Verilog Coding Style

He Weifeng

Outline

- Naming Conventions
- Comments
- Code Style
- Other Coding techniques

Naming Conventions

- Rule 1.1 One design unit per file
  - A file must not contain more than one design unit. Everything contained in a design unit must be completely contained in a single module/endmodule construct.
  - Reason: Simplifies design modifications.

- Rule 1.2 First Character of a name
  - Names must start with a letter, not a number or underscore
  - Reason: Names starting with a number or underscore may cause conflicts with tools.

Naming Conventions

- Rule 1.3 All names must be unique irrespective of case
  - Case must not be used to differentiate cell names or signals.
  - reason: Some tools may not be able to differentiate on the basis of case. Moving designs between Verilog (case sensitive) to VHDL (case insensitive) is facilitated by a case insensitive design style.

- Rule 1.4 Constant, parameter and block label are upper case
  - Consistent upper case spelling of parameter and constant names must be used. Therefore, all letters must be upper case for: constants, parameters, block labels;
  - Provides a mechanism to identify objects that do not go through data changes during simulation.

Naming Conventions

- Rule 1.5 Signal, constructs and instance labels are lower case
  - Consistent lower case spelling of signal and construct names, and instance labels must be used. Therefore, all letters must be lower case for: signals, constructs and instance labels
  - reason: Differentiates signals and constructs from objects that do not change data during simulation, and maintains a consistent look and feel between designs

- Rule 1.6 Meaningful signal and variable names
  - The lower-case name must contain the purpose of the variable/signal.
  - Reason: Description for understanding the design.
  - example: data_bus, set_priority

Naming Conventions

- Rule 1.7 Underscore separate names composed of several words
  - For names composed of several words, underscore separated lower case letters must be used
  - reason: Improves readability.
  - example: ram_addr

- Rule 1.8 Active low signal names use _b
  - Where a signal uses active low polarity, it must use the suffix _b.
  - reason: Meaningful, consistent names aid in understanding the design.
  - example: enable_data_b, reset_b

Naming Conventions

- Rule 1.9 Clock signal names use _clk
- Signals that are used for clocking that do not have the word clock or clk already in their names must use the suffix _clk.
- reason: Meaningful, consistent names aid in understanding the design.
- example: fifo_transmit_clk

- Rule 1.10 State machine signal names use _next
- It is recommended that next state signals of state machine use the suffix _next.
- reason: Meaningful, consistent names aid in understanding the design.
- example: transmit_next

Naming Conventions

- Rule 1.11 Test mode signal names use _test
- It is recommended that test mode signals use the suffix _test.
- reason: Meaningful, consistent names aid in understanding the design.
- example: parallel_clk_test

- Rule 1.12 Consistent names throughout the hierarchy
- It is recommended that signal or design unit names remain the same throughout the hierarchy of the entire IP. Names associated with multiple instances are recommended to have the name indexed by an integer.
- reason: Improves readability, removes confusion, avoids buffer insertion during synthesis.

Naming Conventions

- Rule 1.14 Verilog names are equivalent to documentation names
- All signals and blocks in the Verilog RTL that are referenced in the documentation are recommended to maintain the same name as in the documentation.
- reason: Simplifies understanding of an HDL model.

Course Overview

- Naming Conventions
- Comments
- Code Style
- Other Coding techniques

Comments

- Rule 2.1 Each file must contain a file header
- Every file must contain a header. All fields must be included, even if the data is N/A.
- reason: Provides a standard means of supplying pertinent design information.

- Rule 2.2 Include file construct type
- The header must include the highest level construct contained in the file.
- reason: Provides an easy way to determine what a file contains.
- example: module, macromodule

Comments

- Rule 2.3 Include point of contact information
- Every file header must include the originating department, author, and author's email address.
- reason: Required for inquiries beyond the scope of the documentation for the design.

- Rule 2.4 Include a release history
- Header must include release history only for the IP changes checked into the Repository. This information is useful to the integrator. Local release history should not be included in the header.
- reason: Required to track the revision history of the design.

Comments

- Rule 2.5 Include a purpose section
- The header must contain a purpose section describing the modules functionality. The purpose must describe what the unit provides and not how.
- reason: Aids understanding of module functionality.

- Rule 2.6 Include a parameter description
- Headers must contain information describing the parameters being used in the construct. The default value must be listed.
- reason: Aids understanding of HDL code.

Comments- An example of file header

```
+FHDR----------------------------------------------------------------
// Copyright (c) 1999, Motorola.
// Motorola Confidential Proprietary
// ---------------------------------------------------------------------
// FILE NAME :
// TYPE : TYPE can be module, macromodule
// DEPARTMENT :
// AUTHOR :
// AUTHOR'S EMAIL :
// ---------------------------------------------------------------------
// Release history
// VERSION Date AUTHOR DESCRIPTION
// 1.0 6 Oct 98 name
// ---------------------------------------------------------------------
// KEYWORDS : General file searching keywords, leave blank if none.
```

Comments- An example of file header

```
// ------------------------------------------------------------------
// PURPOSE : Short description of functionality
// ------------------------------------------------------------------
// PARAMETERS
// PARAM NAME RANGE : DESCRIPTION : DEFAULT : VA UNITS
// e.g.DATA_WIDTH_PP [32,16] : width of the data : 32 :
// ------------------------------------------------------------------
// REUSE ISSUES
// Reset Strategy :
// Clock Domains :
// Critical Timing :
// Test Features :
// Asynchronous I/F :
// Scan Methodology :
// Instantiations :
// -FHDR------------------------------------------------------------
```

Course Overview

- Naming Conventions
- Comments
- Code Style
- Other Coding techniques

Code Style

- Rule 3.1 Use two to four space code indentation
- A constant indentation of two to four spaces must be used for code alignment. Do not use tab stops. Use spaces and empty lines to increase the readability of the code.
- reason: Improves readability.

- Rule 3.2 One Verilog statement per line
- One line must not contain more than one statement. Do not concatenate multiple semicolon separated Verilog statement on the same line.
- reason: Improves readability. Easier to parse code with a design tool.

Code Style

- Rule 3.3 Use one line comments
- One line comments (//) must be used. Do not use multi-line (/*...*/) comments.
- reason: Improves readability and improves code parsing.

- Rule 3.4 Explicitly indicate port type
- Explicit port typing indication must be used. One port per line must be declared.
- reason: Improves readability and understanding of the code, as well as parsing the code with scripts.
- example: use: input a; input b; do not use: input a, b;

Code Style

- Rule 3.6 Port declaration order
- It is recommended that ports be declared in the same order as in the port list.
- reason: Improves readability

- Rule 3.7 Comment port listings
- It is recommended that a descriptive comment follow each port listing, preferably on the same line.
- reason: Improves readability.

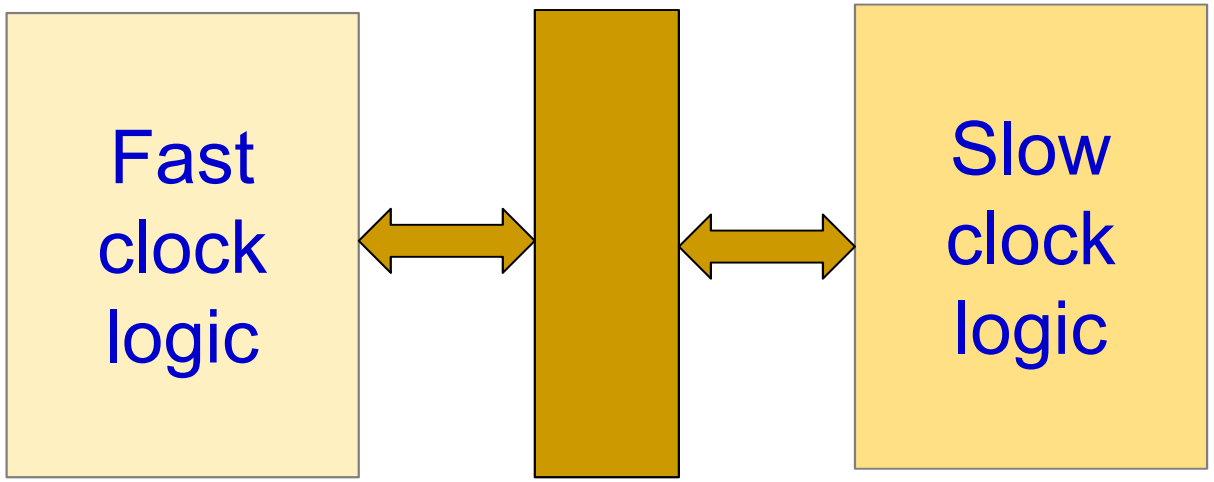
Code Style

- Rule 3.8 Partition clocks into separate block
- clock generation circuit must be kept in a separate module at the top level of the IP module or at the same logical level in the hierarchy as the block to which the clocks apply.
- reason: Eases test strategy generation, and limits exceptions to the coding standards to a small module. It also improves the portability of the code to a different end use clocking scheme.

- Rule 3.9 Register all module outputs
- It is recommended that all module outputs be registered.
- reason: Simplifies the timing interface to other modules and the synthesis process.
- exception: Not applicable to non-synthesizable

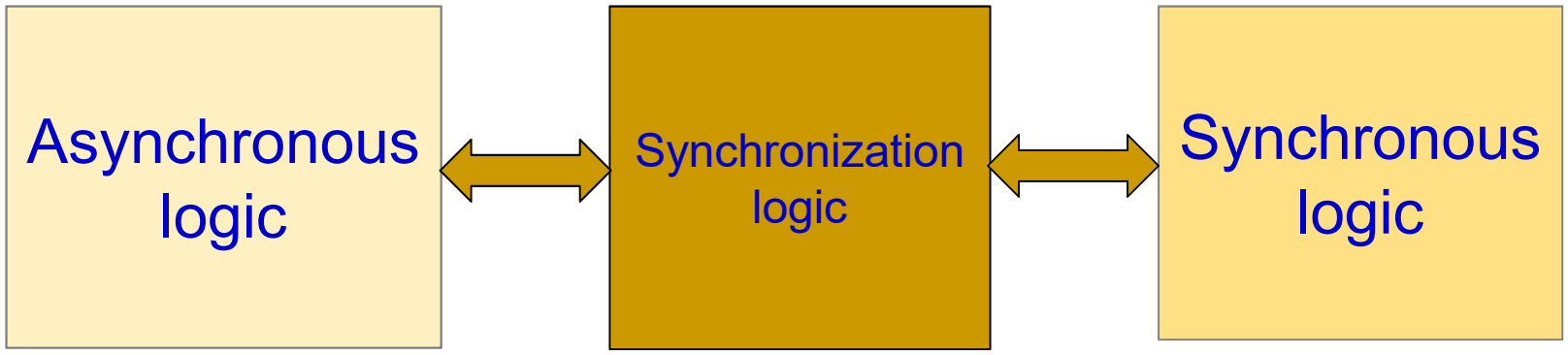
Code Style

- Rule 3.10 Partition clock domains
- It is recommended that separate clock domains be partitioned into separate blocks. The synchronization logic between the two domains is recommended to also be partitioned into a separate block.
- reason: Eases synthesis. Avoids over-constraining one clock domain



Code Style

- Rule 3.11 Partition asynchronous logic
- It is recommended that asynchronous logic be partitioned from synchronous logic.
- reason: Avoids synthesis problems and eases constraints and scripting.
- exception: Not applicable to non-synthesizable blocks



Code Style

- Rule 3.12 Partition state machines
- It is recommended that the FSMs be coded into their own block.
- reason: Avoids synthesis problems and eases constraints and scripting.

- Rule 3.13 Do not mix structural and behavioral RTL code within a construct
- It is recommended that the model partitioning isolate the structural code from the behavioral RTL code. Each construct must be either purely structural or purely behavioral RTL. Structural code is a line of code containing only connectivity information.
- reason: This can result in synthesis problems and limitations.

Code Style

- Rule 3.14 Do not assign signals to x
- Signals must not be assigned to x. Known legal signal values must be assigned to all signals.
- reason: Avoids x propagation through the circuitry.

- Rule 3.15 Do not infer latches in functions
- Latches must not be inferred in any function call
- reason: Functions always synthesize to combinational logic.

Code Style

- Rule 3.16 Synchronously reset storage elements
- It is recommended that synchronous resets be used wherever possible.
- reason: Eases scan chain insertion and test.

- Rule 3.17 Complete always sensitivity list
- All always blocks inferring combinational logic or a latch must have a sensitivity list. The sensitivity list must contain all input signals.
- reason: Synthesis will create a structure that depends on all values read regardless of the sensitivity list, which can lead to potential mismatches between behavioral and gate-level simulation.

Code Style

- Rule 3.18 One clock per always block
- Only one clock per Verilog always block must be used in a synchronous process.
- reason: This is required to restrict each process to a single type of memory-element inference.

- Rule 3.19 Wait statements and # delay statements are not allowed.
- Wait statements, both explicit and implicit, must not be used in the design.
- reason: The wait statement and the # delay are generally not supported by the RTL to gate level synthesis process.

Code Style

- Rule 3.20 Specify combinational expressions completely
- Conditional expressions must be specified completely (i.e., assign a value to a variable or signal under all conditions).
- reason: Synthesis tools infer memory elements if a combinational expression is not completely specified.

```
always @(signal_names)
  case (signal_names)
    3'b000,
    3'b001 : output = 4'b0000;
    3'b010 : output = 4'b1010;
    3'b011,
    3'b100,
    3'b101 : output = 4'b0101;
    3'b110,
    3'b111 : output = 4'b0001;
  endcase
```

```
Always @(signal)
   begin
      if (signal =1'h1)
      output =4'b0;
   end
```

Code Style

- Rule 2.21 Do not use the initial statement
- Reset functionality along with signal and variable initialization must be modeled explicitly. Do not use the initial construct.
- reason: The synthesized gate-level netlist will not use the initialization construct which will result in simulation mismatches between the behavioral and gate-level models.
- exception: Allowed in testbench constructs.

- Rule 2.22 Expressions are not allowed in port connections
- Expressions must not be used in port connections.
- reason: May result in glue logic between blocks. Eases understanding of the HDL.
- exception: Bus concatenations are allowed.

Course Overview

- Naming Conventions
- Comments
- Code Style
- Other Coding techniques

Other Coding techniques

- Rule 4.1 Verilog primitives are prohibited
- Verilog primitive must not be used.
- exception: Eases understanding of the HDL.

- Rule 4.2 Use non-blocking assignments (<=) in edge-sensitive constructs
- Non-blocking assignments (<=) must be used in edge-sensitive sequential blocks. Blocking assignments (=) are not allowed.
- reason: Use of blocking assignments in edge-sensitive sequential code can result in mismatches between pre and post-synthesis simulations.

Other Coding techniques

- Rule 4.2 Use non-blocking assignments (<=) in edge-sensitive constructs

example: Edge-sensitive code written with non-blocking assignments:

```
always @(posedge clk) begin
    regb <= rega;
    rega <= data;
    end
```

example: Code that may result in pre and post-synthesis simulation mismatches:

```
always @(posedge clk) begin
    rega = data;
    regb = rega;
    end
```

Other Coding techniques

- Rule 4.3 Declare all internal wires in one section
- The internal wire declaration must follow the port I/O declarations at the top of the module.
- reason: Eases understanding of code.

- Rule 4.4 Internal wires must be declared
- All internal wires must be declared instead of implied.
- reason: Although Verilog can handle implied wires, all internal wires must be declared to avoid confusion

Other Coding techniques

- Rule 4.5 Use of casex is not allowed
- case or casez must be used for all case statements.
- reason: Casex treats the X and Z states as don't cares in synthesis, which can result in different simulation behavior pre- and post-synthesis.

- Rule 4.6 Use parameters for state encodings
- It is recommended that enumerated parameters be used to encode the different states of a state machine.
- reason: This eases retargeting to different state machine implementations, for example changing from an encoded 1-hot style to gray code.

example:

```
parameter [1:0] // synthesis enum state_info
RESET_STATE = 2'b00,
TX_STATE = 2'b01,
RX_STATE = 2'b10,
```

Other Coding techniques

- Rule 4.7 Top level glue logic is not allowed
- It is recommended that gates not be instantiated or inferred at the top level of the design hierarchy.
- reason: Synthesis results are limited because the top level logic cannot be combined for optimization.

- Rule 4.8 Avoid ports of type inout
- It is recommended to use ports of type input or output. Ports of type inout (bidirectional) should be avoided.
- reason: Bidirectional implementations may cause contention problems. Avoiding bidirectionals also eases synthesis and test insertion.

Other Coding techniques

- Rule 4.9 Blocking assignment usage
- It is recommended that combinational logic use blocking assignments.

- reason: Blocking assignments typically simulate faster than non-blocking. However, to avoid dependencies on execution order, care must be taken when inferring latches.
- exception: Latches in mixed latch/FF designs should be written with non-blocking assignments.

Other Coding techniques

- Rule 4.10 Default case assignments in case statements
- It is recommended that default case assignments be used for all combinational logic case statement descriptions.
- reason: It may be possible to decode unexpected combinations of the case selects, resulting in pre- and post-synthesis simulation mismatches.
- note: Additional logic may be inferred using the default case.
- example: The bus is assigned to an illegal value if a case select combination which is not explicitly decoded is selected, e.g. 3'b000.

```
casez (sbus_sel[2:0])
  3'b100: sbus_data[31:0] = in_bus[31:0];
  3'b?1?: sbus_data[31:0] = 32'b1;
  3'b?01: sbus_data[31:0] = data_rd[31:0];
  default: sbus_data[31:0] = 32'h0;
endcase
```

Reference

- Verilog HDL Coding Semiconductor Reuse Standard, Motorola, 1999