MethodsX 12 (2024) 102602

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/methodsx

Sequential logic circuit gold codes for electronics and communication technologies

Aakanksha Devrari[a], Adesh Kumar[a,*], Piyush Kuchhal[a], Zoltán Illés[b,*], Chaman Verma[b,*]

[a] Department of Electrical & Electronics Engineering, School of Advanced Engineering, UPES, Dehradun, India
[b] Department of Media & Educational Informatics, Faculty of Informatics, Eötvös Loránd, University Budapest, Hungary, 1053

ARTICLE INFO

Method name:
Gold Codes Hardware Chip using LFSR

Keywords:
Communication technology
Seeding LFSR
Xilinx ISIM
FPGA synthesis

ABSTRACT

The linear feedback shift register (LFSR) and Gold codes are used in telecommunications, Global Positioning System (GPS), satellite navigation, wireless systems, and code division multiple access (CDMA) dependent channel schemes for numerous radio communication technologies Gold codes are distinguished by their capacity to provide various orthogonal sequences.

- The objective of the article is to focus on the design and simulation of the LFSR-based gold code generator chip in Xilinx ISE 14.7 software with the logic synthesis in Virtex-5 field programmable gate array (FPGA) and check the switching behavior with large frequency support applicable in fast-switching optical, and wireless electronics systems.
- The methodology comprises design, functional simulation with different test inputs, and FPGA synthesis. The chip design is verified for the 10-bit seeding sequence of LFSRs to result in 1023-bit code with the frequency support of 310 MHz, and 9.285 ns delay.
- The chip design is simulated based on seed data and different tap points of LFSR registers from which the bits are considered to generate the feedback. The design is scalable and has greater potential to extend to a larger extent. The behavior of the gold code depends on the maximum length sequence, absolute cross-correlation, and size of LFSR.

Specifications table

Subject area:	Engineering
More specific subject area:	Electronics and Communication
Name of your method:	Gold Codes Hardware Chip using LFSR
Name and reference of the original method:	Generation and implementation of IRNSS standard positioning signal. *Engineering Science and Technology, an international journal, 19*(3), 1381–1389. Generation of pseudorandom binary sequences by means of linear feedback shift registers (LFSRs) with dynamic feedback. *Mathematical and Computer Modelling, 57*(11–12), 2596–2604.
Resource availability:	Not Applicable

* Corresponding authors.
E-mail addresses: adeshmanav@gmail.com (A. Kumar), illes@inf.elte.hu (Z. Illés), chaman@inf.elte.hu (C. Verma).

https://doi.org/10.1016/j.mex.2024.102602
Received 24 July 2023; Accepted 2 February 2024
Available online 13 February 2024
2215-0161/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Introduction

Electronic communication systems with high-speed requirements are emerging rapidly. The high-speed networks require optical switches and processors to connect the fast-switching electronic systems with the optical data bus at ultra-high speed. The data encryption and decryption in cryptography and digital networks require the use of optical logic, and switching that provides high frequency, larger throughput, and minimum latency. Simple duplication of electrical encryption circuits is not possible. CDMA with direct sequence spread spectrum (DSSS) is the most significant wireless communication technology. CDMA networks [1] have approximately ten times the capacity of analog networks and significantly more GSM, time division multiple access (TDMA) networks. CDMA provides carriers and users with several other benefits, including higher-quality audio and video transmission, expanded coverage, and enhanced security [2].

A Gold code generally known to be a gold binary sequence is used in GPS satellite radio navigation and telecommunication technologies, such as optical, wire, radio, and electromagnetic systems. Robert Gold has given this name to the Gold codes [3,4]. When numerous devices are broadcasting in the same frequency band, Gold codes produce a relatively small cross-correlation [5] proving to be advantageous. There are $(2^n + 1)$ sequences in a set of Gold codes, which exist for a period of $(2^n - 1)$. LFSR plays a major role in the generation of maximum length based on the 'n' number of bits in the register. The two maximum length sequences are grouped with the $(2^n + 1)$ XOR gates, which constitute some set of $(2^n + 1)$ gold code sequences. Linear feedback shift registers (LFSR) [6,7] consist of 'n' successive 2-state memory blocks or flip-flops (FFs) that are synchronized simultaneously by a clock. Every memory unit changes states simultaneously with the input clock pulses, and each clock pulse shifts the data from one memory block to the next. If there is no signal in the first delay element, throughout this operation, all memory units will be empty at the end of 'm' shifts. A feedback loop is required to transform the shift register's content from a delay unit to a sequence generator [8]. The feedback loop provides the contents of a particular memory unit, referred to as taps of the XOR operation [9] based on the modulo 2 adder operation. In this way, a new term is calculated based on some of the previous 'm' terms as input.

In the field of cryptography and encoding, ultrafast communications [10,11] present a new set of obstacles. Because of their capacity to function at extremely high speeds, optical logic gates may be appealing components for these applications. The current bitwise logic record speed of 40 Gb/s [12] could be extended to bitwise logic rates of hundreds of Gb/s by careful design of the nonlinear optical switching elements. However, due to issues in synchronization and fanout, high-speed optical computers will be limited to applications requiring low logic gate counts for the predictable future. Using optical logic to create a linear feedback shift register would limit the system to 40 identical taps. Thus, making the shift register extremely exposed to a sparse matrix.

The most often used technique for raising the rate of generating pseudo-noise (PN) sequences is to multiplex several lower rates of PN sequences together. Multiplexing allows the system to produce PN sequences at any desired rate, but it is important to consider any needs that go beyond. the rate requirement [13]. The PN sequences are employed in a variety of applications that need data generation based on their random behaviour. A binary PN sequence and a binary data stream are XORed together to encrypt [14] the data stream. The resulting bitstream will conceal the original plaintext from an observer who is not familiar with the PN sequence. On the other hand, the plaintext can be extracted by XORing the encrypted stream with someone who knows the PN sequence. PN sequences are employed to encode data, for example, in spread-spectrum applications when white-noise signals are desired for transmission. LFSR is one of the simplest methods for creating PN sequences [15]. At each clock cycle in an LFSR, the bit in the register's last cell is retrieved for output, all remaining bits are moved down one cell, and a new bit is stored in the first cell. The new bit is computed from the bits in the FSR using a function that is linear in the case of LFSRs arbitrarily chosen in the case of LFSRs. Each cell containing the data utilized to generate the new bit is said to have a tap. The contents of the cells obtained via taps are combined according to a feedback function, referred to as the characteristic polynomial in D-transform notation. There are various difficulties associated with rapidly creating PN sequences.

Method detail

The LFSR is a shift register in which some of its outputs are grouped in XOR configurations to create a feedback channel [16]. LFSRs are widely utilized as pseudorandom pattern generators to produce a random assortment of '1' and '0'.

High-toggle-rate configurations are generated while in LFSR test mode are excellent for producing high-fault coverage in communication devices. LFSRs are easy to generate and use in multiple ways based on the system's needs and memory [17]. The logic is checked using a behavioral model from the design perspective. The most common category of an LFSR is shown in Fig. 1. The 8-bit LFSR is a shift register having feedback from two or more nodes or taps, in the register chain. In this case, the taps are at the position of bit-1, and bit-7, which is written as (1,7). The All-register elements have the same input for the clock, which is left out of the symbol to make it easier to understand. The DFF0, DFF1, DFF3, DFF4, DFF5, DFF6, and DFF7 denote the 8-bit data processing using D flip-flop respectively based on the synchronized clock signal. The LFSR out is configured based on the XOR or XNOR operation of tap bits. The rest of the bits operate normally like a shift register. The feedback function and the taps selection decide the order of the LFSR value. Fig. 2 depicts two 4-bit XOR-based LFSRs having different feedback taps. The first LFSR has feedback taps at (1,3) and the second LFSR has feedback taps at (2,3). Initially, both LFSRs start with the same value i.e. (0111), later when the clock pulses are added, the sequences start to change quickly because their taps are different. An LFSR will cycle throughout a loop with only a few values if certain conditions are met. The maximally lengthened LFSRs traverse every possible value (except from all zeros) and then return to their initial values.

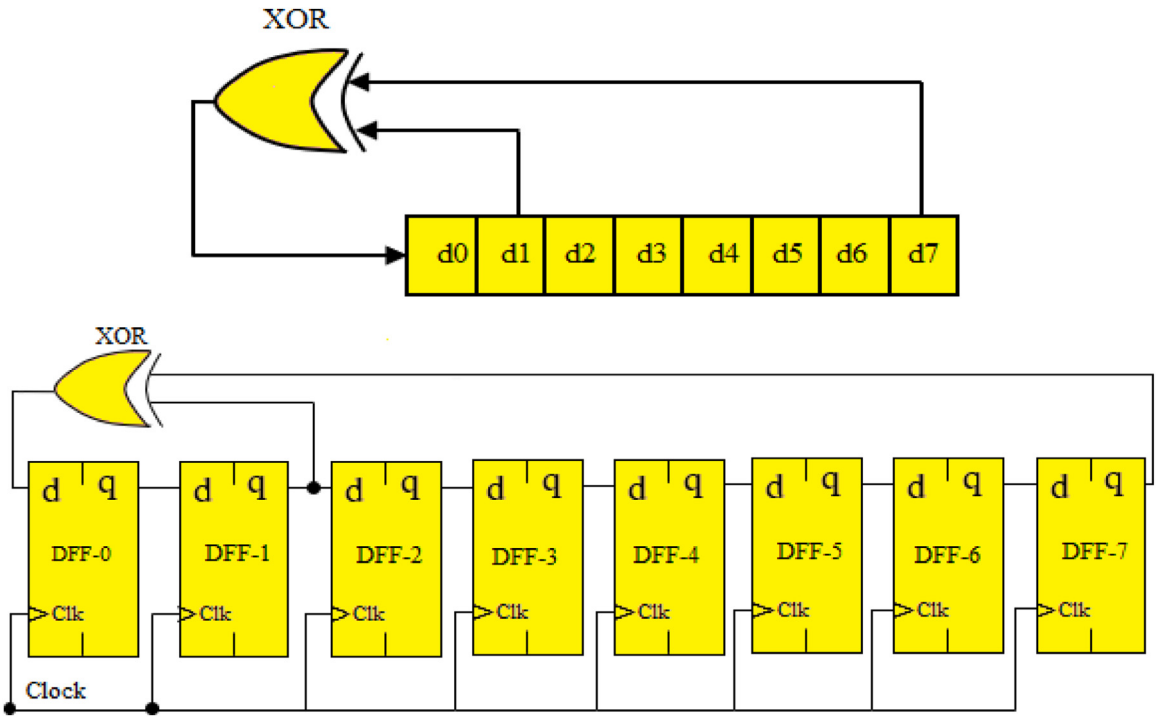


Fig. 1. The circuit diagram and symbol of LFSR (8-bit).

Seeding an LFSR

A strange thing about an XOR-based LFSR is that if it reaches the all-zero value, it will keep shifting all-zero values for as long as it wants. The fact that XORs are utilized as the sum constituent of the easiest form of the half adder allows us to see the XOR function as a component of the digital addition operation. The shift registers subsequent bits, and the overall contents are compared using XOR logic to control the input bit of an LFSR. Similarly, an XNOR-based LFSR will keep shifting all the '1' values. Each register bit starts with either a logic '0' or a logic '1', so the prohibited value wakes up the LFSR. Therefore, the seed value is to be put at the beginning of an LFSR [18]. Registers with reset or set inputs are used to load a seed value. Some reset inputs and set inputs of the registers coupled to the same control signal. When the control signal turns on, the LFSR is loaded with a seed value that is hard-wired into it. In some situations, it proves to help change the seed value. This is done by adding a multiplexer to the input of the LFSR as shown in Fig. 3.

When the data input is chosen, the multiplexer acts like a normal shift register, and it loads any seed value. Once a seed value is loaded, feedback paths are chosen, and the device goes back to the LFSR mode. Using the strange order of values in an LFSR, data is encrypted and decrypted. Data encryption is achieved by XORing the bit stream with the LFSR's output, as shown in Fig. 4. A stream of encrypted data is decrypted by XORing it with the result of a comparable LFSR. This method of encryption is very simple yet not very safe, but more cost-efficient and may be useful in some situations. It is applicable in secured communication such as wireless data communication and telecommunication switching.

Gold code sequence

The data generated by the two LFSRs is combined using Modulo 2 additions to generate a set of small correlation PN codes. This process produces a sequence of gold codes that have a correlation behavior [19]. A gold code sequence is obtained by multiplying the two sequential PN sequences. It has the advantage of being consistent and having a low cross-correlation between two codes. The chosen pair of gold code sequences are generated by the temporary shift of one PN sequence. Minimum cross-correlation analysis is performed on the chosen pair of the gold code sequences, having the lowest cross-correlation. Cross-correlation is helpful for Gold code study to check the similarity between two coding sequences. Let us consider

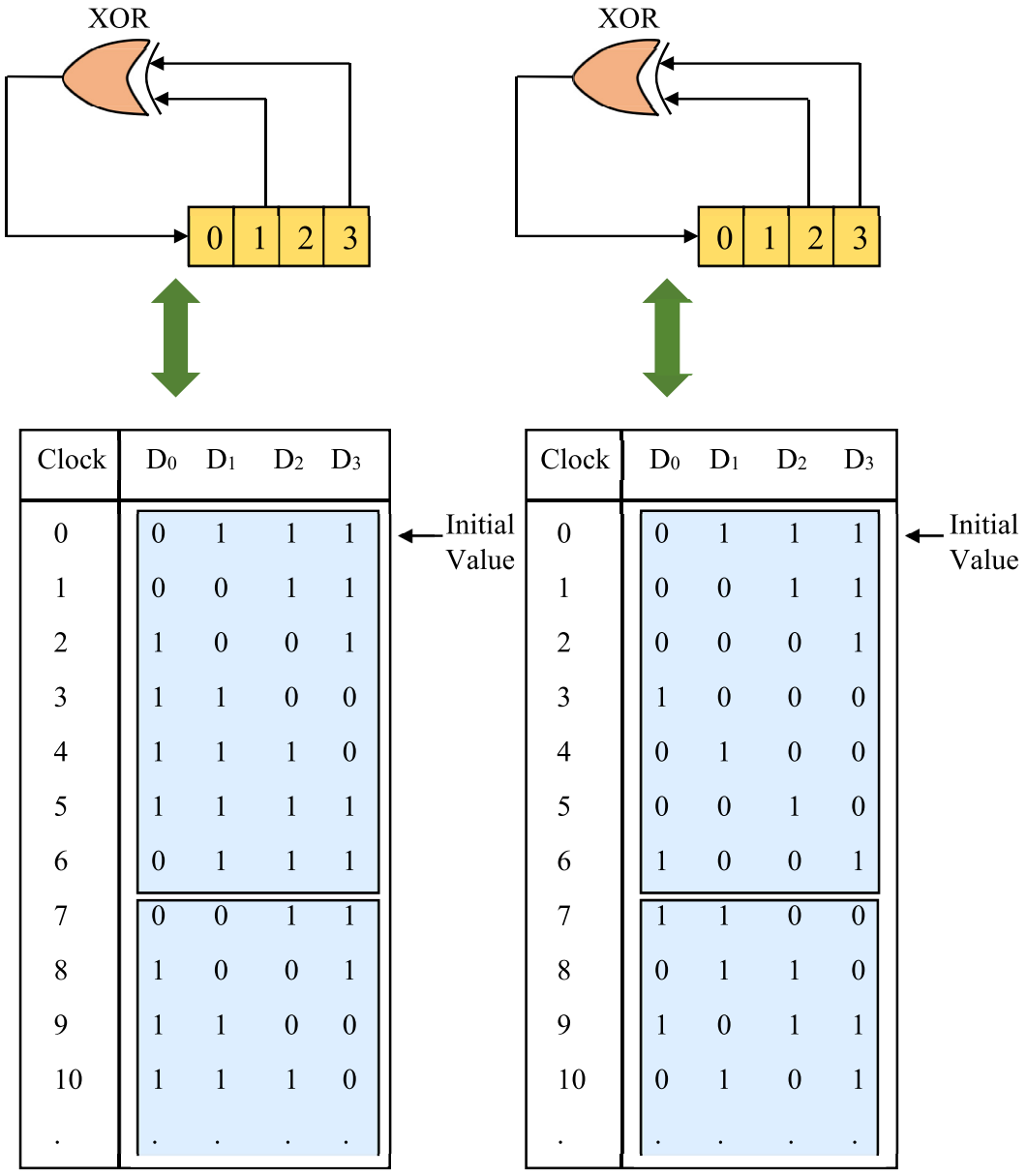
M = Number of the input bits of the Gold generator

L= Length of the sequences generated for the input Gold vector **G**

The sequence number and sampled and quantized at the Q^{th} symbol.

The resultant Gold vector **G' = G [Q]**

The concept of common factor is applied to estimate the maximum performance using a generated common divisor (GCD), Then GCD (M, Q) = 1. Where M = Message input bits and Q and quantized data bits. The sequence is divided into even and odd sequences,



Clock	D_0	D_1	D_2	D_3
0	0	1	1	1
1	0	0	1	1
2	1	0	0	1
3	1	1	0	0
4	1	1	1	0
5	1	1	1	1
6	0	1	1	1
7	0	0	1	1
8	1	0	0	1
9	1	1	0	0
10	1	1	1	0
.	.	.	.	.

Clock	D_0	D_1	D_2	D_3
0	0	1	1	1
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	0	1	0	0
5	0	0	1	0
6	1	0	0	1
7	1	1	0	0
8	0	1	1	0
9	1	0	1	1
10	0	1	0	1
.	.	.	.	.

Fig. 2. Different tap selections and their comparison.

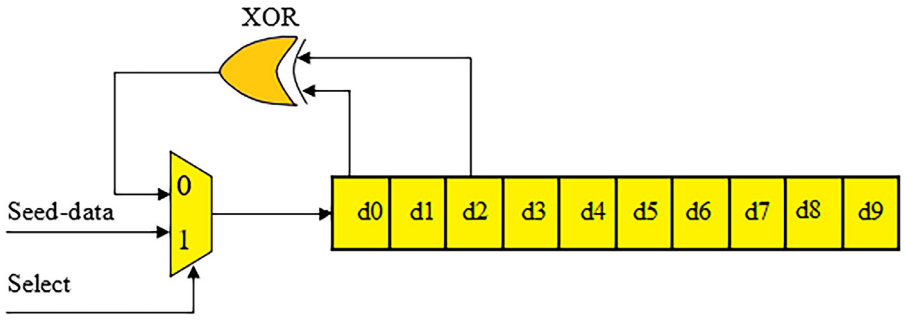


Fig. 3. Seed values loading for a 10-bit LFSR.

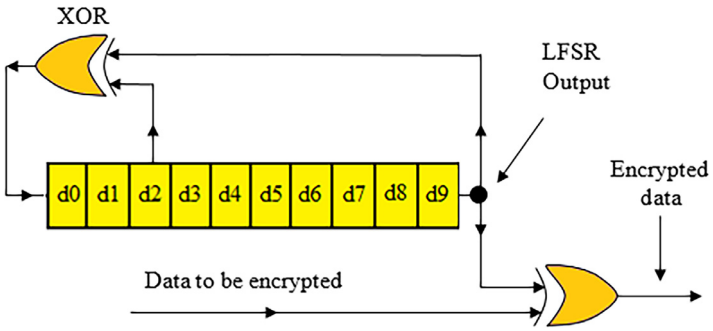


Fig. 4. Encryption of data using 10-bit LFSR.

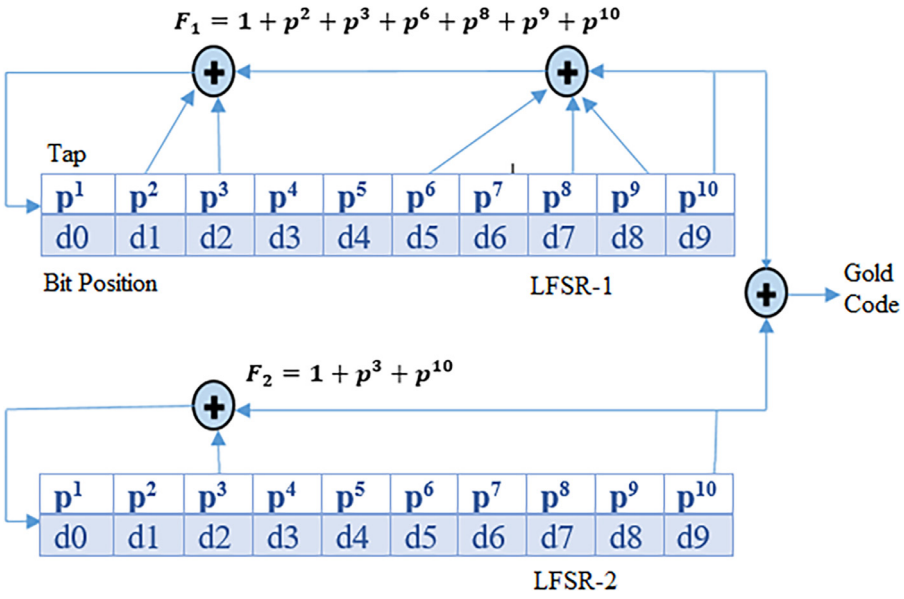


Fig. 5. Gold code generator.

Then the correlation is established using the following equations [20,21].

$$|R_c| \le 2^{\left(\frac{M+1}{2}\right)} + 1 \; for\ odd\ sequences \text{ GCD (M, Q)} = 1$$

$$|R_c| \le 2^{\left(\frac{M+2}{2}\right)} + 2 \; for\ even\ sequences \text{ GCD (M, Q)} = 2$$

The following even sequence and fixed and quantized even samples follow a sampled sequence. For example, sampled sequence 0th, 4th, 8th, 12th, 16th· …. etc. are considered as M mod 4 ≠0, and other values as M mod 4 = 2 is applicable to establish cross-correlation using LFSR shift register. These sequences act as a program component and begin with a fill or seeding of the bits. The LFSR algorithm is used to generate m-Sequences, which are then used as program components. The PN sequence generators will produce two signals against the first and second inputs and two sequences are used to get the final output. Gold-Code Sequence is generated by XORing the previous step outputs. Table 1 lists the LFSR maximal-length sequence with 10-bit input data "1101100110

Gold sequences aid in the generation of additional sequences from a pair of m-sequences, providing a much broader range of sequences to many users. Only one sequence of sufficient length emerged from the m-sequences. Combining two of these sequences with the two m sequences themselves yields sequences. Modulo-2 is used to produce gold codes that show strong cross-correlation behaviour [22]. Gold codes are generated by modulo-2 by adding the desired pair of sequences' specific relative phases (maximally length sequences). In terms of operation. Fig. 5 depicts a schematic representation of a Gold code generator with n =10. Linear feedback shift registers are used in conjunction with a shift register for each of the two m-sequences (LFSR). The diagram above depicts a Gold code sequence processing using a shift register with the length 10-bit, capable of generating output stages ($2^n - 1$). The shifting of the input data sequence with feedback taps (2, 3, 6, 8, 10) and feedback taps (3, 10) is listed in Tables 2 and 3, and the corresponding output is in Table 4. The generated polynomial for LFSR-1 and LFSR-2 are given as

$$F_1 = 1 + p^2 + p^3 + p^6 + p^8 + p^9 + p^{10} \quad (1)$$

$$F_2 = 1 + p^3 + p^{10} \quad (2)$$

Table 1
LFSR maximal length sequence with 10-bit input data "1101100110".

State	Data with Tap (3, 10) for LFSR-1	Output LFSR-1	Data with Tap (2,3, 6,8,9,10) LFSR -2	Output LFSR-2	Gold Sequence
0	1101100110	0	1101100110	0	0
1	0110110011	1	0110110011	1	0
2	1010011001	1	1000010010	0	1
3	1100001100	0	0100001001	1	1
4	0110000110	0	1001001111	1	1
5	0011000011	1	1111101100	0	1
6	1000100001	1	0111110110	1	0
7	1101010000	0	0011111011	1	1
8	0110101000	0	1010110110	0	0
9	0011010100	0	0101011011	1	1
10	0001101010	0	1001100110	0	0
11	0000110101	1	0100110011	1	0
12	1001011010	0	1001010010	0	0
13	0100101101	1	0100101001	1	0
14	1011010110	0	1001011111	1	1
15	0101101011	1	1111100100	0	1
16	1011110101	1	0111110010	0	1
17	1100111010	0	0011111001	1	1
18	0110011101	1	1010110111	1	0
19	1010001110	0	1110010000	0	0
20	0101000111	1	0111001000	0	1
21	1011100011	1	0011100100	0	1
22	1100110001	1	0001110010	0	1
23	1111011000	0	0000111001	1	1
24	0111101100	0	1011010111	1	1
25	0011110110	0	1110100000	0	0
26	0001111011	1	0111010000	0	1
27	1001111101	1	0011101000	0	1
28	1101111110	0	0001110100	0	0
29	0110111111	1	0000111010	0	1
30	1010011111	1	0000011101	1	0
31	1100001111	1	1011000101	1	0
32	1111000111	1	1110101001	1	0
.	.	.	.	.	.
.	.	.	.	.	.
1022	1001001101	1	1101011011	1	0
1023	1101100110	0	1101100110	0	0

Table 2
LFSR maximum-length sequence with (2, 3, 6, 8, 9, 10) feedback taps.

	State change in the shift register										Output Symbol
	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	
Input Sequence	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	
Feedback											
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	1	0	1	1	0	0	1	1
0	0	1	1	1	1	0	1	1	0	0	0
1	1	0	1	1	1	1	0	1	1	0	0
0	0	1	0	1	1	1	1	0	1	1	1
0	0	0	1	0	1	1	1	1	0	1	1
0	0	0	0	1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1	1	1	1	1
1	1	1	0	0	0	1	0	1	1	1	1
1	1	1	1	0	0	0	1	0	1	1	1
0	0	1	1	1	0	0	0	1	0	1	1
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
1023	1	1	0	1	1	0	0	1	1	0	0

Coded sequence: 11001101111….0

Table 3
LFSR maximum-length sequence with (3, 10) feedback taps.

	State change in the shift register										Output Symbol
	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	
Input Sequence	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	
Feedback											
0	0	1	1	0	1	1	0	0	1	1	**1**
0	0	0	1	1	0	1	1	0	0	1	**1**
0	0	0	0	1	1	0	1	1	0	0	**0**
0	0	0	0	0	1	1	0	1	1	0	**0**
0	0	0	0	0	0	1	1	0	1	1	**1**
1	1	0	0	0	0	0	1	1	0	1	**1**
1	1	1	0	0	0	0	0	1	1	0	**0**
0	0	1	1	0	0	0	0	0	1	1	**1**
0	0	0	1	1	0	0	0	0	0	1	**1**
0	0	0	0	1	1	0	0	0	0	0	**0**
0	0	0	0	0	1	1	0	0	0	0	**0**
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
1023	1	1	0	1	1	0	0	1	1	0	**0**

Coded Sequence: 1100110110….0. Table 7 lists the Gold code output parameters estimations for the input data sequence 1101100110 using LFSR 1 with feedback taps (2, 3, 6, 8, 9, 10) and LFSR 2 with feedback taps (3, 10).

Table 4
Gold sequence output.

Stages	LFSR-1 Feedback taps (2, 3, 6, 8, 9, 10)	Output	LFSR-2 Feedback taps (3, 10)	Output	Gold Code output
1	1101100110	1	1101100110	1	0
2		1		1	0
3		0		0	0
4		0		0	0
5		1		1	0
6		1		1	0
7		0		0	0
8		1		1	0
9		1		1	0
10		1		0	1
.		.		.	.
.		.		.	.
1023		0		0	0

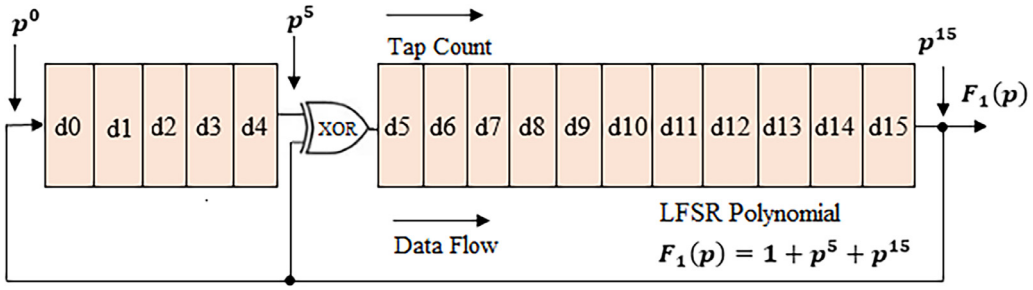


Fig. 6. Galois Implementation.

LFSR implementation

The two types of LFSR implementation are Galois implementation and Fibonacci implementation.

Implementation of galois

Fig. 6 depicts the data flow from the left side to the right side, whereas the feedback path is displayed from the right side to the left side [23]. The polynomials acquire values from the left side to the right side, starting with the p^0 in MSB and p^{16} in LSB. In mathematics, this polynomial is known as a tap polynomial, and it identifies which taps from the shift register should always be

Table 5
Test simulation output for LFSRs and Gold code based on Galois implementation.

Taps	Output (1023-bit)
LFSR1 input = 1101100110 Taps (3, 10) = 0010000001	0110011 00001010 11010111 00011011 11110001 00011110 01111011 01101000 00001010 00010110 10101000 11111011 11001001 01100000 10011001 00010100 01101101 11000000 11110001 11011111 11001000 01100010 11011101 00001101 01011001 11100101 10110010 00001000 10010011 00000010 11000101 00111011 00111000 10111111 01010001 01110110 10110000 11001101 10101000 00111010 01111010 01101010 01001110 00001111 10011100 11011110 10001010 10110111 11000010 01110100 01110101 11110110 10010000 10000101 00101011 00011100 11111110 11000010 00110100 11100100 11110000 11011101 10001100 01111011 11101001 00101000 00011010 00110010 11101001 01101000 10001011 00110100 10100100 01100001 11011011 11000001 01110010 10111001 11011101 11001100 11101010 11101111 01100101 00010011 01100010 00011100 10111110 01010011 00110010 10101001 11111001 10001101 01111001 10101101 00110001 00101110 00010111 10101010 10111111 11010000 01010100 10111100 01010111 10111010 10011011 10010001 11000111 11111110 00000011 10000111 11101110 00100111 11000110 01111101 01100101 10010010 01000000 00010000 00100100 01000001 10010011 01000010 01010100 00111101 01110101 10110110 00000001 10000011 (binary) =330AD71BF11E7B680A16A8FBC96099146DC0F1DFC862DD0D59E5B2089302C53B38BF5176B0CDA83A7A6A4E0F9CDE8AB 7C27475F690852B1CFEC234E4F0DD8C7BE9281A32E9688B34A461DBC172B9DDCCEAEF6513621CBE5332A9F98D79AD312E 17AABFD054BC57BA9B91C7FE0387EE27C67D659240102441934254 3D75B60183 (hexadecimal)
LFSR2 input = 1101100110 Taps (2,3,6,8,9,10) = 0110010111	0101100 10101011 00110000 11000001 11100110 11111100 01001001 11000110 10101101 01100011 11011010 11110011 10101110 01001010 01010111 01111001 00000111 01001011 10011111 10010011 00110101 00000011 00101001 10001101 10001010 10101001 00000001 11001000 11101010 00110010 01001000 10110110 00011110 10111111 10101010 00101000 01000101 01100000 10011011 01001001 01111110 11110100 10001101 11100010 10011110 01011011 11011111 00110001 01100001 00111011 10010100 00010110 10101011 11100000 10101111 01010010 11010011 11111111 01100000 01001011 00100111 11001010 11011011 00111011 01000100 01111000 00011111 11001111 00011001 11110100 00110101 10111011 01110000 01100011 10110010 11000100 11110100 01011101 10001100 00101010 01110100 01101001 10010111 10000111 01111111 10000100 00111110 00111110 11101110 10000000 00110100 00011011 10101101 00001011 11101101 10101001 10111001 10010001 00000100 00001010 00100101 01000011 10001011 11011001 10110010 00010100 10011010 11101001 10100011 10011100 11010010 10001111 11010011 10010111 01010111 00010001 00110000 00010001 10001000 01001000 01100110 01110000 00001011 10000101 10011110 11100011 10000110 11011111 (binary) =2CAB30C1E6FC49C6AD63DAF3AE4A5779074B9F933503298D8AA901C8EA3248B61EBFAA2845609B497EF48DE29E5BDF3 1613B9416ABE0AF52D3FF604B27CADB3B44781FCF19F435BB7063B2C4F45D8C2A746997877F843E3EEE80341BAD0BEDA9 B991040A25438BD9B2149AE9A39CD28FD3975711301188486 6700B859EE386DF (hexadecimal)
Gold code output	0011111 10100001 11100111 11011010 00010111 11100010 00110010 10101110 10100111 01110101 01110010 00001000 01100111 00101010 11001110 01101101 01101010 10001011 01101110 01001100 11111101 01100001 11110100 10000000 11010011 01001100 10110011 11000000 01111001 00110000 10001101 10001101 00100110 00000000 11111011 01011110 11110101 10101101 00110011 01110011 00000100 10011110 11000011 11101101 00000010 10000101 01010101 10000110 10100011 01001111 11100001 11100000 00111011 01100101 10000100 01001110 00101101 00111101 01010100 10101111 11010111 00010111 01010111 01000000 10101101 01010000 00000101 11111101 11110000 10011100 10111110 10001111 11010100 00000010 01101001 00000101 10000110 11100100 01010001 11100110 10011110 10000110 11110010 10010100 00011101 10011000 10000000 01101101 11011100 00101001 11001101 10010110 11010100 10100110 11011100 10000111 10101110 00111011 10111011 11011010 01110001 11111111 11011100 01100011 00101001 10000101 01011101 00010111 10100000 00011011 00111100 10101000 00010101 11101010 00110010 10000011 01110000 00000001 10101100 00001001 11110101 00110010 01011111 10111000 11101011 01010101 10000111 01011100 =1FA1E7DA17E232AEA7757208672ACE6D6A8B6E4CFD61F480D34CB3C079308D8D2600FB5EF5AD3373049EC3ED0285558 6A34FE1E03B65844E2D3D54AFD7175740AD5005FDF09CBE8FD402690586E451E69E86F2941D98806DDC29CD96D4A6DC8 7AE3BBBDA71FFDC6329855D17A01B3CA815EA32837001AC09F5325FB8EB55875C (hexadecimal)

brought back to the computer. The generator LFSR polynomial is $\boldsymbol{F}_1(\boldsymbol{p}) = 1 + \boldsymbol{p}^5 + \boldsymbol{p}^{15}$**.** The Galois implementation is referred to as modular type, M-type, or in-line LFSR implementation because the XOR gate is in the path of the shift register.

Fibonacci implementation

The data flow in Fibonacci implementation occurs from the right side to the left side as depicted in Fig. 7. In the Fibonacci implementation the generated polynomial decrements from the left side to the right side [24,25], p^0 is the final term of the LFSR polynomial. p^0 is the MSB p^{15} is in LSB, p^0 is known as the reverse tap polynomial, and the associated feedback taps are labeled sequentially from the right to the left side, indicating the direction of the shift registers [26]. The generator LFSR polynomial is $\boldsymbol{F}_2(\boldsymbol{p}) = \boldsymbol{p}^{15} + \boldsymbol{p}^5 + 1$**.** Since the XOR gate is located in the feedback route, the Fibonacci implementation is also referred to as a simple type (S-type) or out-of-line LFSR implementation.

Results and discussions

Xilinx ISE 14.7 is used to simulate the LFSR module-1, LFSR module-2, and the gold code sequence generator. The description of the input and output pins of the design is shown using Fig. 8, which presents the register transfer level (RTL) [27,28] of the designed hardware chip. Table 8 contains information on the pins direction utilized in the design. The architecture uses the tap sequence to process and regulate the 10-bit input sequence. For LFSR-1 and LFSR-2, the tap bit locations are (3, 10) and (2, 3, 6, 8, 9, 10), respectively.

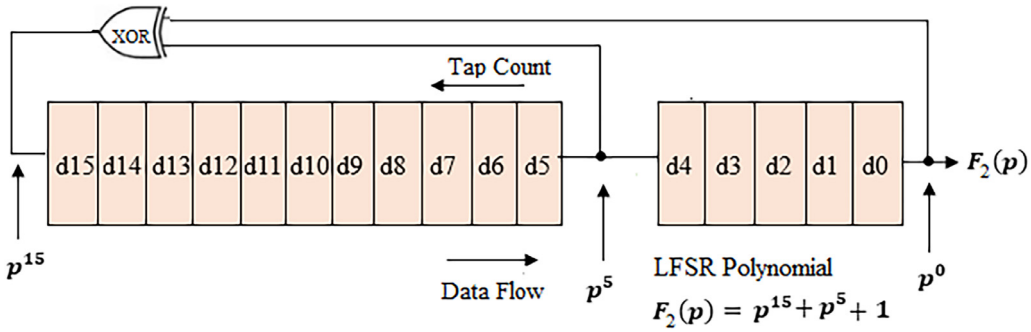


Fig. 7. Fibonacci Implementation.

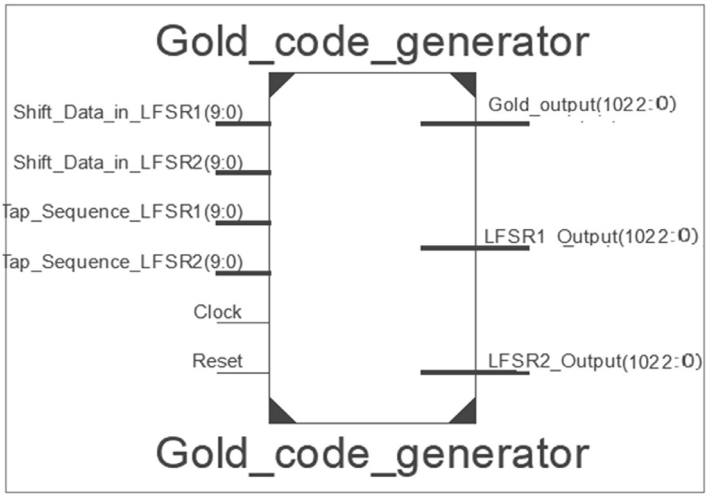


Fig. 8. Gold code sequence generator RTL description with 10-bit LFSR inputs/outputs.

- The LFSR module-1 input is Shift_data_in_LFSR1(9:0) which is used to provide the sequence-1 input of the LFSR-1 module with the 10-bit input.
- The LFSR module-2 input is Shift_data_in_LFSR2(9:0) which is used to provide the sequence-2 input of the LFSR-2 module with the 10-bit input.
- Tap_Sequence_LFSR1(9:0) is the 10-bit tap input for the LFSR-1 chip module. The bit positions called taps are those that affect the next state. The output bit is transmitted back into the leftmost bit after each tap is successively XORed with it.
- Tap_Sequence_LFSR2(9:0) is the output bit that is delivered back into the leftmost bit after each tap is successively XORed, presenting the bit position for feedback for the LFSR-2 chip module.
- The clock (1-bit) is the input used by LFSR chip1 and LFSR chip-2 to operate on the clock signal's functioning rising edge. The design employs a 50% duty cycle.
- Reset (1-bit) presents the input used for the initial contents of the LFSR as zero during reset.
- LFSR_out1 (1022:0) denotes the 1023-bit simulated output result of the LFSR chip-1 module.
- LFSR_out2 (1022:0) denotes the 1023-bit simulated output result of the LFSR chip-2 module.
- Gold_out (1022:0) is the integrated output of the Gold sequence output module, which is obtained from the output of the 1023-bit XOR operation.

Figs. 9 and 10 depict the simulations of the LFSR module-1 and LFSR module-2, respectively. Fig. 11 illustrates the output waveform of the gold code simulation. The simulation waveform is taken from the Xilinx ISIM simulator [29,30]. The test inputs are given below. The software and hardware requirements used in the simulation: 64-bit Window-10, DDR4 memory 8 GB, 1.00 GHz speed, Intel Core i5 8th Gen to run the test cases in Xilinx ISE 14.7 software.

The input sequence for LFSR 1 is given as Shift_data_in_LFSR1 (9:0) = "1101100110", Tap inputs (3, 10) as Tap Sequence_LFSR1 (9:0) = "0010000001", The output presents a 1023-bit sequence for easy understanding in hexadecimal form. The input sequence of LFSR2 is given as Shift_data_in_LFSR2 (9:0) = "1101100110", Tap inputs (2,3,6,8,9,10) as Tap Sequence_LFSR2 (9:0) = "0110010111", The output of LFSR2 is presented in 1023-bit sequence for easy understanding in hexadecimal form. The output of the LFSR1 and LFSR2 data stream after 1023 iterations and generated Gold code is given in Table 5. The chip design is verified for other test inputs as

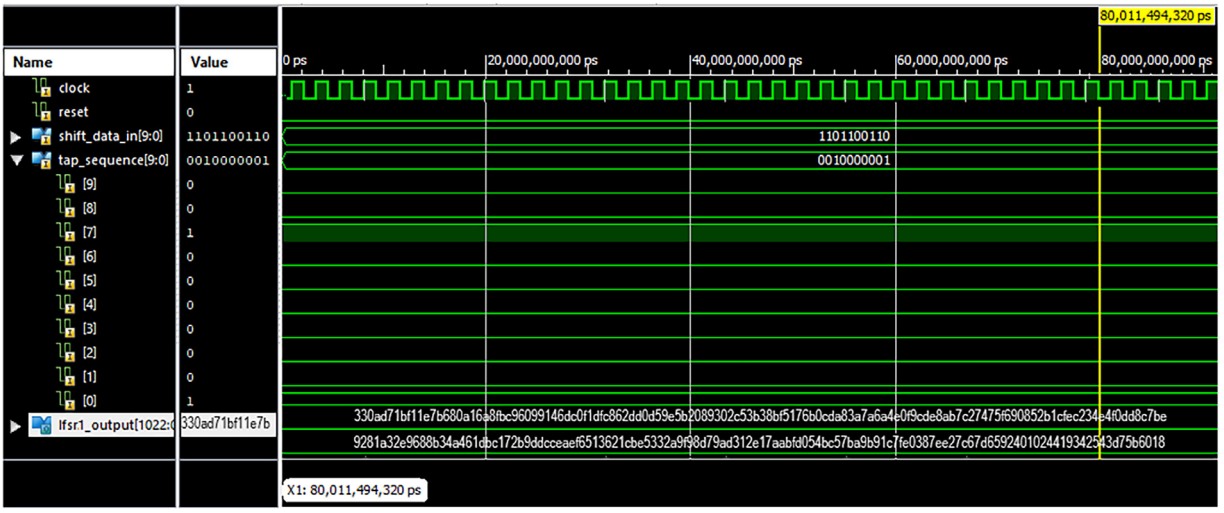


Fig. 9. Output simulation waveform of LFSR-1 module with tap (3, 10).

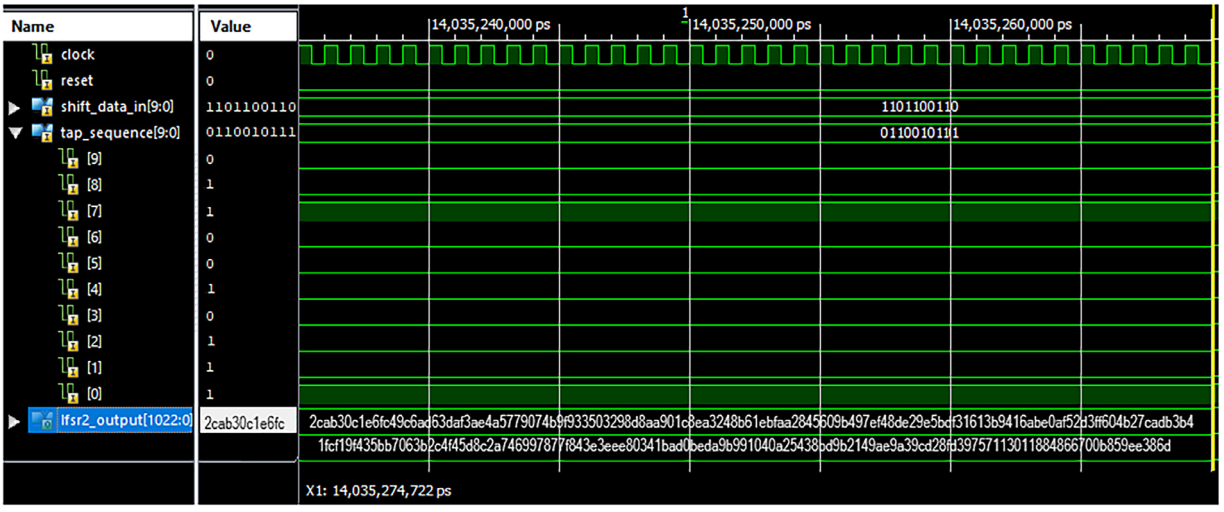


Fig. 10. Output simulation waveform of LFSR-2 module with tap (2, 3, 6, 8, 9, 10).

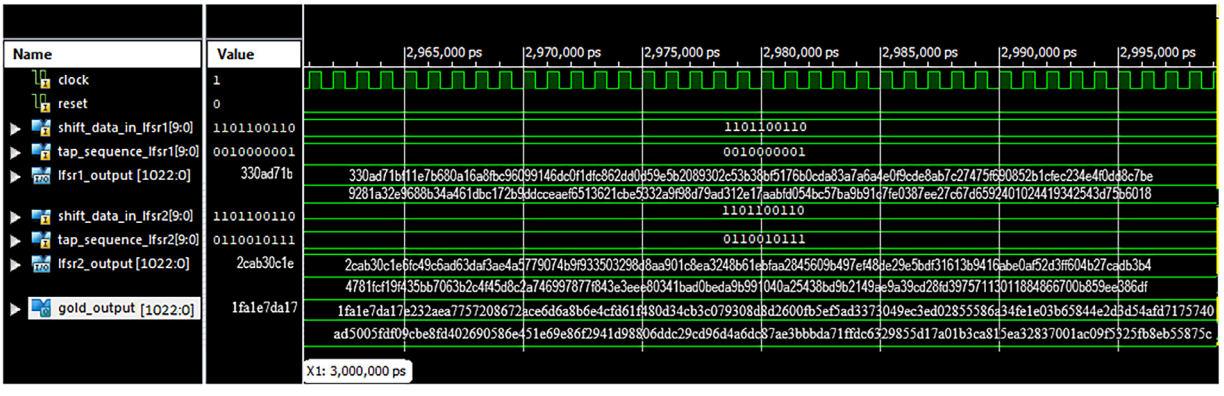


Fig. 11. Simulation waveform of Gold sequence generated output.

Table 6
Test cases for the simulation of Gold code.

Test	Taps	Output (Hexadecimal)
Test-1	LFSR1 input = 1001100101 Taps (2, 4, 6) = 0101010000	574B058A6713A3FB5D2C16299C4E8FED74B058A6713A3FB5D2C16299C4E8FED74B05 8A6713A3FB5D2C16299C4E8FED74B058A6713A3FB5D2C16299C4E8FED74B058A6713A3F B5D2C16299C4E8FED74B058A6713A3FB5D2C16299C4E8FED74B058A6713A3FB5D2C162 99C4E8FED74B058A6713A3FB5D2C16299C4E8FED74B058A
	LFSR2 input = 1001100101 Taps (2, 3, 4, 6,9) = 0111010010	67A6A359758984DCB21DF12E47D2BC68F539AF614C5D948310D231FBC13842A503A0F98 C05815DEFF2032D6AB8046C99A00F6E227FEB6878AA9B60A199DB7E7DDD24A8AD2C719 E6C6F42289094F861C1CEA082FD167E06AC3757F66FB4CAB76A4EE64B671688E48BC9E17 1E51D7CBD73D351ACBAC4C26E590EF89723E95E347A
	Gold code output	30EDA6D3129A2727EF31E707DB9C33858189F7C73D67AB36C213536205D0BC7248A573E B1622A6B2DE1504F6F68B81ED1057C85345D4DDAA6BF9F9657125A936D8AECFBE8F8AC 340796BB4DE1B1515719606F3C729CBD46D55CE32874A1DFC6FC48158C4EDD5C89DE07D 3F6AF0080D2B752096857110AF3B973C7FDD3E1531F0
Test-2	LFSR1 input = 1010101010 Taps (3, 5, 7) = 0010101000	2E4CF32757267993AB933CC9D5C99E64EAE4CF32757267993AB933CC9D5C99E64EAE4CF 32757267993AB933CC9D5C99E64EAE4CF32757267993AB933CC9D5C99E64EAE4CF3275 7267993AB933CC9D5C99E64EAE4CF32757267993AB933CC9D5C99E64EAE4CF327572679 93AB933CC9D5C99E64EAE4CF32757267993AB933CC9D
	LFSR2 input = 1010101010 Taps (2, 3, 7,10) = 0110001001	2F3D2D8C788740127DAC1CB47E579E96C63C43A0093ED60E5A3F2BCF4B631E21D0049F 6B072D1F95E7A5B18F10E8024FB583968FCAF3D2D8C788740127DAC1CB47E579E96C6 3C43A0093ED60E5A3F2BCF4B631E21D0049F6B072D1F95E7A5B18F10E8024FB583968FC AF3D2D8C788740127DAC1CB47E579E96C63C43A0093ED6
	Gold Output	171DEAB2FA13981D63F207DAB9E00F22CD88C927C4CB19760861803D63F87C79EAAD39 8207A39EC740E22B3D93DCBD1D1697240F886A0BF5EB2CD32EB479D52A1ABD7A59F449 31C790046F3D96A27756AD2DB06D2323C84D7EBEB406DB6C64468E8CE8AB7AB1E3FDA D6AE861F444E95DBE3C8F650B165EBE4A1A579193AF24B
Test-3	LFSR1 input = 1001011100 Taps (1, 2, 3, 9) = 1110000010	16152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C2A58 54B0A96152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C 2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96152C2A5854B0A96 152C2A5854B0A96152C2A5854B0A96152C2A5854B0A9
	LFSR2 input = 1001011100, Taps (1, 2, 5, 7,9) = 1000101010	15D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A8B6 04395D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A 8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27EF1A8B604395D27E F1A8B604395D27EF1A8B604395D27EF1A8B60439
	Gold Output	3C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2 B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752D BF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C752DBF0E2B4903C7 52DBF0E2B4903C752DBF0E2B4903C752DBF0E2B490
Test-4	LFSR1 input = 1111000011, Taps (1, 8, 9) = 1000000110	5A795895E4D47F5B22A120A04D9CE1D613BC2C11BD412D110BDF7B3472EA64E2BA422CC AA92514BF2DCA1FBB429A54FF006D8A321CD7130AB2479838180EDD17D37C5AF903DA CFC6C3C1AD4A44F46988E8658F86EE6638C31AB92E7D5A4F9D8CEABFF6DE7B82ECBC4 19B8F5DFA02016D3CAC4AF26A3FAD9150905026CE70EB09
	LFSR2 input = 1111000011, Taps (2, 3, 6, 7,9) = 0110011010	8D764C055F3C9F184306453258BD121F20D4EDBEEA4E489EDCFC599BF2EF117401C4B F5FDEC0B7AC661EE46BB2602AF9E4F8C218322992C5E890F906A76DF7527244F6E7E2CCD F97788BA00E25FAFEF605BD6330F7235D930157CF27C610C1914C962F4487C8353B6FBA 939227B73F1666FCBBC45D00712FD7F7B02DEB1987B
	Gold Output	52AE3C55B127B6AAA69144F3681730F7E1B162CA53E5C998E610BEADCDC495F5FA5E6 73F54C91FC5EBABF1FDF9BC56509E2206139FF58A26ECCE97A872780262F45815977DF6 023FB449174AA6ABC667883E59B5E1140D1A2AAC528F262E9195FE76942A33FE6FEFF76 02664D879729C5AC3814E2FEFAA83ADEF2B2410C17372

listed in Table 6. Test-1 has LFSR1 input = 1001100101with taps (2, 4, 6) = 0101010000, and LFSR2 input = 1001100101with taps (2, 3, 4, 6,9) = 0111010010. Test-2 has LFSR1 input = 1010101010 with taps (3, 5, 7) = 0010101000, and LFSR2 input = 1010101010 with taps (2, 3, 7,10) = 0110001001. Test-3 has LFSR1 input = 1001011100 with taps (1, 2, 3, 9) = 1110000010, and LFSR2 input = 1001011100, with taps (1, 2, 5, 7,9) = 1000101010. Test-4 has LFSR1 input = 1111000011 with taps (1, 8, 9) = 1000000110, and LFSR2 input = 1111000011, with taps (2, 3, 6, 7,9) = 0110011010.

The 10-bit test input for LFSR-1 is processed with the 10-bit tap sequence and produces the corresponding output-1 according to the defined case and the same is simulated. The 10-bit test input for LFSR-2 is processed with the 10-bit tap sequence and produces the corresponding output-2 according to the defined case and the same is simulated. Output-1 and output-2 are used to provide the claimed output of 1024-bit as listed in the binary Gold sequence output. The analysis of the chip design is done according to

Table 7
Results and parameters.

Parameters	Estimation
Delay (ns)	9.285
Power (mW)	15.190
Frequency (MHz)	310.00

Table 8
Input and output pins description.

Pins	Direction
Shift_data_in_LFSR1(9:0)	Input
Shift_data_in_LFSR2(9:0)	Input
Tap_Sequence_LFSR1(9:0)	Input
Tap_Sequence_LFSR2(9:0)	Input
clock (1-bit)	-Input
Reset (1-bit)	-Input
LFSR_out1 (1022:0)	Output
LFSR_out2 (1022:0)	Output
Gold_out (1022:0)	Output

Virtex-5 FPGA hardware and timing summary reported by the software [31]. The design provided 9.285 ns of delay and 310.00 MHz of frequency support.

Gold codes are distinguished by their capacity to provide various orthogonal sequences. The Gold codes are generated by combining two or more LFSR code sequences with modulo-2 addition. The LFSR register takes a linear function and exclusive OR operation is applicable for the shifting positions of the bits to get the output. The outputs of the LFSR shift register are coupled exclusively OR to create a feedback loop, which moves the signal from one bit to the next MSB. The LFSR is applied in numerous communication system coding methods, error detection, cryptography, correction methods, and counting operations. The VHDL design and simulation of the LFSR module and 10-bit gold code generator is completed successfully in Xilinx ISE 14.7. The chip functionality is verified by the RTL and waveform simulation with different test inputs. The LFSR register-1 and LFSR register-2 behaviors are verified for the test inputs provided in the simulation and test benches built for testing the designed chip's functionality. Both LFSR and the gold sequence generator are subjected to behavior model-based simulation, which results in 1023-bit output. The chip design of the Gold code generator is done successfully with the integration of the two LFSR modules. The random test inputs are given to check the functionality of both LFSRs and the Gold code generator. The architecture is expandable depending on the requirements of the communication system because it is scalable and reprogrammable.

Ethics statements

The methods used in the study did not involve any human or animal subjects. No data was used or collected for this work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Aakanksha Devrari: Conceptualization, Methodology, Validation, Software, Writing – original draft. **Adesh Kumar:** Methodology, Validation, Supervision, Resources, Writing – review & editing. **Piyush Kuchhal:** Writing – review & editing. **Zoltán Illés:** Writing – review & editing. **Chaman Verma:** Writing – review & editing, Resources.

Data availability

No data was used for the research described in the article.

Acknowledgments

The Department of Media and Educational Informatics, Faculty of Informatics, Eotvos Lorand University, Budapest, Hungary, supported by Dr. Chaman Verma's and Dr. Zoltán Illés's work.

References

[1] F. Liu, M.M. Karbassian, H. Ghafouri-Shiraz, Novel family of prime codes for synchronous optical CDMA, Opt. Quantum Electron. 39 (1) (2007) 79–90.
[2] A. Tsuneda, T. Yoshida, Performance evaluation of asynchronous DS/CDMA communications using unipolar codes, in: 2011 20th European Conference on Circuit Theory and Design (ECCTD), IEEE, 2011, pp. 648–651.
[3] J.M. Velazquez-Gutierrez, C. Vargas-Rosales, Sequence sets in wireless communication systems: a survey, IEEE Commun. Surv. Tutor. 19 (2) (2016) 1225–1248.
[4] R. Gold, Optimal binary sequences for spread spectrum multiplexing (corresp.), IEEE Trans. Inf. Theory 13 (4) (1967) 619–621.
[5] George, M., Hamid, M., & Miller, A. (2001). Gold code generators in Virtex devices. *Xilinx Application Note xapp217, v1, 1*.
[6] J. Alkasassbeh, A. Al-Qaisi, A. Khalifeh, BER performance using linear phase orthogonal binary codes for multi-users mobile communication, Wirel. Person. Commun. 119 (1) (2021) 259–274.
[7] O.B. Wojuola, S.H. Mneney, Multiple-access interference of gold codes in a DS-CDMA system, SAIEE Afr. Res. J. 106 (1) (2015) 4–10.
[8] S. Sajic, N. Maletic, B.M. Todorovic, M. Sunjevaric, Random binary sequences in telecommunications, J. Electr. Eng. 64 (4) (2013) 230.
[9] R. Sarojini, C. Rambabu, Design and implementation of DSSS-CDMA transmitter and receiver for reconfigurable links using FPGA, Int. J. Recent Technol. Eng. (IJRTE). ISSN (2012) 2277–3878.
[10] A. Kumar, P. Vishnoi, S. SL, Smart grid security with cryptographic chip integration, EAI Endor. Transact. Energy Web 6 (23) (2019).
[11] M. Yadav, K. Singh, A.S. Pandey, A. Kumar, R. Kumar, Smart communication and security by key distribution in multicast environment, Wirel. Commun. Mob. Comput. 2022 (2022).
[12] A. Devrari, A. Kumar, A. Kumar, S. Singh, Design and FPGA implementation of DSSS for near-far effect in MANET, in: Proceeding of International Conference on Intelligent Communication, Control and Devices, Springer, Singapore, 2017, pp. 425–434.
[13] M. Medard, A.H. Chan, J.D. Moores, K.L. Hall, K.A. Rauschenbach, S.A. Parikh, Ultrafast cryptography using optical logic in reconfigurable feedback shift registers, in: Multimedia Networks: Security, Displays, Terminals, and Gateways, 3228, International Society for Optics and Photonics, 1998, pp. 342–353.
[14] K.E. Zoiros, T. Houbavlis, M. Kalyvas, Ultra-high speed all-optical shift registers and their applications in OTDM networks, Opt. Quantum Electron. 36 (11) (2004) 1005–1053.
[15] F. Dridi, S. El Assad, W. El Hadj Youssef, M Machhout, Design, hardware implementation on FPGA and performance analysis of three chaos-based stream ciphers, Fract. Fract. 7 (2) (2023) 197.
[16] G.K. Maity, A.K. Mandal, N.B. Manik, J.D. White, All-optical TOAD based new binary sequence generator, Opt. Quant. Electron. 48 (6) (2016) 1–15.
[17] P. Zode, P. Zode, R. Deshmukh, FPGA based novel true random number generator using LFSR with dynamic seed, in: 2019 IEEE 16th India Council International Conference (INDICON), IEEE, 2019, pp. 1–3.
[18] A. Devrari, A. Kumar, Reconfigurable linear feedback shift registers for wireless communication and coding, Int. J. Reconfigur. Embedd. Syst. 12 (2) (2023) 195–204 JulyISSN: 2089-4864, doi:10.11591/ijres.v12.i2.pp195-204.
[19] V. Shivakumar, C. Senthilpari, Z. Yusoff, A low-power and area-efficient design of a weighted pseudorandom test-pattern generator for a test-per-scan built-in self-test architecture, IEEE Access 9 (2021) 29366–29379.
[20] M.B. Mollah, M.R. Islam, Comparative analysis of Gold Codes with PN codes using correlation property in CDMA technology, in: 2012 International Conference on Computer Communication and Informatics, IEEE, 2012, pp. 1–6.
[21] A.R. Yashaswini, P.S.N. Reddy, G.K. Ramaiah, Generation and implementation of IRNSS standard positioning signal, Eng. Sci. Technol. Int. J. 19 (3) (2016) 1381–1389.
[22] A. Gezer, Improving IEEE 802.15. 4 performance with a switched Gold sequence chip formation, Wirel. Netw. 26 (6) (2020) 4579–4593.
[23] S. Gueron, M. Kounavis, Efficient implementation of the Galois counter mode using a carry-less multiplier and a fast reduction algorithm, Inform. Process. Lett. 110 (14-15) (2010) 549–553.
[24] V. Maksymovych, M. Shabatura, O. Harasymchuk, M. Karpinski, D. Jancarczyk, P. Sawicki, Development of additive Fibonacci generators with improved characteristics for cybersecurity needs, Appl. Sci. 12 (3) (2022) 1519.
[25] K. Kashyap, S. Pathak, N.S. Yadav, Optimization of spreading code using modified differential evolution for wireless communication, Wirel. Person. Commun. 122 (2) (2022) 1283–1304.
[26] A. Peinado, A. Fúster-Sabater, Generation of pseudorandom binary sequences by means of linear feedback shift registers (LFSRs) with dynamic feedback, Math. Comput. Modell. 57 (11-12) (2013) 2596–2604.
[27] A. Kumar, A. Kumar, A. Devrari, Hardware chip performance analysis of different FFT architecture, Int. J. Electron. 108 (7) (2021) 1124–1140.
[28] N. Gupta, K.S. Vaisla, A. Jain, A. Kumar, R. Kumar, Performance analysis of AODV routing for wireless sensor network in FPGA hardware, Comput. Syst. Sci. Eng. 39 (2) (2021) 1–12.
[29] N. Gupta, A. Jain, K.S. Vaisla, A. Kumar, R. Kumar, Performance analysis of DSDV and OLSR wireless sensor network routing protocols using FPGA hardware and machine learning, Multimed. Tool. Applic. 80 (14) (2021) 22301–22319.
[30] A. Kumar, A. Kumar, G.S. Tomar, Hardware chip performance of CORDIC-based OFDM transceiver for wireless communication, Comput. Syst. Sci. Eng. 40 (2) (2022) 645–659.
[31] A. Kumar, P. Sharma, M.K. Gupta, R. Kumar, Machine learning-based resource utilization and pre-estimation for network on chip (NoC) communication, Wirel. Person. Commun. 102 (3) (2018) 2211–2231.