Overview

• Functions and languages
• Designing circuits from functions
• Minterms and the DNF
• Maxterms and CNF
• Circuit complexity
• Algebra of Boolean expressions
• Illustration using the Full Adder
• Dual rail logic
Mathematical Preliminaries

• The Cartesian product $A \times B$ of two sets is the set of pairs, the first from $A$, the second from $B$.
  – If $A = \{0,1\}$, $B = \{2,3\}$,
  – $A \times B = \{02, 03, 12, 13\}$

• The $n$-fold Cartesian product of a set $A$ with itself is denoted $A^n$. E.g. $A^3 = \{0,1\}^3 = \{000,001,010,011, \ldots\}$

• If $a, b$ are in $A$, $a \cdot b$ denotes the string $ab$ in $A \times B$. The operation $\cdot$ is called concatenation.

• The empty string $\epsilon$ satisfies $a \cdot \epsilon = a$, $\epsilon \cdot a = a$. 
Mathematical Preliminaries

• The Kleene star $A^* = \{\epsilon\} \cup A \cup A^2 \cup A^3 \cup ...$, that is, the set of strings of all lengths including the empty string.
  
  $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, ...\}$
Languages

• A **language** is a set of strings over an **alphabet**.
  – A language $L$ over the alphabet $A$ satisfies $L \subseteq A^*$, that is, $L$ is a subset of $A^*$.

• **Examples of languages**
  – The set of 10-bit binary strings with an odd number of 1s is a subset of $\{0,1\}^{10}$.
  – The strings that form acceptable Pascal programs.
Boolean Functions

• A Boolean function – $f : \{0,1\}^k \rightarrow \{0,1\}$
  – $f$ has $k$ inputs and one output.
  – $f(x_1, x_2, \ldots, x_k)$ is the value of $f$ on inputs $x_1, x_2, \ldots, x_k$

• Standard Boolean functions:
  – AND on two binary inputs, $f_{\text{AND}}(x,y) = x \land y$
  – OR on two binary inputs, $f_{\text{OR}}(x,y) = x \lor y$
  – EXOR on two binary inputs, $f_{\text{EXOR}}(x,y) = x \oplus y$
  – NOT on one binary input, $f_{\text{NOT}}(x) = \overline{x}$ or $\neg x$
Languages and Functions

• Let $f : \{0,1\}^k \rightarrow \{0,1\}$

• The $k$-tuples $x$ for which $f(x) = 1$ is a language
  – They form a subset of the strings in $\{0,1\}^k$.

• Given a language $L \subseteq A^*$, there is a natural characteristic function $f_L : A^* \rightarrow \{0,1\}$ associated with the language $L$.
  – If $x$ is in $L$, $f_L(x) = 1$.
  – If $x$ is not in $L$, $f_L(x) = 0$.
Logic Circuits and Functions

• A logic circuit is a directed acyclic graph (DAG) in which node labels are Boolean functions.
  • \( g_7 := g_5 \lor g_6 \);
  • \( g_5 := g_1 \land g_4 \);
  • \( g_6 := g_3 \land g_2 \);
  • \( g_4 := \neg g_2 \);
  • \( g_3 := \neg g_1 \);
  • \( g_1 := x \); \( g_2 = y \)
  • The output function is \( g_7 = x \oplus y \)
Straight-Line Programs

• A *straight-line program (SLP)* is a program that has no branching or looping.
  – An SLP can use arithmetic or Boolean operations

• A *Boolean SLP* is an SLP in which the operations are Boolean.

• The *graph of a Boolean SLP* is a circuit.
  – The graph is directed and acyclic (a *dag*)
Boolean Straight-Line Program

• (1 READ x)
• (2 READ y)
• (3 NOT 1)
• (4 NOT 2)
• (5 AND 1 4)
• (6 AND 3 2)
• (7 OR 5 6)

This SLP computes the Exclusive OR of x and y:

\[ g_7 = x \oplus y = (x \land \neg y) \lor (\neg x \land y) \]
Computing Multi-Output Functions

- Computes \( f : \{0,1\}^3 \to \{0,1\}^2, f(x,y,z) = (c,s) \)
  - This Full Adder uses AND, OR, XOR

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

⊕ is Exclusive OR
Designing Circuits From Functions

• Consider **binary** function $g : \{0,1\}^n \rightarrow \{0,1\}^m$ with $m$ outputs. It can be viewed as $m$ functions, that is, $g = (f_1,f_2, \ldots, f_m)$ where $f_j : \{0,1\}^n \rightarrow \{0,1\}$ are Boolean functions.

• We can realize each $f_j$ separately by a circuit or find common sub-circuits to reduce number of gates (operations).
Designing Circuits From Functions

- Start by realizing one-output functions $f(x_1,\ldots,x_n)$
- A literal is a variable ($x_j$) or its complement (NOT) ($\overline{x}_j$)
- A minterm is the AND of one literal for each variable of a function.
  - E.g. $\overline{x}_1 \land x_2 \land \cdots \land \overline{x}_n$
- A minterm = 1 exactly when each literal = 1.
  - E.g. $x_1 = 0$, $x_2 = 1$, ..., $x_n = 0$. 
Designing Circuits From Functions

• Construct circuit for $f : \{0,1\}^n \rightarrow \{0,1\}$ by forming the OR of those minterms in the inputs to $f$ for which $f$ has value 1.
  - E.g. $c = \overline{x}yz \lor x\overline{y}z \lor xy\overline{z} \lor xyz$

• Minterm expansions can often be simplified.
  - E.g. $c = xy \lor xz \lor yz$

• Tables clear (but long) way to show functional equivalence.

\[
\begin{array}{cccc}
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
\end{array}
\]
Disjunctive Normal Form

• The minterm expansion is also called the disjunctive normal form (DNF).
  – “Disjunct” is an old word for OR.

**Definition:** The DNF is the OR of minterms corresponding to the inputs for which a Boolean function has value 1.

• Why does a formula compute the function?
Maxterms

• The **maxterm** of a Boolean function is the OR of one literal for each variable of the function.
  – E.g. $\overline{x} \lor \overline{y} \lor z$

• A maxterm is 1 **except** when all literals are 0.
  – E.g. The above maxterm is 0 when $x = 1$, $y = 1$, and $z = 0$. 

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</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>
Conjunctive Normal Form

• The conjunctive normal form (CNF) of function $f$ is the AND of the maxterms for which $f = 0$.
  — “Conjunct” is an old word for AND.
CNF Example

• The CNF of $f$ is 1 except where maxterms = 0.
  – E.g. $c = (x \lor y \lor z) \land (x \lor y \lor \overline{z}) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor y \lor z)$

• The CNF can often be simplified.
  – E.g. $c = (x \lor y) \land (x \lor z) \land (y \lor z)$
    (=1 if $\geq 2$ 1s in input)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</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>
Circuit Complexity

• Given a binary function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ its \textbf{circuit size}, denoted $C_\Omega(f)$, is the size of the smallest circuit (fewest gates) drawn from the basis $\Omega$.

• A basis $\Omega$ is \textbf{complete} if every binary function can be realized by a circuit with gates from $\Omega$.
  
  -- $\Omega = \{\text{AND, OR, NOT}\}$ is complete.
  
  -- $\Omega = \{\text{AND, EXOR}\}$ and $\Omega = \{\text{NAND}\}$ are complete.
More on Bases

• For complete bases the circuit size changes by a multiplicative factor when the basis changes.
  – Why is this statement true?

• Monotone functions can be realized over the basis \{\text{AND, OR}\}, which is not complete.
  – The value of a monotone function \{0,1\} does not decrease if inputs are increased from 0 to 1.

• Adding NOT can greatly reduce circuit size of a monotone function.
The Algebra of Boolean Expressions

• Commutativity
  - \( x \lor y = y \lor x \),
  - \( x \land y = y \land x \)

• Distributivity
  - \( x \land (y \lor z) = (x \land y) \lor (x \land z) \) (AND distributes over OR)
  - \( x \lor (y \land z) = (x \lor y) \land (x \lor z) \) (OR distributes over AND)

• Example
  - \((a \land b \land c) \lor (a \land b \land \overline{c}) = (a \land b) \land (c \lor \overline{c}) = (a \land b)\)
  - Treat \( a \land b \) as one variable, apply first distributivity rule

• Absorption Rules
  - \( x \lor x = x \), \( x \lor \overline{x} = 1 \)
  - \( x \land x = x \), \( x \land \overline{x} = 0 \)
The Algebra of Boolean Expressions

• DeMorgan’s Rules
  - \( x \lor y = \overline{x} \land \overline{y} \)
  - \( x \land y = \overline{x} \lor \overline{y} \)

• Exclusive OR expansion (= 1 if 1 or 3 inputs = 1)
  - \( x \oplus y \oplus z = (\overline{x} \land (y \oplus z)) \lor (x \land (y \oplus z)) \)
  - \( = (\overline{x} \land ((\overline{y} \land z) \lor (y \land \overline{z}))) \lor (x \land ((\overline{y} \land z) \lor (y \land \overline{z}))) \)
  - \( (\overline{y} \land z) \lor (y \land \overline{z})) = (\overline{y} \land z) \land (y \land \overline{z}) = (y \land z) \lor (\overline{y} \land \overline{z}) \)
  - \( x \oplus y \oplus z = (x \land y \land z) \lor (\overline{x} \land \overline{y} \land z) \lor (x \land \overline{y} \land \overline{z}) \lor (\overline{x} \land y \land \overline{z}) \)

• Exclusive OR definition
  - \( x \oplus y = (\overline{x} \land y) \lor (x \land \overline{y}) \)
Full Adder

- \( s = x \oplus y \oplus z \)
- \( c = = x y \lor x z \lor y z \)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The formulas use 7 binary operations. The above circuit uses only 5!
Implication Boolean Operator

- Implication is written $x \rightarrow y$. It is interpreted, if $x$ is True, then $y$ is True. The truth table for $z = x \rightarrow y$ is shown below.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Dual Rail Logic

- Instead of representing a truth value by a literal $x$, let’s represent it by $(\bar{x}, x)$. Thus, True and False are represented by $(0,1)$ and $(1,0)$.

- How is AND represented in this new system?
- How about OR and NOT?
- What does a circuit look like?
Dual Rail Logic

- **AND:** Inputs \((\bar{x}, x)\) and \((\bar{y}, y)\). Output \((\bar{z}, z)\) where \(z = x \land y\) and \(z = x \lor y\).
- **OR:** Inputs \((x, x)\) and \((y, y)\). Output \((z, z)\) where \(z = x \lor y\) and \(z = x \land y\).
- **NOT:** Input \((x, x)\), output \((z, z)\) where \(z = x\), \(z = \bar{x}\). Thus, flip the inputs.
- **AND, OR, NOT** circuit can be converted to dual rail by doubling AND, OR gates, dropping NOTs.
Review

• Functions and languages
• Designing circuits from functions
• Minterms and the DNF
• Maxterms and CNF
• Circuit complexity
• Algebra of Boolean expressions
• Illustration using the Full Adder
• Dual rail logic