Digital Systems (Sheet #1)

Marius Gavrilescu

1.1 1 Since $a_1 \oplus a_2 \oplus \dots a_n$ represents the parity of $\sum_{i=1}^{n} a_i$, and addition is both commutative and associative, xor is also associative and commutative.

Not idempotent, since $1 \oplus 1 = 0$. Has a unit: $a \oplus 0 = a$. Does not have a zero.

2 $a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$

3 $a \oplus b = (\neg a \vee \neg b) \wedge (a \vee b)$

4 $u = \neg(a \wedge b)$.
$v = \neg(a \wedge \neg(a \wedge b)) = \neg a \vee (a \wedge b)$.
$w = \neg b \vee (a \wedge b)$.
$z = (\neg(\neg a \vee (a \wedge b))) \vee (\neg(\neg b \vee (a \wedge b))) = (a \wedge (\neg(a \wedge b))) \vee (b \wedge (\neg(a \wedge b))) = (a \wedge \neg b) \vee (b \vee \neg a) = a \oplus b$.

2.4 Map:

a \ c \ b \ d	0	0	0	0
	1	1	0	1
	0	0	0	0
	1	1	0	1

The given expression is the minimal disjunctive form, and disjunctive forms can be implemented directly in multi-input nand gates. So this is also the smallest implementation in nand gates.

a \ c \ b \ d	1	1	1	1
	0	0	1	0
	1	1	1	1
	0	0	1	0

$\neg z = (\neg a \wedge \neg c) \vee (a \wedge c) \vee (b \wedge d)$ and $z = (a \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg d)$.

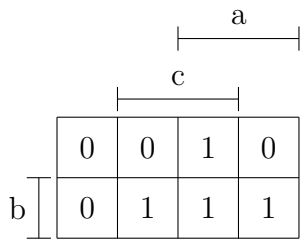
We can read a product of sums form from a Karnaugh map in the same way we would read a sum of products, but covering all zeroes in the matrix instead of all ones (thus getting a sum

of products for $\neg z$), then negating all variables and replacing all $\wedge$s with $\vee$s and all $\vee$s with $\wedge$s.

Any product of sums form can be implemented in a straightforward manner as a two level nor gate circuit: the terms in one parentheses go into one multi-input nor gate, and the outputs of all nor gates go into a single multi-input nor gate on the second level. In this example, the inputs of the first level nor gates are $(a, c; \neg a, \neg c; \neg b, \neg d)$.

2.5 The Karnaugh map is a checkerboard pattern. Both the sum of products and product of sums forms will have 2^{n-1} parentheses, each of which has n terms, where n is the number of variables.

3.1 Map:



$z = (c \wedge a) \vee (b \wedge a) \vee (c \wedge b)$. Three nand gates on the first level (c,a;b,a;c,b), one three-input nand gate on the second level.

3.4 Chain n half adders, such that the first one adds the first bit of the input and a constant 1, and every other adder adds a bit of the input with the carry out of the previous adder.

3.5 Range is from -2^{n-1} to $2^{n-1} - 1$. Largest value is a zero followed by $n - 1$ ones, smallest value is all ones. All numbers from 0 to $2^{n-1} - 1$ have the same representation.

3.6 If bs is not all zeroes, then prepending it to a number will increase its value. So for every length there is a bs of that length (all zeroes) satisfying the condition.

$bin(take\ k\ as) = bin(as)$ as long as the first k bits of as are all zeroes.

3.7 By the same reasoning as above, if as is positive, bs has to be all zeroes. Similarly, if as is negative, bs has to be all ones.

$twoc(take\ k\ as) = twoc(as)$ as long as either as is negative and the first k bits of as are all ones, or as is positive and the first k bits of as are all zeroes.

3.9 $twoc(as \oplus bs) - twoc(as) - twoc(bs) = bin(as \oplus bs) - last(as \oplus bs)2^n - bin(as) - bin(bs) + last(as)2^n + last(bs)2^n$. As the difference between $bin(as \oplus bs)$ and $bin(as) + bin(bs)$ is either 0 or 2^n and all other terms are multiples of 2^n, the difference between $twoc(as \oplus bs)$ and $twoc(as) + twoc(bs)$ is also a multiple of 2^n.

Since the difference is always a multiple of 2^n, and we do not have a $(n + 1)$-th bit, we can simply use a binary adder to add twos complement numerals.

3.10 $bin(as) + bin(map\ not\ as)$ is $2^{\text{length}(as)} - 1$. $twoc(as) + twoc(map\ not\ as)$ is -1.

We have $twoc(as \ominus bs) = twoc(as) - twoc(bs) = twoc(as) + twoc(map\ not\ bs) + 1$. Thus, we can use an adder where every bit of bs is negated and the carry in of the first full adder is set to 1 instead of 0.

For the extension, we can xor every bit of bs with the control input instead of negating them, and set the carry in of the first full adder to the control input.