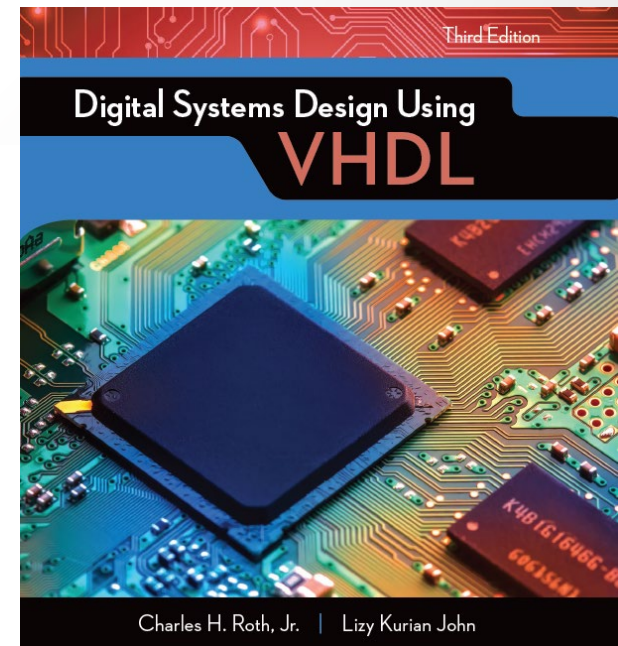


EE2000: Logic Circuit Design

VHDL 2

Acknowledgements: Dr. Wang Hong, Dr. Ray Cheung, Dr. Ricky Lau
Dr. Chang Chip Hong, Dr. Goh Wang Ling



Contents

- 8.1 Component
- 8.2 Instantiation
- 8.3 Conditional Signal Assignments
- 8.4 Selected Signal Assignments
- 8.5 Loops
- 8.6 Using Case statement
- 8.7 Using IF statement
- 8.8 Using Structural model
- 8.9 Enable signal
- 8.10 Encoder
- 8.11 Decimal-to-BCD encoder
- 8.12 BCD to 7 Segment
- 8.13 Demultiplexer

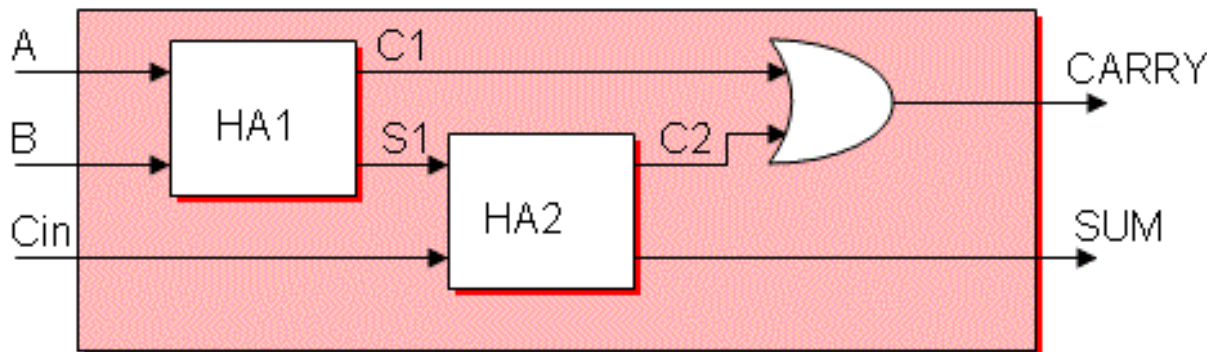
8.1 VHDL - Component

- **Structural modeling** is important for circuit/system design.
- A circuit/system may contain multiple **components** (sub-module/sub-circuits)

Example:

A full adder (FA) contains 2 half adders (HAs).

The half adder can be modeled by a **component**.



8.1 VHDL - Component

An architecture may contain multiple components and they must be declared first.

```
architecture [name] ...  
[signal]  
is
```

```
component XX  
...  
end component;
```

```
component YY  
...  
end component;
```

```
begin  
...  
end [name];
```

8.1 VHDL - Component

Differences between a **component** and an **entity** declaration:

- **Entity** declaration declares a circuit model containing one or multiple architectures.
- **Component** declaration declares a **virtual circuit template**, which must be **instantiated** to take effect during the design.
- **Port map** is required for **component instantiation** (discussed later).

8.1 VHDL - Component

Example: Full adder **entity**

- Create the component entity **halfadder**
- Create the module entity **fulladder**
- Determine the number of units (i.e. **halfadder** in this case) used in the design
- Define signals for inter-connections between **halfadder** (**components**)
- Provide each component a different name
- Then instantiates the declared component

8.1 VHDL - Component

Complete full adder VHDL code:

```
--sub module(half adder) entity declaration
entity halfadder is
Port ( a : in   STD_LOGIC;
        b : in   STD_LOGIC;
        sum : out  STD_LOGIC;
        carry : out  STD_LOGIC
      );
end halfadder;

architecture Behavioral of halfadder is
begin
  sum <= a xor b;
  carry <= a and b;
end Behavioral;
```

8.1 VHDL - Component

```
--top module(full adder) entity declaration
entity fulladder is
    port (a : in std_logic;
          b : in std_logic;
          cin : in std_logic;
          sum : out std_logic;
          carry : out std_logic
    );
end fulladder;
--top module architecture declaration.
architecture behavior of fulladder is
--sub-module(half adder) is declared as a component before the
keyword "begin".
```

```
component halfadder
    port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
        carry : out std_logic
    );
end component;
```

8.1 VHDL - Component

```
--All the signals used inside this Architecture are declared here, which are not a part of the top module.
```

```
signal s1,c1,c2 : std_logic:='0';
```

```
begin
```

```
--Provide a different name for each half adder.
```

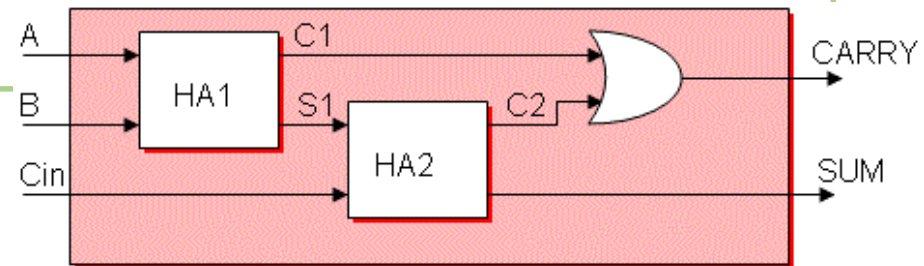
```
--instantiate and do port map for the half adders.
```

```
HA1 : halfadder port map (a,b,s1,c1);
```

```
HA2 : halfadder port map (s1,cin,sum,c2);
```

```
carry <= c1 or c2; --final carry calculation
```

```
end;
```



Two HAs are used to form a FA.

Internal signals **s1**, **c1**, **c2** are used to connect the two HAs.

8.2 VHDL - Instantiation

For Creating connections between components and ports.

3 steps in VHDL instantiation:

- 1) Label: identify a unique **instance** of component
- 2) Component type: select a targeted declared component
- 3) Port Map: Connect component to signals in an architecture

*Signals must be of the **same data type** for the connecting pins.*

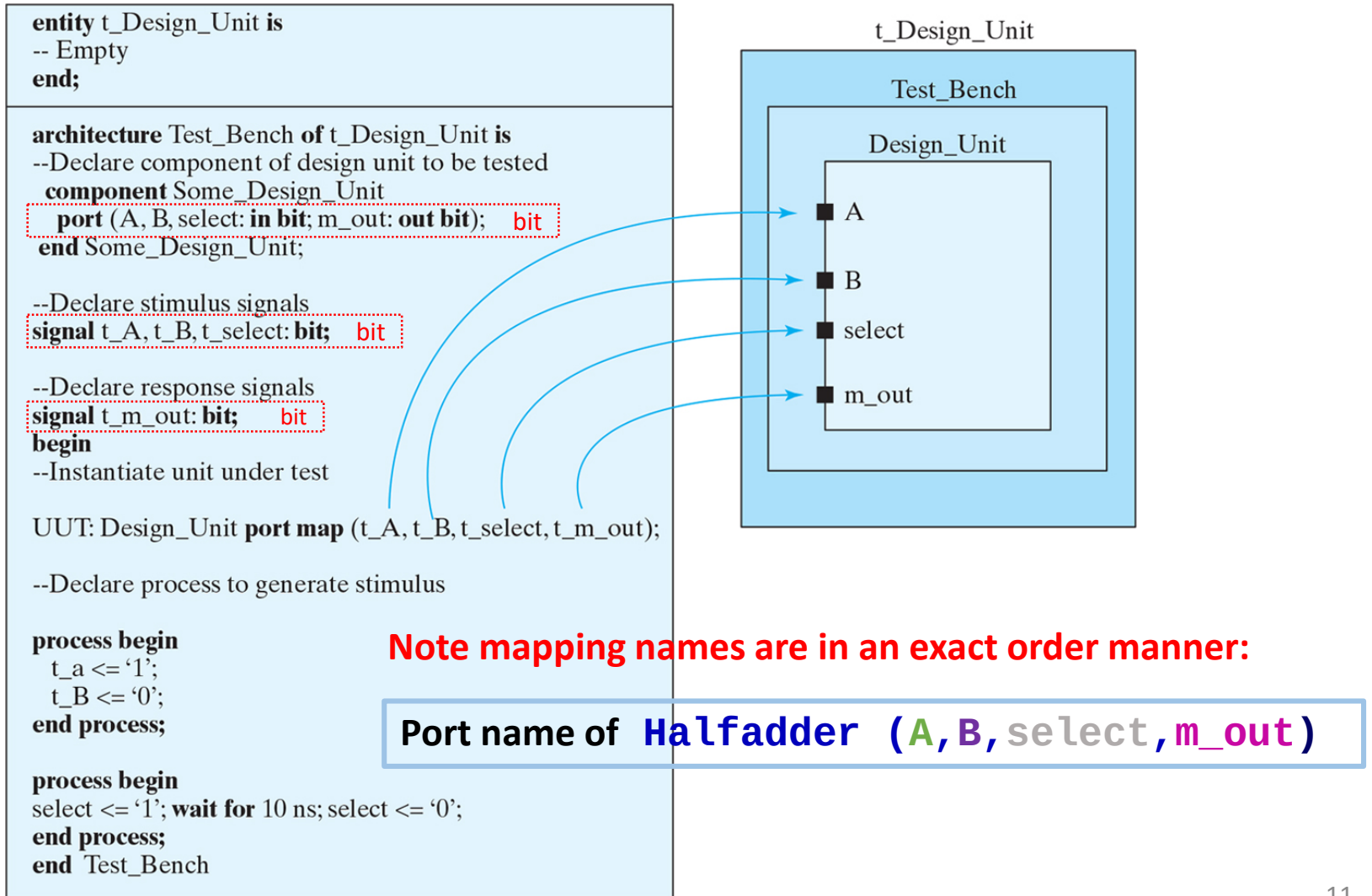
```
HA1 : halfadder port map (a, b, s1, c1);  
HA2 : halfadder port map (s1, cin, sum, c2);
```

↑
Label

↑
Component

↑
Port Map

8.2 VHDL – Instantiation: Port Map



8.2 VHDL – Instantiation: Port Map

```
entity halfadder is
Port ( a : in  STD_LOGIC;
      b : in  STD_LOGIC;
      sum : out STD_LOGIC;
      carry : out STD_LOGIC
);
end halfadder;
```

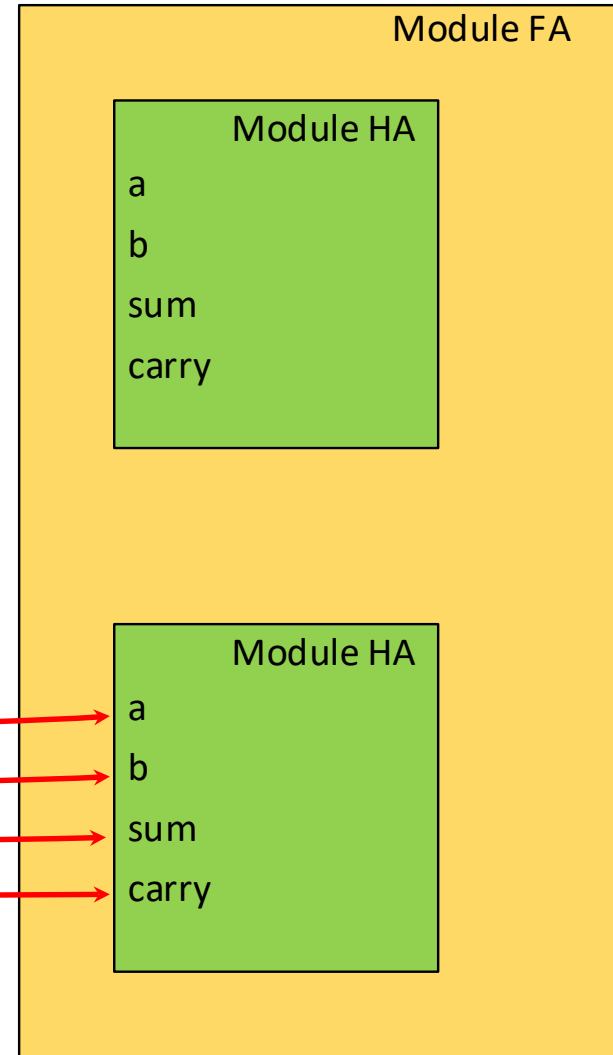
Port name of Halfadder (a, b, sum, carry)

```
begin

--Provide a different name for each half adder.
--instantiate and do port map for the half adders.

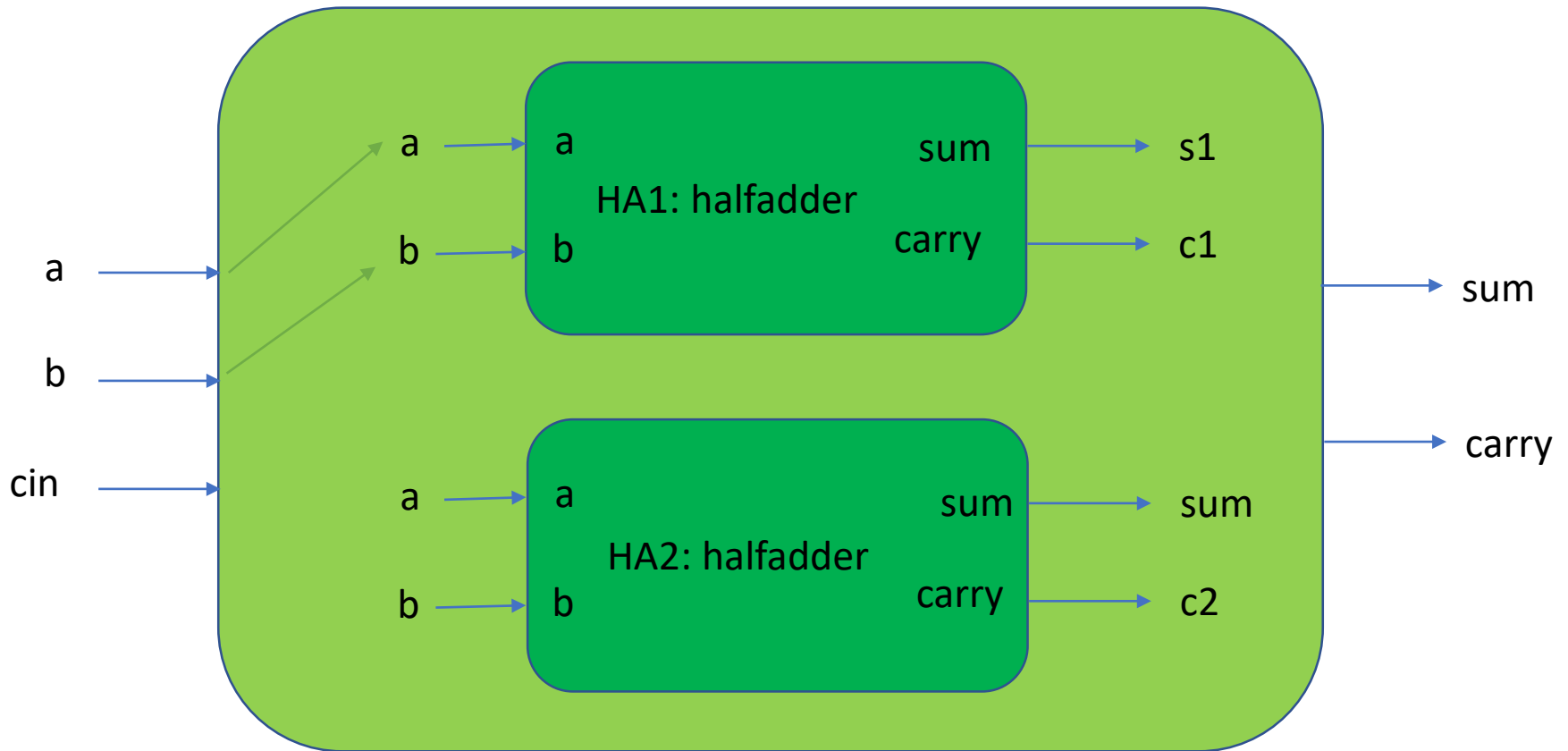
HA1 : halfadder port map (a,b,s1,c1);
HA2 : halfadder port map (s1, cin, sum, c2);
carry <= c1 or c2;  --final carry calculation

end;
```



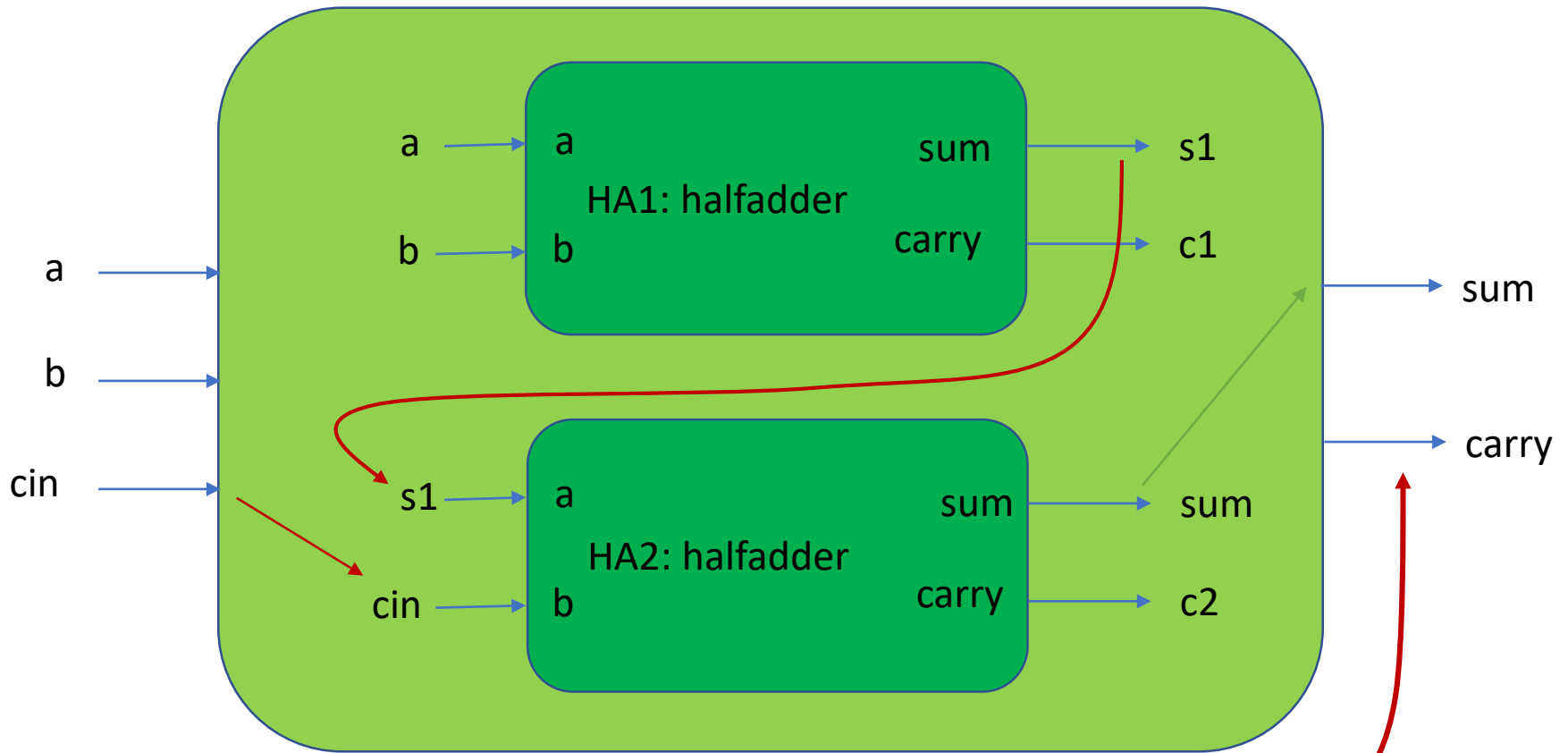
8.2 VHDL – Instantiation: Port Map

```
HA1 : halfadder port map (a,b,s1,c1);
```



8.2 VHDL – Instantiation: Port Map

```
HA2 : halfadder port map (s1,cin,sum,c2);
```



carry is generate by this VHDL statement

```
carry <= c1 or c2;
```

8.3 VHDL Conditional Signal Assignments

Form of a conditional signal assignment statement:

```
signal_name <= expression1 when condition1  
                else expression2 when condition2  
                [else expressionN];
```

- Target output can be a port or a signal.
- Only handle one target output.
- Less flexible than using IF/ELSE/ELSIF statement.

8.3 VHDL Conditional Signal Assignments

Example:

```
Y <= '1' when a = '0' else  
    '1' when b = '0' else  
    '1' when c = '0' else  
    '0';
```

```
Y <= (C and B) when a = '0'      else  
    '0'                when b = '1'      else  
    '1'                when c = (d or e)  else  
    d;
```

```
Z <= "00" when D > "0010" and D <= "0110" else  
    "01" when D = "0101"  
else  
    "10" when D > "1000" and D < "1100"  
else  
    "11";
```

8.4 VHDL Selected Signal Assignments

A selected signal assignment statement has the form:

```
with expression_s select  
    signal_s <= expression1 [after delay-time] when choice1,  
                expression2 [after delay-time] when choice2,  
                ...  
                expression_n [after delay-time] when others;
```

- Target output can be a port or a signal.
- Can only handle one target output.
- Each line ends with ‘,’ and the last line with ‘;’
- “**when others**” is used to handle the default case, and also the don’t case cases.

8.4 VHDL Selected Signal Assignments

Example:

```
with d select
Y <=   '0' when "000",
      '1' when "001",
      '1' when "010",
      '0' when "011",
      '1' when "100",
      '0' when "101",
      '1' when "110",
      '1' when "111",
      NULL when others;
```

```
with d select
Y <=   '0' when "000",
      '0' when "011",
      '0' when "101",
      '1' when others;
```

8.5 VHDL Loops

Activity occurring in a repetitive way.

Statements are sequential.

Kinds of loop statements: **for**, **while**

Infinite loop:

General form:

```
[loop-label:] loop  
    sequential statements  
end loop [loop-label];
```

Can be terminated using exit statements:

```
exit; or exit when condition;
```

8.5 VHDL Loops

for loop:

General form:

```
[loop-label:] for loop-index in range loop  
    sequential statements  
end loop [loop-label];
```

Loop-index is incremented at the end of each loop.
Continues for every value in the range.

while loop:

General form:

```
[loop-label:] while condition loop  
    sequential statements  
end loop [loop-label];
```

Condition is tested at the beginning of each loop.
Loop is terminated if condition is false.

8.6 VHDL Decoder – A) Using Case statement

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder is
port(
    A : in STD_LOGIC_VECTOR(1 downto 0);
    X : out STD_LOGIC_VECTOR(3 downto 0);
);
end decoder;

```

```

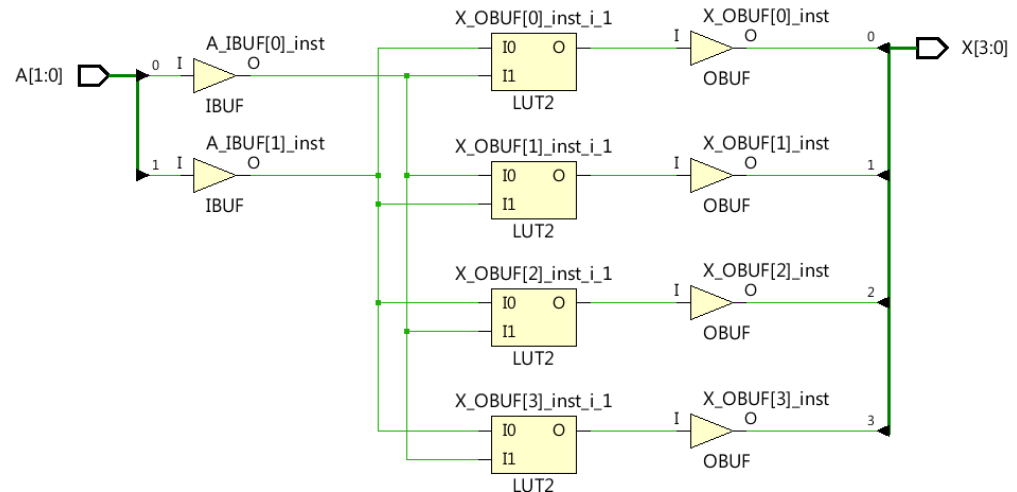
architecture Behavioral of decoder is
begin

```

```

    process(a)
    begin
        case A is
            when "00" => X <= "0001";
            when "01" => X <= "0010";
            when "10" => X <= "0100";
            when "11" => X <= "1000";
        end case;
    end process;
end Behavioral;

```

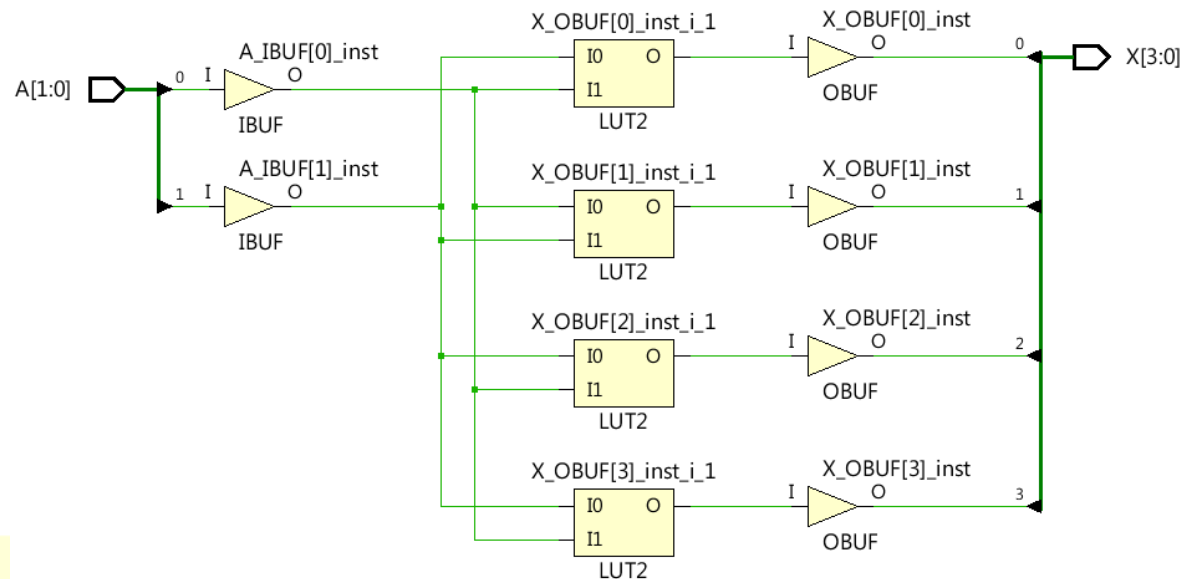


8.7 VHDL Decoder – B) Using IF statement

```
entity decoder is
port(
    A : in STD_LOGIC_VECTOR(1 downto 0);
    X : out STD_LOGIC_VECTOR(3 downto 0)
);
end decoder;
```

```
architecture Behavioral of decoder is
begin
```

```
    process(a)
    begin
        if (A="00") then
            X <= "0001";
        elsif (A="01") then
            X <= "0010";
        elsif (A="10") then
            X <= "0100";
        else
            X <= "1000";
        end if;
    end process;
end Behavioral;
```

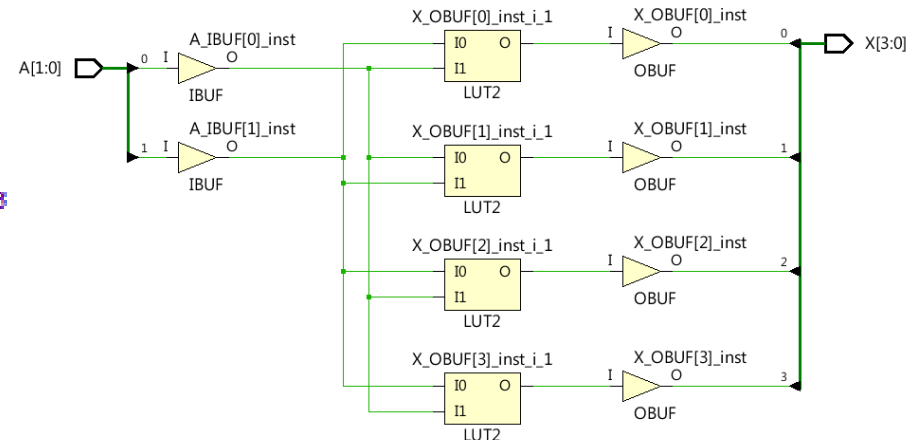


8.8 VHDL Decoder – C) Using Structural model

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder is
port(
    A : in STD_LOGIC_VECTOR(1 downto 0);
    X : out STD_LOGIC_VECTOR(3 downto 0)
);
end decoder;
```

```
architecture Structral of decoder is
begin
    X(0) <= not A(0) and not A(1);
    X(1) <= not A(0) and A(1);
    X(2) <= A(0) and not A(1);
    X(3) <= A(0) and A(1);
end Structral;
```



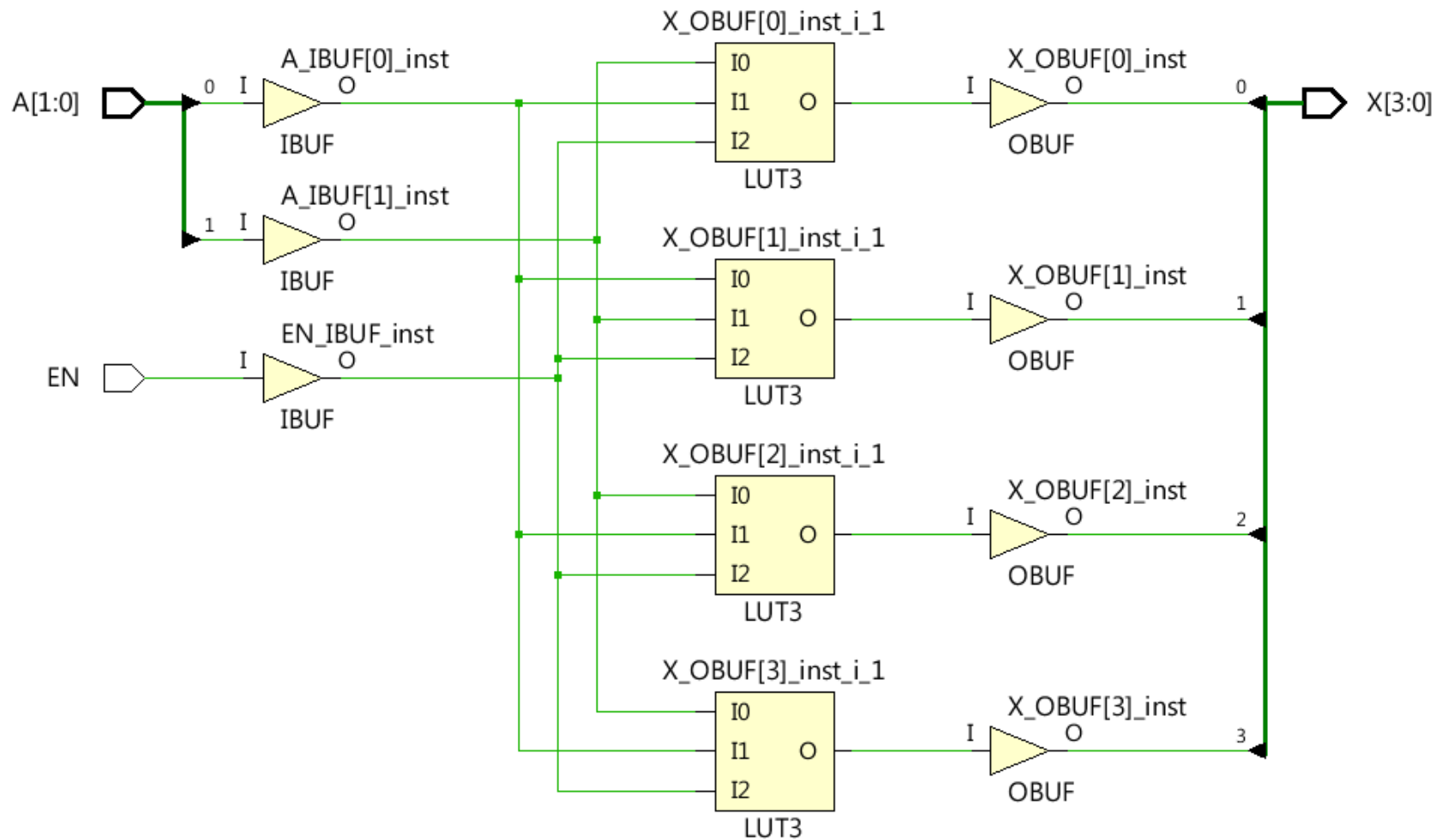
8.9 VHDL Decoder – with Enable signal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decode_2to4_top is
    Port ( A   : in  STD_LOGIC_VECTOR (1 downto 0);    -- 2-bit input
          X   : out STD_LOGIC_VECTOR (3 downto 0);    -- 4-bit output
          EN  : in  STD_LOGIC);                      -- enable input
end decode_2to4_top;

architecture Behavioral of decode_2to4_top is
begin
    process (A, EN)
    begin
        X <= "1111";    -- default output value
        if (EN = '1') then -- active high enable pin
            case A is
                when "00" => X(0) <= '0';
                when "01" => X(1) <= '0';
                when "10" => X(2) <= '0';
                when "11" => X(3) <= '0';
                when others => X <= "1111";
            end case;
        end if;
    end process;
end Behavioral;
```

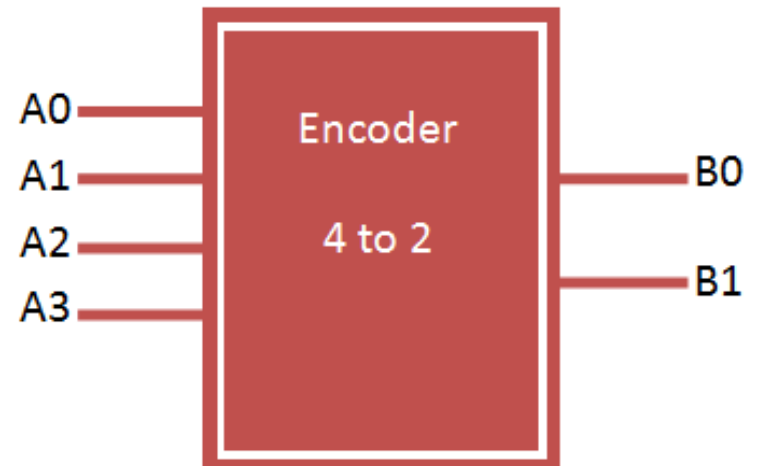
8.9 VHDL Decoder



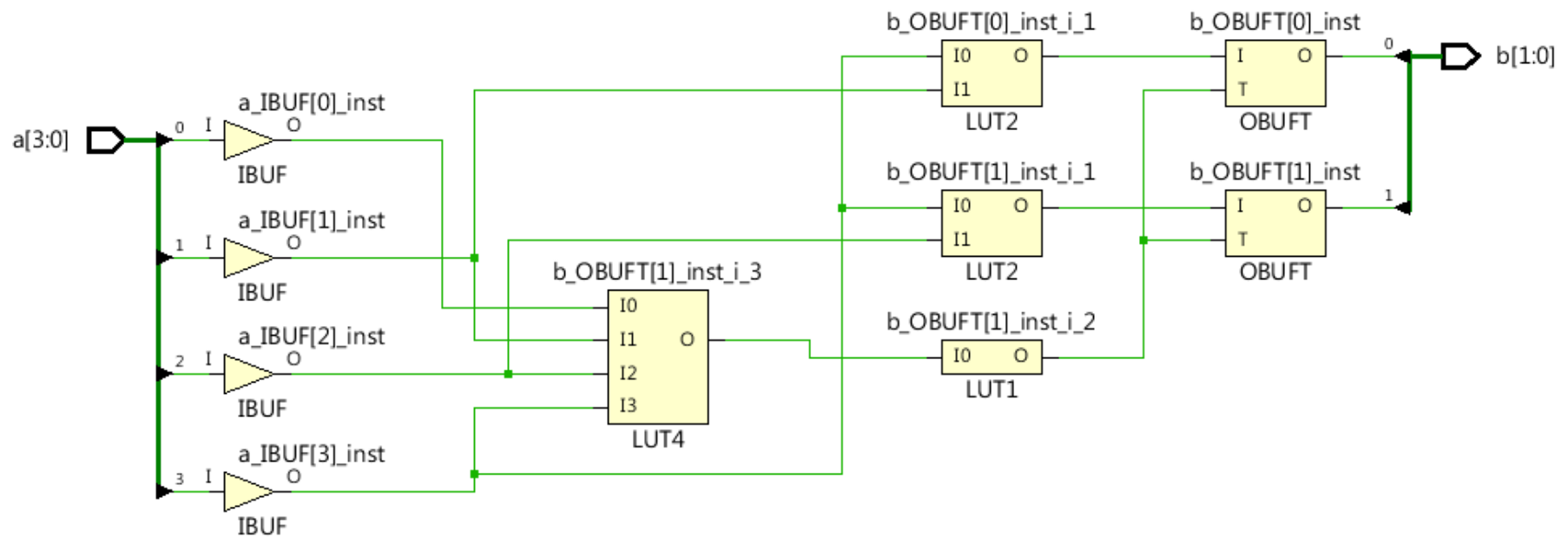
8.10 VHDL Encoder

```
entity encoder is
  port(
    a : in STD_LOGIC_VECTOR(3 downto 0);
    b : out STD_LOGIC_VECTOR(1 downto 0)
  );
end encoder;
```

```
architecture Behavioral of encoder is
begin
  process(a)
  begin
    case a is
      when "1000" => b <= "00";
      when "0100" => b <= "01";
      when "0010" => b <= "10";
      when "0001" => b <= "11";
      when others => b <= "ZZ";
    end case;
  end process;
end Behavioral;
```



8.10 VHDL Encoder – Schematic Diagram



Input 4 bits, output 2 bits

8.10 VHDL Encoder – Simulation

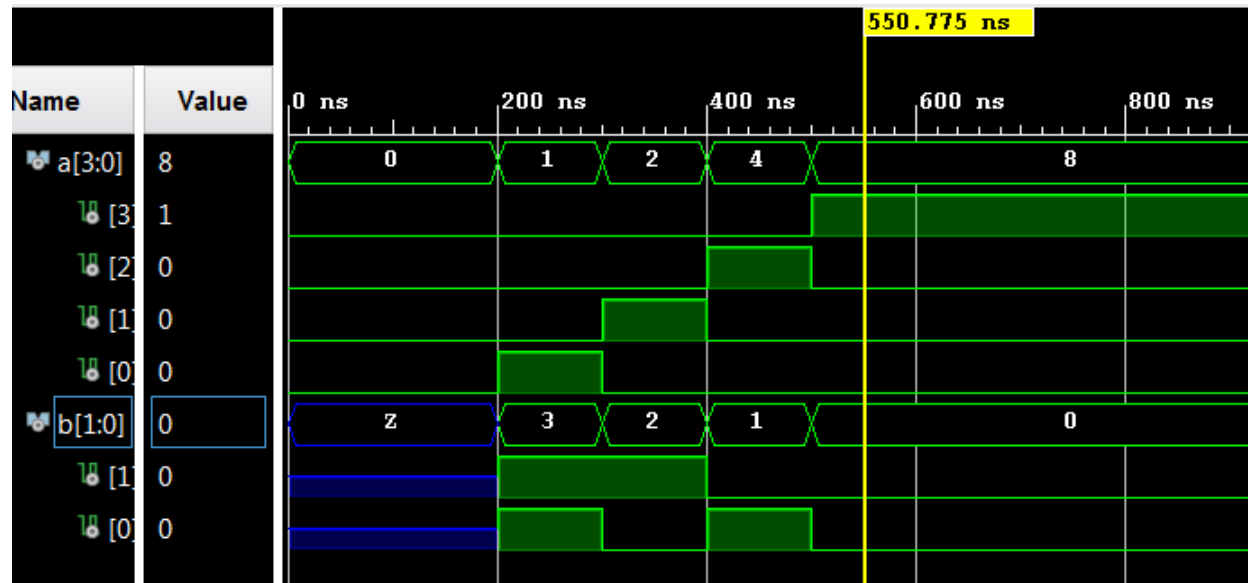
```

ENTITY tb_encoder IS
END tb_encoder;

ARCHITECTURE behavior OF tb_encoder IS
COMPONENT encoder
    PORT (
        a : IN std_logic_vector(3 downto 0);
        b : OUT std_logic_vector(1 downto 0)
    );
END COMPONENT;

signal a : std_logic_vector(3 downto 0) := (others => '0');
signal b : std_logic_vector(1 downto 0);
BEGIN
uut: encoder PORT MAP (a => a, b => b);
stim_proc: process
begin
    -- hold reset state for 100 ns.
    wait for 100 ns;
    a <= "0000";
    wait for 100 ns;
    a <= "0001";
    wait for 100 ns;
    a <= "0010";
    wait for 100 ns;
    a <= "0100";
    wait for 100 ns;
    a <= "1000";
    wait;
end process;
END;

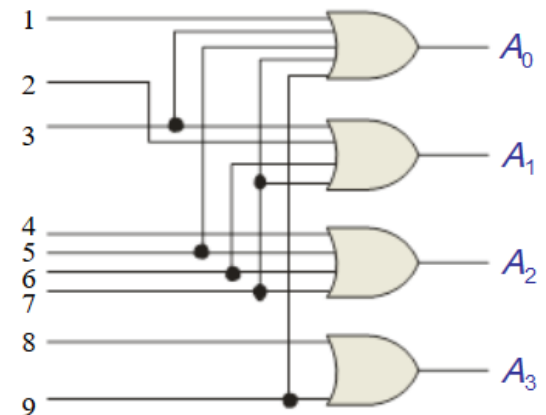
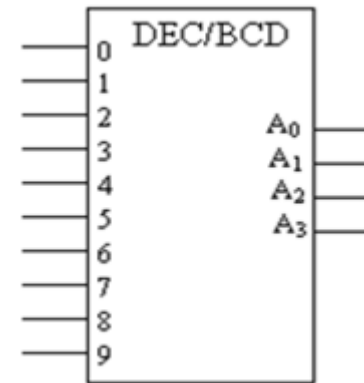
```



8.11 VHDL Encoder – Decimal-to-BCD encoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ENC3 is
  Port ( Q : in std_logic;
        R : in std_logic;
        S : in std_logic;
        T : in std_logic;
        U : in std_logic;
        V : in std_logic;
        W : in std_logic;
        X : in std_logic;
        Y : in std_logic;
        Z : in std_logic;
        OUT0 : out std_logic;
        OUT1 : out std_logic;
        OUT2 : out std_logic;
        OUT3 : out std_logic);
end ENC3;
architecture Behavioral of ENC3 is
begin
  process (Q,R,S,T,U,V,W,X,Y,Z)
  begin
    OUT0 <= R OR T OR V OR X OR Z;
    OUT1 <= S OR T OR W OR X;
    OUT2 <= U OR V OR W OR X;
    OUT3 <= Y OR Z;
  end process;
end Behavioral;
```

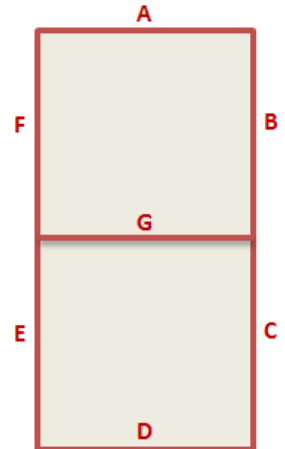
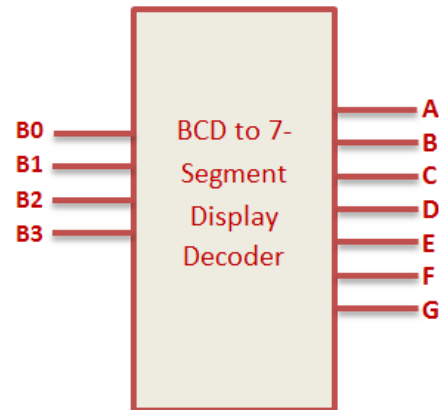


8.12 VHDL Encoder – BCD to 7 Segment

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bcd_7seg is
    Port ( B0,B1,B2,B3 : in STD_LOGIC;
           A,B,C,D,E,F,G : out STD_LOGIC);
end bcd_7seg;
```

```
architecture Behavioral of bcd_7seg is
begin
    A <= B0 OR B2 OR (B1 AND B3) OR (NOT B1 AND NOT B3);
    B <= (NOT B1) OR (NOT B2 AND NOT B3) OR (B2 AND B3);
    C <= B1 OR NOT B2 OR B3;
    D <= (NOT B1 AND NOT B3) OR (B2 AND NOT B3) OR (B1 AND NOT B2 AND B3) OR (NOT B1 AND B2) OR B0;
    E <= (NOT B1 AND NOT B3) OR (B2 AND NOT B3);
    F <= B0 OR (NOT B2 AND NOT B3) OR (B1 AND NOT B2) OR (B1 AND NOT B3);
    G <= B0 OR (B1 AND NOT B2) OR ( NOT B1 AND B2) OR (B2 AND NOT B3);
end Behavioral;
```



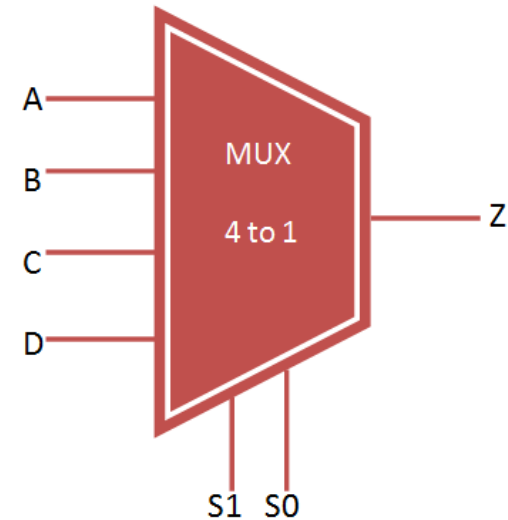
Build the Truth table first.

8.13 VHDL Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux_4to1 is
    port(
        A,B,C,D : in STD_LOGIC;
        S0,S1: in STD_LOGIC;
        Z: out STD_LOGIC
    );
end mux_4to1;

architecture bhv of mux_4to1 is
begin
    process (A,B,C,D,S0,S1) is
    begin
        if (S0 = '0' and S1 = '0') then
            Z <= A;
        elsif (S0 = '1' and S1 = '0') then
            Z <= B;
        elsif (S0 = '0' and S1 = '1') then
            Z <= C;
        else
            Z <= D;
        end if;
    end process;
end bhv;
```



Input		output
S1	S0	Z
0	0	A
0	1	B
1	0	C
1	1	D

```

ENTITY tb_mux IS
END tb_mux;

ARCHITECTURE behavior OF tb_mux IS
    COMPONENT mux_4to1
    PORT(
        A : IN  std_logic;
        B : IN  std_logic;
        C : IN  std_logic;
        D : IN  std_logic;
        S0 : IN  std_logic;
        S1 : IN  std_logic;
        Z : OUT  std_logic
    );
END COMPONENT;

signal A, B, C, D, S0, S1 : std_logic := '0';
signal Z : std_logic;

BEGIN

    uut: mux_4to1 PORT MAP (A => A,B => B,C => C,D => D,S0 => S0,S1 => S1,Z => Z);
    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        wait for 100 ns;
        A <= '1';
        B <= '0';
        C <= '1';
        D <= '0';
        S0 <= '0'; S1 <= '0';
        wait for 100 ns;
        S0 <= '1'; S1 <= '0';
        wait for 100 ns;
        S0 <= '0'; S1 <= '1';
        wait for 100 ns;
        S0 <= '0'; S1 <= '1';
        wait for 100 ns;
        end process;
END;

```

Name	Value
A	1
B	0
C	1
D	0
S0	0
S1	1
Z	1

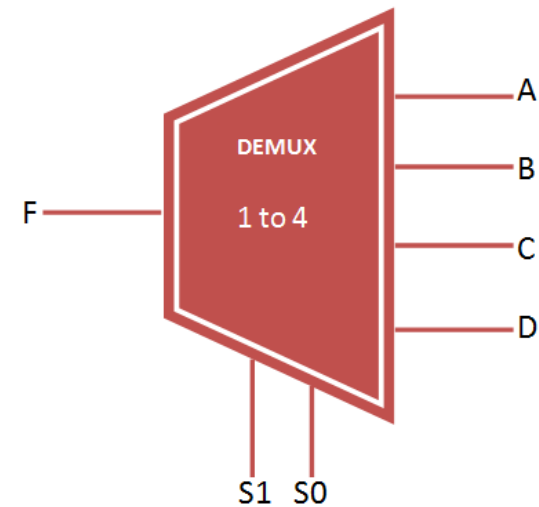
Timing diagram showing eight digital signals (1-8) over 800 ns. The signals are represented by green bars on a black background. The time axis is marked at 0 ns, 200 ns, 400 ns, 600 ns, and 800 ns.

Signal	High Interval(s)
1	0 ns to 800 ns
2	0 ns to 800 ns
3	0 ns to 100 ns
4	100 ns to 800 ns
5	200 ns to 350 ns
6	350 ns to 700 ns
7	100 ns to 250 ns, 400 ns to 700 ns
8	100 ns to 250 ns, 400 ns to 700 ns

8.13 VHDL Demultiplexer

```
entity demux_1to4 is
    port(
        F : in STD_LOGIC;
        S0,S1: in STD_LOGIC;
        A,B,C,D: out STD_LOGIC
    );
end demux_1to4;

architecture bhv of demux_1to4 is
begin
    process (F,S0,S1) is
    begin
        if (S0 ='0' and S1 = '0') then
            A <= F;
        elsif (S0 ='1' and S1 = '0') then
            B <= F;
        elsif (S0 ='0' and S1 = '1') then
            C <= F;
        else
            D <= F;
        end if;
    end process;
end bhv;
```



Input	Selection line		Output			
F	S1	S0	D	C	B	A
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0