announcements

Reading assignment
– Boolean Algebra
  12.1 – 12.3 7th Edition
  11.1 – 11.3 6th Edition

Homework 1 due today
– Hand in at start of class

Homework 2 available online later today

review: a quick combinational logic example

Calendar subsystem:
# of days in a month (to control watch display)

– used in controlling the display of a wrist-watch LCD screen

  – inputs: month, leap year flag
  – outputs: number of days

Example: (March, non-leap year) → 31

review: implementation in software

```
integer number_of_days (month, leap_year_flag) {
  switch (month) {
    case 1: return (31);
    case 2: if (leap_year_flag == 1) then
             return (29) else return (28);
    case 3: return (31);
    default: return (0);
  }
}
```
Encoding:
- how many bits for each input/output?
- binary number for month
- four wires for 28, 29, 30, and 31

<table>
<thead>
<tr>
<th>month</th>
<th>leap</th>
<th>d28</th>
<th>d29</th>
<th>d30</th>
<th>d31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1110</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1111</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Truth-table to logic to switches to gates

\[ d_{28} = \text{"1 when month=0010 and leap=0"} \]
\[ d_{28} = m_8' \cdot m_4' \cdot m_2 \cdot m_1' \cdot \text{leap}' \]
\[ d_{31} = \text{"1 when month=0001 or month=0011 or ... month=1100"} \]
\[ d_{31} = (m_8' \cdot m_4' \cdot m_2' \cdot m_1) + (m_8' \cdot m_4' \cdot m_2 \cdot m_1) + \ldots + (m_8 \cdot m_4 \cdot m_2' \cdot m_1') \]
\[ d_{31} = \text{can we simplify more?} \]

<table>
<thead>
<tr>
<th>month</th>
<th>leap</th>
<th>d28</th>
<th>d29</th>
<th>d30</th>
<th>d31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1110</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1111</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Boolean algebra

- **Boolean algebra to circuit design**
- **Boolean algebra**
  - a set of elements \( E \) containing \( \{0, 1\} \)
  - binary operations \( \{ +, \cdot \} \)
  - and a unary operation \( \{ ' \} \)
  - such that the following axioms hold:

  1. the set \( E \) contains at least two elements: \( a, b \)
  2. closure: \( a + b \) is in \( E \)
  3. commutativity: \( a + b = b + a \)
  4. associativity: \( a + (b + c) = (a + b) + c \)
  5. identity: \( a + 0 = a \)
  6. distributivity: \( a + (b \cdot c) = (a + b) \cdot (a + c) \)
  7. complementarity: \( a + a' = 1 \)

\( \text{Note: a} \cdot \text{a}' = 0 \)
axioms and theorems of Boolean algebra

identity: 1. X + 0 = X 1D. X • 1 = X
null: 2. X + 1 = 1 2D. X • 0 = 0
idempotency: 3. X + X = X 3D. X • X = X
involution: 4. (X')' = X
complementarity: 5. X + X' = 1 5D. X • X' = 0
commutativity: 6. X + Y = Y + X 6D. X • Y = Y • X
associativity: 7. (X + Y) + Z = X + (Y + Z) 7D. (X • Y) • Z = X • (Y • Z)
distributivity: 8. X • (Y + Z) = (X • Y) + (X • Z) 8D. X + (Y • Z) = (X + Y) • (X + Z)

axioms and theorems of Boolean algebra

uniting: 9. X • X + X • X' = X 9D. (X + Y) • (X + Y') = X
absorption: 10. X + X • Y = X 10D. X • (X + Y) = X
11. (X + Y') • Y = X • Y 11D. (X • Y') + Y = X + Y
factoring: 12. (X + Y) • (X' + Z) = Z 12D. X • Y + X' • Z = (X • Z) + (X' • Z)
consensus: 13. (X • Y) + (Y • Z) + (X' • Z) = X • Y + X' • Z 13D. (X • Y) + (Y + Z) + (X' + Z) = (X + Y) • (X' + Z)
de Morgan’s: 14. (X + Y + ...) = X' • Y' • ... 14D. (X + Y • ...)' = X' + Y' + ...

proving theorems (rewriting)

Using the laws of Boolean algebra:

prove the theorem: X • Y + X • Y' = X
distribution (8) complementarity (5) identity (1D)
X • Y + X • Y' = X • (Y + Y')
= X • (1)
= X

prove the theorem: X + X • Y = X
identity (1D) distribution (8) uniting (2) identity (1D)
X + X • Y = X • 1 + X • Y
= X • (1 + Y)
= X • (1)
= X

proving theorems (truth table)

Using complete truth table:

For example, de Morgan’s Law:

\[(X + Y)' = X' \cdot Y'\]

NOR is equivalent to AND with inputs complemented

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X'</th>
<th>Y'</th>
<th>(X + Y)'</th>
<th>X' \cdot Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[(X \cdot Y)' = X' + Y'\]

NAND is equivalent to OR with inputs complemented

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X'</th>
<th>Y'</th>
<th>(X \cdot Y)'</th>
<th>X' + Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
a simple example: 1-bit binary adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Cout</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin

S = A’ B’ Cin + A’ B Cin’ + A B’ Cin’ + A B Cin

apply theorems to simplify expressions

The theorems of Boolean algebra can simplify expressions—e.g., full adder’s carry-out function:

\[
Cout = A' B C + A' B C' + A B C' + A B C
\]

\[
S = A' B' C + A' B C' + A B' C' + A B C
\]

rewrite using xor gates

\[
S = A' B' C + A' B C' + A B' C' + A B C
\]
a simple example: 1-bit binary adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Cout</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>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Cout = A'B Cin + A B' Cin + A B Cin' + A B Cin
S = A xor (B xor Cin)

recall gates

**NOT**
\[ X' \bar{X} \rightarrow X \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

**AND**
\[ X \cdot Y \quad X \land Y \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

**OR**
\[ X + Y \quad X \lor Y \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

some other gates

**NAND**
\[ \neg(X \land Y) \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

**NOR**
\[ \neg(X \lor Y) \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

**XOR**
\[ X \oplus Y \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

**XNOR**
\[ X \leftrightarrow Y, X = Y \]

\[
\begin{array}{ccc|c|c|}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

da 2-bit ripple-carry adder
mapping truth tables to logic gates

Given a truth table:

1. Write the Boolean expression
2. Minimize the Boolean expression
3. Draw as gates
4. Map to available gates

\[ F = A'B'C' + A'BC + AB'C + ABC' + ABC \]

\[ = A'B'C' + A'BC + AB'C + ABC' + ABC \]

\[ = A'B'C' + A'BC + AB'C + ABC' + ABC \]

Canonical forms

- Truth table is the unique signature of a Boolean function
- The same truth table can have many gate realizations
  --we’ve seen this already
  --depends on how good we are at Boolean simplification
- Canonical forms
  -- standard forms for a Boolean expression
  -- we all come up with the same expression

Sum-of-products canonical form

- also known as Disjunctive Normal Form (DNF)
- also known as minterm expansion

\[ F = 001 + 011 + 101 + 110 + 111 \]

\[ F = A'B'C' + A'BC + AB'C + ABC' + ABC \]

Sum-of-products canonical form

Product term (or minterm)
- ANDed product of literals – input combination for which output is true
- each variable appears exactly once, true or inverted (but not both)

\[ F(A, B, C) = \Sigma m(1,3,5,6,7) \]

\[ = m1 + m3 + m5 + m6 + m7 \]

\[ = A'B'C' + A'BC + AB'C + ABC' + ABC \]

Canonical form ≠ minimal form

\[ F(A, B, C) = A'B'C' + A'BC + AB'C + ABC + ABC' \]

\[ = A'B'C' + A'BC + AB'C + ABC + ABC' \]

\[ = (A' + A)(B' + B)C + ABC' \]

\[ = C + ABC' \]

\[ = ABC' + C \]

\[ = AB + C \]
product-of-sums canonical form

- Also known as Conjunctive Normal Form (CNF)
- Also known as maxterm expansion

A B C F F' 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1

F = 000 010 100
F = (A + B + C) (A + B' + C) (A' + B + C)

product-of-sums canonical form

Sum term (or maxterm)
- ORed sum of literals – input combination for which output is false
- each variable appears exactly once, true or inverted (but not both)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>maxterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A + B + C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B + C'</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A + B' + C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A + B' + C'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A' + B + C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A' + B + C'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A' + B' + C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A' + B' + C'</td>
</tr>
</tbody>
</table>

F in canonical form:
F(A, B, C) = Π M(0,2,4) = M0 • M2 • M4 = (A + B + C) (A + B' + C) (A' + B + C)

F in canonical form ≠ minimal form
F(A, B, C) = (A + B + C) (A + B' + C) (A' + B + C) = (A + C) (B + C)

short-hand notation for maxterms of 3 variables