ERROR DETECTION AND CORRECTION

Error detection and correction (EDAC) techniques are used to ensure that data is correct and has not been corrupted, either by hardware failures or by noise occurring during transmission or a data read operation from Memory. There are many different error correction codes in existence. The reason for the different codes being used in different applications has to do with the historical development of the data storage, the types of data errors occurring, and the overhead associated with each of the error detection techniques. We discuss some of the popular techniques here with details of Hamming code.

PARITY CODE

We have discussed parity generation and checking in details in section 4.8. When a word is written into memory, each parity bit is generated from the data bits of the byte it is associated with. This is done by a tree of Exclusive-OR gates. When the word is read back from the memory, the same parity computation is done on the data bits read from the memory, and the result is compared to the parity bits that were read. Any computed parity bit that doesn't match the stored parity bit indicates that there was at least one error in that byte (or in the parity bit itself). However, parity can only detect an odd number of errors. If an even number of errors occur, the computed parity will match the read parity, so the error will go undetected. Since memory errors are rare if the system is operating correctly, the vast majority of errors will be single-bit errors, and will be detected.

Unfortunately, while parity allows for the detection of single bit errors, it does not provide a means of determining which bit is in error, which would be necessary to correct the error. Thus the data needs to be read again if an error is detected. Error Correction Code (ECC) is an extension of the parity concept..

CHECKSUM CODE

This is a kind of error detection code used for checking a large block of data. The checksum bits are generated by summing all the codes of a message and are stored with data. Usually the block of data summed is of length 512 or 1024 and the checksum results are stored in 32 bits that allows overflow. When data is read the summing operation is again done and checksum bits generated are matched with the stored one. If they are unequal then an error is detected. Obviously, it can fool the detection system if error occurring at one place is compensated by the other.

CYCLE REDUNDANCY CODE (CRC)

This is a more robust error checking algorithm than the previous two. The code is generated like this. Take a binary message and convert it to a polynomial then divide it by another predefined polynomial called the *key*. The remainder from this division is the CRC. This is stored with the message. Upon reading the data, memory controller does the same operation i.e. divides the message by the same key and compares with CRC stored. If they differ, the data is wrongly read or stored. Not all keys are equally good. The longer the key, the better is the error checking. On the other hand, the calculations with long keys can get pretty involved. Two of the polynomials commonly used are:

CRC-16 = x16 + x15 + x2+ 1
CRC-32 = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1
Usually, series of Ex-OR gates are used to generate CRC code.

HAMMING CODE

Introduced in 1950 by R.W. Hamming, this scheme allows one bit in the word to be corrected, but is unable to correct events where more than one bit in the word is in error. These multi-bit errors can only be detected, not corrected, and therefore will cause a system to malfunction. Hamming code uses parity bits discussed before but in a different way. For *n* number of data bits, number of parity bits required here if *m* then

$$2^m \geq m + n + 1$$

In the meory word, (i) all bit positions that are of the form 2^i are used as parity bits (like 1,2,4,8,16,32…..) and (ii)the remaining positions are used as data bits(like 3,5,6,7,9,10,11,12,13,14,17,18…….)

Thus code will be in the form of

P1 P2 D3 P4 D5 D6 D7 P8 D9 D10 D11 ……….

Where P1,P2,P4,P8…. are parity bits and D3,D5,D6,D7….are data bits

We discuss Hamming Code generation with an example. Consider the 7-bit data to be coded is 0110101. This requires 4 parity bits in position 1,2,4,8 so that Hamming coded data becomes 11 bit long. To calculate the value of P1 we check parity of zeroth binary locations of data bits. This is shown in 3rd row of Fig. x-1 for this example. Zeroth locations are the places where address ends with an 1. These are D3, D5, D9 and D11 for 7 bit data. Since we have total odd number of 1s in these 4 positions P1=1. This is calculated as done in case of parity generation (refer to section 4.8) by series of Ex-OR gates through equation

$$P1 = D3 \oplus D5 \oplus D9 \oplus D11$$

Similarly for P2, we check locations where we have 1 in address of the 1st bit i.e. D3, D6, D7, D10 and D11. Since there are even number of 1s, P2=1. Proceeding in similar manner and examining parity of 2nd and 3rd position we get P4=0 and P8=0

	0001 P1	0010 P2	0011 D3	0100 P4	0101 D5	0110 D6	0111 D7	1000 P8	1001 D9	1010 D10	1011 D11
Data word (without parity)			0		1	1	0		1	0	1
P1	1		0		1		0		1		1
P2		0	0			1	0			0	1
P4				0	1	1	0				
P8								0	1	0	1
Data with parity	1	0	0	0	1	1	0	0	1	0	1

Fig. x-1 Calculation of parity bits

Next we discuss how error in a hamming coded data is detected and if it is in single bit how it is corrected. We continue with the previous example and consider the data is incorrectly read in position D11 so that 11 bit coded data is 10001100100. Fig. x-2 describes the detection mechanism. First of all, we check the parity of zeroth position and find it is to be even. Since P1=1, this parity check fails and this is equivalent to generating a parity bit at the output (last column) following equation

$$\text{Parity P1 check bit} = D3 \oplus D5 \oplus D9 \oplus D11 \oplus P1$$

This is similar to parity checker in section 4.8. Note that, in addition to data bits we have also included the corresponding parity bit to the input of Ex-OR gate tree. Proceeding similarly for other positions we find except for P4 all other parity checks fail. Note that, even a single failure detects an error. However, to correct the error, we use the output of last column 1011 (in the order P8 P4 P2 P1) and find its decimal equivalent which is 11. So the data of location 11, which is D11 needs to be corrected.

	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Parity check	Parity bit
Received data word	1	0	0	0	1	1	0	0	1	0	0		
P1	1		0		1		0		1		0	Fail	1
P2		0	0			1	0			0	0	Fail	1
P4				0	1	1	0					Pass	0
P8								0	1	0	0	Fail	1

Fig. x-2 Error detection and correction

Note that, this method detects error in more than one position unlike the first method but overhead is more. In simple parity method we add 1 additional bit for 7 bit data while this is 4 here. Also note, further increasing this overhead, error in more than one position can also be corrected. However, more than one bit error is unlikely for memory read. With overhead for 1 bit correction, if there occurs error in more than 1 bit positions the data needs to be read once again from the memory.

SELF TEST

1. Can parity code detect even number of errors?
2. What is the full form of CRC?
3. What is the advantage with Hamming code?
4. What is error detection-correction overhead?

ANSWER TO SELF TEST

1. No.
2. Cycle Redundancy Code.
3. It can detect as well as correct one bit error.
4. Additional bits to be included with data bits for this purpose.

PROBLEM

1. How many parity bits are needed to hamming code 16 bit data?
2. Find Hamming Code of data 11001.
3. Find Hamming code of data 1000111.
4. Show how an error in the 3rd data bit, if occurs is corrected for data of problem 3.

ANSWER TO ODD NUMBERED PROBLEMS

1. Five.
2. 11100001111.