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
- Work Specification

VHDL Design Unit

- Entity Declaration
- Architecture Body
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

definition

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 entity NAND_GATE;

entity NAND_GATE is
    port ( A, B : in BIT;  F : out BIT );
end entity 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 architecture EQUATION1;

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

• Character set - enumeration type
• Identifiers: {A-Z}{A-Z_0-9}*{A-Z0-9}
• 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

• 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

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 <>l
    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: SIG_NAME <= waveform ;
- Waveform: pairs of values, times
- Waveform also name of signal
- Concurrent signal assignment
- Sequential signal assignment
- Assignment of value must wait at least 1 delta time

Example Simple Signal Assignment Statements

F_OUT <= (not A and not B and C ) or (not A and B and not C ) or (A and B and C) or (A and not B and not C ) ;

C_OUT <= ( A and B ) or ( A and C ) or ( B and C ) after 2 * G_DLY;
Conditional Signal Assignment

- Concurrent statement
- Identify options and waveforms
- Syntax:
  \[
  \text{TARGET} \leq \text{waveform1 when COND1 else}
  \text{waveform2 when COND2 else}
  \text{waveform3 when COND3 else}
  \ldots \text{else}
  \text{waveform}_n;
  \]

Selected Signal Assignment

- Concurrent statement
- Identify options and waveforms
- Syntax:
  \[
  \text{with } \text{EXPRESSION select}
  \text{TARGET} \leq \text{waveform1 when } \text{CHOICE}_1,
  \text{waveform2 when } \text{CHOICE}_2,
  \text{waveform3 when } \text{C}_3 \lor \text{C}_4,
  \text{waveform}_k \text{when } \text{C}_k;
  \]
Generate Statement

• Syntax:
  STMT_ID: for ID in RANGE generate
  { concurrent statement }
  end generate;

• Example:
  G_NAME: for I in 0 to 7 generate
    AB(I) <= CD(I) and EF(I);
  end generate;

Process Statement

• Concurrent statement
• Activity in process - sequential
• Delay mechanism user selectable
  – sensitivity list - comma separated list of signals
  – event on any signal in list causes process activity
  – wait statements in process body
• Activity visited once on startup
Process Statement Syntax

PROC_NAME: process( SEN_LIST )

    { declaration area }
begin
    {sequential statement}*
end process {PROC_NAME};

Sequential Statements

- Signal Assignment Statement
- If Statement
- Case Statement
- Loop Statement
- Wait statement
- Assertion Statement
- Report Statement
Sequential Statements (Cont.)

- Variable assignment statement
- Procedure Call Statement
- Next Statement
- Exit Statement
- Null Statement

If Statement Syntax

If BOOL_EXP then
    {sequential statement}
elsif BOOL_EXP then
    {sequential statement}
else
    {sequential statement}
end if;
Case Statement Syntax

```plaintext
  case SEL_EXP is
      when CHOICE => {sequential stmt}
      when CHOICE | CHOICE => {seq stmt}
      when others => {sequential stmt}
  end case ;
```

Loop Statement Syntax

```plaintext
  for ID in RANGE loop
      {sequential statement}
  end loop;

  while BOOL_EXP loop
      {sequential statement}
  end loop;
```
Loop Statement Syntax (Cont.)

go to next iteration:
  next { when \texttt{BOOL\_EXP} } ;

get out of loop:
  exit { when \texttt{BOOL\_EXP} } ;

Wait Statement Syntax

wait on \texttt{SENSITIVITY\_LIST}
  until \texttt{BOOLEAN\_EXP}
  for \texttt{TIME\_EXP} ;

wait on A, B, C for 60 ns;
wait until DATA = FE for 10 ns;
Advanced Features of VHDL:
Overloading

- Subprogram names: correct procedure selected by number, type of parameters:
  - $\text{DFF (CLK, D, Q);}$
  - $\text{DFF (CLK, D, Q, QBAR);}$
  - $\text{DFF (CLK, D, CLR, SET, Q);}$

Advanced Features of VHDL:
Libraries

- Successful analysis: result to library
- Work library - current working info
- Resource libraries - Example: IEEE
- Library contents: primary and secondary units
- Primary units: declarations for entity, pkg, etc
- Secondary units: architectures and pkg bodies
- Libraries have logical, physical names
Advanced Features of VHDL: Configurations

- Simulation after binding to components
- Configuration: method for specifying the models bound to components

Advanced Features of VHDL: File I/O

- To drive model with test vectors
- Information stored on system; read as file
- Files are sequential, regarded as data streams
- Formatted: written by VHDL, not readable
- Text: human readable; use editor/program
- File In or Out, but not both
Advanced Features of VHDL: Text I/O

- type LINE is access STRING;
  type TEXT is file of STRING;

Back to Attributes - More Details

- DELAYED(time) - delayed version of signal
- STABLE(time) - boolean, true when stable
- QUIET(time) - boolean, true when no trans
- EVENT - boolean, true when event just occurred
- TRANSACTION - BIT, toggles on transaction
Subprogram Declaration

- Function - list of input parameters and types, and also the return type
- Input parameters considered constants
- Function can be pure or impure (impure function must contain `impure` in spec)
- Procedure - list of parameters, mode, type
- Procedure inputs - constant; out, inout vars

Advanced Features of VHDL: Libraries

- Successful analysis: result to library
- Work library - current working info
- Resource libraries - Example: IEEE
- Library contents: primary and secondary units
- Primary units: declarations for entity, pkg, etc
- Secondary units: architectures and pkg bodies
- Libraries have logical, physical names
Advanced Features of VHDL: Packages

- Package to hold frequently used declarations
- Package body to hold subprograms
- “use” clause needed to make package visible
- “use” format:
  ```vhdl```
  use LIB_NAME.PKG_NAME.ELE_NAME;
  ```vhdl```
- STANDARD, TEXT_IO, STD_LOGIC

Package Declaration

- Used to declare elements for other associations
- Syntax:
  ```vhdl```
  package PKG_IDENT is
    PKG_DECLARATIVE_ITEM
  end {package}{PKG_IDENT};
  ```vhdl```
Package Declarative Items

- Subprogram declaration
- Type, subtype, constant declaration
- Signal, shared variable declaration
- File, alias declaration
- Component declaration
- Attribute declaration, specification
- Disconnection specification
- Use clause
- Group, group template declaration

Package Body

- Contains work needed in package
- Syntax
  ```
  package body PKG_IDENT is
    PKG_BODYDECLITEM *
  end {package body}{PKG_IDENT};
  ```
Resolution of Signals

- VHDL communication - signals
- Signal generation and drivers
- Process/concurrent statements drive signals
- Multiple processes driving same signal must have resolution mechanism

IEEE - STD_LOGIC_1164

type STD_ULOGIC is ( 'U', -- uninitialized
   'X', -- Forcing Unknown
   '0', -- Forcing 0
   '1', -- Forcing 1
   'Z', -- High impedance
   'W', -- Weak unknown)
‘L’, -- Weak 0
‘H’, -- Weak 1
‘-’ -- Don’t Care

); type STD_ULOGIC_VECTOR is ARRAY (NATURAL range <> ) of STD_ULOGIC;

function RESOLVED ( S :
STD_ULOGIC_VECTOR ) return
STD_ULOGIC;

subtype STD_LOGIC is RESOLVED
STD_ULOGIC;

type STD_LOGIC_VECTOR is array
(NATURAL range <> ) of STD_LOGIC;