b20 February, 2002

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
- DoD requirements
  - Hierarchical design
  - Library support
  - Timing control at all levels
  - Sequential control structure
  - Strong typing -- like Ada
- VHDL 93

Structure of VHDL

- Interface specification
  - `ENTITY` specification_name `IS`
    - begins the interface specification
  - `PORT(` *name*: mode type .... `);`
    - mode can be `IN`, `OUT`, `INOUT`, and `BUFFER`
    - type is `BIT` or `INTEGER RANGE 0 TO 7`
      - or many more …
  - `END` specification_name
- Architecture ***specifications***
  - `ARCHITECTURE` implementation_name
    - `OF` specification_name `IS`
  - VHDL implementation with a choice of style
    - procedural -- conventional programming
    - dataflow -- concurrent programming
    - structural -- component connection
  - `END` implementation_name `;`
- VHDL BNF

Interface specification example

Largely taken from old NCSU ECE 214 lab

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Lab6 is
    port (
        RST: in STD_LOGIC;
        MyIn: in STD_LOGIC;
        CLK: in STD_LOGIC;
        MyOut: out STD_LOGIC
    );
end Lab6;
```

VHDL 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

See table 4-28 (p. 271)

`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)
  - Very ADA, but really a wordier C
  - `mod`, `rem`, `abs`, `and`, `or`, `nand`, `nor`, …
- Arrays
  - Table 4-33 (p. 274)
  - Extremely ADA
    - Array indexes do not need to start with 0
    - Boolean values can index an array
    - Arrays indexes can go downward
  - Array initialization in many varieties

- Conversion
  - What is a zero?
    - `0` integer
    - `'0'` bit
    - `'0'` `STD_ULOGIC` “Forcing 0”
    - `false` Boolean
  - Examples of conversions
    - Tables 4-37 to 4-39 (pp. 278 and 279)

- Functions and procedures
  - Not a lot of surprises

Structural descriptions

- Similar to a schematic
- List of connected components
- Nothing like a "normal" programming language

```
architecture Lab6_arch of Lab6 is
  signal CompRes: STD_LOGIC ;
  signal LastBit: STD_LOGIC ;
  component TwoBitCounter
    port (Clock, Reset, CarryIn:in STD_LOGIC; CarryOut:out STD_LOGIC);
  end component ;
  component DFlipFlop
    port (Clock, Clear, D: in STD_LOGIC; Q: out STD_LOGIC) ;
  end component ;
  component XOR2
    port (X, Y: in STD_LOGIC; Z: out STD_LOGIC) ;
  end component ;
begin
  C1: TwoBitCounter port map(CLK, RST, CompRes, MyOut) ;  -- count to 4
  D1: DFlipFlop port map(CLK, RST, MyIn, LastBit)      ;  -- save 1 bit
  X1: XOR port map(MyIn, LastBit, CompRes)             ;  -- compare
end Lab6_arch;
```

- Component declarations
  - List ports
- Concurrent statements
  - Connect ports
  - See Table 4-44 to 4-47 (p. 285) for exotic examples

Dataflow descriptions

- Concurrent signal-assignment states
- Similar to functional programming languages
- Works well for combinational specification
- Often used with behavioral statements

```
architecture HalfAdder_Arch of HalfAdder is

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*
- May not be implementable by all VHDL "compilers"

```
architecture Lab6_arch of Lab6 is
  type count_int is range 0 to 3 ;

begin
  process(CLK)
  variable trans_count: count_int ;
  begin
    if CLK'event and CLK = '1' then
      if RST='1' then
        trans_count := 0 ;
        MyOut <= '0' ;
      elsif (trans_count mod 2 = 0 AND MyIn = '1')
            OR (trans_count mod 2 = 1 AND MyIn = '0') then
        if (trans_count = 3) then
          trans_count := 0 ;
          MyOut <= '1' ;
        else
          trans_count := trans_count + 1 ;
          MyOut <= '0' ;
        end if ;
      else
        MyOut <= '0' ;
      end if;
    end if ;
  end process ;
end Lab6_arch;
```