Introduction to VHDL

Prepared by:
Eng. Waleed Saad

- VHDL is a language for describing digital hardware used by industry worldwide
  - **VHDL** is an acronym for **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

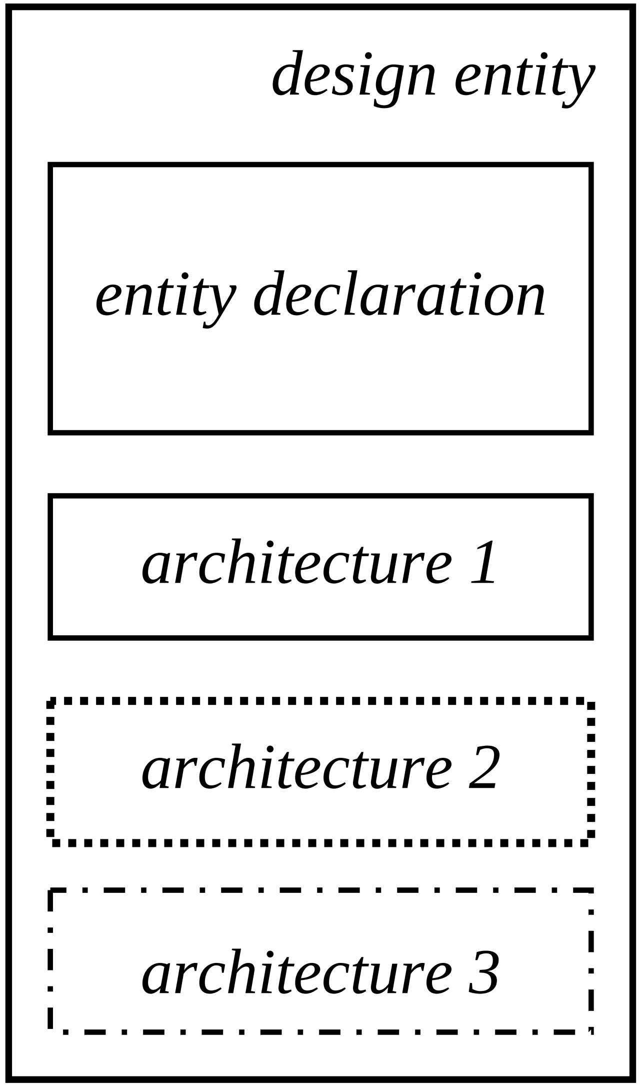
Features of VHDL

- Technology/vendor independent
- Portable
- Reusable

Three versions of VHDL

- VHDL-87
- VHDL-93
- VHDL-01

Design Entity

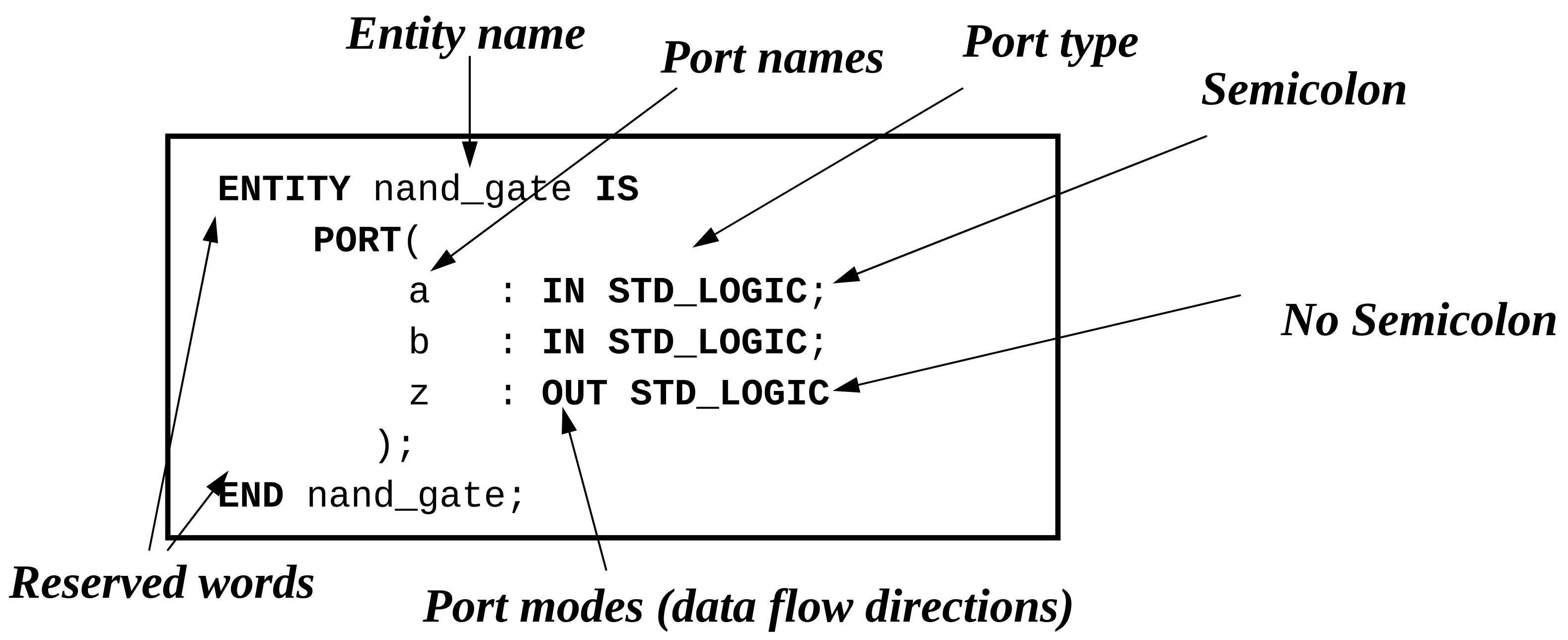


Design Entity - most basic building block of a design.

One *entity* can have many different *architectures.*

Entity Declaration

- *Entity Declaration* describes the interface of the component, i.e. *input* and *output* ports.



```
ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC
       );
END nand_gate;
```

Entity declaration – simplified syntax

```
ENTITY entity_name IS
    PORT (
        port_name :  signal_mode  signal_type;
        port_name :  signal_mode  signal_type;
        ………….
        port_name :  signal_mode  signal_type);
END entity_name;
```

Architecture

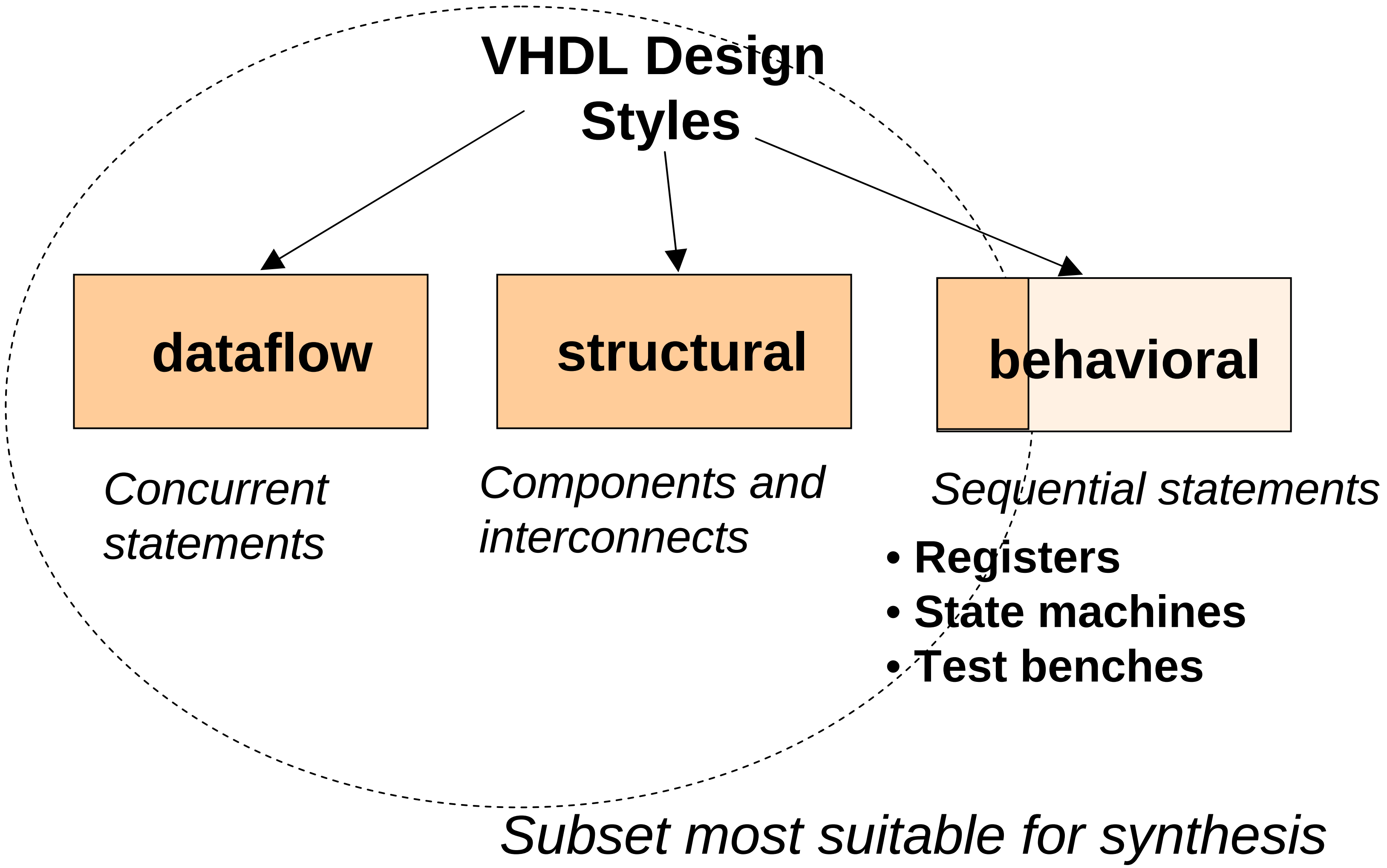
- Describes an implementation of a design entity.
- Architecture example:

```
ARCHITECTURE model OF nand_gate IS
BEGIN
    z <= a NAND b;
END model;
```

Architecture – simplified syntax

```
ARCHITECTURE architecture_name OF entity_name IS
    [ declarations ]
BEGIN
    code
END architecture_name;
```

VHDL Design Styles



Component and Instantiation (1)

- Named association connectivity (recommended)

```
component XOR2 is
        port(
            I1 : in STD_LOGIC;
            I2 : in STD_LOGIC;
            Y : out STD_LOGIC
            );
end component;

U1: XOR2 port map (I1 => A,
                                I2 => B,
                                Y  => U1_OUT);
```

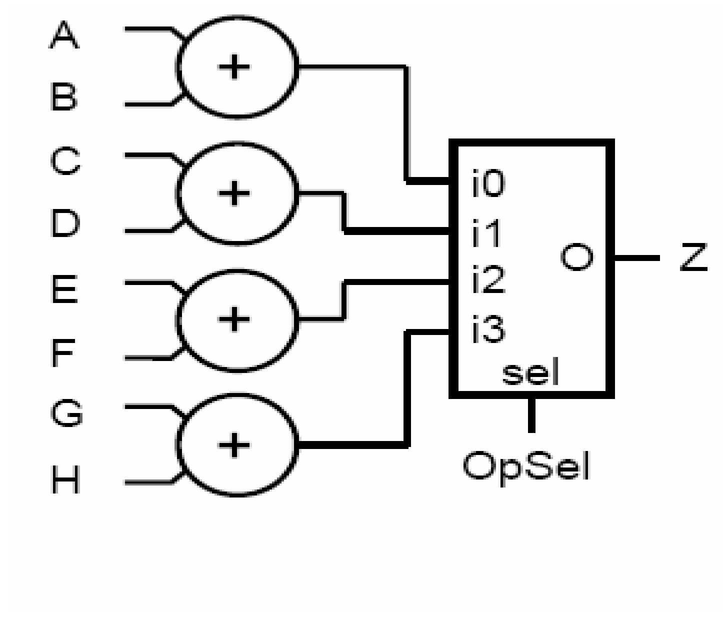
Component and Instantiation (2)

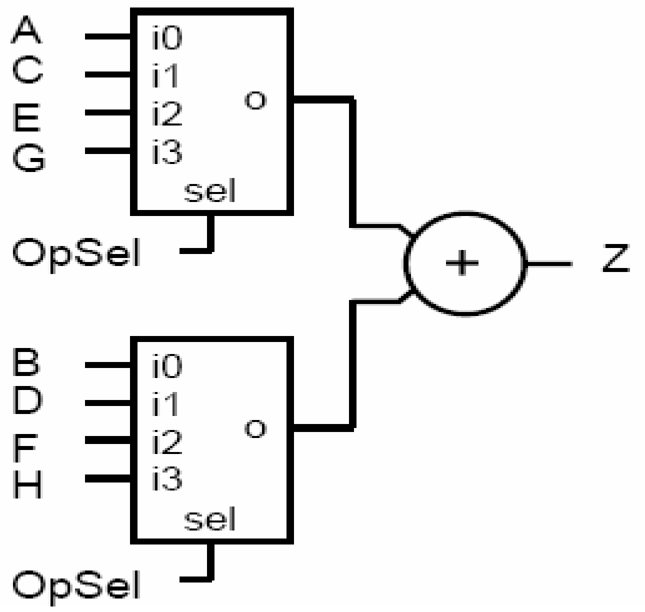
- Positional association connectivity (not recommended)

```
component XOR2 is
        port(
            I1 : in STD_LOGIC;
            I2 : in STD_LOGIC;
            Y : out STD_LOGIC
            );
end component;

U1: XOR2 port map (A, B, U1_OUT);
```

Optimal results:

OpSel	Function
00	A + B
01	C + D
10	E + F
11	G + H

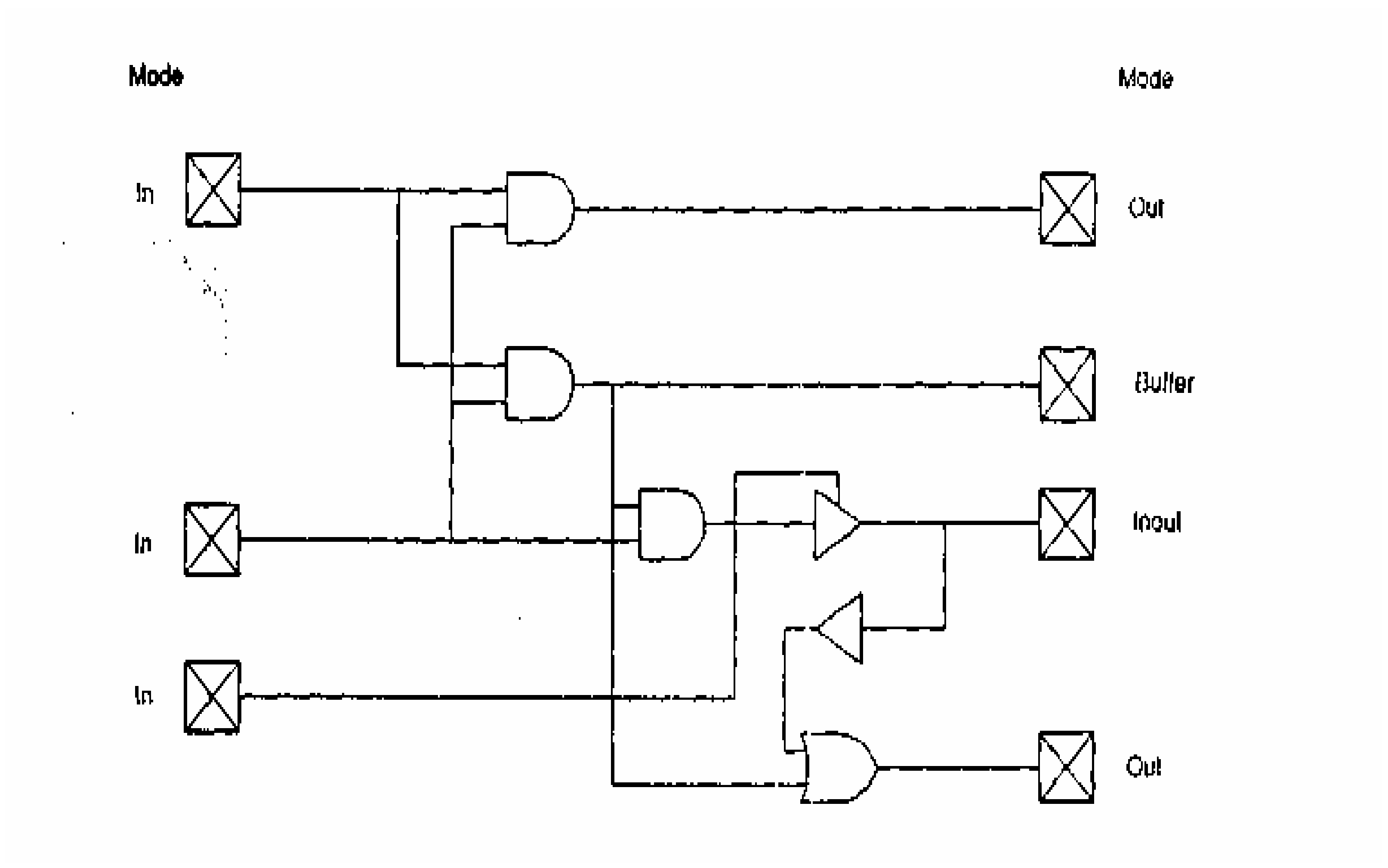




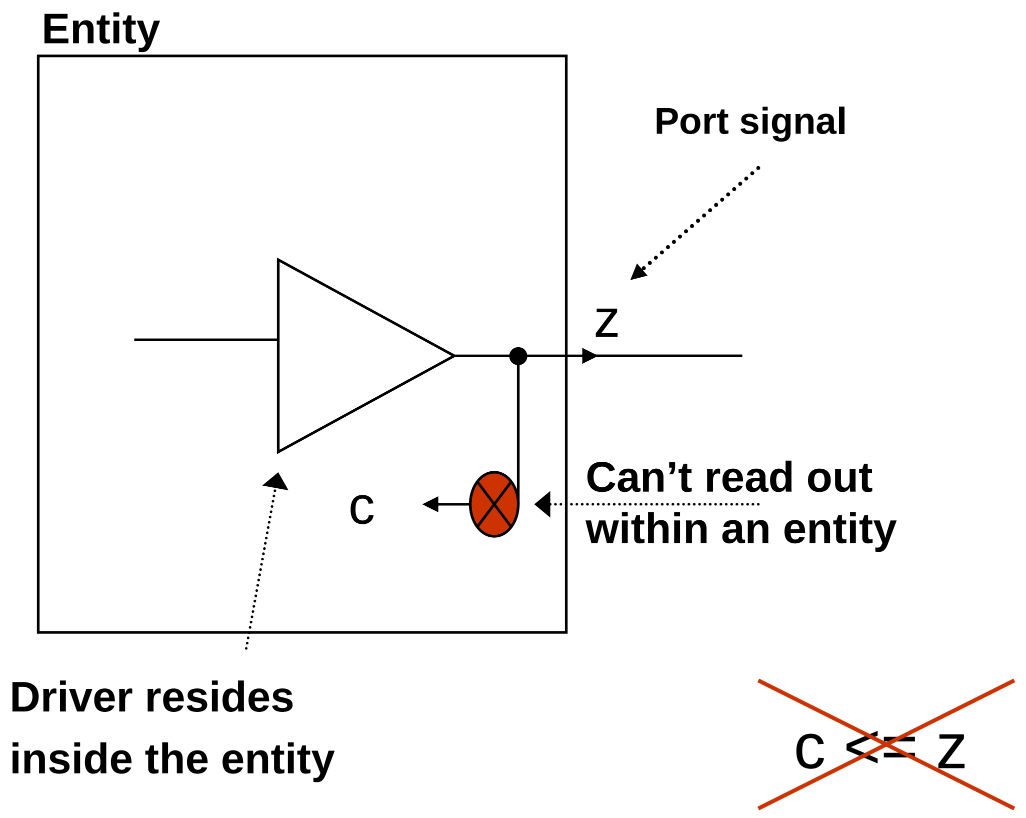
Port Mode

• In	: data flows in this port and can only be read (this is the default mode)
• Out	:data flows out this port and can only be written to
• Buffer	: similar to Out, but it allows for internal feedback
• Inout	: data flow can be in either direction with any number of sources allowed
• Linkage	: data flow direction is unknown

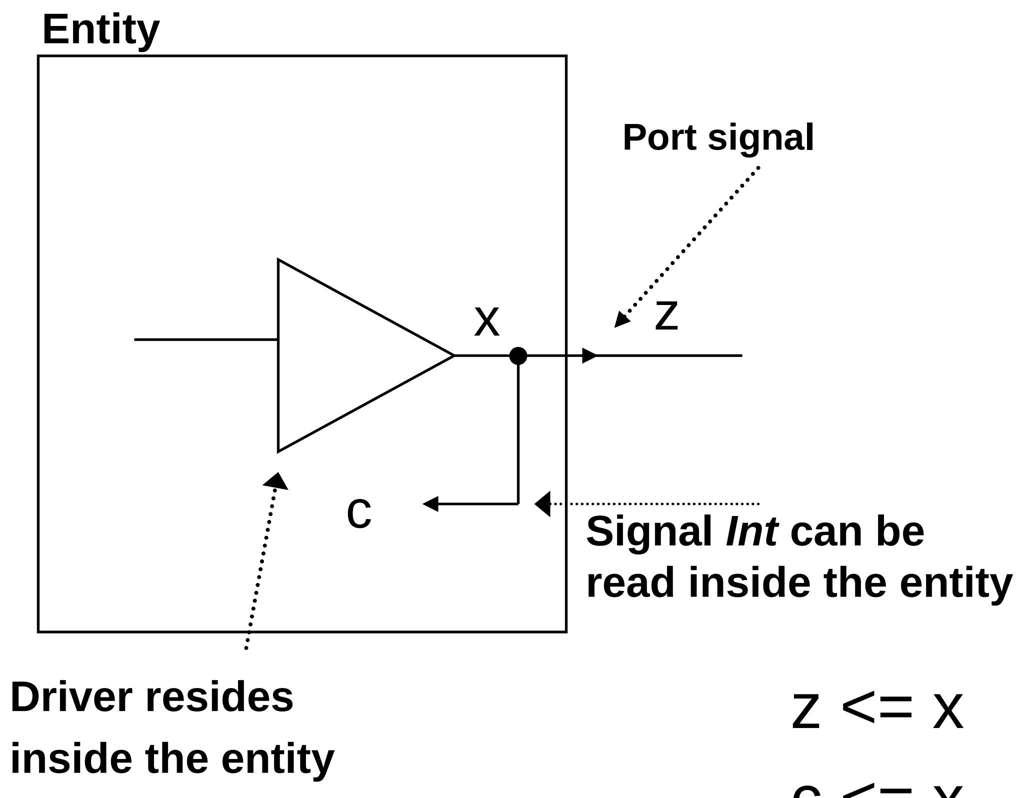
Modes and their signal sources



Mode *out*

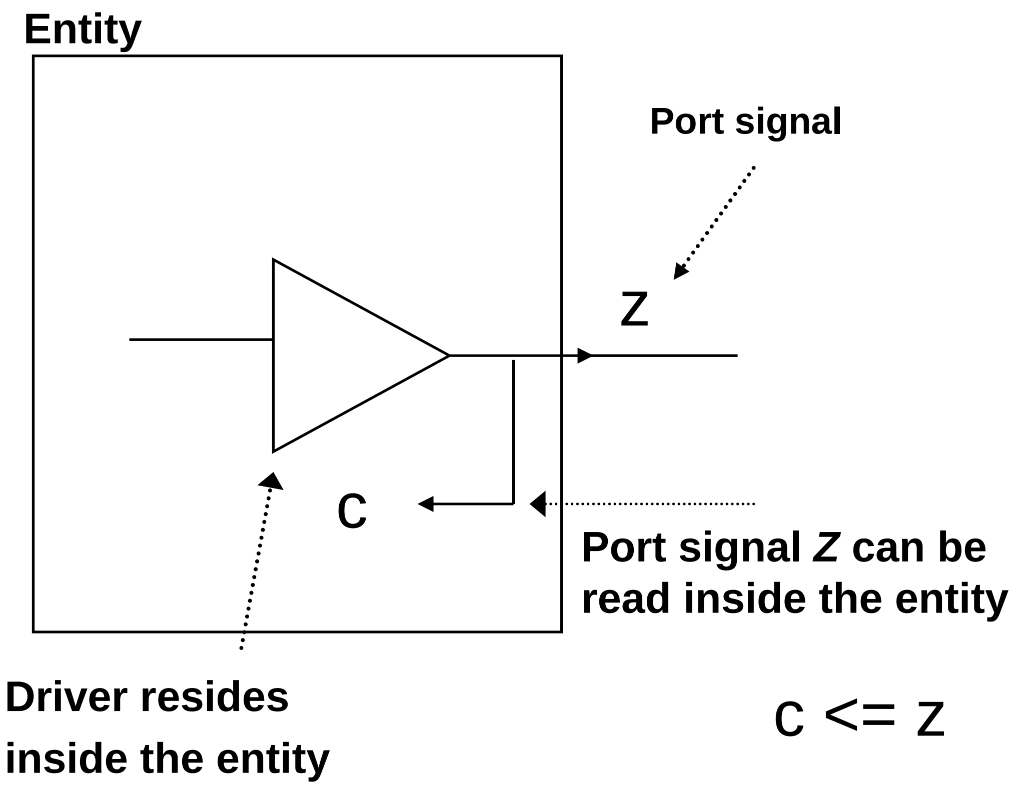

Entity
Port signal
Z
C
Can't read out
within an entity
Driver resides
inside the entity
~~C <= Z~~

Mode *out* with *signal*


Entity
Port signal
x
z
c
Signal *Int* can be read inside the entity
Driver resides inside the entity

```
z <= x
c <= x
```

Mode *buffer*


Entity
Port signal
Z
C
Port signal *Z* can be read inside the entity
Driver resides inside the entity

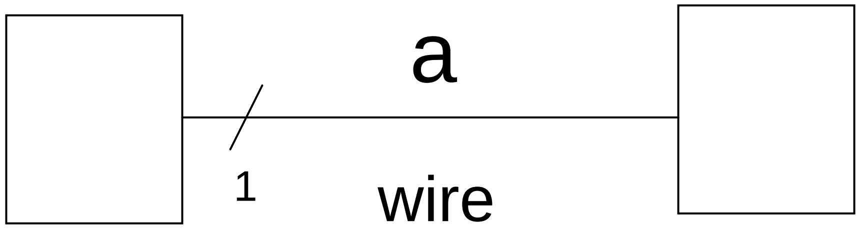
```
c <= z
```

Data Types

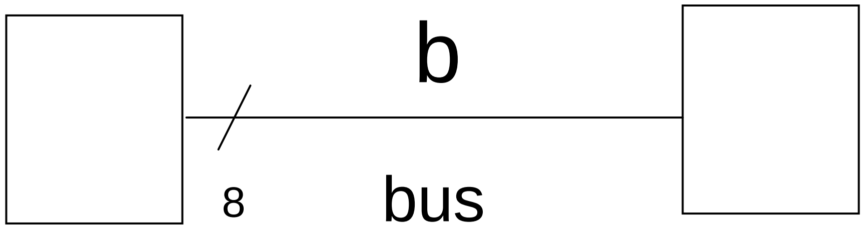
Data Type	***Values***	***Example***
Bit	‘1’,'0'	Q<=‘1’;
Bit_vector	(array of bits)	DataOut<="00010101";
Boolean	True, False	EQ<=True;
Integer	-2, -1,0, 1,2, 3,4. ..	Count <= Count + 2;
Real	1.0, -1.0E5	V1 =V2/5.3
Time	1 ua, 7 ns, 100 ps	Q<=T after 6ns;
Character	'a', 'b', '2, '$', etc.	CharData <= 'X';
String	(Array of characters)	Msg<="MEM:"&Addr

Signals

SIGNAL a : STD_LOGIC;

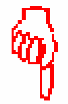


SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



Common VHDL Types

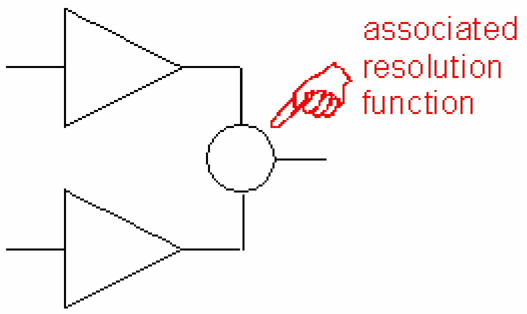
TYPE	Value	Origin
std_ulogic	'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'	std_logic_1164
std_ulogic_vector	array of std_ulogic	std_logic_1164
std_logic	resolved std_ulogic	std_logic_1164
std_logic_vector	array of std_logic	std_logic_1164
unsigned	array of std_logic	numeric_std, std_logic_arith
signed	array of std_logic	numeric_std, std_logic_arith
boolean	true, false	standard
character	191 / 256 characters	standard
string	array of character	standard
integer	$-(2^{31} -1)$ to $(2^{31} - 1)$	standard
real	-1.0E38 to 1.0E38	standard
time	1 fs to 1 hr	standard



std_ulogic

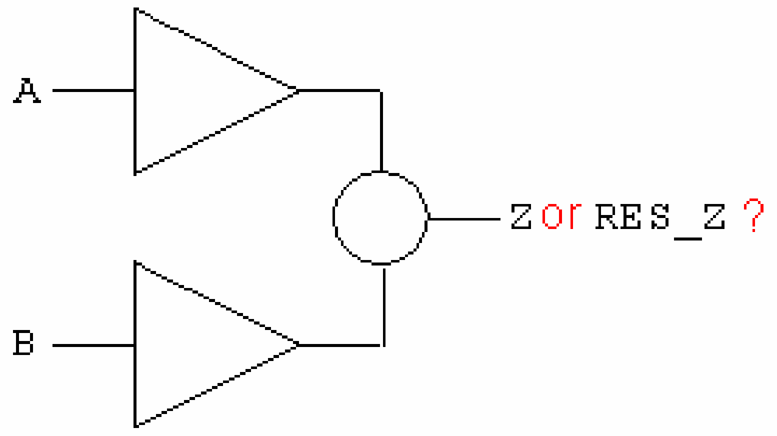
Only one driver!

std_logic



One or more drivers

```
signal A,B,Z : std_ulogic;
signal RES_Z : std_logic;
```



```
Z  <=  A;
Z  <=  B;
```

✗

```
RES_Z  <=  A;
RES_Z  <=  B;
```

✓

Standard Logic Vectors

```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL f: STD_LOGIC_VECTOR(8 DOWNTO 0);
                    ……….
a <= ‘1’;
b <= ”0000”;          -- Binary base assumed by default
c <= B”0000”;         -- Binary base explicitly specified
d <= ”0110_0111”;     -- You can use ‘_’ to increase readability
e <= X”AF67”;         -- Hexadecimal base
f <= O”723”;          -- Octal base
```

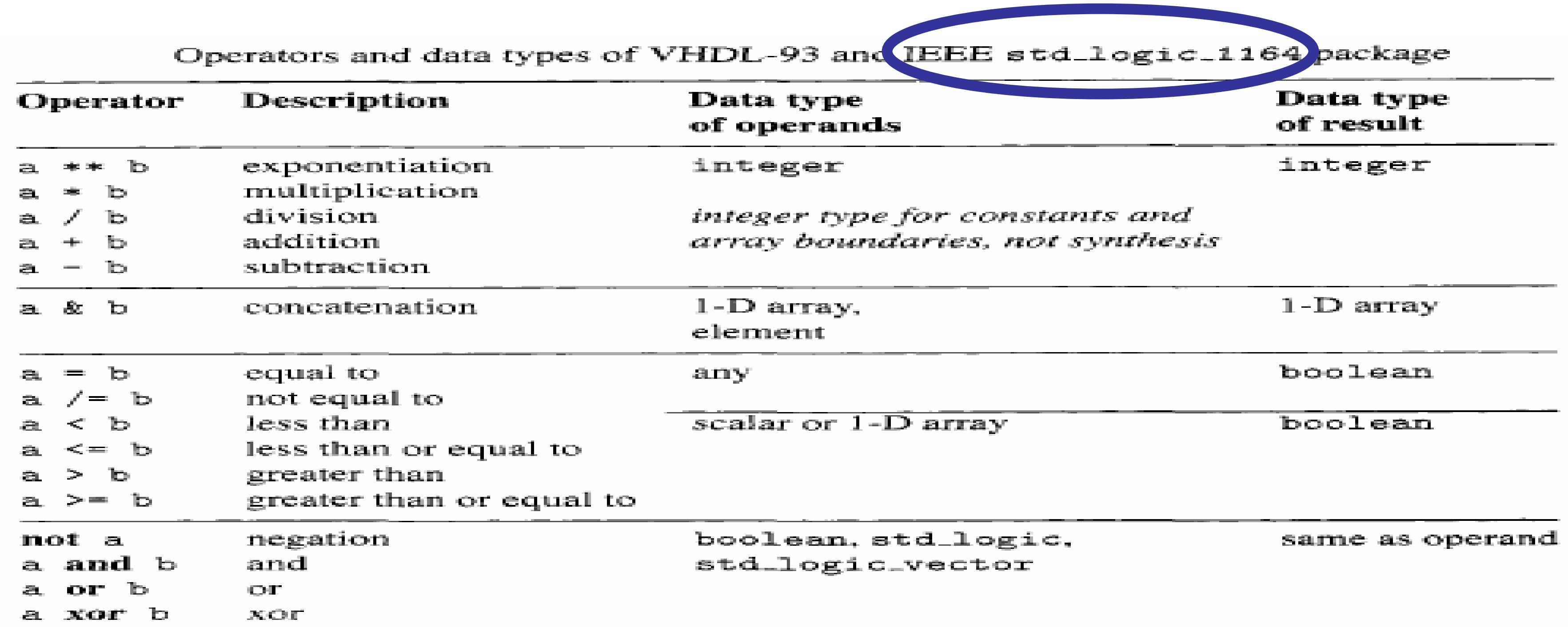
Vectors and Concatenation

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <= ”0000”;
b <= ”1111”;
c <= a & b;                  -- c = ”00001111”

d <= ‘0’ & ”0001111”;        -- d <= ”00001111”

e <= ‘0’ & ‘0’ & ‘0’ & ‘0’ & ‘1’ & ‘1’ &
     ‘1’ & ‘1’;
                             -- e <= ”00001111”
```

Size and type of target

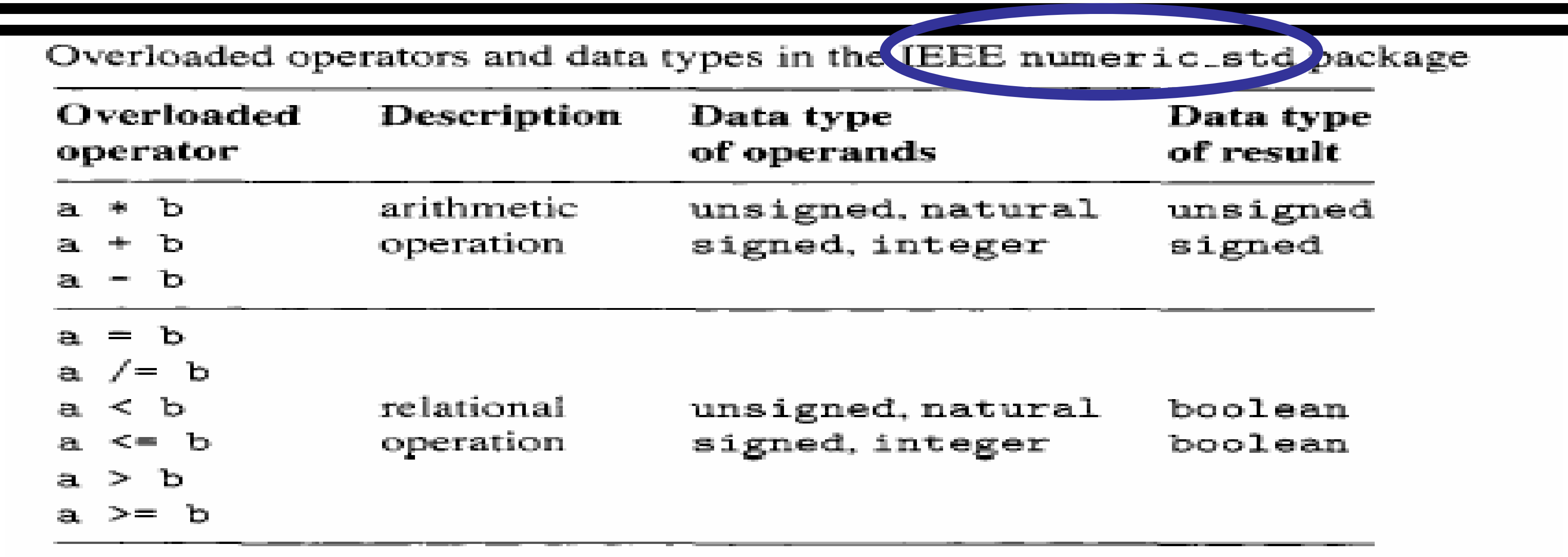
Operation	Size of Y = Size of Expression
`Y <= "10101010" ;`	number of digits in literal
`Y <= X"AA" ;`	4 * (number of digits)
`Y <= A ;`	A'Length = Length of array A
`Y <= A and B ;`	A'Length = B'Length
`W <= A > B ;`	Boolean
`Y <= A + B ;`	Maximum (A'Length, B'Length)
`Y <= A + 10 ;`	A'Length
`V <= A * B ;`	A'Length + B'Length

Packages operators



Operators and data types of VHDL-93 and IEEE std_logic_1164 package

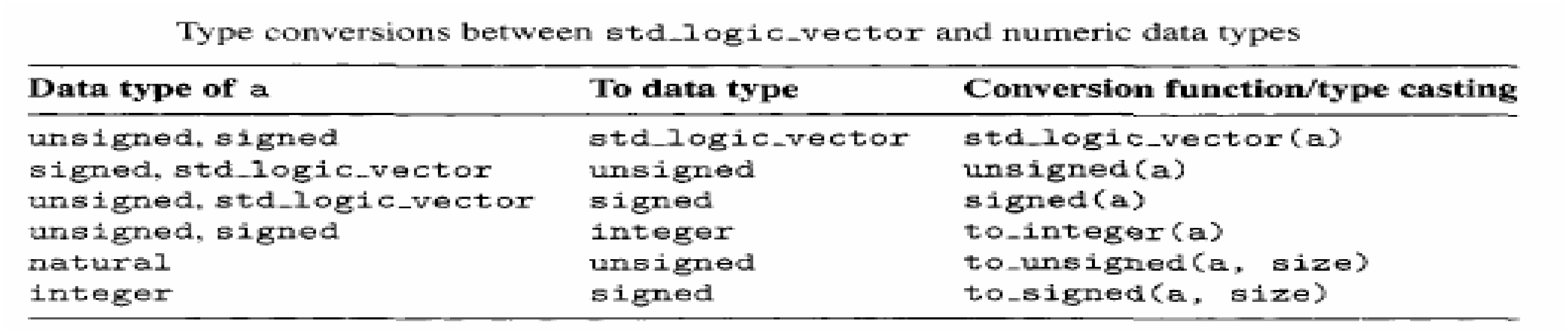
Operator	Description	Data type of operands	Data type of result
a ** b	exponentiation	integer	integer
a * b	multiplication		
a / b	division	*integer type for constants and*	
a + b	addition	*array boundaries, not synthesis*	
a - b	subtraction		
a & b	concatenation	1-D array, element	1-D array
a = b	equal to	any	boolean
a /= b	not equal to		
a < b	less than	scalar or 1-D array	boolean
a <= b	less than or equal to		
a > b	greater than		
a >= b	greater than or equal to		
not a	negation	boolean, std_logic,	same as operand
a **and** b	and	std_logic_vector	
a **or** b	or		
a **xor** b	xor		



Overloaded operators and data types in the IEEE numeric_std package

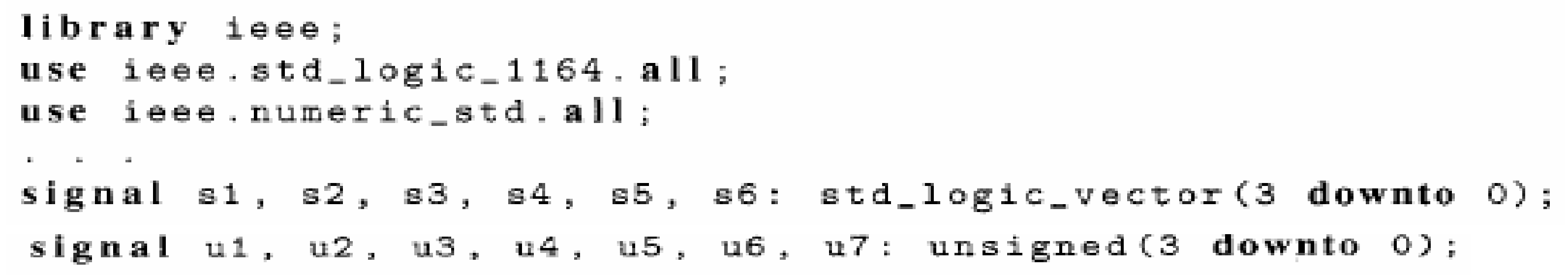
Overloaded operator	Description	Data type of operands	Data type of result
a * b	arithmetic	unsigned, natural	unsigned
a + b	operation	signed, integer	signed
a - b			
a = b			
a /= b			
a < b	relational	unsigned, natural	boolean
a <= b	operation	signed, integer	boolean
a > b			
a >= b			

Data types conversions

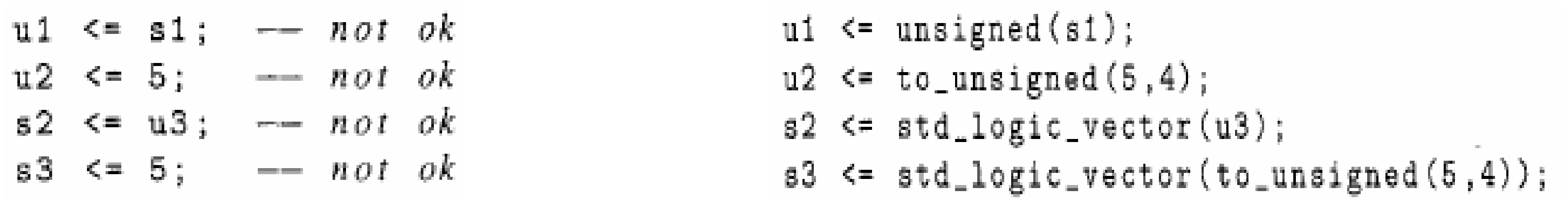


Type conversions between std_logic_vector and numeric data types

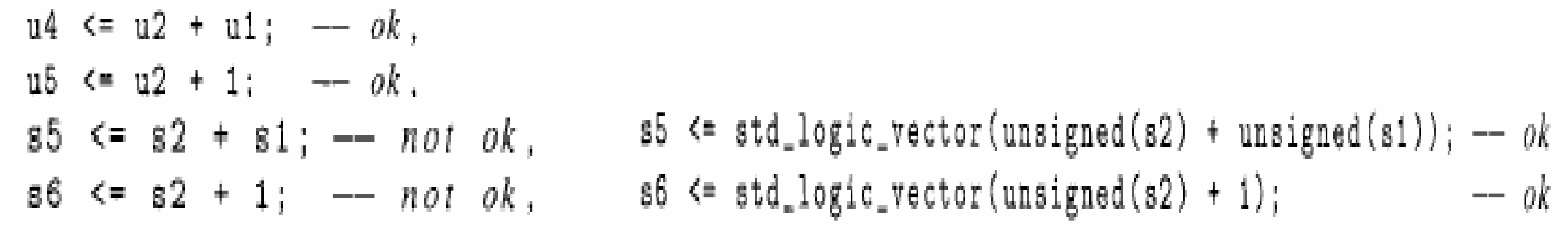
Data type of a	To data type	Conversion function/type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
. . .
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
 signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);
```



```
u1  <=  s1;   -- not ok                u1  <= unsigned(s1);
u2  <=  5;    -- not ok                u2  <= to_unsigned(5,4);
s2  <=  u3;   -- not ok                s2  <= std_logic_vector(u3);
s3  <=  5;    -- not ok                s3  <= std_logic_vector(to_unsigned(5,4));
```



```
u4 <= u2 + u1;  -- ok,
u5 <= u2 + 1;   -- ok,
s5 <= s2 + s1; -- not ok,     s5 <= std_logic_vector(unsigned(s2) + unsigned(s1)); -- ok
s6 <= s2 + 1;  -- not ok,     s6 <= std_logic_vector(unsigned(s2) + 1);            -- ok
```

Packages for Numeric Operations

- Using IEEE Numeric_Std

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

- Using Synopsys Std_Logic_Arith

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.std_logic_arith.all ;
  use ieee.std_logic_unsigned.all ;
```

Recommendation:
Use numeric_std for new designs

Use **numeric_std** or **std_logic_arith**, but never both

Unsigned and Signed Types

- Used to represent numeric values:

TYPE	Value	Notes
unsigned	0 to $2^N - 1$	
signed	$-2^{(N-1)}$ to $2^{(N-1)} - 1$	2's Complement number

- Usage similar to std_logic_vector:

```
signal A_unsigned     : unsigned(3 downto 0) ;
signal B_signed       : signed   (3 downto 0) ;
signal C_slv          : std_logic_vector (3 downto 0) ;
. . .

A_unsigned <= "1111" ;

B_signed   <= "1111" ;

C_slv      <= "1111" ;
```

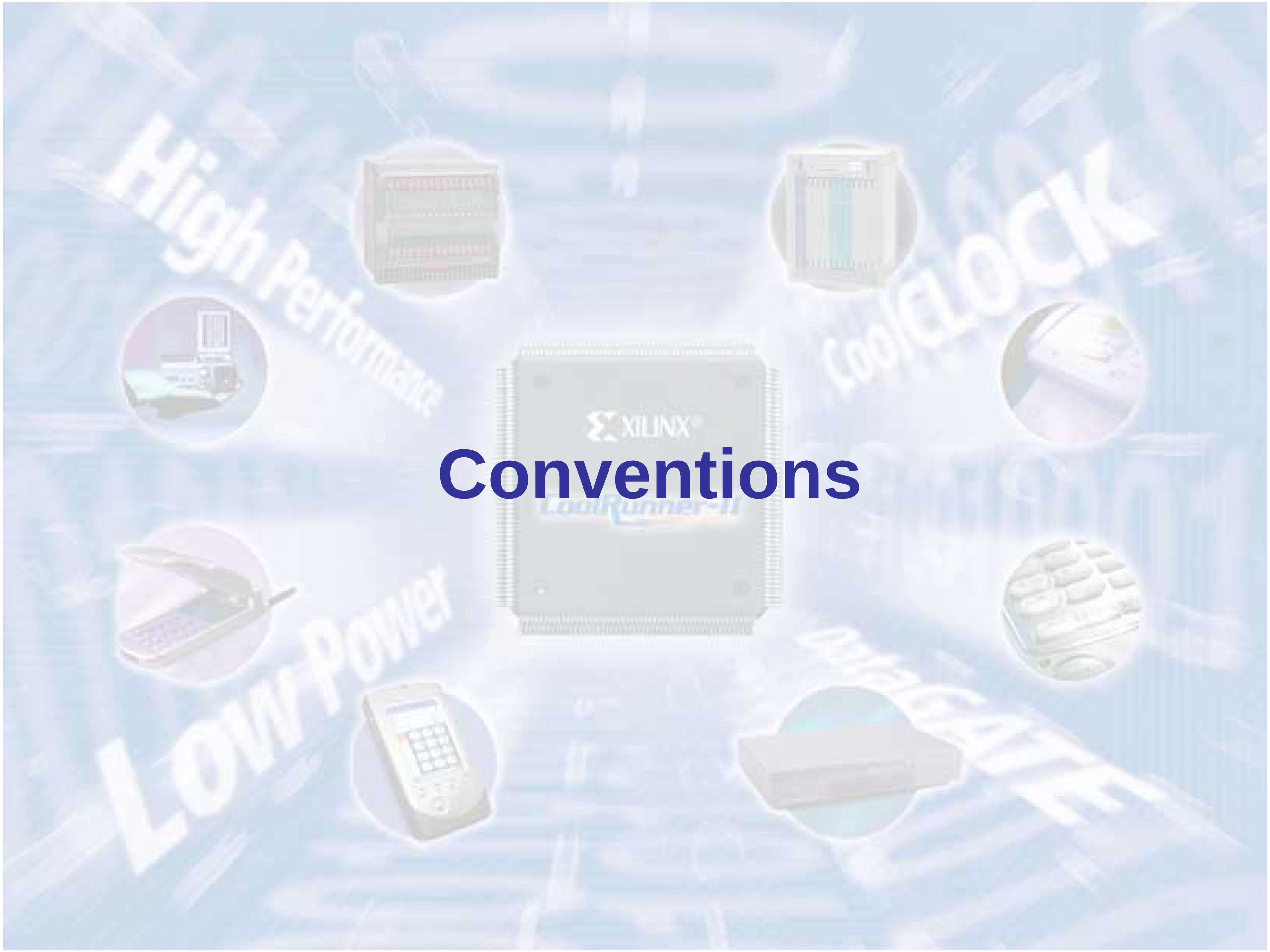
= 15 decimal

= -1 decimal

= 15 decimal only if using std_logic_unsigned

Overloading Examples

```
Signal A_uv, B_uv, C_uv, D_uv, E_uv : unsigned(7 downto 0) ;
Signal R_sv, S_sv, T_sv, U_sv, V_sv : signed(7 downto 0) ;
Signal J_slv, K_slv, L_slv     : std_logic_vector(7 downto 0) ;
signal Y_sv                    : signed(8 downto 0) ;
. . .

-- Permitted
A_uv <= B_uv + C_uv ;       -- Unsigned + Unsigned = Unsigned
D_uv <= B_uv + 1 ;          -- Unsigned + Integer  = Unsigned
E_uv <= 1 + C_uv;           -- Integer  + Unsigned = Unsigned

R_sv <= S_sv + T_sv ;       -- Signed   + Signed   = Signed
U_sv <= S_sv + 1 ;          -- Signed   + Integer  = Signed
V_sv <= 1 + T_sv;           -- Integer  + Signed   = Signed

J_slv <= K_slv + L_slv ;    -- if using std_logic_unsigned

-- Illegal Cannot mix different array types
-- Solution persented later in type conversions
-- Y_sv <= A_uv - B_uv ;    -- want signed result
```



Conventions

Naming and Labeling (1)

- VHDL is <u>not</u> case sensitive

  Example:

  Names or labels

  ```
  databus
  Databus
  DataBus
  DATABUS
  ```

  are all equivalent

Naming and Labeling (2)

General rules of thumb (according to VHDL-87)

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (__) within a name (e.g., Sel__A is invalid)
5. All names and labels in a given entity and architecture must be unique

Free Format

- VHDL is a “free format” language

  No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space and carriage return treated the same way.

  Example:

  ```
  if (a=b) then
  ```

  or

  ```
  if (a=b)         then
  ```

  or

  ```
  if (a =
  b) then
  ```

  are all equivalent

Comments

- Comments in VHDL are indicated with a "double dash", i.e., "--"
  - Comment indicator can be placed anywhere in the line
  - Any text that follows in the <u>same</u> line is treated as a comment
  - Carriage return terminates a comment
  - No method for commenting a block extending over a couple of lines

Examples:

```
-- main subcircuit
Data_in <= Data_bus;     -- reading data from the input FIFO
```

VHDL FAQ

What is the difference between VHDL and Verilog?

Can I use VHDL for the analog part of a design?

How must I write VHDL to make it synthesizable?

How many versions of VHDL are there?

Are there any tools to generate VHDL test benches automatically?

Are there translators from 'C' to VHDL?

I've heard that VHDL is very inefficient for FPGAs. Is that true?

Are freeware / shareware VHDL tools available?