Circuit Design with VHDL

Chapter 7-8: Predefined and user-defined Data Types

Instructor: Ali Jahanian

Outline

- I-CIRCUIT DESIGN

Data Types

- In this chapter:
  - all fundamental data types are described, with special emphasis on those that are **synthesizable**.
  - Discussions on data compatibility and data conversion are also included.

Pre-Defined Data Types

- VHDL contains a series of pre-defined data types, specified through the IEEE 1076 and IEEE 1164 standards.

Pre-Defined Data Types …

- **BIT (and BIT_VECTOR)**: 2-level logic (‘0’, ‘1’).

```
SIGNAL x: BIT;
-- x is declared as a one-digit signal of type BIT.

SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
-- y is a 4-bit vector, with the leftmost bit being the MSB.

SIGNAL w: BIT_VECTOR (0 TO 7);
-- w is an 8-bit vector, with the rightmost bit being the MSB.
```

Pre-Defined Data Types …

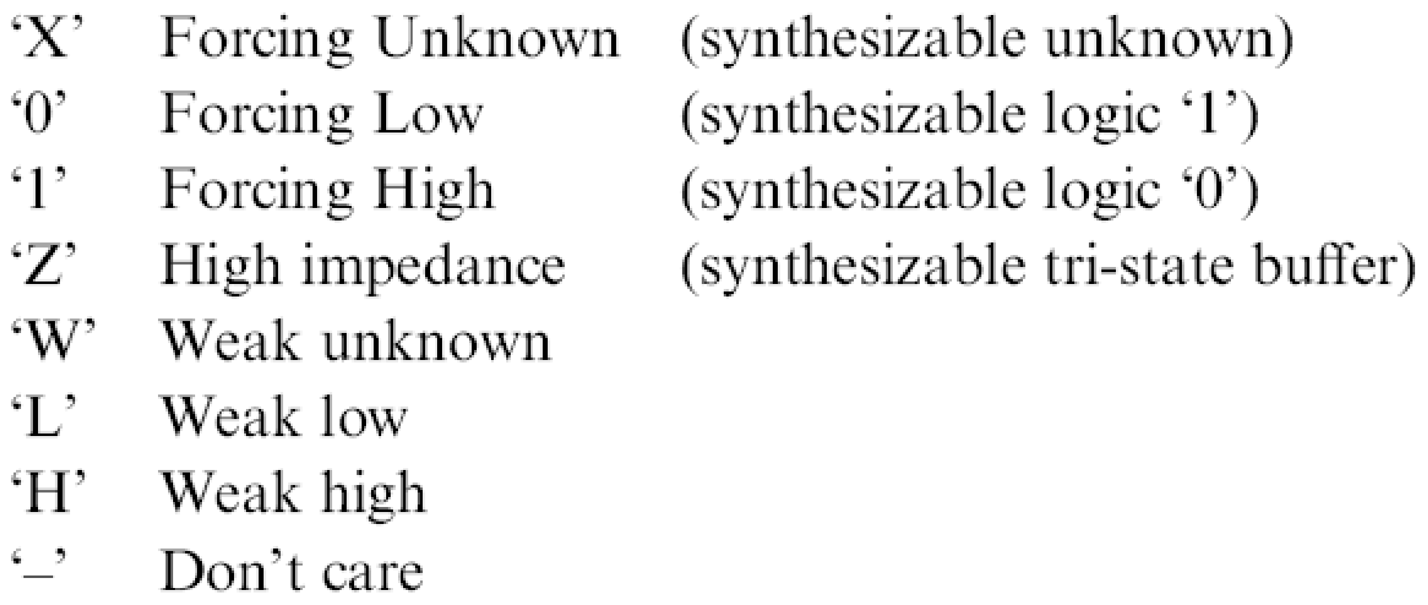
```
x <= '1';
-- x is a single-bit signal (as specified above), whose value is
-- '1'. Notice that single quotes (' ') are used for a single bit.

y <= "0111";
-- y is a 4-bit signal (as specified above), whose value is "0111"
-- (MSB='0'). Notice that double quotes (" ") are used for
-- vectors.

w <= "01110001";
-- w is an 8-bit signal, whose value is "01110001" (MSB='1').
```

Pre-Defined Data Types…

- **STD_LOGIC (and STD_LOGIC_VECTOR)**
  - 8-valued logic system introduced in the IEEE 1164 standard.



'X'	Forcing Unknown	(synthesizable unknown)
'0'	Forcing Low	(synthesizable logic '1')
'1'	Forcing High	(synthesizable logic '0')
'Z'	High impedance	(synthesizable tri-state buffer)
'W'	Weak unknown	
'L'	Weak low	
'H'	Weak high	
'–'	Don't care	

Pre-Defined Data Types …

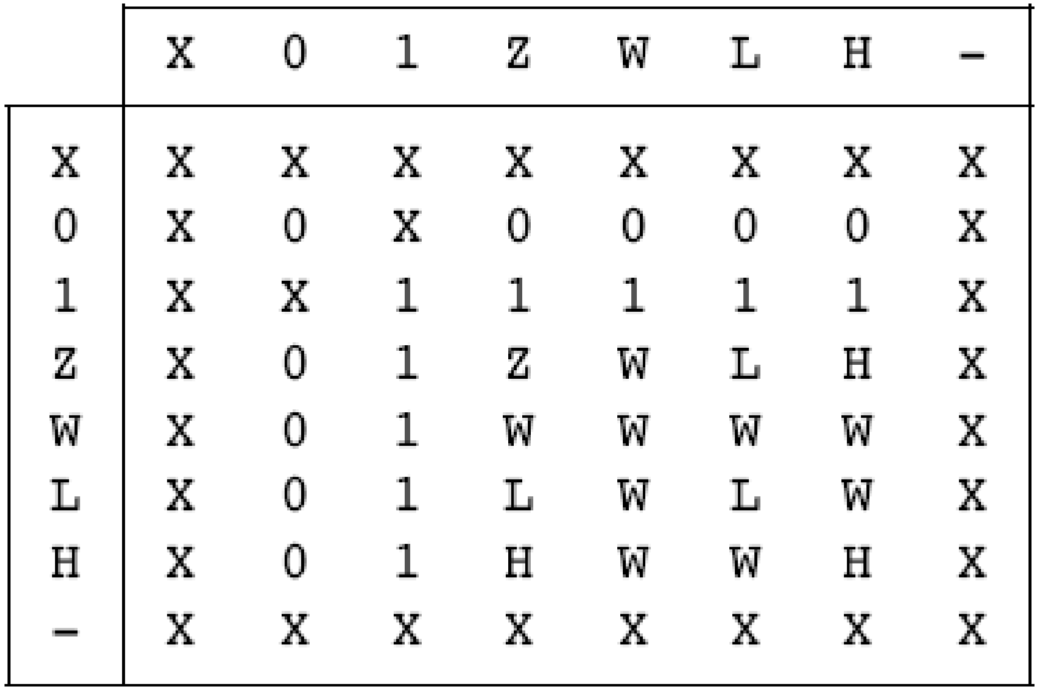
```
SIGNAL x: STD_LOGIC;
-- x is declared as a one-digit (scalar) signal of type STD_LOGIC.

SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001";
-- y is declared as a 4-bit vector, with the leftmost bit being
-- the MSB. The initial value (optional) of y is "0001". Notice
-- that the ":=" operator is used to establish the initial value.
```

Pre-Defined Data Types …

- Resolved logic system

Table 3.1
Resolved logic system (STD_LOGIC).



	X	0	1	Z	W	L	H	–
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
–	X	X	X	X	X	X	X	X

Pre-Defined Data Types …

- **STD_ULOGIC (STD_ULOGIC_VECTOR)**
  - 9-level logic system introduced in the IEEE 1164 standard (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘–’).
    - ‘U’ stands for unresolved
  - Notes:
    - STD_ULOGIC type cannot be used for multiple-drived signals.
    - STD_LOGIC is a resolved version of STD_ULOGC

Pre-Defined Data Types …

- **BOOLEAN**: True, False.
- **INTEGER:** 32-bit integers (from −2,147,483,647 to +2,147,483,647).
- **NATURAL:** Non-negative integers (from 0 to +2,147,483,647).
- **REAL:** Real numbers ranging from 1.0E38 to +1.0E38.
  - Not synthesizable.
- **Physical literals:** Used to inform physical quantities, like time, voltage, etc. Useful in simulations.
  - Not synthesizable.
- **Character literals:** Single ASCII character or a string of such characters.
  - Not synthesizable.

Pre-Defined Data Types …

- Examples

```
x0 <= '0';              -- bit, std_logic, or std_ulogic value '0'
x1 <= "00011111";       -- bit_vector, std_logic_vector,
                        -- std_ulogic_vector, signed, or unsigned
x2 <= "0001_1111";      -- underscore allowed to ease visualization
x3 <= "101111"          -- binary representation of decimal 47
x4 <= B"101111"         -- binary representation of decimal 47
x5 <= O"57"             -- octal representation of decimal 47
x6 <= X"2F"             -- hexadecimal representation of decimal 47
n <= 1200;              -- integer
m <= 1_200;             -- integer, underscore allowed
IF ready THEN...        -- Boolean, executed if ready=TRUE
y <= 1.2E-5;            -- real, not synthesizable
q <= d after 10 ns;     -- physical, not synthesizable
```

Pre-Defined Data Types …

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
```

```
a <= b(5);
b(0) <= a;
c <= d(5);
d(0) <= c;
a <= c;
b <= d;


e <= b;
e <= d;
```

User-Defined Data Types

- Integer

```
TYPE integer IS RANGE -2147483647 TO +2147483647;
-- This is indeed the pre-defined type INTEGER.

TYPE natural IS RANGE 0 TO +2147483647;
-- This is indeed the pre-defined type NATURAL.


TYPE my_integer IS RANGE -32 TO 32;
-- A user-defined subset of integers.

TYPE student_grade IS RANGE 0 TO 100;
-- A user-defined subset of integers or naturals.
```

User-Defined Data Types …

- Enumerated

```
TYPE bit IS ('0', '1');
-- This is indeed the pre-defined type BIT

TYPE my_logic IS ('0', '1', 'Z');
-- A user-defined subset of std_logic.
```

User-Defined Data Types …

- Enumerated

```
TYPE bit_vector IS ARRAY (NATURAL RANGE <>) OF BIT;
-- This is indeed the pre-defined type BIT_VECTOR.
-- RANGE <> is used to indicate that the range is unconstrained.
-- NATURAL RANGE <>, on the other hand, indicates that the only
-- restriction is that the range must fall within the NATURAL
-- range.

TYPE state IS (idle, forward, backward, stop);
-- An enumerated data type, typical of finite state machines.

TYPE color IS (red, green, blue, white);
-- Another enumerated data type.
```

Subtypes

- A SUBTYPE is a TYPE with a constraint.

- The main reason for using a subtype rather than specifying a new type is that, though operations between data of different types are not allowed, they are allowed between a subtype and its corresponding base type.

Subtypes …

```
SUBTYPE natural IS INTEGER RANGE 0 TO INTEGER'HIGH;
-- As expected, NATURAL is a subtype (subset) of INTEGER.

SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO 'Z';
-- Recall that STD_LOGIC=('X','0','1','Z','W','L','H','-').
-- Therefore, my_logic=('0','1','Z').

SUBTYPE my_color IS color RANGE red TO blue;
-- Since color=(red, green, blue, white), then
-- my_color=(red, green, blue).

SUBTYPE small_integer IS INTEGER RANGE -32 TO 32;
-- A subtype of INTEGER.
```

Subtypes …

```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO '1';
SIGNAL a: BIT;
SIGNAL b: STD_LOGIC;
SIGNAL c: my_logic;
...
b <= a;    -- illegal (type mismatch: BIT versus STD_LOGIC)
b <= c;    -- legal (same "base" type: STD_LOGIC)
```

Arrays

- Arrays are collections of objects of the same type.

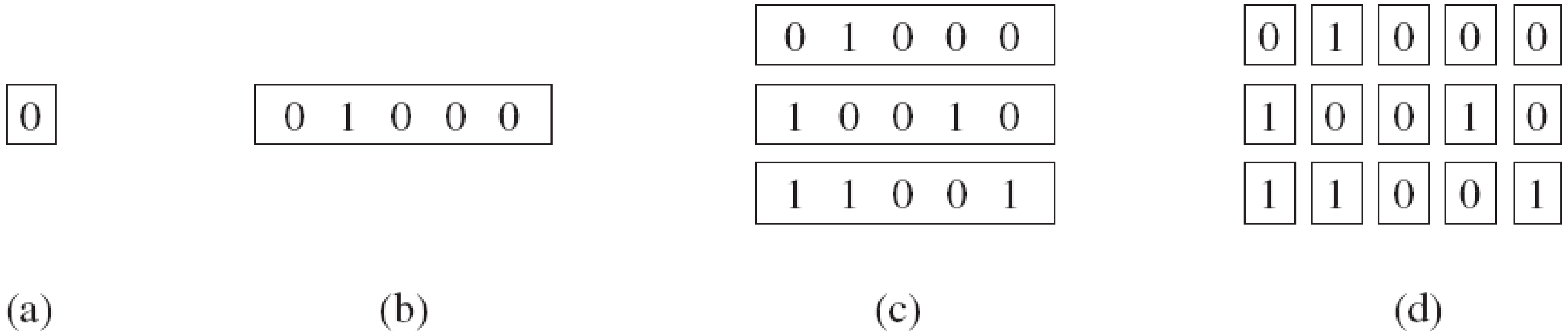


Figure 3.1
Illustration of (a) scalar, (b) 1D, (c) 1Dx1D, and (d) 2D data arrays.

There are no pre-defined 2D or 1Dx1D arrays

Arrays …

To specify a new array type:

```
TYPE type_name IS ARRAY (specification) OF data_type;
```

To make use of the new array type:

```
SIGNAL signal_name: type_name [:= initial_value];
```

Example: 1Dx1D array

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;     -- 1D array
TYPE matrix IS ARRAY (0 TO 3) OF row;            -- 1Dx1D array
SIGNAL x: matrix;                                -- 1Dx1D signal
```

Arrays …

Another Example: 1Dx1D array

```
TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
```

Example: 2D array

```
TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
```

Example:Array initialization

```
 ... :="0001";                                -- for 1D array
 ... :=('0','0','0','1')                      -- for 1D array
 ... :=(('0','1','1','1'), ('1','1','1','0')); -- for 1Dx1D or
                                              -- 2D array
```

Arrays …

Example: Legal and illegal array assignments

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
                                                    -- 1D array
TYPE array1 IS ARRAY (0 TO 3) OF row;
                                                    -- 1Dx1D array
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
                                                    -- 1Dx1D
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
                                                    -- 2D array
SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
```

Arrays …

Example: Legal and illegal array assignments

```
x(0) <= y(1)(2);

x(1) <= v(2)(3);
x(2) <= w(2,1);
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
```

Arrays …

```
x <= y(0);                  -- legal (same data types: ROW)
x <= v(1);                  -- illegal (type mismatch: ROW x
                            -- STD_LOGIC_VECTOR)
x <= w(2);                  -- illegal (w must have 2D index)
x <= w(2, 2 DOWNTO 0);      -- illegal (type mismatch: ROW x
                            -- STD_LOGIC)
v(0) <= w(2, 2 DOWNTO 0);   -- illegal (mismatch: STD_LOGIC_VECTOR
                            -- x STD_LOGIC)
v(0) <= w(2);               -- illegal (w must have 2D index)
y(1) <= v(3);               -- illegal (type mismatch: ROW x
                            -- STD_LOGIC_VECTOR)
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0);      -- legal (same type,
                                        -- same size)
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0);   -- legal (same type,
                                        -- same size)
w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0);   -- illegal (type mismatch)
```

Port Array

- There are no pre-defined data types of more than one dimension

- we might need to specify the ports as arrays of vectors.

Port Array …

```
------- Package: --------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------
PACKAGE my_data_types IS
   TYPE vector_array IS ARRAY (NATURAL RANGE <>) OF
      STD_LOGIC_VECTOR(7 DOWNTO 0);
END my_data_types;
--------------------------------------------
```

Port Array …

```
------- Main code: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_data_types.all;     -- user-defined package
---------------------------
ENTITY mux IS
   PORT (inp: IN VECTOR_ARRAY (0 TO 3);
   ... );
END mux;
   ... ;
--------------------------------------------
```

Records

- Records are similar to arrays, with the only difference that they contain objects of different types.

```
TYPE birthday IS RECORD
   day: INTEGER RANGE 1 TO 31;
   month: month_name;
END RECORD;
```

Signed and Unsigned Data Types

- these types are defined in the *std_logic_arith* package.

```
SIGNAL x: SIGNED (7 DOWNTO 0);
SIGNAL y: UNSIGNED (0 TO 3);
```

Signed and Unsigned Data Types …

Example: Legal and illegal operations with signed/unsigned data types.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;    -- extra package necessary
...
SIGNAL a: IN SIGNED (7 DOWNTO 0);
SIGNAL b: IN SIGNED (7 DOWNTO 0);
SIGNAL x: OUT SIGNED (7 DOWNTO 0);
...
v <= a + b;      -- legal (arithmetic operation OK)
w <= a AND b;    -- illegal (logical operation not OK)
```

Signed and Unsigned Data Types …

Example: Legal and illegal operations with std_logic_vector.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;    -- no extra package required
...
SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
v <= a + b;        -- illegal (arithmetic operation not OK)
w <= a AND b;      -- legal (logical operation OK)
```

Signed and Unsigned Data Types …

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;     -- extra package included
...

SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
v <= a + b;        -- legal (arithmetic operation OK), unsigned
w <= a AND b;      -- legal (logical operation OK)
```

Data Conversion

- Write a piece of VHDL code.
- Invoke a FUNCTION from a pre-defined PACKAGE.

```
TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
...
y <= 2*x + 5;          -- error, type mismatch
y <= long(2*x + 5);    -- OK, result converted into type long
```

straightforward conversion functions

Data Conversion …

- conv_integer(p) :
  - Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an INTEGER value. Notice that STD_LOGIC_VECTOR is not included.
- conv_unsigned(p, b):
  - Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an UNSIGNED value with size b bits.
- conv_signed(p, b):
  - Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to a SIGNED value with size b bits.

Data Conversion …

- conv_std_logic_vector(p, b):
  - Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size b bits.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
...
SIGNAL a: IN UNSIGNED (7 DOWNTO 0);
SIGNAL b: IN UNSIGNED (7 DOWNTO 0);
SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
-- Legal operation: a+b is converted from UNSIGNED to an
-- 8-bit STD_LOGIC_VECTOR value, then assigned to y.
```

Summary

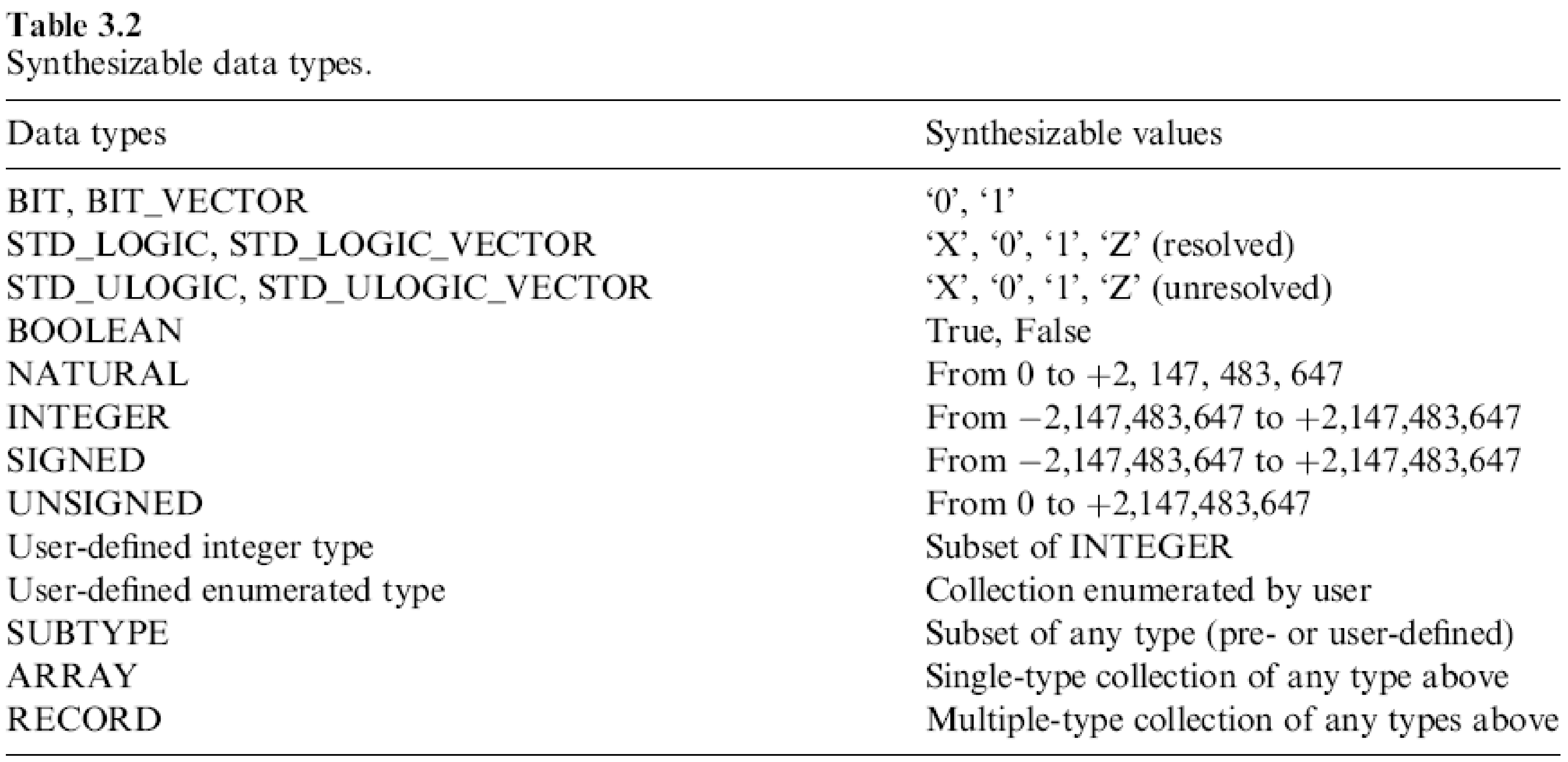


Table 3.2
Synthesizable data types.

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From −2,147,483,647 to +2,147,483,647
SIGNED	From −2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above

Additional Examples (Dealing with Data Types)

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;                  -- 1D
                                                               -- array
TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;          -- 2D
                                                               -- array
TYPE mem2 IS ARRAY (0 TO 3) OF byte;                           -- 1Dx1D
                                                               -- array
TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7);       -- 1Dx1D
                                                               -- array
SIGNAL a: STD_LOGIC;                                  -- scalar signal
SIGNAL b: BIT;                                        -- scalar signal
SIGNAL x: byte;                                       -- 1D signal
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);              -- 1D signal
SIGNAL v: BIT_VECTOR (3 DOWNTO 0);                    -- 1D signal
SIGNAL z: STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);         -- 1D signal
SIGNAL w1: mem1;                                      -- 2D signal
SIGNAL w2: mem2;                                      -- 1Dx1D signal
SIGNAL w3: mem3;                                      -- 1Dx1D signal
```

Additional Examples (Dealing with Data Types) …

```
x(2) <= a;            -- same types (STD_LOGIC), correct indexing
y(0) <= x(0);         -- same types (STD_LOGIC), correct indexing
z(7) <= x(5);         -- same types (STD_LOGIC), correct indexing
b <= v(3);            -- same types (BIT), correct indexing
w1(0,0) <= x(3);      -- same types (STD_LOGIC), correct indexing
w1(2,5) <= y(7);      -- same types (STD_LOGIC), correct indexing
w2(0)(0) <= x(2);     -- same types (STD_LOGIC), correct indexing
w2(2)(5) <= y(7);     -- same types (STD_LOGIC), correct indexing
w1(2,5) <= w2(3)(7);  -- same types (STD_LOGIC), correct indexing
------- Illegal scalar assignments: --------------------
b <= a;              -- type mismatch (BIT x STD_LOGIC)
w1(0)(2) <= x(2);    -- index of w1 must be 2D
w2(2,0) <= a;        -- index of w2 must be 1Dx1D
```

Additional Examples (Dealing with Data Types) …

```
-------- Legal vector assignments: ------------------------
x <= "11111110";
y <= ('1','1','1','1','1','1','0','Z');
z <= "11111" & "000";
x <= (OTHERS => '1');
y <= (7 =>'0', 1 =>'0', OTHERS => '1');
z <= y;
y(2 DOWNTO 0) <= z(6 DOWNTO 4);
w2(0)(7 DOWNTO 0) <= "11110000";
w3(2) <= y;
z <= w3(1);
z(5 DOWNTO 0) <= w3(1)(2 TO 7);
w3(1) <= "00000000";
w3(1) <= (OTHERS => '0');
w2 <= ((OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'));
w3 <= ("11111100", ('0','0','0','0','Z','Z','Z','Z',),
       (OTHERS=>'0'), (OTHERS=>'0'));
w1 <= ((OTHERS=>'Z'), "11110000" ,"11110000", (OTHERS=>'0'));
```

Additional Examples (Dealing with Data Types) …

```
------ Illegal array assignments: ----------------------
x <= y;                             -- type mismatch
y(5 TO 7) <= z(6 DOWNTO 0);         -- wrong direction of y
w1 <= (OTHERS => '1');              -- w1 is a 2D array
w1(0, 7 DOWNTO 0) <="11111111";     -- w1 is a 2D array
w2 <= (OTHERS => 'Z');              -- w2 is a 1Dx1D array
w2(0, 7 DOWNTO 0) <= "11110000";    -- index should be 1Dx1D
```

Additional Examples (Dealing with Data Types) …

```
-- Example of data type independent array initialization:
FOR i IN 0 TO 3 LOOP
   FOR j IN 7 DOWNTO 0 LOOP
      x(j) <= '0';
      y(j) <= '0'
      z(j) <= '0';
      w1(i,j) <= '0';
      w2(i)(j) <= '0';
      w3(i)(j) <= '0';
   END LOOP;
END LOOP;
---------------------------------------------------------
```

Additional Examples (Single Bit Versus Bit Vector)

```
----------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT;
         x: OUT BIT);
END and2;
----------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
----------------------------
```

```
-----------------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT_VECTOR (0 TO 3);
         x: OUT BIT_VECTOR (0 TO 3));
END and2;
-----------------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------------
```

Additional Examples (Single Bit Versus Bit Vector) …

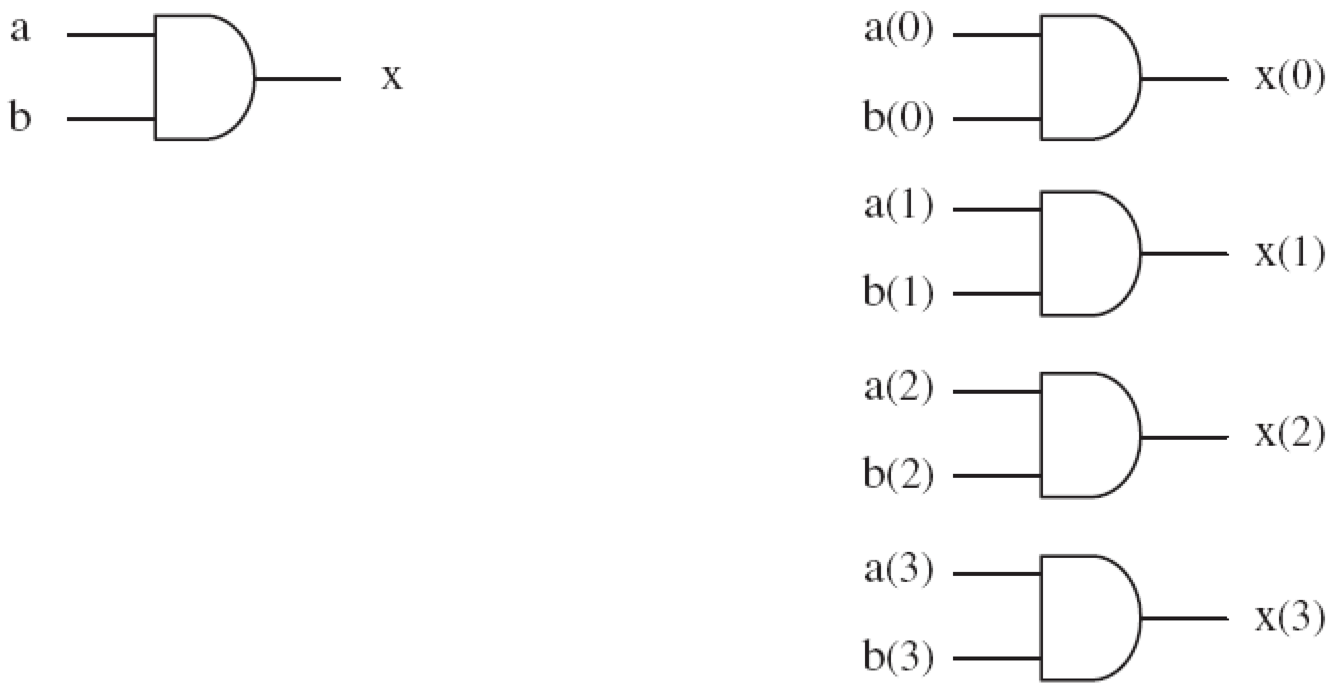


Figure 3.2
Circuits inferred from the codes of example 3.2.

Additional Examples (Adder)

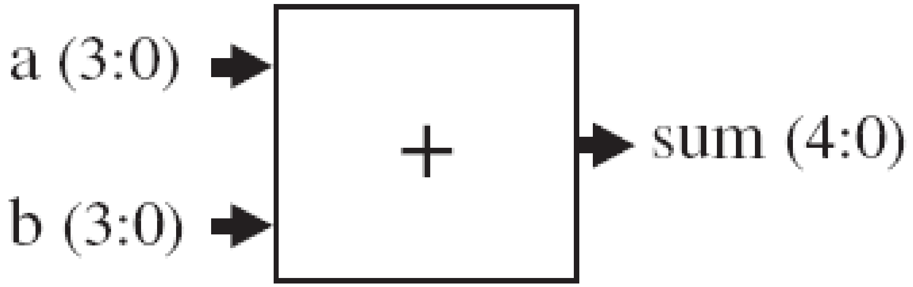


Figure 3.3
4-bit adder of example 3.3.

Additional Examples (Adder) …

```
----- Solution 1: in/out=SIGNED ----------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
-----------------------------------------
ENTITY adder1 IS
   PORT ( a, b : IN SIGNED (3 DOWNTO 0);
          sum : OUT SIGNED (4 DOWNTO 0));
END adder1;
-----------------------------------------
ARCHITECTURE adder1 OF adder1 IS
BEGIN
   sum <= a + b;
END adder1;
-----------------------------------------
```

Additional Examples (Adder) …

```
------ Solution 2: out=INTEGER -----------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
------------------------------------------
ENTITY adder2 IS
   PORT ( a, b : IN SIGNED (3 DOWNTO 0);
          sum : OUT INTEGER RANGE -16 TO 15);
END adder2;
------------------------------------------
ARCHITECTURE adder2 OF adder2 IS
BEGIN
   sum <= CONV_INTEGER(a + b);
END adder2;
------------------------------------------
```

Thanks for your attention

Next session

4 Operators and Attributes

4.1 Operators

4.2 Attributes

4.3 User-Defined Attributes

4.4 Operator Overloading

4.5 GENERIC

4.6 Examples

4.7 Summary