Abstraction Hierarchy Concept

- Structural Domain: component described in terms of interconnection of primitives
- Behavioral Domain: component described by defining its input/output response
- Abstraction Hierarchy: set of interrelated representation levels that allow system to be represented in varying amounts of detail
Digital Systems Diagrams

A → B → C
VHDL Design Unit

Interface Specification

to/from other modules

Work Specification
to/from other modules
Entity Statement

- Identifies interface mechanism
- Can also identify constant values
- Interface elements - signals
- No work can occur in entity
- Checking and passive routines allowed
Entity Syntax

entity ENTITY_NAME is
    generic ( GENERIC_SPECS );
    port ( PORT_SPECS );
end {entity}{ENTITY_NAME};
entity NAND_GATE is
  port ( A : in BIT;
         B : in BIT;
         F : out BIT );
end NAND_GATE;

entity NAND_GATE is
  port ( A, B : in BIT;  F : out BIT );
end NAND_GATE;
Architecture Body

- Holds description of the work
- Has access to signals in port, information in generic of entity
- Local elements identified in declaration area
- Description can be behavioral or structural
- Communication only through port
Architecture Syntax

architecture ARCH_NAME of ENT_NAME is
  { declaration area }
begin
  { specification of parallel activities }
end {architecture}{ARCH_NAME};
architecture EQUATION1 of NAND_GATE is
begin
    F <= not ( A and B );
end EQUATION1;

architecture EQUATION2 of NAND_GATE is
begin
    F <= A nand B;
end EQUATION2;
Abstraction Hierarchy Concept

- Structural Domain: component described in terms of interconnection of primitives
- Behavioral Domain: component described by defining its input/output response
- Abstraction Hierarchy: set of interrelated representation levels that allow system to be represented in varying amounts of detail
Digital Systems Diagrams

A → B → C
VHDL Design Unit

Interface Specification

to/from
other modules

to/from
other modules

Work Specification
VHDL Design Unit

Entity Declaration

Architecture Body

other modules to/from other modules
Entity Statement

- Identifies interface mechanism
- Can also identify constant values
- Interface elements - signals
- No work can occur in entity
- Checking and passive routines allowed
entity ENTITY_NAME is
  generic ( GENERIC_SPECS );
  port ( PORT_SPECS );
end {entity}{ENTITY_NAME};
entity NAND_GATE is
  port ( A : in BIT;
        B : in BIT;
        F : out BIT );
end NAND_GATE;

entity NAND_GATE is
  port ( A, B : in BIT;
         F : out BIT );
end NAND_GATE;
Architecture Body

- Holds description of the work
- Has access to signals in port, information in generic of entity
- Local elements identified in declaration area
- Description can be behavioral or structural
- Communication only through port
Architecture Syntax

architecture ARCH_NAME of ENT_NAME is
   { declaration area }
begin
   { specification of parallel activities }
end { architecture } { ARCH_NAME } ;
architecture EQUATION1 of NAND_GATE is
begin
    F <= not ( A and B );
end EQUATION1;

architecture EQUATION2 of NAND_GATE is
begin
    F <= A nand B;
end EQUATION2;
Syntax Details

- Character set - enumeration type
- Identifiers: \{A-Z\}\{A-Z_0-9\}*{A-Z0-9}\n- Use character case to help communication
- Delimiters and compound delimiters
- Comments -- use liberally
More Syntax Details

• Character literal - one char between ‘s
• String literal - chars between “s
• Bit string literal - String literal with base specifier
  – B”10100101”
  – X”A5”
  – O”245”

More Syntax Details

• Abstract literal - numerical value (int, real)
• Decimal literal - standard decimal notation
  – no negative exponents
  – 1st char must be number
  – reals must contain decimal point
  – int examples: 6 86_153 3E3 0 23e0
  – real examples: 2.5 0.0 0.25 256.375 2.0E5
More Syntax Details

• Based literal
• Base included as first chars in literal
• Pound signs used to enclose literal
• Base spec, any exponent, always in base 10
• Int examples: 16#FFF# 2#1111_1111_1111#
• Real examples: 16#0.FFF#e3
  2#1.1111_1111_111#E11
More Syntax Details

• Representations of values: variables and signals
• Variables: hold value, change instantaneously
• Signal: hold waveform, pairs of values and times. Cannot change instantaneously
More Details

• Classes of Data Types
  – Scalar（one value only）
  – Composite（complex objects - array or record）
  – Access（provide access to other types）
  – File（provide access to files）
More Details

• Enumeration type - lists possible values
• Syntax: type TYPE_NAME is ( LIST );
• Examples: type BIT is ( ‘0’, ‘1’);
  type STATE is (S0, S1, S2, S3, S4);
• Position numbers start at 0, increment through the list
More Details

• Attributes contribute information to representation
• Attribute is value, function, type, range associated with various constructs
• Attribute identified by ELEMENT’ATT_NAME
More Details

• Attributes:
  – LEFT, RIGHT    the left (right) bound of the type
  – HIGH, LOW      the upper (lower) bound of the type
  – POS(X)         the position number of X
  – VAL(X)         the value whose position number is X
  – SUCC(X)        value whose position number 1 greater
  – PRED(X)        value whose position number 1 less
Attribute Examples

Type STATE is (S0, S1, S2, S3, S4);

STATE’POS(S2) = 2
STATE’VAL(3) = S3
STATE’LEFT = S0
STATE’RIGHT = S4
STATE’PRED(S3) = S2
STATE’SUCC(S3) = S4
Subtypes: Subsets of Types

- Syntax:
  subtype ST_IDENT is TYPE_NAME [CON];
- CON is range constraint or index constraint
- Examples:
  subtype OUT_ST is STATE range S2 to S4;
  subtype IN_ST is STATE range STATE’LEFT to S2;
Integer Types

• Integer range implementation dependent
  – must be at least -(2^{31}-1) to 2^{31}-1
• Syntax: type NAME is range RANGE;
• Examples:
  type INT_2C is range -32768 to 32767;
  type ADR_BITS is range 31 downto 0;
  type INFO_BITS is range 2 to 9;
More Integer Stuff

• Predefined integer types:
  type INTEGER is <implementation dep>;
  type NATURAL is INTEGER
       range 0 to INTEGER’HIGH;
  type POSITIVE is INTEGER
       range 1 to INTEGER’HIGH;
Floating Point Types

• Same syntax as Integer types
• Examples:
  type PROBABILITY is range 0.0 to 1.0;
  type CHEAP is range 0.01 to 9.99;
  type VCC is range 1.8 to 5.1;
Composite Data Types

- Array: each element same subtype
- Predefined array types:
  - type STRING is array (POSITIVE range <>) of CHARACTER;
  - type BIT_VECTOR is array (NATURAL range <>) of BIT;
- Example:
  - type D_BUS is BIT_VECTOR ( 31 downto 0 );
Return to Architecture Syntax

architecture ARCH_NAME of ENT_NAME is
   { declaration area }
begin
   {concurrent statement }
end {architecture}{ARCH_NAME};
Concurrent Statement

- Block statement
- Process statement
- Component instantiation
- Generate statement
- Concurrent signal assignment statement
- Concurrent assertion statement
- Concurrent procedure call statement
Component Instantiation

COMP_IDENT: COMP_NAME

generic map ( FORMALS => ACTUALS )
port map ( FORMALS => ACTUALS ) ;
Component Instantiation Example

UUT: NAND2

generic map (  
    T_DELAY => 10 ns  
)

    port map (  
        A => X_A,  
        B => X_B,  
        F => X_F  
    ) ;
Signal Assignment Statement

- Syntax: \texttt{SIG\_NAME} \texttt{<=} waveform ;
- Waveform: pairs of values, times
- Concurrent signal assignment
- Sequential signal assignment
- Assignment of value must wait at least 1 delta time
Steps to Create Testbed for Combinational Logic

• Complete system in Design Capture
  – Use Hierarchical Connectors for In, Out
  – Make sure root is set correctly

• Generate VHDL automatically
  – Result of this step: files in ‘genhdl’
  – May need global file as well

• Copy root VHDL file to ‘tb_name’
Steps to Create Testbed for Combinational Logic (cont)

- Edit tb_{name} to:
  - Include empty entity at top of file
  - Add statement “use WORK.all;”
  - Add line: ‘architecture NAME of NAME is’
  - Change entity statement to component statement by changing ‘entity’ to ‘component’ and inserting ‘component’ at end of statement
  - Add signals for driving/observing port elements
  - Change remainder of file to begin/end architecture
Steps to Create Testbed for Combinational Logic (cont)

- Create a process in body of architecture:
  ```
  process
  begin
    more stuff here
  end process;
  ```
  - Add signal assignment statements for all desired patterns
  - After each statement, add: wait for NN ns;