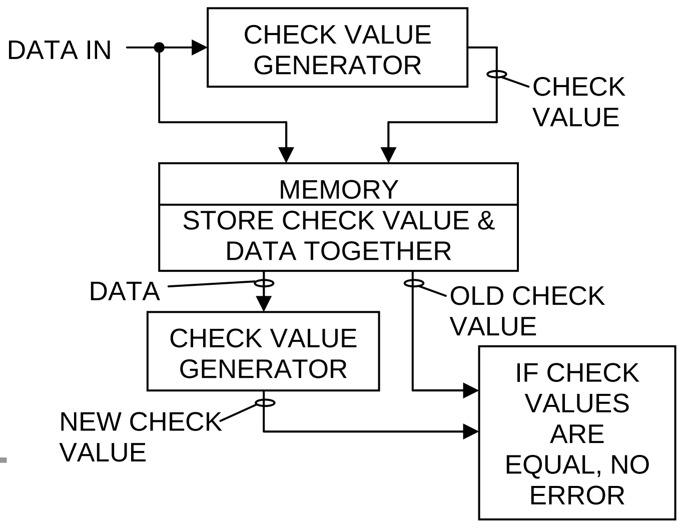
Error Detection and Correction

This worksheet is meant to describe the processes used to detect and correct errors at the bit level. After covering this material, you should be able to design a system to correct a single bit error and detect if a double bit error has occurred in any size word.

PROCESS:

A. Run an algorithm on a piece of data before it is stored to generate a check value.
B. Store the check value with the data.
C. When the piece of data is retrieved, run the algorithm on the data again to generate a second check value. Compare the second check value with the first. If they are equal, then no error occurred. A difference indicates an error.



A Poor Solution:

One possible solution, albeit a rather poor one, is to simply use a parity bit as the check value.

- An odd number of ones in the data value → check value=1
- An even number of ones in the data value → check value=0
- Limitations! Can only detect odd numbers of bit errors (even numbers of bit errors would test okay), and it provides no means for error correction

A Better Solution (Hamming Code):

Divide the stored data value into overlapping groups, each with its own parity bit. Each bit of the data value would belong to a unique set of groups.

- A flipped bit would generate a parity error in every group it was a member of.
- By seeing which groups had erroneous parity bits and matching those groups to the bit that belonged only to those groups, we could detect and correct the flipped bit.
- Limitation! Double bit errors would still cause problems.

Example: Create an error detection scheme for a 4-bit value consisting of the bits D_3, D_2, D_1, and D_0.

Step 1: Design a set of bit groups for the four bits D_3, D_2, D_1, and D_0 such that each bit belongs to a unique set of groups. For example:

Sample Group Assignment:
Group 2: D_3, D_1, and D_0
Group 1: D_3, D_2, and D_0
Group 0: D_3, D_2, and D_1

$\Rightarrow$

Parity Bits:
$P_2 = D_3 \oplus D_1 \oplus D_0$
$P_1 = D_3 \oplus D_2 \oplus D_0$
$P_0 = D_3 \oplus D_2 \oplus D_1$

$\Rightarrow$

D_3 belongs to groups 0, 1, and 2
D_2 belongs to groups 0 and 1
D_1 belongs to groups 0 and 2
D_0 belongs to groups 1 and 2

Step 2: Generate the check value as the group of parity bits P_0, P_1, and P_2.

Step 3: Upon retrieval, generate a new check value and see if $P_0' = P_0$, $P_1' = P_1$, and $P_2' = P_2$. If there are differences, the bits that are different identify which data bit (or sometimes which parity bit) was flipped. (The "primes" indicate the new check values.)

Example (continued): Let's continue the example by putting some data into the system. Try to store the binary value 1011.

$D_3 = 1$
$D_2 = 0$
$D_1 = 1$
$D_0 = 1$

Generate Parity Bits:

$P_2 = D_3 \oplus D_1 \oplus D_0 = 1 \oplus 1 \oplus 1 = 1$
$P_1 = D_3 \oplus D_2 \oplus D_0 = 1 \oplus 0 \oplus 1 = 0$
$P_0 = D_3 \oplus D_2 \oplus D_1 = 1 \oplus 0 \oplus 1 = 0$

Values Stored in Memory:

Data				Parity		
D_3	D_2	D_1	D_0	P_2	P_1	P_0
1	0	1	1	1	0	0

Now assume D_1 has flipped during storage. When the data is retrieved, we get:

Values Retrieved from Memory:

Data				Parity		
D_3	D_2	D_1	D_0	P_2	P_1	P_0
1	0	0	1	1	0	0

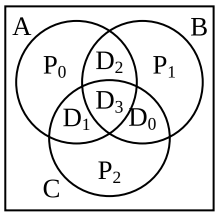
Generate NEW Parity Bits:

$P_2' = D_3 \oplus D_1 \oplus D_0 = 1 \oplus 0 \oplus 1 = 0$
$P_1' = D_3 \oplus D_2 \oplus D_0 = 1 \oplus 0 \oplus 1 = 0$
$P_0' = D_3 \oplus D_2 \oplus D_1 = 1 \oplus 0 \oplus 0 = 1$

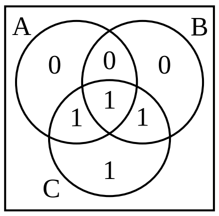
Parity errors in groups 0 & 2. ($P_2' \neq P_2$ and $P_0' \neq P_0$) D_1 is only bit in exactly those groups.

Since the parity errors are in groups 0 and 2, then the bit that flipped must be in exactly groups 0 and 2. This indicates data bit D_1. Therefore, flip D_1 from a 0 back to a 1, and you have corrected data.

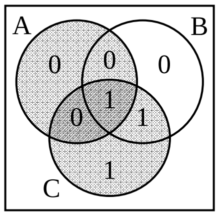
The above example can sometimes be better explained using a Venn diagram. The circle A represents group 0; the circle B represents group 1, and the circle C represents group 2. The sum of ones in a circle must be an even number. Note that the group membership for the data bits matches that contained in each circle.



Original Grouping



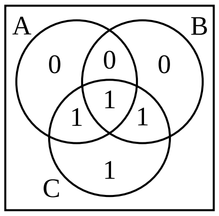
Data/Parity Stored



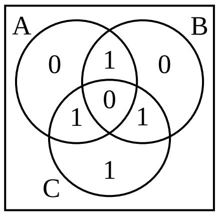
Parity error in A & C

Note that it is possible to have an error in the parity bits. In this case, since each parity bit belongs to only one group, then it should be easy to identify the error as a parity bit.

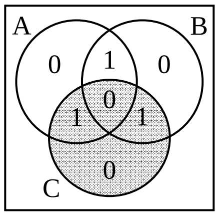
PROBLEM: What happens if there are two errors? In the figures below, both D_2 and D_3 have flipped. ***This ends up looking like a parity error only in circle C implying that it is P_2 that's in error!***



Data/Parity Stored



D_2 and D_3 in error

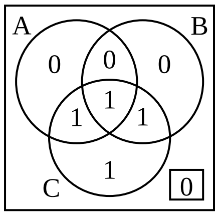


Attempted correction

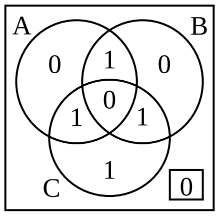
An Even Better Solution:

We need to correct the problem with double errors in our error correction scheme. We can detect that an error has occurred, but we cannot correct it properly. We can solve this problem by adding one more bit that acts as a parity check for all seven data and parity bits. If after we perform a correction the parity bit for the whole diagram is wrong, then we know that a double error has occurred. We cannot recover from a double error, but at least we won't correct it wrongly.

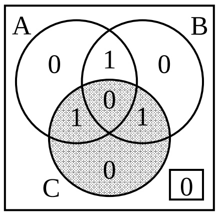
Let's pick up where the previous example left off by adding a parity bit (in the lower right corner of the Venn diagrams below) that is the parity of the seven stored bits, D_3, D_2, D_1, D_0, P_2, P_1, and P_0. This additional parity bit is stored with the rest of the data.



Data/Parity Stored



D_2 and D_3 in error

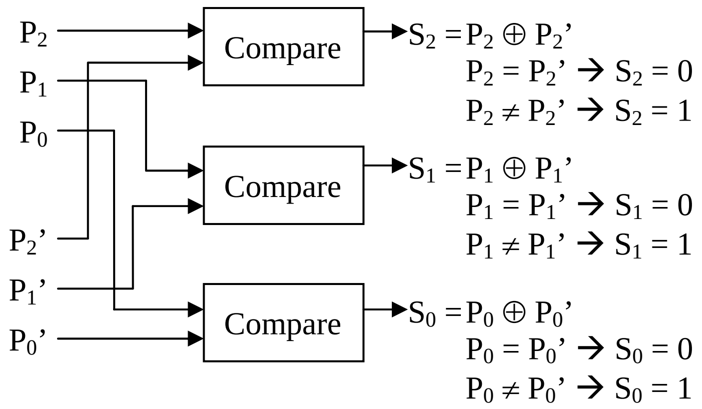


Attempted correction

Notice that in the third figure (the one all of the way to the right), the parity is wrong AFTER the correction. This means the error was not a single bit error and it will remain uncorrectable.

Syndrome Word: Automating the Process

To automate the process of determining which groups have parity errors, we need to compare each stored parity bit with the corresponding parity bit generated after retrieving the data, i.e., compare P_n with P_n'. The diagram to the right represents how this is done using the exclusive-or to generate S_2, S_1, and S_0 for a system with three parity bits such as the one we have in the previous example.


P_2
P_1
P_0
P_2'
P_1'
P_0'
Compare
Compare
Compare
$S_2 = P_2 \oplus P_2'$
$P_2 = P_2' \rightarrow S_2 = 0$
$P_2 \neq P_2' \rightarrow S_2 = 1$
$S_1 = P_1 \oplus P_1'$
$P_1 = P_1' \rightarrow S_1 = 0$
$P_1 \neq P_1' \rightarrow S_1 = 1$
$S_0 = P_0 \oplus P_0'$
$P_0 = P_0' \rightarrow S_0 = 0$
$P_0 \neq P_0' \rightarrow S_0 = 1$

The book calls the combination of all of the S_n bits the ***syndrome word***. The value of a syndrome word identifies the state of the data:

- A syndrome word of all zeros indicates no error has occurred.
- A syndrome word with a single bit set to one indicates that one of the parity bits is in error.
- A syndrome word with two or more bits set to one indicates that one of the data bits is in error.

Continuing our previous example of four data bits and three parity bits, the following list shows how each possible syndrome word (S_2 - S_1 - S_0) corresponds to a data or parity error.

$000_2 \rightarrow$ No error	$010_2 \rightarrow P_1$ in error	$100_2 \rightarrow P_2$ in error	$110_2 \rightarrow D_0$ in error
$001_2 \rightarrow P_0$ in error	$011_2 \rightarrow D_2$ in error	$101_2 \rightarrow D_1$ in error	$111_2 \rightarrow D_3$ in error

Note that there are three bits in the syndrome word which makes for $2^3 = 8$ possible values. This corresponds to the 1 error-free condition, the 3 parity error conditions, and the 4 data error conditions.

Expanding to More than Four Bits

So how can this method be expanded to more than four data bits? The answer is to find out how many groups and therefore parity bits are needed to have each data bit be a member of a unique set of groups. The answer lies in the syndrome word which must be capable of identifying each possible error condition. (Note that we'll add the final additional parity bit for double error detection later.)

Possible values of the syndrome word for M data bits using K parity bits:

- 1 pattern for the error-free condition: all zeros
- K patterns to represent errors in each of the K parity bits: exactly one bit set to 1
- M patterns to represent errors in each of the M data bits: two or more bits set to 1

Since there are K parity bits, the syndrome word must also have K bits. Therefore, there are 2^K possible patterns of ones and zeros in the syndrome word. Since there must be at least as many syndrome words as there are conditions listed above, we get the following expression:

$$2^K \geq 1 + K + M$$

$$2^K - 1 \geq K + M$$

The table shown to the right lists some of the numbers of parity bits required for specific numbers of data bits to perform single-error correction (SEC). To obtain double error detection (DED), a single additional bit is required.

Number of data bits (M)	Number of parity bits (K)	$K + M$	$2^K - 1$
4	3	7	7
8	4	12	15
16	5	21	31
32	6	38	63
64	7	71	127
128	8	136	255

An Example with More Bits

Now let's design an SEC/DED system for 8 data bits. Using the previous table we see that 4 parity bits are going to be required for SEC. Let's begin by assigning the bit patterns for each of the possible syndrome words by doing the following.

- Begin by assigning all zeros to error-free condition
- Identify each pattern where exactly one bit set to 1 and use those to identify parity errors
- Assign each of the 8 remaining data bits to one of the remaining syndrome word patterns

$0000_2 \rightarrow$ No errors	$0100_2 \rightarrow P_2$ in error	$1000_2 \rightarrow P_3$ in error	$1100_2 \rightarrow D_7$ in error
$0001_2 \rightarrow P_0$ in error	$0101_2 \rightarrow D_1$ in error	$1001_2 \rightarrow D_4$ in error	$1101_2 \rightarrow$ Not used
$0010_2 \rightarrow P_1$ in error	$0110_2 \rightarrow D_2$ in error	$1010_2 \rightarrow D_5$ in error	$1110_2 \rightarrow$ Not used
$0011_2 \rightarrow D_0$ in error	$0111_2 \rightarrow D_3$ in error	$1011_2 \rightarrow D_6$ in error	$1111_2 \rightarrow$ Not used

A change in parity bit P_n due to an error is going to set the corresponding position in the syndrome word to a 1. For example, if $P_2 \neq P_2'$, then the syndrome word is going to have bit S_2 set, i.e., 0100_2. Going back to our description of the data bits belonging to groups, there are four groups represented by the parity bits P_0, P_1, P_2, and P_3. We can figure out which groups each of the data bits belongs to by seeing which ones contain which parity bits in their syndrome word. For example D_0 has bits P_0 and P_1 set. Therefore, it belongs to groups 0 and 1.

- In group 0 (represented by P_0) we have data bits D_0, D_1, D_3, D_4 and D_6.
- In group 1 (represented by P_1) we have data bits D_0, D_2, D_3, D_5 and D_6.
- In group 2 (represented by P_2) we have data bits D_1, D_2, D_3, and D_7.
- In group 3 (represented by P_3) we have data bits D_4, D_5, D_6, and D_7.

This gives us our parity expressions for P_0, P_1, P_2, and P_3.

- $P_0 = D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6$
- $P_1 = D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6$
- $P_2 = D_1 \oplus D_2 \oplus D_3 \oplus D_7$
- $P_3 = D_4 \oplus D_5 \oplus D_6 \oplus D_7$