ECE-545
Introduction to VHDL

Prof. K. J. Hintz

Department of Electrical and Computer Engineering
George Mason University
Some of the materials used in this course come from ARPA RASSP Program and are copyright

- **Rapid Prototyping of Application Specific Signal Processors Program**
- [http://rassp.scra.org](http://rassp.scra.org)

Some materials are copyright K. J. Hintz
Lecture Goals

- Introduce VHDL Concept and Motivation for VHDL
- Introduce the VHDL Hierarchy and Alternative Architectures Model
- Start Defining VHDL Syntax
Motivation

- Digital System Complexity Continues to Increase
  - No longer able to breadboard systems
    » Number of chips
    » Number of components
    » Length of interconnects
  - Need to simulate before committing to hardware
    » Not just logic, but timing
Motivation

Different Types of Models are Required at Various Development Stages

– Logic models
– Performance models
– Timing models
– System Models
Motivation

- Non-Proprietary *Lengua Franca*
  - Need a universal language for various levels of system design
  - Replacement for schematics
  - Unambiguous, formal language
  - Partitions problem
    » Design
    » Simulation and Verification
    » Implementation
Motivation

- Standard for Development of Upgrades
  - Testbenches and results
  - System modifications must still pass original testbench
  - Testbench can (and should) be written by people other than designers
Need for VHDL

- Leads to Automatic Implementation--Synthesis
  - Routing tools
  - Standard cell libraries
  - FPGA
  - CPLD
  - Formal Language description is independent of physical implementation
Need for VHDL

- Need a Unified Development Environment
  - Errors occur at translations from one stage of design to another
  - VHDL language the same at all levels
  - All people involved speak the same HDL
  - Testing and verification

- Performance, Reliability, and Behavioral Modeling Available at All Design Levels
Need for VHDL

- Need to Have Power and Flexibility to Model Digital Systems at Many Different Levels of Description
  - Support “mixed” simulation at different levels of abstraction, representation, and interpretation with an ability for step-wise refinement
  - Can model to high or low levels of detail, but still simulate
Languages Other Than VHDL

- **VHDL**: VHSIC (Very High Speed Integrated Circuit) **Hardware Description Language**
  - Not the only hardware description language

- Most others are proprietary
ABEL

- Simplified HDL
- PLD language
- Dataflow primitives, *e.g.*, registers
- Can use to Program XILINX FPGA
ALTERA

- Created by Altera Corporation
- Simplified dialect of HDL
  » AHDL
AHPL

- AHPL: A Hardware Programming Language
  - Dataflow language
  - Implicit clock
  - Does not support asynchronous circuits
  - Fixed data types
  - Non-hierarchical
CDL

- **CDL**: Computer Design Language
  - Academic language for teaching digital systems
  - Dataflow language
  - Non-hierarchical
  - Contains conditional statements
CONLAN: CONsensus LANguage

- Family of languages for describing various levels of abstraction
- Concurrent
- Hierarchical
- IDL: Interactive Design Language
  - Internal IBM language
  - Originally for automatic generation of PLA structures
  - Generalized to cover other circuits
  - Concurrent
  - Hierarchical
ISPS

- ISPS: Instruction Set Processor Specification
  - Behavioral language
  - Used to design software based on specific hardware
  - Statement level timing control, but no gate level control
TEGAS

- TEGAS: TEst Generation And Simulation
  - Structural with behavioral extensions
  - Hierarchical
  - Allows detailed timing specifications
TI-HDL

- TI-HDL: Texas Instruments Hardware Description Language
  - Created at Texas Instruments
  - Hierarchical
  - Models synchronous and asynchronous circuits
  - Non-extendable fixed data types
Verilog

- Essentially identical in function to VHDL
- Simpler and syntactically different
- Gateway Design Automation Co., 1983
- Early *de facto* standard for ASIC programming
- Open Verilog International standard
- Programming language interface to allow connection to non-Verilog code
ZEUS

- Created at General Electric
- Hierarchical
- Functional Descriptions
- Structural Descriptions
- Clock timing, but no gate delays
- No asynchronous circuits
Different Representation Models

- Some, Not Mutually Exclusive, Models
  - Functional
  - Behavioral
  - Dataflow
  - Structural
  - Physical

- From RASSP Taxonomy
Functional Model

- Describes the logical Function of Hardware Independent of Any Specific Implementation or Timing Information
  - Can exist at multiple levels of abstraction, depending on the granularity and the data types that are used in the behavioral description
Behavioral Model

- Describes the Function and Timing of Hardware Independent of Any Specific Implementation
  - Can exist at multiple levels of abstraction, depending on the granularity of the timing that are used in the functional description
Functional & Behavioral Descriptions

- Functional & Behavioral Models May Bear Little Resemblance to System Implementation
  - Structure not necessarily implied
Dataflow Model

- Describes How Data Moves Through the System and the Various Processing Steps
  - Register Transfer Level (RTL)
  - No registers are native to VHDL
  - Hides details of underlying combinational circuitry and functional implementation
Structural Model

- Represents a System in Terms of the Interconnections of a Set of Components
  - Components are interconnected in a hierarchical manner
  - Components themselves are described structurally, behaviorally, or functionally with interfaces between structural and their behavioral-level implementations
Structural Descriptions

- Pre-Defined VHDL Components Are ‘Instantiated’ and Connected Together

- Structural Descriptions May Connect Simple Gates or Complex, Abstract Components
Structural Descriptions

- Mechanisms for Supporting Hierarchical Description

- Mechanisms for Describing Highly Repetitive Structures Easily
Physical Model

- Specifies the Relationship Between the Component Model and the Physical Packaging of the Component.
  - Contains all the timing and performance details to allow for an accurate simulation of physical reality
  - Back annotation allows precise simulations
RASSP Roadmap

RASSP DESIGN LIBRARIES AND DATABASE

Primarily software

VIRTUAL PROTOTYPE

Primarily hardware

SYSTEM DEF.

FUNCTION DESIGN

HW & SW PART.

HW DESIGN

HW FAB

INTEG. & TEST

HW & SW CODESIGN

SW DESIGN

SW CODE

VHDL
Outline

- VHDL Background/History
- VHDL Design Example
- VHDL Model Components
  - Entity Declarations
  - Architecture Descriptions
- Basic Syntax and Lexigraphical Conventions
Reasons for Using VHDL

  
  - VHDL is an acronym for Very High Speed Integrated Circuit Hardware Description Language
Reasons for Using VHDL

- VHDL enables hardware modeling from the gate to system level
- VHDL provides a mechanism for digital design and reusable design documentation
- VHDL Provides a Common Communications Medium
A Brief History of VHDL

- Very High Speed Integrated Circuit (VHSIC) Program
  - Launched in 1980
  - Object was to achieve significant gains in VLSI technology by shortening the time from concept to implementation (18 months to 6 months)
  - Need for common descriptive language
A Brief History of VHDL

- Woods Hole Workshop
  - Held in June 1981 in Massachusetts
  - Discussion of VHSIC goals
  - Comprised of members of industry, government, and academia
A Brief History of VHDL

- July 1983: contract awarded to develop VHDL
  - Intermetrics
  - IBM
  - Texas Instruments
- August 1985: VHDL Version 7.2 released
A Brief History of VHDL

- September 1993: VHDL was restandardized to clarify and enhance the language
- VHDL has been accepted as a Draft International Standard by the IEC
Gajski and Kuhn’s Y Chart

Behavioral

Algorithmic

Functional Block

Processor

Hardware Modules

ALUs, Registers

Gates, FFs

Transistors

Logic

Circuit

Rectangles

Cell, Module Plans

Floor Plans

Clusters

Physical Partitions

Physical/Geometry
VHDL Model

Generic

Behavioral Architecture

Functional Architecture

Dataflow Architecture

Structural Architecture

Entity

Package

Ports
VHDL Design Example

- Problem: Design a single bit half adder with carry and enable

- Specifications
  - Inputs and outputs are each one bit
  - When enable is high, result gets \( x \) plus \( y \)
  - When enable is high, carry gets any carry of \( x \) plus \( y \)
  - Outputs are zero when enable input is low

\[ x \rightarrow \text{Half Adder} \rightarrow \text{carry} \]

\[ y \rightarrow \text{Half Adder} \rightarrow \text{result} \]

\[ \text{enable} \rightarrow \text{Half Adder} \]
As a first step, the entity declaration describes the interface of the component – input and output *ports* are declared.

```vhdl
ENTITY half_adder IS
    PORT( x, y, enable: IN BIT;
         carry, result: OUT BIT);
END half_adder;
```
A high level description can be used to describe the function of the adder

ARCHITECTURE half_adder_a OF half_adder IS
BEGIN
  PROCESS (x, y, enable)
  BEGIN
    IF enable = '1' THEN
      result <= x XOR y;
      carry <= x AND y;
    ELSE
      carry <= '0';
      result <= '0';
    END IF;
  END PROCESS;
END half_adder_a;

The model can then be simulated to verify correct functionality of the component
A high level description can be used to describe the function of the adder

ARCHITECTURE half_adder_b OF half_adder IS
BEGIN
    PROCESS (x, y, enable)
    BEGIN
        IF enable = '1' THEN
            result <= x XOR y after 10ns;
            carry  <= x AND y  after 12 ns;
        ELSE
            carry  <= '0'  after 10ns;
            result <= '0' after 12ns;
        END IF;
    END PROCESS;
END half_adder_b;

The model can then be simulated to verify correct timing of the entity
A Third Method Is to Use Logic Equations to Develop a Data Flow Description

ARCHITECTURE half_adder_c OF half_adder IS
BEGIN
  carry <= enable AND (x AND y);
  result <= enable AND (x XOR y);
END half_adder_c;

● Again, the model can be simulated at this level to confirm the logic equations
VHDL Design Example
Structural Specification

- As a Fourth Method, a Structural Description Can Be Created From Previously Described Components
- These gates can be taken from a library of parts
ARCHITECTURE half_adder_d OF half_adder IS

COMPONENT and2
  PORT (in0, in1 : IN BIT;
        out0 : OUT BIT);
END COMPONENT;

COMPONENT and3
  PORT (in0, in1, in2 : IN BIT;
        out0 : OUT BIT);
END COMPONENT;

COMPONENT xor2
  PORT (in0, in1 : IN BIT;
        out0 : OUT BIT);
END COMPONENT;

FOR ALL : and2 USE ENTITY gate_lib.and2_Nty(and2_a);
FOR ALL : and3 USE ENTITY gate_lib.and3_Nty(and3_a);
FOR ALL : xor2 USE ENTITY gate_lib.xor2_Nty(xor2_a);

-- description is continued on next slide
-- continuing half_adder_d description

    SIGNAL xor_res : BIT; -- internal signal
    -- Note that other signals are already declared in entity

    BEGIN

        A0 : and2 PORT MAP (enable, xor_res, result);
        A1 : and3 PORT MAP (x, y, enable, carry);
        X0 : xor2 PORT MAP (x, y, xor_res);

    END half_adder_d;
VHDL Model Components

- A Complete VHDL Component Description Requires a VHDL Entity and a VHDL Architecture
  - The entity defines a component’s interface
  - The architecture defines a component’s function

- Several Alternative Architectures May Be Developed for Use With the Same Entity
VHDL Model Components

- Three Areas of Description for a VHDL Component:
  - Structural descriptions
  - Functional descriptions
  - Timing and delay descriptions (Behavioral)
Process

- Fundamental Unit for Component Behavior Description Is the *Process*
- Processes may be explicitly or implicitly defined and are packaged in architectures
VHDL Model Components

- Primary Communication Mechanism Is the *Signal* (distinct from a *variable*)
  - Process executions result in new values being assigned to signals which are then accessible to other processes
  - Similarly, a signal may be accessed by a process in another architecture by connecting the signal to ports in the the entities associated with the two architectures

```
Output <= My_id + 10;
```
Entity Declarations

- The Primary Purpose of the Entity Is to Declare the Signals in the Component’s Interface
  - The interface signals are listed in the PORT clause
    » In this respect, the *entity* is akin to the schematic *symbol* for the component
ENTITY half_adder IS
  GENERIC(prop_delay : TIME := 10 ns);
  PORT( x, y, enable : IN BIT;
       carry, result : OUT BIT);
END half_adder;
Entity Declarations

Port Clause

PORT clause declares the interface signals of the object to the outside world

```
PORT (signal_name : mode data_type);
```

Three parts of the PORT clause

- Name
- Mode
- Data type

```
PORT ( input : IN BIT VECTOR(3 DOWNTO 0);
       ready, output : OUT BIT );
```
Entity Declarations

Port Clause (Cont.)

- The Port Mode of the Interface Describes the Direction in Which Data Travels With Respect to the Component

- Five Port Modes
  1. **In**: data comes in this port and can only be read

  2. **Out**: data travels out this port
3. **Buffer**: bidirectional data, but only one signal driver may be enabled at any one time.

4. **Inout**: bidirectional data with any number of active drivers allowed but requires a Bus Resolution Function.

5. **Linkage**: direction of data is unknown.
Entity Declarations

Generics May Be Used for Readability, Maintenance and Configuration

Generic Clause Syntax:

```
GENERIC (generic_name : type [:= default_value]);
```

- If optional `default_value` missing in generic clause declaration, it must be present when component is to be used (i.e. instantiated)
Behavioral Descriptions

- VHDL Provides Two Styles of Describing Component Behavior
  - **Data Flow**: concurrent signal assignment statements
  - **Behavioral**: *processes* used to describe complex behavior by means of high-level language constructs
    » variables, loops, if-then-else statements, *etc.*
Generic Clause

- **Generic Clause Example**:

  ```
  GENERIC (My_ID : INTEGER := 37);
  ```

- The generic `My_ID`, with a default value of 37, can be referenced by any architecture of the entity with this generic clause.

- The default can be overridden at component instantiation.
Architecture Bodies

- Describes the Operation of the Component, Not Just Its Interface

- More Than One Architecture Can (and Usually Is) Associated With Each Entity
Architecture Bodies

- Consist of Two Parts:
  1. **Declarative part** -- includes necessary declarations, *e.g.*:
     - type declarations
     - signal declarations
     - component declarations
     - subprogram declarations
Architecture Bodies

2. **Statement part** -- includes statements that describe organization and/or functional operation of component, *e.g.*:
   - concurrent signal assignment statements
   - process statements
   - component instantiation statements
ARCHITECTURE half_adder_d OF half_adder IS

-- architecture declarative part

SIGNAL xor_res : BIT ;

-- architecture statement part

BEGIN

carry    <= enable AND (x AND y) ;
result   <= enable AND xor_res ;
xor_res  <= x XOR y ;
END half_adder_d ;
Lexical Elements of VHDL

- Comments
  - two dashes to end of line is a comment, e.g.,

  --this is a comment
Lexical Elements of VHDL

- Basic Identifiers
  - Can Only Use
    - alphabetic letters (A-Z, a-z), or
    - Decimal digits (0-9), or
    - Underline character (_)
  - Must Start With Alphabetic Letter (MyVal)
Lexical Elements of VHDL

- Basic Identifiers
  - Not case sensitive
    (LastValue = = lAsTvALue)
  - May NOT end with underline (MyVal_ )
  - May NOT contain sequential underlines
    (My__Val)
Lexical Elements of VHDL

■ Extended Identifiers
  – Any character(s) enclosed by \ \ \ \ 
  – Case IS significant
  – Extended identifiers are distinct from basic identifiers
  – If “ \ ” is needed in extended identifier, use “ \ \ “
Lexical Elements of VHDL

- **Reserved Words**
  - Do not use as identifiers

- **Special Symbols**
  - Single characters
    & ` ( ) * + , - . / : ; < = > |
  - Double characters (no intervening space)
    => ** := /= >= <= <>
Lexical Elements of VHDL

- Numbers
  - Underlines are NOT significant
    \[ 10\#8_192 \]
  - Exponential notation allowed
    \[ 46e5, 98.6E+12 \]
  - Integer Literals \( 12 \)
    » Only positive numbers; negative numbers are preceded by unary negation operator
    » No radix point
Lexical Elements of VHDL

- Real Literals (23.1)
  - Always include decimal point
  - Radix point must be preceded and followed by at least one digit.

- Radix (radix # number expressed in radix)
  - Any radix from binary (2) to hexadecimal (16)
  - Numbers in radices > 10 use letters a-f for 10-15.
Lexical Elements of VHDL

- **String**
  - A sequence of any printable characters enclosed in double quotes
    
    ( "a string" )
  - Quote uses double quote
    
    ( " he said ""no!"" "" )
  - Strings longer than one line use the concatenation operator ( & ) at beginning of continuation line.
Lexical Elements of VHDL

- **Characters**
  - Any printable character including space enclosed in single quotes (`'x'`)

- **Bit Strings**
  - B for binary (b"0100_1001"
  - O for Octal (o"76443"
  - X for hexadecimonal (x"FFFE_F138"

KJH, 545_1, 4/20/01
VHDL Syntax

- Extended Backus-Naur Form (EBNF)
  - Language divided into syntactic categories
  - Each category has a rule describing how to build a rule of that category
  - Syntactic category $<=$ pattern
  - “$<=$“ is read as “...is defined to be...”
VHDL Syntax

- *e.g.*,

```
variable_assignment <= target := expression;
```

- A clause of the category `variable_assignment` is defined to be a clause from the category `target` followed by the symbol `:=` followed by a clause from the `expression` category followed by a terminating `;`
VHDL Syntax

- syntax between outline brackets [ ] is optional

- syntax between outline braces { } can be repeated none or more times, a.k.a. “Kleene Star”
A preceding lexical element can be repeated an arbitrary number of times if ellipses are present, e.g.,

```
case-statement  <=
  case  expression  is
    case_statement_alternative
    { . . . }
  end case;
```
VHDL Syntax

- If a delimiter is needed, it is included with the ellipses as

```vhdl
identifier_list <=
  identifier { , . . . }
```

VHDL Syntax

- “OR” operator, “ | ”, in a list of alternatives, e.g.,
  
  `mode <= in | out | inout`

- When grouping is ambiguous, parenthesis are used, e.g.,
  
  `term <= factor { ( * | / | mod | rem ) factor }`
**VHDL Syntax**

- e.g. an identifier may be defined in EBNF as

  \[
  \text{identifier} \leq \\
  \text{letter} \{ \ [\text{underline}] \ 	ext{letter_or_digit} \ }
  \]
VHDL Lecture 1

The end...