VHDL Primer

Tutorial #2

Mike Goldsmith

Feb 3, 2004, ~ 2 hr duration

Outline

- IEEE 1164 and Built-In Data types
- Arithmetic and Logic operators
- More VHDL Syntax
- Modularization and Instantiation
- Test benches

IEEE 1164 Data types

- std_ulogic
  - ‘U’ => Uninitialized
  - ‘0’ => Strong (forced) Zero
  - ‘1’ => Strong One
  - ‘X’ => Strong Unknown
  - ‘Z’ => High Impedance
  - ‘L’ => Weak Zero
  - ‘H’ => Weak One
  - ‘W’ => Weak Unknown
  - ‘-’ => Don’t Care

SEE: http://www.ecs.soton.ac.uk/~ajr1/vhdl_faq/std_logic_1164.html for gory details.

Built-in Data types

- Bit – ‘0’,’1’
- Boolean – true, false
- Integer – integer numbers, eg: 25
- Real – floating point numbers, eg: 2.57
- Time – an integer value + unit,eg: 5 ms
  - Time has **units** of fs, ps, ns, us, ms, sec, min, hr
- Character – ASCII char set

Arithmetic Operators

• +	addition
• -	subtraction
• /	division
• *	multiplication
• **	exponential
• mod	modulus
• rem	remainder
• abs	absolute value

Logic Operators

- not – negation
- and – logical and
- or – logical inclusive or
- xor – logical exclusive or
- nand – negated logical and
- nor – negated logical inclusive or
- xnor – negated logical exclusive or

Comparison Operators

=	equals
/=	inequality
<=	less than or equals
>=	greater than or equals
<	less than
>	greater than

More VHDL Syntax

- Conditional and Looping constructs **must** be within processes
- Conditional Statements
  - If-then constructs
  - Switch / Case constructs
  - ‘Condensed’ processes (when construct)
- Loops
  - Simple loops
  - While loops
  - For loops

More VHDL Syntax

- If-Then: basic conditional, if ‘a’ then ‘b’
- Sample code:

```
[if_label:] if condition then
   --statements
elsif alt_condition then
   --statements
else
   --statements
end if [if_label];
```

More VHDL Syntax

- Switch / Case – because writing ‘elsif’ 55 times really sucks.
- Sample code:

```
[case_label :] case signal_name is
    when value_1 =>        --if sig = value_1 then
      --statements
    when value_n =>        --elsif sig = value_n then
    when default =>        --else
end case [case label];
```

More VHDL Syntax

- Condensed conditional processes: write a conditional process on one line
- Sample code:

```
signal_1 <= signal_2 when condition else signal_3;
```

Replaces:

```
process( signal_2, signal_3, …) is
begin
  if condition then
          signal_1 <= signal_2;
  else
          signal_1 <= signal_3;
  end if;
end process;
```

More VHDL Syntax

- Simple loops: repeat a sequence of statements multiple times.
- Sample code:

```
[loop_label :] loop
   --statement(s)
   exit [loop_ label] [when condition];
   next [loop_ label] [when condition];
   --conditionally executed statement(s)
end loop [loop_ label];
```

More VHDL Syntax

- While loops: execute loop while exit conditions are unmet.
- Sample code:

[*loop_label* :] while *condition* loop

--statement(s)

next [*loop_ label*] [when *condition*];

--conditionally executed statement(s)

end loop [*loop_ label*];

More VHDL Syntax

- For loops: execute loop a fixed number of times
- Sample code:

[*loop_label* :] for *index* in *range* loop
  --statement(s)
  next [*loop_ label*] [when *condition*];
  --conditionally executed statement(s)
end loop [*loop_ label*];

- Loop index is a **variable** with scope limited to the loop

More VHDL Syntax

- Sequential (clocked) processes
- Sample code:

```
[process_label :] process( clk, d, q )is
begin
   if clk’event and clk = ‘1’  then
     q <= d;           --simple D flip-flop, notice no
                       --‘else’ case
   end if;
end process [process_ label];
```

More VHDL Syntax

- Sequential processes (again)
- Sample code:

```
[process_label :] process( clk, d, q )is
begin
   if rising_edge( clk ) then
     q <= d;          --simple D flip-flop, notice no
                      --'else' case
   end if;
end process [process_ label];
```

More VHDL Syntax

- Sequential processes (yet again)
- Sample code:

```
[process_label :] process is
begin
   wait until clk'event and clk = '1'
     q <= d;           --simple D flip-flop, notice no
                       --'else' case
end process [process_ label];
```

- Processes with 'wait' statements cannot have sensitivity lists

Modularization and Instantiation

- How to make one module talk to another
- All modules are instantiated by other modules; the entire design falls under a ‘top-level’ module
- The *interface* of a module must be defined for that module to be used. The *implementation* of the modules is selectable

Modularization and Instantiation

- Source code:

```
architecture arch_name of entity_name is
component comp_name is
   port(      inport: in type;
              outport: out type
   );
end component comp_name;
begin
--statements
```

Modularization and Instantiation

- Source code:

```
begin
[inst_label :] comp_name
port map( inport => signal_1, outport => signal_2 );
--statements
end architecture arch_name;
```

Modularization and Instantiation

- Example:

```
architecture foo of bar is
   component inv is
   port( d : in std_logic;
            q : out std_logic
   );
   end component inv;
   signal s_in, s_out : std_logic;
begin
   my_inverter: inv port map( d => s_in, q => s_out);
   --statements
end architecture foo;
```

Test Benches

- Used for simulation and verification
- Entity has **no ports**
- Architecture instantiates **one** main module to be tested, plus optionally support modules
- Module to be tested referred to as **device under test** (dut) or **unit under test** (uut)

Test Benches

- Sample code:

```
entity comp_name_tb is
end entity comp_name_tb;
architecture test_name of comp_name_tb is
component comp_name is
…
begin
uut: comp_name port map( … );
```

Test Benches

- Test benches use control and status signals to force operating conditions on the UUT and monitor the results
- Test benches can be executed in simulation and results displayed on a **waveform viewer**
- Test benches can also interact with the computer system, including file reading and writing, display to standard output, etc.

Test Benches

- Example:

```
entity int_tb is
end entity inv_tb;
architecture tb of inv_tb is
   component inv is
   port( d : in std_logic;
            q : out std_logic
   );
   end component inv;
   signal t_in : std_logic := ‘0’;
   signal t_out : std_logic;
```

Test Benches

- Example:

```
begin
   uut: inv port map( d => t_in, q => t_out);
   t_in <= not t_in after 20 us; --create a 50 kHz clk
end architecture tb;
```

- Test bench **must** have some form of signal that changes with time