VHDL 2 – Combinational Logic Circuits

Reference: Roth/John Text: Chapter 2

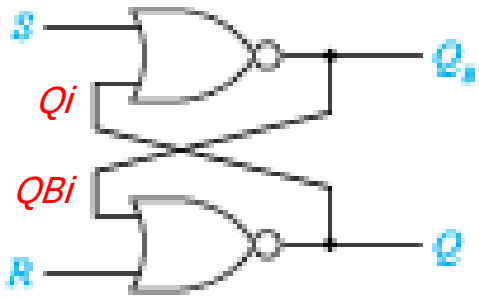
Combinational logic

```
-- Behavior can be specified as concurrent signal assignments
-- These model concurrent operation of hardware elements
entity Gates is
    port (a, b,c: in   STD_LOGIC;
          d:       out STD_LOGIC);
end Gates;
architecture behavior of Gates is
     signal e: STD_LOGIC;
begin
          -- concurrent signal assignment statements
          e <= (a and b) xor (not c);  -- synthesize gate-level ckt
          d <= a nor b and (not e);    -- in target technology
end;
```

Example: SR latch (logic equations)

```
entity SRlatch is
    port (S,R:     in std_logic;      --latch inputs
          Q,QB: out std_logic); --latch outputs
end SRlatch;

architecture eqns of SRlatch is
  signal Qi,QBi: std_logic; -- internal signals
begin
  QBi <= S nor Qi;    -- Incorrect would be: QB <= S nor Q;
  Qi <= R nor QBi;    -- Incorrect would be: Q  <= R nor QB;
  Q <= Qi;        --drive output Q with internal Qi
  QB <= QBi;   --drive output QB with internal QBi
end;
```

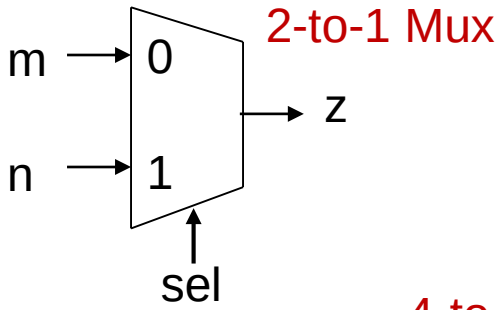


Cannot "reference" output ports.

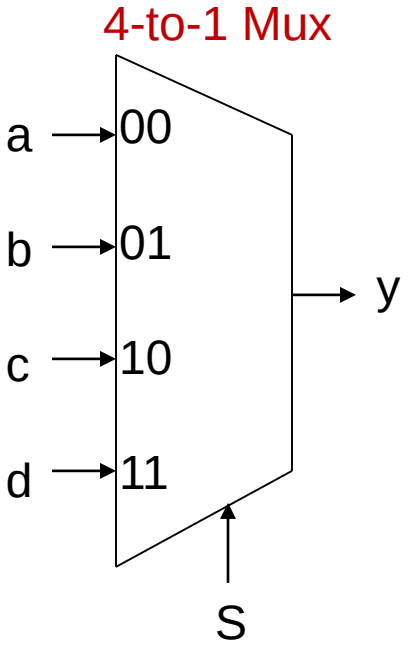
Conditional signal assignment (form 1)

z <= m when sel = '0' else n;

True/False conditions



y <= a when (S="00") else
b when (S="01") else
c when (S="10") else
d;

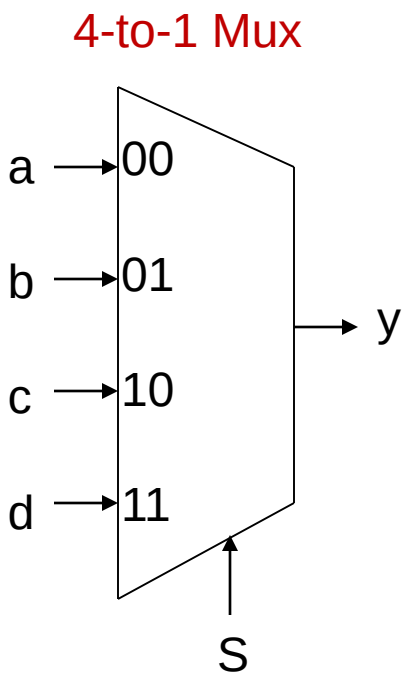


Condition can be any Boolean expression

y <= a when (F='1') and (G='0') …

Conditional signal assignment (form 2)

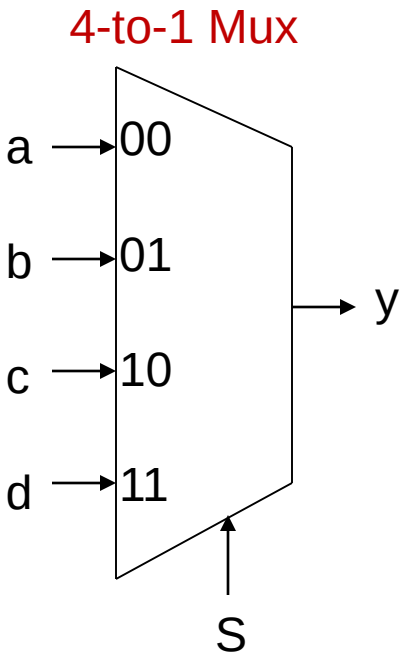
```
-- One signal (S in this case) selects the result
    signal a,b,c,d,y: std_logic;
    signal S: std_logic_vector(0 to 1);
begin
    with S select
        y <=   a when "00",
               b when "01",
               c when "10",
               d when "11";
--Alternative "default" *:
               d when others;
```



* "std_logic" values can be other than '0' and '1'

32-bit-wide 4-to-1 multiplexer

```
signal a,b,c,d,y: std_logic_vector(0 to 31);
signal S: std_logic_vector(0 to 1);
begin
  with S select
      y <=   a when "00",
             b when "01",
             c when "10",
             d when "11";
```



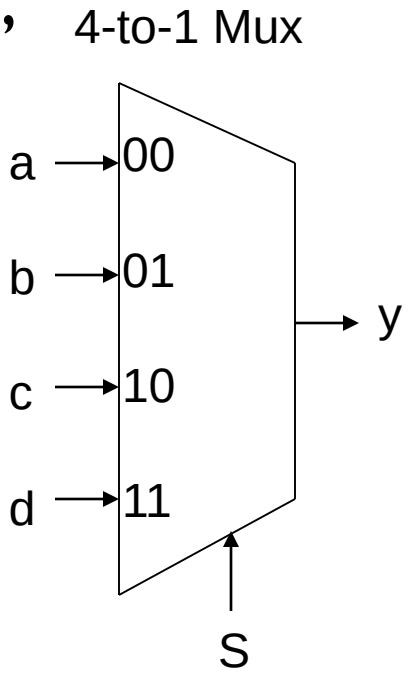
--y, a,b,c,d can be **any type**, as long as they match

32-bit-wide 4-to-1 multiplexer

-- Delays can be specified if desired

```
signal a,b,c,d,y: std_logic_vector(0 to 31);
   signal S: std_logic_vector(0 to 1);
begin
   with S select
        y <=   a after 1 ns when "00",
               b after 2 ns when "01",
               c after 1 ns when "10",
               d            when "11";
```

Optional non-delta delays for each option



a->y delay is 1ns, b->y delay is 2ns, c->y delay is 1ns, **d->y delay is δ**

Truth table model as a conditional assignment

- Conditional assignment can model the truth table of a switching function (without deriving logic equations)

S

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

```
signal S: std_logic_vector(1 downto 0);
begin
    S <= A & B;        -- S(1)=A, S(0)=B
    with S select  -- 4 options for S
      Y <= '0' when "00",
           '1' when "01",
           '1' when "10",
           '0' when "11",
           'X' when others;
```

& is the concatenate operator, merging scalars/vectors into larger vectors

Example: full adder truth table

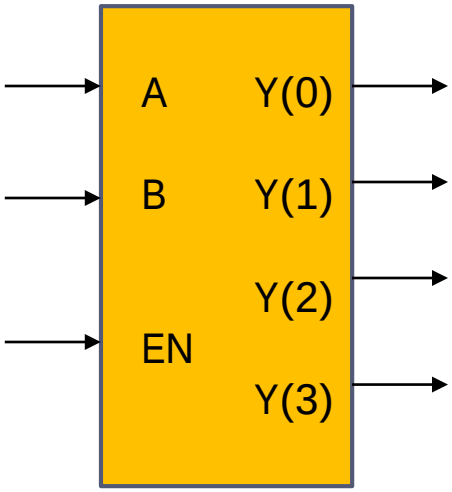
```
ADDin <= A & B & Cin;  --ADDin is a 3-bit vector
S     <= ADDout(0);    --Sum output (ADDout is a 2-bit vector)
Cout <= ADDout(1);     --Carry output

with ADDin select
   ADDout <= "00" when "000",
             "01" when "001",
             "01" when "010",
             "10" when "011",
             "01" when "100",
             "10" when "101",
             "10" when "110",
             "11" when "111",
             "XX" when others;
```

ADDin			ADDout	
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Example: 2-to-4 decoder

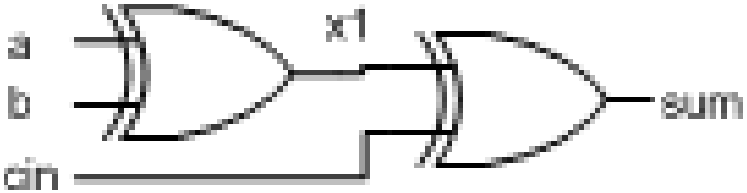
```
library ieee; use ieee.std_logic_1164.all;
entity decode2_4 is
   port (A,B,EN: in std_logic;
            Y: out std_logic_vector(3 downto 0));
end decode2_4;
architecture behavior of decode2_4 is
   signal D: std_logic_vector(2 downto 0);
begin
   D <= EN & B & A; -- vector of the three inputs
   with D select
       Y <= "0001" when "100",  --enabled,  BA=00
            "0010" when "101",  --enabled,  BA=01
            "0100" when "110",  --enabled,  BA=10
            "1000" when "111",  --enabled,  BA=11
            "0000" when others; --disabled (EN = 0)
end;
```



Structural model (no "behavior" specified)

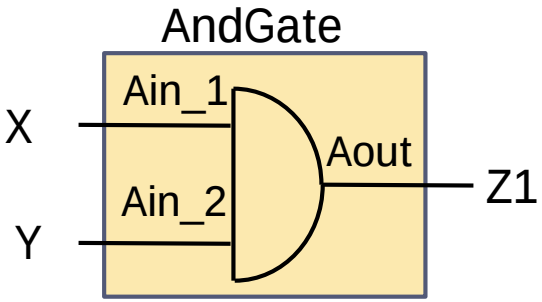
```
architecture structure of full_add1 is
   component xor        -- declare component to be used
         port (x,y: in std_logic;
                    z: out std_logic);
   end component;
   for all: xor use entity work.xor(eqns); -- if multiple arch's in lib.
   signal x1: std_logic;  -- signal internal to this component
begin  -- instantiate components with "map" of connections
   G1: xor port map (a, b, x1);        -- instantiate 1st xor gate
   G2: xor port map (x1, cin, sum);  -- instantiate 2nd xor gate
   …add circuit for carry output…
end;
```

library entity architecture



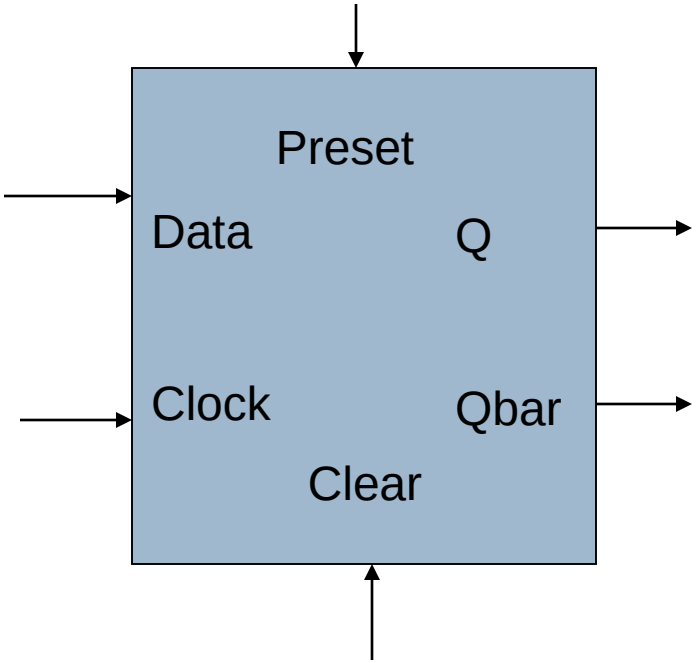
Associating signals with formal ports

```
  component AndGate
    port  (Ain_1,Ain_2 : in std_logic;   -- formal parameters
           Aout : out std_logic);
  end component;
begin
-- positional association of "actual" to "formal"
A1:AndGate port map (X,Y,Z1);
-- named association (usually improves readability)
A2:AndGate port map (Ain_2=>Y, Aout=>Z2, Ain_1=>X);
-- both (positional must begin from leftmost formal)
A3:AndGate port map (X, Aout => Z3, Ain_2 => Y);
```



Example: D flip-flop (equations model)

```
entity DFF is
   port (Preset: in std_logic;
         Clear: in std_logic;
         Clock: in std_logic;
         Data: in std_logic;
         Q: out std_logic;
         Qbar: out std_logic);
end DFF;
```

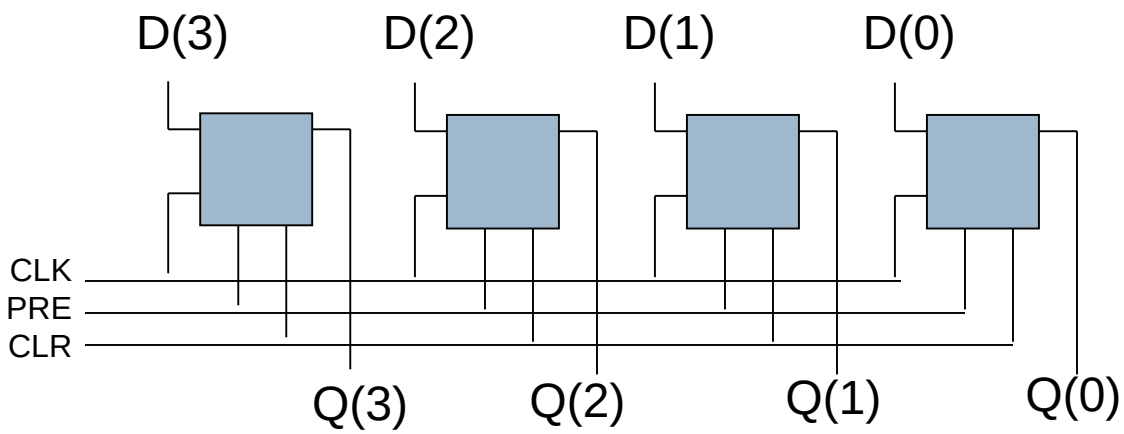


7474 D flip-flop equations

```
architecture eqns of DFF is
        signal A,B,C,D: std_logic;
        signal QInt, QBarInt: std_logic;
begin
        A <= not (Preset and D and B) after 1 ns;
        B <= not (A and Clear and Clock) after 1 ns;
        C <= not (B and Clock and D) after 1 ns;
        D <= not (C and Clear and Data) after 1 ns;
        Qint <= not (Preset and B and QbarInt) after 1 ns;
        QBarInt <= not (QInt and Clear and C) after 1 ns;
        Q <= QInt;          -- Can drive but not read "outs"
        QBar <= QBarInt;    -- Can read & drive "internals"
end;
```

4-bit Register (Structural Model)

```
entity Register4 is
  port (   D: in std_logic_vector(0 to 3);
           Q: out std_logic_vector(0 to 3);
           Clk: in std_logic;
           Clr: in std_logic;
           Pre: in std_logic);
end Register4;
```



Register Structure

```
architecture structure of Register4 is
    component DFF      -- declare library component to be used
      port (Preset: in std_logic;
            Clear: in std_logic;
            Clock: in std_logic;
            Data: in std_logic;
            Q: out std_logic;
            Qbar: out std_logic);
     end component;
     signal Qbar: std_logic_vector(0 to 3); -- dummy for unused FF Qbar outputs
begin
    -- Signals connect to ports in order listed above
    F3: DFF port map (Pre, Clr, Clk, D(3), Q(3), Qbar(3));
    F2: DFF port map (Pre, Clr, Clk, D(2), Q(2), Qbar(2));
    F1: DFF port map (Pre, Clr, Clk, D(1), Q(1), Qbar(1));
    F0: DFF port map (Pre, Clr, Clk, D(0), Q(0), Qbar(0));
end;
```

Register Structure (with open output)

```
architecture structure of Register4 is
   component DFF          -- declare library component to be used
      port (Preset: in std_logic;
            Clear: in std_logic;
            Clock: in std_logic;
            Data: in std_logic;
            Q: out std_logic;
            Qbar: out std_logic);
     end component;
begin
   -- Signals connect to ports in order listed above
   F3: DFF port map (Pre, Clr, Clk, D(3), Q(3), OPEN);
   F2: DFF port map (Pre, Clr, Clk, D(2), Q(2), OPEN);
   F1: DFF port map (Pre, Clr, Clk, D(1), Q(1), OPEN);
   F0: DFF port map (Pre, Clr, Clk, D(0), Q(0), OPEN);
end;
```

Keyword OPEN indicates an unconnected output

VHDL "Process" Construct

(Processes will be covered in more detail in "sequential circuit modeling")

```
[label:]  process (sensitivity list)
              declarations
          begin
              sequential statements
          end process;
```

- Process statements are executed *in sequence*
- Process statements are executed once at start of simulation
- Process halts at "end" until an event occurs on a signal in the "sensitivity list"
- Allows conventional programming language methods to describe circuit behavior

Modeling combinational logic as a process

```
-- All signals referenced in process must be in the sensitivity list.
entity And_Good is
    port (a, b: in std_logic; c: out std_logic);
end And_Good;

architecture Synthesis_Good of And_Good is
   begin
     process (a,b)      -- gate sensitive to events on signals a and/or b
     begin
         c <= a and b;  -- c updated (after delay on a or b  "events"
     end process;
    -- This process is equivalent to the simple signal assignment:
    --        c <= a and b;
end;
```

Bad example of combinational logic

```
-- This example produces unexpected results.
entity And_Bad is
    port (a, b: in std_logic; c: out std_logic);
end And_Bad;
architecture Synthesis_Bad of And_Bad is
   begin
     process (a)        -- sensitivity list should be (a, b)
     begin
         c <= a and b;  -- will not react to changes in b
    end process;
end Synthesis_Bad;
-- synthesis may generate a flip flop, triggered by signal a
```