April 8, 2021

Concurrency

Single-bit

```
wire reset;
reg  start;
```

Multiple-bit

```
[(#bits-1) : 0]
```

```
wire [7:0] phase;
reg [31:0] a;
reg [7:0] b;
b = a[15:8];
```

↑ wire or reg

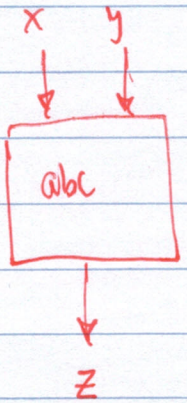
1101 = 9

8 4 2 1

2^3 2^2 2^1 2^0

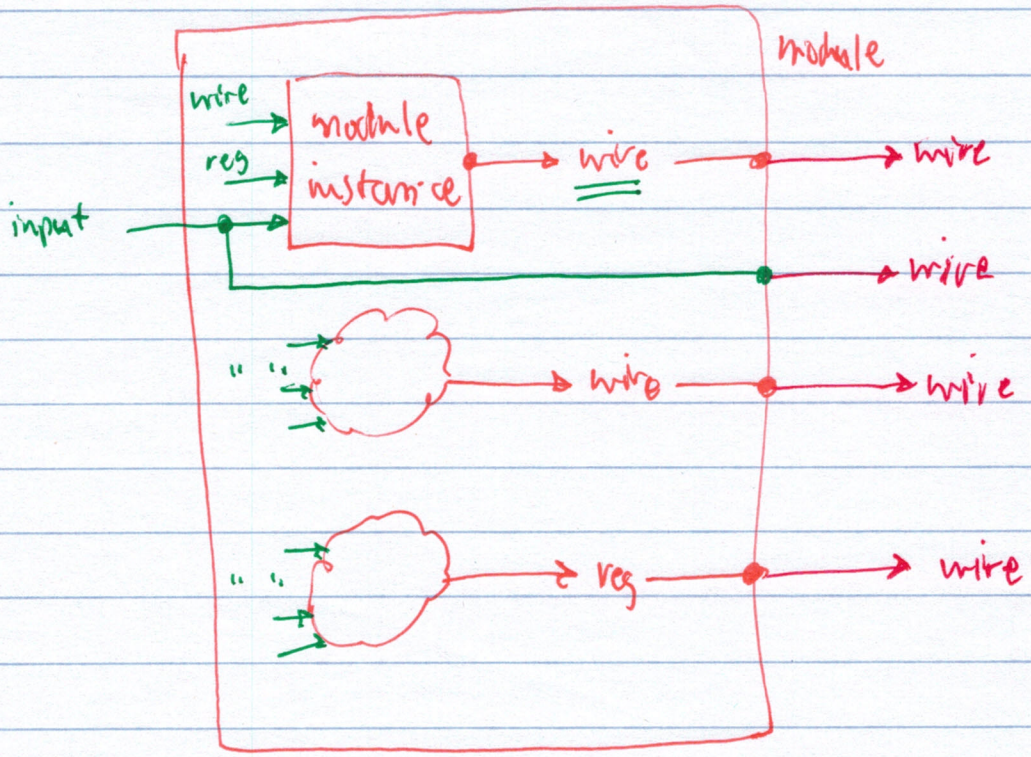
a[3:0]

```
module abc (in1, in2, out);
    input in1;
    input in2;
    output out;
    ==
endmodule
```

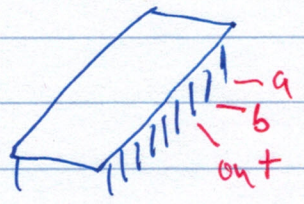



```
x abc instance1 (x, y, z);
```

```
abc instance2 (          // use this
    .in1  (x),
    .in2  (y),
    .out  (z) );
```

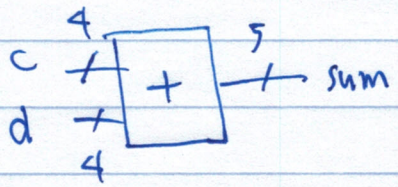


2) Wire, Assign




```
wire out;

assign out = a & b;
```



wire [4:0] sum;

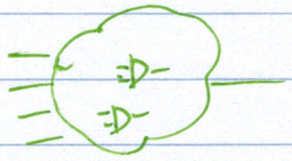
assign sum = {c[3], c} + {d[3], d}; // sign extension

3) reg, always

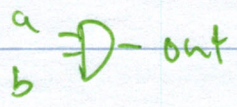
- more general
- if/then/else, case, for
- statements execute in order to specify a circuit

- always @ (sensitivity list) begin
  statements
  end

statements executed when a signal in sensitivity list changes



- V. 2001
  always @(*) begin // incl. all inputs in sens. list




```
reg out;

always @(a or b) begin
    out = a & b;
end
```

```
initial begin
    ===
    ===
    ===
end
```

executes once at beginning of simulation

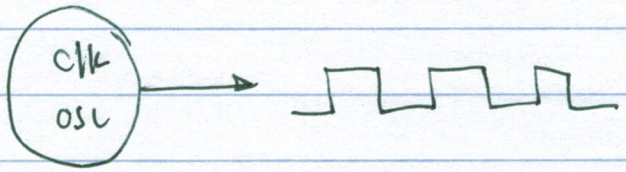
* 1) Test bench
  2) contents of a ROM

```
always begin
    ==
    ==
end
```

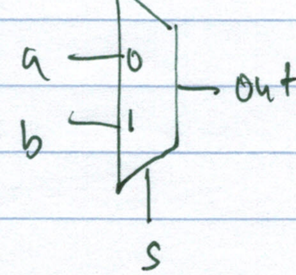
executes repeatedly

needs delay inside

Ex: clock generator

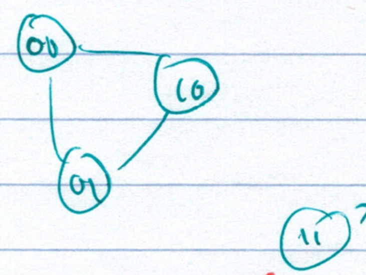


Ex:- 2:1 mux



1)
```
reg out
always @ (a or b or s) begin
    if (s == 1'b0) begin
        out = a;
        end
    else begin
        out = b;
        end
end
```

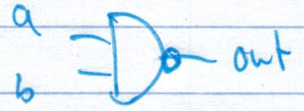
```
case (expr)
    value1: "statement" → begin
                           ≡
                          end
    value2: statement
       ≡
    default: statement    (optional)
endcase      state = xx
```



casez

" ? " wildcard

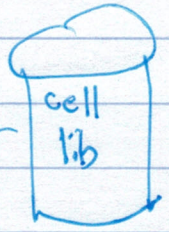
casex - don't use

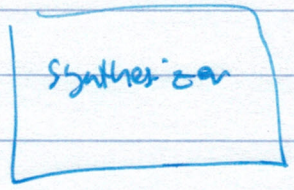


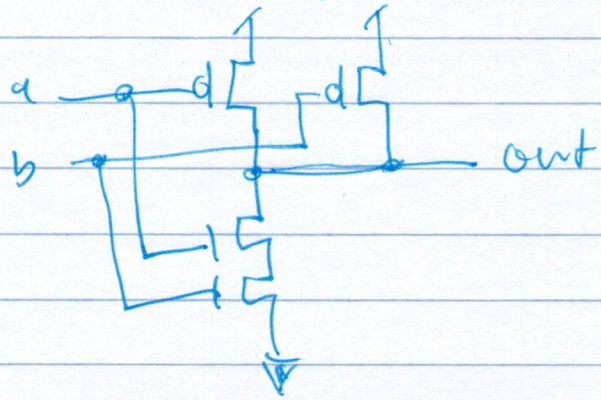
ab	out
00	1
01	1
10	1
11	0

```
reg out;
always @(*) begin
    out = 1'b1;
    if (___)
        out = 1'b0;
    end
end
```

in order







d = {c, r, x, b[4:2], y}