Hardware Description Languages

Introduction and Combinational Logic

Joe Crop

Oregon State University

ECE 271

Outline

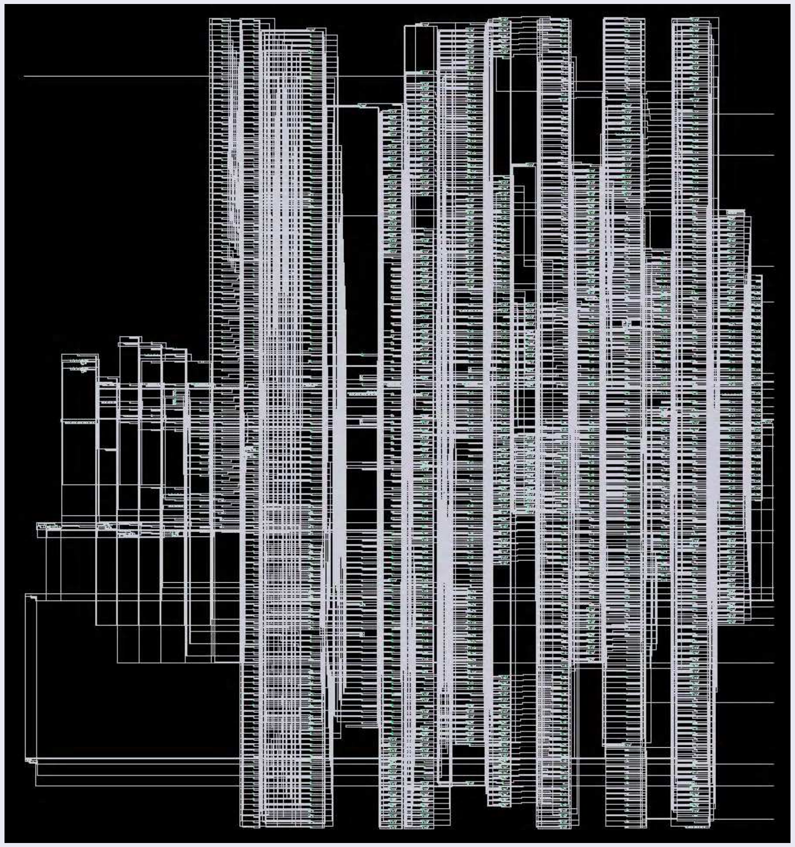
Outline

What is an HDL?

A Programming language that describes hardware.

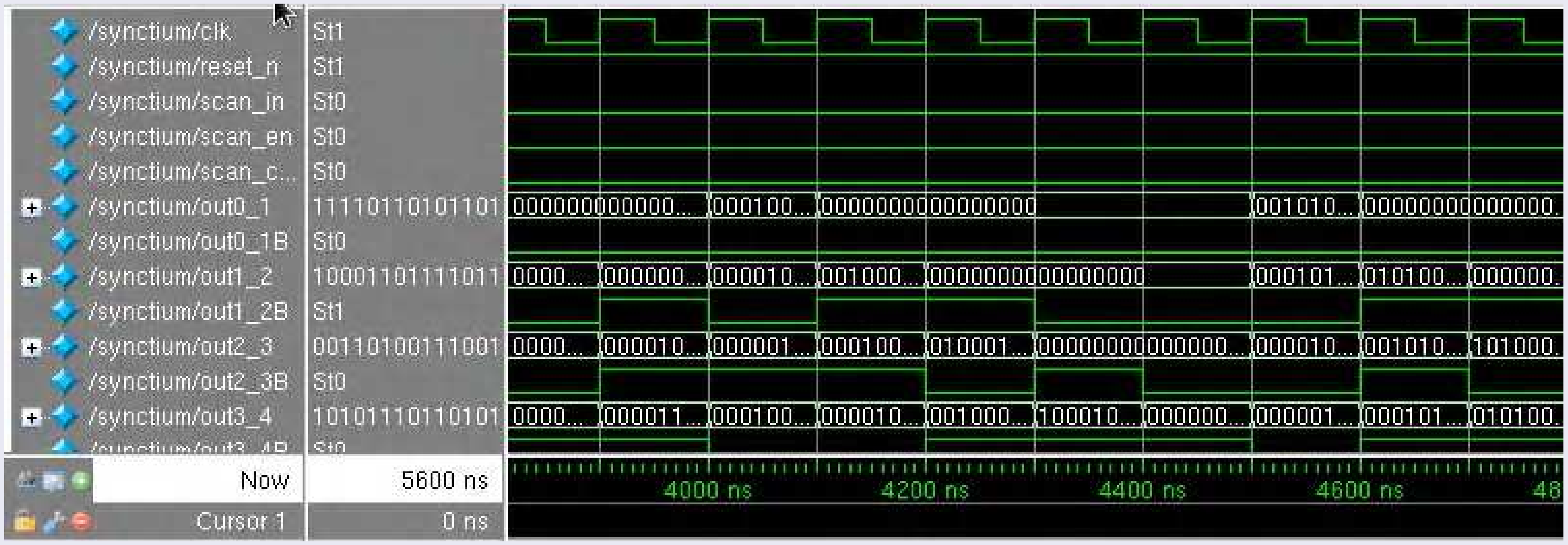
Why HDL?

1.) Schematics can get large and complicated.



Why HDL?

2.) The ability to simulate is imperative.



```
/synctium/clk         St1
/synctium/reset_n     St1
/synctium/scan_in     St0
/synctium/scan_en     St0
/synctium/scan_c...   St0
/synctium/out0_1      1111011010110
/synctium/out0_1B     St0
/synctium/out1_2      1000110111101
/synctium/out1_2B     St1
/synctium/out2_3      0011010011100
/synctium/out2_3B     St0
/synctium/out3_4      1010111011010
Now                   5600 ns
Cursor 1              0 ns
```

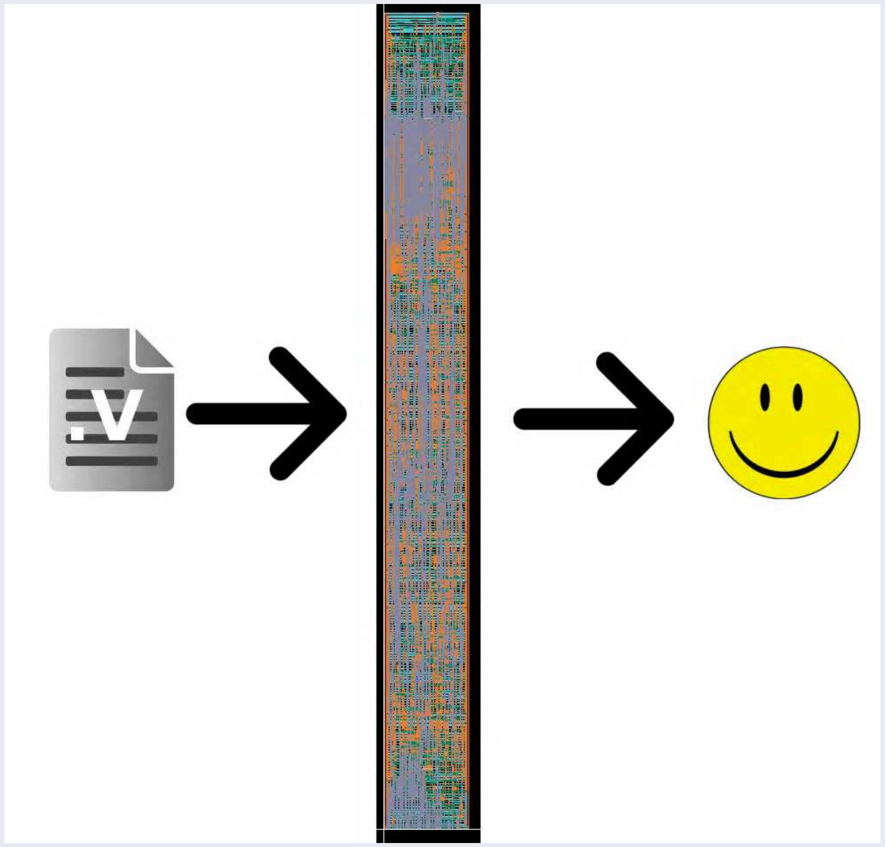
Why HDL?

3.) Logic minimization and optimization are a pain.

```
Beginning Mapping Optimizations  (Ultra High effort)
-------------------------------
 ELAPSED            WORST NEG TOTAL NEG  DESIGN
  TIME      AREA      SLACK     SLACK   RULE COST
--------- --------- --------- --------- ---------
  0:00:18  372324.0      0.00       0.0       0.6
  0:00:19  372324.0      0.00       0.0       0.6
  0:00:21  356715.6      0.00       0.0      34.4
  0:00:23  356836.8      0.00       0.0       0.0
  0:00:24  355791.0      0.00       0.0       0.0
  0:00:28  355689.0      0.00       0.0      31.1
  0:00:33  355379.1      0.00       0.0       0.0
Optimization Complete
```

Why HDL?

4.) Faster design time is needed.



Types of HDLs

- Verilog
- VHDL
- System-C
- Bluespec
- C/C++
- Matlab

Modules

Definition: A block of hardware with inputs and outputs.

Examples:

- AND gate
- multiplexer
- Adder
- CPU

VHDL/Verilog Comparison

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity gates is
 port(in1, in2: in STD_LOGIC;
      out1: out STD_LOGIC);
end;

architecture synth of gates is
 signal in1_b: STD_LOGIC;
begin

in1_b <= not in1;
out1 <= in1_b and in2;

end
```

Verilog

```
module gates(in1, in2, out1);

input  in1;
input  in2;
output out1;

wire in1_b;

assign in1_b = ~in1;
assign out1 = in1_b & in2;

endmodule
```

Typical Design Cycle

1. Concept
2. Design
3. Simulate
4. Synthesize
5. Implement
6. Sign-off

Outline

Bitwise Operators

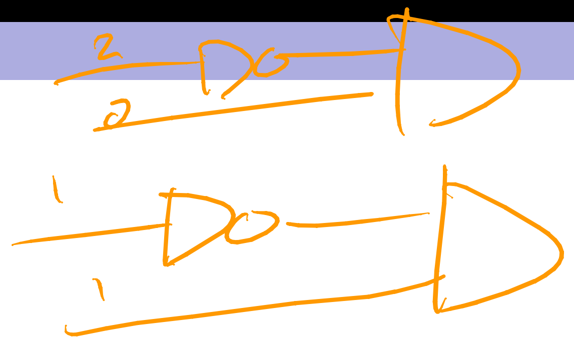
operator	function
~	NOT
*, /, %	Multiply, Divide, Modulo
<<, >>	Logical Shift Left, Right
<<<, >>>	Arithmetic Shift Left, Right
<, <=, >, >=	Relative Comparison
==, !=	Equality Comparison
&	AND
~&	NAND
^	XOR
~^	XNOR
|	OR
~|	NOR
?:	Conditional

Bitwise Operators: Example

gates.v

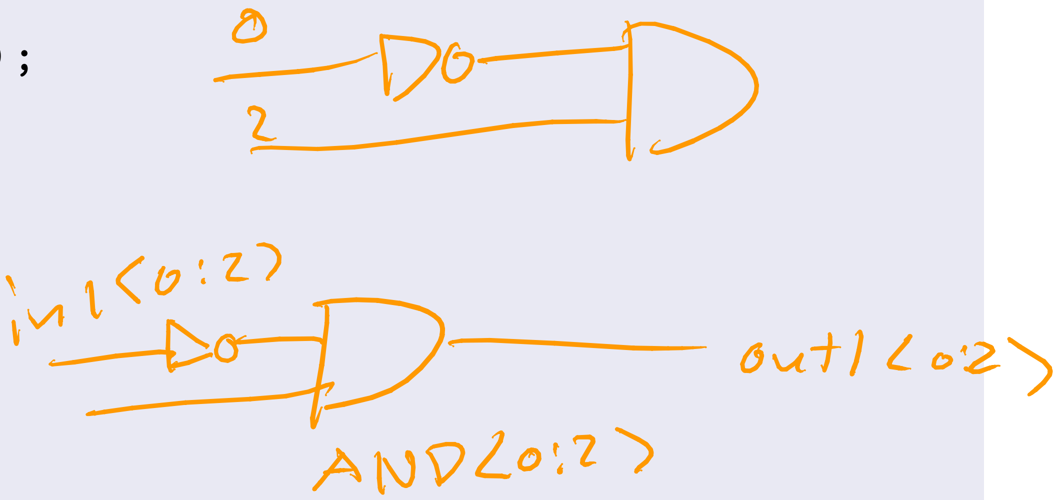
```
module gates( a, b, y1, y2, y3, y4, y5);
    input  a;
    input  b;
    output y1;
    output y2;
    output y3;
    output y4;
    output y5;

    assign y1 = a & b;       // AND
    assign y2 = a | b;       // OR
    assign y3 = a ^ b;       // XOR
    assign y4 = ~(a & b);    // NAND
    assign y5 = ~(a | b);    // NOR
endmodule
```

Buses



Buses of wires

```
module gates(in1, in2, out1);

input  [0:2] in1;
input  [2:0] in2;
output [0:2] out1;

wire [0:2] in1_b;

assign in1_b = ~in1;
assign out1 = in1_b & in2;

endmodule
```



Comments

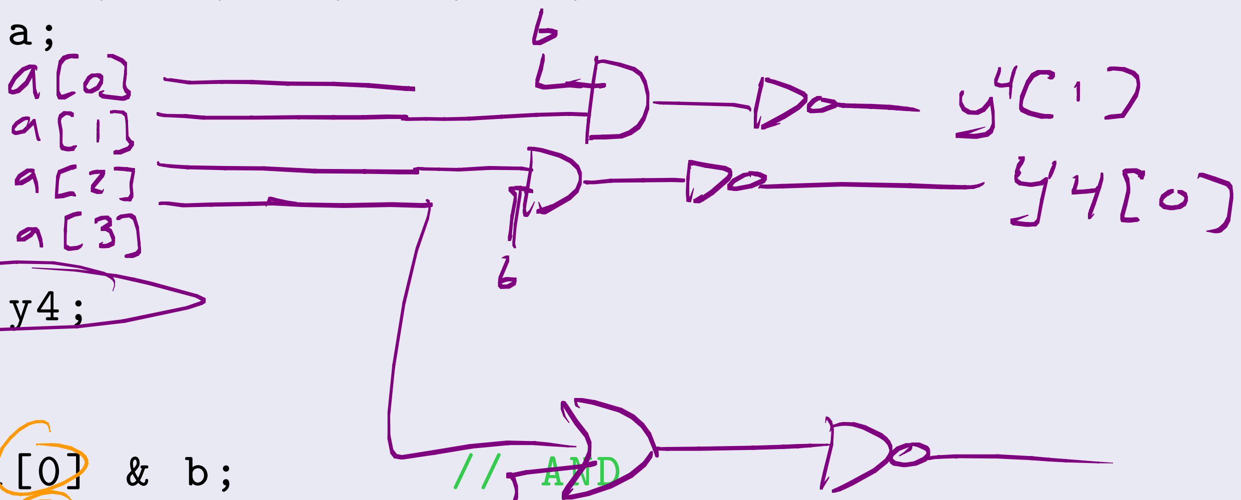
Comments in Verilog

```
module gates( a, y);
    input  a;
    output y;

/* A block coment in Verilog
 * consists of either C/C++
 * style comment statements
 * */

    assign y1 = a & b;       // an and gate

endmodule
```

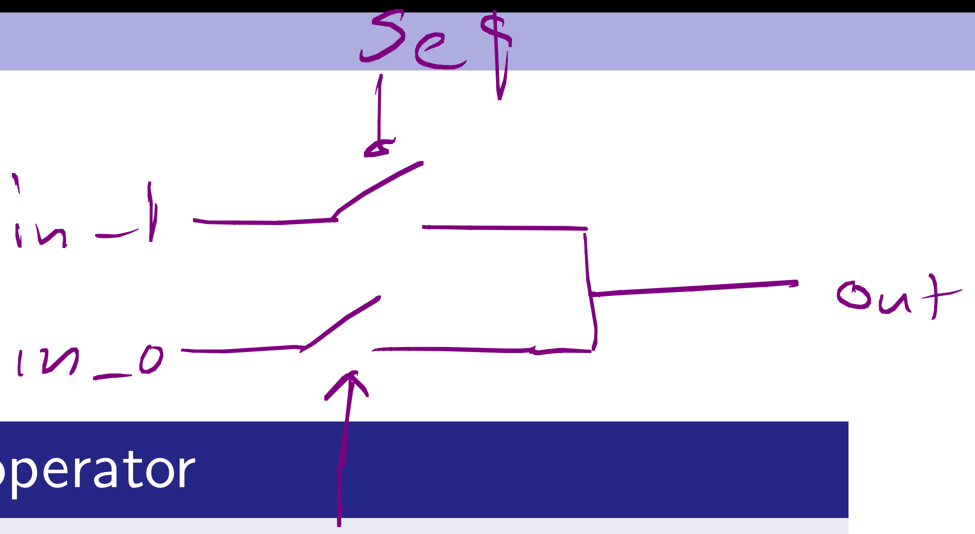
Reduction Operators

Buses of wires

```
module gates( a, b, y1, y2, y3, y4, y5);
     input  [3:0] a;
     input  b;
     output y1;
     output y2;
     output y3;
     output [1:0] y4;
     output y5;

     assign y1 = a[0] & b;          // AND
     assign y2 = a[1] | b;          // OR
     assign y3 = a[2] ^ b;          // XOR
     assign y4 = ~(a[1:2] & b);     // NAND
     assign y5 = ~(a[3] | b);       // NOR
endmodule
```

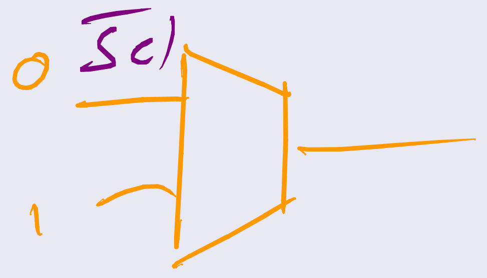


Multiplexers



Multiplexers using the Conditional operator

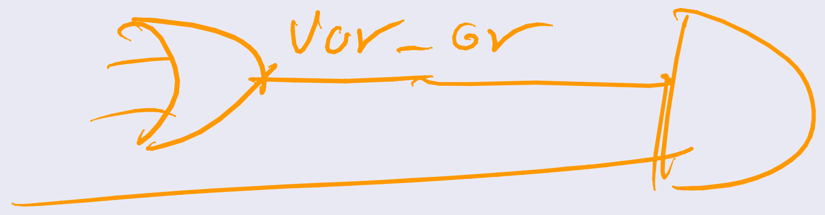
```verilog
module mux( in_0, in_1, sel, y);
     input   in_0;
     input   in_1;
     input   sel;
     output  y;

     assign y = sel ? in_1 : in_0;          // A simple MUX

endmodule
```



Internal Variables

Internal Variables

```
module gates( a, b, y);
    input  a;
    input  b;
    output y;

    /* an internal "variable" */
    wire var_or;

    assign var_or = a | b;        // OR
    assign y = a & var_or;        // AND
endmodule
```



Formatting Numbers

- Format: (# of bits)' (base) (value)
    - 3'b101 = binary 101
    - 8'b101 = binary 00000101
    - 3'd6 = decimal (binary 110)
    - 8'hAB = hexadecimal (binary 10101011)
    - 6'o42 = octal (binary 100010)
- Things to (almost) never do:
    - 'b11 = 000...0011
    - 42 = 000...0101010
    - 4'bz = high-impedance (zzzz)
    - 4'bx = not defined (xxxx)

Tristate Buffers

Example

```verilog
module tristate( in, enable, y);
     input   in;
     input   enable;
     output  y;

     assign y = sel ? in : 1'bz;          // A tristate buffer

endmodule
```

Bit Swizzling

More commonly known as concatenation

```
module gates(in1, in2, out1);

input   [0:2] in1;
input   [2:0] in2;
output  [0:2] out1;

wire [0:2] in1_b;

assign in1_b = {~in1[0:1], in2[2]};
assign out1 = in1_b & in2;

endmodule
```

Simulating Delays

Example

```
module gates( a, b, y1, y2, y3, y4, y5);
    input   a;
    input   b;
    output  y1;
    output  y2;
    output  y3;
    output  y4;
    output  y5;

    assign #2 y1 = a & b;           // AND
    assign #3 y2 = a | b | y1;      // OR
    assign #1 y3 = a ^ b;           // XOR
    assign #1 y4 = ~(a & b);        // NAND
    assign #1 y5 = ~(a | b);        // NOR
endmodule
```

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Outline

Structural Modeling

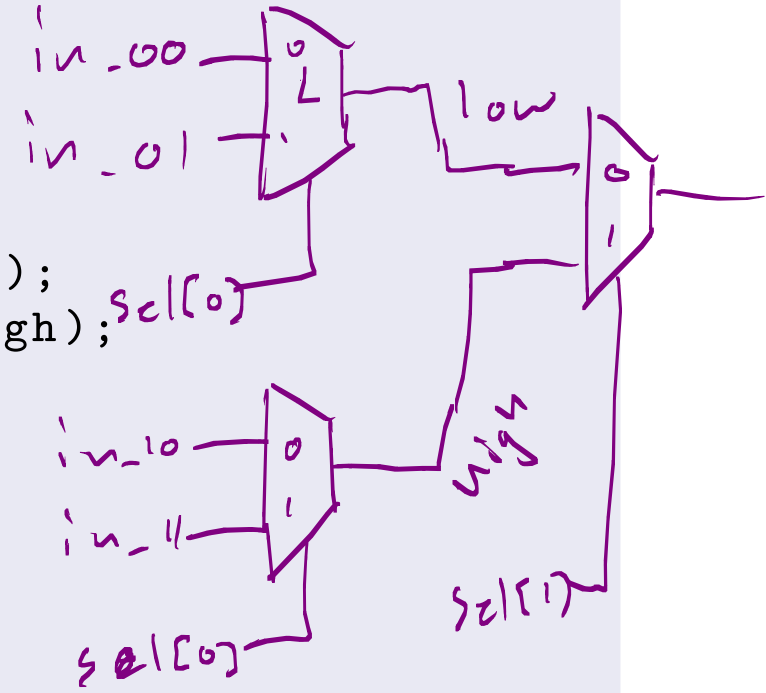
Behavioral Modeling: Describing a module in terms of inputs and outputs.

Structural Modeling: Describing a module in terms of how it is composed of sub-modules.

A Simple Structure

Example

```
module mux4( in_00, in_01, in_10, in_11 sel, y);
     input   in_00, in_01, in_10, in_11;
     input   [1:0] sel;
     output  y;

     wire ~~[3:0]~~ low, high;

     mux LOWMUX(in_00, in_01, sel[0], low);
     mux HIGHMUX(in_10, in_11, sel[0], high);
     mux FINALMUX(low, high, sel[1], y);
endmodule //mux4

module mux( in_0, in_1, sel, y);
     input   in_0, in_1, sel;
     output  y;

     assign y = sel ? in_1 : in_0;          // A simple MUX
endmodule //mux
```



The Correct way to do it!

Example

```
module mux4( in_00, in_01, in_10, in_11 sel, y);
    input  in_00, in_01, in_10, in_11;
    input  [1:0] sel;
    output y;

    wire [3:0] low, high;

    mux LOWMUX(
                .in_0    (in_00),
                .in_1    (in_01),
                .sel     (sel[0]),
                .y       (low)
            );
    mux HIGHMUX(
                .in_0    (in_10),
                .in_1    (in_11),
                .sel     (sel[0]),
                .y       (high)
            );
```

Local

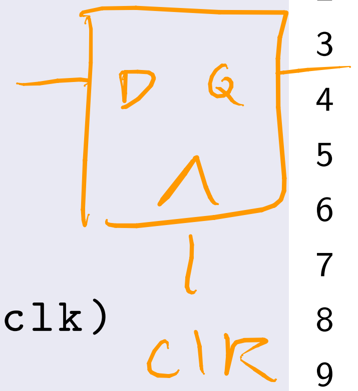
global

Outline

Registers (DFF)

Positive Edge

```
module flop(clk, d, q);
    input clk;
    input d;
    ~~in~~put q;
    out

    reg q;

    always @(posedge clk)
        q <= d;

endmodule //flop
```



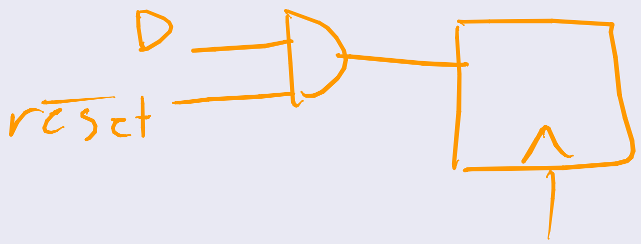
Negative Edge

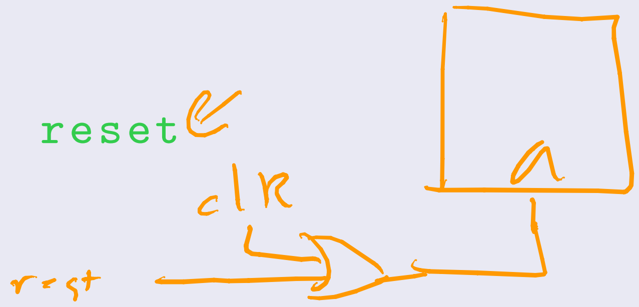
```
module neg_flop(clk, d, q);
    input clk;
    input d;
    input q;

    reg q;

    always @(negedge clk)
   { begin
        q <= d;
   } end

endmodule //neg_flop
```

Resettable Registers (DFFR)

Example

```
module flopr(clk, d, q, reset);
    input clk, reset;
    input [0:1] d;
    input [0:1] q;
   out
    reg [1:0] q;

    always @(posedge clk) //synchronous reset
        if(resest)   q[0] <= 0;
        else         q[0] <= d[0];

    always @(posedge clk, posedge reset) //asynchronous reset
        if(resest)   q[0] <= 0;
        else         q[0] <= d[0];

endmodule //flopr
```





Enable Registers (DFFE)

Example

```
module flope(clk, d, q, reset, en);
    input clk, reset, en, d;
    output q;

    reg q;

    always @(posedge clk, posedge reset) //asynchronous reset
    begin
        if(resest)
        begin
            q <= 0;
        end
        else if(en)
        begin
            q <= d;
        end
    end
endmodule //flope
```

Multiple Registers

Example

```
module flop(clk, d, q);
    input clk, d;
    output q1;

    reg q0, q1;

    always @(posedge clk)
    begin
        q0 <= d;
        q1 <= q0;
    end
endmodule //flop
```

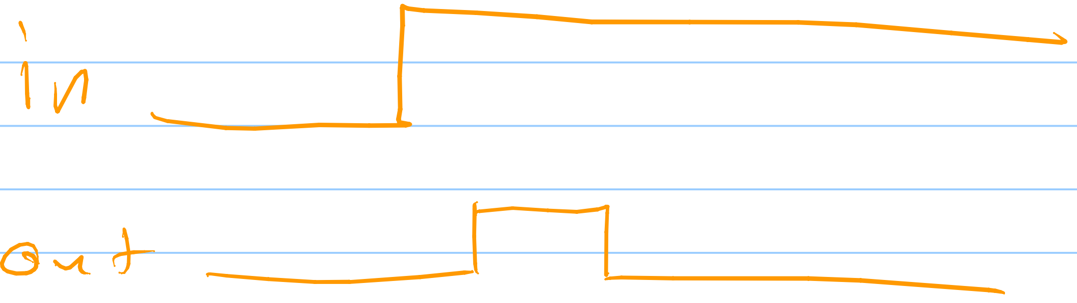
Latches

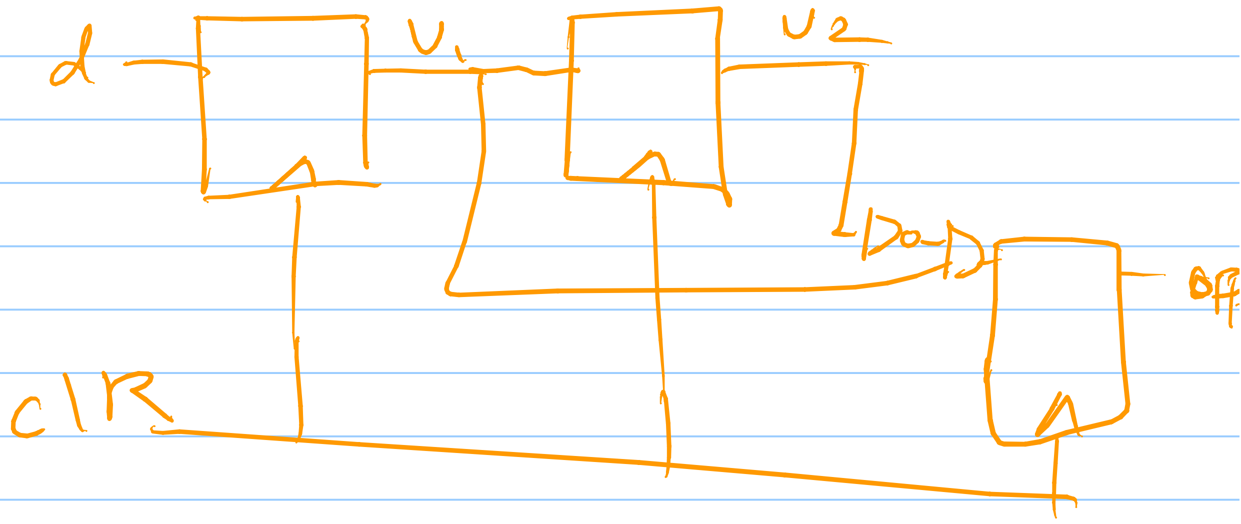
Transparent when clock is high.

Example

```
module latch(clk, d, q);
     input clk, d;
     output q;

     reg q;

     always @(clk, d) //senitivity list
          if(clk) q <= d;

endmodule //latch
```

```
module pulse_generator (d, clk, op);
  input d, clk;
  output op;

  reg op;
  reg v1, v2;

  always @(posedge clk)
    begin
       v1 <= d;
       v2 <= v1;
       op <= v1 & ~v2;
    end
endmodule
```





More Cool stuff

- Case Statements (4.5.1)
- Combinational if/else (4.5.2)
- Blocking / Non-Blocking Assignments (4.5.4)

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Outline

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop