CS2100: Computer Organisation

Tutorial #6: Boolean Algebra, Logic Gates and Simplification

(Week 8: 11 – 15 March 2024)

Answers to Selected Questions

Note: By default, we assume that complemented literals are NOT available, unless otherwise stated.

2. Using Boolean algebra, simplify each of the following expressions into simplified **sum-of-products (SOP) expression**. Indicate the law/theorem used at every step.

   (a) $F(x,y,z) = (x + y \cdot z') \cdot (y' + y) + x' \cdot (y \cdot z' + y)$

   (b) $G(p,q,r,s) = \Pi M(5, 9, 13)$

   Tip: For (b), it is easier to start with the given expression and get done in about 5 steps, rather than to expand it into sum-of-products/sum-of-minterms expression first.

   Answers:

   Note: There are more than one way of derivation.

   (a) $(x + y \cdot z') \cdot \mathbf{(y' + y)} + x' \cdot (y \cdot z' + y)$

   $= (x + y \cdot z') \cdot \mathbf{1} + x' \cdot (y \cdot z' + y)$ [complement]

   $= (x + y \cdot z') + x' \cdot (\mathbf{y \cdot z' + y})$ [identity]

   $= x + \mathbf{y \cdot z'} + \mathbf{x' \cdot y}$ [absorption 1]

   $= \mathbf{x} + \mathbf{x' \cdot y} + y \cdot z'$ [commutative]

   $= x + \mathbf{y} + \mathbf{y \cdot z'}$ [absorption 2]

   $= \mathbf{x + y}$ [absorption 1]

   (b) $G(p,q,r,s) = \Pi M(5, 9, 13)$

   $= \mathbf{(p + q' + r + s')} \cdot (p' + q + r + s') \cdot \mathbf{(p' + q' + r + s')}$

   $= ((\mathbf{p \cdot p'}) + (q' + r + s')) \cdot (p' + q + r + s')$ [distributive]

   $= (\mathbf{0 + (q' + r + s')}) \cdot (p' + q + r + s')$ [complement]

   $= (q' + \mathbf{r + s'}) \cdot (p' + q + \mathbf{r + s'})$ [identity]

   $= (\mathbf{q' \cdot (p' + q)}) + (r + s')$ [distributive]

   $= \mathbf{p' \cdot q' + r + s'}$ [absorption 2]

> Reminder: Write · for AND, and not to leave it out. For example, for "*x* AND *y*", write *x*·*y* and not *xy*.

3. (a) The following K-map layout is used for a 4-variable Boolean function $T(A,B,C,D)$. Fill in the minterm positions m1 to m15 into the respective cells. m0 has been filled for you.

			B	*B*	
	m0	m1	m5	m4	
	m2	m3	m7	m6	*C*
A	m10	m11	m15	m14	*C*
A	m8	m9	m13	m12	
		D	*D*		

T

			B	*B*	
	1	1	1	1	
	1	0	0	X	*C*
A	0	X	X	X	*C*
A	0	1	0	0	
		D	*D*		

Alternative PI: $B'{\cdot}C'{\cdot}D$

(b) Given the following 4-variable Boolean function:

$$T(A,B,C,D) = \Pi M(3,7,8,10,12,13) \cdot X(6,11,14,15)$$

where X's are the don't-cares, write out the Σm notation for $T(A,B,C,D)$.

(c) Draw the K-map for T using the layout above.

(d) How many PIs (prime implicants) are there in the K-map? List out all the PIs.

(e) How many EPIs (essential prime implicants) are there? List out all the EPIs.

(f) What is the simplified SOP expression for T? List out all alternative solutions.

(g) What is the simplified POS expression for T? List out all alternative solutions.

(h) Implement the simplified SOP expression for T using a 2-level AND-OR circuit and a 2-level NAND only circuit.

Answers:

(a) See above.

(b) $T(A,B,C,D) = \Sigma m(0,1,2,4,5,9) + X(6,11,14,15)$.

(c) See K-map above.

(d) 4 PIs: $A'{\cdot}D'$, $A'{\cdot}C'$, $A{\cdot}B'{\cdot}D$ and $B'{\cdot}C'{\cdot}D$.

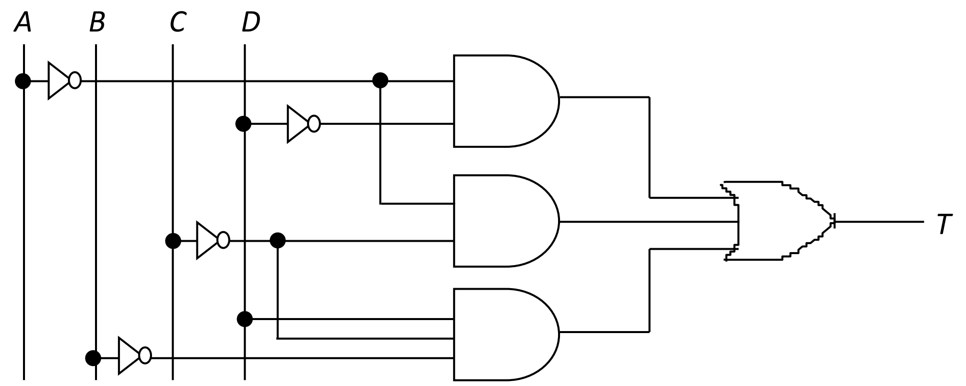
(e) 2 EPIs: $A'{\cdot}D'$ and $A'{\cdot}C'$.

(f) SOP expression: $T(A,B,C,D) = A'{\cdot}D' + A'{\cdot}C' + B'{\cdot}C'{\cdot}D$ or $A'{\cdot}D' + A'{\cdot}C' + A{\cdot}B'{\cdot}D$.

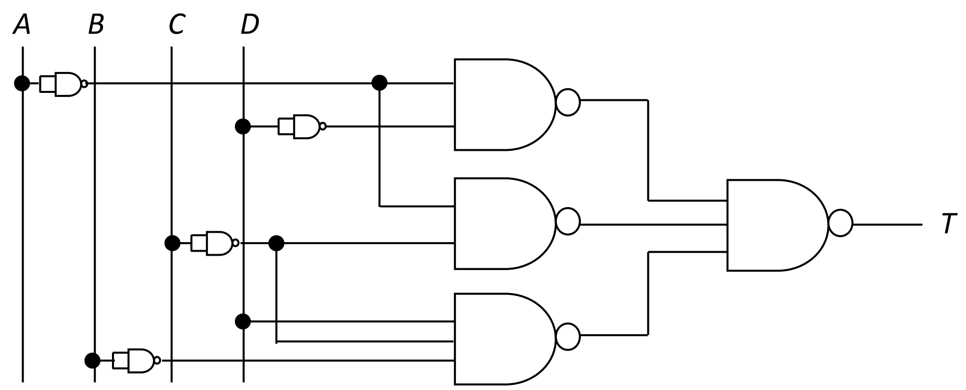
(g) POS expression: $T(A,B,C,D) = (A' + D){\cdot}(C' + D'){\cdot}(A' + B')$.
[Working: $T'(A,B,C,D) = A{\cdot}D' + C{\cdot}D + A{\cdot}B$.]

(h) Take the expression $A'{\cdot}D' + A'{\cdot}C' + B'{\cdot}C'{\cdot}D$

2-level AND-OR circuit:



2-level NAND circuit:



Students: Draw logic diagrams neatly with <u>straight lines</u>. Draw thick dots to represent forks.

Using Quine McCluskey to find the simplified SOP expression for *T*.

(Just for illustration. Quine McCluskey is not in the scope of CS2100, but knowing it will strengthen your understanding of K-map, and appreciate why K-map is faster and easier.)

$T(A,B,C,D) = \Sigma m(0,1,2,4,5,9) + X(6,11,14,15)$.

PI chart:

✓0: 0000	✓0,1: 000- ✓0,2: 00-0 ✓0,4: 0-00	0,1,4,5: 0-0- 0,2,4,6: 0--0
✓1: 0001 ✓2: 0010 ✓4: 0100	✓1,5: 0-01 1,9: -001 ✓2,6: 0-10 ✓4,5: 010- ✓4,6: 01-0	
✓5: 0101 ✓6: 0110 ✓9: 1001	6,14: -110 9,11: 10-1	
✓11: 1011 ✓14: 1110	11,15: 1-11 14,15: 111-	
✓15: 1111		

These are PIs.

These are not PIs because they contain solely don't-cares.

Reduced PI Chart:
Collecting the 4 PIs, we draw this reduced PI chart:

PI	Minterms						Don't-cares	
	0	1	2	4	5	9	6	11
0,1,4,5: 0 - 0 - ($A' \cdot C'$)	■	■		■	■			
0,2,4,6: 0 - - 0 ($A' \cdot D'$)	■		■	■			■	
1,9: -0 0 1 ($B' \cdot C' \cdot D$)		■				■		
9,11: 1 0 - 1 ($A \cdot B' \cdot D$)						■		■

Look under the minterms columns to find any column containing just one dot.
Since minterm m2 is covered only by $A' \cdot D'$, so $A' \cdot D'$ must be an EPI.
Likewise, minterm m5 is covered only by $A' \cdot C'$, so $A' \cdot C'$ must be an EPI.
Minterms m0, m1, m2, m4, m5 are covered by these 2 EPIs, leaving only minterm m9, which can be covered either by $B' \cdot C' \cdot D$ or $A \cdot B' \cdot D$.

4. A circuit takes in four inputs *K,L,M,N* and generates 3 outputs *X,Y,Z* as follow:

$X(K,L,M,N)$ = 1 if $KL = MN$, or 0 otherwise,
where *KL* and *MN* are 2-bit unsigned integers.

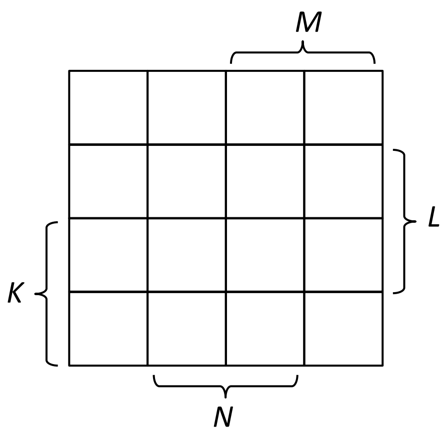
$Y(K,L,M,N)$ = 1 if $KL \leq MN$, or 0 otherwise,
where *KL* and *MN* are 2-bit unsigned integers.

$Z(K,L,M,N)$ = 1 if $KLM < LMN$, or 0 otherwise,
where *KLM* and *LMN* are 3-bit unsigned integers.

For parts (a) – (c) below, you may assume that the input 0000 will not occur.

(a) Fill in the truth table for the circuit. Write 'd' for don't cares.

(b) Fill in the K-maps of *X*, *Y* and *Z* using the layout given below.



(c) Write out the simplified SOP expressions of *X*, *Y* and Z.

(d) After designing the circuit according to the simplified SOP expressions in (c), if you feed the input 0000 into it, what will be the outputs?

Only answers for (c) and (d) are shown:

(c) $X = K' \cdot L \cdot M' \cdot N + K \cdot L' \cdot M \cdot N' + K \cdot L \cdot M \cdot N$

$Y = M \cdot N + K' \cdot N + K' \cdot M + L' \cdot M$

$Z = K'$

(d) Input *KLMN* = 0000; output *XYZ* = 001.