assignment

vectors

number literals

Verilog Assignments

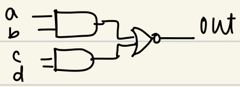
allows for description of a circuits structure

- instantiating gate primitive : and (y, a, b);

- continuous assignment : assign y = a & b   use of boolean statements

allows for description of a circuits function

~	NOT	bitwise
&	AND	
|	OR	
^	XOR	
~&	NAND	
~|	NOR	

eg.


assign out = (a&b) ~| (c&d);

~ & | ^ ~^ ← L to R precedence

eg2. full adder. $S = A \oplus B \oplus C_{in}$ $C = AB + BC + AC = AB + C_{in}(A+B)$

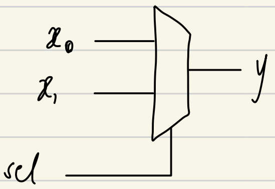
```
module full_adder (input a, b, cin,
                   output s, c    );

   assign s = a ^ b ^ cin ;
   assign c = (a&b) | cin & (a|b) ;

endmodule
```

happen concurrently ∴ order does not matter basically an if statement

- conditional assignment

```
assign y = sel ? x1 : x0 ;   // a multiplexer
```

∴ conditional ≡ mux



- combinational arithmetic

+, -, *, /   (div is not usually synthesizable)
<, <=, >, >=, ==, !=

↑ comparisons give true (1) or false (0)

eg3.
```
assign sum = a+b;
assign diff = curr - prev;
assign max = (a>b) ? a : b;
```

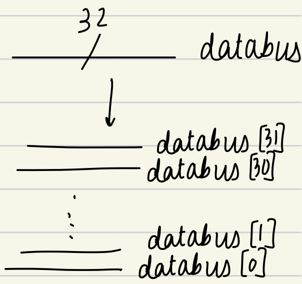
Vectors in Verilog

· multi-bit inputs

MSB ↘ ↙ LSB

```
wire [31:0] databus;
```

range of the wire : 32 bits



eg 4

```
module add16 (input [15:0] a, b,
              output [15:0] sum,
              output cout );

endmodule
```

2 16 bit inputs + o/p
1 1 bit output

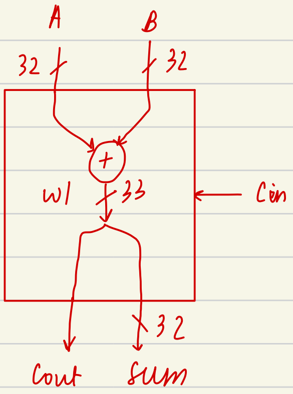
eg5

```
module adder (
        input cin,
        input [31:0] A, B,
        output cout,
        output [31:0] sum);

        wire [32:0] w1;

        assign w1 = A + B + cin;
        assign sum = w1[31:0];
        assign cout = w1[32];
endmodule
```

multibit wires must be declared →

assigning the msb of a vector to another

assign cout = W1 [32 : 10]; also valid
assign A[15] = B[10] ; also valid



- careful:

$$\begin{tikzcd} \text{multi-bit} \arrow[d] & \text{1 bit} \arrow[d] \\ \texttt{assign x[2:0]} & \texttt{= y[1]} \end{tikzcd}$$

≡

x2	0
x1	0
x0	< y[1] >

} verilog adds 0 to cover gap

- concatenation

```
wire [3:0] a, b;
wire [7:0] y;
:
assign y = {a, b};
```

⟹
```
y[7] = a[3]
  :
y[4] = a[0]
y[3] = b[3]
y[0] = b[0]
```

concatenation operators

- replication

```
assign c = {{4{a[3]}}, a[3:0]};
```

a = 1010 ⟹ c = 11111010

Number Literals

- assigning values

  assign s = 1 ;   dec? hexa? oct?
                        how many bits?

- <size> : in bits
  <radix> : b for binary, o.., h..., d...
  <value> : number w/ optional underscores for readibility

  assign s = <size>'<radix><value> ;
  assign s = 4'b0001 ;

- if a literal is assigned to a larger signal, it is zero padded at the MSB
  and if the opposite is done, it is truncated at the MSB

eg 5. assign b = {a[3:0], 4'b0000} ;

$a = 1010_2 \Rightarrow b = 10100000$

Parameter

- constant that is local to a module

→ can be declared in the module header

syntax keyword constant'sname

```
module some_mod #(parameter SIZE=8)
        ( input [SIZE-1:0] X,Y,
          output [SIZE-1:0] Z );
```

can have multiple

⇒ X,Y [7,0] and so on

- when instantiating

```
some_mod #(.SIZE(value)) U1 (.X(a), .Y(b), .Z(c));
```

eg6
```
module adder #(parameter SIZE=32) (
            input Cin,
            input [SIZE-1:0] A,B,
            output Cout,
            output [SIZE-1:0] sum);

   assign {Cout, Sum} = A + B + Cin ;
endmodule
```

MSB [31:0]

operator to generate sum