Digital Design Group Project
EENG 28010

Introduction to VHDL

Dr. Faezeh Arab Hassani
Department of Electrical and Electronic Engineering

Outline

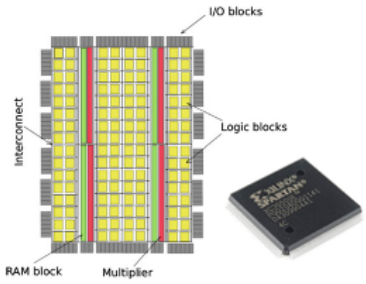
- Why VHDL?
- VHDL Module
- Numeric Types

Why VHDL for Digital System Design?

Transistor Scale of Integration

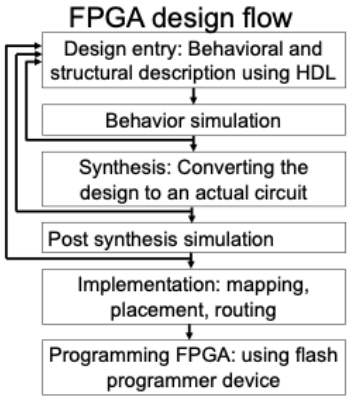
Scale of integration	Number of transistors
Small scale integration (SSI)	2-64
Medium scale integration (MSI)	64-2000
Large scale integration (LSI)	2000-64,000
Very large scale integration (VLSI)	64,000-2,000,000
Ultra large scale integration (ULSI)	<2,000,000

Simplified FPGA Xilinx Spartan-3 block structure



- More transistors → more logic elements and complex digital systems (e.g. field programmable gate arrays (FPGA))

Why VHDL for Digital System Design? (cont.)



- Hardware description language (HDL) (e.g. VHDL) allows designing and debugging of a digital system at a higher level of abstraction
- VHDL was developed as a uniform method for digital system design and standardized by IEEE in 1987

Outline

- Why VHDL?
- VHDL Module
- Numeric Types

VHDL Module

- The general structure of a VHDL module consists of: An entity and an architecture description.

Entity: Defining the black box picture of the module and its external interconnections

Keyword

```
entity two_gates is
  port (A, B, D: in bit; E: out bit);
end two_gates;
```

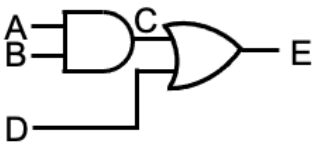
Data type Mode type

Mode types:
in: Input signal to the entity
out: Output signal to the entity
inout: Input-output signal to the entity
buffer: Allows internal feedbacks inside the entity

Data types:
bit: 0 or 1 **boolean**: True or false
integer: All integer values
std_logic: Nine values (U, X, 0, 1, Z, W, L, H, '-')
bit_vector: A vector of bits
std_logic_vector: A vector of type std_logic

VHDL Module (cont.)

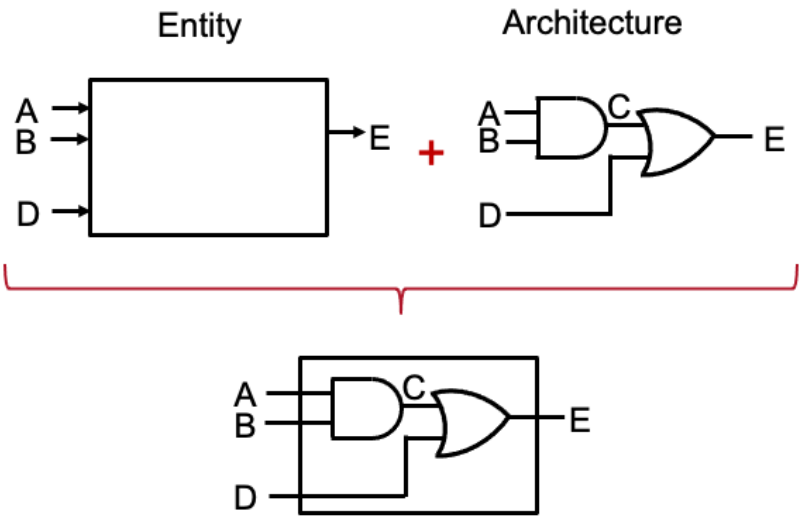
Architecture: Defining the components and internal signals




```
architecture gates of two_gates is
signal C: bit;
begin
  C <= A and B after 5 ns;  -- concurrent statements
  E <= C or D after 5 ns;
end gates;
```

4

VHDL Module (cont.)



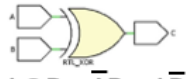
VHDL Libraries and Packages

- VHDL libraries and packages are used to extend the functionality of VHDL by defining types, functions, components, and overloaded operators.
- Some operations are only valid for certain data types.
- If an operation is needed for the other types, one has to use function "overloading" to create an "overloaded" operator.
- How to use library and packages:

```
library IEEE;
use IEEE.std_logic_1164.all; -- defines logic operations, 9 values, std_logic_vectors but not arithmetic operations
use IEEE.numeric_std.all; -- IEEE standard, unsigned and signed binary numbers, std_logic_vectors
use IEEE.std_logic_vector; -- not IEEE standard
```

Levels of Abstraction for Architecture

Behavioral



$A \oplus B = \overline{A}B + A\overline{B}$

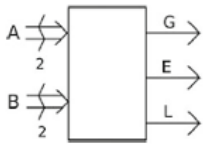
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

```
entity XOR_G is
  port(
  A : in   std_logic;
  B : in   std_logic;
  C : out  std_logic
  );
end XOR_G;

architecture behavioral of XOR_G is
begin
  process(A,B)
  begin
    if A = B then
      C <= '0';
    else
      C <= '1';
    end if;
  end process;
end architecture behavioral;
```

Sequential statements

Data flow

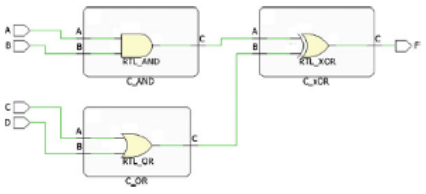


2-bit comparator

```
entity comparator_2bits is
  port(
  A : in   std_logic_vector(1 downto 0);
  B : in   std_logic_vector(1 downto 0);
  G : out  std_logic;
  E : out  std_logic;
  L : out  std_logic
  );
end comparator_2bits;

architecture data_flow of comparator_2bits is
begin

  G <= '1' when A > B else '0';
  E <= '1' when A = B else '0';
  L <= '1' when A < B else '0';

end architecture data_flow;
```

Concurrent statements

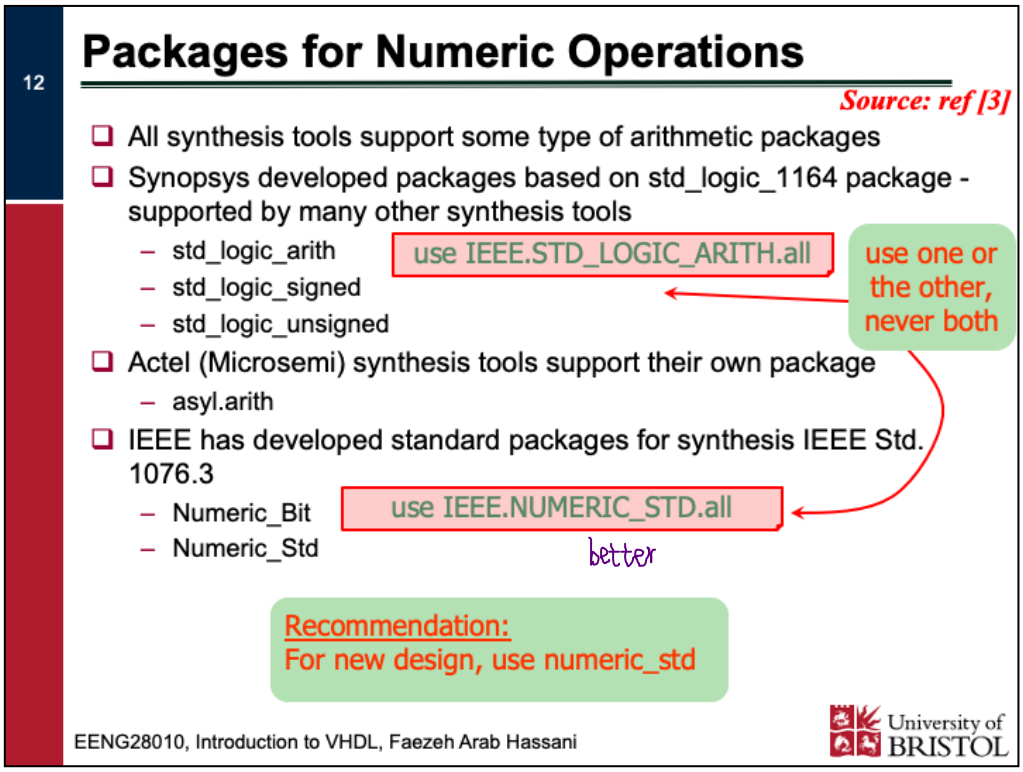
Structural




```
entity example is
    port (
    A : in   std_logic;
    B : in   std_logic;
    C : in   std_logic;
    D : in   std_logic;
    F : out  std_logic
        );
end example;

architecture structural of example is
component AND_G
    port(
    A : in   std_logic;
    B : in   std_logic;
    C : out  std_logic
    );
end component;
component OR_G
    port(
    A : in   std_logic;
    B : in   std_logic;
    C : out  std_logic
    );
end component;
component XOR_G
    port(
    A : in   std_logic;
    B : in   std_logic;
    C : out  std_logic
    );
end component;

signal SI0, SI1, SI2, SI3, SI4 : std_logic;

begin
    DUT1  : AND_G  port map(A,B,SI0);
    DUT2  : XOR_G  port map(SI0,SI1,F);
    DUT3  : OR_G   port map(C,D,SI1);

end structural;
```

Outline

- Why VHDL?
- VHDL Module
- Numeric Types



All synthesis tools support some type of arithmetic package - that is, a package that contains operators like addition, subtraction, multiplication, etc.

When synthesis tools began to appear, each tool vendor created their own arithmetic package. Synopsis created packages for general arithmetic, and signed and unsigned arithmetic called std_logic_arith, std_logic_unsigned, and std_logic_signed. These became “de facto” standards and other tool vendors began to support them. In 1997, the IEEE came out with a set of standard packages for synthesis. These are defined in IEEE std. 1076.3-1997 IEEE Standard VHDL Synthesis Packages.

These packages are called numeric_bit, for arithmetic operations on standard VHDL BIT types, and numeric_std for arithmetic operations on std_logic types.

Unsigned and Signed Types

Source: ref [3]

- Used to represent numeric values:

TYPE	Value	Interpretation
unsigned	0 to 2^N - 1	
signed	- $2^{(N-1)}$ to $2^{(N-1)}$ - 1	2's complement

1's complement: By subtracting each binary number from 1

2' complement: by adding 1 to 1's complement

- Usage similar to std_logic_vector:

```
signal A     : unsigned (3 down to 0);
signal B     : signed (3 downto 0);
signal C     : std_logic_vector (3 downto 0);
....
A <= "1111";     -- decimal 15
B <= "1111";     -- decimal –1
C <= "1111";     -- decimal 15 if using std_logic_unsigned
                 -- (Synopsys library which extends std_logic_arith)
```

Unsigned / Signed vs std_logic_vector

Source: ref [3]

- Type definitions identical to std_logic_vector

TYPE	Value	Interpretation
type UNSIGNED is array (natural range <>) of std_logic;		
type SIGNED is array (natural range <>) of std_logic;		

- How are the types distinguished, and how do they generate unsigned and signed arithmetic?
- For each operator, a unique function is called
  - i.e. operators are overloaded

```
function "+" (L, R: signed) return signed;
function "+" (L, R: unsigned) return unsigned;
-- several more
```

Overloading Examples

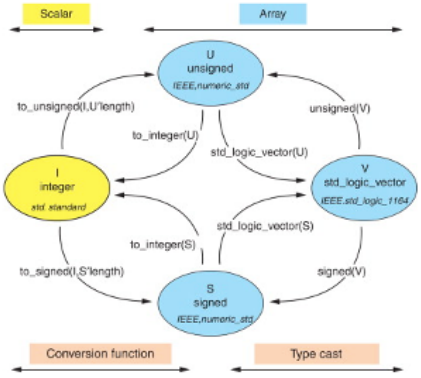
Source: ref [3]

```
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
-- only legal if using std_logic_unsigned (which you should not!)
J_slv <= K_slv + L_slv ;

-- Illegal cannot mix different array sizes
Y_sv <= A_uv - B_uv; -- want signed result
-- Solution is to carry out type conversions
```

Type Conversions

Source: ref [3]

- What conversion functions are needed?
  - Signed & Unsigned (elements) <=> Std_Logic
  - Signed & Unsigned <=> Std_Logic_Vector
  - Signed & Unsigned <=> Integer
  - Std_Logic_vector <=> Integer
- VHDL Built-In Conversions
  - Automatic Type Conversion
  - Conversion by Type Casting
- Conversion functions located in Numeric_Std



Top-Down Digital VLSI Design: From Architectures to Gate-Level Circuits and FPGAs, Chapter 4 - Circuit Modeling with Hardware Description Languages, 2015, pages 179-300.

Automatic Type Conversion: Unsigned, Signed <=> std_logic

Source: ref [3]

- Two types convert automatically when both are subtypes of the same type

```
subtype std_logic is resolved std_ulogic;
```

  - Converting between std_ulogic and std_logic is automatic

```
-- all legal
B_sul <= K_sv(7) ;
L_uv(0) <= C_sl;
M_slv(2) <= N_sv(2);
Y_sl <= A_sl and B_sul and J_uv(2) and K_sv(7) and M_slv(2);
```

vector
logic
ulogic
signed

Type Casting: Unsigned, Signed <=> Std_Logic_Vector

Source: ref [3]

- Use type casting to convert equal sized arrays when:
  - Elements have a common base type (i.e. std_logic)
  - Indices have a common base type (i.e. Integer)

```
A_slv <= std_logic_vector( B_uv ) ;
C_slv <= std_logic_vector( D_sv ) ;
G_uv <= unsigned( H_slv ) ;
J_sv <= signed( K_slv ) ;
```

```
-- Eg: unsigned – unsigned = signed
signal X_uv, Y_uv : unsigned (6 downto 0) ;
signal Z_sv : signed (7 downto 0) ;
. . .
Z_sv <= signed('0' & X_uv) - signed('0' & Y_uv) ;
```

Numeric_Std Function Conversions: Unsigned, Signed <=> Integer

Source: ref [3]

- Converting to and from integer requires a conversion function

```
signal A_uv, C_uv : unsigned (7 downto 0) ;
signal Unsigned_int : integer range 0 to 255 ;
signal B_sv, D_sv : signed( 7 downto 0) ;
signal Signed_int : integer range -128 to 127

-- unsigned, signed => integer
Unsigned_int <= TO_INTEGER ( A_uv ) ;
Signed_int <= TO_INTEGER ( B_sv ) ;

-- integer => unsigned, Signed
C_uv <= TO_UNSIGNED ( unsigned_int, 8 ) ;
D_sv <= TO_SIGNED ( signed_int, 8 ) ;
```

Width of array

Strong Typing – Overflow

```
signal A8, B8, Result8 : unsigned(7 downto 0) ;
signal Result9 : unsigned(8 downto 0) ;
signal Result7 : unsigned(6 downto 0) ;
. . .
-- Simple Addition, no carry out
Result8 <= A8 + B8 ;

-- Carry Out in result
Result9 <= ('0' & A8) + ('0' & B8) ;

-- For smaller result, slice input arrays
Result7 <= A8(6 downto 0) + B8(6 downto 0) ;
```

References

All of the material sourced from:

- Jim Lewis (2003), "VHDL Math Tricks of the Trade," *Tutorial at Military and Aerospace Programmable Logic Devices (MAPLD) Conference* [Online]. Available: http://www.gstitt.ece.ufl.edu/vhdl/refs/vhdl_math_tricks_mapld_2003.pdf
- Reference [3] in Assignment 1
- C. H. Roth and L. K. John (2017), "Digital Systems Design Using VHDL", CENGAGE Learning.