Sequential Logic

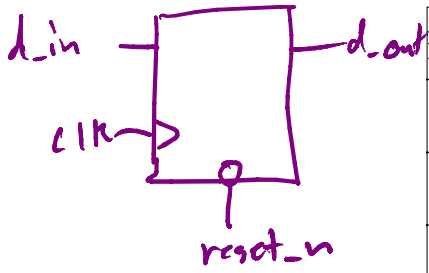
- Outputs are a reflection of past state values
  - Flip-flops and latches hold state
- Only certain input changes cause an change in state
  - Sensitivity only to clock and reset

```
module reg8 (
  input            clk,
  input            reset_n,
  input      [7:0] d_in,
  output reg [7:0] q_out,
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n) q_out <= '0;
   else         q_out <= d_in;
endmodule
```

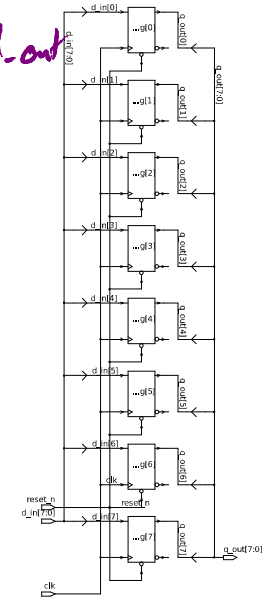
SystemVerilog

- `always_ff` gives clear intent that flip-flops are to be inferred
- `always_ff` must still include sensitivity list for clock and reset
- Nonblocking assignment is used under `always` for sequential logic

Sequential Logic



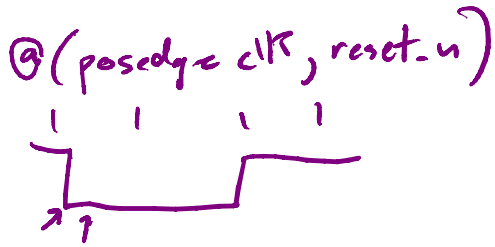

```
module reg8 (
  input            clk,
  input            reset_n,
  input      [7:0] d_in,
  output reg [7:0] q_out
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n) q_out <= '0;
   else         q_out <= d_in;
endmodule
```



Sequential Logic

- An asynchronous input such as reset_n changes the state of the flip-flops independent of the sequential trigger.
- These flip-flops have a level-sensitive, asynchronous reset.
- The reset signal is made dominant by using the `if`.
- Clock input is not tested by an `if`, async inputs are
- `if-else` tests must come before the clocked logic
- Since reset_n is level sensitive, why do we look at the falling edge?

```
module reg8 (
  input            clk,
  input            reset_n,
  input      [7:0] d_in,
  output reg [7:0] q_out
  );
  always_ff @ (posedge clk, negedge reset_n)
   if(~reset_n) q_out <= '0;
    else        q_out <= d_in;
endmodule
```

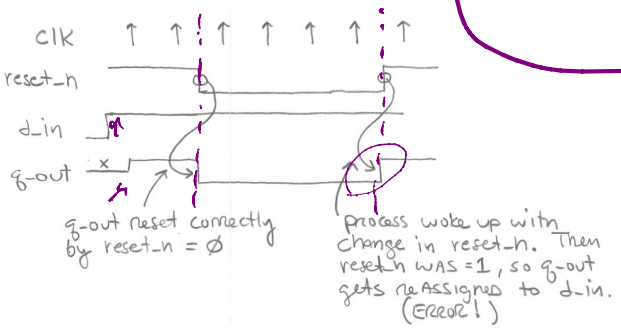


Sequential Logic

- How would it work if we coded this way?... Busted, that's how!

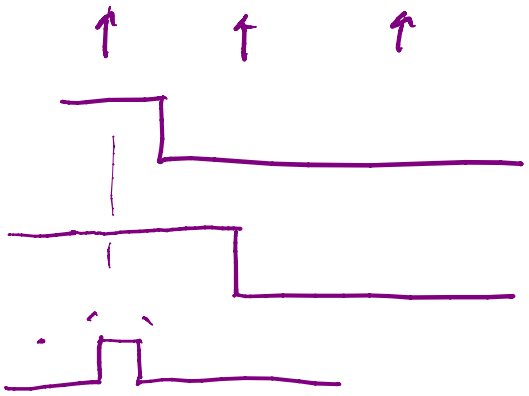
```
module reg8 (
  input            clk,
  input            reset_n,
  input      [7:0] d_in,
  output reg [7:0] q_out
  );
  always_ff @ (posedge clk, reset_n)
   if(~reset_n) q_out <= '0;
   else         q_out <= d_in;
endmodule
```

Not Asynch. Reset.



Sequential Logic

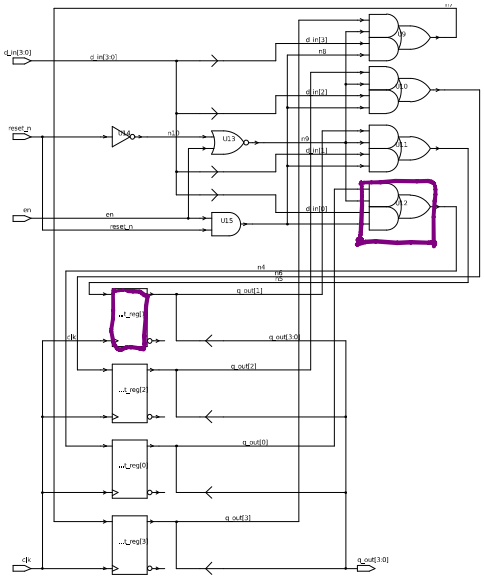
```
module reg_en4 (
  input             clk,
  input             reset_n,
  input             en,
  input      [3:0] d_in,
  output reg [3:0] q_out
  );
  always_ff @ (posedge clk)
    if(~reset_n)   q_out <= '0;
    else if(en)    q_out <= d_in;
endmodule
```



- Fully synchronous logic evaluates only on the clock edge
- What is the difference in this register and the last one?
- Why isn't `reset_n` and `en` in the sensitivity list?

Sequential Logic

```
module reg_en4 (
  input             clk,
  input             reset_n,
  input             en,
  input      [3:0] d_in,
  output reg [3:0] q_out
  );
  always_ff @ (posedge clk)
   if(~reset_n)   q_out <= '0;
   else if(en)    q_out <= d_in;
endmodule
```



What's going on here?

Sequential Logic

- What about latches?
  - Latches are not Registers or Flip-flops
  - Don't ever confuse the two
  - We will model transparent latches
  - Latches combine combo logic with sequential logic
  - Transparent: pass in to out without sequential trigger (combo logic)
  - Latched: input is stored (sequential logic)

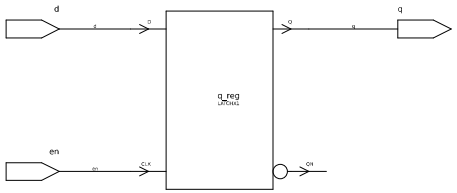
Sequential Logic

- Modeling a latch

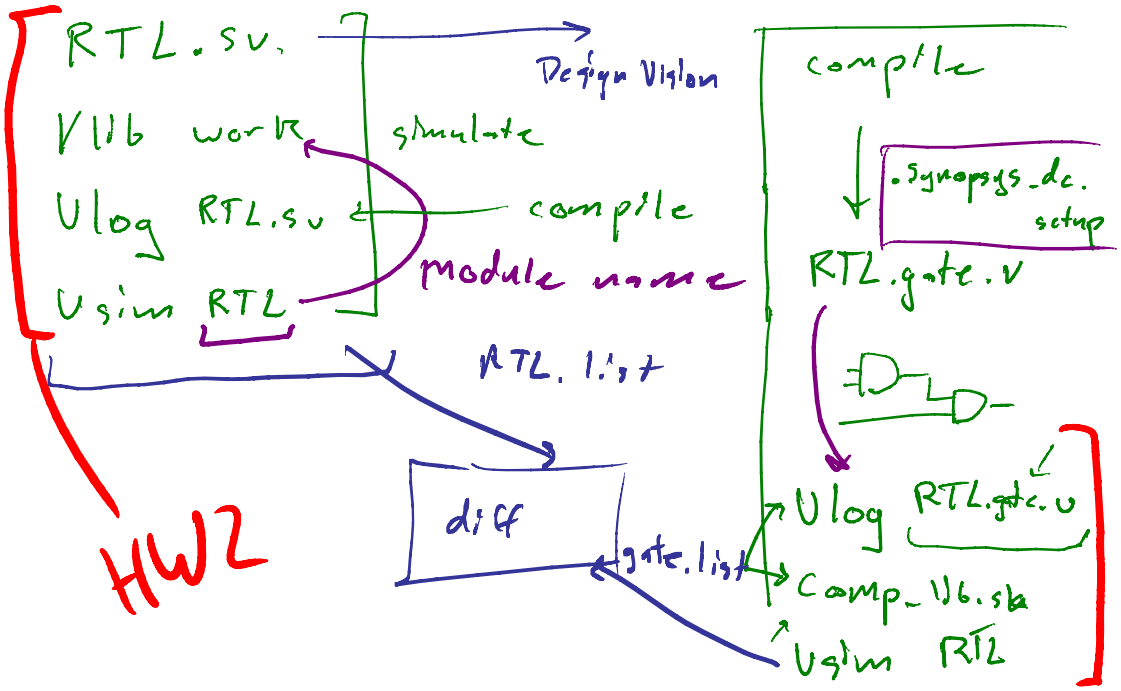
```
module latch (
  input        d, en,
  output   reg q);

always_latch
  if (en) q <= d;  //transparent latch

endmodule
```



always @(en)


RTL.sv.
Design Vision
compile
Vlib work
simulate
.synopsys_dc.
setup
Vlog RTL.sv
compile
Vsim RTL
module name
RTL.gate.v
RTL.list
HW2
diff
Vlog RTL.gate.v
gate.list
Comp_Vb.sh
Vsim RTL