VLSI System Design

Verilog Coding Style

李昆忠
國立成功大學電機系

06-2757575 X 62371
kjlee@mail.ncku.edu.tw

RTL Coding Guidelines

- Basic coding practices
- Coding for portability
- Guideline for clocks & reset
- Coding for synthesis
- Partitioning for synthesis
- Designing with memory

Ref : Reuse Methodology Manual (RMM)
Keating & Bricaud, 3rd Edition, 2002.

General Rules

- Simple: structure, clocking, etc.
- Consistent: naming
- Regular: module size, output registered
- Understandable: comments, meaningful names, parameters & constants

Basic Coding Practices

- Naming convention
  - Lower case for signals, variables, ports
  - Upper case for constants (define directive)
  - Meaningful names
    Ex: `ram_addr` for RAM address, `sys_bus` for system bus.
  - Short but descriptive parameters (for synthesis)
  - Prefix `clk` for clocks, prefix `rst` for reset.
  - `*_x` for active low, e.g., rst_n, act_b.
  - Use `[x:0]` rather than `[0:x]`
  - Use `*_r` for output of a register
  - Use `*_z` for tristate signal

- Do not use HDL reserved words.

Basic Coding Practices (cont.)

- State variables: `<name>_cs` for current state, `<name>_ns` for next state
- Use informational header for each source file:
  - Legal information (confidentiality, copyright, restriction, etc.), filename, author, date, version history, main contains.
- Use concise but explanatory comments where appropriately.
- Avoid multiple statements in one line.
- Use indentation to improve readability.
  - Use indentation of 2 spaces; do not use tab
- Port (core I/O) ordering
  - One port per line; a comment followed each port declaration
  - Follow the following order:
    - clocks, resets, enables, other control signals, data & Address lines

Basic Coding Practices (cont.)

- Port mapping
  - Use explicit mapping
  - Use named association rather than positional association

Ex:

```
 TagRam TagRam (
.Address    (PAddress[‘INDEX]),
.TagIN       (PAddress[‘TAG]),
.Tagout      (TagRAMTag[‘TAG]),
.Write       (Write),
.Clk         (Clk)
);
```

Basic Coding Practices (cont.)

- Use functions wherever possible (for reusability and avoid repetition).
- Use vector operations (array) rather than loops for faster simulation.

Ex:

Poor:
```
for (i = 1, i < k, i ++)
    c_vec[i] = a_vec[i] ^ b_vec[i];
```

Better:
```
c_vec = a_vec ^ b_vec;
```

Coding for Portability

- Use constants instead of hard_coded values.

  Ex: Poor:
  ```
  wire[7:0] in_bus;
  reg[7:0] out_bus;
  ```
  Better:
  ```
  `define BUS_SIZE 8
  wire[BUS_SIZE-1:0] in_bus;
  reg[BUS_SIZE-1:0] out_bus;
  ```

- Use separate constant definition files
  - Specify constant definition file (e.g., DesignName_constances.v) only on tool command line.

- Use technology-independent libraries
  - Use DesignWare Fundation Library of Synopsys for arithmetic components (adders, multipliers,comparators, incrementers/decrementers, sum of product, sin/cos,modulus/divide, square root, arithmetic and barrel shifters)

Coding for Portability (cont.)

- Avoid instantiating technology-specific gates.
- Use Synopsys generic technology library, GTECH, to instantiate a gate.
  - AND, OR, NOR
  - 1-bit adders & half adders
  - Multiplexers
  - Flip-flops
  - Latches
  - Multiple level gates, such as AND-OR, AND-NOT, AOI
- Isolate technology-specific gates in a separate module.
- Do not use those descriptions that cannot be translated from Verilog to VHDL and vice versa.

Clocks & Resets

- Preferably use a single global clock and positive edge-triggered flip-flops only.
- Avoid both positive and negative edge-triggered flip-flops.
- Separate positive-edge and negative-edge triggered flip-flops into different modules **--- make scan design easier.**
- Avoid clock buffers **--- leave it to clock insertion tool.**
- Avoid gated clock **--- to avoid false clock or glitch, and improve** testability.
- Avoid internally generated clocks **--- for testability.**
- If internally generated clocks are necessary, separate it in a top-level module.
- Avoid internally generated resets.

Coding for Synthesis

- Registers (flip-flops) are preferred.
- Latch should be avoided.
- Use design tools to check for latches.
- Poor coding may infer latches:
  - Missing **`else`** statement in **`if-then-else`** structures.
  - Missing assignments or conditions in **`case`** structures.
- To avoid the undesired inferred latches:
  - Assign default values at the beginning of a process.
  - Assign outputs for all input conditions.
  - Use default statements for case structures.
  - Always consider the else case in a if-then-else structure.
- If a latch must be used, well document it and prepare to make it testable via a mux.

Coding for Synthesis (cont.)

- Avoid combinational loops.
- Always use nonblocking assignments in **`always @(posedge clk)`** blocks.
- Use **`case`** instead of nested **`if-then-else`** statements.
- Separate FSM and non-FSM logic in different modules.
- Keep late-arriving signals with critical timing closest to the output of a module, e.g., earlier in if-then-else structures.
- Do not use delay constants in RTL code to be synthesized.

Partitioning for Synthesis

- Partitioning can result in better synthesis results, faster compile and simulation time, and enable simpler synthesis strategy to meet timing requirement.
- Try to register all outputs of each block in a hierarchical design
- Locate related combinational logic in a single module
- Separate modules that have different design goals.
- Avoid asynchronous design except reset.
- If asynchronous is required, put it in a separate module.
- Put relevant resources to be shared in the same module.
- Eliminate glue logic at the top-level.
- For an SOC, the top level contains only I/O pad rings, clock generator logic. The clock generation circuitry should be isolated from the rest of the design.

Design with Memory

- Using synchronous memory in SOC.
- Keep memory interface pins at the top-level to allow user choice of memory implementation, interface and test.
- For embedded memory, use BIST.

Verifiable Verilog Keywords

always	assign	begin	case
casez	default	else	end
endcase	endfunction	endmodule	function
if	initial	inout	input
module	negedge	or	output
parameter	posedge	reg	wire

Synopsys-Unsupported Verilog Constructs

- **Unsupported definitions / declarations**
  - primitive definition
  - time declaration
  - event declaration
  - triand, trior, tri1
  - tri0, trireg net types
  - Ranges and arrays for integers
- **Unsupported statements**
  - initial statement
  - repeat statement
  - delay control
  - event control
  - wait statement
  - fork statement
  - deassign statement
  - force statement
  - release statement

Reference: HDL Compiler (Presto Verilog) Reference Manual

- **Unsupported operators**

-Case equality

-inequality operators (=== and !==)

- **Unsupported gate-level constructs**
  - nmos, pmos, cmos, rnmos,
  - rpmos, rcmos
  - pullup, pulldown, tranif0,
  - tranif1, rtran, rtrainf0,
  - rtrainf1 gate types
- **Unsupported miscellaneous constructs**
  - hierarchical names within a module

If you use an unsupported construct, Presto Verilog issues a syntax error such as **"event is not supported"**