CPE 462
VHDL: Simulation and Synthesis

Topic #04 - c) Operators and Attributes

Nuno Alves (nalves@wne.edu), College of Engineering

Still a foundation

- **Operators** and **attributes** are still "foundations"
- We need to master all foundations before we can actually do some useful designs

Operators

VHDL operators

An operator is an operation between two (or more) data elements.

Here are the different operators:

- Assignment operators ←
- Logical operators ← We've seen these before
- Arithmetic operators ←
- Relational operators
- Shift operators
- Concatenation operators

Assignment operators

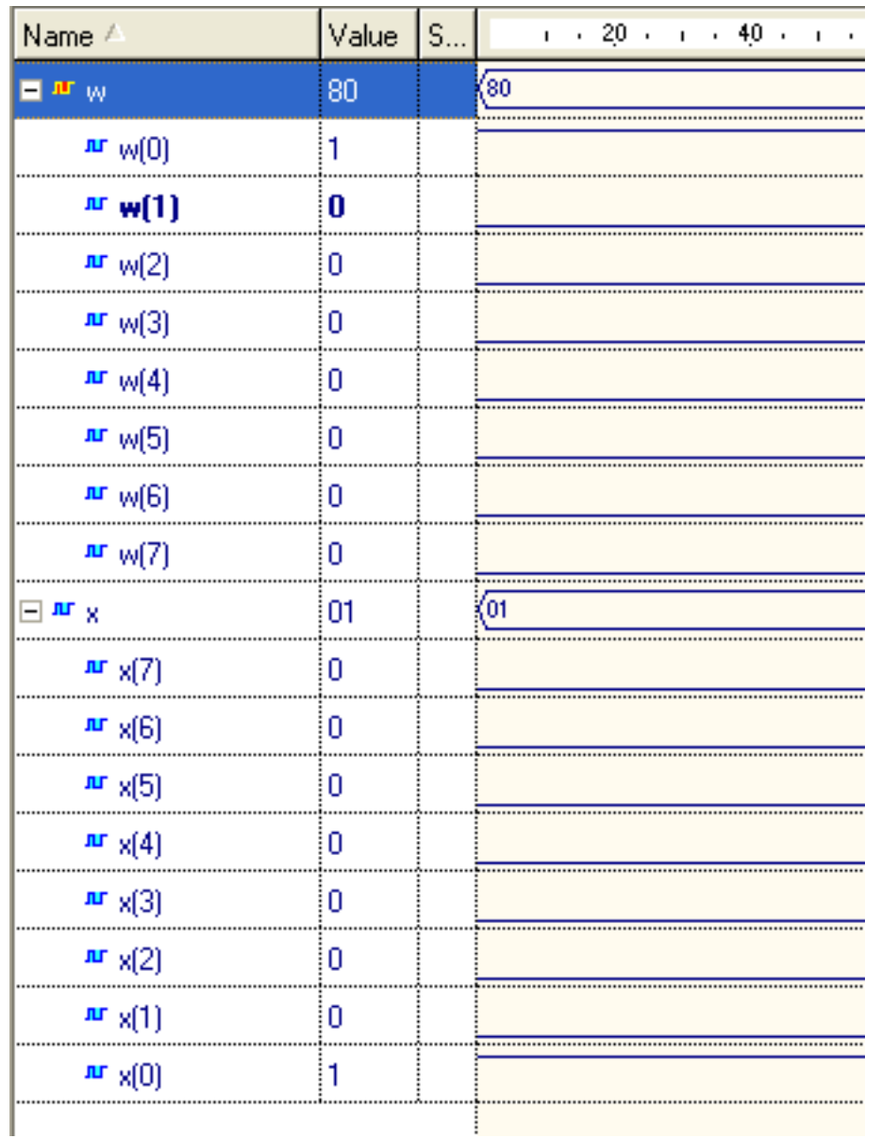
- Are used to assign values to signals, variables, and constants.
- We will talk about variables soon enough
- Here are all assignment operators:
  - <= Used to assign a value to a SIGNAL
  - := We have used this to specify initial values. It is also used to assign a value to a VARIABLE, CONSTANT, or GENERIC.
  - => Used to assign values to individual vector elements or with OTHERS.

Assignment examples

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
-- leftmost bit is the MSB
signal x: std_logic_vector (7 downto 0);

-- rightmost bit is the MSB
signal y: std_logic_vector (0 to 7) := “00001111”;

begin

    y <= “11110000”;
    -- LSB of x is set to 1... rest is set to 0
    x <= (0 =>'1', OTHERS =>'0');

end architecture;
```

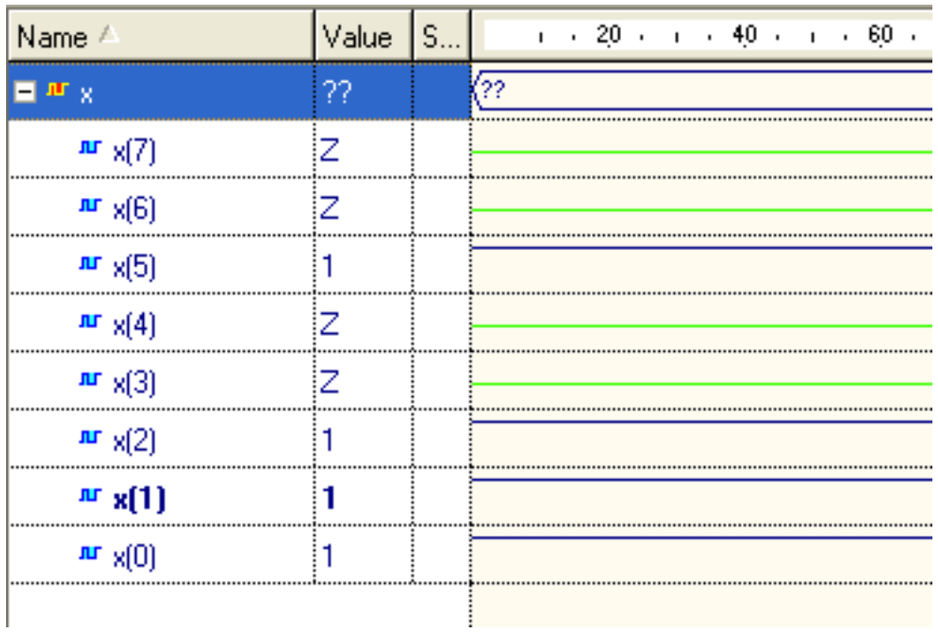
More assignment examples

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
    --rightmost bit is MSB
    signal w: std_logic_vector(0 to 7);

    --leftmost bit is MSB
    signal x: std_logic_vector(7 downto 0);

begin
    -- least significant bit is 1
    -- all others are 0
    w <= (0 =>'1', OTHERS =>'0');

    -- least significant bit is 1
    -- all others are 0
    x <= (0 =>'1', OTHERS =>'0');
end architecture;
```



The => operator is very useful

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
-- leftmost bit is the MSB
signal x: std_logic_vector (7 downto 0);

begin

    -- position 2,1 and 0 are set to 1
    -- position 5 is set to 1
    -- everything else is set to Z
    x <= (2 downto 0 =>'1', 5=> '1', OTHERS =>'Z');

end architecture;
```



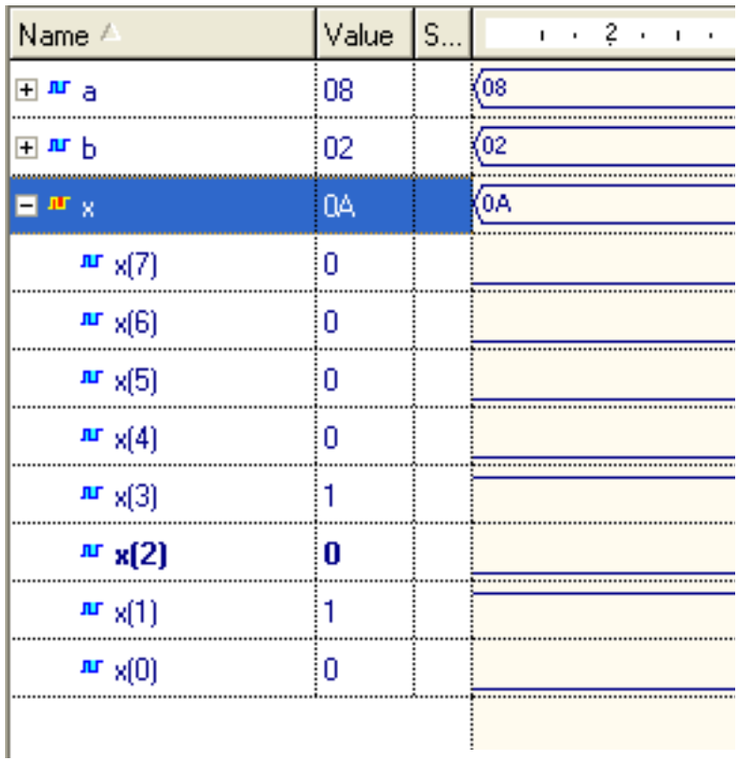
Logical operators

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
-- leftmost bit is the MSB
signal x: std_logic_vector (7 downto 0);
signal y: std_logic_vector (7 downto
0):="00011110";
signal z: std_logic_vector (7 downto
0):="00000000";

begin
     x <= NOT(y) AND z;
end architecture;
```

You've done this before

- Used to perform logical operations
- Data must be of type BIT, STD_LOGIC, or STD_ULOGIC (or, obviously, their respective extensions, BIT_VECTOR, STD_LOGIC_VECTOR, or STD_ULOGIC_VECTOR).
- NOT, AND, OR, NAND, NOR, XOR, XNOR
- The NOT operator has precedence over the others

Arithmetic operators




```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal b: signed (7 downto 0);
signal x: signed (7 downto 0);

begin
     a <= x"08"; --hexadecimal
     --a <= "000001000"; --same as above
     b <= x"02";
     x <= a + b;
end architecture;
```

Other operators →

+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation
MOD	Modulus
REM	Remainder
ABS	Absolute value

Comparison operators

- Used for making comparisons between two data types
- The relational (comparison) operators are:
  - = Equal to
  - /= Not equal to
  - < Less than
  - > Greater than
  - <= Less than or equal to
  - >= Greater than or equal to

Comparison examples

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
    port (clock  : in std_logic;
          output : out integer);
end entity;

architecture myarch of test is
-- leftmost bit is the MSB
signal x: std_logic_vector (7 downto 0):="00011110";
signal z: std_logic_vector (7 downto 0):="00000000";

begin
    process(clock)
    begin
        if (x/=z) then output <= 1;
        end if;
        if (x>=z) then output <= 2;
        end if;
    end process;
end architecture;
```

We haven't fully discussed processes... but you should know what it does.

What will be the output?

Shift operators

- Used for shifting data
- Their syntax is the following: <left operand> <shift operation> <right operand>. The left operand must be of type **BIT_VECTOR**, while the right operand must be an INTEGER (+ or - in front of it is accepted).
- The shift operators are:
  - sll : Shift left logic – positions on the right are filled with '0's
  - srl : Shift right logic – positions on the left are filled with '0's

Examples of shift operators

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
-- leftmost bit is the MSB
signal a: bit_vector (7 downto 0):="00001111";
signal x: bit_vector (7 downto 0):="00000000";
signal y: bit_vector (7 downto 0):="00000000";

begin

    x <= a sll 3;
    y <= a srl 3;

end architecture;
```

requires **bit_vector**

shift **a** 3 positions to the left

Time	Delta	a	x	y
100.000 ns	0	00001111	01111000	00000001

Shift operators types

- Since shift operators require a bit_vector and an integer, you will probably have to perform some sort of conversion...

Conversions supported by std_logic_1164 package

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*)

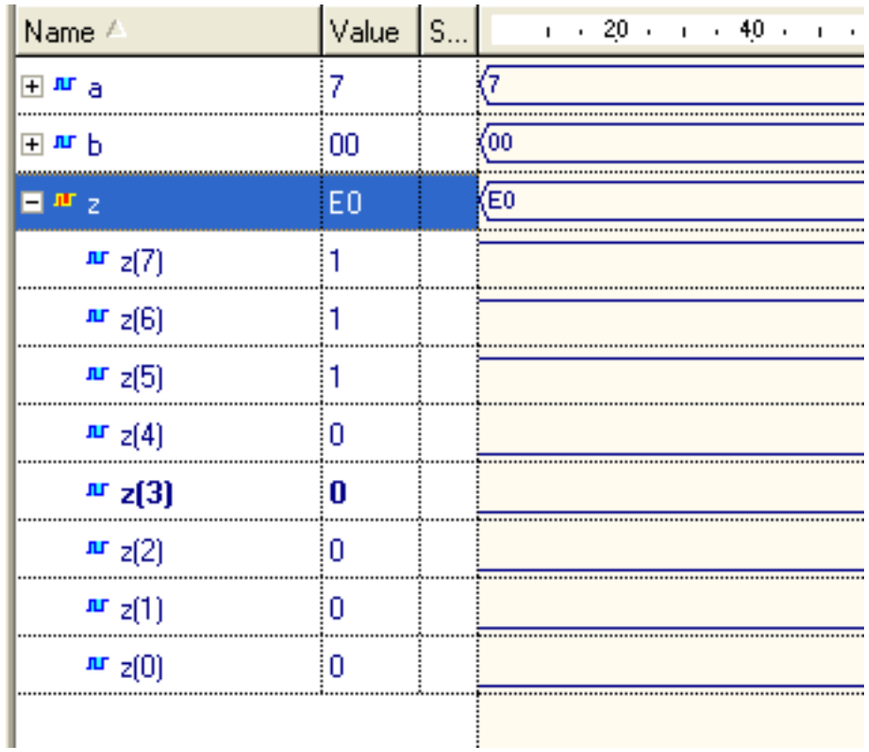
Examples of shift operators with some conversion

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
signal   std_logic_inp   : std_logic_vector(7 downto 0) :="00001100";
signal   bit_vec_inp     : bit_vector(7 downto 0);

signal   bit_vec_outp    : bit_vector(7 downto 0);
signal   std_logic_outp  : std_logic_vector(7 downto 0);

begin

    bit_vec_inp    <= to_bitvector(std_logic_inp);
    bit_vec_outp   <= bit_vec_inp sll 3;
    std_logic_outp <= to_stdlogicvector(bit_vec_outp);

end architecture;
```

Concatenation operators

- The & operator appends bit_vectors or std_logic_vectors

```
entity test is
end entity;

architecture myarch of test is
signal a: bit_vector (2 downto 0):="111";
signal b: bit_vector (4 downto 0):="00000";
signal z: bit_vector (7 downto 0);

begin

    z <= a & b;

end architecture;
```



Attributes

Data attributes

- Data attributes return specific properties of an element of data.

Data attributes

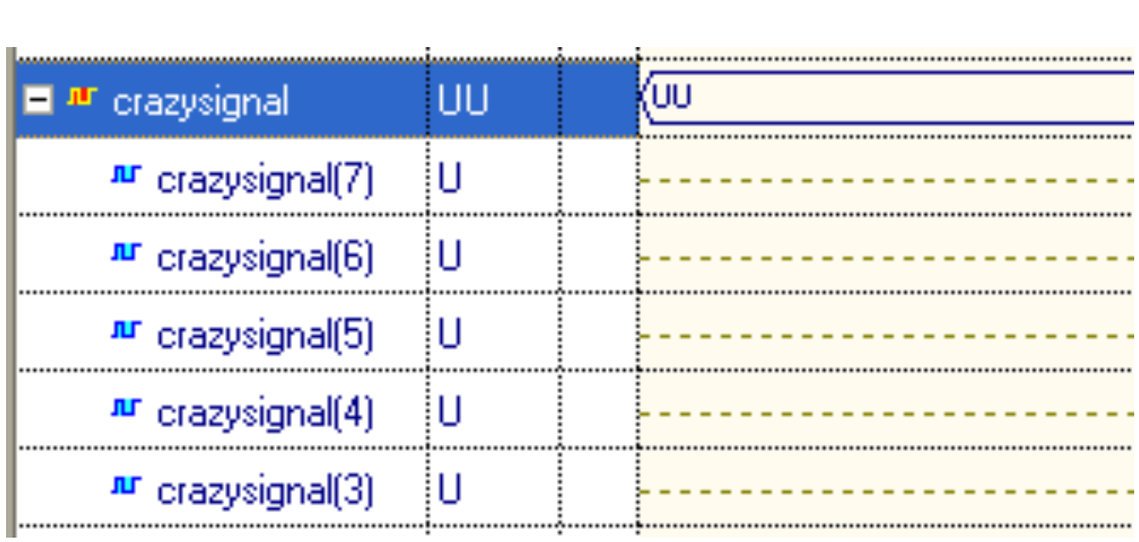
The pre-defined, synthesizable data attributes are the following:

- 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 vector size
- d'RANGE: Returns vector range
- d'REVERSE_RANGE: Returns vector range in reverse order

Examples of data attributes

```
LIBRARY ieee; use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
signal crazysignal: std_logic_vector (7 downto 3);
signal a,b,c,d,e : integer;

begin

    --Returns lower array index (3)
    a<=crazysignal'low;
    --Returns upper array index   (7)
    b<=crazysignal'HIGH;
    --Returns leftmost array index (7)
    c<=crazysignal'LEFT;
    --Returns rightmost array index   (3)
    d<=crazysignal'RIGHT;
    --Returns vector size (5)
    e<=crazysignal'LENGTH;

end architecture;
```

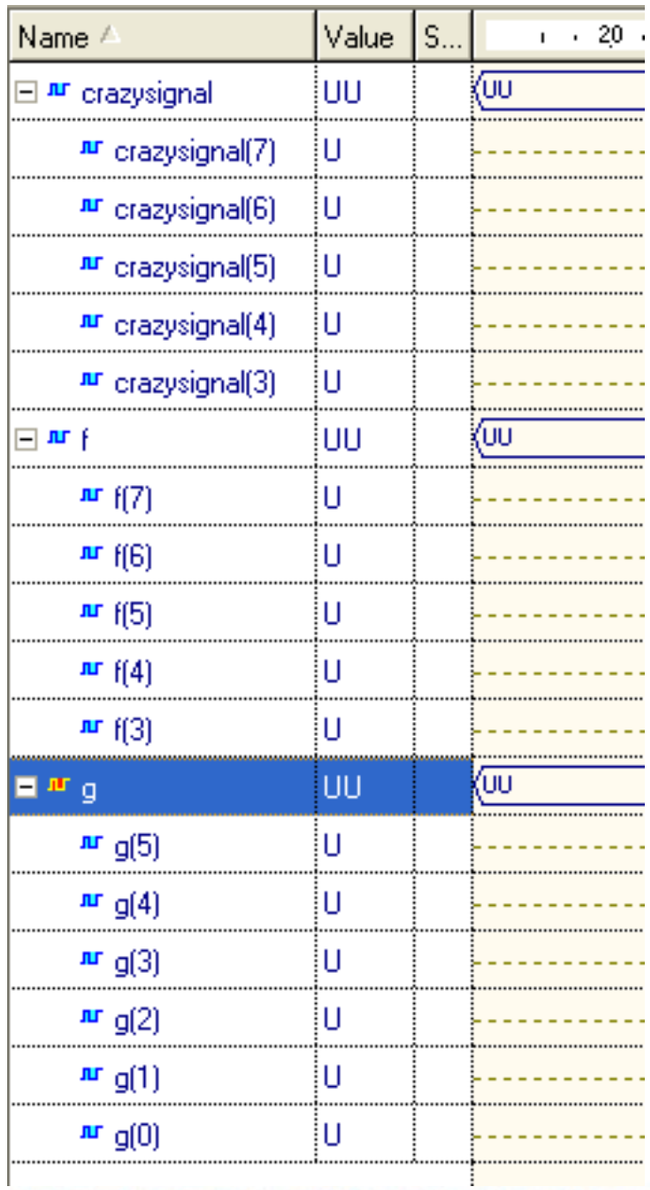
this is weird but it is valid!



More examples of data attributes

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
signal crazysignal: std_logic_vector (7 downto 3);

signal f : std_logic_vector(crazysignal'range);

signal g : std_logic_vector(crazysignal'LENGTH downto 0);

begin

end architecture;
```

even more weird
but it is valid!



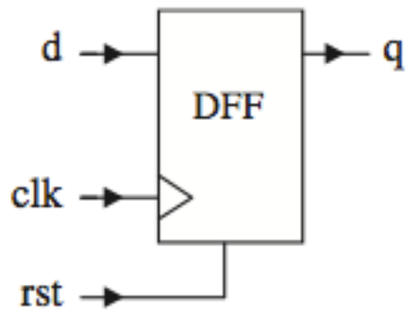
Signal attributes

Let us consider a signal s. Then:

- s'EVENT: Returns true when an event occurs on s
- s'STABLE: Returns true if no event has occurred on s

Signal attribute example

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

entity DFF is
   port (d, clk, rst: IN std_logic;
   q: OUT std_logic);
end DFF;

architecture myarch of DFF is
begin
   PROCESS (rst, clk)
   begin
        if (rst='1') then
            q<='0';
        elsif (clk'event and clk='1') then
            q<=d;
        end if;
   end process;
end myarch;
```



- The PROCESS is entered each time clk or rst changes.
- clk'event means clock has changed... an event has occurred in clk.