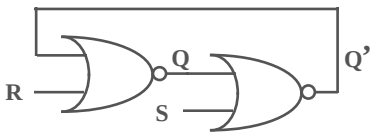
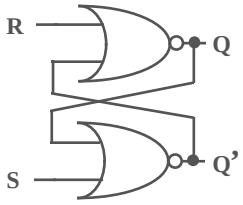
Outline

- Last time:
  - Design versus Implementation
  - Specification, Description, and Depiction
  - Conceptual Blocks & How to Overcome Them
  - Good Versus Bad Design
  - The Role of Language in Design
  - The CS150 Project
- This lecture:
  - Introduction to timing diagrams
  - SR latch
  - D latch
  - Flip-Flops
  - Shift Registers
  - Serial-Parallel Conversion

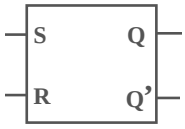
The Set-Reset (SR, RS) Latch



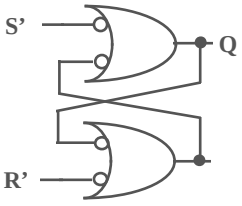


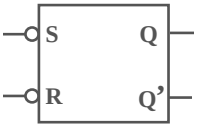
Latch Behavior

S	R	Q	Q'
0	0	last Q	last Q'
0	1	0	1
0	0	1	0
0	1	0	0



The S'-R' Latch

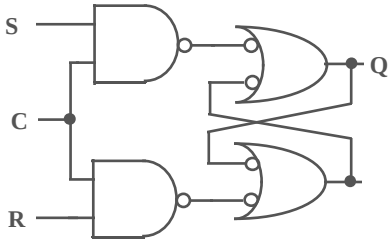


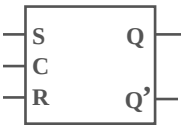


Latch Behavior

S'	R'	Q	Q'
0	0	0	0
0	1	1	0
1	0	0	1
1	1	last Q	last Q'

S-R Latch With Enable

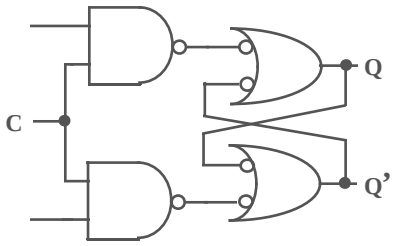




Latch Behavior

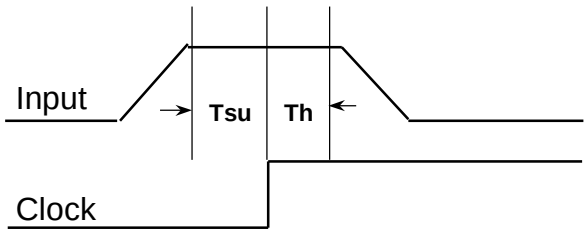
S	R	C	Q	Q'
0	0	1	last Q	last Q'
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
*	*	0	last Q	last Q'

Data (D) Latch

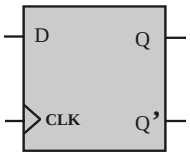


Setup and Hold Times for Latches

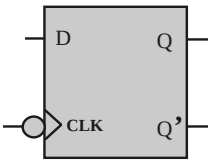
- **Setup Time (Tsu)** is the minimum time interval for which the input signal must be stable (unchanging) *prior to* the sampling event of the clock for the input signal to be recognized correctly.
- **Hold Time (Th)** is the minimum time interval for which the input signal must be stable (unchanging) *following* the sampling event of the clock for the input signal to be recognized correctly.



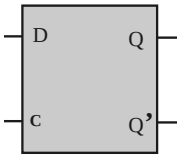
Standard Latch & Flip-Flop Conventions



Positive-Edge-Triggered D Flip-flop



Negative-Edge-Triggered D Flip-flop



D Latch with Enable

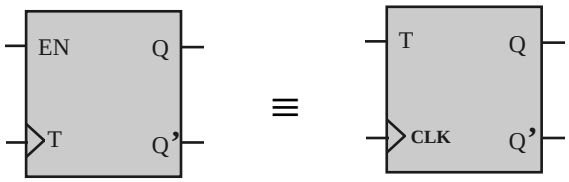
Input/Output Behavior of Latches and Flip-Flops

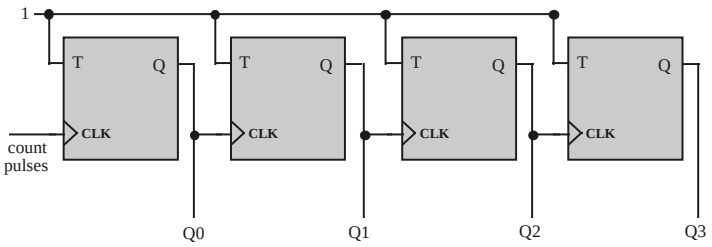
Type	When Inputs Sampled	When Outputs Valid	Tsu, Th Relative to:
Unclocked Latch	always	Tprop from input change	-
Level-Sensitive Latch	clock high	Tprop from input change	Falling clock edge
Positive edge triggered flip-flop	clock low → high transition	Tprop from rising clock edge	Rising clock edge
Negative edge triggered flip-flop	clock high → low transition	Tprop from falling clock edge	Falling clock edge
Master-slave flip-flop	clock high → low transition	Tprop from falling clock edge	Falling clock edge

Latch & Flip-Flop Characteristic Equations

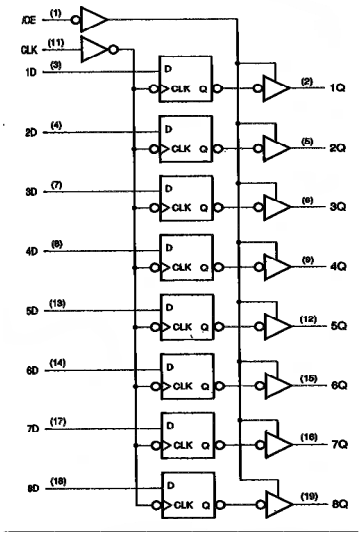
	S-R Latch	Qn+1 = S + R'Qn
	D latch	Qn+1 = D
*	Edge-triggered D Flip-Flop	Qn+1 = D
	Master-slave S-R Flip-Flop	Qn+1 = S + R'Qn
	Master-Slave J-K Flip-Flop	Qn+1 = JQn' + K'Qn
	Edge-Triggered J-K Flip-Flop	Qn+1 = JQn' + K'Qn
*	T Flip-Flop	Qn+1 = Qn'
*	T Flip-Flop with Enable	Qn+1 = EQn' + E'Qn = E ⊕ Qn

Toggle (T) Flip-Flop



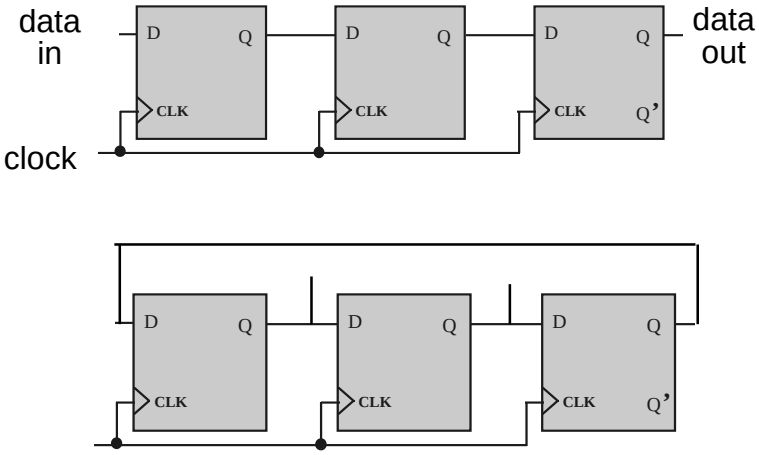


Registers

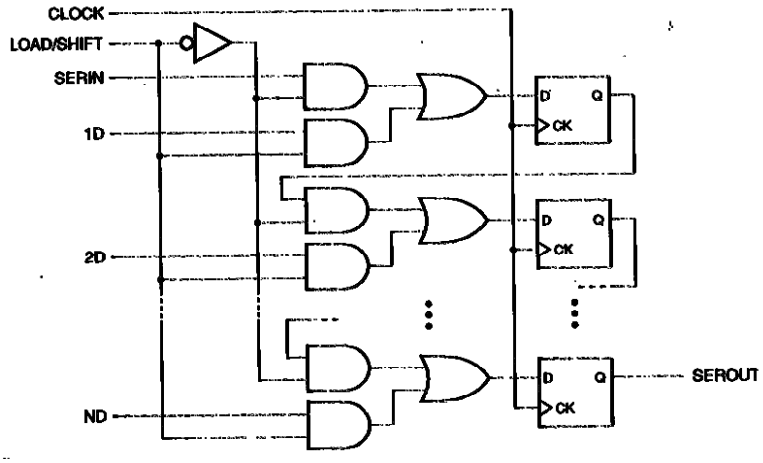


- *Tri-state Outputs*: Output can be logic-0, logic-1, or "high impedence" (off, disconnected). Very important for connections to a "bus".

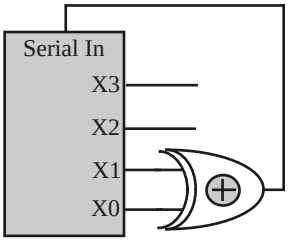
Shift Registers & Ring Counters



Parallel/Serial Shift Register



Linear Feedback Shift Registers (LFSR)



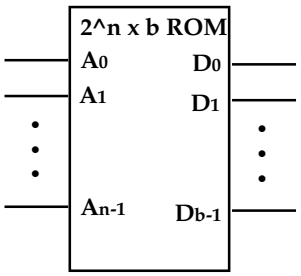
n	Feedback equation
2	X2 = X1 ⊕ X0
3	X3 = X1 ⊕ X0
4	X4 = X1 ⊕ X0
5	X5 = X2 ⊕ X0
6	X6 = X1 ⊕ X0
7	X7 = X3 ⊕ X0
8	X8 = X4 ⊕ X3 ⊕ X2 ⊕ X0
12	X12 = X6 ⊕ X4 ⊕ X1 ⊕ X0
16	X16 = X5 ⊕ X4 ⊕ X3 ⊕ X0
20	X20 = X3 ⊕ X0
24	X24 = X7 ⊕ X2 ⊕ X1 ⊕ X0
28	X28 = X3 ⊕ X0
32	X32 = X22 ⊕ X2 ⊕ X1 ⊕ X0

Outline

- Last time:
  - SR latch
  - D latch
  - Flip-Flops
  - Edge-triggered, Master-Slave
  - Shift Registers
  - Simple Counters
  - Serial-Parallel Conversion
  - LFSR Counters
- This lecture:
  - ROM, RAM
  - Bussing, Tri-state, multi-porting
  - Switch De-bouncing
  - FSM Design With ROMs

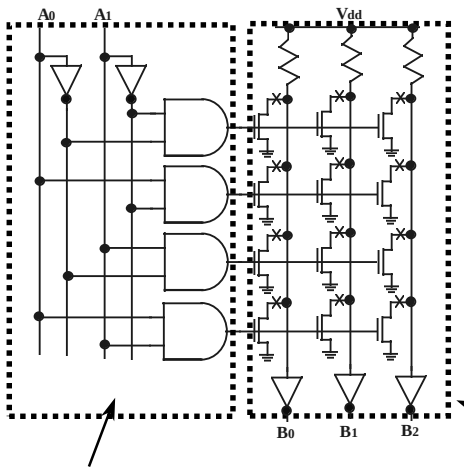
Read-Only Memory (ROM)

- A *Read-Only Memory* (ROM) is a combinational circuit with *n* address-line data inputs (A0, A1, ... An-1) and *b* (D0, D1, ... Db-1) data outputs. A ROM literally store the truth-table (minterms) of an n-input, b-output combinational circuit.



- A ROM is *nonvolatile* memory. That is, its contents are preserved even if no power is applied.

Read-Only Memory

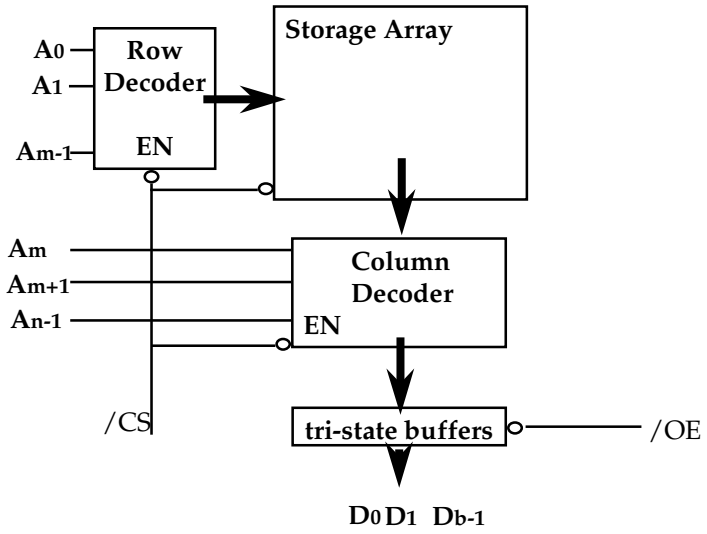


- Often NOR decoder is used (see PLA), but in ROM all minterms must be decoded.
- ROM is often used to implement "table look-up" functions (multiplier, most significant bits (MSBs) of exponential or square-root).

Classification of ROMs

- *Mask-programmable ROM*: Look-up array connections are programmed during semiconductor manufacture using a mask. Very fast (>5ns), very dense (~256Mbits), expensive ($20-50K+n*$0.50-20), 2-4week turn-around, low power.
- *Programmable ROM (PROM)*: Fuses (sometimes anti-fuses) to program once in the field, using PROM Programmer. Very fast, dense (~ 64Mbits), inexpensive ($2-100), 5min to write, low power.
- *Erasable Programmable ROM (EPROM)*: Fuses (sometimes anti-fuses) implemented using floating-gate MOSFETS to program 10K-100K times in the field, using EPROM Programmer. Fast (> 150ns), dense (~ 16Mbits), inexpensive ($2-30), 5min to write, low power, erased by flooding with UV light.
- *Electrically Erasable Programmable ROM (EEPROM, E2PROM)*: Same as EPROM, but specific bits may be erased selectively. Fast (> 150ns), dense (~ 16Mbits), inexpensive ($5-50), 5min to write, low power.
- *Flash Electrically Erasable Programmable ROM (Flash EEPROM, E2PROM)*: Same as E2PROM, but entire ROM may be erased electrically (no UV). Fast (> 100ns), dense (~ 64Mbits), inexpensive ($5-50), 5min to write, low power. Denser than E2PROM. *Very high growth in PCMCIA card/ portable electronics applications.*

Internal ROM Structure

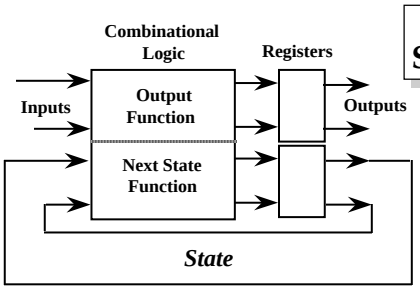


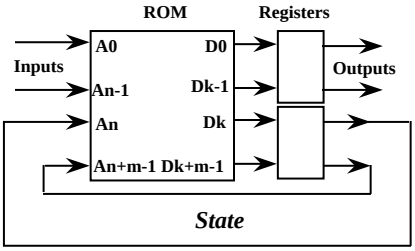
Read-Write Memory

- Static Random-Access Memory (SRAM): Storage Array implemented using flip-flops. Very fast (>3ns), dense (~16Mbits), relativel expensive. To write, Chip-select (/CS) and Write Enable (/WE) must be low. To read, /CS and Output Enable (/OE) must be low and /WE must be high.
- Dynamic Random-Access Memory (DRAM): Storage Array implemented using single transistor & a capacitor. Fast (>100ns), very dense (~64Mbits). Control signals are Row address strobe (/RAS), Column Address Strobe (/CAS) and Write-Enable (/WE).
- DRAMs use a destructive-read cycle, so the data must be written back after it is read.

Rom-Based FSM Design

Block Diagram for Synchronous Mealy Machine





ROM-based Realization

- **Inputs & Current State form the address**
- **ROM data bits form the Outputs & Next State**

Rom-Based FSM Design: Example

Example: BCD to Excess 3 Serial Converter

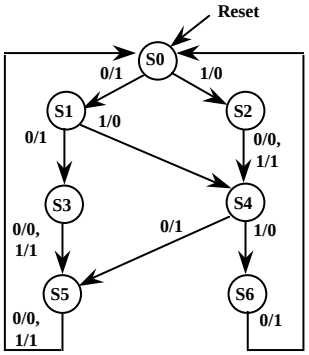
The bits are presented in bit-serial fashion starting with the least significant bit. There is a single input X, single output Z

BCD	Excess 3 Code
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100

Rom-Based FSM Design: Example

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	--	1	--

State Transition Table

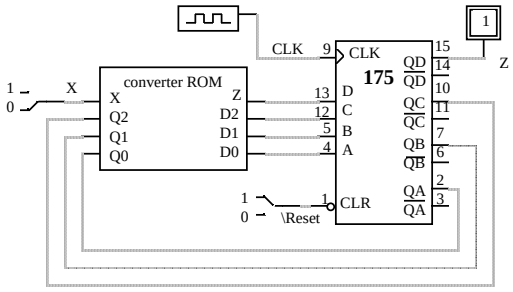


Derived State Diagram

Rom-Based FSM Design: Example

ROM Address X	Q2	Q1	Q0	ROM Outputs Z	D2	D1	D0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	1	0	0	0
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Truth Table/ROM I/Os



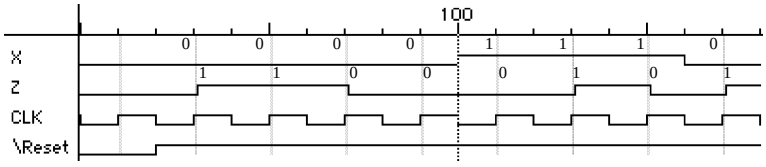
Circuit Level Realization
74175 = 4 x positive edge triggered D FFs

In ROM-based designs, no need to consider state assignment

Rom-Based FSM Design: Example

LSB MSB

Timing Behavior for input strings 0 0 0 0 (0) and 1 1 1 0 (7)

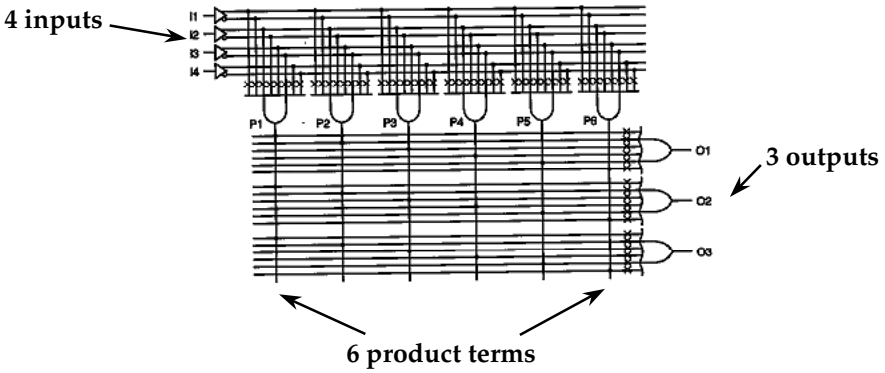


Outline

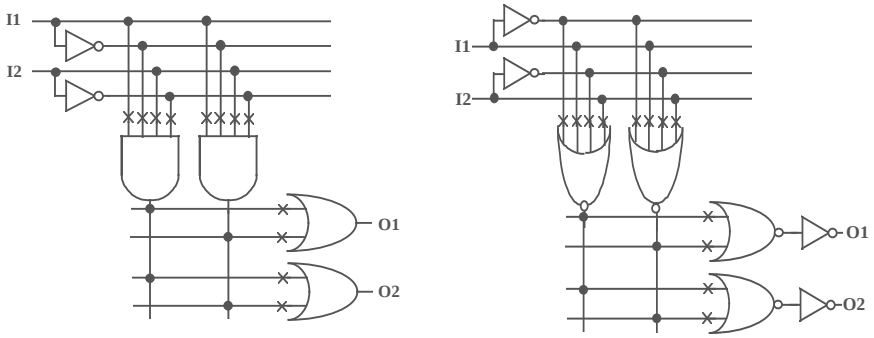
- Last time:
  - ROM, RAM
  - Bussing, Tri-state, multi-porting
  - Switch De-bouncing
  - FSM Design With ROMs
- This lecture:
  - LSI Implementation: PLAs, PLDs, PALs
  - "PLA Format"
  - PLD Structures
  - PLA-Based FSM Design
  - Intro. to FPGAs

Programmable Logic Arrays (PLAs)

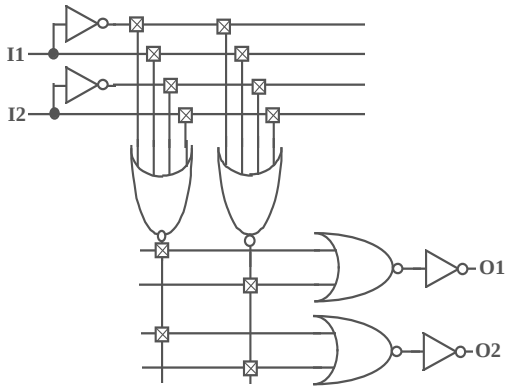
- We know that any logic function can be implemented in Truth-Table form, so can we build a general circuit structure that can implement a *simplified* ROM form directly?



AND-OR and NOR-NOR PLAs

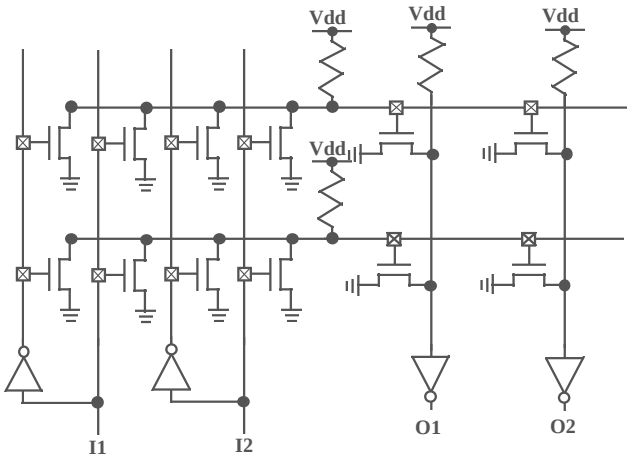


Alternate Programming Approach

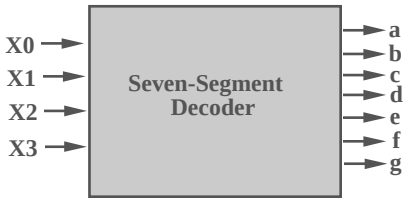


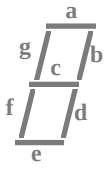
- Connections between horizontal and vertical lines on integrated circuit make by inserting a "via" or "cut" in the SiO2 between the mask layers.

Transistor-Level Implementation



Design Example: Seven-Segment Decoder

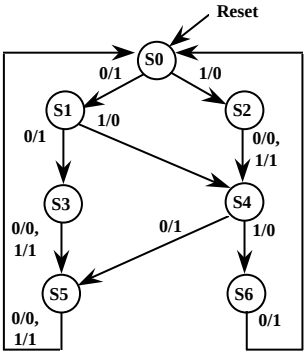




PLA-Based FSM Design: Example

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	--	1	--

State Transition Table



Derived State Diagram

PLA-Based FSM Design: Example

State Assignment with NOVA

```
0 S0 S1 1
1 S0 S2 0
0 S1 S3 1
1 S1 S4 0
0 S2 S4 0
1 S2 S4 1
0 S3 S5 0
1 S3 S5 1
0 S4 S5 1
1 S4 S6 0
0 S5 S0 0
1 S5 S0 1
0 S6 S0 1
```

NOVA input file

S0 = 000
S1 = 001
S2 = 011
S3 = 110
S4 = 100
S5 = 111
S6 = 101

NOVA derived state assignment

9 product term implementation

PLA-Based FSM Design: Example

Espresso Inputs

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 16
0 000 001 1
1 000 011 0
0 001 110 1
1 001 100 0
0 011 100 0
1 011 100 1
0 110 111 0
1 110 111 1
0 100 111 1
1 100 101 0
0 111 000 0
1 111 000 1
0 101 000 1
1 101 - - - -
0 010 - - - -
1 010 - - - -
.e
```

"-" means "don't connect"

Espresso Outputs

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 9
0001      0100
10-0      0100
01-0      0100
1-1-      0001
-0-1      1000
0-0-      0001
-1-0      1000
- -10     0100
- - -0    0010
.e
```

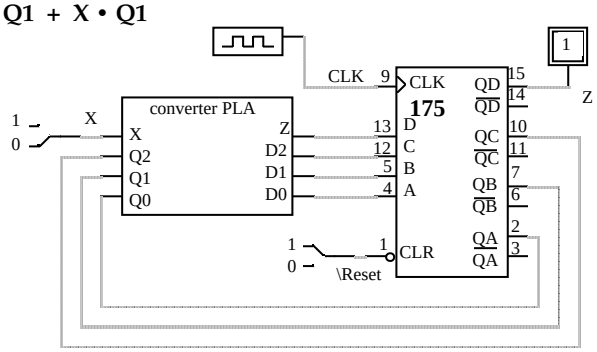
PLA-Based FSM Design: Example

D2 = $\overline{Q2}$ • Q0 + Q2 • $\overline{Q0}$

D1 = $\overline{X}$ • $\overline{Q2}$ • $\overline{Q1}$ • Q0 + X • $\overline{Q2}$ • $\overline{Q0}$ + $\overline{X}$ • Q2 • $\overline{Q0}$ + Q1 • $\overline{Q0}$

D0 = $\overline{Q0}$

Z = X• Q1 + $\overline{X}$ • $\overline{Q1}$



Remember: Good tools cannot guarantee a good design, but if you undertstand their limitations, they can reduce the time spent on the less creative aspects of design.

PAL-Based FSM Design: Example

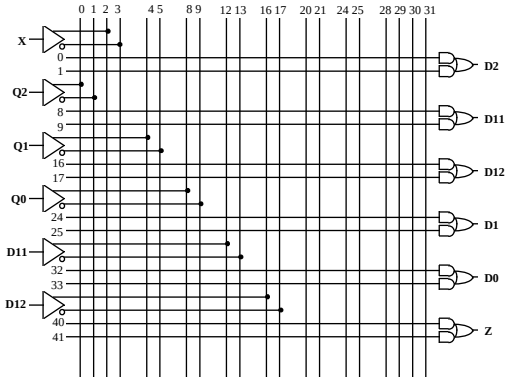
10H8 PAL: 10 inputs, 8 outputs, 2 product terms per OR gate

D1 = D11 + D12

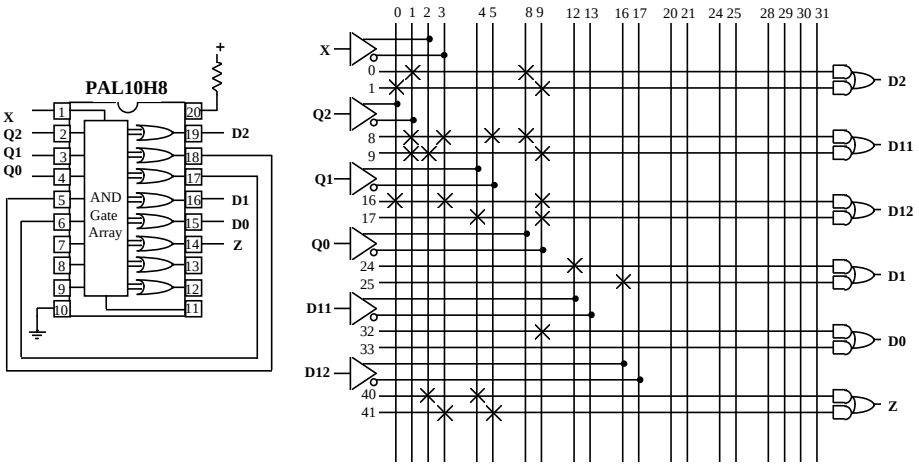
D11 = $\overline{X}$ • $\overline{Q2}$ • $\overline{Q1}$ • Q0 + X • $\overline{Q2}$ • $\overline{Q0}$

D12 = $\overline{X}$ • Q2 • $\overline{Q0}$ + Q1 • $\overline{Q0}$

0. Q2' • Q0
1. Q2 • Q0'
8. X' • Q2' • Q1' • Q0
9. X • Q2' • Q0'
16. X' • Q2 • Q0'
17. Q1 • Q0'
24. D11
25. D12
32. Q0'
33. not used
40. X • Q1
41. X '• Q1'

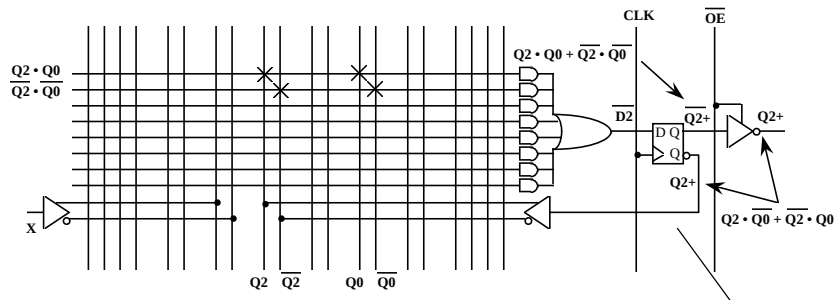


PAL-Based FSM Design: Example



Advanced PAL-Based FSM Design: Example

Registered PAL Architecture



$\overline{D2} = Q2 \cdot Q0 + \overline{Q2} \cdot \overline{Q0}$

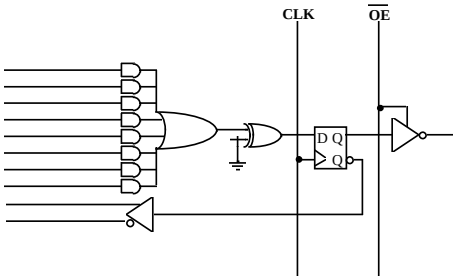
$\overline{D1} = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot \overline{Q0} + X \cdot Q2 + X \cdot Q0 + Q2 \cdot Q0 + Q1 \cdot Q0$

$\overline{D0} = Q0$

$\overline{Z} = X \cdot \overline{Q1} + \overline{X} \cdot Q1$

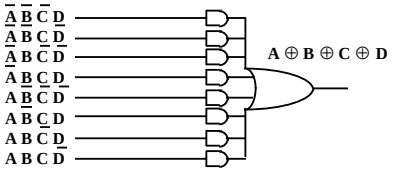
PAL-Based FSM Design: Example

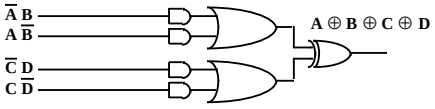
Programmable Output Polarity/XOR PALs



Buried Registers: decouple FF from the output pin

Advantage of XOR PALs: Parity and Arithmetic Operations





PAL-Based FSM Design

Specifying PALs with ABEL

P10H8 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter State Machine'
u1 device 'p10h8';

"Input Pins
 X,Q2,Q1,Q0,D11i,D12i  pin  1,2,3,4,5,6;

"Output Pins
 D2,D11o,D12o,D1,D0,Z  pin  19,18,17,16,15,14;

INSTATE = [Q2, Q1, Q0];
S0 = [0, 0, 0];
S1 = [0, 0, 1];
S2 = [0, 1, 1];
S3 = [1, 1, 0];
S4 = [1, 0, 0];
S5 = [1, 1, 1];
S6 = [1, 0, 1];

equations
 D2 = (!Q2 & Q0) # (Q2 & !Q0);
 D1 = D11i # D12i;
 D11o = (!X & !Q2 & !Q1 & Q0) # (X & !Q2 & !Q0);
 D12o = (!X & Q2 & !Q0) # (Q1 & !Q0);
 D0 = !Q0;
 Z = (X & Q1) # (!X & !Q1);

end bcd2excess3;
```

Explicit equations for partitioned output functions

PAL-Based FSM Design

Specifying PALs with ABEL

P12H6 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter State Machine'
u1 device  'p12h6';

"Input Pins
 X, Q2, Q1, Q0  pin  1, 2, 3, 4;

"Output Pins
 D2, D1, D0, Z  pin  17, 18, 16, 15;

INSTATE = [Q2, Q1, Q0]; OUTSTATE = [D2, D1, D0];
S0in = [0, 0, 0];  S0out = [0, 0, 0];
S1in = [0, 0, 1];  S1out = [0, 0, 1];
S2in = [0, 1, 1];  S2out = [0, 1, 1];
S3in = [1, 1, 0];  S3out = [1, 1, 0];
S4in = [1, 0, 0];  S4out = [1, 0, 0];
S5in = [1, 1, 1];  S5out = [1, 1, 1];
S6in = [1, 0, 1];  S6out = [1, 0, 1];

equations
 D2 = (!Q2 & Q0) # (Q2 & !Q0);
 D1 = (!X & !Q2 & !Q1 & Q0) # (X & !Q2 & !Q0) #
  (!X & Q2 & !Q0) # (Q1 & !Q0);
D0 = !Q0;
 Z = (X & Q1) # (!X & !Q1);

end bcd2excess3;
```

Simpler equations

PAL-Based FSM Design

Specifying PALs with ABEL

P16R4 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter'
u1 device 'p16r4';

"Input Pins
 Clk, Reset, X, !OE    pin  1, 2, 3, 11;

"Output Pins
 D2, D1, D0, Z    pin 14, 15, 16, 13;

SREG = [D2, D1, D0];
S0 = [0, 0, 0];
S1 = [0, 0, 1];
S2 = [0, 1, 1];
S3 = [1, 1, 0];
S4 = [1, 0, 0];
S5 = [1, 1, 1];
S6 = [1, 0, 1];
```

```
state_diagram SREG
state S0: if Reset then S0
    else if X then S2 with Z = 0
    else S1 with Z = 1
state S1: if Reset then S0
    else if X then S4 with Z = 0
    else S3 with Z = 1
state S2: if Reset then S0
    else if X then S4 with Z = 1
    else S4 with Z = 0
state S3: if Reset then S0
    else if X then S5 with Z = 1
    else S5 with Z = 0
state S4: if Reset then S0
    else if X then S6 with Z = 0
    else S5 with Z = 1
state S5: if Reset then S0
    else if X then S0 with Z = 1
    else S0 with Z = 0
state S6: if Reset then S0
    else if !X then S0 with Z = 1

end bcd2excess3;
```

FSM Design with More Sophisticated PLDs

Programmable Logic Devices = PLD

PALs, PLAs = 10 - 1,000s Gate Equivalents

Field Programmable Gate Arrays = FPGAs

- Altera MAX Family
- Actel Programmable Gate Array
- Xilinx Logical Cell Array

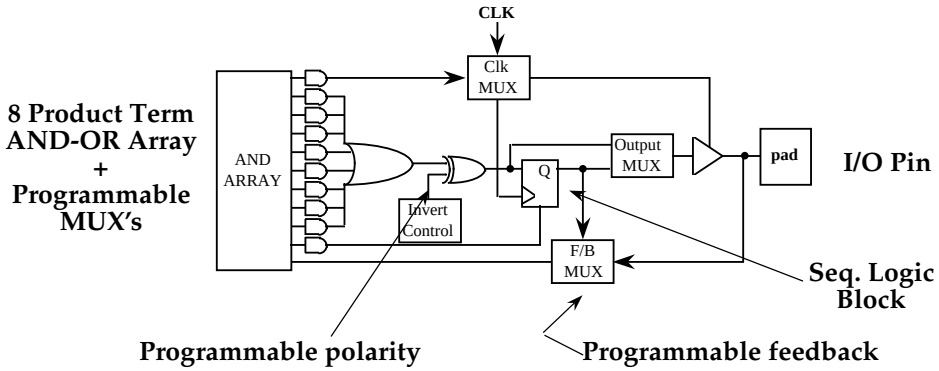
1,000s - 10,000s of Gate Equivalents!

FSM Design with More Sophisticated PLDs

Altera EPLD (Erasable Programmable Logic Devices)

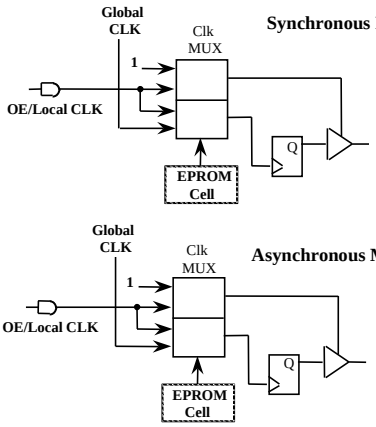
Historical Perspective:
PALs – same technology as programmed once bipolar PROM
EPLDs — CMOS erasable programmable ROM (EPROM)
erased by UV light

Altera building block = MACROCELL



FSM Design with More Sophisticated PLDs

Altera EPLDs contain 8 to 48 independently programmed macrocells
Personalized by EPROM bits:



Flipflop controlled by global clock signal

local signal computes output enable

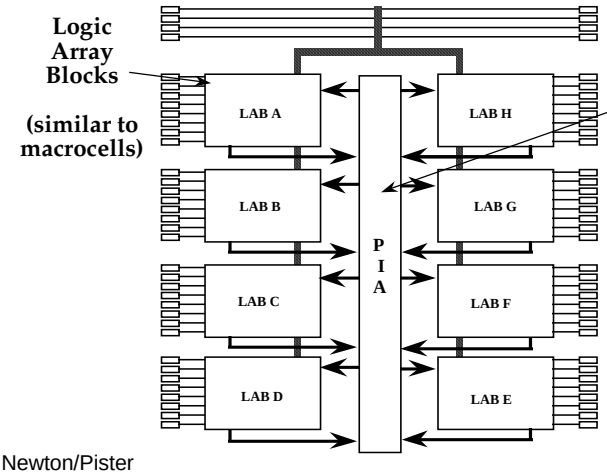
Flipflop controlled by locally generated clock signal

- **Seq Logic: could be D, T positive or negative edge triggered**
- **product term to implement clear function**

FSM Design with More Sophisticated PLDs

AND-OR structures are relatively limited
Cannot share signals/product terms among macrocells

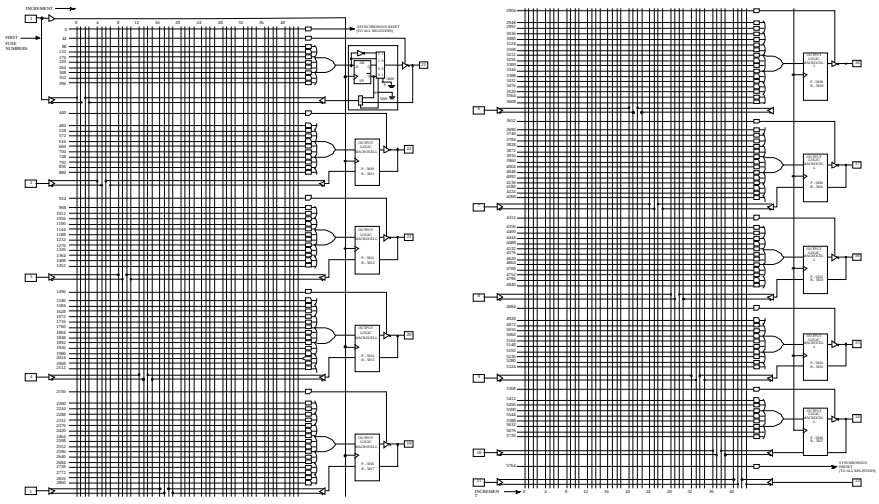
Altera solution: Multiple Array Matrix (MAX)



EPM5128:
8 Fixed Inputs
52 I/O Pins
8 LABs
16 Macrocells/LAB
32 Expanders/LAB

FSM Design with More Sophisticated PALs

P22V10 PAL



Supports large number of product terms per output
Latches and muxes associated with output pins

Outline

- Last time:
  - LSI Implementation: PLAs, PLDs, PALs
  - "PLA Format"
  - PLD Structures
  - PLA-Based FSM Design
  - Intro. to FPGAs
- This lecture:
  - Field-Programmable Logic Arrays
  - FPGA-Based FSM Design
  - Intro. to the Xilinx FPGA

FSM Design with FPGAs

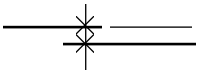
Actel Programmable Gate Arrays

Rows of programmable logic building blocks

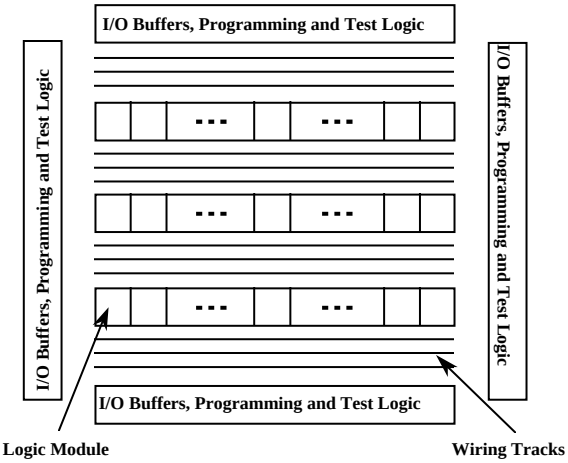
+

rows of interconnect

Anti-fuse Technology: Program Once



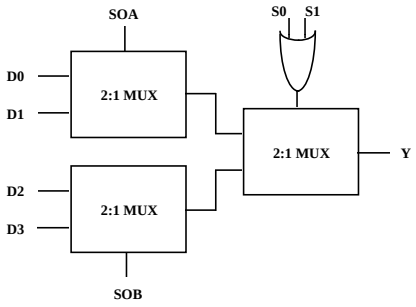
Use Anti-fuses to build up long wiring runs from short segments



8 input, single output combinational logic blocks
FFs constructed from discrete cross coupled gates

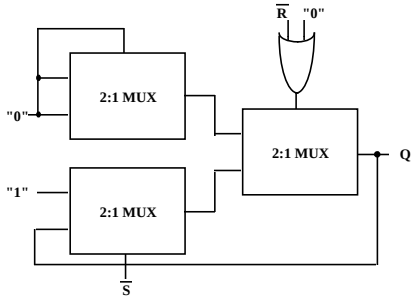
FSM Design with FPGAs

Actel Logic Module



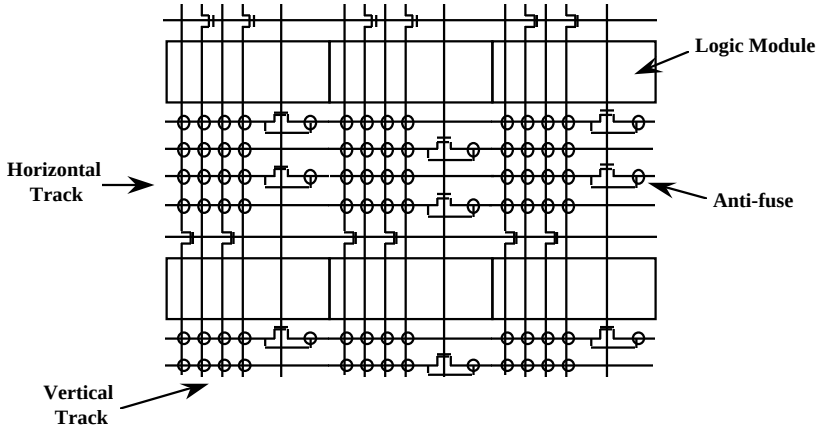
Basic Module is a Modified 4:1 Multiplexer

Example:
Implementation of S-$\overline{R}$ $\overline{\text{Latch}}$



FSM Design with FPGAs

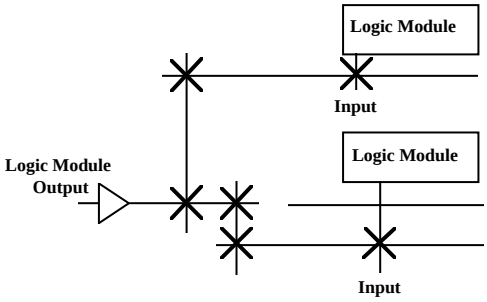
Actel Interconnect



Interconnection Fabric

FSM Design with FPGAs

Actel Routing Example



Jogs cross an anti-fuse

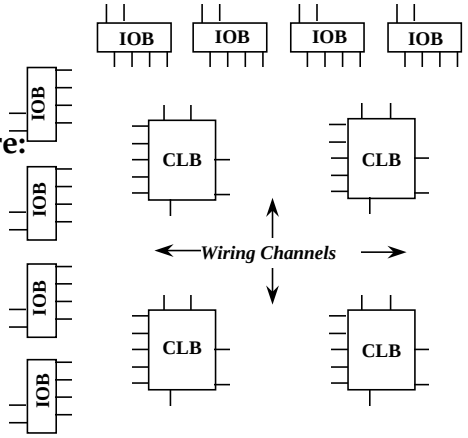
minimize the # of jobs for speed critical circuits

2 - 3 hops for most interconnections

FSM Design with FPGAs
Xilinx Logic Cell Arrays

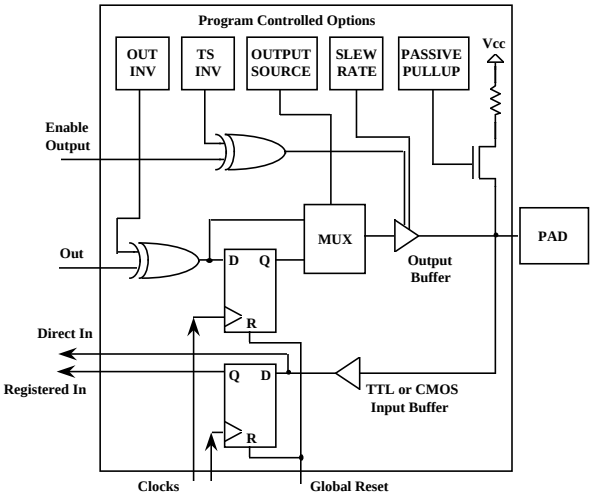
CMOS Static RAM Technology: programmable on the fly!
All personality elements connected into serial shift register
Shift in string of 1's and 0's on power up

General Chip Architecture:
- **Logic Blocks (CLBs)**
- **IO Blocks (IOBs)**
- **Wiring Channels**



FSM Design with FPGAs
Xilinx LCA Architecture

- **Inputs:**
  Tri-state enable bit to output input, output clocks
- **Outputs:**
  input bit
- **Internal FFs for input & output paths**
- **Fast/Slow outputs**
  5 ns vs. 30 ns rise
- **Pull-up used with unused IOBs**

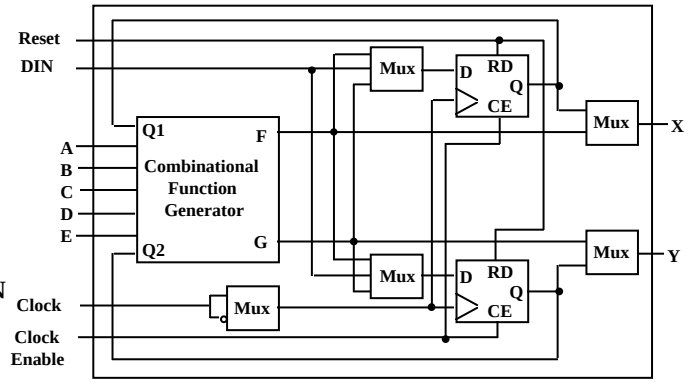


FSM Design with FPGAs

Xilinx LCA Architecture

Configurable Logic Block: CLB

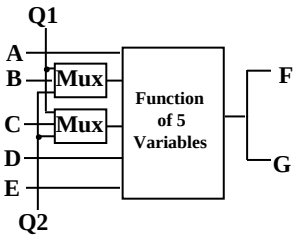
- 2 FFs
- Any function of 5 Variables
- Global Reset
- Clock, Clock Enb
- Independent DIN



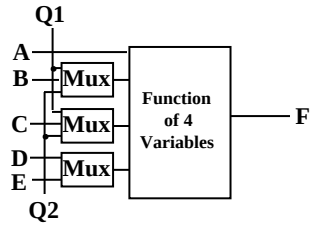
FSM Design with FPGAs

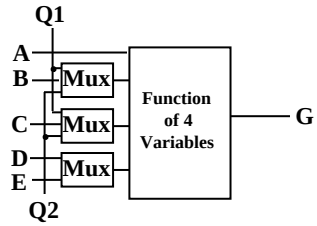
Xilinx LCA Architecture

CLB Function Generator



Any function of 5 variables



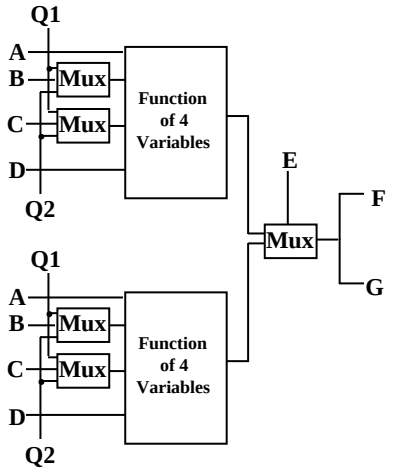


Two Independent Functions of 4 variables each

FSM Design with FPGAs

Xilinx LCA Architecture

CLB Function Generator



Certain Limited Functions of 6 Variables

FSM Design with FPGAs

Xilinx Application Examples

5-Input Parity Generator

Implemented with 1 CLB:

$$F = \overline{A \text{ xor } B \text{ xor } C \text{ xor } D \text{ xor } E}$$

2-bit Comparator: A B = C D or A B > C D

Implemented with 1 CLB:

(GT) $F = A\bar{C} + AB\bar{D} + \bar{B}C\bar{D}$

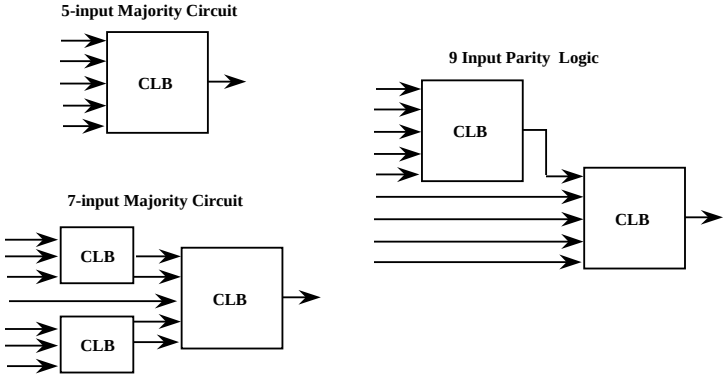
(EQ) $G = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + A\bar{B}C\bar{D}$

FSM Design with FPGAs

Xilinx Application Examples

n-Input Majority Circuit

Assert 1 whenever n/2 or greater inputs are 1



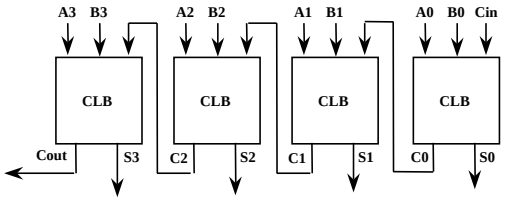
n-input Parity Functions

5 input = 1 CLB, 2 Levels of CLBs yield up to 25 inputs!

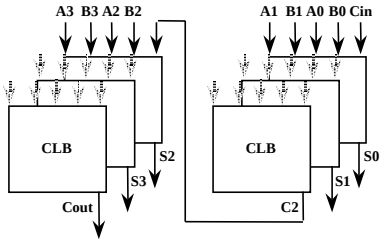
FSM Design with FPGAs

Xilinx Application Examples

4-bit Binary Adder



Full Adder, 4 CLB delays to final carry out



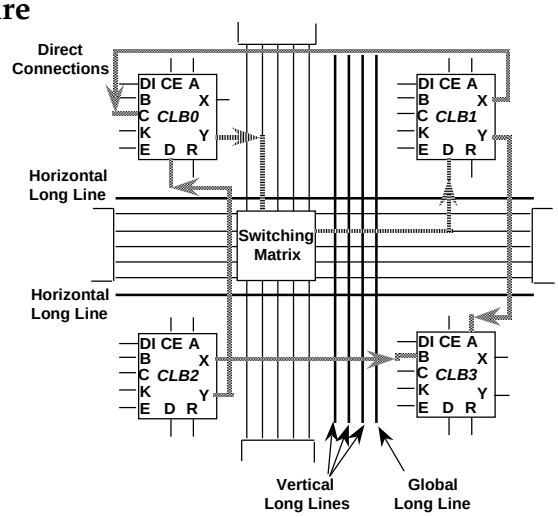
2 x Two-bit Adders (3 CLBs each) yields 2 CLBs to final carry out

FSM Design with FPGAs

Xilinx LCA Architecture

- Interconnect
- Direct Connections
- Global Long Line
- Horizontal/Vertical Long Lines

vSwitching Matrix Connections



FSM Design with FPGAs: Example

Xilinx LCA Architecture

Implementing the BCD to Excess 3 FSM

$Q2+ = \overline{Q2} \cdot Q0 + Q2 \cdot \overline{Q0}$

$Q1+ = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + X \cdot \overline{Q2} \cdot \overline{Q0} + \overline{X} \cdot Q2 \cdot \overline{Q0} + Q1 \cdot \overline{Q0}$

$Q0+ = \overline{Q0}$

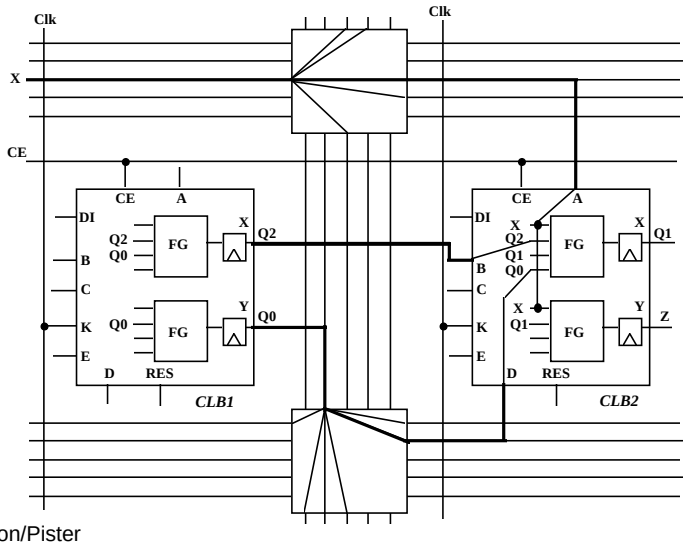
$Z = Z \cdot Q1 + \overline{X} \cdot \overline{Q1}$

- No function more complex than 4 variables
  4 FFs implies 2 CLBs
- Synchronous Mealy Machine
- Global Reset to be used
- Place Q2+, Q0+ in once CLB
  Q1, Z in second CLB
  maximize use of direct & general purpose interconnections

FSM Design with FPGAs: Example

Xilinx LCA Architecture

Implementing the BCD to Excess 3 FSM

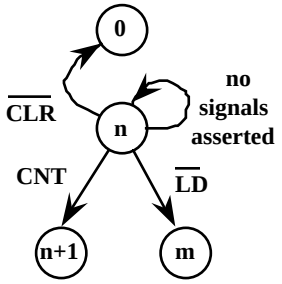


FSM Design with Counters

Synchronous Counters: CLR, LD, CNT

Four kinds of transitions for each state:

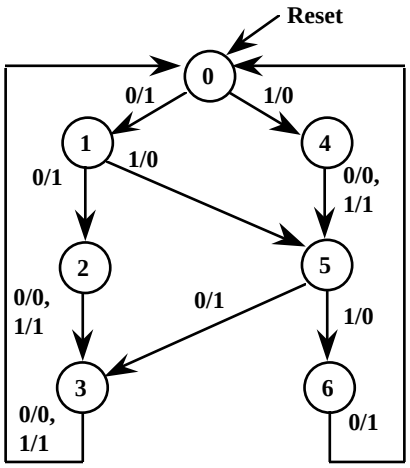
1. **to State 0 (CLR)**
2. **to next state in sequence (CNT)**
3. **to arbitrary next state (LD)**
4. **loop in current state**



Careful state assignment is needed to reflect basic sequencing of the counter

FSM Design with Counters: Example

Excess 3 Converter Revisited



Note the sequential nature of the state assignments

FSM Design with Counters

Excess 3 Converter

Inputs/Current State				*Next State*			*Outputs*						
X	Q2	Q1	Q0	Q2+	Q1+	Q0+	Z	$\overline{CLR}$	$\overline{LD}$	EN	C	B	A
0	0	0	0	0	0	1	1	1	1	1	X	X	X
0	0	0	1	0	1	0	1	1	1	1	X	X	X
0	0	1	0	0	1	1	0	1	1	1	X	X	X
0	0	1	1	0	0	0	0	0	X	X	X	X	X
0	1	0	0	1	0	1	1	1	1	1	X	X	X
0	1	0	1	0	1	1	0	1	0	X	0	1	0
0	1	1	0	0	0	0	1	0	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	1	0	0	0	1	0	X	1	0	0
1	0	0	1	1	0	1	0	1	0	X	1	0	1
1	0	1	0	0	1	1	1	1	1	1	X	X	X
1	0	1	1	0	0	0	1	0	X	X	X	X	X
1	1	0	0	1	0	1	0	1	1	1	X	X	X
1	1	0	1	1	1	0	1	1	1	1	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

CLR signal dominates LD which dominates Count

FSM Design with Counters

Excess 3 Converter

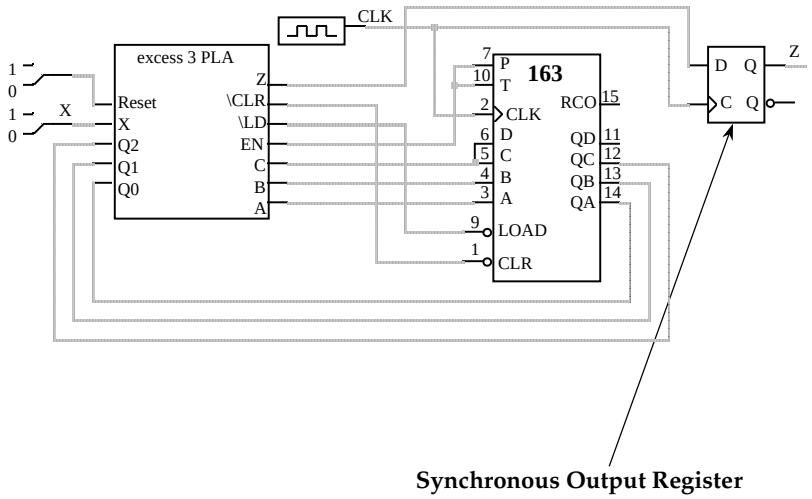
Espresso Input File

```
.i 5
.o 7
.ilb res x q2 q1 q0
.ob z clr ld en c b a
.p 17
1---- -0-----
00000 1111---
00001 1111---
00010 0111---
00011 00-----
00100 0111---
00101 110-011
00110 10-----
00111 -------
01000 010-100
01001 010-101
01010 1111---
01011 10-----
01100 1111---
01101 0111---
01110 -------
01111 -------
.e
```

Espresso Output File

```
.i 5
.o 7
.ilb res x q2 q1 q0
.ob z clr ld en c b a
.p 10
0-001 0101101
-0-01 1000000
-11-0 1000000
0-0-0 0101100
-000- 1010000
-0--0 0010000
0-10- 0101011
--11- 1000000
-11-- 0010000
-1-1- 1010000
.e
```


FSM Design with Counters



Synchronous Output Register

Design Case Study

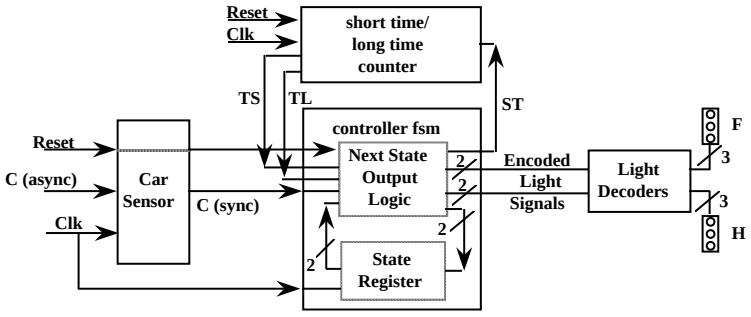
Traffic Light Controller

Decomposition into primitive subsystems

- **Controller FSM**
  next state/output functions
  state register
- **Short time/long time interval counter**
- **Car Sensor**
- **Output Decoders and Traffic Lights**

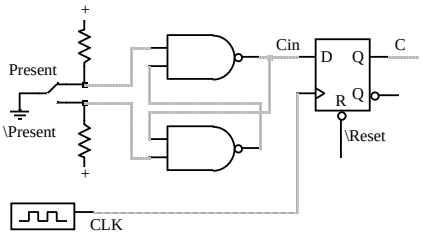
Design Case Study

Block Diagram



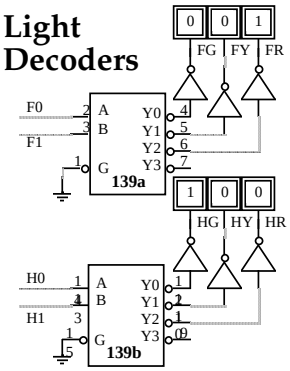
Design Case Study

Subsystem Logic

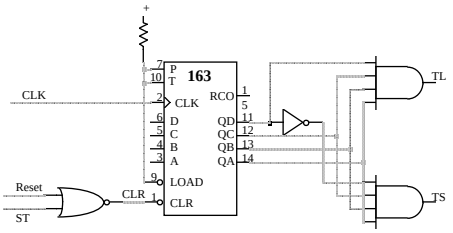


Car Detector

Light Decoders



Interval Timer



Design Case Study

Next State Logic

State Assignment: HG = 00, HY = 10, FG = 01, FY = 11

P1 = C TL $\overline{Q1}$ + $\overline{TS}$ Q1 $\overline{Q0}$ + $\overline{C}$ $\overline{Q1}$ Q0 + $\overline{TS}$ Q1 Q0

P0 = TS Q1 $\overline{Q0}$ + $\overline{Q1}$ Q0 + $\overline{TS}$ Q1 Q0

ST = C TL $\overline{Q1}$ + $\overline{C}$ $\overline{Q1}$ Q0 + TS Q1 $\overline{Q0}$ + TS Q1 Q0

HL[1] = TS Q1 Q0 + $\overline{Q1}$ Q0 + $\overline{TS}$ Q1 Q0

HL[0] = $\overline{TS}$ Q1 $\overline{Q0}$ + TS Q1 $\overline{Q0}$

FL[1] = $\overline{Q0}$

FL[0] = TS Q1 Q0 + $\overline{TS}$ Q1 Q0

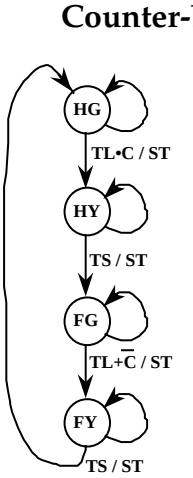
PAL/PLA Implementation:
5 inputs, 7 outputs, 8 product terms
PAL 22V10 -- 11 inputs, 10 prog. IOs, 8 to 14 prod terms per OR

ROM Implementation:
32 word by 8-bit ROM (256 bits)
Reset may double ROM size

Design Case Study

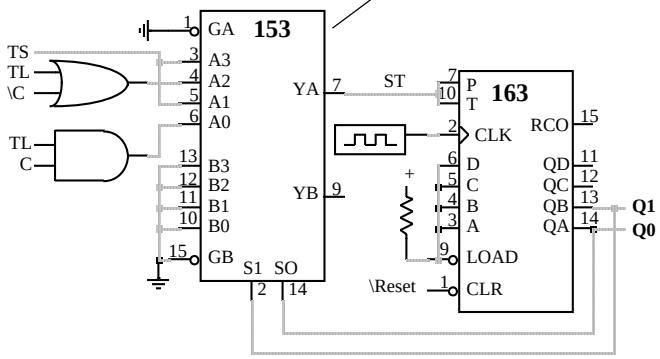
Next State Logic

Counter-based Implementation



ST = Count

2 x 4:1 MUX



TTL Implementation with MUX and Counter

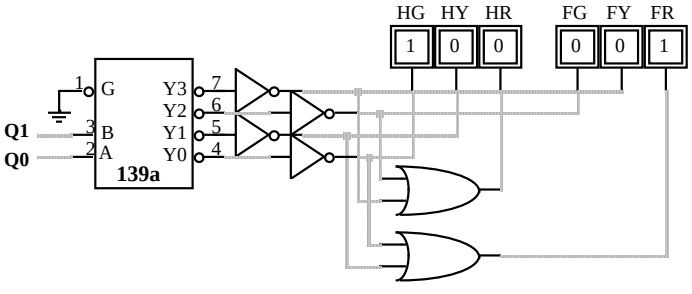
Can we reduce package count by using an 8:1 MUX?

Design Case Study

Next State Logic

Counter-based Implementation

Dispense with direct output functions for the traffic lights
Why not simply decode from the current state?



ST is a Synchronous Mealy Output

Light Controllers are Moore Outputs

Design Case Study

LCA-Based Implementation

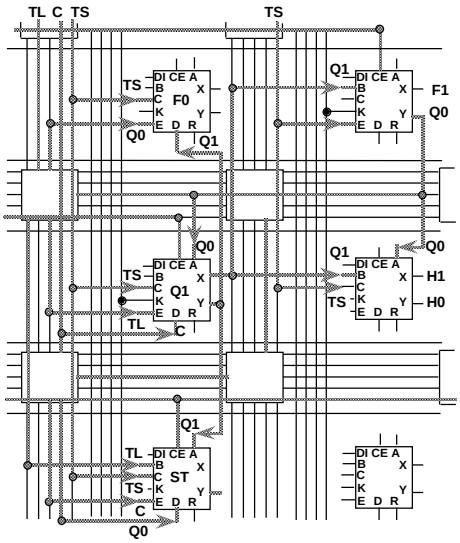
Discrete Gate Method:

- **None of the functions exceed 5 variables**
- **P1, ST are 5 variable (1 CLB each)**
- **P0, HL1, HL0, FL0 are 3 variable (1/2 CLB each)**
- **FL1 is 1 variable (1/2 CLB)**
- **4 1/2 CLBs total!**

Design Case Study

LCA-Based Implementation

Placement of functions selected to maximize the use of direct connections



Design Case Study

LCA-Based Implementation

Counter/Multiplexer Method:

- **4:1 MUX, 2 Bit Upcounter**
- **MUX: six variables (4 data, 2 control)**
  >> but this is the kind of 6 variable function that can be implemented in 1 CLB!
  >> 2nd CLB to implement TL • C and TL + C'
  >> But note that ST/Cnt is really a function of TL, C, TS, Q1, Q0
  >> 1 CLB to implement this function of 5 variables!
- **2 Bit Counter: 2 functions of 3 variables (2 bit state + count)**
  >> Also implemented in one CLB
- **Traffic light decoders: functions of 2 variables (Q1, Q0)**
  >> 2 per CLB = 3 CLB for the six lights
- **Total count = 5 CLBs**