VHDL Introduction, Part I

Figures in this lecture are from:
*Rapid Prototyping of Digital Systems, Second Edition*

Dr. Robert A. Walker
Computer Science Department
Kent State University
Kent, OH 44242 USA
http://www.cs.kent.edu/~walker

VHDL History

- **VHDL = VHSIC Hardware Description Language**
  - VHSIC = Very High Speed Integrated Circuits
  - Both US Department of Defense (DOD) programs

- Initially developed under DOD auspices, later standardized as IEEE standards 1076-1987, 1076-1993, & 1076-1164 (standard logic data type)

- Syntax similar to ADA and Pascal

- A concurrent language, initially aimed at simulation, later at synthesis
  - Specific subsets and “cookbook” design styles supported by logic and behavioral synthesis tools
  - Write code like this & you’ll get this expected design…
Signals, Time, and Simulation

- Variables vs. signals
  - VHDL variables change value without delay
  - VHDL signals have an associated delay

- A signal is given a value at a specific point in time, and retains that value until it is given a new value
  - A waveform is a sequence of values over time
  - Example: in1 <= ‘0’, ‘1’ after 5 ns, ‘0’ after 15 ns;
  - A variable has a single value, whereas a signal has multiple value / time pairs

- A discrete event simulator executes VHDL code by advancing time to the next event, updating signal values, then possibly scheduling new events

Simple Gate Network (Desired Hardware)
Simple Gate Network (VHDL Code)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY gate_network IS
PORT(A, B, C : IN STD_LOGIC;
      D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      X, Y : OUT STD_LOGIC);
END gate_network;

ARCHITECTURE behavior OF gate_network IS
BEGIN
  -- Concurrent assignment statements operate in parallel
  X <= A AND NOT( B OR C ) AND ( D(1) XOR D(2) );
  -- Process must declare a sensitivity list,
  -- In this case it is (A, B, C, D)
  -- List includes all signals that can change the outputs
  PROCESS ( A, B, C, D ) BEGIN
    -- Statements inside process execute sequentially
    Y <= A AND NOT( B OR C ) AND ( D(1) XOR D(2) );
  END PROCESS;
END behavior;

VHDL Design Styles

- VHDL was designed to support both behavioral and structural designs as various levels of abstraction
  - *Behavioral description* says what the design does, but not how it is implemented
  - *Structural description* specifies the interconnection of a set of components, but not what the components do
  - The design process tends to convert high-level behavioral descriptions to low-level structural ones

- VHDL design entities include
  - *Entity* declaration — interface (named I/O ports)
  - *Architecture* declaration — behavioral or structural description of the design entity
Simple Gate Network (Entity)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY gate_network IS
PORT(A, B, C : IN STD_LOGIC;
     D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     X, Y : OUT STD_LOGIC);
END gate_network;

ARCHITECTURE behavior OF gate_network IS
BEGIN
   X <= A AND NOT(B OR C) AND (D(1) XOR D(2));
   Y <= A AND NOT(B OR C) AND (D(1) XOR D(2));
END behavior;

Simple Gate Network (Architecture)

LIBRARY IEEE:
USE IEEE.STD_LOGIC_1164.ALL:

ENTITY gate_network IS
PORT(A, B, C : IN STD_LOGIC;
     D : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     X, Y : OUT STD_LOGIC);
END gate_network;

ARCHITECTURE behavior OF gate_network IS
BEGIN
   X <= A AND NOT(B OR C) AND (D(1) XOR D(2));
   Y <= A AND NOT(B OR C) AND (D(1) XOR D(2));
END behavior;
**VHDL Data Types**

- **Boolean, integer, real**

- **STD_LOGIC to model logic values**
  - 0, 1, Z, U, X, –, L, W, H
  - Z = tri-state, U = uninitialized, X = unknown, – = don’t care, L = weak “0”, W = weak unknown, H = weak “1”

  Table 6.2 STD_LOGIC conversion functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_STDLOGICVECTOR( bit_vector )</td>
<td>TO_STDLOGICVECTOR( X&quot;FFFF&quot; )</td>
</tr>
<tr>
<td>Converts a bit vector to a standard logic vector.</td>
<td>Generates a 16-bit standard logic vector of ones. &quot;X&quot; indicates hexadecimal and &quot;B&quot; is binary.</td>
</tr>
<tr>
<td>CONV_STD_LOGIC_VECTOR( integer, bits )</td>
<td>CONV_STD_LOGIC_VECTOR( 7, 4 )</td>
</tr>
<tr>
<td>Converts an integer to a standard logic vector.</td>
<td>Produces a standard logic vector of &quot;0111&quot;.</td>
</tr>
<tr>
<td>CONV_INTEGER( std_logic_vector )</td>
<td>CONV_INTEGER( &quot;0111&quot; )</td>
</tr>
<tr>
<td>Converts a standard logic vector to an integer.</td>
<td>Produces an integer value of 7.</td>
</tr>
</tbody>
</table>

**VHDL Operations**

**precedence:**

**,** ABS, NOT
*,/,% MOD, REM
+,− (sign)
+,−,\&
SLL, SRL, SLA, ...
=, /=, <, <=, >, >=
AND, NOT, OR, NOT, XOR, XNOR

*note: all at same level*

### Table 6.1 VHDL Operators

<table>
<thead>
<tr>
<th>VHDL Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication*</td>
</tr>
<tr>
<td>/</td>
<td>Division*</td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus*</td>
</tr>
<tr>
<td>REM</td>
<td>Remainder</td>
</tr>
<tr>
<td>&amp;</td>
<td>Concatenation (used to combine bits)</td>
</tr>
<tr>
<td>SLL**</td>
<td>Logical shift left</td>
</tr>
<tr>
<td>SRL**</td>
<td>Logical shift right</td>
</tr>
<tr>
<td>SLA**</td>
<td>Arithmetic shift left</td>
</tr>
<tr>
<td>SRA**</td>
<td>Arithmetic shift right</td>
</tr>
<tr>
<td>ROL**</td>
<td>Rotate left</td>
</tr>
<tr>
<td>ROR**</td>
<td>Rotate right</td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
<tr>
<td>=&lt;</td>
<td>Inequality</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR</td>
</tr>
<tr>
<td>NAND</td>
<td>Logical NAND</td>
</tr>
<tr>
<td>NOR</td>
<td>Logical NOR</td>
</tr>
<tr>
<td>XNOR</td>
<td>Logical XNOR</td>
</tr>
<tr>
<td>XNOR**</td>
<td>Logical XNOR</td>
</tr>
</tbody>
</table>

*Note:
VHDL is not case sensitive

*Supported in many VHDL synthesis tools. In the MAX+PLUS II tools:
only multiply and divide by powers of two (shifts) are supported. Mod and Rem are not supported in MAX+PLUS II. Efficient design of multiply or divide hardware typically requires the user to specify the arithmetic algorithm and design in VHDL.

**Supported only in 1076-1993 VHDL.
Concurrent and Sequential Statements

- Concurrent statements
  - Signal assignment
  - Conditional signal assignment (WHEN-ELSE)
  - Selected signal assignment (WITH-SELECT-WHEN)
  - Process

- Statements inside a process are executed sequentially
  - Variables, arrays, queues
  - Variable assignments (no delay)
  - IF-THEN-ELSE, CASE-WHEN, LOOP
  - WAIT UNTIL, WAIT FOR, WAIT ON
  - WARNING — not everything you do in a process may be synthesizable by your synthesis tools!

Four Multiplexers

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY multiplexer IS
  PORT( A, B, Mux_Control : IN
        Mux_Out1, Mux_Out2, Mux_Out3, Mux_Out4 : OUT
        STD_LOGIC
        );
END multiplexer;

ARCHITECTURE behavior OF multiplexer IS
BEGIN

  Mux_Out1 <= A WHEN Mux_Control = '0' ELSE B;
  -- / with Select Statement
  Mux_Out2 <=
    A WHEN '0',
    B WHEN '1',
    A WHEN OTHERS;
  -- OTHERS case required since STD_LOGIC has values other than '0' or '1'
  Mux_Out3 <= A;
  -- Statements inside a process execute sequentially.
  Mux_out3 <= B;
  CASE Mux_Control IS
    WHEN '0' =>
      Mux_Out4 <= A;
    WHEN '1' =>
      Mux_Out4 <= B;
    WHEN OTHERS =>
      Mux_Out4 <= A;
  END CASE;
END behavior;
```

Spring 2006, Lecture 18
Signal Assignment

- **Concurrent signal assignment**
  - Example: \( a \leq \text{‘}1\text{’}; \quad z \leq a \text{ XOR } b; \)
  - LHS signal type must match RHS

- **Conditional signal assignment**
  - Example: \( z \leq s0 \text{ WHEN } \text{sel=}’0\text{’ } \text{ELSE } s1; \)
  - Last ELSE should have no condition to handle all cases not otherwise covered

- **Selected signal assignment**
  - Example: \( \text{WITH sel SELECT } \)
    \( \quad z \leq s0 \text{ WHEN } ‘0’, s1 \text{ WHEN } ‘1\text{’}; \)
  - Cover all cases in mutually-exclusive fashion, possibly use “WHEN OTHERS” for last case

Processes

- **A process may begin “main: process (A, B)”**
  - Name of process is “main”
  - Sensitivity list for process is “A, B”

- **Sensitivity list**
  - If one of these signals changes, the process executes
  - Should contain any signals on the right-hand-side of an assignment, or in any boolean condition

- **Conditionals in IF statements must return a boolean**
  - OK to write: \( \text{if reset } = ‘1\text{’ then } \cdots \)
  - NOT OK to write: \( \text{if reset then } \cdots \)
  - Returns a standard logic type (!)
Sequential Statements (Inside Process)

- **IF-THEN-ELSE**
  - Only statements in the first condition matched will be executed
  - Nesting allowed, each level adds more multiplexing or other additional logic, so should be done carefully

- **CASE-WHEN**
  - Good when all branching is based on single condition

- **LOOP, WHILE-LOOP, FOR-LOOP**
  - Repetition

- **WAIT UNTIL** (boolean), **WAIT FOR** (time expression), **WAIT ON** (signal) -- waits for event on that signal