Company Confidential

PROJECT STRETCH

File Memo #22

SUBJECT: The Description of an Error Detecting and Correcting Code and its Possible Application to Stretch

BY: H. K. Wild

DATE: February 29th 1956

The inclusion of multiple check bits in a binary word to provide a means for the detection and correction of single bit errors is described by R. W. Hamming in an article entitled "Error Detecting and Correcting Codes" appearing in V 29 of the Bell Telephone Technical Journal, pp 147-160, April 1950. It appears that the method discussed would be singularly adaptable to the problem of single error correcting the information channels in the Stretch machine.

Hamming forms an error detecting code by a unique arrangement of check bits and information bits in the binary word. Each check bit, depending upon its position in the word, is associated with certain of the information bits. When checking, all check bits must be in agreement with their respective information bits to insure correctness. If there is a disagreement, the location of the bit can be ascertained uniquely from the check bits which signified the error.

The number of check bits necessary to detect and correct a single error can be represented by the relationship:

$$2^k = m + k + 1$$

where m is the number of information bits and
k is the number of check bits.

Therefore, if m is 4, the number of check bits required for k would be 3. If m is 60, the number of check bits required to detect and correct single errors would be 7.

As an example, let us first consider a 4 bit information word checked by 3 bits. The result is a 7 bit single error correcting code word. The check bits occupy all bit positions whose decimal representation is a power of 2, i.e., 1, 2, 4, 8 etc. (Decimal numbering of bit positions is from left to right).

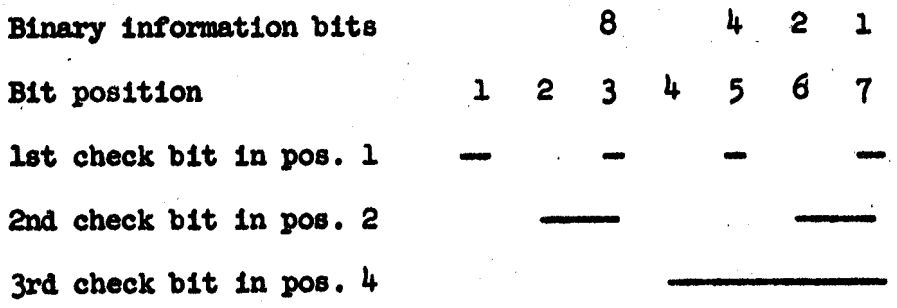

FIG. 1

The information bits occupy positions 3, 5, 6, 7 with the units position in 7 as indicated. The check bits occupy positions 1, 2 and 4. The check bit in position 1 checks 1, 3, 5 and 7. The check bit in position 2 checks 2, 3 and 6, 7. The position of the check bit determines which information bit positions it is associated with. The first check bit in position 1 checks all bit positions whose binary representation has a 1 in the units position, i.e. 1, 3, 5 etc. The second check bit in position 2 checks all bit positions whose binary representation has a 1 bit in the 2's column. The third check bit in position 4 checks all bit positions whose binary representation has a 1 bit in the 4's column.

The following table can be constructed:

Check Bit	In Position	Checks Position
1	1	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, etc.
2	2	2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, etc.
3	4	4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, etc.
4	8	8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31, etc.

FIG. 2

The value of a check bit is not dependent upon the value of any other check bit. Check bit 1 does not check position 2, 4 or 8. Nor does 2 check 1, 4 or 8.

Considering again a 4 bit information word checked by 3 bits, the following table can be constructed. Odd parity count shall be assumed.

Binary column			8		4	2	1	
Bit position	1	2	3	4	5	6	7	Decimal Value
	1	1	0	1	0	0	0	0
	0	0	0	0	0	0	1	1
	1	0	0	0	0	1	0	2
	0	1	0	1	0	1	1	3
	0	1	0	0	1	0	0	4
	1	0	0	1	1	0	1	5
	0	0	0	1	1	1	0	6
	1	1	0	0	1	1	1	7
	0	0	1	1	0	0	0	8
	1	1	1	0	0	0	1	9
	0	1	1	0	0	1	0	10
	1	0	1	1	0	1	1	11
	1	0	1	0	1	0	0	12
	0	1	1	1	1	0	1	13
	1	1	1	1	1	1	0	14
	0	0	1	0	1	1	1	15

FIG. 3

The decimal 3 is written in the binary form as 0101011. Suppose that a 4 bit were picked up in position 5 so that what should be a decimal 3 appears as 0101111. Applying our check bits, we find that our parity count does not agree for the 1st and third check bits. Supposing a checking system with 3 output lines corresponding to positions 1, 2 and 4 we would find outputs on lines 1 and 4 signifying an error in position 5. Suppose what should be 0101011 appears as 0111011, i.e. a bit was picked up in position 3. Then parity would not agree for positions 1 and 2 and this would signify an error in position 3. Note that an error in a check bit will be signified by a single error indication by the particular check bit which is in error and by that alone.

To discriminate between single and double errors, a total parity bit is included which gives the parity count for the entire word. This parity bit does not enter into the calculations when one is determining the number of bits necessary to detect and correct single bit errors but is added to the coded word.

The chart labeled Fig. 4 on the attached sheet described the error detection and correction method extended to a word of 60 information bits and 7 check bits. One overall parity check bit is added for a total word of 68 bits. The chart shows the association of each check bit with the positions it checks by underscoring the positions checked by each check bit.

It can rapidly be seen from this chart which check bits will indicate an error for any particular bit position. It is not quite as obvious that this chart also represents the logical circuitry necessary to generate the check bits or check the word for error and that the number of logical connections necessary to do this is also represented.

For convenience, this will be illustrated with the 7 bit arrangement described in Fig. 1. The addition of the total parity check bit brings the total number of bits to 8 bits. The total parity bit is in position 8.

			8		4	2	1	
	1	2	3	4	5	6	7	8
Total parity bit	—	—	—	—	—	—	—	—
1st check bit	—		—		—		—	
2nd check bit		—	—			—	—	
3rd check bit				—	—	—	—	

FIG. 5

Fig. 6 is a schematic representation of a check or parity bit generator/checker utilizing trees of Exclusive OR's for the 7 bit code described in Fig. 5. Generation of the parity count for each check bit position is accomplished by combining those sections of the word with which the particular check bit is associated in Exclusive OR combinations. When generating the parity counts the check bit positions are held to 0. When checking, these positions should contain whatever parity count applies for the particular positions checked. For the higher order check bits, levels of the tree prior to the final level contain the results of parity counts of sections of the word pertaining the lower order check bits. These section counts can be utilized by the lower order check bit generators so that a complete tree is needed for only the first section of bits covered by that check bit. The parity count for the first section then should be combined with the parity counts of the other sections on the same level which are covered by the higher order check bits.

The total parity check can be obtained by determining the parity for all information bits by properly combining the parity counts of non-overlapping sections and combining this count with the parity of the generated check bits.

A double error is indicated if the check bits signify an error but the total parity check does not.

If the check bits indicate no error in the information bits and the overall parity check indicates an error, it is likely that it is in error and should be changed. There are particular multiple errors that can occur which can cause this indication. These unique multiple errors statistically have a lesser likelihood of occurrence than a single bit error and, therefore, it is highly probable if such an indication exists that the error is in the overall check bit.

When generating the check bits the parity counts appear on the respective output lines and are stored in the proper locations.

As a checker, the check bits and information bits are entered and if the word is correct there is no output. If there is a disparity caused by a single bit error, the output lines are decoded to force the complement into the proper bit position of the register.

It can be seen that the chart shown in Fig. 5 is merely another and less cumbersome method of describing the parity bit generator/checker of Fig. 6, excluding the decoder, inverters and special error detecting circuits.

To generate the parity count for n bit positions requires n-1 Exclusive OR networks. Referring to Fig. 4, it can be seen that a distinct tree is necessary for the first section of each order of the check bits. For the 2nd , 3rd, 4th and 5th check bits, the parity counts for other than the first section are available at the respective levels of the higher order check bit trees and can be combined by Exclusive OR's to obtain the proper parity count for each check bit. The total parity is obtained by combining the parity counts for the first section associated with each check bit and combining the result with the parity count of the check bits. The number of Exclusive OR's necessary for the 67 bit coded word can be obtained directly from Fig. 4.

For example, the 4th check bit checks 32 positions, 8-15, 24-31, 40-47, and 56-63. Positions 8-15 are the first section. n equals 8, therefore, the number of Exclusive OR's necessary is 7. The parity counts for the other three sections already exist because they are levels of trees formed for generating higher order check bits. The four section counts are combined by 3 Exclusive OR's to obtain the parity count for the 4th check bit for a total of 7 + 3 = 10 Exclusive OR's.

To obtain the 7 check bits require a total of 121 Exclusive OR's. The overall check bit requires 12 additional for a total of 133 Exclusive OR logical elements.

Referring to Fig. 6, a positive error indication is obtained if there is an output from the checker on one or more of the output lines and if the overall parity count does not agree with the overall parity bit. Double error detection occurs if the multiple parities do not agree but the overall parity agrees. This is also the case for multiple double errors.

It appears necessary to reroute the word through the checker again after correction has taken place to discover whether a triple or multiple odd error was the case.

It should be pointed out that when considering multiple errors the system described is not fail safe. It can be shown that there are certain multiple double errors involving both information and check bits which will not be detected. Also, for each combination of the 7 check bits in the 67 bit coded word there are associated 2^{53} configurations of the 60 information bits. Multiple errors could occur in the information channels to result in what is an apparently correct word but not the proper word. If total parity did not check, it might be assumed that the total parity bit is in error, which is statistically probable but could be erroneous. However, in most of these cases, it is necessary to both pick up and drop bits in the information to get this result.

I believe it can be safely said that the system will detect and correct all single bit errors, detect all double bit errors and most multiple bit errors.

To obtain a somewhat more powerful checking arrangement, the seventh check bit (see Fig.4) can be extended to cover positions 5 through 15 and 17 through 31 and still allow the checking system to signify uniquely the position in error through modifying the decoding procedure.

The circuitry discussed is used to describe a possible system utilizing the error correction principle. It does not necessarily represent the minimum amount of circuitry necessary or the optimum configuration.

H. K. Wild

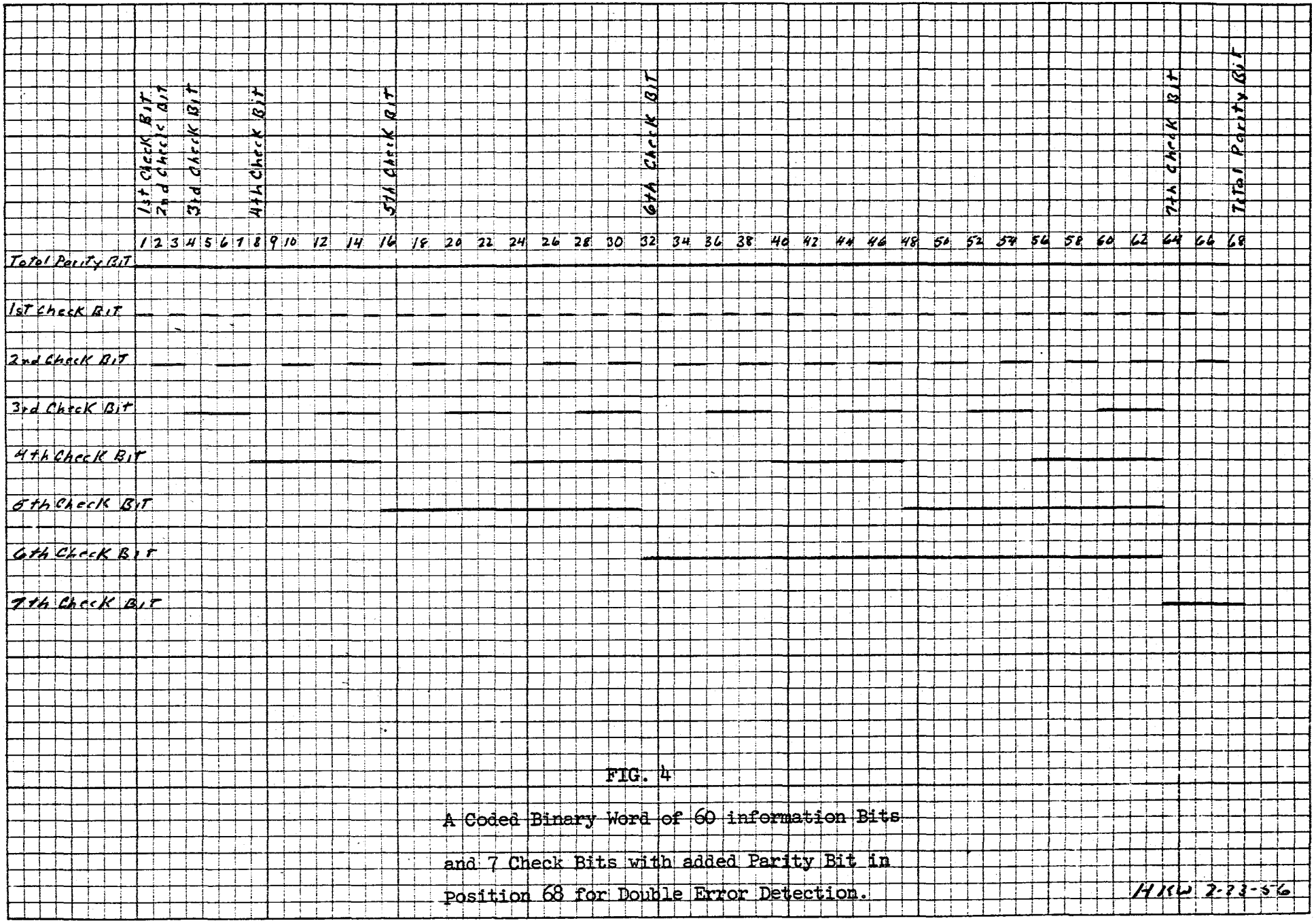

FIG. 4

A Coded Binary Word of 60 information Bits and 7 Check Bits with added Parity Bit in Position 68 for Double Error Detection.

HHW 2-23-56

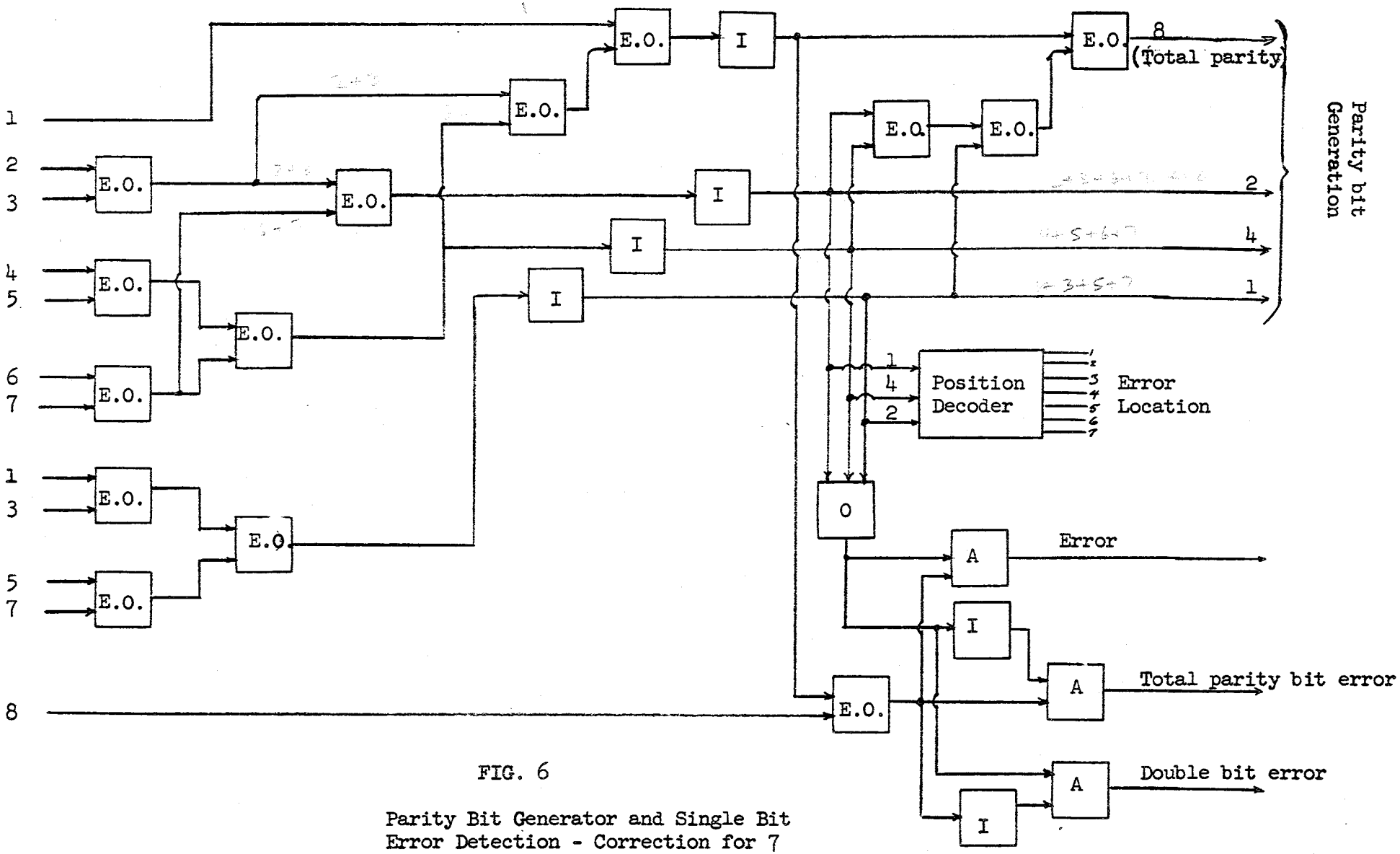

FIG. 6

Parity Bit Generator and Single Bit Error Detection - Correction for 7 Bit Coded Binary Word

HKW. 2-23-56