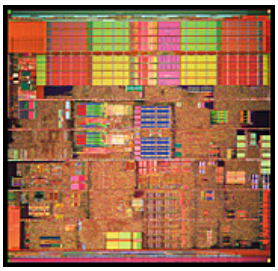
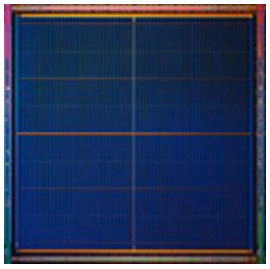
Programmable Logic

- Regular logic
  - Programmable Logic Arrays
  - Multiplexers/Decoders
  - ROMs
- Field Programmable Gate Arrays
  - Xilinx Vertex



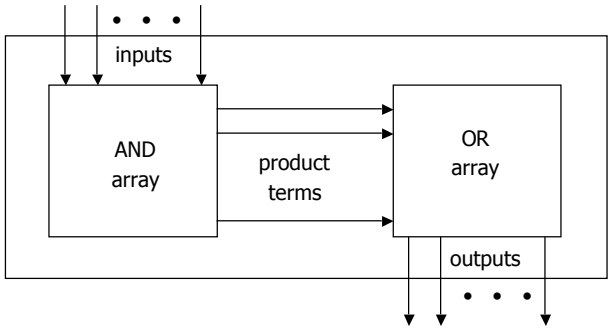
"Random Logic"
Full Custom Design



"Regular Logic"
Structured Design

Programmable Logic Arrays (PLAs)

- Pre-fabricated building block of many AND/OR gates
  - Actually NOR or NAND
  - "Personalized" by making or breaking connections among gates
  - Programmable array block diagram for sum of products form



Enabling Concept

- Shared product terms among outputs

example:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

personality matrix

product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

input side:

1 = uncomplemented in term
0 = complemented in term
– = does not participate

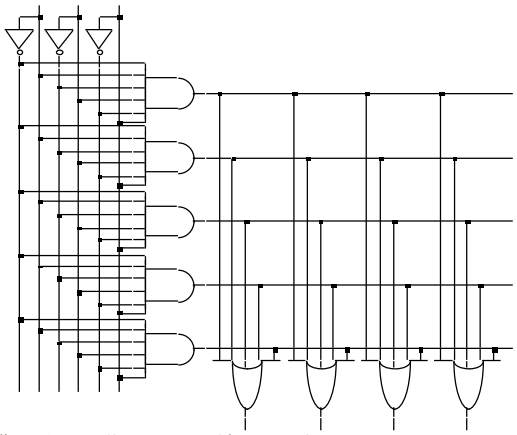
output side:

1 = term connected to output
0 = no connection to output

reuse of terms

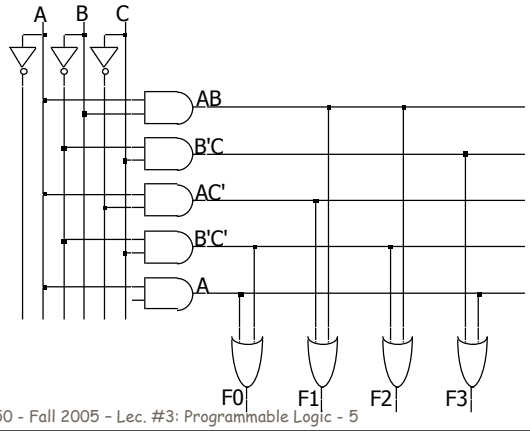
Before Programming

- All possible connections available before "programming"
  - In reality, all AND and OR gates are NANDs



After Programming

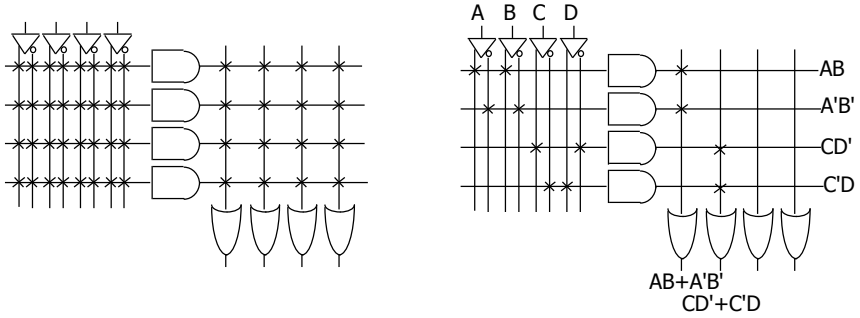
- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)



Alternate Representation for High Fan-in Structures

- Short-hand notation--don't have to draw all the wires
  - Signifies a connection is present and perpendicular signal is an input to gate

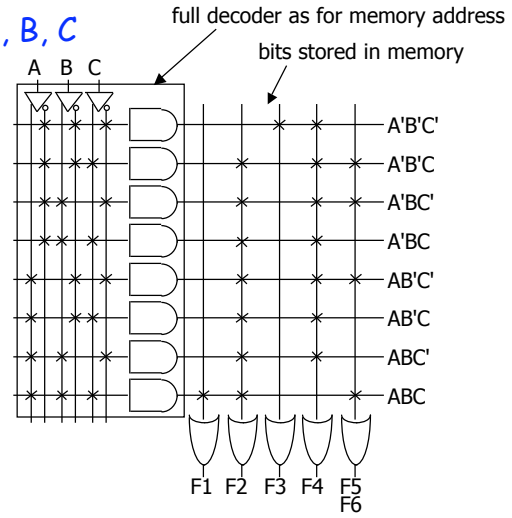
notation for implementing
F0 = A B + A' B'
F1 = C D' + C' D



Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A' B' C'
  - F4 = A' + B' + C'
  - F5 = A xor B xor C
  - F6 = (A xnor B xnor C)'

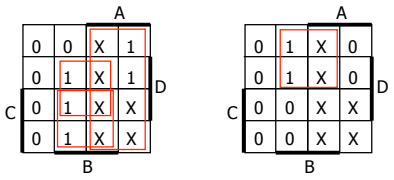
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



PLA Design Example

- BCD to Gray code converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

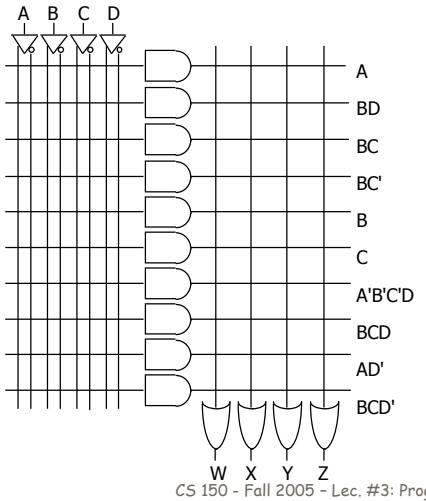


minimized functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A'B'C'D + B C D + A D' + B' C D'

PLA Design Example (cont'd)

- Code converter: programmed PLA



minimized functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A'B'C'D + B C D + A D' + B' C D'

not a particularly good candidate for PLA implementation since no terms are shared among outputs

however, much more compact and regular implementation when compared with discrete AND and OR gates

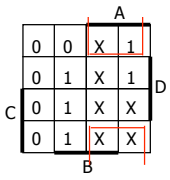
PLA Design Example

- BCD to Gray code converter

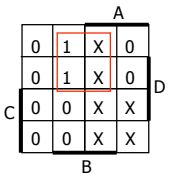
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

minimized functions:

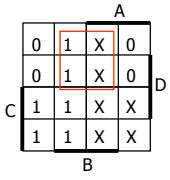
W =
X =
Y =
Z =



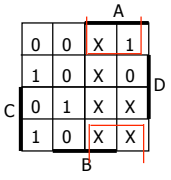
K-map for W



K-map for X



K-map for Y



K-map for Z

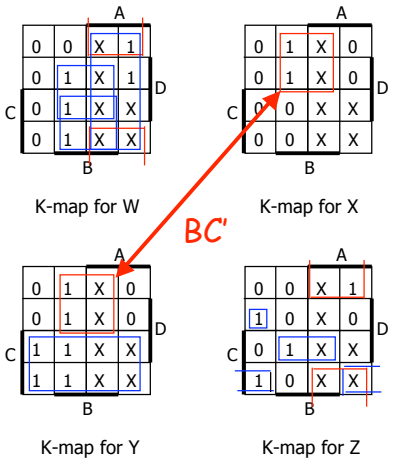
PLA Design Example #1

- BCD to Gray code converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

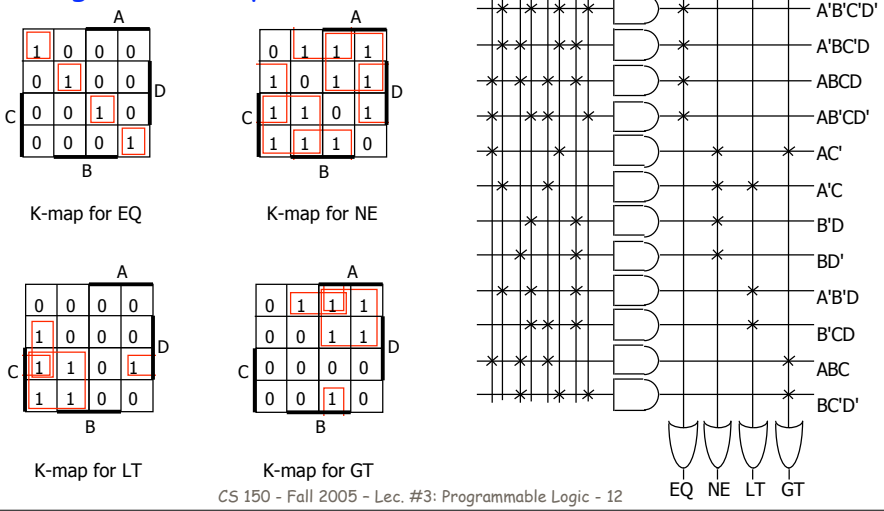
minimized functions:

W =
X =
Y =
Z =



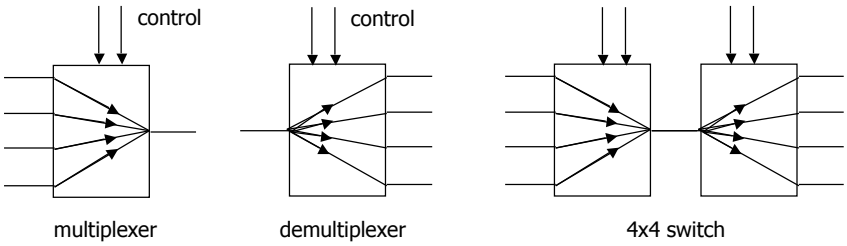
PLA Design Example #2

- Magnitude comparator



Multiplexer/Demultiplexer: Making Connections

- Direct point-to-point connections between gates
- *Multiplexer:* route one of many inputs to a single output
- *Demultiplexer:* route single input to one of many outputs



Multiplexers/Selectors

- Multiplexers/Selectors: general concept
  - 2^n data inputs, n control inputs (called "selects"), 1 output
  - Used to connect 2^n points to a single point
  - Control signal pattern forms binary index of input connected to output

$Z = A' I_0 + A I_1$

A	Z
0	I_0
1	I_1

I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

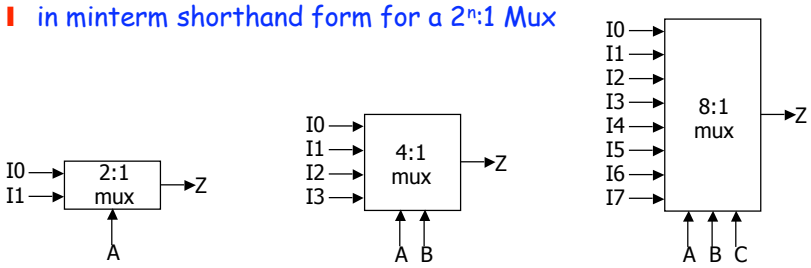
functional form

logical form

two alternative forms for a 2:1 Mux truth table

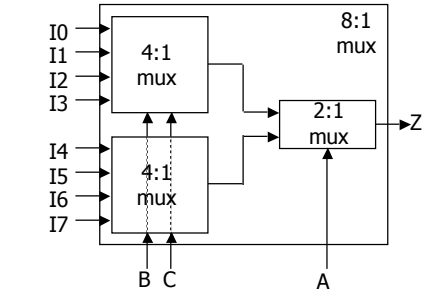
Multiplexers/Selectors (cont'd)

- 2:1 mux: $Z = A' I0 + A I1$
- 4:1 mux: $Z = A' B' I0 + A' B I1 + A B' I2 + A B I3$
- 8:1 mux: $Z = A'B'C'I0 + A'B'CI1 + A'BC'I2 + A'BCI3 + AB'C'I4 + AB'CI5 + ABC'I6 + ABCI7$
- In general, $Z = \sum_{k=0}^{2^n-1} (m_k I_k)$
  - in minterm shorthand form for a 2^n:1 Mux



Cascading Multiplexers

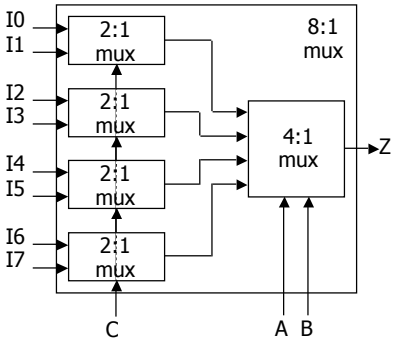
- Large multiplexers implemented by cascading smaller ones



control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

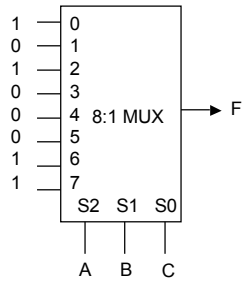
control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation



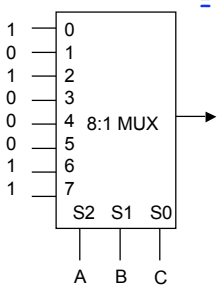
Multiplexers as Lookup Tables (LUTs)

- 2^n:1 multiplexer implements any function of n variables
  - With the variables used as control inputs and
  - Data inputs tied to 0 or 1
  - In essence, *a lookup table*
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



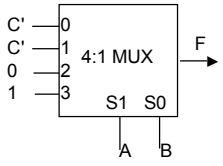
Multiplexers as LUTs (cont'd)

- 2^{n-1}:1 mux can implement any function of n variables
  - With n-1 variables used as control inputs and
  - Data inputs tied to the last variable or its complement
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



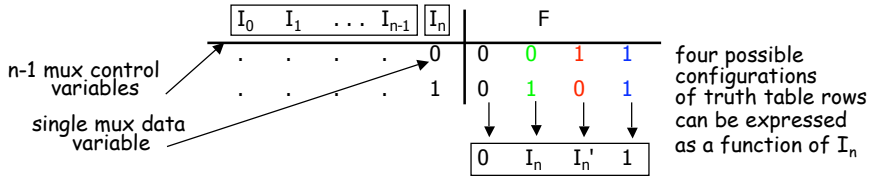
F

A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	

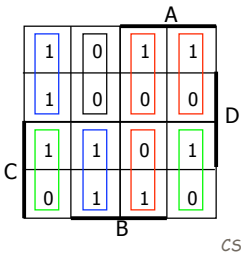


Multiplexers as LUTs (cont'd)

- Generalization

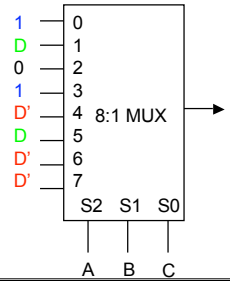


- Example: F(A,B,C,D) implemented by an 8:1 MUX



choose A,B,C as control variables

multiplexer implementation



Announcements

- We took everyone on the wait list into the class
  - Result is that Tu labs are very crowded!
  - Th night lab is very light -- think of switching to get more TA face time!
  - Send email to pokai@berkeley.edu to request a lab change
- First HW due Friday at 2 PM … just before Lab Lecture
  - CS 150 hand-in box outside and just to the right of 125 Cory doors
- Second HW on class web site
- Use ucb.class.cs150 newsgroup for lab, hw, course questions!

Demultiplexers/Decoders

- Decoders/demultiplexers: general concept
  - Single data input, n control inputs, 2^n outputs
  - Control inputs (called "selects" (S)) represent binary index of output to which the input is connected
  - Data input usually called "enable" (G)

1:2 Decoder:

O0 = G • S'
O1 = G • S

2:4 Decoder:

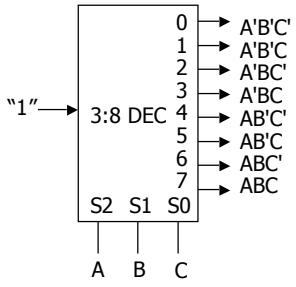
O0 = G • S1' • S0'
O1 = G • S1' • S0
O2 = G • S1 • S0'
O3 = G • S1 • S0

3:8 Decoder:

O0 = G • S2' • S1' • S0'
O1 = G • S2' • S1' • S0
O2 = G • S2' • S1 • S0'
O3 = G • S2' • S1 • S0
O4 = G • S2 • S1' • S0'
O5 = G • S2 • S1' • S0
O6 = G • S2 • S1 • S0'
O7 = G • S2 • S1 • S0

Demultiplexers as General-Purpose Logic

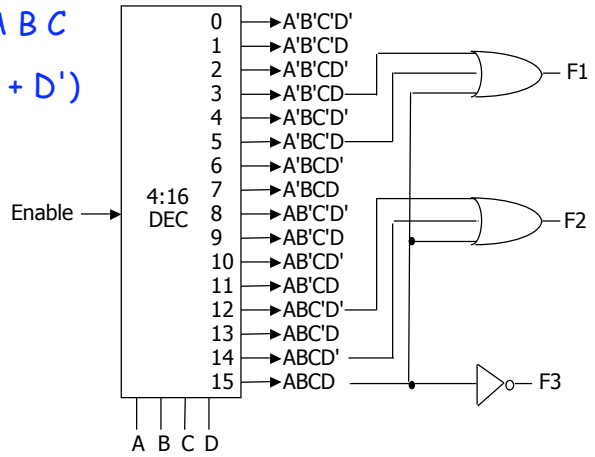
- n:2^n decoder implements any function of n variables
  - With the variables used as control inputs
  - Enable inputs tied to 1 and
  - Appropriate minterms summed to form the function



demultiplexer generates appropriate minterm based on control signals (it "decodes" control signals)

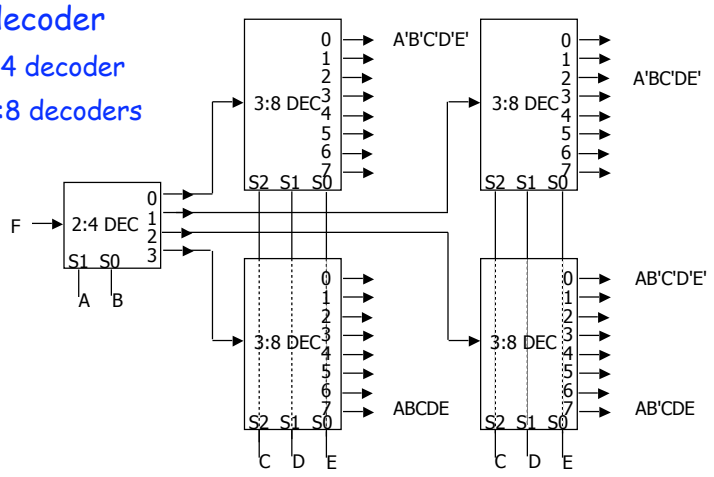
Demultiplexers as General-Purpose Logic (cont'd)

- F1 = A' B C' D + A' B' C D + A B C D
- F2 = A B C' D' + A B C
- F3 = (A' + B' + C' + D')



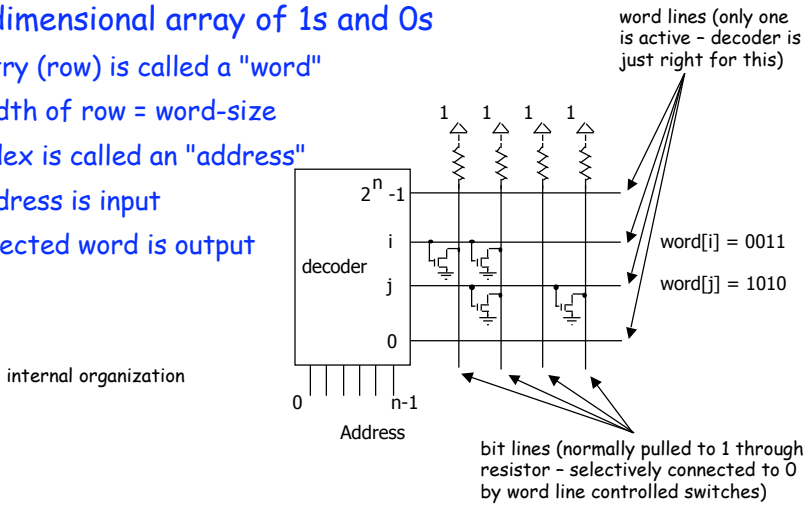
Cascading Decoders

- 5:32 decoder
  - 1x2:4 decoder
  - 4x3:8 decoders



Read-only Memories

- Two dimensional array of 1s and 0s
  - Entry (row) is called a "word"
  - Width of row = word-size
  - Index is called an "address"
  - Address is input
  - Selected word is output



ROMs and Combinational Logic

- Combinational logic implementation (two-level canonical form) using a ROM

F0 = A' B' C + A B' C' + A B' C

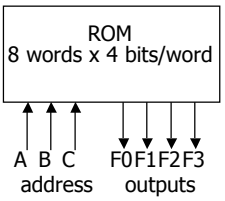
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

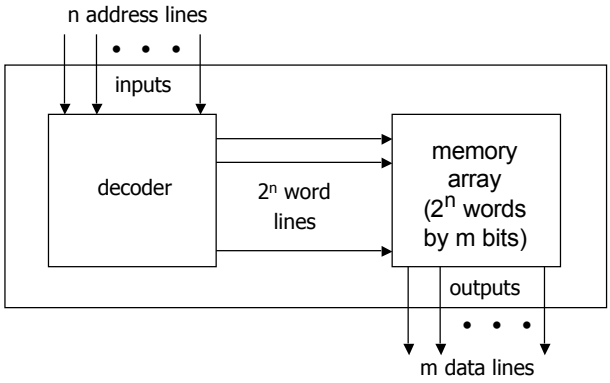
truth table



block diagram

ROM Structure

- Similar to a PLA structure but with a fully decoded AND array
  - Completely flexible OR array (unlike PAL)

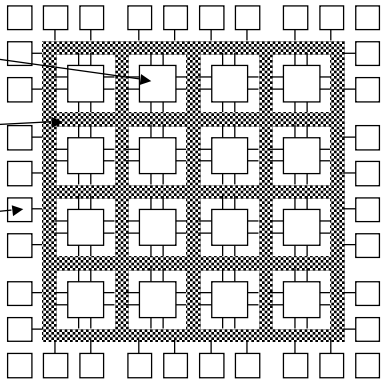


ROM vs. PLA

- ROM
  - Design time is short (no need to minimize output functions)
  - Most input combinations are needed (e.g., code converters)
  - Little sharing of product terms among output functions
  - Size doubles for each additional input
  - Can't exploit don't cares
  - Cheap (high-volume component)
  - Can implement any function of n inputs
  - Medium speed
- PLA
  - Design tools are available for multi-output minimization
  - There are relatively few unique minterm combinations
  - Many minterms are shared among the output functions
  - Most complex in design, need more sophisticated tools
  - Can implement any function up to a product term limit
  - Slow (two programmable planes)

Field-Programmable Gate Arrays

- PLAs: 100s of gate equivalents
- FPGAs: 1000-10000s gates
- Logic blocks
  - Implement combinational and sequential logic
- Interconnect
  - Wires to connect inputs and outputs to logic blocks
- I/O blocks
  - Special logic blocks at periphery of device for external connections
- Key questions:
  - How to make logic blocks programmable?
  - How to connect the wires?
  - *After the chip has been fabbed*

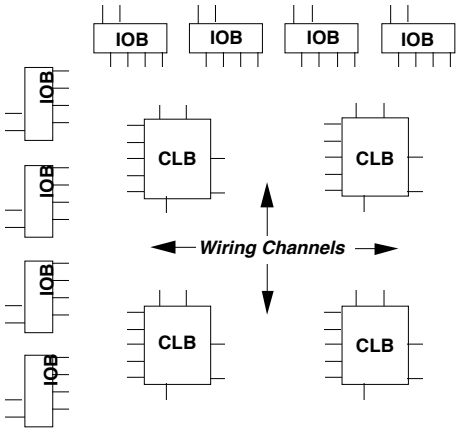


Tradeoffs in FPGAs

- Logic block - how are functions implemented: fixed functions (manipulate inputs) or programmable?
  - Support complex functions, need fewer blocks, but they are bigger so less of them on chip
  - Support simple functions, need more blocks, but they are smaller so more of them on chip
- Interconnect
  - How are logic blocks arranged?
  - How many wires will be needed between them?
  - Are wires evenly distributed across chip?
  - Programmability slows wires down – are some wires specialized to long distances?
  - How many inputs/outputs must be routed to/from each logic block?
  - What utilization are we willing to accept? 50%? 20%? 90%?

Xilinx 4000 Series Programmable Gate Arrays

- CLB - Configurable Logic Block
  - 5-input, 1 output function
  - or 2 4-input, 1 output functions
  - optional register on outputs
- Built-in fast carry logic
- Can be used as memory
- Three types of routing
  - direct
  - general-purpose
  - long lines of various lengths
- RAM-programmable
  - can be reconfigured



The Xilinx 4000 CLB

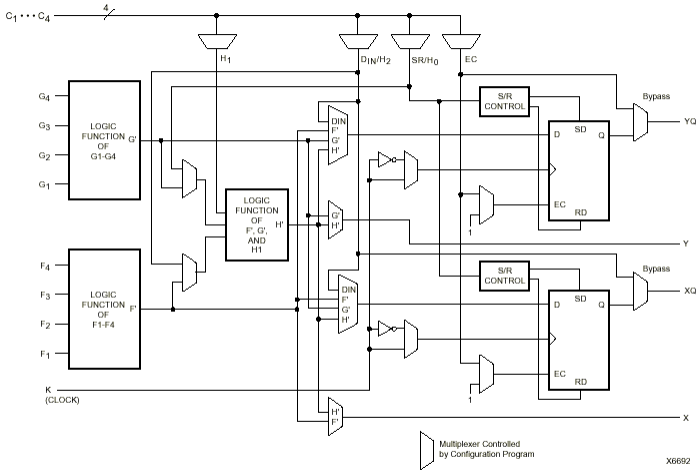


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

Two 4-Input Functions, Registered Output

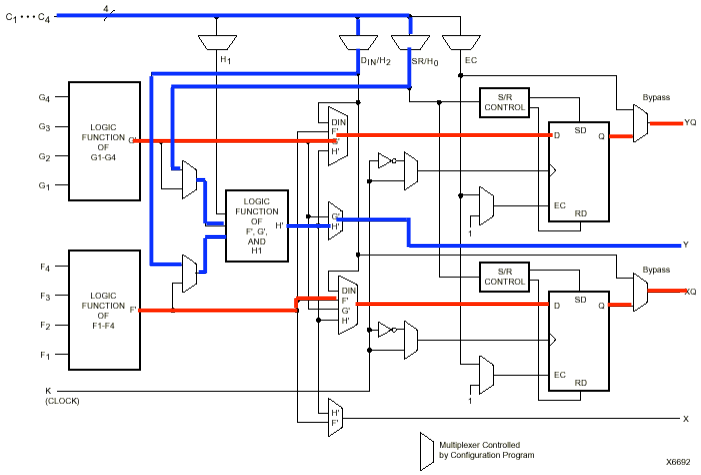


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

5-Input Function, Combinational Output

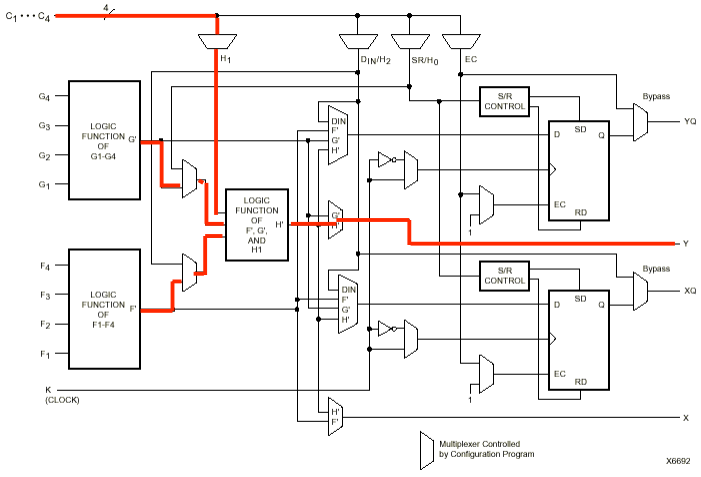


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

CLB Used as RAM

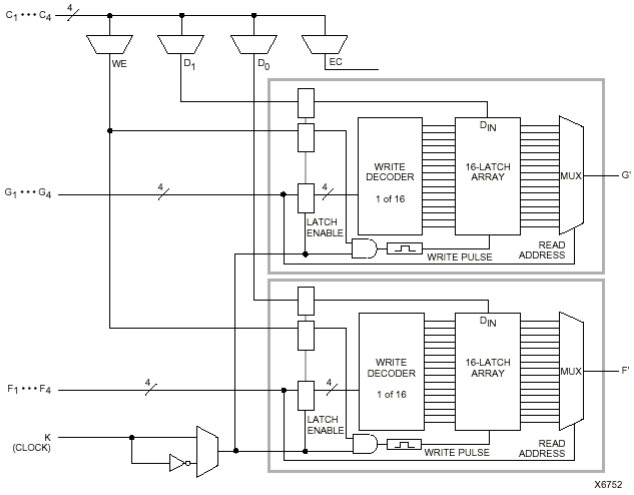


Figure 4: 16x2 (or 16x1) Edge-Triggered Single-Port RAM

Xilinx 4000 Interconnect

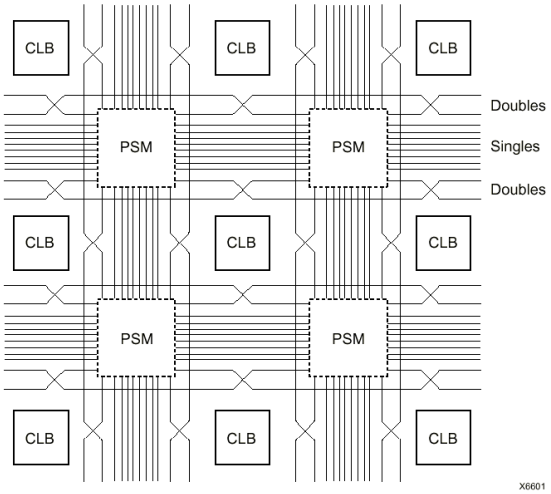


Figure 28: Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)

Xilinx FPGA Combinational Logic Examples

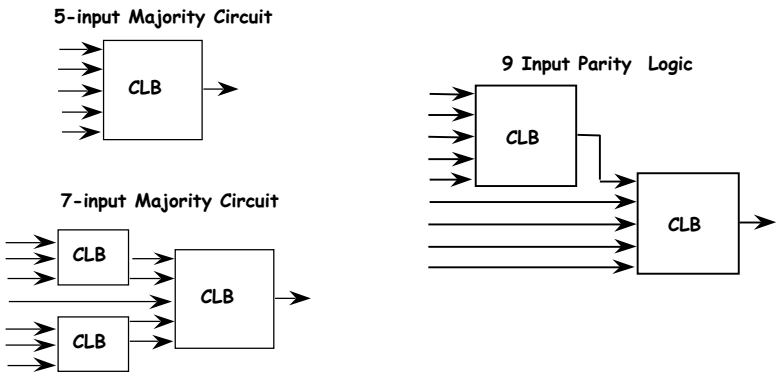
- Key: General functions are limited to 5 inputs
  - (4 even better - 1/2 CLB)
  - *No limitation on function complexity*
- Example
  - 2-bit comparator:
    A B = C D and A B > C D implemented with 1 CLB

```
(GT)  F = A C' +  A B D' +  B C' D'
(EQ)  G = A'B'C'D'+ A'B C'D + A B'C D'+ A B C D
```

- Can implement some functions of > 5 input

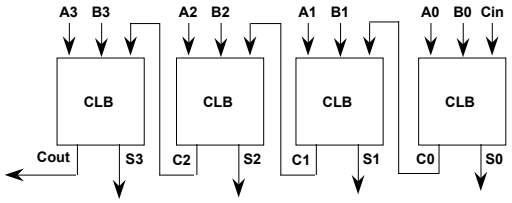
Xilinx FPGA Combinational Logic

- Examples
  - N-input majority function: 1 whenever n/2 or more inputs are 1
  - N-input parity functions: 5 input/1 CLB; 2 levels yield 25 inputs!

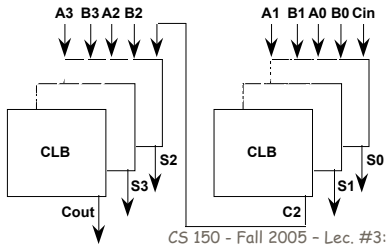


Xilinx FPGA Adder Example

- Example
  - 2-bit binary adder - inputs: A1, A0, B1, B0, CIN
    outputs: S0, S1, Cout



Full Adder, 4 CLB delays to final carry out



2 x Two-bit Adders (3 CLBs each) yields 2 CLBs to final carry out

Combinational Logic Implementation Summary

- Regular Logic Structures
  - Programmable Logic Arrays
    - Programmable connections: AND-OR (NOR-NOR) Arrays
  - Multiplexers/decoders
    - Multipoint connections for signal routing
    - Lookup Tables
  - ROMs
    - Truth table in hardware
  - Field Programmable Gate Arrays (FPGAs)
    - Programmable logic (LUTs, Truth Tables) and connections
  - Advantages/disadvantages of each