Greg Matthews
ECEC 520
May 29th, 2017

ECEC 520: Dependable Computing
Assignment II

1. **(20 points)** A simple technique for error correction is to pick the nearest code word as the correct word once an error has been detected. Develop a hardware design, that is a functioning digital circuit, that performs the error correction for a separable 3-of-6 code by choosing the nearest valid code word. Make the necessary assumptions and state them clearly in your answer. Points will be awarded, in part, based on the efficiency of your design.

Original Code	**3-of-6 Code**
000	000 111
001	001 110
010	010 101
011	011 100
100	100 011
101	101 010
110	110 001
111	111 000

The 3-of-6 code system has a Hamming Distance of 2, and therefore can detect all single bit flip errors, but cannot accurately correct all single bit flip errors. The circuit will perform correction of a detected fault in the 3-of-6 code by choosing the nearest valid code word, however there is the uncertainty of picking the wrong code word because H_d is 2. We will assume only 1 bit errors are possible, and also that the 3-of-6 Code is known before computation, to allow for error checking.

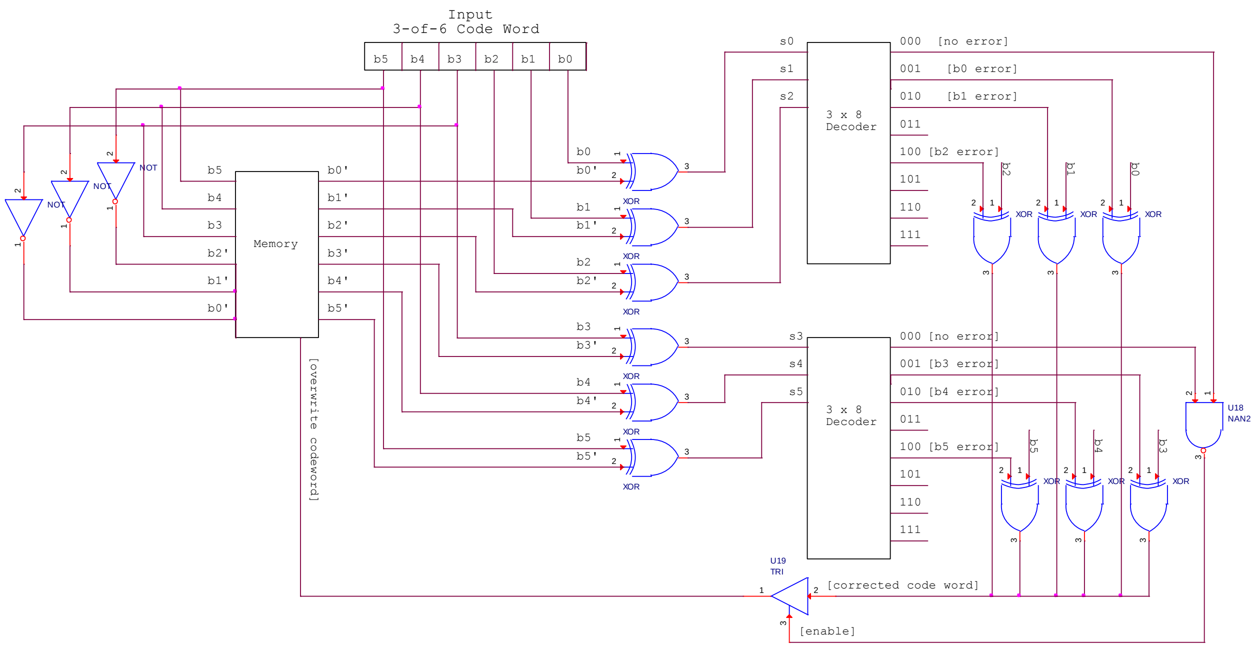
For our hardware design, we will need an encoder and a decoder system for our original code, with the assumptions that M = 3, and N = 6. The encoding process will be fairly straight forward because the code word is separable, making computation complexity and cost, relatively low. The encoding will be done by inverting the original code and producing the necessary 3 added code word bits. This will produce $\{b2, b1, b0\}$, which will then be appended to the original code to form the code word $\{b5, b4, b3, b2, b1, b0\}$, and be stored into memory.

The generated code bits $\{b2, b1, b0\}$, will need to be checked for correctness by XOR-ing them with their pre-computation value which are known to be correct. Supplying this output into a 3x8 Decoder will allow for the detection of errors, and because we are assuming only 1 bit flip errors are possible at any given time, the possible outputs of the decoder is:

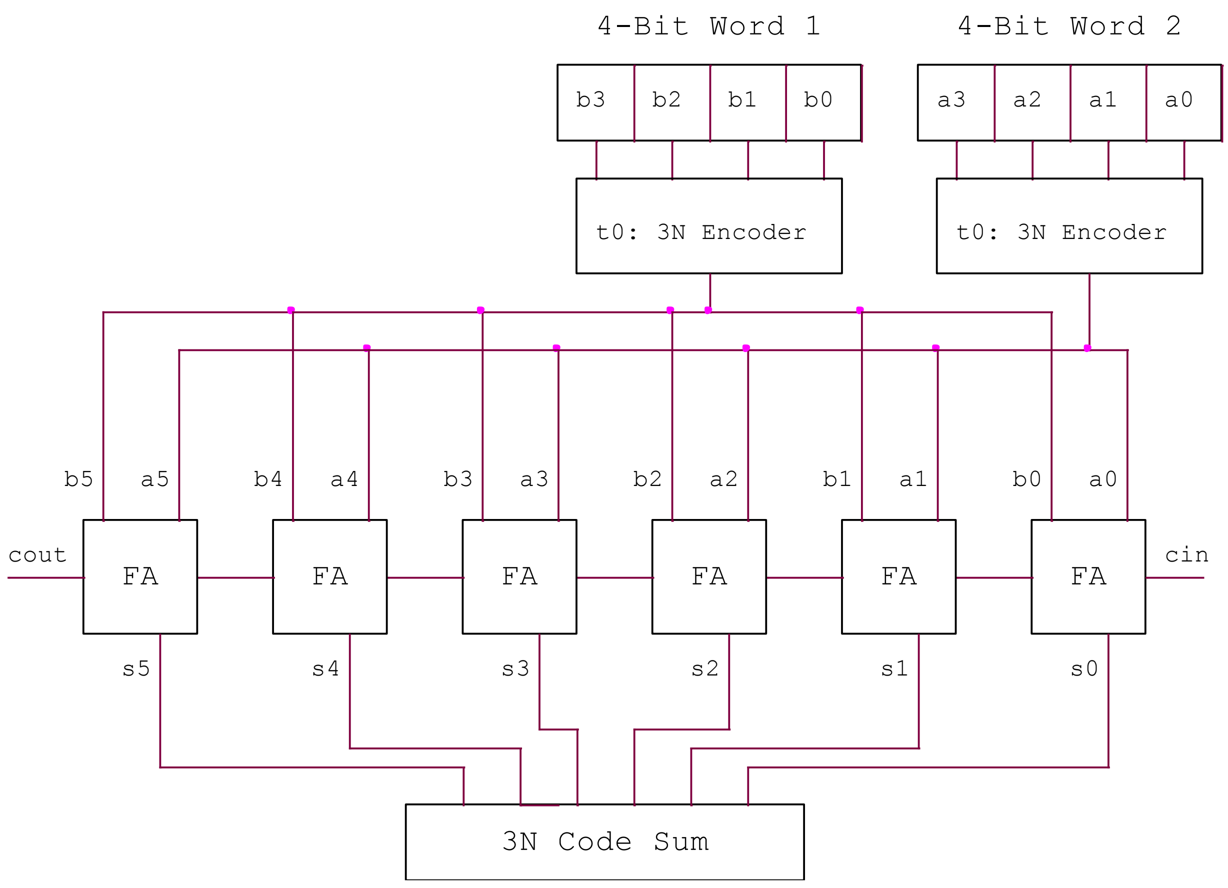
Decoder Input	**Decoder Output**
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	0

This will detect whether the error has arisen from $b2, b1,$ or $b0$. Likewise for code bits $\{b5, b4, b3\}$ we cannot assume that storing these bits into memory has not caused a bit flip error, therefore we check the stored values from memory with the pre-computation values that are known, and again supply the XOR of bits $\{b5, b4, b3\}$ into a 3x8 Decoder to detect any possible errors. Since only 1 bit flip errors are assumed, we will not have any redundant correction of errors with code bits $\{b2, b1, b0\}$. If an error is detected, the decoder will XOR the error bit with 1, essentially inverting the bit and correcting it. If there is no error detected in any of the bits, the two decoder outputs at line 000 will serve as an disable switch for overwriting the code word in memory. This is performed by using a combination of the two 000 line outputs from the decoder, connected to a NAND gate, and used as the enable switch for a tristate buffer that lets the corrected code word (if there's an error) be overwritten into memory. Therefore the system initially supplies the original code word into memory, and then detects and checks for errors. If there's a bit flip error, the system will overwrite the code word initially stored in memory. Extracting the data from the code word in memory is assumed to be obvious because the code word is separable.

The output of the system is therefore concluded to be either error free and correct, or have 1 bit flip error that has been corrected to the nearest valid code word. This is proven by the fact that a bit flip error would have either caused the number of 1's to be $m + 1$ or $m - 1$, and via error correction, the system would correct an erroneous 1/0 flip with its inverse, reverting the system back to having m 1's. The full implementation of the 3-of-6 code system can be seen below.



2. **(20 points)** Investigate the error detection capability of the following time redundancy approach when used on a ripple-carry adder. During the first addition, the operands are encoded using a 3N arithmetic code. During the second addition, the operands are encoded using a 5N arithmetic code. Will this scheme detect any single error that can occur in the adder, and why? Will this approach detect any double errors that can occur in the adder, and why?



Pictured above is an example of a 3N encode system for 2 4-bit words, however this system will apply for any arbitrary length data word. During the first addition which is performed at time t0, the system will encode the data words, which will apply an additional 2 bits of overhead. Then the system will perform the summation using the ripple carry adder implementation, creating the 3N code sum. During t0, we can already detect SSL faults in the system, for example:

Let Word1 = {1, 1, 0, 1} and Word2 = {0, 0, 1, 1}

The 3N Encode of Word1 = {1, 0, 0, 1, 1, 1} and the 3N encode of Word2 = {0, 0, 1, 0, 0, 1}

Summed together, S = {1, 1, 0, 0, 0, 0} = 48, which is evident to be divisible by 3, and thus is a valid code word.

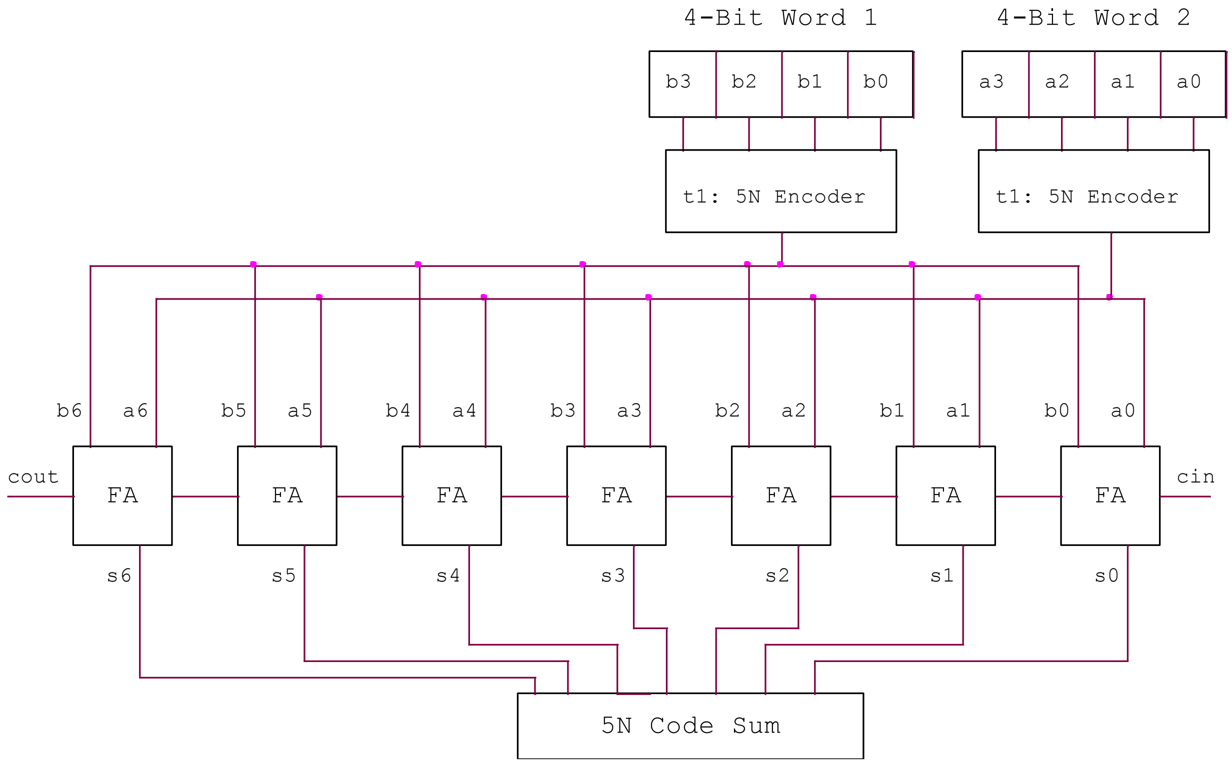
Say however there is a s/1 fault at bit S_1, then S = {1, 1, 0, 0, 1, 0} = 50, which is not divisible by 3 and is therefore not a valid code word. Therefore the system can currently detect SSL faults.

Let's assume there are 2 errors, a s/1 fault at bit S_1 and a s/1 fault at S_0, then

S = {1, 0, 0, 0, 1, 1} = 51, which is divisible by 3 and therefore will not detect the 2 faults, however in some instances it will detect 2 bit flip errors, because in some instances 2 stuck lines may not provide a multiple of 3 to the code word, however we cannot accurately assume then this can detect all double errors.

Table 1: 3N Encoding Double Error Detection

Valid Code Word (Dec)	**Valid Code Word (Bin)**	**Detect S_1 & S_0 s/1 error?**
48	0110000	No
51	0110011	No
54	0110110	Yes
57	0111001	Yes



At time t1, the second addition is performed however with a 5N encoding, whereby now the code has 3 additional bits of overhead. The system is portrayed in the figure above, and computes the 5N code sum. Going back to the previously given example, we currently can accurately detect SSL faults, but not 2. We will now check whether the addition of this time redundancy with a 5N encoding can perhaps give us detection of 2 faults. Using the same bit words as before, the 5N encoded words will now be:

Word1 = {1, 0, 0, 0, 0, 0, 1} and Word2 = {0, 1, 0, 1, 1, 0, 1}

Summed together, S = {1, 1, 0, 1, 1, 1, 0} = 110, which is divisible by 5, and therefore a valid code word.

Like before with the 3N encoded words, let's assume that there is a s/1 fault at S_1, then

S = {1, 1, 0, 1, 1, 1, 0} = 110, which is identical to the original sum since the SSL fault hasn't flipped any bits.

Now let's assume that there are 2 stuck lines, a s/1 fault at bit S_1 and a s/1 fault at S_0, then

S = {1, 1, 0, 1, 1, 1, 1} = 111, which isn't divisible by 5, and therefore detects the 2 stuck lines, however this is not always the case for much the same reason as using the 3N encoding; in some instances the 2 stuck lines may or may not create another multiple of 5. From the data of different valued code words provided in Table 2, it becomes apparent that we get full coverage of detecting S_1 & S_0 s/1 faults, assuming that at least one of the bits becomes flipped. This however does not conclusively answer whether this time redundancy will detect all double errors, because at most, having S_1 & S_0 s/1 will only add/sub 3 to the code word, which will sensitize detection of the 3N encoding, but not the 5N. Let's also experiment with S_2 & S_1 s/1 faults, which will sensitize the 5N encoding by adding/subtracting 5.

Table 2: 5N Encoding Double Error Detection

Valid Code Word (Dec)	**Valid Code Word (Bin)**	**Detect S_1 & S_0 s/1 error?**
110	1101110	Yes
115	1110011	No
120	1111000	Yes
125	1111101	Yes

Reviewing the data shown in Tables 3 and 4 that provide information on detecting S_2 & S_1 s/1 faults for both the 3N and 5N encodings, we can conclude that this system gets total coverage in detecting 2 stuck lines in the system, assuming that at least 1 stuck line changes the value of the code. This is evident in code word 54 from Table 3, and code word 120 from Table 4, which simply didn't detect the stuck lines because the fault did not change any of the bits in the code word, and therefore isn't regarded as an "error" per-say, which thus satisfies the conditions of "detecting double errors" as specified.

This time redundancy system of using a 3N and 5N encoding system can detect double errors, because if there is 2 errors in the system that make a 3N encoding a multiple of 3, and therefore doesn't detect the errors, this will not produce a respective multiple of 5 in the 5N encoding, which therefore will be detected, and vice versa. In conclusion, this system will accurately detect single and double errors.

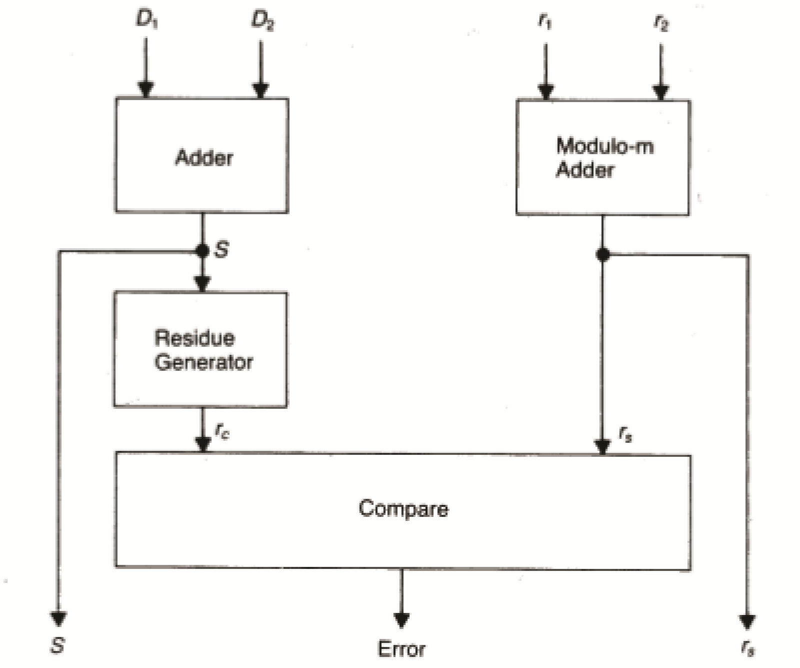
Table 3: 3N Encoding Double Error Detection

Valid Code Word (Dec)	**Valid Code Word (Bin)**	**Detect S_2 & S_1 s/1 error?**
48	0110000	Yes
51	0110011	Yes
54	0110110	No
57	0111001	Yes

Table 4: 5N Encoding Double Error Detection

Valid Code Word (Dec)	**Valid Code Word (Bin)**	**Detect S_2 & S_1 s/1 error?**
110	1101110	No
115	1110011	Yes
120	1111000	No
125	1111101	Yes

3. **(10 points)** Show that a residue check with the modulus $A = 2^a - 1$ can detect all errors in a group of $a - 1$ or fewer adjacent bits. Such errors are called burst errors of length $a - 1$ (or less).

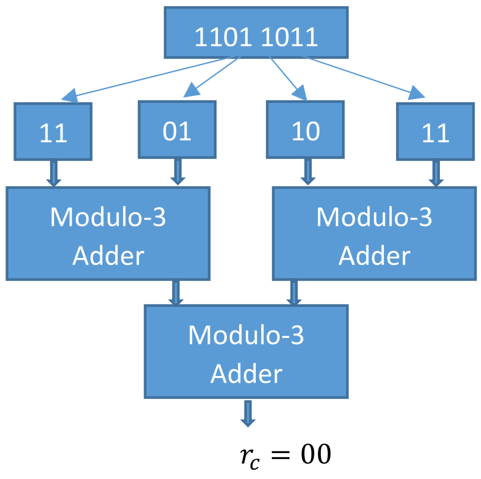


From the above residue code checking system, error detection is performed by checking the computed residue values from r_c and r_s, where r_c is computed by extracting the residue from the sum of data words $D1$ and $D2$, while r_s is computed by extracting the residue, of the 2 known residue values from $D1$ and $D2$ separately. This is possible because residue codes are invariant to the addition operation, and can therefore be handled separately as a good error checking algorithm. To check whether a residue code with a modulus of $A = 2^a - 1$ can detect errors in a grouping of $a - 1$ bits, we'll experiment with the operation of the system and see the output of an example input of:

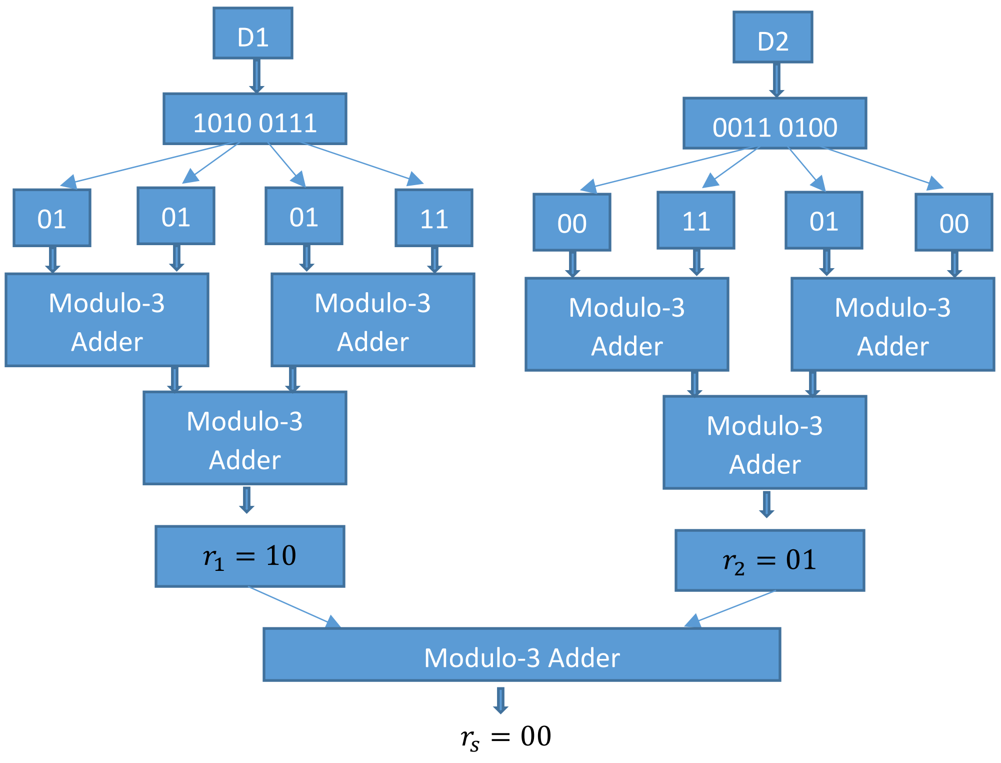
$D1 = 1010\ 0111$ and $D2 = 0011\ 0100$, were $A = 3$, and therefore $a = 2$.

Then $D1 + D2 = 1101\ 1011$

Computing r_c:



Computing r_s:



We can conclude that the system is working when no errors have occurred, as a result of r_c and r_s being equivalent. Now let's experiment with detecting $a - 1$ grouped erroneous bits, where a is 2, and therefore just 1 error bit. Let's assume that the sum of $D1 + D2$ has an erroneous bit flip in it's LSB, therefore $D1 + D2 = 1101\ 1011$. Going back through the algorithm;

$r_c = (D1 + D2)\ mod3 = 1101\ 1010\ Mod3 = 10$

$r_s = (D1\ mod3)\ modulu_{3add}\ (D2\ mod3)\ \ = 10\ modulo_{3add}\ 01 = 00$

Because $r_c \neq r_s$, the system can accurately detect this error with a grouping of 1 error bit. Now that we have successfully proved the base case, let's check higher order values of a.

Let's now assume that $A = 2^4 - 1 = 15$, where $a = 4$. Using the same data words $D1$ and $D2$ we get:

$r_c = (D1 + D2)\ mod15 = 1101\ 1011\ Mod15 = 1001$

$r_s = (D1\ mod3)\ modulu_{15add}\ (D2\ mod3)\ \ = 0010\ modulo_{15add}\ 0111 = 1001$

With no erroneous bits, the system is shown to work, to fully prove detection coverage, we must check all possible combinations of burst errors of 1 bit:

Table 6: Detecting Burst Errors of 1 in Residue Check System

Bits in Error:	**r_c value**	**r_s value**	**Detected?**
b0	1000	1001	Yes
b1	0111	1001	Yes
b2	1101	1001	Yes
b3	0001	1001	Yes
b4	1000	1001	Yes
b5	1011	1001	Yes
b6	0101	1001	Yes
b7	0001	1001	Yes

From Table 6, we can accurately state that the system is capable of detecting a burst errors of 1. Now let's assume that the sum of of $D1 + D2$ has 2 erroneous bit flips starting from its LSB, let's check all possible combinations of this burst error.

Table 7: Detecting Burst Errors of 2 in Residue Check System

Bits in Error:	**r_c value**	**r_s value**	**Detected?**
b0,b1	0101	1001	Yes
b1,b2	1011	1001	Yes
b2,b3	0101	1001	Yes
b3,b4	0000	1001	Yes
b4,b5	1010	1001	Yes
b5,b6	0111	1001	Yes
b6,b7	1100	1001	Yes

From Table 7, we can accurately state that the system is capable of detecting a burst errors of 3 . Now let's assume that the sum of of $D1 + D2$ has 3 erroneous bit flips starting from its LSB, therefore $D1 + D2 = 1110\ 1100$. Going back through the algorithm;

$r_c = (D1 + D2)\ mod15 = 1101\ 1100\ Mod15 = 1010$

$r_s = (D1\ mod3)\ modulu_{15add}\ (D2\ mod3)\ = 0010\ modulo_{15add}\ 0111 = 1001$

$r_c \neq r_s$, and therefore the system can detect this burst error of bits b0, b1, and b2. Let's also check this for a grouping of 3 bit errors for all possible arrangements:

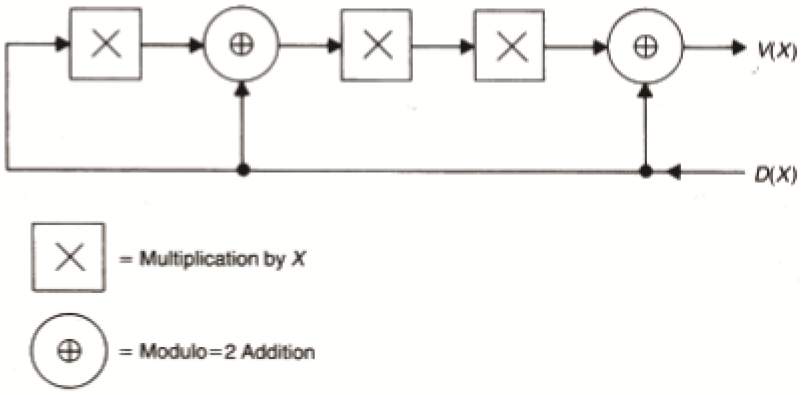
Table 8: Detecting Burst Errors of 3 in Residue Check System

Bits in Error:	**r_c value**	**r_s value**	**Detected?**
b0,b1,b2	1010	1001	Yes
b1,b2,b3	0011	1001	Yes
b2,b3,b4	0100	1001	Yes
b3,b4,b5	0010	1001	Yes
b4,b5,b6	0110	1001	Yes
b5,b6,b7	0001	1001	Yes

From the gathered data shown in Table 8, we can now conclude that the residue code algorithm can detect all bursts errors of $a - 1$.

4. **(10 points)** Consider an n-bit code word and show that if the generating polynomial $G(X)$ of a cyclic code has more than one term, all single-bit errors will be detected.

Let's assume we are using a generator polynomial $G(X) = 1 + X^2 + X^3$, where the circuit implementation looks something like this:



For testing purposes, we will assume a 4-bit data word $D(X)$ for which we will check the single-bit error capabilities of the code word $V(X)$. For simplicity, let's consider a data word $D(X) = 1_{dec} = 1000_{bin}$, then the output code word $V(X) = G(X) * D(X) = 1 + X^2 + X^3$. To tell if this is a valid code word, we must represent the code word in terms of $R(X)$ where $R(X) = D(X) * G(X) + S(X)$. So long as $S(X)$ is 0, then the code word is assumed to be correct. In this example, the retrieved code word $R(X)$ will be evenly divisible by $G(X)$, and therefore $S(X)$ is 0. Now let us assume that the LSB of $D(X)$ has been flipped, whereby now $D(X) = 1 + X^3$. Then,

$$R(X) = (1 + X^3) * (1 + X^2 + X^3) = (1 + X^2 + 2X^3 + X^5 + X^6) = (1 + X^2 + X^5 + X^6)$$

$$R(X) = (1 + X^2 + X^5 + X^6) = (1 + X^2 + X^3) + S(X)$$

$$S(X) = X^3 + X^5 + X^6 \neq 0$$

Because $S(X)$ has equated to a non-zero value, a syndrome has been detected, and therefore the system acknowledges an error has occurred in the code word. Let's now check all single bit errors in the system:

Table 8: Detecting Syndromes from single bit errors of Cyclic Codes

D(x) errors	**D(X) value**	**S(X) value**
no error	1	0
b0	$1 + X^3$	$X^2 + X^5 + X^6$
b1	$1 + X^2$	$X^2 + X^4 + X^5$
b2	$1 + X$	$X + X^3 + X^4$
b3	0	$1 + X^2 + X^3$

From Table 8, it becomes apparent that the cyclic code system will detect all single bit errors so long as the number of terms in the generator polynomial $G(X)$ is more than one. This rationally makes sense because if there were to be a bit flip in $D(X)$ then the code word would always be uniquely different from a single error bit flip because this would either remove terms, or provide more terms whereby $R(X)$ will not be evenly divisible by $G(X)$, therefore any single bit error will not successfully generate an erroneous valid code word.