

Homework 1 Solutions

Prof. Loh

CS3220 - Processor Design - Spring 2005

1. Boolean functions, K-maps, Minimization, Gates

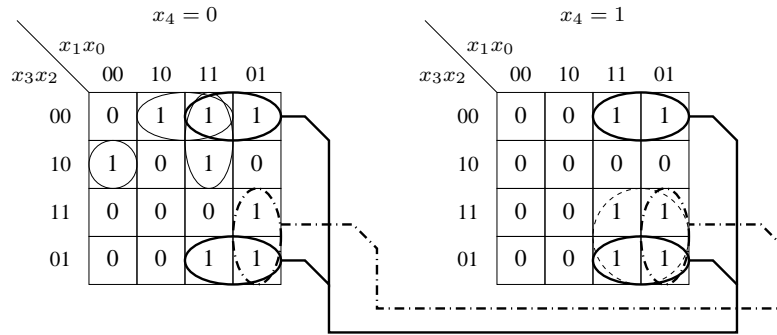
Consider the a five-input function $f(x_4, x_3, x_2, x_1, x_0)$ where the inputs form a 5-bit unsigned value (i.e. $X = x_4x_3x_2x_1x_0 = \sum_{i=0}^4 x_i \cdot 2^i$). Let $f()$ be a function that evaluates to true if X is either a prime number (0 and 1 are not prime) or if X is a Fibonacci number (1,2,3,5,8,...).

The minterms are: 1,2,3,5,7,8,11,13,17,19,21,23,29,31.

The truth table for this function is:

x_4	x_3	x_2	x_1	x_0	$f()$
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

(a) Use a 5-variable Karnaugh map to minimize $f()$.



$$\overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot x_1 + \overline{x_4} \cdot \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot x_0 + x_4 \cdot x_2 \cdot x_0 + \overline{x_3} \cdot x_0$$

(b) Use the Quine-McCluskey method to minimize $f()$.

00001 ✓	000-1 ✓	00- -1 ✓	-0- -1 *
00010 ✓	00-01 ✓	-00-1 ✓	
01000 *	-0001 ✓	-0-01 ✓	
	0001- *		
00011 ✓		-0-11 ✓	
00101 ✓	00-11 ✓	-01-1 ✓	
10001 ✓	0-011 *	- -101 *	
	-0011 ✓	10- -1 ✓	
00111 ✓	001-1 ✓		
01011 ✓	0-101 ✓	1-1-1 *	
01101 ✓	-0101 ✓		
10011 ✓	100-1 ✓		
10101 ✓	10-01 ✓		
10111 ✓	-0111 ✓		
11101 ✓	-1101 ✓		
	10-11 ✓		
11111 ✓	101-1 ✓		
	1-101 ✓		
	1-111 ✓		
	111-1 ✓		

The prime implicants are written in **bold** with a star (*). In this problem, all prime implicants are required, and so the final equation is:

$$01000 + 0001- + 0-011 + - -101 + 1-1-1 + -0- -1$$

$$= \overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot x_1 + \overline{x_4} \cdot \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot x_0 + x_4 \cdot x_2 \cdot x_0 + \overline{x_3} \cdot x_0$$

(c) Use DeMorgen's theorem to write a product-of-sums form for $\overline{f()}$.

$$\overline{(\overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot x_1 + \overline{x_4} \cdot \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot x_0 + x_4 \cdot x_2 \cdot x_0 + \overline{x_3} \cdot x_0)}$$

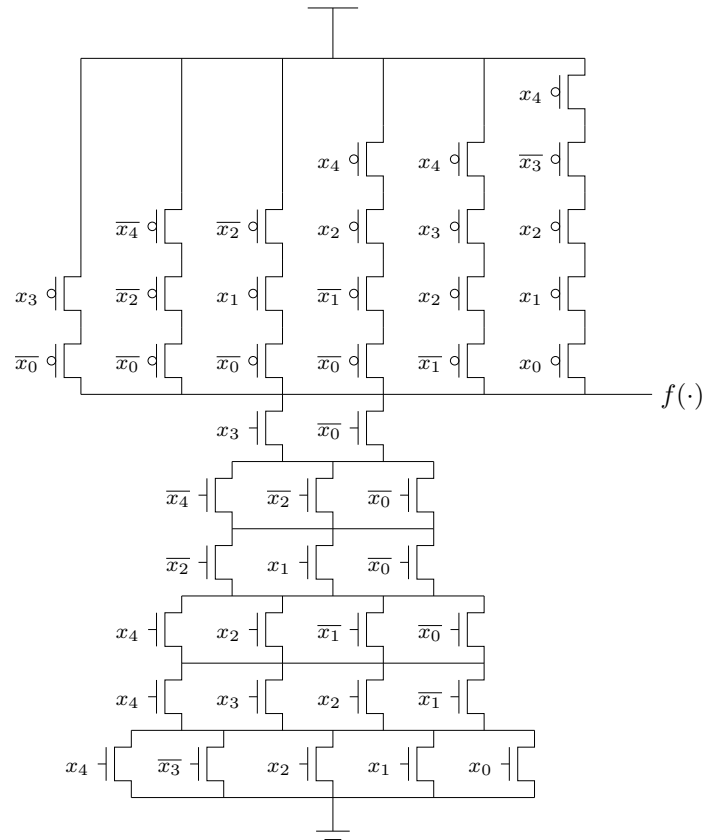
$$\Downarrow$$

$$\overline{(\overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0}) \cdot (\overline{x_4} \cdot x_3 \cdot \overline{x_2} \cdot x_1) \cdot (\overline{x_4} \cdot \overline{x_2} \cdot x_1 \cdot x_0) \cdot (x_2 \cdot \overline{x_1} \cdot x_0) \cdot (x_4 \cdot x_2 \cdot x_0) \cdot (\overline{x_3} \cdot x_0)}$$

$$\Downarrow$$

$$(x_4 + \overline{x_3} + x_2 + x_1 + x_0) \cdot (x_4 + x_3 + x_2 + \overline{x_1}) \cdot (x_4 + x_2 + \overline{x_1} + \overline{x_0}) \cdot (x_2 + x_1 + \overline{x_0}) \cdot (\overline{x_4} + \overline{x_2} + \overline{x_0}) \cdot (x_3 + \overline{x_0})$$

(d) Use the solution from 1(c) to implement a CMOS gate for $f(\cdot)$. Note that the NMOS pulldown network will pull down if $\overline{f(\cdot)}$ evaluates to true, which will result in a gate that implements the function $f(\cdot)$.



2. Parallel Prefix Adders

In class we discussed a Look-Ahead Carry Adder where the carry-propagate circuit was implemented as a binary tree. As the circuit makes its way up the tree, the wire distances that must be traveled between nodes increases. Sometimes it makes sense to collect multiple inputs together, perform a larger computation, and then move on to the next node; that is, use a ternary- or even a quaternary-tree.

- (a) Consider a node for a ternary parallel-prefix adder tree. The inputs from the “bottom” are pgk_L, pgk_M, pgk_R and the input from the top is pgk_{in} . The outputs out the bottom to the subtrees are PGK_L, PGK_M, PGK_R and the output up to the next root is PGK_{out} . Fill in three “truth tables” for the output PGK_{out}, PGK_L and PGK_M (you don’t need to do PGK_R since it is always equal to pgk_{in}). Input values should be either K, P, G or X (don’t care) and the output values should be K, P or G.

Each tables should be of the form (I’ve filled in an arbitrary row from each as an example):

pgk_R	pgk_{in}	PGK_M
K	X	K
P	K	K
P	P	P
P	G	G
G	X	G

pgk_M	pgk_R	pgk_{in}	PGK_L
K	X	X	K
P	K	X	K
P	P	K	K
P	P	P	P
P	P	G	G
P	G	X	G
G	X	X	G

pgk_L	pgk_M	pgk_R	PGK_{out}
K	X	X	K
P	K	X	K
P	P	K	K
P	P	P	P
P	P	G	G
P	G	X	G
G	X	X	G

- (b) Draw a 9-bit look-ahead carry adder using a ternary parallel-prefix tree (for each node, you can just use blank blocks like the figure on the next page). Like the example in class, annotate each wire in your 9-bit parallel prefix tree with a P, G or K assuming inputs of $x = 011011100$ and $y = 010110010$.

