Coding Guidelines

Revision 2: Changed indentation example and wording to emphasize that only **`always_ff`** *is the only* **`always`** *allowed.*

Mandatory Guidelines

It's not enough that your design work. Marks will be deducted if you do not follow the guidelines in this section.

File-level Comments

Include, near the beginning of each Verilog file, comments showing: the file name, a line describing the purpose of the file, the author's name, and the date[1].

Why? These help to quickly identify the source and purpose of your code.

Example:

```
// lab1.sv
// Display digits of ID on 7-segment display.
// Jane Doe, 2020-9-15
```

Synchronous Design

Use a single clock. The same clock signal must appear in every **`always_ff @(posedge`** *<clock>***`)`** expression. You can verify this by checking that this is the only signal that appears as a Clock in **Processing > Compilation Report > Fitter > Resource Section > Control Signals.**

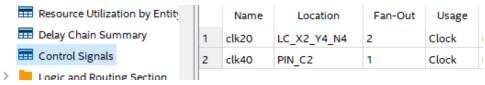
Why? Programmable hardware and design tools assume synchronous (one clock) design. "Computed" clocks, such as in the ripple counter below, are inefficient and difficult to verify.

Example:

```
// NOT allowed:
  always_ff @(posedge clk40) clk20 <= ~clk20 ;
  always_ff @(posedge clk20) clk10 <= ~clk10 ;
```

[1]Don't confuse this with the information on your report's cover page.

Example: A report showing multiple clocks:



logic Type

Use the **`logic`** type for synthesis.

Why? System Verilog's **`logic`** can replace both **`wire`** and **`reg`**. Use it in new code.

Example:

```
// NOT allowed:
  wire clk ;
  reg [15:0] cnt ;

// OK:
  logic clk ;
  logic [15:0] cnt ;
```

Consistent Indentation

A line with **`end`** should be indented the same as the line with the corresponding **`begin`**. Each level of indentation should increase by the same amount (no less than 3 and no more than 8). Avoid use of tab characters.

Why? This makes it much easier to find errors.

Example:

```
  // NOT allowed:
  assign cnt_next =
  cnt == 0 ? 25 :
  cnt - 1 ;

 always_ff @(posedge clk)
      cnt <= cnt_next ;

  // OK:
  assign cnt_next =
     cnt == 0 ? 25 :
     cnt - 1 ;

  always_ff @(posedge clk)
     cnt <= cnt_next ;
```

Use always_ff

Use **always_ff**, not **always**, **always_comb** or **always_latch**.

Why? To avoid unintended latches.

Example:

```
// NOT allowed:
always @(posedge clk)
  cnt <= cnt_next ;

// OK:
always_ff @(posedge clk)
  cnt <= cnt_next ;
```

Single Assignment in always_ff

An **always_ff** block may only contain one assignment.

Don't follow examples that use sequential statements such as **if** or **case** within the **always_ff**.

Why? This ensures each **always_ff** corresponds to one register.

Example:

```
// NOT allowed:
always_ff @(posedge clk) begin
   if ( cnt[15] )
      cnt <= cnt + 1'b1 ;
end

// OK:
always_ff @(posedge clk) cnt <=
   cnt[15] ? cnt + 1'b1 : cnt ;
```

Recommended Guidelines

The following guidelines will make your code easier to understand and help you avoid mistakes. However, their use won't be marked.

Add comments next to port and signal declarations and for non-obvious parts of your design. These should explain why you're doing something rather than repeating what is obvious from the code. It should be possible to figure out how your design works just by reading the comments.

```
// AVOID redundant or missing comments:
module ...
  (
  input logic reset_n, clk, // active-low reset, clk
  input logic [15:0] a, b, c,
  ...
  // set usea to 1 if a>b and a<c or a>=c and a <=b
  assign usea = a > b && a < c || a >= c && a <= b ;

// DO describe non-obvious signals and code
module ...
  (
  input logic reset_n, clk,
  input logic [15:0] a, b, c, // filter inputs
  ...
  // is a the median value?
  assign usea = a > b && a < c || a >= c && a <= b ;
```

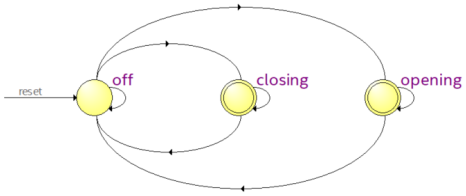
Use consistent signal naming to avoid confusion. The following are widely-recognized conventions:

- append **_n** to active-low signals
- append **_t** to type names
- append **_in** to names of input ports that have a corresponding internal signal such as a synchronised or debounced version
- append **_next** to the name of a register output to derive the name of the register input

Use enumerated types for states to make your code easier to read and to help the synthesizer optimize the design.

The garage door controller below is an example of a state machine. The synthesizer will choose an appropriate representation for the state variables.

```
module controller
  (
    input logic clk, pb_in,  // clock, pushbutton
    input logic top, bottom, // position sensors
    output logic up, down    // motor control
    ) ;

   // declare and use an enum state type
   typedef enum int unsigned
                { off, opening, closing } state_t ;
   state_t state = off, state_next ;

   logic pb ;
   debounce db0 ( pb_in, clk, pb ) ;

   always @(posedge clk) state <=
       state == off && pb ?
         ( top ? closing : opening ) :
       state == opening && top ? off :
       state == closing && bottom ? off :
       state ;

   assign up = state == opening ;
   assign down = state == closing ;

endmodule
```



AI Tools

Generative AI software such as ChatGPT can indent your code, find syntax errors, add comments, and help explain how your design works – or why it doesn't.

However, do not rely on these program to write your code because (1) you will need to write your own code on the quizzes, exams and some labs; and (2) these programs are unlikely to generate code that complies with the course coding guidelines without significant prompting.

As always, credit any work that is not your own. You must cite the source if you submit code generated by AI software.