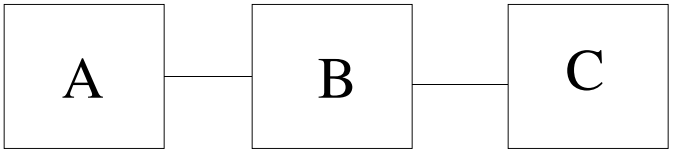
Pollard's Introduction to VHDL (Session 1)

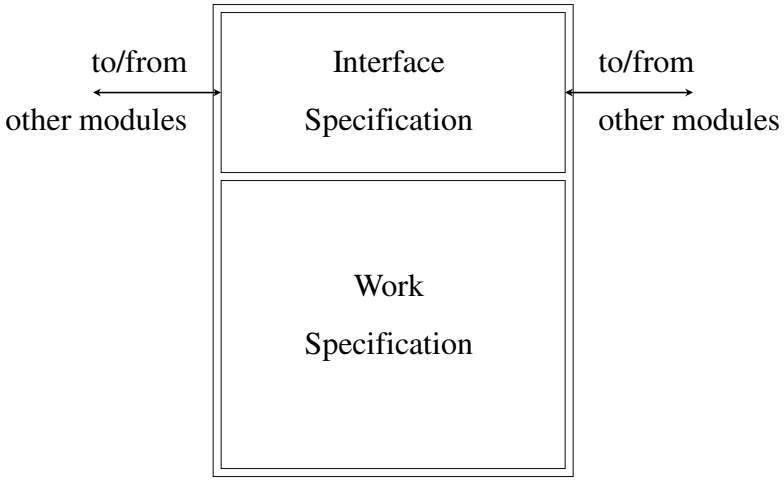
Abstraction Hierarchy Concept

- 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

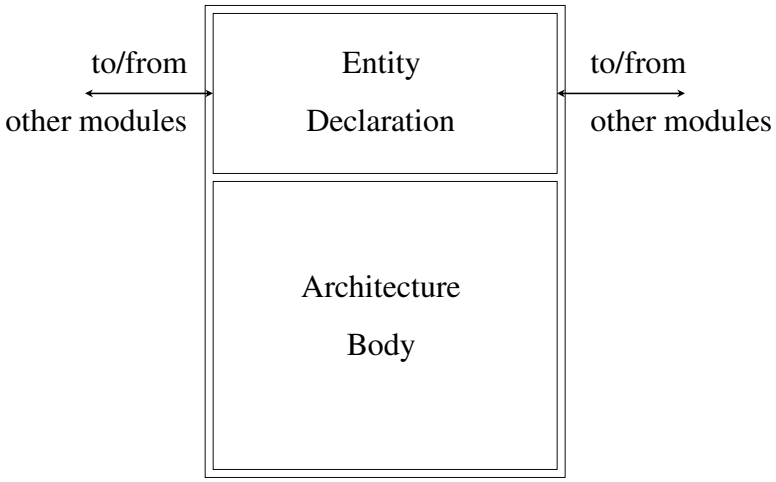
Digital Systems Diagrams



VHDL Design Unit



VHDL Design Unit



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 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

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: 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:

```
TARGET <= waveform1 when COND1 else
          waveform2 when COND2 else
          waveform3 when COND3 else
                …                  else
          waveform_n;
```

Selected Signal Assignment

- Concurrent statement
- Identify options and waveforms
- Syntax:

```
with EXPRESSION select
TARGET <= waveform1 when CHOICE1,
          waveform2 when CHOICE2,
          waveform3 when C3 | C4,
          waveformk when Ck;
```

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

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

Loop Statement Syntax

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 *BOOL_EXP* } ;

get out of loop:

exit { when *BOOL_EXP* } ;

Wait Statement Syntax

wait on *SENSITIVITY_LIST*
until *BOOLEAN_EXP*
for *TIME_EXP* ;

wait on A, B, C for 60 ns;
wait until DATA = FE for 10 ns;

Advanced Features of VHDL: Overloading

- Change meaning of literals, operators, functions, and procedures
- Overloading literal: BIT vs STD_LOGIC….
- Overloading of operators: and for BIT, STD_LOGIC
- Correct “and” operator chosen by parameter profile
- function “and” (L, R : STD_LOGIC……)

Advanced Features of VHDL: Overloading

- From Armstrong & Gray:

  “overloading gives one the ability to alter the meaning of a name without having to change the VHDL code in which it is invoked.”

Advanced Features of VHDL: Overloading

- Subprogram names: correct procedure selected by number, type of parameters:

```
DFF ( CLK, D, Q );
DFF ( CLK, D, Q, QBAR );
DFF ( CLK, D, CLR, SET, Q );
```

Advanced Features of VHDL: Visibility

- REGION: logically continuous bounded portion of text
- DECLARATION REGION: region in which name can be used to unambiguously refer to a declared item
- Item declared in declaration region, name visible to end of unit
- Names made visible outside: library, use clauses (visibility by selection)

Advanced Features of VHDL: Visibility

- General declaration regions
  entity declaration, associated architecture
  process
  block
  subprogram
  package

Advanced Features of VHDL: Visibility

- Specialized declaration regions
  record types - names of fields
  loops - index control variable
  component declarations - name, port, generic
  configuration - have declaration region for
    attribute specifications, use clauses

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;
  ```
  ```

Advanced Features of VHDL: driving and driving_value

- Port directions: in, out, inout
- Output cannot be read directly
- Output port can be read with **`PORT_NAME'driving_value`**
- Connection tested with **`PORT_NAME'driving`**

Advanced Features of VHDL: unaffected

- Method added in 93 to preserve waveform
- Signal assignment with ability to maintain old transactions, not introduce new ones
- **`A <= '1' after 10 ns when CON = '1' else unaffected;`**

Advanced Features of VHDL: Shared Variables

- VHDL 87 - no shared variables
- VHDL 93 - shared variables, but no synchronization primitives
- Allows communication between elements without the use of signals
- User responsible for semaphore activity

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:
  `use LIB_NAME.PKG_NAME.ELE_NAME;`
- STANDARD, TEXT_IO, STD_LOGIC

Package Declaration

- Used to declare elements for other associations
- Syntax:
  package PKG_IDENT is
  PKG_DECLARATIVE_ITEM
- end {package}{PKG_IDENT};

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_BODY_DECL_ITEM *
  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

Rudiments of MVL4 System

```
type MVL4 is ('X', '0', '1', 'Z');
type MVL4_VECTOR is array
  (NATURAL range <>) of MVL4;
type MVL4_TAB1D is array (MVL4) of MVL4;
type MVL4_TABLE is array (MVL4, MVL4)
  of  MVL4;
```

Rudiments of MVL4 System, Mod.

Now - build resolution function RESFUN

Resolution function made to resolve conflicts on signals

subtype MVL4RES is RESFUN MVL4;

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;
```

More VHDL Syntax: Aggregate

- Aggregate - collection of expressions that represent a composite value (array, record)
- Syntax: (comma separated list of vals)
- With => operator, can have out-of-order vals
- With discrete range, can specify multiple elements
- With others can give value to all remaining elements

More VHDL Syntax: Expression

- Expression: formula for computing value
- **`LEFT operator RIGHT`**
- Expression can be logical, relational, shift, mathematical

Logical Expression

- Left operand: BIT or BOOLEAN or 1D array of BIT or BOOLEAN
- Right operand: Same as left (also, same length)
- Result: Same as left
- Operators: **`and, or, nand, nor, xor, xnor`**
- All logical operators have same precedence

Relational Expression

- For **`=`** and **`/=`**, left operand can be any type, with right operand of same type, result is BOOLEAN
- Other relational operators: **`<, <=, >, >=`**
- Left operand type: scalar type or discrete array
- Right operant type: same as left
- Result type: BOOLEAN

Shift Expression

- Left operand: 1D array of BIT or BOOLEAN
- Right operand: integer
- Result operand: same as left operand
- Available operators: **`SLL, SRL, SLA, SRA, ROL, ROR`**
- Not available in VHDL-87

Simple Expression: Addition

- Left operand type: numeric
- Right operand type: same type as left
- Result type: same type as left
- Operators: **`+, –`**

Simple Expression: Concatenation

- Left operand type: array or element type
- Right operand type: array or element type
- Result type: array type
- Operator: `&`

Simple Expression: Sign Operators

- Right operand type: numeric type
- Result type: same type as left
- Operators: `+, –`

Simple Expression: Multiplication

- Left operand type: numeric
- Right operand type: same type as left
- Result type: same type as left
- Operators: **`*, /`**

Simple Expression: Modulo, Remainder

- Left operand type: integer
- Right operand type: same type as left
- Result type: same type as left
- Operators: **`mod, rem`**

Simple Expression: Exponentiation

- Left operand type: numeric
- Right operand type: integer
- Result type: same type as left
- Operator: **`**`**

Simple Expression: Miscellaneous

- Right operand type: numeric
- Result type: same type as right
- Operator: **`abs`**
- Right operand type: BIT/BOOLEAN (or 1D array of BIT/BOOLEAN)
- Result type: same as right
- Operator: **`not`**

Assertion Statement

- Syntax:
  assert BOOL_EXP -- condition to check
  report STRING_EXP -- message for operator
  severity SEVER_LEVEL ;
- BOOL_EXP checked; if FALSE, then STRING_EXP printed to operator, SEVER_LEVEL reported to simulator
- If BOOL_EXP is TRUE, no action

Report Statement

- Syntax:
  report STRING_EXP -- message for oper
  severity SEVER_EXP; -- for simulator
- Behaves like assertion statement with an assertion value of FALSE

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;

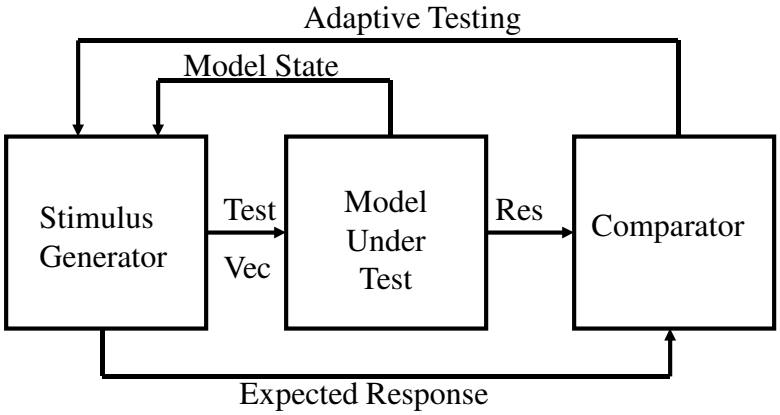
Block Statement

- Syntax:
  LABEL: process [(guard expression)][is]
  block_header
  block_declarative_part
  begin
  block_statement_part
  end block [LABEL];

Block Statement (cont.)

- Used to contain declarations
- Used to allow guarded execution
- Guard expression is of type BOOLEAN
- Signal assignment statements can be guarded – include keyword ‘guarded’ to identify when the action takes place

Test Bench Development



Syntax: Alias

- Alias – alternate name for an existing named entity
- Syntax:
  alias <alias_desig> [:subtype_indication] is <name> [signature];
- Alias_designator is identifier, character literal, or operator symbol
- Signature is parameter profile

Alias Example

```
variable REAL_NUM: BIT_VECTOR ( 31 downto 0);
alias SIGN: BIT is REAL_NUM(31);
alias EXP: BIT_VECTOR ( 7 downto 0) is REAL_NUM ( 30
   downto 23 );
alias MANTISSA: BIT_VECTOR ( 22 downto 0 );
alias "+" is SYNT.BIT_AR."+"[SYNT.BIT_AR.SIGNED,
   SYNT.BIT_AR.SIGNED return
   SYNT.BIT_AR.SIGNED];
```

More Syntax: Physical Types

- Type – representation of measurements; integral multiple of primary unit
- Syntax:
  type <NAME> is range <end> to <end>
  units
  <unit list>
  end units;

Physical Type: Example

- Type TIME is range –2E9 to 2E9
  units
  fs;
  ps = 1000 fs;
  ns = 1000 ps;
  us = 1000 ns;
  ms = 1000 us;
  sec = 1000 ms;
  min = 60 sec;
  end units;

Syntax: Record Types

- Record: composite type, objects are named elements
- Syntax:
  type <NAME> is
    record
      {element_declaration}
    end record [NAME];

Record Type Example

```
type DATE is
  record
    DAY  : INTEGER range 1 to 31;
    MONTH: MONTH_NAME;
    YEAR: INTEGER range 0 to 4000;
  end record;
```

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);

Advanced Features of VHDL: Visibility

- REGION: logically continuous bounded portion of text
- DECLARATION REGION: region in which name can be used to unambiguously refer to a declared item
- Item declared in declaration region, name visible to end of unit
- Names made visible outside: library, use clauses (visibility by selection)

Advanced Features of VHDL: Visibility

- General declaration regions
  entity declaration, associated architecture
  process
  block
  subprogram
  package

Advanced Features of VHDL: Visibility

- Specialized declaration regions
  record types - names of fields
  loops - index control variable
  component declarations - name, port, generic
  configuration - have declaration region for
    attribute specifications, use clauses

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;`

Advanced Features of VHDL: driving and driving_value

- Port directions: in, out, inout
- Output cannot be read directly
- Output port can be read with **`PORT_NAME'driving_value`**
- Connection tested with **`PORT_NAME'driving`**

Advanced Features of VHDL: unaffected

- Method added in 93 to preserve waveform
- Signal assignment with ability to maintain old transactions, not introduce new ones
- `A <= '1' after 10 ns when CON = '1' else unaffected;`

Advanced Features of VHDL: Shared Variables

- VHDL 87 - no shared variables
- VHDL 93 - shared variables, but no synchronization primitives
- Allows communication between elements without the use of signals
- User responsible for semaphore activity