BASIC VHDL LANGUAGE ELEMENTS AND SEMANTICS

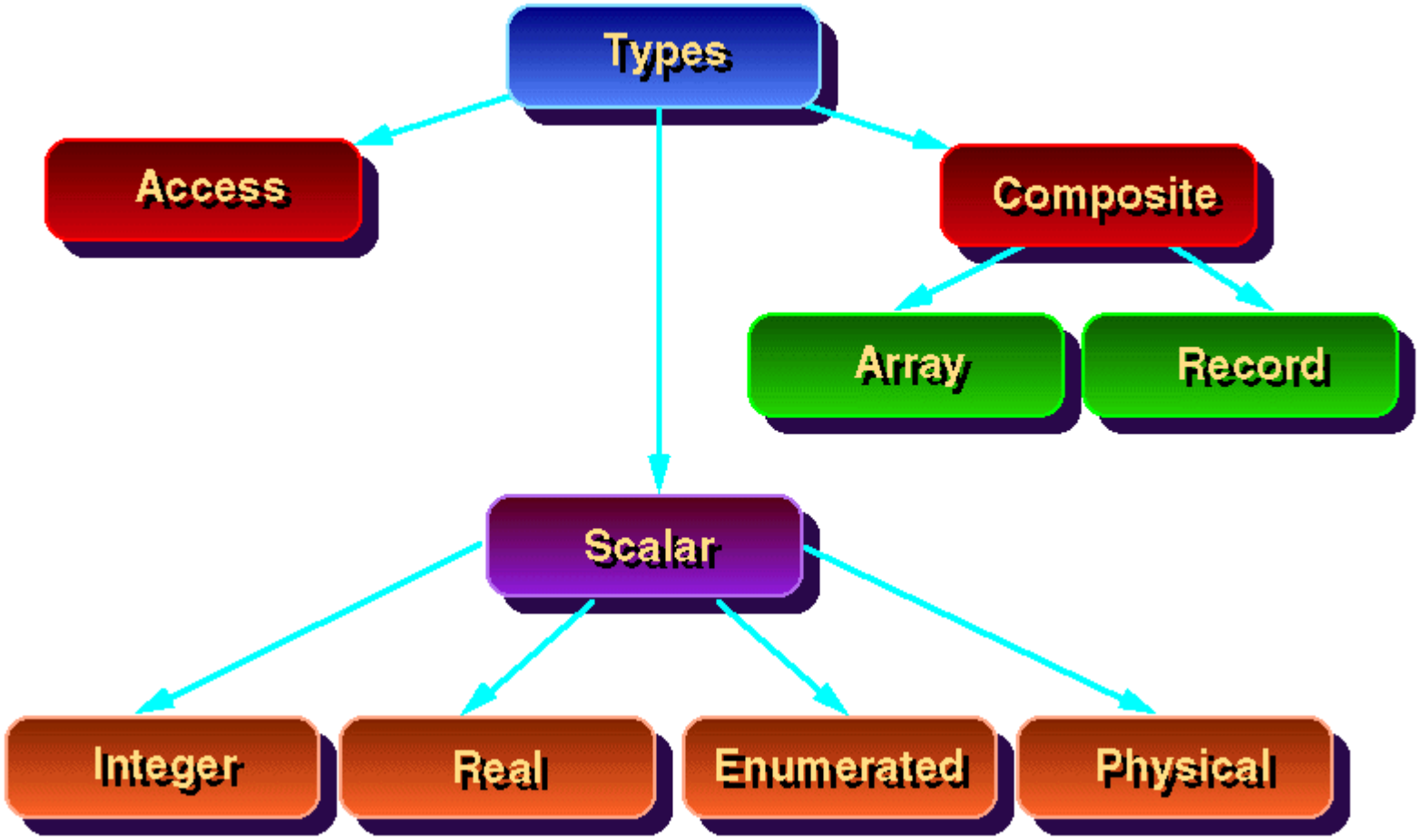
Lecture 7 & 8

Dr. Tayab Din Memon

Outline

- Data Objects
- Data Types
- Operators
- Attributes

VHDL Data Types



VHDL Data Objects

- Signal
- Constant
- Variable
- File

VHDL Data Objects

- Signal
  - An object with a current value and projected (future) values. The projected values can be changed, as many times as desired, using signal assignment statements.
- Constant
  - An object whose value cannot be changed after it is initially specified
- Variable
  - Variables are only used inside the process statement or subprograms (functions and procedures), and must be declared in corresponding declaration regions. A variable's value can be changed, as many times as desired, using variable assignment statements
- File
  - An object that consists of a sequence of values of a specified type

Efficient use of variables: Especially notice how the variable (tmp) is passed to the port signal (C) at the end of the loop

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

entity variabletest is
      port (A: in std_logic_vector(7 downto 0);
               C: out std_logic);
      end variabletest;

architecture vartest of variabletest is

begin
    vtst: process (a)  -- a is a sensitivity list
          variable tmp:std_logic; -- variable declaration region
    begin
      tmp:='1'; -- immediate varaible assignment
          for i in 7 downto 0 loop
                tmp:=A(i) and tmp;
        end loop;
        C<=tmp; -- Passing the variable to a signal
      end process;

end vartest;
```

VHDL Data Types: Scalar

- VHDL is a strongly typed language (you cannot assign a signal of one type to the signal of another type)
- Scalar Types
  - Bit – the only values allowed here are 0 or 1
    - port (I_1,I_2: in bit; I_3: out bit)
  - Boolean – this type has two values: false (0) or true (1)
    - port (I_1,I_2: in bit; I_3: out Boolean)
  - Integer – Covers all the integer values; can be negative or positive
    - port (I_1: in natural; I_2: in bit; I_3: integer)
  - Real – accepts the fraction as well (i.e. 0.45, -5.2E-10)
    - port (I_1: in natural; I_2: in real; I_3: out integer);
  - Character – report (“Variable x is greater than y”);

VHDL Data Types: Scalar (cont…)

- Scalar Type
  - Physical – values that can be measured in units
    - constant delay_inv : time :=1ns;
  - Severity type – used with assert statement

```
-- severity type definition

assert (Flag_full = false);
report "The stack is full";
severity failure
```

VHDL Data Types: composite (cont…)

- Composite Types
  - Bit_vector Type – represents an array of bits
    - port (I1: in bit_vector(3 downto 0); I2: out bit_vector (3 downto 0));
  - Array Types – declared by using the predefined word *array*
    - subtype wordN is integer

```
subtype wordN is integer;
type intg is array (7 downto 0) of wordN;
.........
variable memory: intg;
```

VHDL Data Types: composite (cont…)

- Composite Types
  - Record Types – An object of record type is composed of elements of the same or different types
- Access Types
  - File Types
    - Object file read/write

```
-- Record Type definition
Type forecost is
record
Tempr : integer range -100 to 100;
Day   : real;
Cond  : bit;
end   record;
..............
variable temp : forecast
```

VHDL Data Types: Signed/Unsigned (cont…)

- Signed
  - signed is a numeric type
  - declared in the external package numeric_std
  - Variable d: signed (3 downto 0) :=1010;
- Unsigned
  - represents unsigned integer data in the form of an array of std_logic
  - part of package numeric_std
  - Variable g : unsigned (3 downto 0) :=1010;

Operations with signed/unsigned data types

Example – I:

```
library ieee;
use ieee.std_logic_1164.all;
-- defined arithmatic library
-- for signed/unsigned data types
-- opertions
use ieee.std_logic_arith.all;

entity signedoperation is
port (x,y: in signed (3 downto 0);
        z: out signed (3 downto 0));
end signedoperation;

architecture dataflow of signedoperation is
begin
z <= x + y; -- legal operation
z <= x xor y; -- illegal operation
-- encountered compilation error
end dataflow;
```

Example – II:

```
library ieee;
use ieee.std_logic_1164.all;
-- extra package
--use ieee.std_logic_unsigned.all;

entity operationvector is
port (x,y: in std_logic_vector (3 downto 0);
        z: out std_logic_vector (3 downto 0));
end operationvector;

architecture dataflow of operationvector is
begin
z <= x + y; -- illegal operation
-- encountered compilation error
z <= x and y; -- legal operation
-- but by uncommenting the arithmatic
-- package both operations will become
-- legal
end dataflow;
```

Pre-Defined data types

- Pre-defined in IEEE 1076 and IEEE 1164 standards
- Package standard of library std:
  - Defines Bit, Boolean, Integer, and Real data types
- Package std_logic_1164 of library ieee:
  - Defines Std_logic and Std_ulogic data types
- Package std_logic_arith of library ieee:
  - Defines signed and unsigned data types plus several data conversion functions
- Package std_logic_signed and std_logic_unsigned of library ieee:
  - Works with signed and unsigned data types of std_logic_vector

IEEE Standard Signal Nine Values

- Logical values versus metalogical values
- The state of std_ulogic value denotes its logical level
- The strength of a std_ulogic value denotes
  - the electrical characteristics of the source that drives the signal
- Deriving strengths:
  - forcing, weak and high impedance

Table: State and strength properties of std_ulogic

Value	State	Strength
U	**Uninitialized**	None
X	**Unknown**	Forcing
0	**0**	Forcing
1	**1**	Forcing
Z	**None**	High impedance
W	**Unknown**	Weak
L	**0**	Weak
H	**1**	Weak
-	Don't care	None

Source: VHDL for engineers

U : uninitialized value is the default value given to all std_ulogic signals before the start of a simulation

Signals driven by active output drivers are referred to as forcing strength signals

User-Defined Data Types (Type & Subtype)

- VHDL allows the user to define his own data types
  - Defined by a reserved keyword ‘type’
  - Can be of any type: scalar or composite type
- A subtype is a type with a constraint
  - Mostly used to define objects of a base type with a constraint

Here newtype is userdefined datatype of type an integer & signal mytype is of type newtype

```
type newtype is integer range 0 to 128
signal mytype: newtype;

subtype ttype is std_logic_vector(3 downto 0);

signal x,y: ttype;
signal ad,ac: ttype;

signal x,y: std_logic_vector (7 downto 0);
signal ad,ac: std_logic_vector (7 downto 0);
```

Signal x, y are of type ttype that is of type std_logic_vector

Data Conversion functions

- VHDL doesn't allow direct operations (arithmetic, logic, et) between data of different types
- Can be done by two ways
  - Write a vhdl code
  - Invoke a pre-defined function
- If data are closely related then ieee.std_logic_1164 provides straightforward conversion (see the code fragment)

Exp: Legal and Illegal operations with subsets

```
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
```

Source: Circuit Design with VHDL by Volnei A. Pedroni

Conversion Functions (cont…)

- Various conversion functions are available in the package ieee.std_arith.all are:
  - conv_integer(p)
  - conv_unsigned(p,b)
  - conv_signed(p,b)
  - conv_std_logic_vector(p,b)

```
library ieee;
use ieee.std_logic_1164.all;
-- defind arithmatic package
use ieee.std_logic_arith.all;

entity dataconversion is
port (x,y: in unsigned (7 downto 0);
      -- input type: unsigned
      z: out std_logic_vector(7 downto 0));
       end dataconversion;

architecture behaviour of dataconversion is
begin
z<=conv_std_logic_vector((x+y),8);
-- addition operation of signed data types
-- x+y is converted to std_logic_vector
-- and passed to z (as defined above).
end behaviour;
```

Data Conversion (Alternative Approach)

- Here the use of ieee.std_logic_signed /unsigned.all package mitigates the operation and no need of conversion functions

```
library ieee;
use ieee.std_logic_1164.all;
-- defind arithmatic package
use ieee.std_logic_arith.all;
-- defined signed/unsigned packages
-- to get rid of from the conversion
-- opertions
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity dataconversionalt is
port (x,y: in unsigned (7 downto 0);
      -- input type: unsigned
      z: out std_logic_vector(7 downto 0));
       end dataconversionalt;

architecture behaviour of dataconversion is
begin
z<= x + y;
-- No conversion operation required due to
-- addition of ieee.std_logic_unsinged/signed.all;
end behaviour;
```

VHDL Operators

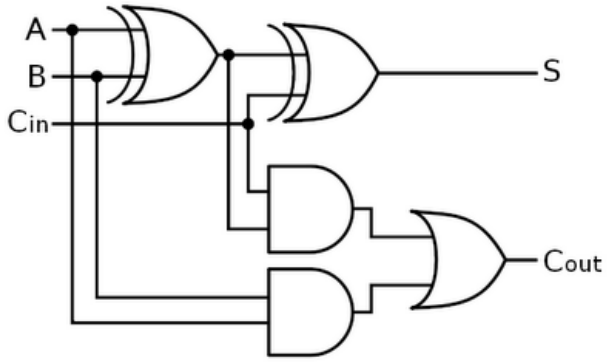
- Assignment
- Logical
- Relational
- Arithmetic
- Shift
- Concatenation

VHDL Operators (cont…)

- Assignment operators are
  - <=, :=, => used to assign values to a signal, variable, or individual vector elements
- Logical
  - AND, NAND
  - OR, NOR
  - XOR, XNOR
  - NOT

VHDL Operators (cont…)

- Arithmetic
  - + (addition)
  - - (subtraction)
  - * (Multiplication)
  - / (Division)
  - ** (Exponentiation)
  - MOD (Modulus)
  - REM (Remainder)
  - ABS (Absolute Value)



VHDL Operators (cont…)

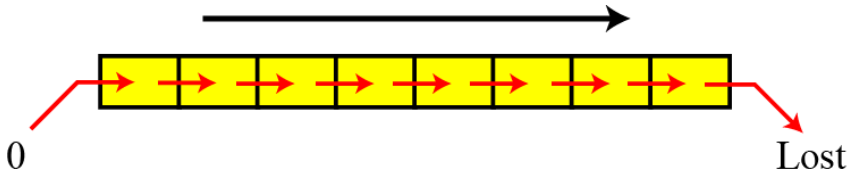
- Relational
  - = (equal to)
  - /= (not equal to)
  - < (less than)
  - <= (less than or equal to)
  - > (greater than)
  - >= (greater than or equal to)

Shift Operators (Cont…)

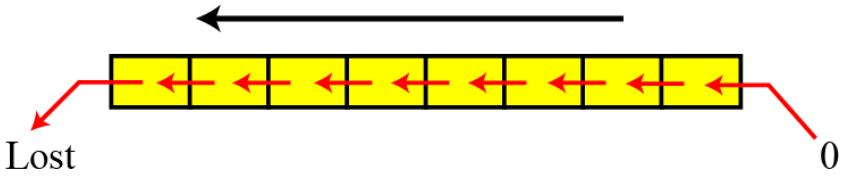
- Shift Operators
  - Shift operations move the bits in a pattern, changing the positions of the bits. They can move bits to the left or to the right. We can divide shift operations into two categories: logical shift operations and arithmetic shift operations.

- Logical Shift Operators – used for unsigned numbers
- Two types of logical shift operations
  - Logical Shift and
  - Logical Circular Shift (Rotate)

Logical Shift Operations



a. Logical right shift

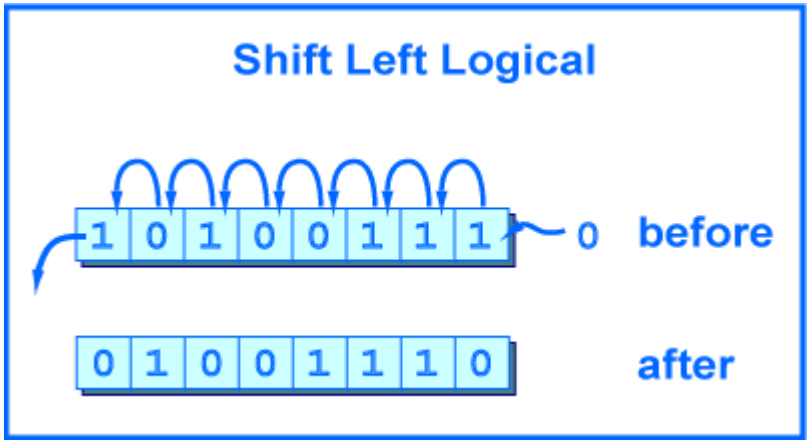


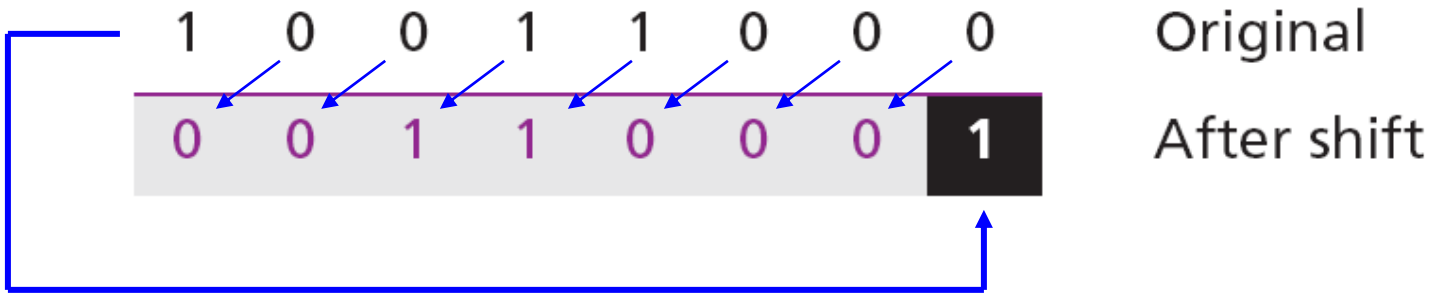
b. Logical left shift

a. Circular right shift

b. Circular left shift

Logical Shift Left and Right

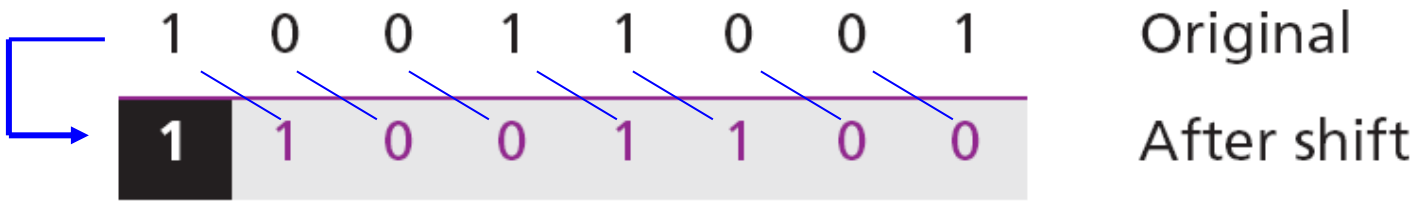




Circular Left Shift Operation

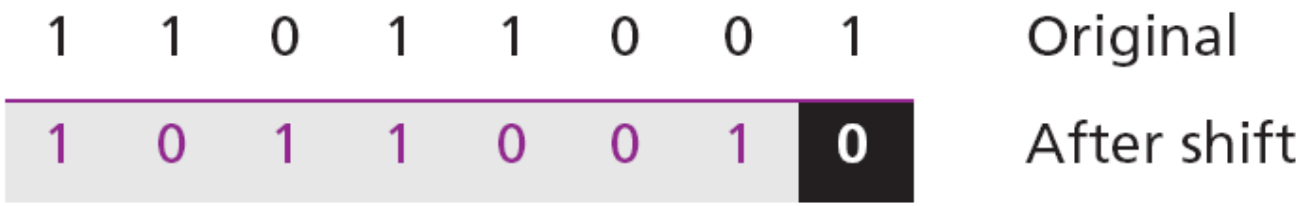
Arithmetic Shift Operations

- Arithmetic operations involve adding, subtracting, multiplying and dividing. We can apply these operations to *integers and floating-point* numbers.
- Following is given arithmetic right shift operation on bit pattern of 10011001. what is difference between original and new number.



Arithmetic shift left – An Example

- Arithmetic Left Shift Operation on bit pattern 11011001 is given below.
- What difference is between original and after shift?



VHDL Shift Operators (cont…)

- Shift Operators
  - sll – shift left logic
  - srl – shift right logic
  - sla – shift left arithmetic
  - sra – shift right arithmetic
  - rol – rotate left logic
  - ror – rotate right logic

- For example A= “010011”
- B<= A sll 2:= 001100
- B<= A srl 2:= 000100
- B<= A sla 2:= 001111
- B<= A sra 2:= 001100
- B<= A rol 2 := 001101
- B<= A srl 2 := 110100

VHDL Attributes

- Data Attributes
- Signal Attributes

Data Attributes

Attribute	Description
D'LOW	Returns lower array index
D'HIGH	Returns upper array index
D'LEFT	Returns leftmost array index
D'RIGHT	Returns rightmost array index
D'LENGTH	Returns array size
D'RANGE	Returns array range
D'REVERSE_RANGE	Returns array range in reverse order
If the signal is of enumerated type then:	
D'VAL (position)	Returns value in the position specified
D'POS (value)	Returns position of value specified
D'LEFTOF (value)	Returns value in the position to the left of the value specified
D'VAL (row, column)	Returns value in the position specified; etc.

Source: circuit design with VHDL

Data attributes – Example

- Consider the following signal:
- Signal X : std_logic_vector (3 downto 0);
  - X’LOW = 0, X’HIGH = 3, X’LEFT = 3,
  - X’RIGHT = 0, X’LENGTH=4,
  - X’RANGE (3 downto 0),
  - X’REVERSE_RANGE = (0 to 3).

Signal Attributes

Attribute	Description
s'EVENT	Returns TRUE when an event occurs on s
s'STABLE	Returns TRUE if no even has occurred on s during the optional time interval t
s'ACTIVE	Returns TRUE when a transaction (assignment) occurs on s (value might not change)
s'QUIET	Returns TRUE if no transaction or event occurred on s during the optional time t
s'LAST_VALUE	Returns the value of s before the last event
s'LAST_EVENT	Returns the time elapsed since the last event of s
s'LAST_ACTIVE	Returns the time elapsed since the last transaction (assignment)

Source: circuit design with VHDL

Operator Overloading

- User-defined operators are so called operator overloading in one way
- The ‘+’ operator is defined by IEEE 1076 standard to operate on numeric types (integers, floating point, and physical types) but not with enumeration types like std_logic or bit_vector.
- To introduce a new kind of addition by the operator ‘+’, supporting bit_vector and std_logic types will be called a operator overloading

Example – Operator Overloading

User-defined *functions* will be discussed later on

Exp: Addition of an integer to a binary 1-bit number

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

entity operatoroverl is
end operatoroverl;

architecture overloading of operatoroverl is

Function "+" (a: integer; b: bit) return integer is
begin
  if (b='1') then return a+1;
  else return a;
  end if;
end "+";

signal inp1, outp: integer range 0 to 15;
signal inp2: bit;

begin
outp<= 5 + inp1 + inp2;

end overloading;
```

GENERIC

- As the name suggests, GENERIC is a way of specifying a generic parameter
- The purpose is to confer the code more flexibility and reusability
- A GENERIC statement must be declared in the entity part as:
  - GENERIC (parameter_name: parameter_type := parameter_value);
  - For example: generic (size: natural := 4);
- More than one generic parameters can be defined in an ENTITY in a same way

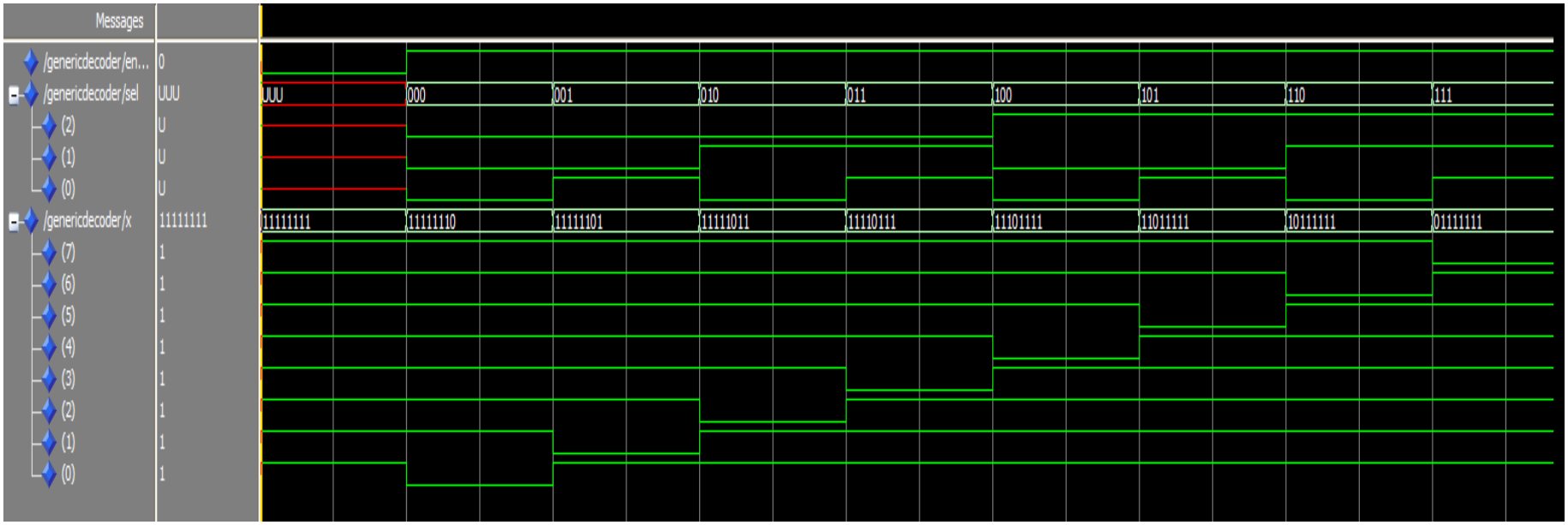
One Final Example: Generic Decoder

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

entity genericdecoder is
generic (m    : integer := 3;
         size : integer := 8);
port (enable: in std_logic;
      sel    : in std_logic_vector (m-1 downto 0);
      x      : out std_logic_vector (size-1 downto 0));
    end genericdecoder;

architecture behavioural of genericdecoder is

begin
  process (enable, sel)
variable temp1 : std_logic_vector (x'high downto 0);
-- or can be (size-1 downto 0);
variable temp2 : integer range 0 to x'high;
    begin
      temp1:= (others => '1');
      -- it means temp1 is always 1
      temp2 := 0;
      if (enable ='1') then
        for i in sel'range loop
          -- can be i in 0 to m-1 (as m defines sel range)
          if (sel(i)='1') then
            temp2:=2*temp2+1;
          else
            temp2:=2*temp2;
          end if;
        end loop;
        temp1(temp2):='0';
      end if;
      x<=temp1;
    end process;
  end behavioural;
```

Use of GENERIC, Operator (+), and signal/ variable assignment symbols as well

Waveforms given on the second slide

Waveform of 3x8 decoder



It can easily be seen that after enable has been asserted, only one output bit is turned low step-by-step.

END OF THE LECTURE 7 & 8

Thanks for your patience