INTRODUCTION TO CYCLIC CODES AND ILLUSTRATION OF THEIR IMPLEMENTATION THROUGH LINEAR FEEDBACK SHIFT REGISTERS WITHIN A PROGRESSIVE WEB APPLICATION

Ioana Lavinia SOFIA
Department of Engineering and Management in Military, Land Forces Academy "Nicolae Bălcescu, Sibiu
ioanasofia30@yahoo.com

Abstract: This paper provides an introductory overview of cyclic codes of Hamming type and showcases their effectiveness in detecting and correcting errors in digital communications. Thus, a Progressive Web Application has been developed through React.js and p5.js JavaScript libraries to simulate the coding operations using Linear Feedback Shift Registers. This GUI serves an educational purpose, being used in Information Transmission Theory laboratories so that students could easier comprehend this particular coding scheme. The results obtained highlight the efficiency of developed software application in the e-learning process of cyclic code of Hamming type and its adeptness in error detection and correction within digital communications.

Keywords: *cyclic codes, digital communications, LFSR, React.js, p5.js, Progressive Web Application, GUI*

I. INTRODUCTION

The direct transmission of information produced by a source is feasible only under circumstances where there is compatibility with the transmission or storage medium, propagation issues and interferences being minimal. In most cases, before the information is transmitted or stored, it undergoes processes such as encoding, modulation, and synchronization, as illustrated in Figure 1.

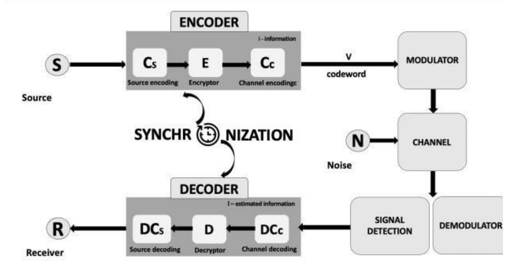


Figure 1. The structural model of a digital or numerical data transmission system

In his work titled "A Mathematical Theory of Communication", C.E. Shannon introduced the concept now known as Shannon's second theorem, which provides a solution for managing disruptions in the transmission and storage of information [1].The theorem states that on a noise channel having a certain capacity *C*, it is possible to achieve real- time transmission of a source of information with a rate $\dot{D}$ lower than the channel's capacity, with an extremely low probability of errors *P(E)*, by using a code with a certain length n that includes redundancy through the k control bits. This code mathematically is equal to the sum of the information bits *m* and the control bits *k*. In practice, the strategy of intentionally adding redundancy before transmitting the information is used to protect it against disruptions [2].

Cyclic codes are an example of codes that include redundancy, which are appreciated in practice due to their easy implementation using feedback shift registers, making them suitable in terms of hardware resource optimization.

The purpose of this work is to create a Progressive Web Application (PWA) that shows how coding works for the single-error correcting Cyclic Hamming Code through an animated representation of the operation mode of LFSR (Linear Feedback Shift Register). Thus, the developed application becomes an educational tool, facilitating the learning process for students regarding such codes.

II. THEORETICAL BACKGROUND

A. Definition of cyclic code of Hamming type

Unlike the Hamming group code, the single-error correcting cyclic code of Hamming type is a systematic code, characterized by the relationship:

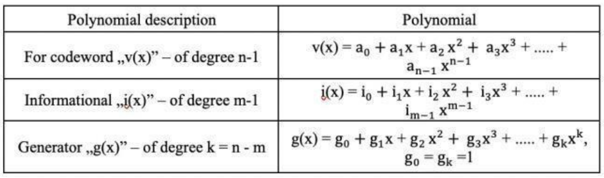
$$n = 2^k - 1 \qquad (1)$$

The code word is defined by the following structure:

$$v = [a_0 a_1 a_2 \dots a_k a_{k-1} \dots a_{n-1}] \qquad (2)$$

The initial *k* bits denote the control bits, whereas the remaining *n - k* bits denote the information bits, labeled as *m*. In a systematic code, the first *m* bits of a codeword are the original or information bits, while the remaining *k* bits are redundant or error-checking bits.
Cyclic codes are represented mathematically in polynomial form, as can be observed in the Table 1.

Manuscript received March 22, 2024; revised June 2, 2024



Polynomial description	Polynomial
For codeword „v(x)" – of degree n-1	$v(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_{n-1}x^{n-1}$
Informational „i(x)" – of degree m-1	$i(x) = i_0 + i_1x + i_2x^2 + i_3x^3 + ... + i_{m-1}x^{m-1}$
Generator „g(x)" – of degree k = n - m	$g(x) = g_0 + g_1x + g_2x^2 + g_3x^3 + ... + g_kx^k$, $g_0 = g_k = 1$

Table 1 – Polynomial representation of the generator polynomial g(x), informational polynomial i(x), and the codeword polynomial v(x)

The single-error correcting cyclic code of Hamming type is actually a BCH (Bose-Chaudhuri-Hocquenghem) code, capable of correcting a single error *t=1*, with a length of *n*, where the generator polynomial *g(x)* is the primitive polynomial of degree *k* which is the order extension of binary Galois Field. The generator polynomial determines the number of redundant bits added to the information bits before transmission over the communication channel [2].

B. Definition of LFSR

A shift register is a linear circuit comprising a sequence of flip-flops utilized for storing multiple bits of data and transferring information from one memory cell to another upon the application of a clock signal, whether within or outside the system. To form a shift register with *k* bits, *k* flip-flops are required. A feedback shift register operates independently, solely based on the feedback signal. The connections within the register correspond to the generator polynomial:

$$g(x) = g_0 + g_1x + \cdots + g_{k-1}x^{k-1} + g_kx^k \quad (3)$$
$$g_k = 1, g_i \in \{0,1\}$$

C. Description of coding process

The encoding and decoding operations in the case of the single-error correcting cyclic Hamming code can be performed using both external and internal LFSRs (Linear Feedback Shift Registers) with external modulo-2 adders. In the developed software application, only the circuit containing external XOR gates will be addressed. The corresponding schema for the encoding process can be found in Figure 2.

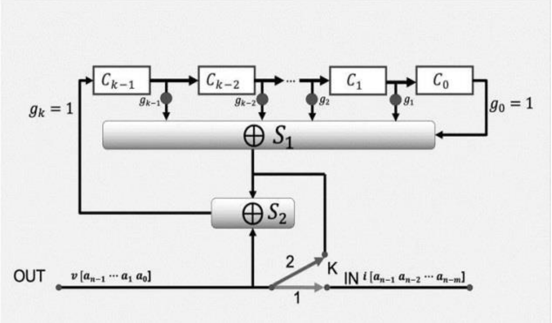


Figure 2. Block diagram of a systematic cyclic encoder

The switch *K* operates differently depending on its position: *1* or *2*. Thus, *K* is in position *1* for *m* clock cycles until the LFSR is filled with the information bits. After these *m* clock cycles, *K* switches to position *2*, this time for *k* clock cycles, during which the LFSR will compute the control bits through the division operation $\frac{i(x)}{g(x)}$ [3]. At the output, we obtain the cyclic codeword in systematic form, as can be observed:

$$v(x) = x^k i(x) + r(x) \quad (4)$$

The remainder r(x) is defined as:

$$r(x) = \frac{x^k i(x)}{g(x)} \quad (5)$$

At time *t=n*, the state of the LFSR becomes zero [2]. At reception, errors can be detected and/or corrected. The corresponding block diagram for error detection is presented below:

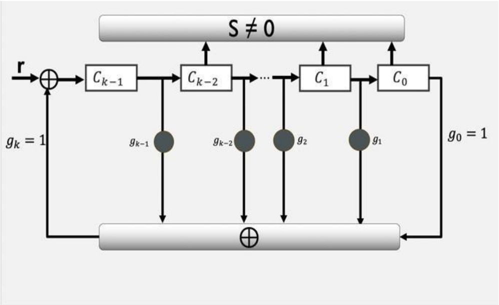


Figure 3. Block diagram of a systematic cyclic decoder

Both at encoding and decoding, the state of the LFSR at the end of the *n* clock cycles is zero. A necessary condition for detecting errors that occur during transmission (or storage) is that the syndrome defined as $s(x) = \frac{r(x)}{g(x)}$ be non-zero. This division is the task of the decoder as it can be observed in Figure 3 (S represents the syndrome).

The syndrome provides information about the nature and location of errors in a transmitted signal, allowing the system to determine how to correct or manage these errors to restore the correct information.

If *s(x) ≠ 0* the error is detected. For error correction with cyclic codes, *2n* clock cycles are required: once the entire word is received (after the first *n* clock cycles), the error can become detectable, and the correction is performed during cycles *(n+1, 2n)* [3], by adding the value *1* to the position containing the error [2].

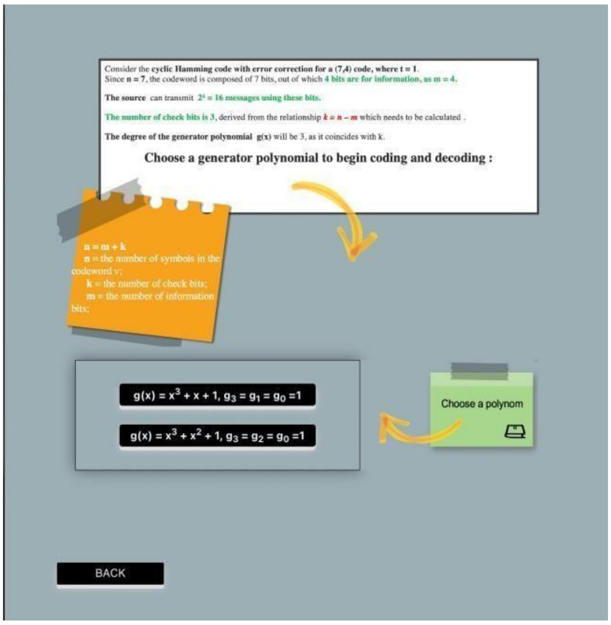
D. Applications of cyclic codes

Cyclic codes have a wide range of applications in digital technology, such as in storage devices (CDs and DVDs) to protect data against corruption, as well as for error correction in RAM memories [3]. They are also used in satellite telecommunications and wireless communications (technologies like Wi-Fi, LTE), contributing to ensuring the quality and integrity of signals in interference-prone environments.

III. IMPLEMENTATION

A. Progressive Web Application

To demonstrate the encoding and decoding mechanism through LFSR, a progressive web application (PWA) was developed using web technology like React.js and p5.js. This application is accessible online, even offline offering the functionality to be installed on both mobile phones and desktops, regardless of the device's platform [4]. Users can simulate the encoding for the cyclic code C(7,4) by selecting a polynomial of their choice, as depicted in the figure below. Figure 4 depicts a GUI (Graphical User Interface) component that includes explanations regarding the dimensioning of the cyclic Hamming error-correcting code treated by the software application, namely, *C(7,4)*, as well as the option to select one of the two generator polynomials that contribute to the information coding process. Thus, the user can observe how the same information sequence can be encoded depending on the chosen polynomial, while the theoretical explanations help them grasp the knowledge necessary to understand the operating principle of the *C(7,4)* code through LFSR.



Consider the **cyclic Hamming code with error correction for a (7,4) code, where t = 1**.
Since **n = 7**, the codeword is composed of 7 bits, out of which **4 bits are for information, as m = 4.**

The source can transmit **$2^4 = 16$ messages using these bits.**

The number of check bits is 3, derived from the relationship ***k = n – m*** which needs to be calculated .

The degree of the generator polynomial g(x) will be 3, as it coincides with k.

Choose a generator polynomial to begin coding and decoding :

n = m + k
n = the number of symbols in the codeword v;
k = the number of check bits;
m = the number of information bits;

$g(x) = x^3 + x + 1, g_3 = g_1 = g_0 = 1$
$g(x) = x^3 + x^2 + 1, g_3 = g_2 = g_0 = 1$
Choose a polynom
BACK

Figure 4. GUI with generator polynomial for coding simulation process

React.js is a library used to render UI (User Interface) components, working in parallel with ReactDOM, which enables the creation of web applications. React.js essentially extends the syntax of the JavaScript programming language with HTML, resulting in JSX.[5].

Visual Studio Code was used as the software development environment and encompasses the best web technologies specific to programming languages such as JavaScript and Node.js. Node.js is an open-source execution environment built on the V8 JavaScript engine developed by Google [6].

B. Auxiliary libraries

Material-UI represents a comprehensive open-source library of user interface components.

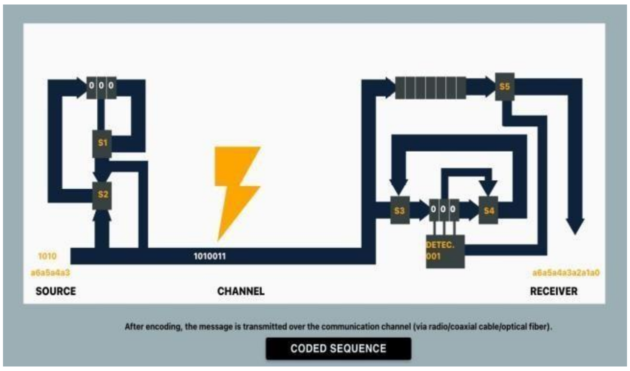
React-router-dom was designed to simplify routing management in React applications, examples of routing components used include: <Routes>, <Route path="/path" element ={<Component Name />} />, <BrowserRouter>.

React-i18next was used as a library for managing translations in React applications; the developed application s translated into the two widely spoken international languages: English and French.

React Testing Library has provided the fundamental tools to thoroughly inspect the source code. With the aid of its accessible and intuitive methodologies, it is possible to emulate user interactions and validate the behavior of the created components in various contexts. When testing with React Testing Library, developers typically write test cases that mimic how a user would interact with the application. For example, they might simulate clicking on a button, entering text into a form field, or navigating to a different page.

Once these interactions are simulated, developers can use assertions to check whether the application responds correctly. This could involve verifying that the correct data is displayed on the screen, that certain components are rendered or hidden under specific conditions, or that the application behaves as expected in response to user input.

P5.js is a library for creative programming and interactive drawing in web environments [7]. In the developed application, it was used to simulate shift registers, the communication channel, both the encoder, and the decoder, as it can be observed in Figure 5.



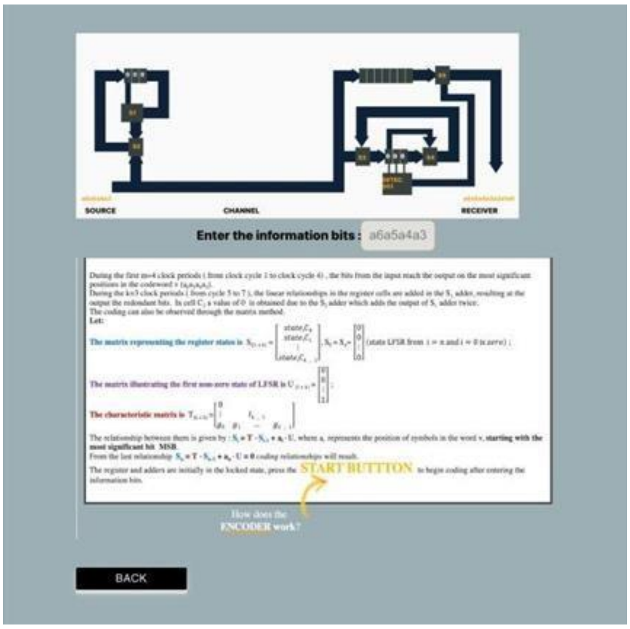
After encoding, the message is transmitted over the communication channel (via radio/coaxial cable/optical fiber).

Figure 5.Animation with encoder, decoder, LFSR and communication channel through p5.js

These were installed using the NPM package manager, which is a widely used software library globally and can be freely utilized by any programmer, being installed alongside Node.js. Public software packages were downloaded through the Command Line Interface (CLI).

To facilitate user access to the application via the internet, it was hosted on a server using the Static Web Apps service provided by Microsoft Azure, a cloud computing platform that offers numerous cloud infrastructure services, including storage, computing, and many others.

The application initially displays the "Home" page, which contains an introduction regarding the necessity of coding the communication channel and information about the inventor of the Hamming code. In the header's left part, the user can access the other components of the application: "Course", "Notes" (a minimal application where students can jot down ideas from the course), "Laboratory" (in this section, the user can simulate cyclic encoding and decoding through LFSR in the case of the single-error correcting cyclic code of Hamming type), "Test" (the user can evaluate their acquired knowledge), "About" (provides details about the application authors and its objectives).



Enter the information bits : a6a5a4a3

During the first m=4 clock periods (from clock cycle 1 to clock cycle 4) , the bits from the input reach the output on the most significant positions in the codeword v ($a_6a_5a_4a_3$).
During the k=3 clock periods (from cycle 5 to 7), the linear relationships in the register cells are added in the S_1 adder, resulting at the output the redundant bits. In cell C_1 a value of 0 is obtained due to the S_2 adder which adds the output of S_1 adder twice.
The coding can also be observed through the matrix method.
Let:
The matrix representing the register states is $S_{(1\times k)}=\begin{bmatrix} state_{C_k} \\ state_{C_1} \\ \vdots \\ state_{C_{k-1}} \end{bmatrix}$, $S_0=S_n=\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ (state LFSR from i = n and i = 0 is zero) ;
The matrix illustrating the first non-zero state of LFSR is $U_{(1\times k)}=\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$;
The characteristic matrix is $T_{(k\times k)}=\begin{bmatrix} 0 & & I_{k-1} & \\ \vdots & & & \\ g_0 & g_1 & \cdots & g_{k-1} \end{bmatrix}$
The relationship between them is given by : $S_i = T \cdot S_{i-1} + a_i \cdot U$, where a_i represents the position of symbols in the word v, **starting with the most significant bit MSB**.
From the last relationship $S_n = T \cdot S_{n-1} + a_n \cdot U = 0$ coding relationships will result.
The register and adders are initially in the locked state, press the START BUTTON to begin coding after entering the information bits.

How does the ENCODER work?

Figure 6. Page containing coding process using LFSR

The user can introduce their own information bits from the keyboard (in Figure 6 it can be observed “Enter the information bits” input), select the generator polynomial (Figure 4), indicate the error position in the transmitted message, and observe how encoding occurs at each clock cycle using the registers, as well as error detection and correction (Figure 7). The created animation simulates how the information bits from the source are transmitted (stored) over the noise channel and decoded at the receiver (Figure 5).

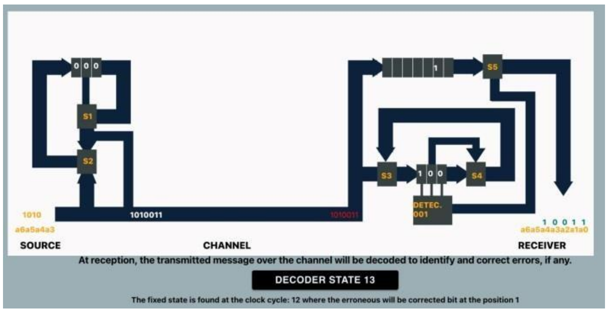
In the realm of e-learning and blended learning techniques, the development and utilization of interactive web applications hold significant promise, particularly in enhancing the study of complex concepts like cyclic codes. By harnessing widely-used software libraries like NPM and cloud computing platforms like Microsoft Azure, educators can create accessible and dynamic learning environments.

The integration of functionalities such as simulating cyclic encoding and decoding processes, as exemplified in this application, empowers students to engage directly with theoretical concepts. Thus, by leveraging modern technology and e-learning methodologies, educators can foster more engaging and effective learning experiences in the study of cyclic codes.

IV. EXPERIMENTAL RESULTS

It is assumed that the user wants to encode the following sequence containing *4* bits of information, respecting the order *a6a5a4a3*: *1010* (which can be observed in the application above the Source field). According to the cyclic code of Hamming type correcting a single error, denoted *C(7,4)*, the number of redundant bits is *3*, thus, the total number of bits, as observed in the graphical interface, is *7*. The encoded sequence *1010011 (n=7)* is obtained after seven clock cycles: in the first *m = 4* cycles, the information is encoded, and in the remaining *k = 3* cycles, the redundant bits are also encoded using LFSR. Subsequently, an animation simulates the transmission of the message over the communication channel. Since the cyclic Hamming code is capable of correcting a single error, the user can choose the position of the distorted bit during transmission. In the screenshot above, the user introduced the error at position *1* in the message, and its detection is possible in the next *(1, n)* cycles, while the correction occurs over *(n+1, 2n)* cycles (the correction is performed when the register reaches the fixed state by adding the value *1* to the erroneous bit in the *12th* cycle). It can be observed in the Figure 7 that the sequence at the Receiver coincides with the one before being transmitted over the communication channel.

In addition to this, the application has been tested by the first generation of students from the Land Forces Academy “Nicolae Balcescu” during the laboratory of Information Transmission Theory lecture, and their positive feedback confirms its functionality and effectiveness in the learning process of cyclic codes of Hamming type.



At reception, the transmitted message over the channel will be decoded to identify and correct errors, if any.

The fixed state is found at the clock cycle: 12 where the erroneous will be corrected bit at the position 1

Figure 7. Encoding and decoding process of 1010 information bits through C(7,4) correcting a single error and LFSR

The animations provided have assisted students in gaining a better understanding of the coding process using LFSR by simulating various scenarios. Through the available options, such as selecting the generator polynomial, introducing as inputs their own information block, choosing the error position resulting from message transmission over the communication channel, and observing the register states at each clock cycle, the developed application serves as a valuable educational tool.

V. CONCLUSIONS

In conclusion, the Progressive Web Application titled "Cyclic Encoding and Decoding through Feedback Shift Registers" stands out as a key educational resource for exploring error detection and correction codes, representing a significant advancement towards a more efficient and contemporary learning method. In a continuously evolving academic environment, this application places the study of cyclic encoding and decoding with feedback shift registers in a modern and easily accessible framework, highlighting the potential for innovation in the field of education.

REFERENCES

[1] Claude Shannon, Warren Weaver, *The Mathematical Theory of Communication*, The University of Illinois Press, 1963 [Accessed: February 23, 2024].

[2] M. Borda, *Fundamentals in Information Theory and Coding*, Springer, 2010 [Accessed : February 23, 2024].

[3] S. M. Sunita, V. S. Kanchana Bhaaskaran, Deepakakumar Hegde and Pavan Dhareshwar, *Error Detection and Correction in Embedded Memories Using Cyclic Code*, Conference: Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking [Accessed : February 23, 2024].

[4] https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps [Accessed: February 24, 2024].

[5] https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing [Accessed: February 24, 2024].

[6] https://developer.mozilla.org/en-US/docs/Glossary/Node.js [Accessed: February 24, 2024].

[7] https://p5js.org/ [Accessed: February 24, 2024].