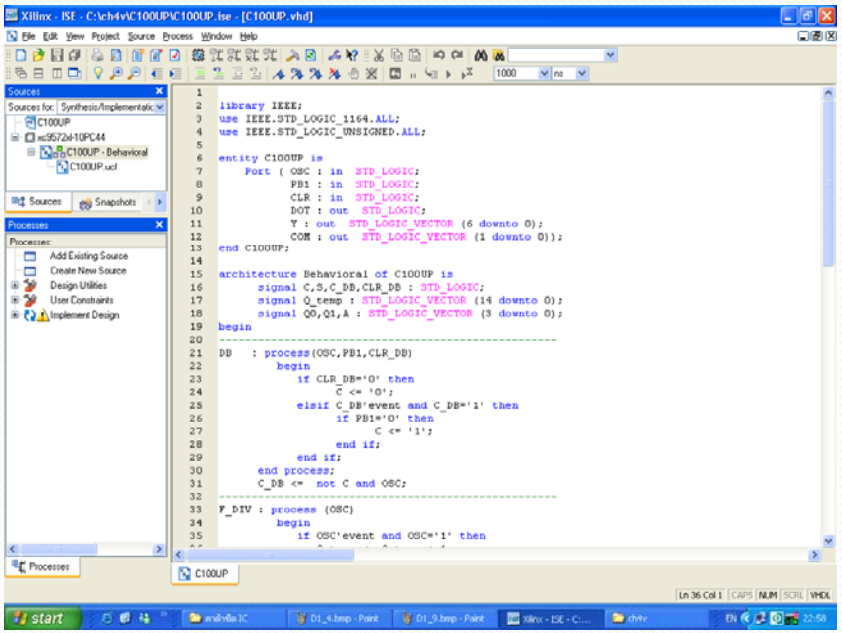
- **Design testing & Verification**
  - Behavioral simulation
  - Functional simulation
  - Timing simulation
- **Design implementation**
  - Translate (Netlist + Constrains)
  - Mapping
  - Place & Route
  - Programming file generation
  - Fitting (CPLD)
- **Programming (.bit, .jed)**

ISE WebPACK™

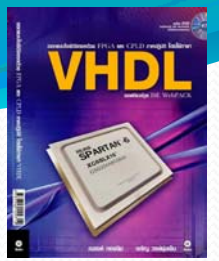

VHDL

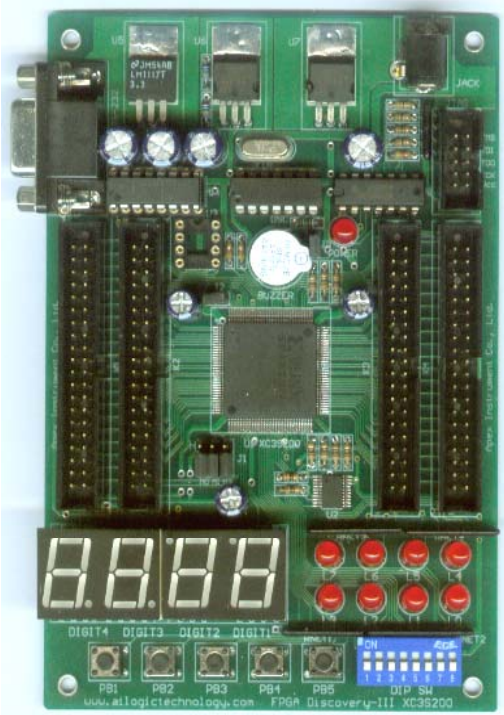

Xilinx - ISE - C:\ch4v\ch4vf_c10_ce_updn\ch4vf_c10_ce_updn.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources
Sources for: Behavioral Simulation
ch4vf_c10_ce_updn
xc3s200-4tq144
ch4vf_c10_ce_updn_tb1 (ch
Sources
Snapshots
Processes
Hierarchy of ch4vf_c10_ce_updn_tb1:
Now:
8100 ns
0 ns
1620
3240 ns
4860
6480 ns
8100
c 0
ce 1
clr 0
dir 1
ceo 0
q[3:0] 7


Xilinx - ISE - C:\ch4sch\ex4_4_5cxl\ex4_4_5cxl.ise - [ex4_4_5cxl.sch]
CB16CE
D2_4E
Dot display
Frequency divider
32.768kHz
128Hz
64Hz
1Hz
SCAN circuit
ch4cxl_sec_min_hr
ch4cxl_4bits_4to1mux
ch4cxl_sevensegment

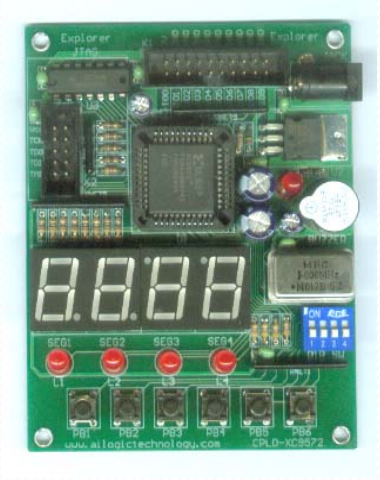

Xilinx - ISE - C:\ch4v\C100UP\C100UP.ise - [C100UP.vhd]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity C100UP is
Port (OSC : in STD_LOGIC;
PB1 : in STD_LOGIC;
CLR : in STD_LOGIC;
DOT : out STD_LOGIC;
Y : out STD_LOGIC_VECTOR (6 downto 0);
COM : out STD_LOGIC_VECTOR (1 downto 0));
end C100UP;
architecture Behavioral of C100UP is
signal C,S,C_DB,CLR_DB : STD_LOGIC;
signal Q_temp : STD_LOGIC_VECTOR (14 downto 0);
signal Q0,Q1,A : STD_LOGIC_VECTOR (3 downto 0);
begin
DB : process(OSC,PB1,CLR_DB)
begin
if CLR_DB='0' then
C <= '0';
elsif C_DB'event and C_DB='1' then
if PB1='0' then
C <= '1';
end if;
end if;
end process;
C_DB <= not C and OSC;
F_DIV : process (OSC)
begin
if OSC'event and OSC='1' then

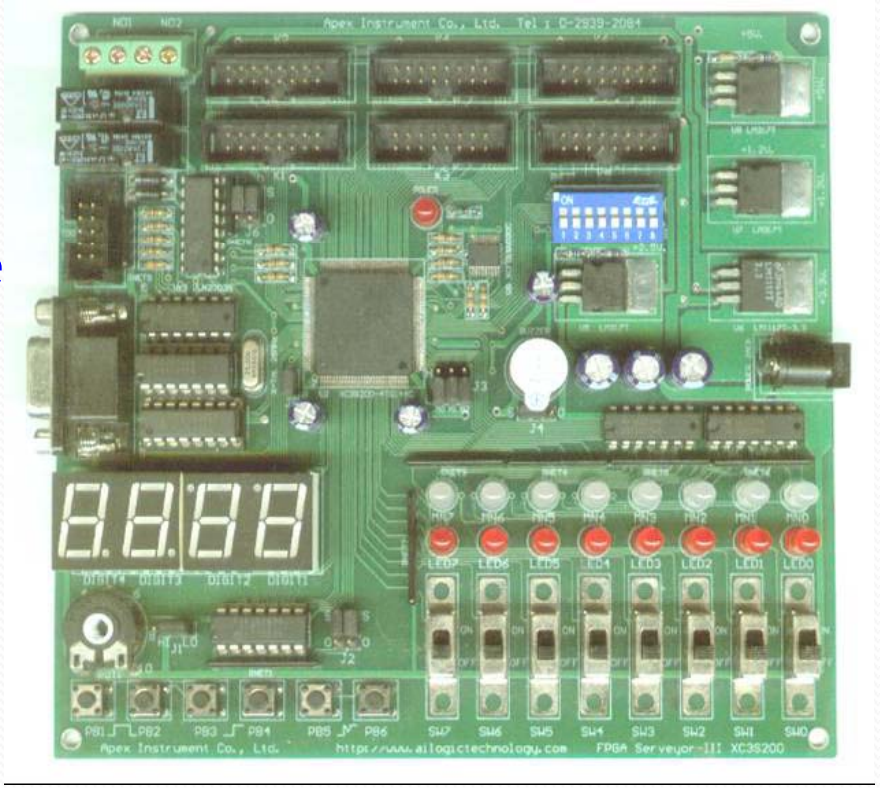
Development board





- Performance
- Reliability
- Functionality
- Reasonable price





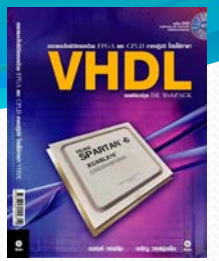
www.ailogictechnology.com

VHDL

VHDL

Basic & introduction

VHDL Basic & Introduction

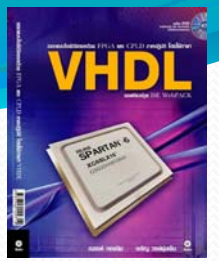


VHDL = VHSIC Hardware Description Language

(VHSIC = Very High Speed Integrated Circuit)

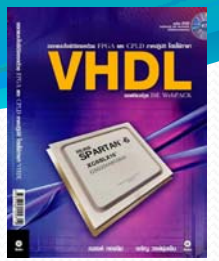
- Started by DoD, USA for military used
- High level language look like Pascal
- IEEE standard 1076 - 1987 (VHDL87)
- IEEE standard 1076 - 1993 (VHDL93)
- IEEE standard 1076 - 2000
- IEEE standard 1076 - 2002
- IEEE standard 1164 – 1993 (std_logic, std_ulogic)
- IEEE standard 1076.3 – 1997 (Synthesis packages)

VHDL Structural elements



- **Entity declaration**
  - Interface definition
  - Input and output signal (Ports)
- **Architecture**
  - Architecture body
  - Relation of input and output
  - Circuit behavior
- **Package**
  - Library : Constants, Procedures, Data types, Components
- **Configuration**
  - Match entity declaration and architecture

VHDL Basic : Example



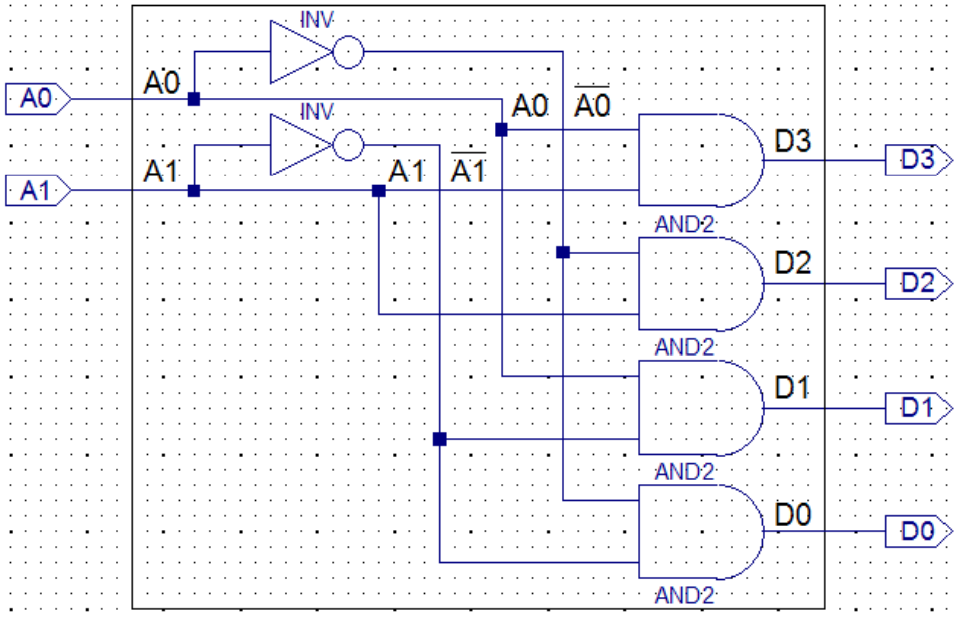
Input		Output			
A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$D0 = \overline{A1} \cdot \overline{A0}$

$D1 = \overline{A1} \cdot A0$

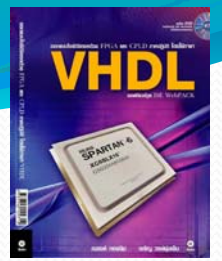
$D2 = A1 \cdot \overline{A0}$

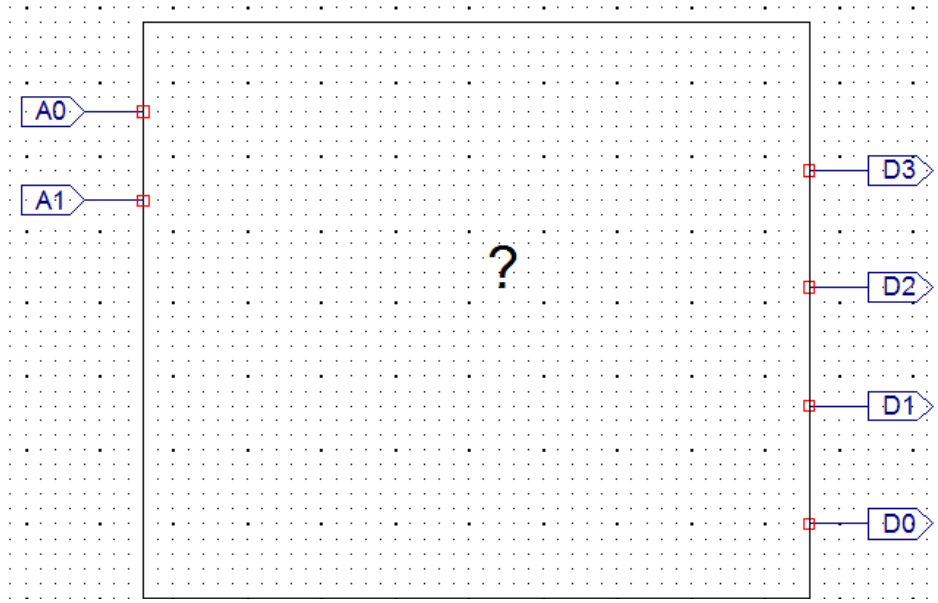
$D3 = A1 \cdot A0$

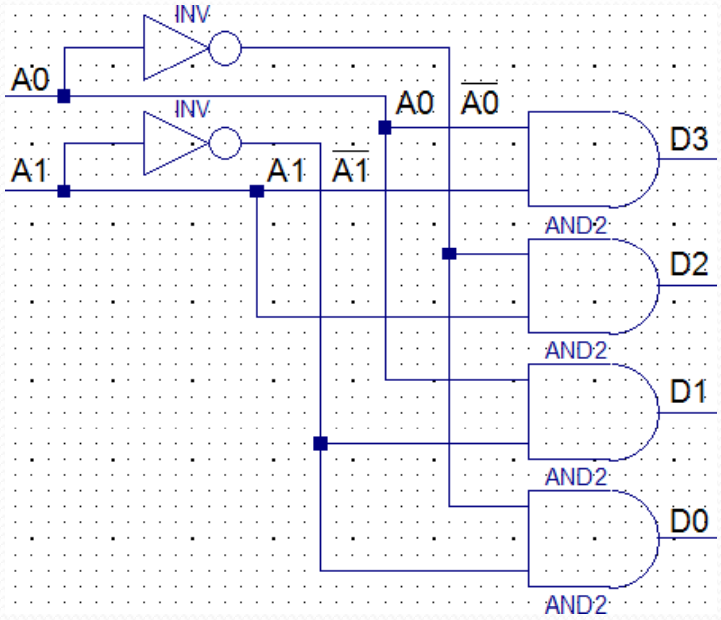


Example of 2 to 4 decoder

VHDL Basic : Example

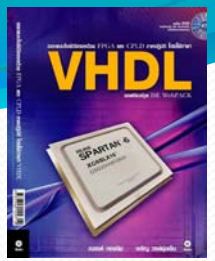






Entity declaration **Architecture**

VHDL Basic : Example




```
2   entity DECODER2TO4 is
3       port ( A0 : in  bit;
4              A1 : in  bit;
5              D0 : out  bit;
6              D1 : out  bit;
7              D2 : out  bit;
8              D3 : out  bit);
9   end DECODER2TO4;
10
11  architecture BEHAVIORAL of DECODER2TO4 is
12     begin
13        D0 <= (not A1)and(not A0);
14        D1 <= (not A1)and A0;
15        D2 <= A1 and(not A0);
16        D3 <= A1 and A0;
17  end BEHAVIORAL;
```

ชื่อเอนติตี้

Mode

ชนิดข้อมูล bit

จบด้วย ";"

บรรทัดที่ 2-9 เป็นประกาศใช้เอนติตี้ ซึ่งส่วนนี้จะบอกว่าวงจรมีอินพุตและเอาต์พุตอะไรบ้าง

ชื่ออาชิเทคเจอร์

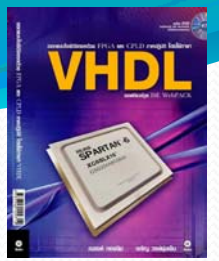
ชื่อเอนติตี้

บรรทัดที่ 11-17 เป็นอาชิเทคเจอร์บอดี้ ซึ่งส่วนนี้จะบอกว่าวงจรทำงานอย่างไร

Operator เช่น and และ not

VHDL 87 code

VHDL Basic : Example



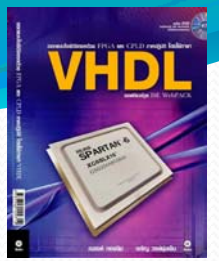

```
 2  entity DECODER2TO4 is
 3      port ( A0 : in  bit;
 4             A1 : in  bit;
 5             D0 : out  bit;
 6             D1 : out  bit;
 7             D2 : out  bit;
 8             D3 : out  bit);
 9  end entity DECODER2TO4;
10
11  architecture BEHAVIORAL of DECODER2TO4 is
12     begin
13        D0 <= (not A1)and(not A0);
14        D1 <= (not A1)and A0;
15        D2 <= A1 and(not A0);
16        D3 <= A1 and A0;
17  end architecture BEHAVIORAL;
```

บรรทัดที่ 9 อาจจะใช้ end entity แทนคำว่า end

บรรทัดที่ 17 อาจจะใช้ end architecture แทนคำว่า end

VHDL 93 code

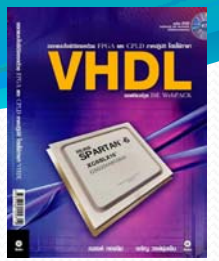
Signal Concurrency




```
11  architecture BEHAVIORAL of DECODER2TO4 is
12     begin
13          D3 <= A1 and A0;
14          D2 <= A1 and(not A0);
15          D1 <= (not A1)and A0;
16          D0 <= (not A1)and(not A0);
17  end architecture BEHAVIORAL;
```

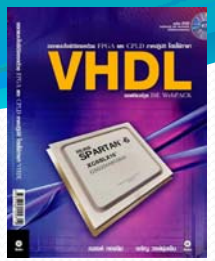
Concurrent statement

Compact style

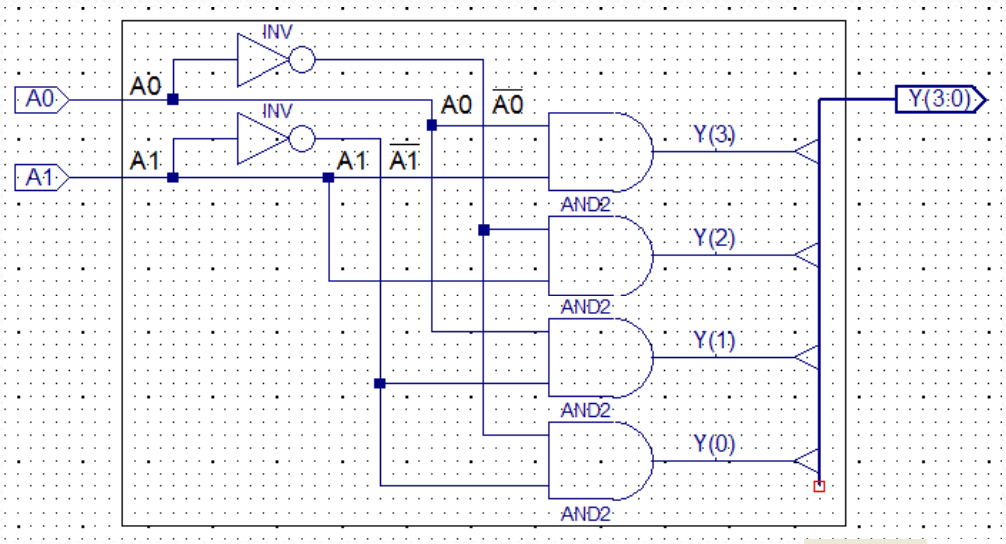



```
2   entity DECODER2TO4 is
3       port ( A0,A1 : in  bit;
4              D0,D1,D2,D3 : out  bit);
5   end DECODER2TO4;
6
7   architecture BEHAVIORAL of DECODER2TO4 is
8      begin
9         D0 <= (not A1)and(not A0);
10        D1 <= (not A1)and A0;
11        D2 <= A1 and(not A0);
12        D3 <= A1 and A0;
13  end BEHAVIORAL;
```

การเขียนอินพุตหรือเอาต์พุตหลายตัวจะต้องมี “,” กั้นระหว่างอินพุตหรือเอาต์พุตทุกตัว



Bus style

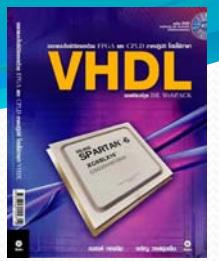



```
 2   entity DECODER2TO4 is
 3       port ( A0,A1 : in  bit;
 4              Y : out  bit_vector(3 downto 0));
 5   end DECODER2TO4;
 6
 7   architecture BEHAVIORAL of DECODER2TO4 is
 8      begin
 9          Y(0) <= (not A1)and(not A0);
10          Y(1) <= (not A1)and A0;
11          Y(2) <= A1 and(not A0);
12          Y(3) <= A1 and A0;
13   end BEHAVIORAL;
```

VHDL

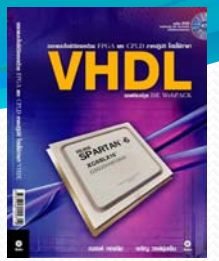
#2

VHDL General rules



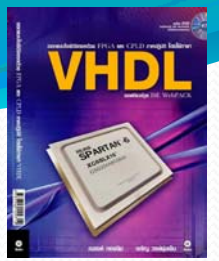
- **Case insensitive**
  - A_Bus = a_bus
  - Address_aBC = aDDRESS_Abc
- **Each statement end with “;”**
  - A_Variable := B_Variable;
  - C := X; D := Y;
  - Data_0 <= Data_5; Data_1 <= Data_2;
- **Comment with “-”**
  - Data_0 <= Data_5; -- Signal Data_5 to signal Data_0

VHDL General rules




```
 2  ------------------------------------------------
 3  -------------------- Example -------------------
 4  ---------  VHDL code of 2 to 4 decoder  ----------
 5  ------------------------------------------------
 6
 7  entity DECODER2TO4 is
 8      port ( A0,A1 : in  bit;          -- A0,A1=Input
 9             D0,D1,D2,D3 : out  bit); -- D0-D3=Output
10  end DECODER2TO4;
11
12  architecture BEHAVIORAL of DECODER2TO4 is
13     begin
14        D0 <= (not A1)and(not A0);
15        D1 <= (not A1)and A0;
16        D2 <= A1 and(not A0);
17        D3 <= A1 and A0;
18  end BEHAVIORAL;
```

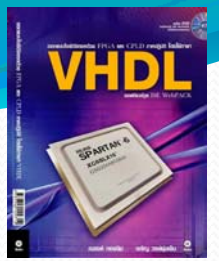
Naming



VHDL 87 identifiers

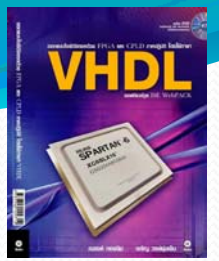
- **Use only**
  - Letter
  - Number
  - Underscore “_”
- **Rules**
  - Start with letter
  - No space
  - No repetitive underscore
  - Not end with underscore
  - No reserved word

Reserved words



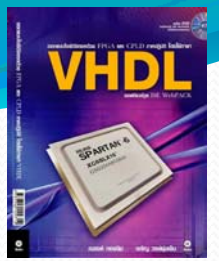
abs	bus	function	literal	others	return	transport
access	case	generate	loop	out	rol	type
after	component	generic	map	package	ror	unaffected
alias	configuration	group	mod	port	select	units
all	constant	guarded	nand	postponed	severity	until
and	disconnect	if	new	procedure	signal	use
architecture	downto	impure	next	process	shared	variable
array	else	in	nor	pure	sla	wait
assert	elsif	inertial	not	range	sll	when
attribute	end	inout	null	record	sra	while
begin	entity	is	of	register	srl	with
block	exit	label	on	reject	subtype	xnor
body	file	library	open	rem	then	xor
buffer	for	linkage	or	report	to	

Example of naming



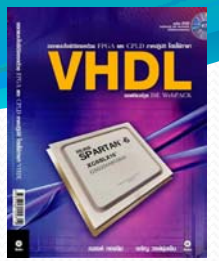
ARCHI, aRChi , archi	Right; Same name
HALF_ADDER	Right
HALF_ _ADDER	No; Double underscore
HALF ADDER_	No; Space and end with underscore
74LS00	No; Start with number
open	No; Reserved word

Port mode



- **In : receive signal from other circuit**
- **Out : send signal to other circuit**
- **Inout : receive/send signal to/from other circuit (bidirectional)**
- **Buffer : Output that can feed back signal to circuit (Avoid this mode)**

std_logic & std_logic_vector



'U' = Uninitialized

'X' = Unknown

'0' = Low (Logic‘0’)

'1' = High (Logic‘1’)

'Z' = High impedance

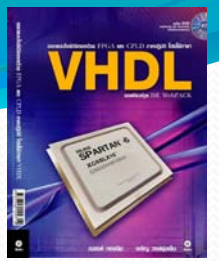
'W' = Weak unknown

'L' = Weak low

'H' = Weak high

' - ' = Don't care

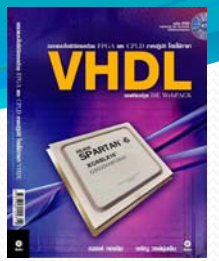
std_ulogic VS std_logic



- bit has only two state, “0” and “1”
- std_logic is a subtype of std_ulogic.

  (including std_ulogic_vector)
- Same logic with std_logic, 9 states
- Resolve function for multiple driven
- Not predefine type in IEEE Std 1076
- Have to use IEEE std_logic_1164 package

std_ulogic & std_logic



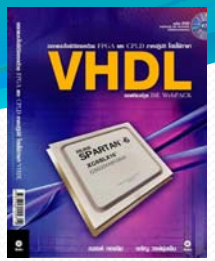

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DECODER2TO4 is
 6      port ( A0,A1 : in  STD_LOGIC;
 7             D0,D1,D2,D3 : out  STD_LOGIC);
 8  end DECODER2TO4;
 9
10  architecture BEHAVIORAL of DECODER2TO4 is
11     begin
12        D0 <= (not A1)and(not A0);
13        D1 <= (not A1)and A0;
14        D2 <= A1 and(not A0);
15        D3 <= A1 and A0;
16  end BEHAVIORAL;
```

บรรทัดที่ 2 และ 3 เป็นการเรียกใช้ Package ชื่อ std_logic_1164 ที่อยู่ในไลบรารี ชื่อ ieee

ชนิดข้อมูล std_logic

2 to 4 decoder; std_logic type

std_ulogic & std_logic



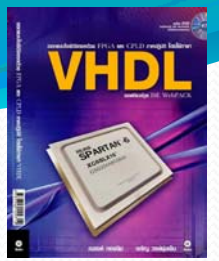

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DECODER2TO4 is
 6      port ( A0,A1 : in  STD_LOGIC;
 7             Y : out  STD_LOGIC_VECTOR(3 downto 0));
 8  end DECODER2TO4;
 9
10  architecture BEHAVIORAL of DECODER2TO4 is
11     begin
12        Y(0) <= (not A1)and(not A0);
13        Y(1) <= (not A1)and A0;
14        Y(2) <= A1 and(not A0);
15        Y(3) <= A1 and A0;
16  end BEHAVIORAL;
```

บรรทัดที่ 2 และ 3 เป็นการเรียกใช้ Package ชื่อ std_logic_1164 ที่อยู่ในไลบรารี ชื่อ ieee

ชนิดข้อมูล std_logic และ std_logic_vector

2 to 4 decoder; std_logic_vector type

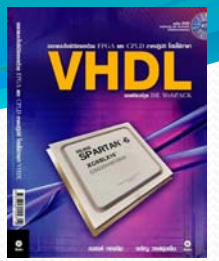
Objects



Four type of objects in VHDL

- Constant
- Signal
- Variable
- File

Constant



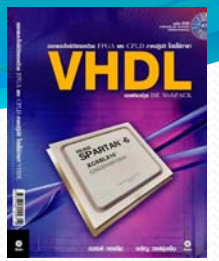
Use define constant value (never change)

constant CONSTANT_NAME : TYPE [:= VALUE];

Ex. **constant WIDTH : integer := 16 ;**

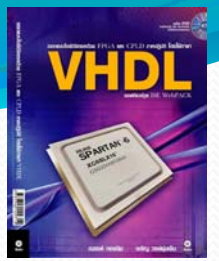
constant PI : real := 3.141592 ;

Signal



- Signal is an internal wire, not port
- Use to communicate with internal circuit (Concurrent statement)
- Use “<=” (signal assignment) to assign value
- Have to be the same data type
- Global, can use in every architect even procedure
- If use in sequential statement, Update its value when end of process, procedure or function

Signal

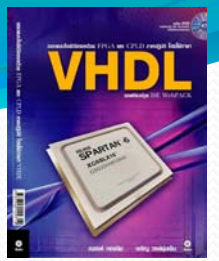


Use define internal signal (can change its value)

signal SIGNAL_NAME : TYPE [:= INITIAL_VALUE];

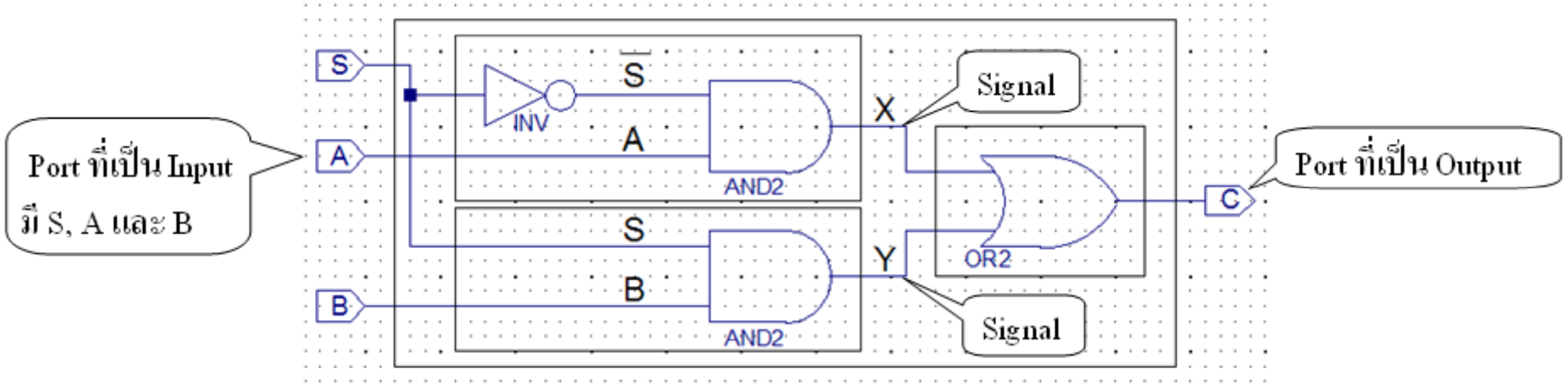
```
Ex.  signal  Q_tmp      :  integer  range  0 to 9 ;
     signal  AX         :  integer  range  0 to 15 := 0 ;
     signal  B          :  std_logic_vector (0 to 31) ;
          - - B(0) = MSB, B(31) = LSB
     signal  C     :  std_logic_vector (3 downto 0) :=  “0000” ;
          - - C(3) = MSB, C(0) = LSB
     signal  X_CLK      :  std_logic ;
```

Signal



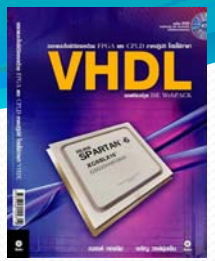
$X = \overline{S} . A$		
$Y = S . B$		
$C = X + Y$	หรือ	$C = \overline{S} . A + S . B$

MUX 2:1



Define only the signal not in port list

Signal

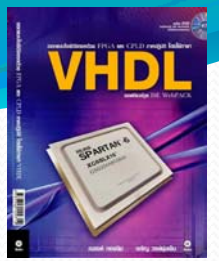



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX2TO1 is
 6      port ( A,B : in  STD_LOGIC;
 7             S : in  STD_LOGIC;
 8             C : out  STD_LOGIC);
 9  end MUX2TO1;
10
11  architecture BEHAVIORAL of MUX2TO1 is
12     signal X,Y : STD_LOGIC;
13  begin
14     X <= (not S)and A;
15     Y <= S and B;
16     C <= X or Y;
17  end BEHAVIORAL;
```

การประกาศใช้ Signal X และ Signal Y นั้นเราจะประกาศใช้ก็ต่อเมื่อชื่อของ Signal นั้นไม่มีชื่อปรากฏใน port ของ entity

MUX 2:1, Three partial circuits

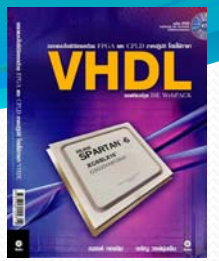
Signal




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX2TO1 is
 6      port ( A,B : in  STD_LOGIC;
 7             S : in  STD_LOGIC;
 8             C : out  STD_LOGIC);
 9  end MUX2TO1;
10
11  architecture BEHAVIORAL of MUX2TO1 is
12  begin
13        C <= ((not S)and A)or(S and B);
14  end BEHAVIORAL;
```

MUX 2:1, One big circuit (No signal needed)

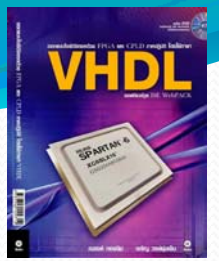
Variable



- Like signal, But use only in sequential statement
- Use “:=” (variable assignment) to assign value
- Have to be the same data type
- Local, can use only in its process, procedure of function
- Update its value immediately
- Can not exchange data outside sequential statement

  (have to assign to signal or port to exchange data)
- Temporary storage in process

Variable



Use define internal process storage

signal VARIABLE_NAME : TYPE [:= INITIAL_VALUE];

Ex. variable A_IN : integer range 0 to 9 ;

variable B : std_logic_vector (3 downto 0) := “0000” ;

- - B(3) = MSB, B(0) = LSB

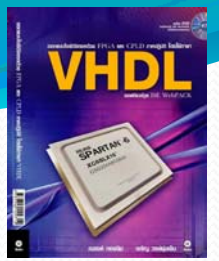
variable QX : std_logic_vector (0 to 7) ;

- - QX(0) = MSB, QX(7) = LSB

variable X : std_logic ;

variable Y : std_logic := ‘0’ ;

Data Types & Subtypes

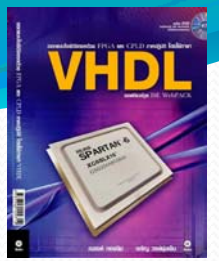



```
type BIT is ('0', '1');

-- The predefined operators for this type are as follows:

-- function "and" (anonymous, anonymous: BIT) return BIT;
-- function "or" (anonymous, anonymous: BIT) return BIT;
-- function "nand" (anonymous, anonymous: BIT) return BIT;
-- function "nor" (anonymous, anonymous: BIT) return BIT;
-- function "xor" (anonymous, anonymous: BIT) return BIT;
-- function "xnor" (anonymous, anonymous: BIT) return BIT;

-- function "not" (anonymous: BIT) return BIT;

-- function "=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "/=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "<" (anonymous, anonymous: BIT) return BOOLEAN;
-- function "<=" (anonymous, anonymous: BIT) return BOOLEAN;
-- function ">" (anonymous, anonymous: BIT) return BOOLEAN;
-- function ">=" (anonymous, anonymous: BIT) return BOOLEAN;
```

Bit type in IEEE Std 1076

Note : can not use with + - *

Data Types & Subtypes

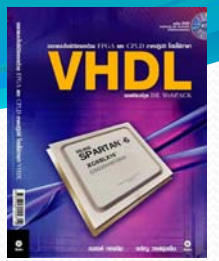



```
type INTEGER is range implementation_defined;

-- The predefined operators for this type are as follows:

-- function "**" (anonymous: universal_integer; anonymous: INTEGER)
--                return universal_integer;
-- function "**" (anonymous: universal_real; anonymous: INTEGER)
--                return universal_real;
-- function "="  (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "/=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "<"  (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "<=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function ">"  (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function ">=" (anonymous, anonymous: INTEGER) return BOOLEAN;
-- function "+"  (anonymous: INTEGER) return INTEGER;
-- function "-"  (anonymous: INTEGER) return INTEGER;
-- function "abs"(anonymous: INTEGER) return INTEGER;
-- function "+"  (anonymous, anonymous: INTEGER) return INTEGER;
-- function "?"  (anonymous, anonymous: INTEGER) return INTEGER;
-- function "*"  (anonymous, anonymous: INTEGER) return INTEGER;
-- function "/"  (anonymous, anonymous: INTEGER) return INTEGER;
-- function "mod"(anonymous, anonymous: INTEGER) return INTEGER;
-- function "rem"(anonymous, anonymous: INTEGER) return INTEGER;
-- function "**" (anonymous: INTEGER; anonymous: INTEGER) return INTEGER;
```

Integer type in IEEE Std 1076

Note : range -2,147,483,647 to 2,147,483,647

Data Types



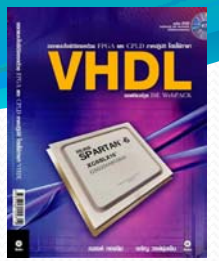
Four class of data types in VHDL

- Scalar
- Composite
- Access
- File

Note : Only scalar and composite types are synthesizable,

The other two are for simulation or RAM initialization.

Scalar Type



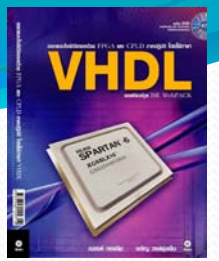
Four types of scalar

- **Enumeration**
- **Integer**
- **Physical**
- **Floating point**

Note : Enumeration and integer are “Discrete types”,

Integer, physical and floating point are “Numeric types”.

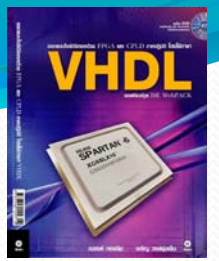
Enumeration Type



Predefined in IEEE Std 1076 (Standard package)

- Character : Alphabets in ISO 8859-1:1987; 256 chars
- Bit : Logic '0' and '1'
- Boolean : FALSE or TRUE
- SEVERITY_LEVEL : NOTE, WARNING, ERROR and FAILURE
- FILE_OPEN_KIND : READ_MODE, WRITE_MODE and APPEND_MODE
- FILE_OPEN_STATUS : OPEN_OK, STATUS_ERROR, NAME_ERROR, MODE_ERROR

User Defined : Enumeration Type

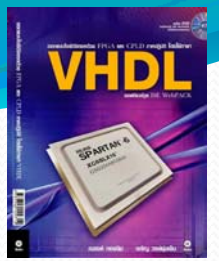


- type BOOLEAN is (FALSE, TRUE) ;
- type BIT is ('0', '1') ;
- type COLOR is (BLUE, GREEN, YELLOW, RED);
- type LOGIC_LEVEL is ('0', '1', 'Z'); - - Base type
- subtype BIT_LOGIC is LOGIC_LEVEL range '0' to '1' ; - - Subtype

Subtype is better than new type,

Can use the same operator with base type.

Integer Type



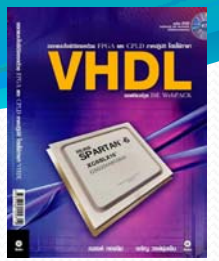
Predefined in IEEE Std 1076 (Standard package)

- – (2^{31} - 1) range -2,147,483,647 to 2,147,483,647
- "Natural" is a predefined subtype of integer (base type)

  range 0 to 2,147,483,647
- "Positive" is a predefined subtype of integer (base type)

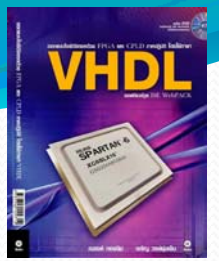
  range 1 to 2,147,483,647

Integer Type



- type INTEGER is range -2147483647 to 2147483647;
- subtype NATURAL is integer range 0 to 2147483647;
- subtype POSITIVE is integer range 1 to 2147483647;
- type WORD_INDEX is range 31 downto 0;
- type BYTE_LENGTH is range 0 to 255;
- subtype HIGH_BIT_LOW is BYTE_LENGTH range 0 to 127;

Physical Type



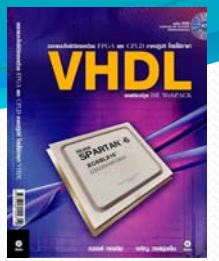
Predefined in IEEE Std 1076 (Standard package)

is “time” range -2,147,483,647 to 2,147,483,647

```
type  DURATION  is  range  -2147483647  to 2147483647
        units
                fs;                         - - femtosecond
                ps      = 1000 fs;          - - picosecond
                 ns     = 1000 ps;          - - nanosecond
                us      = 1000 ns;          - - microsecond
                ms      = 1000 us;          - - millisecond
                sec     = 1000 ms;          - - second
                min     = 60 sec;           - - minute
      end units ;
```

Note : Xilinx synthesis tool is ignore.

Floating Point Type (Real)



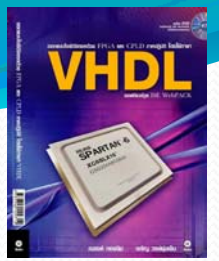
Predefined in IEEE Std 1076 (Standard package)

range $-1.0E^{38}$ to $+1.0E^{38}$

- **10.2**
- **- 15.264**
- **12**
- **15.3 ns**
- **12.7 E-6**
- **- 2.35 E10**

Note: Which one is wrong

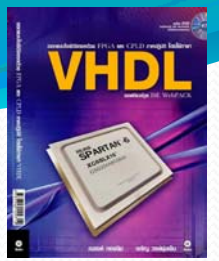
Composite Type



Tow types of composite types

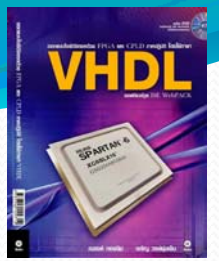
- **Array**
- **Record**

Array Type



- **Multi-value of the same type**
- **Up to 3 dimensional array (Xilinx tools)**
- **Predefined in IEEE Std 1076 (Standard package) are "string" and "bit_vector"**
  - **"string" is one dimensional array of character**
  - **"bit_vector" is on dimensional array of bit**

Array Type



User defined : Constrained and Un constrained array

- type BIT_VECTOR is array (NATURAL range <>) of BIT;

  - - Unconstrained array

- type STRING is array (POSITIVE range <>) of CHARACTER;

  - - Unconstrained array

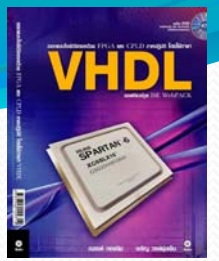
- type NIBBLE is array (3 downto 0) of BIT;

  - - Constrained array

- subtype BYTE is BIT_VECTOR (7 downto 0);

  - - Constrained array

Array Type



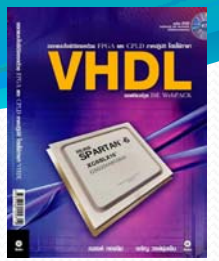
1 X 1 Dimensional

- type DATA_ROM is array (4095 downto 0) of BIT_VECTOR (7 downto 0);

- - If all ready defined BYTE

- subtype BYTE is BIT_VECTOR (7 downto 0);
- type DATA_ROM is array (4095 downto 0) of BYTE;

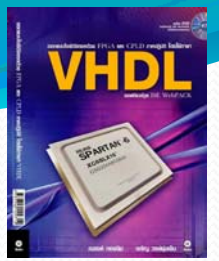
Record Type



- Same as array type BUT
- Multi-value of the “DIFFERENCE” type

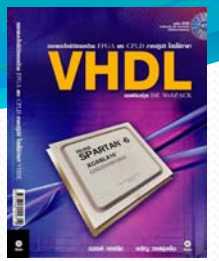
```
type  CONTROL_BUS  is  record
                        R_W          : bit;
                        DATA_BUS     : bit_vector (7 downto 0);
                        ADDR_BUS     : integer  range 0 to 1023;
                        CE           : bit;
                    end record;
```

Access & File



Later explain.

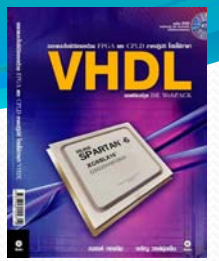
IEEE Std 1164-1993 Data Type



- In practical “0”, “1” and “Z” (high impedance) are enough
- But “Bit” have only “0” and “1”
- std_logic (std_logic_vector) and

  std_ulogic (std_ulogic_vector) are needed
- std_logic & std_logic_vector are “Resolved” type
- std_ulogic & std_ulogic_vector are “Unresolved” type
- Resovled type use resolved function for “Multiple driven”

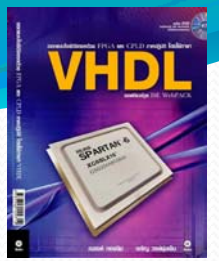
std_logic and std_ulogic



- **Predefined in IEEE Std 1164**

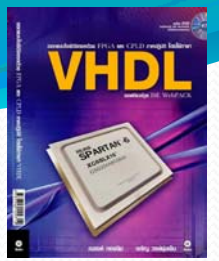
```
type  STD_ULOGIC  is (        'U',  - - Uninitialized
                              'X',  - - Forcing Unknown
                              '0',  - - Forcing 0
                              '1',  - - Forcing 1
                              'Z',  - - High Impedance
                              'W',  - - Weak Unknown
                              'L',  - - Weak 0
                              'H',  - - Weak 1
                              ' - '  - - Don't care
                      ) ;
```

std_logic and std_ulogic



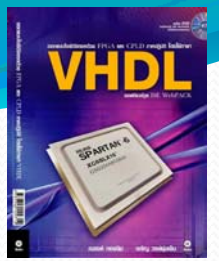

```
subtype STD_LOGIC is resolved STD_ULOGIC;
subtype X01 is resolved STD_ULOGIC range 'X' to '1';
    - - ('X', '0', '1')
subtype X01Z is resolved STD_ULOGIC range 'X' to 'Z';
    - - ('X', '0', '1', 'Z')
subtype UX01 is resolved STD_ULOGIC range 'U' to '1';
    - - ('U', 'X', '0', '1')
subtype UX01Z is resolved STD_ULOGIC range 'U' to 'Z';
    - - ('U', 'X', '0', '1', 'Z')
```

Resolved Function in Std_logic 1164



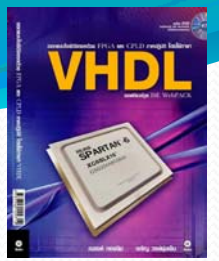

```
--------------------------------------------------------------------
--    resolution function
--------------------------------------------------------------------
   CONSTANT resolution_table : stdlogic_table := (
   -- ---------------------------------------------------------
   -- |  U    X    0    1    Z    W    L    H    -          |    |
   -- ---------------------------------------------------------
      ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
      ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
      ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
      ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
      ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
      ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
      ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
      );
```

std_logic_vector and std_ulogic_vector




```
type STD_ULOGIC_VECTOR is array (natural range <>) of STD_ULOGIC;
    - - Unconstrained array
type STD_LOGIC_VECTOR is array (natural range <>) of STD_LOGIC;
    - - Unconstrained array
```

User Predefined




```
type  NIBBLE  is  array (3 downto 0)  of  STD_LOGIC;
subtype  BYTE  is  std_logic_vector (7 downto 0);
```

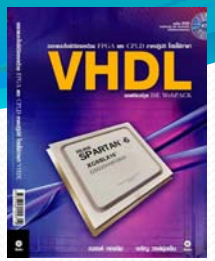
- 1 X 1 dimensional array

```
type  DATA_ROM  is  array (4095 downto 0)  of  std_logic_vector (7 downto 0);
```

- Or

```
type  DATA_ROM   is  array (4095 downto 0)  of  BYTE;
```

2 and 3 Dimensionals User Predefined



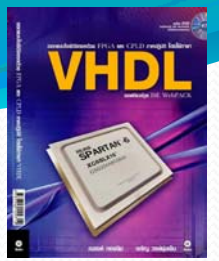
- 2 dimensionals array

```
type  DATA_2D  is  array (15 downto 0, 7 downto 0)  of  std_logic
type  A_DATA  is  array (0 to 255, 7 downto 0)  of  std_logic
```

- 3 dimensionals array

```
type  DATA_3D  is  array (0 to 255, 15 downto 0, 7 downto 0)  of  std_logic
```

IEEE Std 1164 Operators



- **Edge detection function**

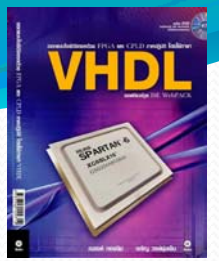
➢ **rising_edge (CLK) is CLK'event and CLK = '1'**

; CLK trigged with positive edge

➢ **falling_edge (CLK) is CLK'event and CLK = '0'**

;CLK trigged with negative edge

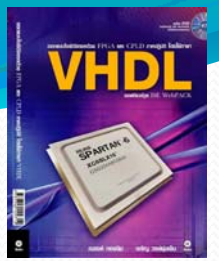
IEEE Std 1164 Functions



- **Conversion function (Types)**

Conversion	Function
std_ulogic to bit	to_bit (expression)
std_logic_vector to bit_vector	to_bitvector (expression)
std_ulogic_vector to bit_vector	to_bitvector (expression)
bit to std_ulogic	to_StdULogic (expression)
bit_vector to std_logic_vector	to_StdLogicVector (expression)
bit_vector to std_ulogic_vector	to_StdULogicVector (expression)
std_ulogic to std_logic_vector	to_StdLogicVector (expression)
std_logic to std_ulogic_vector	to_StdULogicVector (expression)

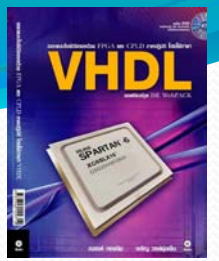
IEEE Std 1076.3 Data Types



- **"Unsigned" and "Signed" in Numeric_std or Numeric_bit**
- **In form of 2's complement**
- **Use the same operators with integer**

```
type UNSIGNED is array (NATURAL range <>) of  STD_LOGIC;
    - - Unconstrained array
type SIGNED is  array (NATURAL range <>) of  STD_LOGIC;
    - - Unconstrained array
```

IEEE Std 1076.3 Functions

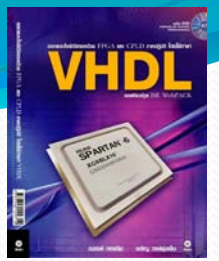


- **Conversion function (Types)**

Conversion	Function (Numeric_std package)	Function (std_logic_arith package)
unsigned to integer	to_integer (expression)	conv_integer (expression)
signed to integer	to_integer (expression)	conv_integer (expression)
integer to unsigned	to_unsigned (expression, size)	conv_unsigned (expression, size)
integer to signed	to_unsigned (expression, size)	conv_signed (expression, size)
Resizing : unsigned	resize (expression, size)	conv_unsigned (expression, size)
Resizing : signed	resize (expression, size)	conv_signed (expression, size)
std_logic_vector to unsigned	unsigned (expression) *	conv_unsigned (expression)
std_logic_vector to signed	signed (expression) *	conv_signed (expression)
unsigned to std_logic_vector	std_logic_vector (expression) *	conv_std_logic_vector (expression)
signed to std_logic_vector	std_logic_vector (expression) *	conv_std_logic_vector (expression)

"*" = Explicit type conversions

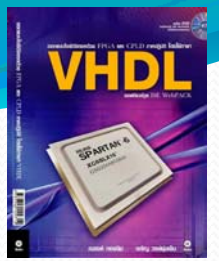
7 Segments Decoder



- **Patterns and Truth table**

No.	Input				Output						
	A(3)	A(2)	A(1)	A(0)	Y(6)=g	Y(5)=f	Y(4)=e	Y(3)=d	Y(2)=c	Y(1)=b	Y(0)=a
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	1	1	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	0	1	1	1	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	1	1	1	0	1
7	0	1	1	1	0	0	0	0	1	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	1	1	1	1
10 = A	1	0	1	0	1	1	1	0	1	1	1
11 = b	1	0	1	1	1	1	1	1	1	0	0
12 = C	1	1	0	0	0	1	1	1	0	0	1
13 = d	1	1	0	1	1	0	1	1	1	1	0
14 = E	1	1	1	0	1	1	1	1	0	0	1
15 = F	1	1	1	1	1	1	1	0	0	0	1

7 Segments Decoder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ROM1 is
 7      Port ( A : in  STD_LOGIC_VECTOR(3 downto 0);
 8             Y : out STD_LOGIC_VECTOR(6 downto 0));
 9  end ROM1;
10  architecture Behavioral of ROM1 is
11    type rom_type is array (15 downto 0) of std_logic_vector(6 downto 0);
12                            --  gfedcba   gfedcba
13    constant ROM : rom_type :=("1110001","1111001", --F,E
14                               "1011110","0111001", --d,C       Y(0)=a
15                               "1111100","1110111", --b,A         ---
16                               "1101111","1111111", --9,8 Y(5)=f|    | Y(1)=b
17                               "0000111","1111101", --7,6         --- Y(6)=g
18                               "1101101","1100110", --5,4 Y(4)=e|    | Y(2)=c
19                               "1001111","1011011", --3,2         ---
20                               "0000110","0111111");--1,0       Y(3)=d
21  signal N : integer range 15 downto 0;
22  begin
23        N <= to_integer( unsigned(A) );
24        Y <=  ROM(N);
25  end Behavioral;
```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

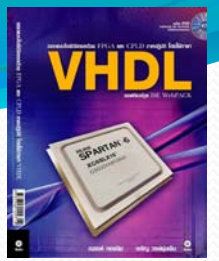
ประกาศใช้ชนิดข้อมูล rom_type แบบ Constrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง unsigned แปลงชนิดข้อมูล std_logic_vector เป็น unsigned แล้วใช้ to_integer แปลงเป็น integer

std_logic_vector 1 x 1 dimensional constrained array

7 Segments Decoder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ROM1 is
 7      Port ( A : in  STD_LOGIC_VECTOR(3 downto 0);
 8             Y : out STD_LOGIC_VECTOR(6 downto 0));
 9  end ROM1;
10  architecture Behavioral of ROM1 is
11    type rom_type is array (integer range <>) of std_logic_vector(6 downto 0);
12                                          --  gfedcba   gfedcba
13    constant ROM : rom_type(15 downto 0):=("1110001","1111001", --F,E
14                                           "1011110","0111001", --d,C      Y(0)=a
15                                           "1111100","1110111", --b,A        ---
16                                           "1101111","1111111", --9,8 Y(5)=f|   | Y(1)=b
17                                           "0000111","1111101", --7,6        --- Y(6)=g
18                                           "1101101","1100110", --5,4 Y(4)=e|   | Y(2)=c
19                                           "1001111","1011011", --3,2        ---
20                                           "0000110","0111111");--1,0      Y(3)=d
21  signal N : integer range 15 downto 0;
22  begin
23        N <= to_integer( unsigned(A) );
24        Y <=  ROM(N);
25  end Behavioral;
```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

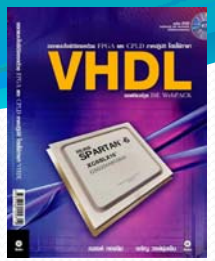
ประกาศใช้ชนิดข้อมูล rom_type แบบ Unconstrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง unsigned แปลงชนิดข้อมูล std_logic_vector เป็น unsigned แล้วใช้ to_integer แปลงเป็น integer

std_logic_vector 1 x 1 dimensional unconstrained array

7 Segments Decoder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ROM1 is
 7      Port ( A : in  unsigned(3 downto 0);
 8             Y : out unsigned(6 downto 0));
 9  end ROM1;
10  architecture Behavioral of ROM1 is
11    type rom_type is array (15 downto 0) of unsigned(6 downto 0);
12                              --  gfedcba   gfedcba
13    constant ROM : rom_type :=("1110001","1111001", --F,E
14                               "1011110","0111001", --d,C      Y(0)=a
15                               "1111100","1110111", --b,A        ---
16                               "1101111","1111111", --9,8 Y(5)=f|   | Y(1)=b
17                               "0000111","1111101", --7,6        --- Y(6)=g
18                               "1101101","1100110", --5,4 Y(4)=e|   | Y(2)=c
19                               "1001111","1011011", --3,2        ---
20                               "0000110","0111111");--1,0      Y(3)=d
21  signal N : integer range 15 downto 0;
22  begin
23        N <= to_integer(A);
24        Y <=  ROM(N);
25  end Behavioral;
```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

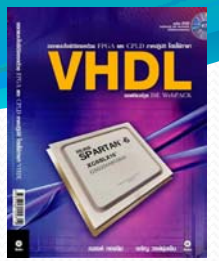
ประกาศใช้ชนิดข้อมูล rom_type แบบ Constrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

ใช้คำสั่ง to_integer แปลงชนิดข้อมูล unsigned เป็นชนิดข้อมูล integer

unsigned 1 x 1 dimensional constrained array

7 Segments Decoder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ROM1 is
 7      Port ( A : in  unsigned(3 downto 0);
 8             Y : out unsigned(6 downto 0));
 9  end ROM1;
10  architecture Behavioral of ROM1 is
11    type rom_type is array (integer range <>) of unsigned(6 downto 0);
12                                          --  gfedcba   gfedcba
13    constant ROM : rom_type(15 downto 0) :=("1110001","1111001", --F,E
14                                            "1011110","0111001", --d,C      Y(0)=a
15                                            "1111100","1110111", --b,A        ---
16                                            "1101111","1111111", --9,8 Y(5)=f|   | Y(1)=b
17                                            "0000111","1111101", --7,6        --- Y(6)=g
18                                            "1101101","1100110", --5,4 Y(4)=e|   | Y(2)=c
19                                            "1001111","1011011", --3,2        ---
20                                            "0000110","0111111");--1,0      Y(3)=d
21  signal N : integer range 15 downto 0;
22  begin
23        N <= to_integer(A);
24        Y <=  ROM(N);
25  end Behavioral;
```

เรียกใช้ numeric_std package ที่อยู่ใน Library ieee เพราะใช้คำสั่ง to_integer และใช้ unsigned

ประกาศใช้ชนิดข้อมูล rom_type แบบ Unconstrained array

การประกาศใช้ Constant ที่มีการกำหนดค่าเริ่มต้น

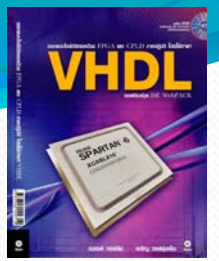
ใช้คำสั่ง to_integer แปลงชนิดข้อมูล unsigned เป็นชนิดข้อมูล integer

unsigned 1 x 1 dimensional unconstrained array

VHDL

#3

VHDL Operator

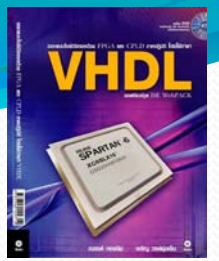


Predefined operator IEEE Std 1076

- Logical operator : and , or , nand , nor , xor และ xnor
- Relational operator : = , /= , < , <= , > และ >=
- Shift operator : sll , srl , sla , sra , rol และ ror
- Adding operator : + , – และ &
- Sign : + , –
- Multiplying operator : * , / , mod และ rem
- Miscellaneous operator : ** , abs และ not

Order by priority

Logical Operators



IEEE Std 1076

- bit, bit_vector and boolean only
- Array : same size

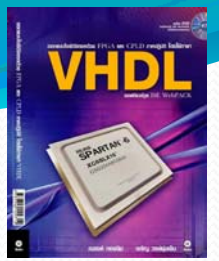
IEEE Std 1164

- std_logic (std_ulogic), std_logic_vector (std_ulogic_vector)
- Array : same size
- Overloaded logical operators, Overloaded function from 1076

IEEE Std 1076.3

- unsigned, signed
- Array : same size
- Overloaded operators

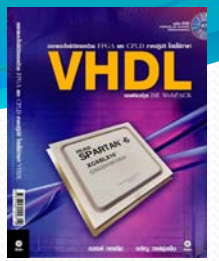
Highest priority : not




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DECODER2TO4 is
 6      port ( A0,A1 : in  STD_LOGIC;
 7             D0,D1,D2,D3 : out  STD_LOGIC);
 8  end DECODER2TO4;
 9
10  architecture BEHAVIORAL of DECODER2TO4 is
11     begin
12        D0 <= not A1 and not A0;
13        D1 <= not A1 and A0;
14        D2 <= A1 and not A0;
15        D3 <= A1 and A0;
16  end BEHAVIORAL;
```

No ( ) needed

Relational Operators



=, /=, >=, >, <=, <

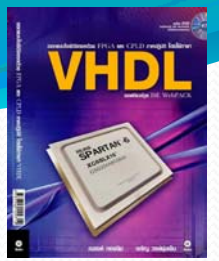
- Equal priority
- IEEE Std 1076 : Scalar and Discrete
- IEEE Std 1076.3 : Signed and Unsigned
- IEEE Std 1164 : Not supported
- But Xilinx Synthesis Tool (XST) supported

  std_logic (std_ulogic), std_logic_vector (std_ulogic_vector)
- Left alignment with unequal array

  01111 (Compare ?)

  1110

Concatenation Operators



&

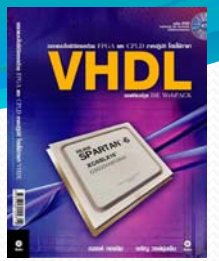
- **Concatenate two of 1 dimensional arrays**
- **IEEE Std 1076, 1076.3 supported**
- **IEEE Std 1164 : Not supported**
- **But Xilinx Synthesis Tool (XST) supported**

A <= "01" & "1101"; Equal to A <= "011101";

B <= '0' & "111" & "0011"; Equal to B <= "01110011";

C <= A & B; Equal to C <= "01110101110011"

Aggregate



Use to concatenate signal

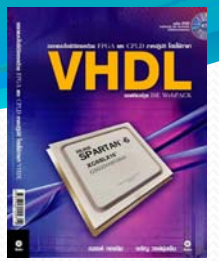
Aggregate ::= ([choice =>] identifier {, [choice =>] identifier })

::= mean “can be replacement with”

Ex.

```
signal     A : std_logic_vector (3 downto 0);  - - Assume
           A  <= (‘0’, ‘1’, ‘0’, ‘0’);                  - - Positional notation
           A  <= (0 => ‘0’, 2 => ‘1’, 1 => ‘0’, 3 => ‘0’);  - - Named notation
           A  <= “0100”;
```

Arithmetic Operators



+, - , *, /, **, MOD, REM, ABS

IEEE Std 1076

- +, -, ABS : numeric (Integer, Floating point, Physical)
- *, /, ** : Integer, Floating point
- MOD, REM : Integer
- None for bit and bit_vector

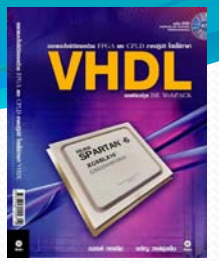
IEEE Std 1164

- None for std_logic (std_ulogic), std_logic_vector (std_ulogic_vector)
- Have to use packages : std_logic_signed or std_logic_unsigned (IEEE; Synopsis)

IEEE Std 1076.3

- unsigned, signed
- Have to use packages : Numeric_std

Type conversion



VHDL did not allow to assign value to difference data type,

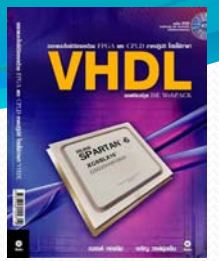
So type conversion is needed.

Explicit type conversion, Closely related data type

Type_conversion ::= type_mark (expression)

::=	**mean “can be replacement with”**
Type_mark	**mean resulted type**

Explicit Type Conversion

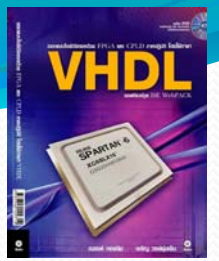


1) Integer or floating point

- **Same type but difference range**
- **Integer to Floating point or vice versa**

```
type X  is  range 0  to  100 ;          - - defined data
type Y  is  range 0  to  1000;
signal  QT  :  X;                       - - defined signal
signal  Q  :  Y ;

Q  <=  Y (QT) ;                         - -  type conversion
```

Explicit Type Conversion

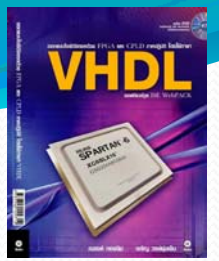


2) Array

- **Same type of member and dimensional**

Conversion	Explicit type conversion
std_logic_vector to unsigned	unsigned (expression)
std_logic_vector to signed	signed (expression)
unsigned to std_logic_vector	std_logic_vector (expression)
signed to std_logic_vector	std_logic_vector (expression)

Explicit Type Conversion

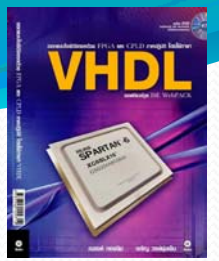


2) Array

- **Same type of member and dimensional**

```
signal  QT  :  unsigned (3 downto 0) ;          - - defined data
signal  Q   :  std_logic_vector (3 downto 0) ;

Q     <=  std_logic_vector (QT) ;               - - type conversion

Q(0) <= QT(0) ;                                 - -  same result
Q(1) <= QT(1) ;
Q(2) <= QT(2) ;
Q(3) <= QT(3) ;
```

Adder



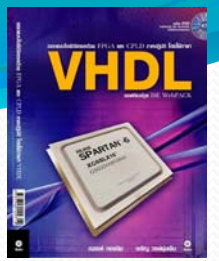

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity ADDER_2BIT is
 7      port ( A,B : in  STD_LOGIC_VECTOR (1 downto 0);
 8             C : out  STD_LOGIC_VECTOR (2 downto 0));
 9  end ADDER_2BIT;
10
11  architecture BEHAVIORAL of ADDER_2BIT is
12  begin
13
14     C <= ('0'& A) + ('0'& B);
15
16  end BEHAVIORAL;
```

ต้องเรียกใช้ std_logic_unsigned package ใน Library ieee เพราะใช้ "+" กับชนิดข้อมูล std_logic_vector

ใช้ Operator "+" กับชนิดข้อมูล std_logic_vector

Package : std_logic_unsiged

Adder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ADDER_2BIT is
 7      port ( A,B : in  STD_LOGIC_VECTOR (1 downto 0);
 8             C : out  STD_LOGIC_VECTOR (2 downto 0));
 9  end ADDER_2BIT;
10
11  architecture BEHAVIORAL of ADDER_2BIT is
12  begin
13
14     C <= std_logic_vector(unsigned('0'&A) + unsigned('0'&B));
15
16  end BEHAVIORAL;
```

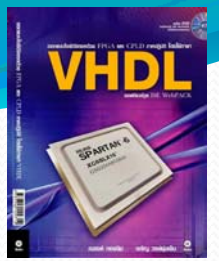
ต้องเรียกใช้ Numeric_std package ใน Library ieee เพราะมีชนิดข้อมูล unsigned

แปลงชนิดข้อมูลกลับไปเป็น std_logic_vector

แปลงชนิดข้อมูลเป็น unsigned

Package : Numeric_std; In/Out std_logic_vector

Adder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity ADDER_2BIT is
 7      port ( A,B : in  unsigned (1 downto 0);
 8             C : out  unsigned (2 downto 0));
 9  end ADDER_2BIT;
10
11  architecture BEHAVIORAL of ADDER_2BIT is
12  begin
13
14     C <= ('0'&A) + ('0'&B);
15
16  end BEHAVIORAL;
```

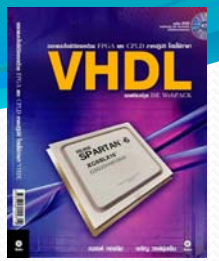
ต้องเรียกใช้ Numeric_std package ใน Library ieee เพราะใช้ชนิดข้อมูล unsigned

ชนิดข้อมูล unsigned

Package : Numeric_std; In/Out unsigned

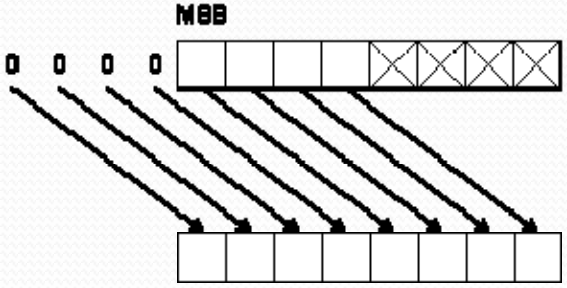
Note : Top-level port declaration must be std_logic or std_logic_vector

Shift Operators

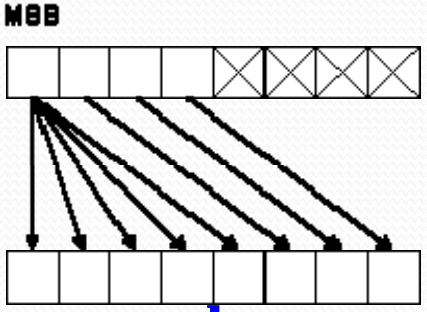


IEEE Std 1076

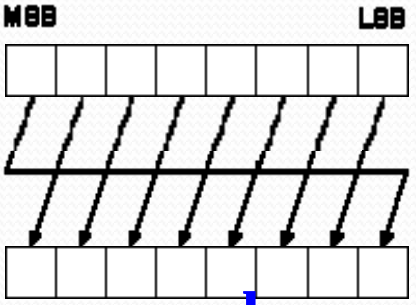
- sll : Shift Left Logical
- srl : Shift Right Logical
- sla : Shift Left Arithmetic
- sra : Shift Right Aritgmetic
- rol : Rotate Left
- ror : Rotate Right



sll

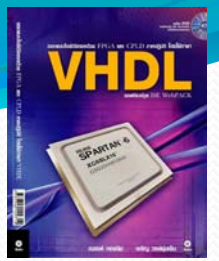


srl



rol

Shift Operators : IEEE Std 1076



signal A : bit_vector (7 downto 0) := “01110001”; - - Assume

B <= A sll 2; B <= A(5 downto 0) & “00”; B <= “11000100”

C <= A srl 2; C <= “00” & A(7 downto 2); C <= “00011100”

D <= A sla 2; B <= A(5 downto 0) & A(0) & A(0);

D <= “11000111”

E <= A sra 2; E <= A(7) & A(7) & A(7 downto 2);

E <= “00011100”

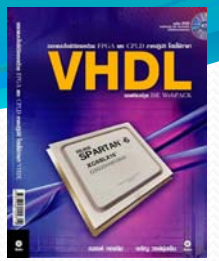
1) A(5 downto 0) is “Slices of array” A =

A(5)&A(4)&A(3)&A(2)&A(1)&A(0)

2) A(7 downto 2) is “Slices of array” A =

A(7)&A(6)&A(5)&A(4)&A(3)&A(2)

Shift Operators : IEEE Std 1164

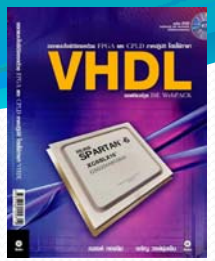


Can not use shift operator with std_logic_vector (Not supported)

But can use this technique.

```
signal  A : bit_vector (7 downto 0) := "01110001"; - - Assume

B <=  A sll 2;                                    - - not supported

B <=  A(5 downto 0) & "00";  B <= "11000100"      - - use this technique
```

Shift Operators : IEEE Std 1076.3



Unsigned and Signed.

signal U1 : UNSIGNED (7 downto 0) := "01101011" ;

signal U2 : UNSIGNED (7 downto 0) := "11101011" ;

signal S1 : SIGNED (7 downto 0) := "01101011" ;

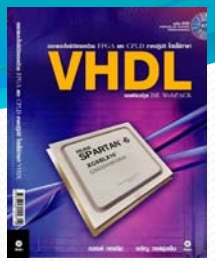
signal S2 : SIGNED (7 downto 0) := "11101011" ;

signal N : NATURAL := 3 ;

SHIFT_LEFT (U1, N) = "01011000"	SHIFT_RIGHT (U1, N) = "00001101"
SHIFT_LEFT (S1, N) = "01011000"	SHIFT_RIGHT (S1, N) = "00001101"
SHIFT_LEFT (U2, N) = "01011000"	SHIFT_RIGHT (U2, N) = "00011101"
SHIFT_LEFT (S2, N) = "01011000"	SHIFT_RIGHT (S2, N) = "11111101"
U1 sll N = "01011000"	U1 srl N = "00001101"
S1 sll N = "01011000"	S1 srl N = "00001101"
U2 sll N = "01011000"	U2 srl N = "00011101"
S2 sll N = "01011000"	S2 srl N = "11111101"

U1 rol N = "01011011"	U1 ror N = "01101101"
S1 rol N = "01011011"	S1 ror N = "01101101"
U2 rol N = "01011111"	U2 ror N = "01111101"
S2 rol N = "01011111"	S2 ror N = "01111101"

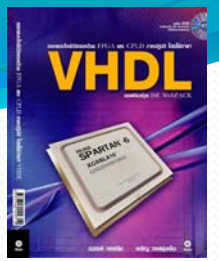
Shift Operator




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity SHIFT_LEFT_A is
 6      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
 7             B,C,D : out  STD_LOGIC_VECTOR (3 downto 0));
 8  end SHIFT_LEFT_A;
 9
10  architecture Behavioral of SHIFT_LEFT_A is
11  begin
12     B <= A(2 donwto 0) & '0';
13     C <= A(1 donwto 0) & "00";
14     D <= A(0) & "000";
15  end Behavioral;
```

Use & instead of “sll” in IEEE Std 1164

Shift Operator




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5
 6  entity SHIFT_LEFT_A is
 7      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
 8             B,C,D : out  STD_LOGIC_VECTOR (3 downto 0));
 9  end SHIFT_LEFT_A;
10
11  architecture Behavioral of SHIFT_LEFT_A is
12     signal X : UNSIGNED (3 downto 0);
13  begin
14     X <= UNSIGNED (A);
15     B <= STD_LOGIC_VECTOR ( X sll 1 );
16     C <= STD_LOGIC_VECTOR ( X sll 2 );
17     D <= STD_LOGIC_VECTOR ( X sll 3 );
18  end Behavioral;
```

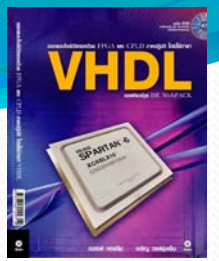
ต้องเรียกใช้ Numeric_std package ใน Library ieee เพราะใช้ชนิดข้อมูล unsigned และใช้ “sll”

Use “sll” in IEEE Std 1076.3

VHDL

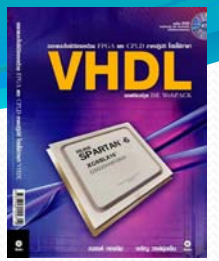
General coding

General Coding in VHDL



- **Only std_logic or std_logic_vector for top-level design (I/O port)**
- **Three necessary parts**
  - **Using the package**
  - **Entity declaration**
  - **Architecture body**

General Coding in VHDL




```
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity DECODER2TO4 is
6      port ( A0,A1 : in  STD_LOGIC;
7             Y : out  STD_LOGIC_VECTOR(3 downto 0));
8  end DECODER2TO4;
9
10 architecture BEHAVIORAL of DECODER2TO4 is
11    begin
12       Y(0) <= (not A1)and(not A0);
13       Y(1) <= (not A1)and A0;
14       Y(2) <= A1 and(not A0);
15       Y(3) <= A1 and A0;
16 end BEHAVIORAL;
```

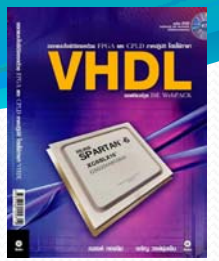
Using the package

Entity declaration

Architecture body

2 to 4 Decoder

Package Using



- **Must be place on top of the code**

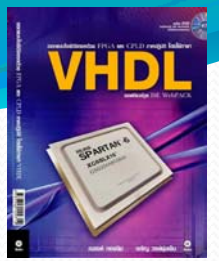
```
library  LIBRARY_NAME;
use      LIBRARY_NAME.PACKAGE_NAME.PACKAGE_PARTS;
```

- **Library : IEEE**
- **Package : std_logic_1164**

```
library  IEEE;
use      IEEE.std_logic_1164.all;
```

Note : std.standard and work.work are already used

Entity Declaration

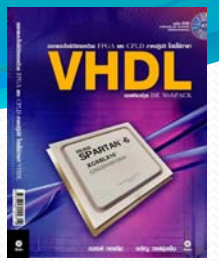


- **Define input output of the system, Port.**

```
entity  ENTITY_NAME  is
                    [ generic ( GENERIC DECLARATIONS) ; ]
                    [ port ( PORT DECLARATIONS) ; ]
end  [entity]  [ENTITY_NAME] ;

generic (  [GENERIC_NAME :  TYPE [ := VALUE ] ] { ;  GENERIC_NAME : TYPE [ := VALUE ] } ) ;

port (  [ PORT_NAME :  mode  TYPE ]  { ;   PORT_NAME :  mode  TYPE } ) ;
```

- **{...} = Multiple existed or not**
- **[...] = Existed or not**

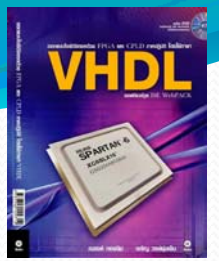
Architecture



- **Describe behavior of designed circuit.**

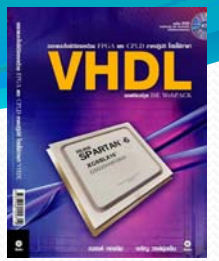
```
architecture ARCHITECTURE_NAME of ENTITY_NAME is
                { DECLARATIVE_ITEM }
begin
                { CONCURRENT_STATEMENT }
end [architecture] [ARCHITECTURE_NAME] ;
```

Declarative Items



- Subprogram declaration and Subprogram body
- Type declaration and/or Subtype declaration
- Constant declaration and/or Signal declaration
- Component declaration
- File declaration
- Attribute declaration and/or Attribute specification

Attributes



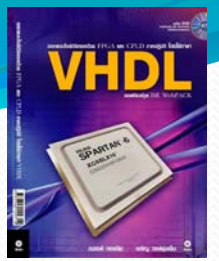
- More detail on VHDL objects. (Signal or Variable)
- Ex. Trig
- Use for simulation and subprogram.

X ’ attribute_name

- attribute_name = Name of attribute
- X = Name of objects
- “ ’ ” = Tick

Note : (not “,”)

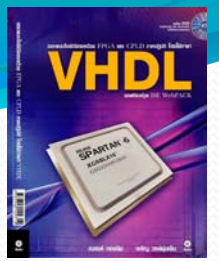
Attributes



- **LENGTH**
- **LOW**
- **RANGE**

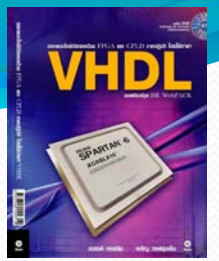
```
PACKAGE BODY Std_logic_1164 IS
-------------------------------------------------------------------
-- local types
-------------------------------------------------------------------
TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
-------------------------------------------------------------------
-- resolution function
-------------------------------------------------------------------
   CONSTANT resolution_table : stdlogic_table := (
-------------------------------------------------------------------
   -- |  U    X    0    1    Z    W    L    H    -         |   |
   -- -------------------------------------------------------
      ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
      ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
      ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
      ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
      ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
      ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
      ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
      );
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
VARIABLE result : std_ulogic := 'Z'; -- weakest state default
BEGIN
-- the test for a single driver is essential; otherwise, the
-- loop would return 'X' for a single driver of '-' and that
-- would conflict with the value of a single driver unresolved
-- signal.
IF (s'LENGTH = 1) THEN RETURN s(s'LOW);
ELSE
FOR i IN s'RANGE LOOP
result := resolution_table(result, s(i));
END LOOP;
END IF;
RETURN result;
```

Attributes



- Have 2 classes
  - Predefined attributes
  - User defined attributes
- Kinds of attribute depend on its return value
  - Value
  - Type
  - Range
  - Function
  - Signal

Predefined Attributes

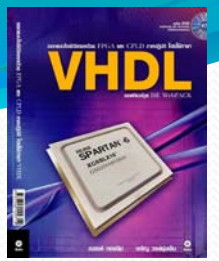


1) Type attribute

Use with Enumeration, Integer and Physical

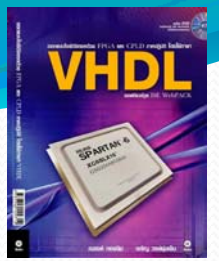
Attribute	Returns	Kind of attribute
T'left	Left bound of T	Value
T'right	Right bound of T	Value
T'high	Upper bound of T	Value
T'low	Lower bound of T	Value
T'pos(X)	Position number of X in T	Function
T'val(X)	value in T of position X	Function
T'succ(X)	T'val(T'pos(X) +1)	Function
T'pred(X)	T'val(T'pos(X) -1)	Function
T'leftof(X)	T'pred(X) if T is ascending T'succ(X) if T is descending	Function
T'rightof(X)	T'succ(X) if T is ascending T'pred(X) if T id descending	Function
T'image(X)	String representing value of X	Function
T'value(X)	Value of string X	Function

Type Attribute



- **Ascending range (0 to 15)**
  - **T'left = T'low**
  - **T'right = T'high**
  - **T'leftof(X) = T'pred(X)**
  - **T'rightof(X) = T'succ(X)**
- **Descending range (15 downto 0)**
  - **T'left = T'high**
  - **T'right = T'low**
  - **T'leftof(X) = T'succ(X)**
  - **T'rightof(X) = T'pred(X)**

Type Attribute



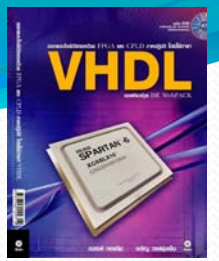
type WORD is range 15 downto 0; - - 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0 Descending

type ADDR is range -5 to 5 - - -5,-4,-3,-2,-1,0,1,2,3,4,5 Ascending range

type COLOR is (BL, G, Y, R);

WORD'left	= 15	WORD'right	= 0
WORD'high	= 15	WORD'low	= 0
ADDR'left	= - 5	ADDR'right	= 5
ADDR'high	= 5	ADDR'low	= -5
COLOR'left	= BL	COLOR'right	= R
COLOR'high	= R	COLOR'low	= BL
COLOR'pos(BL)	= 0	COLOR'val(0)	= BL
COLOR'pos(R)	= 3	COLOR'value("BL")	= BL
COLOR'succ(G)	= Y	COLOR'pred(Y)	= G
COLOR'leftof(Y)	= G	COLOR'rightof(Y)	= R

Predefined Attributes



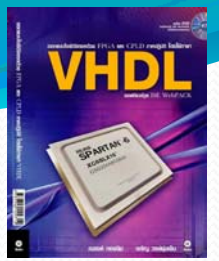
2) Array attribute

Unconstrained 1 dimensional array,

Constrained X dimensional array.

Attribute	Returns	Kind of attribute
A'left(N)	left bound of Nth index of A	Function
A'right(N)	right bound of Nth index of A	Function
A'high(N)	high bound of Nth index of A	Function
A'low(N)	low bound of Nth index of A	Function
A'range(N)	range of Nth index of A	Range
A'reverse_range(N)	reverse range of Nth index of A	Range
A'length(N)	number of values in Nth index of A	Value
A'ascending	true if array range ascending	Value

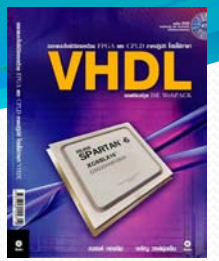
Array Attribute




```
type A1D is array (3 downto 0) of std_logic;
type A2D is array (0 to 15, 7 downto 0) of std_logic;
signal Y : std_logic_vector (31 downto 0);
```

A1D'left	= 3	A1D'right	= 0
A1D'high	= 3	A1D'low	= 0
A1D'range	= 3 downto 0	A1D'reverse_range	= 0 to 3
A1D'length	= 4	A1D'ascending	= False
A2D'left(1)	= 0	A2D'right(1)	= 15
A2D'high(1)	= 15	A2D'low(1)	= 0
A2D'range(1)	= 0 to 15	A2D'reverse_range(1)	= 15 downto 0
A2D'length(1)	= 16	A2D'ascending(1)	= True
A2D'left(2)	= 7	A2D'right(2)	= 0
A2D'high(2)	= 7	A2D'low(2)	= 0
A2D'range(2)	= 7 downto 0	A2D'reverse_range(2)	= 0 to 7
A2D'length(2)	= 8	A2D'ascending(2)	= False
Y'left	= 31	Y'right	= 0
Y'high	= 31	Y'low	= 0
Y'range	= 31 downto 0	Y'reverse_range	= 0 to 31
Y'length	= 32	Y'ascending	= False

Predefined Attributes



3) Signal attribute

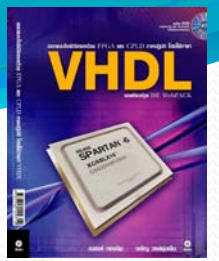
Use with signal.

Attribute	Returns	Kind of attribute
S'event	true if an event has just occurred on S in current cycle	Function
S'active	true if signal S is active in the current cycle	Function
S'last_value	value of S prior to latest change	Function
S'last_event	time at which S last changed	Function
S'last_active	time at which S last active	Function
S'delayed[(T)]	value of S delayed by T time units	Signal
S'stable[(T)]	true if no event on S over last T time units	Signal
S'quiet[(T)]	true if S quiet for T time units	Signal
S'transaction	bit value which toggles each time signal S changes	Signal

Normally, Use for simulation.

'event is use for synthesis.

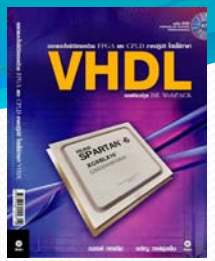
Type Attribute



- C’event : Transition of signal, Edge trig.
- C’event and C = ‘1’ : rising_edge(C)
- C’event and C = ‘0’ : falling_edge(C)

- Rising_edge and Falling_edge are predefined in IEEE Std 1164
- XST supported only : high, low, left, right, range, reverse_range, length, pos, ascending, event and last_value
- Other are ignored.

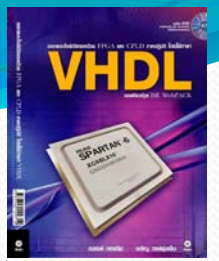
User Defined Attribute



- User defined attribute is user constraints
- Later explain

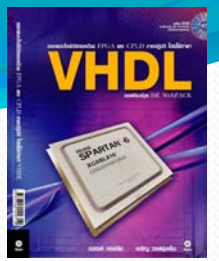
VHDL Coding

VHDL Statements



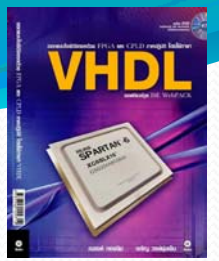
- **Concurrent statement**
  - **All lines work in parallel.**
- **Sequential statement**

Concurrent Statements



- **Concurrent statements (Dataflow descriptions)**
  - **Simple signal assignment statement**
  - **Selected signal assignment statement**
  - **Conditional signal assignment statement**
  - **Component instantiation statement**
  - **Generate statement**
  - **Block statement**
  - **Process statement**
  - **Concurrent procedure calls**

Simple signal assignment statement



- **Assign value to signal or port.**

[label :] TARGET <= EXPRESSION ;

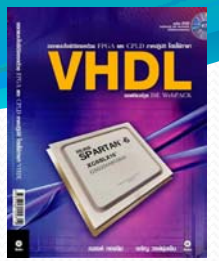
C <= A and B ;

X <= “00000000” ;

X <= (others => ‘0’) equal to X <= “00000000”

X <= (others => ‘1’) equal to X <= “11111111”

Selected signal assignment statement



- **Assign EXPRESSION to TARGET**

  when CHOICE match CHOICE_EXPRESSION.

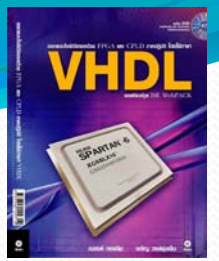
[label :] with CHOICE_EXPRESSION select

TARGET <= { EXPRESSION when CHOICE { | CHOICE}, }

EXPRESSION when CHOICE { | CHOICE};

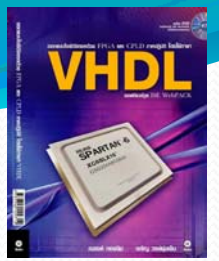
- **{...} = Multiple existed or not**
- **[...] = Existed or not**

Selected signal assignment statement



- **Static expression - - > 3**
- **Static range - - > 7 downto 4**
- **Multi-value - - > “0001” | “1000”**
- **Other left value - - > “OTHERS”**

Conditional signal assignment statement



- **Assign EXPRESSION to TARGET**

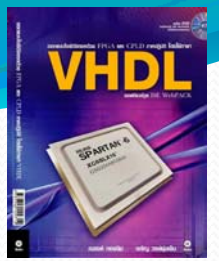
  when CONDITION is “true”.

[label :] TARGET <= { EXPRESSION when CONDITION else }

EXPRESSION ;

- **{...} = Multiple existed or not**
- **[...] = Existed or not**

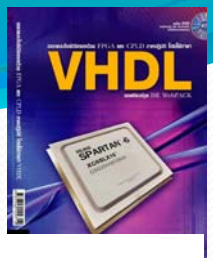
MUX 4 : 1



Control		Output	Remark
S1	S0	O	
0	0	A	O = A
0	1	B	O = B
1	0	C	O = C
1	1	D	O = D

Truth table

MUX 4 : 1



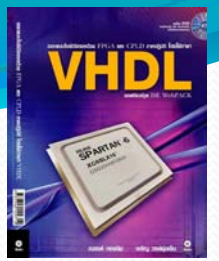

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12  begin
13        SEL <= S1&S0;          -- Concatenation
14     with SEL select
15        O <=  A when "00",
16              B when "01",
17              C when "10",
18              D when others; -- SEL="11"
19  end BEHAVIORAL;
```

ต้องประกาศใช้ signal เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิตโดยใช้ “&” (Concatenation)

Selected signal assignment, MUX 4 : 1

MUX 4 : 1



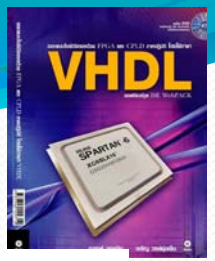

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12  begin
13      SEL <= S1&S0;                    -- Concatenation
14        O <=  A when SEL="00" else
15              B when SEL="01" else
16              C when SEL="10" else
17              D;                       -- SEL="11"
18  end BEHAVIORAL;
```

ต้องประกาศใช้ signal เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิตโดยใช้ “&” (Concatenation)

Conditional signal assignment, MUX 4 : 1

MUX 4 : 1

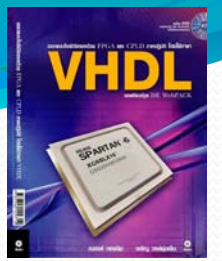



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11  begin
12     O <=  A when (S1='0' and S0='0') else
13           B when (S1='0' and S0='1') else
14           C when (S1='1' and S0='0') else
15           D;                                --(S1='1' and S0='1')
16  end BEHAVIORAL;
```

Conditional signal assignment (No concatenation),

MUX 4 : 1

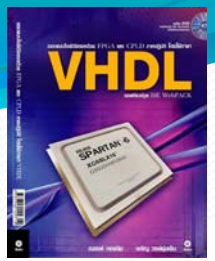
MUX 4 : 1 (Bus)



Control S	Output O	Remark
“00”	A	O = A
“01”	B	O = B
“10”	C	O = C
“11”	D	O = D

Truth table

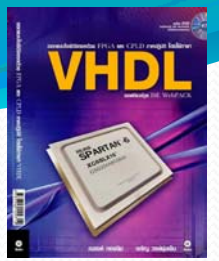
MUX 4 : 1 (Bus)




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D : in  STD_LOGIC;
 7             S : in STD_LOGIC_VECTOR(1 downto 0);
 8             O : out  STD_LOGIC);
 9  end MUX4TO1;
10
11  architecture BEHAVIORAL of MUX4TO1 is
12     begin
13        with S select
14        O <=  A when "00",
15              B when "01",
16              C when "10",
17              D when others;   -- S="11"
18  end BEHAVIORAL;
```

Selected signal assignment, MUX 4 : 1 (Bus)

MUX 4 : 1 (Bus)



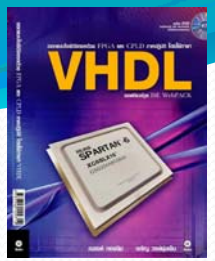

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D : in  STD_LOGIC;
 7             S : in STD_LOGIC_VECTOR(1 downto 0);
 8             O : out  STD_LOGIC);
 9  end MUX4TO1;
10
11  architecture BEHAVIORAL of MUX4TO1 is
12  begin
13     O <=  A when S="00" else
14           B when S="01" else
15           C when S="10" else
16           D;                    -- S="11"
17  end BEHAVIORAL;
```

Conditional signal assignment, MUX 4 : 1 (Bus)

VHDL

#4

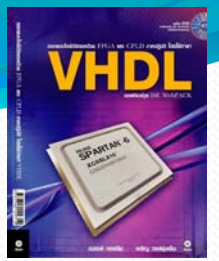
Component



Hierarchical model or Structural model.

- **Component declaration.**
  - **Declare the component before use.**
- **Component instantiation statement.**
  - **Create the declared component.**

Component Declaration

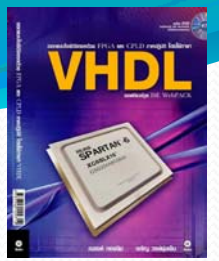


- **Before using component.**
- **Declare in architecture or package.**

```
component   COMPONENT_NAME
                    [ generic (GENERIC_DECLARATIONS) ]
                    [ port (PORT_DECLARATIONS) ]
end component ;
```

- **Generic and Port use the same format as Entity**

Component Instantiation Statement

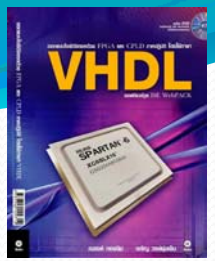


- **Concurrent statement.**
- **Create declared component as circuit.**

```
INSTANCE_NAME  :  COMPONENT_NAME
   [ generic map ( GENERIC_NAME => VALUE { , GENERIC_NAME => VALUE } ) ]
   port map ( [PORT_NAME =>]  SIGNAL_NAME { , [PORT_NAME =>]  SIGNAL_NAME } );
```

- **Generic and Port use the same format as Entity**

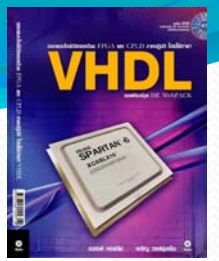
Positional and Named Association



[PORT_NAME =>] is the name of input/out of the component.

- **Positional association**
  - **No PORT_NAME.**
  - **SIGNAL_NAME must place in exactly order.**
- **Named association**
  - **Must have PORT_NAME.**
  - **Order in not necessary.**

3 Bits adder using component




```
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ADDER3BIT_1DIGIT is
6      Port ( A,B : in  STD_LOGIC_VECTOR (2 downto 0);
7             Y : out  STD_LOGIC_VECTOR (6 downto 0));
8  end ADDER3BIT_1DIGIT;
9
10 architecture Behavioral of ADDER3BIT_1DIGIT is
11    component ADDER_2BIT
12        generic ( N : integer := 2);
13        port ( A,B : in  STD_LOGIC_VECTOR (N-1 downto 0);
14               C : out  STD_LOGIC_VECTOR (N downto 0));
15    end component;
16    component HEX_TO_7SEGMENT
17        Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
18               Y : out  STD_LOGIC_VECTOR (6 downto 0));
19    end component;
20    signal C : STD_LOGIC_VECTOR (3 downto 0);
21 begin
22 ----------------------------IC1=ADDER3BIT--------------------------
23 IC1 : ADDER_2BIT  generic map (N => 3)      --ADDER_2BIT=>ADDER_3BIT
24                   port map (A,B,C);         --Positional association
25
26 ----------------------IC2=7SEGMENT DECODER----------------------
27 IC2 : HEX_TO_7SEGMENT port map (A => C, Y => Y);--Named association
28
29 end Behavioral;
```

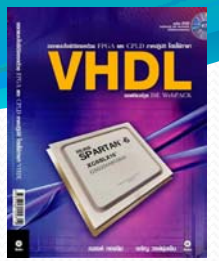
บรรทัด 11-19 เป็นการประกาศใช้คอมโพแนนต์

Signal C ใช้ต่อจากเอาต์พุต IC1 ไปที่อินพุต IC2

INSTANCE_NAME

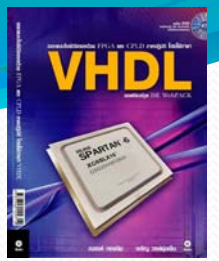
บรรทัดที่ 23-24 และ 27 เป็นคำสั่งใช้คอมโพแนนต์

Generate Statement



- **For/Generate**
  - **Suitable for repeatable statement.**
  - **Suitable for duplication circuit.**
- **If/Generate**
  - **Operate only condition is TRUE.**

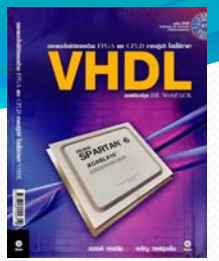
Generate Statement




```
generate_label : for INDEX in DISCRETE_RANGE generate
                    {CONCURRENT_STATEMENT}
                end generate [ generate_label ] ;
```

```
generate_label : if CONDITION generate
                    {CONCURRENT_STATEMENT}
                end generate [ generate_label ] ;
```

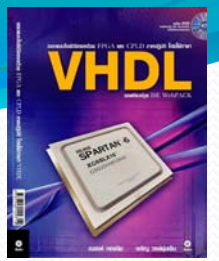
1 Bit full adder



Input			Output	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

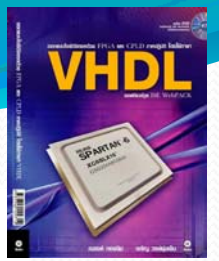
Truth table

1 Bit full adder




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity FULL_ADDER_1BIT is
 6      Port ( A,B,Cin : in  STD_LOGIC;
 7             Cout,Sum : out  STD_LOGIC);
 8  end FULL_ADDER_1BIT;
 9
10  architecture Behavioral of FULL_ADDER_1BIT is
11     signal X : STD_LOGIC_VECTOR(2 downto 0);
12  begin
13      X <= A&B&Cin;
14    Sum <= '1' when (X="001" or X="010" or X="100" or X="111") else
15           '0';
16   Cout <= '1' when (X="011" or X="101" or X="110" or X="111") else
17           '0';
18  end Behavioral;
```

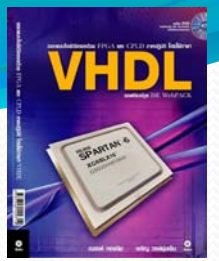
4 Bits adder using Generate




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity ADDN_FOR_GEN is
 6      Port ( A,B : in  STD_LOGIC_VECTOR (3 downto 0);
 7             Cin : in  STD_LOGIC;
 8             Cout : out  STD_LOGIC;
 9             Sum : out  STD_LOGIC_VECTOR (3 downto 0));
10  end ADDN_FOR_GEN;
11
12  architecture Behavioral of ADDN_FOR_GEN is
13     component FULL_ADDER_1BIT
14         Port ( A,B,Cin : in  STD_LOGIC;
15                Cout,Sum : out  STD_LOGIC);
16     end component;
17     signal C_tmp : STD_LOGIC_VECTOR (4 downto 0);
18  begin
19
20     C_tmp(0) <= Cin;
21
22  ADDN_GEN : for I in  3 downto 0  generate
23        ADDN_BIT : FULL_ADDER_1BIT
24                   port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
25             end generate;
26
27         Cout <= C_tmp(4);
28
29  end Behavioral;
```

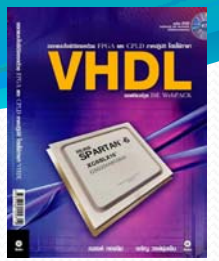
generate_label ซึ่งควรตั้งชื่อให้สื่อความหมายด้วย

N Bits adder using Generate



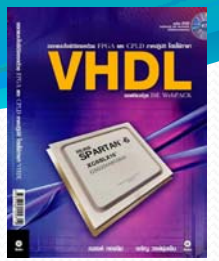

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity ADDN_FOR_GEN is
 6      generic (N : integer := 4);
 7      Port ( A,B : in  STD_LOGIC_VECTOR (N-1 downto 0);
 8             Cin : in  STD_LOGIC;
 9             Cout : out  STD_LOGIC;
10             Sum : out  STD_LOGIC_VECTOR (N-1 downto 0));
11  end ADDN_FOR_GEN;
12
13  architecture Behavioral of ADDN_FOR_GEN is
14     component FULL_ADDER_1BIT
15         Port ( A,B,Cin : in  STD_LOGIC;
16                Cout,Sum : out  STD_LOGIC);
17     end component;
18     signal C_tmp : STD_LOGIC_VECTOR (N downto 0);
19  begin
20     C_tmp(0) <= Cin;
21
22  ADDN_GEN : for I in  A'range  generate --A'range = N-1 downto 0
23        ADDN_BIT : FULL_ADDER_1BIT
24                   port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
25             end generate;
26
27         Cout <= C_tmp(C_tmp'high);       --C_tmp'high = N
28  end Behavioral;
```

2 – 32 Bits adder



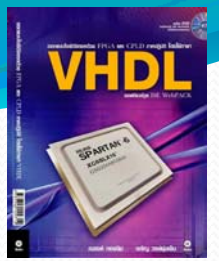

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity ADDN_IF_GEN is
 6      generic (N : integer := 4);
 7      Port ( A,B : in  STD_LOGIC_VECTOR (N-1 downto 0);
 8             Cin : in  STD_LOGIC;
 9             Cout : out  STD_LOGIC;
10             Sum : out  STD_LOGIC_VECTOR (N-1 downto 0));
11  end ADDN_IF_GEN;
12
13  architecture Behavioral of ADDN_IF_GEN is
14     component FULL_ADDER_1BIT
15         Port ( A,B,Cin : in  STD_LOGIC;
16                Cout,Sum : out  STD_LOGIC);
17     end component;
18     signal C_tmp : STD_LOGIC_VECTOR (N downto 0);
19  begin
20     C_tmp(0) <= Cin;
21  ------------------------------IF/generate loop-----------------------
22  ADDN_2TO32 : if (N>=2 and N<=32) generate
23  ----------------------------For/generate loop-----------------------
24        ADDN_GEN : for I in  A'range  generate --A'range = N-1 downto 0
25           ADDN_BIT : FULL_ADDER_1BIT
26                      port map(A(I),B(I),C_tmp(I),C_tmp(I+1),Sum(I));
27                   end generate;
28  ---------------------------END for/generate loop---------------------
29              end generate;
30  ---------------------------END if/generate loop----------------------
31         Cout <= C_tmp(C_tmp'high);             --C_tmp'high = N
32  end Behavioral;
```

Block Statement



- Partition groups of concurrent statement.
- For manage and verify.
- Can have many blocks in code.
- Nested block is acceptable.
- If have the same object name in nested block, use local name firstly.
- Simple and Guarded block
- Only Simple is allow by XST

Simple Block

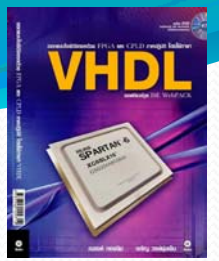



```
label  :  block
                    {BLOCK_DECLARATIVE_PART}
          begin
                    {CONCURRENT_STATEMENT}
end  block [ label ] ;
```

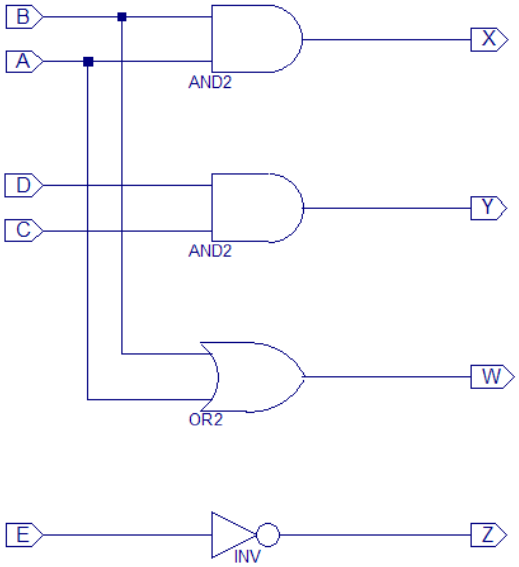
1) **BLOCK_DECLARATIVE_PART**
   - use clause
   - Subprogram declaration and Subprogram body
   - Type declaration and/or Subtype declaration
   - Constant declaration
   - Signal declaration
   - Component declaration

2) **CONCURRENT_STATEMENT**

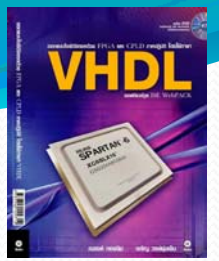
Block Statement




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity EX_BLOCK is
 6      Port ( A,B,C,D,E : in  STD_LOGIC;
 7             W,X,Y,Z : out  STD_LOGIC);
 8  end EX_BLOCK;
 9
10  architecture Behavioral of EX_BLOCK is
11  begin
12
13     Z <= not E;
14
15  B1: block
16     signal S: STD_LOGIC;     --Declaration of signal S in block B1
17  begin
18     S <= A and B;            --signal S from B1
19     B2: block
20        signal S: STD_LOGIC; --Declaration of signal S in block B2
21     begin
22           S <= C and D;      --signal S from B2 (Declaration of S from B1 is overrided.)
23        B3: block
24        begin
25           Y <= S;           --signal S from B2 (Declaration of S from B1 is overrided.)
26        end block B3;
27     end block B2;
28        X <= S;              --signal S from B1
29  end block B1;
30
31     W <= A or B;
32
33  end Behavioral;
```



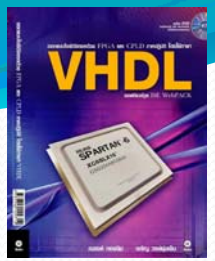
Process Statement




```
[ label : ] process [ (SENSITIVITY  LIST) ]
                {PROCESS_DECLARATIVE_PART}
        begin
                {SEQUENTIAL_STATEMENT}
        end process [ label ] ;
```

1) **PROCESS_DECLARATIVE_PART**
   - use clause
   - Subprogram declaration and Subprogram body
   - Type declaration and/or Subtype declaration
   - Constant declaration
   - Variable declaration
2) **SENSITIVITY LIST; Every input that direct effect to the process**
3) **SEQUENTIAL_STATEMENT**

Concurrent Procedure Calls

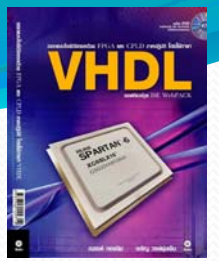


PROCEDURE_NAME [([NAME =>] EXPRESSION

{ ,[NAME =>] EXPRESSION })] ;

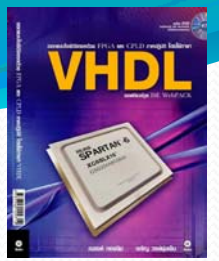
- **Call subprogram inside architecture or block**
- **Later explain**

Sequential Statement



- Orders are necessary.
- Only in Process, Function or Procedure.
- Every process is concurrent statement.
- Describe behavior of the circuit.

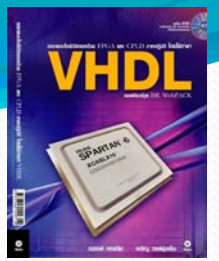
Sequential Statement



Sequential Statement (Behavioral descriptions)

- Signal assignment statement
- Variable assignment statement
- if statement
- case statement
- Wait statement
- Loop statement

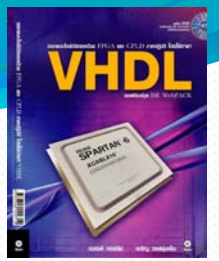
Signal Assignment Statement



- **Assign value to signal or port.**

TARGET <= EXPRESSION ;

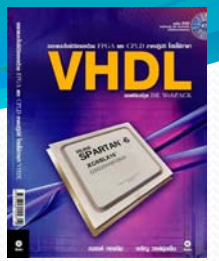
Variable Assignment Statement



- **Assign value to variable in Process, Function or Procedure only.**

TARGET := EXPRESSION ;

If Statement

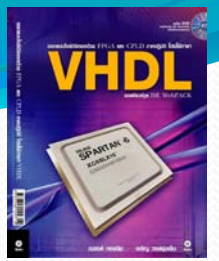


- **Look like Conditional signal assignment.**
- **Only in Process.**
- **Priorities of CONDITION are in order.**

```
if      CONDITION  then {SEQUENTIAL_STATEMENT}
{ elsif   CONDITION  then {SEQUENTIAL_STATEMENT} }
[ else {SEQUENTIAL_STATEMENT} ]
end if;
```

- **If no “else”, Latch must be create to hold previous output.**

Case Statement

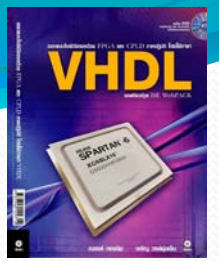


- **Look like Selected signal assignment.**
- **Only in Process.**
- **Priorities of CHOICES are NOT in order.**

```
case  EXPRESSION  is
        when CHOICES => {SEQUENTIAL_STATEMENT}
      { when CHOICES => {SEQUENTIAL_STATEMENT} }
end case;
```

If no “OTHERS”, Latch must be create to hold previous output.

MUX 4 : 1 using If Statement




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12  begin
13     SEL <= S1&S0;            -- Concatenation
14     process(A,B,C,D,SEL)
15        begin
16           if    SEL="00" then  O <= A;
17           elsif SEL="01" then  O <= B;
18           elsif SEL="10" then  O <= C;
19           else   O <= D;    -- SEL="11"
20           end if;
21     end process;
22  end BEHAVIORAL;
```

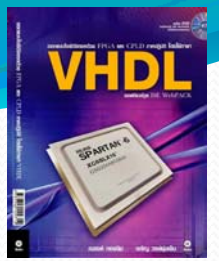
ต้องประกาศใช้ signal SEL เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิต โดยใช้ “&” (Concatenation)

อินพุต A, B, C, D และ SEL ที่มีการเปลี่ยนแปลงแล้วมีผลต่อเอาต์พุต O โดยตรง (ทันที) จึงต้องเขียนอินพุต A, B, C, D และ SEL ไว้ใน Sensitivity list

With concatenation

MUX 4 : 1 using If Statement

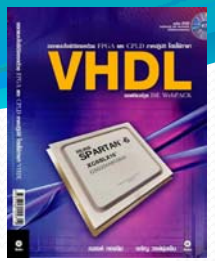



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11  begin
12     process(A,B,C,D,S1,S0)
13        begin
14           if    (S1='0' and S0='0') then  O <= A;
15           elsif (S1='0' and S0='1') then  O <= B;
16           elsif (S1='1' and S0='0') then  O <= C;
17           else   O <= D;    -- (S1='1' and S0='1')
18           end if;
19     end process;
20  end BEHAVIORAL;
```

อินพุต A, B, C, D, S1 และ S0 ที่มีการเปลี่ยนแปลงแล้วมีผลต่อเอาต์พุต O โดยตรง (ทันที) จึงต้องเขียนอินพุต A, B, C, D, S1 และ S0 ไว้ใน Sensitivity list

Without concatenation

MUX 4 : 1 using Case Statement

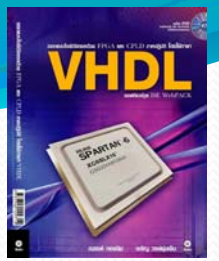



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity MUX4TO1 is
 6      port ( A,B,C,D,S1,S0 : in  STD_LOGIC;
 7             O : out  STD_LOGIC);
 8  end MUX4TO1;
 9
10  architecture BEHAVIORAL of MUX4TO1 is
11     signal SEL : STD_LOGIC_VECTOR(1 downto 0);
12  begin
13     SEL <= S1&S0;                    -- Concatenation
14     process(A,B,C,D,SEL)
15        begin
16           case SEL is
17              when "00"    =>  O <= A;
18              when "01"    =>  O <= B;
19              when "10"    =>  O <= C;
20              when others  =>  O <= D; -- SEL="11"
21           end case;
22     end process;
23  end BEHAVIORAL;
```

ต้องประกาศใช้ signal เนื่องจาก SEL ไม่มีชื่อใน port ของ entity

การรวมชนิดข้อมูลเป็นแบบหลายบิตโดยใช้ “&” (Concatenation)

อินพุต A, B, C, D และ SEL มีการเปลี่ยนแปลงแล้วมีผลต่อเอาต์พุต O โดยตรง (ทันที) จึงต้องเขียนอินพุต A, B, C, D และ SEL ไว้ใน Sensitivity list



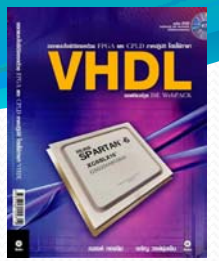
D Flip-Flop with Asynchronous Clear

```
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity DFF_CLR is
6       port ( D,C,CLR : in  STD_LOGIC;
7              Q : out  STD_LOGIC);
8   end DFF_CLR;
9
10  architecture BEHAVIORAL of DFF_CLR is
11  begin
12  process(C,CLR)
13     begin
14        if    CLR='1' then Q <= '0';
15        elsif (C'event and C='1') then Q <= D;
16        end if;
17  end process;
18  end BEHAVIORAL;
```

การเปลี่ยนแปลงของอินพุต C และ CLR นั้น จะมีผลต่อเอาต์พุต Q โดยตรง (ทันที) จึงต้องเขียน C และ CLR ไว้ใน Sensitivity list

C'event and C= '1' คือ C ทริกด้วยขอบขาขึ้น หรือขอบบวก ซึ่งจะใส่วงเล็บหรือไม่ใส่ก็ได้

D Flip-Flop with Synchronous Clear

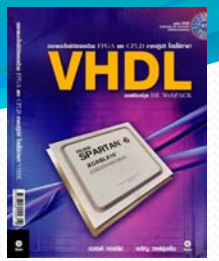



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DFF_RESET is
 6      port ( D,C,RESET : in  STD_LOGIC;
 7             Q : out  STD_LOGIC);
 8  end DFF_RESET;
 9
10  architecture BEHAVIORAL of DFF_RESET is
11  begin
12  process(C)
13     begin
14        if (C'event and C='1') then
15                if RESET='1' then Q <= '0';
16                else Q <= D;
17                end if;
18        end if;
19  end process;
20  end BEHAVIORAL;
```

การ RESET ไม่มีผลโดยตรงต่อเอาต์พุต Q (ทันที) เพราะต้องรอการทริกจาก C แต่การทริกด้วย C จะมีผลต่อเอาต์พุต Q โดยตรง (ทันที) ดังนั้นจึงต้องเขียน C เพียงตัวเดียวเท่านั้นไว้ใน Sensitivity list

C'event and C= '1' คือ C ทริกด้วยขอบขาขึ้นหรือขอบบวก ซึ่งจะใส่วงเล็บหรือไม่ใส่ก็ได้

4 bits Countup Binary Counter



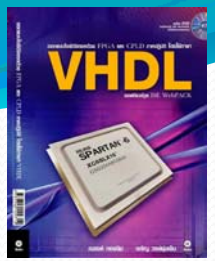

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity COUNTER4BIT is
 7      port ( C,CLR : in  STD_LOGIC;
 8             Q : out  STD_LOGIC_VECTOR (3 downto 0));
 9  end COUNTER4BIT;
10
11  architecture Behavioral of COUNTER4BIT is
12     signal Q_temp : STD_LOGIC_VECTOR (3 downto 0);
13  begin
14     process(C,CLR)
15        begin
16           if CLR='1' then Q_temp <= "0000";
17           elsif C'event and C='1' then Q_temp <= Q_temp + 1;
18           end if;
19     end process;
20           Q <= Q_temp;
21  end Behavioral;
```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

signal Q_temp ไม่ใช่ Port จึงไม่มีความจำเป็นต้องใช้ Mode “buffer”

กำหนดค่า (Assign) Q_temp ซึ่งเป็น Signal ให้กับ Q นั้นต้องทำภายนอก Process เนื่องจากการ Update ค่าของ Signal จะกระทำเมื่อจบ Process

Decade Countup Binary Counter

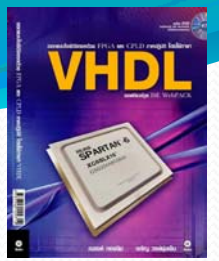



```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity COUNTER10UP is
 7      Port ( C,CLR : in  STD_LOGIC;
 8             Q : out  STD_LOGIC_VECTOR(3 downto 0));
 9  end COUNTER10UP;
10
11  architecture Behavioral of COUNTER10UP is
12     signal QT : STD_LOGIC_VECTOR (3 downto 0);
13  begin
14     process(C,CLR)
15        begin
16           if  CLR='1' then  QT <= "0000";
17           elsif  C'event and C='1'  then
18                 if   QT >= 9  then QT <= "0000";
19                 else QT <= QT + 1;
20                 end if;
21           end if;
22     end process;
23        Q <= QT;
24  end Behavioral;
```

เรียกใช้ Package ชื่อ std_logic_unsigned เพื่อให้ชนิดข้อมูล std_logic_vector สามารถใช้กับ “+” ได้

เมื่อค่า QT >= 9 (หรือ Q >= “1001”) แล้วมีสัญญาณนาฬิกา C ทริกด้วยขอบขาขึ้นจะทำให้ QT มีค่าเป็น “0000”

Wait Statement

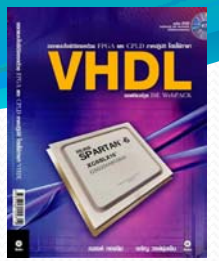


- **Use in Processes without sensitivity lists.**
- **For stop process operation.**

```
wait  [ on  SENSITIVITY_LIST ] ;
      [ until  CONDITION ] ;
      [ for  TIME¬_EXPRESSION ] ;
```

- **Wait on SENSITIVITY_LIST; wait till event occur.**
- **Wait until CONDITION; wait till condition is true.**
- **Wait for TIME_EXPRESSION; time delay.**

Wait Statement

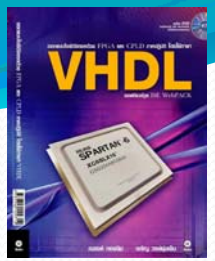


- **Use in Processes without sensitivity lists.**
- **For stop process operation.**

```
wait  [ on  SENSITIVITY_LIST ] ;
      [ until  CONDITION ] ;
      [ for  TIME¬_EXPRESSION ] ;
```

- **Wait on, Wait until and Wait for.**

Wait Statement as Rising Edge Clock

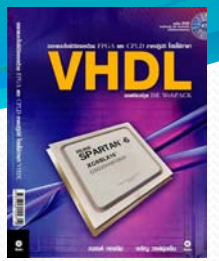


- **Same operation as Rising_Edge(SIGNAL_CLOCK).**

```
wait on  CLK  until  CLK = '1';
wait until  CLK =  '1';
wait until  CLK'event  and  CLK =  '1';
```

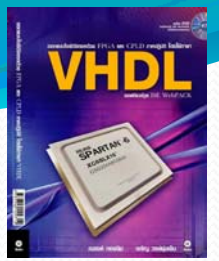
Must be first statement in the process.

D Flip-Flop with Synchronous Clear



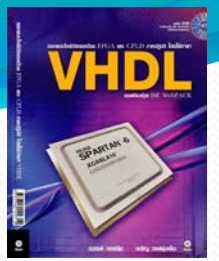

```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DFF_RESET is
 6      port ( D,C,RESET : in  STD_LOGIC;
 7             Q : out  STD_LOGIC);
 8  end DFF_RESET;
 9
10  architecture BEHAVIORAL of DFF_RESET is
11  begin
12  process(C)--Processes with sensitivity lists
13     begin
14        if (C'event and C='1') then
15                if RESET='1' then Q <= '0';
16                else Q <= D;
17                end if;
18        end if;
19  end process;
20  end BEHAVIORAL;
```

D Flip-Flop with Synchronous Clear




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity DFF_RESET is
 6      port ( D,C,RESET : in  STD_LOGIC;
 7             Q : out  STD_LOGIC);
 8  end DFF_RESET;
 9
10  architecture BEHAVIORAL of DFF_RESET is
11  begin
12  process     --Processes without sensitivity lists
13  begin
14     wait until  C'event and C='1';
15           if RESET='1' then Q <= '0';
16           else Q <= D;
17           end if;
18  end process;
19  end BEHAVIORAL;
```

Loop Statement

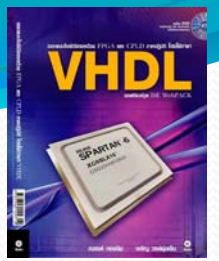


- **Like Generate statement, but use in sequencial.**
- **For repeating the operation.**

```
[ loop_label : ]  [ ITERATION_SCHEME ]  loop
                        {SEQUENTIAL_STATEMENT}
                        { next [ loop_label ]  [ when condition ] ; }
                        { exit  [ loop_label ]  [ when condition ] ; }
                end  loop  [ loop_label ] ;
```

1) **ITERATION_SCHEME; Loop, For, While.**
2) **next; skip to next round(iteration)**
3) **exit; end loop**

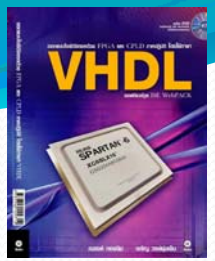
Basic Loop



- **Loop without ITERATION_SCHEME.**
- **End with “next” and “exit”.**
- **At least one “wait” in the loop.**

```
[ loop_label : ]   loop
                        {SEQUENTIAL_STATEMENT}
                   end  loop  [ loop_label ] ;
```

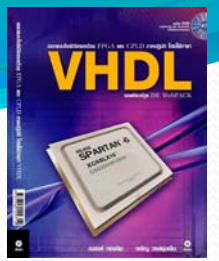
While/Loop



- **Loop while CONDITION is TRUE.**
- **Can not synthesis to circuit.**

```
[ loop_label : ]  while   CONDITION   loop
                              {SEQUENTIAL_STATEMENT}
                    end  loop  [ loop_label ] ;
```

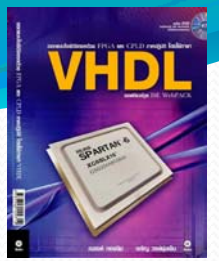
For/Loop



- **Loop while INDEX_NAME is in DISCRETE_RANGE.**
- **No “Wait” in the loop.**

```
[ loop_label : ]  INDEX_NAME  in  DISCRETE_RANGE  loop
                        {SEQUENTIAL_STATEMENT}
              end  loop  [ loop_label ] ;
```

4 Bits Shift Left Register




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity SHIFT_REGISTER_4BIT is
 6      Port ( C,D_IN : in  STD_LOGIC;
 7             Q : out  STD_LOGIC_VECTOR (3 downto 0));
 8  end SHIFT_REGISTER_4BIT;
 9
10  architecture Behavioral of SHIFT_REGISTER_4BIT is
11     signal QT :  STD_LOGIC_VECTOR (3 downto 0);
12  begin
13     process(C) --Processes with sensitivity lists
14        begin
15           if (C'event and C='1') then
16                    QT(0) <= D_IN;
17                    for i in 2 downto 0 loop
18                       QT(i+1) <= QT(i);
19                    end loop;
20           end if;
21     end process;
22           Q <= QT;
23  end Behavioral;
```

VHDL

Package, Library, Subprogram
ROM, RAM

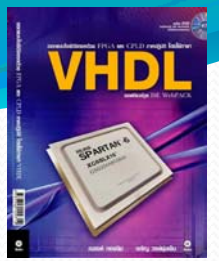
VHDL

#6

VHDL

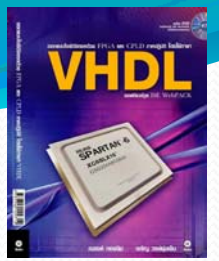
Testbench and TEXTIO

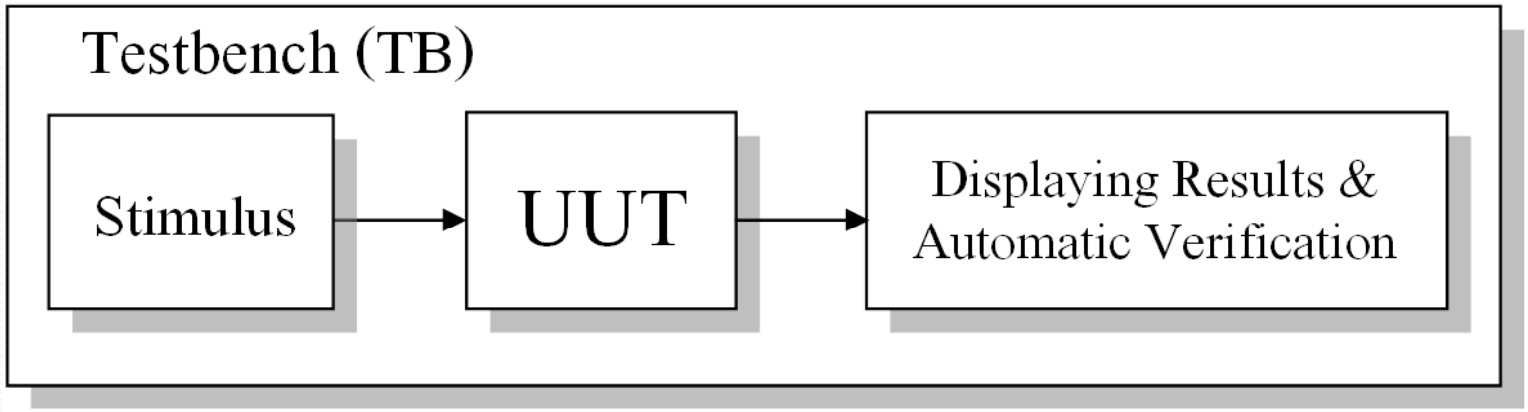
Testbench



- Use to test circuit with simulation technique.
- Test bench will provide test vector to testing component, Unit under test (UUT) or Design under test (DUT) .
- Output will be show in waveform, error report or data file.
- Testbench entity has no port (internal work only).

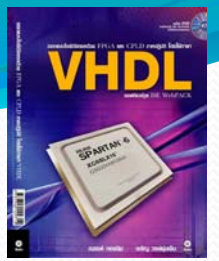
Testbench





- **Compare actual results to expected results are complex.**
- **Report error on transcript windows in Xilinx should be better.**

Testbench Basic Statement



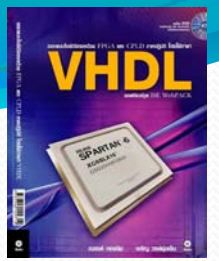
- **Wait : Stop the process.**
- **Wait for : Stop the process for the time “wait for 100 ns;”**
- **Assert : Working when condition is “FALSE”**

  working with “Report” and “Severity”

```
assert  CONDITION_EXPRESSION
        report  TEXT_STRING
        severity  SEVERITY_LEVEL;
```

SEVERITY_LEVEL : NOTE, WARNING, ERROR and FAILURE

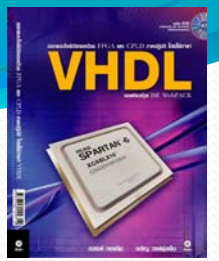
Testbench




```
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4
 5  entity andgate2 is
 6      Port ( A,B : in  STD_LOGIC;
 7             Y : out  STD_LOGIC);
 8  end andgate2;
 9
10  architecture Behavioral of andgate2 is
11  begin
12  Y <= A and B;
13  end Behavioral;
```

And gate 2 inputs

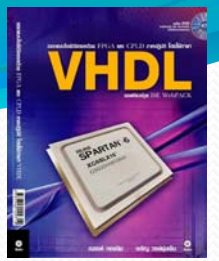
Testbench

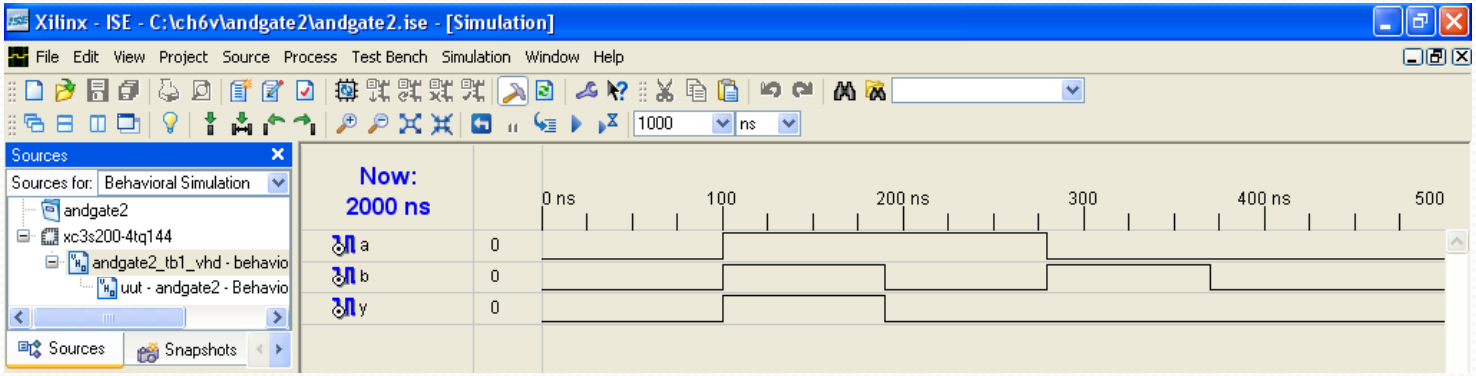



```
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.ALL;
 4
 5  ENTITY andgate2_tb1_vhd IS
 6  END andgate2_tb1_vhd;
 7
 8  ARCHITECTURE behavior OF andgate2_tb1_vhd IS
 9     COMPONENT andgate2   -- Component Declaration for the UUT
10     PORT(A : IN std_logic;
11          B : IN std_logic;
12          Y : OUT std_logic);
13     END COMPONENT;
14     SIGNAL A :  std_logic := '0'; --Input
15     SIGNAL B :  std_logic := '0'; --Input
16     SIGNAL Y :  std_logic;        --Output
17  BEGIN
18
19     uut: andgate2 PORT MAP(A => A,B => B,Y => Y); -- Instantiate the UUT
20
21     tb : PROCESS
22     BEGIN
23        wait for 100 ns;
24        A <= '1';
25        B <= '1';
26        wait for 90 ns;
27        A <= '1';
28        B <= '0';
29        wait for 90 ns;
30        A <= '0';
31        B <= '1';
32        wait for 90 ns;
33        A <= '0';
34        B <= '0';
35        wait; -- will wait forever
36     END PROCESS tb;
37  END;
```

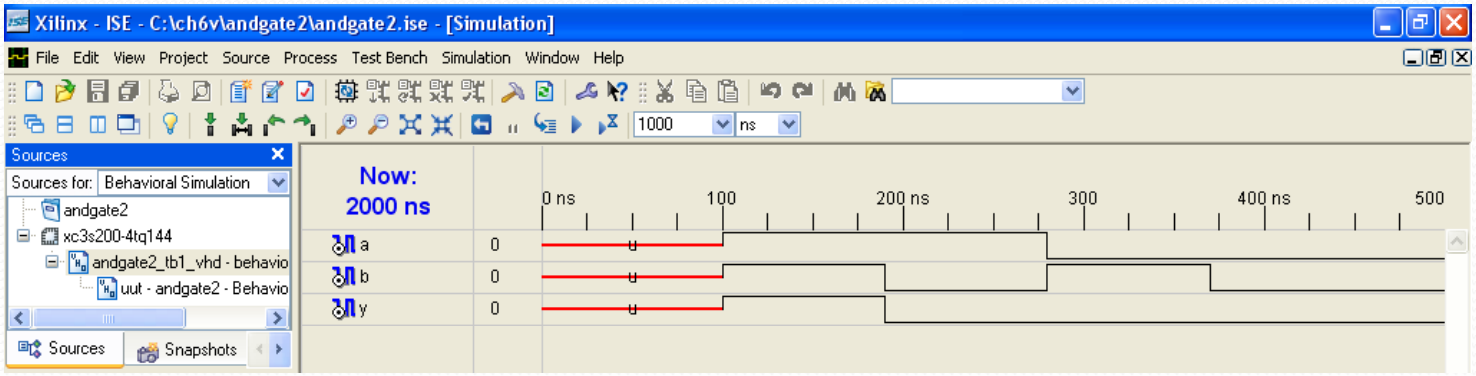
And gate 2 inputs

Testbench

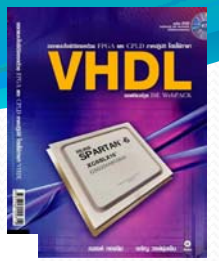




And gate 2 inputs

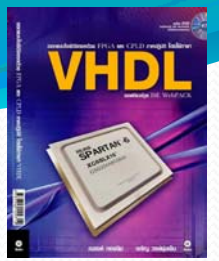


Testbench with Assert Statement



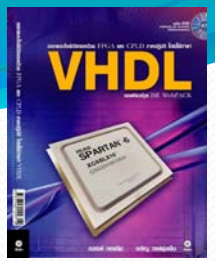

```
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.ALL;
 4
 5  ENTITY andgate2_tb1_vhd IS
 6  END andgate2_tb1_vhd;
 7
 8  ARCHITECTURE behavior OF andgate2_tb1_vhd IS
 9     COMPONENT andgate2                    -- Component Declaration for UUT
10        PORT (A : IN std_logic;
11              B : IN std_logic;
12              Y : OUT std_logic);
13     END COMPONENT;
14     SIGNAL A :  std_logic := '0';--Inputs
15     SIGNAL B :  std_logic := '0';--Inputs
16     SIGNAL C :  std_logic := '0';--Inputs
17     SIGNAL Y :  std_logic;       --Outputs
18  BEGIN
19
20     uut: andgate2 PORT MAP(A => A,B => B,Y => Y);-- Instantiate the UUT
21
22  tb : PROCESS
23        constant gatedelay : time := 10 ns;
24     BEGIN
25        wait for 100ns;
26        A <= '1';
27        B <= '1';
28        wait for gatedelay;
29        assert (Y = '1') report "Test1 failed!" severity ERROR;
30        wait for 90ns;
```

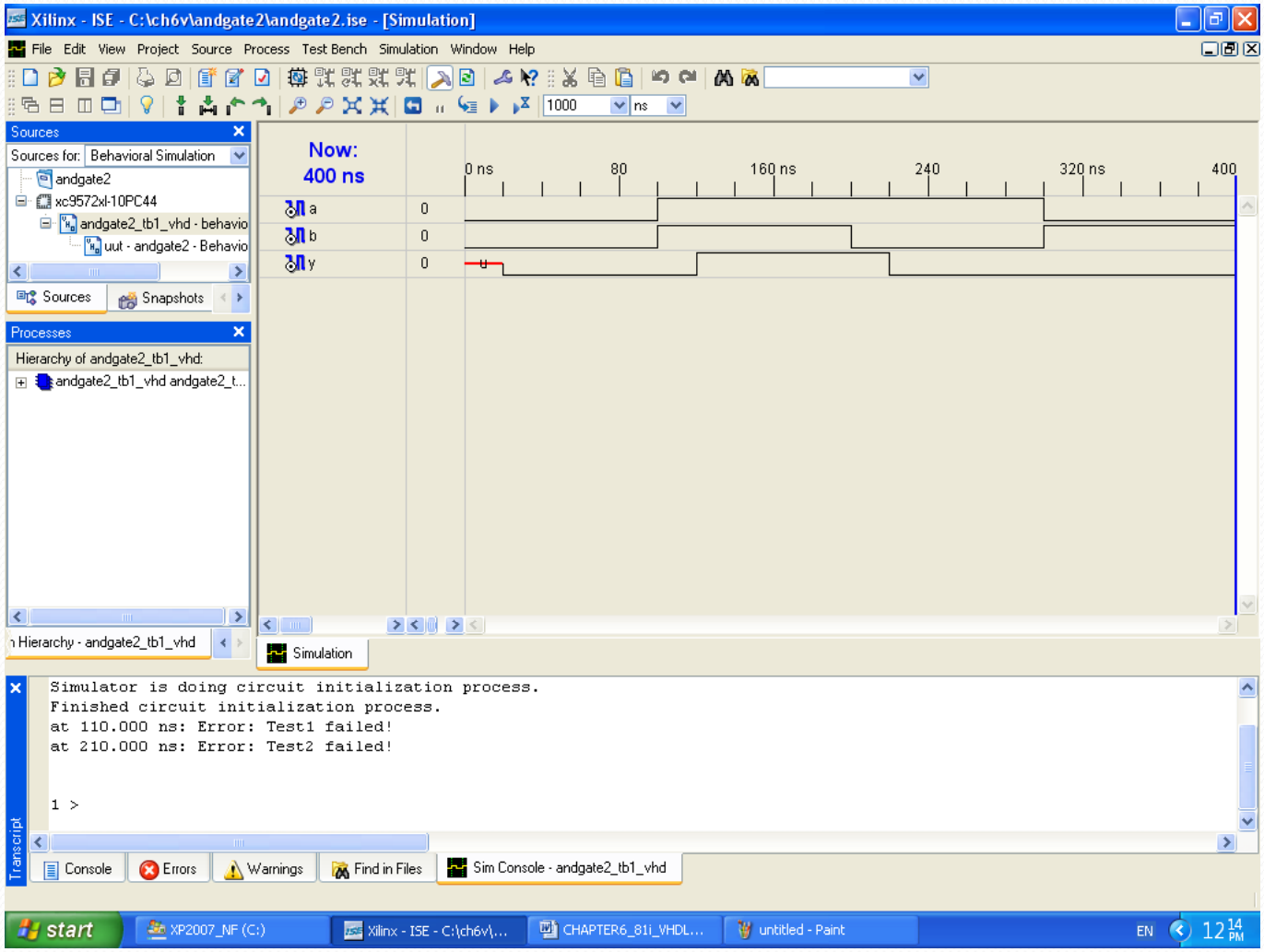
Testbench with Assert Statement



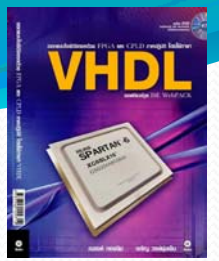

```
31          A <= '1';
32          B <= '0';
33          wait for gatedelay;
34          assert (Y = '0') report "Test2 failed!" severity ERROR;
35          wait for 90ns;
36          A <= '0';
37          B <= '1';
38          wait for gatedelay;
39          assert (Y = '0') report "Test3 failed!" severity ERROR;
40          wait for 90ns;
41          A <= '0';
42          B <= '0';
43          wait for gatedelay;
44          assert (Y = '0') report "Test4 failed!" severity ERROR;
45          wait; -- will wait forever
46      END PROCESS tb;
47      END behavior;
```

Testbench : Too much propagation delay



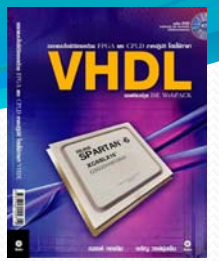


Testbench Construction



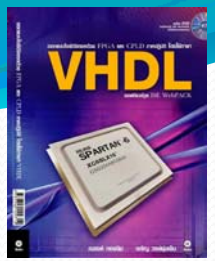
- Not synthesis circuit.
- Consists of 4 parts.
  - Entity declaration and Architecture (No ports).
  - Signal declaration; connect stimulus signal to UUT port.
  - Instantiation of Top-level design.
  - Provide stimulus; Gen clock, reset and test vectors.

Clock Generation



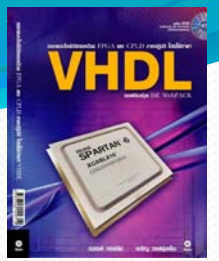

```
constant   Period  :  TIME := 200 ns;       - - Declare a clock period constant

Clock <=  not  Clock  after   Period / 2;   - - Clock generation


Clock <= not  Clock   after  100 ns;        - - Clock generation : Period =
                                                (100+100) ns, F = 5 MHz
```

Clock Generation



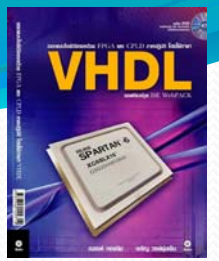

```
Process
    constant   Period  :  TIME  := 200 ns;
        - - Declare a clock period constant
begin
    Clock <= '1';
    wait  for (Period / 2);
    Clock <= '0';
    wait  for (Period / 2);
end process;
```

Stimulus Generation



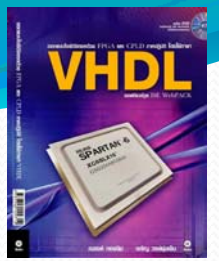

```
Stimulus1 : process
begin
        Reset <= '1';
        wait for 100 ns;
        CE    <= '1';
        Reset <= '0';
        wait for 800 ns;
        CE    <= '0';
        wait for 100 ns;
        wait ;
end  process Stimulus1;
```

Stimulus Generation



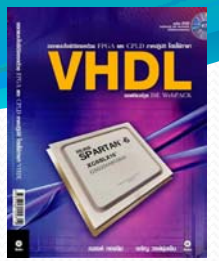

```
S <= "00",
     "01" after 30 ns,
    "11" after 120 ns,
    "01" after 150 ns;
```

Stimulus Generation Using TEXTIO



- Read text file for test vector and send to UUT.
- Using TEXTIO package.
- STD_LOGIC_TEXTIO package for std_logic and std_logic_vector.

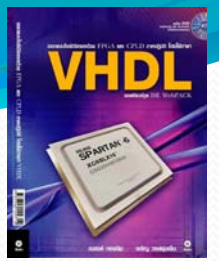
File Access



- **VHDL have no display statement, Using read and write file for input and output.**
- **Using TEXTIO package.**

```
type LINE is access STRING ;    - - A LINE is a pointer to a STRING value.
type TEXT is file of STRING ;   - - A file of variable-length ASCII records.
```

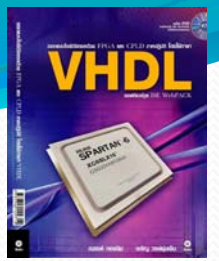
File Type Declaration



type FILE_TYPE_NAME is file of TYPE_NAME;

TYPE_NAME = Scalar type, Record type or Constrained array subtype

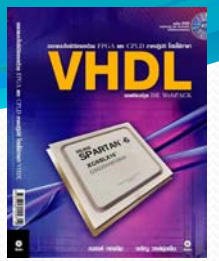
Object Declaration : File




```
file INPUT_OBJECT_FILE : text open READ_MODE is “INPUT_FILE.text” ;
                                                        - - VHDL93
file OUTPUT_OBJECT_FILE : text open WRITE_MODE is “OUTPUT_FILE.text” ;
                                                        - - VHDL93
file INPUT_OBJECT_FILE : text in is “INPUT_FILE.text” ;
                                                        - - VHDL87
file OUTPUT_OBJECT_FILE : text out is “OUTPUT_FILE.text” ;
                                                        - - VHDL87
```

- INPUT_FILE.text,OUTPUT_FILE.text = Text file.
- INPUT_OBJECT_FILE,OUTPUT_OBJECT_FILE = Temporary storage.

File Command



readline (INPUT_OBJECT_FILE, INPUT_LINE) - - Reads new line from temp file

read (INPUT_LINE, VALUE) ; - - Reads a new object from the line.

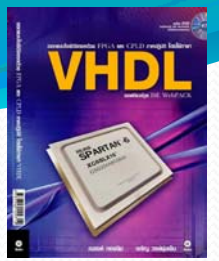
write (OUTPUT_LINE, VALUE) ; - - Writes a new object into the line.

writeline (OUTPUT_OBJECT_FILE, OUTPUT_LINE) - - Writes a new line to temp

fileendfile (FILE_NAME) ; - - Returns boolean true if the end of file is reached.

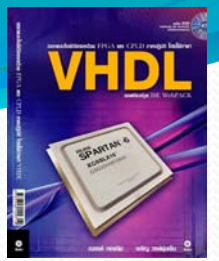
- **use STD.TEXTIO.all ; or use IEEE. STD_LOGIC_TEXTIO.all ;**

4 Bits Adder, Testbench Using File



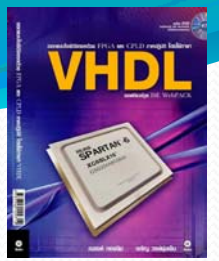

```
 2  LIBRARY ieee;
 3  USE ieee.std_logic_1164.ALL;
 4
 5  USE std.textio.all;
 6  USE ieee.std_logic_textio.all;
 7
 8  ENTITY ADDRE4BIT_TEXTIO_tb2_vhd IS
 9  END ADDRE4BIT_TEXTIO_tb2_vhd;
10
11  ARCHITECTURE behavior OF ADDRE4BIT_TEXTIO_tb2_vhd IS
12     -- Component Declaration for the Unit Under Test (UUT)
13     COMPONENT ADDER4BIT_TEXTIO
14     PORT(A : IN std_logic_vector(3 downto 0);
15          B : IN std_logic_vector(3 downto 0);
16          C : OUT std_logic_vector(4 downto 0));
17     END COMPONENT;
18     --Inputs
19     SIGNAL A :  std_logic_vector(3 downto 0) := (others=>'0');
20     SIGNAL B :  std_logic_vector(3 downto 0) := (others=>'0');
21     --Outputs
22     SIGNAL C :  std_logic_vector(4 downto 0);
23  BEGIN
24     -- Instantiate the Unit Under Test (UUT)
25     uut: ADDER4BIT_TEXTIO PORT MAP(A => A,B => B,C => C);
26     -- Stimulus
27     process
28     -- File declaration and variable declaration
29  --     FILE file_ABCin : TEXT IS IN "data_ABCin.txt";                --VHDL87
30         FILE file_ABCin : TEXT OPEN READ_MODE IS "data_ABCin.txt";    --VHDL93
31  --     FILE file_SUM : TEXT IS OUT "data_SUMout.txt";                --VHDL87
32         FILE file_SUM : TEXT OPEN WRITE_MODE IS "data_SUMout.txt";    --VHDL93
33         variable line_ABCin : LINE;
34         variable line_SUM : LINE;
35         variable Ain,Bin : std_logic_vector(3 downto 0);
```

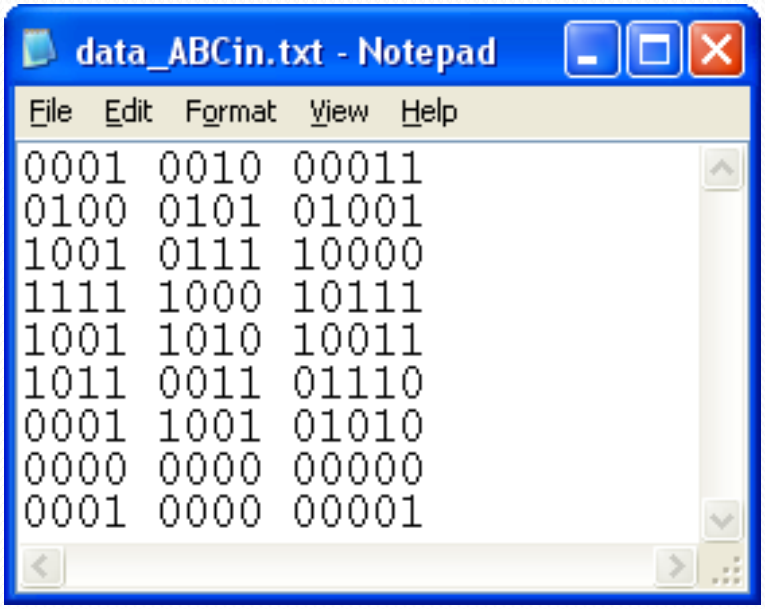
4 Bits Adder, Testbench Using File

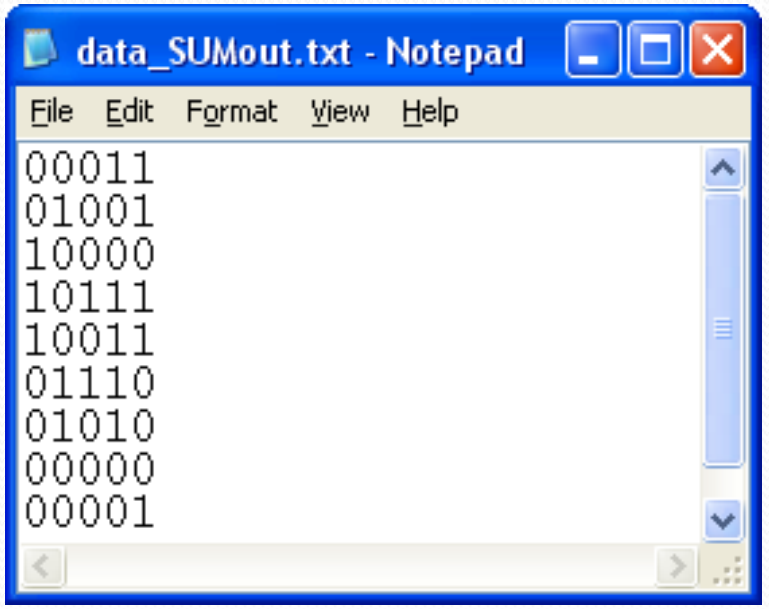



```
36        variable Cin,Cout : std_logic_vector(4 downto 0);
37     begin
38           while  NOT  ENDFILE(file_ABCin)  loop
39              READLINE(file_ABCin,line_ABCin);     --Get line of input file.
40              READ(line_ABCin,Ain);                --Get 1st operand.
41              READ(line_ABCin,Bin);                --Get 2nd operand.
42              READ(line_ABCin,Cin);                --Get expected result.
43              A <= Ain;
44              B <= Bin;
45              wait for 10 ns;
46              WRITE(line_SUM,C);                   --Save result to line.
47              WRITELINE(file_SUM,line_SUM);        --Write line to output file.
48              ASSERT  C = Cin  REPORT "Test failed!" SEVERITY error;
49              wait for 30 ns;
50           end loop;
51           wait; -- will wait forever
52     end process;
53  END behavior;
```

4 Bits Adder, Testbench Using File



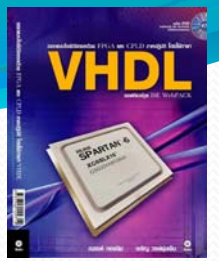


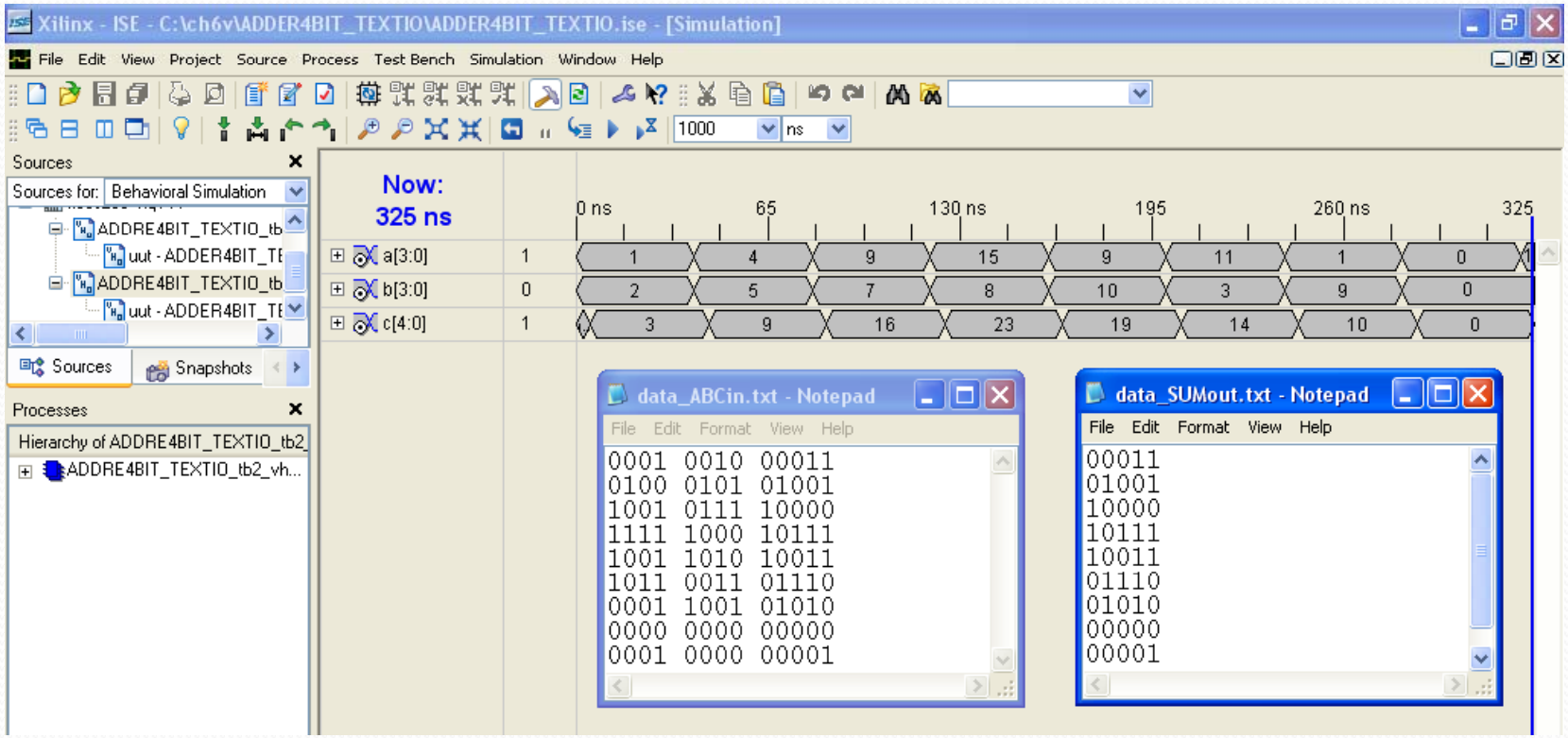


Input file

Output file

4 Bits Adder, Testbench Using File





VHDL

END, EPISODE I