PLAs

- A 2-level *and-or* structure is replicated many times in a programmable array called a *PLA* (*programmable logic array*)

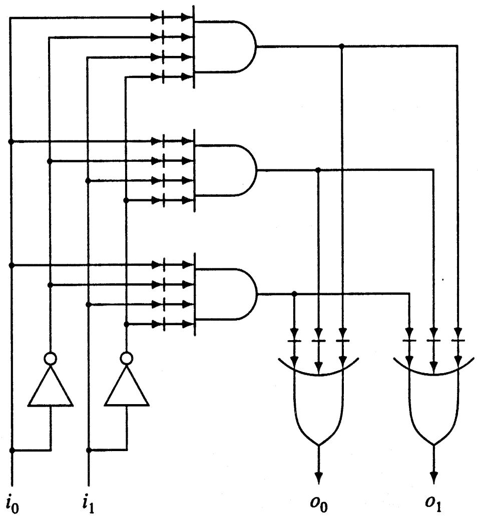


Diagram from *Computer Systems*, Maccabe, Irwin 1993

- This PLA has 2 inputs, 2 outputs, and can represent up to 3 product terms

Using PLAs

- Anytime you have a large combinational circuit, one possibility for implementing it is to use PLAs
  - Parts of a CPU's datapath or next-state logic can be built out of PLAs
  - Small circuits can be built out of PLAs
- At the input of each gate, there's a "fuse" which can be left whole, or broken
  - So the designer can control which inputs go to each and gate, and which outputs of the and gates go to each or gate
- A PLA can be either
  - mask programmable — customer orders a programmed PLA from the manufacturer
  - field programmable — customer can program PLA (once)

PLA Example

- This is an *abstract* diagram of a PLA with 6 inputs, 4 outputs, which can represent up to 12 product terms

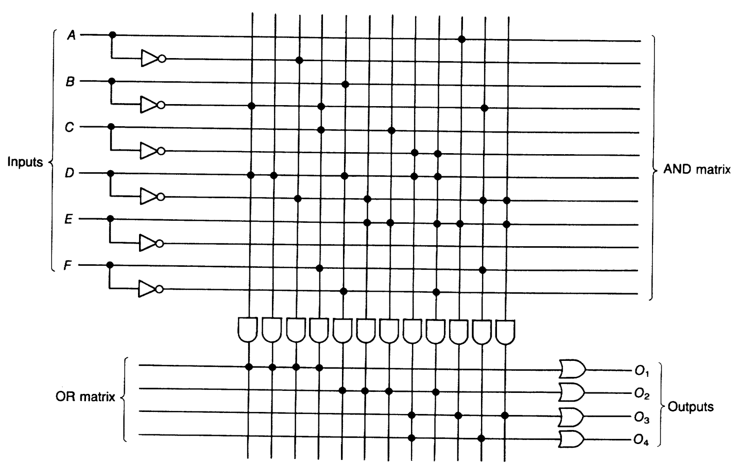


Diagram from *Digital Design*, Johnson & Karim, PWS-Kent 1987

- Try the Java KMap->PLA animation at http://tech-www.informatik.uni-hamburg.de/applets/kvd

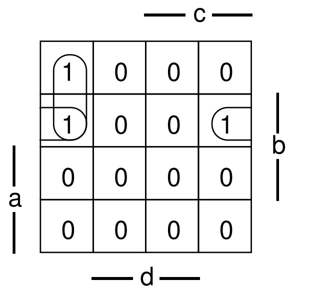
Field-Programmable Logic Device

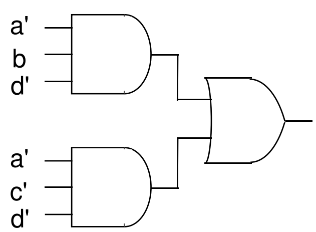
- The next evolutionary step beyond the PLA is the *field-programmable logic device* (FPLD), also called the:
  - Field-programmable gate array (FPGA)
  - Complex programmable logic device (CPLD)
- Some FPLDs are based on an array of PLA-like *and-or* structures
  - May include D (or more complex) flip-flops, to easily build sequential circuits
  - May include RAM
  - Plus connections to 4 nearest neighbors
    - "Longer" connections across chip
  - Many can be "field-programmed" repeatedly
  - Available in different sizes up to 500,000 gates (100MhZ, 2.5 volt, 0.25μ, 5 metal)

Implementing a Truth Table Using An And-Or Structure (Review)

- Given a truth table, we can use a Karnaugh map to find the minimum 2-level SOP implementation

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



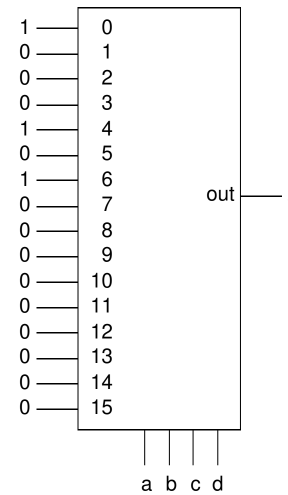


z = a'bd' + a'c'd'

Implementing a Truth Table Using a Multiplexor

- Another alternative is to use a 4-input multiplexor

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

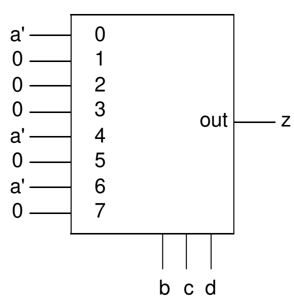


- Any function of N inputs can be implemented using a 2^N to 1 multiplexor

Implementing a Truth Table Using a Multiplexor (cont.)

- An alternative is to "fold" the truth table, and tie each input to either 1, 0, or the MSB, and only use a 3-input multiplexor

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

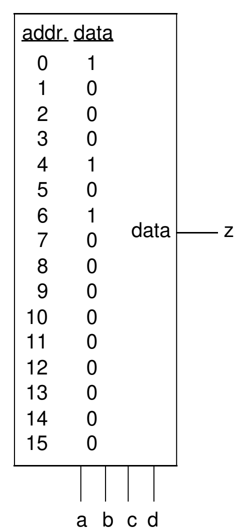


- Any function of N inputs can be implemented using a 2^{N-1} to 1 multiplexor
  - Some FPLDs are based on multiplexors, and attach simple gates to selector lines

Implementing a Truth Table Using a ROM

- Yet another alternative is to use a ROM

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



- Any function of N inputs can be implemented using a 2^{N-1} x 1 bit ROM
  - Some FPLDs are based on static RAMs (SRAMS) loaded at power-up; these are said to use *look-up tables* (LUTs)

Different Implementation Styles