

A-1 Appendix A - Digital Logic

Principles of Computer Architecture

Miles Murdocca and Vincent Heuring

Appendix A: Digital Logic

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-2 Appendix A - Digital Logic

Chapter Contents

| | |
|--|--------------------------------------|
| A.1 Introduction | A.11 Sequential Logic |
| A.2 Combinational Logic | A.12 Design of Finite State Machines |
| A.3 Truth Tables | A.13 Mealy vs. Moore Machines |
| A.4 Logic Gates | A.14 Registers |
| A.5 Properties of Boolean Algebra | A.15 Counters |
| A.6 The Sum-of-Products Form, and Logic Diagrams | |
| A.7 The Product-of-Sums Form | |
| A.8 Positive vs. Negative Logic | |
| A.9 The Data Sheet | |
| A.10 Digital Components | |

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-3 Appendix A - Digital Logic

Some Definitions

- **Combinational logic:** a digital logic circuit in which logical decisions are made based only on combinations of the inputs. e.g. an adder.
- **Sequential logic:** a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs. e.g. a memory unit.
- **Finite state machine:** a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state. e.g. a vending machine controller.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-4 Appendix A - Digital Logic

The Combinational Logic Unit

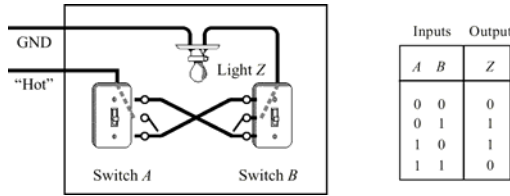
- Translates a set of inputs into a set of outputs according to one or more mapping functions.
- Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 v. and 0 v. for example.
- The outputs of a CLU are strictly functions of the inputs, and the outputs are updated immediately after the inputs change. A set of inputs $i_0 - i_n$ are presented to the CLU, which produces a set of outputs according to mapping functions $f_0 - f_m$

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

Truth Tables

- Developed in 1854 by George Boole
- further developed by Claude Shannon (Bell Labs)
- Outputs are computed for all possible input combinations (how many input combinations are there?)

Consider a room with two light switches. How must they work?*



*Don't show this to your electrician, or wire your house this way. This circuit definitely violates the electric code. The practical circuit never leaves the lines to the light "hot" when the light is turned off. Can you figure how?

Alternate Assignments of Outputs to Switch Settings

- Logically identical truth table to the original (see previous slide), if the switches are configured up-side down.

| Inputs | | Output |
|--------|---|--------|
| A | B | Z |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Truth Tables Showing All Possible Functions of Two Binary Variables

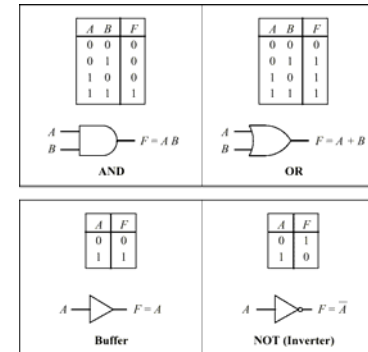
| Inputs | | Outputs | | | | | | | |
|--------|---|---------|-----|-----------------|---|-----------------|---|-----|----|
| A | B | False | AND | \overline{AB} | A | \overline{AB} | B | XOR | OR |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| Inputs | | Outputs | | | | | | | |
|--------|---|---------|------|----------------|--------------------|----------------|--------------------|------|------|
| A | B | NOR | XNOR | \overline{B} | $A + \overline{B}$ | \overline{A} | $\overline{A + B}$ | NAND | True |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

- The more frequently used functions have names: AND, XOR, OR, NOR, XNOR, and NAND. (Always use upper case spelling.)

Logic Gates and Their Symbols

Logic symbols for AND, OR, buffer, and NOT Boolean functions



- Note the use of the "inversion bubble."
- (Be careful about the "nose" of the gate when drawing AND vs. OR.)

A-9 Appendix A - Digital Logic

Logic symbols for NAND, NOR, XOR, and XNOR Boolean functions

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NAND

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

NOR

| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Exclusive-OR (XOR)

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Exclusive-NOR (XNOR)

Principles of Computer Architecture by M. Muroccia and V. Heuring © 1999 M. Muroccia and V. Heuring

A-10 Appendix A - Digital Logic

Variations of Basic Logic Gate Symbols

(a)

(b)

(c)

(a) 3 inputs (b) A Negated input (c) Complementary outputs

Principles of Computer Architecture by M. Muroccia and V. Heuring © 1999 M. Muroccia and V. Heuring

A-11 Appendix A - Digital Logic

The Inverter at the Transistor Level

(a)

Power Terminals

(b)

Transistor Symbol

(c)

A Transistor Used as an Inverter

(d)

Inverter Transfer Function

Principles of Computer Architecture by M. Muroccia and V. Heuring © 1999 M. Muroccia and V. Heuring

A-12 Appendix A - Digital Logic

Allowable Voltages in Transistor-Transistor-Logic (TTL)

(a)

(b)

Principles of Computer Architecture by M. Muroccia and V. Heuring © 1999 M. Muroccia and V. Heuring

A-13 Appendix A - Digital Logic

Transistor-Level Circuits For 2-Input a) NAND and b) NOR Gates

(a) NAND; (b) NOR

(a) (b)

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-14 Appendix A - Digital Logic

Tri-State Buffers

- Outputs can be 0, 1, or “electrically disconnected.”

| C | A | F |
|---|---|-------------|
| 0 | 0 | \emptyset |
| 0 | 1 | \emptyset |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$F = AC$
or
 $F = \emptyset$

Tri-state buffer

| C | A | F |
|---|---|-------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | \emptyset |
| 1 | 1 | \emptyset |

$F = A\bar{C}$
or
 $F = \emptyset$

Tri-state buffer, inverted control

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-15 Appendix A - Digital Logic

The Basic Properties of Boolean Algebra

Principle of duality: The dual of a Boolean function is gotten by replacing AND with OR and OR with AND, constant 1s by 0s, and 0s by 1s

| | Relationship | Dual | Property |
|------------|--------------------------------------|--|-----------------------|
| Postulates | $AB = BA$ | $A+B = B+A$ | Commutative |
| | $A(B+C) = AB+AC$ | $A+BC = (A+B)(A+C)$ | Distributive |
| | $1A = A$ | $0+A = A$ | Identity |
| | $A\bar{A} = 0$ | $A+\bar{A} = 1$ | Complement |
| Theorems | $0A = 0$ | $1+A = 1$ | Zero and one theorems |
| | $AA = A$ | $A+A = A$ | Idempotence |
| | $A(BC) = (AB)C$ | $A+(B+C) = (A+B)+C$ | Associative |
| | $\bar{\bar{A}} = A$ | | Involution |
| | $\overline{AB} = \bar{A} + \bar{B}$ | $\overline{A+B} = \bar{A}\bar{B}$ | DeMorgan's Theorem |
| | $AB + \bar{A}C + BC = AB + \bar{A}C$ | $(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$ | Consensus Theorem |
| | $A(A+B) = A$ | $A+\bar{A}B = A$ | Absorption Theorem |

Postulates

Theorems

A, B, etc. are Literals; 0 and 1 are constants.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-16 Appendix A - Digital Logic

DeMorgan's Theorem

| A | B | $\overline{AB} = \bar{A} + \bar{B}$ | $\overline{A+B} = \bar{A}\bar{B}$ |
|---|---|-------------------------------------|-----------------------------------|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

DeMorgan's theorem: $A+B = \overline{\bar{A}\bar{B}} = \overline{\bar{A}}\overline{\bar{B}}$

$F = A+B \equiv F = \overline{\bar{A}\bar{B}}$

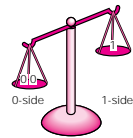
Discuss: Applying DeMorgan's theorem by “pushing the bubbles,” and “bubble tricks.”

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

The Sum-of-Products (SOP) Form

Fig. A.15—Truth Table for The Majority Function

| Minterm Index | A | B | C | F |
|---------------|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |



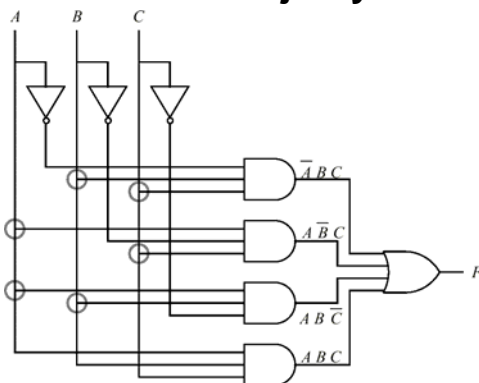
A balance tips to the left or right depending on whether there are more 0's or 1's.

- Transform the function into a two-level AND-OR equation
- Implement the function with an arrangement of logic gates from the set {AND, OR, NOT}
- M is true when A=0, B=1, and C=1, or when A=1, B=0, and C=1, and so on for the remaining cases.
- Represent logic equations by using the sum-of-products (SOP) form

The SOP Form of the Majority Gate

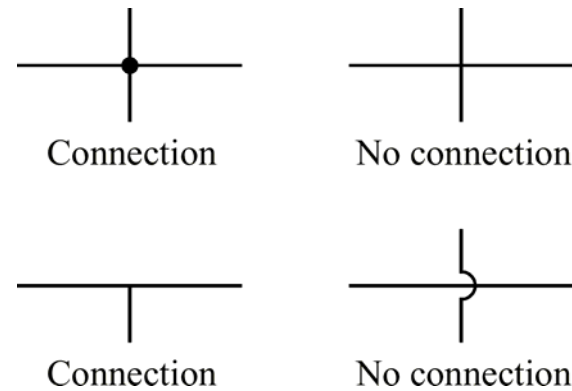
- The SOP form for the 3-input majority gate is:
- $M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC = m_3 + m_5 + m_6 + m_7 = \Sigma(3, 5, 6, 7)$
- Each of the 2^n terms are called minterms, running from 0 to $2^n - 1$
- Note the relationship between minterm number and boolean value.
- Discuss: common-sense interpretation of equation.

A 2-Level AND-OR Circuit that Implements the Majority Function

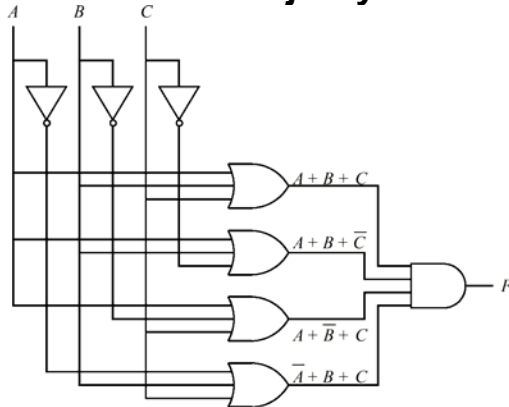


Discuss: What is the Gate Count?

Notation Used at Circuit Intersections



A 2-Level OR-AND Circuit that Implements the Majority Function



Positive vs. Negative Logic

- Positive logic: truth, or assertion is represented by logic 1, higher voltage; falsity, de- or unassertion, logic 0, is represented by lower voltage.
- Negative logic: truth, or assertion is represented by logic 0, lower voltage; falsity, de- or unassertion, logic 1, is represented by lower voltage

Gate Logic: Positive vs. Negative Logic

Normal Convention: Positive Logic/Active High
Low Voltage = 0; High Voltage = 1

Alternative Convention sometimes used: Negative Logic/Active Low

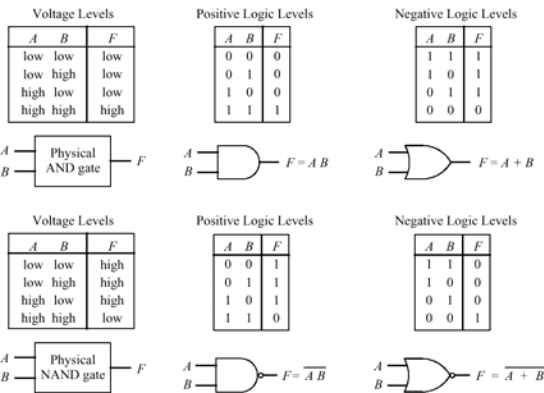
| Voltage Truth Table | | | Positive Logic | | | Negative Logic | | |
|---------------------|------|------|----------------|---|---|----------------|---|---|
| A | B | F | A | B | F | A | B | F |
| low | low | low | 0 | 0 | 0 | 1 | 1 | 1 |
| low | high | low | 0 | 1 | 0 | 1 | 0 | 1 |
| high | low | low | 1 | 0 | 0 | 0 | 1 | 1 |
| high | high | high | 1 | 1 | 1 | 0 | 0 | 0 |

Behavior in terms of Electrical Levels

Two Alternative Interpretations
Positive Logic AND
Negative Logic OR

Dual Operations

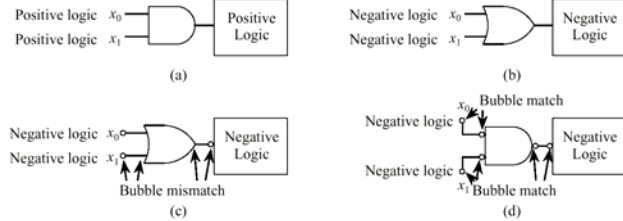
Positive and Negative Logic (Cont'd.)



Bubble Matching

- Active low signals are signified by a prime or overbar or /.
- Active high: enable _____
- Active low: enable', enable, enable/
- Discuss microwave oven control:
- Active high: Heat = DoorClosed • Start
- Active low: ? (hint: begin with AND gate as before.)

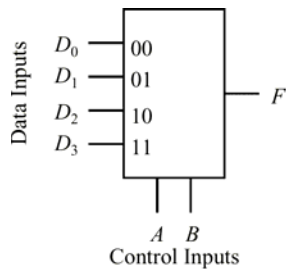
Bubble Matching (Cont'd.)



Digital Components

- High level digital circuit designs are normally made using collections of logic gates referred to as components, rather than using individual logic gates. The majority function can be viewed as a component.
- Levels of integration (numbers of gates) in an integrated circuit (IC):
 - Small scale integration (SSI): 10-100 gates.
 - Medium scale integration (MSI): 100 to 1000 gates.
 - Large scale integration (LSI): 1000-10,000 logic gates.
 - Very large scale integration (VLSI): 10,000-upward.
- These levels are approximate, but the distinctions are useful in comparing the relative complexity of circuits.
- Let us consider several useful MSI components:

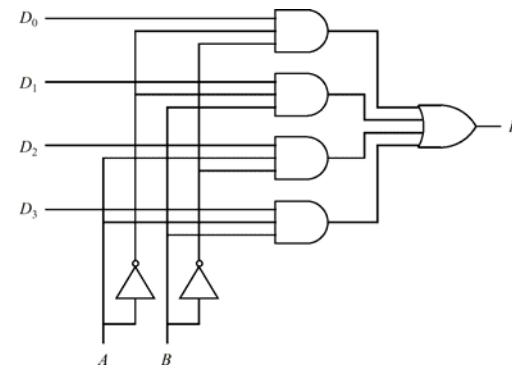
The Data Sheet



| A | B | F |
|---|---|-------|
| 0 | 0 | D_0 |
| 0 | 1 | D_1 |
| 1 | 0 | D_2 |
| 1 | 1 | D_3 |

$$F = \bar{A} \bar{B} D_0 + \bar{A} B D_1 + A \bar{B} D_2 + A B D_3$$

The Multiplexer



A-29 Appendix A - Digital Logic

Implementing the Majority Function with an 8-1 Mux

| A | B | C | M |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Principle: Use the mux select to pick out the selected minterms of the function.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-30 Appendix A - Digital Logic

More Efficiency: Using a 4-1 Mux to Implement the Majority Function

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Principle: Use the A and B inputs to select a pair of minterms. The value applied to the MUX input is selected from {0, 1, C, C} to pick the desired behavior of the minterm pair.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-31 Appendix A - Digital Logic

The Demultiplexer (DEMUX)

| D | A | B | F ₀ | F ₁ | F ₂ | F ₃ |
|---|---|---|----------------|----------------|----------------|----------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

$F_0 = D\bar{A}\bar{B}$ $F_2 = D\bar{A}B$
 $F_1 = D\bar{A}B$ $F_3 = DAB$

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-32 Appendix A - Digital Logic

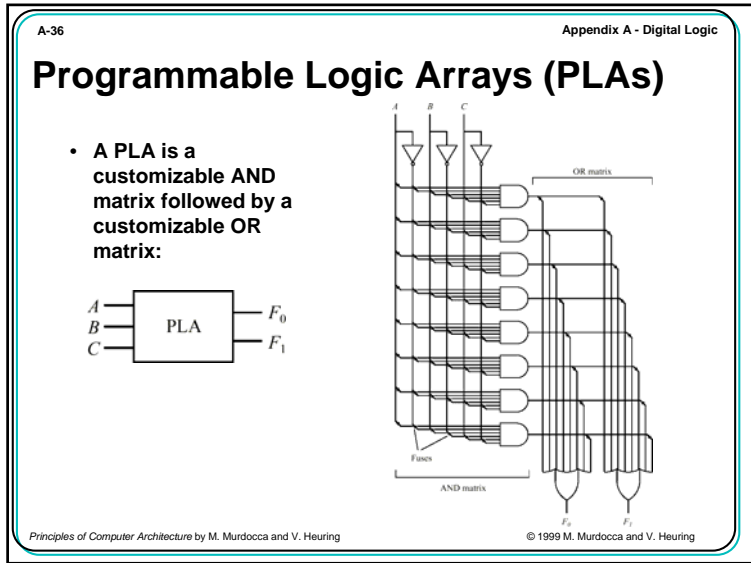
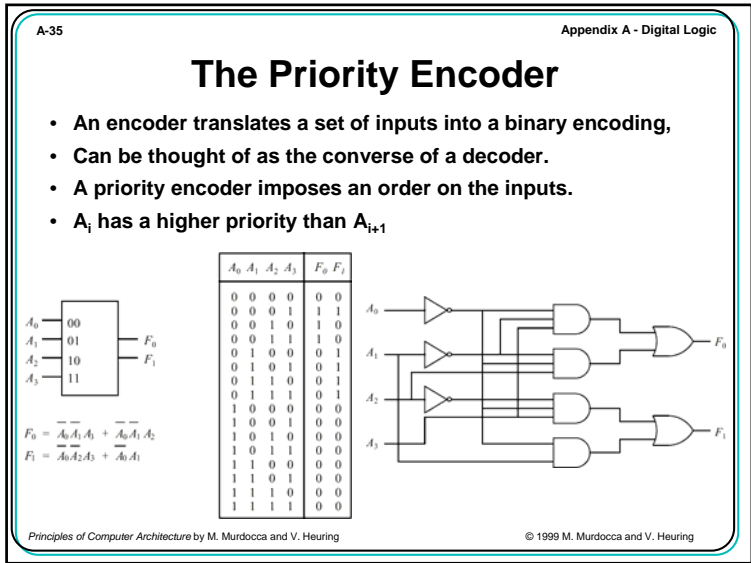
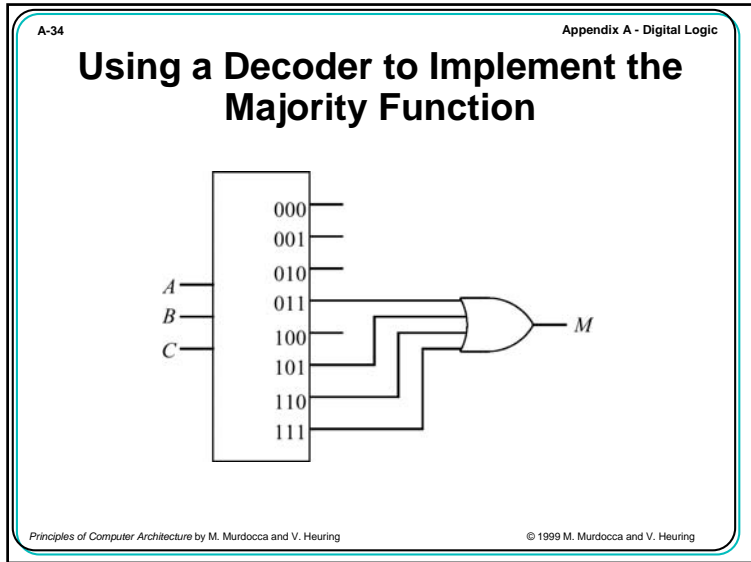
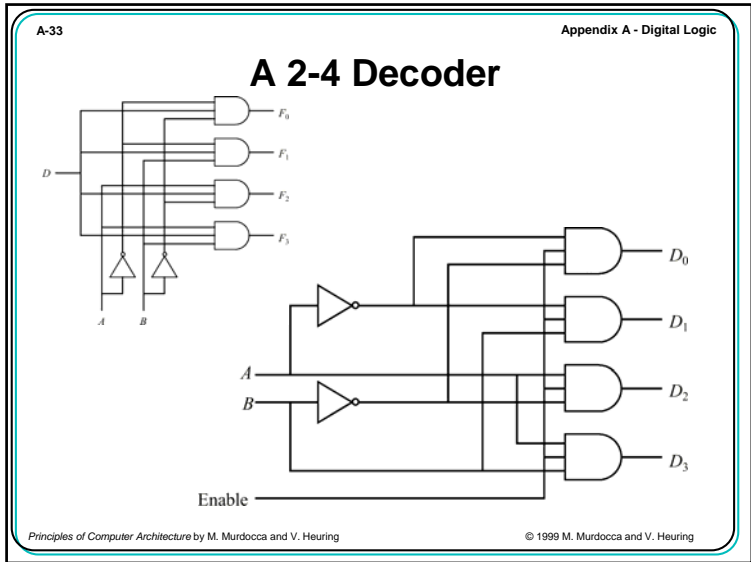
The Demultiplexer is a Decoder with an Enable Input

Compare to Fig A.29

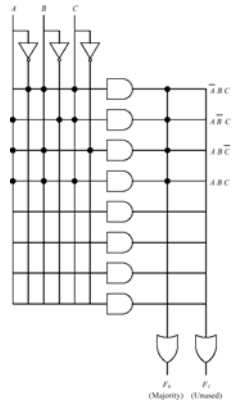
| Enable = 1 | | Enable = 0 | | | | | |
|------------|---|----------------|--|---|---|----------------|--|
| A | B | D ₀ | D ₁ D ₂ D ₃ | A | B | D ₀ | D ₁ D ₂ D ₃ |
| 0 | 0 | 1 | 0 0 0 | 0 | 0 | 0 | 0 0 0 |
| 0 | 1 | 0 | 1 0 0 | 0 | 1 | 0 | 0 0 0 |
| 1 | 0 | 0 | 0 1 0 | 1 | 0 | 0 | 0 0 0 |
| 1 | 1 | 0 | 0 0 1 | 1 | 1 | 0 | 0 0 0 |

$D_0 = \bar{A}\bar{B}$ $D_1 = \bar{A}B$ $D_2 = A\bar{B}$ $D_3 = AB$

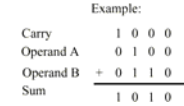
Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring



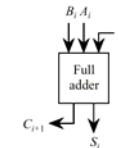
Using a PLA to Implement the Majority Function



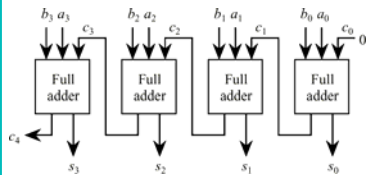
Using PLAs to Implement an Adder



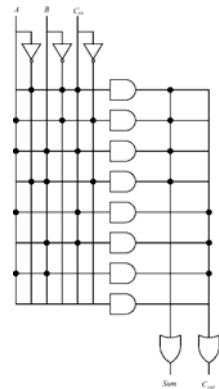
| A_i | B_i | C_i | S_i | C_{i+1} |
|-------|-------|-------|-------|-----------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



A Multi-Bit Ripple-Carry Adder



PLA Realization of a Full Adder

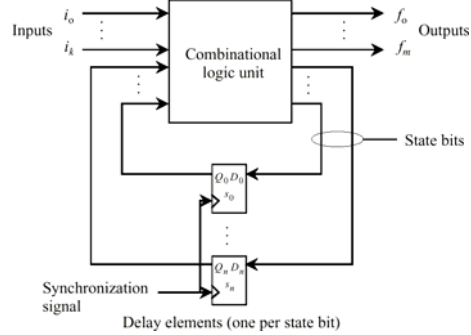


Sequential Logic

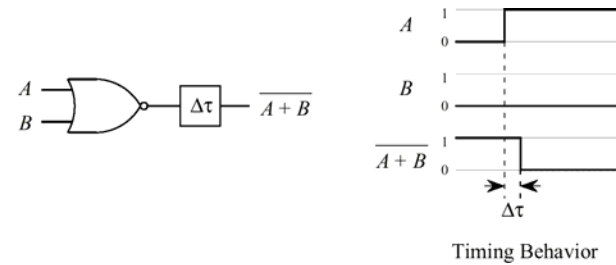
- The combinational logic circuits we have been studying so far have no memory. The outputs always follow the inputs.
- There is a need for circuits with memory, which behave differently depending upon their previous state.
- An example is a vending machine, which must remember how many and what kinds of coins have been inserted. The machine should behave according to not only the current coin inserted, but also upon how many and what kinds of coins have been inserted previously.
- These are referred to as *finite state machines*, because they can have at most a finite number of states.

Classical Model of a Finite State Machine

- An FSM is composed of a combinational logic unit and delay elements (called *flip-flops*) in a feedback path, which maintains state information.



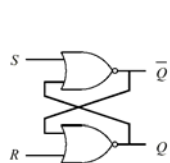
NOR Gate with Lumped Delay



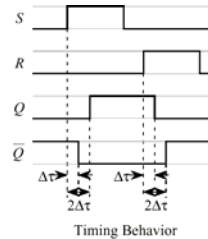
- The delay between input and output (which is lumped at the output for the purpose of analysis) is at the basis of the functioning of an important memory element, the *flip-flop*.

S-R Latch

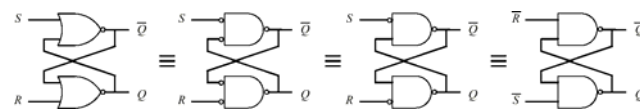
- The S-R latch is an active high (positive logic) device.



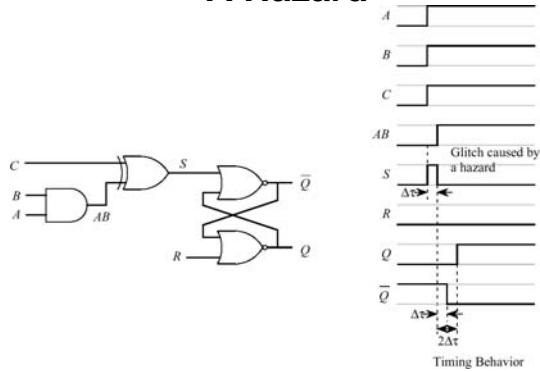
| Q_t | S_t | R_t | Q_{t+1} |
|-------|-------|-------|--------------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | (disallowed) |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | (disallowed) |



NAND Implementation of S-R Latch

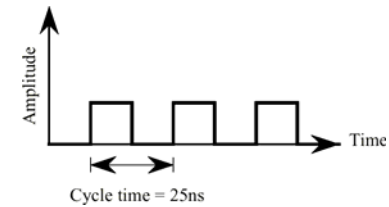


A Hazard



- It is desirable to be able to “turn off” the latch so it does not respond to such hazards.

A Clock Waveform: The Clock Paces the System



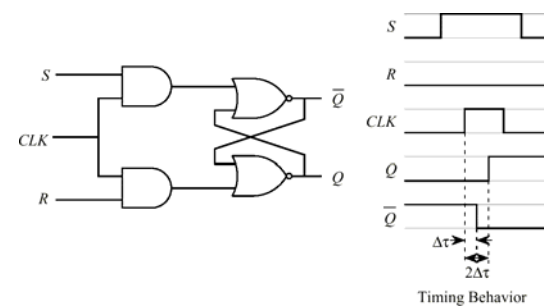
- In a positive logic system, the “action” happens when the clock is high, or positive. The low part of the clock cycle allows propagation between subcircuits, so their inputs settle at the correct value when the clock next goes high.

Scientific Prefixes

- For computer memory, $1K = 2^{10} = 1024$. For everything else, like clock speeds, $1K = 1000$, and likewise for 1M, 1G, etc.

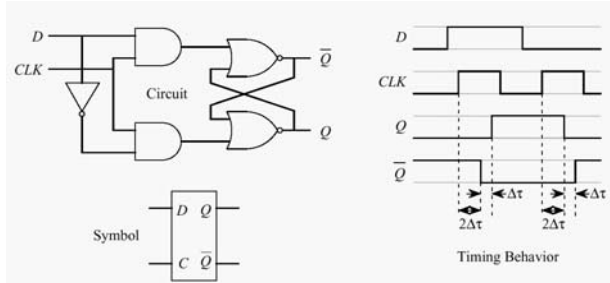
| Prefix | Abbrev. | Quantity | Prefix | Abbrev. | Quantity |
|--------|---------|------------|--------|---------|-----------|
| milli | m | 10^{-3} | Kilo | K | 10^3 |
| micro | μ | 10^{-6} | Mega | M | 10^6 |
| nano | n | 10^{-9} | Giga | G | 10^9 |
| pico | p | 10^{-12} | Tera | T | 10^{12} |
| femto | f | 10^{-15} | Peta | P | 10^{15} |
| atto | a | 10^{-18} | Exa | E | 10^{18} |

Clocked S-R Latch



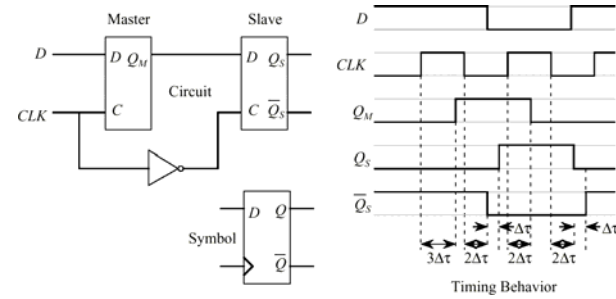
- The clock signal, CLK, enables the S and R inputs to the latch.

Clocked D Latch



- The clocked D latch, has a potential problem: If D changes while the clock is high, the output will also change. The Master-Slave flip-flop (next slide) addresses this problem.

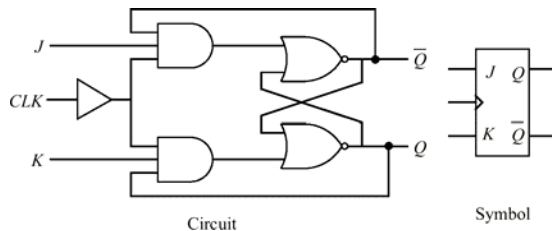
Master-Slave Flip-Flop



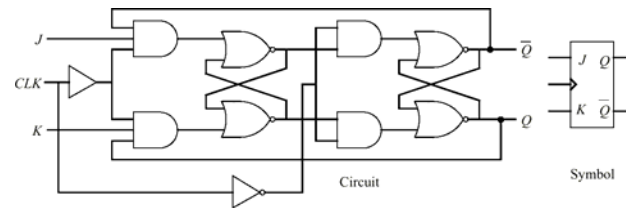
- The rising edge of the clock loads new data into the master, while the slave continues to hold previous data. The falling edge of the clock loads the new master data into the slave.

Clocked J-K Latch

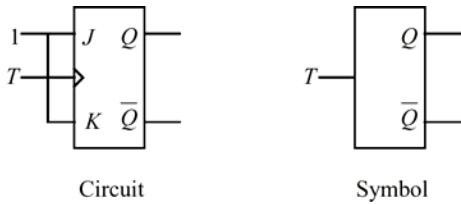
- The J-K latch eliminates the disallowed S=R=1 problem of the S-R latch, because Q enables J while Q' disables K, and vice-versa.
- However, there is still a problem. If J goes momentarily to 1 and then back to 0 while the latch is active and in the reset state, the latch will "catch" the 1. This is referred to as "1's catching."
- The J-K Master-Slave flip-flop (next slide) addresses this problem.



Master-Slave J-K Flip-Flop



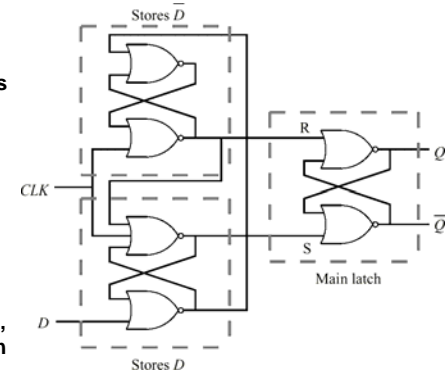
Clocked T Flip-Flop



- The presence of a constant 1 at J and K means that the flip-flop will change its state from 0 to 1 or 1 to 0 each time it is clocked by the T (Toggle) input.

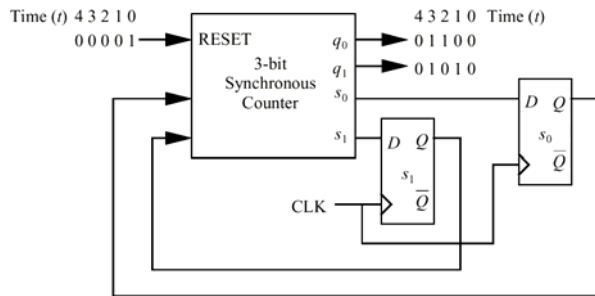
Negative Edge-Triggered D Flip-Flop

- When the clock is high, the two input latches output 0, so the Main latch remains in its previous state, regardless of changes in D.
- When the clock goes high-to-low, values in the two input latches will affect the state of the Main latch.
- While the clock is low, D cannot affect the Main latch.

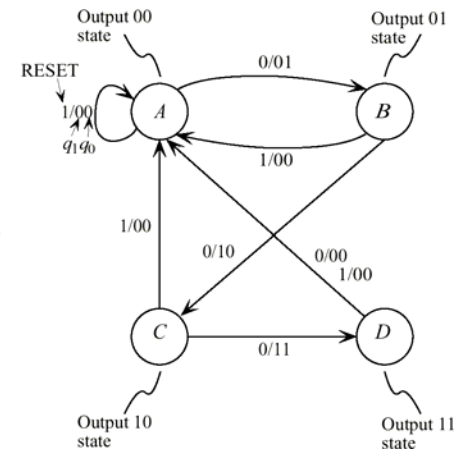


Example: Modulo-4 Counter

- Counter has a clock input (CLK) and a RESET input.
- Counter has two output lines, which take on values of 00, 01, 10, and 11 on subsequent clock cycles.



State Transition Diagram for Mod-4 Counter



State Table for Mod-4 Counter

| Present state \ Input | RESET | |
|-----------------------|-------|------|
| | 0 | 1 |
| A | B/01 | A/00 |
| B | C/10 | A/00 |
| C | D/11 | A/00 |
| D | A/00 | A/00 |

Next state
Output

State Assignment for Mod-4 Counter

| Present state (S _i) \ Input | RESET | |
|---|-------|-------|
| | 0 | 1 |
| A:00 | 01/01 | 00/00 |
| B:01 | 10/10 | 00/00 |
| C:10 | 11/11 | 00/00 |
| D:11 | 00/00 | 00/00 |

Truth Table for Mod-4 Counter

| RESET r(t) | s ₁ (t) | s ₀ (t) | s ₁ s ₀ (t+1) | q ₁ q ₀ (t+1) |
|---------------|--------------------|--------------------|-------------------------------------|-------------------------------------|
| 0 | 0 | 0 | 01 | 01 |
| 0 | 0 | 1 | 10 | 10 |
| 0 | 1 | 0 | 11 | 11 |
| 0 | 1 | 1 | 00 | 00 |
| 1 | 0 | 0 | 00 | 00 |
| 1 | 0 | 1 | 00 | 00 |
| 1 | 1 | 0 | 00 | 00 |
| 1 | 1 | 1 | 00 | 00 |

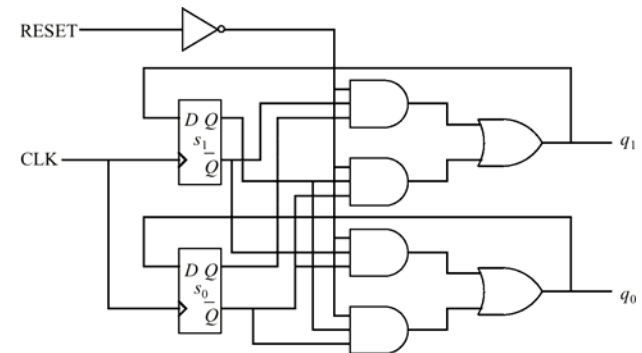
$$s_0(t+1) = \overline{r(t)}\overline{s_1(t)}\overline{s_0(t)} + \overline{r(t)}s_1(t)s_0(t)$$

$$s_1(t+1) = \overline{r(t)}\overline{s_1(t)}s_0(t) + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

$$q_0(t+1) = \overline{r(t)}\overline{s_1(t)}s_0(t) + \overline{r(t)}s_1(t)s_0(t)$$

$$q_1(t+1) = \overline{r(t)}\overline{s_1(t)}s_0(t) + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

Logic Design for Mod-4 Counter

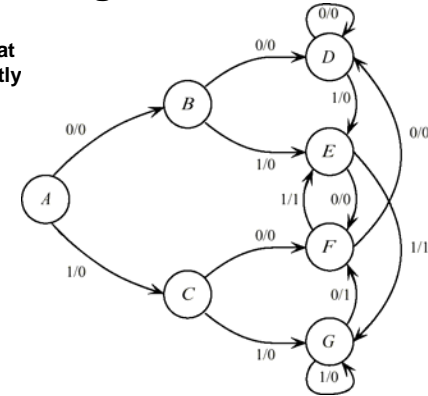


Example: A Sequence Detector

- **Example:** Design a machine that outputs a 1 when exactly two of the last three inputs are 1.
- e.g. input sequence of 011011100 produces an output sequence of 001111010.
- Assume input is a 1-bit serial line.
- Use D flip-flops and 8-to-1 Multiplexers.
- Start by constructing a state transition diagram (next slide).

Sequence Detector State Transition Diagram

- Design a machine that outputs a 1 when exactly two of the last three inputs are 1.



Sequence Detector State Table

| Present state \ Input | X | |
|-----------------------|-----|-----|
| | 0 | 1 |
| A | B/0 | C/0 |
| B | D/0 | E/0 |
| C | F/0 | G/0 |
| D | D/0 | E/0 |
| E | F/0 | G/1 |
| F | D/0 | E/1 |
| G | F/1 | G/0 |

Sequence Detector State Assignment

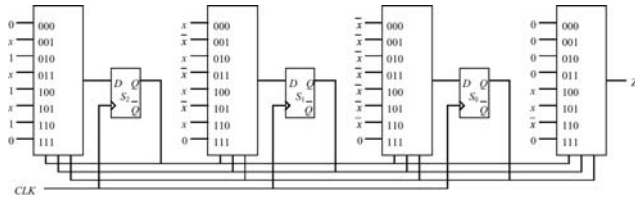
| Present state \ Input | X | |
|-----------------------|-----------------|-----------------|
| | 0 | 1 |
| $s_2 s_1 s_0$ | $s_2 s_1 s_0 Z$ | $s_2 s_1 s_0 Z$ |
| A: 000 | 001/0 | 010/0 |
| B: 001 | 011/0 | 100/0 |
| C: 010 | 101/0 | 110/0 |
| D: 011 | 011/0 | 100/0 |
| E: 100 | 101/0 | 110/1 |
| F: 101 | 011/0 | 100/1 |
| G: 110 | 101/1 | 110/0 |

(a)

| Input and state at time t | | | | Next state and output at time t+1 | | | |
|---------------------------|-------|-------|---|-----------------------------------|-------|-------|---|
| s_2 | s_1 | s_0 | X | s_2 | s_1 | s_0 | Z |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | d | d | d | d |
| 1 | 1 | 1 | 1 | d | d | d | d |

(b)

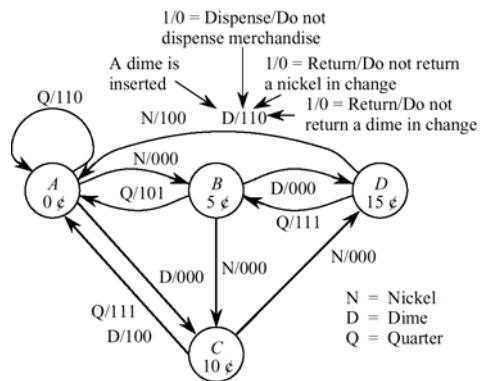
Sequence Detector Logic Diagram



Example: A Vending Machine Controller

- Example: Design a finite state machine for a vending machine controller that accepts nickels (5 cents each), dimes (10 cents each), and quarters (25 cents each). When the value of the money inserted equals or exceeds twenty cents, the machine vends the item and returns change if any, and waits for next transaction.
- Implement with PLA and D flip-flops.

Vending Machine State Transition Diagram



Vending Machine State Table and State Assignment

| Input P.S. | N 00 | D 01 | Q 10 |
|---------------|---------|---------|---------|
| A | B/000 | C/000 | A/110 |
| B | C/000 | D/000 | A/101 |
| C | D/000 | A/100 | A/111 |
| D | A/100 | A/110 | B/111 |

(a)

| Input P.S. | N x_1x_0 00 | D x_1x_0 01 | Q x_1x_0 10 |
|---------------|----------------------|---------------------|---------------------|
| s_1s_0 | $s_1s_0 / z_2z_1z_0$ | | |
| A:00 | 01/000 | 10/000 | 00/110 |
| B:01 | 10/000 | 11/000 | 00/101 |
| C:10 | 11/000 | 00/100 | 00/111 |
| D:11 | 00/100 | 00/110 | 01/111 |

(b)

A-69 Appendix A - Digital Logic

PLA Vending Machine Controller

(a)

| Present state | Next state | Output |
|--|--|-------------------------------|
| Base 10 A ₁ A ₀ | Base 10 A ₁ A ₀ | Z ₁ Z ₀ |
| 0 | 0 0 | 0 1 |
| 1 | 0 0 | 1 0 |
| 2 | 0 0 | 1 1 |
| 3 | 0 1 | d d |
| 4 | 0 1 | 1 0 |
| 5 | 0 1 | 1 1 |
| 6 | 0 1 | 0 1 |
| 7 | 0 1 | d d |
| 8 | 1 0 | 1 1 |
| 9 | 1 0 | 0 1 |
| 10 | 1 0 | 0 1 |
| 11 | 1 0 | 1 1 |
| 12 | 1 1 | 0 1 |
| 13 | 1 1 | 0 1 |
| 14 | 1 1 | 0 1 |
| 15 | 1 1 | d d |

(b)

(c)

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-70 Appendix A - Digital Logic

Moore Counter

- Mealy Model: Outputs are functions of Inputs and Present State.
- Previous FSM designs were Mealy Machines, in which next state was computed from present state and inputs.
- Moore Model: Outputs are functions of Present State only.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-71 Appendix A - Digital Logic

Four-Bit Register

- Makes use of tri-state buffers so that multiple registers can gang their outputs to common output lines.

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

A-72 Appendix A - Digital Logic

Left-Right Shift Register with Parallel Read and Write

| Control | Function |
|---------|---------------|
| 0 0 | No change |
| 0 1 | Shift left |
| 1 0 | Shift right |
| 1 1 | Parallel load |

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring

Modulo-8 Counter

- Note the use of the T flip-flops, implemented as J-K's. They are used to toggle the input of the next flip-flop when its output is 1.

