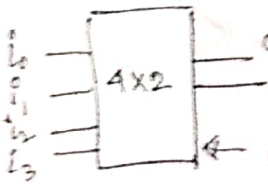
7/01/20

Hardware Description Language (HDL)

- VHDL
  - ISE Design Suite
- Verilog.

FPGA → Filled Programmable Gate Array.

Encoder in VHDL:



← entity - the block diagram

↑ defines inputs & outputs.

```
entity fourtotwo encoder is port
        ( i: in bit_vector (3 down to 0);
          o: out std_logic_vector(1 down to 0));
end fourtotwo encoder;
```

Data types

- bit_vector() {0,1}
- std_logic() → not only 0 and 1, don't cares, etc.
- std_logic_vector() → 9 values
- boolean - true or false

Truth table

i_3	i_2	i_1	i_0	O_0	O_1
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

if 0 0 0 0. So in high impedance state. (Z)

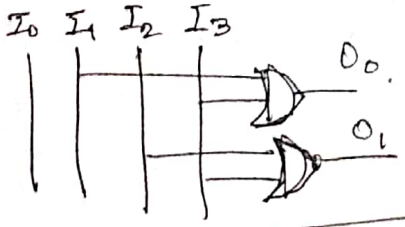
Now we have to define the architecture.

(1) Data flow modelling. (Boolean eqn)
(2) Structural modelling. (Smaller components to larger)
(3) Behavioral modelling.

→ from circuit diagram.

$O_0 = \bar{I_1} + \bar{I_3}$

$O_1 = \bar{I_2} + \bar{I_3}$



Structural modelling

```
architecture of fourtotwoencoder is.
begin.
    o(0) <= i(1) or i(3);
    o(1) <= i(2) or i(3);
end behavioral;
```

- Behavioral modelling. (sequential & concurrent stmts).

1) Concurrent

```
begin
with i select   o <=
        "00" when "0001",
        "01" when "0010",
        "10" when "0100",
        "11" when "1000"
        "ZZ" when others;
end behavioral.
```

0 0 0 0 0 0 0 1	0 0 0
0 0 0 0 0 0 1 0	0 0 1
0 0 0 0 0 1 0 0	0 1 0

2) Sequential stmt → (Process stmt)

```
begin
Process(i)
    case i is
    when "0001" => o <= "00";
    when "0010" => o <= "01";
    when "0100" => o <= "10";
    when "1000" => o <= "11";
    when others => o <= "ZZ";
    end case;
end process;
end behavioral;
```

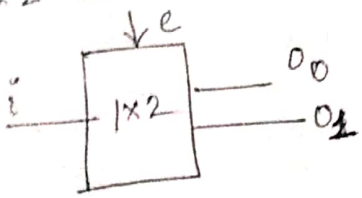
Assgn1: Design a 8×3 encoder. ins all 3.

Assgn2: Design a decimal to BCD encoder.

Lab Report: Name, TT, Block Diag, Structural design, Output.

Decoder:

1×2 decoder

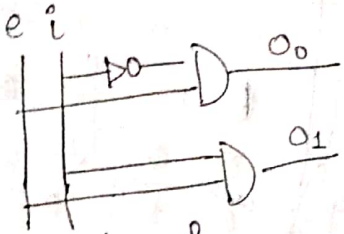


Truth table

e	i	O_0	O_1
0	X	0	0
1	0	0	1
1	1	1	0

$O_0 = e\bar{i}$

$O_1 = ei$



Structural.

```
O0 <= e and (not(i))
O1 <= e and i
```

```
with (e & i) select O <=
    "00" when "0_",
    "01" when "10",
    "10" when "11";
```

```
process (e, i)
begin
    if (e = '0') then
        o <= "00";
    else if (i = '0') then
        o <= "01";
    else if (i = '1') then
        o <= "10";
    end if;
end process;
```

Assgn 3: (a) Design 1x2 decoder using structural

(b) " " " behavioral.

(i) using with select

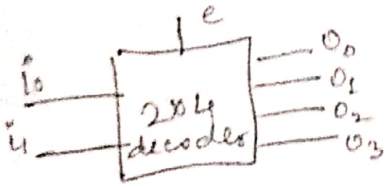
(ii) using process stmt and if else.

(iii) using " " " case.

~~Desi~~ 2x4 decoder using 1x2

→ call function

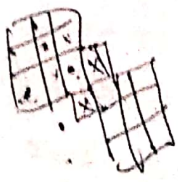
→ component. (instantiation).



e	i_1	i_0	o_3	o_2	o_1	o_0
0	X	X	0	0	0	0
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

1x2 decoder.

3 @ 1 x 2 decoders needed.




```
architecture a of
signal a : bit_vector (1 downto 0)
begin
d1: onetotwodecoder port
        map (e, i(1), a);
d2: onetotwodecoder port
        map (a(0), i(0), o(1 downto 0));
d3: onetotwodecoder port
        map (a(1), i(0), o(3 downto 2));
end
```

i1	i0	o3	o2	o1	o0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

o(0) =

```
with e select
    o <= "00" when '0',
with (e & i) select
    o <= "01" when
```

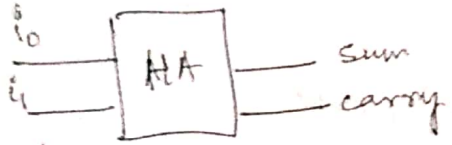
VLSI Lab.

(1) functions → can return value
(2) Procedure. → cannot return value.

Component has to be implemented first.

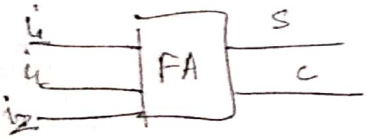
Implement FA using procedure HA

- 4 bit ripple carry adder using FA.



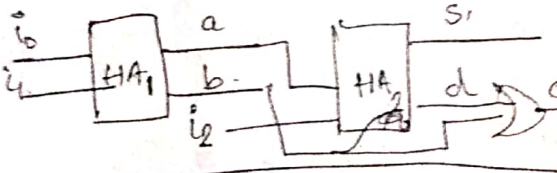
i_1	i_0	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
procedure halfadder (signal i : in bit_vector(1 downto 0);
                     signal sum : out bit;
                     signal carry : out bit) is
begin
    sum   <= i(0) xor i(1);
    carry <= i(0) and i(1);
end procedure;
```



T.T.

i_2	i_1	i_0	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

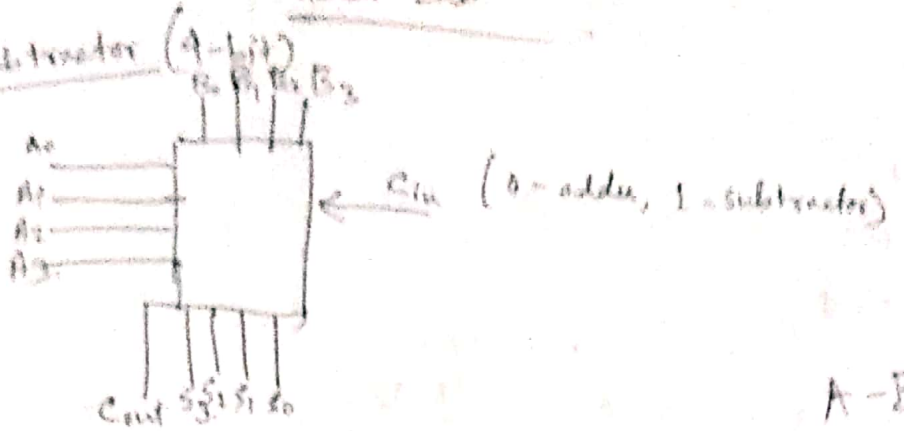



```
architecture of full adder is
    procedure . — — — —
    end pro . —
    signal a, b : bit

begin
    ha1 : halfadder (i(1 downto 0), a, b);
    ha2 : halfadder (i(2) & a, s, c);
```

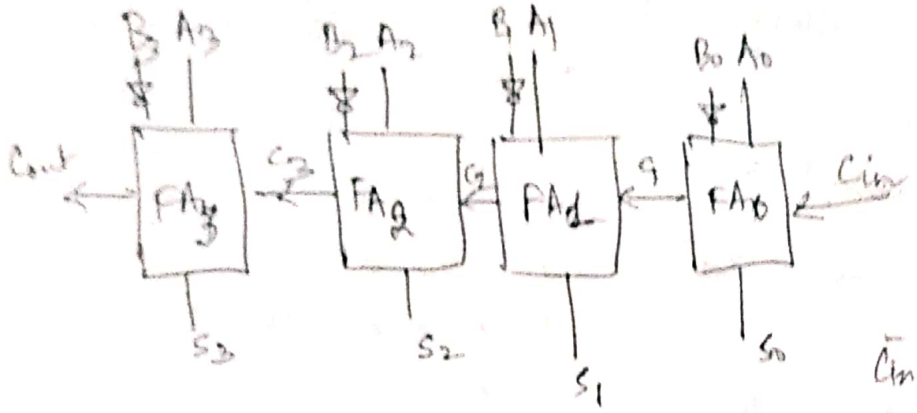
```
c <= b or d;
end arch;
```

Adder/Subtractor (4-bit)



A - B

1000
0111
+1



$\bar{C_{in}}b + C_{in}\bar{b}$

$C_{in} \oplus b$

S for subtractor

Logic: inputb : signal bit_vector(3 downto 0)

$C_0 <= C_{in}$.

gen1 : for i 0 to 3 generate inputb(i), C(i)

fa: fulladder port map (a(i), b(i), s(i), c(c+1));

end generate gen1; ↑ only for component.

inputb <= b when C_{in} = '0' else not(b);

if carry generated ignore.

if carry not gen then 2's complement.

Cout <= '0' when C_{in} = '1' and Cout = '1';

* BCD adder using 4 bit

A(3) <= not A(0)

process (A)

begin

if (A(0) = '0') then

A(1) <= A(1)

elsif

else

if (A(1))

elsif (A(0) = '1') then

B(0) <= A(0)

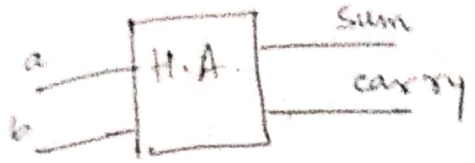
B(1) <= not (A(1))

1100 0100

↓↓

0011 1100

H.A.



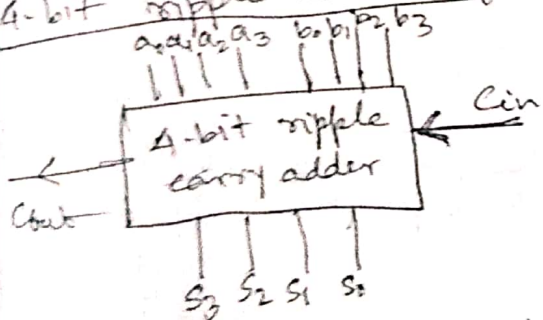
a	b	carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

```
architecture - -
  procedure halfadder ( a,b: in bit,
                        Sum, carry: out bit);
begin.
    Sum := a xor b;
    carry := a and b;
end.

process (aa, bb, cc)
variable s,c: bit
begin
    h1: halfadder (bb, cc, s, c);
    h2: halfadder (aa, s, ss, carry1);

    Sum Sum := ss;
    carrycarry := c or carry1;
end process.
end behavioral.
```

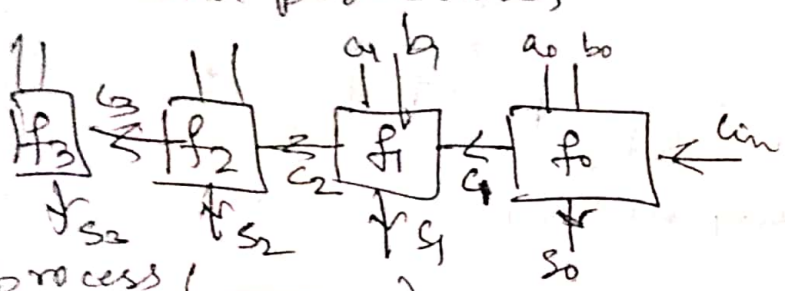
4-bit ripple carry adder:




```
architecture - - begin.
procedure halfadder ( a,b: in bit, sum,carry: out bit) is.

begin
    sum := a xor b;
    carry := a and b;
end procedure;
```

procedure fulladder(aa, bb, cc: in bit,
sum, sumsum, carry, carrycarry: out bit)
begin
write full adder code without process.
end procedure;



process ( )
begin variable c
c(0) <= cin;
for i in 0 to 3 loop
f: fulladder(a(i), b(i), c(i), s(i), c(i+1));
end loop;
cout <= c(4);
end process.
end beh.

9. Using 4 bit ripple carry adder subtract