Intro to VHDL

ESE170 Spring 2012

What is VHDL?

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PaulIsAJerk is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in STD_LOGIC;
           S : out STD_LOGIC);
end PaulIsAJerk;

architecture Behavioral of PaulIsAJerk is
   -- Component/Signal declarations go here

   signal internalSignal : STD_LOGIC_VECTOR(3 downto 0);

begin
   -- Descriptions of module "behavior" go here

   -- These can be structural, describing gates
   internalSignal(0) <= A or B;
   internalSignal(2 downto 1) <= (B xor C) & (A xor C);

   -- Or more like a behavioral description
   with(internalSignal(1 downto 0)) select
      S <= '1' when "11",
           '0' when others;

end Behavioral;
```

- HDL = Hardware Description Language
- VHDL = ...?
- Why should we use HDL's?

Basic Concepts

- Primary "data object" in VHDL is a ***signal***
- Declaration syntax: **signal <name> : <type>;**
  - Example: **signal A : STD_LOGIC;**
- ***Signals*** are like wires:
  - All things connected to A will see the same logic value
- Like variables in C/Java, ***signals*** have *types* and *values*
  - Many possible types in VHDL (next slides)
- There are also ***variables*** and ***constants***
  - Forget them for now (will cover in Lab 6)

Signal Types: Standard Logic

- ***Standard Logic***: probably the simplest possible type
- Keyword: **STD_LOGIC**
  - Example: **signal A : STD_LOGIC;**
- Use this to represent single-bit/wire logic values
- Two useful values: '0' and '1'
  - Others actually exist, less useful
- Single-bit values represented with **single-quotes**
  - e.g. '0' and '1'

Signal Types: Standard Logic Vector

- ***Standard Logic Vector***: a "collection" of **STD_LOGIC**
- Keyword: **STD_LOGIC_VECTOR(a downto b)**
  - **signal X : STD_LOGIC_VECTOR(3 downto 0);**
  - Generally, keep a > b
- Declares a group of logic values numbered 3 *down to* 0
  - How many bits is that?
  - Vector length is *a property of the signal type*
- Specify value as a sequence of 1's and 0's in **double-quotes**
- Use this to represent multi-bit values
  - E.g. an unsigned integer: $(13)_{10}$ = "1101"

Manipulating Logic Vectors

- There are a few ways to interact with vectors
- Access the ith bit of A: **A(i)**
  - Result is a **STD_LOGIC** (not a vector)
- Can also access a range of bits a to b: **A(a downto b)**
  - Result is a **STD_LOGIC_VECTOR**
- As an example:
  - **signal A : STD_LOGIC_VECTOR(3 downto 0);**
  - If **A** = "1001", what are the type and value of:
  - **A(2)** = ? **A(0)** = ?
  - **A(3 downto 1)** = ? **A(2 downto 3)** = ?

Basic Operators: Assignment

- Assignment operator: **a <= b**
  - Not a *less-than-or-equal* operator!
  - Left operand (a) takes the value of the right (b)

- Using the following declarations:

```
signal A : STD_LOGIC;
signal B : STD_LOGIC_VECTOR(3 downto 0);
```

Examples of assignments:

1. `A <= '1';`
2. `B <= "0011";`
3. `B(3) <= A;`
4. `B(3 downto 2) <= "10"`

- Types on each side must match (including vector width)

Assignment Concurrency

What does this VHDL do?

```
signal A : STD_LOGIC_VECTOR(3 downto 0);
A <= "0011";
A(3) <= '1';
```

Q: What is the value of A?

Assignment Concurrency (cont'd)

What does this VHDL do?

```
signal A : STD_LOGIC_VECTOR(3 downto 0);
A <= "0011";
A(3) <= '1';
```

Q: What is the value of A?

A: This is (syntactically valid) nonsense.
*Assignment is not sequential, it is **concurrent** (all at once)*

Think of assignments like "connecting" signals together

Basic Operators: Logical and Arithmetic

- You can also perform basic logical operations
  - **and or nand nor xor xnor not**

- Examples:
  - **(A and B) or (A and C) or (B and C)**
  - **A xor B xor C**

- Operator precedence is almost nonexistent here:
  - This is invalid syntax: **A and B or C**
  - Do this instead: **(A and B) or C**
  - **not** has higher precedence: **(A and B) or not C**

- Unless using all the same operator, use parentheses

Basic Operators: Logical and Arithmetic

- Logical operators work on vectors as well:

```
A <= "0101";
B <= "1100";
C <= A and B;
```

  - What is the type and value of C?

- You can also perform arithmetic. Examples:

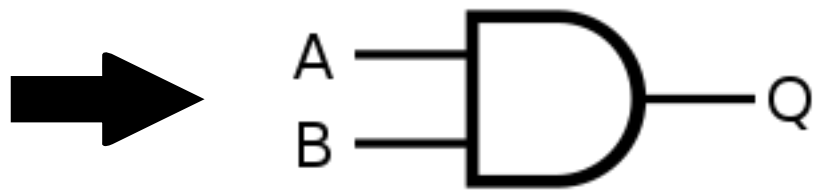
```
S <= A + B;
P <= A * B;
```

- In general, you should test arithmetic operators

VHDL Structure: Entity Declaration

- Defines the module's interface
- Inputs and outputs
- Same types as a **signal**
- Declared using the "port" keyword

```
entity MYAND2 is
  Port ( A : in STD_LOGIC;
         B : in STD_LOGIC;
         Q : out STD_LOGIC);
end MYAND2;
```



VHDL Structure: Architecture

- Defines the module
- Two options: Behavioral and Structural
- Behavioral
  - Define what the module does
  - Let the software figure out the hardware
  - e.g. with-select
- Structural
  - Explicitly state how hardware is arranged
  - e.g. logical operators

Behavioral Description: With Select

- ***With select*** = brute force
  - You describe the output value for every input case

- Syntax

```
with input_signal select
    output_signal <= value0 when case0,
                     value1 when case1,
                     ...
                     valueN when others;
```

- **value**'s must match type of **output_signal**
  - **case**'s must match **input_signal**
- You ***must*** include the **others** case
  - Helps you avoid programming 2^n cases!

With-Select Syntax

Given these declarations:

```
signal A : STD_LOGIC;
signal B : STD_LOGIC_VECTOR(2 downto 0);
signal out : STD_LOGIC_VECTOR(3 downto 0);
```

Is the following code valid?

```
with A select:
        out <= "0001"   when "0";
            <= B        when "1";
```

Aside: Concatenation Operator

- VHDL has a concatenation operator: **&**
- It can be inconsistent to work with...
- You definitely ***can*** do this:
  - `A <= B & C;`
  - Assuming widths match
- You definitely ***can't*** do this:
  - `B & C <= A`
- Other situations: just try it, remove it if it won't compile
  - Never necessary, just declare intermediate signal

Hierarchical Design

If you have this module:

```
entity MYAND2 is
  Port( A : in STD_LOGIC,
        B : in STD_LOGIC,
        C : out STD_LOGIC);
end MYAND2;

architecture Behavioral
of     MYAND2 is:

begin

    C <= A and B;

end Behavioral;
```

You can use it in another:

```
architecture Behavioral
of     ANOTHERMODULE is:
-- Declaration
COMPONENT MYAND2 is
  Port( A : in STD_LOGIC,
        B : in STD_LOGIC,
        C : out STD_LOGIC);
end COMPONENT;

signal C : STD_LOGIC;
begin
   -- Instantiation
   myInst : MYAND2 port map (
                          A=>'0',
                          B=>'1',
                          C=>C);
end Behavioral;
```

Behavioral Description: Processes

- Allows for sequential statements
- Declared within the architecture block
- Multiple processes executed concurrently
- Statements within each process executed sequentially

Processes: Syntax

- Syntax:

```
<name> : process(sensitivity list)
begin
    <statements>
end process;
```

- Sensitivity List
  - Defines what a process is dependent on
  - List of signals in the system
  - Process will execute when **any** of these signals change

- Can use constructs such as *if statements* in a process

Processes: Example

```
architecture Behavioral of ABSOLUTE is
    signal A : STD_LOGIC_VECTOR(3 downto 0);
    signal B : STD_LOGIC_VECTOR(3 downto 0);
begin
    abs: process(A)
    begin
        if A > "1000" then
            B <= not A + "0001";
        else if A <= "0111" then
            B <= A;
        else
            B <= "0000";
        end if;
    end process;
```

Conclusion

- Hardware (HDL) ≠ Software (code)
  - can't reuse signals like you can with variables
  - defining the layout of hardware
- Concurrent vs. Sequential execution
- Behavioral vs. Structural architecture