ELEX 7860 : Wireless System Design
2022 Winter Term

Introduction to Coding

After this lecture you should be able to:

- *distinguish between source coding, encryption and channel coding*
- *compute the minimum distance of a code from its codewords*
- *find n, k and the code rate for systematic block codes*
- *convert a codeword to a polynomial*
- *multiply and divide polynomials with coefficients from GF(2)*
- *compute a simple CRC by dividing a message by a generator polynomial*
- *draw the schematic of circuit that does polynomial division by a specific generator polynomial using flip-flops and XOR gates*
- *compute the undetected error probability for an $(n-k)$-bit CRC for burst errors lengths $\leq (n-k)$ and for a BER of 0.5*
- *describe why a simple CRC does not detect extra leading or trailing zeros and how this can be fixed*

Revised Feb 2 - added learning objectives.

Coding

The term coding has three different meanings when used in communication systems:

Source Coding Often called "compression," source coding attempts to reduce the data rate to more closely match the information rate by removing redundancy. This reduces the complexity of the remainder of the communication system.

Security Techniques such as signatures and encryption can be used to ensure the integrity, authenticity and the privacy of the information being transmitted over the channel.

Channel Coding Used to detect and correct errors introduced by the channel.

The first two are typically covered in courses in signal processing and data communications respectively. In this course we will concentrate on the third meaning.

Parity

Another technique for detecting errors in received frames is for the transmitter to compute one or more bits called "parity bits" and append them to the frame. The receiver computes the parity bits itself from the received data and compares them to the received parity bits. If the computed and received parity bits match then either there were no errors or the received bits were corrupted in such a way the received parity bits are valid for the received data.

The probability of the latter event is called the *undetected error probability*. Good error detecting codes try to make this probability as low as possible.

Single Parity Bits

The simplest type of parity is a single parity bit. Typically the parity bit is computed as the modulo-2 sum of all of the bits in the message.

Exercise 1: What is a modulo-2 sum? What is the modulo-2 sum of 1, 0 and 1? What is the modulo-2 sum if the number of 1's is an even number?

The modulo-2 sum can be easily computed as the exclusive-or (XOR) of the bits.

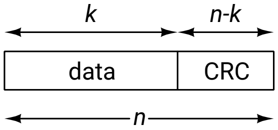
A common use of a single parity bit is a parity bit added to each ASCII character. Most serial interfaces can be configured to compute and append a parity bit to each ASCII character. This bit can be either the sum of all of the bits ("even parity") or it's complement ("odd parity").

The receiver computes the parity bit from the data bits and compares the computed parity bit to the transmitted parity bit. If the computed and received parity bits match then there was either no error or there were an even number of errors.

Block Codes

More complex channel codes use multiple parity bits. Each parity bit is computed from a different subset of data bits. This makes the code more "powerful" in the sense that it can detect (and potentially correct) more errors.

A block code where each block of n bits contains k data bits is called an (n, k) code:



Note that an (n, k) code contains $n - k$ parity bits.

A "code" is defined by the set of all possible n-bit codewords.

Exercise 2: A (5,3) code computes the two parity bits as: $p_0 = d_0 \oplus d_1$ and $p_1 = d_1 \oplus d_2$ where d_i is the i'th data bit. What codeword is transmitted when the data bits are $(d_0, d_1, d_2) = (0, 0, 1)$? How many different codewords are there in the code? What are the first four codewords? In general, how many codewords are there for an (n, k) code?

Hamming Distance

The *Hamming Distance*, D, is the number of bits that differ between two code words.

The performance of a particular code is mainly determined by the minimum (Hamming) distance (D_{min}) between any two code words in the code.

Exercise 3: What is the Hamming distance between the codewords 11100 and 11011? What is the minimum distance of a code with the four codewords 0111, 1011, 1101, 1110?

A code constructed as above (2^k codewords each with $n - k$ parity bits) is a linear code because each codeword can be created by (modulo-2) sum of the other codewords. The weight of a codeword is the number of 1's. For a linear code the minimum Hamming distance is equal to the minimum weight of the non-zero codewords.

Exercise 4: A (5,2) block code has codewords constructed as $d_0, d_1, d_0 \oplus d_1, d_0, d_1$. Is this a linear code? List the codewords. What is the minimum weight of this code? What is the minimum distance?

Code Rate

The *rate* of a code is the ratio of information bits to total bits, or k/n. This is a measure of the efficiency of the code. As we add parity bits the code rate decreases but, for a well-designed code, the minimum distance and thus the error-correcting ability increases.

Exercise 5: What is the code rate of a code with 4 codewords each of which is 4 bits long? *Hint: If a code has 2^k codewords, what is k?*.

Exercise 6: The data rate over the channel is 50 Mb/s; a rate 1/2 code is used. What is the throughput?

Codewords as Polynomials

Polynomials in GF(2)

A Galois field, denoted as $GF(q)$, is a set of integers and two operations that have certain properties. One of the properties is closure – the result of any operation on two elements of the field is also in the field.

For example, $GF(2)$ includes two integers (0 and 1) and the addition and multiplication operations are defined as addition and multiplication with the result taken modulo-2.

Exercise 7: Write the addition and multiplication tables for $GF(2)$. What logic function can be used to implement modulo-2 addition? Modulo-2 multiplication?

Instead of representing messages as a sequence of bits we can also represent codewords as polynomials with coefficients from $GF(2)$. For example, the polynomial:

$$1x^3 + 0x^2 + 1x^1 + 1x^0 = x^3 + x^1 + 1$$

can be used to represent the codeword 1011.

Exercise 8: What is the polynomial representation of the codeword 01101?

Polynomial arithmetic allows for a simple description of various operations on codewords.

Note that it is the coefficients of the polynomial that are important. The polynomial itself is never evaluated and the variable x is a dummy variable. These polynomials can thus also be viewed as binary numbers or bit strings where the order of each term indicates the bit position.

Polynomial Arithmetic

We can add, subtract, multiply and divide polynomials with coefficients in GF(2). These operations are the basis for some useful communication-related functions including convolutional codes for

FEC (Forward Error Correction), CRCs (Cyclic Redundancy Checks), and PRBS (Pseudo-Random Bit Sequence) generators.

Exercise 9: What is the result of multiplying $x^2 + 1$ by $x^3 + x$ if the coefficients are regular integers? If the coefficients are values in $GF(2)$? Which result can be represented as a bit sequence?

Cyclic Redundancy Checks

A Cyclic Redundancy Check (CRC) is a code used to detect errors in a sequence of k data bits. A "codeword" of n bits is transmitted for each k data bits. The length of the CRC is thus $n - k$.

The algorithm used to compute the CRC is as follows:

The data to be transmitted, treated as a polynomial, is multiplied by the polynomial x^{n-k}. This increases the order of each term by $n - k$ (or equivalently, appends $n-k$ zero bits). This new polynomial, $M(x)$, is divided by a *generator polynomial*, $G(x)$[1]. The result is a quotient and a remainder:

$$\frac{M(x)}{G(x)} = Q(x) \,\text{remainder}\, R(x)$$

We then replace the last $n - k$ bits of $M(x)$ (which were zero due to us having multiplied by x^{n-k}) with $R(x)$. This is equivalent to adding (or subtracting since polynomial addition and subtraction are the same for coefficients in GF(2)) $R(x)$ from $M(x)$. This ensures that the new polynomial will be divisible by $G(x)$.

Note that $n-k$ is one less than the number of terms in $G(x)$ since the remainder is always less than the divisor. If we number the terms by the order of x, then the highest order term will be x^{n-k}.

The receiver carries out the same polynomial division operation on the combination of the message bits and CRC. If the remainder is not zero then at least one of the bits must have changed and an error has been detected.

Computing the CRC

Computing the CRC requires polynomial division. The process involves repeated subtraction of the generator polynomial from the message polynomial. Unlike regular division, to compute the CRC we only need to compute the remainder.

Exercise 10: If the generator polynomial is $G(x) = x^3 + x + 1$ and the data to be protected is 1001, what are $n-k, M(x)$ and the CRC? Check your result. Invert any one to three bits of the message and compute the remainder again. Add the generator polynomial, or a shift of it, to the message and compute the CRC again.

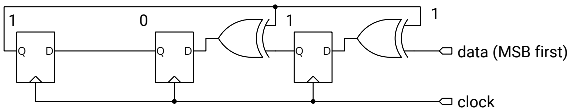
A circuit to perform the division of the polynomials can be implemented using a shift register that holds the result of the intermediate remainder after each subtraction. The shift register only has to hold $(n-k)$ bits.

Digital Implementation of Polynomial Arithmetic

Arithmetic on polynomials with $GF(2)$ coefficients can be implemented with simple digital logic circuits. Flip-flops, organized as shift registers, store the bits of the message (coefficients equal to 1 or 0) and XOR gates compute modulo-2 addition. The bits corresponding to codeword(s)/message(s) can be input and output sequentially, bit by bit, into the circuits.

It's much simpler to do arithmetic using polynomials in GF(2) than using regular integers because we do not need to compute carries when computing results.

A circuit which can compute the CRC for the 4-bit generator polynomial in the example above is shown below:



The shift register bits are initialized to zero so that the first $n - k$ (data) bits are loaded unchanged. At each subsequent step in the division the generator polynomial is subtracted by the xor gates from the intermediate remainder in the shift register if the leftmost bit is 1, otherwise the intermediate remainder is unchanged. After a total of n clock cycles the shift register holds the $n - k$ bits of the final remainder, $R(x)$. This is shifted out at the transmitter (as an appended CRC) or checked (for zero) at the receiver.

Checking the CRC

At the receiver the same circuit can be used to divide the received message and the appended remainder

[1] Generator polynomials "generate" other codewords, in this case the CRC.

polynomial by the generator polynomial. If the remainder is zero then the received polynomial must be a multiple of the generator polynomial. This is always the case when we subtract the remainder $R(x)$ from the message polynomial. Therefore if the remainder in the shift register is non-zero then there must have been an error.

CRC Error Detection Performance

CRC error detection will fail only if the error pattern is a multiple of $G(x)$.

If all the errors are located within an "error burst" of length $n - k$ then the error pattern cannot be a multiple of $G(x)$ and is guaranteed to be detected. However, the CRC will also detect most longer bursts since they are unlikely to be a multiple of $G(x)$.

Exercise 11: Is a 32-bit CRC guaranteed to detect 30 consecutive errors? How about 30 errors evenly distributed within the message?

A common situation is where the received bits are completely random (e.g. noise being detected as data). In this case the probability of not detecting an error is the probability that a random sequence of $n - k$ bits matches the required checksum.

Exercise 12: What is the probability that a CRC of length $n - k$ bits will be the correct CRC for a randomly-chosen codeword? Assuming random data, what is the undetected error probability for a 16-bit CRC? For a 32-bit CRC?

Detecting Added/Deleted Zero Bits

We can add or remove any number of leading zero coefficients to $M(x)$ without affecting its value or the CRC. To allow the CRC to detect missing/added leading zero bits, most implementations require that some initial data bits (typically the first $n-k$) be complemented.

Similarly, appending or deleting zeros to the end of the message will also result in a zero remainder. We can avoid this problem by complementing the CRC before sending it. This generates a non-zero remainder but the value will be a specific value (the same for all messages) if there are no errors.

Another way to detect missing/added leading/-trailing zero bits is to include the length of the message in the CRC computation.

Standard CRC Generator Polynomials

There are several CRC generator polynomials in common use. The most common lengths are 16 and 32 bits since these are multiples of 8 bits. All(?) IEEE 802 standards use the same 32-bit CRC polynomial typically called "CRC-32". The ITU has defined a 16-bit CRC generator polynomial ("CRC-16-CCITT") that is also used in various standards.