# *Logic Design*

A Review

# *Boolean Algebra*

- Two Values: zero and one
- Three Basic Functions: And, Or, Not
- Any Boolean Function Can be Constructed from These Three

| And | 0 | 1 |
|---|---|---|
| 0 | **0** | **0** |
| 1 | **0** | **1** |

| Or | 0 | 1 |
|---|---|---|
| 0 | **0** | **1** |
| 1 | **1** | **1** |

| Not | |
|---|---|
| 0 | **1** |
| 1 | **0** |

# *Algebraic Laws*

| Classification | Law |
|---|---|
| **Identity** | $a1=1a=a$<br>$a+0=0+a=a$ |
| **Dominance** | $a0=0a=0$<br>$1+a=a+1=1$ |
| **Commutativity** | $a+b=b+a$<br>$ab=ba$ |
| **Associativity** | $a(bc)=(ab)c$<br>$a+(b+c)=(a+b)+c$ |
| **Distributive** | $a(b+c)=ab+ac$<br>$a+bc=(a+b)(a+c)$ |
| **Demorgan's Laws** | $(a+b)' = a'b'$<br>$(ab)' = a'+b'$ |

# *Boolean Expressions*

- Addition represents OR
- Multiplication represents AND
- Not is represented by a prime a’ or an overbar $\overline{a}$
- Examples:
- s = a’bc + ab’c + abc’ + a’b’c’
- q = ab + bc + ac + abc

# *Superfluous Terms*

- The following Two Equations Represent The Same Function.

$$q = ab + bc + ac + abc$$

$$q = ab + bc + ac$$

| a | b | c | q |
|---|---|---|---|
| 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 |

# *Prime Implicants*

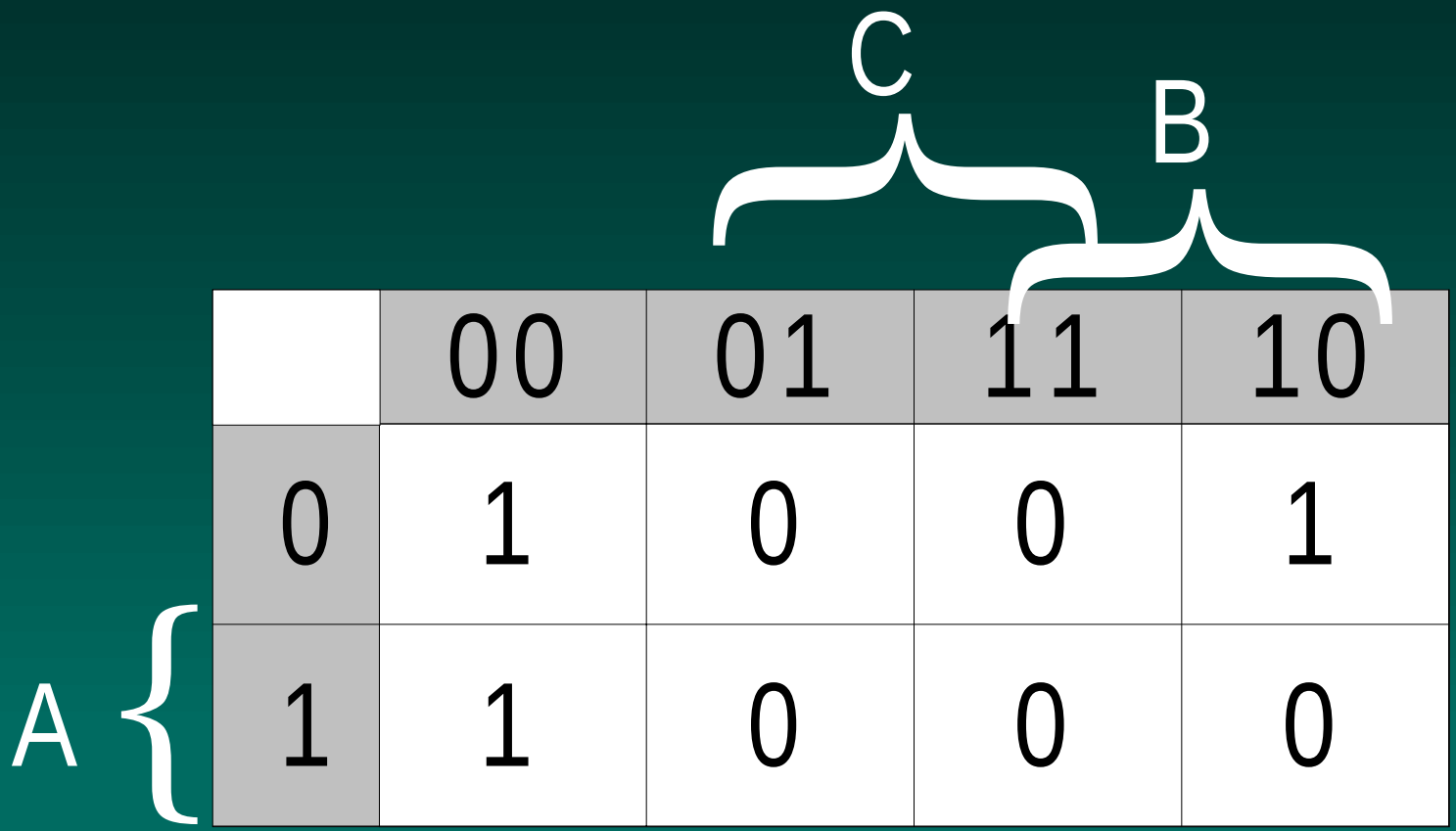
- A Prime Implicant is a Product of Variables or Their Complements, eg. ab'cd'
- If a Prime Implicant has the Value 1, then the Function has the Value 1
- A Minimal Equation is a Sum of Prime Implicants

# *Minimization and Minterms*

- Minimization Reduces the Size and Number of Prime Implicants
- A MinTerm is a Prime Implicant with the Maximum Number of Variables
- For a 3-input Function a'bc is a MinTerm, while ab is not.
- Prime Implicants can be Combined to Eliminate Variables, abc'+abc = ab

# *Minimization with Maps*

- A Karnaugh Map



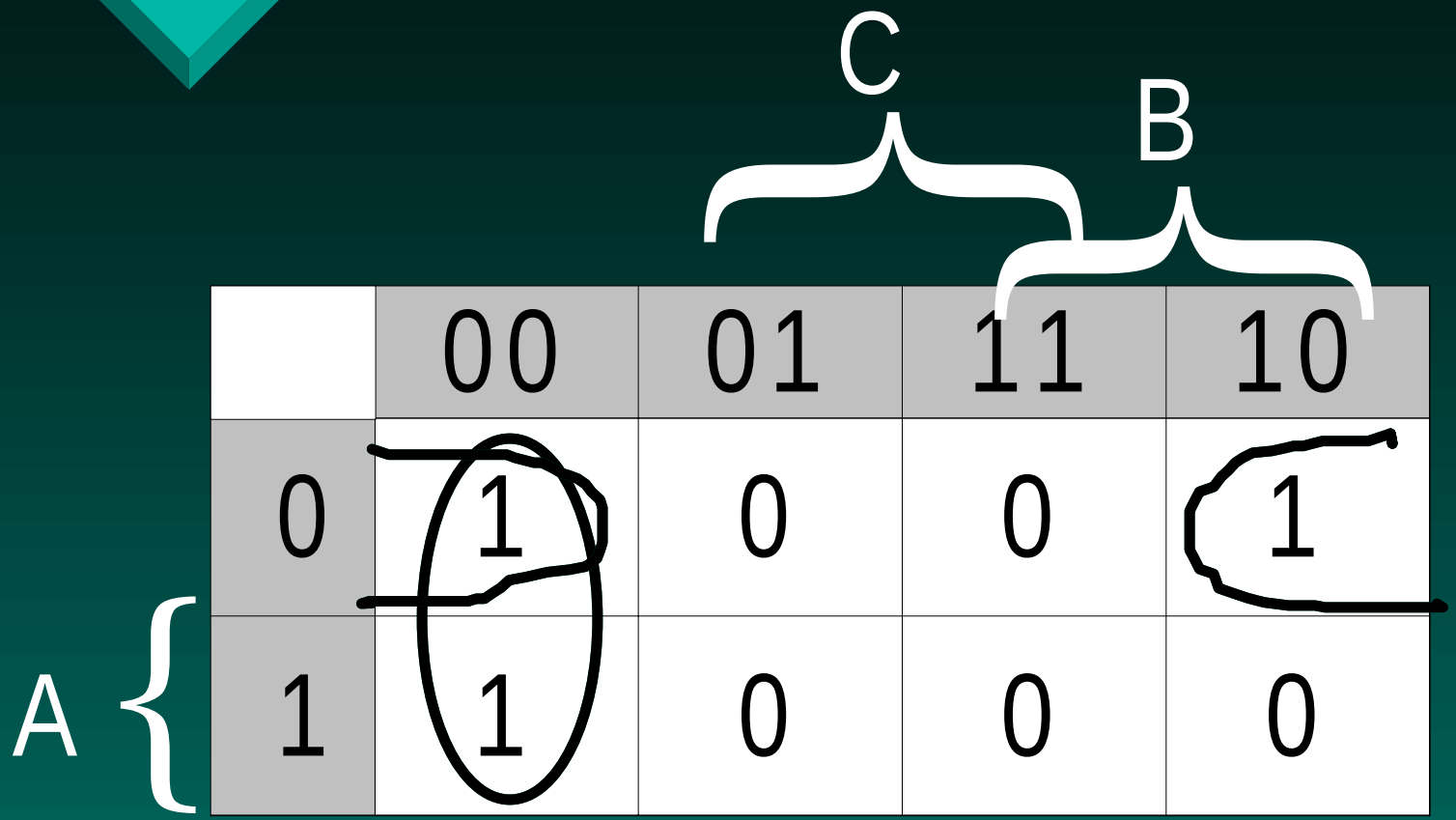
# *Procedure*

- Select Regions Containing All 1's
- Regions should be as Large as Possible
- Regions must contain $2^k$ cells
- Regions should overlap as little as possible
- The complete set of regions must contain all 1's in the map

# *Procedure 2*

- Top and Bottom of Map are Contiguous
- Left and Right of Map are Contiguous
- Regions represent Prime Implicants
- Use Variable name guides to construct equation
  - Completely inside the region of a variable means prime implicant contains variable
  - Completely outside the region of a variable means prime implicant contains negation

# *Applied to Previous Map*

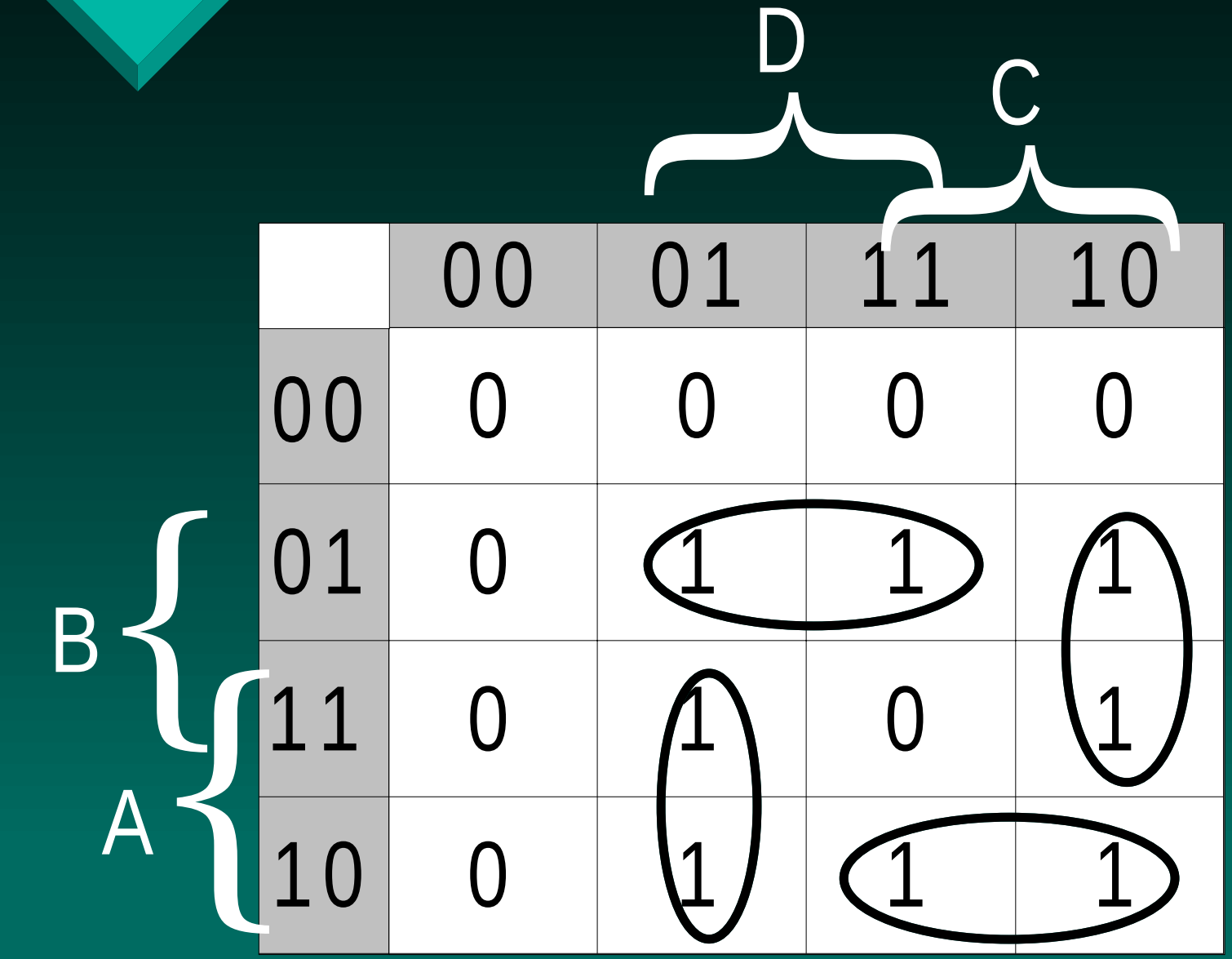



q=c’b’+c’a’

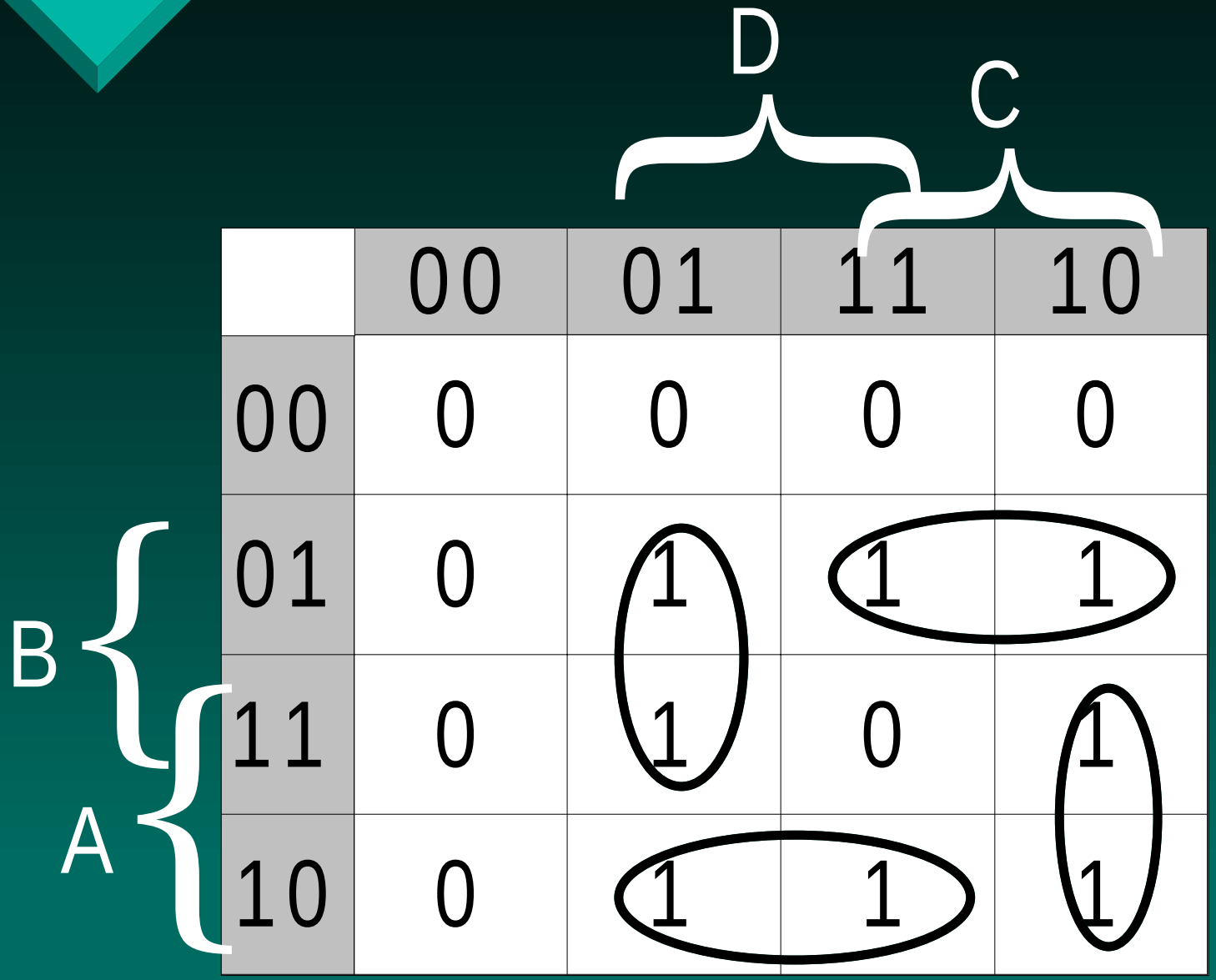
# A 4-Variable Karnaugh Map

| B \ A | D C = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 1 | 1 |

# First Minimization



# *Second Minimization*



# *Minimal Forms for Previous Slides:*

- $ab'd + bc'd + a'bc + acd'$
- $ac'd + a'bd + bcd' + ab'c$
- Moral: A Boolean Function May Have Several Different Minimal Forms
- Karnaugh Maps are Ineffective for Functions with More than Six Inputs.

# *Quine McClusky Minimization*

- Amenable to Machine Implementation
- Applicable to Circuits with an Arbitrary Number of Inputs
- Effective Procedure for Finding Prime Implicants, but …
- Can Require an Exponential Amount of Time for Some Circuits

# *Quine-McClusky Procedure*

- Start with The Function Truth Table
- Extract All Input Combinations that Produce a TRUE Output (MinTerms)
- Group All MinTerms by The Number of Ones They Contain
- Combine Minterms from Adjacent Groups

# *More Quine-McClusky*

- Two Min-Terms Combine If They Differ by Only One Bit
- The Combined MinTerm has an x in the Differing Position
- Create New Groups From Combined Min-Terms
- Each Member of A New Group Must Have the Same Number of 1's and x's

# *Yet More Quine-McClusky*

- Each Member of A Group Must Have x's in The Same Position.
- Combine Members of the New Groups To Create More New Groups
- Combined Terms Must Differ By One Bit, and Have x's in the Same Positions
- Combine as Much as Possible
- Select Prime Implicants to "Cover" All Ones in the Function

# *Quine-McClusky Example 1*

Numbers in Parentheses are Truth-Table Positions.

| |
|---|
| **0011(3) 1100(12)** |
| **0111(7) 1011(11) 1101(13) 1110(14)** |
| **1111(15)** |

# *Quine-McClusky Example 2*

New Groups After Combining MinTerms

| | |
|---|---|
| 0x11(3,7) | 110x(12,13) |
| 1x11(11,15) | 111x(14,15) |
| x011(3,11) | 11x0(12,14) |
| x111(7,15) | 11x1(13,15) |

# *Quine-McClusky Example 3*

The Final Two Groups

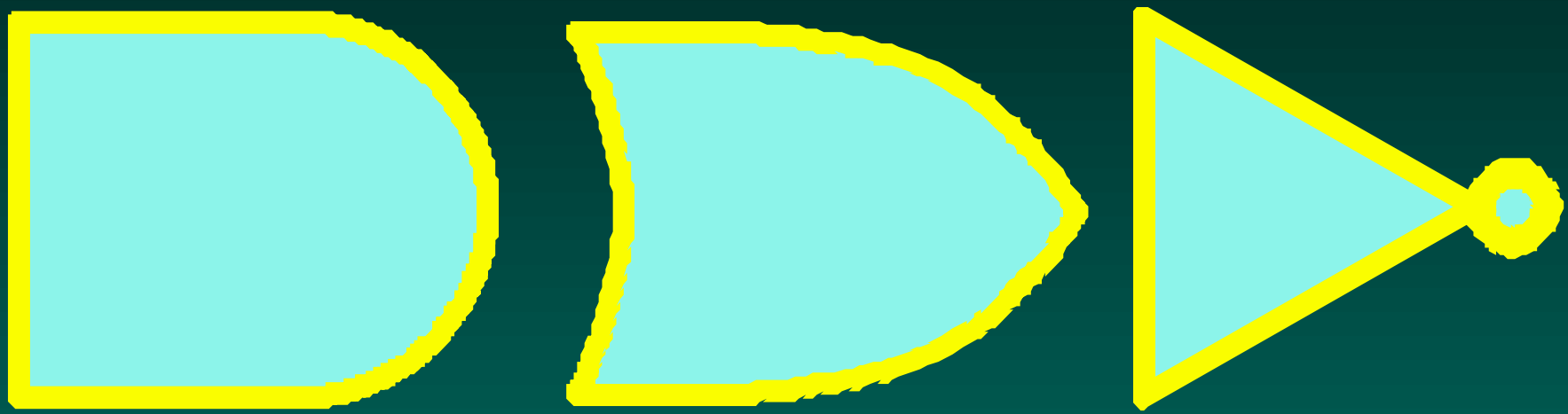
Note That These Two Elements Cover All Truth-Table Positions

| |
|---|
| xx11(3,7,11,15) |
| 11xx(12,13,14,15) |

# *Quine-McClusky Example 4*

- Each Group Element Represents a Prime Implicant
- It is Necessary to Select Group Elements to Cover All Truth-Table Positions.
- In This Case, ab+cd is the Minimal Formula.
- In General, Selecting a Minimal Number of Prime Implicants is NP-Complete

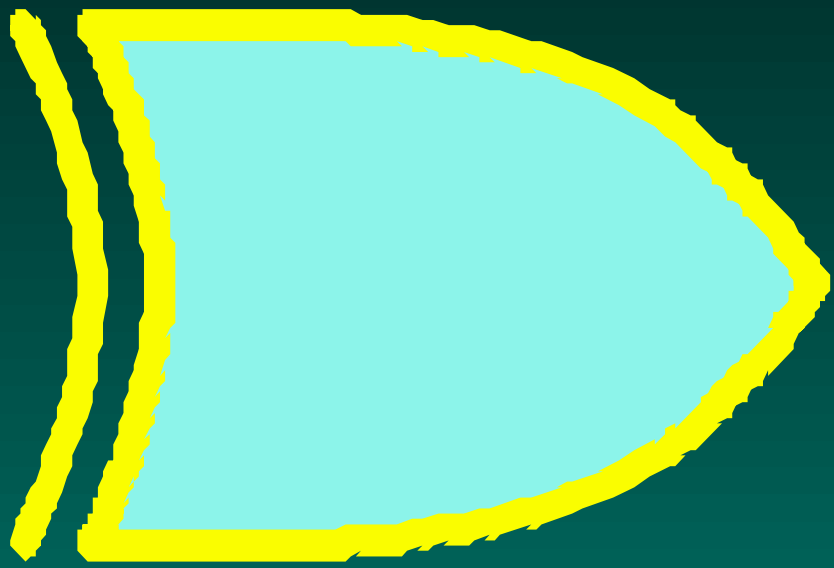
# *Basic Logic Symbols*



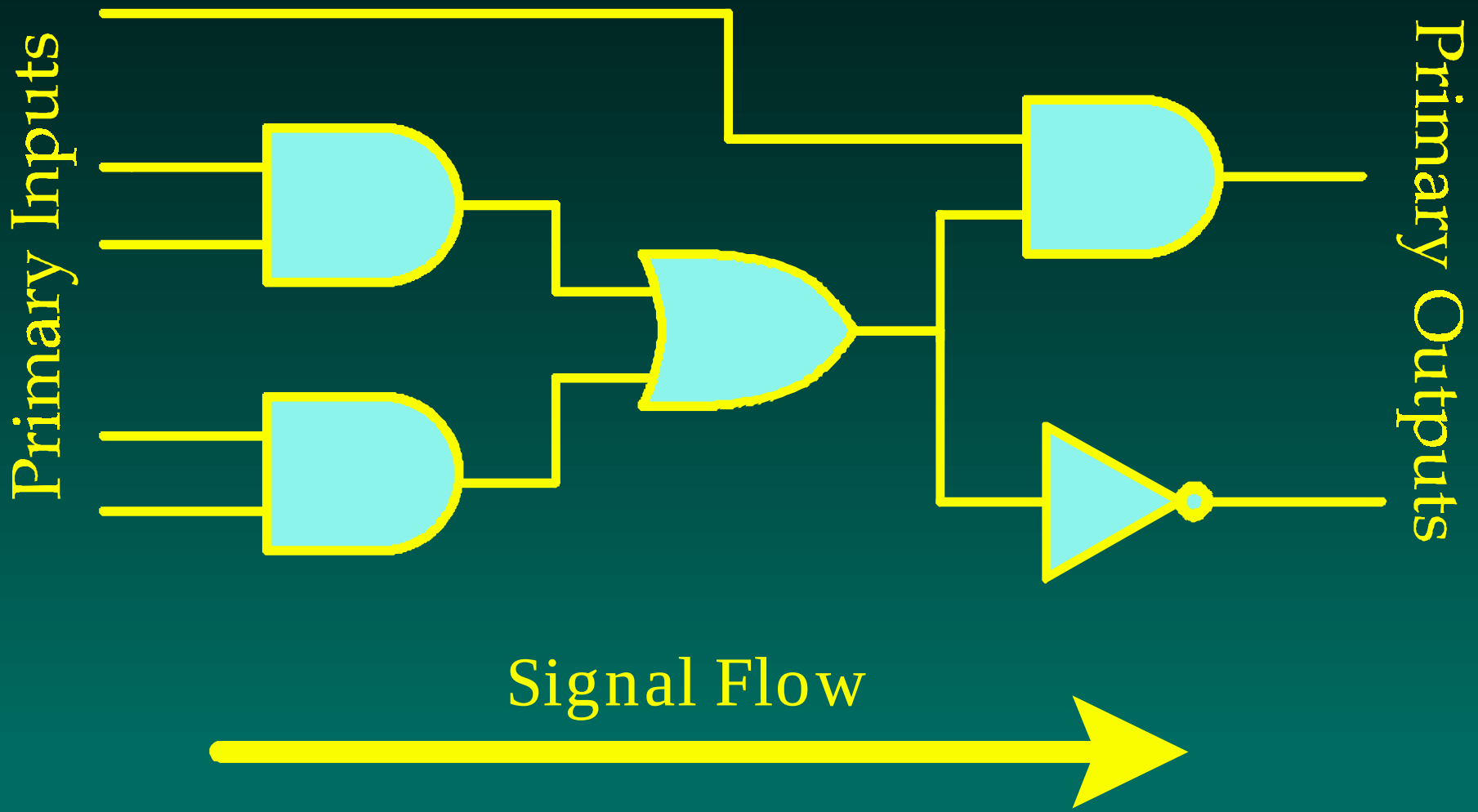
And Or Not

# *The Exclusive Or Function*

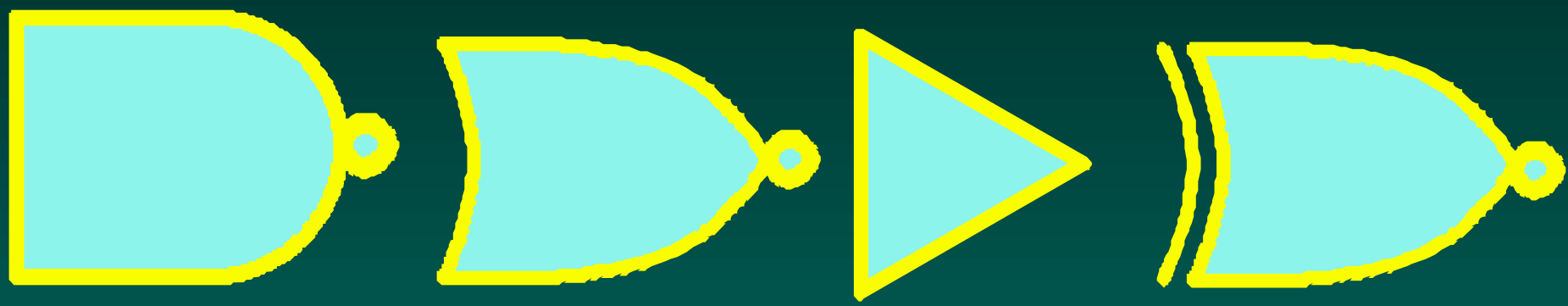
| Xor | 0 | 1 |
|---|---|---|
| 0 | **0** | **1** |
| 1 | **1** | **0** |



# *A Simple Logic Diagram*



# *Additional Logic Symbols*

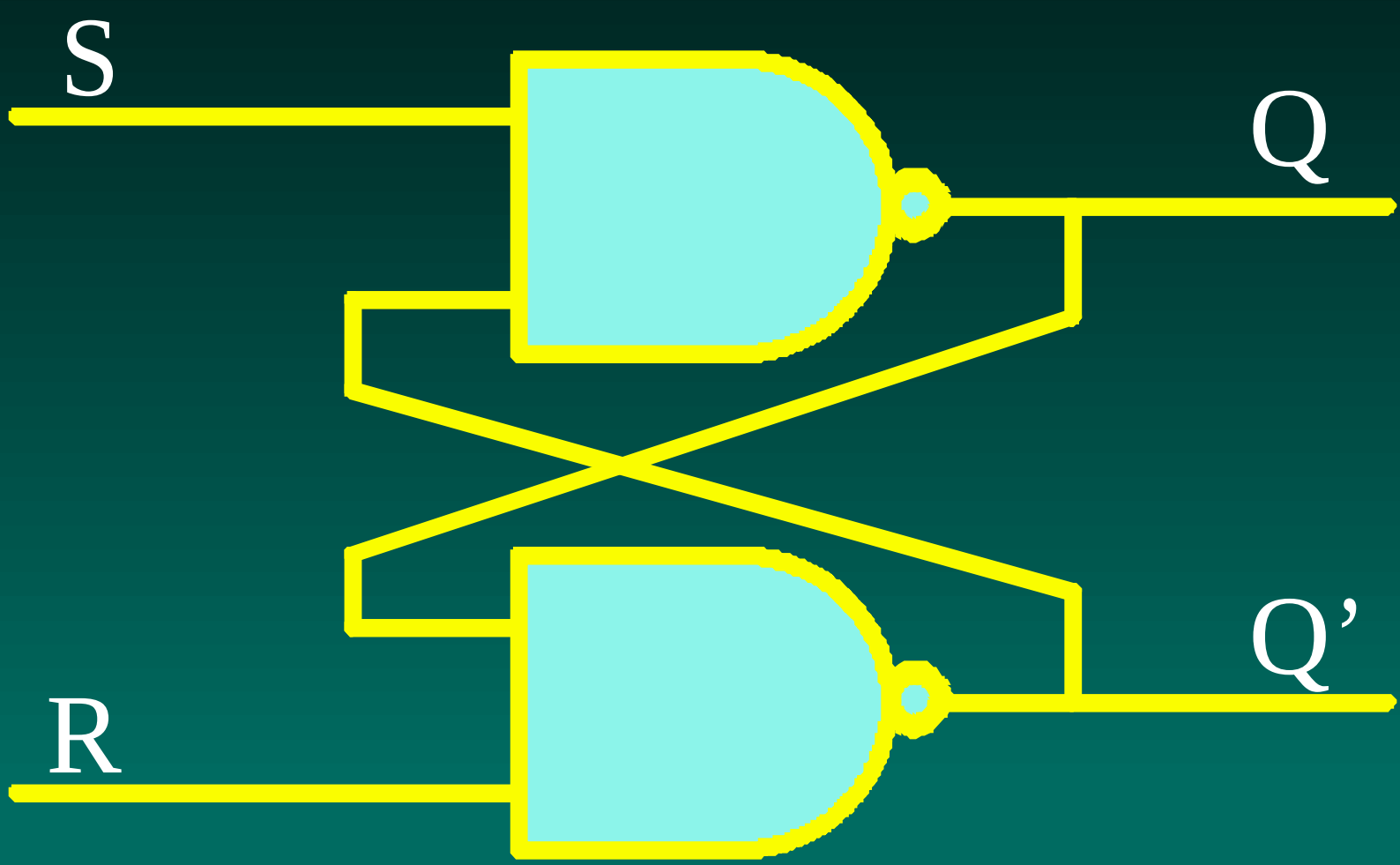


Nand Nor Buffer Xnor

# *Sequential Logic*

- Contains Memory Elements
- Memory is Provided by Feedback
- Circuit diagrams generally have implicit or explicit cycles
- Two Styles: Synchronous and Asynchronous

# *An RS Flip-Flop*



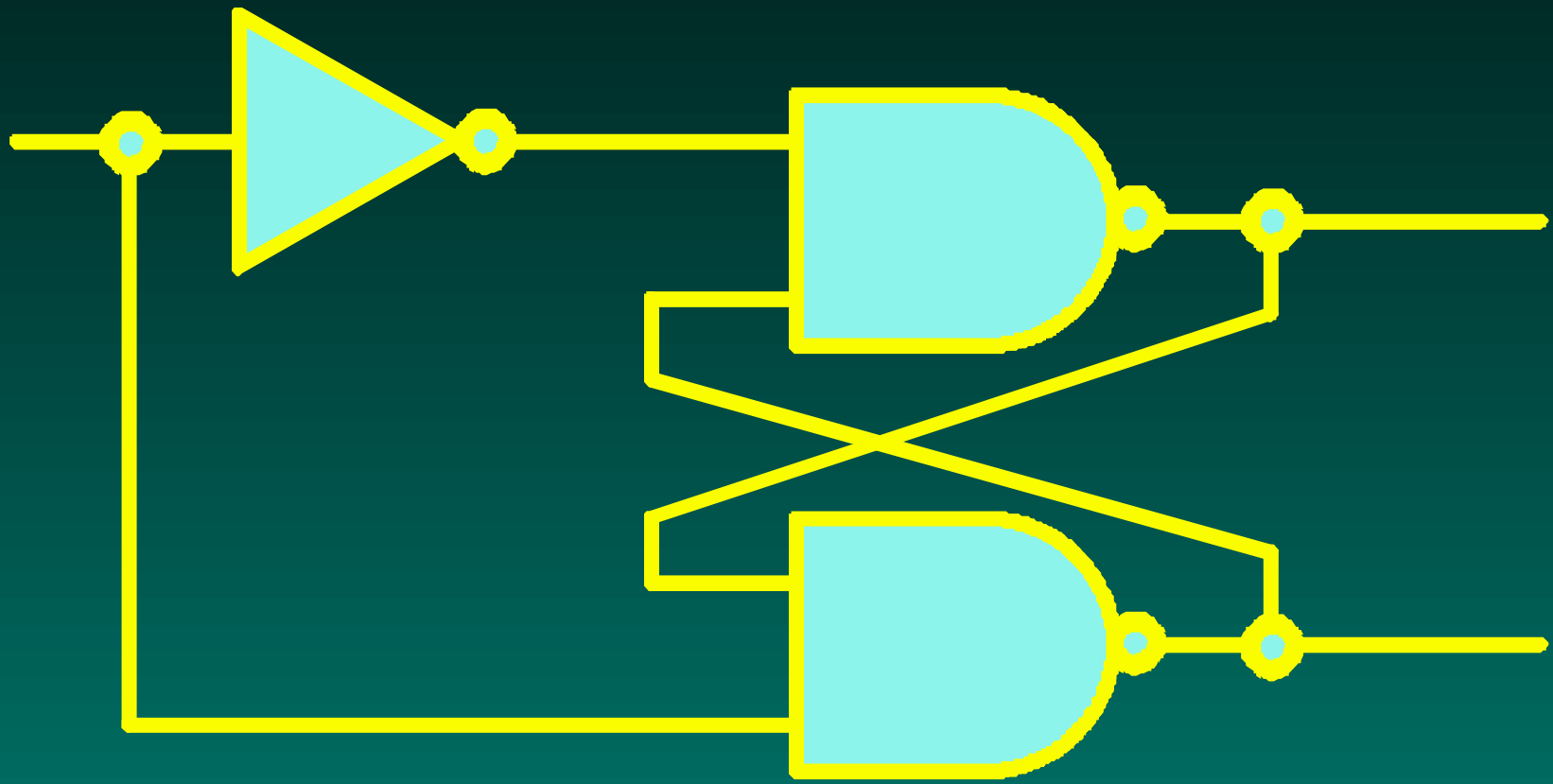
# *RS Characteristics*

- If S=0 and R=1, Q is set to 1, and Q' is reset to 0
- If R=0 and S=1, Q is reset to 0, and Q' is set to 1
- If S=1 and R=1, Q and Q' maintain their previous state.
- If S=0 and R=0, a transision to S=1, R=1 will cause oscillation.

# *Instability*

- RS flip-flops can become unstable if both R and S are set to zero.
- All Sequential elements are fundamentally unstable under certain conditions
  - Invalid Transisions
  - Transisions too close together
  - Transisions at the wrong time

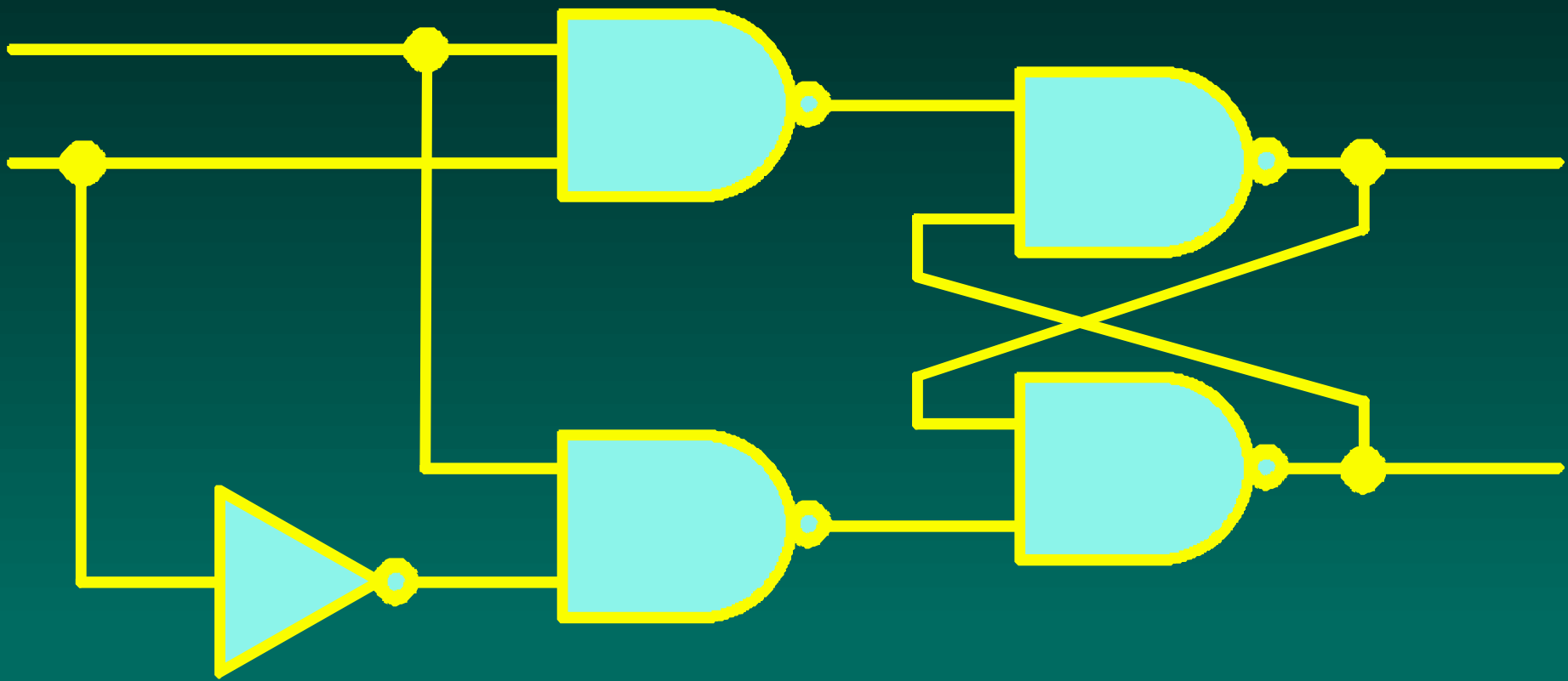
# D Flip-Flops



# *D-Flip Flop Characteristics*

- Avoids the instability of the RS flip-flop
- Retains its last input value
- Formally known as a “Delay” flip-flop
- May become unstable if transisions are too close together
- Is generally implemented as a special circuit, *not* as pictured here.

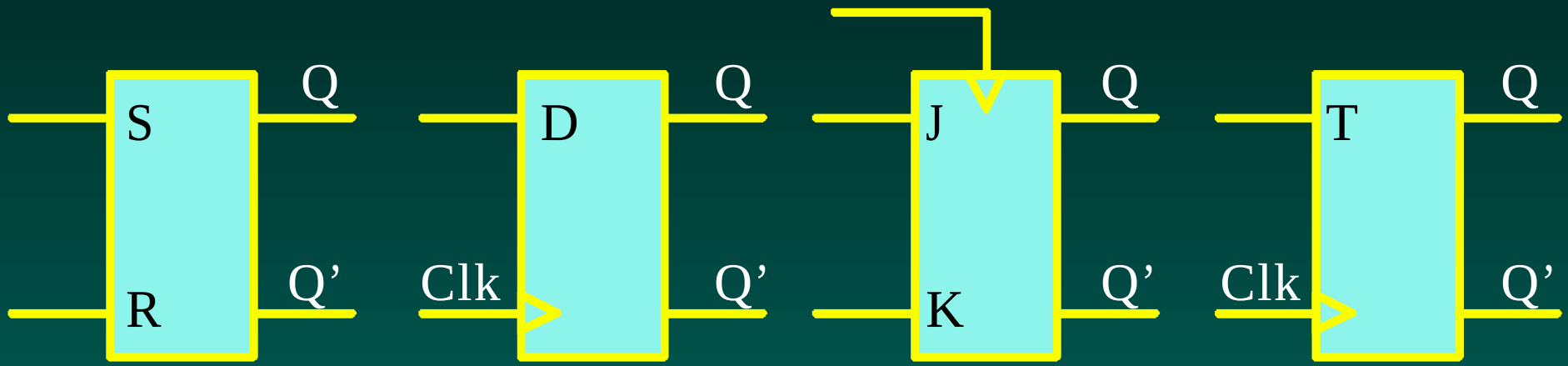
# A Clocked D Flip-Flop



# *Clocked D-Flip Flop Characteristics*

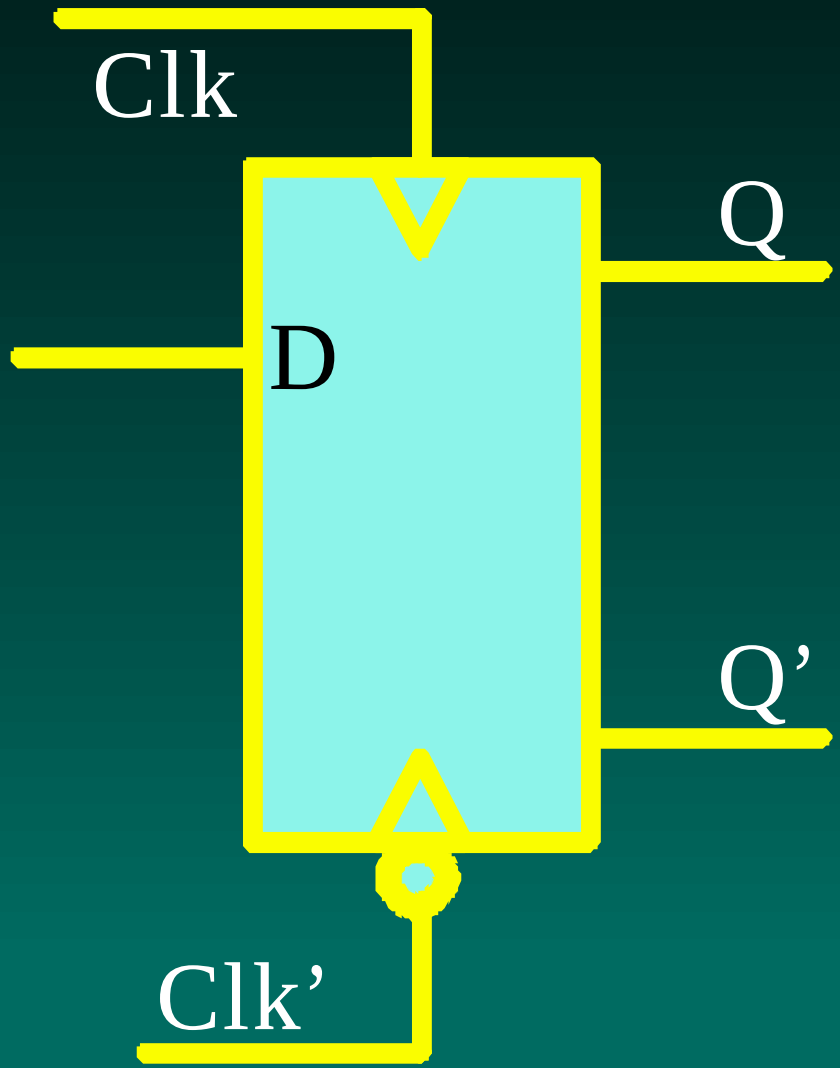
- Synchronizes transisions with a clock
- Input should remain stable while clock is active
- Transision at the wrong time can cause instability
  - Changes while clock is active
  - Changes simultaneous with clock

# *Flip-Flop Symbols*




Flip-Flop Symbols Contain Implicit Feedback Loops

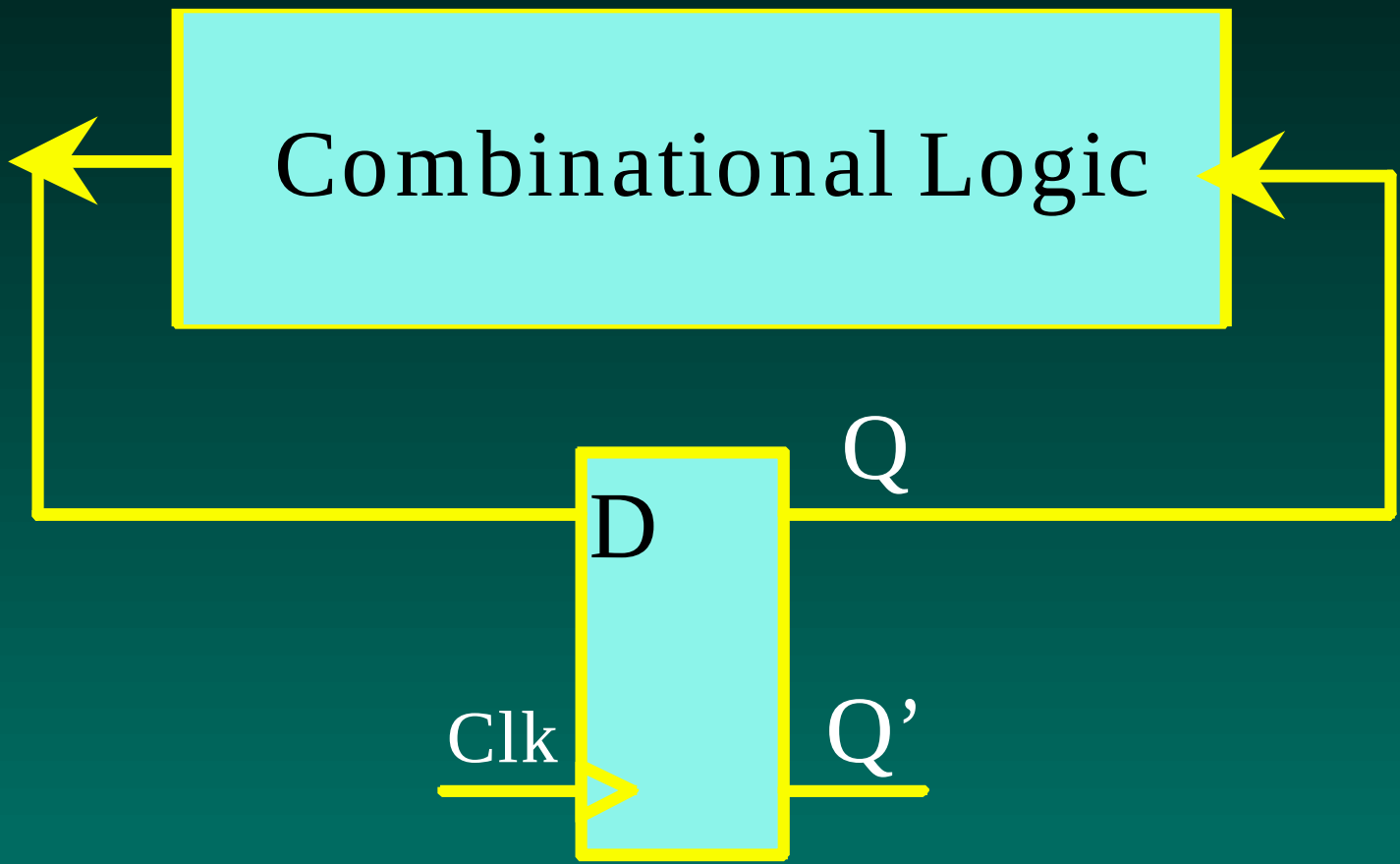
# A CMOS Flip-Flop



# *CMOS Logic Elements*

- CMOS = Complementary MOS
- CMOS Elements Often Require 2 Clocks or 2 Controls
- Clocks or Controls must be Complements of One another
- Clock-Skew (Non-Simultaneous changes in both clocks) can cause problems

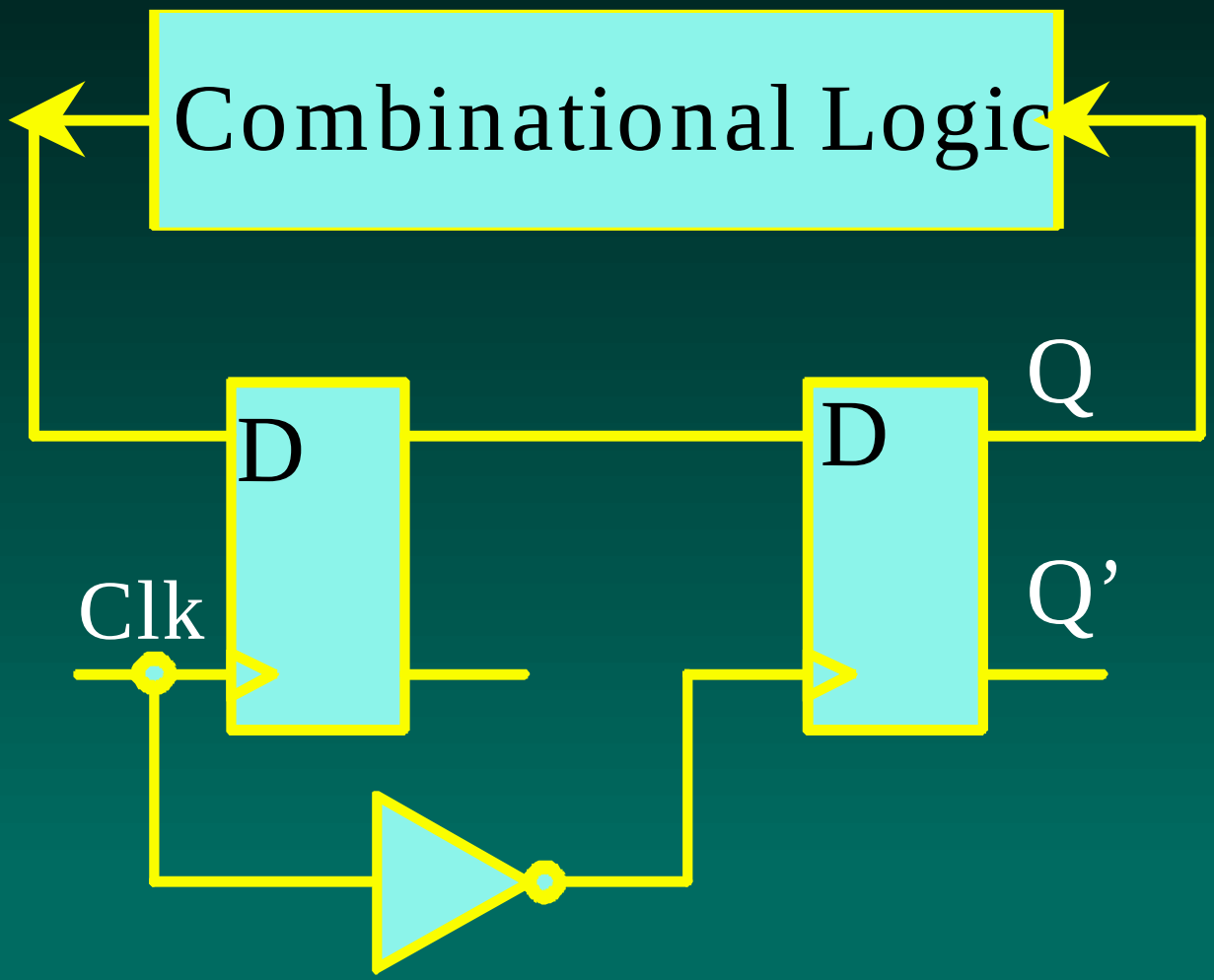
# *An Asynchronous Sequential Circuit*



# *Asynchronous Circuits*

- Combinational Logic is used:
  - To Compute New States
  - To Compute Outputs
- State is maintained in Asynchronous Circuit Elements
- Care must be used to avoid oscillations

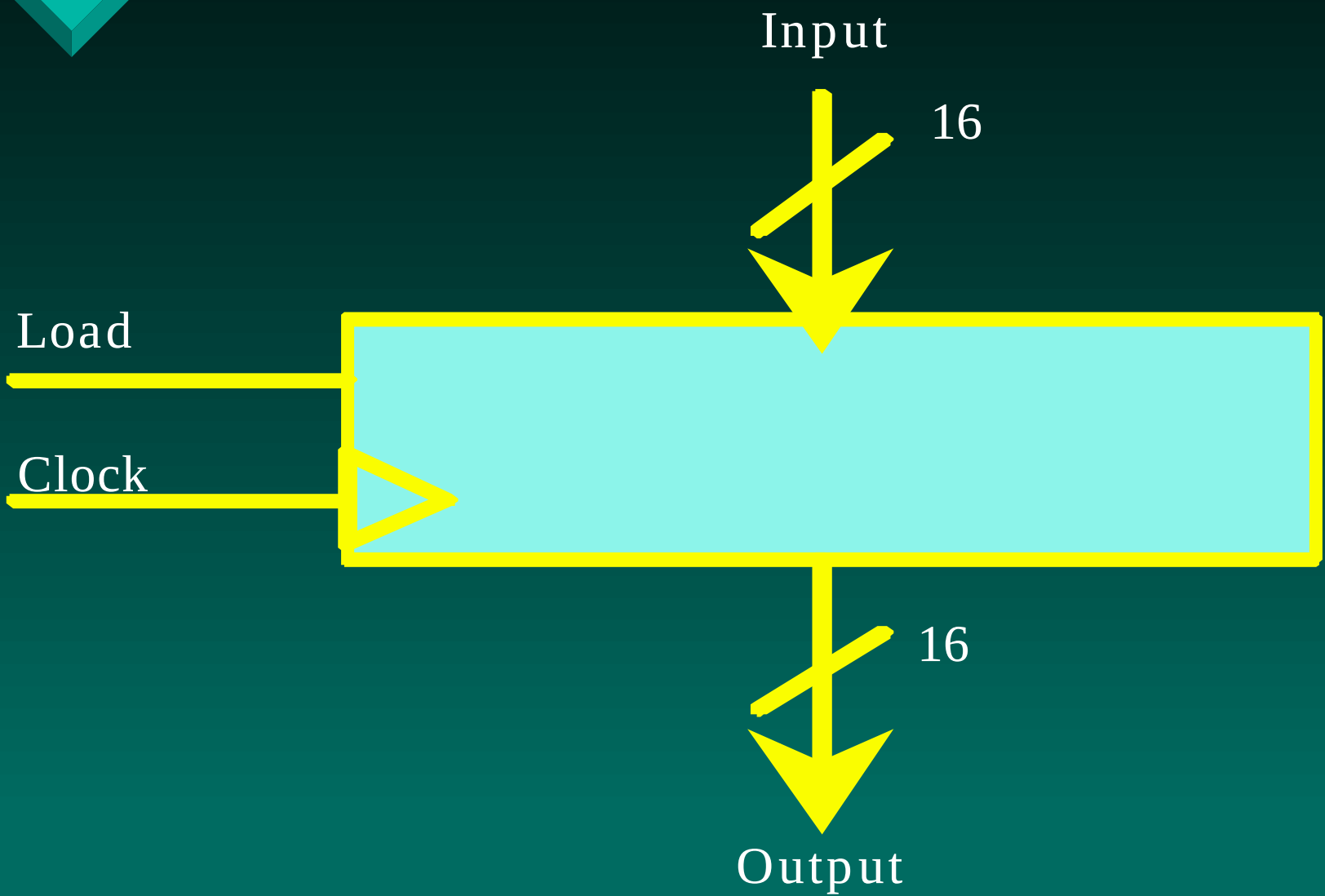
# A Synchronous Sequential Circuit



# *Synchronous Circuits*

- Combinational Logic is used to:
  - Compute New States
  - Compute Outputs
- State is maintained in Synchronous Flip-Flops
- State Changes can be made only when clock changes
- Combinational Logic Must be Stable when Clock is Active

# *Register Symbol*



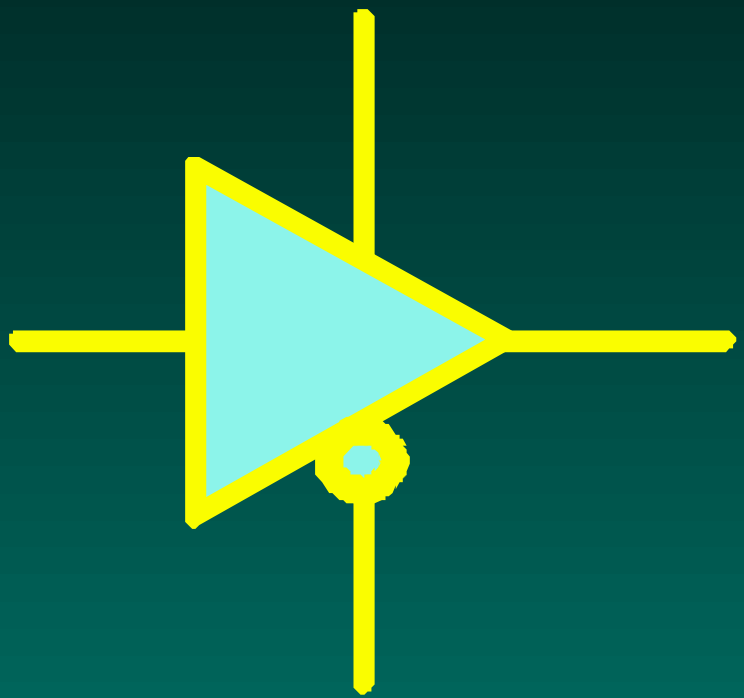
# *Register Issues*

- Generally A Collection of D Flip-Flops
- Can be Synchronous or Asynchronous
- Default is Assumption is Synchronous
- May have internal wiring to:
  - Perform Shifts
  - Set / Clear
  - All-Zero Status Flag

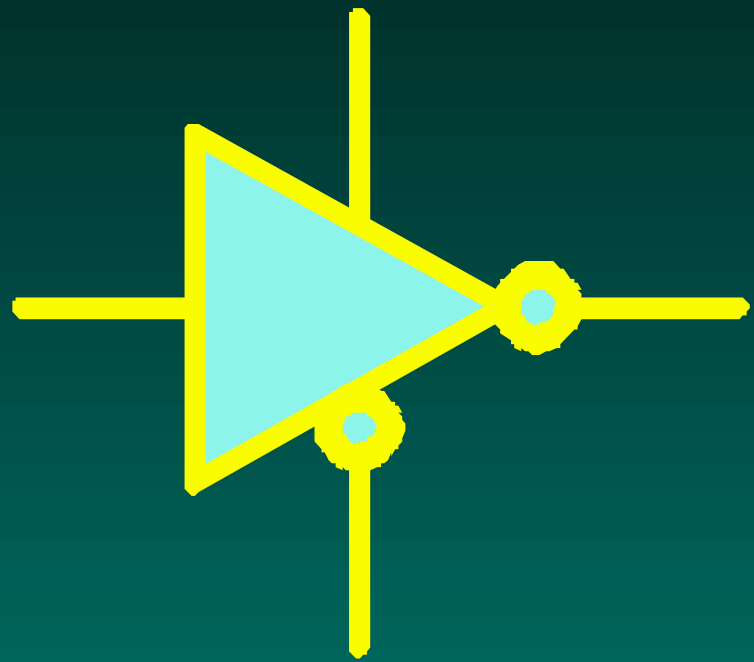
# *Tristate Elements*

- Three States:
  - Zero (Output is grounded)
  - One (Output connected to Power Terminal)
  - High-Impedance (Output Not Connected to Either Power Or Ground)
- Can be Used to Construct Cheap Multiplexors

# *CMOS Tri-state Buffers*



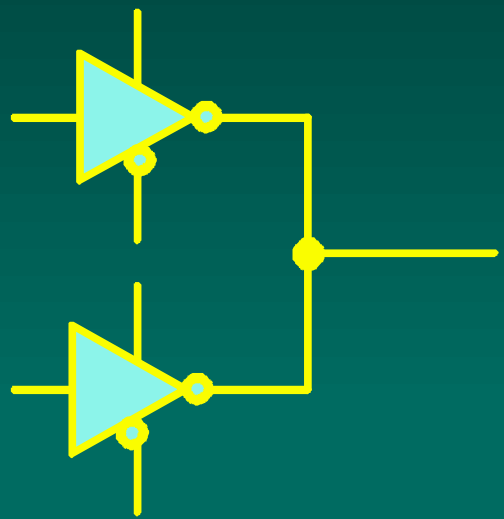
Non-Inverting



Inverting

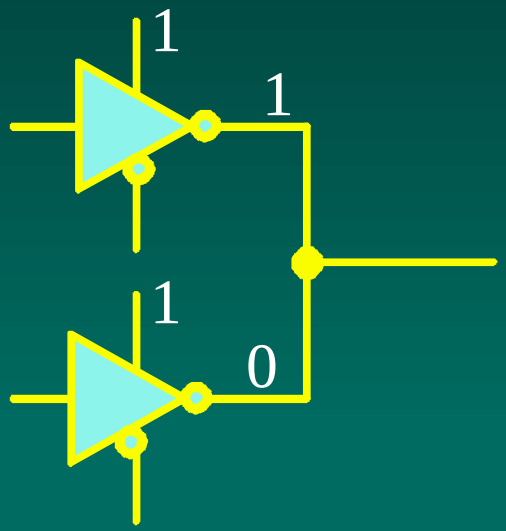
# *Tri-State Buffer Issues*

- The Gate Amplifies its Signal
- May be Inverting or Non-Inverting
- Often used to Construct Multiplexors Using Wired-Or Connections



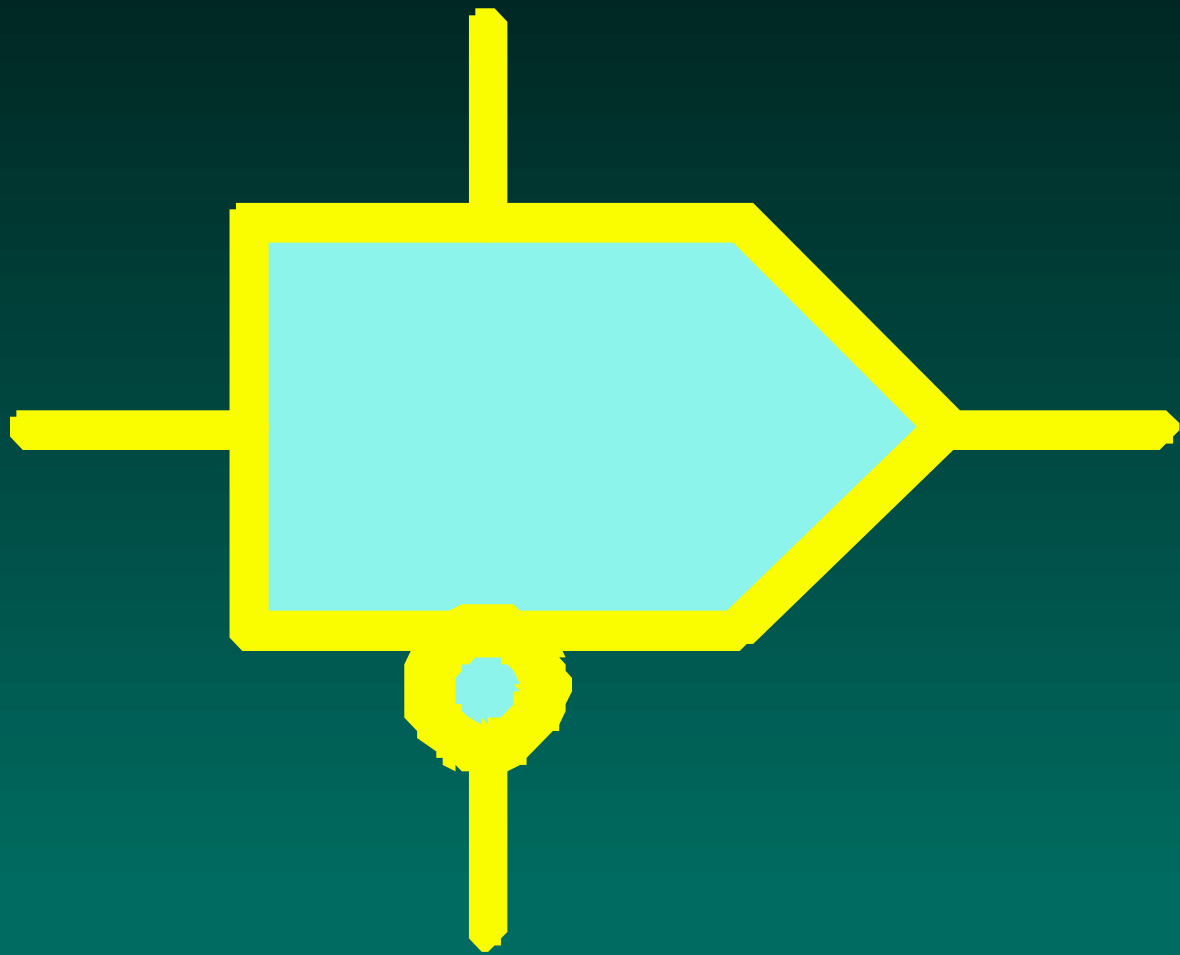
# *More Tri-State Issues*

- In a Wired-Or Connection, Only One Buffer can be in Non-Tristate State
- Violating This Rule Can Destroy The Circuit Due a Power/Ground Short



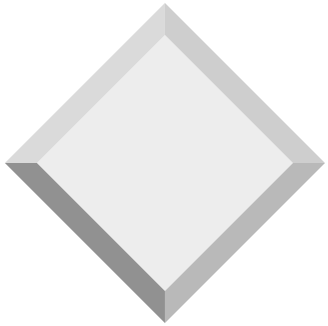

DANGER!

# *The CMOS Transmission Gate*



# *Transmission Gate Issues*

- Similar to Tristate Buffer
- Has No Amplification
- Number of Consecutive Transmission Gates is Limited
- Similar Problems With Wired-Or Connections



# *Logic Design*

A Review

# *Boolean Algebra*

- Two Values: zero and one
- Three Basic Functions: And, Or, Not
- Any Boolean Function Can be Constructed from These Three

| **And** | 0 | 1 |
|---|---|---|
| 0 | **0** | **0** |
| 1 | **0** | **1** |

| **Or** | 0 | 1 |
|---|---|---|
| 0 | **0** | **1** |
| 1 | **1** | **1** |

| **Not** | |
|---|---|
| 0 | **1** |
| 1 | **0** |

# *Algebraic Laws*

| Classification | Law |
|---|---|
| **Identity** | $a1=1a=a$<br>$a+0=0+a=a$ |
| **Dominance** | $a0=0a=0$<br>$1+a=a+1=1$ |
| **Commutativity** | $a+b=b+a$<br>$ab=ba$ |
| **Associativity** | $a(bc)=(ab)c$<br>$a+(b+c)=(a+b)+c$ |
| **Distributive** | $a(b+c)=ab+ac$<br>$a+bc=(a+b)(a+c)$ |
| **Demorgan's Laws** | $(a+b)' = a'b'$<br>$(ab)' = a'+b'$ |

# *Boolean Expressions*

- Addition represents OR
- Multiplication represents AND
- Not is represented by a prime a’ or an overbar $\overline{a}$
- Examples:
- s = a’bc + ab’c + abc’ + a’b’c’
- q = ab + bc + ac + abc

# *Superfluous Terms*

- The following Two Equations Represent The Same Function.

$$q = ab + bc + ac + abc$$

$$q = ab + bc + ac$$

| a | b | c | q |
|---|---|---|---|
| 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 |

# *Prime Implicants*

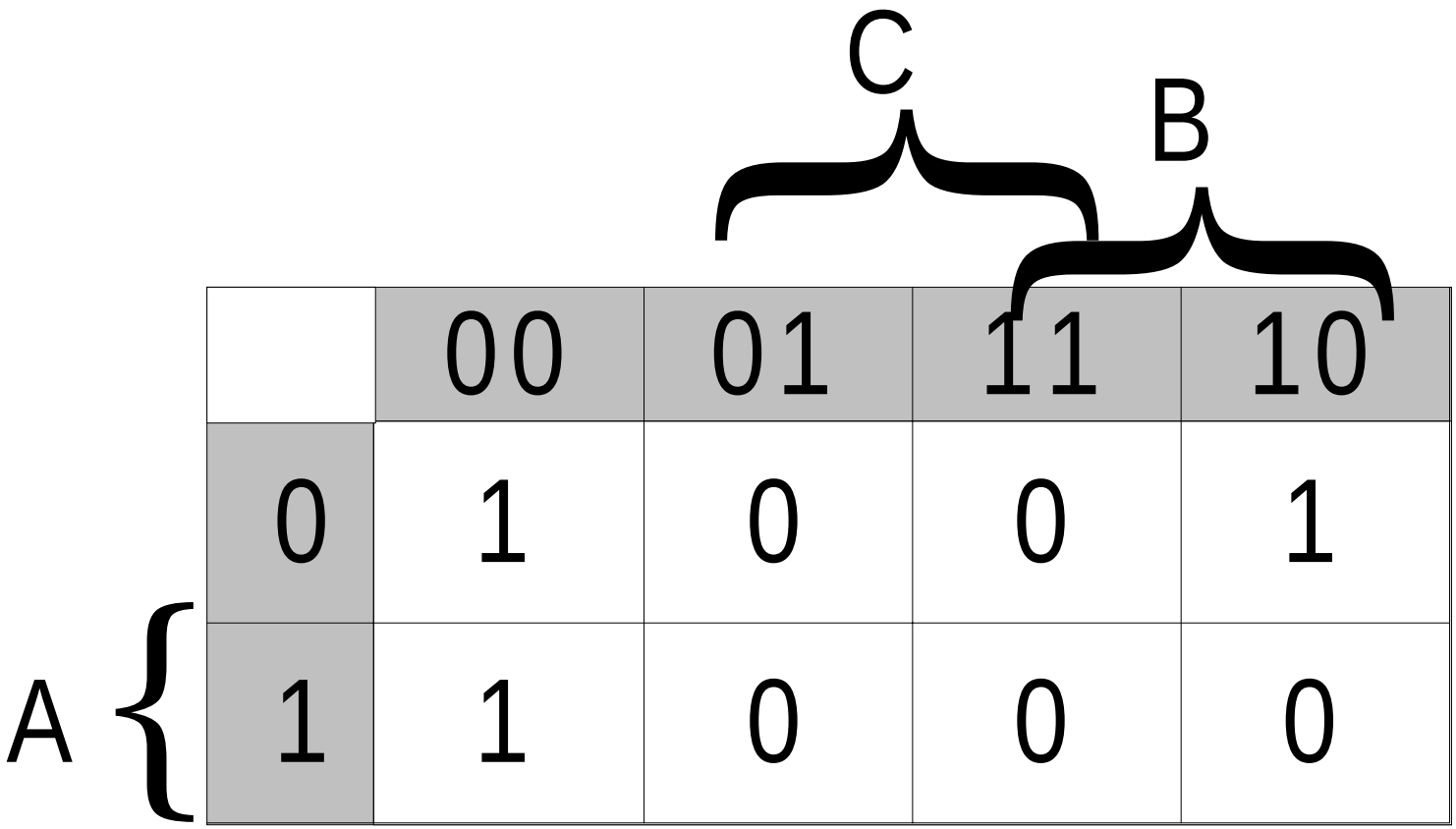
- A Prime Implicant is a Product of Variables or Their Complements, eg. ab'cd'
- If a Prime Implicant has the Value 1, then the Function has the Value 1
- A Minimal Equation is a Sum of Prime Implicants

# *Minimization and Minterms*

- Minimization Reduces the Size and Number of Prime Implicants
- A MinTerm is a Prime Implicant with the Maximum Number of Variables
- For a 3-input Function a'bc is a MinTerm, while ab is not.
- Prime Implicants can be Combined to Eliminate Variables, abc'+abc = ab

# *Minimization with Maps*

- A Karnaugh Map



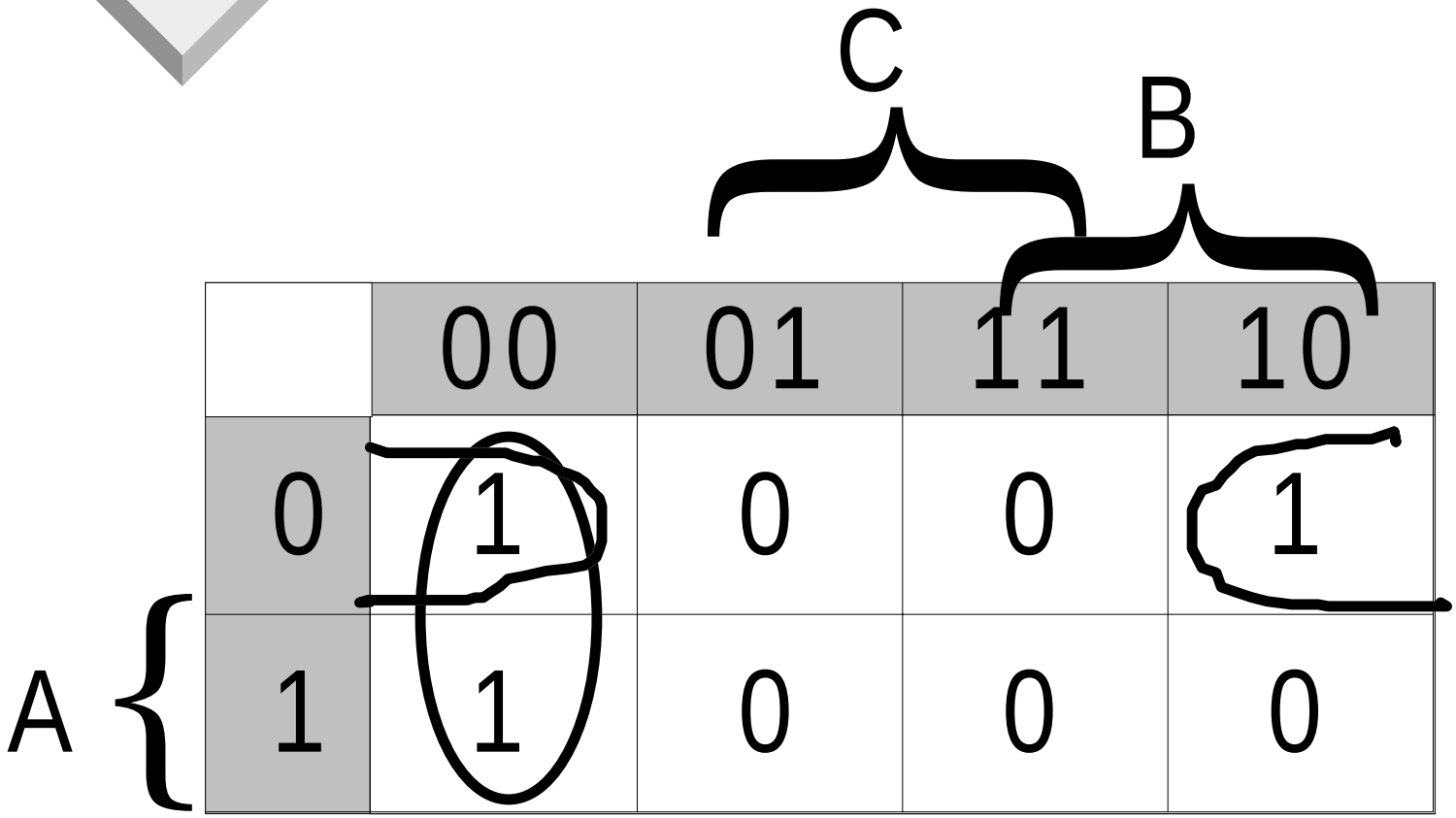
# *Procedure*

- Select Regions Containing All 1's
- Regions should be as Large as Possible
- Regions must contain $2^k$ cells
- Regions should overlap as little as possible
- The complete set of regions must contain all 1's in the map

# *Procedure 2*

- Top and Bottom of Map are Contiguous
- Left and Right of Map are Contiguous
- Regions represent Prime Implicants
- Use Variable name guides to construct equation
  - Completely inside the region of a variable means prime implicant contains variable
  - Completely outside the region of a variable means prime implicant contains negation

# *Applied to Previous Map*

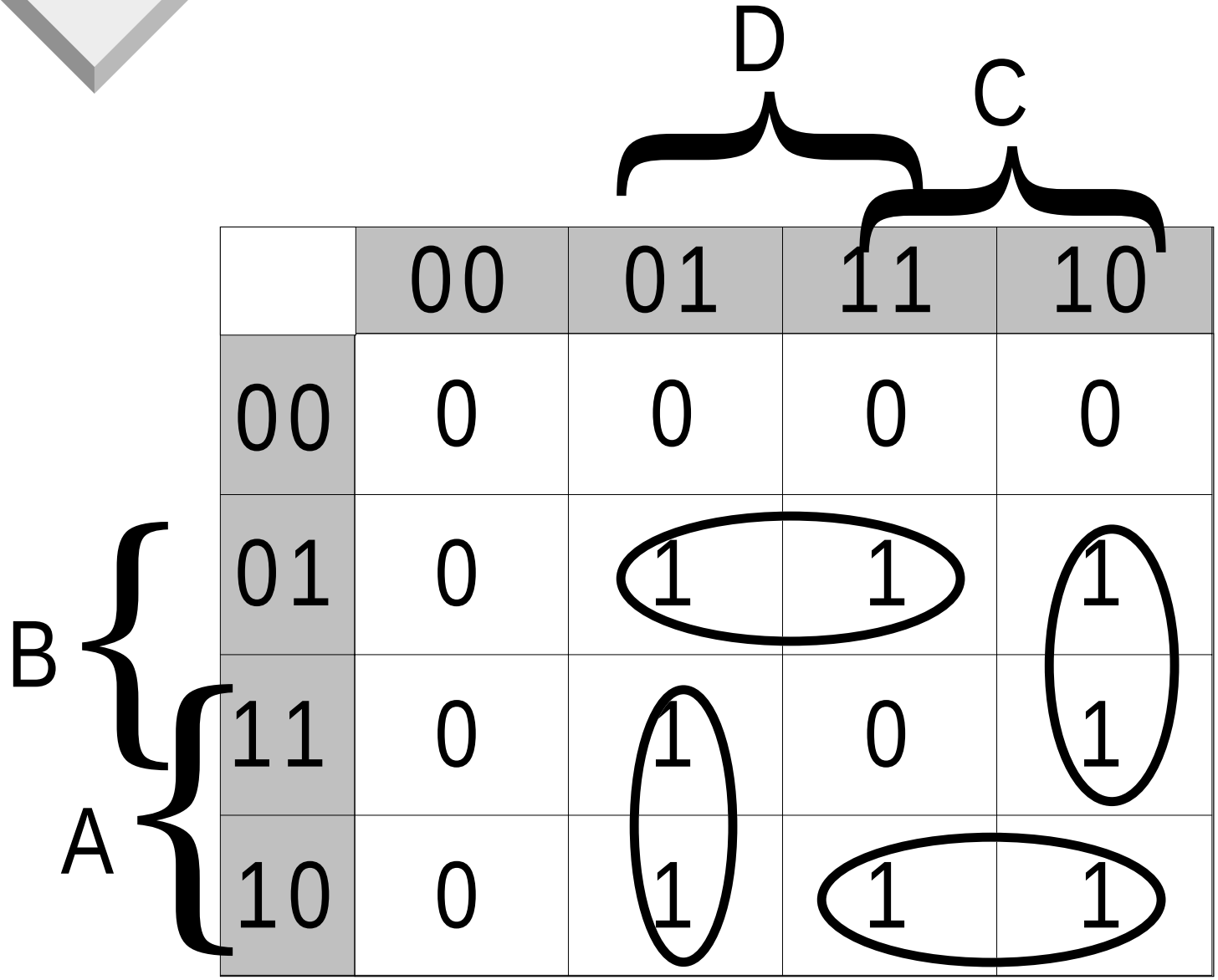



q=c’b’+c’a’

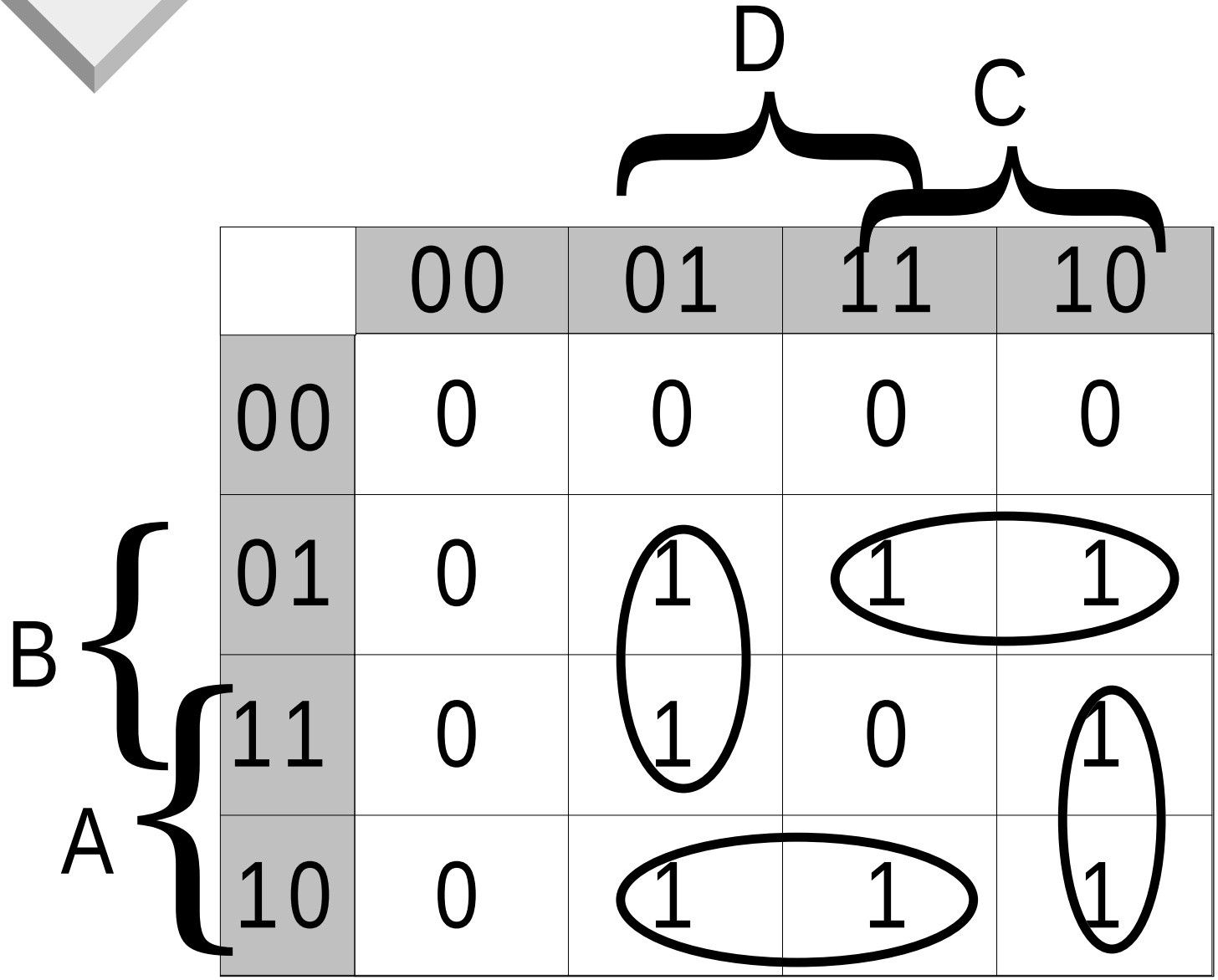
# *A 4-Variable Karnaugh Map*

| B \ A | D \ C: 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 1 | 1 |

# *First Minimization*



# *Second Minimization*



# *Minimal Forms for Previous Slides:*

- $ab'd + bc'd + a'bc + acd'$
- $ac'd + a'bd + bcd' + ab'c$
- Moral: A Boolean Function May Have Several Different Minimal Forms
- Karnaugh Maps are Ineffective for Functions with More than Six Inputs.

# *Quine McClusky Minimization*

- Amenable to Machine Implementation
- Applicable to Circuits with an Arbitrary Number of Inputs
- Effective Procedure for Finding Prime Implicants, but …
- Can Require an Exponential Amount of Time for Some Circuits

# *Quine-McClusky Procedure*

- Start with The Function Truth Table
- Extract All Input Combinations that Produce a TRUE Output (MinTerms)
- Group All MinTerms by The Number of Ones They Contain
- Combine Minterms from Adjacent Groups

# *More Quine-McClusky*

- Two Min-Terms Combine If They Differ by Only One Bit
- The Combined MinTerm has an x in the Differing Position
- Create New Groups From Combined Min-Terms
- Each Member of A New Group Must Have the Same Number of 1's and x's

# *Yet More Quine-McClusky*

- Each Member of A Group Must Have x's in The Same Position.
- Combine Members of the New Groups To Create More New Groups
- Combined Terms Must Differ By One Bit, and Have x's in the Same Positions
- Combine as Much as Possible
- Select Prime Implicants to "Cover" All Ones in the Function

# *Quine-McClusky Example 1*

Numbers in Parentheses are Truth-Table Positions.

| |
|---|
| **0011(3) 1100(12)** |
| **0111(7) 1011(11) 1101(13) 1110(14)** |
| **1111(15)** |

# *Quine-McClusky Example 2*

New Groups After Combining MinTerms

| | |
|---|---|
| 0x11(3,7) | 110x(12,13) |
| 1x11(11,15) | 111x(14,15) |
| x011(3,11) | 11x0(12,14) |
| x111(7,15) | 11x1(13,15) |

# *Quine-McClusky Example 3*

The Final Two Groups

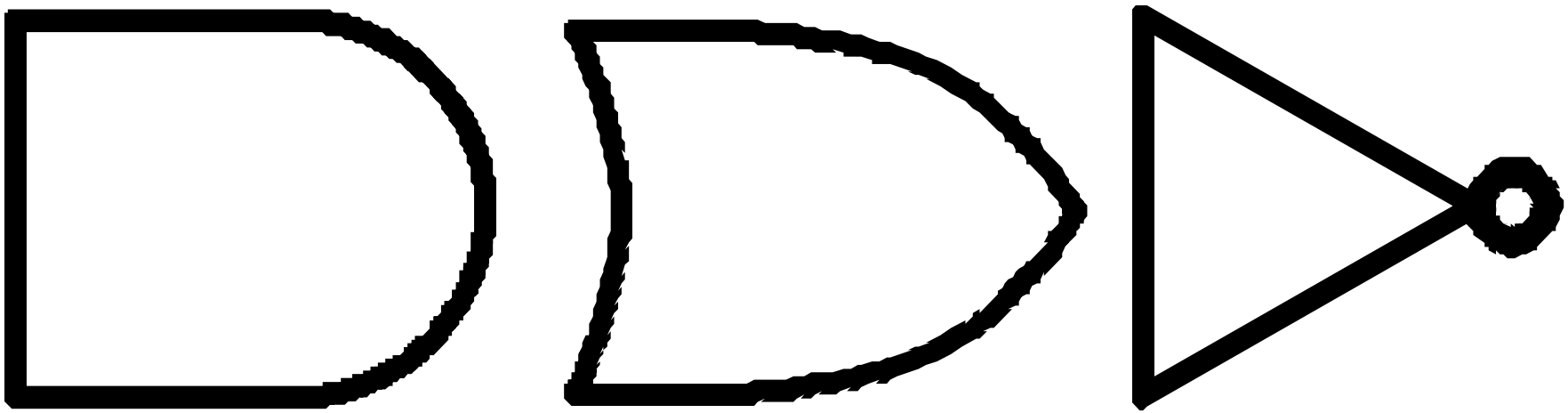
Note That These Two Elements Cover All Truth-Table Positions

| |
|---|
| xx11(3,7,11,15) |
| 11xx(12,13,14,15) |

# *Quine-McClusky Example 4*

- Each Group Element Represents a Prime Implicant
- It is Necessary to Select Group Elements to Cover All Truth-Table Positions.
- In This Case, ab+cd is the Minimal Formula.
- In General, Selecting a Minimal Number of Prime Implicants is NP-Complete

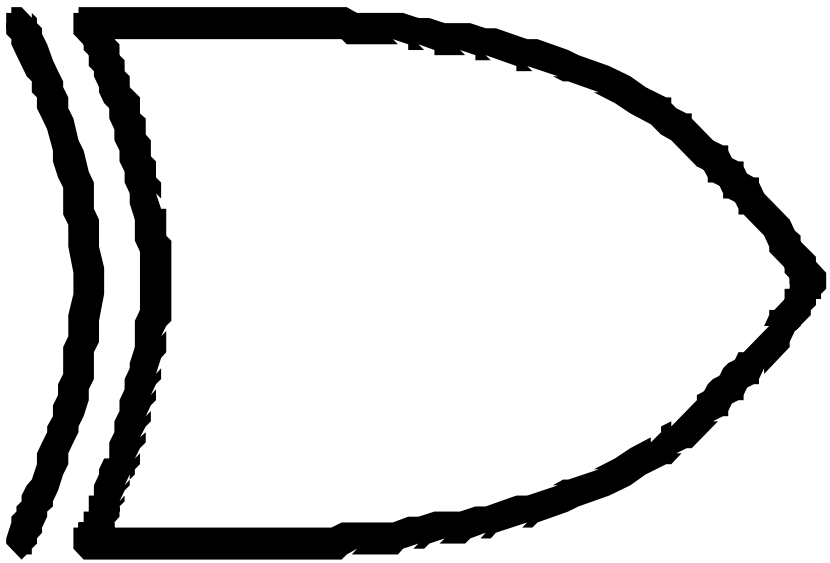
# *Basic Logic Symbols*



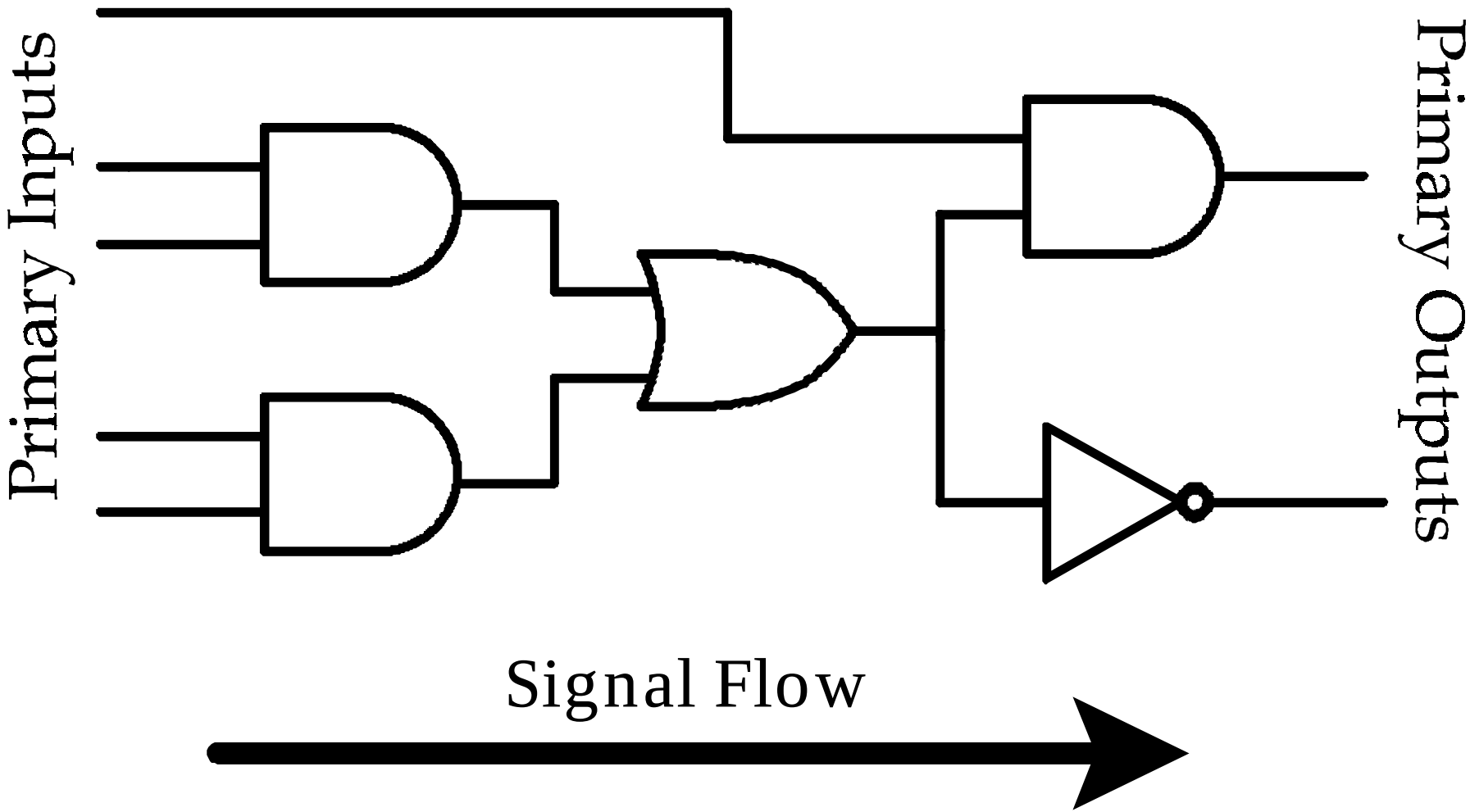
And Or Not

# *The Exclusive Or Function*

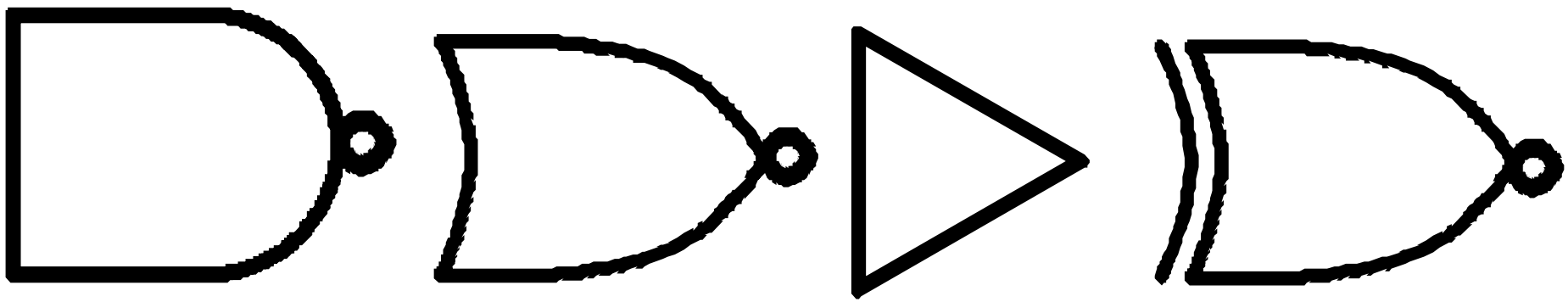
| **Xor** | 0 | 1 |
|---|---|---|
| 0 | **0** | **1** |
| 1 | **1** | **0** |



# *A Simple Logic Diagram*



# *Additional Logic Symbols*

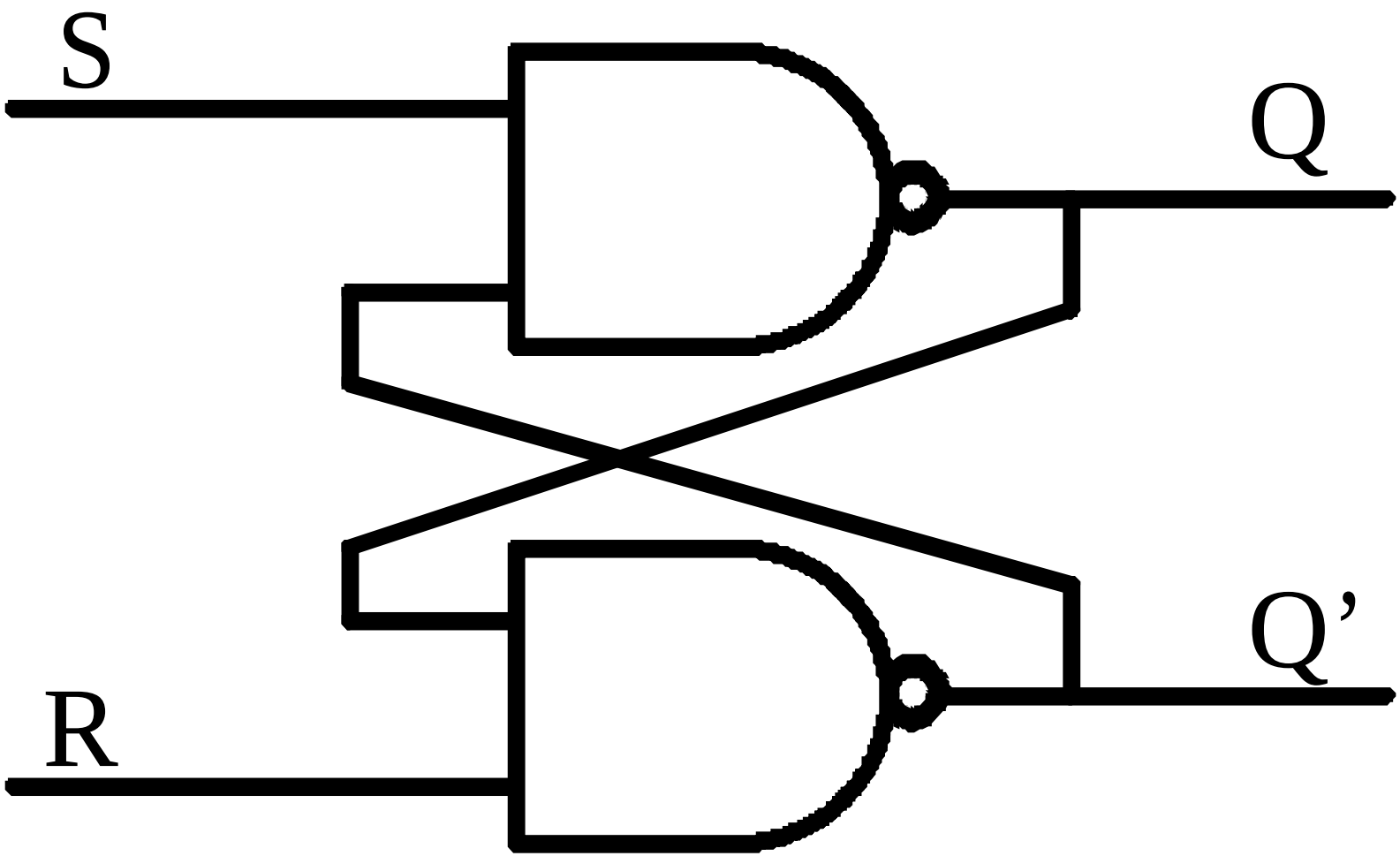


Nand Nor Buffer Xnor

# *Sequential Logic*

- Contains Memory Elements
- Memory is Provided by Feedback
- Circuit diagrams generally have implicit or explicit cycles
- Two Styles: Synchronous and Asynchronous

# *An RS Flip-Flop*



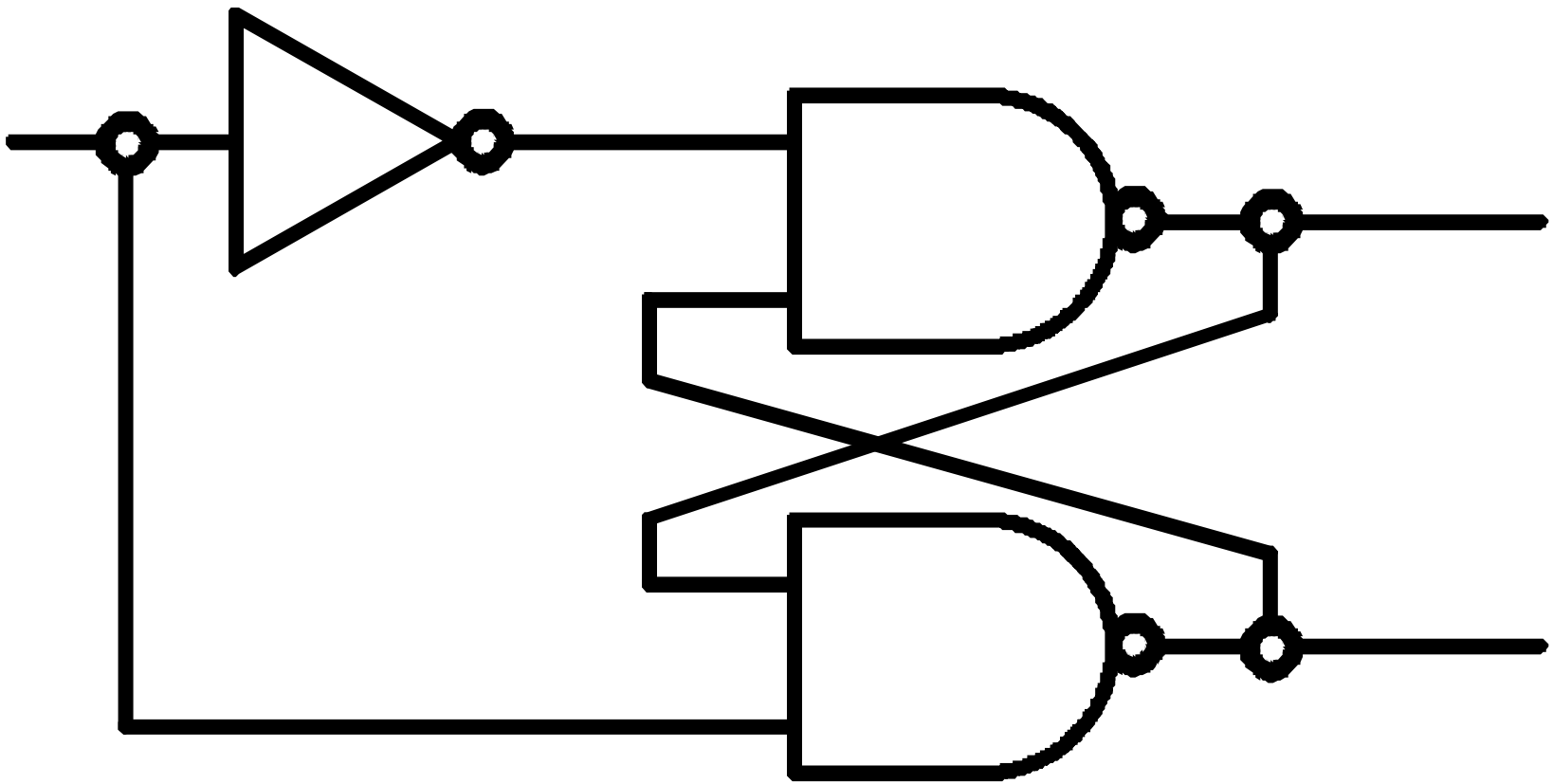
# *RS Characteristics*

- If S=0 and R=1, Q is set to 1, and Q' is reset to 0
- If R=0 and S=1, Q is reset to 0, and Q' is set to 1
- If S=1 and R=1, Q and Q' maintain their previous state.
- If S=0 and R=0, a transision to S=1, R=1 will cause oscillation.

# *Instability*

- RS flip-flops can become unstable if both R and S are set to zero.
- All Sequential elements are fundamentally unstable under certain conditions
  - Invalid Transisions
  - Transisions too close together
  - Transisions at the wrong time

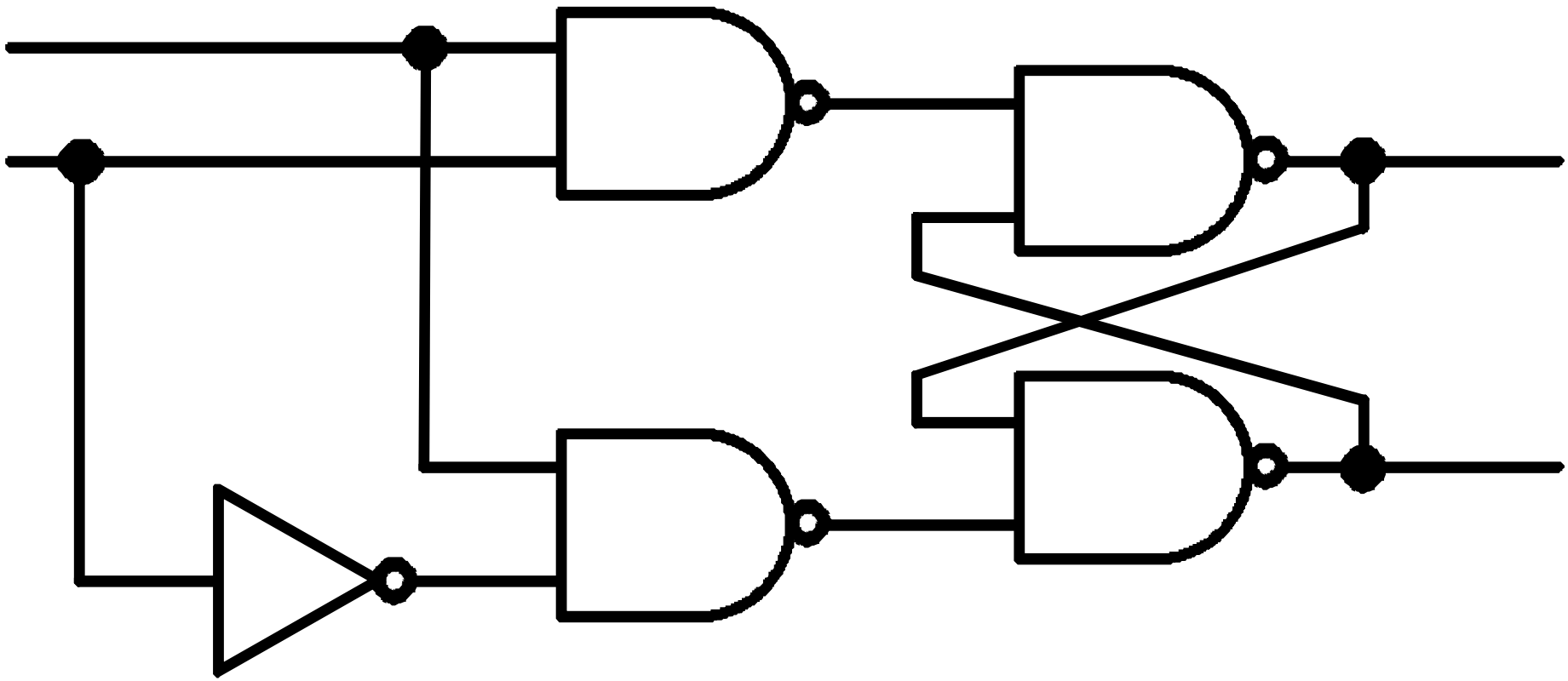
# *D Flip-Flops*



# *D-Flip Flop Characteristics*

- Avoids the instability of the RS flip-flop
- Retains its last input value
- Formally known as a “Delay” flip-flop
- May become unstable if transisions are too close together
- Is generally implemented as a special circuit, *not* as pictured here.

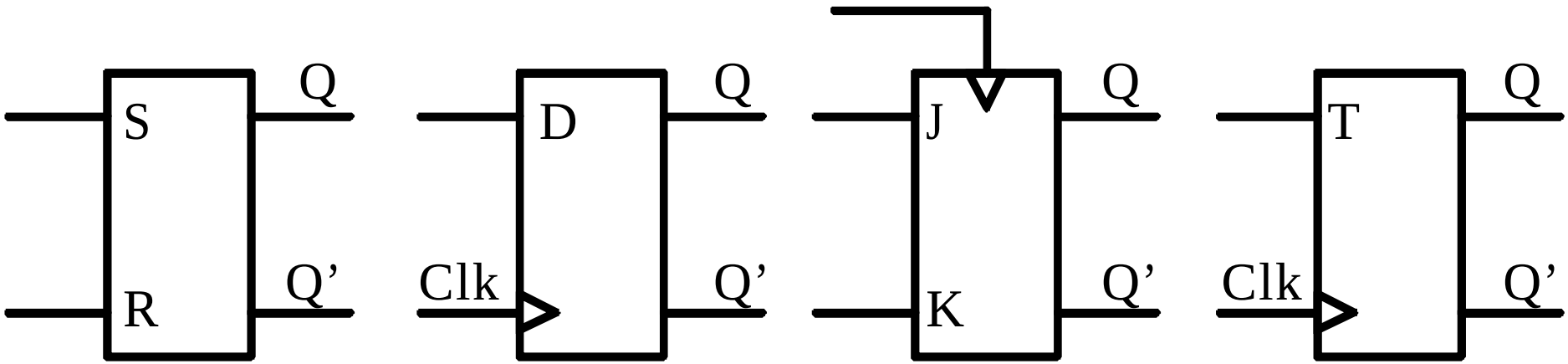
# *A Clocked D Flip-Flop*



# *Clocked D-Flip Flop Characteristics*

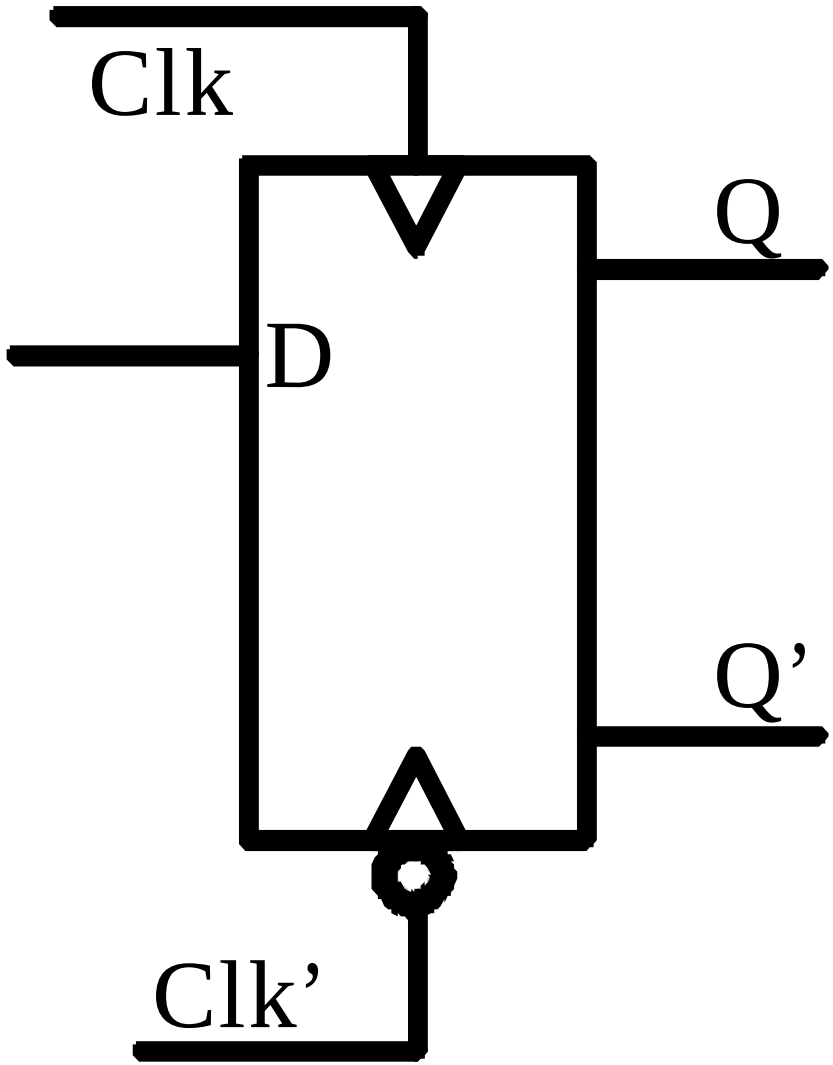
- Synchronizes transisions with a clock
- Input should remain stable while clock is active
- Transision at the wrong time can cause instability
  - Changes while clock is active
  - Changes simultaneous with clock

# *Flip-Flop Symbols*




Flip-Flop Symbols Contain Implicit Feedback Loops

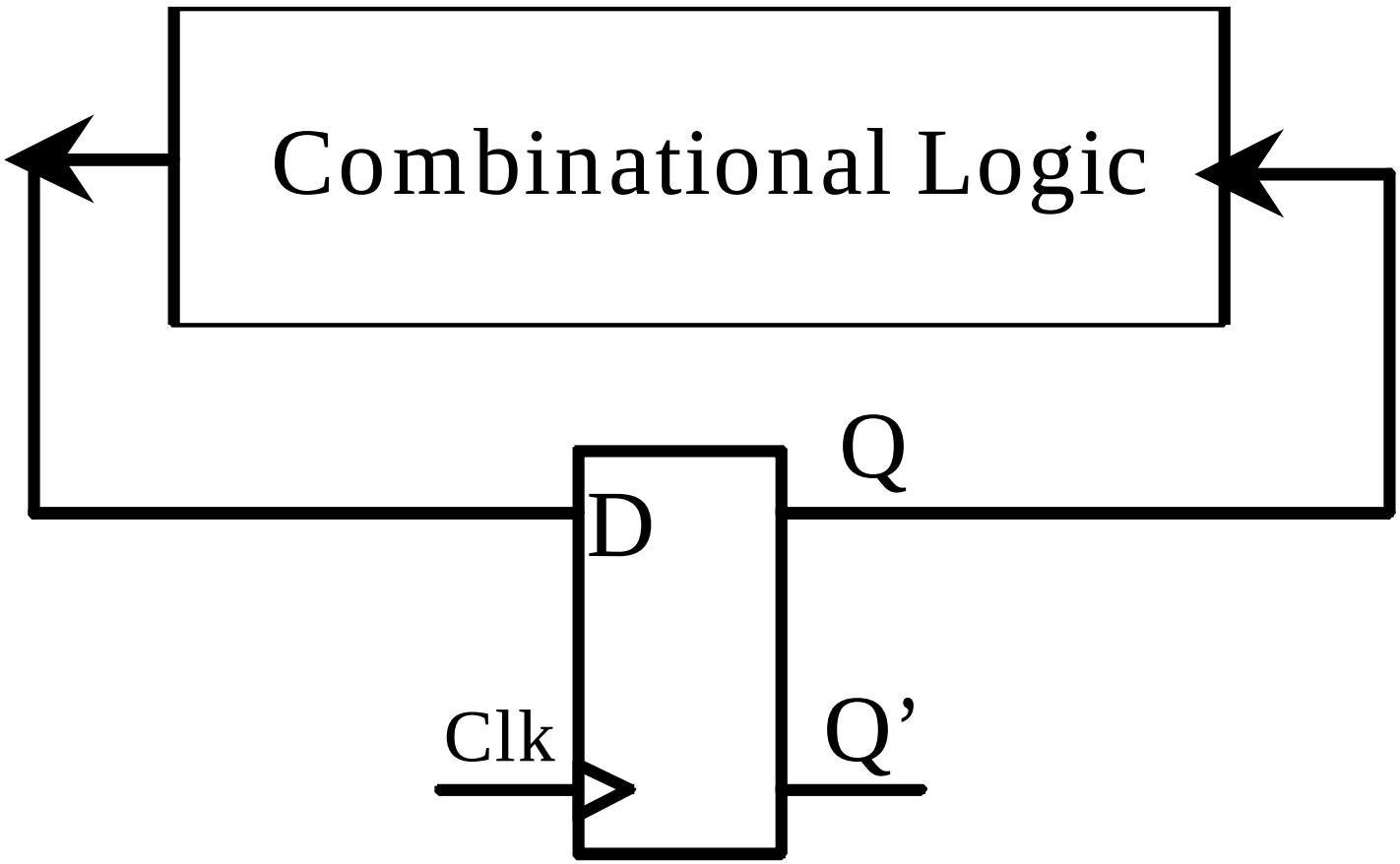
# *A CMOS Flip-Flop*



# *CMOS Logic Elements*

- CMOS = Complementary MOS
- CMOS Elements Often Require 2 Clocks or 2 Controls
- Clocks or Controls must be Complements of One another
- Clock-Skew (Non-Simultaneous changes in both clocks) can cause problems

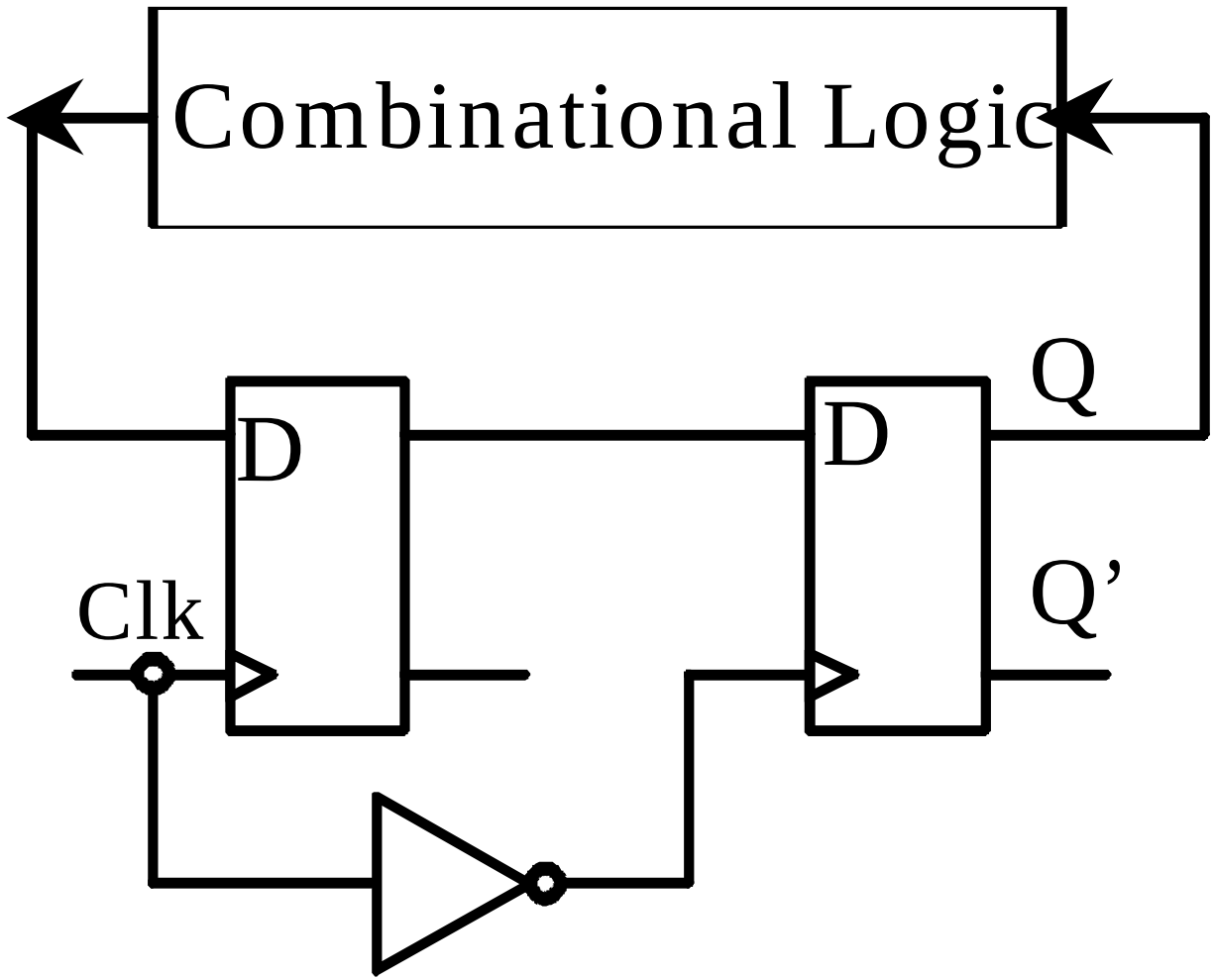
# *An Asynchronous Sequential Circuit*



# *Asynchronous Circuits*

- Combinational Logic is used:
  - To Compute New States
  - To Compute Outputs
- State is maintained in Asynchronous Circuit Elements
- Care must be used to avoid oscillations

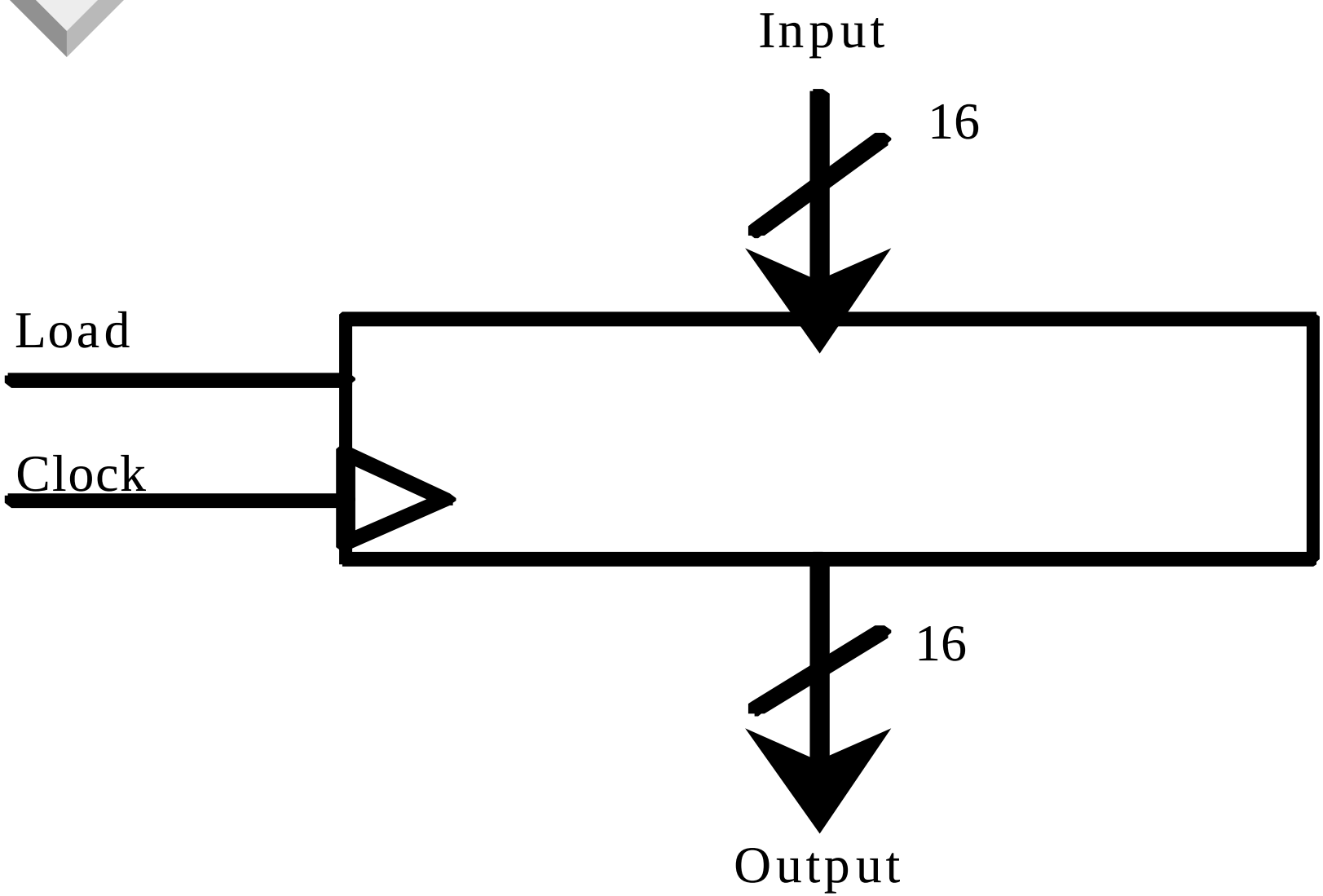
# *A Synchronous Sequential Circuit*



# *Synchronous Circuits*

- Combinational Logic is used to:
  - Compute New States
  - Compute Outputs
- State is maintained in Synchronous Flip-Flops
- State Changes can be made only when clock changes
- Combinational Logic Must be Stable when Clock is Active

# *Register Symbol*



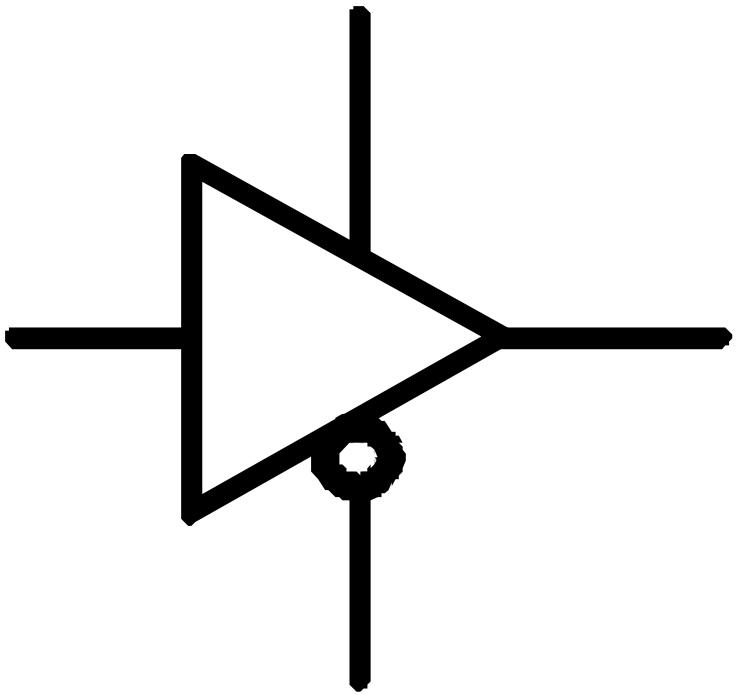
# *Register Issues*

- Generally A Collection of D Flip-Flops
- Can be Synchronous or Asynchronous
- Default is Assumption is Synchronous
- May have internal wiring to:
  - Perform Shifts
  - Set / Clear
  - All-Zero Status Flag

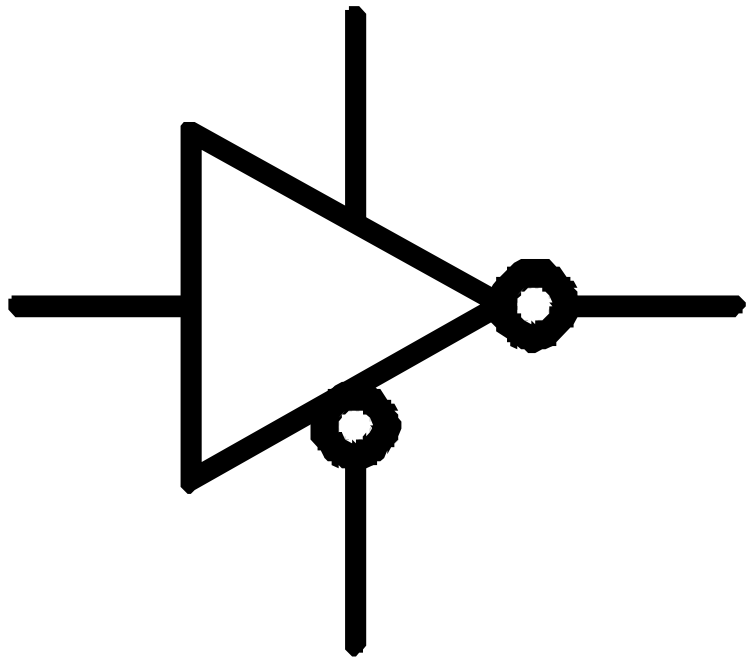
# *Tristate Elements*

- Three States:
  - Zero (Output is grounded)
  - One (Output connected to Power Terminal)
  - High-Impedance (Output Not Connected to Either Power Or Ground)
- Can be Used to Construct Cheap Multiplexors

# *CMOS Tri-state Buffers*



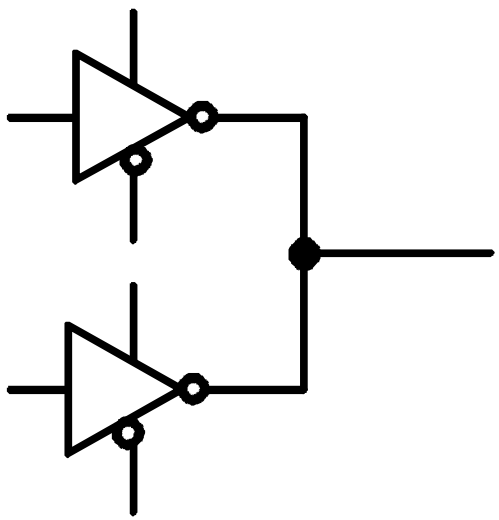
Non-Inverting



Inverting

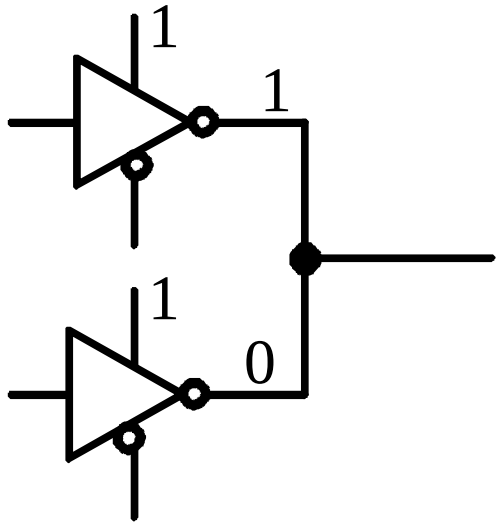
# *Tri-State Buffer Issues*

- The Gate Amplifies its Signal
- May be Inverting or Non-Inverting
- Often used to Construct Multiplexors Using Wired-Or Connections



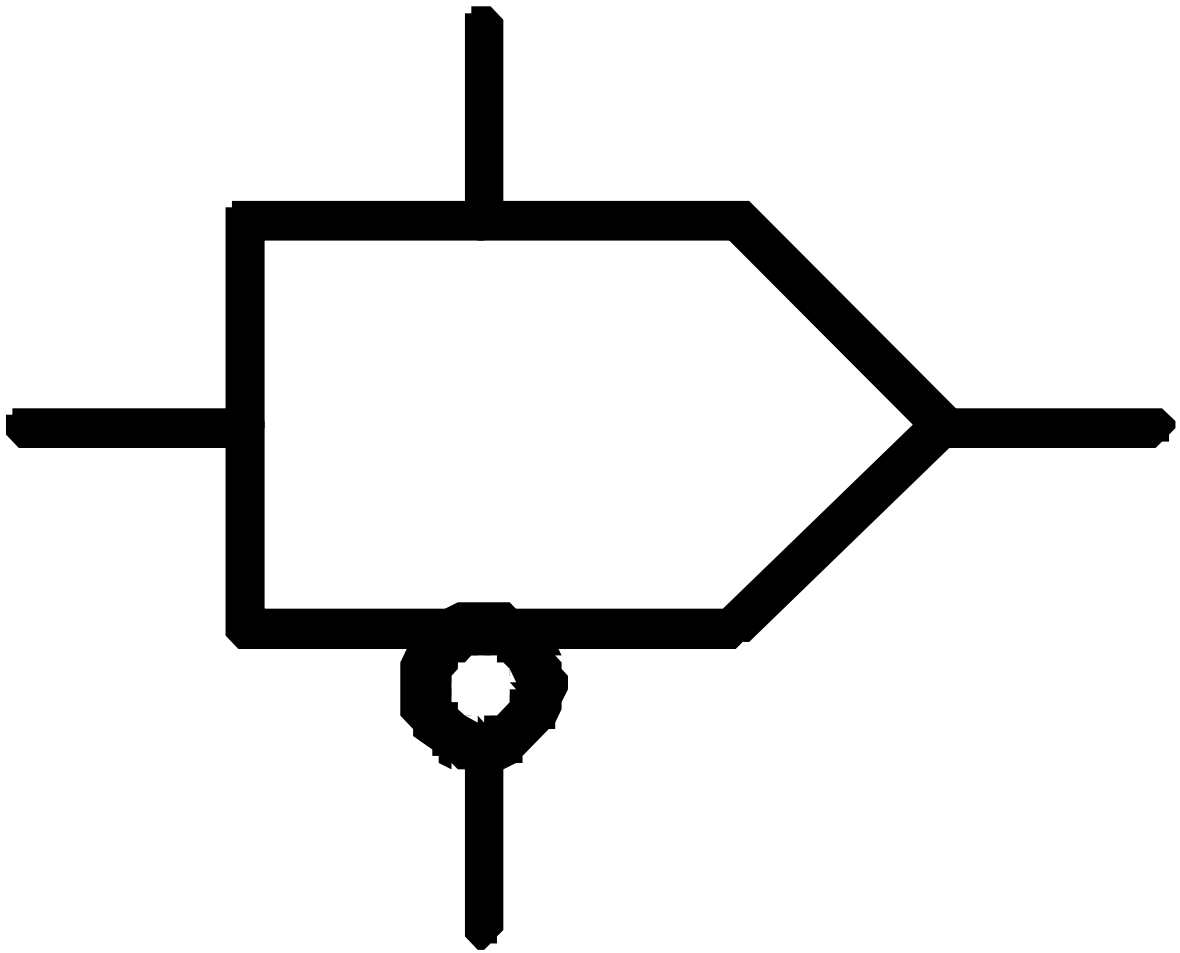
# *More Tri-State Issues*

- In a Wired-Or Connection, Only One Buffer can be in Non-Tristate State
- Violating This Rule Can Destroy The Circuit Due a Power/Ground Short




DANGER!

# *The CMOS Transmission Gate*



# *Transmission Gate Issues*

- Similar to Tristate Buffer
- Has No Amplification
- Number of Consecutive Transmission Gates is Limited
- Similar Problems With Wired-Or Connections