VHDL Introduction, Part I

Figures in this lecture are from:

Rapid Prototyping of Digital Systems, Second Edition
James O. Hamblen & Michael D. Furman,
Kluwer Academic Publishers, 2001,
ISBN 0-7923-7439-8

Dr. Robert A. Walker

Computer Science Department
Kent State University
Kent, OH 44242 USA

http://www.cs.kent.edu/~walker

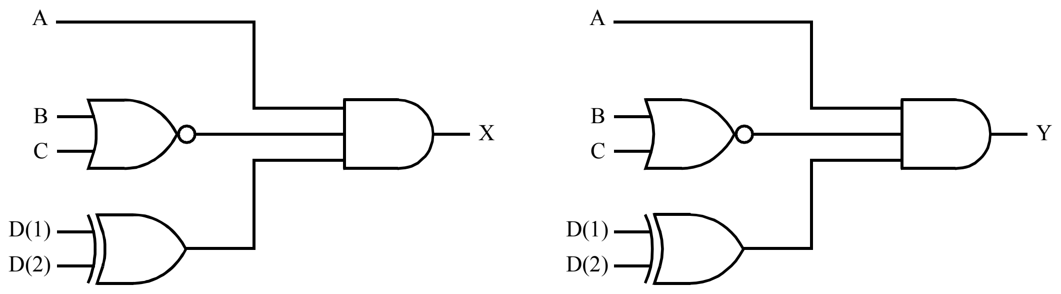
VHDL History

- **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;                              -- Include Libraries for standard logic data types
USE  IEEE.STD_LOGIC_1164.ALL;
                                           -- Entity name normally the same as file name
ENTITY gate_network IS                     -- Ports: Declares module inputs and outputs
    PORT(A, B, C      : IN       STD_LOGIC;
                                           -- Standard Logic Vector ( Array of 4 Bits )
         D            : IN       STD_LOGIC_VECTOR( 3 DOWNTO 0 );
                                           -- Output Signals
         X, Y         : OUT      STD_LOGIC );

END gate_network;

                                 -- Defines internal module architecture
ARCHITECTURE behavior OF gate_network IS
BEGIN                                      -- Concurrent assignment statements operate in parallel
                                 -- D(1) selects bit 1 of standard logic vector D
    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;
```

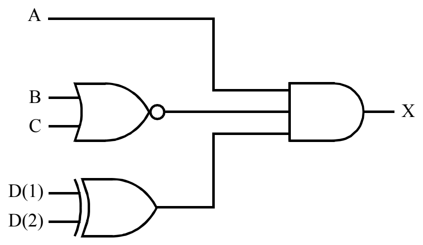
VHDL Design Styles

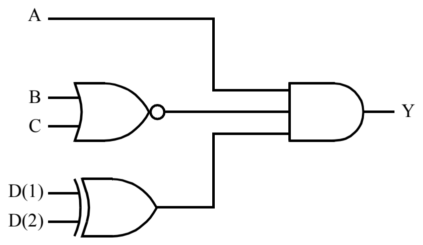
- **VHDL was designed to support both behavioral and structural designs as various levels of abstraction**
  - *Behavioral description* says what the design does, but not how it is implemented
  - *Structural description* specifies the interconnection of a set of components, but not what the components 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

Robert A. Walker

Simple Gate Network (Entity)

```
LIBRARY IEEE;                               -- Include Libraries for standard logic data types
USE  IEEE.STD_LOGIC_1164.ALL;
                                            -- Entity name normally the same as file name
ENTITY gate_network IS                      -- Ports: Declares module inputs and outputs
   PORT(A, B, C      : IN      STD_LOGIC;
                                            -- Standard Logic Vector ( Array of 4 Bits )
        D            : IN      STD_LOGIC_VECTOR( 3 DOWNTO 0 );
                                            -- Output Signals
        X, Y         : OUT     STD_LOGIC );

END gate_network;
```

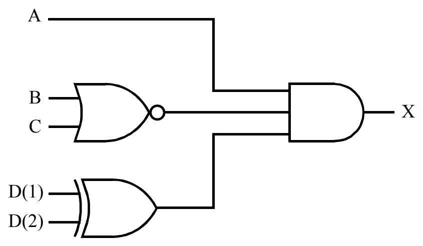


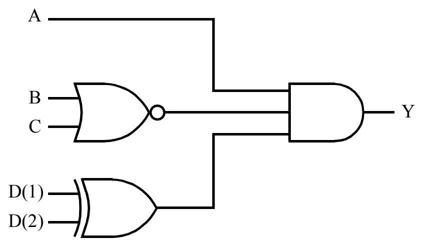


Robert A. Walker

Simple Gate Network (Architecture)

```
                                  -- Defines internal module architecture
ARCHITECTURE behavior OF gate_network IS
BEGIN                                       -- Concurrent assignment statements operate in parallel
                                  -- D(1) selects bit 1 of standard logic vector D
   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 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.

Function	Example:
TO_STDLOGICVECTOR(*bit_vector*)	TO_STDLOGICVECTOR(X"FFFF")
Converts a bit vector to a standard logic vector.	Generates a 16-bit standard logic vector of ones. "X" indicates hexadecimal and "B" is binary.
CONV_STD_LOGIC_VECTOR(*integer, bits*)	CONV_STD_LOGIC_VECTOR(7, 4)
Converts an integer to a standard logic vector.	Produces a standard logic vector of "0111".
CONV_INTEGER(*std_logic_vector*)	CONV_INTEGER("0111")
Converts a standard logic vector to an integer.	Produces an integer value of 7.

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.

VHDL Operator	Operation
+	Addition
-	Subtraction
*	Multiplication*
/	Division*
MOD	Modulus*
REM	Remainder*
&	Concatenation ξ used to combine bits
SLL**	logical shift left
SRL**	logical shift right
SLA**	arithmetic shift left
SRA**	arithmetic shift right
ROL**	rotate left
ROR**	rotate right
=	equality
/=	Inequality
<	less than
<=	less that or equal
>	greater than
>=	greater than or equal
NOT	logical NOT
AND	logical AND
OR	logical OR
NAND	logical NAND
NOR	logical NOR
XOR	logical XOR
XNOR*	logical XNOR

*Not 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.

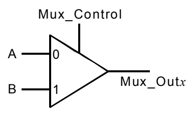
Note:
VHDL is
not case
sensitive

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 <u>sequentially</u>**
  - 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

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

ENTITY multiplexer IS                          -- Input Signals and Mux Control
        PORT(  A, B, Mux_Control       : IN    STD_LOGIC;
                Mux_Out1, Mux_Out2,
                Mux_Out3, Mux_Out4     : OUT   STD_LOGIC   );
END multiplexer;

ARCHITECTURE behavior OF multiplexer IS
BEGIN                                          -- selected signal assignment statement…

        Mux_Out1 <= A WHEN Mux_Control = '0' ELSE B;
                                               -- …  with Select Statement
        WITH mux_control SELECT

        Mux_Out2 <=    A WHEN    '0',
                       B WHEN    '1',
                       A WHEN OTHERS;          -- OTHERS case required since STD_LOGIC
                                               --      has values other than "0" or "1"
        PROCESS ( A, B, Mux_Contro l)
        BEGIN                                  -- Statements inside a process
                IF Mux_Control = '0' THEN      --      execute sequentially.
                   Mux_Out3 <= A;
                ELSE
                   Mux_out3 <= B;
                END IF;

                CASE Mux_Control IS
                        WHEN '0' =>
                                Mux_Out4 <= A;
                        WHEN '1' =>
                                Mux_Out4 <= B;
                        WHEN OTHERS =>
                                Mux_Out4 <= A;
                END CASE;
        END PROCESS;
END behavior;
```



Signal Assignment

- **Concurrent signal assignment**
  - Example: a <= ‘1’; z <= a XOR b;
  - LHS signal type must match RHS
- **Conditional signal assignment**
  - Example: z <= s0 WHEN sel=‘0’ ELSE s1;
  - Last ELSE should have no condition to handle all cases not otherwise covered
- **Selected signal assignment**
  - Example: WITH sel SELECT
    z <= s0 WHEN ‘0’, s1 WHEN ‘1’;
  - 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: if reset = ‘1’ then…
  - NOT OK to write: if reset then…
    - 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 signle condition
- **LOOP, WHILE-LOOP, FOR-LOOP**
  - Repetition
- **WAIT UNTIL (boolean), WAIT FOR (time expression), WAIT ON (signal) -- waits for event on that signal**