

EECS 470 Lab 3

SystemVerilog Style Guide

Department of Electrical Engineering and Computer Science
College of Engineering
University of Michigan

Thursday, 25th January 2024

Overview

Administrivia

Verilog Style Guide

Verilog Style Guide

- Brevity

- Indentation and Alignment

- SystemVerilog Features

Finite State Machines

Administrivia

Homework

- ▶ Homework 2 is due Friday, 2nd February at 10PM (via Gradescope)

Projects

- ▶ Project 2 is due Tuesday, 30th January at 11:59PM (via submission script)

Help

We are available to answer questions on anything here. Office hours can be found on the [course web site](#).

Verilog Style

What is good style?

- ▶ Easy to read
- ▶ Easy to understand
- ▶ Easy to maintain
- ▶ High reusability
- ▶ Mostly a matter of personal/team preference, or "First-Come First-Serve"

Verilog Style

Why should I use good style?

- ▶ Easier to debug: save your sanity
- ▶ Important for group work: give your teammate(s) an easier life :)
- ▶ Mandatory in projects 1-3

Verilog Style Rules

Goal: Clarity

- ▶ We'll look at a bad example
- ▶ Then how to fix it
- ▶ Derive a rule

Brevity by Example

Example

```
always_comb
begin
  if (foo[3] == 1'b1)
  begin
    bar[3] = 1'b1;
    bar[2] = 1'b0;
    bar[1] = 1'b1;
    bar[0] = 1'b1;
  end else if (foo[2] == 1'b1)
  begin
    bar[3] = 1'b0;
    bar[2] = 1'b1;
    bar[1] = 1'b0;
    bar[0] = 1'b0;
  end
end
end
```

Brevity by Example

Example

```
always_comb
begin
  if (foo[3] == 1'b1)
  begin
    bar[3] = 1'b1;
    bar[2] = 1'b0;
    bar[1] = 1'b1;
    bar[0] = 1'b1;
  end else if (foo[2] == 1'b1)
  begin
    bar[3] = 1'b0;
    bar[2] = 1'b1;
    bar[1] = 1'b0;
    bar[0] = 1'b0;
  end
end
end
```

VS.

Example Reformatted

```
always_comb
begin
  if      (foo[3]) bar = 4'b1011;
  else if (foo[2]) bar = 4'b0100;
end
```


Brevity by Example

Example

```
logic [5:0] shift;

always_ff @(posedge clock)
begin
    if (reset)
    begin
        shif_reg <= 6'b0;
    end else begin
        shift[0] <= foo;
        shift[1] <= shift[0];
        shift[2] <= shift[1];
        shift[3] <= shift[2];
        shift[4] <= shift[3];
        shift[5] <= shift[4];
    end
end
```

Brevity by Example

Example

```
logic [5:0] shift;

always_ff @(posedge clock)
begin
    if (reset)
    begin
        shif_reg <= 6'b0;
    end else begin
        shift[0] <= foo;
        shift[1] <= shift[0];
        shift[2] <= shift[1];
        shift[3] <= shift[2];
        shift[4] <= shift[3];
        shift[5] <= shift[4];
    end
end
end
```

VS.

Example Reformatted

```
logic [5:0] shift;

always_ff @(posedge clock)
begin
    if (reset)
    begin
        shift <= 6'b0;
    end else begin
        shift <= {shift[4:0], foo};
    end
end
end
```

Brevity Rule

Rule

Brevity is (often) strongly correlated with the optimal solution. Be brief, where you can.

Brevity Rule

Is brevity always better? Not necessarily...You also need to be clear

General Requirements

Clarity Rules

- ▶ Use meaningful names for signal; `wire wire`; is confusing
- ▶ Comment your designs; `(a ^ b ~^ c) | (&d)` is unintelligible without an explanation
- ▶ Conceptualize what you need to build before you start writing Verilog. A state machine diagram will be make the Verilog much easier. . .

General Requirements

Clarity Rules

- ▶ Starter guide on signal namings: <https://github.com/lowRISC/style-guides/blob/master/VerilogCodingStyle.md#naming>

Clarity by Example

Example

```
logic enable;

always_comb
begin
    enable =
        (op[0]^op[1] | op[2] & op[3])?
            1'b1: 0'b0;
end
```

Clarity by Example

Example

```

logic enable;

always_comb
begin
    enable =
        (op[0]^op[1] | op[2] & op[3])?
        1'b1: 0'b0;
end

```

VS.

Example Reformatted

```

logic enable;

always_comb
begin
    if(op[3])
    begin
        if(op[2])
        begin
            enable = 1'b1;
        end else if(op[0] ^ op[1])
        begin
            enable = 1'b1;
        end else begin
            enable = 1'b0;
        end
    end else begin
        enable = 1'b0;
    end
end
end

```


Brevity & Clarity Rules

- ▶ Use commonsense and intuition.
- ▶ Ask your teammate(s) to peer-review.
- ▶ Use the names to describe what is happening
- ▶ For example:
 - ▶ "_d" suffix for combinational wires
 - ▶ "_q" suffix for sequential registers
 - ▶ "_n" for active low signals
 - ▶ Macros or parameters for describing names of states.

Indentation

Why are we interested in indentation?

- ▶ Readability – easier to trace down
- ▶ Clarity – easier to check what is in a given scope

Indentation by Example

Example

```
always_comb
begin
  if(cond)
  begin
    n_state = `IDLE;
    n_gnt = `NONE;
  end else begin
    n_state = `TO_A;
    n_gnt = `GNT_A;
  end
end
```

Indentation by Example

Example

```
always_comb
begin
  if(cond)
  begin
    n_state = `IDLE;
    n_gnt = `NONE;
  end else begin
    n_state = `TO_A;
    n_gnt = `GNT_A;
  end
end
```

VS.

Example Reformatted

```
always_comb
begin
  if (cond)
  begin
    n_state = `IDLE;
    n_gnt = `NONE;
  end else begin
    n_state = `TO_A;
    n_gnt = `GNT_A;
  end
end
```

Indentation Rule

Rule

Items within the same scope should have the same indentation.

Alignment

Why are we interested in alignment?

- ▶ Readability – easier to trace down
- ▶ Clarity – easier to check that everything is assigned

Alignment by Example

Example

```
always_comb
begin
  if (cond)
    begin
      n_state = `IDLE;
      n_gnt = `NONE;
    end else begin
      n_state = `TO_A;
      n_gnt = `GNT_A;
    end
end
end
```

Alignment by Example

Example

```
always_comb
begin
    if (cond)
        begin
            n_state = `IDLE;
            n_gnt = `NONE;
        end else begin
            n_state = `TO_A;
            n_gnt = `GNT_A;
        end
    end
end
```

VS.

Example Reformatted

```
always_comb
begin
    if (cond)
        begin
            n_state = `IDLE;
            n_gnt = `NONE;
        end else begin
            n_state = `TO_A;
            n_gnt = `GNT_A;
        end
    end
end
```


Alignment by Example

Example

```
assign mux_out = (cond1) ? (foo1&bar): (cond2) ? (foo2+cnt3) :  
(cond3) ? (foo3&~bar2) : 0;
```

Alignment by Example

Example

```
assign mux_out = (cond1) ? (foo1&bar): (cond2) ? (foo2+cnt3) :  
(cond3) ? (foo3&~bar2) : 0;
```

Example Reformatted

```
assign mux_out = (cond1) ? (foo1 & bar)  :  
                  (cond2) ? (foo2 + cnt3) :  
                  (cond3) ? (foo3 & ~bar2) : 0;
```

Alignment Rule

Rule

- ▶ Assignments should be aligned by column.
- ▶ Ternary statements should have the conditionals aligned, and each “if” should be on a new line.
- ▶ Try to align bit width declaration and indices when possible
- ▶ Spaces are preferable over tabs for portability across IDEs

SystemVerilog Types

User-defined Types

- ▶ Useful for cleaning repeated declarations, specifically bundling connections
- ▶ Types can be named informatively, e.g. `arch_reg_t`

Structs

About struct

- ▶ A package of signals (wire or logic)
- ▶ Basically follow C conventions
 - ▶ List of signal declarations
 - ▶ Named with `_t` ending

Syntax

- ▶ `struct`
- ▶ List of signals between braces (`{}`)
- ▶ Name after braces, followed by a semicolon (`;`)

Structs

Example

```
typedef struct {  
    logic [7:0] a; //Structs can contain  
    logic      b; //other structs, like  
    arch_reg_t c; //<-- this line  
} example_t; //named with _t
```

Example

```
typedef struct packed {  
    addr_t pc;  
    logic  valid;  
} prf_entry_t;
```

Usage Example

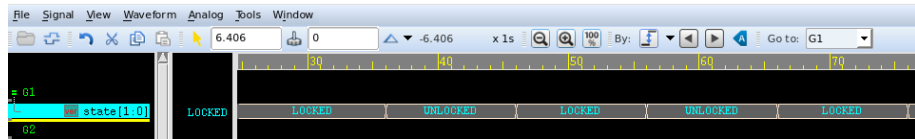
```
prf_entry_t [31:0] prf;  
assign prf[1].valid = 1'b0;
```

Enums

About enum

- ▶ List of possible values, but named instead of numbered
- ▶ Good for state machine states
- ▶ Can be shown in Verdi instead of the associated value

Verdi Example



Syntax

- ▶ enum
- ▶ List of values between braces ({})
- ▶ Name after braces, followed by a semicolon (;)

Enums

Example

```
typedef enum logic [3:0] {  
    IDLE,      // = 0, by default  
    GNT[0:7], // Expands to GNT0=1, ... GNT7=8  
    RESET     // = 9  
} arb_state;
```

Example

```
typedef enum logic [1:0] {  
    ADD    = 2'b00, // The value associated with  
    MULT   = 2'b10, // a particular name can be  
    NOT    = 2'b11, // assigned explicitly.  
    AND    = 2'b01  
} opcode;
```

Usage Example

```
arb_state state, n_state;  
assign n_state = IDLE;
```


Typedef

About typedef

- ▶ Necessary for reuse of a `struct` or `enum`
 - ▶ Without a `typedef`, a `struct`/`enum` must be redefined at each instance declaration
- ▶ Also useful in clearly naming commonly sized buses

Syntax

- ▶ `typedef`
- ▶ by any signal declaration or `struct` or `enum` declaration
- ▶ Name for the type followed by a semicolon (`;`)

Typedef by Example

Example: Typedef'd Enum

```
//typedef, then definition, then name;  
typedef enum logic [3:0] {  
    IDLE,  
    GNT[0:7],  
    RESET  
} arb_state;
```

Example: Type Synonym

```
//typedef, then definition, then name;  
typedef logic [63:0] addr;
```

Procedural FSM Design

FSM Process

- ▶ All states should be `typedef enum`
- ▶ All next state logic should go into a combinational block, following all combinational rules
- ▶ All resets should be synchronous (to the clock)
- ▶ All output assignments should go in their own combinational block
- ▶ The only logic in the sequential block should be the state assignment (to the computed next state)

Finite State Machine Skeleton

```
typedef enum logic [(NUM_STATES-1):0] { STATES } fsm_state;

module fsm(
    input wire inputs,
    output logic outputs
);

    fsm_state state, next_state;

    always_comb begin
        /* Transitions from a diagram go here */
        /* next_state = f(inputs, state) */
    end

    always_ff @(posedge clock) begin
        if(reset) begin
            state <= DEFAULT;
        end else begin
            state <= next_state;
        end
    end
end
endmodule
```

Finite State Machine Example

```
typedef enum logic { LOCKED, UNLOCKED } ts_state;

module turnstile(
    input wire coin, push,
    input wire clock, reset,
    output ts_state state
);

    ts_state next_state;

    always_comb begin
        next_state = state;
        if (state==LOCKED && coin) next_state = UNLOCKED;
        if (state==UNLOCKED && push) next_state = LOCKED;
    end
    always_ff @(posedge clock) begin
        if (reset) state <= LOCKED;
        else      state <= next_state;
    end
endmodule
```

Coverage

- ▶ So last time we talked about why to write a testbench
- ▶ We also talked about how to write a good testbench
- ▶ All metrics that we talked about are rather qualitative, though
- ▶ How do we quantitatively measure how good a testbench is? Code coverage!

Coverage Cont.

Code Coverage Types

- ▶ **Line Coverage** — measures statements in your HDL code that have been executed in the simulation
- ▶ **Toggle Coverage** — measures the bits of logic that have toggled during simulation. A toggle simply means that a bit changes from 0 to 1 or from 1 to 0.
 - ▶ One of the oldest metrics of coverage in hardware designs and can be used at both the register transfer level (RTL) and gate level.
 - ▶ May give low numbers if you have wide signals (ex: adding 64 bit numbers)
- ▶ **Condition Coverage** — measures how the variables or sub-expressions in the conditional statements are evaluated during simulation. It can find errors in the conditional statements that cannot be found by other coverage analysis.

Coverage Cont.

Code Coverage Types Cont.

- ▶ **Branch Coverage** — measures the coverage of expressions and case statements that affect the control flow (such as if-statement and while-statement) of the HDL. It focuses on the decision points that affect the control flow of the HDL execution.
- ▶ **FSM Coverage** — verifies that every legal state of the state machine has been visited and that every transition between states has been covered.
- ▶ For more information about coverage, feel free to ask us or search online!

Coverage Cont.

Coverage compile time options

- ▶ `-cm line|cond|fsm|tgl|branch|assert`
- ▶ The arguments specifies the types of coverage:
 - ▶ `line` - Compile for line or statement coverage.
 - ▶ `cond` - Compile for condition coverage.
 - ▶ `fsm` - Compile for FSM coverage.
 - ▶ `tgl` - Compile for toggle coverage.
 - ▶ `branch` - Compile for branch coverage
 - ▶ `assert` - Compile for SystemVerilog assertion coverage.
- ▶ If you want VCS to compile for more than one type of coverage, use the plus (+) character as a delimiter between arguments, for example:

- ▶ `vcs -sverilog -xprop=tmerge +vc -Mupdate -line -full64 -kdb -lca -nc -debug_access -cm fsm+tgl mydesign.sv mytest.sv -o simv`
- ▶ `./simv -cm fsm+tgl`

Coverage example

Run `make verdi_cov` in the lab 3 scripts to open Verdi in coverage mode.

Verdi Coverage

The screenshot displays the Verdi Coverage tool interface, which is divided into three main panes:

- Summary:** A table showing coverage statistics for different components.

Line	Toggle	FSM	Condition
82.50%	100.00%	50.00%	62.50%
91.39%	100.00%	94.44%	62.50%
- CovSrc1: test.bench.tbp:** A window showing the source code for the `two_bit_pred` module. The code includes input signals (`clock`, `reset`, `taken`, `transit`), an output (`prediction`), and logic for state transitions.


```

module two_bit_pred(
    input clock, reset, taken, transit,
    output prediction);

    logic [1:0] state;
    logic [1:0] next_state;

    assign prediction = state[1];

    always_comb begin
        case(state)
            2'b01, 2'b10 : next_state = taken
            ? 2'b11 : 2'b00;
            2'b00 : next_state = taken ? 2'b01
            : 2'b00;
        endcase
    end
endmodule
      
```
- CovDetail:** A table showing detailed coverage for variables.

Variable	Type	Coverage
clock	port	100.00%
next_state[1:0]	signal	100.00%
prediction	port	100.00%
reset	port	50.00%
state[1:0]	signal	100.00%

Coverage example Cont.

Verdi Coverage

The screenshot displays the Verdi Coverage tool interface. The left pane shows the source code for 'testbench.tbp' with coverage annotations. The right pane shows the 'CovDetail' window with a table of transition coverage.

Source Code (CovSrc.1: testbench.tbp):

```

4
5
6  logic [1:0] state;
7  logic [1:0] next_state;
8
9  assign prediction = state[1];
10
11  always_comb begin
12    case(state)
13      2'b01, 2'b10 : next_state = taken ? 2'b11 : 2'b00;
14      2'b00 : next_state = t
15      2'b11 : next_state = t
16    endcase
17  end
18
19  always_ff @(posedge clock)
20    if(reset)

```

CovDetail Window:

Line Toggle FSM Condition Branch Assert

Line	Toggle	FSM	Condition	Branch	Assert
*					
state		62.50%	100.00%		

Mode: States & Transitions

	0	1	2	3
0 'h0	-	✓	-	-
1 'h1	✓	✓	-	✓
2 'h2	✓	✗	-	✗
3 'h3	-	✗	✓	-

Transition 'h1->'h0 Covered
 Transition 'h1->'h3 Covered
 Transition 'h2->'h0 Covered
 Transition 'h2->'h3 Uncovered
 Press 'F3' to save in Tooltip Viewer

Coverage Cont.

We will come back to this later in the final project. As part of milestone 1, you will need to submit a module along with a testbench and we will grade you based on your coverage percentage, but that'll be much later. In the meanwhile... Give it a try on your project 2 testbench!

Lab Assignment

- ▶ Assignment is posted to the [course website](#) as Lab 3 Assignment.
- ▶ If you get stuck. . .
 - ▶ Ask a neighbor, quietly
 - ▶ Put yourself in the [help queue](#)
- ▶ When you are finished, put yourself on help queue to get checked off.
- ▶ If you are unable to finish today, the assignment needs to be checked off by next Friday.