HDL Coding Rules and Guidelines

Gord Allan

September 3, 2003

Contents

1 HDL Coding Guidelines

Many of these items are taken, with permission, from "HDL Coding Guidelines," by Damjan Lampret and Jamil Khatib, June 7, 2001, www.opencores.org

1.1 Description

The guidelines are of different importance, and fall into three classes

- *Good practice* - signifies a rule that is common good practice and should be used in most cases. This means that in some cases there are specific problems that violate this rule.
- *Recommendation* - signifies a rule that is recommended. It is uncommon that a problem can not be solved without violating this rule.
- *Strong recommendation* - signifies a hard rule, this should be used in all situations unless a very good reason exists to violate it.

1.2 Resets

Resets make the design deterministic. It prevents reaching prohibited states and avoides simultation/synthesis mismatches.

- Recommendation: All flip-flops should have a reset. *Prevents simulation/synthesis mismatches.*
- Recommendation: Resets should be active-low. *Cell libraries contain active-low reset flops. Coding them as such prevents the insertion of unwanted buffering on the reset logic.*
- Recommendation: Resets should be asynchronous. *Most flops have them. Maintains compatibility between ASIC/FPGA code. Easier debugging.*
- Good Practice: The active-low reset should be applied asynchronously, de-asserted synchronously.

```
// synchronize the external reset
always @(posedge clk)
    rst_sn <= rst_an_pushbutton;

// reset comes off once when pushbutton is 'high' AND posedge clk
assign rst_an = rst_sn & rst_an_pushbutton;
```

All flops reset as soon as the pushbutton is applied — eases debugging. The reset track has a full clock cycle to de-assert after a clock edge — eases timing.

- Strong Recommendation: Active-low, asynchronously reset flops are coded as follows:

```
always @(posedge clock or negedge rst_an)
    if(~rst_an)  q <= 0;
    else         q <= d;
```

- Strong Recommendation: On an FPGA or CPLD the reset should be globally connected. *FPGAs and CPLDs have fixed routing that are connected to all device resources.*

1.3 Clocks

- Recommendation: Signals that cross different clock domains should be sampled before and after the crossing domains (double sampling is preferred). *Prevent meta-stability state.*

- Good practice: Use as few clock domains as possible in any design.

- Recommendation Do not use clocks or reset as data or as enable. Do not use data as clock or as reset. Code such as this must be prevented:

```
always @(posedge signal) begin ... end
```

  Synthesis results may be different than HDL, causes timing verification problems.

- Recommendation: Don't use gated clocks. *It negatively effects timing and can cause unwanted glitching. If necessary, they will be implemented at the top level of an IC.*

- Strong Recommendation: Clock signal must be connected to global dedicated reset or clock pin on an FPGA or CPLD. *This is because such pins provide low skew routing channels.*

1.4 Naming Conventions

- Good Practice: Try to write one module in one file. *The File name should be the same as the module's name.*

- Recommentation: Try to use named notation for instantiating instead of positional notation. *For easier debugging and understanding the code.*

- Good Practice: Keep the same signal name through different hierarchies. *So tracing after the signal will be easy. Enable easy netlist debugging.*

- Good Practice: Suffix signal names with *_a* for asynchronous and *_n* for active-low. eg. *rst_an* is an active-low asynchronous reset signal. *Helps keep logic clear.*

- Recommendation: Start buses at bit 0. *Some tools don't support buses that don't start at bit 0.*
- Recommendation: Use MSB to LSB for busses. *This is to avoid misinterpretation through the design hierarchy.*

1.5 Synchronous design and timing optimization

- Strong Recommendation: Use only synchronous design. *It avoids problems in synthesis, in timing verification and in simulations.*
- Recommendation: Avoid using latches. *They causes synthesis, testing, and timing verification problems.*
- Strong Recommendation: Do not use delay elements.
- Strong Recommendation: All blocks external IOs should be registered. *It prevents long timing paths.*
- Good Practice: Block internal IOs should be registered. *This is a design issue but is recommended in most cases.*
- Recommendation: Avoid using FlipFlop with negedge clock. *Causes synthesis problems and timing verification problems.*
- Strong recommendation: Include all signals that are read inside a combinational process in its sensitivity list. (i.e. Signals on Right Hand Side RHS of signal assignments or conditions. *This is to prevent simulation/synthesis mismatches.*
- Strong recommendation: Ensure variables are assigned in every branch of a combinational logic process. *Prevents inferring of unwanted latches.*

1.6 General rules

- Strong Recommendation: In RTL, never initialize registers in their declaration. Use proper reset logic. *Initialization statements can not be synthesised.*
- Recommended: Write fsms in two always blocks — one for sequential assignments (registers) and the other for combinational logic. *This provides more readability and prediction of combinational logic size.*
- Strong Recommendation: Use non blocking assignment (<=) in clocked blocks, and blocking assignment (=) in combinational blocks. *Synthesis tools expects for this format. Makes the simulation respond deterministically.*
- Recommendation: Try to use the 'include' command without a path. *HDL should be environment independent.*

- Good Practice: Compare buses with the same width. *The missing bits may have unexpected value in the comparison process.*

- Strong recommendation: Avoid using long if-then-else statements and use case statement instead. *This is to prevent inferring of large priority decoders and makes the code easier to be read.*

- Strong Recommendation: Avoid using internal tri-state signals. *They increase power consumption and make backend tuning more difficult.*

1.7 Simulation and Debugging

- Strong Recommendation: Test benches should be intelligent enough to determine sucessfull operation without user interaction. *Reduces development time and human oversights.*

- Strong Recommendation: The same test-bench should be used for RTL and gate-level simulations. *Ensures that synthesis and optimization is sucessfull.*

- Recommendation: Try to write the test bench in two parts, one for data generation and checking and one for interfacing to the device-under-test. The interface to the device should be written with normal hardware coding rules in place. *This is to isolate data (results checking) from the hardware interfacing. By writing the interface logic with conventional hardware description (ie. registers), it allows for interchangable RTL and gate level simulation.*

- Good Practice: Use *$display("%t - (%m) Message", $time, vars...)* liberally to provide information while debugging a design.

- Good Practice: Ensure the 'timescale command is specified only once. *Different 'timescale causes simulation problems: races and too long paths.*