YORK
UNIVERSITÉ
UNIVERSITY

EECS 3201: Digital Logic Design Lecture 8

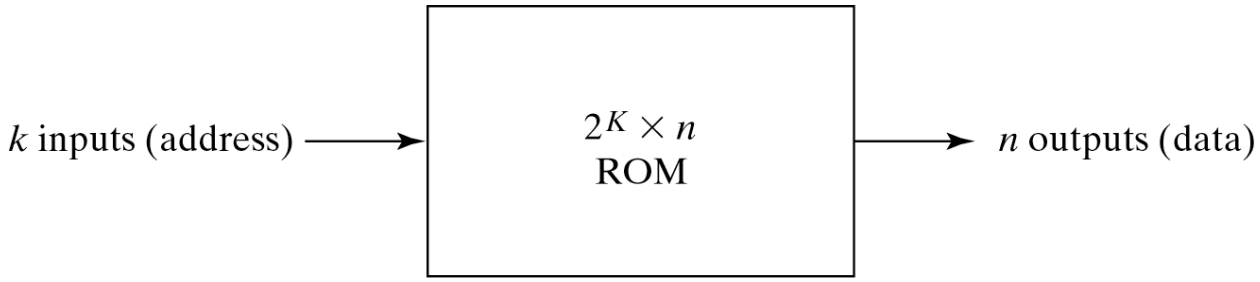
Ihab Amer, PhD, SMIEEE, P.Eng.

Midterm Coming Soon!

- Midterm will be on Wednesday, October 22nd, 2014
- Exam will take place during the lecture time
- Exam will include all material of Lectures (1–8). This includes material on website + related material in textbook
- Verilog HDL is part of the content

Read-Only Memory (ROM)

- A memory device in which permanent binary information is stored
- It is sometimes looked at as a programmable logic device



ROM Block Diagram

Types of ROM

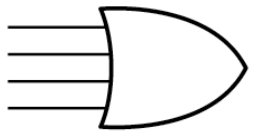
- Mask ROM: The *programming* is done by the semiconductor company during the last fabrication process of the unit
- PROM: A fused link is burned open during the programming process. Once the PROM is programmed, it cannot be reversed
- EPROM: An erasable PROM that can be erased by exposure to UV light
- EEPROM: Can be erased and programmed with electrical pulses
- Flash Memory: High-density read/write memories that are nonvolatile. They have the ability to retain charge for years with no applied power

Programmable Logic Devices (PLDs)

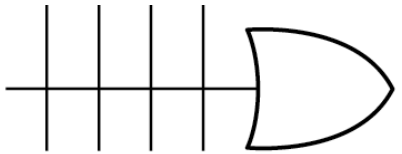
- An IC with internal logic gates connected through electronic paths
- A logic function can be *programmed* into a PLD after manufacture
- Programming here refers to a hardware procedure, which specifies the bits that are inserted into the hardware configurations of the device

Gate Arrays

- A typical PLD may have an array of hundreds to millions of gates interconnected through internal paths
- The figure shows the conventional and array logic symbols for a multiple input OR gate

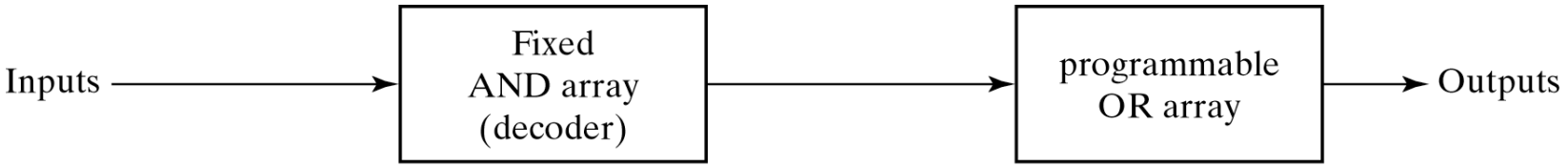


(a) Conventional symbol

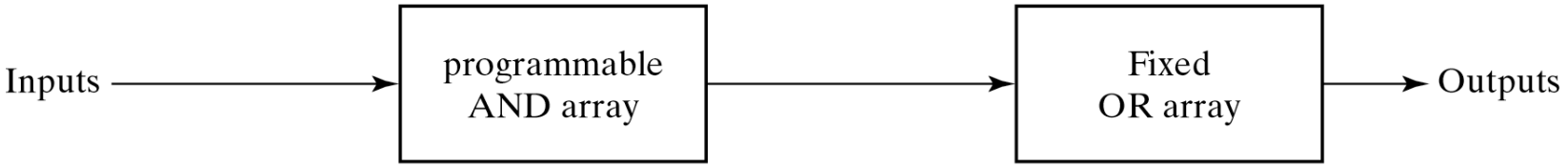


(b) Array logic symbol

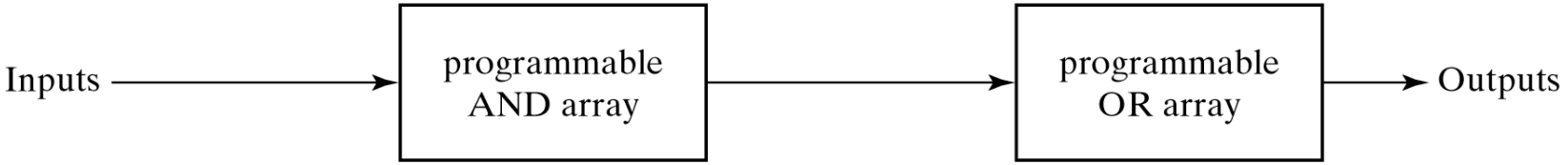
Types of Combinational PLDs



(a) Programmable read-only memory (PROM)



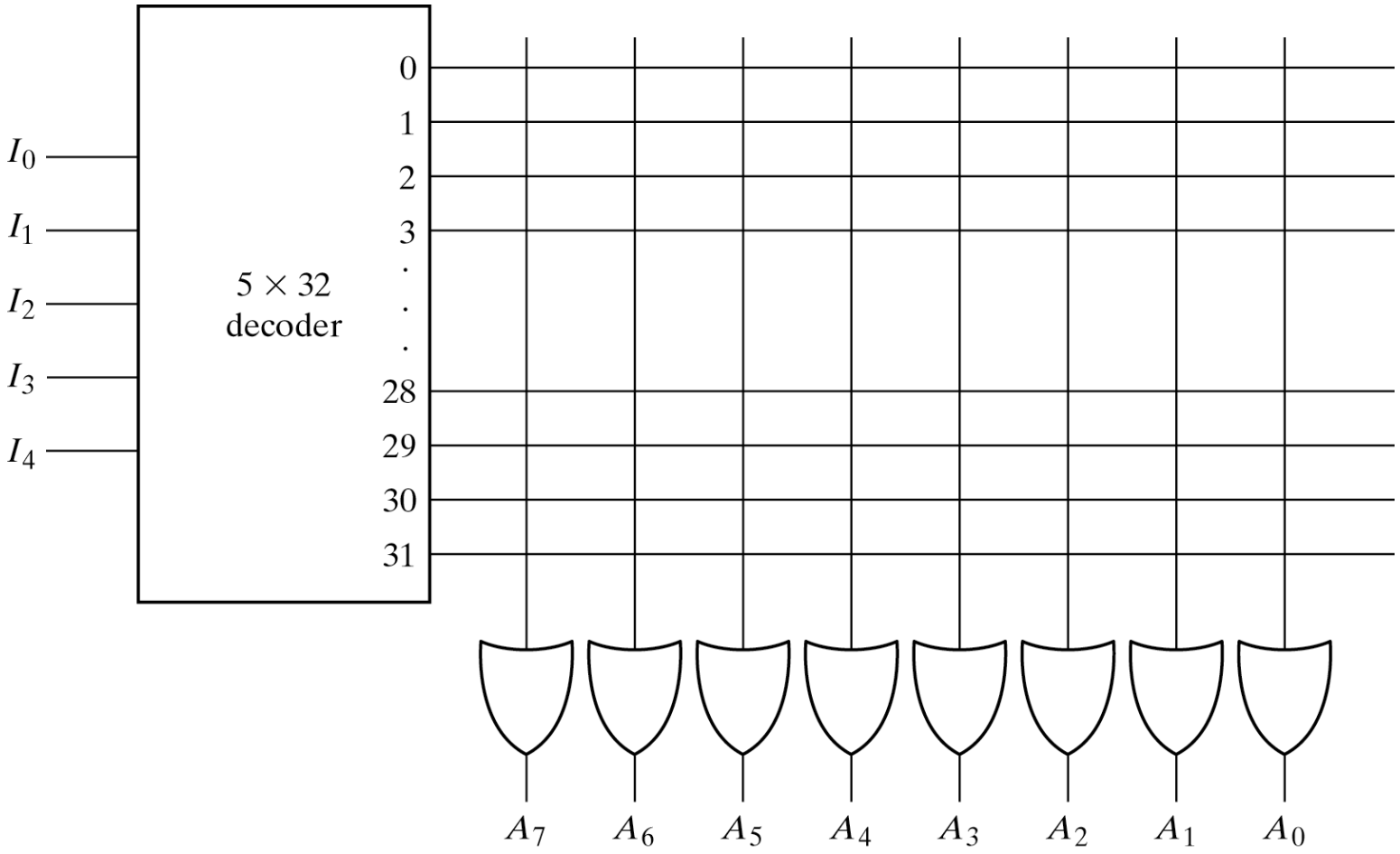
(b) Programmable array logic (PAL)



(c) Programmable logic array (PLA)

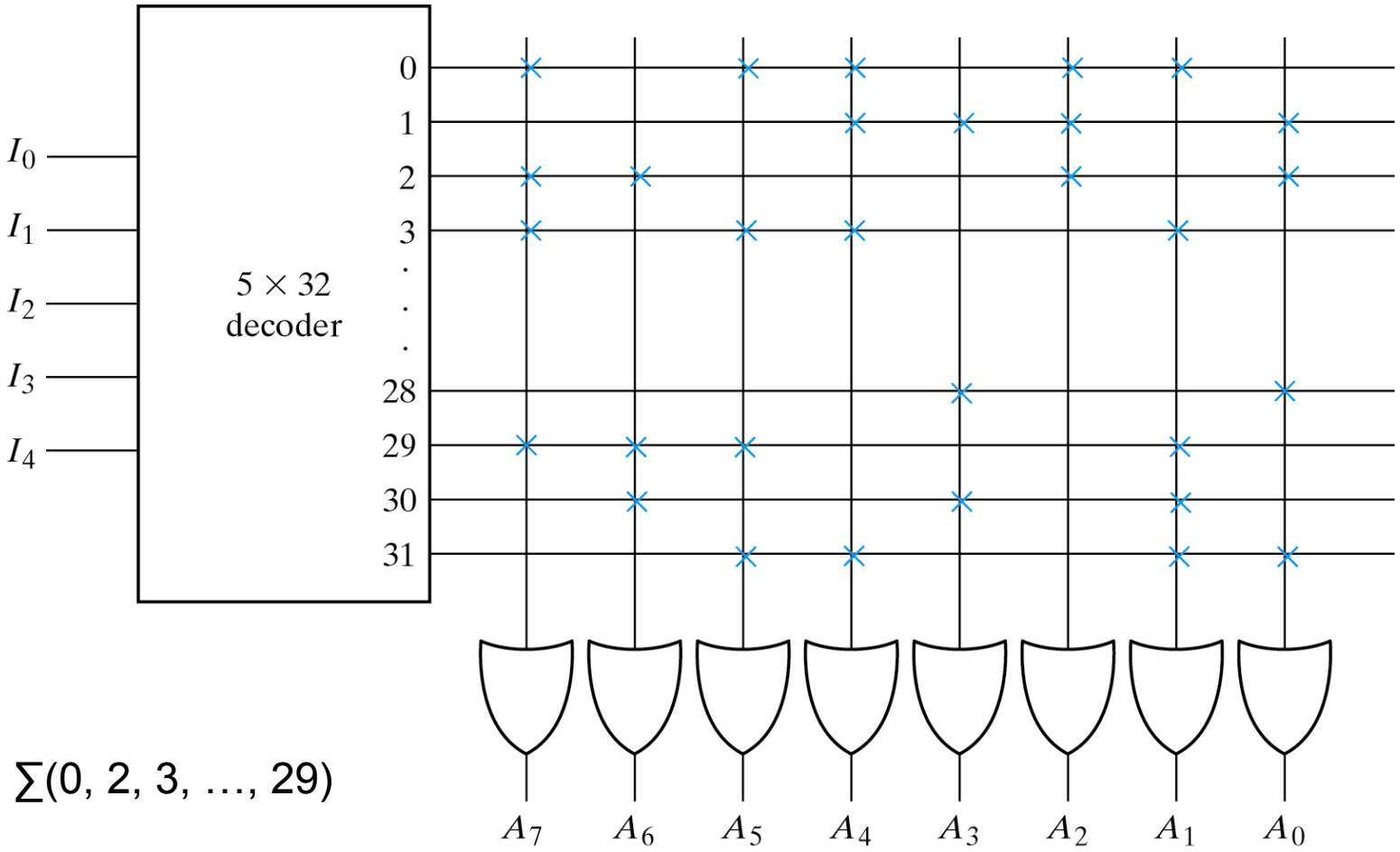
Basic Configuration of Three PLDs

PROM Internal Structure



Internal Logic of a 32 × 8 ROM

Programmed PROM



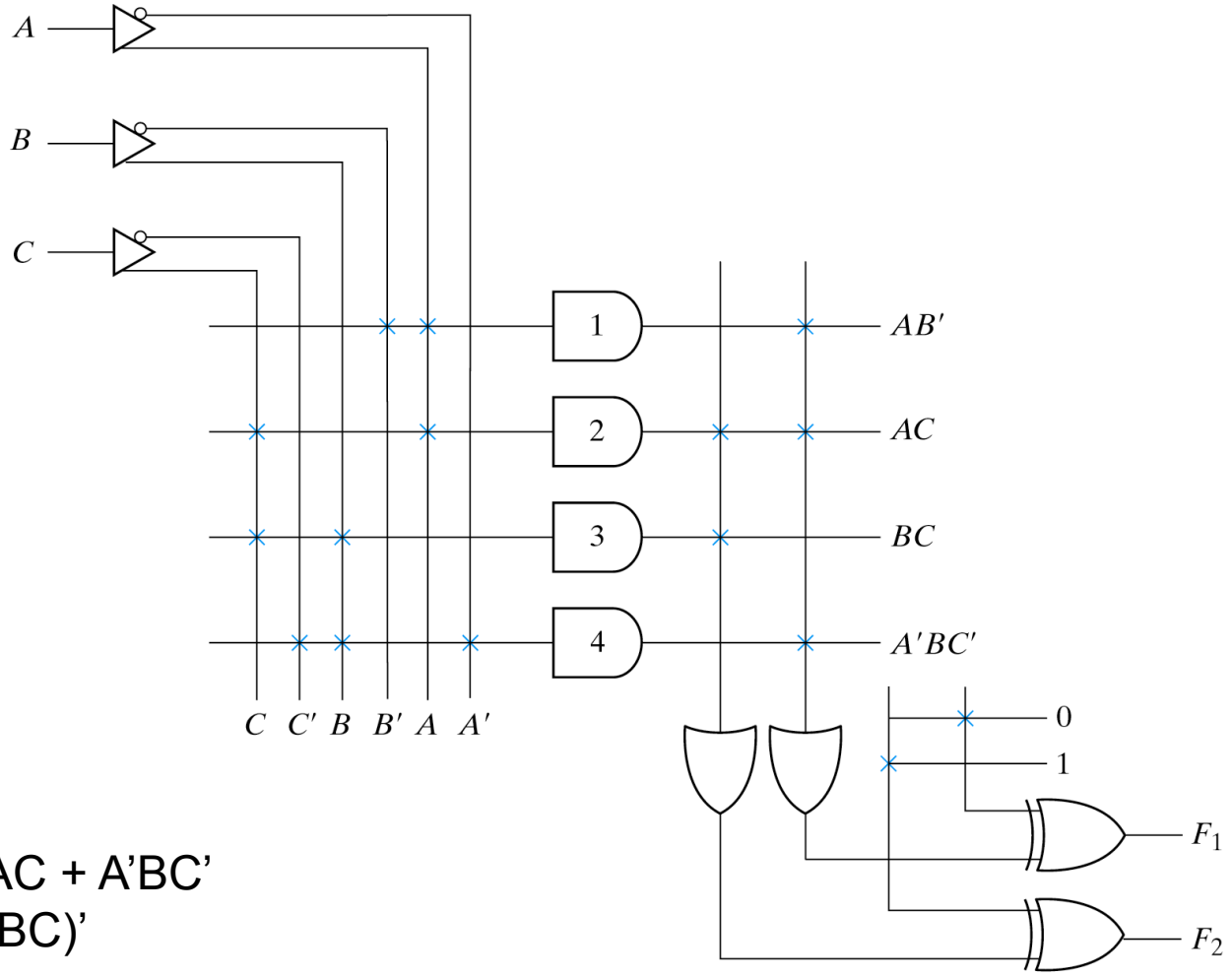
$A_7 = \sum(0, 2, 3, \ldots, 29)$

Programming the ROM

Reading Assignment

- Example 7.1 in Mano textbook

PLA



$F_1 = AB' + AC + A'BC'$

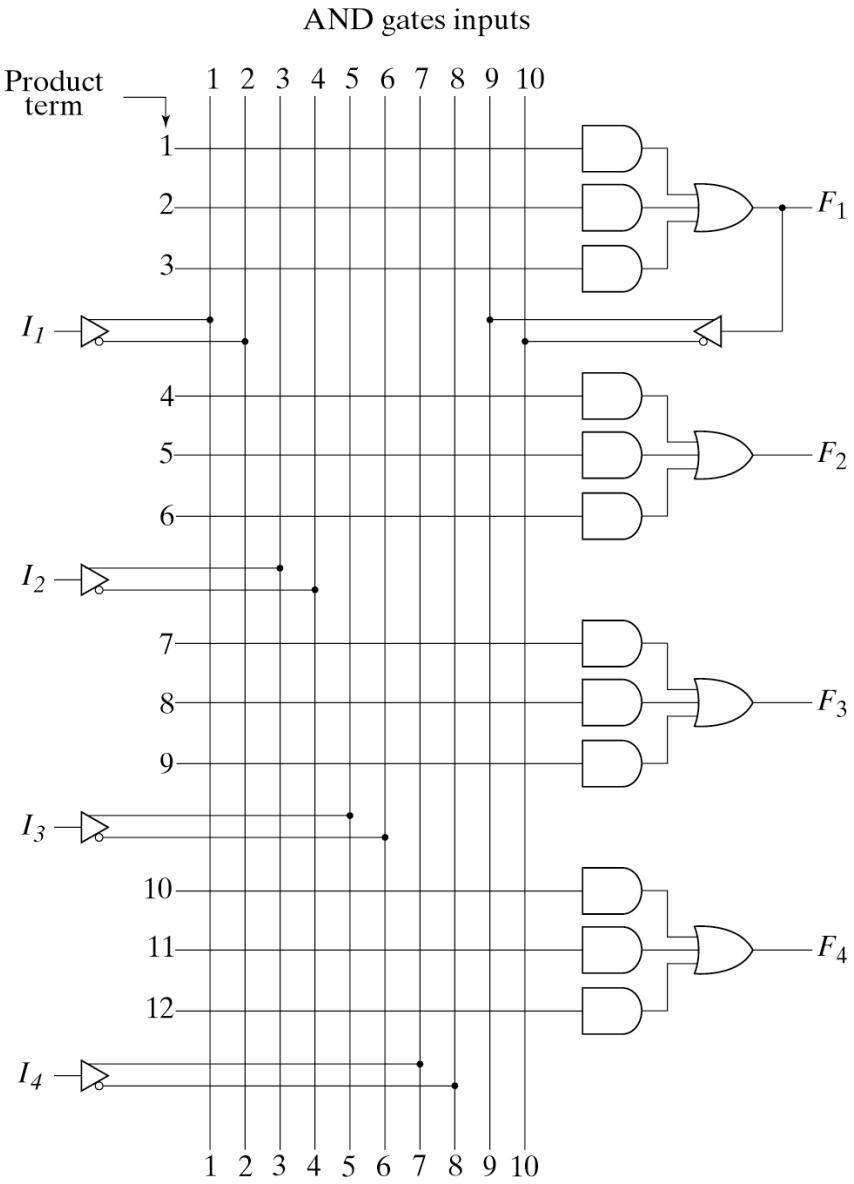
$F_2 = (AC + BC)'$

PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

Reading Assignment

- Refer to Table 7.5 in Mano textbook for the PLA Programming Table
- Example 7.2 in Mano textbook

PAL



PAL with Four Inputs, Four Outputs, and Three-Wide AND-OR Structure

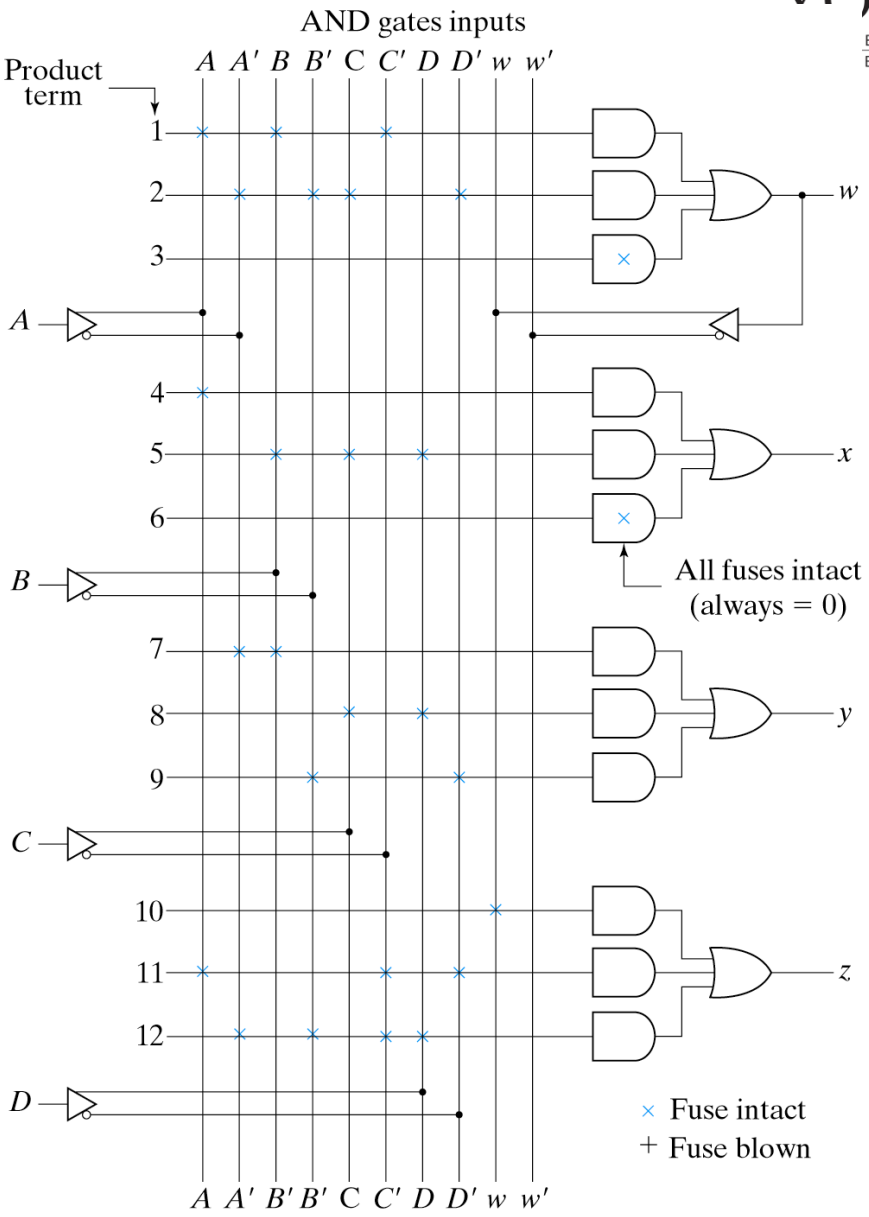
Programmed PAL

w = ABC’ + A’B’CD’

x = A + BCD

y = A’B + CD + B’D’

z = ABC’ + A’B’CD’ + AC’D’ + A’B’C’D
= w + AC’D’ + A’B’C’D



Fuse Map for PAL

Reading Assignment

- Table 7.6 in Mano textbook

Binary Coded Decimal (BCD)

- Using 4 binary numbers to represent decimal digits
- 82 => 1000_0010
- 18 => 1_1000
- How do you add these numbers?
  - If X + Y = Z < 9 nothing new
  - What if Z > 9?

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

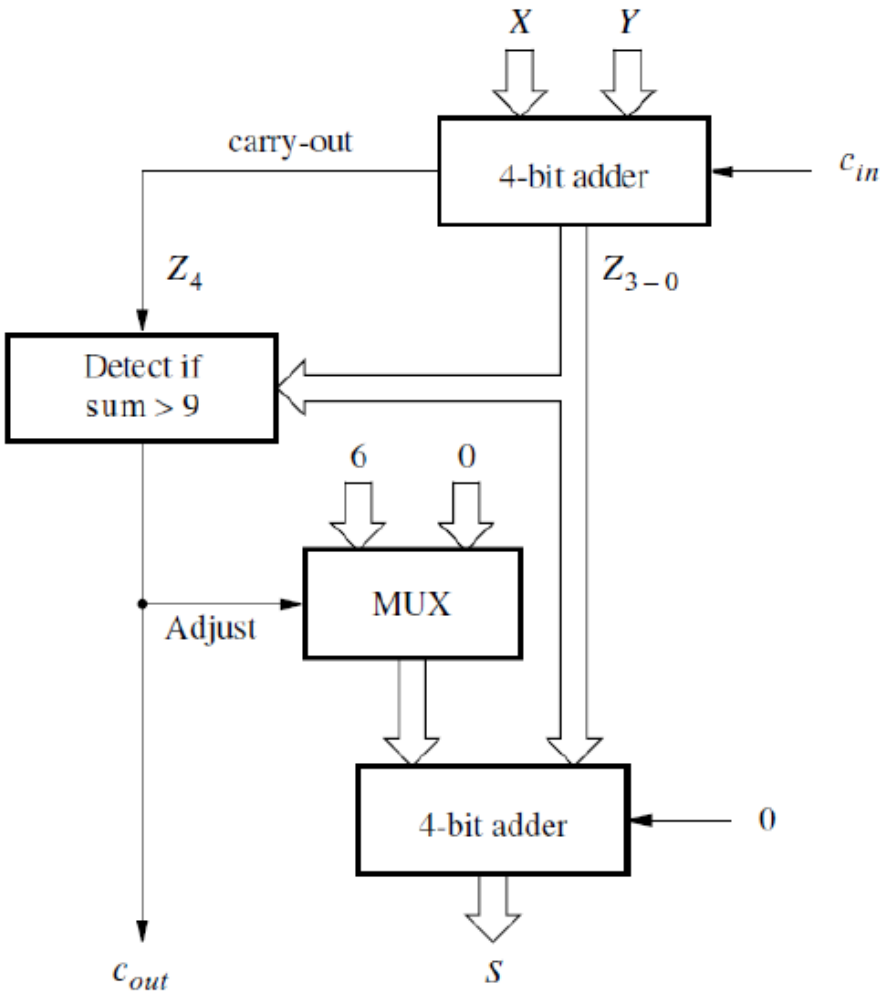
BCD Addition

- 4-bit numbers addition is subject to a modulo-16 scheme (values 0-15)
- If they are used to represent numbers 0-9 only, whenever I exceed 9 I have to add 6 to the sum to get the correct BCD representation

X	0 1 1 1	7
+ Y	+ 0 1 0 1	+ 5
Z	1 1 0 0	12
	+ 0 1 1 0	
carry →	1 0 0 1 0	
	S = 2	

X	1 0 0 0	8
+ Y	+ 1 0 0 1	+ 9
Z	1 0 0 0 1	17
	+ 0 1 1 0	
carry →	1 0 1 1 1	
	S = 7	

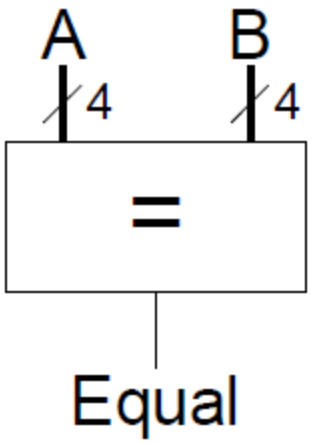
BCD 1-digit Adder



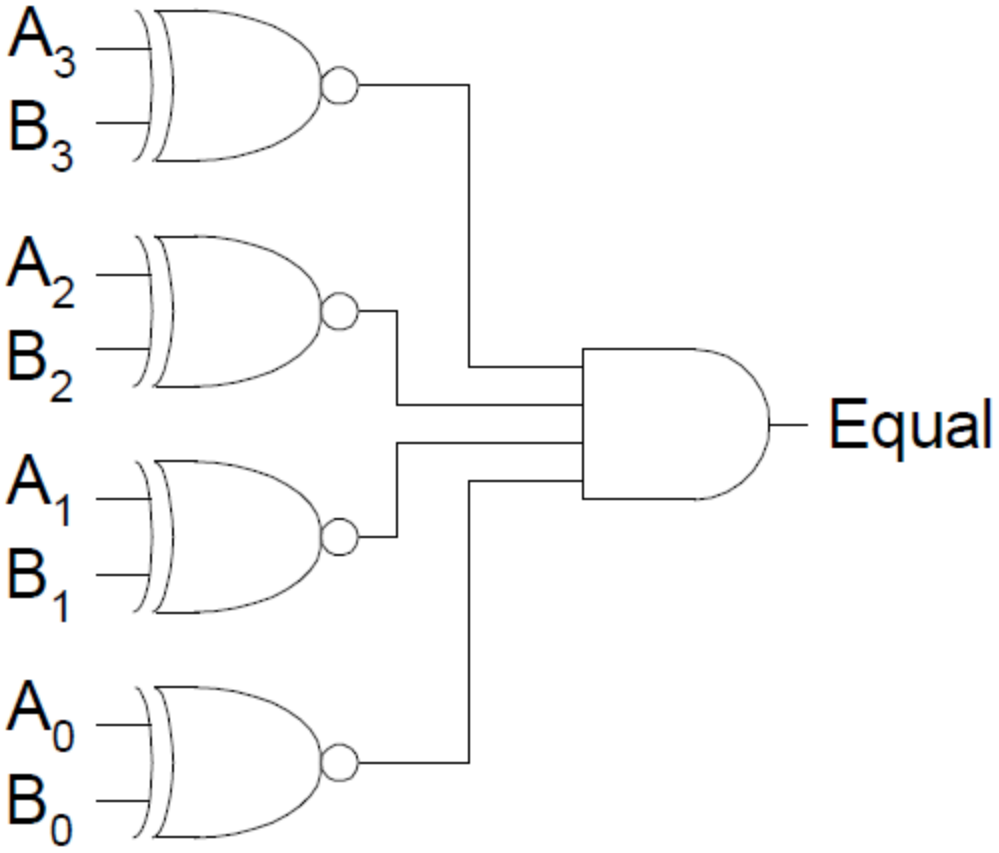

```
module bcdadd(Cin, X, Y, S, Cout);
    input Cin;
    input [3:0] X, Y;
    outputreg [3:0] S;
    outputreg Cout;
    reg [4:0] Z;

    always @ (X, Y, Cin)
    begin
        Z = X + Y + Cin;
        if (Z < 10)
            {Cout, S} = Z;
        else
            {Cout, S} = Z + 6;
    end

endmodule
```

Comparator: Equality

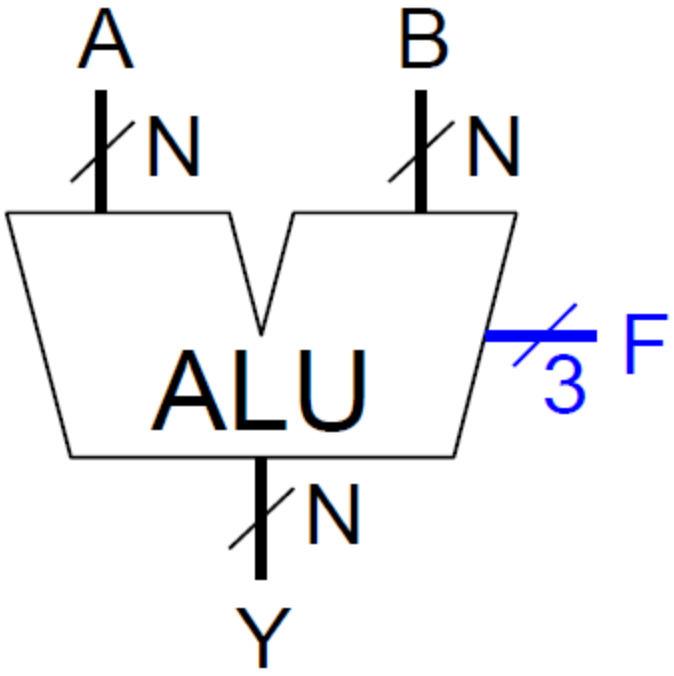
Symbol



Implementation

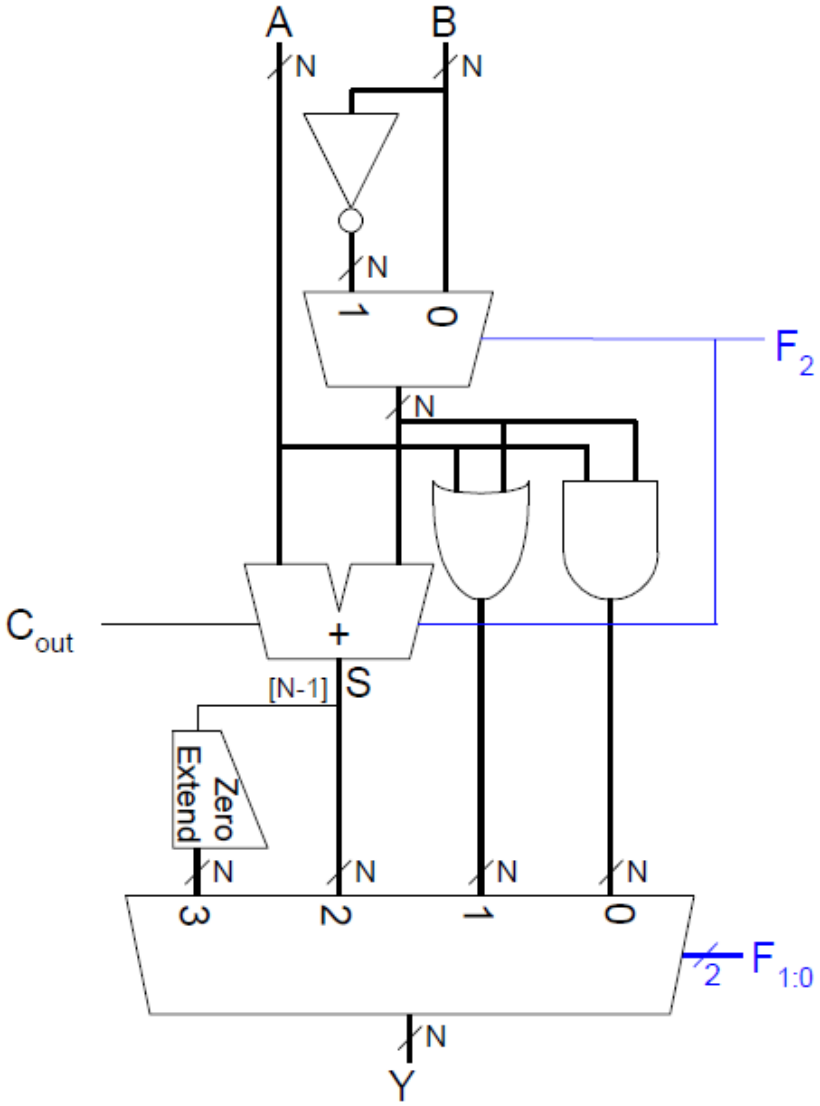


Arithmetic Logic Unit (ALU)



$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

ALU Design



$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

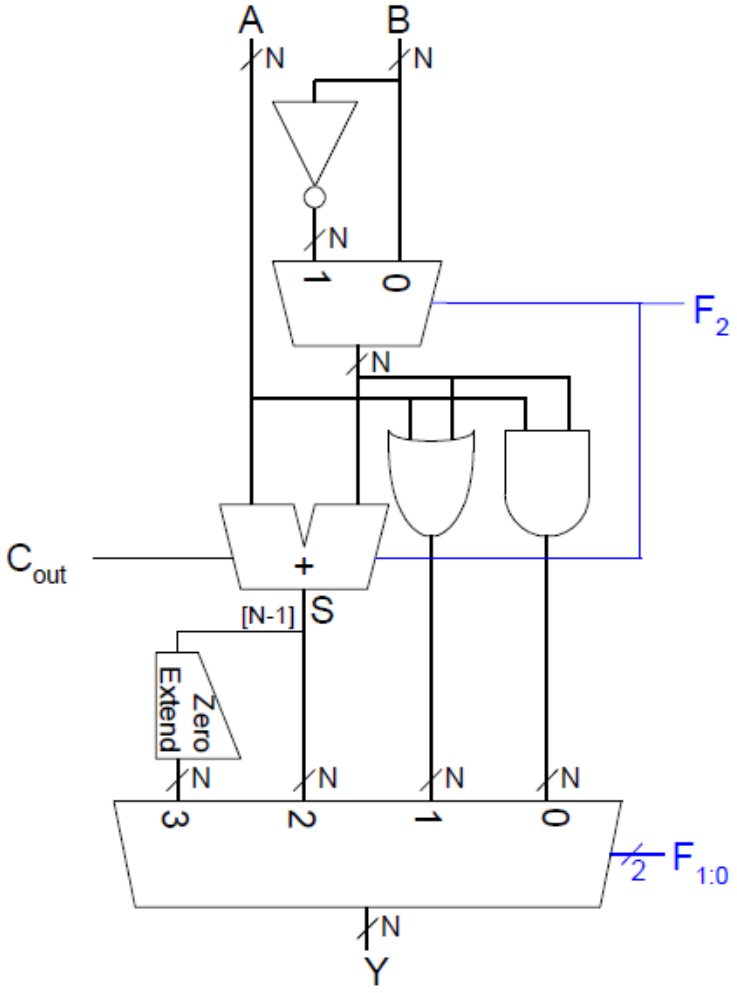
SLT Function

- slt function is defined as:

$$A \text{ slt } B = \begin{cases} 000 \ldots 001 & \text{if } A < B, \text{ i.e. if } A - B < 0 \\ 000 \ldots 000 & \text{if } A \geq B, \text{ i.e. if } A - B \geq 0 \end{cases}$$

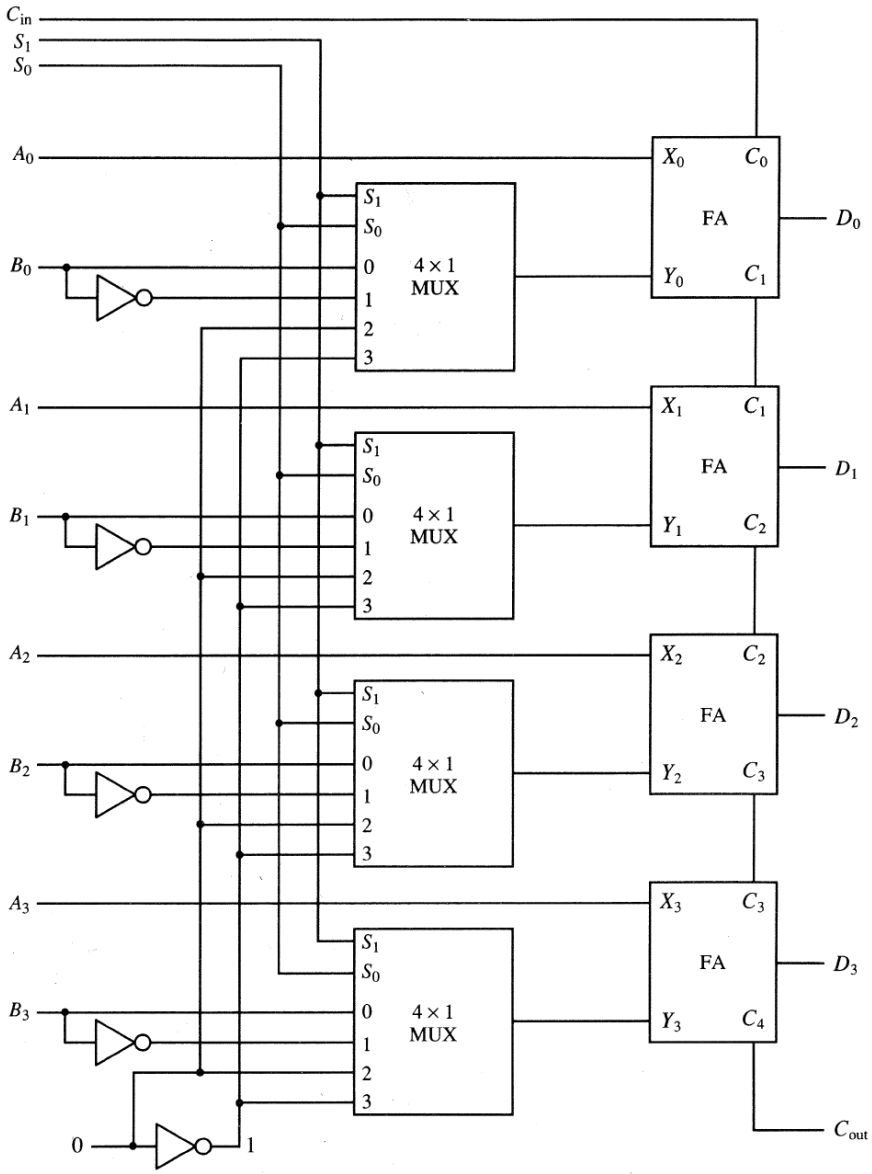
- Thus, each 1-bit ALU should have an additional input (called "Less"), that will provide results for slt function. This input has value 0 for all but 1-bit ALU for the least significant bit.
- For the least significant bit Less value should be sign of A – B

SLT Example



- Configure 32-bit ALU for SLT operation: $A = 25$ and $B = 32$
  - $A < B$, so Y should be 32-bit representation of 1 (0x00000001)
  - $F_{2:0} = 111$
    - $F_2 = 1$ (adder acts as subtracter), so 25 - 32 = -7
    - -7 has 1 in the most significant bit ($S_{31} = 1$)
    - $F_{1:0} = 11$ multiplexer selects $Y = S_{31}$ (zero extended) = 0x00000001.

4-bit Arithmetic Circuit



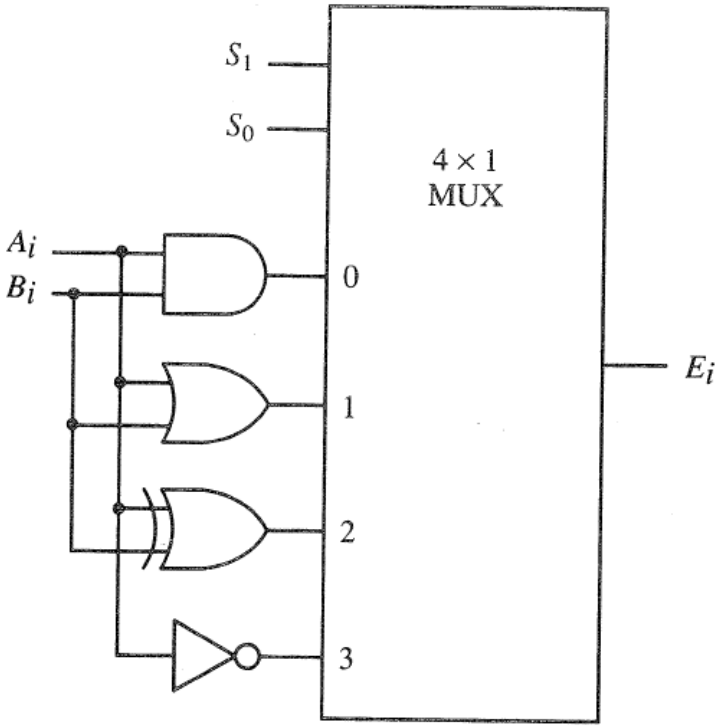
Function Table

Select S_1	Select S_0	Select C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Thoughts!

Can you reverse-engineer this circuit?

Hardware Implementation



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

One stage of a logic circuit

References

- Lecture Notes of Dr. Sebastian Magierowski – Fall 2013
- “Digital Design, Morris Mano , Prentice Hall