Sistemas Digitais I

LESI - 2º ano

Unit 6 - Combinational Design Practices

João Miguel Fernandes
www.di.uminho.pt/~jmf

DEP. DE INFORMÁTICA
ESCOLA DE ENGENHARIA
UNIVERSIDADE DO MINHO

6. Combinational Practices

- Summary -

- PLDs
- Decoders
- 7-Segment Decoders
- Encoders
- Multiplexers
- XOR and Parity Circuits
- Comparators
- Adders, Subtractors and ALUs
- Multipliers

6. Combinational Practices

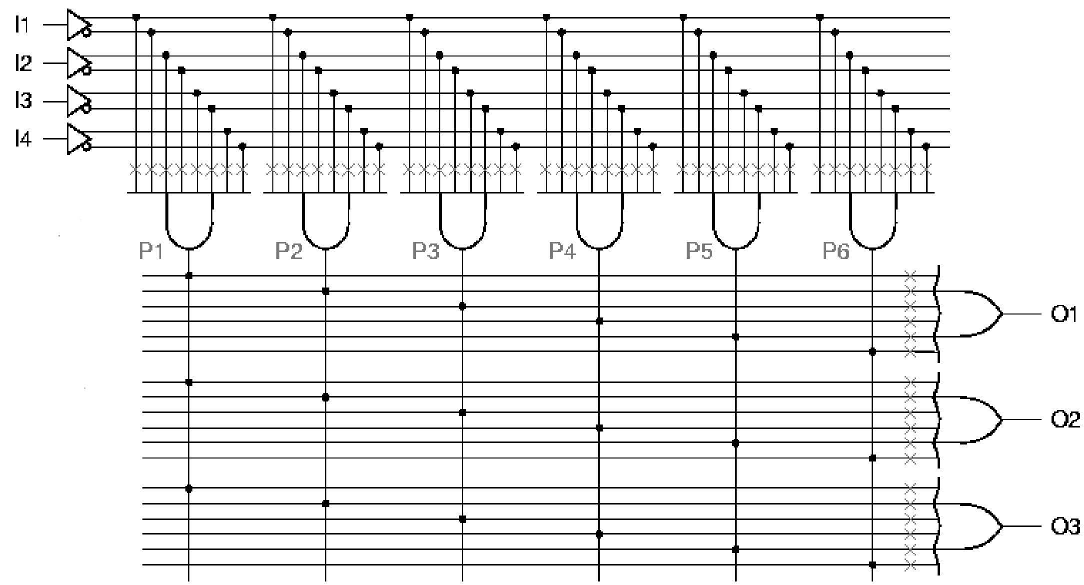
- PLDs (1) -

- The first PLDs were Programmable Logic Arrays (PLAs).
- A PLA is a combinational, 2-level AND-OR device that can be programmed to realise any sum-of-products logic expression.
- A PLA is limited by:
  - **the number of inputs (n)**
  - **the number of outputs (m)**
  - **the number of product terms (p)**
- We refer to an "n x m PLA with p product terms". Usually, $p << 2^n$.
- An n x m PLA with p product terms contains p 2n-input AND gates and m p-input OR gates.

6. Combinational Practices

- PLDs (2) -

- Each input is connected to a buffer that produces a true and a complemented version of the signal.
- Potential connections are indicated by Xs.
- The device is programmed by establishing the needed connections.
- The connections are made by fuses.

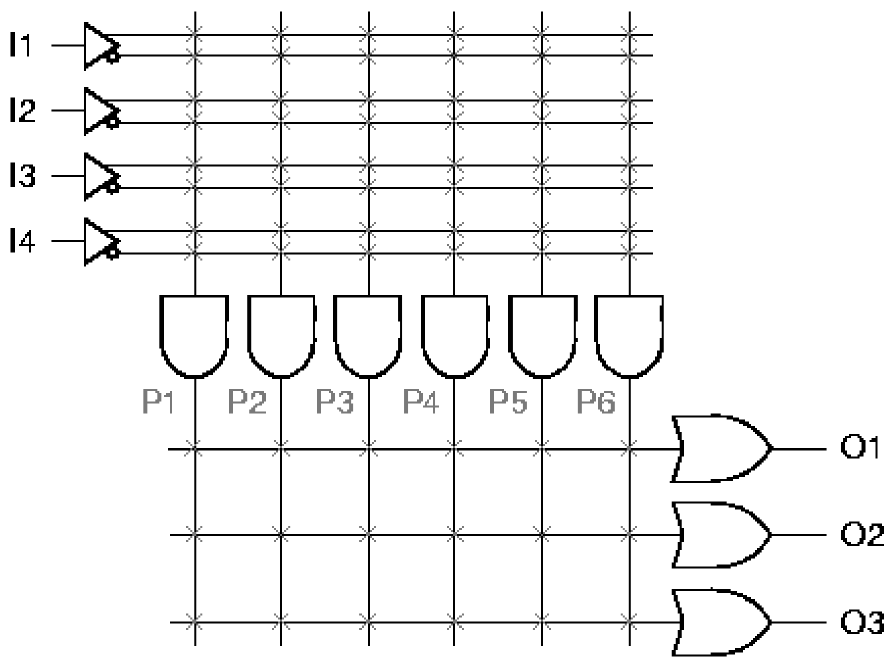


A 4x3 PLA with 6 product terms.

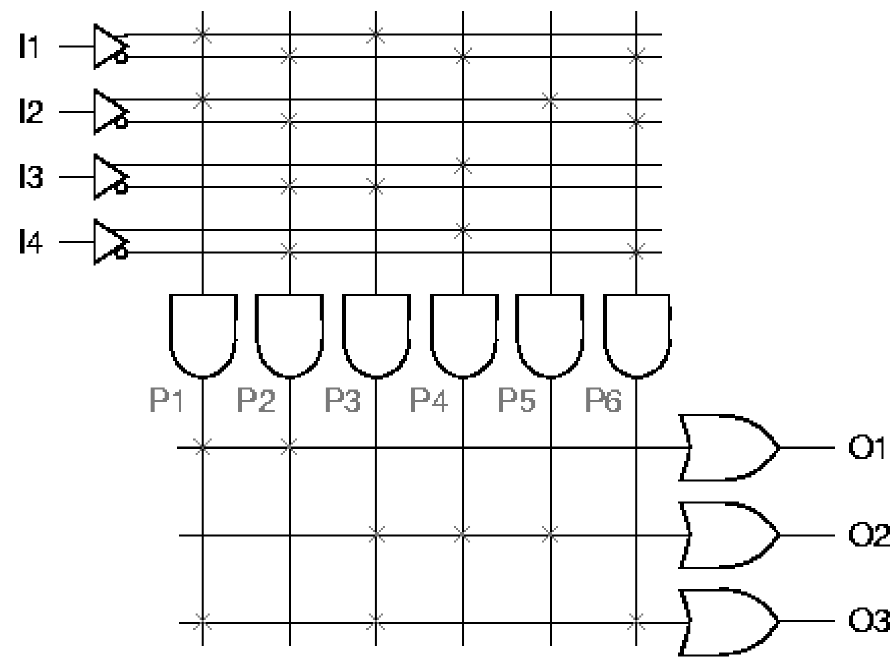
6. Combinational Practices

- PLDs (3) -

- Compact representation of the 4x3 PLA with 6 product terms.



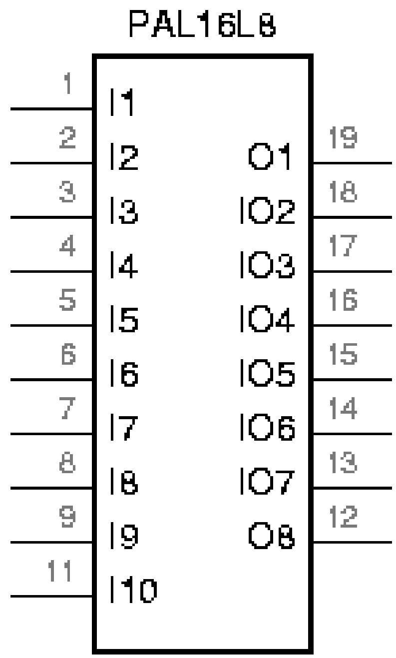
- O1 = I1·I2 + I1'·I2'·I3'·I4'
  O2 = I1·I3' + I1'·I3·I4 + I2
  O3 = I1·I2 + I1·I3' + I1'·I2'·I4'



6. Combinational Practices

- PLDs (4) -

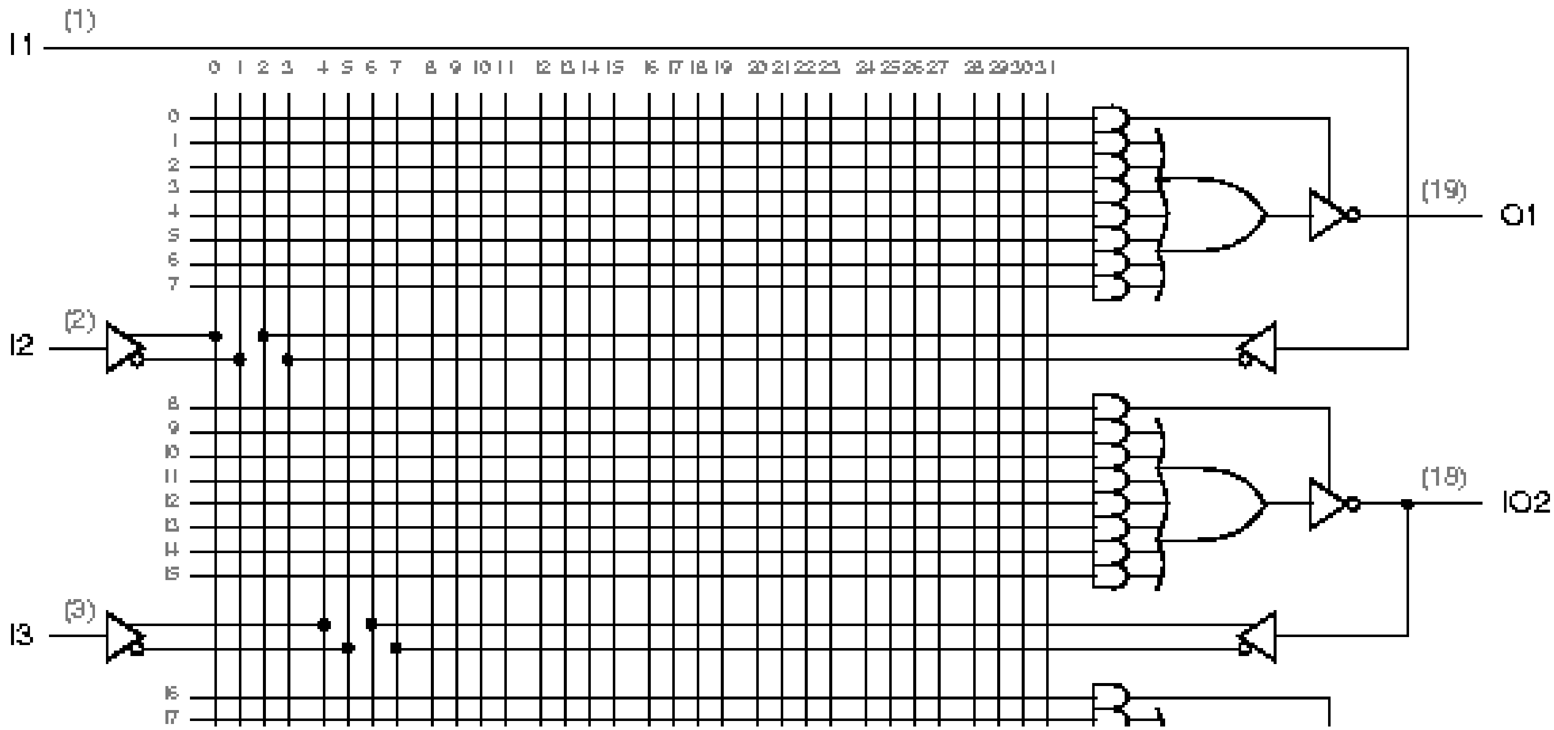
- Another PLD is PAL (Programmable Array Logic).
- A PAL device has a fixed OR array.
- In a PAL, product terms are not shared by the outputs.
- Each output has a fixed and unique set of product terms that it can use.
- A PAL is usually faster than a similar PLA.



6. Combinational Practices

- PLDs (5) -

- Part of the logic diagram of the PAL 16L8.

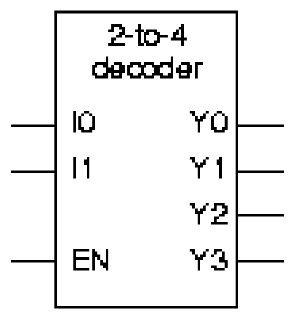


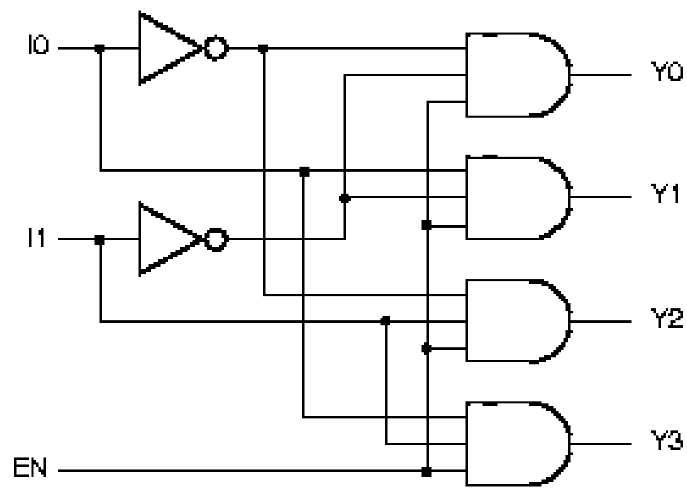
6. Combinational Practices

- Decoders (1) -

- A decoder is a circuit that converts coded inputs into coded outputs.
- Usually, the input code has fewer bits than the output code.
- The most common decoder is an n-to-2^n or binary decoder.
- A binary decoder is used when one of 2^n outputs needs to be activated based on an n-bit input value.

Inputs			*Outputs*			
EN	I1	I0	Y3	Y2	Y1	Y0
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

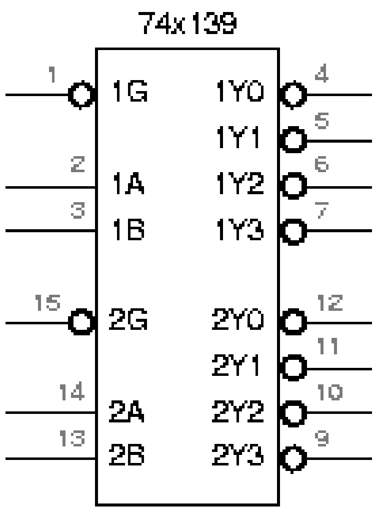




6. Combinational Practices

- Decoders (2) -

- A 74x139 IC has two independent 2-to-4 decoders.

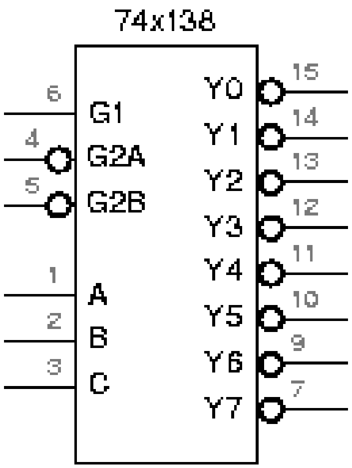


Inputs			*Outputs*			
G_L	B	A	Y3_L	Y2_L	Y1_L	Y0_L
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

6. Combinational Practices

- Decoders (3) -

- A 74x138 IC has one 3-to-8 decoder.

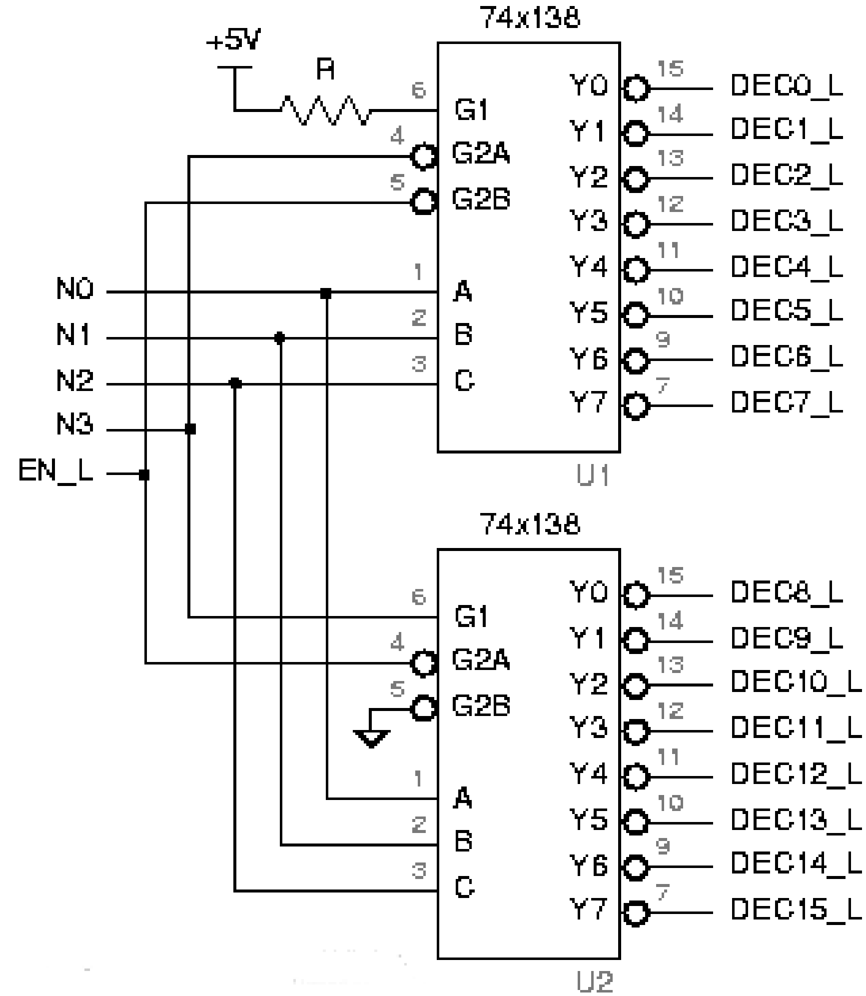


Inputs						*Outputs*							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	l	1	1	1	1
x	1	x	x	x	x	1	1	1	l	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

6. Combinational Practices

- Decoders (4) -

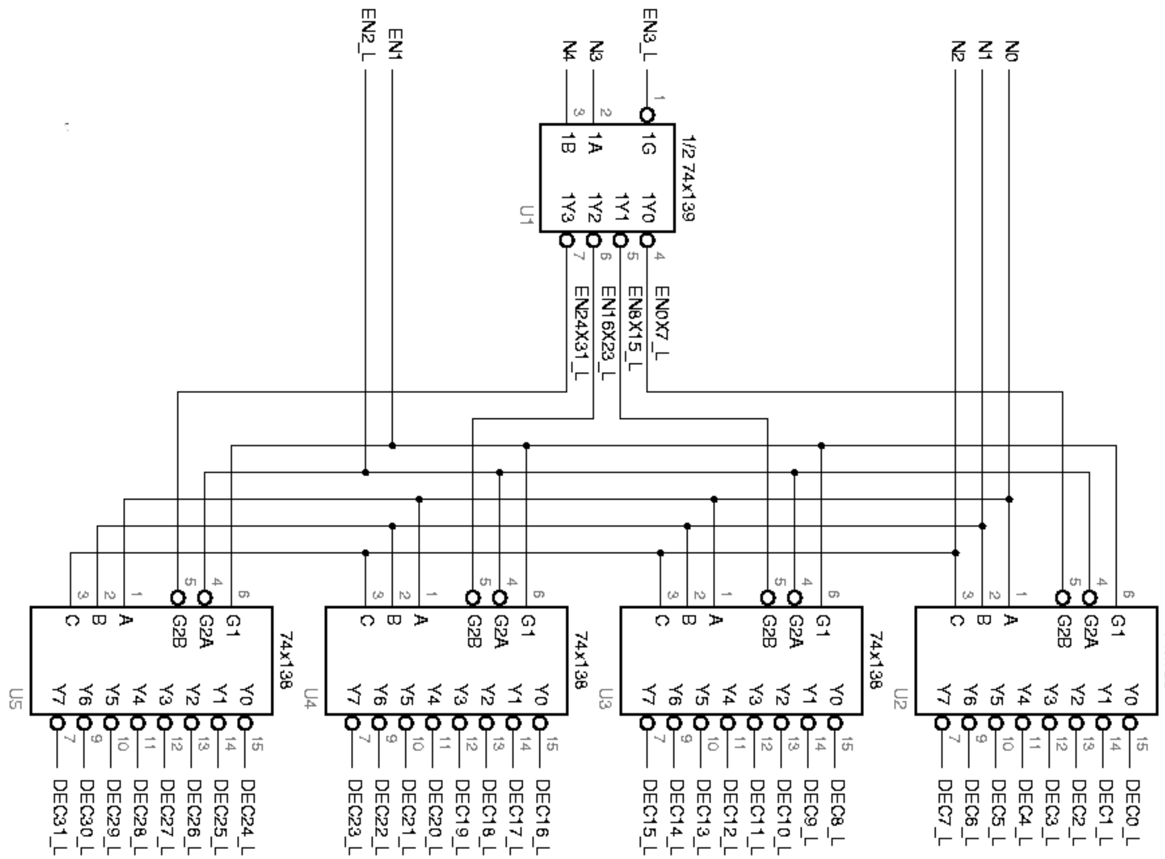
- Multiple decoders can be used to decode larger code words.
- The top decoder (U1) is enabled when N3 is 0, and the bottom decoder (U2) is enabled when N3 is 1.
- To handle larger code words, decoders can be cascaded hierarchically.



6. Combinational Practices

- Decoders (5) -

- To handle larger code words, decoders can be cascaded hierarchically.
- A 5-to-32 decoder can be built with one 2-to-4 and four 3-to-8 decoders.
- The 2-to-4 decoder treats the high order bits.
- The 3-to-8 decoders treat the low-order bits.



6. Combinational Practices

- Decoders (6) -

- There are several ways to write decoders in VHDL.
- The most primitive would be to write a structural description equivalent to the logic circuit on slide 7.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity V2to4dec is
  port (I0, I1, EN: in STD_LOGIC;
        Y0, Y1, Y2, Y3: out STD_LOGIC );
end V2to4dec;
architecture V2to4dec_s of V2to4dec is
  signal NOTI0, NOTI1: STD_LOGIC;
  component inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
  component and3 port (I0, I1, I2: in STD_LOGIC; O: out STD_LOGIC ); end component;
begin
  U1: inv port map (I0,NOTI0);
  U2: inv port map (I1,NOTI1);
  U3: and3 port map (NOTI0,NOTI1,EN,Y0);
  U4: and3 port map (   I0,NOTI1,EN,Y1);
  U5: and3 port map (NOTI0,   I1,EN,Y2);
  U6: and3 port map (   I0,    I1,EN,Y3);
end V2to4dec_s;
```

6. Combinational Practices

- Decoders (7) -

- The second alternative is using the dataflow style.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x138 is
    port (G1, G2A_L, G2B_L: in STD_LOGIC;       -- enable inputs
        A: in STD_LOGIC_VECTOR (2 downto 0);    -- select inputs
        Y_L: out STD_LOGIC_VECTOR (0 to 7) );  -- decoded outputs
 end V74x138;

architecture V74x138_a of V74x138 is
  signal Y_L_i: STD_LOGIC_VECTOR (0 to 7);
begin
  with A select Y_L_i <=
    "01111111" when "000",
    "10111111" when "001",
    "11011111" when "010",
    "11101111" when "011",
    "11110111" when "100",
    "11111011" when "101",
    "11111101" when "110",
    "11111110" when "111",
    "11111111" when others;
  Y_L <= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else "11111111";
end V74x138_a;
```

6. Combinational Practices

- Decoders (8) -

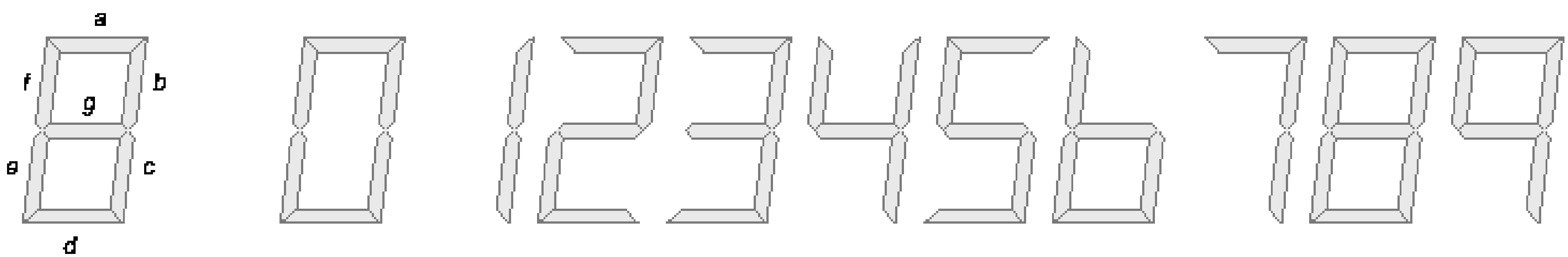
- Another alternative is using the behavioral style.

```
architecture V3to8dec_c of V3to8dec is
begin
process (G1, G2, G3, A)
  variable i: INTEGER range 0 to 7;
  begin
    Y <= "00000000";
    if (G1 and G2 and G3) = '1' then
      for i in 0 to 7 loop
        if i=CONV_INTEGER(A) then Y(i) <= '1'; end if;
      end loop;
    end if;
  end process;
end V3to8dec_c;
```

6. Combinational Practices

- 7-Segment Decoders (1) -

- A <u>7-segment display</u> is used in watches, calculators, and devices to show decimal data.
- A digit is displayed by illuminating a subset of the 7 line segments.



- A <u>7-segment decoder</u> has a 4-bit BCD as its input and the 7-segment code as its output.

6. Combinational Practices

- 7-Segment Decoders (2) -

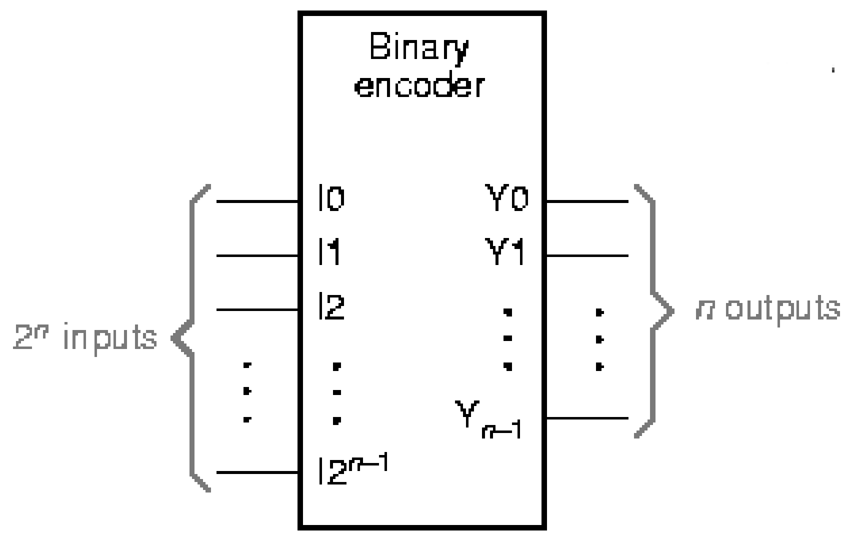
Inputs					*Outputs*						
BI_L	D	C	B	A	a	b	c	d	e	f	g
0	x	x	x	x	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
1	0	0	1	0	1	1	0	1	1	0	1
1	0	0	1	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	1	0	1	1	0	1	1
1	0	1	1	0	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1	0	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	0	0	1	1
1	1	0	1	0	0	0	0	1	1	0	1
1	1	0	1	1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	0	0	0	1	1
1	1	1	0	1	1	0	0	1	0	1	1
1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0

- Exercise 1:
  Obtain minimised expressions for outputs of the 7-segment decoder.
- Exercise 2:
  Write a VHDL description of a 7-segment decoder.

6. Combinational Practices

- Encoders (1) -

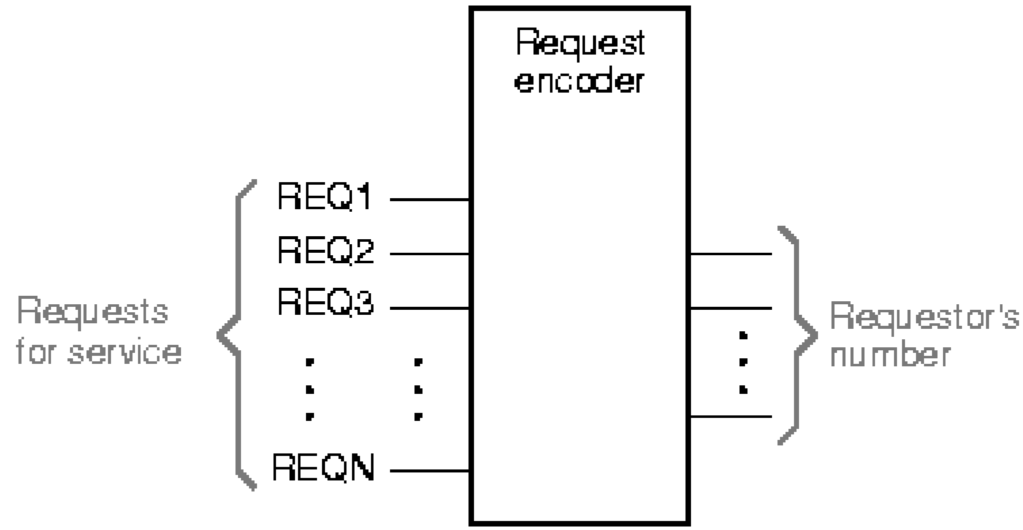
- An encoder is a circuit whose output code has normally fewer bits than its input code.
- The simplest encoder to build is a 2^n-to-n or binary encoder. It has the opposite function as a binary decoder.
- Equations for an 8-to-3 encoder :
  Y0 = I1 + I3 + I5 + I7
  Y1 = I2 + I3 + I6 + I7
  Y2 = I4 + I5 + I6 + I7
- Only 1 input is active at a time. What happens if 2 inputs are asserted (ex: I2 and I4)?



6. Combinational Practices

- Encoders (2) -

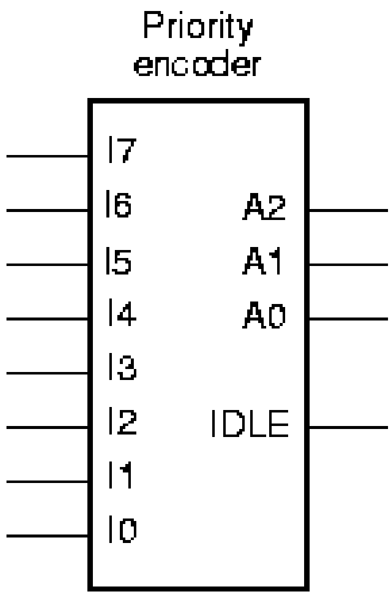
- To implement a <u>request encoder</u>, the binary encoder does not work!
- It assumes that only 1 input is asserted.



- If multiple requests can be made simultaneously, a <u>priority</u> must be assigned to the input lines.
- When multiple requests are made, the device (<u>priority encoder</u>) produces the number of the highest-priority requestor.

6. Combinational Practices

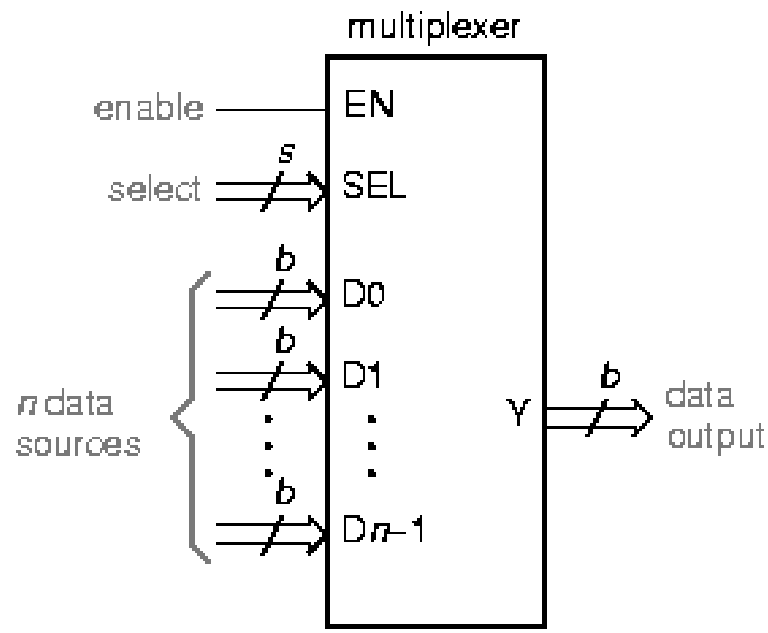
- Encoders (3) -



- Input I7 has the highest priority.
- Outputs A2-A0 contain the number of the highest-priority asserted input, if any.
- The IDLE output is asserted if no inputs are asserted.
- Intermediate variable Hi is 1, if Ii is the highest priority 1-input:
  H7 = I7　　　　H6 = I6·I7'
  H5 = I5·I6'·I7'　　H4 = I4·I5'·I6'·I7'
  ... (similar equations for H3-H0)
- A0 = H1 + H3 + H5 + H7
  A1 = H2 + H3 + H6 + H7
  A2 = H4 + H5 + H6 + H7
- IDLE= I0'·I1'·I2'·I3'·I4'·I5'·I6'·I7'

6. Combinational Practices

- Multiplexers (1) -

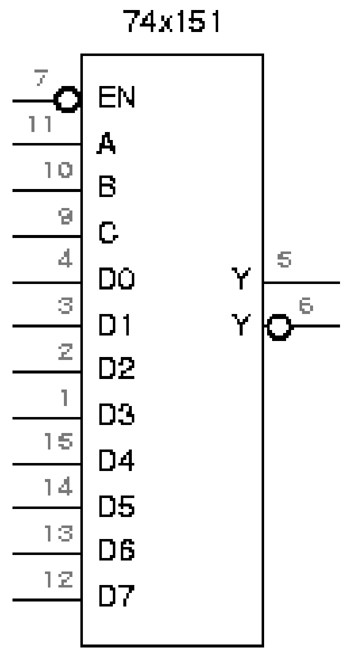


- A <u>multiplexer</u> (mux) is a digital switch.
- It connects data from one of n sources to its output.
- The SEL input selects among the n sources, so $s = \lceil \log_2 n \rceil$.
- When EN=0, Y=0;
  When EN=1, the mux is working.
- Multiplexers are used in computers between the processor's registers and its ALU, to select among a set of registers which one is connected to the ALU.

6. Combinational Practices

- Multiplexers (2) -

- A 74x151 IC has one 8-input, 1-bit multiplexer.
- The select inputs are named A,B,C, where C is the MSB.
- The enable input EN_L is active low.
- Both active-low and high versions of the output are provided

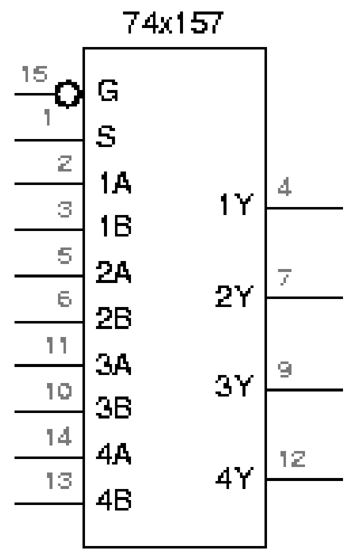


Inputs				Outputs	
EN_L	C	B	A	Y	Y_L
1	x	x	x	0	1
0	0	0	0	D0	D0′
0	0	0	1	D1	D1′
0	0	1	0	D2	D2′
0	0	1	1	D3	D3′
0	1	0	0	D4	D4′
0	1	0	1	D5	D5′
0	1	1	0	D6	D6′
0	1	1	1	D7	D7′

6. Combinational Practices

- Multiplexers (3) -

- A 74x157 IC has one 2-input, 4-bit multiplexer.
- The select input is S.
- The enable input G_L is active low.
- The truth table was extended and inputs appear at the outputs columns.

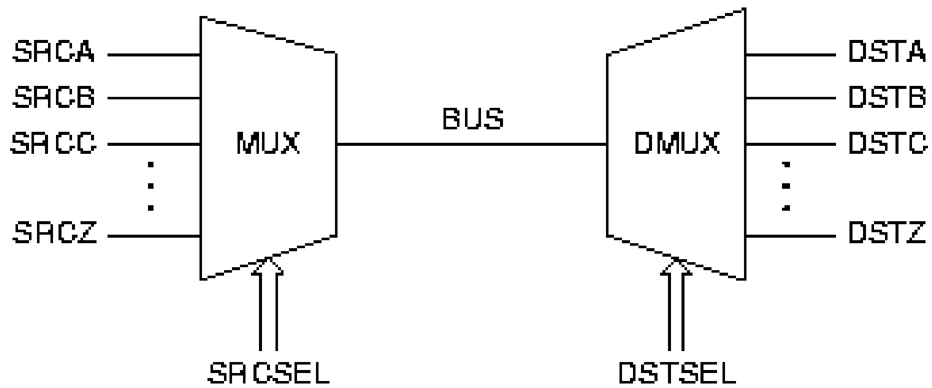


Inputs		Outputs			
G_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

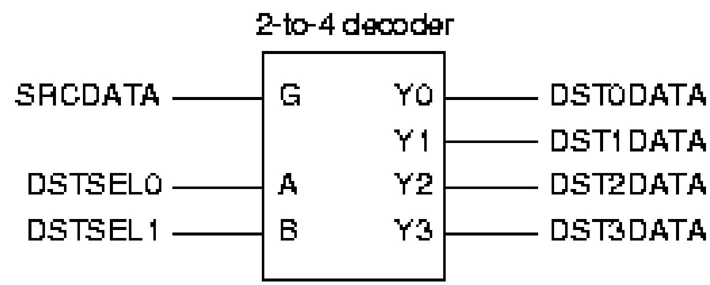
6. Combinational Practices

- Multiplexers (4) -

- A multiplexer can be used to select one of n sources of data to transmit on a bus.
- At the other end, a demultiplexer can be used to route the bus to one of m destinations.



- The function of a multiplexer is the inverse of a demultiplexer's.
- A 1-bit, n-output demultiplexer has one data input and s inputs to select one of the $n=2^s$ data outputs.



6. Combinational Practices

- Multiplexers (5) -

- It is easy to describe multiplexers in VHDL.
- In the dataflow style, a `SELECT` statement is required.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4in8b is
    port (
        S: in STD_LOGIC_VECTOR (1 downto 0);       -- Select inputs, 0-3 ==> A-D
        A, B, C, D: in STD_LOGIC_VECTOR (1 to 8); -- Data bus input
        Y: out STD_LOGIC_VECTOR (1 to 8)           -- Data bus output
    );
end mux4in8b;

architecture mux4in8b of mux4in8b is
begin
    with S select Y <=
      A when "00",
      B when "01",
      C when "10",
      D when "11",
      (others => 'U') when others; -- this creates an 8-bit vector of 'U'
end mux4in8b;
```

6. Combinational Practices

- Multiplexers (6) -

- In a behavioural architecture, a `CASE` statement is used.

```
architecture mux4in8p of mux4in8b is
begin
process(S, A, B, C, D)
  begin
    case S is
      when "00" => Y <= A;
      when "01" => Y <= B;
      when "10" => Y <= C;
      when "11" => Y <= D;
      when others => Y <= (others => 'U');  -- 8-bit vector of 'U'
    end case;
  end process;
end mux4in8p;
```

- It is easy to customise the selection criteria in a VHDL multiplexer program.

6. Combinational Practices

- XOR and Parity Circuits (1) -

- An <u>Exclusive-OR (XOR)</u> gate is a 2-input gate whose output is 1, if exactly one of its inputs is 1.
- An XOR gate produces a 1 output if its input are different.

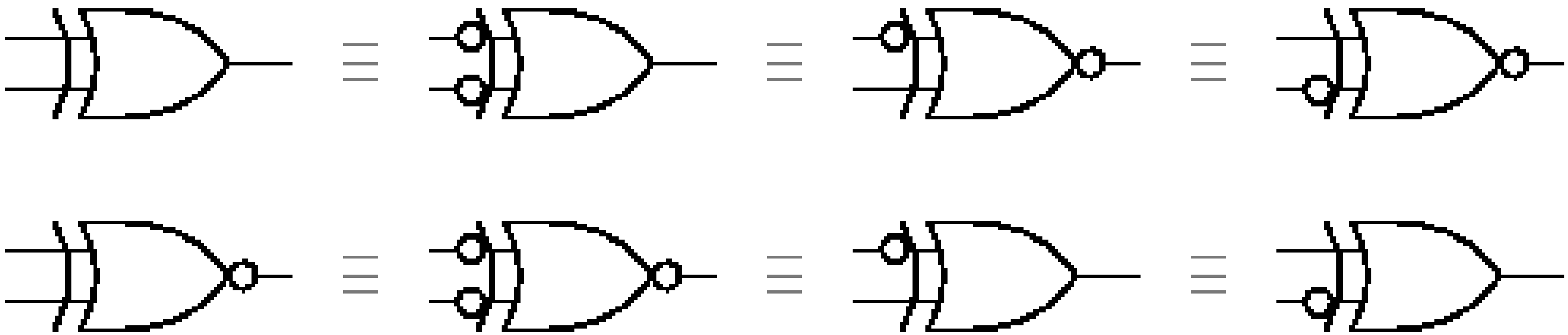
X	Y	$X \oplus Y$ (XOR)	$(X \oplus Y)'$ (XNOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

- An <u>Exclusive-NOR (XNOR)</u> is just the opposite: it produces a 1 output if its inputs are the same.
- The XOR operation is denoted by the symbol $\oplus$.
- $X \oplus Y = X' \cdot Y + X \cdot Y'$

6. Combinational Practices

- XOR and Parity Circuits (2) -

- There are 4 symbols for each XOR and XNOR function.

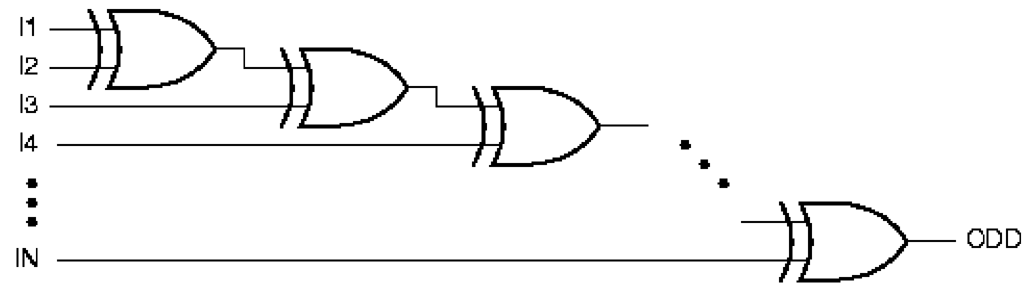


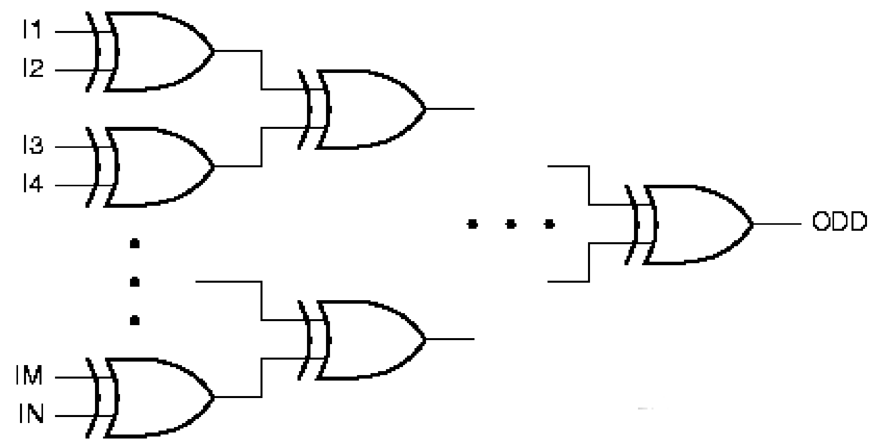
- These alternatives are a consequence of the following rule:
  - **Any two signals (inputs or output) of an XOR or XNOR gate may be complemented without changing the resulting logic function.**
- In bubble-to-bubble design we choose the symbol that is most expressive of the logic function being performed.

6. Combinational Practices

- XOR and Parity Circuits (3) -

- n XOR gates may be cascaded to form a circuit with n+1 inputs and a single output. This is a <u>odd-parity circuit</u>, because its output is 1 if an odd number of its inputs are 1.
- If the output of either circuit is inverted, we get an even-parity circuit, whose output is 1 if an even number of its inputs are 1.





6. Combinational Practices

- XOR and Parity Circuits (4) -

- VHDL provides the primitive operators `xor` and `xnor`.
- A 3-input XOR device can be specified in VHDL dataflow style program.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity vxor3 is
  port (A, B, C: in STD_LOGIC;
           Y: out STD_LOGIC);
end vxor3;
architecture vxor3 of vxor3 is
begin
  Y <= A xor B xor C;
end vxor3;
```

6. Combinational Practices

- XOR and Parity Circuits (5) -

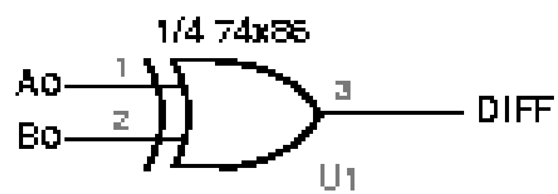
- A 9-input parity function can be specified behaviourally.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity parity9 is
  port (I: in STD_LOGIC_VECTOR (1 to 9);
        EVEN, ODD: out STD_LOGIC );
end parity9;
architecture parity9p of parity9 is
begin
process (I)
  variable p : STD_LOGIC;
  begin
    p := I(1);
    for j in 2 to 9 loop
      if I(j) = '1' then p := not p; end if;
    end loop;
    ODD <= p;
    EVEN <= not p;
  end process;
end parity9p;
```

6. Combinational Practices

- Comparators (1) -

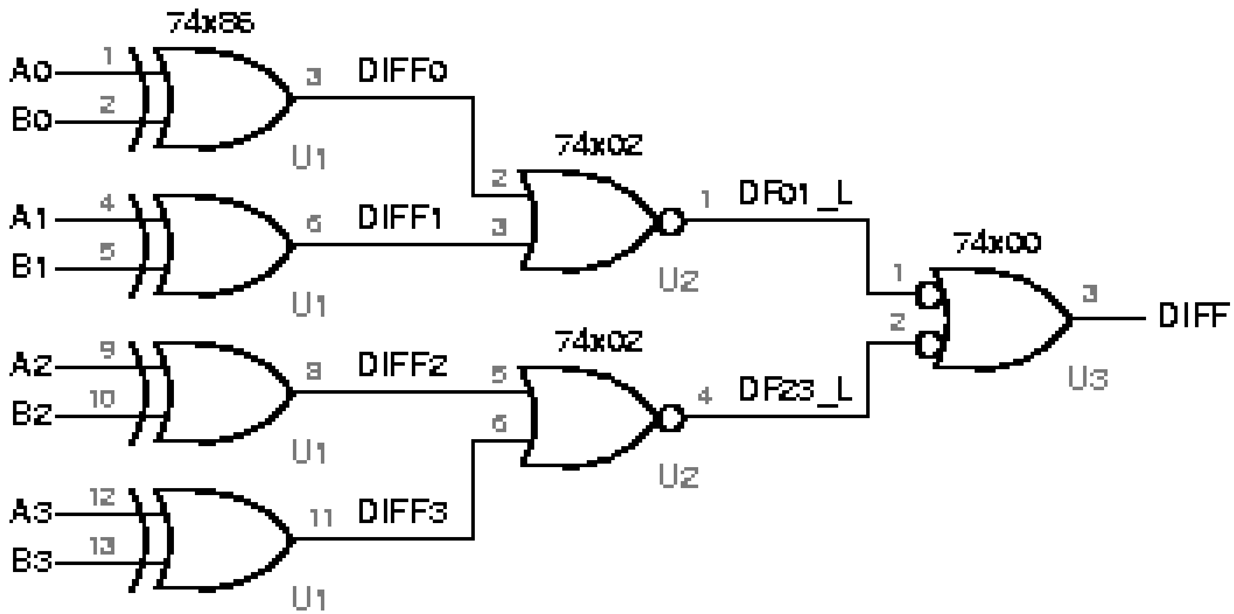
- Comparing two binary words is a common operation in computers.
- A circuit that compares 2 binary words and indicates whether they are equal is a comparator.
- Some comparators interpret their input as signed or unsigned numbers and also indicate an arithmetic relationship (greater or less than) between the words.
- These circuits are often called magnitude comparators.
- XOR and XNOR gates can be viewed as 1-bit comparators.
- The DIFF output is asserted if the inputs are different.



6. Combinational Practices

- Comparators (2) -

- The outputs of 4 XOR gates can be ORed to create a 4-bit comparator.

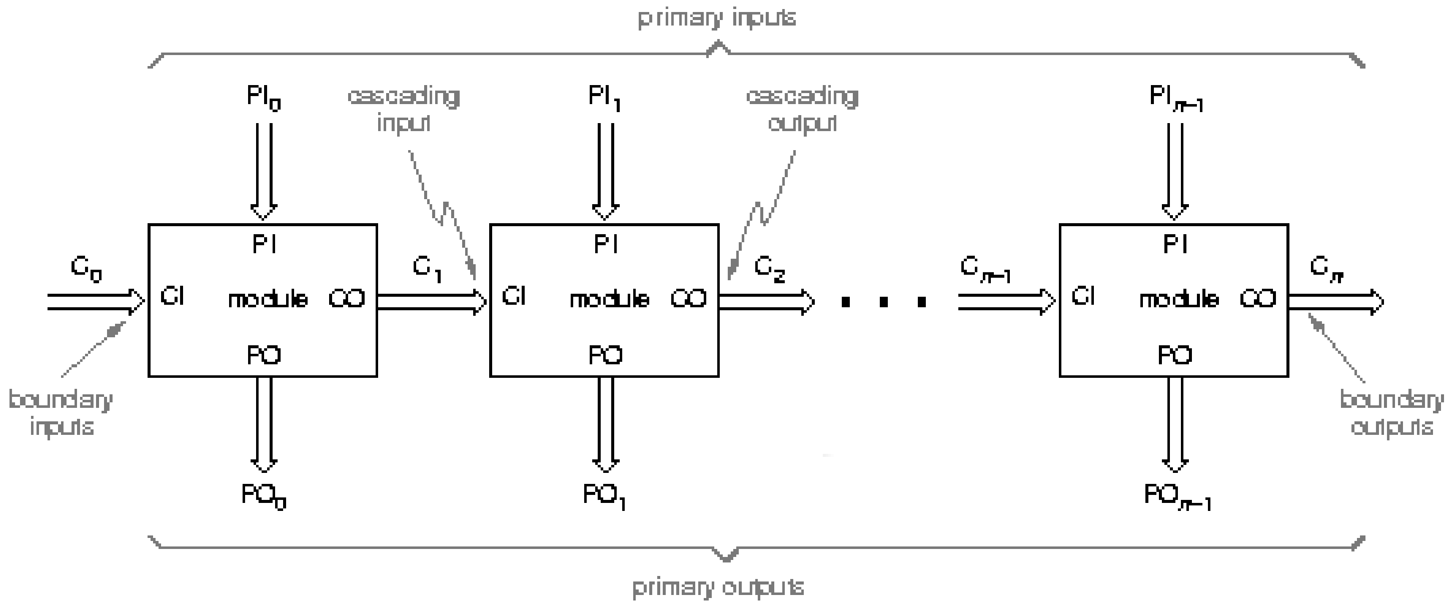


- The DIFF output is asserted if any of the input-bit pairs are different.
- This circuit can be easily adapted to any number of bits per word.

6. Combinational Practices

- Comparators (3) -

- An <u>iterative circuit</u> is a combinational circuit with the following structure.

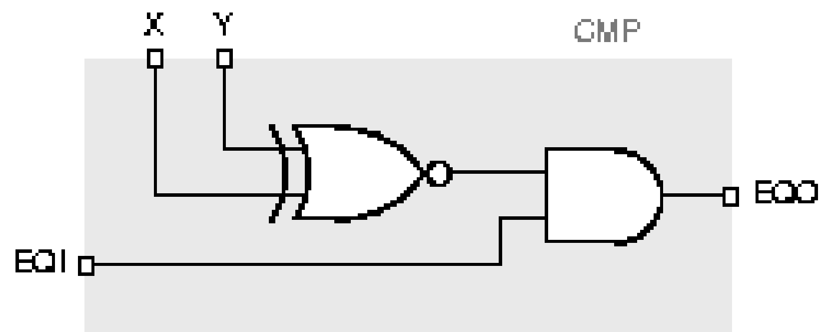


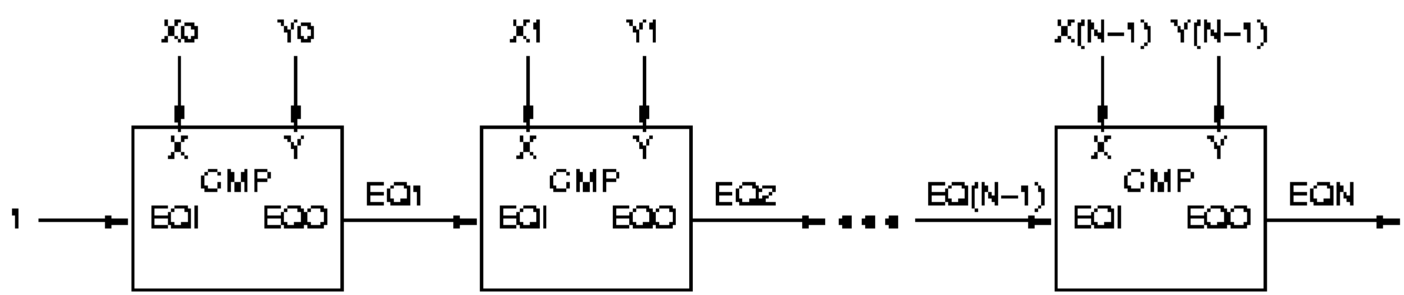
- The circuit contains n identical modules, each of which has <u>both primary inputs and outputs</u> and <u>cascading inputs and outputs</u>.
- The left-most cascading inputs are usually connected to fixed values.

6. Combinational Practices

- Comparators (4) -

- Two n-bit values X and Y can be compared one bit at a time using a single bit EQ_i at each step to keep track of whether all of the bit-pairs have been equal so far:
- 1. Set EQ_0 to 1 and set *i* to 0.
  2. If EQ_i is 1 and X_i=Y_i, set EQ_{i+1} to 1.
     Else set EQ_{i+1} to 0.
  3. Increment *i*.
  4. If *i* < *n*, go to step 2.

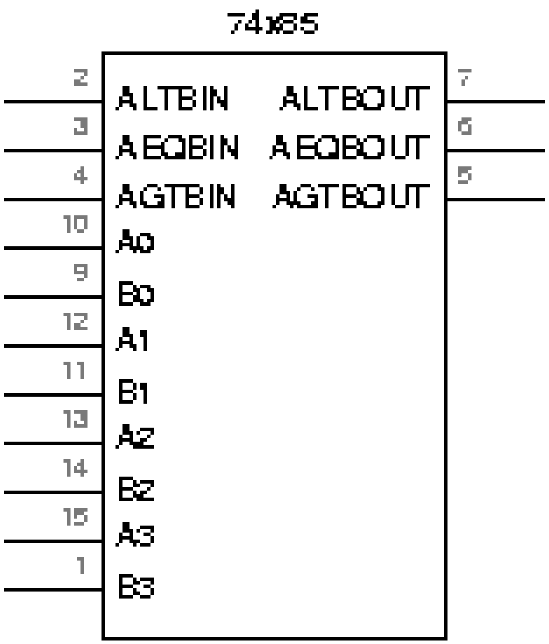




6. Combinational Practices

- Comparators (5) -

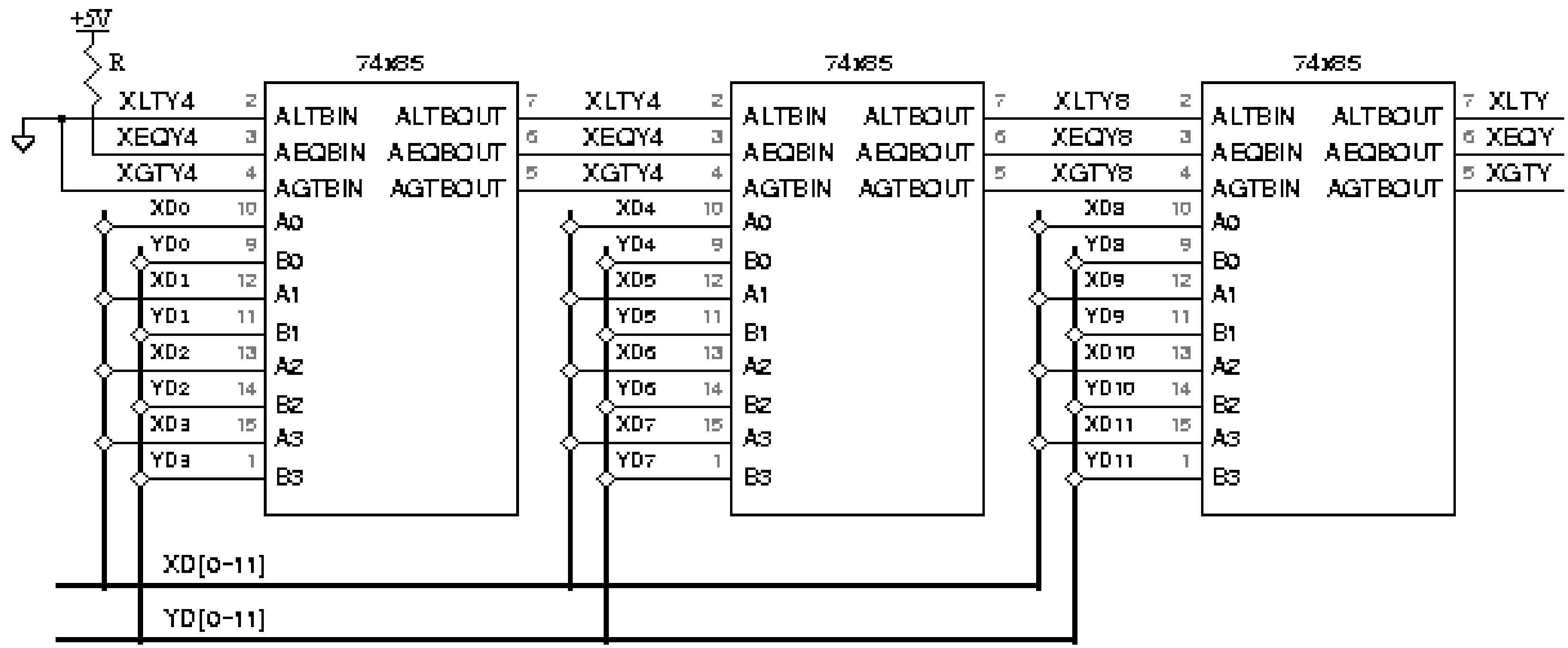
- Several MSI comparators have been developed commercially.
- The 74x85 is a 4-bit comparator.
- It provides a greater-than output, a less-than output and an equal output.
- The 74x85 also has cascading inputs for combining multiple chips to create comparators for more than 4 bits.
- AGTBOUT = (A>B) + (A=B) · AGTBIN
  AEQBOUT = (A=B) · AEQBIN
  ALTBOUT = (A<B) + (A=B) · ALTBIN



6. Combinational Practices

- Comparators (6) -

- With three 74x85 circuits, a 12-bit comparator can be built.



6. Combinational Practices

- Comparators (7) -

- VHDL has comparison operators for all of its built-in types.
- Equality ($=$) and inequality ($/=$) operators apply to all types.
- For array and record types, the operands must have equal size and structure, and the operands are compared component by component.
- VHDL's other comparison operators ($>$, $<$, $>=$, $<=$) apply only to integers, enumerated types and one-dimensional arrays of enumeration or integer types.

6. Combinational Practices

- Adders, Subtractors and ALUs (1) -

- Addition is the most commonly performed arithmetic operation in digital systems.
- An adder combines two arithmetic operands using the addition rules.
- The same addition rules, and hence the same adders, are used for both unsigned and 2’s complement numbers.
- An adder can perform subtraction as the addition of the minuend and the complemented subtrahend.
- A subtractor can also be built to perform subtraction directly.
- An ALU (Arithmetic and Logic Unit) performs addition, subtraction, and other logical operations.

6. Combinational Practices

- Adders, Subtractors and ALUs (2) -

- The simplest adder, called a half adder, adds two 1-bit operands X and Y, producing a 2-bit sum.
- The sum can range from 0 to 2, which requires two bits to express.
- The low-order bit of the sum may be named HS (half sum).
- The high-order bit of the sum may be named CO (carry out).
- The following equations can be written:
  $HS = X \oplus Y = X \cdot Y' + X' \cdot Y$
  $CO = X \cdot Y$
- To add operands with more than one bit, carries between bit positions must be provided.

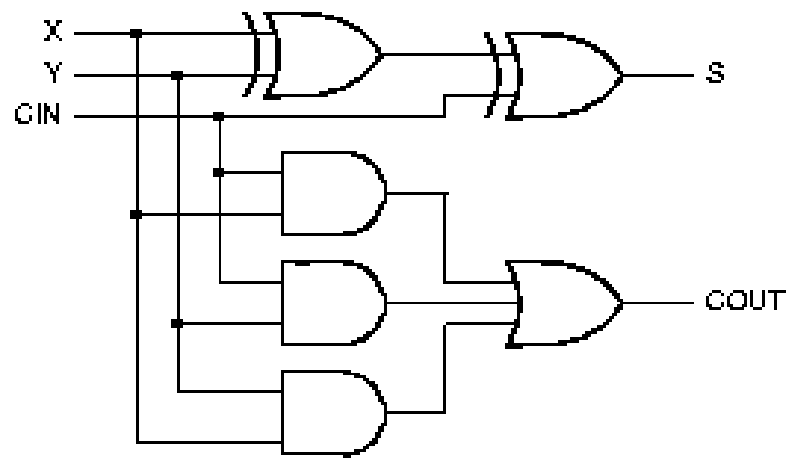
6. Combinational Practices

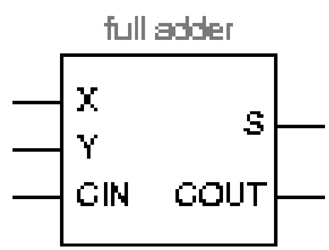
- Adders, Subtractors and ALUs (3) -

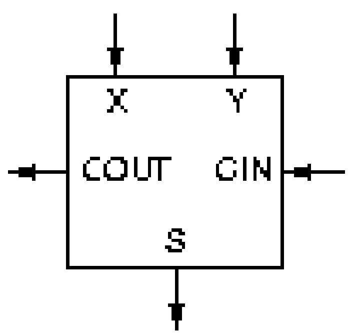
- The building block for this operation is called a full adder.
- Besides the addend-bit inputs X and Y, a full adder has a carry-bit input, CIN.
- The sum of the 3 bits can range from 0 to 3, which can still be expressed with just two output bits, S and COUT.
- The following equations can be written:

  $S = X \oplus Y \oplus CIN$

  $COUT = X \cdot Y + X \cdot CIN + Y \cdot CIN$



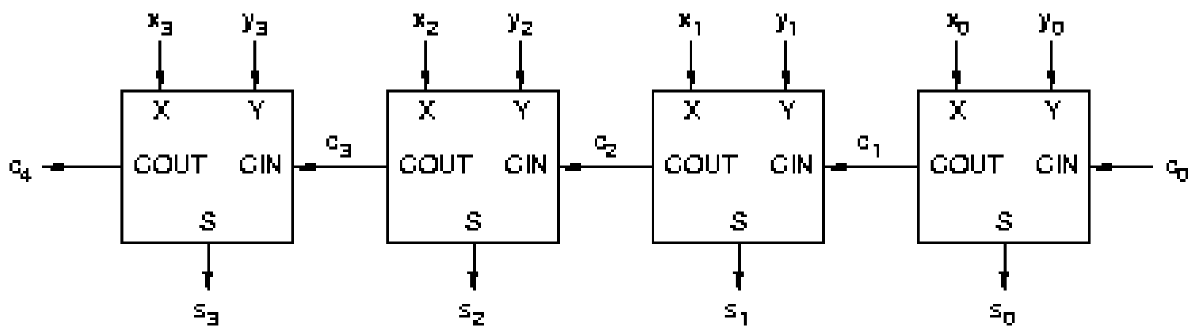




6. Combinational Practices

- Adders, Subtractors and ALUs (4) -

- Two binary words, each with n bits, can be added using a ripple adder.
- A <u>ripple adder</u> is a cascade of n full-adders stages, each of which handles one bit.



- The carry input to the least significant bit (c_0) is usually set to 0.
- The carry output of each full adder is connected to the carry input of the next most significant full adder.

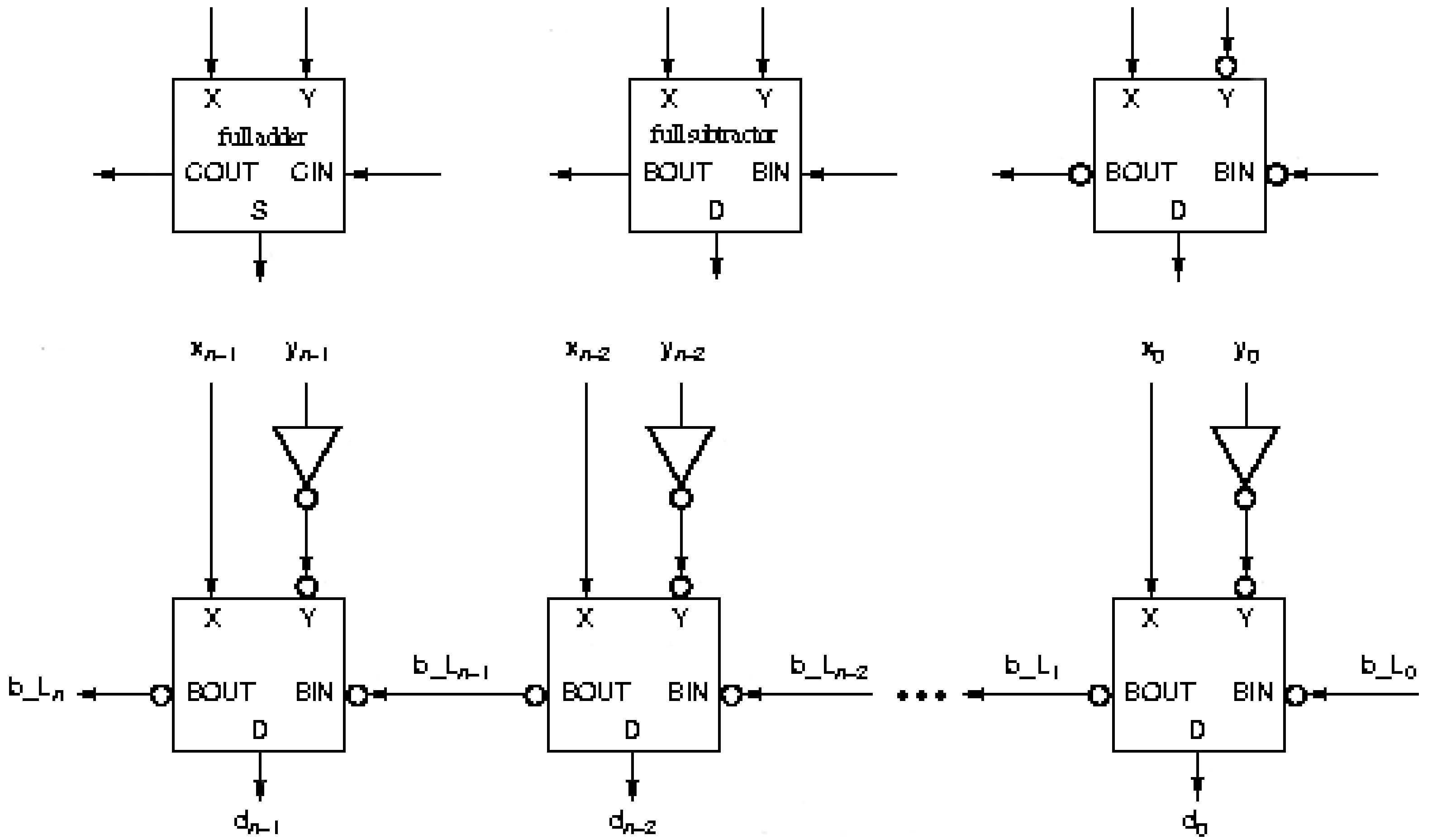
6. Combinational Practices

- Adders, Subtractors and ALUs (5) -

- The binary subtraction operation is analogous to binary addition.
- A <u>full subtractor</u> has inputs X (minuend), Y (subtrahend) and BIN (borrow in) and outputs D (difference) and BOUT (borrow out).
- The following equations can be written:
  $D = X \oplus Y \oplus BIN$
  $BOUT = X' \cdot Y + X' \cdot BIN + Y \cdot BIN$
- These equations are similar to the equations for a full adder.
  $D = X \oplus Y' \oplus BIN'$
  $BOUT = X \cdot Y' + X \cdot BIN' + Y' \cdot BIN'$
- A full subtractor can be built from a full adder. $X-Y = X+Y'+1$

6. Combinational Practices

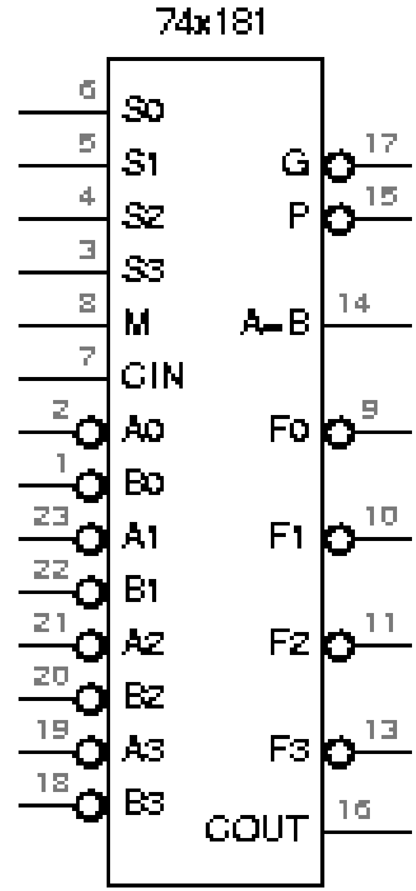
- Adders, Subtractors and ALUs (6) -



6. Combinational Practices

- Adders, Subtractors and ALUs (7) -

- An ALU is a combinational circuit that can perform several arithmetic and logical operations on a pair of b-bit operands.
- The operation to be performed is specified by a set of function-select inputs.
- Typical MSI ALUs have 4-bit operands and three to five function-select inputs, allowing up to 32 different functions to be performed.
- A 74x181 IC has one 4-bit ALU.
- The operation performed by the 74x181 is selected by the M and S3-S0 inputs.



6. Combinational Practices

- Adders, Subtractors and ALUs (8) -

Inputs				Function	
S3	S2	S1	S0	M = 0 (arithmetic)	M = 1 (logic)
0	0	0	0	F = A minus 1 plus CIN	F = A'
0	0	0	1	F = A · B minus 1 plus CIN	F = A' + B'
0	0	1	0	F = A · B' minus 1 plus CIN	F = A' + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B') plus CIN	F = A' · B'
0	1	0	1	F = A · B plus (A + B') plus CIN	F = B'
0	1	1	0	F = A minus B minus 1 plus CIN	F = A ⊕ B'
0	1	1	1	F = A + B' plus CIN	F = A + B'
1	0	0	0	F = A plus (A + B) plus CIN	F = A' · B
1	0	0	1	F = A plus B plus CIN	F = A ⊕ B
1	0	1	0	F = A · B' plus (A + B) plus CIN	F = B
1	0	1	1	F = A + B plus CIN	F = A + B
1	1	0	0	F = A plus A plus CIN	F = 0000
1	1	0	1	F = A · B plus A plus CIN	F = A · B'
1	1	1	0	F = A · B' plus A plus CIN	F = A · B
1	1	1	1	F = A plus CIN	F = A

6. Combinational Practices

- Adders, Subtractors and ALUs (9) -

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vaddshr is
    port (
        A, B, C, D: in SIGNED (7 downto 0);
        SEL: in STD_LOGIC;
        S: out SIGNED (7 downto 0)
    );
end vaddshr;

architecture vaddshr_arch of vaddshr is
begin
    S <= A + B when SEL = '1' else C + D;
end vaddshr_arch;
```

6. Combinational Practices

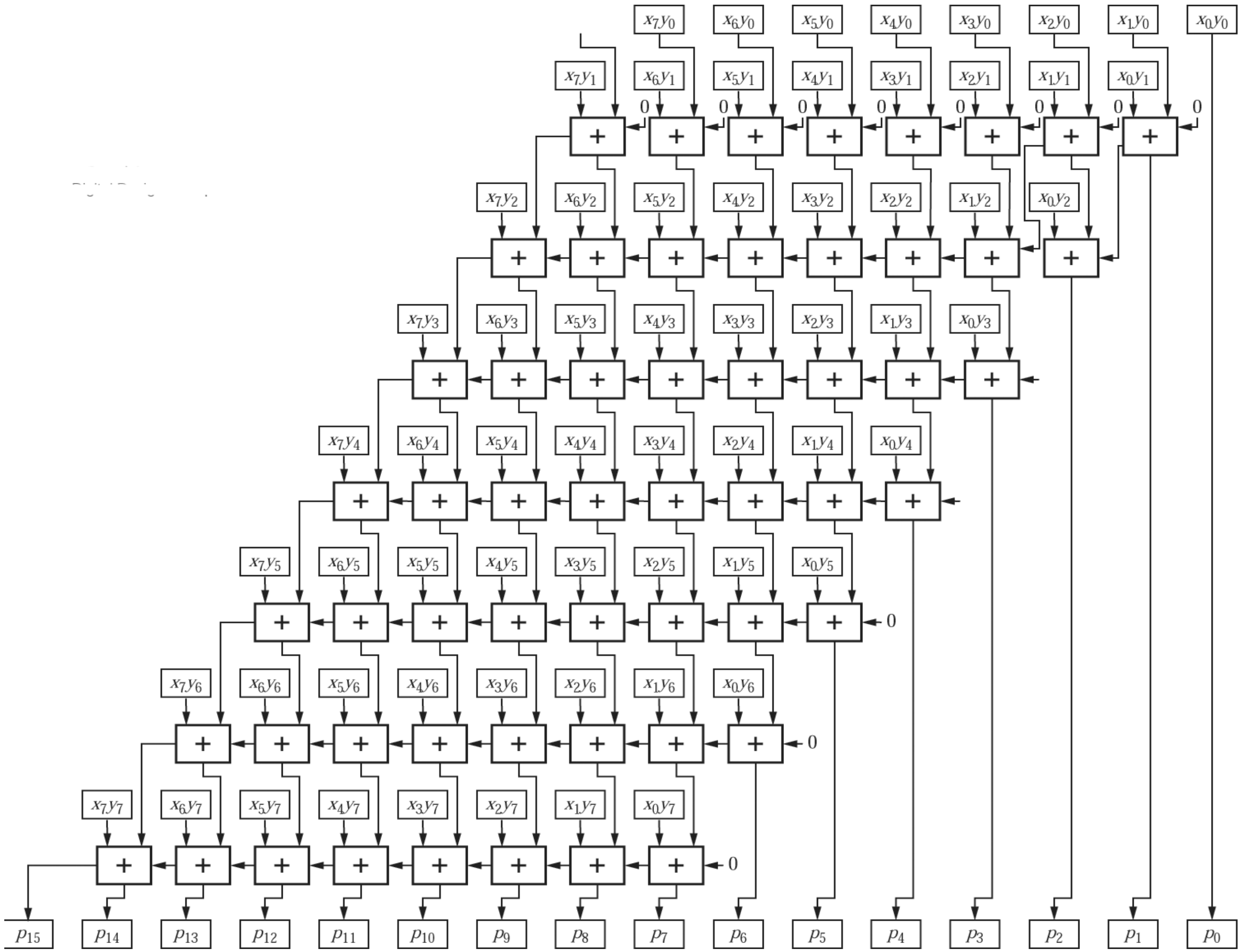
- Multipliers (1) -

- The traditional algorithm to multiply binary numbers uses shifts and adds to obtain the result.
- However, it is not the only solution to implement a multiplier.
- Given 2 n-bit inputs (X, Y), we can write a truth table that expresses the 2n-bit product P=X×Y as a combinational function of X and Y.
- Most approaches to combinational multipliers are based on the traditional shift-and-add algorithm.

									y_0x_7	y_0x_6	y_0x_5	y_0x_4	y_0x_3	y_0x_2	y_0x_1	y_0x_0
								y_1x_7	y_1x_6	y_1x_5	y_1x_4	y_1x_3	y_1x_2	y_1x_1	y_1x_0	
							y_2x_7	y_2x_6	y_2x_5	y_2x_4	y_2x_3	y_2x_2	y_2x_1	y_2x_0		
						y_3x_7	y_3x_6	y_3x_5	y_3x_4	y_3x_3	y_3x_2	y_3x_1	y_3x_0			
					y_4x_7	y_4x_6	y_4x_5	y_4x_4	y_4x_3	y_4x_2	y_4x_1	y_4x_0				
				y_5x_7	y_5x_6	y_5x_5	y_5x_4	y_5x_3	y_5x_2	y_5x_1	y_5x_0					
			y_6x_7	y_6x_6	y_6x_5	y_6x_4	y_6x_3	y_6x_2	y_6x_1	y_6x_0						
+		y_7x_7	y_7x_6	y_7x_5	y_7x_4	y_7x_3	y_7x_2	y_7x_1	y_7x_0							
	p_{15}	p_{14}	p_{13}	p_{12}	p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

6. Combinational Practices

- Multipliers (2) -



6. Combinational Practices

- Multipliers (3) -

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vmul8x8i is
    port (
        X: in UNSIGNED (7 downto 0);
        Y: in UNSIGNED (7 downto 0);
        P: out UNSIGNED (15 downto 0)
    );
end vmul8x8i;

architecture vmul8x8i_arch of vmul8x8i is
begin
  P <= X * Y;
end vmul8x8i_arch;
```