administrivia

Homework #1 Due Friday before class.

Please try out Gradescope before then!

(You can submit multiple times, so do a test run on the first homework.)

Office hours now posted on the web page:

TA	Office hours	Room
Evan McCarty	Thu, 3-4pm	CSE 021
Mert Saglam	Tue, 11-12pm	CSE 021
Krista Holden	Thu, 10:30-11:30am	CSE 220
Gunnar Onarheim	Tue, 3-4pm	CSE 021
Ian Turner	Wed, 12-1pm	CSE 218
Junhao (Ian) Zhu	Thu 4-5pm	CSE 021

Sections start this week:

Section	Day/Time	Room
AA Evan	Th, 1230-120	THO 234
AB Mert	Th, 130-220	DEN 217
AC Ian	Th, 230-320	MGH 254
AD Krista	Th, 1130-1220	MGH 251

Class e-mail list, Discussion board

a combinatorial 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

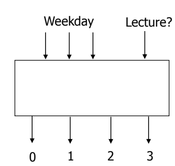
implementation in software

```
public int classesLeft (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 classesLeft (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 ⇒ logic (part one)

DAY	d2d1d0	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	-	-	-	-	-

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)

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

truth table ⇒ logic (part two)

DAY	d2d1d0	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	-	-	-	-	-

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

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

$c2 = d2'\cdot d1\cdot d0'\cdot L + d2'\cdot d1\cdot d0\cdot L$

truth table ⇒ logic (part three)

DAY	d2d1d0	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	-	-	-	-	-

$c3 = d2'\cdot d1'\cdot d0'\cdot L + d2'\cdot d1'\cdot d0\cdot L$
$c2 = d2'\cdot d1\cdot d0'\cdot L + d2'\cdot d1\cdot d0\cdot L$

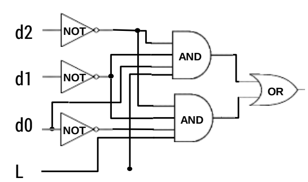
c1 =

[you do this one]

$c0 = d2\cdot d1'\cdot d0\cdot L' + d2\cdot d1\cdot d0'$

logic ⇒ gates

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

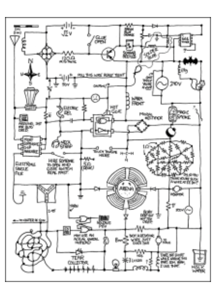


(multiple input AND gates)

DAY	d2d1d0	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

Spring 2015
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 • b is in B
3. commutativity:	a + b = b + a	a • b = b • a
4. associativity:	a + (b + c) = (a + b) + c	a • (b • c) = (a • b) • c
5. identity:	a + 0 = a	a • 1 = a
6. distributivity:	a + (b • c) = (a + b) • (a + c)	a • (b + c) = (a • b) + (a • c)
7. complementarity:	a + a' = 1	a • 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 • Y + X • Y' = 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) = X • Z + X' • Y — 12D. X • Y + X' • Z = (X + Z) • (X' + Y)

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

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)
uniting (2)	= X • (1)
identity (1D)	= X

proving theorems (truth table)

Using complete truth table:

For example, de Morgan's Law:

(X + Y)' = X' • Y'
NOR is equivalent to AND
with inputs complemented

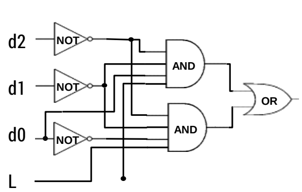
X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

(X • 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		

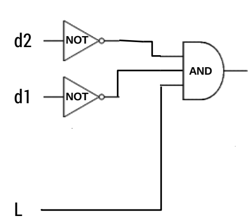
simplifying using Boolean algebra

c3 = d2'•d1'•d0'•L + d2'•d1'•d0•L
= d2'•d1'•(d0' + d0)•L
= d2'•d1'•(1)•L
= d2'•d1'•L



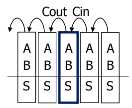
simplifying using Boolean algebra

c3 = d2'•d1'•d0'•L + d2'•d1'•d0•L
= d2'•d1'•(d0' + d0)•L
= d2'•d1'•(1)•L
= d2'•d1'•L



1-bit binary adder

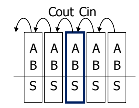
- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



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

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

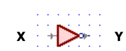
Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin
= A' B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= A' B Cin + A B Cin + A B' Cin + A B Cin' + A B Cin
= (A' + A) B Cin + A B' Cin + A B Cin' + A B Cin
= (1) B Cin + A B' Cin + A B Cin' + A B Cin
= B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= B 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

adding extra terms creates new factoring opportunities

more gates

NOT

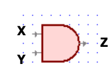
X' $\bar{X}$ $\neg X$



X	Y
0	1
1	0

AND

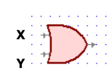
$X \cdot Y$ XY $X \wedge Y$



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

OR

$X + Y$ $X \vee Y$

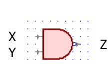


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

more gates

NAND

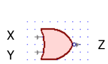
$\neg(X \wedge Y)$ $(XY)'$



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

NOR

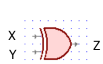
$\neg(X \vee Y)$ $(X + Y)'$



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

XOR

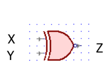
$X \oplus Y$



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

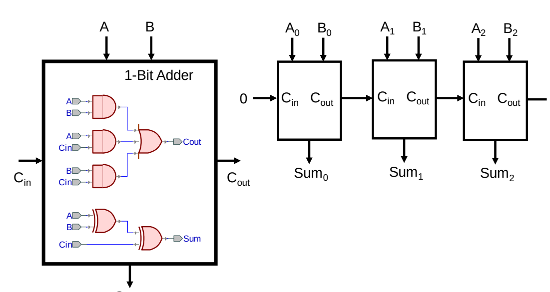
XNOR

$X \leftrightarrow Y$



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

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

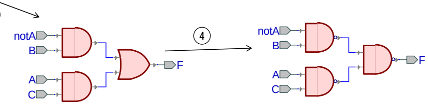
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

① F = A'BC'+A'BC+AB'C+ABC

② = A'B(C'+C)+AC(B'+B)

= A'B+AC

③ ④



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

F(A, B, C) = (A + B + C) (A + B′ + C) (A′ + B + C)
= (A + B + C) (A + B′ + C)
(A + B + C) (A′ + B + C)
= (A + C) (B + C)