

**ECE 465, Fall 2009**  
**Homework 1**

1. Prove that the single parity code method for a set of information bits  $x_{n-1}, x_{n-2}, \dots, x_0$  can detect any number of odd errors in the information bits and the check bit  $x_c$ . Recall that the check bit is determined as:

$$x_c = x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0$$

Note also that the method used for detecting if there is an error in the received bits  $\hat{x}_{n-1}, \hat{x}_{n-2}, \dots, \hat{x}_0, \hat{x}_c$  is to check if the following equality holds:

$$\hat{x}_c = \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0$$

If the equality holds, then there is no error (with high probability), otherwise an error is detected. You can use an approach similar to the one used in class to prove that this parity scheme will not detect errors that are even in number. You can also use the fact that the number of 1's in the correct information plus check bits is even.

**Solution:**

*First Proof:*

If there is an error in bit  $\hat{x}_i (i = 0, \dots, n-1)$ , the received bit can be written as  $\hat{x}_c = \bar{x}_c = x_c \oplus 1$ , where  $\bar{x}_i$  is the complemented form of  $x_i$ .

Assume that there are odd errors in the received bits. There are two cases:

Case 1: All the errors are in information bits.

In this case, we know that the number of  $x_i$  changed to  $\bar{x}_i$  is odd, and each  $\bar{x}_i = x_i \oplus 1$ .

So RHS of the check:

$$\begin{aligned} \hat{x}_c &= \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0 \\ &= x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 \oplus 1 \oplus 1 \oplus \dots \oplus 1 \text{ here, the number of 1's is odd} \\ &= x_c \oplus 1 = \bar{x}_c \neq x_c = \hat{x}_c = \text{LHS} \end{aligned}$$

Thus, the check fails and an error is detected.  $\square$

Case 2: One error occurs in check bit  $x_c$ , and the other errors occur in the information bits.

In case 2, we know that  $x_c$  is changed to  $\bar{x}_c = x_c \oplus 1$ , and the number of  $x_i$  changed to  $\bar{x}_i$  is even,  $\bar{x}_i = x_i \oplus 1$

So RHS of the check:

$$\begin{aligned} \hat{x}_c &= \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0 \\ &= x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 \oplus 1 \oplus 1 \oplus \dots \oplus 1 \text{ the number of 1's is even} \\ &= x_c \oplus 0 = x_c \neq \bar{x}_c = \hat{x}_c = \text{LHS} \end{aligned}$$

Thus, in case 2 as well, the check fails, and an error is detected.  $\square$

If the number of errors in received bits is even, there are also two cases.

Case 1: all the errors are in information bits.

Similar to case 2 above, we can obtain that in the RHS of the check

$$\begin{aligned}\hat{x}_c &= \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0 \\ &= x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 \oplus 1 \oplus 1 \oplus \dots \oplus 1 \text{ the number of 1's is even} \\ &= x_c \oplus 0 = x_c = \hat{x}_c = \text{LHS}\end{aligned}$$

Thus the check passes, error cannot be detected. □

Case 2: One error occurs in check bit  $x_c$ , and the other errors occur in information bits.

Similar to case 1, we can obtain that in the RHS of the check:

$$\begin{aligned}\hat{x}_c &= \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0 \\ &= x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 \oplus 1 \oplus 1 \oplus \dots \oplus 1 \text{ the number of "1s" is odd} \\ &= x_c \oplus 1 = \bar{x}_c = \hat{x}_c \text{ (since the check bit } x_c \text{ is changed to } \bar{x}_c \text{)} = \text{LHS}\end{aligned}$$

Thus again the check passes, and the error cannot be detected. □

In conclusion, the single parity code method can detect any number of odd errors in the information bits and the check bit  $x_c$ , but it cannot detect an even number of errors.

*Second (Alternate) Proof:* There are many ways of proving the same result. Provided here is an alternate proof for establishing that any odd number of errors are detected by the single parity code (the even error case was proved in class—that the single parity code cannot detect an even number of errors).

Let  $k$  be the no of 1's in a given correct code word  $x_c x_{n-1} \dots x_0$ . By definition of  $x_c$ ,  $k$  is even. Consider an odd number  $m$  of errors. For an error that changes a 0 → 1, the number of 1's increase by 1 (+1 1's), and for the 1 → 0 error, the number of 1's decrease by 1 (-1 1's). Thus after a generic error, we can say that the number of 1's change by ± 1. Thus after  $m$  errors, the number of 1's change by ± 1 repeated  $m$  times, i.e., by  $(\pm 1)(\pm 1)\dots(\pm 1)$  [ $m$  times]. Combining the various + 1's and -1's in the above sequence, we will have  $p$  number of +1's and  $q$  number of -1's, where  $1 \leq p, q \leq m$ , and  $p+q = m$  (since  $p+q$  is the total number of errors). This means that the number of 1's change to  $k + p$  (corresponding to  $p$  number of +1's) –  $q$  (corresponding to  $q$  number of -1's). Considering that  $p+q = m$ , and that  $m$  is odd, the only possible odd, even status combinations for  $p, q$  are:  $p$  is odd and  $q$  is even →  $p-q$  is odd, or  $p$  is even and  $q$  is odd →  $p-q$  is odd. Thus  $p-q$  is always odd, and the total number of 1's after any  $m$  errors =  $k + (p-q)$  = an even number  $k$  + an odd number  $p-q$  = an odd number. Thus the number of 1's in the code word after an odd number of errors in it is odd, and thus the check:

$$\hat{x}_c = \hat{x}_{n-1} \oplus \hat{x}_{n-2} \oplus \dots \oplus \hat{x}_0$$

will *always fail* since:

- (a) If all the 1's ( $k+p-q$  of them) are in the info bits, then the info bits have an odd number of 1's and the RHS is thus 1 (by the known property of the XOR computation), while (by our assumption that all the 1's are in the info bits),

$$\text{the LHS } \hat{x}_c = 0. \quad \square$$

(b) Or,  $\hat{x}_c = 1$ , and thus the info bits have an even number of 1's (total number of 1's in the code word,  $k+p-q$ , is odd). Thus the RHS of the above check is 0 (by the known property of the XOR computation), while

the LHS  $\hat{x}_c = 1$ .

Since the check always fails when there is an odd number of errors, an odd number of errors in the code word is always detected.  $\square$

2. Prove that for two implicants  $g$  and  $h$  of a function  $f$  to be *adjacent* (two implicants are adjacent if their product terms are the same in all variables except one, which occurs in complemented form in one implicant and in uncomplemented form in the other), the number of 1's in their ternary notations need to differ by exactly one. (Note that, as the statement asserts, this condition is *necessary* but not *sufficient* for  $g$  and  $h$  to be adjacent).

**Solution:**

*Proof:*

Assume that  $g$  and  $h$  differ in the  $j^{\text{th}}$  bit, then:

$$g = g_{n-1}g_{n-2} \cdots g_{j+1}g_j g_{j-1} \cdots g_0$$

$$h = h_{n-1}h_{n-2} \cdots h_{j+1}h_j h_{j-1} \cdots h_0$$

$$= g_{n-1}g_{n-2} \cdots g_{j+1}g_j' g_{j-1} \cdots g_0$$

Assume the number of 1's in  $g$ 's ternary notation is  $m$ , there are two cases:

Case 1:

$$g_j = 0$$

we obtain  $\bar{g}_j = 1$ , so there are  $m+1$  1s in  $h$ 's ternary notation.  $\square$

Case 2:

$$g_j = 1$$

then  $\bar{g}_j = 0$  and there are  $m-1$  1's in  $g_{n-1}g_{n-2} \cdots g_{j+1}g_j g_{j-1} \cdots g_0$

So there are  $m-1$  1's in  $h$ 's ternary notation.  $\square$

In conclusion, the number of 1s in  $g$ 's and  $h$ 's ternary notation differ by exactly one.

3. Prove that in QM's tabular method for forming PIs, if an implicant cannot be combined with (i.e., is not adjacent to) any other implicant in its column, then it is a PI. **Hint:** Use the fact that any implicant covers or contains  $2^i$  minterms, where  $i \geq 0$ .

**Solution:**

*Proof:*

We first show that in the 1<sup>st</sup> stage (PI formation stage) of QM, an implicant  $g$  cannot be combined with another implicant in a different column from its own column, say, col.  $i$ , (in which all implicants cover  $2^i$  minterms [MTs]). Assume that the implicant  $g$  in column  $i$  can be combined with another implicant  $h$  in column  $j$ , where  $0 \leq i \leq n-1$ ,  $0 \leq j \leq n-1$  and  $i \neq j$  and there are  $n$  columns in QM's tabular form.

Then,  $g + h$  contains  $2^i + 2^j$  MTs. Because  $2^i + 2^j = 2^i(1 + 2^{j-i})$  (where we assume without loss of generality that  $j > i$ ; the same conclusions would hold for  $i > j$ ) and  $1 + 2^{j-i} \neq 0$  or  $2$ , it contradicts the fact that any implicant covers or contains  $2^m$  MTs,  $m > 0$  thus,  $g$  and  $h$  cannot be combined into an implicant. It means that an implicant  $g$  in column  $i$  cannot be combined with any other implicants which are not in  $i$  column. So if an implicant  $g$  cannot be combined with other implicant in its column, then  $g$  cannot be combined with any other implicant in any column.

We now show that  $g$  has to be a PI. Suppose it is not. Then it must be covered by some other PI  $h$ . Since  $g$  has  $2^i$  MTs (by our assumption),  $h$  has at least  $2^{i+1}$  MTs (otherwise it cannot cover  $g$ ). Suppose first that  $h$  has exactly  $2^{i+1}$  MTs. Then it must include another implicant  $p$  that is disjoint from  $g$  (i.e., without any common MTs) and has  $2^i$  MTs. The only way 2 disjoint PIs,  $g$  and  $p$ , each with  $2^i$  MTs can be contained in another implicant ( $h$  in this case) with  $2^{i+1}$  MTs, is if  $g$  and  $p$  are adjacent (since, otherwise, there is no implicant that contains both  $g$  and  $p$  and exactly  $2^{i+1}$  MTs, and this is also easily proved, and is left it to you as an exercise). But this means that  $g$  can be combined with  $p$ , and this contradicts the given assumption that  $g$  cannot be combined with any other implicant in its column.

Now suppose  $h$  covers  $2^r$  MTs, with  $r > i+1$ . In this case, it must cover an implicant  $q$  that covers a set of  $2^{i+1}$  MTs that contains the MTs of  $g$ . Then the case of  $q$  is the same as the above case for  $h$  ( $h$  has exactly  $2^{i+1}$  MTs), and we reach a similar conclusion that  $g$  must be adjacent to another implicant  $p$  that covers the other half MTs of  $q$ . This is another contradiction.

Thus we reach contradictions for all cases if we start with the assumption that  $g$  is not a PI. Thus  $g$  has to be a PI.  $\square$

4. (a) Determine a minimized SOP expression for the  $\log_2$  function of a 4-bit number  $N$ . Specifically, this function gives the rounded-up integer value (i.e., the *ceiling*) of  $\log_2 N$ . Thus, e.g.,  $f(7) = 3$ ,  $f(8) = 3$ ,  $f(9) = 4$ ,  $f(15) = 4$ . The minimized expression for each output bit can be obtained using either the K-map or the QM technique. 75

(b) Implement and simulate your gate-based design obtained above using the Quartus II CAD

software as specified below.

- Choose the schematic capture tool in Quartus to specify your design.
- Perform simulations based upon the input  $_le$  provided by the TA.
- Device family to be used for the project is Cyclone which is selected by default in Quartus.
- Report: (i) the logic/hardware cost of the Quartus implementation in terms of the total number of gate inputs, and (ii) the circuit delay obtained by Quartus simulation. This is the maximum of all delays you obtain by simulating the different inputs provided to you.
- Submit the printed Quartus schematic of the design.
- Submit the design file to the TA by email.
- Submit the simulation output file to the TA by email.

### Solution:

#### 4(a)

Assume that  $a_3, a_2, a_1, a_0$  are the four bits of number  $N$ , in which  $a_3$  is the most significant bit and  $a_0$  is the least significant bit.  $b_2, b_1, b_0$  are the output bits with  $b_2$  the most significant and  $b_0$  the least significant bit.

The truth table is as follows:

$N$	$a_3$	$a_2$	$a_1$	$a_0$	$b_2$	$b_1$	$b_0$
0	0	0	0	0	X	X	X
1	0	0	0	1	0	0	0
2	0	0	1	0	0	0	1
3	0	0	1	1	0	1	0
4	0	1	0	0	0	1	0
5	0	1	0	1	0	1	1
6	0	1	1	0	0	1	1
7	0	1	1	1	0	1	1
8	1	0	0	0	0	1	1
9	1	0	0	1	1	0	0
10	1	0	1	0	1	0	0
11	1	0	1	1	1	0	0
12	1	1	0	0	1	0	0
13	1	1	0	1	1	0	0
14	1	1	1	0	1	0	0
15	1	1	1	1	1	0	0

$$b_0 = \sum m(2,5,6,7,8) + d(0)$$

the k-map for  $b_0$  is:  $b_0 = \overline{a_3}a_2a_0 + \overline{a_3}a_1\overline{a_0} + \overline{a_2}a_1a_0$

$a_1a_0 \setminus a_3a_2$

	00	01	11	10
00	X 0			1 8
01		1 5		
11		1 7		
10	1 8	1 6		

$$b_1 = \sum m(3,4,5,6,7,8) + d(0)$$

$b_1 = \overline{a_3}a_2 + \overline{a_3}a_1a_0 + \overline{a_2}a_1a_0$

$a_1a_0 \setminus a_3a_2$

	00	01	11	10
00	X 0	1 4		1 8
01		1 5		
11	1 3	1 7		
10		1 6		

$$b_2 = \sum m(9,10,11,12,13,14,15) + d(0)$$

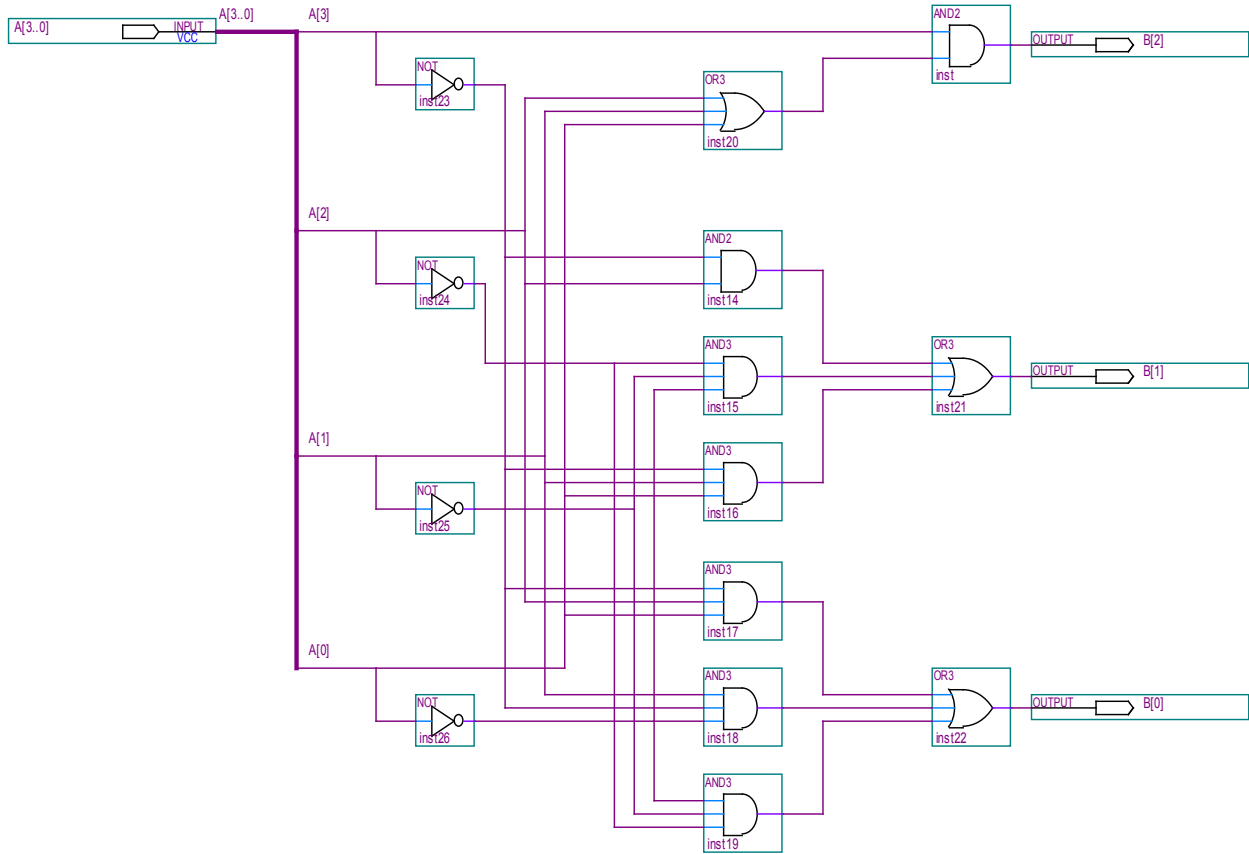
$b_2 = a_3a_2 + a_3a_0 + a_3a_1$

$a_1a_0 \setminus a_3a_2$

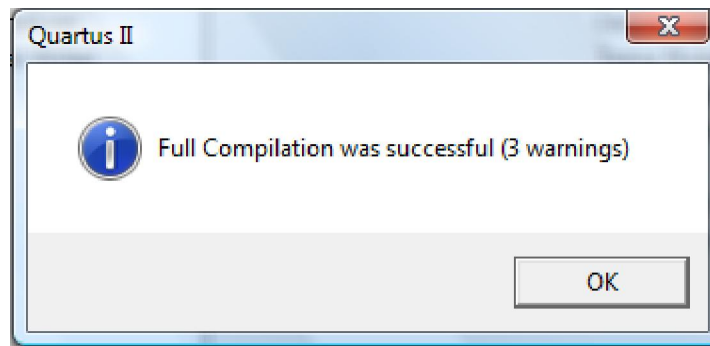
	00	01	11	10
00	X 0		1 12	
01			1 13	1 9
11			1 15	1 11
10			1 14	1 10

4(b)

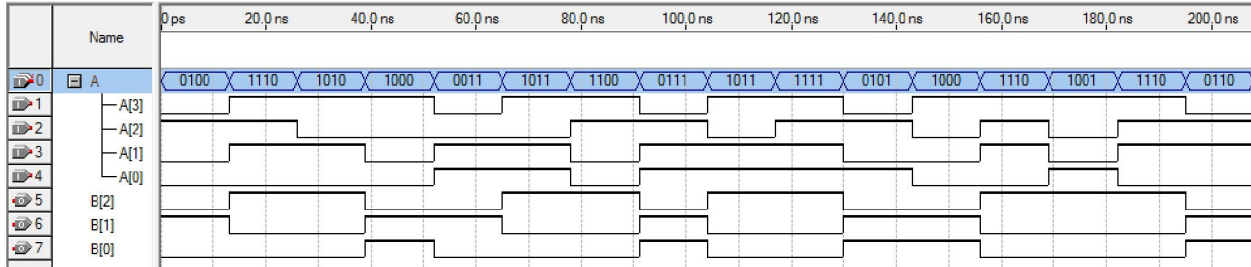
Schematic diagram:



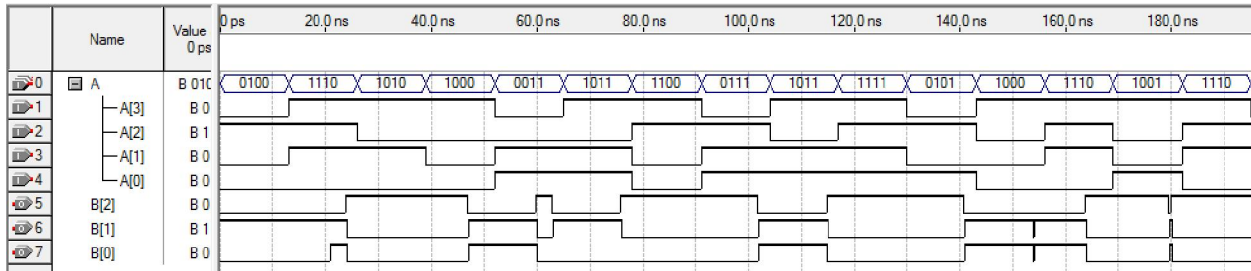
Compilation:



### Functional simulation output:



### Timing simulation output:



### Propagation Delay:

Propagation delay info is generated by **Classical Timing Analysis** tool in Quartus II. The worst case propagation delay path is from A[0] to B[0] and delay is 11.139 ns.

tpd					
	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	11.139 ns	A[0]	B[0]
2	N/A	None	11.137 ns	A[0]	B[1]
3	N/A	None	10.990 ns	A[3]	B[0]
4	N/A	None	10.989 ns	A[3]	B[1]
5	N/A	None	10.889 ns	A[0]	B[2]
6	N/A	None	10.791 ns	A[2]	B[0]
7	N/A	None	10.782 ns	A[2]	B[1]
8	N/A	None	10.742 ns	A[3]	B[2]
9	N/A	None	10.539 ns	A[2]	B[2]
10	N/A	None	8.008 ns	A[1]	B[0]
11	N/A	None	7.992 ns	A[1]	B[1]
12	N/A	None	7.755 ns	A[1]	B[2]