13 April, 2004
UNCA CSCI 311, NCSU ECE 212

VHDL

VHSIC Hardware Description Language

Very High Speed Integrated Circuits

Developed by IBM, Texas Instruments, and Intermetics in 1983-84

Under DoD contract

Language and software transfered to IEEE in 1985

Structure of VHDL

Interface specification

`ENTITY` *specification_name* `IS`

begins the interface specification

`PORT (` *name*: mode type .... `);`

mode can be `IN`, `OUT`, `INOUT`, and more

type is `BIT` or `INTEGER` or more

`END` *specification_name*

Architecture ***specifications***

`ARCHITECTURE` *impl_name* `OF` *spec_name* `IS`

VHDL implementation with a choice of style

procedural -- conventional programming

dataflow -- concurrent programming

structural -- component connection

`END` *impl* `;`

VHDL BNF

Interface specification example

```
library ieee ;
use ieee.std_logic_1164.all ;

entity VOTE3CB is
  port (
    X  : in  STD_LOGIC;
    Y  : in  STD_LOGIC;
    Z  : in  STD_LOGIC;
    M  : out STD_LOGIC
  ) ;
end VOTE3CB ;
```

VHDL types

atomic types

`bit`, `boolean`, `character`, `real`, ...

enumerated types

```
type light_color is (red, yellow, green) ;
```

ranges

```
type op_code is integer range 0 to 31 ;
```

arrays

```
type op_code_field is array (4 downto 0) of STD_LOGIC ;
```

IEEE standard logic library types

STD_LOGIC	
'U'	Unitialized
'X'	Forcing Unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High Impedance
'W'	Weak Unknown
'L'	Weak 0
'H'	Weak 1
'-'	Don't care

Ports

Connections to the outside world

`in`	
`out`	value cannot be "read" into the entity
`buffer`	output value that can be "read"
`inout`	

Architecture Interface

`architecture` *architecture-name* `of` *entity-name* `is`

type declarations

generally names for structured types, like fields of an instruction

signal declarations

outside connections, but without `mode`'s

constant declarations

useful constants, like opcode for ADD

function definitions

procedure definitions

component declarations

to connect in VHDL "schematics"

`begin`

concurrent-statements

generally VHDL assignments or components

`end` *architecture-name*

Some preliminaries

Operators: Table 4-30 (p. 272)

`mod`, `rem`, `abs`, `and`, `or`, `nand`, `nor`, …

Arrays: Table 4-33 (p. 274)

Array indexes do not need to start with 0

Boolean values can index an array

Arrays indexes can go downward

`V(5)`, not `V[5]`

Array initialization in many varieties

Types

`0`	`integer`
`'0'`	`bit`
`'0'`	`STD_ULOGIC` (forcing 0)
`false`	`boolean`

Functions and procedures

Structural description

Similar to a schematic

Lists of connected components

Nothing like a "normal" programming language

```
begin  -- VOTE3CS_arch_JDB
  -- The NANDs of the AND plane
  N0: NAND2 port map(A => X, B => Y, N => TermXY) ;
  N1: NAND2 port map(A => X, B => Z, N => TermXZ) ;
  N2: NAND2 port map(A => Y, B => Z, N => TermYZ) ;
  -- The NAND of the OR plane
  N3: NAND3 port map(A=>TermXY, B=>TermXZ, C=>TermYZ, N=>M) ;
end VOTE3CS_arch_JDB;
```

Dataflow descriptions

Concurrent signal-assignment states

Similar to functional programming languages

Works well for combinational specification

Often used with behavioral statements

```
begin
   M <= X and Y or X and Z or Y and Z ;
end
```

```
begin
  Sum <= '1' when not (In0 = In1) else '0' ;
  Carry <= '1' when In0 = '1' AND In1 = '1' else '0' ;
end HalfAdder_Arch ;
```

Behavioral description

Similar to a conventional program

Generally a process

`process` (*signalvar*)

Sequential program "called" by change of signal

`:=` assigns to a *variable*

`<=` assigns to a *signal*

```
process(Clock, Reset)
  VARIABLE CycleCount : threecount := 0 ;
  VARIABLE OneCount   : threecount := 0 ;
begin
  if Reset = '1' then
    CycleCount := 0 ;
    OneCount := 0 ;
    SerOut <= '0' ;
  elsif Clock'event and Clock = '1' then
    if CycleCount = 2 then
      if OneCount = 2 or (OneCount = 1 and SerIn = '1') then
        SerOut <= '1' ;
      else
        SerOut <= '0' ;
      end if ;
      CycleCount := 0 ;
      OneCount := 0 ;
    else
      SerOut <= '0' ;
      CycleCount := CycleCount + 1 ;
      if SerIn = '1' then
        OneCount := OneCount + 1 ;
      end if ;
    end if ;
  end if ;
end process ;
```

By use of enumerated types and bit variables, you can specify

State transitions

Flip-flop transitions