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

<table>
<thead>
<tr>
<th>And</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Or</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
## Algebraic Laws

<table>
<thead>
<tr>
<th>Classification</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identity</strong></td>
<td>$a1 = 1a = a$</td>
</tr>
<tr>
<td></td>
<td>$a + 0 = 0 + a = a$</td>
</tr>
<tr>
<td><strong>Dominance</strong></td>
<td>$a0 = 0a = 0$</td>
</tr>
<tr>
<td></td>
<td>$1 + a = a + 1 = 1$</td>
</tr>
<tr>
<td><strong>Commutativity</strong></td>
<td>$a + b = b + a$</td>
</tr>
<tr>
<td></td>
<td>$ab = ba$</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td>$a(bc) = (ab)c$</td>
</tr>
<tr>
<td></td>
<td>$a + (b + c) = (a + b) + c$</td>
</tr>
<tr>
<td><strong>Distributive</strong></td>
<td>$a(b + c) = ab + ac$</td>
</tr>
<tr>
<td></td>
<td>$a + bc = (a + b)(a + c)$</td>
</tr>
<tr>
<td><strong>Demorgan’s Laws</strong></td>
<td>$(a + b)' = a'b'$</td>
</tr>
<tr>
<td></td>
<td>$(ab)' = a' + b'$</td>
</tr>
</tbody>
</table>
Boolean Expressions

- Addition represents OR
- Multiplication represents AND
- Not is represented by a prime $a'$ or an overbar $\overline{a}$

Examples:
- $s = a'bc + ab'c + abc' + a'b'c'$
- $q = ab + bc + ac + abc$
The following Two Equations Represent The Same Function.

\[ q = ab + bc + ac + abc \]
\[ q = ab + bc + ac \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A

C

B
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
\[ q = c'^b' + c'a' \]
A 4-Variable Karnaugh Map

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### First Minimization

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
## Second Minimization

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Variables: A, B, C, D
Minimal Forms for Previous Slides:

- $ab'c'd + bc'dd + a'bc + ac'd'$
- $ac'd + a'bd + bcd' + ab'c$

- 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
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.

<table>
<thead>
<tr>
<th>0011(3)</th>
<th>1100(12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111(7)</td>
<td>1011(11)</td>
</tr>
<tr>
<td>1110(14)</td>
<td></td>
</tr>
<tr>
<td>1111(15)</td>
<td></td>
</tr>
</tbody>
</table>
Quine-McClusky Example 2

New Groups After Combining MinTerms

<table>
<thead>
<tr>
<th>0x11(3,7)</th>
<th>110x(12,13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x11(11,15)</td>
<td>111x(14,15)</td>
</tr>
<tr>
<td>x011(3,11)</td>
<td>11x0(12,14)</td>
</tr>
<tr>
<td>x111(7,15)</td>
<td>11x1(13,15)</td>
</tr>
</tbody>
</table>
Quine-McClusky Example 3

The Final Two Groups

Note That These Two Elements Cover All Truth-Table Positions

\[
\begin{array}{c}
xx11(3,7,11,15) \\
11xx(12,13,14,15)
\end{array}
\]
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

<table>
<thead>
<tr>
<th>Xor</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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

\[ S \]
\[ Q \]
\[ Q' \]
\[ R \]
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 transition 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 Transitions
  - Transitions too close together
  - Transitions 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 transitions 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 transitions 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

Combinational Logic

D Q

Clk Q'

D

Q

Clk

Q'
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

Combinational Logic

D

Q

D

Q'

Clk

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

Input

Load

16

Clock

16

Output
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

<table>
<thead>
<tr>
<th>And</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Or</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
# Algebraic Laws

<table>
<thead>
<tr>
<th>Classification</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identity</strong></td>
<td>( a1 = 1a = a )</td>
</tr>
<tr>
<td></td>
<td>( a + 0 = 0 + a = a )</td>
</tr>
<tr>
<td><strong>Dominance</strong></td>
<td>( a0 = 0a = 0 )</td>
</tr>
<tr>
<td></td>
<td>( 1 + a = a + 1 = 1 )</td>
</tr>
<tr>
<td><strong>Commutativity</strong></td>
<td>( a + b = b + a )</td>
</tr>
<tr>
<td></td>
<td>( ab = ba )</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td>( a(bc) = (ab)c )</td>
</tr>
<tr>
<td></td>
<td>( a + (b + c) = (a + b) + c )</td>
</tr>
<tr>
<td><strong>Distributive</strong></td>
<td>( a(b + c) = ab + ac )</td>
</tr>
<tr>
<td></td>
<td>( a + bc = (a + b)(a + c) )</td>
</tr>
<tr>
<td><strong>Demorgan’s Laws</strong></td>
<td>((a + b)' = a'b' )</td>
</tr>
<tr>
<td></td>
<td>((ab)' = a' + b' )</td>
</tr>
</tbody>
</table>
Boolean Expressions

- Addition represents OR
- Multiplication represents AND
- Not is represented by a prime $a'$ or an overbar $\overline{a}$
- Examples:
  - $s = a'bc + ab'c + abc' + a'b'c'$
  - $q = ab + bc + ac + abc$
The following Two Equations Represent The Same Function.

\[ q = ab + bc + ac + abc \]
\[ q = ab + bc + ac \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A \{ 0, 1 \}

C \{ 00, 01 \}

B \{ 01, 10 \}
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

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### First Minimization

**Table:**

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>00</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>01</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>11</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Diagrams:**

- **A**
  - 00
  - 01
  - 11
  - 10

- **B**
  - 00
  - 01
  - 11
  - 10

- **C**
  - 00
  - 01
  - 11
  - 10

- **D**
  - 00
  - 01
  - 11
  - 10
### Second Minimization

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **A**: {00, 01, 10, 11}
- **B**: {00, 01}
- **C**: {11, 10}
- **D**: {00, 01, 10}
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.

<table>
<thead>
<tr>
<th>0011(3)</th>
<th>1100(12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111(7)</td>
<td>1011(11)</td>
</tr>
<tr>
<td>1110(14)</td>
<td></td>
</tr>
<tr>
<td>1111(15)</td>
<td></td>
</tr>
</tbody>
</table>
### Quine-McClusky Example 2

#### New Groups After Combining MinTerms

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11(3,7)</td>
<td>110x(12,13)</td>
<td>1x11(11,15)</td>
<td>111x(14,15)</td>
</tr>
<tr>
<td>x011(3,11)</td>
<td>11x0(12,14)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x111(7,15)</td>
<td>11x1(13,15)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quine-McClusky Example 3

The Final Two Groups

Note That These Two Elements Cover All Truth-Table Positions

\[
\begin{array}{c}
\text{xx}11(3,7,11,15) \\
11\text{xx}(12,13,14,15)
\end{array}
\]
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

<table>
<thead>
<tr>
<th>Xor</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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 transition to $S=1$, $R=1$ will cause oscillation.
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 transitions 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 transitions with a clock
- Input should remain stable while clock is active
- Transition at the wrong time can cause instability
  - Changes while clock is active
  - Changes simultaneous with clock
Flip-Flop Symbols Contain Implicit Feedback Loops
A CMOS Flip-Flop

Clk

D

Q

Q'

Clk'

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

Combinational Logic

D

Q

Q'

Clk
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

Combinational Logic

D

D

Clk

Q

Q'

Q'
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

- Load
- Clock
- Input 16
- Output 16
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