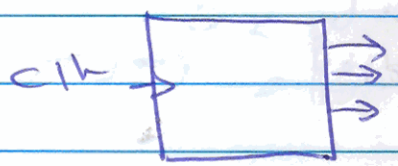
⊛ Two ways to see bus

1- STD_LOGIC_VECTOR (3 downto 0)
mem suitable to describe buses on which logical operations are made (and, or, not, ... etc).

2- Bus seen as number
(INTEGER Range 0 to 15)
mem suitable for arithmetic operations.

Ex: 3 bit Counter



use ieee.std_logic_arith.all

```
Entity counter3 IS
  port ( clk: in std_logic;
         Count: inout integer range 0 to 7);
end;

Architecture behav of counter3 is
begin
  process (clk)
    begin
      if (rising_edge(clk)) then
        Count <= Count+1;
      end if;
    end process;
end;
```

⊛ open Ports

Sometimes not all the ports of an entity are needed to be connected. These ports can be left unconnected.

Ex: universal gate with 3 inputs: x, y & invert, 2 outputs a, and b

a = x and y if invert = 0
a = x nand y " " = 1
b = x or y if invert = 0
b = x nor y if invert = 1

```
Entity universal is
    port ( x, y, invert : in bit;
           a, b         : out bit );
end;
architecture univ of universal is
begin
    a <= x and y when invert = '0'
         else x nand y;
    b <= x or y when invert = '0'
         else x nor y;
end;
```

⟹ now we need to use and gate only ⟹

```
U0 : entity work.universal(univ) port map
     (x, y, '0', a, OPEN);
```

- outputs can be left unconnected but inputs must be connected EXCEPT if the have a default value.

e.g entity universal is

```
port (x,y: in bit;
      invert: in bit := '0';
      q,b : out bit);
end;
```

⇒

```
U0 : universal port map (x, y, open, q, open);
```

⊛ GENERIC

- not only for n-bit entities (it has many applications).

Ex: many and gates are required, each has different delay ⇒

```
entity and2 is
    generic (delay: delay_length := 5ns)
    port (x, y : in
          z : out)
;
```

g1: entity work.and2(simple) port map (a, b, c);

g2: // // // generic map(5 ns)
port map ( )

g3: generic map (open) port map(
default

⊛ Assert and Report statement

The assert and report statement can be used to check that expected conditions are met within the design. Otherwise, the simulator will print a statement to indicate a specific violation.

Ex: in RS flip-flop we must avoid the case where $S = R = 1 \Rightarrow$

```
architecture simple of rs is
begin
  process (clk)
  begin
    assert (not (s='1' and r='1'))
    report "set and rset are active in
            the same time!!"
    Severity warning;
  if (s='0' and r='0') then q2<=q;
  elsif ... ..
```

Ex: in a D-FF we need that D is stable for at least 2 ns before the +ve edge of the clock (setup time)

```
assert (not (clk='1' and clk'event and
            not clk'stable(2 ns))
report "setup time violation"
Severity note;
```

- There are 4 types of severity:-
  - note
  - warning
  - error
  - failure

- The assert statement is passive, meaning that there is no signal assignment. Passive statements may be included in the entity part of a declaration (to avoid repetition in different architectures).

Ex:
```
entity rs is
    port(                    );

begin
    assert ( not (s='1' and r='1'))
      report "          "
      severity "          ";
```

- default report: "assert violation"
  default severity: error

- different simulators (tools) have different behaviors for assert statem

* Loop statement :
- Sequential statement

Ex: ~~entity counter is port(clk : in bit; count : out~~
Process
variable count : integer := 0;
begin
:
loop
wait until clk = '1';
count := (count + 1) mod 16;
:
end loop;
end process;
:

- For Loop
Ex: for i in 0 to 7 loop
:
end loop;

- while loop
Ex: while index > 0 loop
:
end loop

(while (conditions) loop
:
end loop ;)

⊛ Exit statement

Ex.

... loop

:

if (Condition) then EXIT;

(exit when condition) END IF;

:

end loop;

⊛ Next statement

loop

:

next when (condition);

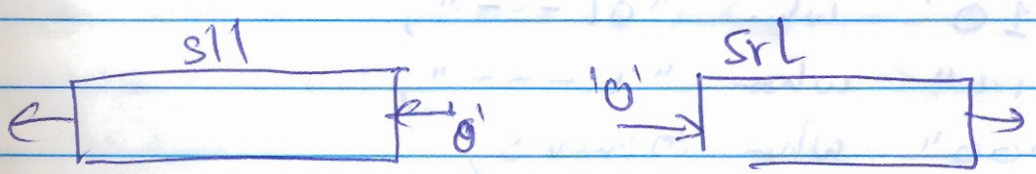
:

end loop

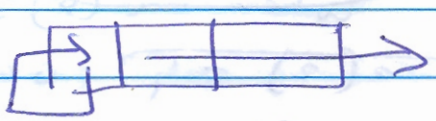
Ex: Decoder (n x 2^n decoder)

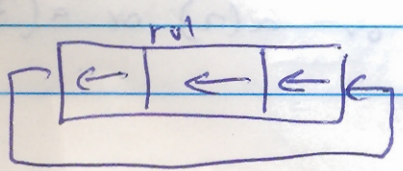
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity decoder IS
    generic (n: positive);
    port (a: in std_logic_vector(n-1 downto 0);
          z: out std_logic_vector(2**n-1 downto 0));
end;

architecture behav of decoder is
CONSTANT z_out : std_logic_vector(2**n-1
  downto 0) := (0=>'1', others=>'0');
begin
z <= z_out SLL (to_integer(unsigned(a)));
end architecture;
```



sla, sra





ror

Ex: Priority encoder

A_0	A_1	A_2	A_3	Y_1	Y_0	V
0	0	0	0	0	0	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

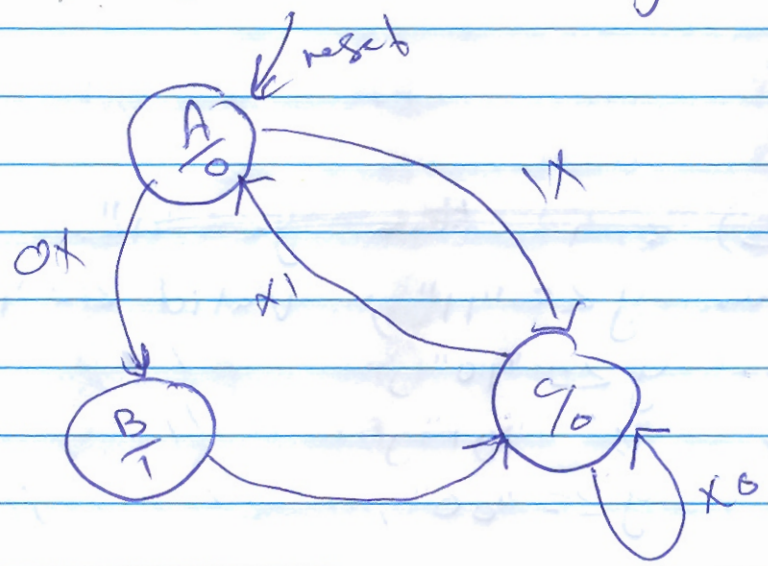
```
entity priority is
  port( a: in std_logic_vector (3 downto 0);
        y: out std_logic_vector (1 downto 0);
        v: out std_logic);
end;

architecture dontcare of priority is
begin
  with a select
  y <= "00" when "0001",
       "01" when "001-",
       "10" when "01--",
       "11" when "1---",
       "00" when others;
  valid <= '1' when a(0)='1' or a(1)='1'
           or a(2)='1' or
  valid <= a(0) or a(1) or a(2) or a(3);
end;
```

```
architecture sequential of priority is
  begin
    process(a)
    begin
      if a(3) = '1' then y<= "11" ; valid <= '1';
    elsif a(2)= '1' then y<= "10" ;        //
    elsif a(1) = '1'  //   <= "01" ;       //
    elsif a(0) = "1"  //   y<= "00",       //      ;
    else  y <= "0" ; valid <= '0';
    end if ;
    ad process;

  end architecture;
```

Ex. State machine design




```
library ieee;
use ieee.std_logic_1164.all;

entity sb_machin is
  port (clk, rsnt : in  std_logic;
        inpt1, inpt2 : in  std_logic;
        outpt        : out std_logic);
end;

architecture simple of sb_machin is
  Type state_type IS (State_A, State_B, State_C);
  SIGNAL state : STATE_TYPE;
BEGIN
    process (rsnt, clk)
    if reset = '1' then state <= state_A;
    elsif (rising_edge(clk)) then
```

```
CASE state IS
  WHEN state_A =>
    IF input1 = '0' THEN state <= state_B;
    ELSE state <= state_C;
    END IF;

  when state_B =>
       state <= state_C;

  when state_C =>
    if input2 = '1' then state <= state_A;
    end if;

  when others => state <= state_A;
  END CASE;
END IF;
END PROCESS;

with state select
  output <= '0' when state_A,
            '1' when state_B,
            '0' when state_C;
end simple;
```