Guidelines for VHDL-based Design

VHDL coding basics

- Here, we describe a set of basic guidelines for coding in VHDL
- The objective is to produce readable VHDL source that is targeted for synthesis
- Many factors affect the ultimate performance of a VHDL-based design including
  - VHDL coding style
  - EDA tool(s) used: synthesis and place/route
  - Characteristics of the target device
  - User/designer awareness of how the above 3 items interact and depend on each other
Sources of information

• Various sources of information exist that should help you learn and use good coding practice
  – Altera online help in Quartus gives a set of guidelines for VHDL coding
  – Altera website provides numerous design examples
  – Various VHDL-related websites
    • Use with caution

• Do not attempt to code what you do not understand

Sources of information (continued)

• Various internet resources
  – Altera provided design examples
    • www.altera.com/support/examples/exm-index.html
  – Literature for various Altera device families
    • CYCLONE V -- www.altera.com/literature/lit-cyclone-v.jsp
  – Various related sources
    • www.eetimes.com – Numerous IC resources including FPGAs
    • www.fpga-faq.com – FPGA FAQ
    • www.fpga4fun.com – Various FPGA projects
Simulation accuracy and speed

- **Accuracy**
  - Ensure the sensitivity list of `process` statements is complete (correct)

- **Speed**
  - Use a `process` statement in preference to concurrent signal assignments
    - Reduces the number of signals a simulator must continually monitor for changes and so improves simulation speed
  - Design models to minimize the number of signals in a `process` statement
    - Less signals to monitor will improve simulation speed
  - Do not model many small `process` statements
    - If there are many registers being clocked from the same clock source, it is better to put them in one process.

Synthesis modeling recommendations

- When modeling purely combinational logic, ensure signals are assigned in every branch of conditional assignment statements
- Use `case` statements in preference to `if` statements containing `else-if` clauses
- The `others` default `case` branch ensures all branch values are covered.
- There is no real need to use a wait statement to infer flip-flops.
  - The `if` statement can do all that wait does and has the added advantage that combinational logic can be inferred.

General HDL modeling recommendations

- Before attempting to code a model, know and understand what it is you are modeling
- Divide your design into sub-designs when possible
  - Promotes short, easy to read, reusable code
- Make models as generic as possible
  - Parameterize bus widths
- Use meaningful signal names.
  - For active low signals use `<signal_name>_n` for clearer understanding and easier debugging
- Use comments liberally.
  - A header should describe each module and each signal declaration should have a comment
  - Dated, inconsistent comments are worse than none at all.

Assigning Values in Different Number Formats

- Assignment statements (with VHDL 1993 syntax and newer) can use different number formats as appropriate
  - X (hex), O (Octal)
- Assume `reg` is STD_LOGIC_VECTOR(7 DOWNTO 0)

  \[
  \text{reg} \leftarrow "10101111"; \text{ is the same as } \text{reg} \leftarrow x "AF";
  \]

- Underscores can be used to separate fields in long strings to enhance readability

  \[
  \text{instruction} \leftarrow "000000000100001000010000100000100000";
  \text{instruction} \leftarrow "000000_00001_00001_00001_00000_00000_100000";
  \]
VHDL Aggregates

- An **aggregate** is a collection of items that are gathered together to form a total quantity.

```vhdl
signal v : std_logic_vector(7 downto 0);
```

- ```vhdl
   v <= (others => '0'); -- "00000000"
   v <= ('1', '0', others => '0'); -- "10000000"
   v <= (4 => '1', others => '0'); -- "00010000"
   v <= (3 DOWNTO 0 => '0', others => '1'); -- "11110000"
   -- v <= ("0000", others => '1'); -- illegal!
```

VHDL Generics

- VHDL generics may be used to parameterize code making components as modular (and hence) reusable as possible.

For example:

```vhdl
entity counters is
  generic (width : integer := 8);
  port(
    d   : in   std_logic_vector(width-1 downto 0);
    clk : in   std_logic;
    clear : in std_logic;
    load : in std_logic;
    up_down : in std_logic;
    qd   : out std_logic_vector(width-1 downto 0));
end counters;
```