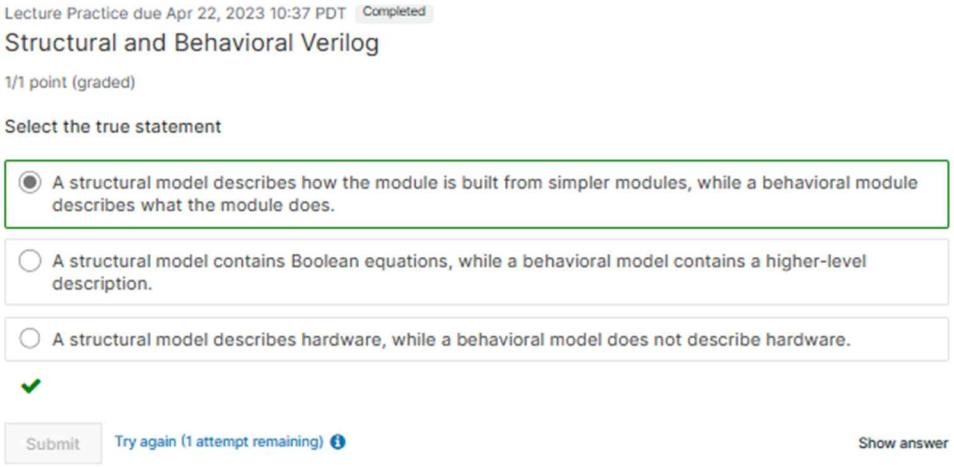
❖ Chapter 4: Verilog / Introduction



Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Verilog Logic Functions

1/1 point (graded)

Keyboard Help

Drag each Verilog symbol to the corresponding logic function.

Reset

FEEDBACK

Good work! You know the symbols for logic functions in Verilog.

For the following problems, consider signals declared below:

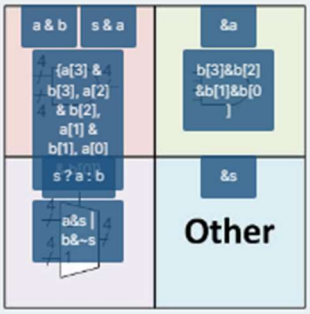
```
logic [3:0] a, b;

logic       s;
```

Schematics from Verilog

1/1 point (graded)

Keyboard Help

Drag each Verilog expression to the corresponding schematics.



Reset

Bit Swizzling

1/1 point (graded)

How many bits long is the expression {a[3:2], {4{s}}, b, 3'b010, s}?

14 ✔

14

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Benefits of Delays

1/1 point (graded)

Delays have the following purpose in HDLs (check all that apply)

- [ ] They imply logic must have the specified propagation delay.
- [ ] They imply logic must have the specified contamination delay.
- [x] They help space events in time so it is easier to understand causality from simulation waveforms.
- [x] They are used in test benches to specify the relative order of inputs changing.
- [ ] They are purely cosmetic and have no practical function.

✔

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Flip-flop idiom

1/1 point (graded)

The preferred SystemVerilog idiom for a flip-flop is

- ○ always @(clk) q <= d;
- ○ always @(posedge clk) q <= d;
- ◉ always_ff @(posedge clk) q <= d;
- ○ always_ff @(posedge clk) q = d;
- ○ always_latch @(clk) q gets d;

✔

Submit Try again (1 attempt remaining) Show answer

The proper way to design digital systems with Verilog is to think of the hardware that you want, and then to write the appropriate idiom for that hardware. The lecture slides and textbook give the idioms for essentially all of the hardware that I use in my professional life as an accomplished chip designer. If you find yourself looking for a component that is not in the book, you are probably not thinking about designing with hardware.

Many programmers become accustomed to learning language features by looking them up in a search engine. While this is usually effective for typical programming languages, it is dangerous with Verilog because the Internet is full of an astonishing amount of bad Verilog code. The following examples are taken from the top page of hits in a Google search for "verilog flip-flop" in September 2020. Nearly every search I have made for Verilog examples has produced a substantial fraction of hits with defective or poor code.

Although SystemVerilog became a standard in 2009 and introduce the "`logic`" data type, all of these examples use the old-style "`reg`" data type instead. A "`reg`" in Verilog is simply a variable that appears on the left hand side of <= or = in an "`always`" block. It might indicate a register, but also might indicate combinational logic, and is unnecessarily confusing. SystemVerilog introduced "`logic`" that works with both `assign` and `always` statements, and is the preferred type, as taught in this class.

Many of the examples also use old-style "`always @(posedge clk)`" instead of the preferred SystemVerilog "`always_ff @(posedge clk)`", missing out on a safety check that you didn't accidentally imply something other than a flip-flop.

Consider the following Verilog code purporting to be a flip-flop:

https://technobyte.org/verilog-code-d-flip-flop-dataflow-gate-behavioral/

```
module dff_behavioral(d,clk,clear,q,qbar);
input d, clk, clear;
output reg q, qbar;
always@(posedge clk)
begin
if(clear== 1)
q <= 0;
qbar <= 1;
else
q <= d;
qbar = !d;
end
endmodule
```

Bad Flop 1 Failings

1/1 point (graded)

What is bad about the flip-flop above: (check all that apply)

- [x] This code actually implies two flip-flops, not one, and takes twice as much hardware as necessary. Your chip will be twice as expensive to build as that of your competitors.
- [x] The lack of indentation makes it unnecessarily hard to read the code.
- [x] The code could have just written if (clear) instead of if (clear==1)
- [x] If the flip-flop were expanded to more than 1 bit, qbar = !d would not produce the complement of q because ! is a logical rather than bitwise NOT. It treats a multiple-bit signal as either false (all 0s) or true (not all zeros). It produces an output that is either true (00...001) or false (00...000), rather than the bitwise complement of the input.

✔

Submit Try again (1 attempt remaining) Show answer

Consider the following Verilog code purporting to be a flip-flop:

https://technobyte.org/verilog-code-d-flip-flop-dataflow-gate-behavioral/

```
module d_ff_struct(q,qbar,d,clk);
input d,clk;
output q, qbar;
not_gate not1(dbar,d);
nand_gate nand1(x,clk,d);
nand_gate nand2(y,clk,dbar);
nand_gate nand3(q,qbar,y);
nand_gate nand4(qbar,q,x);
endmodule
```

Bad Flop 2 Failings

1/1 point (graded)

What is bad about the flip-flop above: (check all that apply)

- [x] This code actually a latch, not a flip-flop.
- [x] Because this code is structural, a logic synthesis tool can't map it onto a library of latches or flip-flops and instead will produce random gates. A timing analyzer won't recognize the random gates as a synchronous sequential circuit and won't properly perform setup or hold time analysis and hence the chip may not operate as intended.
- [x] The lack of indentation makes it unnecessarily hard to read the code.
- [x] The code gradutiously uses new cells called "not_gate" and "nand_gate" rather than the built-in "not" and "nand" elements in Verilog.
- [ ] Your mother was a hamster and your father smelt of elderberries.

✔

Submit Try again (1 attempt remaining) Show answer

Consider the following Verilog code purporting to be a flip-flop:

https://technobyte.org/verilog-code-d-flip-flop-dataflow-gate-behavioral/

```
module RisingEdge_DFlipFlop(D,clk,Q);
input D; // Data input
input clk; // clock input
output Q; // output Q
always @(posedge clk)
begin
 Q <= D;
end
endmodule
```

Bad Flop 3 Failings

1/1 point (graded)

What is bad about the flip-flop above: (check all that apply)

- [x] Q appears on the left hand side of an assignment in an always block, but is not declared as either reg or logic, so this code will not compile.
- [x] The module name is too long and impractical to repeatedly type.

✔

Submit | Try again (1 attempt remaining) | Show answer

Consider the following Verilog code purporting to be a flip-flop:

http://ftp.smart-dv.com/examples/verilog/d_ff.html

```
//-----------------------------------------------------
// Design Name : dff_async_reset
// File Name   : dff_async_reset.v
// Function    : D flip-flop async reset
// Coder       : Deepak Kumar Tala
//-----------------------------------------------------
module dff_async_reset (
data  , // Data Input
clk   , // Clock Input
reset , // Reset Input
q       // Q output
);
//----------Input Ports--------------
input data, clk, reset ;

//----------Output Ports--------------
output q;

//------------Internal Variables--------
reg q;

//-------------Code Starts Here---------
always @ ( posedge clk or negedge reset)
if (~reset) begin
  q <= 1'b0;
end  else begin
  q <= data;
end

endmodule  //End Of Module dff_async_reset
```

Bad Flop 4 Failings

1/1 point (graded)

What is bad about the flip-flop above: (check all that apply)

- [x] This flop has an asynchronous active-low reset. Although active-low resets were common as inputs to small chips such as the 74LS74, they are rarely used inside modern chips. Furthermore, the signal should have been named resetb or resetn so the user wouldn't be caught by surprise that the reset was nonstandard.
- [x] The module is 30 lines long, as compared to 10 or less in the preferred idiom in the textbook. Bloated code is harder to read and obscures the big picture.

✔

Submit Try again (1 attempt remaining) Show answer

Consider the following Verilog code purporting to be a flip-flop:

https://www.chipverify.com/verilog/verilog-tff

```
module tff (  input clk,
              input rstn,
              input t,
            output reg q);

  always @ (posedge clk) begin
    if (!rstn)
      q <= 0;
    else
       if (t)
            q <= ~q;
       else
            q <= q;
  end
endmodule
```

Bad Flop 5 Failings

1/1 point (graded)

What is bad about the flip-flop above: (check all that apply)

- [x] Although you were probably Googling for a D flip-flop, this is actually a T-flip-flop that toggles the opposite value on a clock edge when t is asserted. You've probably never heard of a T flip-flop because they aren't in the textbook. They aren't in the textbook because they are almost never used in industry. When you need the output to toggle, you can do it with an inverter.
- [x] The second else statement is unnecessary.
- [x] Beware of another active low reset (but at least this time the name resetn warns you of the polarity).

✔

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

always_comb

1/1 point (graded)

Use always_comb in SystemVerilog when you wish to (check all that apply)

- [x] Describe logic as a truth table using case or casez
- [x] Describe logic to choose an output based on an input condition using if/else
- [ ] Describe logic with Boolean equations using assign
- [ ] Describe flip-flops and latches with reset or enable

✔

Submit Try again (1 attempt remaining) Show answer

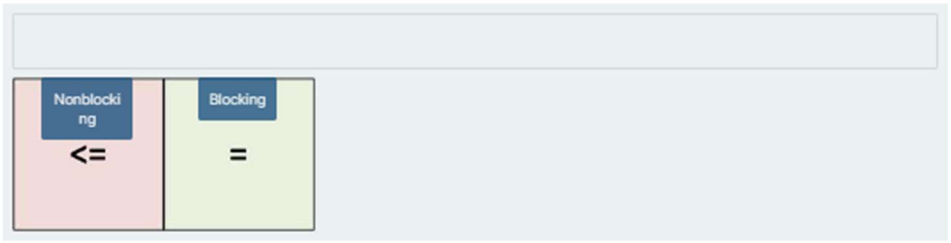
Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Blocking and Nonblocking Assignments

1/1 point (graded)

Keyboard Help

Drag the type of assignment to its corresponding symbol



Reset

FEEDBACK

Good work! You have shown the symbols for blocking and nonblocking assignments.

Using Blocking and Nonblocking Assignments

1/1 point (graded)

Check all that are true:

- [x] Use nonblocking assignments when defining sequential elements because these elements all operate concurrently and could simulate incorrectly if modeled with blocking assignments.
- [x] Use assign statements to concisely define simple combinational logic with Boolean equations.
- [x] Use blocking assignments in complicated combinational logic for the sake of simulation efficiency.
- [x] Each output or internal node should be given a value in exactly one always or assign statement.

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Consider the following FSM:

```
module l2p(input  logic clk, reset, L,
           output logic P);
  typedef enum logic [1:0] {S00, S01, S10} statetype;
  statetype state, nextstate;
  always_ff@(posedge clk, posedge reset)
    if (reset) state <= S00;
    else       state <= nextstate;
  always_comb
    case (state)
      S00: if (L) nextstate = S01;
           else   nextstate = S00;
      S01: if (L) nextstate = S10;
           else   nextstate = S00;
      S10: if (L) nextstate = S10;
           else   nextstate = S00;
   endcase
  assign P = (state == S01);
endmodule
```

State machine type

1/1 point (graded)

The FSM above is a:

- (●) Moore machine because the output P is just a function of the current state
- ( ) Mealy machine because the output P depends on the current state and the inputs.
- ( ) None of the above

✔

Submit Try again (1 attempt remaining) Show answer

SystemVerilog FSM Coding

1/1 point (graded)

The FSM above (check all that apply)

- [ ] Is poor coding style because nonblocking assignments should be used everywhere
- [x] Has an asynchronously resettable 2-bit state register
- [x] Implies an undesirable latch because the default case was omitted
- [ ] Has a total of four bits of state

✔

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Consider the following SystemVerilog code:

```
module counter
       #(parameter N=16)
       (input  logic         clk, reset,
        output logic [N-1:0] q);

  flop #(N) f(clk, reset, q+1, q);
endmodule
module flop
       #(parameter N=4)
       (input  logic         clk, reset,
        input  logic [N-1:0] d,
        output logic [N-1:0] q);

  always @(posedge clk, posedge reset)
    if (reset) q <= 0;
    else       q <= d;
endmodule
```

Parameterized Module

1/1 point (graded)

How many bits of state does the counter use?

- ○ 0
- ○ 4
- ◉ 16

✔

Submit Try again (1 attempt remaining) Show answer

Lecture Practice due Apr 22, 2023 10:37 PDT Completed

Testbench Qualities

1/1 point (graded)

A good testbench (choose all that apply)

- ☑ Applies test vectors to the inputs of a DUT
- ☑ Reports whether the outputs of the DUT match expectations
- ☑ Guides you to find any failing outputs
- ☑ Works automatically in a way that is easy to use, even for somebody unfamiliar with the requirements or inner workings of the DUT
- ☑ Includes a comprehensive set of test vectors that are likely to uncover any error in the DUT, yet still be able to run in a reasonable amount of time.

✔

Submit Try again (1 attempt remaining) Show answer