Σχεδίαση Ψηφιακών Συστημάτων

Τμήμα Ψηφιακών Συστημάτων,
Πανεπιστήμιο Πελοποννήσου
Αθανάσιος Παπαδημητρίου

Τι είναι η VHDL;

- Ορισμός: VHDL σημαίνει VHSIC Hardware Description Language. Είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται για τη μοντελοποίηση ψηφιακών συστημάτων σε πολλαπλά επίπεδα αφαίρεσης
- Ιστορία: Αναπτύχθηκε τη δεκαετία του 1980 για το Υπουργείο Άμυνας των Η.Π.Α. για την τεκμηρίωση της συμπεριφοράς των ASIC (Ολοκληρωμένα κυκλώματα ειδικών εφαρμογών) και των FPGA
- Χρήση: Χρησιμοποιείται κυρίως για ηλεκτρονικό αυτοματισμό σχεδιασμού για την περιγραφή ψηφιακών συστημάτων όπως τα ολοκληρωμένα κυκλώματα

Οι ρόλοι της VHDL στον σχεδιασμό ψηφιακών συστημάτων

Simulation vs. Synthesis:

- **Simulation**: Χρησιμοποιείται για την επαλήθευση της λογικής και της ορθότητας των ψηφιακών μοντέλων. Μπορεί να προσομοιώσει όλα τα μέρη της γλώσσας.
- **Synthesis**: Μετατρέπει τον κώδικα VHDL σε κύκλωμα (π.χ. ASIC, FPGA). Μόνο οι συνθέσιμες κατασκευές μπορούν να μετατραπούν σε υλικό
- **Industry Applications**: Παραδείγματα όπου χρησιμοποιείται VHDL, όπως τηλεπικοινωνίες, αυτοκινητοβιομηχανία, αεροδιαστημική και ηλεκτρονικά είδη ευρείας κατανάλωσης

Εστίαση σε συνθέσιμη VHDL

Synthesizable VHDL:

- Ορισμός: Υποσύνολο της VHDL που μπορεί να μεταφραστεί απευθείας σε υλικό.
- Synthesizable Constructs: Περιγράφει μόνο τις δυνατότητες που μπορούν να εφαρμοστούν από το υλικό, όπως λογικές πύλες, καταχωρητές και θύρες εισόδου/εξόδου.
- Non-Synthesizable: Περιλαμβάνει καθυστερήσεις χρονισμού, προηγμένες λειτουργίες εισόδου/εξόδου αρχείων και ορισμένους τύπους procedural κώδικα που χρησιμοποιούνται μόνο σε προσομοιώσεις.

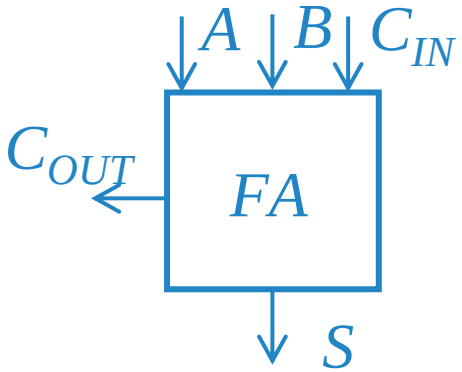
Focus on Synthesizable VHDL - Example

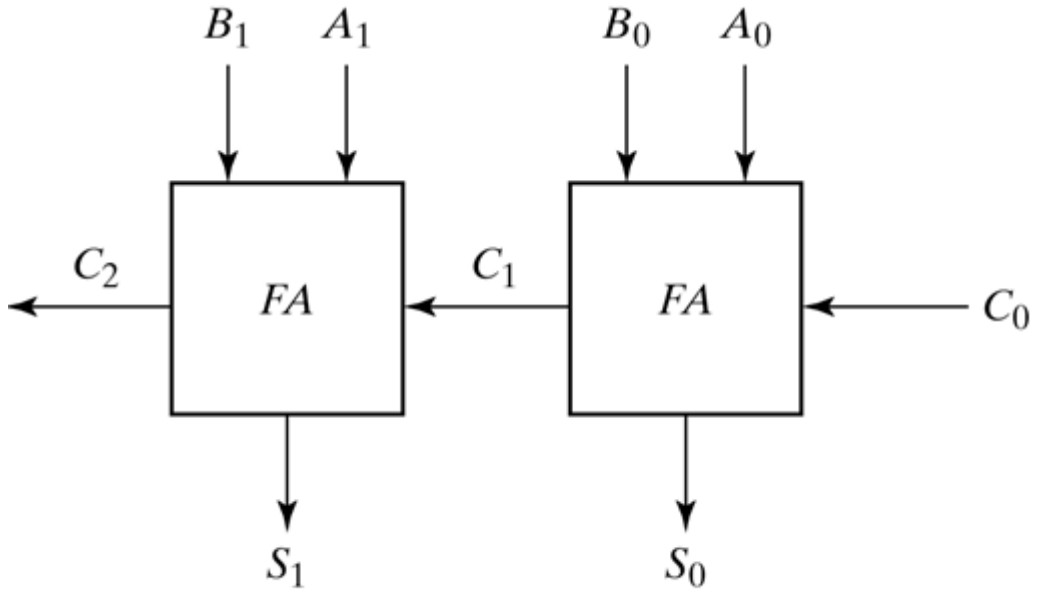
```
1  entity AndGate is
2      port (A, B : in bit; Y : out bit);
3  end AndGate;
4
5  architecture Behavior of AndGate is
6  begin
7      Y <= A and B;
8  end Behavior;
```

```
1  Y <= A and B after 10 ns;
```

Ripple Carry Adder

- Πλήρης αθροιστής: 3 είσοδοι 2 έξοδοι
- Ιεραρχική σχεδίαση με χρήση ενός βασικού κυκλώματος






```
s0 <= (A(0) xor B(0)) xor Cin;
c0 <= (A(0) and B(0)) or (Cin and A(0)) or (Cin and B(0));
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RippleCarryAdder is
    Port ( A : in STD_LOGIC_VECTOR(1 downto 0);
           B : in STD_LOGIC_VECTOR(1 downto 0);
           Cin : in STD_LOGIC;
           Sum : out STD_LOGIC_VECTOR(1 downto 0);
           Cout : out STD_LOGIC);
end RippleCarryAdder;

architecture Behavioral of RippleCarryAdder is
    signal s0, c0, c1: STD_LOGIC;
begin
    -- Full adder for bit 0
    s0 <= (A(0) xor B(0)) xor Cin;
    c0 <= (A(0) and B(0)) or (Cin and A(0)) or (Cin and B(0));

    S <= A XOR B XOR Cin ;
    c0 <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

    -- Full adder for bit 1
    s0(1) <= (A(1) xor B(1)) xor c0;
    c1 <= (A(1) and B(1)) or (Cin and A(1)) or (C0 and B(1));

    -- Output assignments
    Sum(0) <= s0;
    Cout <= c1;

end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RippleCarryAdder_tb is
-- This testbench entity does not need any ports.
end RippleCarryAdder_tb;

architecture sim of RippleCarryAdder_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component RippleCarryAdder
        Port (
            A : in  STD_LOGIC_VECTOR(1 downto 0);
            B : in  STD_LOGIC_VECTOR(1 downto 0);
            Cin : in  STD_LOGIC;
            Sum : out  STD_LOGIC_VECTOR(1 downto 0);
            Cout : out STD_LOGIC
        );
    end component;

    -- Inputs
    signal A : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal B : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal Cin : STD_LOGIC := '0';

    -- Outputs
    signal Sum : STD_LOGIC_VECTOR(1 downto 0);
    signal Cout : STD_LOGIC;

    -- Instantiate the Unit Under Test (UUT)
    begin
        uut: RippleCarryAdder
            Port map (
                A => A,
                B => B,
                Cin => Cin,
                Sum => Sum,
                Cout => Cout
            );

    -- Stimulus process to apply test vectors
    process
    begin
        -- Test case 1
        A <= "00"; B <= "00"; Cin <= '0';
        wait for 10 ns;

        -- Test case 2
        A <= "01"; B <= "01"; Cin <= '0';
        wait for 10 ns;

        -- Test case 3
        A <= "10"; B <= "11"; Cin <= '1';
        wait for 10 ns;

        -- Test case 4
        A <= "11"; B <= "11"; Cin <= '0';
        wait for 10 ns;

        -- Add more test cases as required

        -- Complete the simulation
        wait;
    end process;
end sim;
```

Majority - LUT

- Output is 1 if 2 out of 3 inputs are equal to 1, otherwise 0

x	y	z	out1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Majority_LUT is
    Port (
        x : in STD_LOGIC_VECTOR (2 downto 0);
        out1 : out STD_LOGIC
    );
end Majority_LUT;

architecture Behavioral of Majority_LUT is
begin
    process(x)
    begin
        case (x) is
            when "000" => out1 <= '0';
            when "001" => out1 <= '0';
            when "010" => out1 <= '0';
            when "011" => out1 <= '1';
            when "100" => out1 <= '0';
            when "101" => out1 <= '1';
            when "110" => out1 <= '1';
            when "111" => out1 <= '1';
            when others => out1 <= '0'; -- Covers any unexpected cases
        end case;
    end process;
end Behavioral;
```

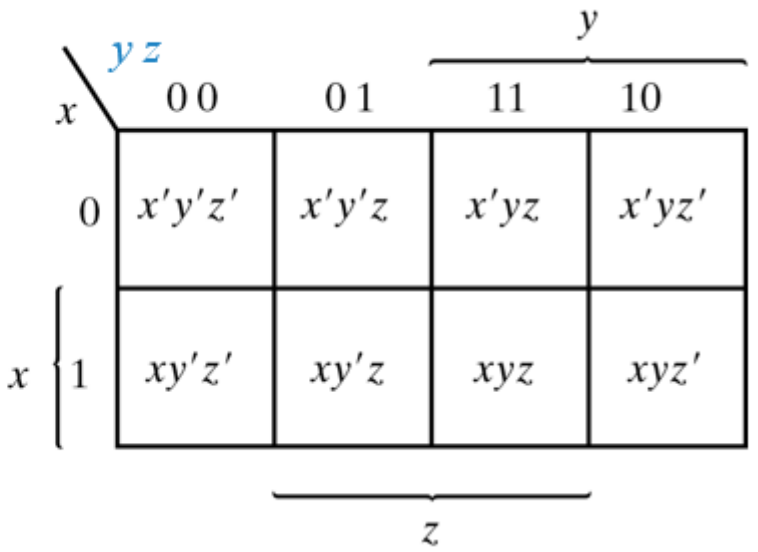
Testbench – Majority - LUT

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Majority_LUT_tb is
-- Testbench has no ports
end Majority_LUT_tb;

architecture sim of Majority_LUT_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component Majority_LUT
        Port (
            x : in  STD_LOGIC;
            y : in  STD_LOGIC;
            z : in  STD_LOGIC;
            out1 : out STD_LOGIC
        );
    end component;

    -- Inputs
    signal x : STD_LOGIC := '0';
    signal y : STD_LOGIC := '0';
    signal z : STD_LOGIC := '0';

    -- Outputs
    signal out1 : STD_LOGIC;

    -- Instantiate the Unit Under Test (UUT)
    begin
        uut: Majority_LUT Port map (
            x => x,
            y => y,
            z => z,
            out1 => out1
        );

    -- Stimulus process to apply test vectors
    process
    begin
        -- Apply test cases
        x <= '0'; y <= '0'; z <= '0'; wait for 10 ns;
        x <= '0'; y <= '0'; z <= '1'; wait for 10 ns;
        x <= '0'; y <= '1'; z <= '0'; wait for 10 ns;
        x <= '0'; y <= '1'; z <= '1'; wait for 10 ns;
        x <= '1'; y <= '0'; z <= '0'; wait for 10 ns;
        x <= '1'; y <= '0'; z <= '1'; wait for 10 ns;
        x <= '1'; y <= '1'; z <= '0'; wait for 10 ns;
        x <= '1'; y <= '1'; z <= '1'; wait for 10 ns;

        -- Complete the simulation
        wait;
    end process;
end sim;
```

x	y	z	out1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	Ελαχιστόροι Όρος	Ονομασία
0	0	0	x'y'z'	m_0
0	0	1	x'y'z	m_1
0	1	0	x'yz'	m_2
0	1	1	x'yz	m_3
1	0	0	xy'z'	m_4
1	0	1	xy'z	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



Majority - Boolean

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Majority is
    Port (
        x : in  STD_LOGIC;
        y : in  STD_LOGIC;
        z : in  STD_LOGIC;
        out1 : out STD_LOGIC
    );
end Majority;

architecture Behavioral of Majority is
begin
    -- Output is '1' if two or more inputs are '1'
    out1 <= (x AND y) OR (y AND z) OR (z AND x);
end Behavioral;

```

x	y	z	out1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Majority – Boolean - Testbench

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Majority_tb is
-- Testbench has no ports
end Majority_tb;

architecture sim of Majority_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component Majority
        Port (
            x : in  STD_LOGIC;
            y : in  STD_LOGIC;
            z : in  STD_LOGIC;
            out1 : out STD_LOGIC
        );
    end component;

    -- Inputs
    signal x : STD_LOGIC := '0';
    signal y : STD_LOGIC := '0';
    signal z : STD_LOGIC := '0';

    -- Outputs
    signal out1 : STD_LOGIC;

    -- Instantiate the Unit Under Test (UUT)
    begin
        uut: Majority Port map (
            x => x,
            y => y,
            z => z,
            out1 => out1
        );

    -- Stimulus process to apply test vectors
    process
    begin
        -- Apply test cases
        x <= '0'; y <= '0'; z <= '0'; wait for 10 ns;
        x <= '0'; y <= '0'; z <= '1'; wait for 10 ns;
        x <= '0'; y <= '1'; z <= '0'; wait for 10 ns;
        x <= '0'; y <= '1'; z <= '1'; wait for 10 ns;
        x <= '1'; y <= '0'; z <= '0'; wait for 10 ns;
        x <= '1'; y <= '0'; z <= '1'; wait for 10 ns;
        x <= '1'; y <= '1'; z <= '0'; wait for 10 ns;
        x <= '1'; y <= '1'; z <= '1'; wait for 10 ns;

        -- Complete the simulation
        wait;
    end process;
end sim;
```

Carry Lookahead Adder

- Σήματα διάδοσης (propagation) και γέννησης (generation) κρατουμένου
- Για κάθε FA το κρατούμενο C_{i+1} είναι ίσο με 1 αν μια από τις ακόλουθες συνθήκες είναι αληθής:
  - $x_i = 1 \; και \; y_i = 1$

  ή
  - $(x_i = 1 \; \eta \; y_i = 1) \; και \; c_i = 1$
- Συνολικά: $c_{i+1} = x_i y_i + c_i(x_i + y_i)$
- Θέτουμε: $g_i = x_i y_i$ και $p_i = x_i + y_i$
- Άρα: $c_{i+1} = g_i + p_i \; c_i$
- Τριών επιπέδων: ένα για τα p και γ και δύο για τα αθροίσματα γινομένων
- Κάποια απλοποίηση? C0=?

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + p_1(g_0 + p_0 c_0) \\
&= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + p_2 \, (g_1 + p_1 g_0 + p_1 p_0 c_0) \\
&= g_2 + p_2 \, g_1 + p_2 p_1 g_0 + p_2 p_1 \, p_0 c_0 \\
c_4 &= g_3 + p_3 c_3 \\
&= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\
&= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}
$$

VHDL Entity Declaration

- **Τι είναι μια VHDL Entity?**
  - **Δήλωση οντότητας: Καθορίζει την εξωτερική διεπαφή μιας μονάδας υλικού**
  - **Συστατικά:**
    - **Όνομα οντότητας: Αναγνωριστικό για τη λειτουργική μονάδα**
    - **Περιγραφή θυρών: Περιγράφει σήματα εισόδου και εξόδου**

Introduction to VHDL Entity Declaration

- Entity Name:
  - Καθορίζει το όνομα της υπο-μονάδας
- Περιγραφή θυρών:
  - port_name
  - mode: Κατεύθυνση (πχ., in, out, inout)
  - data_type: (e.g., bit, std_logic)

```
entity entity_name is
  port(
    port_name1: mode data_type;
    port_name2: mode data_type;
    ...
    port_nameN: mode data_type
  );
end entity_name;
```

```
entity Adder8Bit is
  port(
    A    : in  std_logic_vector(7 downto 0); -- 8-bit input A
    B    : in  std_logic_vector(7 downto 0); -- 8-bit input B
    Cin  : in  std_logic;                    -- Carry-in
    Sum  : out std_logic_vector(7 downto 0); -- 8-bit sum
output
    Cout : out std_logic                     -- Carry-out
  );
end Adder8Bit;
```

VHDL Architecture Body

- **Τι είναι η Αρχιτεκτονική VHDL;**
- **Architecture Body**: Περιγράφει την εσωτερική υλοποίηση μιας οντότητας
  - **Όνομα Αρχιτεκτονικής:** Αναγνωριστικό για την αρχιτεκτονική
  - **Όνομα οντότητας:** Συσχετίζει την αρχιτεκτονική με μια οντότητα
  - **Δηλώσεις:** Εσωτερικά σήματα, εξαρτήματα και σταθερές
  - **Concurrent Statements (**Ταυτόχρονες δηλώσεις): Διεργασίες και εκχωρήσεις σημάτων

Simplified Syntax of VHDL Architecture Body

```
architecture arch_body of entity_module is
  declarations;
begin
  concurrent_statement1;
  concurrent_statement2;
  concurrent_statement3;
  ...
end arch_body;
```

- **Όνομα Αρχιτεκτονικής:** Καθορίζει το όνομα της αρχιτεκτονικής (arch_body)
- **Ονομα οντότητας:** Συνδέει την αρχιτεκτονική με μια συγκεκριμένη οντότητα (entity_module)
- **Declarations:** Εσωτερικά σήματα, μεταβλητές και εξαρτήματα
- **Concurrent Statements:** Ταυτόχρονες δηλώσεις: Περιγράφει τη συμπεριφορά χρησιμοποιώντας διεργασίες και εκχωρήσεις σημάτων (concurrent_statement1, concurrent_statement1, κ.λπ.)

Example of 8-bit Adder Architecture

- **Όνομα Αρχιτεκτονικής**: Behavioral
- **Όνομα οντότητας**: Adder8Bit
- **Δηλώσεις**:
  - Το σήμα sum_temp
- **Concurrent Statements**:
  - Διεργασία για την πράξη της πρόσθεσης των A, B, και Cin
  - Το αποτέλεσμα ανατίθεται στο Sum και το Cout

```
architecture Behavioral of Adder8Bit is
  signal A_ext, B_ext: std_logic_vector(8 downto 0);
  signal Cin_ext: std_logic_vector(8 downto 0);
  signal sum_temp: std_logic_vector(8 downto 0); -- 9-bit
temporary sum
begin
  A_ext <= '0' & A;
  B_ext <= '0' & B;
  Cin_ext <= "0000000" & Cin;
  process(A_ext, B_ext, Cin_ext)
  begin
    -- Addition of A and B with Cin
    sum_temp <= A_ext + B_ext + Cin_ext;
     -- Assign the result to the outputs
    Sum  <= sum_temp(7 downto 0);  -- Assign lower 8 bits to Sum
    Cout <= sum_temp(8);                -- Assign the carry out
  end process;
end Behavioral;
```

Operators in Standard VHDL

Operator	Description	Operand A Data Type	Operand B Data Type	Result Data Type
a ** b	Exponentiation	integer	integer	integer
abs a	Absolute value	integer		integer
not a	Logical negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	Multiplication	integer	integer	integer
a / b	Division	integer	integer	integer
a mod b	Modulo	integer	integer	integer
a rem b	Remainder	integer	integer	integer

Operators in Standard VHDL

Operator	Description	Operand A Data Type	Operand B Data Type	Result Data Type
+ a	Identity	integer		integer
- a	Negation	integer		integer
a + b	Addition	integer	integer	integer
a - b	Subtraction	integer	integer	integer
a & b	Concatenation	1-D array of elements	1-D array of elements	1-D array of elements
a or b	Logical OR	boolean, bit, bit_vector	boolean, bit, bit_vector	boolean, bit, bit_vector
a and b	Logical AND	boolean, bit, bit_vector	boolean, bit, bit_vector	boolean, bit, bit_vector
a xor b	Logical XOR	boolean, bit, bit_vector	boolean, bit, bit_vector	boolean, bit, bit_vector

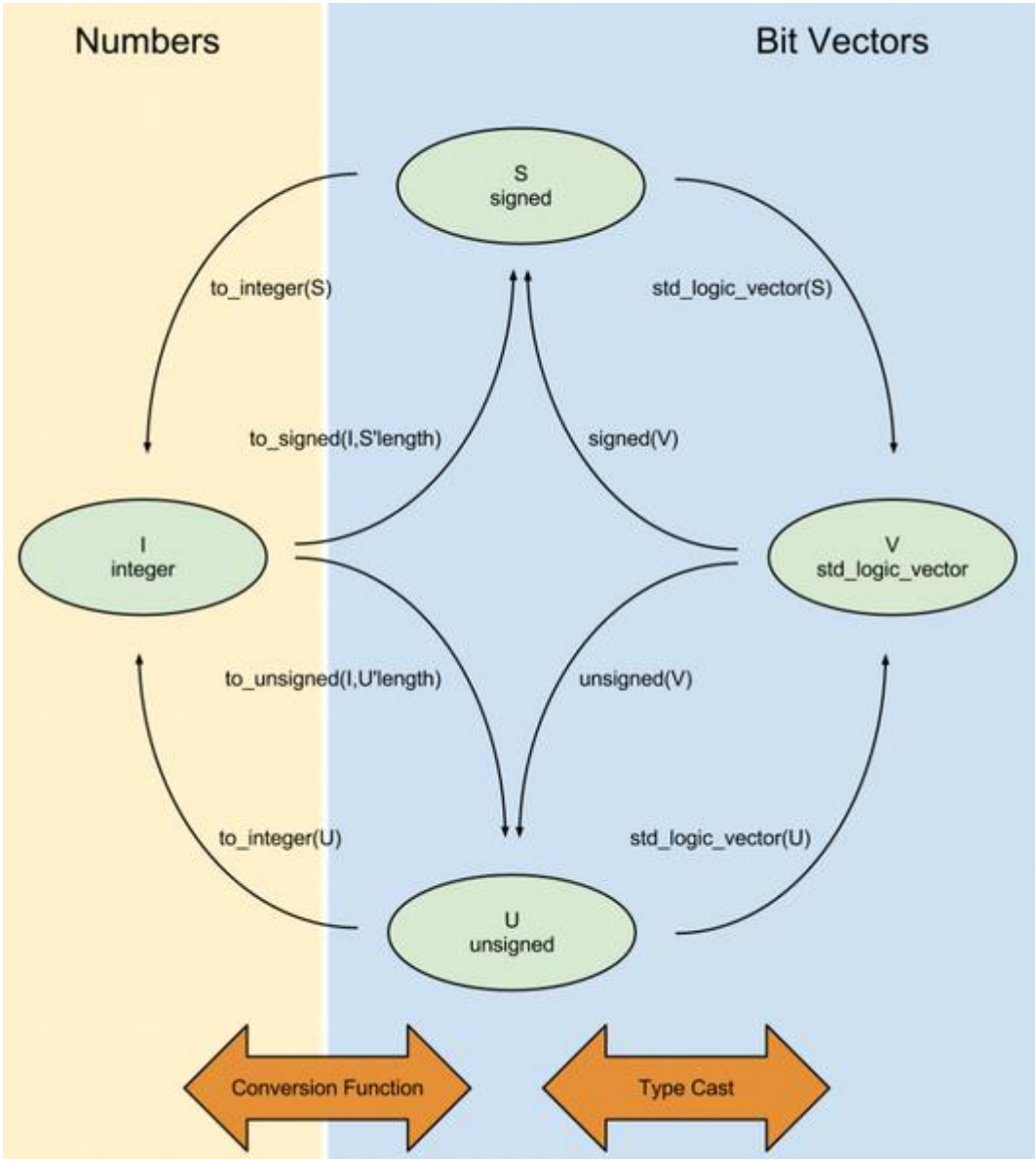
Synthesizable Signal Types in VHDL

- Synthesizable Types: Types that can be implemented in hardware using VHDL
- Common Synthesizable Types
  - Bit – Represents a single binary value ('0' or '1'):
    - signal bit_signal : bit;
  - std_logic – Multi-valued logic type, more versatile than bit – Values include '0', '1', 'Z', 'X', etc
    - signal std_logic_signal : std_logic;
  - std_logic_vector – Array of std_logic values – Used for buses and multi-bit signals
    - signal std_logic_vector_signal : std_logic_vector(7 downto 0);

Συνθέσιμοι τύποι της VHDL

- Συνθέσιμοι τύποι: Τύποι που μπορούν να υλοποιηθούν σε υλικό
  - Ακέραιοι – Χρησιμοποιούνται για παράδειγμα για μετρητές, δείκτες, …
    - signal integer_signal : integer range 0 to 255 := 0;
  - unsigned and signed – Αναπαριστούν προσημασμένους και απρόσημους αριθμούς – Ορίζονται στη βιβλιοθήκη numeric_std
    - signal unsigned_signal : unsigned(7 downto 0) := (others => '0');
    - signal signed_signal : signed(7 downto 0) := (others => '0');

Συναρτήσεις μετατροπής της VHDL



Συνδιαστικές Διεργασίες στη VHDL

- **Τι είναι μια συνδιαστική διεργασία?**
- **Συνδιαστηκή Διεργασία**: Μια διεργασία (process) της VHDL που περιγράφει συνδιαστική λογική, όπου οι έξοδοι εξαρτώνται μόνο από τις τρέχουσες εισόδους
- **Βασικά χαρακτηριστικά**:
  - Δεν υλοποιούν στοιχεία μνήμης
  - Η έξοδοι αλλάζουν όταν αλλάζουν οι είσοδοι
  - Χρησιμοποιούνται για τη σχεδίαση κυκλωμάτων όπως: multiplexers, adders, …

Structure of a Combinational Process

- Λίστα ευαισθησίας:
  - Λίστα σημάτων που ενεργοποιούν τη διεργασία όταν αλλάξουν τιμές

  - A, B, Sel: Είσοδοι
  - Y: Έξοδοι
  - Λίστα ευαισθησίας: (A, B, Sel)

```
process (sensitivity_list)
begin
  -- Combinational logic
end process;
```

```
process (A, B, Sel)
begin
  if Sel = '1' then
    Y <= A;
  else
    Y <= B;
  end if;
end process;
```

Case Statement in VHDL

- Τι είναι μια δήλωση Case;
  - Δομή ελέγχου της VHDL
    - Ελέγχει μια έκφραση και επιλέγει μία από τις πολλές εναλλακτικές με βάση την τιμή της έκφρασης

```
case expression is
  when value1 =>
    -- Sequential statements
  when value2 =>
    -- Sequential statements
  ...
  when others =>
    -- Sequential statements
end case;
```

Case Statement in VHDL

- expression: Η τιμή που πρέπει να αξιολογηθεί
- when value: Οι πιθανές τιμές της expression
- others: Προαιρετική συνθήκη για μη καθορισμένες τιμές

```
case expression is
  when value1 =>
    -- Sequential statements
  when value2 =>
    -- Sequential statements
  ...
  when others =>
    -- Sequential statements
end case;
```

Παράδειγμα δήλωσης Case: 4-to-1 Multiplexer

- Όνομα Αρχιτεκτονικής: Behavioral
- Όνομα Οντότητας: Mux4to1
- Λειτουργία:
  - Δήλωση case που επιλέγει μια απο τις εισόδους (I0 έως I3) σύμφωνα με το σήμα Sel
  - Η προεπιλεγμένη ενέργεια ορίζεται από το when others

```
architecture Behavioral of Mux4to1 is
begin
  process (I0, I1, I2, I3, Sel)
  begin
    case Sel is
      when "00" => Y <= I0;
      when "01" => Y <= I1;
      when "10" => Y <= I2;
      when "11" => Y <= I3;
      when others => Y <= (others => '0');
    end case;
  end process;
end Behavioral;
```

Δήλωση If της VHDL

- Τι είναι μια δήλωση If;
  - Δήλωση If: Μια δομή ελέγχου της VHDL
    - Εκτελεί δηλώσεις με βάση την αξιολόγηση μιας ή περισσότερων συνθηκών
    - Μπορεί να περιλαμβάνει πολλές συνθήκες και ένθετες δηλώσεις

Δομή της δήλωσης If

- Components:
  - if: Πρώτη συνθήκη
  - elsif: Προαιρετικές συνθήκες
  - else: Προαιρετική συνθήκη else για όλες τις υπόλοιπες περιπτώσεις

```
if condition1 then
  -- Sequential statements
elsif condition2 then
  -- Sequential statements
else
  -- Sequential statements
end if;
```

Παράδειγμα δήλωσης If: 4-to-1 Multiplexer

- Όνομα Αρχιτεκτονικής: Behavioral
- Όνομα Όντότητας: Mux4to1
  - Η δήλωση if ελέγχει την τιμή του σήματος Sel και θέτει την αντίστοιχη είσοδο (I0 έως I3) στην έξοδο Y
  - Η συνθήκη else χειρίζεται τυχόν απροσδιόριστες τιμές του Sel

```
architecture Behavioral of Mux4to1 is
begin
  process (I0, I1, I2, I3, Sel)
  begin
    if Sel = "00" then
      Y <= I0;
    elsif Sel = "01" then
      Y <= I1;
    elsif Sel = "10" then
      Y <= I2;
    elsif Sel = "11" then
      Y <= I3;
    else
      Y <= (others => '0');
    end if;
  end process;
end Behavioral;
```

VHDL Testbenches

- Ορισμός: Ένα VHDL testbench χρησιμοποιείται για την επαλήθευση της λειτουγικότητας ενός κυκλώματος που έχει περιγραφεί με VHDL
- Σκοπός: Η προσομοίωση του σχεδίου για να διαπιστωθεί ότι λειτουργεί όπως απαιτείται, πριν την υλοποίηση του κυκλώματος
- Περιλαμβάνει σήματα εισόδου για τη λειτουργία του κυκλώματος και για την παρατήρηση των εξόδων του

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MUX4to1 IS
   PORT(
      a, b, c, d : IN  STD_LOGIC;
      sel : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
      y : OUT STD_LOGIC
   );
END MUX4to1;

ARCHITECTURE behavior OF MUX4to1 IS
BEGIN
   PROCESS(a, b, c, d, sel)
   BEGIN
      CASE sel IS
         WHEN "00" => y <= a;
         WHEN "01" => y <= b;
         WHEN "10" => y <= c;
         WHEN "11" => y <= d;
         WHEN OTHERS => y <= 'X';
      END CASE;
   END PROCESS;
END behavior;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_MUX4to1 IS
END tb_MUX4to1;

ARCHITECTURE behavior OF tb_MUX4to1 IS
  -- Component Declaration
  COMPONENT MUX4to1
  PORT(
     a, b, c, d : IN  STD_LOGIC;
     sel : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
     y : OUT STD_LOGIC
  );
  END COMPONENT;

  -- Signals for DUT
  SIGNAL a, b, c, d : STD_LOGIC;
  SIGNAL sel : STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL y : STD_LOGIC;

BEGIN
  -- Instantiate the Unit Under Test (UUT)
  UUT: MUX4to1 PORT MAP (
     a => a,
     b => b,
     c => c,
     d => d,
     sel => sel,
     y => y
  );
```

```
 -- Stimulus process
  stim_proc: PROCESS
  BEGIN
     -- Test vector 1
     a <= '0'; b <= '1'; c <= '0'; d <= '1'; sel <=
"00";
     WAIT FOR 10 ns;

     -- Test vector 2
     sel <= "01";
     WAIT FOR 10 ns;

     -- Test vector 3
     sel <= "10";
     WAIT FOR 10 ns;

     -- Test vector 4
     sel <= "11";
     WAIT FOR 10 ns;

     -- End of test
     WAIT;
  END PROCESS;
END behavior;
```