

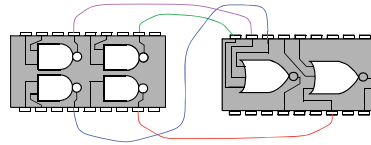


## Constructing Digital Circuits

### Hand Wired Circuits

Circa 1970-85

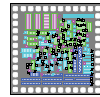
- Make 2 to 4 silicon gates in a package.
- Connect with wires.



### VLSI circuits

Start with a silicon wafer and make:

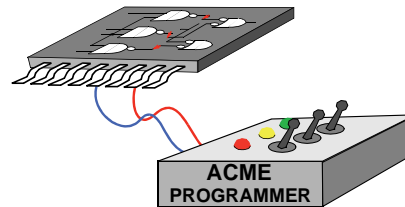
- the gates
- the interconnections on top both made together.



### Field Programmable circuits

Start with a silicon wafer and make:

- gates with no connections.
- Make connections later using:
  - 1) electrical means
    - blow fuses, grow anti fuses
    - use memory to hold connections
  - 2) deposit metal lines on top of silicon.



### Micro Controllers

Printed: 11/02/09  
Modified: February 11, 2009

Department of Electronics, Carleton University  
© John Knight

Slide 1  
Array Logic p. 1

## Constructing Logic Circuits

### Connecting Small Gates

Labour intensive, power hungry, large. Good for one or two gates and undergraduate experiments.

### VLSI

Very-large-scale integration. The glamour chips are made this way. Several million gates can be but on a cm<sup>2</sup> of silicon.

The initial masks and manufacturing setups cost a million dollars. After that, production costs are a few dollars per chip.

### Programmable Logic

These are very common. The basic design can be made by the millions. Since they are programmable one basic circuit can be adapted to many applications without having an initial million dollar setup cost.

### Microcontrollers

Another way of effectively doing logic is with a small microcontroller. These are small microcomputers which are made in quantities of tens of millions. They can be bought for about a dollar and are usually the best way to implement digital control circuits. For some applications, they may not be fast enough, they may not have enough outputs of the right type, or they may take too much power, but they should be a designers first thought when they considers implementation.

Every microwave oven is controlled by such a microcontroller. If you have a keypad on an electrical device, suspect a microcontroller.



### Programmable Logic

#### Different Ways of Connecting Logic

- 1. Connect Wires**  
Few gates on a chip
- 2. Electrically Programmable**  
Many gates on a chip

**Electrically Controlled Switches**

Fuses; blow to remove unwanted connections

Antifuses; grow to make connections
- 3. Mask Programmable**  
Many gates on a chip

**Metal blobs deposited over ends of wires to be connected**

## Connecting Logic

### 1. Connecting with wires

Done for small amounts of logic, which will have only a few copies made.  
Labour intensive.

### 2. Electrically controlled switches make the connections

Most common way to implement logic.  
Used for up to 500,000 gates.  
Some types are programmed once, for example by blowing internal fuses to disconnect unwanted connections. Other types have to be reprogrammed every time the power is shut off.  
The programming is relatively slow, milliseconds to seconds.  
Good for a few copies but mask programming is cheaper for over say fifty thousand copies.

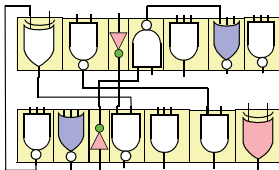
### 3. Mask programmed

This must be done in the factory.  
Making the underlying silicon takes at least a hundred million dollar factory. Placing the last metal connections on is a much simpler process and can be done in a half million dollar factory.  
Used for up to 2,000,000 gates.  
Good when hundreds of thousands of copies will be made.  
Chips are put in a vacuum. Metal is exploded over the top to place a thin metal layer over everything. A photo sensitive polymer coating is placed on top and exposed to light where the metal is to be removed. The light exposed coating is dissolved, uncovering most of the metal. This uncovered metal is removed with acid, leaving metal only over the connections.

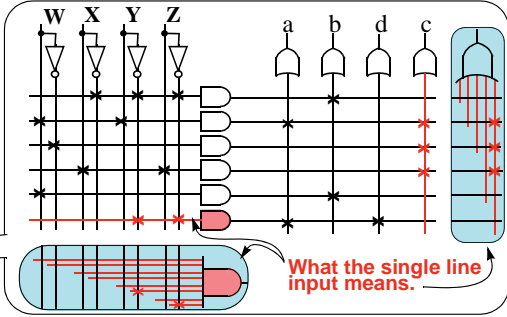


### Arrangements of Logic On Silicon

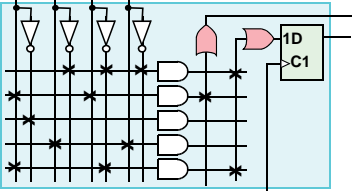
**Random Logic**  
Not really random but rows of gates



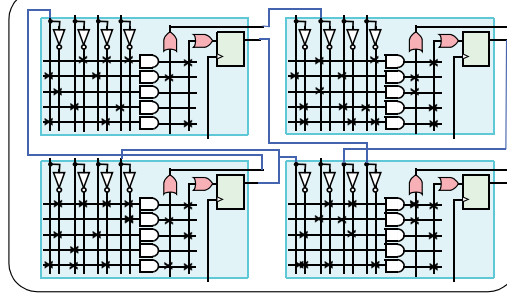
**Array Logic**  
Sum of Products logic.  
Logic arranged in regular array.  
Change cross connections × to change logic



**Logic Blocks**  
Small block of array logic  
With flip-flop



**Many blocks on a chip**



## Arrangement of Logic

### Random Logic

This is the way most widely sold commercial chips are built. Microprocessors and memory are the major exceptions. Modems, graphics chips and sound chips and smoke detectors are usually built this way.

The circuit is fabricated on silicon with rows of predesigned gates and the leads interconnecting them. This is called *standard cell* design.

### Array Logic

The logic is simple,  $\Sigma$  of  $\Pi$  and ranges in size from a T-bird tail light controller to USB interface for your PC. The logic can usually be programmed after it if fabricated either electrically or by a mask.

### Logic Blocks

The array logic is usually grouped into logic blocks. Each block has one or more storage units on the output of the array.

If each array is fairly large, ten to twenty inputs, and there are not too many on a chip, they are called CPLDs (complex logic devices) as well as many other things. If each array has only four or five inputs, with hundreds on the chip, they are called FPGAs (Field Programmable Gate Arrays). In FPGAs, there are many wires interconnecting logic blocks which can be connected during programming. In CPLDs the connections between blocks are usually very simple with little programmability. However this makes them faster.

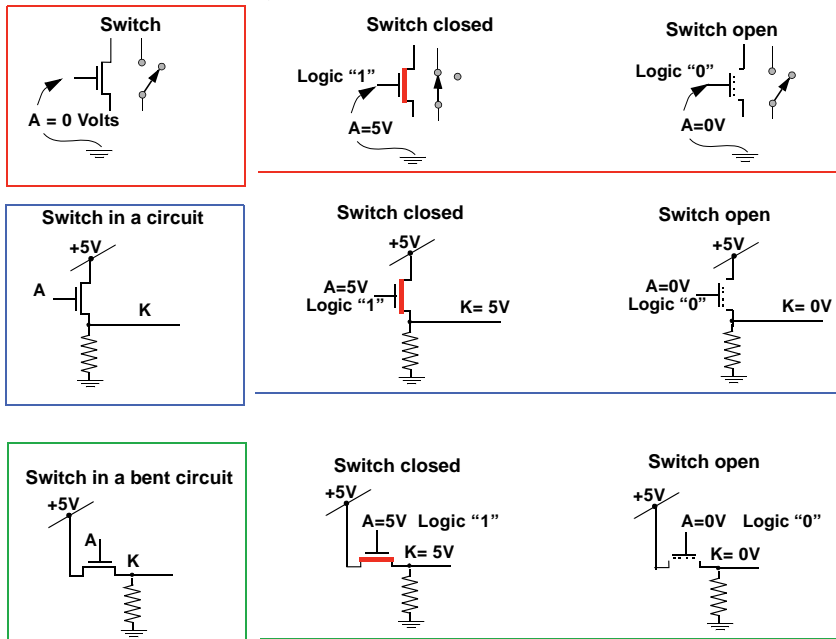
### Microcontrollers

Small microprocessors are both programmable and very inexpensive. Their price and flexibility make them the method of choice for many logic applications. Their disadvantage is that they are slower and use more power than the methods above. The logic in microwave ovens is controlled by a small microcontroller, which costs under a dollar.



### Array Logic

Transistors as electrically controlled switches.



Printed: 11/02/09  
Modified: February 11, 2009

Department of Electronics, Carleton University  
© John Knight

Slide 4  
Array Logic p. 7

### Transistors as switches

#### Transistor Switches

When these transistors are on, there is a conducting band of electrons under the gate. This is shown by a red band. When the transistor is off, the electrons are removed, which we represent with a few dots.

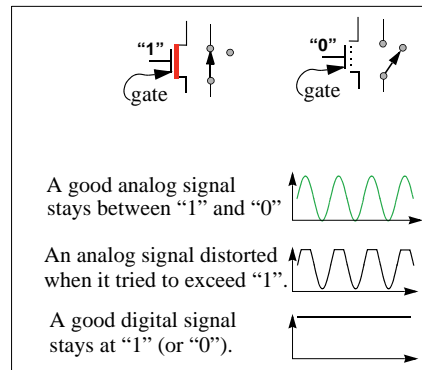
#### For Persons Taking an Electronics Course

##### Digital Circuits Would Distort Analog Signals

Analog courses deal with linear amplifiers. In these amplifiers one does not want the output to be 0 or 1, but somewhere in between.

##### Offset Details about NMOS FETs

The transistor shown is an NMOS (N-type metal-oxide-semiconductor) FET (field-effect transistor). The circuit is over simplified. The transistor shown will not be a perfect switch. There will be about a one volt<sup>1</sup> drop across the switch when it is on. In a practical circuit one might use a PMOS transistor which would avoid the drop for the circuits shown. However PMOS is confusing in a first course, because it turns on for 0 V and off for 5V. Hence we will draw the transistors as NMOS and ignore the drop for now.

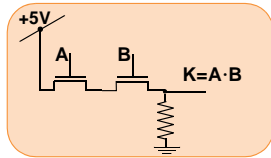


1. This "threshold" drop used to be about one volt. In modern NMOS transistors it is likely closer to a half volt.



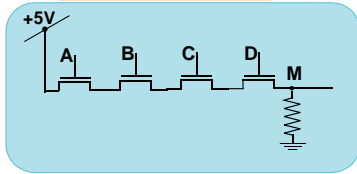
### Array Logic

#### Transistor AND logic.



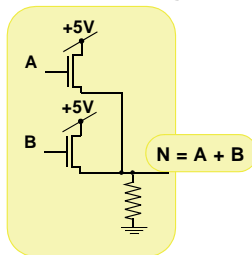
Voltages			Logic levels		
A	B	K	A	B	K
5V	5V	5V	1	1	1
5V	0V	0V	1	0	0
0V	5V	0V	0	1	0
0V	0V	0V	0	0	0

$K = A \cdot B$

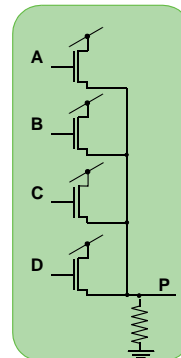


$M = A \cdot B \cdot C \cdot D$

#### Transistor OR logic.



Logic levels		
A	B	N
1	1	1
1	0	1
0	1	1
0	0	0



$P = A + B + C + D$

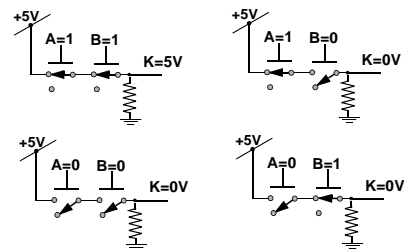
### Transistor ANDs and ORs Used In Array Logic

#### AND gate operation

When the inputs A and B are both “1” think of the two transistors as being like switches that are both closed

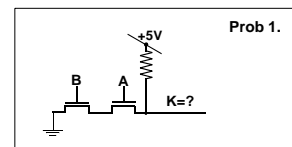
When the inputs A = “1” and B = “0” think of the A transistors as a closed switch and the B transistor as an open switch. Then K is connected to 0V (ground) through a resistor. Since no current is flowing in the resistor, it has no voltage across it, and K is at “0” volts.

When both inputs A and B are “0” the output K is at 0V.



1. • PROBLEM

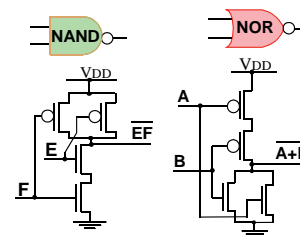
What logic function is performed by the circuit on the right?



#### Compare with Previous gate Circuits

The previous gates are shown. These gates take twice as many transistors, and thus more space. However they are faster than the array logic gates above, and we shall soon see that array logic must waste space for unused logic combinations.

The main advantage of the array logic gates is their connections are simple and easily programmed.





### Array Logic

Combine the AND and OR logic

$$P = A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D$$

### A Fuse Programmed Logic Array

Program the logic  
Short all transistors with fuses.  
Blow fuses across transistors  
needed for the AND terms.

$$P = A \cdot B + B \cdot C + C \cdot D + A \cdot D$$

### Fuse Programming

Start with all transistors shorted.

Zap the short where you want a transistor

Leave a working transistor

Printed: 11/02/09  
Modified: February 11, 2009

Department of Electronics, Carleton University  
© John Knight

Slide 6  
Array Logic p. 11

## Fuse Programmable Array Logic

2. • PROBLEM

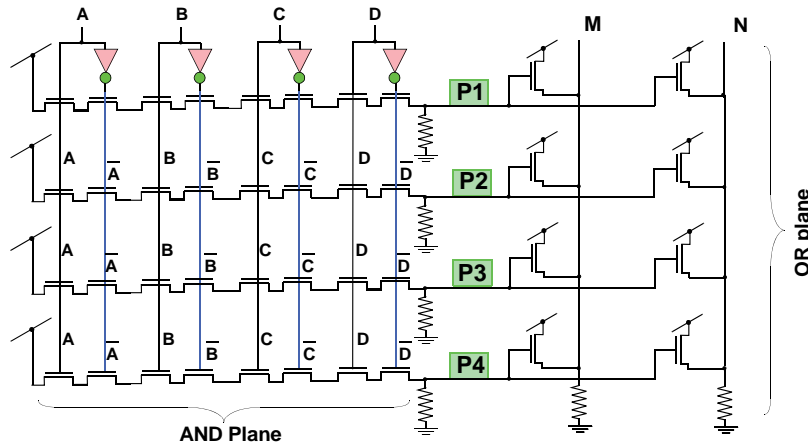
To implement  $P=A(B + CD)$ :

Into what form does one have to change the expression for P?

Program  $P=A(B + CD)$  on the array above on the left?

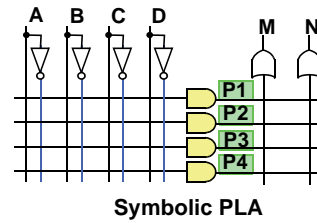


### Programmable Logic Array (PLA)



Two outputs M and N  
 Can implement two  $\Sigma$  of  $\Pi$  functions  
 with 4 inputs which includes  $x$  and  $\bar{x}$   
 and with 4 product terms, P1, P2, P3 and P4.

Use the simple symbol for logic design



### A Complete PAL

#### Number of Product lines Needed

Four product lines is small. The worst case  $\Sigma$  of  $\Pi$  expression for a PAL is one where the Karnaugh map looks like a checker board with "1"s in all the diagonal squares. This has  $2^N/2$  product terms for N inputs. This is the worst case because if any more "1"s are added, they can be circled with other "1"s and thus need no more product lines.

However, most circuits can get by with far fewer product terms.



### Programming the PLA

**AND Plane**  
Blow fuses for letters you want in the AND term

**OR Plane**  
Blow fuses to remove unwanted ORs

**Symbolic PLA**  
Put an X where you want a connection

Slide 8  
Array Logic p. 15

## Programming a PLA

### Blowing the Fuses

#### PLAs which are programmed by blowing fuses

This are programmed by special circuitry which puts a higher than normal voltage on the fuses that are to be blown. Other types of PLA use an antifuse which make a conductive path by melting a thin insulating layer between two conducting lines.

#### The symbolic PLA

This type of symbol will be used from now on for array type logic. It shows all connections as X, independent of whether they were fuses or antifuses, or whether they were transistors that appeared when a short across them was removed, or they were the connections that survived when all others were blown.

#### 3. PROBLEM

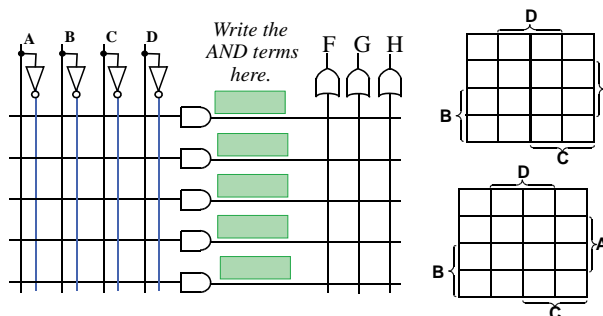
Program the PLA shown to implement

$$F = AC\bar{D} + ABC\bar{C} + CDB\bar{C} + \bar{A}BCD;$$

$$G = A\bar{D} + CDB\bar{C} + AC + AB$$

The PLA is over generous on the OR plane, but you will need to work to squeeze in enough product terms.

*Hint use the Karnaugh maps and share terms.*





**PLA for 7-Segment Display Driver,**

**These maps were minimized for a gate implementation**

$J = YZ$   
 $K = \bar{X}\bar{Z}$   
 $L = \bar{Y}\bar{Z}$   
 $M = \bar{X}Y$   
 $N = XYZ$   
 $P = Y\bar{Z}$   
 $R = X\bar{Y}$

$a = J + W + K + N$

$b = J + L + \bar{X}$

$c = Z + \bar{Y} + X$

$d = N + M + P + K$

$e = K + P$

$f = L + W + X$

$g = W + M + P + R$

**With PLAs circles of all sizes cost the same  
Halfmaps have no advantage.**

**PLA Must Have**  
 4 inputs  
 7 outputs => 7 OR plane (vertical) lines  
 $7 + W + X + \bar{X} + \bar{Y} + Z$   
 = 12 product terms (horizontal lines)

### The Minimum PLA

The equations shown on Slide 11, were picked to reduce gate and gate-input count. This is not optimum for a PLA. The PLA size is governed by the number of product terms. The number of inputs (4 here) and the number of outputs (7 here) is fixed by the problem. The only simplification is to reduce the number of product terms down from 12. To do this, forget some rules:

- A smaller map circle saves nothing. Each AND takes one product line whether it has one fuse blown or four.
- Also the output OR lines take the same area whether they have one fuse blown or all twelve.

To reduce size in a PLA, the only alternative is to reduce the number of product lines.

The 7-segment display logic was redesigned below to reduce product lines. The single letter terms, W, X,  $\bar{X}$  ... which were so efficient before, are mostly gone. They are replaced by a smaller number of single square (or single plus a *d* square) terms like S, N, T and V. These circles look inefficient, and they are if you count the number of letters. However the number of product terms has been reduced by 25%. This will give a similar reduction in silicon area.

$P = Y\bar{Z}$   
 $T = \bar{X}YZ$   
 $V = \bar{W}Y\bar{Z}$   
 $S = \bar{W}\bar{Y}\bar{Z}$   
 $H = \bar{W}X\bar{Z}$   
 $M = \bar{X}Y$   
 $Q = \bar{X}\bar{Y}\bar{Z}$   
 $N = X\bar{Y}Z$   
 $W$

$a = T + N + W + Q + M$

$b = Q + S + H + T + M$

$c = Q + H + S + N + T + V$

$d = Q + M + N + V$

$e = Q + P$

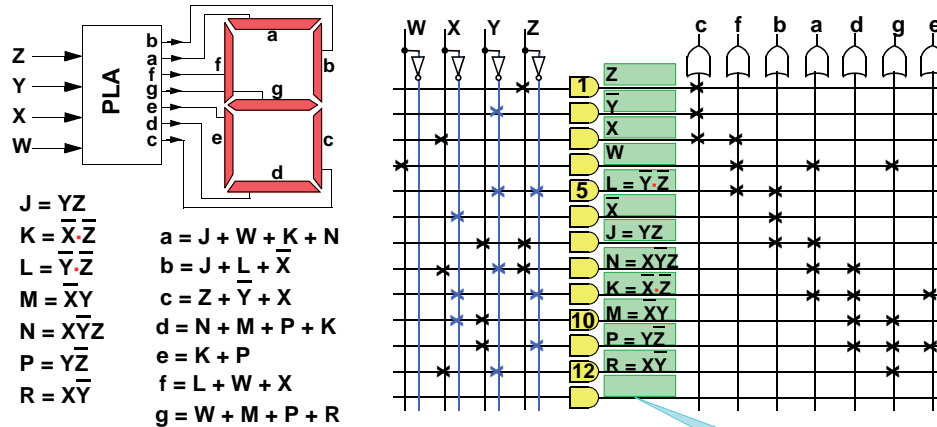
$f = Q + W + S + N + T + V$

$g = W + M + S + N + P$

9 Product terms (was 12)  
55 letters (was 37)



PLA for 7-Segment Display Driver,



This shows how the PLA is programmed.  
 Is a nice orderly array.  
 But notice most of the area is unmade connections.  
 In integrated circuits using logic gates takes less silicon area.

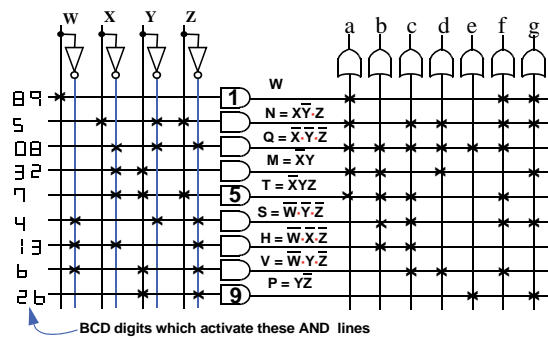
Typically a few product lines wasted  
 PLAs are preproduced in standard sizes

Designers observed  
 An output OR line usually connects to only a few of the OR lines  
 Wastes area  
 They tried making OR plane fixed.

The Smaller PLA

$N = X\overline{Y} \cdot Z$   
 $Q = \overline{X} \cdot \overline{Y} \cdot \overline{Z}$   
 $M = \overline{X} Y$   
 $T = \overline{X} Y Z$   
 $S = \overline{W} \cdot \overline{Y} \cdot \overline{Z}$   
 $H = \overline{W} \cdot \overline{X} \cdot \overline{Z}$   
 $V = \overline{W} \cdot Y \cdot \overline{Z}$   
 $P = YZ$

$a = W + N + Q + M + T$   
 $b = Q + M + T + S + H$   
 $c = N + Q + T + S + H + V$   
 $d = N + Q + M + V$   
 $e = Q + P$   
 $f = W + N + Q + T + S + V$   
 $g = W + N + M + S + P$



Standard sizes

The great advantage of the PLA and other array logic, is that the silicon can be produced in large quantities before the problem is defined. Manufacturers produce several standard sizes. The designer picks the next larger size that will hold the logic he/she needs.

Now logic is so low cost that companies save money by using only a few sizes. 50% utilization will not a cause concern about the cost of unused silicon. Only 10% utilization probably would.



**Programmable Array Logic (PAL).**

**PAL Concept**

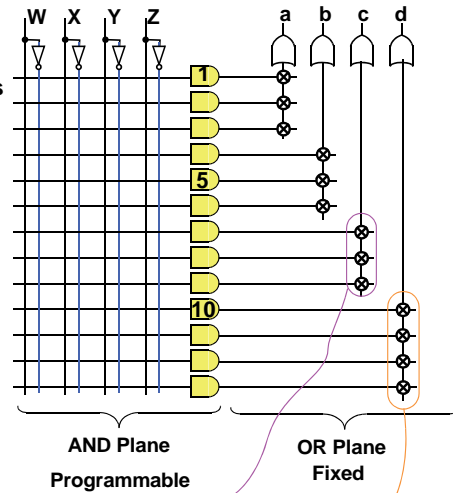
OR outputs use only a few product connections  
 Give them a few fixed connections ⊗  
 This should be enough for most logic.  
 This saves space  
 Perhaps 75% of OR plane area.  
 Makes logic less costly.

Sometimes the logic won't fit  
 Then you use a bigger PAL  
 and waste AND terms.

**Example**

The 7-segment display logic.

Use a PAL with:  
 Three 3-product line OR outputs.  
 Four 4-product line OR outputs.  
 ⊗ is a permanently wired connection.



**PALs,**

**Helper or Expander Logic**

This can often allow squeezing more terms into a PAL

Here the helper line  $H = \overline{W} \cdot \overline{X} \cdot \overline{Y}$

$$\overline{H} = W + X + Y$$

and the  $Z\overline{H}$  term will supply d with

$$Z\overline{H} = Z(W + X + Y)$$

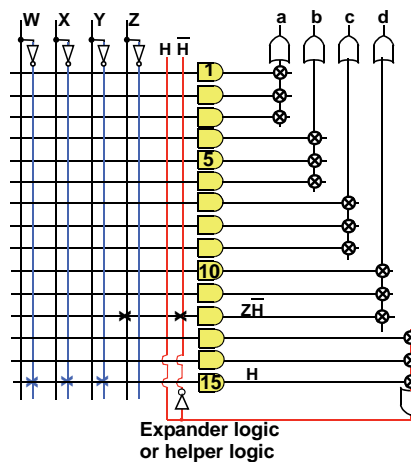
**Next Slide**

**External Helper input**

Using a 5th input,  $e = P + K$  is fed into that input.

Thus  $d = P + K + M + N$  is implemented as

$$d = e + M + N$$

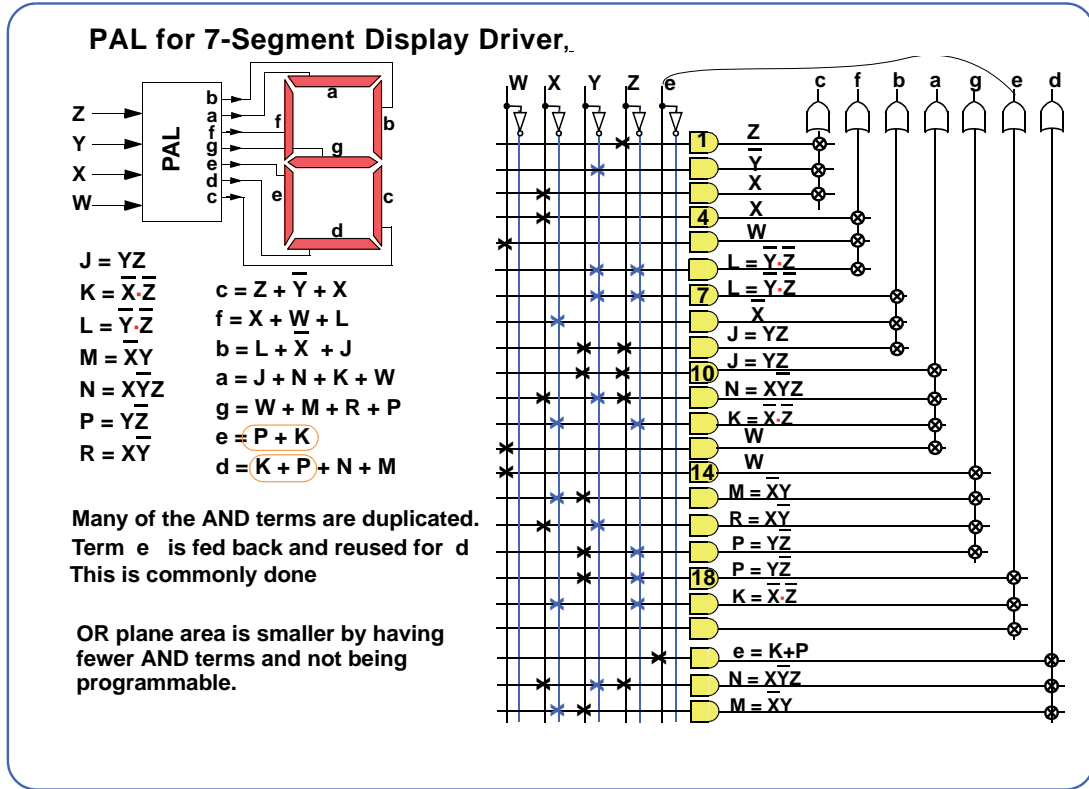


**4. PROBLEM**

Using the maps on Slide 9, define a new expression  $U = W + M + N = W + \overline{X}Y + X\overline{Y}Z$

a) You will need U to help you define a, d and g using only three product terms instead of the four terms shown on the next page. This allows them to fit in the PAL shown on page 24 which has only three terms for each output.

b) Finish programming the PAL on page 24.



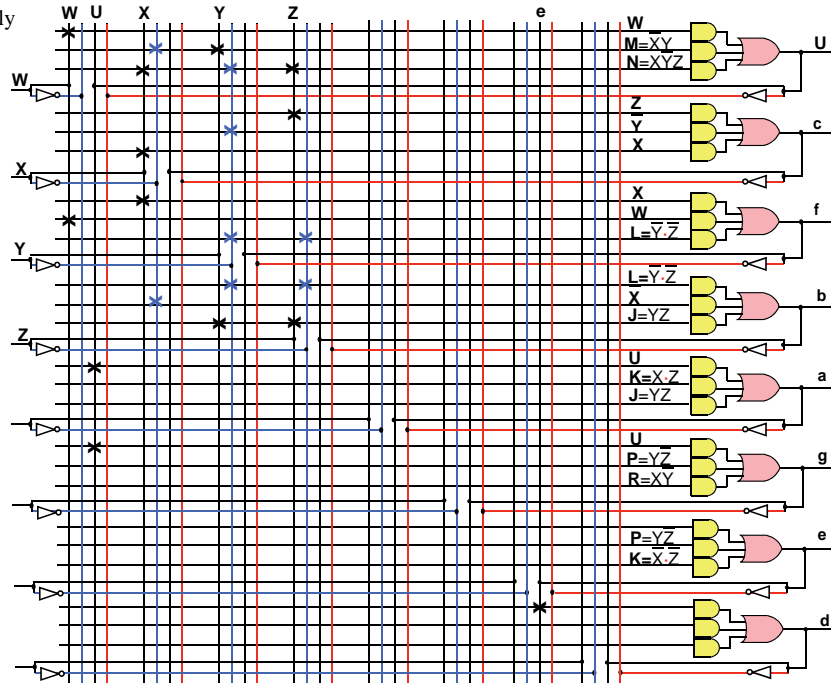
### A PAL Layout More Like the Diagrams in Manufacturers Data Pages

Modern PALs usually have an equal number of product lines for each output. Three are shown here. Most PALs have 8 or more.

All the outputs can also be used as feedback for helper logic.

Most PALs, but not this one, will also have a flip-flop for each output.

The “.” are permanent connections  
The “x” show part of the programmed connections for the 7-segment display.





**Fixed AND Plane Programmable OR Plane,**

**Concept**

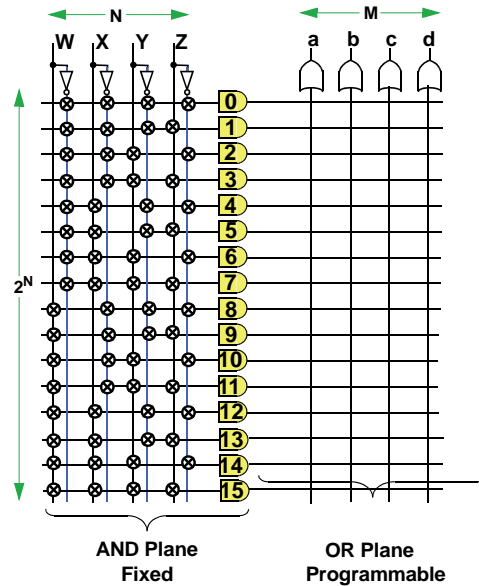
- N inputs, M outputs
- Preprogram all  $2^N$  possible AND terms
- OR plane programmed for desired output

**No restrictions**

- Can do **any** N-input, M output function

**Wasteful of AND terms**

**WHAT DO YOU CALL IT?**



**Fixed AND Plane, Programmable OR Plane**

- There is an AND line for each binary number from 0 to  $2^N-1$ .
- With  $2^N$  AND lines, this can get large
- This is a very common circuit, but not for doing logic.
- It is a ROM (Read Only Memory)
- It is not used in mainstream logic because it is large and slow.



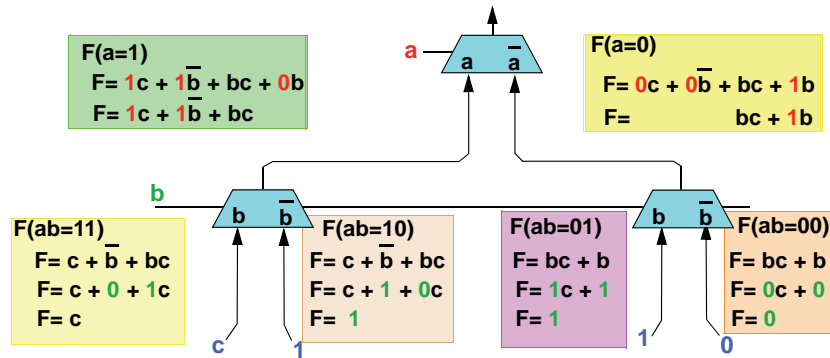
## MUX Logic

### Building Circuits Using MUXs

Break  $F$  into two parts;  
one when  $a=1$ , one when  $a=0$

Use a MUX controlled by "a" to select which expression to use.

$$F = ac + a\bar{b} + bc + \bar{a}b$$



## Mux Logic

### Breaking up Equations

Any algebraic expression with ANDs, OR and NOT can be broken into two parts along a single variable

$$F = cba(d + \bar{e}\bar{b}\bar{d}) + e(a + d(\bar{b} + \bar{0}))$$

Break around c:

$$\begin{aligned} \text{When } c=1; F(c=1) &= 1ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d(\bar{b} + \bar{0})) \\ &= ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d\bar{b}) \end{aligned}$$

$$\begin{aligned} \text{When } c=0; F(c=0) &= 0ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d(\bar{b} + 1)) \\ &= e(a + d) \end{aligned}$$

$$\text{We can write this as } F = c[ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d\bar{b})] + \bar{c}[e(a + d)]$$

This is known as the *Shannon Expansion* around c.

### Mux

The expression for a MUX controlled by c is  $F = cA + \bar{c}B$

If, as above,  $A = ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d\bar{b})$

and  $B = e(a + d)$

then F from the MUX is

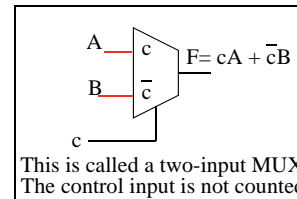
$$F = c[ba(d + \bar{e}\bar{b}\bar{d}) + e(a + d\bar{b})] + \bar{c}[e(a + d)]$$

This shows one can always implement a logic expression from a MUX and two simpler expressions.

### 5. PROBLEM

Read the completion of the above example on the next page.

Then use two input (plus control input) MUXs to implement the parity function  $F = a\bar{b}\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}\bar{b}c$

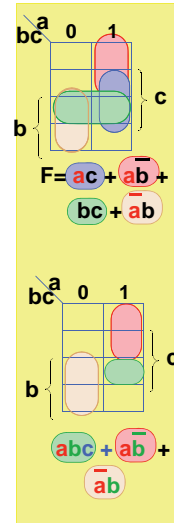
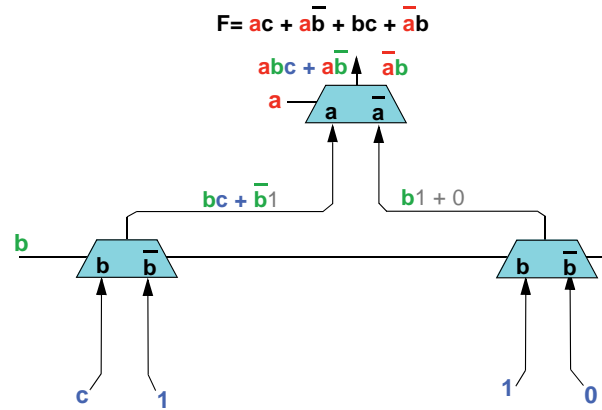




## MUX Logic

### Building Circuits Using MUXs

Work backwards;  
signal flows upwards

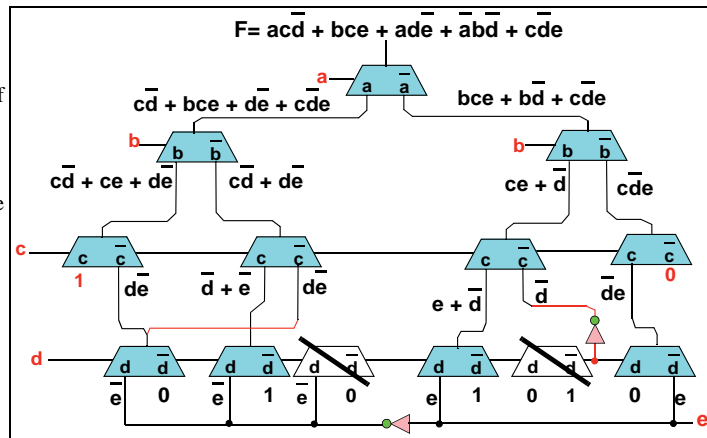


### Mux Logic

#### Simplification

If one looks at a single level of MUX, one may find two or more MUXs with the same inputs. The MUXs in the “d” level having inputs  $\bar{e}$  and 0 are an example. These MUXs do the same thing so one can replace them with one MUXs, as was done on the right.

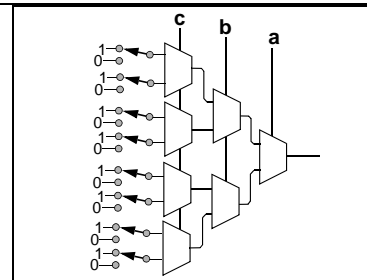
Also one does not need the MUX to calculate  $\bar{D}$ , an inverter is enough.



### MUX Logic Can Be Programmable

Mux logic can be made programmable by making the input connections programmable to either 1 or 0, as shown.

In the figure 8 inputs are programmable and any function of three inputs can be implemented. To make any function of four variables, one would need to have 16 inputs programmable to 1 or 0.

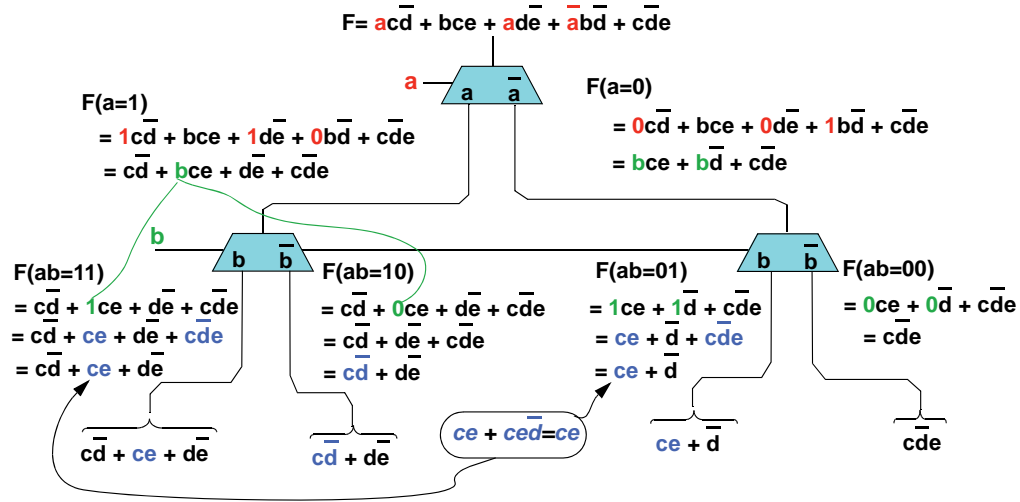




### MUX Logic

#### Breaking up Equations Using MUXs

Break F into two parts;  
 one when a=1, one when a=0  
 Use a MUX controlled by "a" to select which expression to use.



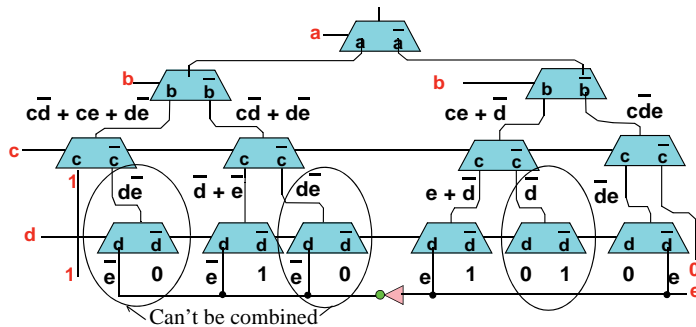
#### Four Input MUXs Logic

Logic can be made with 4-input MUXs as well as 2-input MUXs.

Draw trapezoids around three MUXs, two lower level one they supply. These trapezoids becomes a 4-input MUXs.

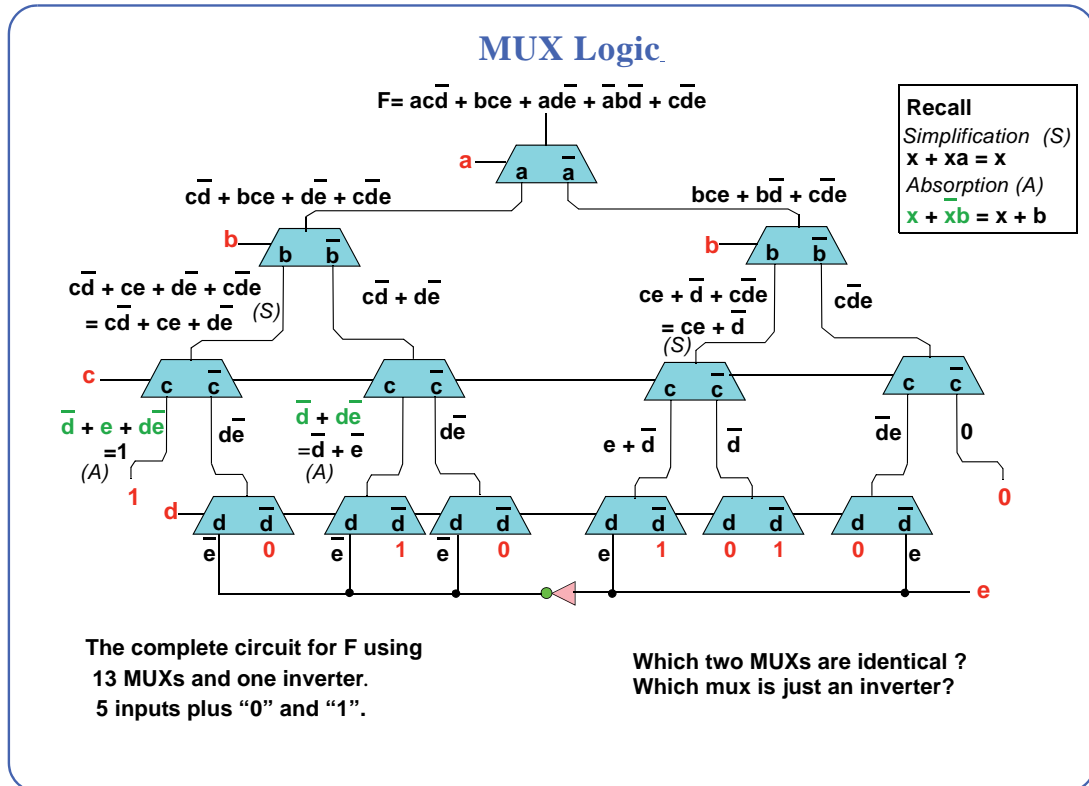
Some of the simplifications possible with 2-input MUX logic are not possible with 4-input MUX logic. They would require connections between the insides of the 4-input MUXs.

Many of the commercial FPGAs are divided into logic blocks containing a 5-input, 2-output logic arrays. Manufacturers have found that it is often most efficient to implement these arrays using 4-input MUX logic.



6. • PROBLEM

Convert the circuit on Slide 17 (with order d-e-c-b-a) to a circuit using 4-input MUXs.



### Reordering MUX Logic

**Reordering**

Ordering of the control inputs for the MUXs can make a large difference in complexity. How to find the optimal ordering is not well understood.

**Sharing MUXs**

On the right side, at the "c" MUX level, two MUX inputs are the same  $\bar{a}b$ . These two can share all lower level MUXs. In this case there is only one at the b level.

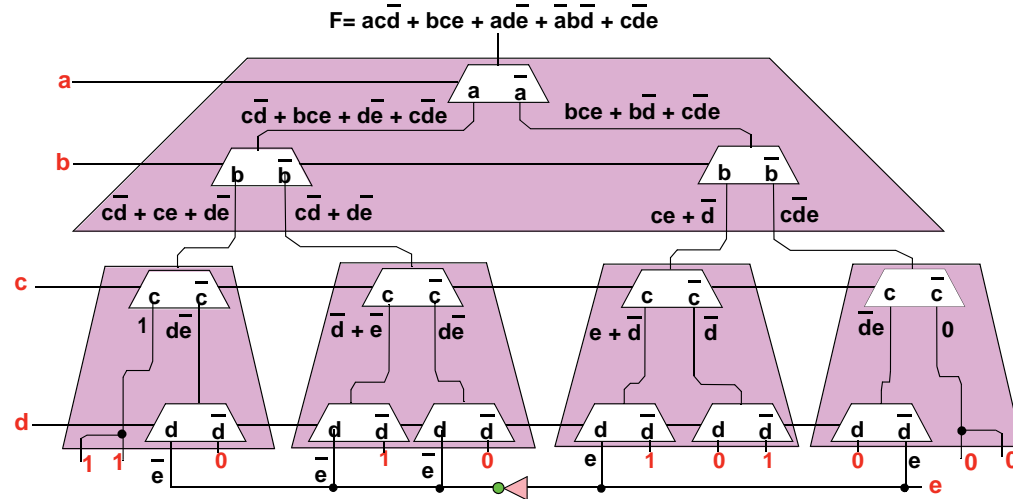
7. • PROBLEM

Find a MUX implementation of  $F = ac\bar{d} + bce + ade + \bar{a}b\bar{d} + c\bar{d}e$

Order the MUXs c-e-b-a-d



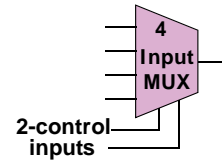
4-Input MUXs



Same circuit as before except trapezoids group MUXs

Combines 2-input MUXS into 4-input MUXs

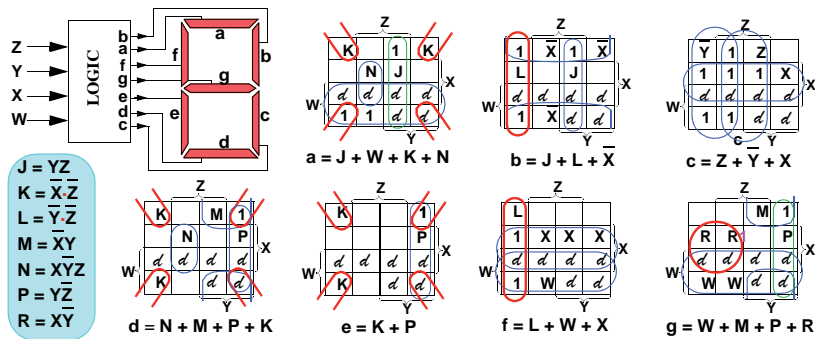
Shows how to do logic with 4-input MUXs. often the most efficient way to make programmable logic.



Appendix

Reduce four-term expressions to three term

Three terms will fit into the three input ORs of many PALs. See "A PAL Layout More Like the Diagrams in Manufacturers Data Pages" on page 24. ,



Feed backs term are

$$U = W + N + M = W + XYZ + \bar{X}Y$$

$$e = K + P$$

$$a = J + W + K + N = W + N + M + K + J = U + K + J$$

$$g = W + N + M + P + R = U + P + R$$

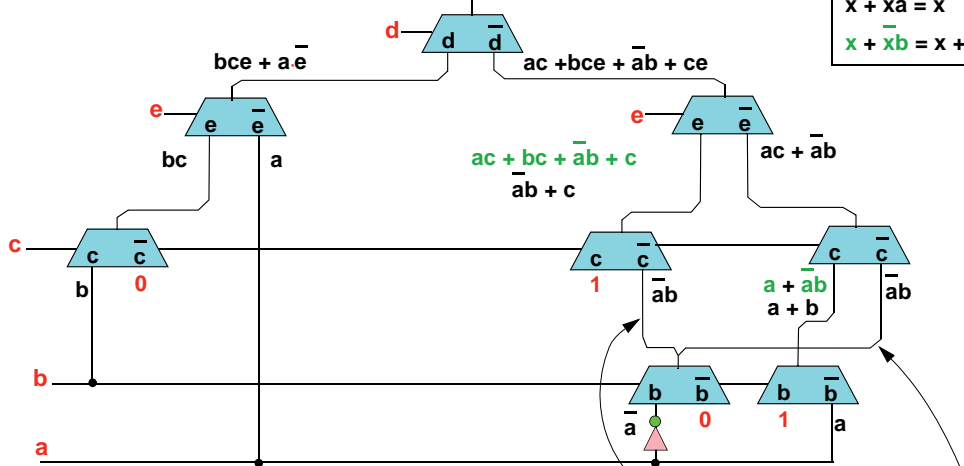
$$d = N + M + e$$



Reordering Can Make Big Changes.

$$F = ac\bar{d} + bce + ade + \bar{a}bd + c\bar{d}e$$

Recall  
 $x + xa = x$   
 $x + \bar{x}b = x + b$



The same function, with a different ordering of MUXs.  
 Order a-b-c-d-e used 12 MUXs  
 Order d-e-c-b-a used only 8 MUXs  
 There is an optimum ordering for most expressions.  
 Hard to find without trying them all.



