Chapter 2 – Coding Techniques

The key to detecting and correcting errors in codes is the amount of **information redundancy** in the code.

Coding theory has become a mainstay as digital devices came into existence (rather recent in terms of history ~ 1960) and has made significant gains as processing power has increased in recent years. An overall understanding of codes is desirable in the study of reliability since the techniques depend on redundancy and are a critical component in FT computing systems.

The type of codes used depends on the types of errors that occur coupled with a sensible amount of redundancy and the overall reliability requirement (including speed and $).

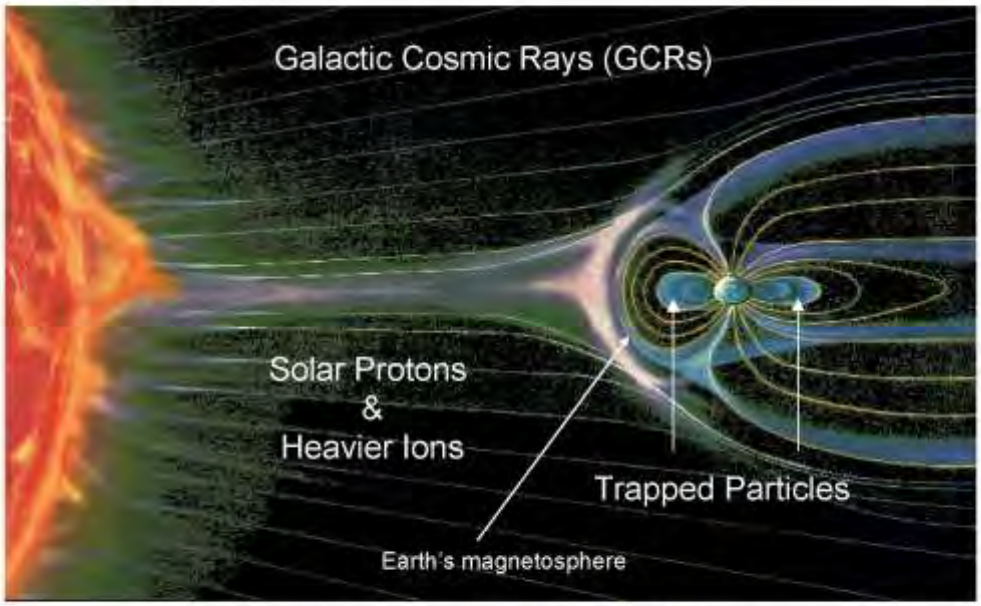
Information transmission errors have become 'invisible' today because of the underlying error detection/correction schemes built into almost all digital devices. One can not tolerate bit errors especially if one is obtaining his bank balance electronically.

Error Sources

1. Component Failure: stuck high/stuck low, chip-wide failure, etc.
2. Cross-talk: LC coupling in wires; first order dependency on the frequency and amplitude of the signal
3. Atmospheric (lightning, fading) disturbances: normally results in burst errors
4. Power Disturbances (both internal and external): noisy power sources, power supply decoupling, grounding
5. Radiation Effects: Single Event Upsets (SEUs), soft vs. hard failures, latch-up → destructive/nondestructive
6. Electromagnetic (EM) Fields: Other radiation sources, EM Interference (EMI), Electromagnetic Pulse (EMP) from nuclear devices
7. Channel Noise: SNR (dB) of transmission path, signal detection (is it a 1 or a 0?)
8. Phenomenon associated with the basic physics of devices and information theory: Brownian motion, quantization, Shannon Channel Capacity, $1/r^2$ signal losses

Radiation effects are a big driver in fault tolerant computing. Textbook Table 2.1 lists soft faults associated with airplane types. This is dependent on the computational equipment (speed, IC fabrication), altitude, IC fabrication techniques (registration resolution, line width, semiconductor type/material).

The Space Radiation Environment



Nikkei Science, Inc. of Japan, by K. Endo

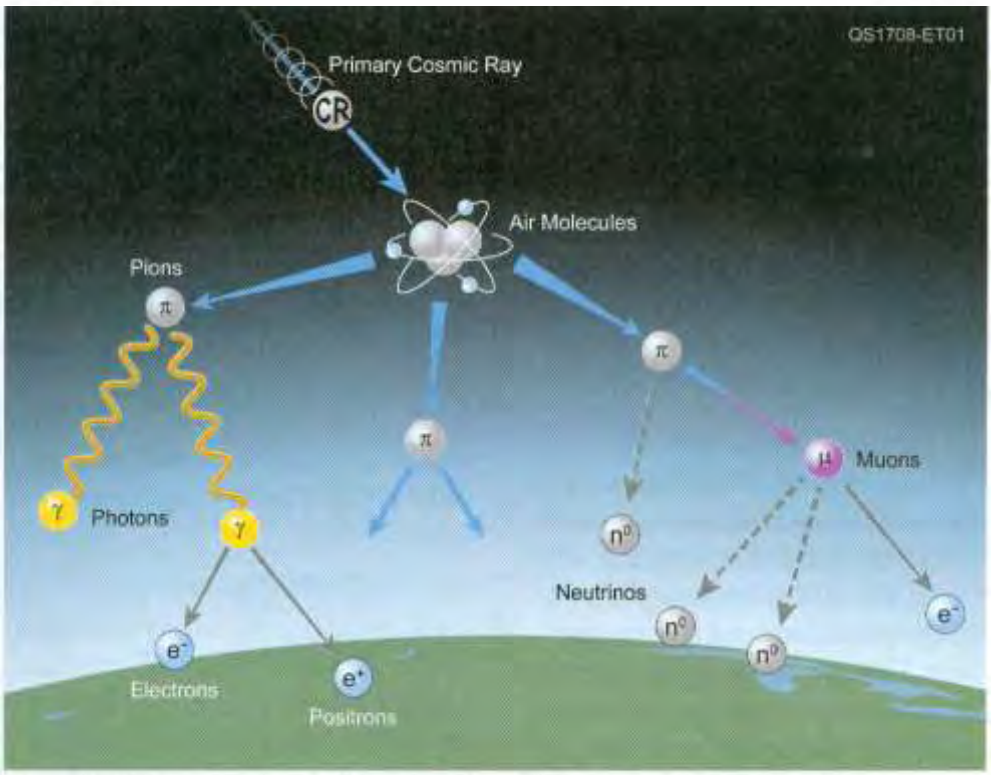


Figure 1 — When a cosmic ray slams into a molecule of air in our upper atmosphere, it produces a shower of secondary particles such as pions (π), photons (γ), electrons (e), and many others.

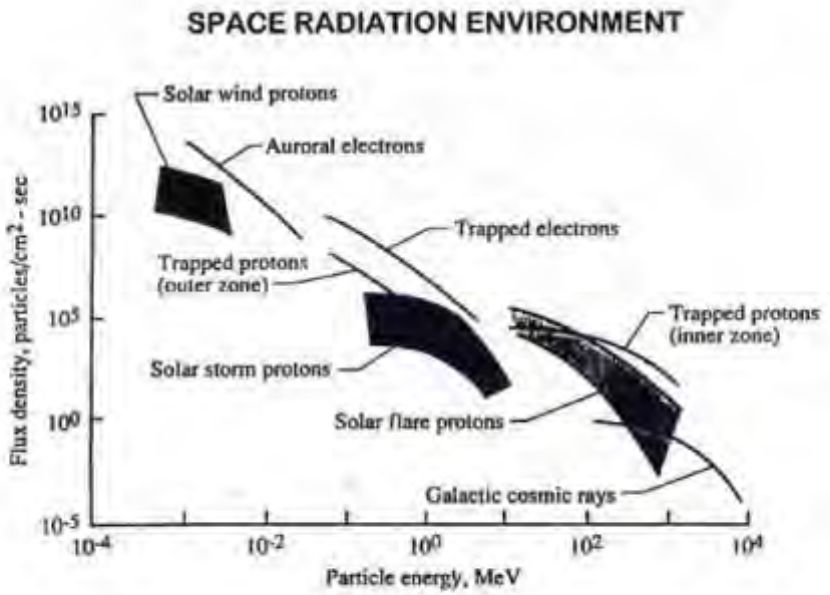
The Space Radiation Environment

Important Notes:

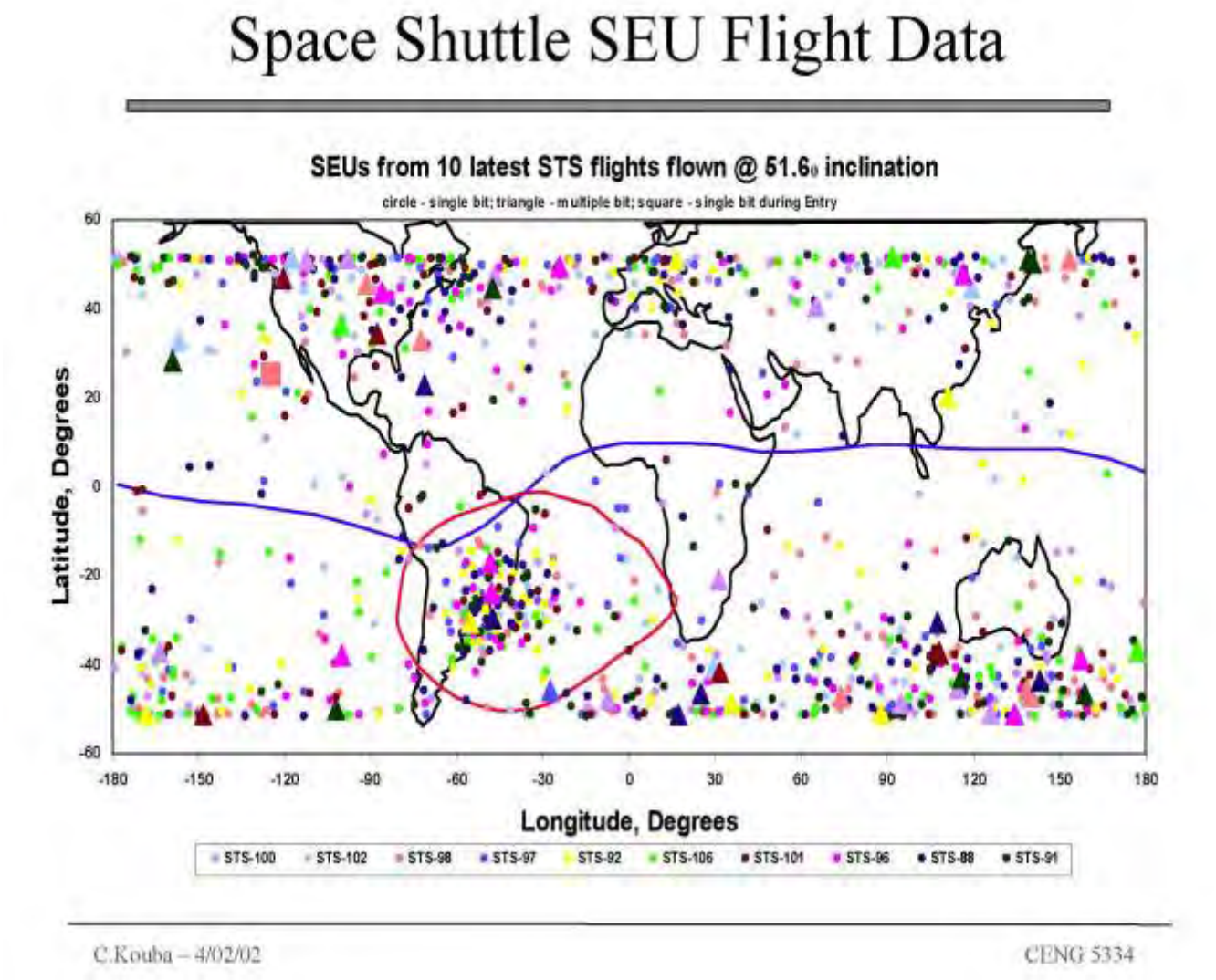
- Particles are omnidirectional and have energies ranging from keV to TeV [1]
- Altitude – generally, the higher the altitude, the greater the radiation threat
- Inclination – the magnetosphere provides some natural protection; more near the equator, less protection towards the polar regions
- South Atlantic Anomaly – an area where the magnetosphere dips closer to the Earth's surface (due to difference between true north and magnetic north)

(1) one eV is the energy gained by one electron in accelerating through a potential difference of one volt
1eV = 1.6×10^{-19} joules

The Space Radiation Environment



C.Rouiso – 4/02/02 CENG 5334



Note the South American Anomaly where solar radiation is higher than normal.

Critical actions in LEO (low earth orbit) are not executed in this geographical region.

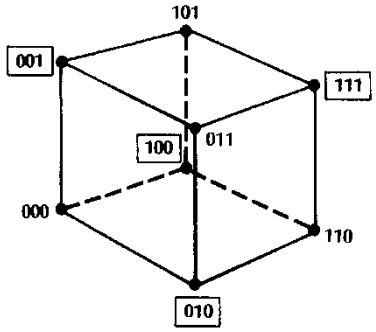
Also note that single event upsets (SEU) also occur when the Shuttle is on the launch pad at Cape Kennedy, Florida

Basic Principles

Coding is all about legal versus illegal code words (bits, nibbles, bytes, blocks).

Differentiation based on legal/illegal code distance, the *minimum* number of bits by which any one **valid** code word differs from another – the ***Hamming Distance***

The greater the error detection/error correction capability of a code → the greater the Hamming distance, the greater the redundancy (and vice versa).



Valid Code Words: 001, 010, 100, 111 Errors: 000, 011, 101, 110
Hamming Distance: 2

Simplest scheme: even/odd parity adds one redundant bit to differentiate between a valid and invalid code word.

Odd-Parity: total # of 1's in the code word (incld the parity bit) adds to an odd number
Even-Parity: total number of 1's in the code word is an even number

(Not necessarily the mirror images of each other, odd parity can detect stuck-low/catastrophic type hardware errors whereas all 0's will pass with even parity.)

Even/Odd Parity detects an odd number of errors (1, 3, 5, etc); can't detect an even number of errors/failures (double errors: 2, 4, 6, etc.).

Probability of undetected parity errors P'_{ue} with ***r*** failures in ***n*** bits
(r-out-of-n) with a bit failure probability of ***q*** is given by the binomial distribution:

$$B(r: n, q) = \begin{pmatrix} n \\ r \end{pmatrix} q^r (1 - q)^{n-r}$$

P_{ue} = undetected error without parity
P'_{ue} = undetected error with parity

which for single bit parity would be failures of r = 2, 4, 6, …. etc. For high reliability/ small q ($q < 10^{-4}$), P'_{ue} for $r \geq 2$ is negligible as compared to r = 2 so drop higher order failures of P'_{ue} (2-out-of-n bits)

For an 8-bit code word (a byte) with one bit parity (n = 9 bits) the most likely case of an undetected error would be the probability of occurrence for two errors (false positive)

$$P'_{ue} = B(2: 9, q) = \binom{9}{2} q^2 (1 - q)^7 = 36\, q^2 (1 - q)^7 \quad n = 9,\ r = 2,\ q = \text{BER (failure)}$$

To compute the improvement of a 1-bit parity scheme P'_{ue} compared to an undetected transmission error for 8-bits without parity P_{ue} - it is easiest to compute:

$$P_{ue} = 1 - P(\text{no errors}) = 1 - B(0: 8, q) = 1 - \binom{8}{0} q^0 (1 - q)^{8-0} = 1 - (1 - q)^8$$

as compared to calculating 1 to 7 bit failure possibilities

then comparing the ratio between P_{ue} / P'_{ue} Replacing $(1 \pm q)^n$ with $1 \pm nq$ and replacing $[1/(1 - q)]$ with $1 + q$ results in the formula (2.7)

$$P_{ue} / P'_{ue} = [\, 2(1 + 7q) / 9q \,] = \text{Improvement Ratio of Undetected error no Parity / Undetected error w/Parity}$$

For normal q (10^{-4} or less), the improvement ratio is significant (Table 2.3) with little overhead for the one extra parity bit in the 8-bit word which is (9 - 8) / 8 = 12.5%

Taking into account the coder/decoder reliabilities which have been neglected so far:

$P'_{ue} = P(A + B) = P(A) + P(B)$ mutually exclusive events associated with not detecting an error with parity where A is the event where the hardware doesn't fail but the parity error is undetected (h/w works but the error gets by with an even number of bit errors) and B is a hardware failure (error not detected because of a hardware failure)

P'_{ue} = P(no coder or decoder failure during transmission) x P(two bit errors undetected)
+ P(coder or decoder failure during transmission)

Based on the MIL-HDBK-217 reliability models, the hardware failure rate of ICs is proportional to the square root of the number of gates (g) in an *equivalent* logic model

$\lambda_b = c(g)^{½} = 1.67 \times 10^{-8}$ failure/hr for an equivalent logic model of the SN74180

Assuming a constant failure rate (Poisson) for λ_b then the coder + decoder reliability

$R(t) = e^{-2 \lambda_b t}$ and the probability of coder/decoder failure is $(1 - e^{-2 \lambda_b t}) = q(t)$

P(no coder or decoder failure during transmission) = $R(t) = e^{-2 \lambda_b t}$

P(hardware failure during transmission) = $1 - e^{-2 \lambda_b t}$

P(two errors in the even/odd parity) = P(undetected error) = $P'_{ue} = 36\, q^2 (1 - q)^7$

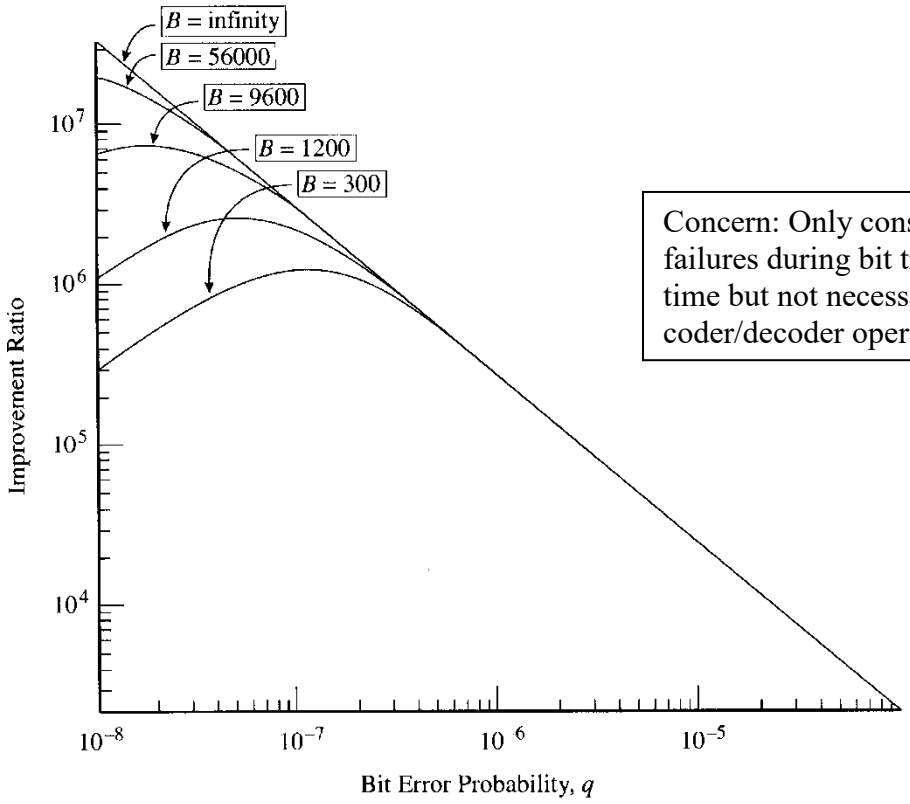
The time t associated with a hardware failure during the 9-bit transmission (8 bits message + 1 bit parity) where B = transmission rate in bps

$t = 9 \text{ bits} \times (1 / B \text{ bps}) = 9 \times 1/B \times 3600 \text{ sec/hour} = 9/3600B$

$$P'_{ue} = \underbrace{[\, e^{-2\lambda_b t} \,]}_{\text{No hardware failure}} [36\, q^2 (1-q)^7] + \underbrace{[1 - e^{-2\lambda_b t}]}_{\text{h/w failure}} \quad (2.10)$$

Probability of an undetected error for 1-bit parity including hardware failures

The probability of an undetected error without parity for 8-bits is still $P_{ue} = 1 - (1 - q)^8$
Thus the improvement ratio P_{ue} / P'_{ue} is graphed as follows for various data rates which includes the possibility of coder-decoder (hardware) failures during the transmission in B bps



Concern: Only consider hardware failures during bit transmission time but not necessarily during the coder/decoder operational times ??

If the hardware failure rate is small or the data rate B is large (small t), then the hardware failures have little effect on the improvement ratio. A good example would be memory read/write operations (large B or small t which minimizes the h/w failure rate)

The hardware failure rate has little impact if the bit error probability (BER) is $10^{-4} \rightarrow 10^{-6}$ (the large BER swamps out the impact of the hardware failures) but it does have an impact if the bit error rate is very small.

When t becomes larger (B = 300 bps for example), then the chip failures become significant as shown above Figure 2.5 in the textbook). Compare to Figure 2.3 which is the improvement ratio where hardware failures are not considered → straight line.

MIL - HDBK - 217B RELIABILITY MODEL

ASSUME EXPONENTIAL DISTRIBUTION WITH A FAILURE RATE λ FOR A SINGLE UNIT

$$\lambda = \pi_L \pi_Q (C_1 \pi_T + C_2 \pi_E) \pi_P$$

WHERE

π_L - LEARNING FACTOR (10 FOR NEW DEVICE, 1 FOR MATURE DEVICE)

π_Q - QUALITY FACTOR (16 FOR CLASS C SCREENING)

π_T = TEMPERATURE ACCELERATION FACTOR

π_E - ENVIRONMENTAL FACTOR (1 FOR GROUND, FIXED)

π_P - PIN MULTIPLIER FACTOR

C_1, C_2 - COMPLEXITY FACTORS

MIL - HDBK - 217E

$$\lambda = \pi_L \pi_Q (C_1 \pi_T \pi_V + C_2 \pi_E)$$

π_V - VOLTAGE STRESS FACTOR

The MIL-HDBK-217*X* (*A,B,C,D,E,F* → *changes due to technology*) is a linear formulation for a reliability model based on known empirical factors that impact component reliability that was validated with actual failure data gathered by DoD. It is no longer supported by the Gov't and has been replaced with more complex models such as Prism. MIL-HDBK-217*X* still provides a foundation for the basic principles of R(t) in electronic systems, e.g., high temps are bad, new technology without a track record is bad, the learning factor.

Hamming Codes (error detection/error correction)

To make a decision on detecting an error or correcting an error, there must be a clear choice between the code word in error and nearest correct code word. For error correction codes, the separation distance (the Hamming distance) must be large enough to allow valid decisions between legal and illegal code words.

For 1-bit parity, a single error produces an incorrect code word that is a Hamming distance of 1 from the correct word (d = 1). For a single error to be detected, the Hamming distance d between correct code words must be at least 2 so that the resultant error with its d = 1 will map into the illegal code word set and thus be detected. The even/odd parity bit scheme has a Hamming distance d = 2, which is the Hamming distance between correct code words. Note that for a detected parity bit error, there are a number of equally possible correct code words that are a d =1 distance away from the detected error, so correcting the error is not possible with a parity bit coding scheme.

TABLE 2.2 Examples of 3- and 4-Bit Code Words

(a) 3-Bit Code Words			(b) 4-Bit Code Words: 3 Original Bits plus Added Even-Parity (Legal Code Words)				(c) Illegal Code Words for the Even-Parity Code of (b)			
x_1 b_1	x_2 b_2	x_3 b_3	x_1 p_1	x_2 b_1	x_3 b_2	x_4 b_3	x_1 p_1	x_2 b_1	x_3 b_2	x_4 b_3
0	0	0	0	0	0	0	1	0	0	0
0	0	1	1	0	0	1	0	0	0	1
0	1	0	1	0	1	0	0	0	1	0
0	1	1	0	0	1	1	1	0	1	1
1	0	0	1	1	0	0	0	1	0	0
1	0	1	0	1	0	1	1	1	0	1
1	1	0	0	1	1	0	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1

If you examine the legal code words in Table 2.2(b) for the even parity scheme, note that the Hamming distance d between all of the legal code words is 2 or greater.

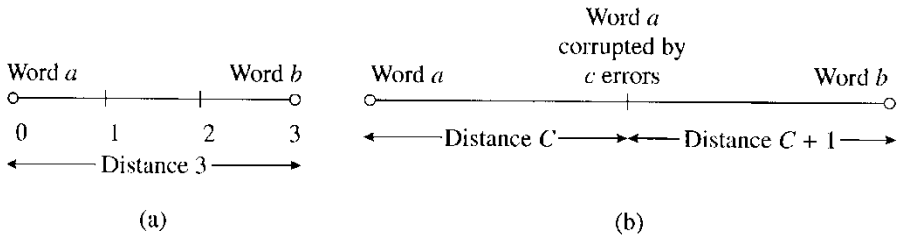
Thus to detect D errors, the Hamming distance $d \geq D + 1$

For single bit parity code where d =2 gives you a code able to detect one error D = 1

Using the following nomenclature:

d = the Hamming distance of a code (the least number of bits that are different in the legal code words)
D = the number of errors that a code can **D**etect
C = the number of errors that a code can **C**orrect
n = the total number of bits in a code word
m = the number of message bits (the information bits which are the no overhead bits)
c = the number of check bits thus c = n – m (the check bits are the overhead or redundant bits)

For CORRECTING bit errors, consider a code scheme with d = 3 shown below



An error in the code will have a Hamming distance d = 1 at a minimum. To correct the error by deciding if the corrupted word was originally Word ***a*** or Word ***b*** (the two nearest neighbors to the corrupted word), the decision must be clear and thus the Hamming distance of a correctable code d > 2 (the Hamming distance between valid code members)

$d \geq 2C + 1$ To correct C errors, the distance of the code must be 2C + 1

Combining our Hamming distances detection and correction formulas can be done by seeing that for the same Hamming distance (for example d = 4) the two formulas show

$d \geq D + 1$ $4 \geq 3 + 1$ thus D = 3 (three detectable errors)

$d \geq 2C + 1$ $4 \geq 2(1) + 1$ thus C = 1 thus

$D \geq C$ the # of detectable errors must be ≥ the # of correctable errors

Rewriting $d \geq 2C + 1 = C + C + 1$ and using the smallest value of D or D = C we see

$d \geq D + C + 1$ the Hamming distance d must be ≥ the # or detectable errors + the # of correctable errors + 1. We can trade our code Hamming distance d between the # of detections and the # of corrections

TABLE 2.5 Relationships Among *d*, *D*, and *C*

d	*D*	*C*	Type of Code
1	0	0	No code possible
2	1	0	Parity bit
3	1	1	Single error detecting; single error correcting
3	2	0	Double error detecting; zero error correcting
4	3	0	Triple error detecting; zero error correcting
4	2	1	Double error detectiug; single error correcting
5	4	0	Quadruple error detecting; zero error correcting
5	3	1	Triple error detecting; single error correcting
5	2	2	Double error detecting; double error correcting
6	5	0	Quintuple error detecting; zero error correcting
6	4	1	Quadruple error detecting; single error correcting
6	3	2	Triple error detecting; double error correcting
etc.			

For those examples with values for both D and C, note that the error corrected is also an error detected since $d \geq 2C + 1$ and $d \geq D + C + 1$ (for d = 3: C = 1 D = 1 or D = 2)

solely for correcting for correcting + detecting SECSED DED

Hamming SECSED Code

Codes with the ability to correct errors have more 'redundancy' or as we've seen and increased Hamming distance d: $d \geq 2C + 1$ or $d \geq D+C+1$

How the Hamming codes 'correct' is described in the textbook from a hardware implementation viewpoint and not from a basic mathematical view. Below, it will be shown that the described (7,4) SECSED code is a very unique code in that the binary representation of the detected error specifically identifies the numerical position of the bit in error within the code word.

From a matrix viewpoint (more understandable, see Siewiorek text for example):

$\mathbf{H} \bullet \mathbf{R}^T = \mathbf{S}$ where **H** is a Parity Check Matrix, **R** is the received code word and **S** is resultant Syndrome which points to the error that can be corrected.

The parity check matrix relates the m and c bits through the parity equations such as

$c_1 = m_1 \oplus m_2 \oplus m_3$

If the resultant column vector Syndrome = 0 then no errors are detected.

If $\mathbf{S} \neq 0$ then an error is detected. For the assumed single error, the resultant S points to the column in the parity check matrix which identifies the specific bit in error (either m or c).

For SECSED, double bit errors result in a invalid non-zero S since the bit identified as in error will be incorrect, e.g., SECSED fails for double bit errors just like even/odd parity. For triple bit errors, $\mathbf{S} = 0$ and the errors are not detected and therefore can't be corrected.

The following **H** is a parity-check matrix (PCM) for a single error correcting (7,4) Hamming code (SECSED)

	c_1	c_2	d_1	c_3	d_2	d_3	d_4
c_1	1	0	1	0	1	0	1
c_2	0	1	1	0	0	1	1
c_3	0	0	0	1	1	1	1

$\mathbf{H} \bullet \mathbf{R}^T = \mathbf{S}$

3 x 7 7 x 1 3 x 1

Writing the equations for calculating the values of the check bits c constrained so that S = 0 = no errors.

$$\mathbf{H} \bullet \mathbf{R}^T = \mathbf{H} \bullet [\, c_1\, c_2\, d_1\, c_3\, d_2\, d_3\, d_4 \,]^T = \mathbf{S} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix}$$

$$S_1 = c_1 \oplus d_1 \oplus d_2 \oplus d_4 = 0$$
$$S_2 = c_2 \oplus d_1 \oplus d_3 \oplus d_4 = 0$$
$$S_3 = c_3 \oplus d_2 \oplus d_3 \oplus d_4 = 0$$

Note that the **H** matrix column associated with the check bits consists of a single 1 which generates these equations which produce a unique binary value for the check bits c.

If $d_1\ d_2\ d_3\ d_4 = [\ 1\ 1\ 0\ 1\ ]$, calculate the corresponding code word.

The correct code word must satisfy the above equations with a zero syndrome [**S**] = 0. Substituting data values into $\mathbf{R}^T$ such that $\mathbf{H} \bullet \mathbf{R}^T = [\ \mathbf{S}\ ] = 0$ results in the following equations:

$S_1 = c_1 \oplus 1 \oplus 1 \oplus 1 = 0$
$S_2 = c_2 \oplus 1 \oplus 0 \oplus 1 = 0$
$S_3 = c_3 \oplus 1 \oplus 0 \oplus 1 = 0$

by ex-or operations/inspection $c_1 = 1$ $c_2 = 0$ and $c_3 = 0$ thus the corresponding code word is given by **R = c_1 c_2 d_1 c_3 d_2 d_3 d_4 = [1 0 1 0 1 0 1]** (from the format of the code word for **H** shown above)

Example: The following code word was received: 0 0 0 0 1 0 1 . Assuming bit-wide symbols, are there any errors? If so, which bit is in error and what is its correct value? If not, why not?

For the (7,4) SECSED code $[\ \mathbf{H}\ ] \bullet [\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ ]^T = [\ \mathbf{S}\ ] = [\ 1+1,\ 1,\ 1+1\ ] = [2\ \ 1\ \ 2\ ]_{10} = [0\ 1\ 0]_2$
Modulo 2

$$[\ \mathbf{S}\ ] = \begin{matrix} 0 = S_1 \\ 1 = S_2 \\ 0 = S_3 \end{matrix}$$

[**S** *should be shown as a 3 X 1 column vector above*]

The syndrome points to (matches with) the 2nd column or c_2 being in error. With the second bit in error, the correct code word is: 0 **1** 0 0 1 0 1 This is uniqueness of the (7,4) SECSED Hamming code. The resultant **S** = $(010)_2$ is the same as the binary position of the error, the 2nd bit. If S = 0 → no error exists.

The arrangement of the m and c bits does not need to be in any specific order. Best to consider all bits the same as long as the check bits satisfy the ⊕ equations which produce a column with one 1.
For SECSED, if the resultant **S** ≠ 0 then S will match bit for bit a specific column in the parity check matrix **H** which will uniquely identify either the c or d bit that must be corrected (for **S** to equal 0).

The hardware is the key which is not implemented by using this matrix scheme which is the math behind the correction scheme. COTS (commercial off-the-shelf) hardware is used and is done for very large word lengths (number of bits >> 8 bits).

Hamming (n, m) code where n = total # of bits in the word length and m = # of message bits thus the number of check bits c = n - m

For a (n, m) SECSED code, the # of check bits c that *cover* a maximum of word bits *n* is:

$$\mathbf{2^c \geq n + 1}$$ thus for example with n = 10 bits c must equal 4 ($2^4 \geq 10 + 1$)

Table 2.7 shows the SECSED Hamming code relationship for different code word lengths (different n's, e.g., for n =16 bit code word contains m = 11 msg bits and c = 5 check bits).

Reduction in Undetected Errors (for a Hamming SECSED Code)

One oddity of the SECSED is that the resultant non-zero Syndrome (matrix **S**) can be viewed as the correction for a single error OR the detection of a double error. Remember that for detection $d \geq D + 1$ Thus a Hamming distance of 3 required for a SECSED results in the ability to detect two errors: If Hamming Distance d = 3 then

$d \geq 3 \geq D + 1 \geq 2 + 1$ for detected errors D = 2 (but also $d \geq 3 \geq 2C + 1$ for C = 1)

One could call this a dual-error detecting code (DED). The scheme to do this is TBD.

To evaluate the resultant reduction in undetected errors for the Hamming SECSED code, the dominant term in the computation of the probability of an undetected error P'_{ue} is the probability of 3 errors for SECSED (an undetected error occurs when the SECSED fails for 3 bits in error because it can only detect 2 bits in error or correct 1bit in error).

The improvement in the undetected error ratio for a (12, 8) SECSED where m = 8 message bits (1 byte) and c = 4 check bits for n = 12 message bits is:

$P_{ue} / P'_{ue} = [\ 2(1 + 9q) / 55q^2\ ]$ where P_{ue} = undetected error in 8 message bits w/o any checking

Comparing this to the single bit parity code undetected error ratio (Eq 2.7)

$P_{ue} / P'_{ue} = [\ 2(1 + 7q) / 9q\ ]$

shows a significant improvement (a 10^3 to 10^7 improvement for the same BER) with the same m = 8 message bits but with n = 9 bits for the parity case and n = 12 bits for the SECSED case → **more redundancy** (more check bits) (Table 2.11 versus Table 2.3)

Effect of Coder-Decoder Failures (on SECSED)

As was done with parity codes, the effect of the hardware failing in the SECSED is evaluated in Section 2.4.6 The SECSED evaluation has much smaller undetected error probability (P'_{ue}) since

$$P'_{ue} = [\ e^{-\lambda_b t}\ ]\ [220\ q^3\ (1 - q)^9] + [1 - e^{-\lambda_b t}\ ]$$

From Figure 2.7 for a Hamming SECSED implementation with its more complex hardware, the hardware failure rate is four times that of the parity hardware implementation.

The improvement ratio P_{ue} / P'_{ue} is graphed in Figure 2.8 for the different data rates B which includes possible coder-decoder failures during the SECSED transmission.

Figure 2.8 has the same 'general' characteristics as the parity case Figure 2.5 but with lower values for the probability of an undetected error. The interesting result is that the impact of hardware failures is far more pronounced for SECSED. In fact for very small bit error probabilities (BER) coupled with slow transmission rates B = 300 bps or 1200 bps, the parity-bit scheme is SUPERIOR (bigger is not always better, remember the complexity term in the MIL-HDBK-217 reliability model).

Thus more is not necessarily better and for more complex error detection schemes, one should evaluate the impact of h/w failures since these may be considerable for small q.

1. Consider simpler coding schemes where h/w failures might severely impact the undetected error probabilities (P'_{ue}).
2. Use larger-scale integration which improves the resultant hardware reliability with less equivalent logic gates (g) as in the MIL-HDBK-217 reliability model.

Reliability of SECSED Code

Section 2.5.2 calculates the SECSED reliability by comparing no error correction for 1 byte (8 bits)

$$R = (1 - q)^8$$

versus the (12,8) SECSED where single errors are corrected.

$$R_{SECSED} = P(\text{no errors} + 1 \text{ error})$$

where the two events are mutually exclusive. With the binomial distribution

$$R_{SECSED} = (1 - q)^{11}(1+11q) \quad \text{where q = bit error probability} \quad \text{(Eq 2.33)}$$

Table 2.15 lists the gain in reliability for various values of q for the (12,8) SECSED code.

Reliability of a Retransmitted Code (for parity & SECSED used as a DED 2-error detection code)

Taking into consideration codes that both codes must retransmit after detecting an error

$$R = P(\text{no error}) + P(\text{detected error}) \times P(\text{no error on retransmission})$$

Comparing 9-bit parity and (12,8) DED, both of which require retransmission, Tables 2.15 and 2.16 show that both retransmit schemes are better than the (12,8) SECSED and that the parity code is best with a small margin over (12,8) SECSED. However both retransmit schemes have a 100% overhead as compared to typical SECSED whose overheads are 11% – 50% (decreases with longer code words m). The correcting scheme also takes less overall time if the error correction processing is quicker than retransmission (always a good assumption).

All of these comparisons neglect hardware failures (generator/checker and retransmission control logic) however the previous analysis results provide examples on how to incorporate hardware failures into the reliability equations.

Burst Error-Correction Codes

Parity and Hamming codes are intended for communication channels with good SNR (memory cycles, transmission schemes only subject to single bit errors, etc.) Many applications are subject to burst errors (CDs, DVDs, hard drives, magnetic media, RF communications, etc.)

Burst errors are errors in bit patters where adjacent bits are in error (error bits x) and may extend over a large field of bits m_1 = bbxxxbbxbbbb shows 4 errors over a 12 bit field.

The error detection scheme is to start with the requirement to detect bursts of length t where t in the above example could be t = 6, a burst length of 6 even though 2 bits of the burst length field were not in error. (Could design t = 3 then m_1 would have two bursts.) Check bit equations are developed by writing a set of equations for the message bits (m) in which the burst error bits are in a field of t bit positions.

If the resultant received syndrome is 0, then no errors are detected. The scheme begins to fail for burst lengths > t but a fair percentage of these conditions are still detected depending on the specific burst length pattern when the # of errors > t.

Some key properties of Burst Codes:

1. For a burst length of t, t check bits are required (overhead) to detect errors, which is independent of the message length m.
2. There are t check-bit equations where the t^{th} check-bit equation starts with bit t and contains all the bits that are 2t, 3t, ….. kt where $kt \leq n$
3. The resultant code word length need not be an integer multiple of t but it is normally padded to produce this (code words are normally multiple bytes long).
4. Hardware implementation schemes can be linear feedback shift registers (LFSR) or EXOR tree circuits where LFSR schemes have a greater 'processing' delay times (register shift times versus EXOR gate-switching times)

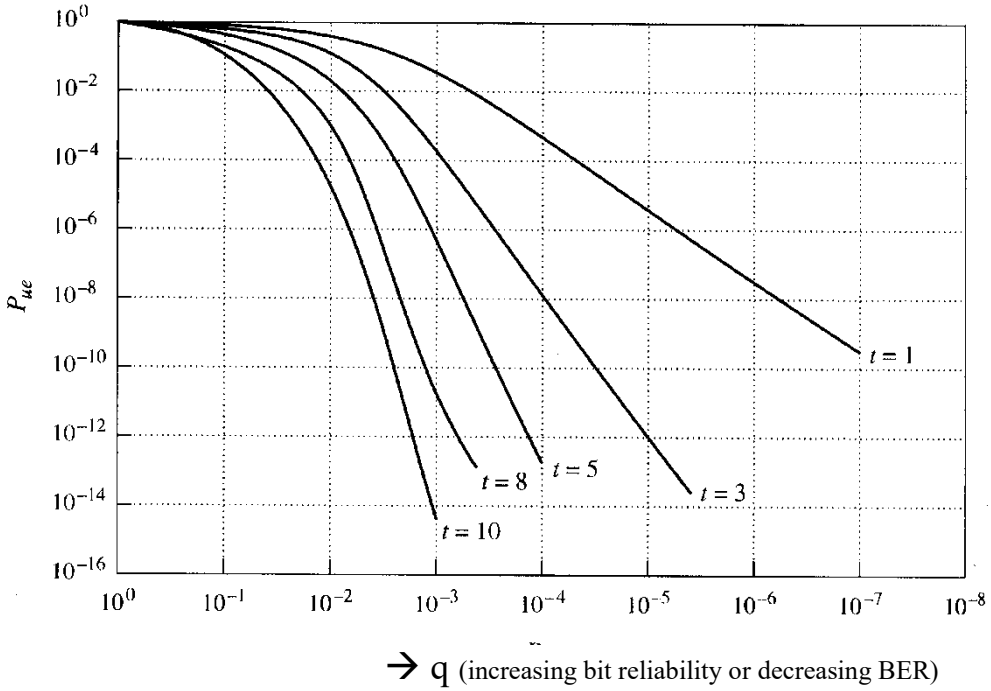
Error Correction

Burst error correction schemes rely on shifting of the error syndrome of a burst code and a number theory theorem called the Chinese Remainder Theorem. The schemes are complex and require numerous computational stages, all of which becomes more convoluted in a sequential logic implementation (textbook pages 66 – 72). However, burst codes are not the only way to detect and correct burst errors.

Reed-Solomon Codes – RS Codes

RS codes are block-type codes which operate on multiple rather than individual bits (the previously discussed bit-wide codes). Data is handled in blocks where each block is composed of n symbols where symbols are m bits long. For example, with a message length n = 255 bits where we want to correct up to 10 errors (t = 10) we'll have 235 message symbols and 20 check symbols with a code efficiency of 92.15%.

Probability of an uncorrected error in a RS code with the message parameters graphed for various t's where t = the number of errors to be corrected is:



Hardware failures haven't been considered but the analysis method is the same as before and again hardware failures can have an impact for some common ranges of the system parameters. The graph above shows that the probability of an undetected error is less for the more robust RS codes but the redundancy is also obviously higher in these codes (t = 10 versus t = 8).

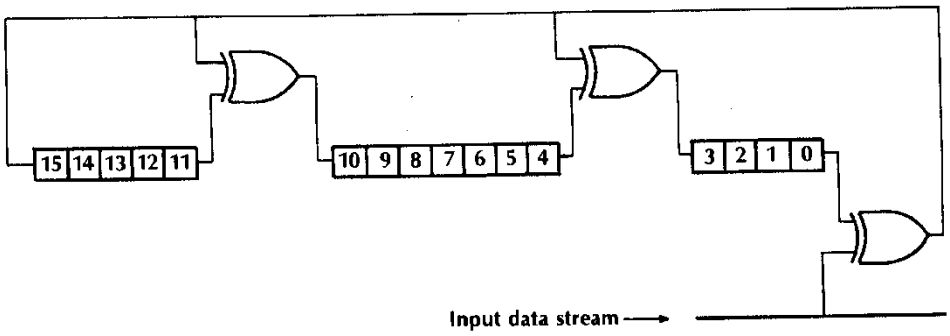
Interleaving is a technique of shifting individual bits by spreading them out over several code blocks and thus spreading the long burst errors so that error correction can occur even for code bursts > t. This is very common in today's error correcting codes such as Reed-Solomon and Turbo Codes.

Other Codes

CRC Codes (cyclic redundancy check) – a linear separable code which is where the check bits (c) can be separated from the message bits (m) and thus can be processed in parallel. CRC codes are a very common coding scheme that detects single bit and word-length burst errors with 100% reliability and has very good detection properties for even longer burst errors (~ 99.9969 %)

Even though the CRC check bits can be generated in software, a hardware implementation for example of the CRC-CCITT code with a generator polynomial of

$G(x) = x^{16} + x^{12} + x^5 + 1$ using linear feedback shift registers:



The generator and the checker are essentially the same. For the checker, when the message bits along with the CRC check are shifted thru the LFSR, if at the end of the block (data + CRC) the LFSR registers are all zero, then no error has been detected. If an error is detected, then retransmission is required. Very low overhead (small # of c bits)

The software process generates the CRC check bits by XOR'ing the message bits with a CRC constant (different for the various CRC codes like CRC16, CRC-CCITT, etc). For the receive end of the process, the received data is XOR'ed with the known CRC constant. At the end of this XOR process, if the data along with the received CRC result in the known CRC constant, then no errors have been detected. For example, the CRC constant for the CRC-CCITT is $(0F01)_{16}$

More codes:

M-of-N Codes
Duplication Codes
Parity Codes: bit-per-word parity, bit-per-byte parity, interlaced parity, chip-wide parity
Checksum Codes: single-precision, extended precision, residue code
Arithmetic Codes
BCH Codes
Concatenated Codes
Convolutional Codes – nested codes or the combination of different codes (inner & outer)
Residue Codes
Viterbi Decoding
Golay Codes
FEC (forward error correcting) Codes

See the error-correcting code (ECC) web page at http://www.eccpage.com/ for some programs that implement some of these various coding schemes.

Turbo Codes – the hot code today (G3/G4 cell phones).

A 'turbo' encoder is a combination of two simple encoders. The input is a block of K information bits. The two encoders generate parity symbols from two simple recursive convolution codes, each with a small number of states. The information bits are also sent uncoded. The key innovation of turbo codes is an interleaver, which permutes the original K information bits before input to the second encoder. The permutation allows that input sequences for which one encoder produces low-weight code words will usually cause the other encoder to produce high-weight code words. Thus, even though the constituent codes are individually weak, the combination is surprisingly powerful. The resulting code has features similar to a 'random' block code with K information bits.

Random block codes are known to achieve Shannon-limit performance as K gets large, but at the price of a prohibitively complex decoding algorithm. This is the holy grail of digital communications.

Shannon Capacity Formula: $\mathbf{C = B \log_2(1 + SNR)}$ where

C = channel capacity in bps, B = bandwidth in Hz, SNR = ratio of transmit power / noise power both in watts

Formula assumes white noise (thermal noise)
Factors not accounted for:

1. Impulse noise
2. Attenuation distortion or delay distortion – not constant over frequency range of signal

The 1948 creation of Information Theory and the concept of the bit as the fundamental unit of information, the Shannon formula represents the theoretical maximum for reliable data communications that can be achieved in a noisy channel. In practice, only much lower rates are achieved – until the advent of turbo codes.

Turbo codes are used in 3rd generation (3G) / 4th generation (4G) cell phones.

Background story to the development of Turbo Codes in 1993 from two French engineers (Berrou and Glavieus) versus a more typical mathematician's creation of a new code
(see February 2007 IEEE Spectrum article *Closing in on the Perfect Code* by Erico Guizzo)

See course web page for the URL (link) to a Turbo Code Graphical representation.