VHDL Primer

Tutorial #1

Mike Goldsmith

Jan 27th, 2004, ~1 hr duration

Outline

- Usage of an HDL
- Steps of HDL flow
- Basic VHDL Syntax
- Example design flow

Using an HDL

- Why use a Hardware Description Language?
  - Schematic designs for large circuits are cumbersome
  - Complicated logic difficult to render at the schematic level
- Why use VHDL?
  - Supports Object Oriented-style design patterns and good abstraction/ modularization
  - Syntax is easy to learn and different *enough* from 'programming' languages to differentiate

Steps of the HDL Design Flow

- 1) Design / Compilation
  - Paper design of ‘functionality’
  - VHDL coding of functionality
  - VHDL compilers will check for *syntax* errors
- 2) Simulation
  - Testing of *logic* errors: proving the VHDL functionality meets the design functionality

Steps of the HDL Design Flow

- 3) Synthesis
  - Mapping process whereby functionality is assigned to gate-level design netlist
- 4) Place and Route (PAR)
  - Synthesized netlist is mapped onto specific mfg process in terms of physical location (place) and data interconnect layer (route)

Steps of the HDL Design Flow

- 5) Verification
  - Ongoing process parallel to Simulation, Synthesis, PAR and post-fab
  - Ensure functionality integrity between flow steps
  - Ensure that PAR'd design meets timing, power consumption, load balanced requirements

Basic VHDL Syntax

- Entity block: describes the *interface* of the module
- Sample code:

```
entity entity_name is
    port(     in_port : in std_logic;          --an input port
              out_port : out std_logic         --an output port
    );
end entity entity_name;
```

Basic VHDL Syntax

- Architecture block: describes the *implementation* of the module
- One entity can have **multiple** architectures
- Sample code:

```
architecture arch_name of entity_name is
begin
    --body contents
end architecture arch_name;
```

Basic VHDL Syntax

- Library inclusion: types, functions and other bits can be stored in a **library** which can be included in other designs (for reuse).
- Sample code:

```
library ieee;          --contains all base types and some
use ieee.std_logic_1164.all;    --type conversions
library work;          --your current design library
use work.my_package.entity_name;
use work.my_package.function_name;
```

Basic VHDL Syntax

- Process block: Structure within architecture to establish signal **dependencies**
  - Some logic **requires** the use of process blocks
  - Process is enacted by changing signals in its **sensitivity list**
- Sample code:

```
[process_name :]process( signal_name,… )is
begin
   --block body (process name is optional)
end process [process_name];
```

Basic VHDL Syntax

- Signals, Variables, Constants: your data
  - Variables are 'ignored' in simulation and cannot be used within the scope of the architecture or entity blocks*
- Sample code:

```
architecture arch_name of entity_name is
    signal signal_name,... : type [:= initial_value];
    constant constant_name : type := value;
begin
```

*Shared variables must be *declared* within the architecture scope, but are only *accessible* within process or function scope

Basic VHDL Syntax

- Sample code:

```
[process_name:]process(signal_name1,… )is
   variable variable_name,… : type [:= initial_value];
   signal signal_name2,… : type;
begin
   signal_name2 <= signal_name1; --signal assign
   variable_name := signal_name2; --variable assign
```

- Initial values are ignored in synthesis

Example Design Flow

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

entity inverter is
port(           input : in std_ulogic;          --values of ‘U’,‘1’,’0’,’X’,’H’,’L’,’W’,’Z’,’-’
                output: out std_ulogic
      );
end entity inverter;

--architecture number 1: a good (behavioural) design
architecture good of inverter is
begin
      output <= not input;                      -- not '1' = '0', not '0' = '1';
                                                -- all other inputs result in 'X' for 'unknown'
end architecture good;
```

Example Design Flow

```
--architecture number 2: a better behavioural design that accounts for all inputs
architecture better of inverter is
begin
        output <= '1' when input = '0' else '0';          -- input = '0' means output = '1'; all other inputs
                                                          -- (including '1') result in an output of '0';
end architecture better;
```

Example Design Flow

```
--architecture number 3: the best architecture that handles all inputs in the most appropriate manner
architecture best of inverter is
begin
        process (input) is
        begin
                case (input) is
                when '0' =>                             --force logic zero
                                output <= '1';          --  output a logic one
                when '1' =>                             --force logic one
                                output <= '0';          --  output a logic zero
                when 'X' =>                             --force unknown
                                output <= 'X';          --  output an unknown
                when 'L' =>                             --weak logic zero
                                output <= '1';          --  output a logic one
                when 'H' =>                             --weak logic one
                                output <= '0';          --  output a logic zero
                when 'W' =>                             --weak unknown
                                output <= 'X';          --  output an unknown
                when others =>                          --all unspecified values
                                output <= 'X';          --  output an unknown
                end case;
        end process;
end architecture best;
```