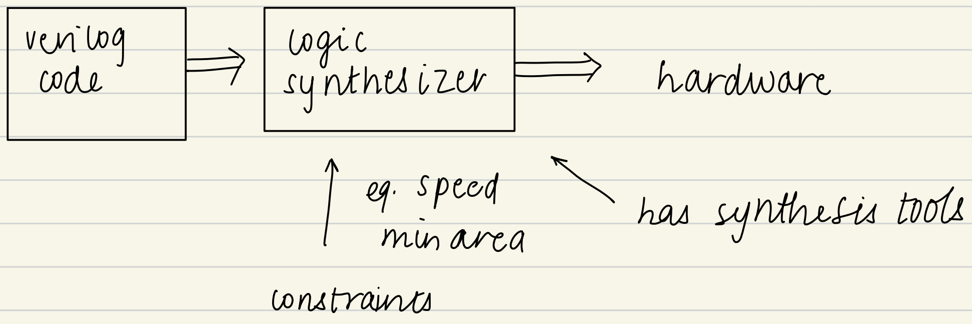
- combinational circuits in VL
- sequential circuits (memory)
- state machines

Intro to Verilog

- hardware - description language

  - special languages with special constructs for describing hardware
  - can describe hardware at various levels of abstraction and hence enables hierarchial design

- .V extension



-

HDL	Prog. Lang.
synthesize and optimise hardware primitives	· compiled to primitive instructions to run on an existing piece of hardware

· modules

- designs are broken-down into modules
- container a designer uses to encapsulate a unit of functionality

- and gate, adder, etc

- can contain instances of other modules

- good designs have sufficient hierarchy with modules containing instances of modules containing instances of modules

· declaration

only assign & read

eg1 module <name> $\underbrace{\text{(port 1 , port2, port3)}}_{\text{single wire, means of communication}}$;

$\boxed{}$

endmodule

eg2 module <name> (input port1, port2,
output port3);

$\boxed{}$

end module

structural design in verilog

- describing a circuit by its internal structure

- 1. declare module
  2. introduce gates
  3. have wires

- primitives in verilog to model gates: and, nand, or, nor, not, xor, xnor

eg1 and (y, a, b)   (y, a, b are signals)

$\underset{\uparrow \text{ o/p}}{y}$, $\underbrace{a, b}_{\text{input}}$ ↳ either ports or wires

eg2 module w/ 2 input and.

```
module simple_AND ( input in1, in2,
                    output out );

        and ( out , in1, in2) ;

endmodule
```

* ports ≡ wires but o/p port can't be connected to an input port.

* we wouldn't create a module for a gate that already exists as a primitive

- wires → anchor pts to which you can connect logic

- internal wires can be declared in a module using wire keyword.

```
wire int_signal;
```
← 1-bit wire connects the gates

- verilog module with gate level primitives

  - and-or-inverter
  - boolean exp: $out = ((a \cdot b) + (c \cdot d))'$

```
module andorinv (input a, b, c, d,
                 output out);

    wire o1, o2;

    and g1 (o1, a, b);
    and g2 (o2, c, d);
    nor g3 (out, o1, o2);

endmodule
```
← no need to declare 1-bit wires

gate identifiers

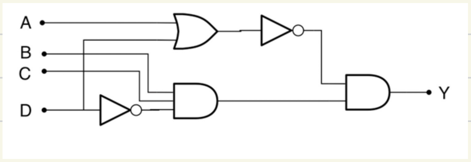
· half adder in verilog.

sum . $A \oplus B$ 2 i/p 2 o/u
carry : $A \cdot B$

```
module halfadder ( input a, b,
                   output sum, carry);

       and g1 (carry, a ; b) ;
       xor g2 (sum, a , b) ;

endmodule
```

· 

$(A+D)' \cdot BCD' = Y$

verilog rules

- order of instantiation doesnt matter
- case sensitive
- ;
- C-style comments (// or /*....*/)
- whitespace is ignored
- identifiers
  - letter / underscore as 1st char
  - letter / numbers / $
  - mustn't clash with keywords

module instantiation

$S_2 = (A \oplus B) \oplus C$
$C_2 = AB + C(A \oplus B)$

FA = HA + HA + OR

```
module halfadder (input a, b,
                  output sum, cout)
                  i/p first
                  o/p second
    xor g1 (sum, a, b..);
    and g2 (cout, a, b);
endmodule
```

```
module fulladder (input A, B, CIN, output SUM, COUT);

    or g1 (COUT, w3, w2)
   half adder M2 (CIN, w1, SUM, w3)
   half-adder M1 (A, B, w1, w2)
endmodule
```

alt: halfadder M1 (.a(A), .b(B), .sum(w1), .Cout(w2));

unused ports should be left as : .b(), ← preferred or completely left out