CAL POLY
SAN LUIS OBISPO

EE 531: ADVANCED VLSI DESIGN

Introduction to Verilog

Nishith N. Chakraborty

January, 2025

OUTLINE

- HDL versus schematic entry
- Basic structure of a Verilog module
- Common syntax of Verilog
- Continuous versus Procedural Assignments
- The Verilog always block Sensitivity lists
- Nets versus variables

SCHEMATIC VERSUS HDL

Schematic	HDL
✓ Good for multiple data flow	✓ Flexible and parameterizable
✓ Provides overview picture	✓ Excellent for optimization
✓ Relates directly to hardware	✓ Excellent for top-down synthesis
✓ No need for programming skills	✓ Direct mapping to algorithms
✓ High information density	✓ Excellent for datapaths
✓ Easy back annotations	✓ Easy to handle electronically
✓ Useful for mixed analog/digital	
✗ Not good for algorithms	✗ Serial representation
✗ Not good for datapaths	✗ May not show overall picture
✗ Poor interface for optimization	✗ Need good programming skills
✗ Difficult to reuse Difficult to parameterize	✗ Divorsed from physical hardware

VERILOG BASICS

- Similar to C language but for describing hardware
- Description can be at different levels:
  - Behavioral
  - Structural or gate level
- Not just a specification language – also associated with simulation environment
- Considered easier and “lighter weight” compared to VHDL
- Very popular in industry

STRUCTURE OF A MODULE

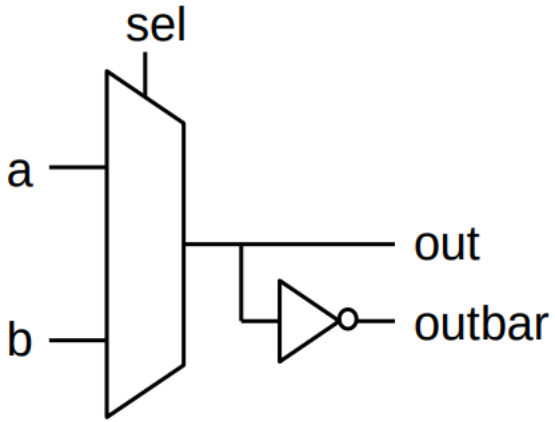
- Verilog design contains interconnected modules
- A module has collections of low-level gates, statements and instances of other modules
- Example:

```
// Function: 2-to-1 multiplexer
module mux2to1 (out, outbar, a, b, sel);

  output out, outbar;
  input  a, b, sel;

  assign out = sel ? a : b;
  assign outbar = ~out;

endmodule;
```



STRUCTURE OF A MODULE

- Verilog design contains interconnected modules
- A module has collections of low-level gates, statements and instances of other modules
- Example:

```
// Function: 2-to-1 multiplexer
module mux2to1 (out, outbar, a, b, sel);

  output out, outbar;
  input  a, b, sel;

  assign out = sel ? a : b;
  assign outbar = ~out;

endmodule;
```

- Use // to comment; can also use /* … */
- Declare and name module; list the ports; terminate with ';'
- Specify port type as input, output or inout if bidirectional
- Express module behavior
- Each statement executes concurrently
- Order does not matter!

CONTINUOUS ASSIGNMENT

- Keyword assign specifies continuous assignment to describe combinational logic
- Right-hand continuously evaluated, immediate response to inputs
- Left-hand is a *net* driven with evaluated value
- Operators are low-level:
  - Conditional assignment: (condition) ? vTrue : vFalse
  - Boolean: ~, &, |
  - Arithmetic: +, -, *
- Operators can be nested:

```
assign out = s1 ? (s0 ? i3 : i2) : (s0 ? i1 : i0);
```

VERILOG OPERATORS

{ }	concatenation
+ - * /	
	arithmetic
%	modulus
> >= < <=	
	relational
!	logical NOT
&&	logical AND
||	logical OR
==	logical equality
!=	logical inequality
? :	conditional

~	bit-wise NOT
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR
^~ ~^	bit-wise XNOR
&	reduction AND
|	reduction OR
~&	reduction NAND
~|	reduction NOR
^	reduction XOR
~^ ^~	reduction XNOR
<<	shift left
>>	shift right

GATE LEVEL DESCRIPTION

- Built-in logic primitives:
  - and, nand, or, nor, xor, xnor, not, buf
- Tri-state buffers: bufif1 and bufif0
- Declare nets (signals, wires) using wire keyword
- An instantiation of an AND gate:

```
and a1 (out1, a, sel);
```

```
// Function: 2-to-1 multiplexer
module mux_gate (out, outbar,
                 a, b, sel);

  output out, outbar;
  input  a, b, sel;
  wire   out1, out2, selb;

  not i1 (selb, sel);
  and a1 (out1, a, sel);
  and a2 (out2, b, selb);
  or  o1 (out, out1, out2);
  not i2 (outbar, out);

endmodule;
```

PROCEDURAL ASSIGNMENT & "ALWAYS"

- Keywords *always* and *initial* are used to define procedural assignments, similar to procedures in software
- Good for behavioral descriptions of hardware
- Support for C-like constructs such as if, for, while, case, etc.

```
module mux_2_to_1 (out, outbar, a, b, sel);
  output out, outbar;
  input  a, b, sel;

  reg    out, outbar;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;

    outbar = ~out;
  end
```

- Input and output declarations as before
- Assignment in an always block must be declared as variable type such as reg
- always runs whenever a signal in the sensitivity list changes values
- Statements inside the always block are executed sequentially
- Sandwiched between begin & end

VERILOG "REGISTER" – NOT WHAT IT APPEARS

- Registers normally represent storage elements in digital logic – they need clock signals to update their output value
- In Verilog, reg artifacts are not same as physical registers, used only to declare a variable for holding the value
- Values of variables (declared as reg) can be changed anytime in simulation, and can be used for nets of combinational circuit

MIXING PROCEDURAL & CONTINUOUS

- Procedural and continuous assignments can co-exist
- In procedural assignments, variables declared as reg changed only when procedural block is invoked via sensitivity list
- In continuous assignments, right-hand expression constantly evaluated and the left-side net updated instantaneously

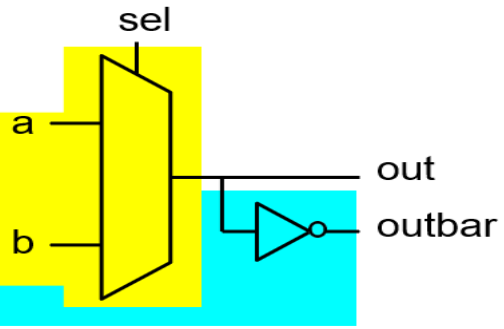
```
module mux_2_to_1 (out, outbar, a, b,
sel);
  output out, outbar;
  input  a, b, sel;
  reg    out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

  assign outbar = ~out;
endmodule;
```

procedural

continuous



THE "CASE" STATEMENT

- The case statement can often replace if-else constructs inside an always block, and provides better abstraction
- An example:

```
always @ (a or b or sel)
begin

  case (sel)
    1'b0: out = b;
    1'b1: out = a;
  endcase

end
```

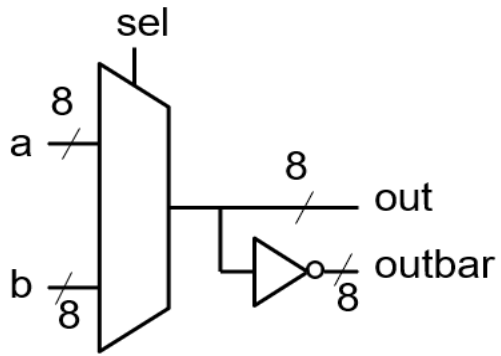
```
if (sel) out = a;
else out = b;
```

Notation for numbers:
<size>'<base><number>

2'b10	2-bit binary, value=2
'b10	Unsigned binary 32-bit, value=2
31	Unsigned decimal, value=31
8'hAf	8-bit hex, value=175
-16'd47	16-bit negative decimal, value=-47

BUSES – N-BIT SIGNALS

- Verilog is powerful in specifying buses
- Example: 8-bit wide 2-to-1 MUX




```
module mux_2_to_1 (out, outbar, a, b,
sel);
  input[7:0]   a, b,
  input        sel;
  output[7:0]  out, outbar;
  reg[7:0]     out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

  assign outbar = ~out;
endmodule;
```

Concatenate signals using the { } operator

```
assign {b[7:0], b[15,8]} = {a[15:8], a[7:0]};
// effects a byte swap
```

SYNTHESIZED NETLISTS: USUALLY VERILOG

```
// Generated by Cadence Encounter(R) RTL Compiler v07.10-s021_1

module reg4(d0, d1, d2, d3, en, clk, q0, q1, q2, q3);
  input d0, d1, d2, d3, en, clk;
  output q0, q1, q2, q3;
  wire d0, d1, d2, d3, en, clk;
  wire q0, q1, q2, q3;
  wire n_0;
  LATCH stored_d0_reg(.CLK (n_0), .D (d0), .Q (q0));
  LATCH stored_d1_reg(.CLK (n_0), .D (d1), .Q (q1));
  LATCH stored_d3_reg(.CLK (n_0), .D (d3), .Q (q3));
  LATCH stored_d2_reg(.CLK (n_0), .D (d2), .Q (q2));
  AND2X1 g21(.A (en), .B (clk), .Y (n_0));
endmodule
```

SUMMARY

- Verilog basics for digital system design
- Use either Verilog or SystemVerilog for class projects

- Challenge: Wasn't this a hardware design class?
  - YES! … but programming skills are critical to simplifying the design of very/ultra large scale integrated systems

Thank you!