Behavioral Modeling

- describes how the circuit behaves, not how it is constructed : behavioral design
- logic synthesisers read the description and builds the circuit

Combinational Always Block

eg 1

```
always @ (a, b)   ← sensitivity list (must contain the names of any signals that will affect the output of the block)
  begin
     x = a & b ;
     y = a | b ;      } → procedural statements
  end                   order matters
                        or sequential statements
```

if there is more than 1 statement

- instead of writing out every signal in the sensitivity list, just put a *

  always @ *

- dont ever use an assign statement in the always blk
- for a signal inside an always block, keyword reg is used outside the block to declare it

eg.

```
module ......
   reg w1 ;
   always @ *
      w1 = a & b ;
   assign out = w1 ;
endmodule
```

- reg is more like a variable while a wire is
- can't assign to reg using an assign statement
- if output is assigned within the always block, then in the module desc it should say :

  output reg <name>) ;
- reg [3 : 0] x = 4'b0000; is valid

  ↑ multi-bit   ↳ assign initial values
- can use if statements (java syntax) in always blocks (nested too)

  ↑ if there's more than one statement in any of the blocks, a begin ... end must be used in that block.

  ↓ = { } in c

  ⌐ in verilog {} = concatenation
- group signals in always blocks only if it makes logical sense

- case statements (almost like switch)

eg 2.
```
always @ *
  case (sel)
        2'b00  :  y = a;
        2'b01  :  y = b;
        2'b10  :  begin
                       y = c;
                       z = d;
                  end
        default :  y = 4'b0101;
  endcase
```

↑ used in Decoder

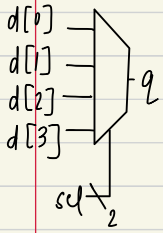
- decoder is a one-hot output

- 3-to-8 decoder

```
module decoder3_8 ( output reg [7:0] d_out,
                    input [2:0] ival);
   always @ *
   case (ival)
      3'b000 : d_out = 8'b00000001;
      :
      :
      3'b111 : d_out = 8'b10000000;
   endcase
endmodule
```

single statement →

Implementing a multiplexer



```
module mux4 (output reg q,
             input [3:0] d,
             input [1:0] sel);

  always @ *
   begin
     case (sel)
       2'b00 : q = d[0];
       2'b01 : q = d[1];
       2'b10 : q = d[2];
       2'b11 : q = d[3];
     endcase
   end
endmodule
```

unnecessary

```
module Y (output reg q,
          input [3:0] d,
          input [1:0] sel);

  always @ *
    assign q = d[sel];

endmodule
```

DOUBT

converts to dec?

But what is an always block?

- acts as an infinite loop
- order matters as it is executed sequentically
- order of placement of always block in the module doesn't really matter