Essential "RTL" Verilog :
an excerpt from 18-643 Lecture 7

James C. Hoe
Department of ECE
Carnegie Mellon University

Verilog is not RTL

- Verilog in essence
  - a multithreaded programming language +
  - modeled time +
  - scheduling queue +
  - modules and ports
- Verilog describes how a module behaves, not its construction
  - no notion of “combinational” vs “sequential” logic
  - no notion of a register or even of a clock
  - perfectly happy describing non-hardware

Verilog Synthesis is Interpretation

```
// Ex 1
always@(a or b)
  if (a)
    c = b;
```

```
// Ex 2
always@(a)
  if (a)
    c = b;
  else
    c = 0;
```

```
// Ex 3
always@(a or b)
  if (a)
    c = b;
  else
    c = 0;
```

```
// Ex 4
always@(a or b)
begin
  c = 0;
  if (a)
    c = b;
end
```

```
// Ex 5
always@(a)
  if (a)
    c = b;
  else
    c = 0;

always@(b)
  if (a)
    c = b;
  else
    c = 0;
```

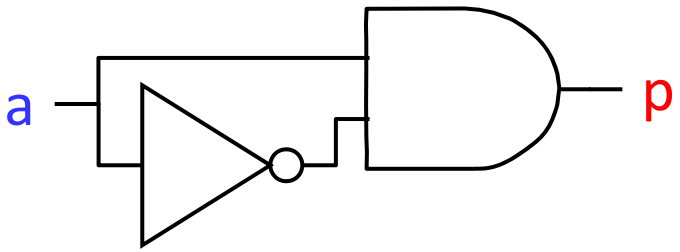
- All have well-defined behaviors
- According to Verilog semantics, c depends combinationally on a and b in Ex 3, 4 and 5
- Verilog doesn't say they are "combinational" or they are synthesizable

Synthesizable Verilog

- Verilog becomes an RTL language and becomes synthesizable only when used in a stylized way dictated by the synthesis tool
- So called "synthesizable subset" is really a different language
- Difficult even to define what is "correct" synthesis with respect to simulation behavior

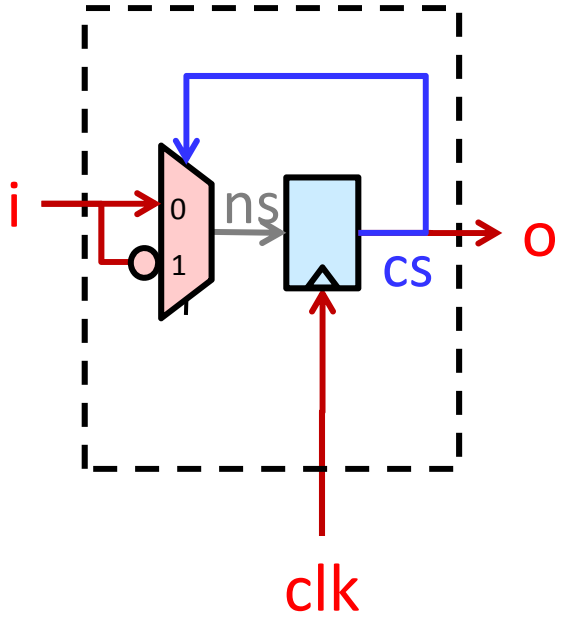
```
always@(a)
   p = a & (!a);
```

=?



Is p combinational?

RTL by Discipline



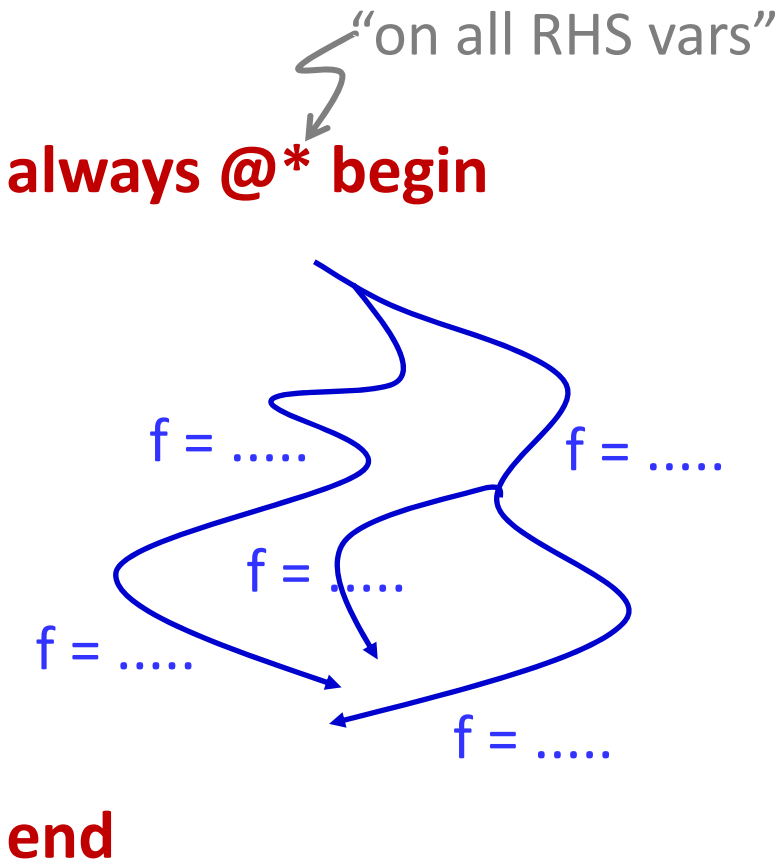

```
module contrived(input i, clk,
                 output o);
  reg cs; // sequential
  reg ns; // combinational

  assign o = cs;

  always @ ( i or cs)
      if (cs) ns = ~i;
      else    ns = i;

  always @ ( posedge clk ) begin
          cs <= ns;
  end

endmodule
```

combinational

synch. sequential

Crib sheet: Combinational "always"

- f must be assigned in all possible control paths; use "blocking" assigns
- f can depend on f only if f has been assigned
- repeated assigns to f okay; the last one holds
- f cannot be assigned in any other process
- multiple LHS vars okay; all rules above apply

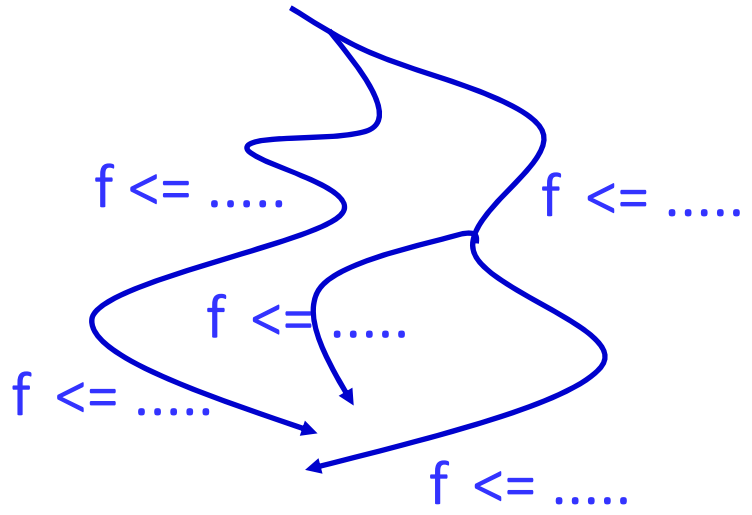


Use continuous assigns for simple expressions

Crib sheet: Synchronous "always"

- use "non-blocking" assigns; effect of assign not visible until after all triggered processes are done
- f can depend on f; RHS f stays at starting value
- repeated assigns to f okay; the last one holds
- f cannot be assigned in any other process
- multiple LHS vars okay; all rules above apply

always @(posedge clk) begin



end

Helpful Resources

- Visit HDLBits at https://hdlbits.01xz.net to self-teach or review RTL Verilog fundamentals