Announcements

- Homework 2
  - Available for download
- Reading assignments
  - Boolean Algebra
    - 12.1 – 12.3 7th Edition
    - 11.1 – 11.3 6th Edition
    - 10.1 – 10.3 5th Edition
  - Predicates and Quantifiers
    - 1.4 7th Edition
    - 1.3 5th and 6th Edition

Ugrad Autumn Career Events

We strongly urge all CSE undergrads to attend our first two CSE Autumn Career Events. These events feature vital employment information for all future CSE job seekers. (Note: for CSE majors only.)

Event 1: Employer Panel
Date/Time: Wednesday, October 5, 2010 5:30-6:30pm
Place: EE125
Our first career event of autumn will provide an overview of the job search process from both the CSE student and employer perspective.

Event 2: Resume Review Roundtable Workshop
Date/Time: Tuesday, October 11, 3:00-6:00 pm
Place: CSE Atrium
In this workshop, HR reps, recruiters and engineers sit with small groups of CSE students to critique resumes, offer suggestions, and help refine the way you present yourself on paper.

Boolean logic

- Combinational logic
  - output, = f(input,)
- Sequential logic
  - output, = f(output, , input,)
  - output dependent on history
  - concept of a time step (clock)
- An algebraic structure consists of
  - a set of elements B = {0, 1}
  - binary operations (+, •) (OR, AND)
  - and a unary operation (') (NOT)

A quick combinational logic example

- Calendar subsystem: number 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

Implementation in software

```java
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);
        ... case 12: return (31);
        default: return (0);
    }
}
```
Implementation as a combinational digital system

- Encoding:
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31

Combinational example (cont’d)

- Truth-table to logic to switches to gates
  - \( d_{28} = 1 \) when \( \text{month}=0010 \) and \( \text{leap}=0 \)
  - \( d_{28} = \text{m8'•m4'•m2•m1'•leap} \)
  - \( d_{31} = 1 \) when \( \text{month}=0001 \) or \( \text{month}=0011 \) or \( \ldots \) \( \text{month}=1100 \)
  - \( d_{31} = (\text{m8'•m4'•m2'•m1}) + (\text{m8'•m4'•m2•m1}) + \ldots + (\text{m8•m4'•m2'•m1}) + (\text{m8•m4•m2'•m1'}) \)
  - \( d_{31} \) can we simplify more?

Switches: basic element of physical implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1”):
  - close switch (if \( A \) is “1” or asserted) and turn on light bulb (\( Z \))
  - open switch (if \( A \) is “0” or unasserted) and turn off light bulb (\( Z \))
Transistor networks

- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and normally-closed switches
- MOS transistors act as voltage-controlled switches
  - similar, though easier to work with than relays.

Multi-input logic gates

- CMOS logic gates are inverting
  - Easy to implement NAND, NOR, NOT while AND, OR, and Buffer are harder

Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:
  - in general, there are \(2^{2^n}\) functions of \(n\) inputs

Boolean algebra

- An algebraic structure consists of
  - a set of elements \(B\)
  - binary operations \((+, \cdot)\)
  - and a unary operation \((\cdot')\)
  - such that the following axioms hold:

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. identity</td>
<td>(X + 0 = X)</td>
</tr>
<tr>
<td>2. null</td>
<td>(X + 1 = 1)</td>
</tr>
<tr>
<td>3. idempotency</td>
<td>(X + X = X)</td>
</tr>
<tr>
<td>4. involution</td>
<td>((X')' = X)</td>
</tr>
<tr>
<td>5. complementarity</td>
<td>(X + X' = 1)</td>
</tr>
<tr>
<td>6. commutativity</td>
<td>(X + Y = Y + X)</td>
</tr>
<tr>
<td>7. associativity</td>
<td>((X + Y) + Z = X + (Y + Z))</td>
</tr>
<tr>
<td>8. distributivity</td>
<td>((X + Y) \cdot Z = X \cdot Z + Y \cdot Z)</td>
</tr>
</tbody>
</table>

Logic functions and Boolean algebra

Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: \(^{'}, +, \cdot\)

<table>
<thead>
<tr>
<th>Function</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X \cdot Y)</td>
<td>(X \cdot Y)</td>
</tr>
<tr>
<td>(X + Y)</td>
<td>(X + Y)</td>
</tr>
<tr>
<td>(X')</td>
<td>(X')</td>
</tr>
</tbody>
</table>
Axioms and theorems of Boolean algebra (cont’d)

uniting:
9. \( X \cdot Y + X \cdot Y' = X \)
9D. \( (X + Y) \cdot (X + Y') = X \)

absorption:
10. \( X + X \cdot Y = X \)
10D. \( X \cdot (X + Y) = X \)
11. \( (X + Y') \cdot Y = X \cdot Y \)
11D. \( (X \cdot Y') + Y = X + Y \)

factoring:
12. \( (X + Y) \cdot (X' + Z) = X \cdot Z + X' \cdot Y \)
12D. \( X \cdot Y + X' \cdot Z = (X + Z) \cdot (X' + Y) \)

consensus:
13. \( (X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = X \cdot Y + X' \cdot Z \)
13D. \( (X + Y) \cdot (Y + Z) \cdot (X' + Z) = (X + Y) \cdot (X' + Z) \)

de Morgan’s:
14. \( (X + Y + ...)' = X' \cdot Y' \cdot ... \)
14D. \( (X \cdot Y \cdot ...)' = X' + Y' + ... \)

generalized de Morgan’s:
15. \( f'(X_1, X_2, ..., X_n, 0, 1, +, \cdot) = f(X_1', X_2', ..., X_n', 1, 0, \cdot, +) \)

Proving theorems (rewriting)
• Using the laws of Boolean algebra:
- e.g., prove the theorem: \( X \cdot Y + X \cdot Y' = X \)
- e.g., prove the theorem: \( X + X \cdot Y = X \)

Proving theorems (perfect induction)
• Using perfect induction (complete truth table):
- e.g., de Morgan’s:
  \( (X + Y)' = X' \cdot Y' \)
  \( (X \cdot Y)' = X' + Y' \)

A simple example: 1-bit binary adder
• Inputs: A, B, Carry-in
• Outputs: Sum, Carry-out

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

A simple example: 1-bit binary adder
• Inputs: A, B, Carry-in
• Outputs: Sum, Carry-out
From Boolean expressions to logic gates

- **NOT** \( \overline{X} \) \( \sim X \) \( \overline{X} \) \( \overline{X} \) \( X \) \( Y \) \( \bar{x} \) \( \bar{x} \) \( \bar{y} \) \( x \) \( y \) \( 1 \) \( 0 \)

- **AND** \( X \land Y \) \( \land \) \( X \) \( Y \) \( \land \) \( X \) \( Y \) \( 0 \) \( 1 \) \( 0 \) \( 0 \) \( 1 \) \( 1 \) \( 1 \)

- **OR** \( X \lor Y \) \( \lor \) \( X \) \( Y \) \( \lor \) \( X \) \( Y \) \( 0 \) \( 1 \) \( 1 \) \( 1 \) \( 1 \) \( 1 \) \( 1 \)

- **NAND** \( \overline{X \land Y} \) \( \overline{X} \) \( \overline{Y} \) \( \overline{Z} \) \( 0 \) \( 1 \) \( 0 \) \( 0 \) \( 1 \) \( 1 \) \( 1 \)

- **NOR** \( \overline{X \lor Y} \) \( \overline{X} \) \( \overline{Y} \) \( \overline{Z} \) \( 0 \) \( 1 \) \( 0 \) \( 0 \) \( 1 \) \( 1 \) \( 1 \)

- **XOR** \( \overline{X} \oplus \overline{Y} \) \( \oplus \) \( X \) \( Y \) \( \oplus \) \( X \) \( Y \) \( 0 \) \( 1 \) \( 1 \) \( 1 \) \( 1 \) \( 0 \) \( 0 \)

- **XNOR** \( \overline{X} \equiv \overline{Y} \) \( \equiv \) \( X \) \( Y \) \( \equiv \) \( X \) \( Y \) \( 0 \) \( 1 \) \( 1 \) \( 1 \) \( 0 \) \( 0 \) \( 0 \)

Full adder: Carry-out

Before Boolean minimization: \( C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in} \)

After Boolean minimization: \( C_{out} = BC_{in} + AC_{in} + AB \)

Full adder: Sum

Before Boolean minimization: \( \text{Sum} = A'B'C_{in} + A'BC_{in} + AB'C_{in} + ABC_{in} \)

After Boolean minimization: \( \text{Sum} = (A \oplus B) \oplus C_{in} \)

Preview: A 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

\[ F = ABC + ABC + AB'C + ABC + AB(C + C) + AC(B + B) = AB + AC \]
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 forms

- Also known as conjunctive normal form
- Also known as maxterm expansion

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F in canonical form</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A'B'C' + A'BC' + AB'C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A'B'C + A'BC + AB'C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A'B'C + A'BC + AB'C'</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A'B'C + A'BC + AB'C'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A'B'C + A'BC + AB'C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A'B'C + A'BC + AB'C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A'B'C + A'BC + AB'C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A'B'C + A'BC + AB'C</td>
</tr>
</tbody>
</table>

Product-of-sums canonical form

- Also known as disjunctive normal form
- Also known as minterm expansion

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

Product-of-sums canonical form (cont’d)

- 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>maximal form</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>

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

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

- Complement of function in product-of-sums form
  - \( F' = (A + B + C)(A + B' + C)(A' + B + C)(A' + B' + C) \)

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

- Complement of function in product-of-sums form
  - \( F' = A'B'C + A'BC + AB'C \)