William Stallings

Computer Organization and Architecture

10th Edition

Chapter 11

Digital Logic

+

Boolean Algebra

- Mathematical discipline used to design and analyze the behavior of the digital circuitry in digital computers and other digital systems
- Named after George Boole
  - English mathematician
  - Proposed basic principles of the algebra in 1854
- Claude Shannon suggested Boolean algebra could be used to solve problems in relay-switching circuit design
- Is a convenient tool:
  - Analysis
    - It is an economical way of describing the function of digital circuitry
  - Design
    - Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function

+

Boolean Variables and Operations

- Makes use of variables and operations
  - Are logical
  - A variable may take on the value 1 (TRUE) or 0 (FALSE)
  - Basic logical operations are AND, OR, and NOT
- AND
  - Yields true (binary value 1) if and only if both of its operands are true
  - In the absence of parentheses the AND operation takes precedence over the OR operation
  - When no ambiguity will occur the AND operation is represented by simple concatenation instead of the dot operator
- OR
  - Yields true if either or both of its operands are true
- NOT
  - Inverts the value of its operand

Table 11.1 Boolean Operators

(a) Boolean Operators of Two Input Variables

P	Q	NOT P ($\overline{P}$)	P AND Q ($P \bullet Q$)	P OR Q ($P + Q$)	P NAND Q ($\overline{P \bullet Q}$)	P NOR Q ($\overline{P + Q}$)	P XOR Q ($P \oplus Q$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

(b) Boolean Operators Extended to More than Two Inputs (A, B, . . .)

Operation	Expression	Output = 1 if
AND	$A \bullet B \bullet \ldots$	All of the set {A, B, …} are 1.
OR	$A + B + \ldots$	Any of the set {A, B, …} are 1.
NAND	$\overline{A \cdot B \cdot \ldots}$	Any of the set {A, B, …} are 0.
NOR	$\overline{A + B + \ldots}$	All of the set {A, B, …} are 0.
XOR	$A \oplus B \oplus \ldots$	The set {A, B, …} contains an odd number of ones.

Table 11.2
Basic Identities of Boolean Algebra

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

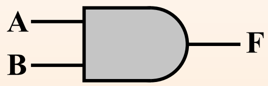
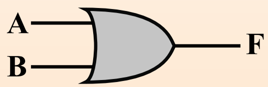
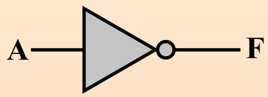
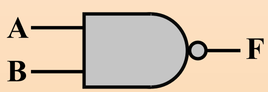
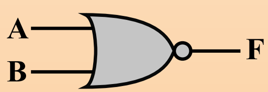
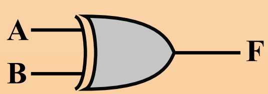
Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B F 0 0 0 0 1 0 1 0 0 1 1 1
OR		$F = A + B$	A B F 0 0 0 0 1 1 1 0 1 1 1 1
NOT		$F = \overline{A}$ or $F = A'$	A F 0 1 1 0
NAND		$F = \overline{AB}$	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$F = \overline{A + B}$	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR		$F = A \oplus B$	A B F 0 0 0 0 1 1 1 0 1 1 1 0

Figure 11.1 Basic Logic Gates

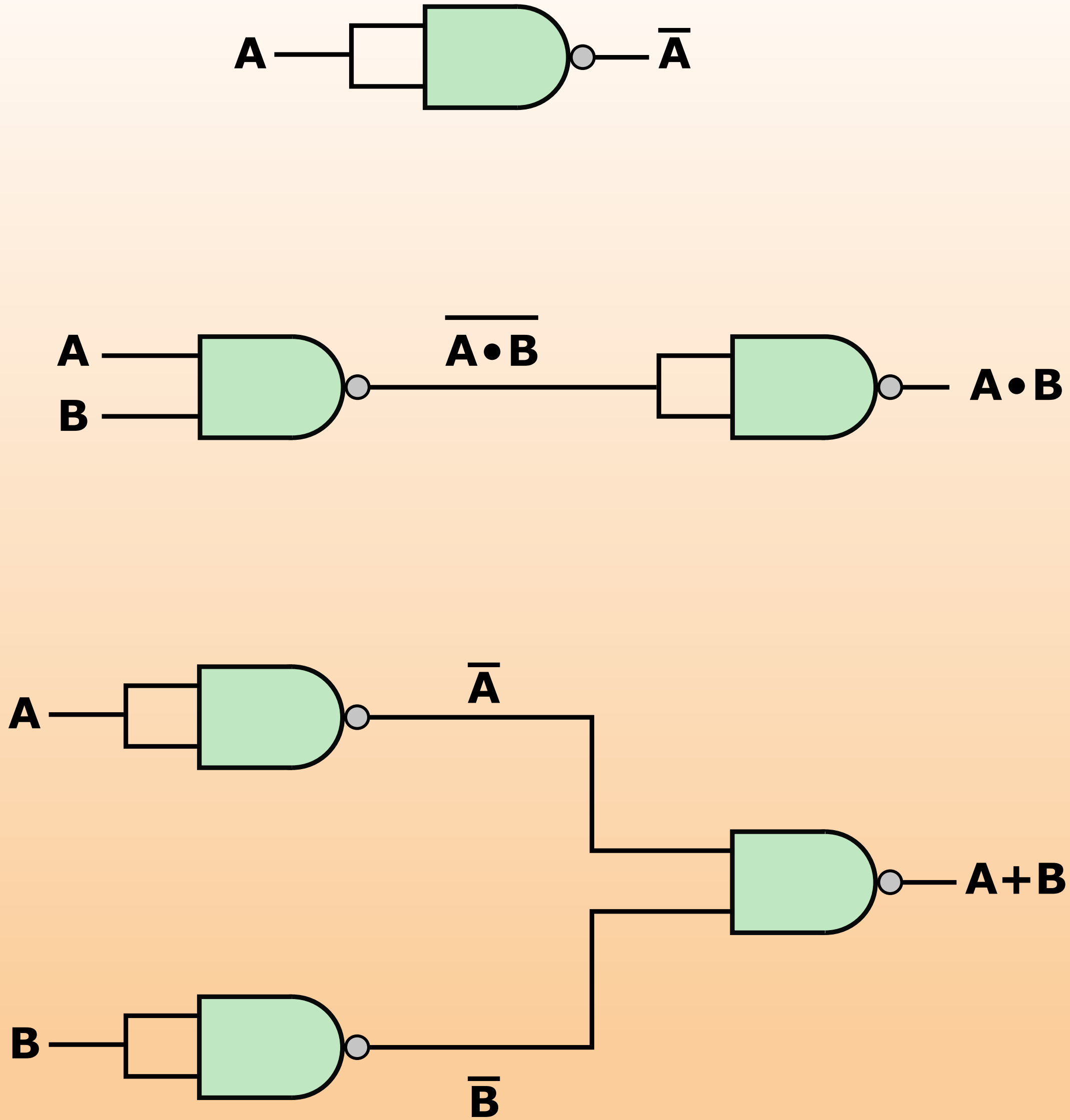


Figure 11.2 Some Uses of NAND Gates

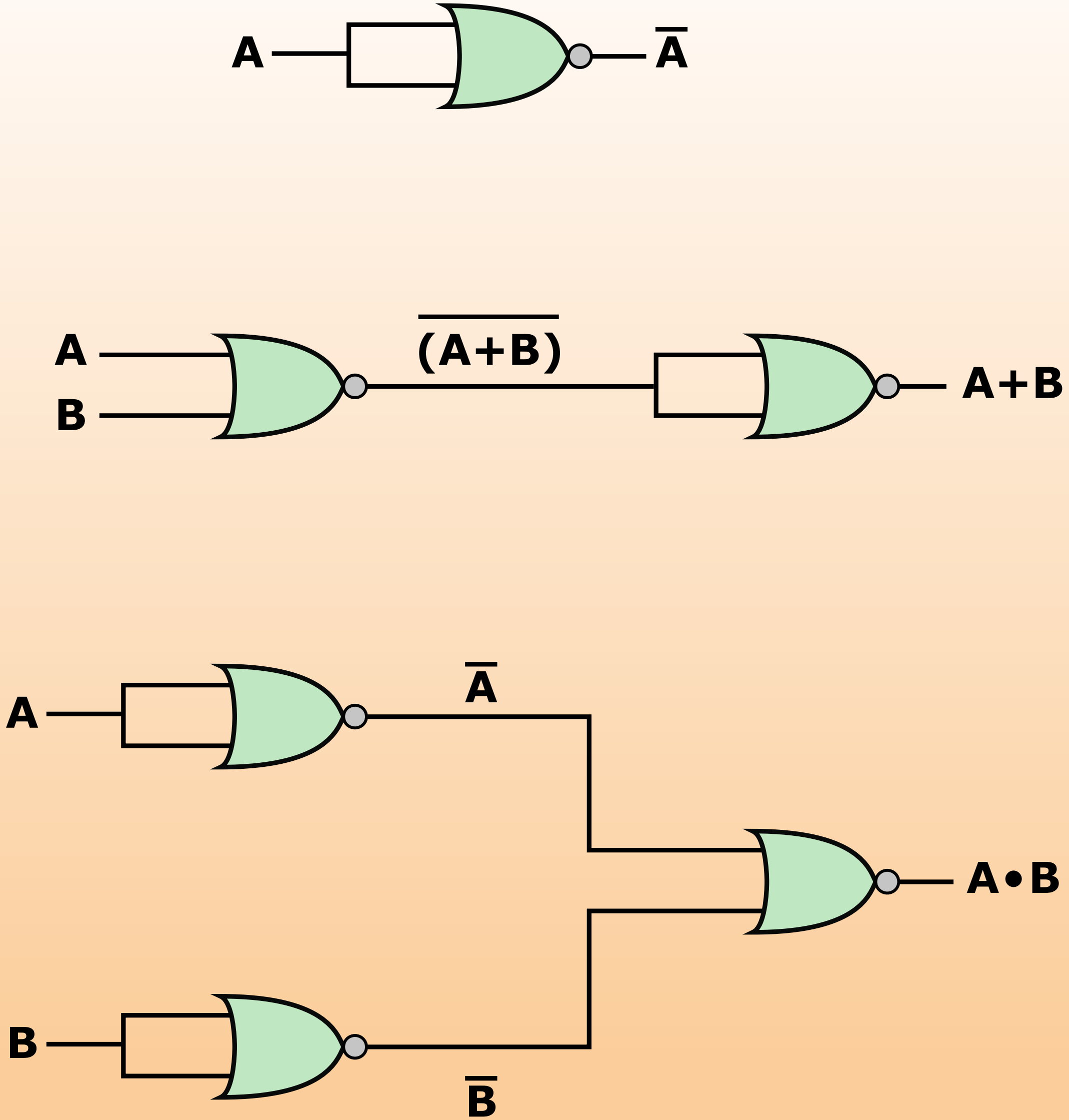
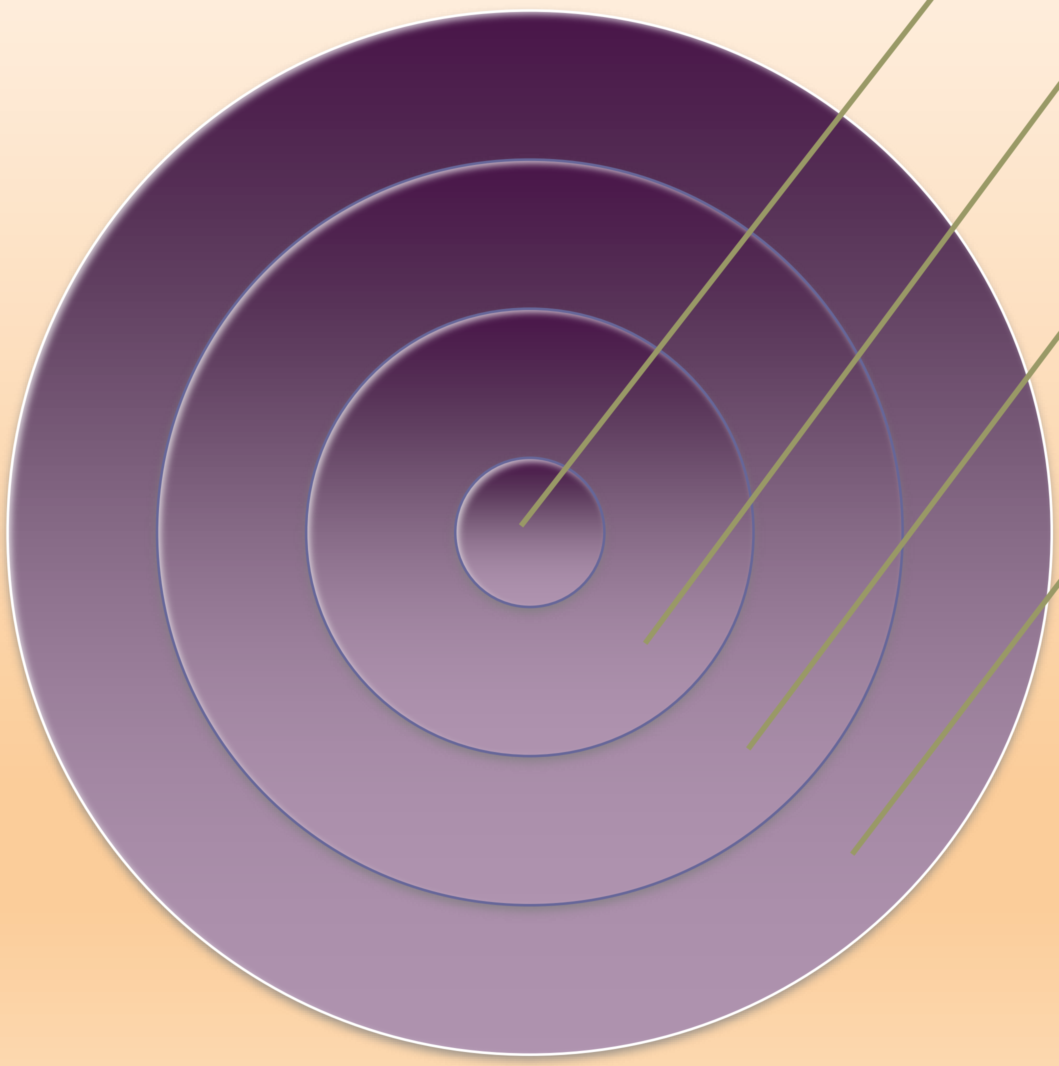


Figure 11.3 Some Uses of NOR Gates

Combinational Circuit



An interconnected set of gates whose output at any time is a function only of the input at that time

The appearance of the input is followed almost immediately by the appearance of the output, with only gate delays

Consists of *n* binary inputs and *m* binary outputs

Can be defined in three ways:

- Truth table
  - For each of the 2^n possible combinations of input signals, the binary value of each of the *m* output signals is listed
- Graphical symbols
  - The interconnected layout of gates is depicted
- Boolean equations
  - Each output signal is expressed as a Boolean function of its input signals

Table 11.3
A Boolean Function of Three Variables

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	0
1	1	0	1
1	1	1	0

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

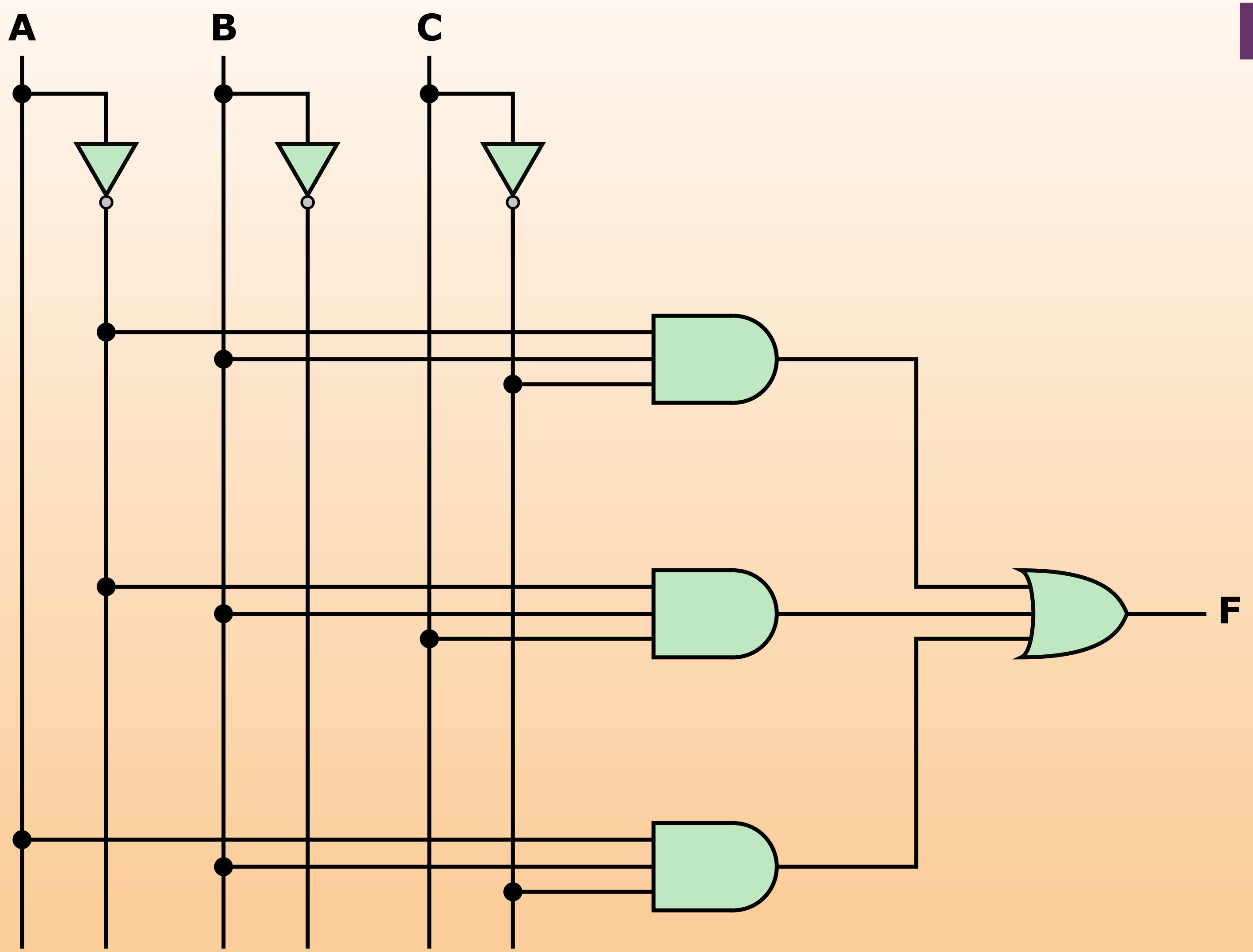


Figure 11.4 Sum-of-Products Implementation of Table 11.3

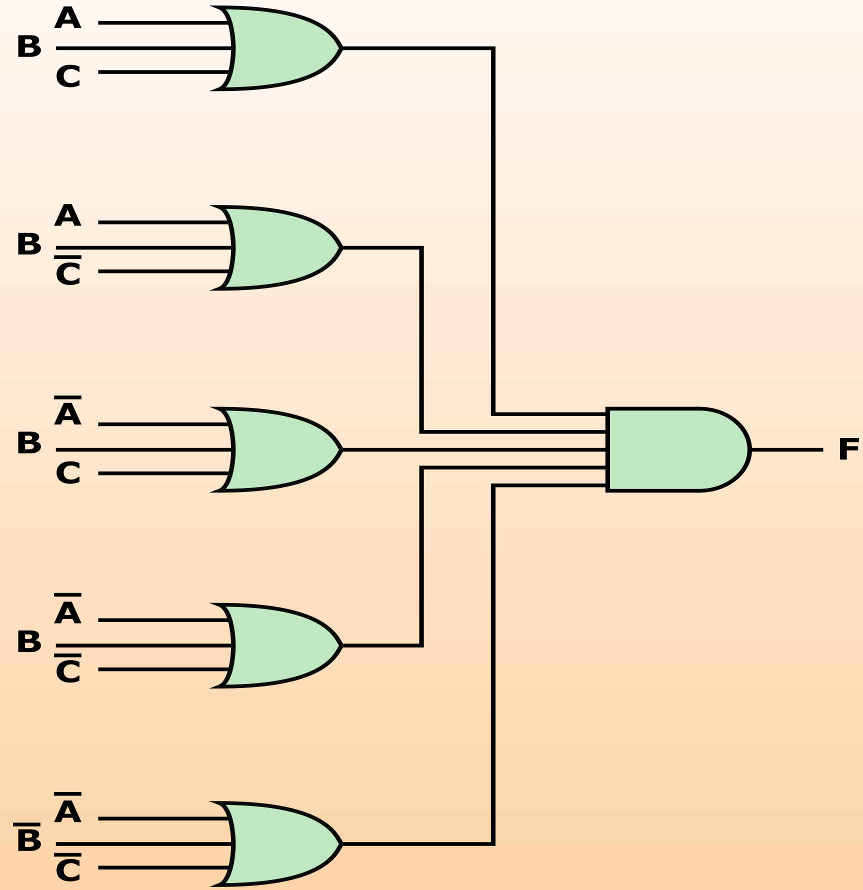


Figure 11.5 Product-of-Sums Implementation of Table 11.3

$$\begin{aligned} F &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \end{aligned} \quad (11.2)$$

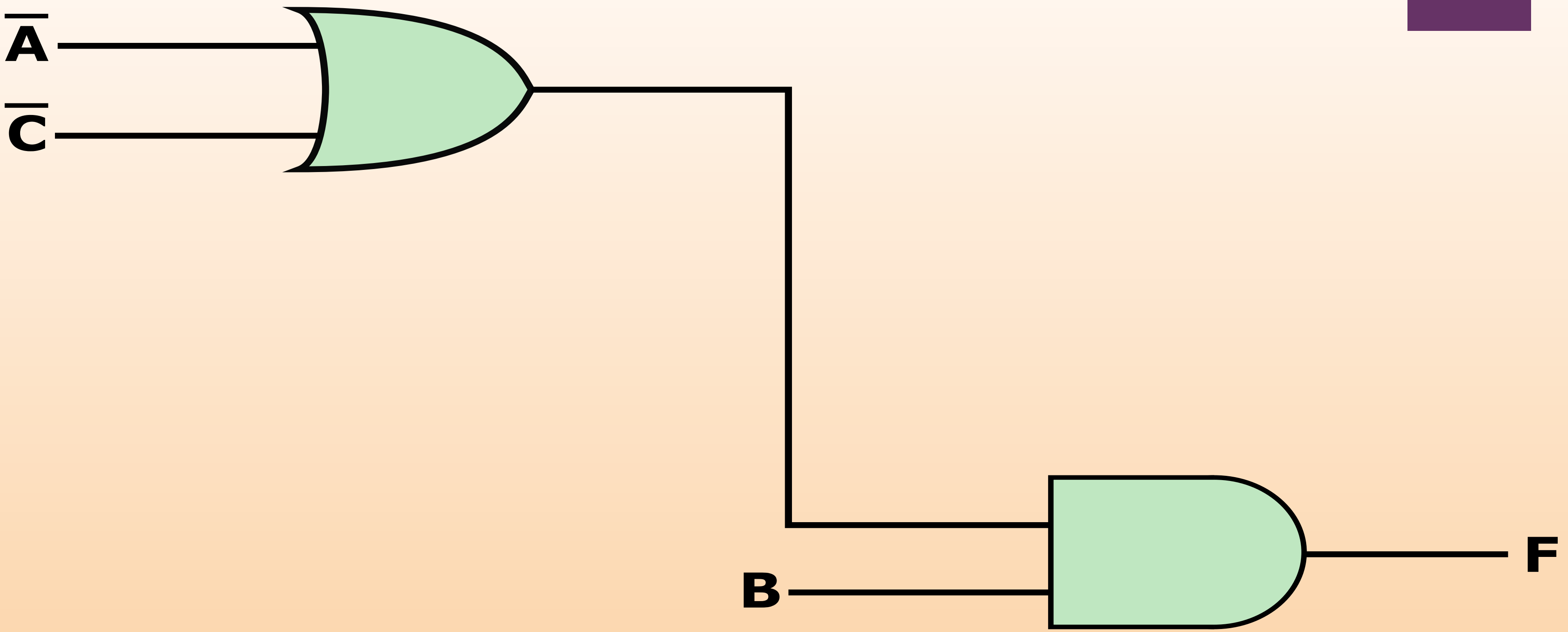


Figure 11.6 Simplified Implementation of Table 11.3

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

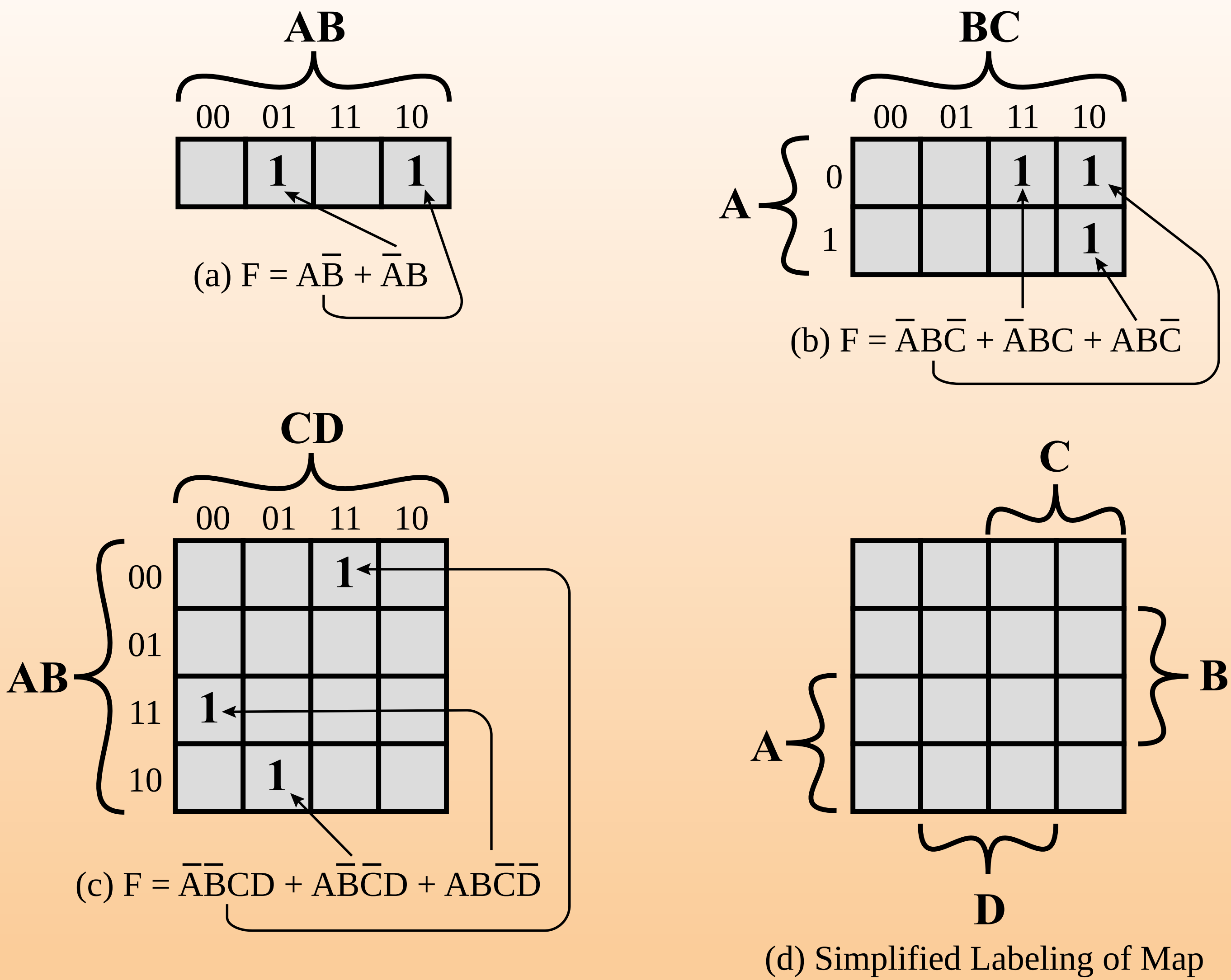


Figure 11.7 The Use of Karnaugh Maps to Represent Boolean Functions

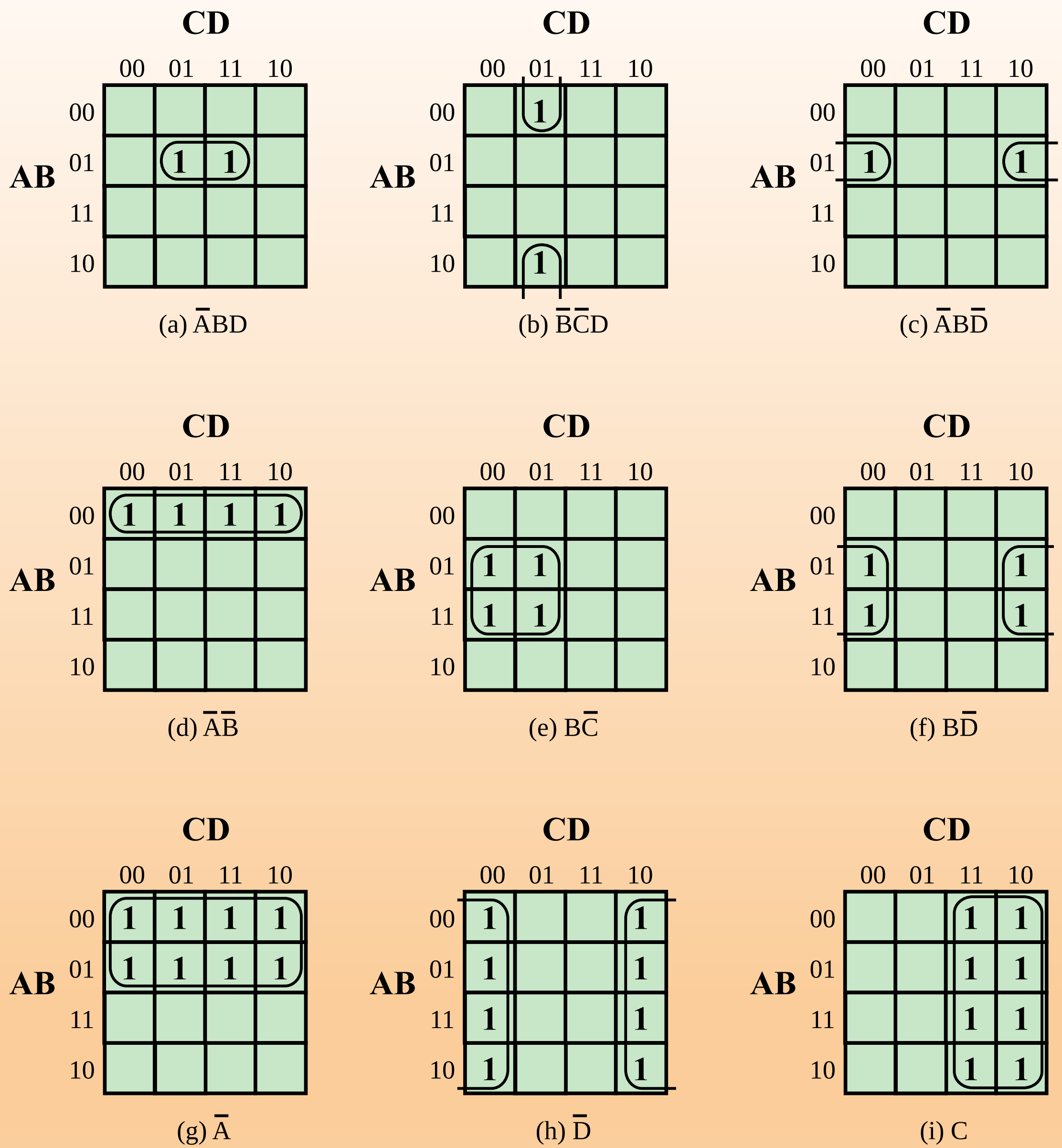
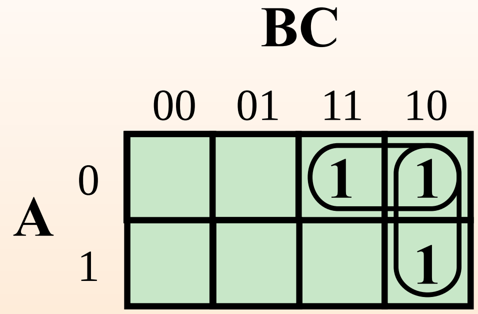
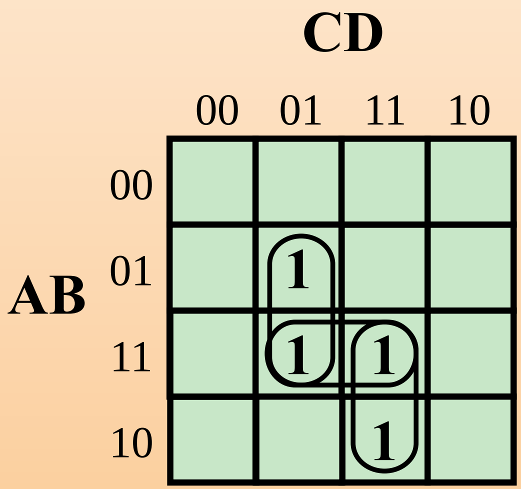


Figure 11.8 Example Use of Karnaugh Maps



(a) $F = \overline{A}B + B\overline{C}$



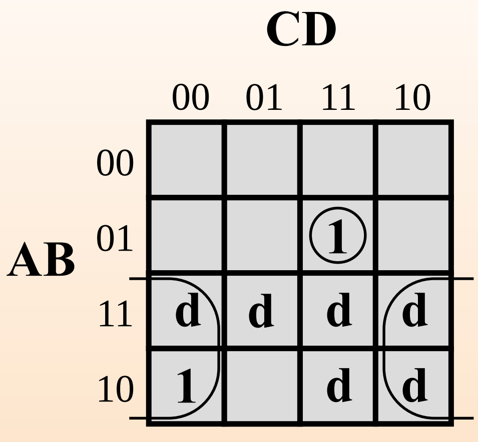
(b) $F = B\overline{C}D + ACD$

Figure 11.9 Overlapping Groups

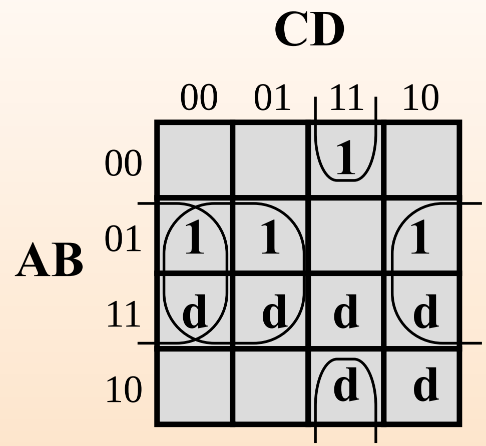
Table 11.4
Truth Table for the One-Digit Packed Decimal Incrementer

	Input					Output			
Number	A	B	C	D	Number	W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Don't care condition	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d

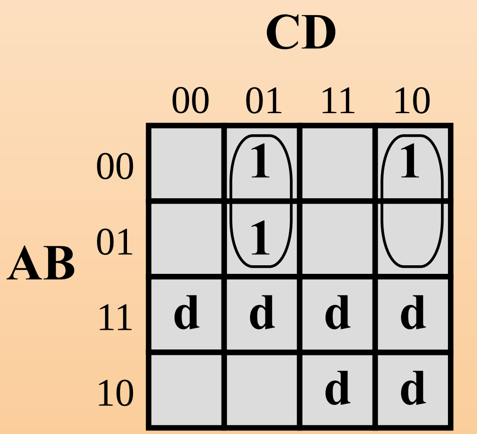
Table 11.4 Truth Table for the One-Digit Packed Decimal Incrementer



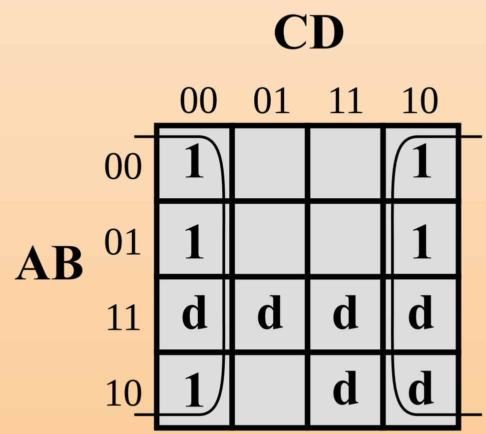
(a) $W = A\overline{D} + \overline{A}BCD$



(b) $X = B\overline{D} + B\overline{C} + BCD$



(c) $Y = \overline{A}\overline{C}D + \overline{A}C\overline{D}$



(d) $Z = \overline{D}$

Figure 11.10 Karnaugh Maps for the Incrementer

Table 11.5
First Stage of Quine-McCluskey Method

(for F = ABCD + AB D + AB + A CD + BCD + BC + B D + D)

Product Term	Index	A	B	C	D	
$\overline{A}\,\overline{B}\,\overline{C}D$	1	0	0	0	1	✓
$\overline{A}B\overline{C}D$	5	0	1	0	1	✓
$\overline{A}BC\overline{D}$	6	0	1	1	0	✓
$AB\overline{C}\,\overline{D}$	12	1	1	0	0	✓
$\overline{A}BCD$	7	0	1	1	1	✓
$A\overline{B}CD$	11	1	0	1	1	✓
$AB\overline{C}D$	13	1	1	0	1	✓
$ABCD$	15	1	1	1	1	✓

Table 11.6
Last Stage of Quine-McCluskey Method

(for F = ABCD + AB D + AB + A CD + BCD + BC + B D + D)

	ABCD	$AB\overline{C}D$	$AB\overline{C}\,\overline{D}$	$A\overline{B}CD$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$	$\overline{A}B\overline{C}D$	[illegible]D
BD	X	X			X		X	
$\overline{A}\,\overline{C}D$							$\boxed{X}$	Ä
$\overline{A}BC$					$\boxed{X}$	Ä		
$AB\overline{C}$		$\boxed{X}$	Ä					
ACD	$\boxed{X}$			Ä				

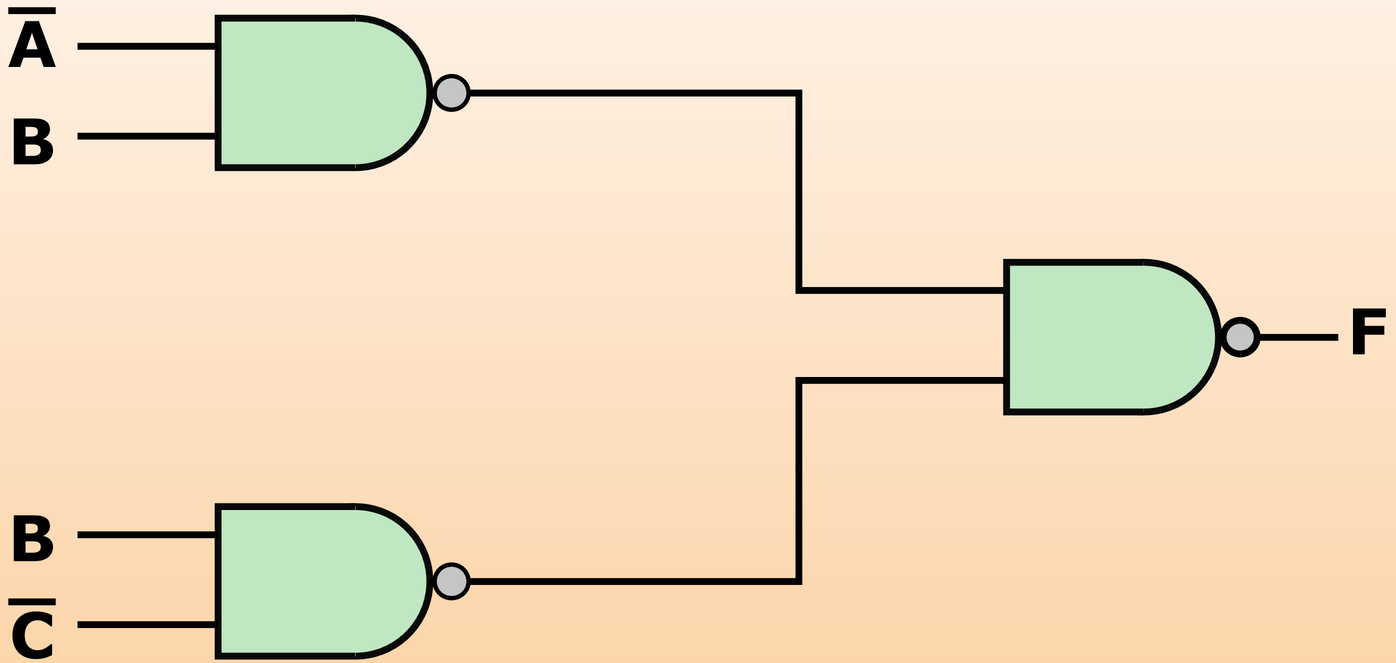


Figure 11.11 NAND Implementation of Table 11.3

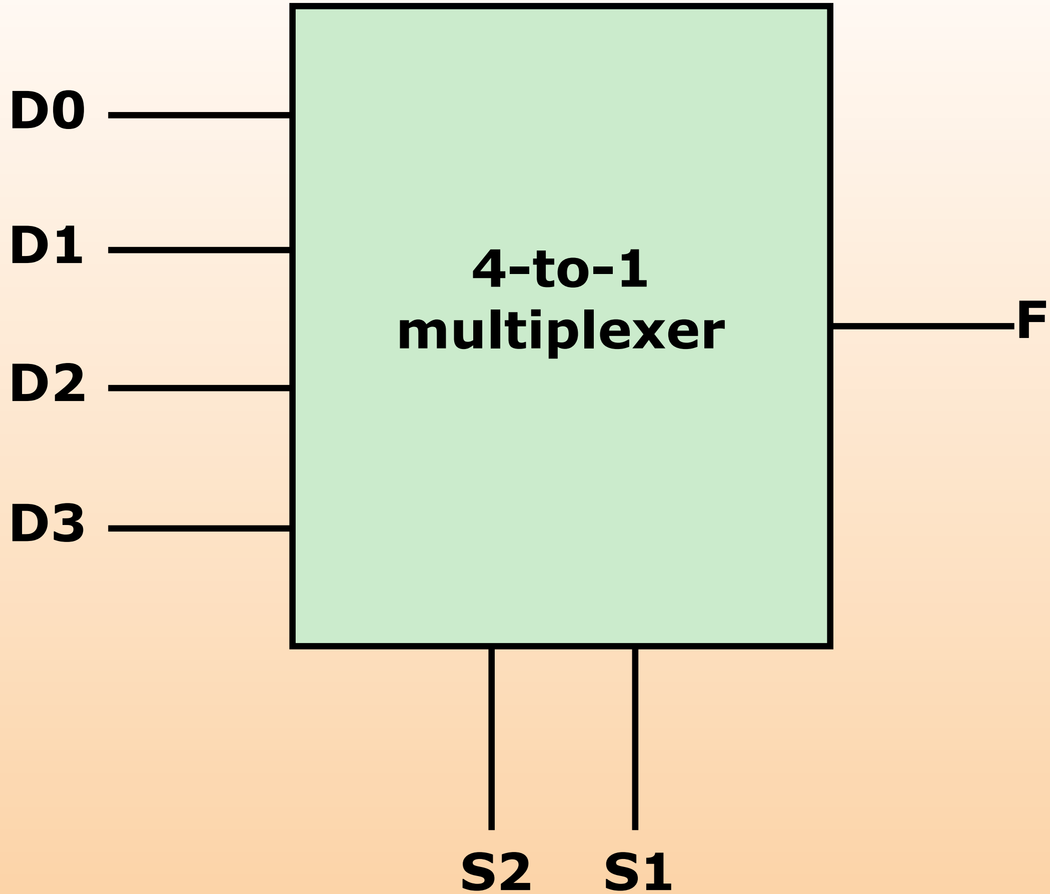


Figure 11.12 4-to-1 Multiplexer Representation

Table 11.7
4-to-1 Multiplexer Truth Table

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

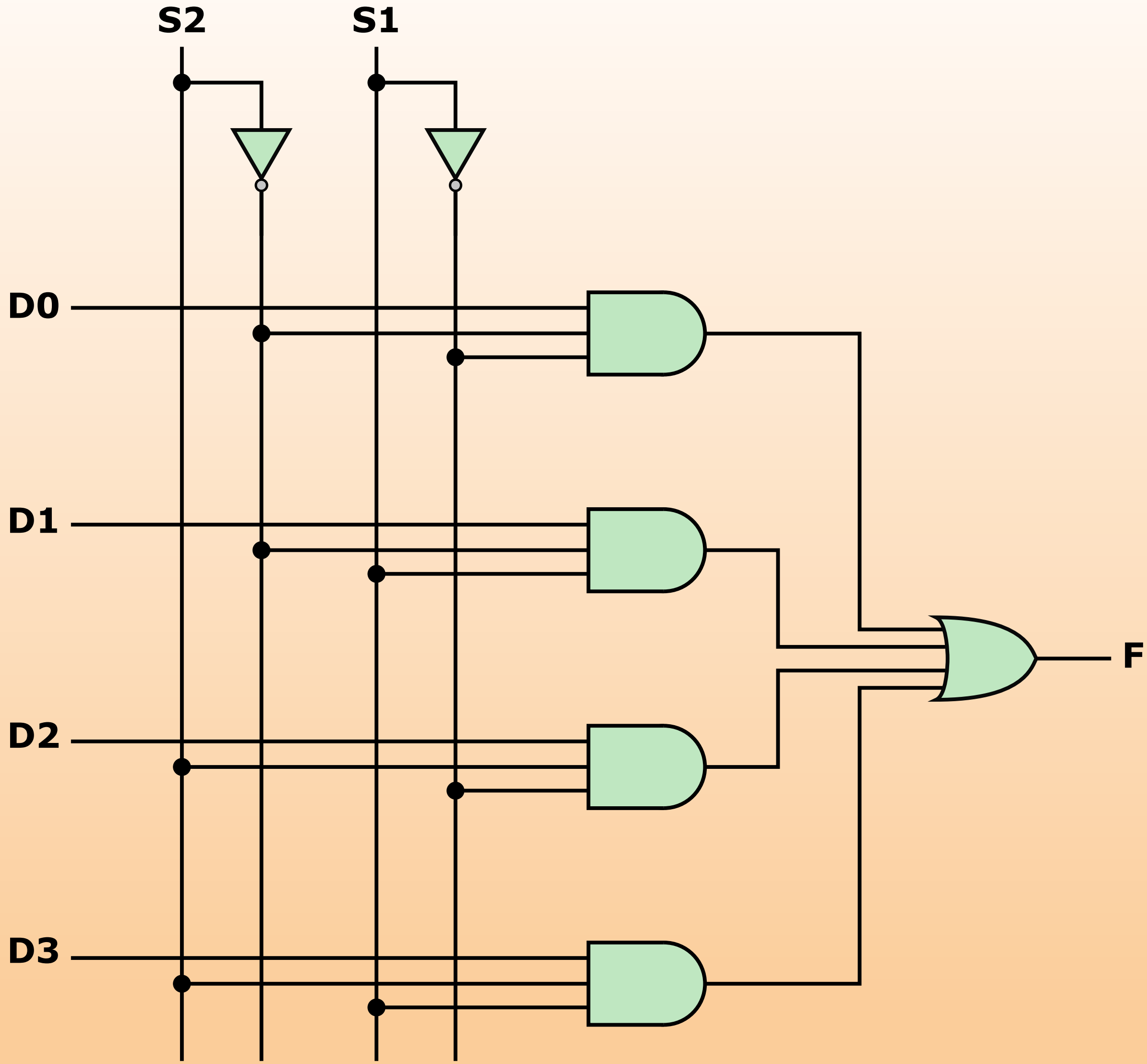


Figure 11.13 Multiplexer Implementation

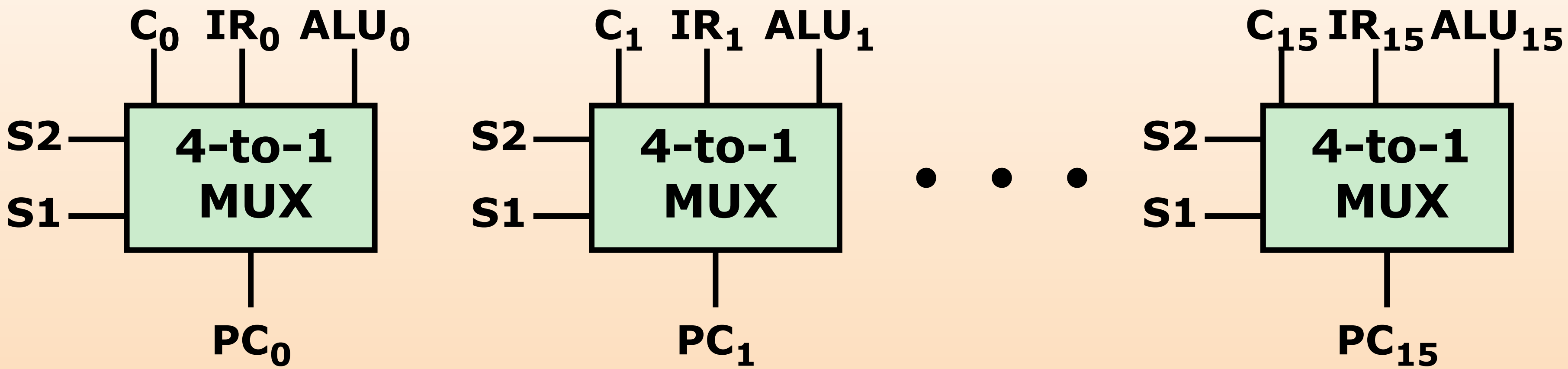


Figure 11.14 Multiplexer Input to Program Counter

Multiplexers are used in digital circuits to control signal and data routing. An example is the loading of the program **counter** (PC). The value to be loaded into the program counter may come from one of several different sources:

* A binary counter, if the PC is to be incremented for the next instruction

*The instruction **register,** if a branch instruction using a direct address has just been executed

*The output of the ALU, if the branch instruction specifies the address using a displacement mode

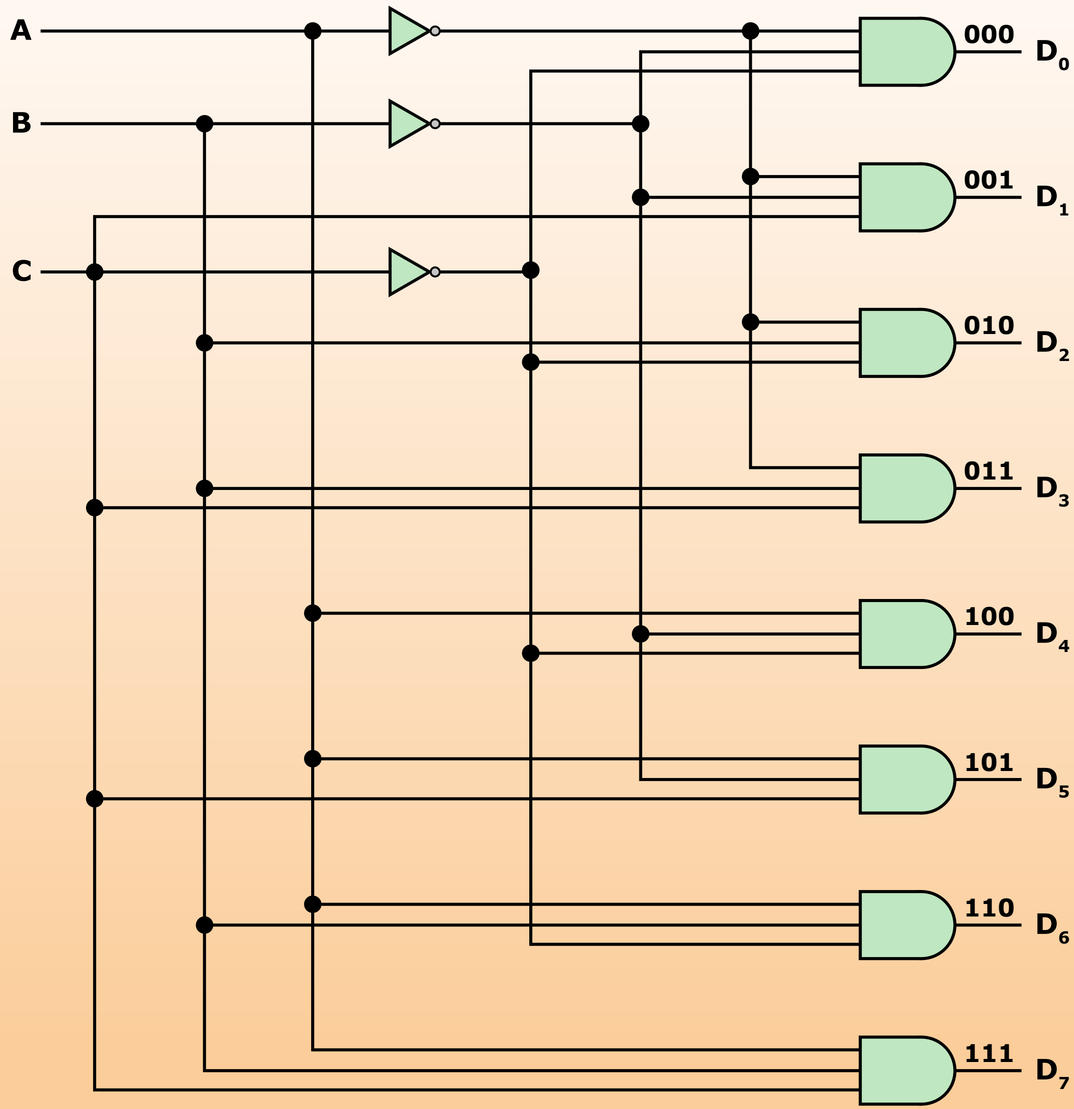


Figure 11.15 Decoder with 3 Inputs and $2^3 = 8$ Outputs

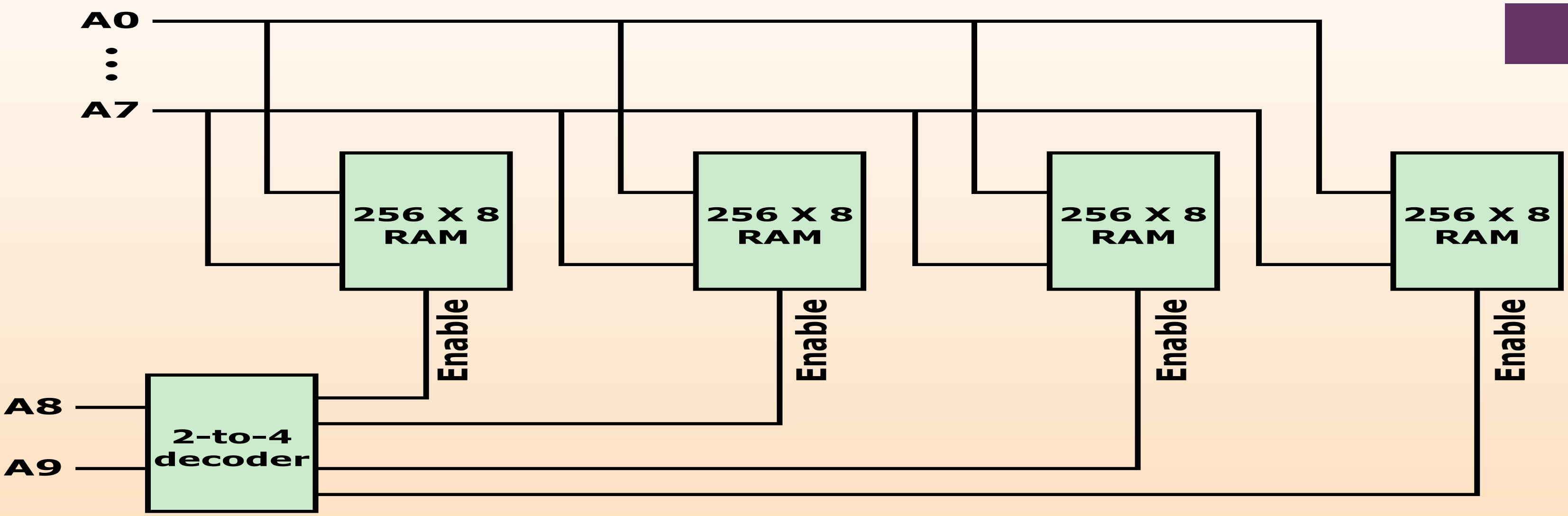


Figure 11.16 Address Decoding

Decoders find many uses in digital computers. One example is address decoding. Suppose we wish to construct a 1K-byte memory using four 256 * 8-bit RAM chips. We want a single unified address space, which can be broken down as follows:

Address	Chip
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Each chip requires 8 address lines, and these are supplied by the lower-order 8 bits of the address. The higher-order 2 bits of the 10-bit address are used to select one of the four RAM chips. For this purpose, a 2-to-4 decoder is used whose output enables one of the four chips, as shown in Figure 11.16.

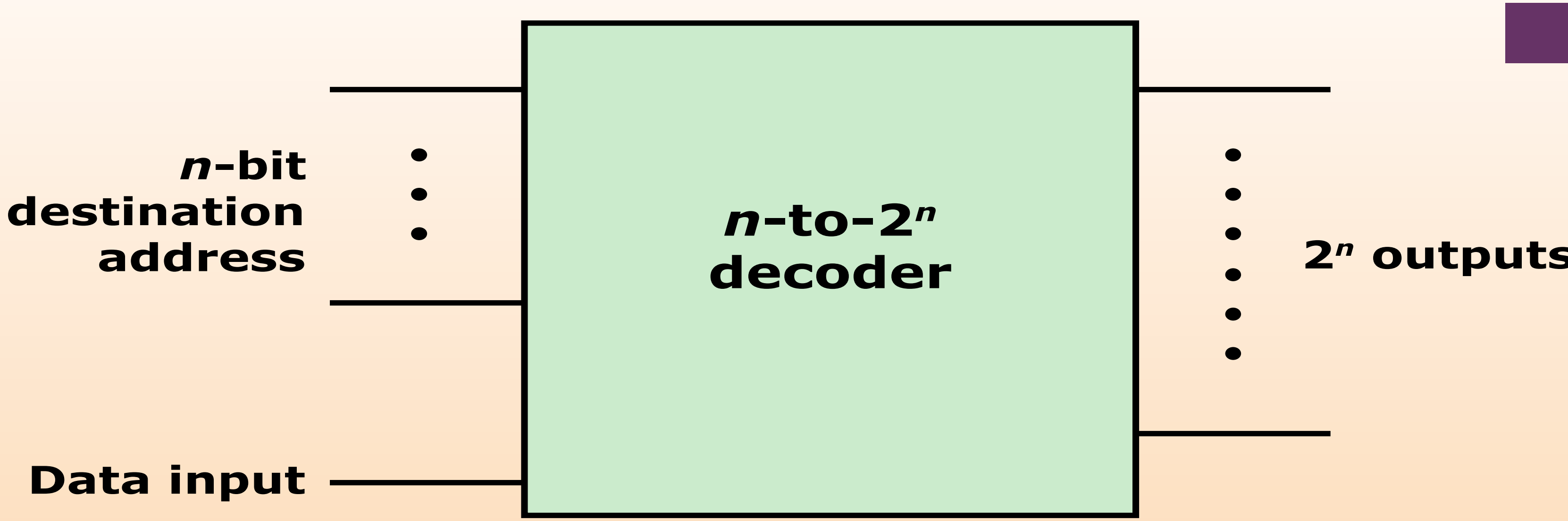


Figure 11.17 Implementation of a Demultiplexer Using a Decoder

With an additional input line, a decoder can be used as a demultiplexer. The demultiplexer performs the inverse function of a multiplexer; it connects a single input to one of several outputs. This is shown in Figure 11.17. As before, *n* inputs are decoded to produce a single one of 2^n outputs. All of the 2^n output lines are ANDed with a data input line. Thus, the *n* inputs act as an address to select a particular out- put line, and the value on the data input line (0 or 1) is routed to that output line.

+
Read-Only Memory (ROM)

- Memory that is implemented with combinational circuits
  - Combinational circuits are often referred to as "memoryless" circuits because their output depends only on their current input and no history of prior inputs is retained
- Memory unit that performs only the read operation
  - Binary information stored in a ROM is permanent and is created during the fabrication process
  - A given input to the ROM (address lines) always produces the same output (data lines)
  - Because the outputs are a function only of the present inputs, ROM is a combinational circuit

Table 11.8
Truth Table for a ROM

Input				Output			
X_1	X_2	X_3	X_4	Z_1	Z_2	Z_3	Z_4
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	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

A ROM can be implemented with a decoder and a set of OR gates. As an example, consider Table 11.8. This can be viewed as a truth table with four inputs and four outputs. It can also be viewed as defining the contents of a 64-bit ROM consisting of 16 words of 4 bits each. The four inputs specify an address, and the four outputs specify the contents of the location specified by the address.

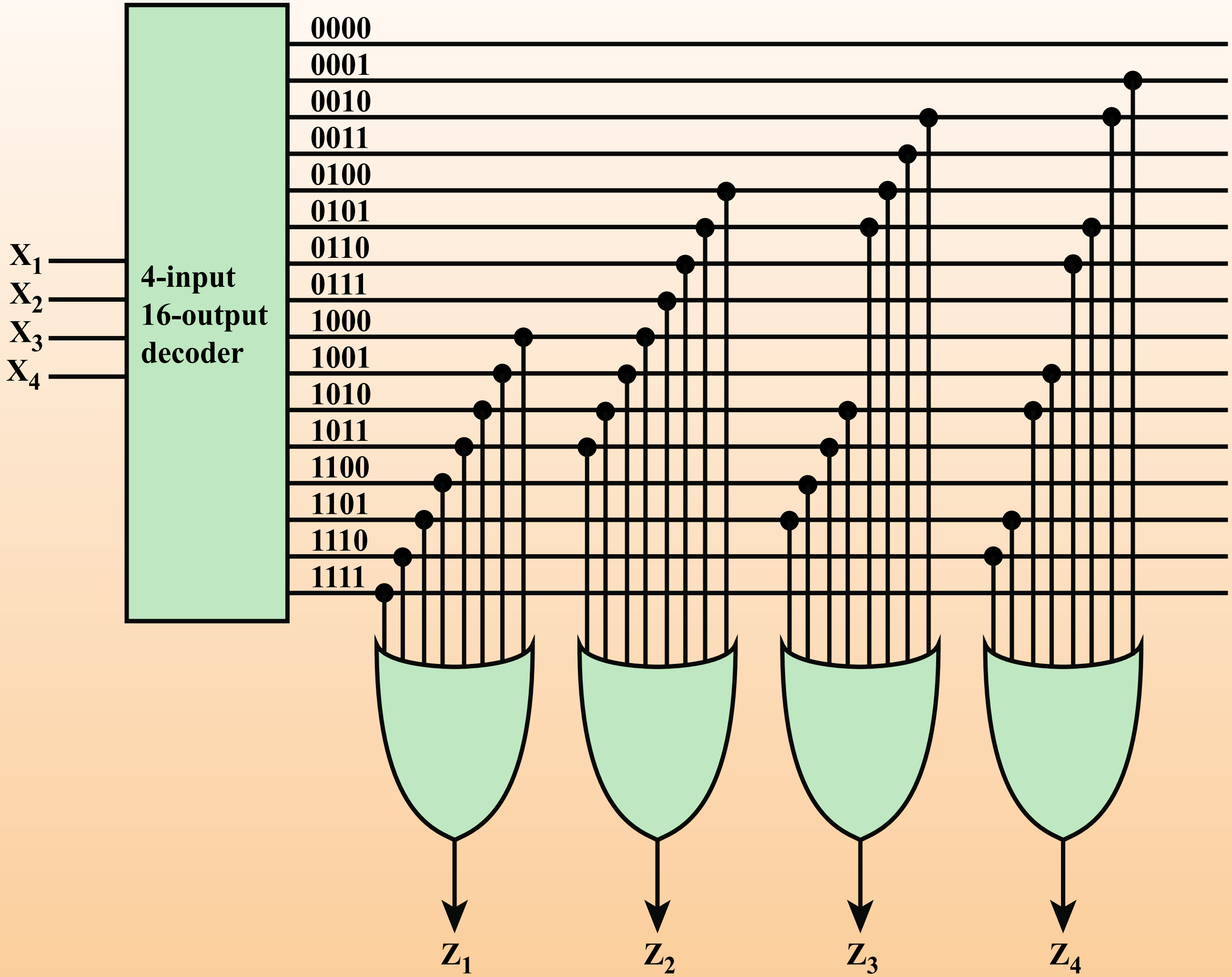


Figure 11.18 shows how this memory could be implemented using a 4-to-16 decoder and four OR gates. As with the PLA, a regular organization is used, and the interconnections are made to reflect the desired result.

Figure 11.18 A 64-Bit ROM

Table 11.9
Binary Addition Truth Tables

(a) Single-Bit Addition

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Addition with Carry Input

C_{in}	A	B	Sum	C_{out}
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

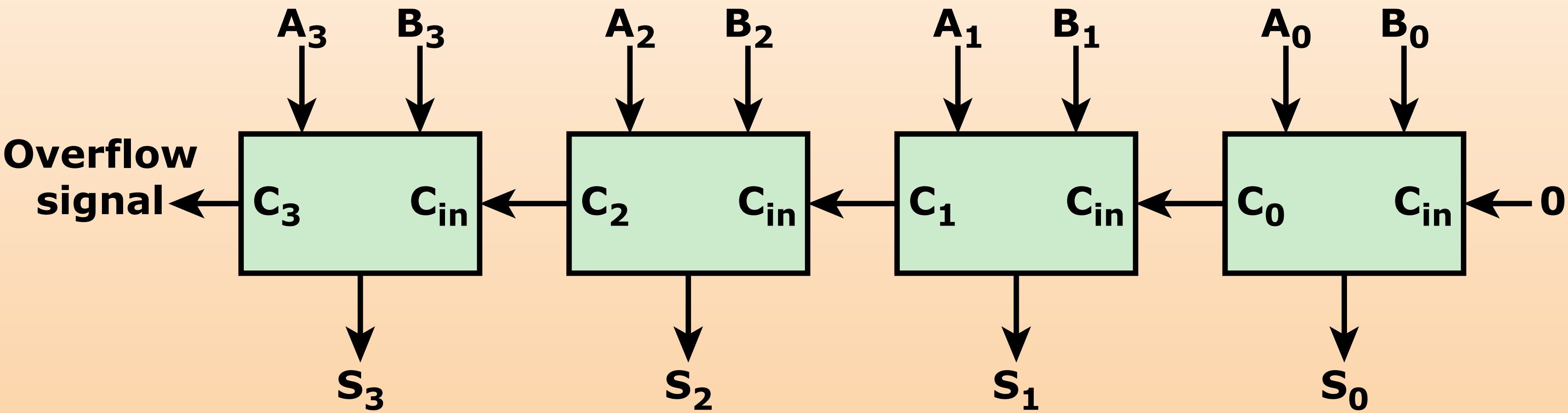


Figure 11.19 4-Bit Adder

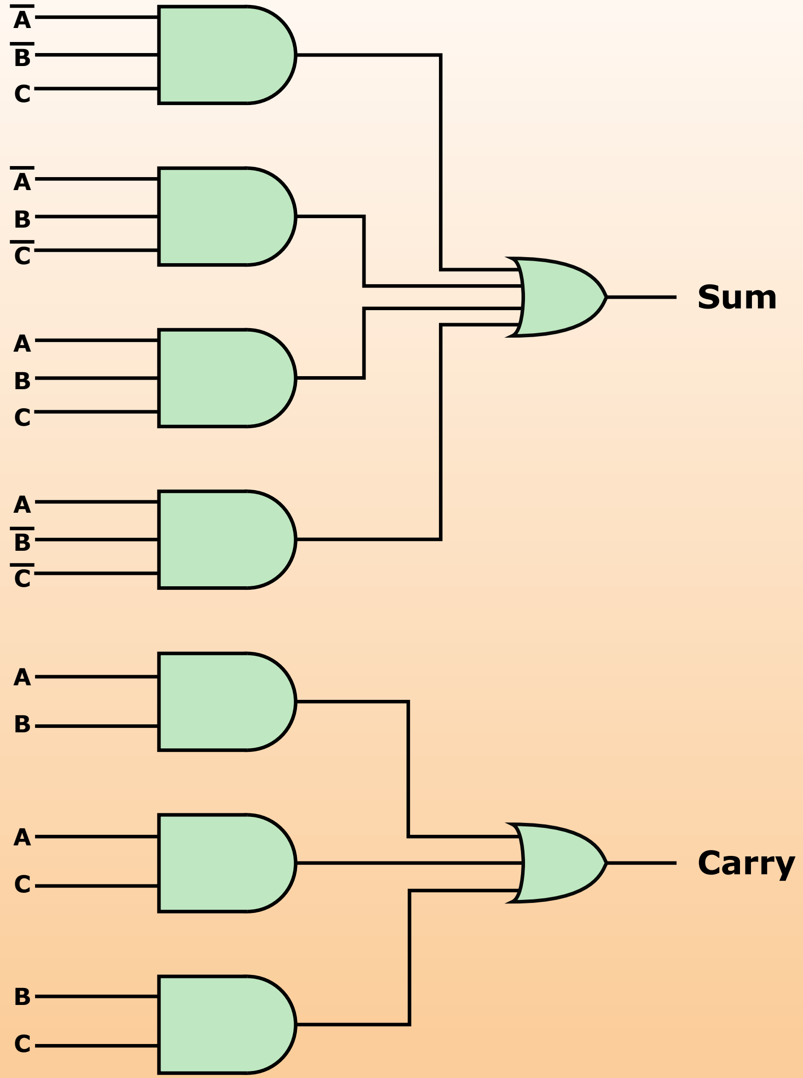


Figure 11.20 Implementation of an Adder

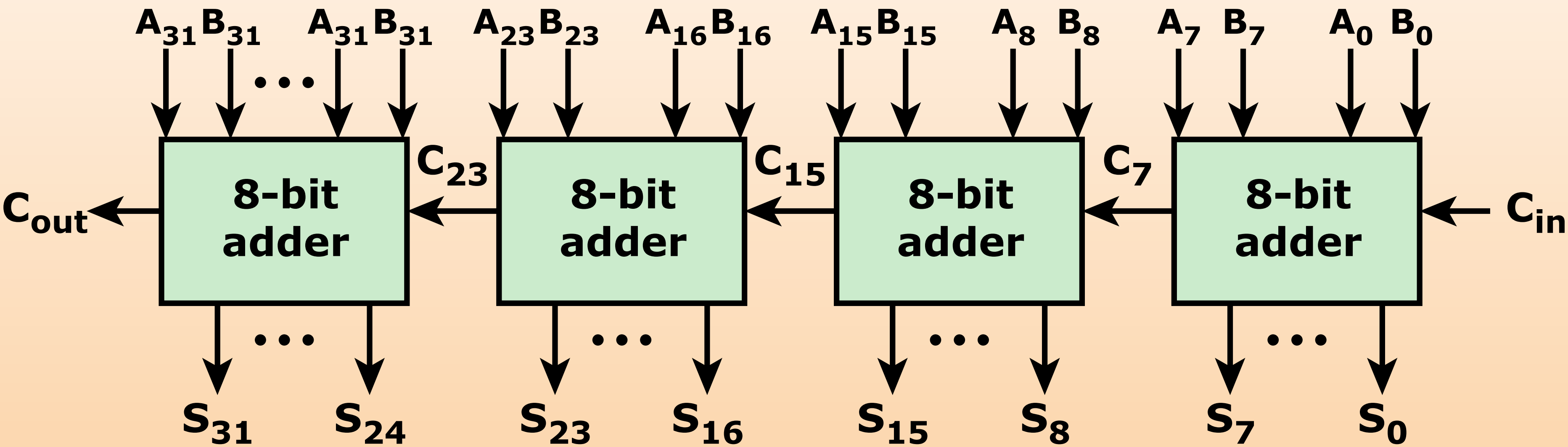


Figure 11.21 Construction of a 32-Bit Adder Using 8-Bit Adders

Sequential Circuit

Sequential Circuit

Current output depends not only on the current input, but also on the past history of inputs

Makes use of combinational circuits

+
Flip-Flops

- Simplest form of sequential circuit
- There are a variety of flip-flops, all of which share two properties:

1. The flip-flop is a bistable device. It exists in one of two states and, in the absence of input, remains in that state. Thus, the flip-flop can function as a 1-bit memory.
2. The flip-flop has two outputs, which are always the complements of each other.

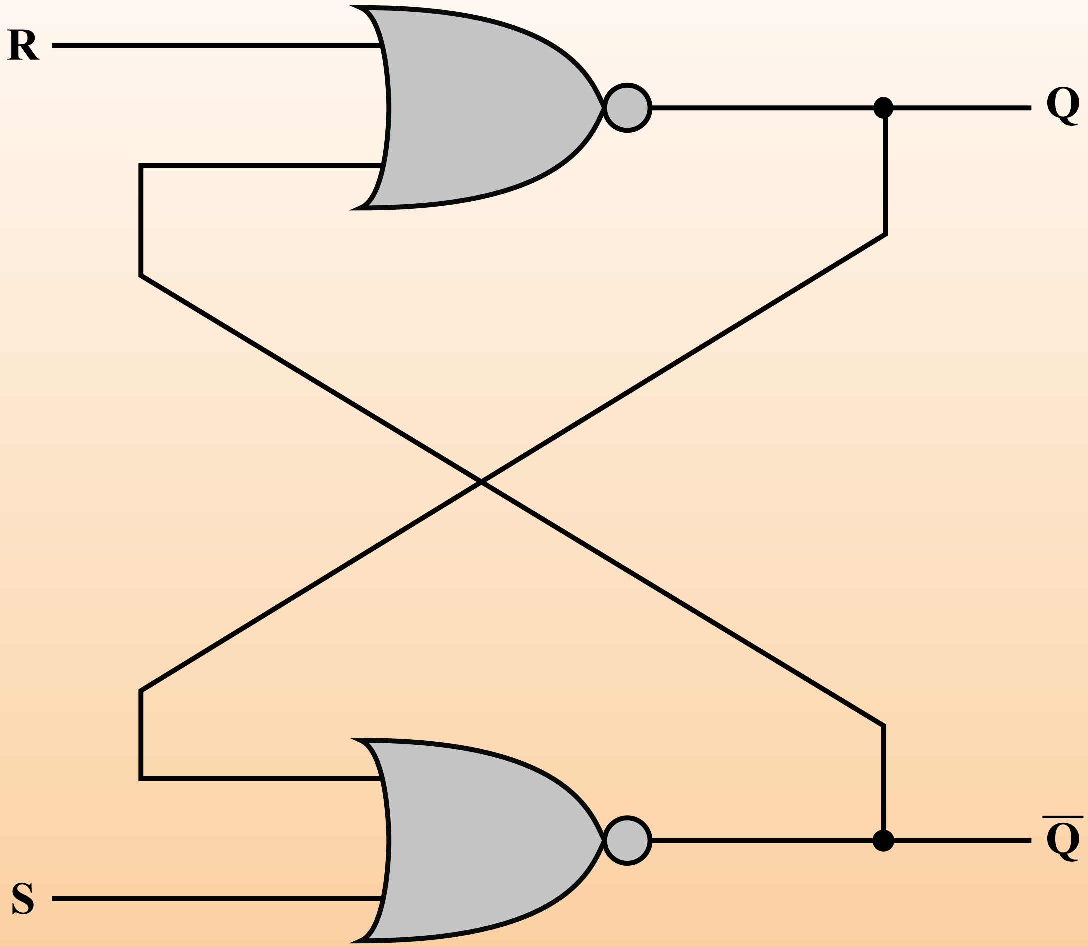


Figure 11.22 The S-R Latch Implemented with NOR Gates

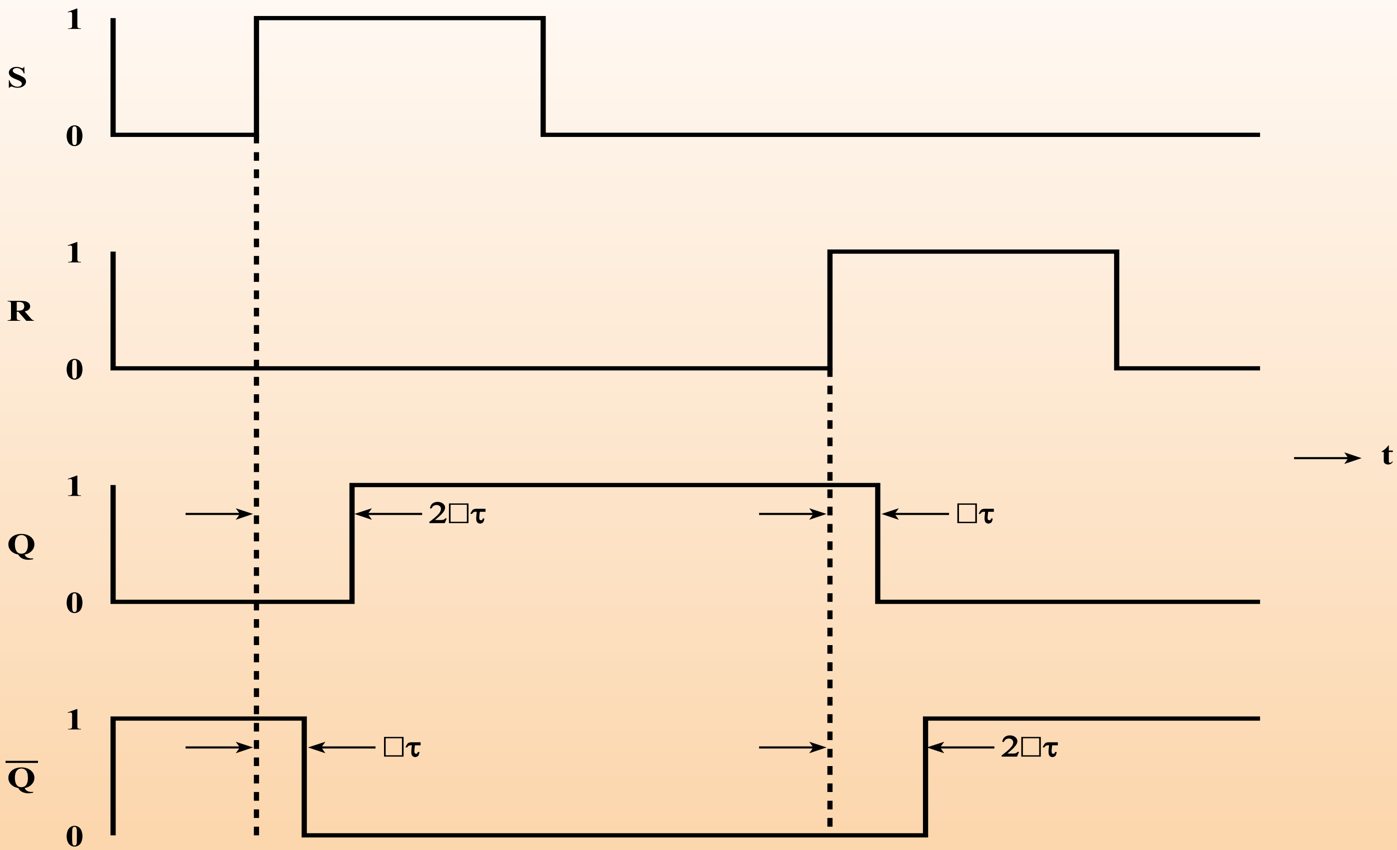


Figure 11.23 NOR S-R Latch Timing Diagram

The output of the S–R latch changes, after a brief time delay, in response to a change in the input.

Table 11.10 The S-R Latch

(a) Characteristic Table

Current Inputs SR	Current State Q_n	Next State Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

(b) Simplified Characteristic Table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

Observe that the inputs S = 1, R = 1 are not allowed, because these would produce an inconsistent output (both Q and Q equal 0).

(c) Response to Series of Inputs

t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

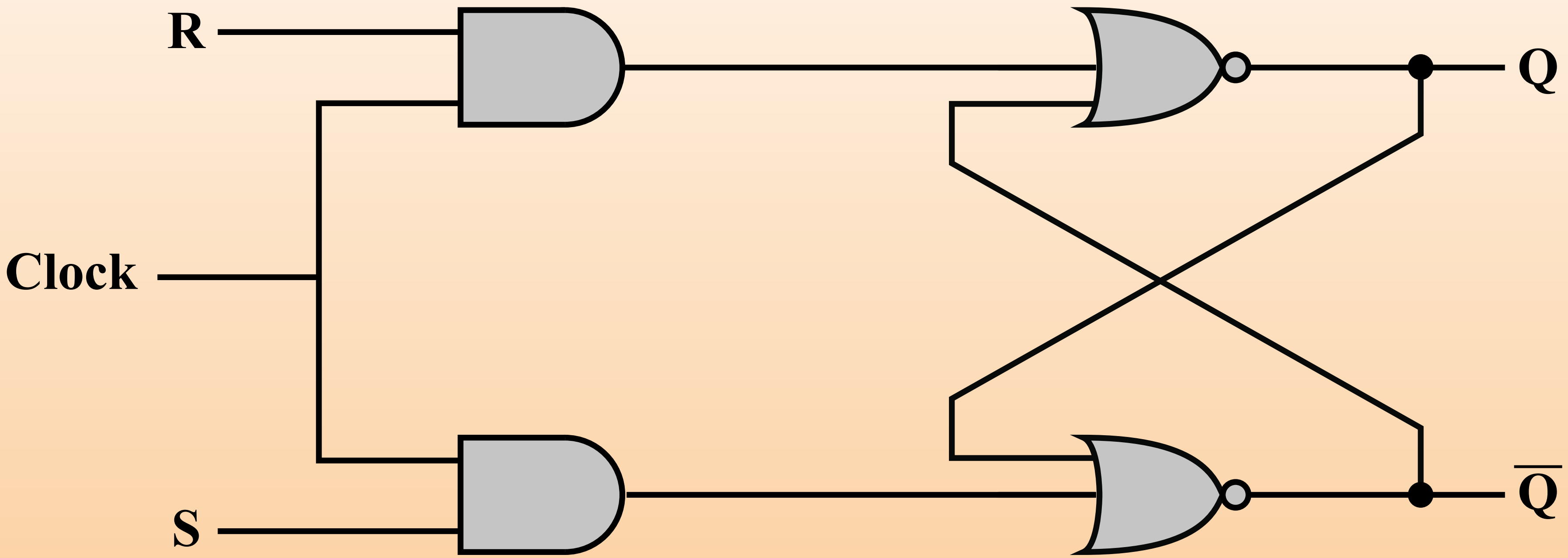


Figure 11.24 Clocked S-R Flip Flop

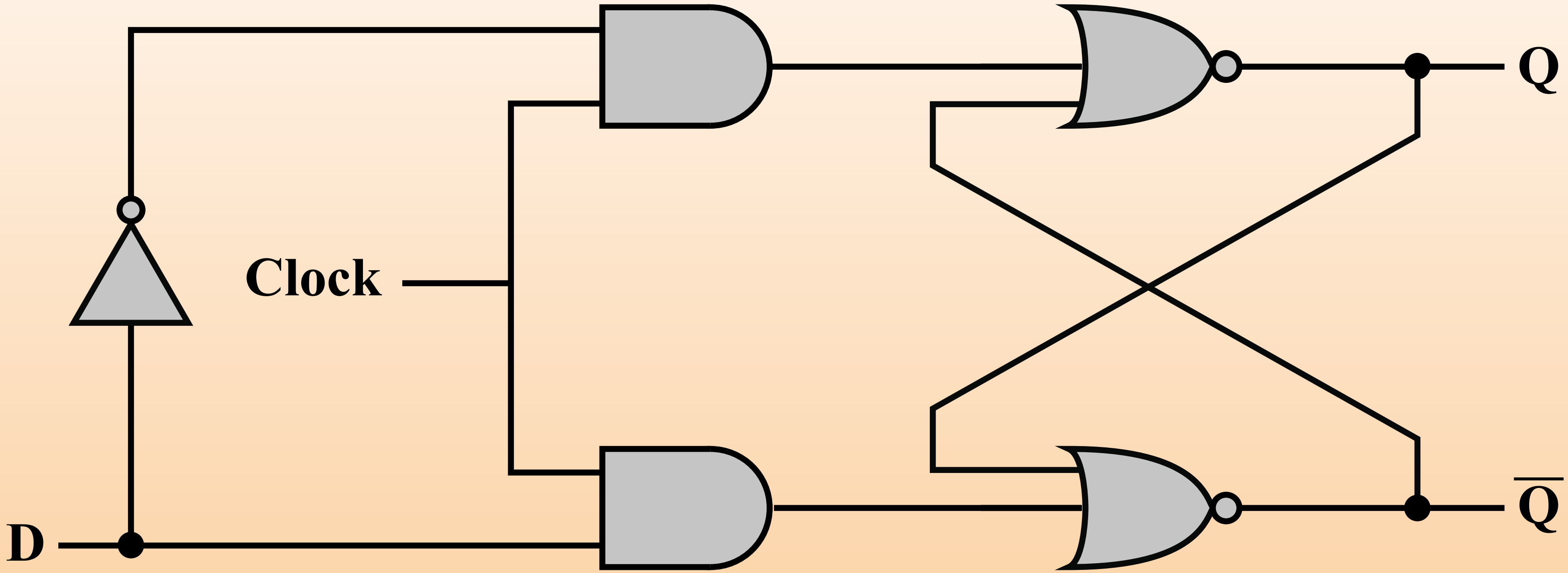


Figure 11.25 D Flip Flop

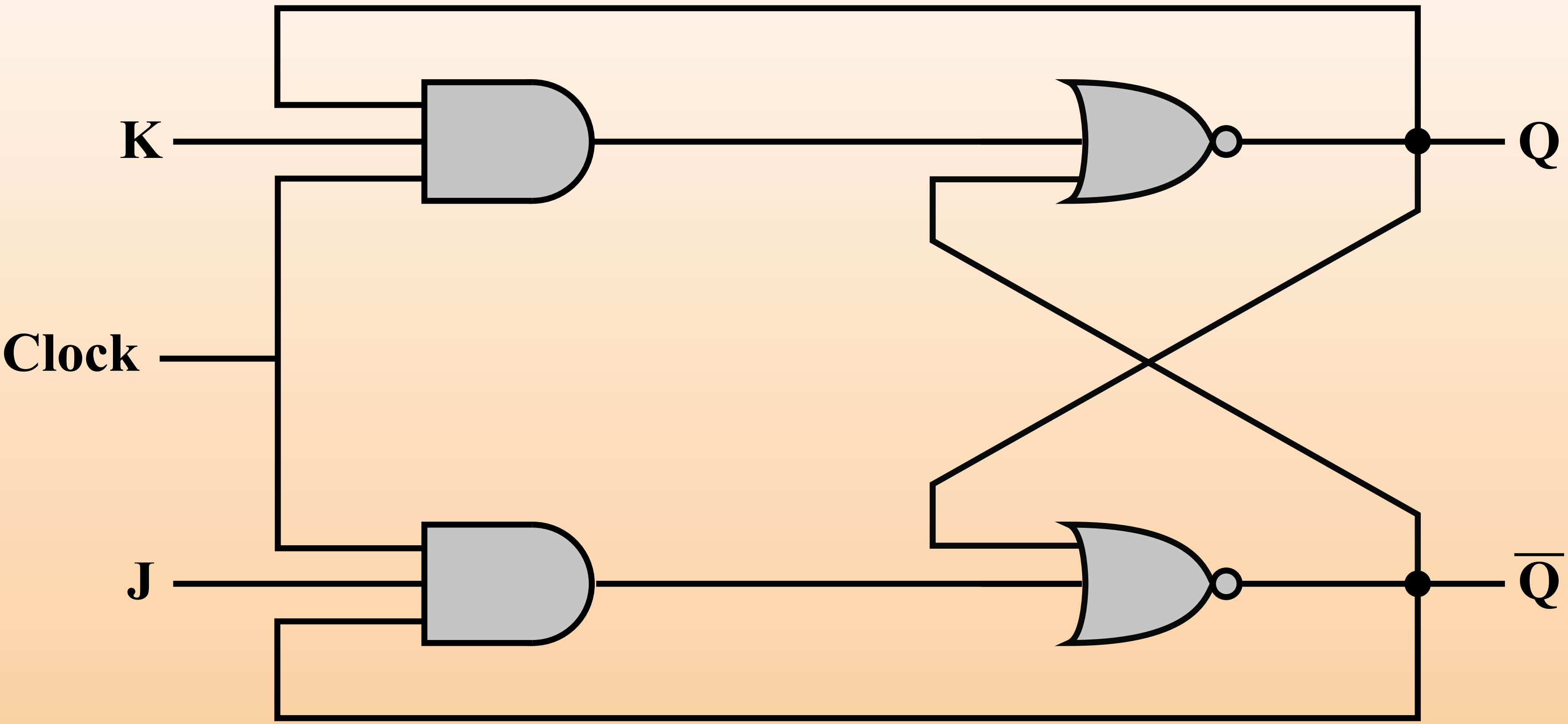


Figure 11.26 J-K Flip Flop

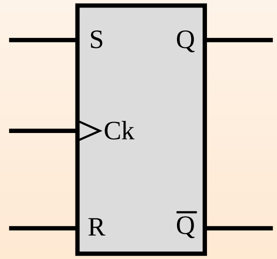
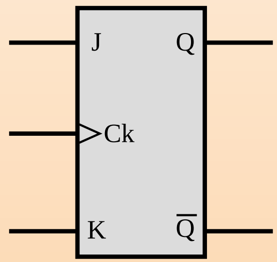
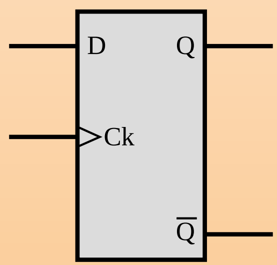
Name	Graphical Symbol	Truth Table
S-R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J-K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q_n}$
D		D Q_{n+1} 0 0 1 1

Figure 11.27 Basic Flip-Flops

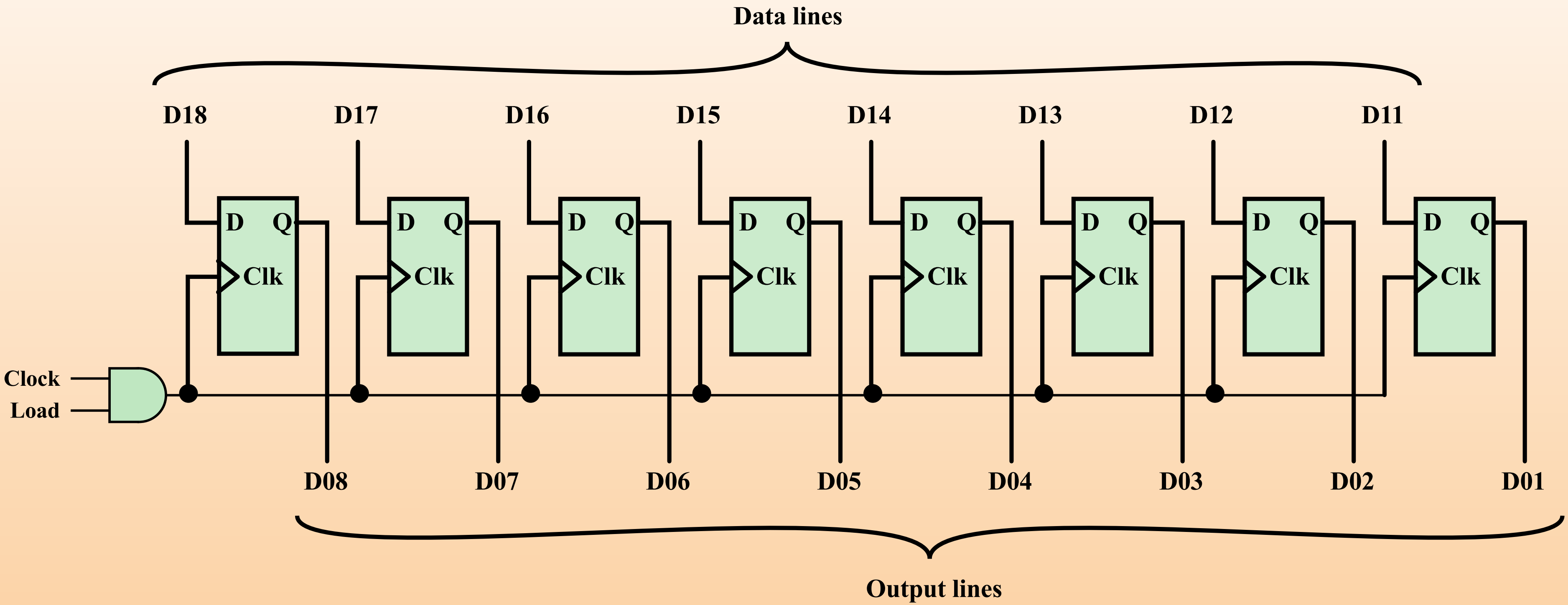


Figure 11.28 8-Bit Parallel Register

The 8-bit register of Figure 11.28 illustrates the operation of a parallel register using D flip-flops. A control signal, labeled *load*, controls writing into the register from signal lines, D11 through D18. These lines might be the output of multiplexers, so that data from a variety of sources can be loaded into the register.

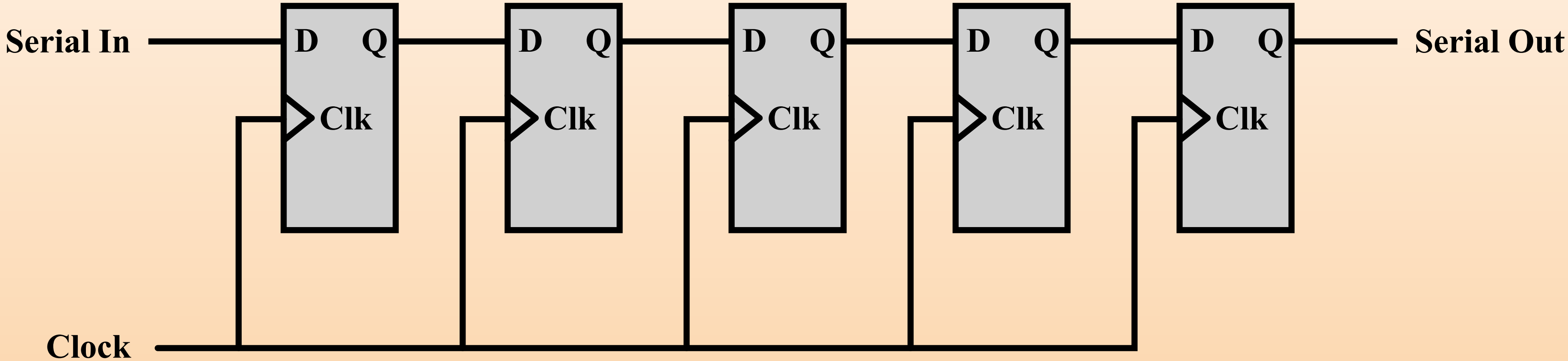
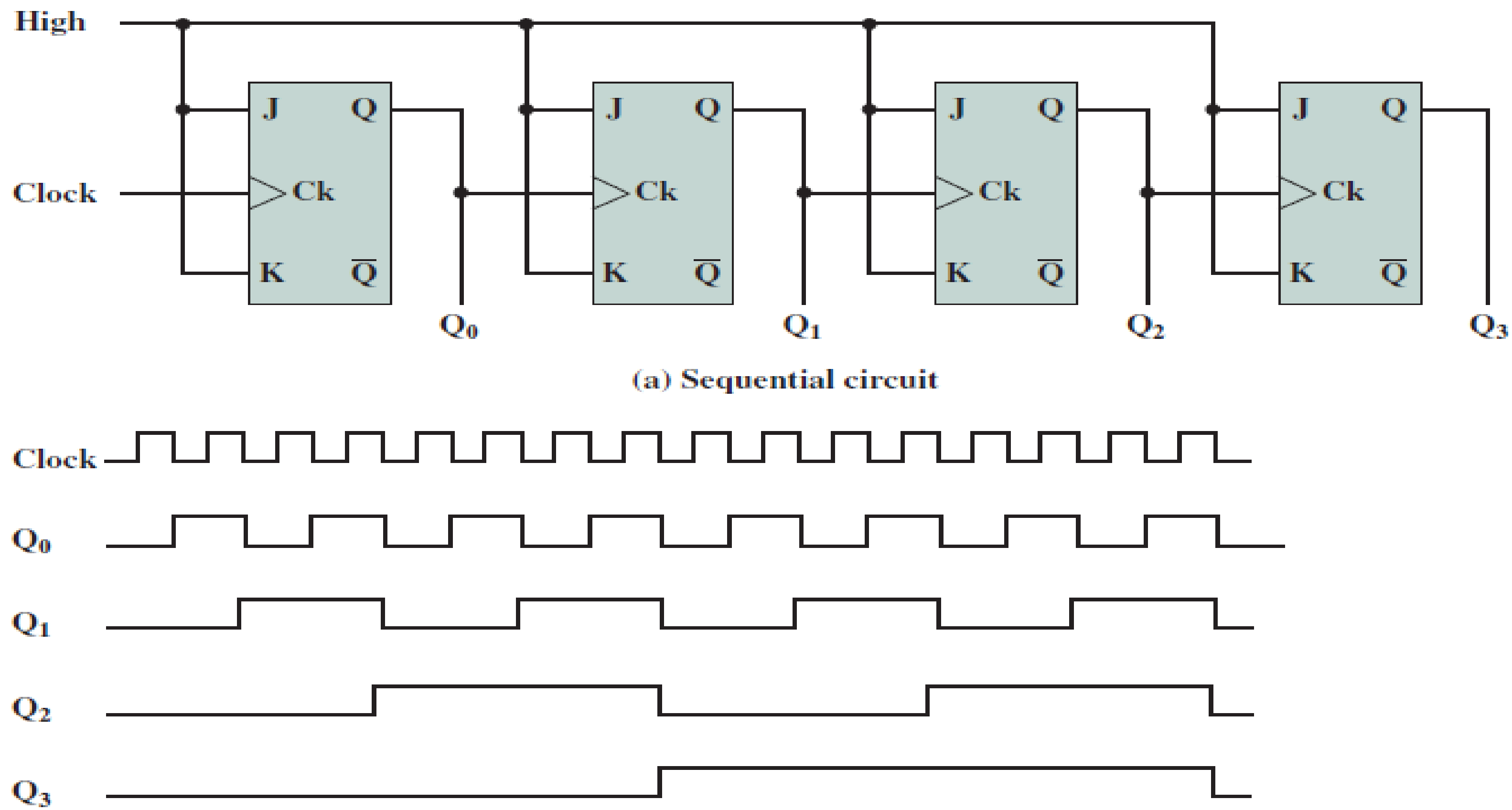


Figure 11.29 5-Bit Shift Register

+ # Counter

- A register whose value is easily incremented by 1 modulo the capacity of the register
- After the maximum value is achieved the next increment sets the counter value to 0
- An example of a counter in the CPU is the program counter
- Can be designated as:
  - Asynchronous
    - Relatively slow because the output of one flip-flop triggers a change in the status of the next flip-flop
  - Synchronous
    - All of the flip-flops change state at the same time
    - Because it is faster it is the kind used in CPUs



(a) Sequential circuit

(b) Timing diagram

An asynchronous counter is also referred to as a **ripple counter,** because the change that occurs to increment the counter starts at one end and "ripples" through to the other end. Figure 11.30 shows an implementation of a 4-bit counter using J–K flip-flops, together with a timing diagram that illustrates its behavior.

In the illustrated implementation, the counter is incremented with each clock pulse. The J and K inputs to each flip-flop are held at a constant 1. This means that, when there is a clock pulse, the output at Q will be inverted (1 to 0; 0 to 1).

(a) Truth table

C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1

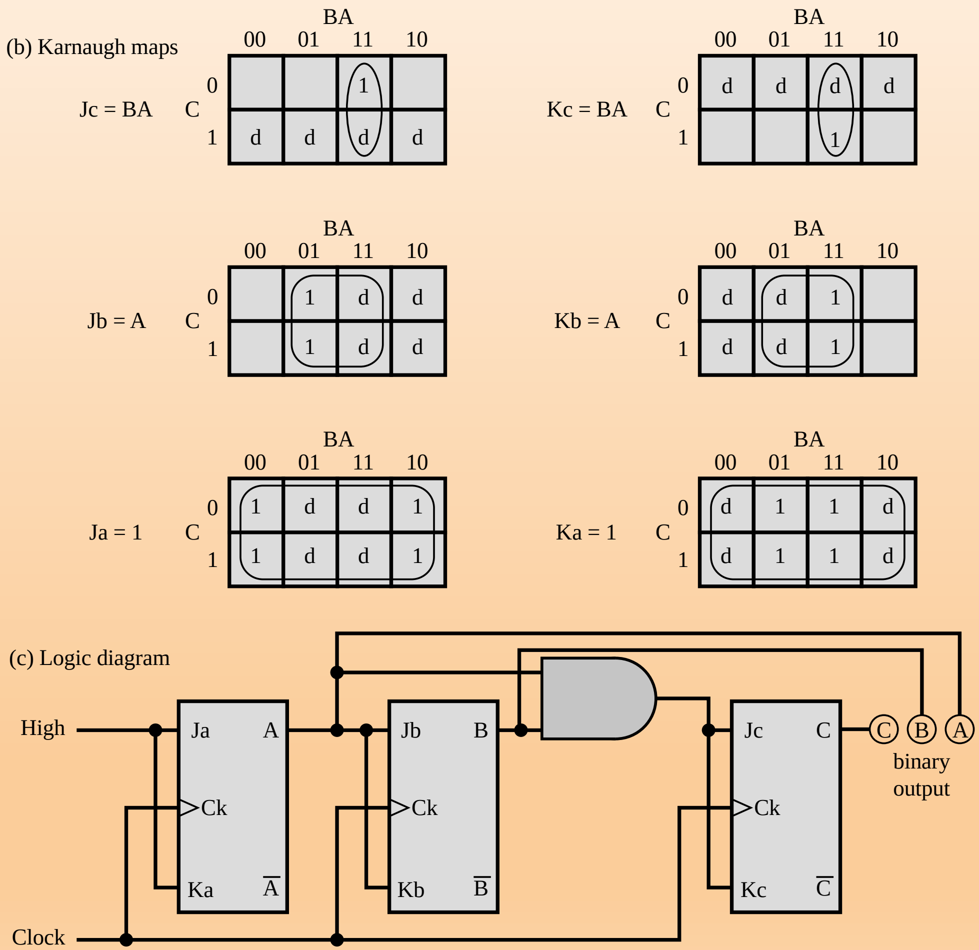


Figure 11.31 Design of a Synchronous Counter

The ripple counter has the disadvantage of the delay involved in changing value, which is proportional to the length of the counter. To overcome this disadvantage, CPUs make use of synchronous counters, in which all of the flip-flops of the counter change at the same time. In this subsection, we present a design for a 3-bit synchronous counter. In doing so, we illustrate some basic concepts in the design of a synchronous circuit.

Programmable Logic Device (PLD)
A general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured "in-system". Also referred to as a field-programmable device (FPD).

Programmable Logic Array (PLA)
A relatively small PLD that contains two levels of logic, an AND-plane and an OR-plane, where both levels are programmable.

Programmable Array Logic (PAL)
A relatively small PLD that has a programmable AND-plane followed by a fixed OR-plane.

Simple PLD (SPLD)
A PLA or PAL.

Complex PLD (CPLD)
A more complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.

Field-Programmable Gate Array (FPGA)
A PLD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs.

Logic Block
A relatively small circuit block that is replicated in an array in an FPD. When a circuit is implemented in an FPD, it is first decomposed into smaller sub-circuits that can each be mapped into a logic block. The term logic block is mostly used in the context of FPGAs, but it could also refer to a block of circuitry in a CPLD.

Table 11.11

PLD Terminology

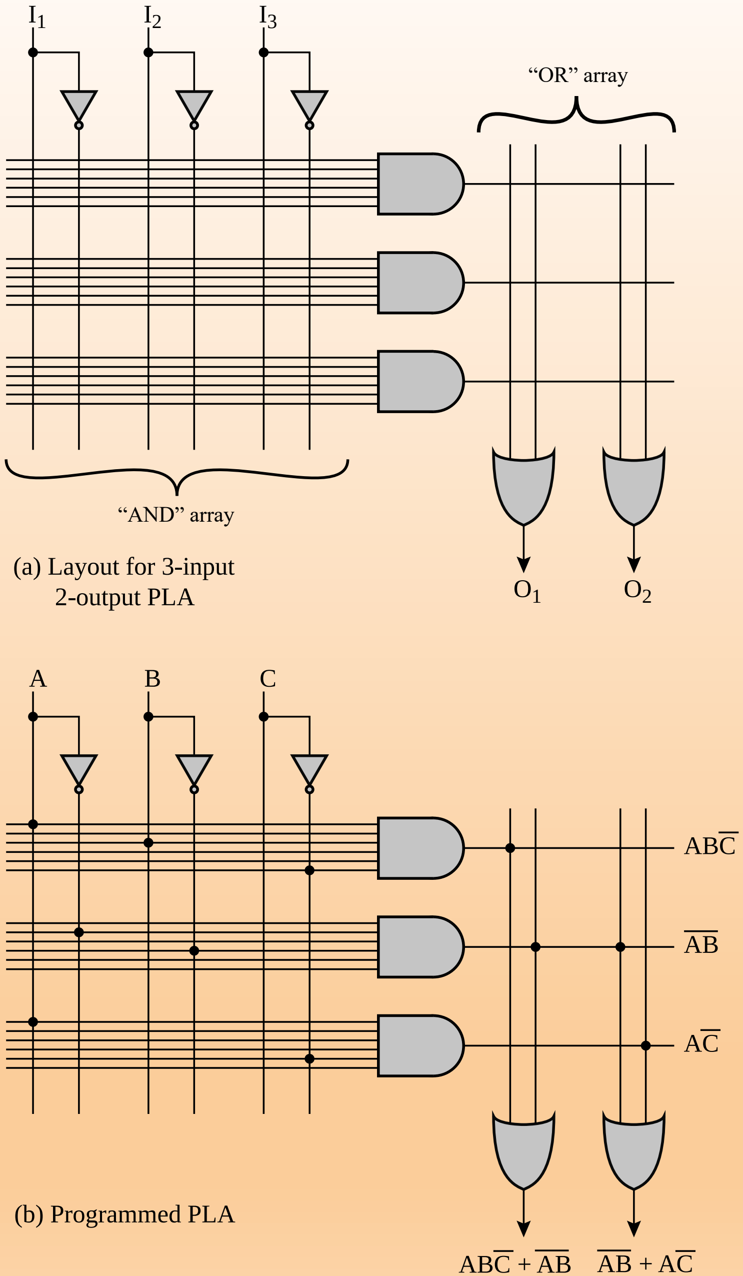


Figure 11.32 An Example of a Programmable Logic Array

The AND array is programmed by establishing a connection between any PLA input or its negation and any AND gate input by connecting the corresponding lines at their point of intersection. On the right is a programmable OR array, which involves connecting AND gate outputs to OR gate inputs.

PLAs are manufactured in two different ways to allow easy programming (making of connections). In the first, every possible connection is made through a fuse at every intersection point. The undesired connections can then be later removed by blowing the fuses. This type of PLA is referred to as a *field-programmable logic array.* Alternatively, the proper connections can be made during chip fabrication by using an appropriate mask supplied for a particular interconnection pattern

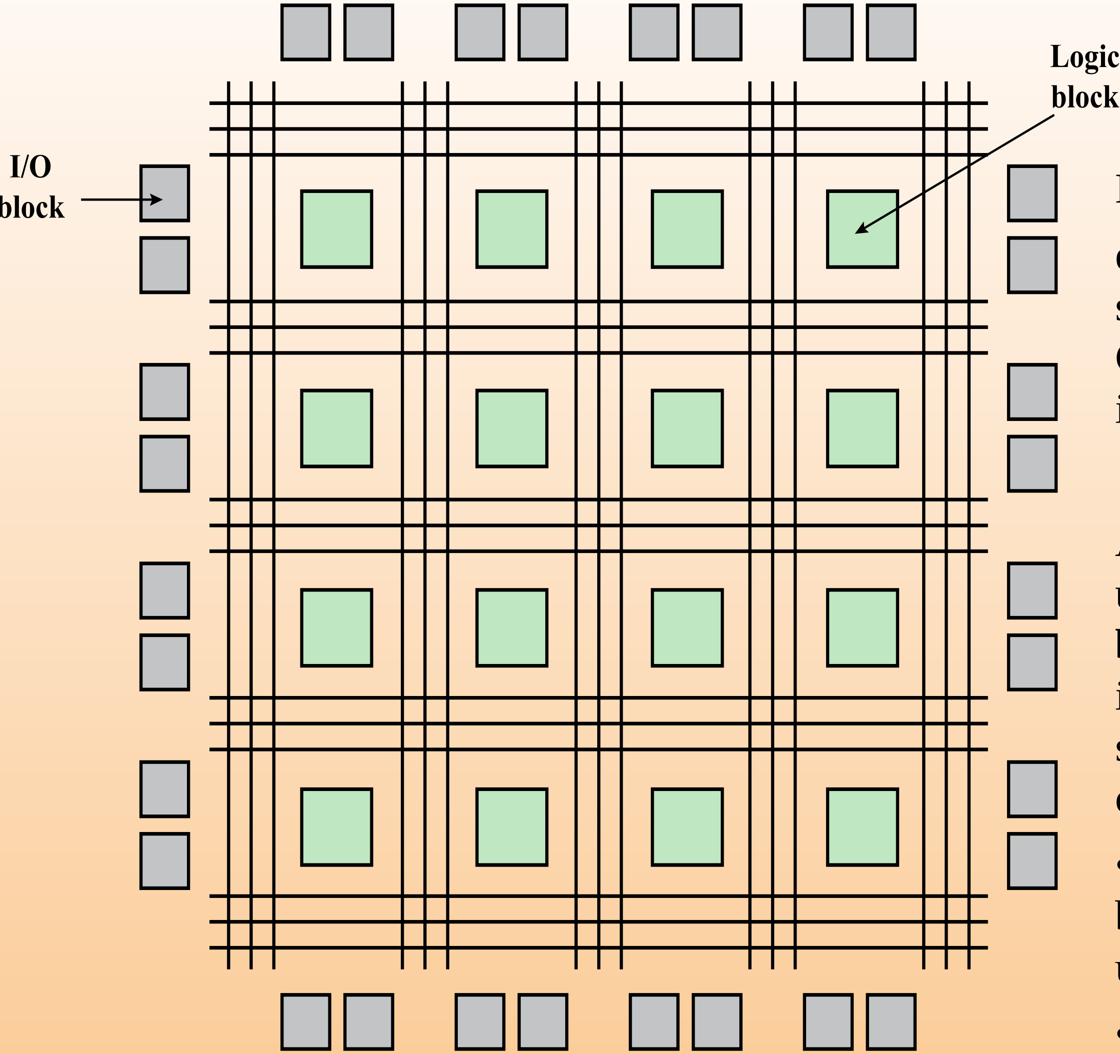


Figure 11.33 Structure of an FPGA

Many commercial PLD products

exist on the market today with this basic structure, and are collectively referred to as Complex PLDs (CPLDs). The most important type of CPLD is the FPGA.

An FPGA consists of an array of uncommitted circuit elements, called **logic blocks,** and interconnect resources. An illustration of a typical FPGA architecture is shown in Figure 11.33. The key components of an FPGA are;

- **Logic block:** The configurable logic blocks are where the computation of the user's circuit takes place.
- **I/O block:** The I/O blocks connect I/O pins to the circuitry on the chip.
- **Interconnect:** These are signal paths available for establishing connections among I/O blocks and logic blocks.

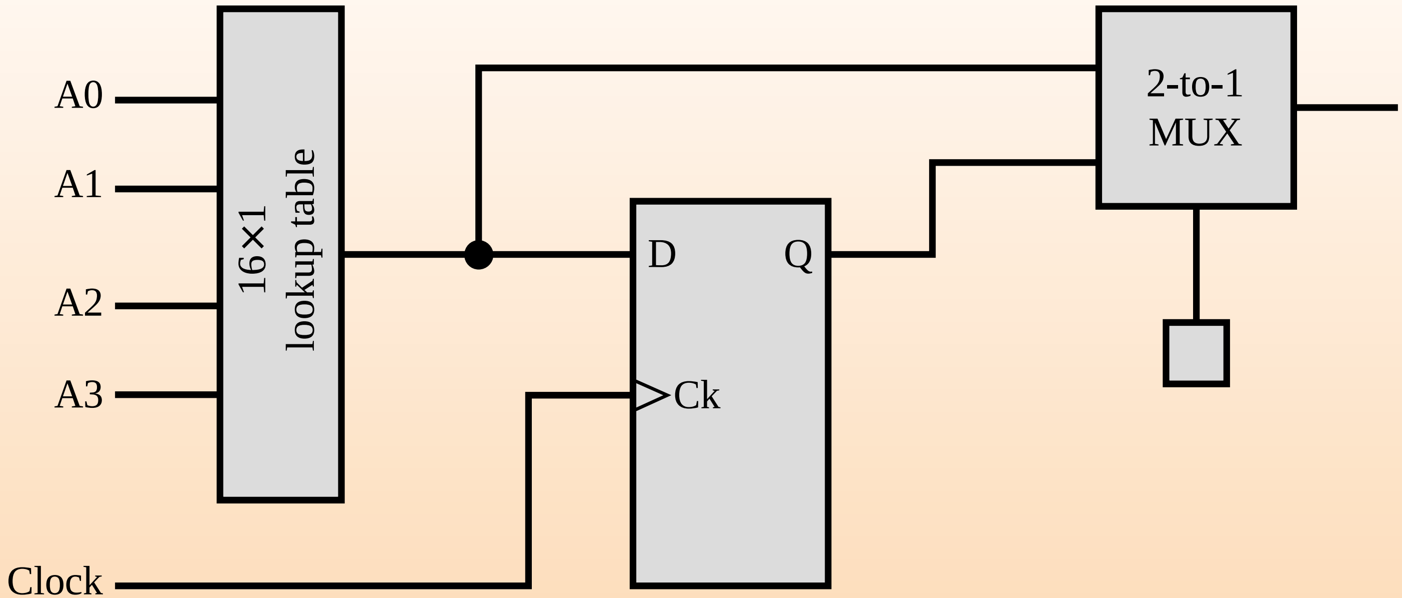


Figure 11.34 shows an example of a simple logic block consisting of a D flip-flop, a 2-to-1 multiplexer, and a 16-bit **lookup table.** The lookup table is a memory consisting of 16 1-bit elements, so that 4 input lines are required to select one of the 16 bits. Larger logic blocks have larger lookup tables and multiple interconnected lookup tables. The combinational logic realized by the lookup table can be output directly or stored in the D flip-flop and output synchronously. A separate one-bit memory controls the multiplexer to determine whether the output comes directly from the lookup table or from the flip-flop.

By interconnecting numerous logic blocks, very complex logic functions can be easily implemented.

Figure 11.34 A Simple FPGA Logic Block

+ Summary

Chapter 11

Digital Logic

- Boolean Algebra
- Gates
- Combinational Circuits
  - Implementation of Boolean Functions
  - Multiplexers
  - Decoders
  - Read-Only-Memory
  - Adders
- Sequential Circuits
  - Flip-Flops
  - Registers
  - Counters
- Programmable Logic Devices
  - Programmable Logic Array
  - Field-Programmable Gate Array