CHAPTER 10

PRACTICE SET

Questions

Q10-1. In this case, $k = 20$, $r = 5$, and $n = 20$. Five redundant bits are added to the dataword to create the corresponding codeword.

Q10-2. We have $k = 5$ and $n = 8$. The size of the dividend is the same as the size of the codeword (8 bits). We need to augment the dataword with three 0s. The size of the remainder is $r = n - k = 3$ bits. The divisor is $r + 1 = 4$ bits.

Q10-3. The value of a checksum can be all 0s (in binary). This happens when the value of the sum (after wrapping) becomes all 1s (in binary).

Q10-4. The Hamming distance $d_{min} = s + 1$. Since $s = 2$, we have $d_{min} = 3$.

Q10-5. We have $n = 2^r - 1 = 7$ and $k = n - 3 = 7 - 3 = 4$. A dataword has four bits and a codeword has seven bits. Although it is not asked in the question, we give the datawords and valid codewords below. Note that the minimum distance between the two valid codewords is 3.

Data	*Code*	*Data*	*Code*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Q10-6. We have $k = 8$, $r = 2$, and $n = 8 + 2 = 10$.

a. The number of valid codewords is $2^k = 2^8 = 256$.

b. The number of invalid codewords is $2^n - 2^k = 2^{10} - 2^8 = 768$.

Q10-7.

a. The generator has three bits (more than required). Both the rightmost bit and leftmost bits are 1s; it can detect all single-bit errors.

b. This cannot be used as a generator: the rightmost bit is 0.

c. This cannot be used as a generator; it has only one bit.

Q10-8. A *linear block* code is a block code in which the exclusive-OR of any two codewords results in another codeword.

Q10-9. In a *single-bit error* only one bit of a data unit is corrupted; in a *burst error* more than one bit is corrupted (not necessarily contiguous).

Q10-10. The following shows that L is the weighted sum of the data items.

Beginning	$R = 0$	$L = 0$
Iteration 1	$R = 0 + D_1$	$L = L + R = 0 + D1$
Iteration 2	$R = R + D_2 = D_1 + D_2$	$L = L + R = 2D_1 + D_2$
Iteration 3	$R = R + D_3 = D_1 + D_2 + D_3$	$L = L + R = 3D_1 + 2D_2 + D_3$
…	…	…
Iteration *n*	$R = D_1 + D_2 + \ldots + D_n$	$L = nD_1 + (n-1)D_2 + \ldots + D_n$

Q10-11. In this case $r = 7 - 1 = 6$.

a. The length of the error is $L = 5$, which means $L \leq r$. All burst errors of this size will be detected.

b. The length of the error is $L = 7$, which means $L = r + 1$. This CRC will detect all burst errors of this size with the probability $1 - (0.5)^5 \approx 0.9688$. Almost 312 out of 10,000 errors of this length may be passed undetected.

c. The length of the error is $L = 10$, which means $L > r$. This CRC will detect all burst errors of this size with the probability $1 - (0.5)^6 \approx 0.9844$. Almost 156 out of 10,000 errors of this length may be passed undetected. Although the length of the burst error is increased, the probability of errors being passed undetected is decreased.

Q10-12.

a. The generator 10111 is qualified and divisible by 11 (the quotient is 1101); it can always detect an odd number of errors.

b. The generator 101101 is qualified and divisible by 11 (the result is 11011); it can always detect an odd number of errors.

c. The generator 111 is qualified, but not divisible by 11; it can detect an odd number of errors sometimes, but not always.

Q10-13. The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Q10-14. The error cannot be detected because the sum of items is not affected in this swapping.

Q10-15. The following shows that L is the weighted sum of the data items.

Beginning	R = 1	L = 0
Iteration 1	R = 1 + D_1	L = L + R = 0 + 1 + D1
Iteration 2	R = R + D_2 = 1 + D_1 + D_2	L = L + R = 2 + 2D_1 + D_2
…	…	…
Iteration *n*	R = 1 + D_1 + D_2 + … + D_n	L = *n* + *n*D_1 + …+ D_n

Problems

P10-1. Answers are given below:

a. error **b.** error **c.** 0000 **d.** 1101

P10-2. According to the Hamming distance principle, the code can correct one single-bit error ($d = 2t + 1$). The distance between codewords is 3. Assume both the sender and the receiver have the list. Now the sender sends the codeword `01011`, but it is corrupted and received as `11011` (the leftmost bit is corrupted). The receiver compares the received codewords with all valid codewords to find only one bit difference. This can be done by XORing the received codeword with all valid codewords. The one which has only one 1 bit in the result is the correctly sent codeword.

a. `00000` ⊕ `11011` = `11011`

b. `01011` ⊕ `11011` = `10000` → The original code should be `01011`.

c. `10101` ⊕ `11011` = `01110`

d. `11110` ⊕ `11011` = `00101`

P10-3. If we need to correct *m* bits in an *n* bit codeword, we need to think about the combination of *n* objects taking no object at a time or *Com*(*n*, 0), which means the state of no error, the combination of *n* objects taking one object at a time or *Com*(*n*, 1), which means the state of one-bit error, the combination of *n* objects taking two objects at a time or *Com*(*n*, 2), which means the state of two-bit error, and so on. We can have the following relationship between the value of *r* (number of redundant bits) and the value of *m* (the number of errors) we need to correct.

$$2^r \geq Com(n, m) + Com(n, m-1) + \ldots + Com(n, 1) + Com(n, 0)$$

P10-4.

a. 101110 $\rightarrow$ $x^5 + x^3 + x^2 + x$

b. 101110 $\rightarrow$ 101110**000** (Three 0s are added to the right)

c. $x^3 \times (x^5 + x^3 + x^2 + x) = x^8 + x^6 + x^5 + x^4$

d. 101110 $\rightarrow$ 10 (The four rightmost bits are deleted)

e. $x^{-4} \times (x^5 + x^3 + x^2 + x) = x$ (Note that negative powers are deleted)

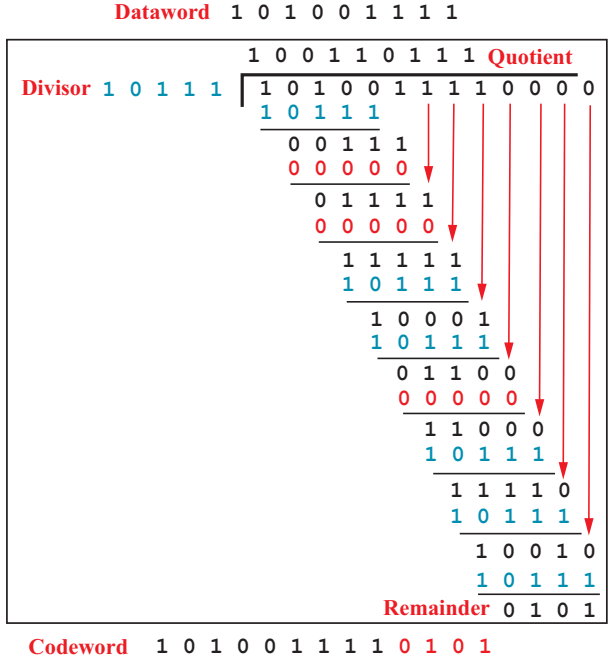
P10-5. The following shows the steps:

a. We first add the numbers in two's complement to get 212,947.

b. We divide the above result by 65,536 (or 2^{16}). The quotient is 3 and the remainder is 16,339. The sum of the quotient and the remainder is 16,342.

c. Finally, we subtract the sum from 65,535 (or $2^{16} - 1$), simulating the complement operation, to get 49,193 as the checksum.

P10-6. The following shows the calculation of the codeword.



P10-7.

a. We calculate R and L values in each iteration of the loop and then concatenate L and R to get the checksum. All calculations are in hexadecimal and modulo 256 or $(FF)_{16}$. Note that R needs to be calculated before L in each iteration ($L = L_{previous} + R$).

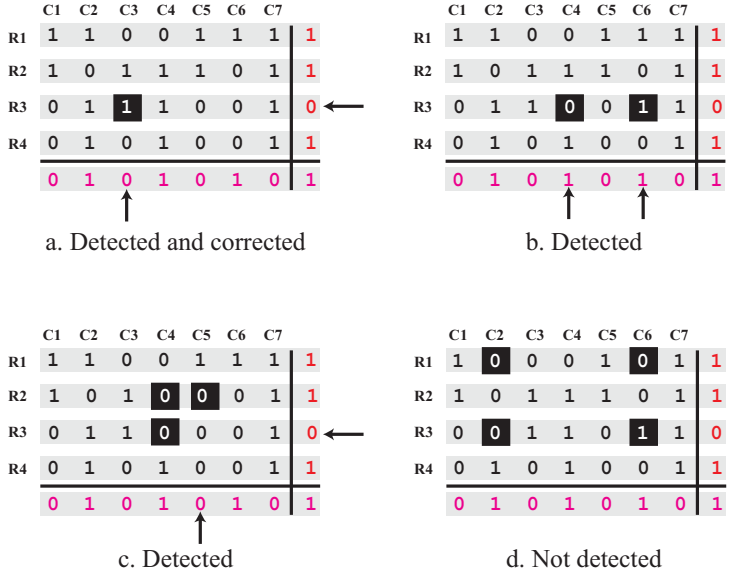
Initial values:	`R = 00`	`L = 00`
Iteration 1:	`R = 00 + 2B = 2B`	`L = 00 + 2B = 2B`
Iteration 2:	`R = 2B + 3F = 6A`	`L = 2B + 6A = 95`
Iteration 3:	`R = 6A + 6A = D4`	`L = 95 + D4 = 69`
Iteration 4:	`R = D4 + AF = 83`	`L = 69 + 83 = EC`
Checksum = `EC83`		

b. The L and R values can be calculated as shown below (D_i is the corresponding bytes), which shows that L is the weighted sum of bytes.

$$R = D_1 + D_2 + D_3 + D_4 = 2B + 3F + 6A + AF = 83$$
$$L = 4 \times D_1 + 3 \times D_2 + 2 \times D_3 + 1 \times D_4 = EC$$

P10-8. The codeword for dataword 10 is 101. If a 3-bit burst error occurs, the codeword will be changed to 010. This pattern is not one of the valid codewords, so the receiver detects the error and discards the received pattern.

P10-9. The following shows the errors and how they are detected.



a. In the case of one error, it can be detected and corrected because the two affected parity bits can define where the error is.

b. Two errors can definitely be detected because they affect two bits of the column parity.The receiver knows that the message is somewhat corrupted (although not where). It discards the whole message.

c. Three errors are detected because they affect two parity bits, one of the column parity and one of the row parity. The receiver knows that the message is somewhat corrupted (although not where). It discards the whole message.

d. The last case cannot be detected because none of the parity bits are affected.

P10-10.

a. If we rotate 0101100 one bit, the result is 0010110, which is in the code.

b. The XORing of the two codewords (0010110) $\oplus$ (1111111) = 1101001, which is in the code.

P10-11. The redundant bits in this case need to find $(n + 1)$ different states because the corruption can be in any of the n bits or in no bits (no corruption). A set of r bits can define 2^r states. This means that we need to have the following relationship: $2^r \geq n + 1$. We need to solve the equation for each value of k using trial and error to find the minimum value of r.

a. If $k = 1$, then $r = 2$ and $n = 3$ because $(2^2 \geq 3 + 1)$, which means C(3, 1).

b. If $k = 2$, then $r = 3$ and $n = 5$ because $(2^3 \geq 5 + 1)$, which means C(5, 1).

c. If $k = 5$, then $r = 4$ and $n = 9$ because $(2^4 \geq 9 + 1)$, which means C(9, 5).

d. If $k = 50$, then $r = 6$ and $n = 56$ because $(2^6 \geq 56 + 1)$, which means C(56, 50).

e. If $k = 1000$, then $r = 10$ and $n = 1010$ because $2^{10} \geq 1010 + 1$, which means C(1010, 1000).

P10-12. CRC-8 generator is $x^8 + x^2 + x + 1$.

a. It has more than one term and the coefficient of x^0 is 1. It can detect a single-bit error.

b. The polynomial is of degree 8, which means that the number of checkbits (remainder) r = 8. It will detect all burst errors of size 8 or less.

c. Burst errors of size 9 are detected most of the time, but they slip by with probability $(1/2)^{r-1}$ or $(1/2)^{8-1} \approx 0.008$. This means **8 out of 1000** burst errors of size 9 are left undetected.

d. Burst errors of size 15 are detected most of the time, but they slip by with probability $(1/2)^{r}$ or $(1/2)^{8} \approx 0.004$. This means **4 out of 1000** burst errors of size 15 are left undetected.

P10-13. We use modulo-11 calculation to find the check digit:

$$\mathbf{C} = (\mathbf{1} \times 0) + (\mathbf{2} \times 0) + (\mathbf{3} \times 7) + (\mathbf{4} \times 2) + (\mathbf{5} \times 9) + (\mathbf{6} \times 6) + (\mathbf{7} \times 7) + (\mathbf{8} \times 7) + (\mathbf{9} \times 5) \bmod 11 = 7$$

P10-14. The sum in this case is $(FFFF)_{16}$ and the checksum is $(0000)_{16}$. The problem shows that the checksum can be all 0s in hexadecimal. It can be all Fs in the hexadecimal only if all data items are all 0s, which makes no sense.

P10-15. The following shows the result. Part d shows that the Hamming distance between a word and itself is 0.

a. d (10000, 00000) = 1

b. d (10101, 10000) = 2

c. d (00000, 11111) = 5

d. d (00000, 00000) = 0

P10-16. The exclusive-OR of the second and the third codewords (01011) ⊕ (10111) is 11100, which is not in the code. The code is not linear.

P10-17. The receiver misses samples 21, 23, 25, 27, 29, 31, 33, 35, 37, and 39. However, the even-numbered samples are received and played. There may be some glitches in the audio, but that passes immediately.

P10-18. Adler is a byte-oriented algorithm; data needs to be divided into bytes. For this reason, we need to represent each 16-bit data words in the problem into two bytes. The result is $(FB)_{16}$, $(FF)_{16}$, $(EF)_{16}$, and $(AA)_{16}$.

a. We calculate R and L values in each iteration of the loop and then concatenate L and R to get the checksum. All calculations are in hexadecimal and modulo 65521 or $(FFF1)_{16}$. Note that R needs to be calculated before L in each iteration ($L = L_{previous} + R$). Since the result in each iteration is smaller than $(FFF1)_{16}$, modular calculation does not show here, but we need to remember that it needs to be applied continuously.

Initial:	`R = 0001`	`L = 0000`
Iteration 1:	`R = 0001 + FB = 00FC`	`L = 0000 + 00FC = 00FC`
Iteration 2:	`R = OOFC + FF = 01FB`	`L = 00FC + 01FB = 02F7`
Iteration 3:	`R = 01FB + EF = 02EA`	`L = 02F7 + 02EA = 05E1`

Iteration 4:	R = 02EA + AA = 0394	L = 05E1 + 0394 = 0975
Checksum = 09750394		

b. The L and R values can be calculated as shown below (D_i is the corresponding bytes), which shows that L is the weighted sum of bytes.

$$R = 1 + D_1 + D_2 + D_3 + D_4 = 1 + FB + FF + EF + AA = 0394$$

$$L = 4 + 4 \times D_1 + 3 \times D_2 + 3 \times D_2 + 3 \times D_2 = 0975$$

P10-19. This generator is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

a. It has more than one term and the coefficient of x^0 is 1. It detects all single-bit error.

b. The polynomial is of degree 32, which means that the number of checkbits (remainder) r = 32. It will detect all burst errors of size 32 or less.

c. Burst errors of size 33 are detected most of the time, but they are slip by with probability $(1/2)^{r-1}$ or $(1/2)^{32-1} \approx 465 \times 10^{-12}$. This means **465 out of 10^{12}** burst errors of size 33 are left undetected.

d. Burst errors of size 55 are detected most of the time, but they are slipped with probability $(1/2)^r$ or $(1/2)^{32} \approx 233 \times 10^{-12}$. This means **233 out of 10^{12}** burst errors of size 55 are left undetected.

P10-20. We first calculate the sum modulo 10 of all digits. We then let the check digit to be 10 − sum. In this way, when the check digit is added to the sum, the result is 0 modulo 10.

$$\textbf{sum} = [(1 \times 9) + (3 \times 7) + (1 \times 8) + (3 \times 0) + (1 \times 0) + (3 \times 7) + (1 \times 2) + (3 \times 9) + (1 \times 6) + (3 \times 7) + (1 \times 7) + (3 \times 5)] \bmod 10 = 137 \bmod 10 = 7 \rightarrow \textbf{C} = 10 - 7 = 3$$

P10-21.

a. $(x^3 + x^2 + x + 1) + (x^4 + x^2 + x + 1) = x^4 + x^3$

b. $(x^3 + x^2 + x + 1) - (x^4 + x^2 + x + 1) = x^4 + x^3$

c. $(x^3 + x^2) \times (x^4 + x^2 + x + 1) = x^7 + x^6 + x^5 + x^2$

d. $(x^3 + x^2 + x + 1) / (x^2 + 1) = x + 1$ (remainder is 0)

P10-22. The following shows the steps:

a. We first add the numbers to get $(0002A3BE)_{16}$. This corresponds to the first loop in Figure 10.17.

b. We extract the leftmost four digits, $(0002)_{16}$, and the rightmost four digits, (A3BE)16, and add them together to simulate the second loop in Figure

10.17. The result is $(A3C0)_{16}$. We stop here because the result does not create a carry.

c. Finally, we complement the result to get the checksum as $(5C3F)_{16}$.

P10-23. To detect single bit errors, a CRC generator must have at least two terms and the coefficient of x^0 must be nonzero.

a. $x^3 + x + 1 \rightarrow$ It meets both critic.

b. $x^4 + x^2 \rightarrow$ It meets the first criteria, but not the second.

c. $1 \rightarrow$ It meets the second criteria, but not the first.

d. $x^2 + 1 \rightarrow$ It meets both criteria.

P10-24. This is a binomial distribution if we think of each bit as the outcome of tossing a coin. A corrupted bit is the *head* outcome; an uncorrupted bit is the *tail* outcome. The probability of x errors in an n-bit data unit is the probability of tossing a non-fair coin n times and expecting x heads:

$$P[x \text{ bits in error}] = C(n, x)\, p^x\, (1-p)^{n-x}$$

$$P[\text{1-bit error in 8-bit unit}] = C(8, 1)\,(0.2)^1\,(0.8)^7 \approx 0.34$$

$$P[\text{3-bit error in 16-bit unit}] = C(16, 3)\,(0.3)^3\,(0.7)^{13} \approx 0.15$$

$$P[\text{10-bit error in 32-bit unit}] = C(32, 10)\,(0.4)^{10}\,(0.6)^{22} \approx 0.09$$

P10-25. The following shows the results. In the interpretation, 0 means a word of all 0 bits, 1 means a word of all 1 bits, and ~X means the complement of X.

a. (10001) ⊕ (10001) = (00000) **Interpretation: X ⊕ X → 0**

b. (11100) ⊕ (00000) = (11100) **Interpretation: X ⊕ 0 → X**

c. (10011) ⊕ (11111) = (01100) **Interpretation: X ⊕ 1 → ~X**

P10-26. Two bits can be corrupted anywhere in the codeword, one bit can be corrupted anywhere in the codeword, and the codeword can be uncorrupted. The sequence of r bits in this case needs to define $Com(n, 2) + n + 1$ states, in which $Com(n, 2)$ means combination of n objects taken two at a time. A set of r bits can define 2^r states. This means that we need to have the following relationship: $2^r \geq com(n, 2) + n + 1$. We need to solve the equation for each value of k using trial and error to find the minimum value of r.

a. If $k = 1$, then $r = 4$ and $n = 5$ because $(2^4 \geq Com(5, 2) + 5 + 1)$, which means C(5, 1).

b. If $k = 2$, then $r = 5$ and $n = 7$ because $(2^5 \geq Com(7, 2) + 7 + 1)$, which means C(7, 2).

c. If $k = 5$, then $r = 7$ and $n = 12$ because $(2^7 \geq Com(12, 2) + 12 + 1)$, which means C(12, 5).

d. If $k = 50$, then $r = 11$ and $n = 61$ because $(2^{11} \geq Com(61, 2) + 61 + 1)$, which means C(61, 50).

e. If $k = 1000$, then $r = 19$ and $n = 1019$ because $(2^{19} \geq Com(1019, 2) + 1019 + 1)$, which means C(1019, 1000).

P10-27. We have (vulnerable bits) = (data rate) × (burst duration). The last example shows how a noise of small duration can affect a large number of bits if the data rate is high.

a.	vulnerable bits	=	$(1500) \times (2 \times 10^{-3})$	=	3 bits
b.	vulnerable bits	=	$(12 \times 10^3) \times (2 \times 10^{-3})$	=	24 bits
c.	vulnerable bits	=	$(100 \times 10^3) \times (2 \times 10^{-3})$	=	200 bits
d.	vulnerable bits	=	$(100 \times 10^6) \times (2 \times 10^{-3})$	=	200,000 bits

P10-28. Sending only high-resolution packets means sending only 700 bits per packet, or 70,000 bits. Sending both high-resolution and low-resolution means sending 1100 bits per packet or 110,000 bits. We are sending 40,000 extra bits. The overhead is (110,000 – 70,000) / 70,000 ≈ 57 percent.

P10-29. The CRC-8 is 9 bits long, which means $r = 8$.

a. It has more than one bit and the rightmost and leftmost bits are 1s; it can detect a single-bit error.

b. Since $6 \leq 8$, a burst error of size 6 is detected.

c. Since $9 = 8 + 1$, a burst error of size 9 is detected most of the time; it may be left undetected with probability $(1/2)^{r-1}$ or $(1/2)^{8-1} \approx 0.008$.

d. Since $15 > 8 + 1$, a burst error of size 15 is detected most of the time; it may be left undetected with probability $(1/2)^r$ or $(1/2)^8 \approx 0.004$.

P10-30. We need to add all bits modulo-2 (XORing). However, it is simpler to count the number of 1s and make them even by adding a 0 or a 1. We have shown the parity bit in the codeword in color and separated for emphasis.

	Dataword		Number of 1s		Parity	Codeword
a.	1001011	→	4 (even)	→	0	10010110
b.	0001100	→	2 (even)	→	0	00011000
c.	1000000	→	1 (odd)	→	1	10000001
d.	1110111	→	6 (even)	→	0	11101110