Combinational Logic Design

Today’s lecture

- **Combinational Logic**
  - Different Representations of Boolean Functions (review)
- **How to design any circuit**
  - Write a truth table
  - Sum-of-Products implementation
  - Example
- **Other gates you should know about (XOR, NAND, NOR)**
- **Divide-and-Conquer design**

Combinational Logic

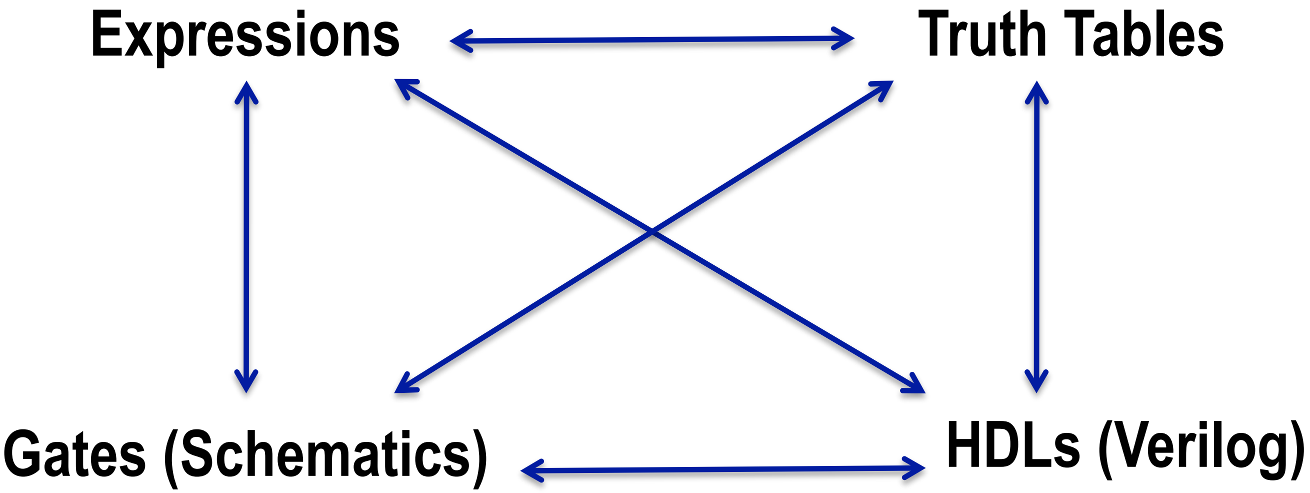
- **Definition: Boolean circuits where the output is a pure function of the present input only.**

- **Circuits made up of gates, that don't have any feedback**
  - No feedback: **outputs** are not connected to **inputs**
  - If you change the inputs, *and wait for a while*, the correct outputs show up.
    - Real circuits have delays (more on this later)

- **Can be represented by Boolean Algebra**

Four representations of Boolean functions

- Equivalent functionality



Expressions

Truth Tables

Gates (Schematics)

HDLs (Verilog)

- Relatively mechanical to translate between these formats

A fifth representation

- **An English description/specification**

Example: A sandwich shop has the following rules for making a good (meat) sandwich:

(1) All sandwiches must have at least one type of meat.
(2) Don't put both roast beef and ham on the same sandwich.
(3) Cheese only goes on sandwiches that include turkey.

Write a Boolean expression for the allowed combinations of sandwich ingredients using the following variables:

c = cheese
h = ham
t = turkey
r = roast beef

English → Truth Table example

- **Most reliable method**
  1. Write a truth table
  2. Every row evaluating to 1 becomes a term
  3. OR all the terms together

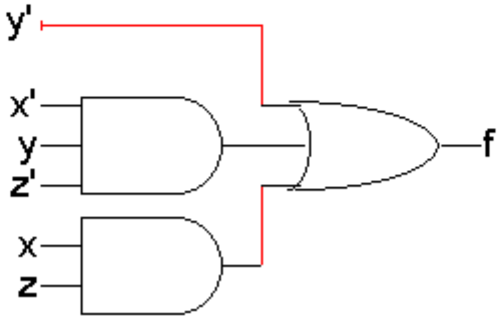
- **This will give an un-optimized expression**
  - (we can write computer programs to optimize expressions)
    - (or better yet, use the ones that other people wrote…)
  - (we can’t write programs to design circuits for us.)

Sum of Products (SOP) form

- **A useful way to represent any Boolean expression**
- **A sum of products (SOP) expression contains:**
  - only OR (sum) operations at the “outermost” level
  - Each term that is summed must be a product of literals

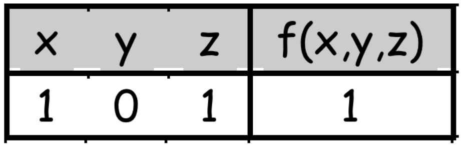
$$f(x,y,z) = y' + x'yz' + xz$$

- **The advantage is that any sum of products expression can be implemented using a two-level circuit**
  - literals and complements at “0th” level
  - AND gates at the first level
  - a single OR gate at the second level



Truth tables to Boolean expressions

1. **For each row in truth table where output is 1**
   - Write a product term that is true for that set of inputs
     - And only for that set of inputs



x	y	z	f(x,y,z)
1	0	1	1

→ **xy'z**

   - This product will include each terminal exactly once
2. **OR all the product terms together**

   product-term1 + product-term2 + product-term3 + …

Truth table -> Boolean -> gates example

Step 1. Write a truth table

Rules:

(1) must have at least one meat.

(2) not both roast beef and ham.

(3) cheese only if turkey.

Ingredients:

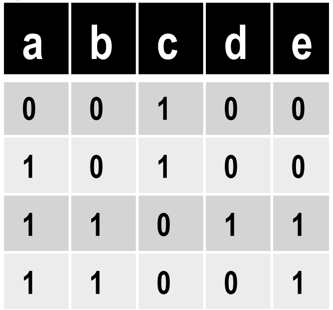
c = cheese

h = ham

t = turkey

r = roast beef

c	h	t	r	f(..)
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



a	b	c	d	e
0	0	1	0	0
1	0	1	0	0
1	1	0	1	1
1	1	0	0	1

Step 2. Every 1 becomes a term

Truth table -> Boolean -> gates example

Step 1. Write a truth table

Rules:

(1) must have at least one meat.
(2) not both roast beef and ham.
(3) cheese only if turkey.

c	h	t	r	f(..)	
0	0	0	0	0	
0	0	0	1	1	c'h't'r
0	0	1	0	1	c'h'tr'
0	0	1	1	1	c'h'tr
0	1	0	0	1	c'ht'r'
0	1	0	1	0	
0	1	1	0	1	c'htr'
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	ch'tr'
1	0	1	1	1	ch'tr
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	chtr'
1	1	1	1	0	

Ingredients:

c = cheese	
h = ham	
t = turkey	
r = roast beef	

Step 2. Every 1 becomes a term

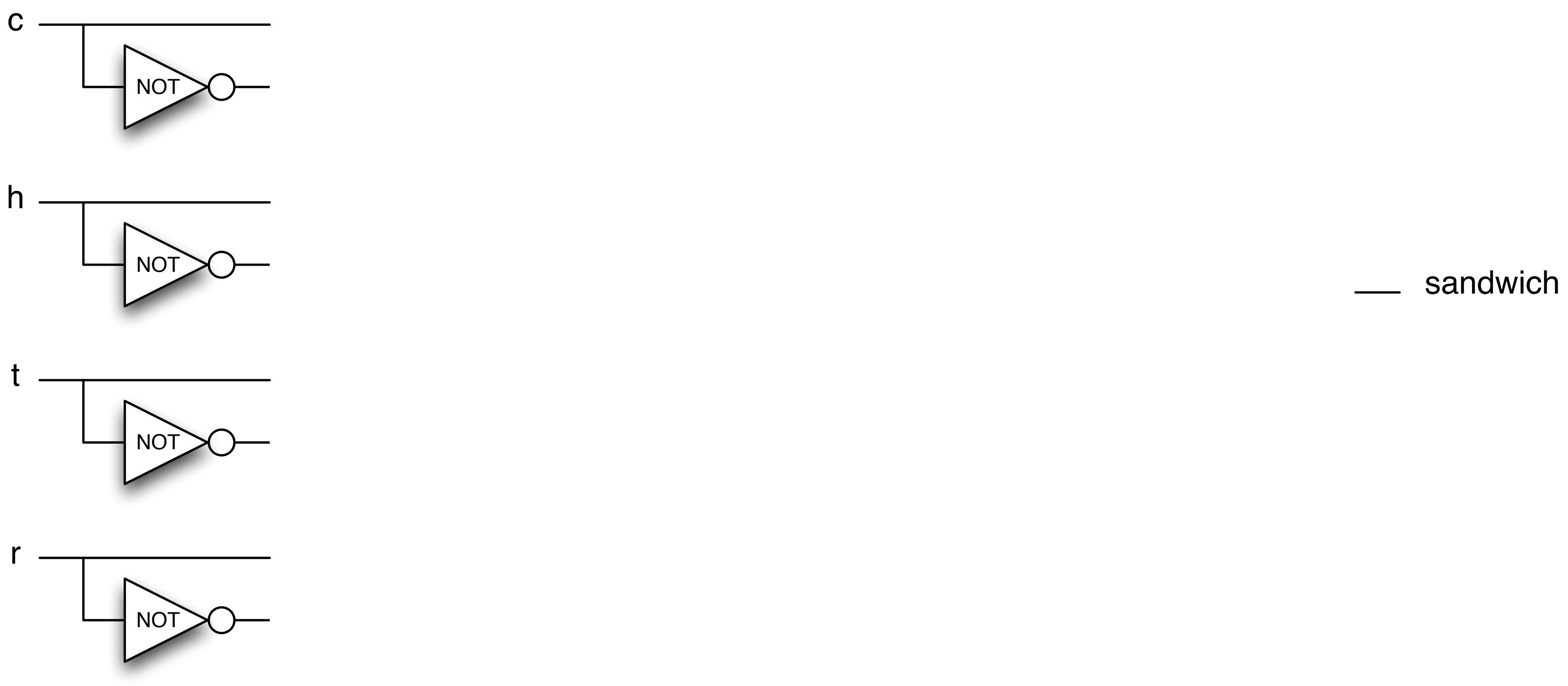
Truth table -> Boolean -> gates example

Step 3. OR all the terms together

c	h	t	r	f(..)	
0	0	0	0	0	
0	0	0	1	1	c'h’t’r
0	0	1	0	1	c'h’tr’
0	0	1	1	1	c'h’tr
0	1	0	0	1	c'ht’r’
0	1	0	1	0	
0	1	1	0	1	c'htr’
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	ch’tr’
1	0	1	1	1	ch’tr
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	chtr’
1	1	1	1	0	

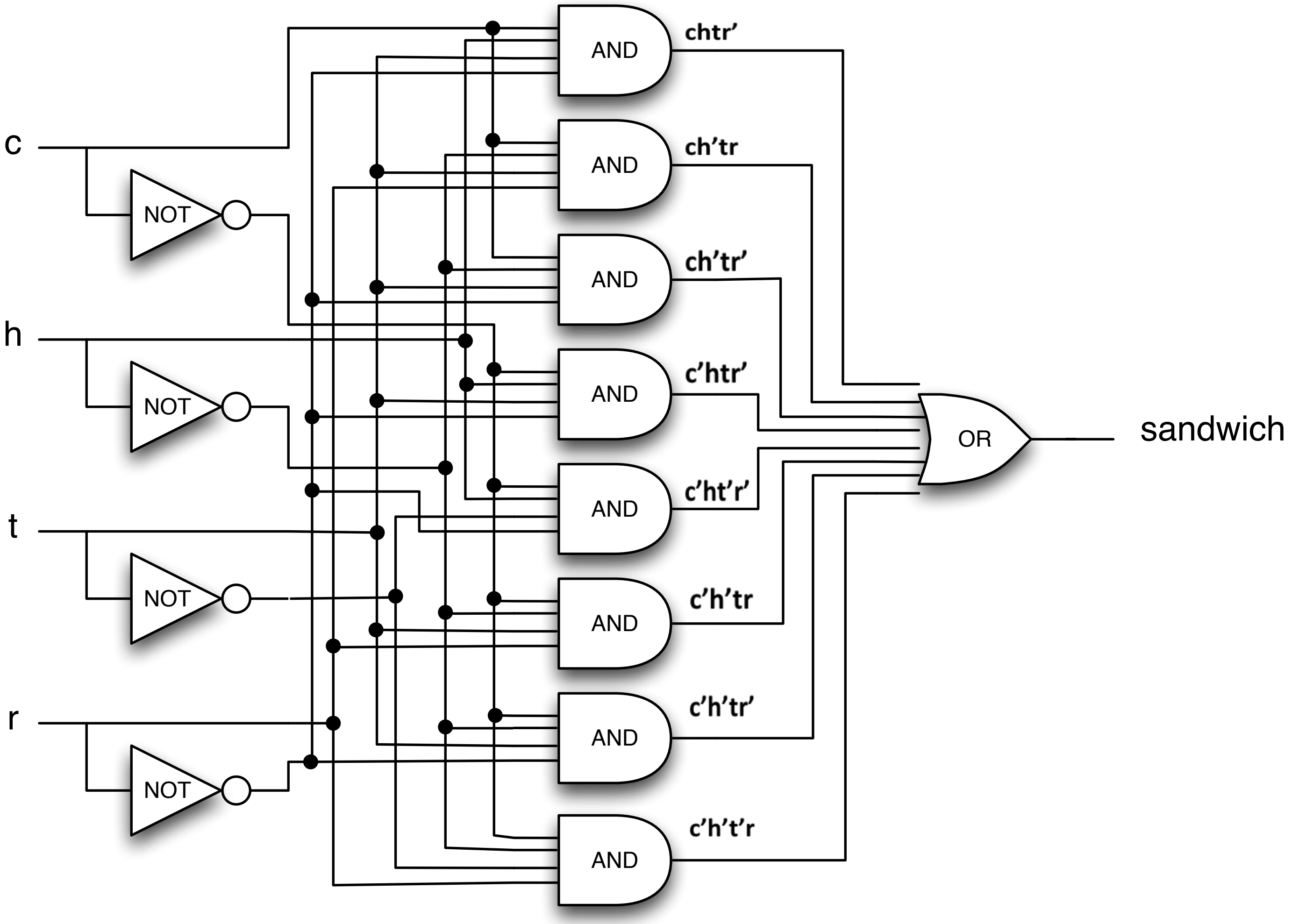
Truth table -> Boolean -> gates example

c’h’t’r + c’h’tr’ + c’h’tr + c’ht’r’ + c’htr’ + ch’tr’ + ch’tr + chtr’



Truth table -> Boolean -> gates example

c'h't'r + c'h'tr' + c'h'tr + c'ht'r' + c'htr' + ch'tr' + ch'tr + chtr'



Three other notable 2-input functions

- **Remember this table?**

			AND						OR								
x	y	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Additional Boolean operations

Operation:	NAND (NOT-AND)	NOR (NOT-OR)	XOR (eXclusive OR)
Expressions:	$(xy)' = x' + y'$	$(x + y)' = x'y'$	$x \oplus y = x'y + xy'$

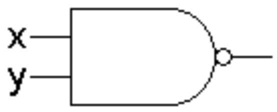
Truth table:

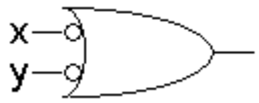
x	y	(xy)'
0	0	1
0	1	1
1	0	1
1	1	0

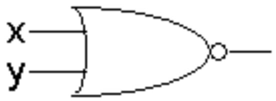
x	y	(x+y)'
0	0	1
0	1	0
1	0	0
1	1	0

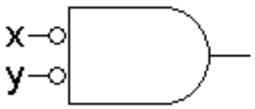
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

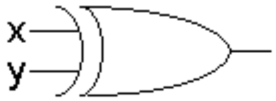
Logic gates:











XOR gates

- A two-input XOR gate outputs true when exactly one of its inputs is true:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

$$x \oplus y = x'y + xy'$$

- XOR corresponds more closely to typical English usage of "either … or," *either of the two, but not neither nor both.*
- Several fascinating properties of the XOR operation:

$x \oplus 0 = x$	$x \oplus 1 = x'$
$x \oplus x = 0$	$x \oplus x' = 1$
$x \oplus (y \oplus z) = (x \oplus y) \oplus z$	[Associative]
$x \oplus y = y \oplus x$	[Commutative]

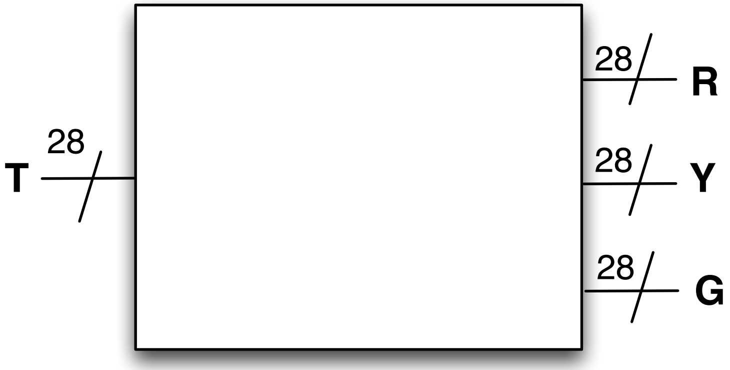
x XOR y = x’ XOR y’

a) True

b) False

Divide-and-Conquer Design

- **Consider the following problem**
  - You are building system to help avoid train collisions on subways.
  - Each of the 28 segments of track:
    - Senses if there is a train on it (T = 1) or no train (T = 0)
    - Has a red/yellow/green stoplight, where exactly 1 light is on at a time
      - The red light is on (R = 1) if there is a train in the next segment
      - Otherwise, yellow is on (Y = 1) if a train is 2 segments away
      - Else, green is on (G = 1)
- **We could implement this as one big circuit.**

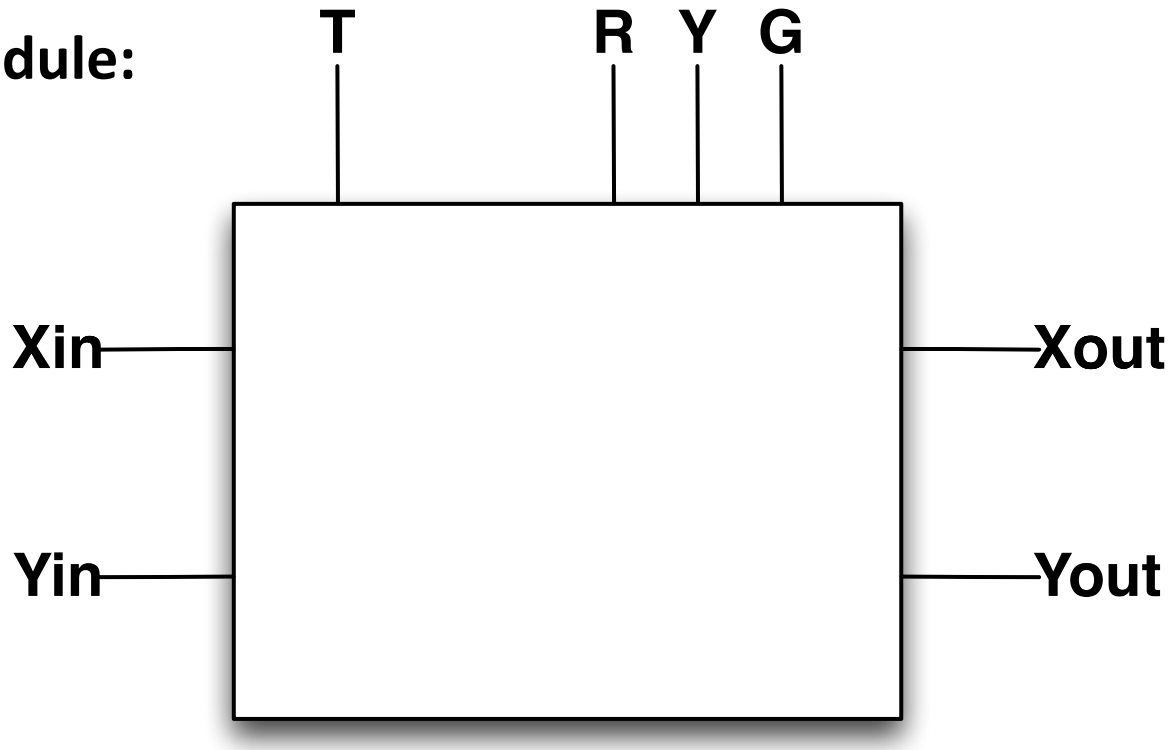


Proximity sensor

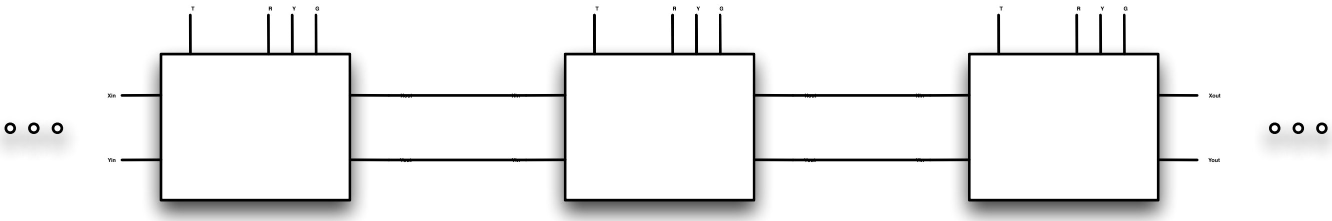
Why would that be a bad idea?

Divide-and-Conquer Design

- Instead build a module:



- And replicate:



What is Y(Xin, Yin)?

a) Xin ⊕ Yin

b) Xin + Yin’

c) Xin’ + Yin

d) Xin • Yin’

e) Xin’ • Yin

What is G(Xin, Yin)?

a) Xin OR Yin

b) Xin NOR Yin

c) Xin AND Yin

d) Xin NAND Yin

e) Xin XOR Yin