Laura Guijarro
Daniel Olivares

Analysis of LFSR structure for n<13 bits

Content

1- Introduction

In this project, we are going to describe and analyse a linear feedback shift register (LFSR) and make some applications using the board CoolRunner-II CPLD and VHDL code. Also we are going to study the LFSR as a Stream Ciphers to make a cryptography application implementing it in Matlab.

2- Main properties of FSR (Feedback Shift Register)

A feedback shift register FSR is constituted by two parts: a shift register and a feedback function. The shift register is composed of storage units of one bit, so it can be said that a shift register is a sequence of bits. The length of the shift register depends on the number of storage units or the number of stages that it contains, a shift register with length L, has L stages numbered from 0 to L-1. A feedback register has one input, one output and a clock input, that controls the data flows. During a clock period these operations are made:

1. The bit of the stage cero is the output of the register and it is a part of the output sequence.
2. The content of the stage *i* goes to the stage *i-1*, $1 \leq i \leq L-1$.
3. The new content of the stage *L-1* is the feedback bit.

$$S_j = f(s_{j-1}, s_{j-2}, \dots \dots \dots \dots .., s_{j-L})$$

Where the feedback function f is a Boolean function and s_{j-i} is the content of the previous stage.

If the content for the initial stage is $s_i \in \{0,1\}$ for every $0 \le i \le L-1$ the sequence $[s_{L-1}, \dots\dots\dots\dots.., s_{1,} s_0]$ is called the initial value of the FSR.

The figure number 1 shows a FSR, if f is a linear function the FSR is a LFSR *(Linear Feedback Shift Register)*. Otherwise it is a NLFSR *(No-Linear Feedback Shift Register).*

If the feedback function f of a FSR can be written like

$$f(s_{j-1}, s_{j-2}, \dots\dots\dots\dots.., s_{j-L}) = (C_1 s_{j-1} \oplus \dots\dots\dots \oplus C_L S_{j-L})$$

in every unit of time, then is said that the FSR is lineal, where $C_{1,} C_2 \dots\dots.., C_L$ are the feedback coefficients and $\oplus$ is the sum module2.

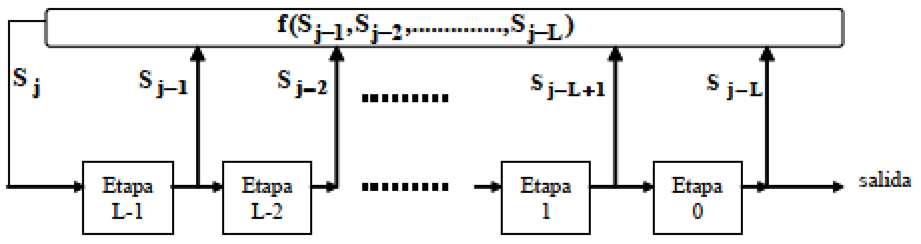


Figure 1

3- LFSR (Linear Feedback Shift Register)

The linear feedback Shift Register produce a sequence of $2^n - 1$, where n is the number of stages in the LFSR. The content of the register is the movement to the right of one position, for each positive transition in the clock. There is a feedback produced by an XOR or XNOR operation between certain stages; the result of this operation is a bit that is inserted to the left of the register. A register that is initialized with only values "1" will not be convenient because if the feedback function is an XNOR, the value of this register will not change and will remain in the same state. Similarly, it is not convenient for a register to be initialized with only zeros when its feedback function is an XOR, because the record will not change state. In these two cases the states are illegal because the counter would remain in a single state (locked-up).

For example, a 4-bit LFSR produces a sequence of 2^4 - 1 = 15 (state 1111 is an illegal state, using an XNOR feedback function). On the other hand, a 4-bit ascending binary counter would produce a sequence of 2^4 = 16 with non-illegal states. LFSR counters are very fast since they do not use carry signals. The LFSR can replace conventional binary counters in the realization of critical applications where the sequence counter is not important. LFSRs are used as generators of bitstream in pseudo-sequences of stream bits. They are also important in the implementation of encryption and decryption algorithms. We have to considerate:

- **The length of the shift register (N).** It refers to the degree and in general to the length of the shift register, the longest duration of the pseudo-noise sequence before it is repeated. For a fixed length shift register N, the number and duration of the sequences that can be generated are determined by the number and position of the steps used to generate the feedback bit.

- **Stages of the shift register**. The combination of stages and their location is regularly referred to as a polynomial and is expressed as:

$$P(x) = 1 + x^3 + x^7$$

Where the first "1" represents x^0, which is the output of the XOR and represents the last stage of the register. X^3 is the output of the register of the stage 3 and x^7 is the first stage of the register and the output of the LFSR.

- **Maximum length sequence (L).** The maximum length sequence for a register of length N refers to an m-sequence and is defined as:

$$L = 2^N-1$$

The LFSR also have many variables to consider as:

- The number of stages in the shift register
- The number of stages in the feedback path
- The position of each stage in the shift register
- The initial condition of the start of the shift register.

4- Implementation of the LFSR

There are two styles of implementations of the LFSR, the implementation of Galois and the implementation of Fibonacci.

4.1 Implementation of Galois

As shown in Figure 2, the data flow is from left to right and the feedback path is from right to left. The increment of the polynomial is from left to right with the term x^0 ("1" in the polynomial) as the first term in the polynomial. This polynomial indicates which stages of the shift register are fed back. The XOR gate is in the trajectory of the shift register, therefore, the implementation of Galois is also known as in line, modular type or LFSR type M [6].

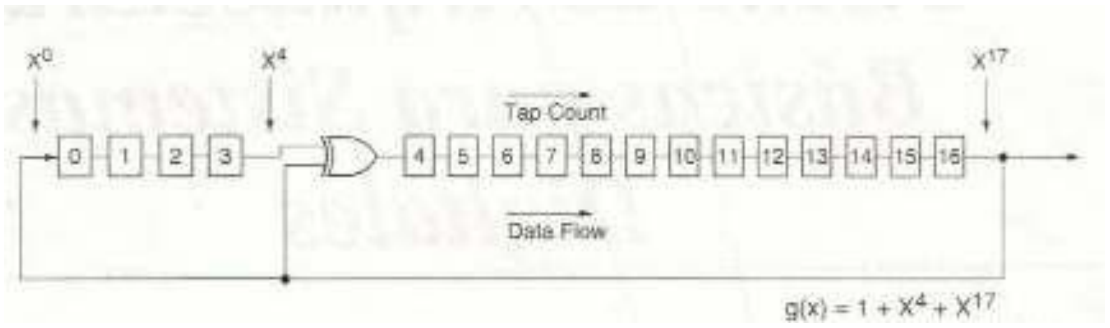


Figure 2

4.2 Implementation of Fibonacci

In Figure 3 the flow of data is from the left to the right and the feedback path is from right to left, similar to the implementation of Galois. However, the Fibonacci implementation the polynomial decreases from left to right with X^0 as the last term in the polynomial. This polynomial is mentioned as a polynomial of reciprocal stage and the feedback stages are incremented from right to left along the shift register. The XOR gateway is the feedback path, therefore, the implementation of Fibonacci is also known as offline, simple type or LFSR type S.

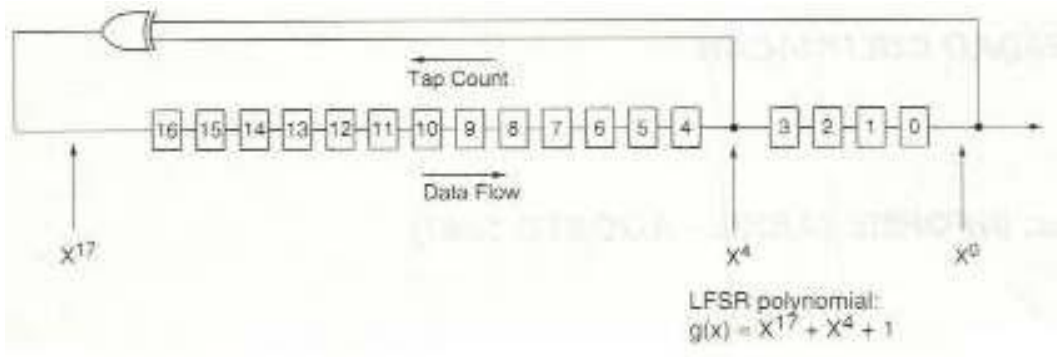


Figure 3

5- Description of the hardware used

CoolRunner-II:

The CoolRunner-II Board is a USB-powered circuit development platform for Xilinx's CoolRunnerII CPLD. The board includes power supplies, a programmable 8MHz fixed-frequency oscillator, several I/O devices, and a USB2 port for board power and CPLD programming. The board also includes five expansion connectors that make 64 CPLD signals available to external circuits. The board is show in the figure 4.

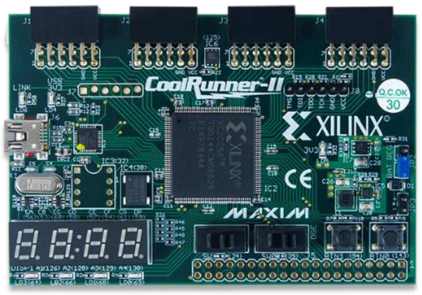


Figure 4

Due to the architecture of the board, it's impossible to update the value of more than one position in the BCD display at the same time. To solve that, by reducing the clock speed, each position is updated in a different cycle, but undetectable for the human eye.

6- APLICATTION 1. LFSR of 4 bits

We are going to create a LFSR only with 4 bits to show in the 7 segment of the board the successive displacements of the shift register. As the 4 segments are connected in parallel (as we have explained before) we will use the period of the clock to show successively the 4 bits before making the displacement.

The example that we took is described below.

- It will have length 4.
- It will be built with a primitive polynomial $1 + x + x^4$, of degree 4. Figure 5 shows the correspondence of the value 9 in the VHDL code and the polynomial $1 + x + x^4$.
- The seed will be 1011 and according to figure 4 the value will be 11 in the VHDL code.

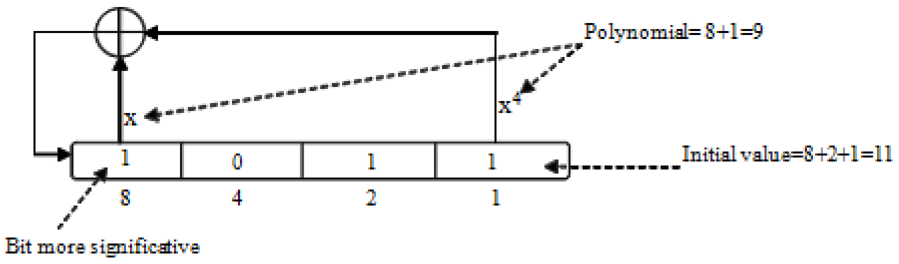
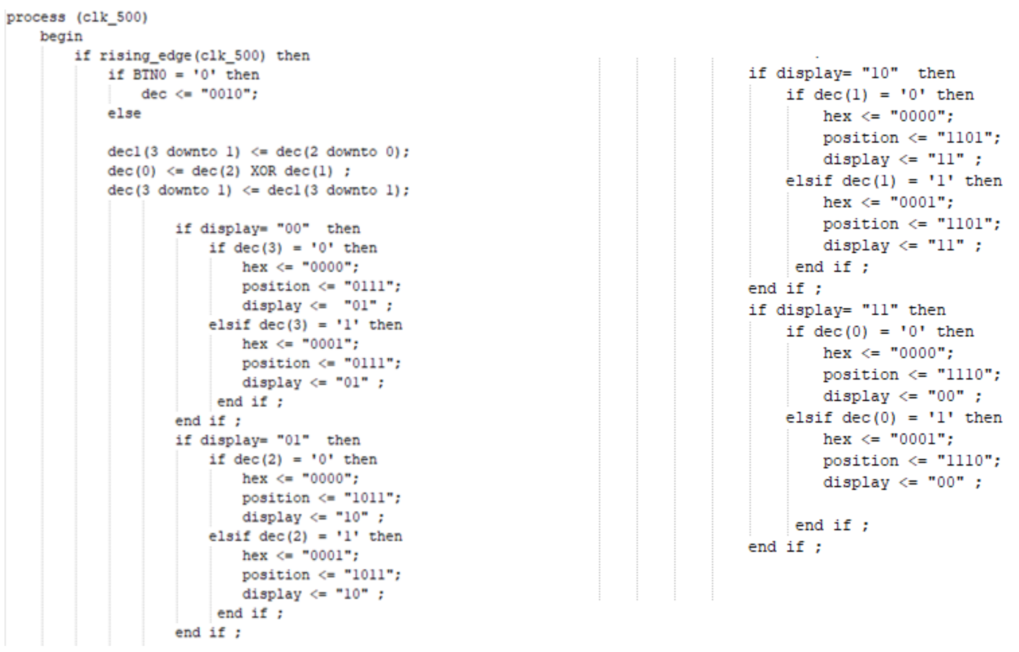


Figure 5

Code



```
process (clk_500)
    begin
        if rising_edge(clk_500) then
            if BTN0 = '0' then
                dec <= "0010";
            else

            decl(3 downto 1) <= dec(2 downto 0);
            dec(0) <= dec(2) XOR dec(1) ;
            dec(3 downto 1) <= decl(3 downto 1);

                    if display= "00"  then
                        if dec(3) = '0' then
                            hex <= "0000";
                            position <= "0111";
                            display <=  "01" ;
                        elsif dec(3) = '1' then
                            hex <= "0001";
                            position <= "0111";
                            display <= "01" ;
                         end if ;
                    end if ;
                    if display= "01"  then
                        if dec(2) = '0' then
                            hex <= "0000";
                            position <= "1011";
                            display <= "10" ;
                        elsif dec(2) = '1' then
                            hex <= "0001";
                            position <= "1011";
                            display <= "10" ;
                         end if ;
                    end if ;

if display= "10"  then
    if dec(1) = '0' then
        hex <= "0000";
        position <= "1101";
        display <= "11" ;
    elsif dec(1) = '1' then
        hex <= "0001";
        position <= "1101";
        display <= "11" ;
     end if ;
end if ;
if display= "11" then
    if dec(0) = '0' then
        hex <= "0000";
        position <= "1110";
        display <= "00" ;
    elsif dec(0) = '1' then
        hex <= "0001";
        position <= "1110";
        display <= "00" ;

     end if ;
end if ;
```

Figure 6

7- Cryptography

With the entry into the information age, the problem of security and data authentication has begun to be an inseparable aspect of communication systems. Each time a greater amount of information is stored and transmitted by electronic means and therefore they are exposed to attacks. Individual privacy as well as organizations depend on the possibility of protecting information from unauthorized access and modification.

Cryptographic systems provide privacy in information through the use of transformations. In the transmitter the clear text is transformed into the encrypted text under the control of an encryption key, which makes this information incomprehensible to other people. Only the correct receiver under the control of a decryption key retransforms the encrypted text in the clear text.

Cryptographic systems are divided into those that are secret or **symmetric key** and public or **asymmetric key**. In public key or asymmetric cryptography, the transmitter uses a publicly known key to send a message to the receiver. The receiver uses a secret key to recover the clear text. In secret key cryptography, the transmitter and receiver previously agree to use a private key that will encrypt and decrypt the clear text. This key must remain secret to protect the clear text against possible cryptanalysts.

General Structure

The symmetric key cryptographic systems can be divided into **block ciphers** and **stream ciphers**. When the encryption operates in each entity of the clear text independently, then we are talking about block encryption. Block encryption is a simple substitution cipher and should be made to large portions of the clear text to prevent brute force attacks. The block name is used to limit the size of the clear text entity to be encrypted. The stream cipher, in contrast, encrypts each clear text entity by means of a Boolean function, where the clock frequency governs the internal states of the stream cipher. For this encryption principle, the clear text entity does not need to be large, since the data in it is bit-bit encrypted. After each bit is encrypted the flow cipher changes state according to some type of rule. Therefore, two occurrences of the same clear-text character usually do not result in the same character as the encrypted text.

To obtain a clear distinction between block cipher and stream cipher, we have the Figures 7 and 8.

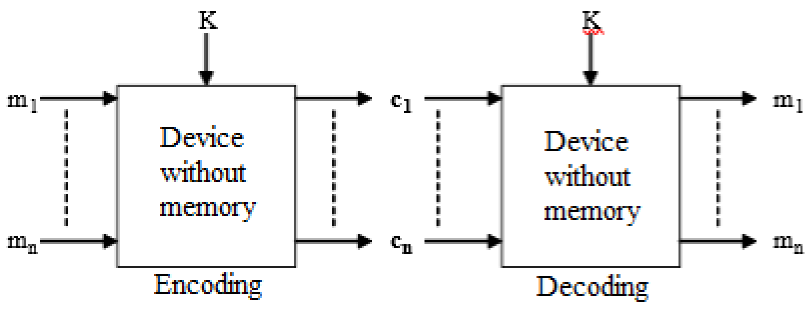


Figure 7 - Block cipher

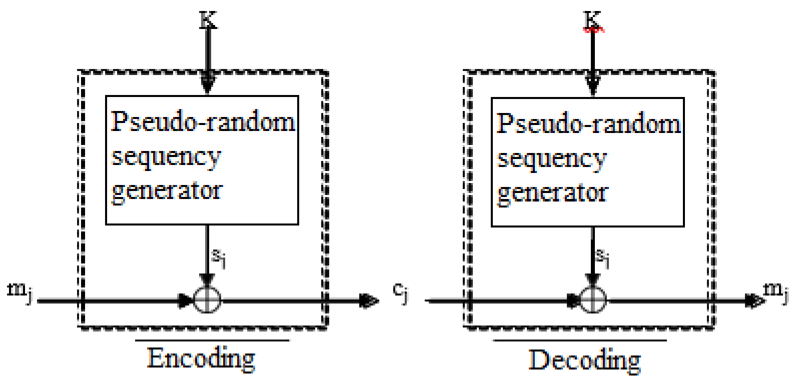

K
Pseudo-random sequency generator
sj
mj
cj
Encoding
K
Pseudo-random sequency generator
sj
mj
Decoding

Figure 8- Stream cipher

The stream cipher as shown in Figure 7 can be performed by means of pseudorandom sequence generators, which use a function *f* and a key, which is generally made by

the initial value of the generator. Using the pseudorandom sequence every bit of the clear text is encrypted using a Boolean function, for example an XOR. For generate this pseudorandom sequence we are going to use the LFSR.

8.1 Stream coding based on LFSR.

The encryption of flow encrypts the data of a clear text bit by bit, using a coding sequence and a Boolean function. This encryption can be done using LFSR (Displacement Register with Linear Feedback), which generates the key (pseudorandom sequence) to encrypt a clear text. This encryption, made with LFSR, has the following important properties:

1. LFSR can be easily performed on hardware.
2. They can generate sequences of large period T, with T = 2L-1 and where L is the length of the LFSR.
3. They produce sequences with good statistical propertie.
    - The difference in the number of ones and zeros within each period of the sequence should not exceed unity.
    - Throughout the period, $(1/2^i)$ is the number of i-grams (of the total grams throughout the period) of length i.
    - The autocorrelation fuction AC (k) out-of-phase is constant for every value k.

4. Due to their structure, they can be analyzed using algebraic techniques, one of which is linear complexity.

In telecommunications, flow encryption is widely used under the following conditions:

• Storage space is limited.

• The characters of a message must be processed individually and received in the same way.

• The transmission error has a high probability.

• The end-to-end delay should not be significantly increased (affects interactive applications).

• The data flow to be encrypted is generated continuously and has a considerable volume.

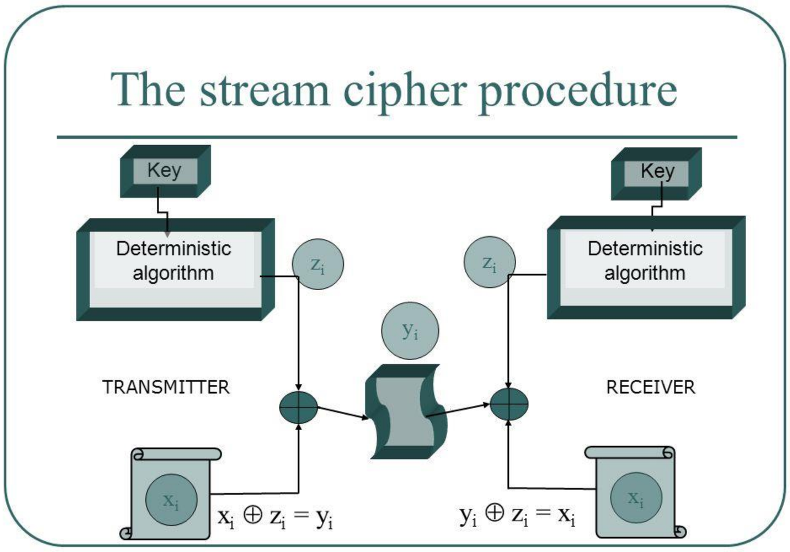


Figure 9

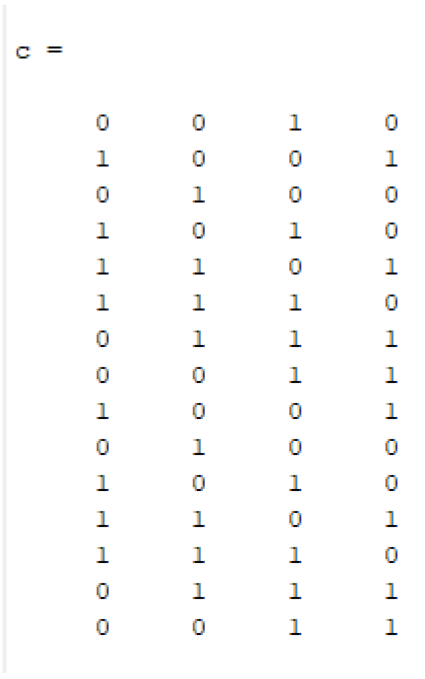
8.2 APPLICATION OF STREAM CIPHER LFSR.

First we have to create our LFSR, For that we are going to use the function f(x)= 1+x^2+x^3 with length 4.

MATLAB Code

```
%Implementation of LFRS

s=[0  0  1  0 ] %Initial value
t=[3  2] %fuction f(x)= 1+x^2+x^3

n=length(s);
c(1,:)=s;
m=length(t);
for k=1:2^n-2;
b(1)=xor(s(t(1)), s(t(2)));
if m>2;
    for i=1:m-2;
    b(i+1)=xor(s(t(i+2)), b(i));
    end
end
j=1:n-1;
s(n+1-j)=s(n-j);
s(1)=b(m-1);
c(k+1,:)=s;
end
seq=c(:,n)';
```

We obtain our pseudorandom sequence matrix. Each file will cypher the different messages.



```
c =

     0     0     1     0
     1     0     0     1
     0     1     0     0
     1     0     1     0
     1     1     0     1
     1     1     1     0
     0     1     1     1
     0     0     1     1
     1     0     0     1
     0     1     0     0
     1     0     1     0
     1     1     0     1
     1     1     1     0
     0     1     1     1
     0     0     1     1
```

As we can see we have all the possible combinations 2^4-1= 15, and no prohibit states are show.

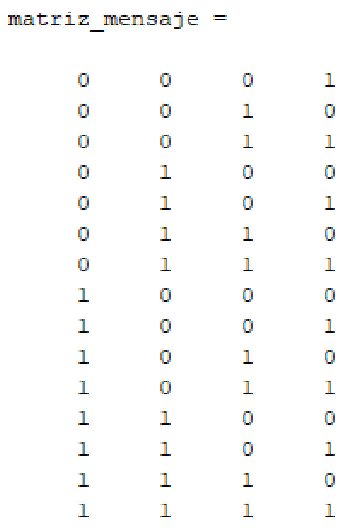
Matrix 1.Pseudorandom sequence

Now we have to do the stream cypher. For the message that we have to send, I am going to use the real numbers from 1 to 15.

MATLAB Code

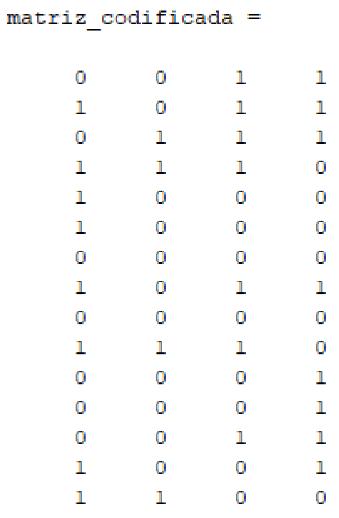
```
n=4; %length of the cypher

for i=1:2^n -1
    numero = i
    numero_binario =dec2bin(i,n)

    vector_numero_binario=num2str(numero_binario)-'0'

    codigo_cod=c(i,:)
    numero_codificado =xor (vector_numero_binario,c(i,:))
    numero_salida = xor (numero_codificado,c(i,:))

    matriz_mensaje(i,:)=(vector_numero_binario)
    matriz_codificada(i,:)=(numero_codificado)
    matriz_salida(i,:)=(numero_salida)
end
```

The process of that is:



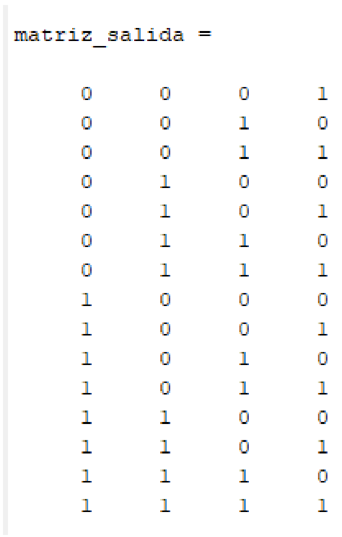
```
matriz_mensaje =

     0     0     0     1
     0     0     1     0
     0     0     1     1
     0     1     0     0
     0     1     0     1
     0     1     1     0
     0     1     1     1
     1     0     0     0
     1     0     0     1
     1     0     1     0
     1     0     1     1
     1     1     0     0
     1     1     0     1
     1     1     1     0
     1     1     1     1
```

Matrix 1.1 The message that we want to transmit (numbers)



```
matriz_codificada =

     0     0     1     1
     1     0     1     1
     0     1     1     1
     1     1     1     0
     1     0     0     0
     1     0     0     0
     0     0     0     0
     1     0     1     1
     0     0     0     0
     1     1     1     0
     0     0     0     1
     0     0     0     1
     0     0     1     1
     1     0     0     1
     1     1     0     0
```

Matrix 1.2. Encrypted message



```
matriz_salida =

     0     0     0     1
     0     0     1     0
     0     0     1     1
     0     1     0     0
     0     1     0     1
     0     1     1     0
     0     1     1     1
     1     0     0     0
     1     0     0     1
     1     0     1     0
     1     0     1     1
     1     1     0     0
     1     1     0     1
     1     1     1     0
     1     1     1     1
```

Matrix 1.3 Received message at the output of the decryption

In this result it is shows that we are able to encrypt and decrypt without errors and we see that we recuperate the clear text.

8- CONCLUSION

A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state.

Most common applications for LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

LFSRs have long been used as pseudo-random number generators for use in stream ciphers, due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed output streams.