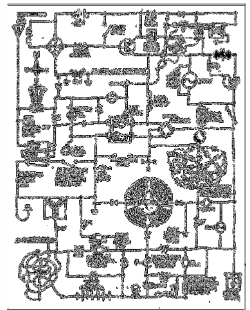
CSE 311



Foundations of Computing I

Fall 2014

Administrivia

Homework 1 Due Today

– Hand in at start of class

Homework 2 Available Online Later Today

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

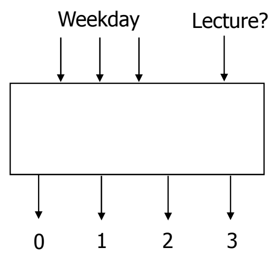
Implementation in Software

```
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;
    }
}
```

Implementation with Combinational Logic

Encoding:

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



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;
    }
}
```

Weekday	Number	Binary
Sunday	0	$(000)_2$
Monday	1	$(001)_2$
Tuesday	2	$(010)_2$
Wednesday	3	$(011)_2$
Thursday	4	$(100)_2$
Friday	5	$(101)_2$
Saturday	6	$(110)_2$

Converting to a Truth Table!

Weekday	Number	Binary
Sunday	0	$(000)_2$
Monday	1	$(001)_2$
Tuesday	2	$(010)_2$
Wednesday	3	$(011)_2$
Thursday	4	$(100)_2$
Friday	5	$(101)_2$
Saturday	6	$(110)_2$

Weekday	Lecture?	c0	c1	c2	c3
000	0	0	1	0	0
000	1	0	0	0	1
001	0	0	1	0	0
001	1	0	0	0	1
010	0	0	1	0	0
010	1	0	0	1	0
011	0	0	1	0	0
011	1	0	0	1	0
100	-	0	1	0	0
101	0	1	0	0	0
101	1	0	1	0	0
110	-	1	0	0	0
111	-	-	-	-	-

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) ||
(d2 == 0 && d1 == 0 && d0 == 1 && L == 1)

$c3 = d2' \cdot d1' \cdot d0' \cdot L + d2' \cdot d1' \cdot d0 \cdot L$

DAY	d2 d1 d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

Truth Table to Logic (Part 2)

c3 = d2’•d1’•d0’•L + d2’•d1’•d0•L

c2 = (DAY == TUE and LEC) or
(DAY == WED and LEC)

c2 = d2’•d1•d0’•L + d2’•d1•d0•L

DAY	d2 d1 d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

Truth Table to Logic (Part 3)

c3 = d2’•d1’•d0’•L + d2’•d1’•d0•L
c2 = d2’•d1 •d0’•L + d2’•d1 •d0•L

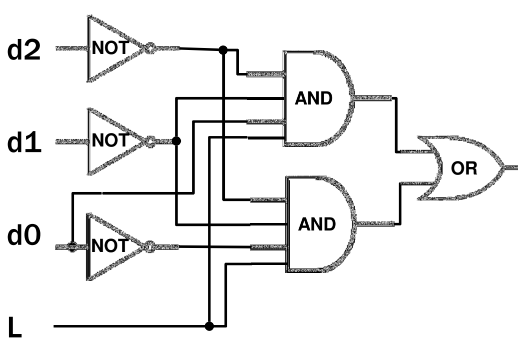
c1 = On your homework for next week!

c0 = d2•d1’• d0 •L’ + d2•d1 •d0’

DAY	d2 d1 d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

Logic to Gates

c3 = d2’•d1’•d0’•L + d2’•d1’•d0•L

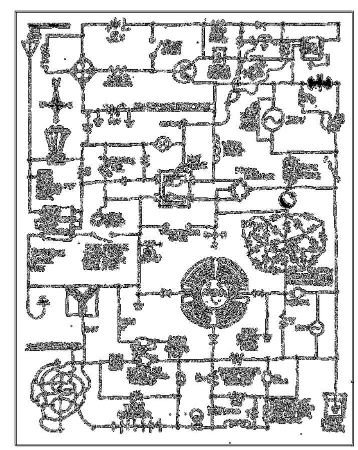


Multi-input AND gates

DAY	d2 d1 d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

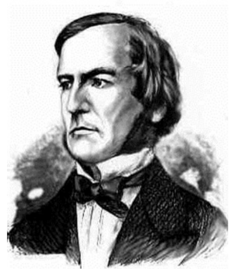
CSE 311: Foundations of Computing

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	$a \bullet b$ is in B
3. commutativity:	$a + b = b + a$	$a \bullet b = b \bullet a$
4. associativity:	$a + (b + c) = (a + b) + c$	$a \bullet (b \bullet c) = (a \bullet b) \bullet c$
5. identity:	$a + 0 = a$	$a \bullet 1 = a$
6. distributivity:	$a + (b \bullet c) = (a + b) \bullet (a + c)$	$a \bullet (b + c) = (a \bullet b) + (a \bullet c)$
7. complementarity:	$a + a' = 1$	$a \bullet a' = 0$

Axioms and Theorems of Boolean Algebra

identity:
- 1. $X + 0 = X$ — 1D. $X \bullet 1 = X$

null:
- 2. $X + 1 = 1$ — 2D. $X \bullet 0 = 0$

idempotency:
- 3. $X + X = X$ — 3D. $X \bullet X = X$

involution:
- 4. $(X')' = X$

complementarity:
- 5. $X + X' = 1$ — 5D. $X \bullet X' = 0$

commutatively:
- 6. $X + Y = Y + X$ — 6D. $X \bullet Y = Y \bullet X$

associativity:
- 7. $(X + Y) + Z = X + (Y + Z)$ — 7D. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$

distributivity:
- 8. $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$ — 8D. $X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$

Axioms and Theorems of Boolean Algebra

uniting:
- 9. $X \bullet Y + X \bullet Y' = X$ — 9D. $(X + Y) \bullet (X + Y') = X$

absorption:
- 10. $X + X \bullet Y = X$ — 10D. $X \bullet (X + Y) = X$
- 11. $(X + Y') \bullet Y = X \bullet Y$ — 11D. $(X \bullet Y') + Y = X + Y$

factoring:
- 12. $(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$ — 12D. $X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$

consensus:
- 13. $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$ — 13D. $(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$

de Morgan’s:
- 14. $(X + Y + \ldots)' = X' \bullet Y' \bullet \ldots$ — 14D. $(X \bullet Y \bullet \ldots)' = X' + Y' + \ldots$

Proving Theorems (Rewriting)

Using the laws of Boolean Algebra:

prove the theorem: $X \bullet Y + X \bullet Y' = X$

distributivity (8)	$X \bullet Y + X \bullet Y' = X \bullet (Y + Y')$
complementarity (5)	$= X \bullet (1)$
identity (1D)	$= X$

prove the theorem: $X + X \bullet Y = X$

identity (1D)	$X + X \bullet Y = X \bullet 1 + X \bullet Y$
distributivity (8)	$= X \bullet (1 + Y)$
uniting (2)	$= X \bullet (1)$
identity (1D)	$= X$

Proving Theorems (Truth Table)

Using complete truth table:

For example, de Morgan's Law:

$(X + Y)' = X' \bullet Y'$
NOR is equivalent to AND
with inputs complemented

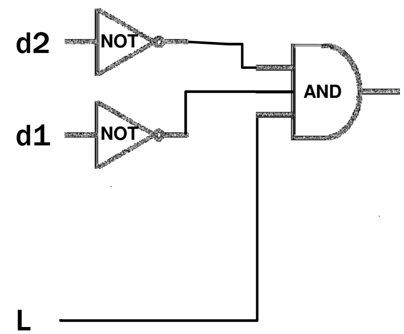
X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

$(X \bullet Y)' = X' + Y'$
NAND is equivalent to OR
with inputs complemented

X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

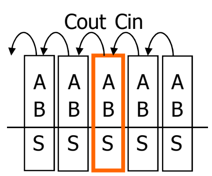
Simplifying using Boolean algebra

$$
\begin{aligned}
c3 &= d2' \bullet d1' \bullet d0' \bullet L + d2' \bullet d1' \bullet d0 \bullet L \\
&= d2' \bullet d1' \bullet (d0' + d0) \bullet L \\
&= d2' \bullet d1' \bullet (1) \bullet L \\
&= d2' \bullet d1' \bullet L
\end{aligned}
$$



1-bit Binary Adder

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



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A'\,B'\,Cin + A'\,B\,Cin' + A\,B'\,Cin' + A\,B\,Cin$$

$$Cout = A'\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin$$

Apply Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

– **e.g., full adder's carry-out function**

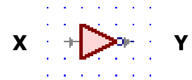
$$
\begin{aligned}
Cout &= A'\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= A'\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + \boxed{A\,B\,Cin + A\,B\,Cin} \\
&= A'\,B\,Cin + A\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= (A' + A)\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= (1)\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,B'\,Cin + A\,B\,Cin' + \boxed{A\,B\,Cin + A\,B\,Cin} \\
&= B\,Cin + A\,B'\,Cin + A\,B\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,(B' + B)\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,(1)\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,Cin + A\,B\,(Cin' + Cin) \\
&= B\,Cin + A\,Cin + A\,B\,(1) \\
&= B\,Cin + A\,Cin + A\,B
\end{aligned}
$$

adding extra terms
creates new factoring
opportunities

Gates Again!

NOT

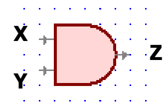
$X' \quad \bar{X} \quad \neg X$



X	Y
0	1
1	0

AND

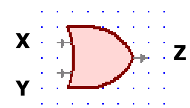
$X \cdot Y \quad XY \quad X \wedge Y$



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR

$X + Y \quad X \vee Y$



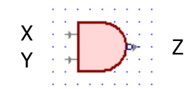
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

1

More Gates!

NAND

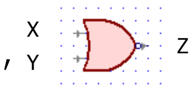
$\neg(X \wedge Y) \quad (XY)'$



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR

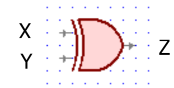
$\neg(X \vee Y) \; (X + Y)'$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR

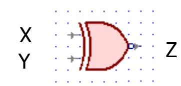
$X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$X \leftrightarrow Y$

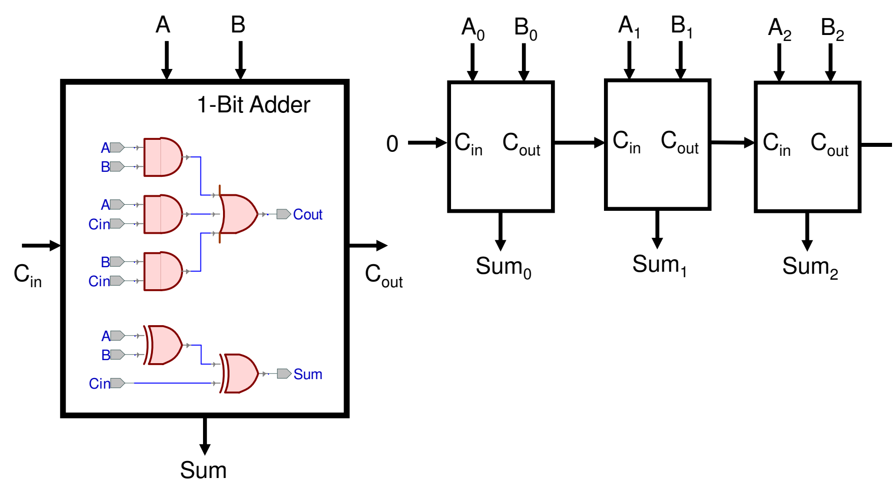


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Slide 22

1 can we omit this slide? we mentioned it on the hw and this feels like info overload?
Adam Blank, 9/27/2014

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

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

1

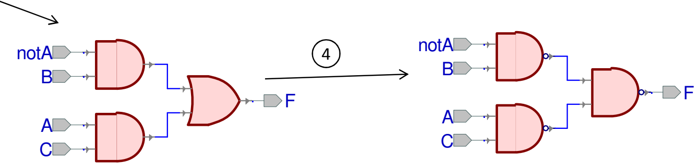
F = A'BC'+A'BC+AB'C+ABC

2

= A'B(C'+C)+AC(B'+B)

= A'B+AC

3



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

A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

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)

A	B	C	minterms
0	0	0	A′B′C′
0	0	1	A′B′C
0	1	0	A′BC′
0	1	1	A′BC
1	0	0	AB′C′
1	0	1	AB′C
1	1	0	ABC′
1	1	1	ABC

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′

= ((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

F = *000* *010* *100*

$F = (A + B + C)\ (A + B' + C)\ (A' + B + C)$

A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

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:

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

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)

A	B	C	maxterms
0	0	0	$A+B+C$
0	0	1	$A+B+C'$
0	1	0	$A+B'+C$
0	1	1	$A+B'+C'$
1	0	0	$A'+B+C$
1	0	1	$A'+B+C'$
1	1	0	$A'+B'+C$
1	1	1	$A'+B'+C'$

F in canonical form:

$F(A, B, C) = (A + B + C)\ (A + B' + C)\ (A' + B + C)$

canonical form ≠ minimal form

$$\begin{aligned} F(A, B, C) &= (A + B + C)\ (A + B' + C)\ (A' + B + C) \\ &= (A + B + C)\ (A + B' + C) \\ &\quad\ (A + B + C)\ (A' + B + C) \\ &= (A + C)\ (B + C) \end{aligned}$$