Programmable Logic

- There has to be a better way to implement a logic function than to hook together discrete 74XX packages or create a custom Integrated Circuit.
- Memory - can use semiconductor memory to implement logic equations
- Programmable Logic - can use integrated circuits known as “Programmable Logic Devices” to implement logic.

Memory

Typically think of a memory device as used for storing data.

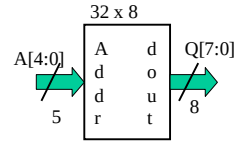
A Memory chip is characterized by how many locations it contains and how many bits per location it can hold.

Memories are classified as K x N devices, K is the # of locations, N is the number bits per location (16 x 2 would be 16 locations, each storing 2 bits).

To access a LOCATION within a memory device, a group of inputs known as the ADDRESS BUS is used. The number of address lines needed is log2(K) (I.e., for 16 locations would need 4 address lines).

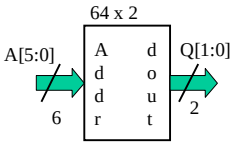
The data at a location is placed on some outputs known as the DATAOUT bus.

Memory Examples



32 x 8 memory (32 locations, 8 bits per location).

Needs 5 address lines, 8 data lines



64 x 2 memory (64 locations, 2 bits per location).

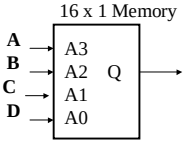
Needs 6 address lines, 2 data lines

LookUp Table (LUT)

Loc	A B C D	F(A,B,C,D)
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	0
6	0 1 1 0	1
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	1
11	1 0 1 1	0
12	1 1 0 0	0
13	1 1 0 1	1
14	1 1 1 0	1
15	1 1 1 1	1

A memory device can be thought of as a LookUp Table (LUT), where each location contains 1 or more bits.

16 x 1 Memory



Using Memory to implement a Boolean Function

Need to use one address line for each boolean variable.

3 variables use 3 address lines, need 8 locations in memory.
4 variables use 4 address lines, need 16 locations in memory
5 variables use 5 address lines, need 32 locations in memory… etc.

For N variables, need 2^N locations in memory.

For each FUNCTION to be implemented, need a bit at each location.

K x 1 Memory can implement 1 boolean function of log2(K) variables (16 x 1 memory can implement F(A,B,C,D).
K x 2 Memory can implement 2 boolean functions of the same log2(K) variables (32 x 2 memory can implement F(A,B,C,D,E) and G(A,B,C,D,E). Etc..

Implement 2 functions of 3 variables

F (A,B,C) = A xor B xor C G = AB + AC + BC

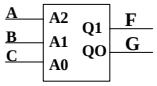
A B C	F	G
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1
1 1 1	1	1

Recall that Exclusive OR (xor) is

A B	Y
0 0	0
0 1	1
1 0	1
1 1	0

Y = A⊕B
= A xor B

8 x 2 Memory



LookUp Table (LUT)

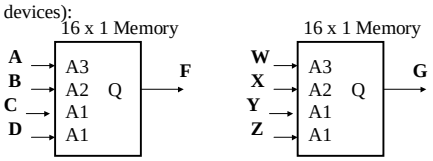
A[2:0] is 3 bit address bus, D[1:0] is 2 bit output bus.
Location 0 has “00”,
Location 1 has “10”,
Location 2 has “10”,
etc….

Memory can be INEFFICIENT

What if I wanted to implement:
F(A,B,C, D) and G (W,X,Y, Z)

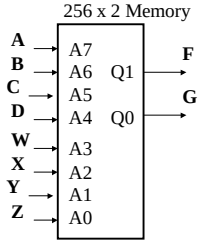
These are two INDEPENDENT functions - they use DIFFERENT inputs!

Note that the variables are different. I could use two different memory devices (need 32 locations total between the two devices):



Memory can be INEFFICIENT (cont).

What if I wanted to use only a single memory device???



Went from 32 locations to 256 locations!!!

Most of the locations are wasted because I have to repeat the G function for every F input.

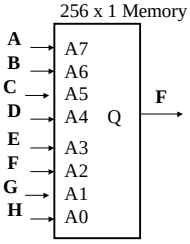
Another Inefficiency

Implement:

F(A,B,C,D,E,F,G,H) = ABCD + EFGH

Looks like a simple equation, would only take two 4 input NANDs, one two input NAND (NAND-NAND form).

Yet takes a 256 x 1 memory device because of the number of input variables!!!



Memory Summary

- To implement N functions of the same K variables, need a memory with 2^K locations and N bits per location (use one address line for each variable, use data out line for each function).
- Memory is not efficient at implementing wide functions (functions with lots of input variables) or multiple functions with different inputs.

Programmable Logic Devices (PLDs)

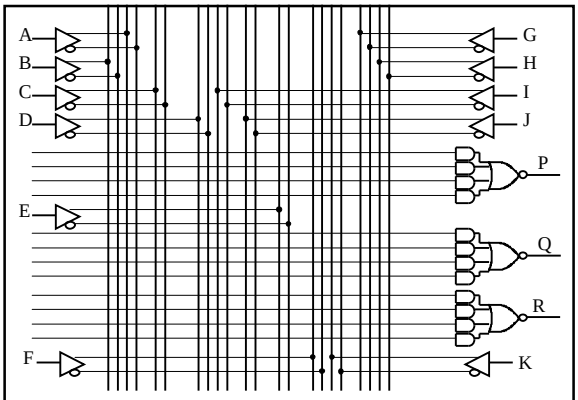
- PLDs were invented to address the inefficiencies of implementing logic using memories.
- PLDs can implement wide functions efficiently (functions with many input variables).
- PLDs can implement multiple functions of different variables efficiently.
- The logic in PLDs is programmable -- it can be defined by the user and programmed on the desktop
  - Most PLDs can be erased and reprogrammed many times.

PLD types

- There are MANY different types of PLDs.
- Densities ranges from from 10's of gates to 100's of thousands of gates.
- We will look at a type called PALs (Programmable Array Logic).
- The Digital System class (EE 4743) uses a programmable logic type called a Field Programmable Gate Array (FPGA).

PALs (Programmable Array Logic)

- An early type of programmable logic - still in common use today.
- Logic is represented in SOP form (Sum of Products)
- The number of PRODUCTs in an SOP form will be limited to a fixed number (usually 4-10 Product terms).
- The number of VARIABLEs in each product term limited by number of input pins on PLD (usually a LOT, minimum of 10 inputs
- The number of independent functions limited by number of OUTPUT pins.

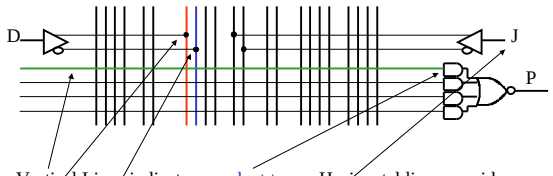


A Sample PAL

Comments on Sample PAL

- 11 inputs, 3 outputs
  - Can implement three functions - functions can share inputs or not share inputs
- Each output implements a SOP equation with Four product terms.
  - Each product term can include complemented or uncomplemented form of an input.

Understanding the Diagram

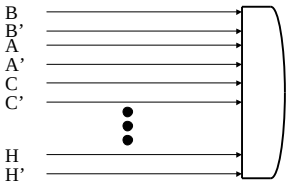


Vertical Lines indicate a product term. Horizontal lines provide True and Complemented forms of external inputs.

Even though a product term looks like it has only one input, it actually has 2 * N inputs, where N is the number of external inputs.

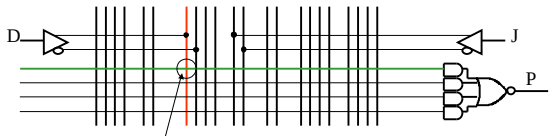
Product Term

This looks like an AND gate with one input. Is actually:



Only drawn with a single line to save space.

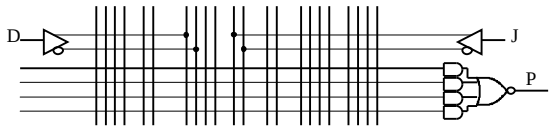
Fuse Points



A cross over of a Vertical input line and a horizontal product term line is a FUSE LOCATION. When the PAL is in its blank or erased state, all FUSES are connected. This means that each product term implements the equation:

(A A' B B' C C'……. KK') will be '0'! This means that the output will be high!

Programming

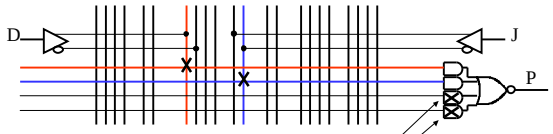


To program, will want to BLOW most of the fuses (break the vertical/horizontal crossover connection). To indicate a logic function, will use a ' X ' over a fuse that I want to KEEP INTACT.

X ← Will mark Intact fuse location.

When a fuse is blown, that product term input acts as a '1' so that the input no longer effects the product term.

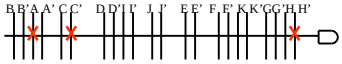
P' = D + J'



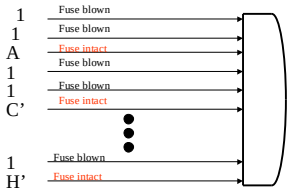
When implementing an equation, sometimes will not want to use all available product terms. If ALL fuses along product term are left intact, then product term value will be '0' and will not affect equation. Mark unused PT's by placing an X over them -- all fuses in that PT row are assumed intact.

Note that P' must be implemented!

Example Product Term AC'H'

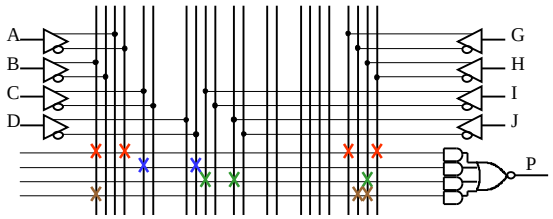


The connections will be:



Actually, fuses are not 'blown' in eraseable PLDs - the connection is broken in a non-destructive way for eraseable PLDs.

Another Example



P' = A'BGH' + CD' + HIJ + BG'H

An Optimization Question

The following two equations are the same:

P' = A'B'C'D' + A'BC'D' + A'B'CD'

P' = A'C'D' + A'B'D' (minimized form)

Does it make a difference which form we implement in the previous PLD?

NO!!!!!!!!!!

Each SOP equation in the PLD has four Product terms allocated. An unused PT cannot be used elsewhere - so makes no difference in terms of the amount of resources used in the PLD!!!

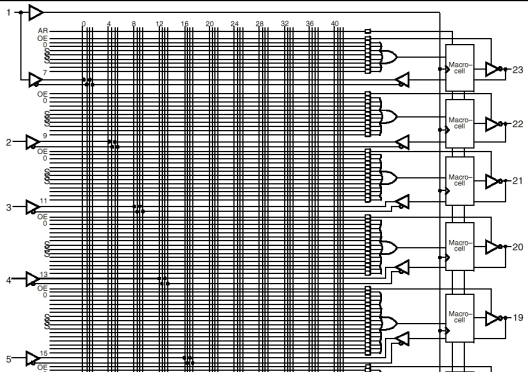
Optimization is Dependent upon Implementation Technology!!!

How we minimize Boolean equations is affected by the IMPLEMENTATION TECHNOLOGY!!!!

When wiring up individual 74xx packages, want as few as gates as possible, and as few connections as possible.

For PALs, want get the SOP equation into the number of allowed Products terms - getting fewer than this will not help us unless it can eliminate an input variable.

For other PLD technologies, saving product terms may definitely help. It all DEPENDS on the ARCHITECTURE of the programmable logic device!!!!



Part of a 22V10 PLD

Comments on 22V10 PLD

- The 22V10 PLD is a example of a more complex PAL. We will use this device in Lab.
- The 'macrocell' has a memory device called a Flip-Flop inside of it - we will discuss this later in the course.
  - In our initial use of the 22V10 PLD, the macrocell will just pass the OR gate output unchanged to the output pin.
- The output of the Macrocell is passed back into the array as an input - this way complex functions be created by 'chaining' functions together.

Other PLD types

- Other types of programmable logic have much different internal structures from PALs.
- You must understand the internal structure of whatever PLD you are using so that you can understand the LIMITATIONS and ADVANTAGES of the particular PLD that you are using.
  - These advantages/limitations can affect how you implement a particular logic function

What do you need to know?

- How to implement boolean equations in memory devices.
- Structure of PAL programmable logic
- How show the implementation of a boolean equation on a PAL architecture diagram
- Optimization goals for boolean equations targeted at a PAL architecture