A Combinational Logic Example

Sessions of Class:
We would like to compute the number of lectures or quiz sections remaining at the start of a given day of the week.

- Inputs: Day of the Week, Lecture/Section flag
- Output: Number of sessions left

Examples: Input: (Wednesday, Lecture) Output: 2
Input: (Monday, Section) Output: 1
Implementation with Combinational Logic

Encoding:
- How many bits for each input/output?
- Binary number for weekday
- One bit for each possible output

Lecture? Weekday

0 1 2 3

Weekday Number Binary
Sunday 0 (000)
Monday 1 (001)
Tuesday 2 (010)
Wednesday 3 (011)
Thursday 4 (100)
Friday 5 (101)
Saturday 6 (110)

Converting to a Truth Table!

Weekday Lecture? c0 c1 c2 c3
Sunday 000 0 1 0 0
Monday 000 1 0 0 1
Tuesday 001 0 0 1 0
Wednesday 001 1 0 0 1
Thursday 010 0 0 1 0
Friday 010 1 0 0 1
Saturday 011 0 0 1 0
011 1 0 0 1
100 - 0 1 0 0
101 0 1 0 0
101 1 0 1 0
110 - 1 0 0 0
111 - - - -

Defining Our Inputs!

public int classesLeftInMorning(weekday, lecture_flag) {
    switch (day) {
    case SUNDAY: case MONDAY:
        return lecture_flag ? 3 : 1;
    case TUESDAY: case WEDNESDAY:
        return lecture_flag ? 2 : 1;
    case THURSDAY:
        return lecture_flag ? 1 : 1;
    case FRIDAY:
        return lecture_flag ? 1 : 0;
    case SATURDAY:
        return lecture_flag ? 0 : 0;
    }
}

Truth Table to Logic (Part 1)

\[ c3 = (DAY == SUN and LEC) or (DAY == MON and LEC) \]
\[ c3 = (d2 == 0 & & d1 == 0 & & d0 == 0 & & L == 1) or (d2 == 0 & & d1 == 0 & & d0 == 1 & & L == 1) \]
\[ c3 = d2' * d1' * d0' * L + d2' * d1' * d0 * L \]
Truth Table to Logic (Part 2)

\[
\begin{align*}
c_3 &= d_2 \cdot d_1 \cdot d_0 \cdot L + d_2 \cdot d_1 \cdot d_0 \cdot L \\
c_2 &= (\text{DAY} == \text{TUE} \text{ and LEC}) \text{ or } (\text{DAY} == \text{WED} \text{ and LEC}) \\
c_2 &= d_2 \cdot d_1 \cdot d_0 \cdot L + d_2 \cdot d_1 \cdot d_0 \cdot L
\end{align*}
\]

Truth Table to Logic (Part 3)

\[
\begin{align*}
c_3 &= d_2 \cdot d_1 \cdot d_0 \cdot L + d_2 \cdot d_1 \cdot d_0 \cdot L \\
c_2 &= d_2 \cdot d_1 \cdot d_0 \cdot L + d_2 \cdot d_1 \cdot d_0 \cdot L \\
c_1 &= \text{On your homework for next week!} \\
c_0 &= d_2 \cdot d_1 \cdot d_0 \cdot L' + d_2 \cdot d_1 \cdot d_0'
\end{align*}
\]

Logic to Gates

\[
\begin{align*}
c_3 &= d_2 \cdot d_1 \cdot d_0 \cdot L + d_2 \cdot d_1 \cdot d_0 \cdot L
\end{align*}
\]

CSE 311: Foundations of Computing

Fall 2014
Lecture 4: Boolean Algebra and Circuits
Boolean Algebra

- Boolean algebra to circuit design

Boolean algebra
- a set of elements B containing {0, 1}
- binary operations { +, • }
- and a unary operation { ’ }
- such that the following axioms hold:

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

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)

Proving Theorems (Rewriting)

Using the laws of Boolean Algebra:

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

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

\[\begin{array}{c|c|c|c|c|c}
X & Y & X' & Y' & (X + Y)' & X' \cdot Y' \\
\hline
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}\]

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

NAND is equivalent to OR with inputs complemented

\[\begin{array}{c|c|c|c|c|c}
X & Y & X' & Y' & (X \cdot Y)' & X' + Y' \\
\hline
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}\]

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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</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>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[S = A' B' \text{Cin} + A' B \text{Cin'} + A B' \text{Cin'} + A B \text{Cin}\]

\[\text{Cout} = A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin'} + A B \text{Cin}\]

Apply Theorems to Simplify Expressions

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

\[\text{Cout} = A' B' \text{Cin} + A' B \text{Cin'} + A B' \text{Cin} + A B \text{Cin}\]

\[= A' B' \text{Cin} + A' B' \text{Cin'} + A B' \text{Cin'} + A B \text{Cin}\]

\[= (A' + A) B' \text{Cin} + A B' \text{Cin'} + A B \text{Cin}\]

\[= (1) B' \text{Cin} + A B' \text{Cin'} + A B \text{Cin}\]

\[= B' \text{Cin} + A B' \text{Cin'} + A B \text{Cin}
\]

\[= B' \text{Cin} + A B' \text{Cin'} + A B \text{Cin} + A B \text{Cin}\]

\[= B \text{Cin} + A B' \text{Cin'} + A B \text{Cin} + A B \text{Cin}\]

\[= B \text{Cin} + A B' \text{Cin'} + A B \text{Cin} + A B \text{Cin}\]

\[= B \text{Cin} + A (B' + B) \text{Cin} + A B \text{Cin}\]

\[= B \text{Cin} + A (1) \text{Cin} + A B \text{Cin}\]

\[= B \text{Cin} + A \text{Cin} + A B\]

adding extra terms creates new factoring opportunities
Gates Again!

**NOT**
\[ X' \equiv \bar{X} \quad \neg X \]

\[ X \]
\[ Y \]
\[ 0 \quad 1 \]
\[ 1 \quad 0 \]

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

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 1 \quad 0 \]
\[ 0 \quad 1 \quad 0 \]
\[ 1 \quad 0 \quad 0 \]
\[ 1 \quad 1 \quad 1 \]

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

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 1 \quad 0 \]
\[ 1 \quad 0 \quad 1 \]
\[ 1 \quad 1 \quad 1 \]

More Gates!

**NAND**
\[ \neg (X \land Y) \quad (XY)' \]

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 0 \quad 1 \]
\[ 0 \quad 1 \quad 1 \]
\[ 1 \quad 0 \quad 1 \]
\[ 1 \quad 1 \quad 0 \]

**NOR**
\[ \neg (X \lor Y) \quad (X + Y)' \]

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 0 \quad 1 \]
\[ 0 \quad 1 \quad 1 \]
\[ 1 \quad 0 \quad 1 \]
\[ 1 \quad 1 \quad 0 \]

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

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 0 \quad 1 \]
\[ 0 \quad 1 \quad 1 \]
\[ 1 \quad 0 \quad 1 \]
\[ 1 \quad 1 \quad 0 \]

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

\[ X \]
\[ Y \]
\[ z \]
\[ 0 \quad 0 \quad 1 \]
\[ 0 \quad 1 \quad 1 \]
\[ 1 \quad 0 \quad 1 \]
\[ 1 \quad 1 \quad 0 \]

---

Slide 22

A 2-bit Ripple-Carry Adder

---

1. can we omit this slide? we mentioned it on homework and this feels like info overload?

Adam Blank, 2/7/2014
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'BC' + A'BC + AB'C + ABC
= A'B(C' + C) + AC(B' + B)
= A'B + AC

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 = 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 in canonical form:
F(A, B, C) = 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' + A'B + AB' + AB)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

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

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

Complement again and apply de Morgan’s and get the product-of-sums form:

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

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

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>
<th>F in canonical form:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A+B+C</td>
<td>F(A, B, C) = (A + B + C) (A + B' + C) (A' + B + C)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A+B+C'</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A+B'+C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A+B'+C'</td>
<td></td>
</tr>
</tbody>
</table>
| 1 | 0 | 0 | A'+B+C   | (A + B + C) (A + B' + C)
| 1 | 0 | 1 | A'+B'+C  | (A + B + C) (A' + B + C)
| 1 | 1 | 0 | A'+B'+C' | (A + C) (B + C)
| 1 | 1 | 1 | A'+B+C   | (A + B + C) (A + B' + C) |

s-o-p, p-o-s, and de Morgan’s theorem

Complement of function in sum-of-products form:
- \( F' = A'B'C' + A'BC' + AB'C' \)

Complement again and apply de Morgan’s and get the product-of-sums form:
- \( (F')' = (A'B'C' + A'BC' + AB'C')' \)
- \( F = (A + B + C) (A + B' + C) (A' + B + C) \)