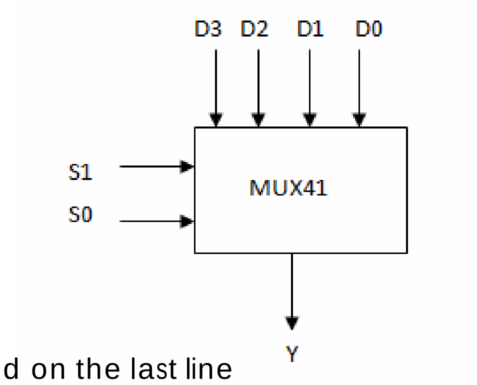
VHDL Tutorial

<u>Behavioral VHDL</u>

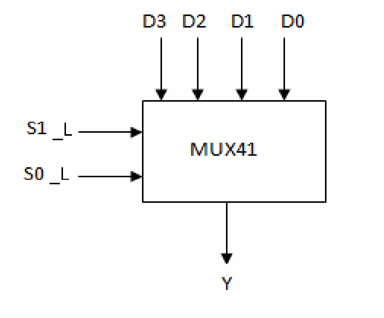
4 to 1 Mux

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

entity MUX41 is
port            --define inputs and outputs
      (
                S1  :  inbit;      -- input S1
                S0  :  inbit;
                D3  :  in          bit;
                D2  :  in          bit;
                D1  :  in          bit;
                D0  :  in          bit;
                Y   :  out  bit -- output Y, note: NO ‘;’ used on the last line
      );
end MUX41;

architecture logic of MUX41 is      -- Note MUX41 is the same as entity name above
begin
        Y       <=      (D0 and (not S1) and (not S0)) or
                        (D1 and (not S1) and      S0 ) or
                        (D2 and      S1  and (not S0)) or
                        (D3 and      S1  and      S0 ) ;
end logic;      -- Note matching names ‘logic’
```



4 to 1 Mux (S1 and S0 active low / Mixed Logic)

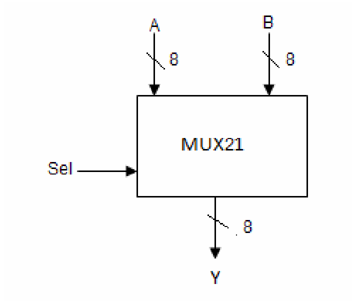
```
library ieee;
use ieee.std_logic_1164.all;

entity MUX41 is
port
      (
        S1_L, S0_L, D3, D2, D1, D0  :  in       std_logic;
        --std_logic same as bit, multiple inputs of the same
        type can be defined on the same line separated by commas

        Y   :  out  std_logic
      );
end MUX41;
```



```
architecture logic of MUX41 is

signal S1, S0 : std_logic;
--define signals. Signals are like temp variables which are not defined in entity but are
needed for the behaviour
begin

        S1 <= not S1_L;                 --relating S1 to S1_L;
        S0 <= not S0_L;

        -- No _L in equation below since logic equation does not change

        Y       <=      (D0 and (not S1) and (not S0)) or
                        (D1 and (not S1) and     S0 ) or
                        (D2 and      S1  and (not S0)) or
                        (D3 and      S1  and     S0 ) ;
end logic;
```

2 to 1 Mux (using IF/ELSE)

```
library ieee;
use ieee.std_logic_1164.all;
entity MUX2to1 is port(
      A, B:     in      std_logic_vector(7 downto 0);
      Sel:      in      std_logic;
      Y:        out     std_logic_vector(7 downto 0));
end MUX2to1;
```

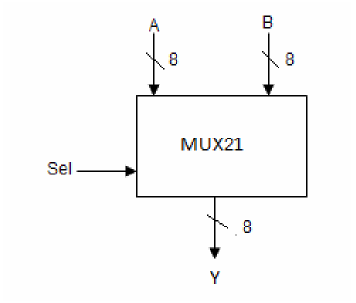


```
architecture behavior of MUX2to1 is
begin
      process (Sel, A, B) -- rerun process if any changes, sensitivity list, all inputs
      begin
        if (Sel = '1') then
          Y <= B;
        else
          Y <= A;
        end if; -- note that *end if* is two words
      end process;
end behavior;
```

Note:

- Std_logic_vector used to define a signal of more than 1 bit. In this case A, B and Y are all 8 bits and can be referred to as a vector or as individual components such as A(7), A(6),.. etc

- Process(Sel, A, B) is the sensitivity list. ALL inputs must be listed here for proper functionality
- If you had more than one case then use elsif (condition) then ← note that elsif is one word
- Sel is 1 bit so the syntax is *if (Sel = '1') then*. In the case that Sel were more than 1 bit, double quotes are to be used, for example *if (Sel = "0010") then*.
- Note the use of "end if" and "end process".

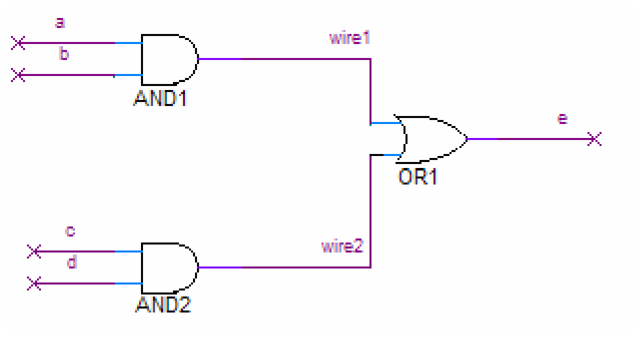


```
2 to 1 Mux (using Case)
ibrary ieee;
use ieee.std_logic_1164.all;
entity MUX2to1 is port(
        A, B:     in      std_logic_vector(7 downto 0);
        Sel:      in      std_logic;
        Y:        out     std_logic_vector(7 downto 0));
end MUX2to1;

architecture behavior of MUX2to1 is
begin
        process (Sel, A, B) -- rerun process if any changes, sensitivity list, all inputs
        begin
           case Sel is
                        when '0' =>
                                Y <= A;
                        when '1' =>
                                Y <=B;
                end case;
        end process;
end behavior;
```

Structural VHDL

Structural VHDL uses component description and connection descriptions (i.e. how the components are connected to each other).

For the following example, assume that a VHDL component for an AND gate (called “and”) and a component for the OR gate (called “or”) has already been developed using any of the behavioral methods described above.



```
ibrary ieee;
use ieee.std_logic_1164.all;
use work.all;

entity comb_logic is
   port(     a,b,c,d : in std_logic;
              e: out std_logic
   );
end comb_logic;

architecture behv of comb_logic is

--Assume you have an "AND" and an "OR" defined first
--define similar to other examples

component and is
port (a,b : in std_logic;
        output : out std_logic
        );
end component;

component or is
port (a,b : in std_logic;
        output : out std_logic
        );
end component;

--structural definition of e = ab + cd

signal wire1, wire2 : std_logic;
begin
        AND1: and port map ( a, b, wire1 );
        AND2: and port map ( c, d, wire2 );
        OR1: or port map ( wire1, wire2, e);
end behv;
```

Note:

- The port statement for entity comb_logic contain a,b,c,d, and e.
- Components are instantiated by including the entity description as follows:

```
component and is  --declaring an “and” component
port (a,b : in std_logic;
      output : out std_logic
      );
end component;
```

- Ports a, b, c, d, and e are defined but we need ‘temp’ signals (which are called wire1 and wire2 in this case) to connect the components defined above.
- AND1 is a component of type “and”. The port map maps signals to the port of the “and” component. AND2 is another instance of the “and” component, while OR1 is an instance of the previously defined “or” component.
- The port map entries have to correspond to the component entity ports. For example component “or” has a port definition

```
    Port (a,b : in std_logic;
output : out std_logic
);
```

So the port map of OR1 is (wire1, wire2, e) where wire1, and wire2 correspond to the two inputs of the “or” gate called a and b, and e is mapped onto the output of the “or” component. When doing port maps order matters since there is a one-to-one correspondence between the port declaration and port map. If you do not want to worry about order, you can use explicit definition as follows:

```
OR1: or port map ( a<=wire1, b<=wire2, output<=e);
```

Some examples were taken from the course website (although they may have been modified a little). For more examples see the course website (Examples → VHDL Examples). Follow the tutorial on Creating Graphical Components found in either Examples → VHDL Examples or Software/Docs → Quartus to include your VHDL components in your design, compile and simulate.