

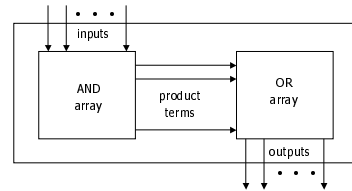
Programmable Logic

- Regular logic
 - Programmable Logic Arrays
 - Multiplexers/Decoders
 - ROMs
- Field Programmable Gate Arrays (FPGAs)
 - Xilinx

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 1

Programmable Logic Arrays

- 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



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 2

Enabling Concept

- Shared product terms among outputs

example:
 $F_0 = A + B' C'$
 $F_1 = A C' + A B$
 $F_2 = B' C' + A B$
 $F_3 = B' C' + A$

product term	inputs			outputs			
	A	B	C	F ₀	F ₁	F ₂	F ₃
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

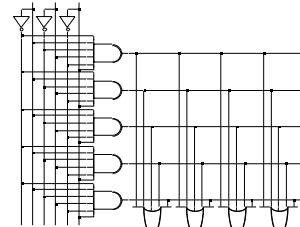
output side:
 1 = term connected to output
 0 = no connection to output

reuse of terms

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 3

Before Programming

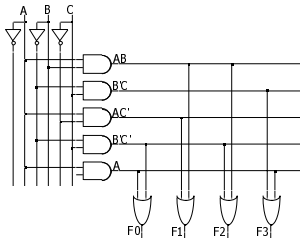
- All possible connections available before "programming"
 - In reality, all AND and OR gates are NANDs



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 4

After Programming

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

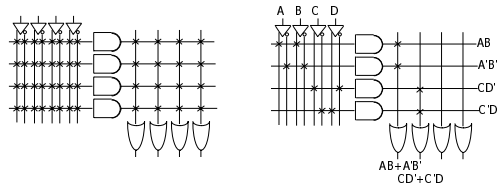


CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 5

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
 $F_0 = A B + A' B'$
 $F_1 = C D' + C' D$



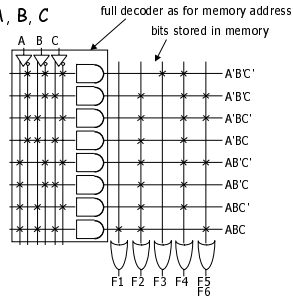
CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 6

Programmable Logic Array Example

Multiple functions of A, B, C

- $F1 = A B C$
- $F2 = A + B + C$
- $F3 = A' B' C'$
- $F4 = A' + B' + C'$
- $F5 = A \text{ xor } B \text{ xor } C$
- $F6 = A \text{ xnor } B \text{ xnor } C$

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

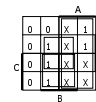


CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 7

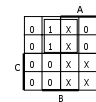
PLAs Design Example

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	-	-	-	-	-	-



K-map for W



K-map for X

minimized functions:

$$W = A + B D + B C$$

$$X = B C'$$

$$Y = B + C$$

$$Z = A' B' C' D + B C D + A D' + B' C D'$$



K-map for Y

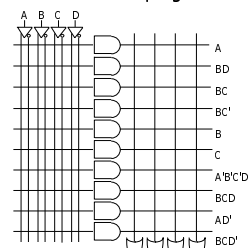


K-map for Z

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 8

PLAs Design Example (cont'd)

Code converter: programmed PLA



minimized functions:

$$W = A + B D + B C$$

$$X = B C'$$

$$Y = B + C$$

$$Z = A' B' C' D + B C D + A D' + B' C D'$$

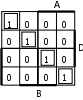
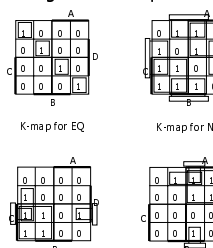
not a particularly good candidate for PLA implementation since no terms are shared among outputs

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

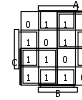
CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 9

PLA Second Design Example

Magnitude comparator



K-map for EQ



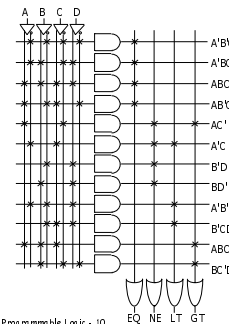
K-map for NE



K-map for LT



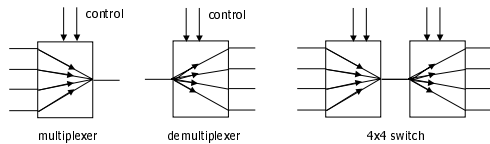
K-map for GT



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 10

Multiplexer/Demultiplexer: Making Connections

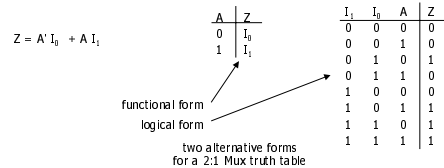
- Direct point-to-point connections between gates
- **Multiplexer:** route one of many inputs to a single output
- **Demultiplexer:** route single input to one of many outputs



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 11

Multiplexers/Selectors

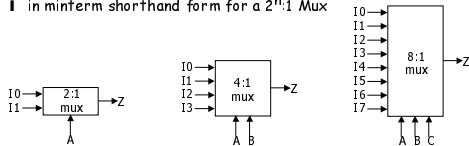
- **Multiplexers/Selectors: general concept**
 - 2^n data inputs, n control inputs (called "selectors"), 1 output
 - Used to connect 2^n points to a single point
 - Control signal pattern forms binary index of input connected to output



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 12

Multiplexers/Selectors (cont'd)

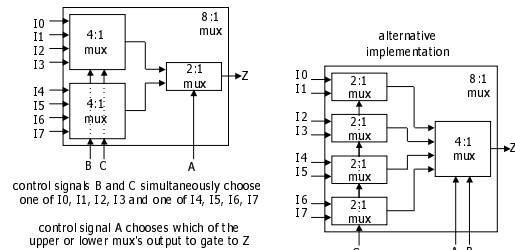
- 2:1 mux: $Z = A' I_0 + A I_1$
- 4:1 mux: $Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3$
- 8:1 mux: $Z = A' B' C' I_0 + A' B' C I_1 + A' B C' I_2 + A' B C I_3 + A B' C' I_4 + A B' C I_5 + A B C' I_6 + A B C I_7$
- In general, $Z = \sum_{k=0}^{2^n-1} (m_k I_k)$
 - in minterm shorthand form for a $2^n:1$ Mux



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 13

Cascading Multiplexers

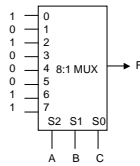
- Large multiplexers implemented by cascading smaller ones



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 14

Multiplexers as General-purpose Logic

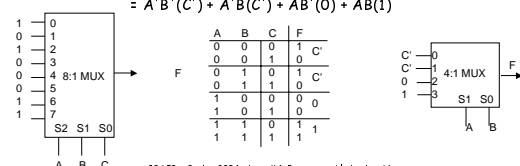
- $2^n:1$ multiplexer implements any function of n variables
 - With the variables used as control inputs and
 - Data inputs tied to 0 or 1
 - In essence, a lookup table
- Example:
 - $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
 - $= A'B'C' + A'BC' + ABC' + ABC$
 - $= A'B'(C') + A'B(C') + AB'(0) + AB(1)$



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 15

Multiplexers as General-purpose Logic (cont'd)

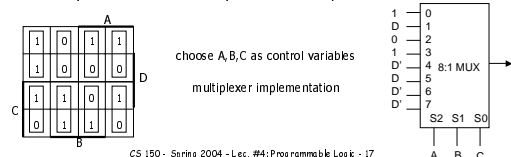
- $2^{n-1}:1$ mux can implement any function of n variables
 - With $n-1$ variables used as control inputs and
 - Data inputs tied to the last variable or its complement
- Example:
 - $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
 - $= A'B'C' + A'BC' + ABC' + ABC$
 - $= A'B'(C') + A'B(C') + AB'(0) + AB(1)$



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 16

Multiplexers as General-purpose Logic (cont'd)

- Generalization
 - $n-1$ mux control variables
 - single mux data variable
 - four possible configurations of truth table rows can be expressed as a function of I_n
- Example: $F(A,B,C,D)$ implemented by an 8:1 MUX



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 17

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:

$$\begin{aligned} O_0 &= G \cdot S' \\ O_1 &= G \cdot S \end{aligned}$$

3:8 Decoder:

$$\begin{aligned} O_0 &= G \cdot S_2' \cdot S_1' \cdot S_0' \\ O_1 &= G \cdot S_2' \cdot S_1' \cdot S_0 \\ O_2 &= G \cdot S_2' \cdot S_1 \cdot S_0' \\ O_3 &= G \cdot S_2' \cdot S_1 \cdot S_0 \\ O_4 &= G \cdot S_2 \cdot S_1' \cdot S_0' \\ O_5 &= G \cdot S_2 \cdot S_1' \cdot S_0 \\ O_6 &= G \cdot S_2 \cdot S_1 \cdot S_0' \\ O_7 &= G \cdot S_2 \cdot S_1 \cdot S_0 \end{aligned}$$

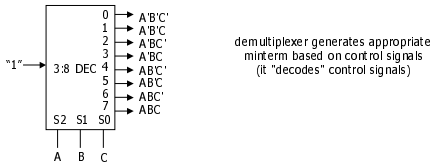
2:4 Decoder:

$$\begin{aligned} O_0 &= G \cdot S_1' \cdot S_0' \\ O_1 &= G \cdot S_1' \cdot S_0 \\ O_2 &= G \cdot S_1 \cdot S_0' \\ O_3 &= G \cdot S_1 \cdot S_0 \end{aligned}$$

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 18

Demultiplexers as General-purpose Logic

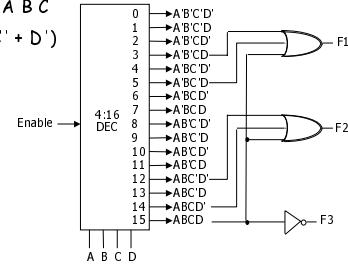
- $n:2^n$ decoder implements any function of n variables
 - With the variables used as control inputs
 - Enable inputs tied to 1 and
 - Appropriate minterms summed to form the function



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 19

Demultiplexers as General-purpose Logic (cont'd)

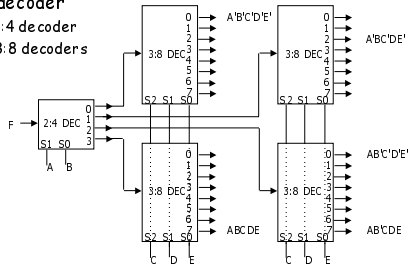
- $F1 = A'B'C'D + A'B'CD + ABC'D$
- $F2 = ABC'D + ABC$
- $F3 = (A' + B' + C' + D')$



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 20

Cascading Decoders

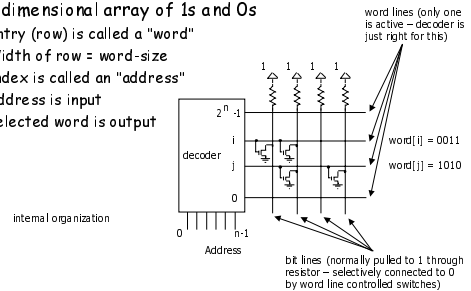
- 5:32 decoder
 - 1x2:4 decoder
 - 4x3:8 decoders



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 21

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



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 22

ROMs and Combinational Logic

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

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

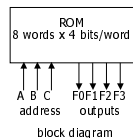
$$F1 = A'B'C + A'B'C' + ABC$$

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

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

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

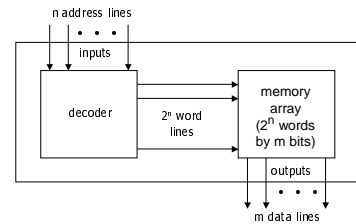


block diagram

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 23

ROM Structure

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



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 24

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)

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 25

Field-Programmable Gate Arrays

PLAs: 100s of gate equivalents

FPGAs: 1000-10000s gates

Logic blocks

- ! Implement combinational and sequential logic

Interconnect

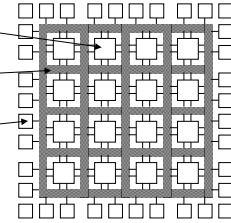
- ! Wires to connect inputs and outputs to logic blocks

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*



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 26

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%?

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 27

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

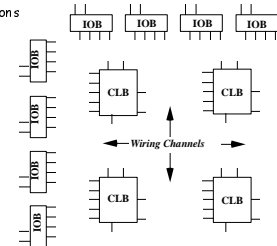
Can be used as memory

Three types of routing

- ! direct
- ! general-purpose
- ! long lines of various lengths

RAM-programmable

- ! can be reconfigured



CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 28

The Xilinx 4000 CLB

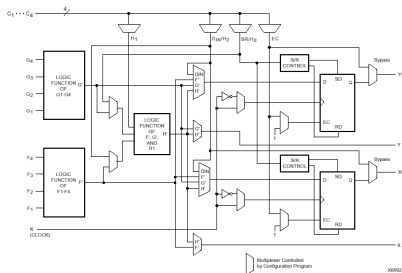


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 29

Two 4-input functions, registered output

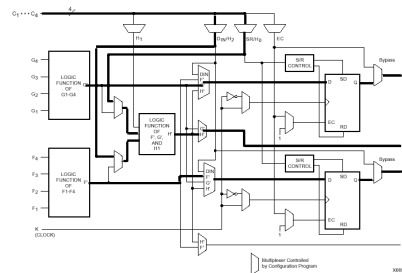


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

CS 150 - Spring 2004 - Lec. #4: Programmable Logic - 30

5-input function, combinational output

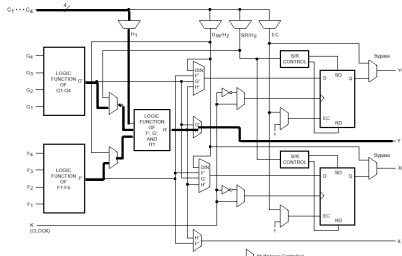


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

CLB Used as RAM

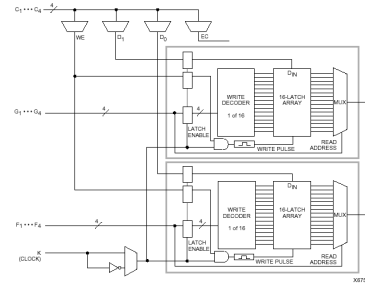


Figure 4: 16x2 (or 16x1) Edge-Triggered Single-Port RAM

Xilinx 4000 Interconnect

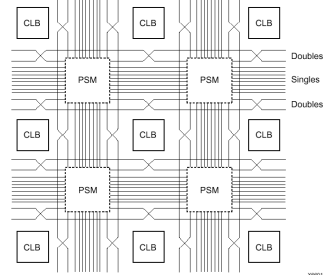


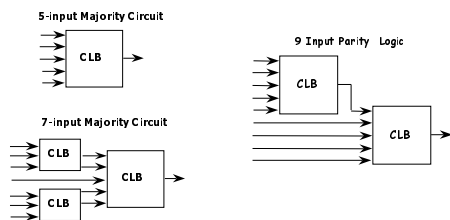
Figure 28: Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)

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:
 - $AB = CD$ and $AB > CD$ implemented with 1 CLB
 - (GT) $F = A C' + A B D' + B C' D'$
 - (BQ) $G = A'B'C'D' + A'B C'D + A B'C'D' + A B C D$
- Can implement some functions of > 5 input

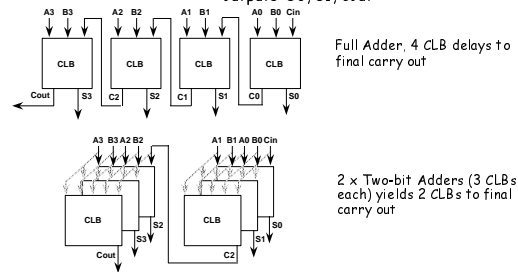
Xilinx FPGA Combinational Logic

- 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



Combinational Logic Implementation Summary

- Regular Logic Structures
 - PLAs/PALs
 - Multiplexers/decoders
 - ROMs
 - Advantages/disadvantages of each