

CS232 VHDL Lecture

VHSIC Hardware Description Language [VHDL] is a language used to define and describe the behavior of digital circuits. Unlike most other programming languages, VHDL is explicitly parallel. It must be able to describe parallel computations because VHDL programs often describe the behavior of thousands, if not millions of gates, each gate acting independently as a function of its inputs and outputs.

VHDL programs have two parts.

- **entity** - the entity section is like a prototype in C, or an interface in Java. It contains a port statement that describes the name of the circuit and its inputs and outputs. All inputs and outputs are signals, which you can think of as wires. Each input or output must be labeled as to its usage (IN, OUT, INOUT) and type.
- **architecture** - the architecture is the circuit definition. The actual code that describes the circuit goes here. There are many ways to describe circuits in VHDL. A netlist definition instantiates gates and specifies which wires connect which inputs and outputs. A dataflow representation describes the flow of data through the circuit. A behavioral description uses more complex control structures to specify the functionality of the circuit over time.

Types

There are built-in types for VHDL, but they are not commonly used. Instead, most programs use a set of types defined by a IEEE standard library. The base type is `std_logic`, which represents one bit. The `std_logic` type uses the characters `0` and `1` to represent binary values. In addition, the `std_logic` type can take on values such as `X`, which is undefined, or `H` or `L`, which correspond to weak high and low values, which are important in some circuit designs.

VHDL has a flexible array mechanism that permits arbitrary range indexing with specification of the indexing direction: left or right with increasing index value. The basic IEEE array type is the `std_logic_vector`, which is an array of

type `std_logic`. The following example shows an 8 bit register, indexed from 7 to 0, with 7 being the index of the leftmost bit, or the most significant bit in the array.

```
signal register: std_logic_vector( 7 downto 0 );
```

The IEEE library also defines vector types for which addition and subtraction are defined. These types are signed and unsigned, and they require a range when defined, as below. The range of the indices determines the number of bits required by the signal.

```
signal counter: unsigned( 7 downto 0 );
```

Note that, because VHDL represents hardware, the programmer must specify the number of bits used in the type definition. An unsigned type does not have a pre-defined precision. If you need 64 bits of information, then you define the variable to have a range covering 64 bits. If you just need 3 bits, then you define the variable to have 3 bits.

Why not use the built-in VHDL types? The primary reason is that the IEEE types represent a standard that clearly states the hardware behavior of each type in the IEEE library. Simulators, therefore, ensure that the IEEE types work according to the standard, and the standard ensures that if you use your VHDL code to synthesize a real circuit, the behavior in simulation matches the behavior of the actual circuit.

If you don't care about creating hardware, there are a number of VHDL data types and control structures you can use.

Statements

Statements in VHDL all run continuously, in parallel. There are three major types of statements: assignments, port maps, and processes.

An assignment statement is, in effect, a wire connecting two nodes of the circuit. Any change in the right side of an assignment immediately affects the left side.

There are several variations of assignment statement, however, including conditional signal assignments that allow an exterior signal to control which value gets routed to the left-hand symbol.

— connects the value of B to the value of A with a wire
`A <= B;`

— connects the value of B to the value of A with a wire and a time delay
`A <= B after 5 ns;`

— selects B or (not B) to connect to A depending upon signal S
`A <= B when S = 0 else (not B);`

— combines a switch and delay
`A <= B after 5 ns when S = 0 else (not B) after 5 ns`

The **port map** statement is similar to creating an instance of a particular class. A port map creates an instance of an entity and connects the inputs and outputs of the entity with local signals. The syntax of a port map is as follows.

```
<label> : <entity> port map ( <argument list> );  
— example  
A0: andgate port map (I1 => A, I2 => B, Q => C);
```

In the example code, the label of the instance, which must be unique within scope, is A0, the entity, defined elsewhere, is called andgate, and it has three arguments. The => symbol connects the formal parameter names I1, I2, Q to their local signals A, B, C. Once port mapped, the andgate entity becomes part of the VHDL program and runs in parallel to other statements. Any time the inputs A or B change, the li!andgate! entity executes and updates the value of C.

Just as you can create many instances of a class, so you can create many instances of an entity using a port map. If you need ten andgate circuits, you can port map the entity ten times, linking them together using local signals. Each instance requires a unique label.

The **process** statement is somewhat more similar to a procedure or block statement. The purpose of a process is to permit the programmer to write a functional description of the circuit using more traditional control flow techniques.

Inside a process statement you can use structures like for loops and if-then-else branching. You can also use variables, as opposed to signals, but the use of variables is not recommended as it is difficult to compile code using variables into hardware.

When the process executes, the statements in the process get interpreted in serial order, which permits control flow statements to act as usual. However, despite the apparently normality of control flow and signal assignments inside a process, they are still not like a regular programming language.

The primary difference is how VHDL handles signal assignments inside a process. At the beginning of the process, each signal used in the process has a value. That value does not change as the system executes the behavior defined in the process. Instead, any assignments calculate the value of the right hand side based on the current value of the signals and then schedule that value for assignment when the process completes. One of the issues that arises is that a signal should receive a new value only once within one traversal of a process block. There may be many assignment statements that give a new value to a signal, but only one of those should actually execute given the control flow.

Note the similarity of the recommendation that a signal be assigned a new value only once within a process block and the same requirement in single-assignment C. In both cases, parallel computation makes multiple assignments to the same value difficult to resolve.

Unlike SA-C, however, VHDL does not forbid multiple assignments to a signal. One of the attributes of the IEEE std logic package types is that a subset of them contain resolution tables that specify the proper value when a signal receives two inputs simultaneously. In hardware, this enables bus-style hardware where any one of a number of signals can pull the bus low, but in the absence of a low signal the bus floats high. As noted earlier, the std logic type has a number of possible values besides 1 and 0. An H and a 0, for example, resolve to 0 because the H is a weak high.

Consider the process example below. First, note that a process must have a sensitivity list. The sensitivity list indicates which signals should cause the process to execute. The instructions inside the process only execute when a signaling variable changes. The process statement, therefore, is how you implement variables that change state only when an event occurs, such as a clock signal changing. Looking at the process itself, the if-statement is sensitive to the reset and clk signals, so they are in the sensitivity list.

Second, note that, while there are two signal assignments for each of A and B, at most one will execute in a single pass through the process. If the reset

signal is high, then the variables asynchronously receive the specified values (the clock is irrelevant). On the other hand, if the reset signal is not high, then the second condition specifies the rising edge of the clock: the clk signal is high and the clk signal changing caused the process to execute.

Finally, making the signal assignments in the rising edge case works because the assignments schedule the assignment to take place, they do not actually modify the value of A or B during the process execution. Instead, all updates take place simultaneously at the end of the process.

```

library ieee;
use ieee.std_logic_1164.all;

entity swap is
  port( reset , clk: in std_logic;
        Q: out std_logic );
end swap;

architecture test_of_swap is

  signal A : std_logic := 0 ; -- register A
  signal B : std_logic := 1 ; -- register B

begin -- test

  main: process (reset , clk)
  begin -- process main
    if reset = 1 then
      A <= 0 ;
      B <= 1 ;
    elsif clk = 1 and clk'event then
      A <= B; -- swap the values
      B <= A; -- works b/c swaps occurs simultaneously
    end if;
  end process main;

  Q <= A; -- runs simultaneously with the process

end test;

```

Inside a process, you can use for loops and while loops, which are most appropriately used to reduce the number of lines required to describe a process. For loops and while loops can use variables, which are not signals and do not have a physical analog. If you want to ensure your VHDL code can compile

to an actual circuit, the loop variables should not occur on the right side of an assignment except as index variables. Loop variables do follow standard rules of assignment within a process (the assignment occurs at the statement in which it is made). Variables also require a different syntax $j := j + 1$ rather than $A \leq B$.

To use variables properly, it is important to understand their role in the circuit description. A variable is not a wire or a bit of storage. It is not available to the circuit as an actual physical value. The role of a variable is to enable the programmer to describe the circuit more efficiently, such as looping over a sequence of wires that all behave similarly. A for loop, therefore, will almost always resolve into a set of copies of a circuit, which is important to remember, since chip size is a fixed quantity.

Examples

The following is an example of a process that incorporates an if-statement, signal assignments, and a case statement. The process describes a circuit that outputs different values depending upon the value of state, which must be a two-bit std logic vector.

```
process( reset , clock ) begin
  if reset = 1 then
    A <= 0 ;
    B <= 1 ;
  elsif clock = 1 and clockevent then
    case state is
      when "00" =>
        A <= 0 ;
        B <= not B;
      when "01" =>
        A <= 1 ;
        B <= not B;
      when "10" =>
        A <= B;
        B <= A;
      when others =>
        A <= 1 ;
        B <= 1 ;
      end case;
    end if;
  end process;
```

A testbench is a VHDL program designed to test other VHDL entities. The

example below is a testbench program for the swap entity described above. Note how the output of the S instance becomes the clock for the T instance. The delay form of the signal assignment generates the original clock and reset inputs. Since the testbench has no inputs or outputs, its entity statement is empty.

```

library ieee;
use ieee.std_logic_1164.all;

entity testswap is
end testswap;

architecture test of testswap is
  component swap
  port( reset , clk: in std_logic;
        Q: out std_logic );
  end component;

  signal clk , reset : std_logic;
  signal Q, R : std_logic;

begin — test

  clk <= 0 , 1 after 50 ns, 0 after 100 ns, 1 after 150 ns,
        0 after 200 ns, 1 after 250 ns, 0 after 300 ns;
  reset <= 1 , 0 after 10 ns;

  S: swap port map ( reset , clk , Q ); — these circuits are executing in parallel
  T: swap port map ( reset , Q, R );

end test;

```

Notes from Stephanie

- A test bench file is a main program that allows us to test a particular circuit; it provides input and reports the output. (Note: We need to write a test bench file if we are going to simulate VHDL using ghdl.)
- Concurrency. If we have 3 assignment statements, we are to assume they are processed concurrently (at the same time). Why? All three signal assignments are executing simultaneously and continuously be-

cause, in this case, they represent wires in a circuit, not memory locations.

- Note from Bruce's 333 notes:. The process statement in VHDL does implement a form of sequential semantics for a block of code, in the sense that the ordering of operations can matter to program functionality. However, later signal assignments within a process statement do not see the full effects of earlier signal assignments. While this may not make much sense from the point of view of a typical programming language, VHDL is intended to describe the behavior of physical circuits consisting of wires, logic gates, and flip-flops (1 bit storage units). Wires do not behave like memory locations in a computer, but VHDL permits us to attach identifiers to a wire and specify which wires connect to which other wires.

Note: These notes were adapted from Bruce Maxwell's VHDL notes from his fall 2010 CS333 lectures.