Multi-bit throughput LFSRs [*]

Theory and Verilog Implementation

Rev 2.01

Kaveh Fazli

January 2, 2026

Contents

1 Introduction to LFSRs

LFSRs (Linear feedback shift register) are fascinating circuits. They are so simple to implement while their mathematical theory is so complicated that an engineer cannot fully comprehend. In this article we discuss LFSRs from digital design point of view with a little mathematical theories.

Let's start with a simple example. Figure 1 shows a 4-bit LFSR. Four registers are connected as shift-right registers. At every clock (not shown here) a new value is calculated for the MSB bit b_3 by XORing the old values of bits b_3 and b_0. This new value is pushed to the MSB bit and all bits shift to the right. Old LSB bit b_0 is lost or pushed out to be used as the output of this LFSR circuit.

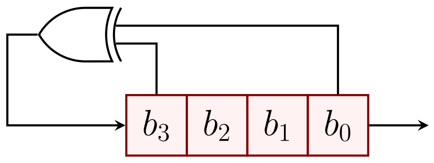


Figure 1: A 4-bit LFSR

An n bit LFSR has 2^n states. The LFSR in Figure 1 has 16 states. Its state diagram is shown in Figure 2. If this LFSR starts with the state $'h0$, it stays in this state forever. Otherwise it goes through all $2^n - 1 = 15$ states periodically. Note that each state has 3 bits similar to its previous and the next state.

Figure 3 shows the bitstream generated by our LFSR. the bitstream also has the period of 15 bits. The states of the registers are also shown on the bitstream. The 3-bit overlap of the states is obvious.

If we look at the state of the registers every 4 clocks as shown in Figure 4, then the states wont have fixed similarities with the neighboring states. Because the period of the bitstream 15 is not a multiple of 4, the states go beyond one period of the bitstream before repeating. Actually, the period of the new state transition is also 15. It means we go through all 4-bit states except 0 before getting back to the first state, but with a different sequence. The state diagram is shown in Figure 5

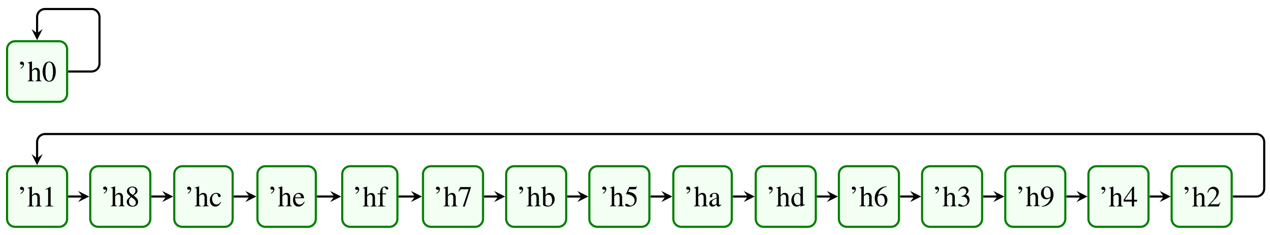


Figure 2: The state diagram for the 4-bit LFSR in Figure 1

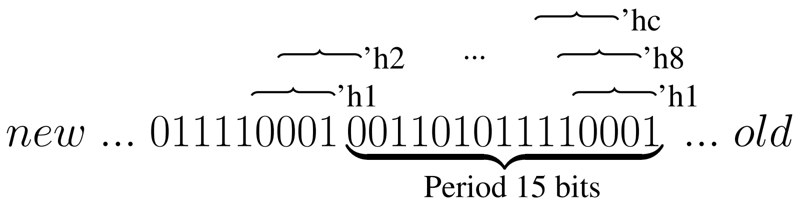


Figure 3: The bitstream generated by the 4-bit LFSR in Figure 1

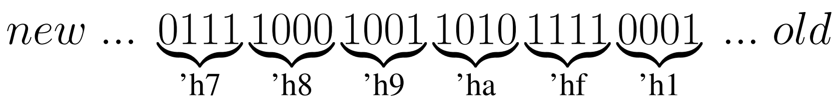


Figure 4: The bitstream with 4-bit groups

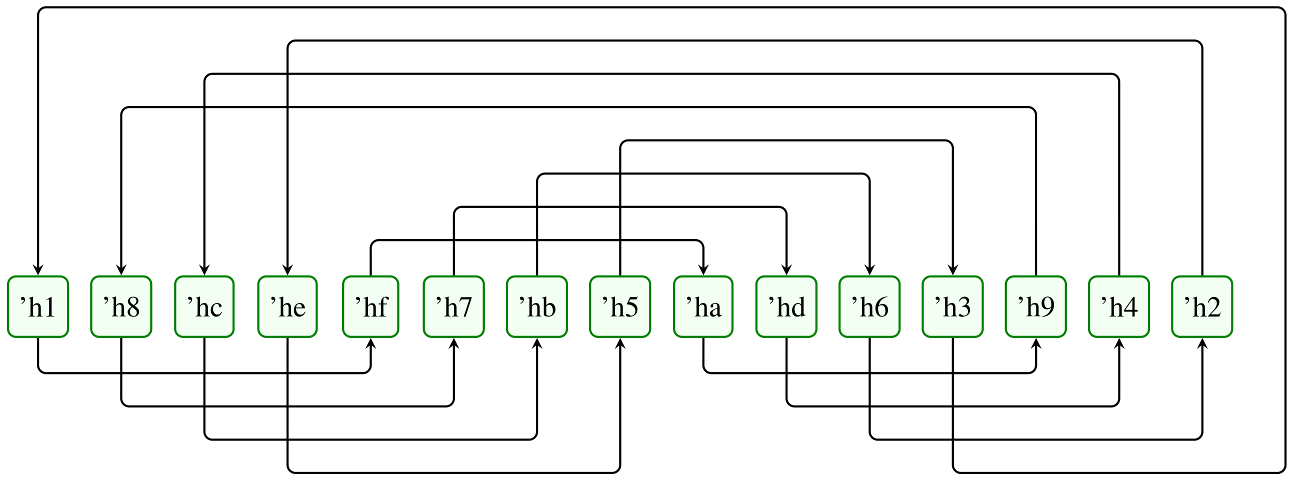


Figure 5: The state transition every 4 clocks

2 The randomness of the bitstream generated by an LFSR

Solomon Golomb (1932 – 2016) was the first mathematician who studied and proved many properties of LFSRs in his 1967 book *"Shift Register Sequences"*[1][2]. In his book he answered the question of how we can define randomness in a sequence of bits that is basically deterministic. He provided three postulates to define a sequence randomness and then mathematically proved that sequences generated by an LFSR satisfy these three postulates to a high degree[3].

We describe the three postulates and examine them on a 6-bit LFSR shown in Figure 6. One period of the sequence (bitstream) that it generates is shown in the Figure 7 and its sequence properties shown in Table 1.

1. **Balance Property:** In a full cycle, the number of 1s and 0s must be equal.
   As the period of the sequence generated by an LFSR is always an odd number $(2^n - 1)$, the number of 1s and 0s are always different by 1. This disparity becomes negligible for LFSRs with higher number of bits (n).

2. **Run Property:** Runs are consecutive sequences of 1s or 0s. There should be equal numbers of 0-runs and 1-runs for each length. Also runs should follow a geometric pattern: half are length 1, one-fourth length 2, one-eighth length 3, and so on.
   as seen in Table 1, the run property is also satisfied with a little disparity. Note that one and only one run of 1s with the length of (n) always exists and a run of 0s with the length of (n) never exists.

3. **Autocorrelation Property:** Autocorrelation function for a period of a sequence can be described this way: We make t bits of circular shift to the sequence. Circular shift means that as we shift, we inject the bits that are pushed out back to the other side. Then, we compare the shited sequence with the original sequence and count the number of similar bits. As we do the same process for different value of t, we get the autocorrelation function. For $t = 0$, which means comparing the sequence with itself the autocorrelation value is $(2^n - 1)$ the length of the sequence period. for other t values it is less. For a random sequence, the value of the autocorrelation function for $t \neq 0$ should be constant. In our sequence of Figure 7, the autocorrelation value for $t = 0$ is 63 and for any $t \neq 0$ is a constant number 31, as shown in Figure 8.

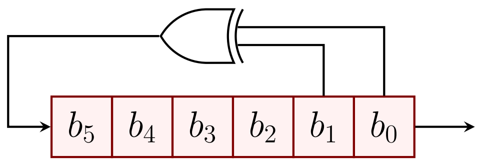


Figure 6: A 6-bit LFSR

0 1 0 1 0 11 00 11 0 111 0 11 0 1 00 1 00 111 000 1 0 1111 00 1 0 1 000 11 0000 1 00000 111111

Figure 7: The bitstream of the 6-bit LFSR in Figure 6

Properties	0s	1s
Total bits	63	
Total runs	32	
Bits	31	32
Runs	16	16
Runs of length 1	8	8
Runs of length 2	4	4
Runs of length 3	2	2
Runs of length 4	1	1
Runs of length 5	1	
Runs of length 6		1

Table 1: Properties of the bitstream shown in Figure 7

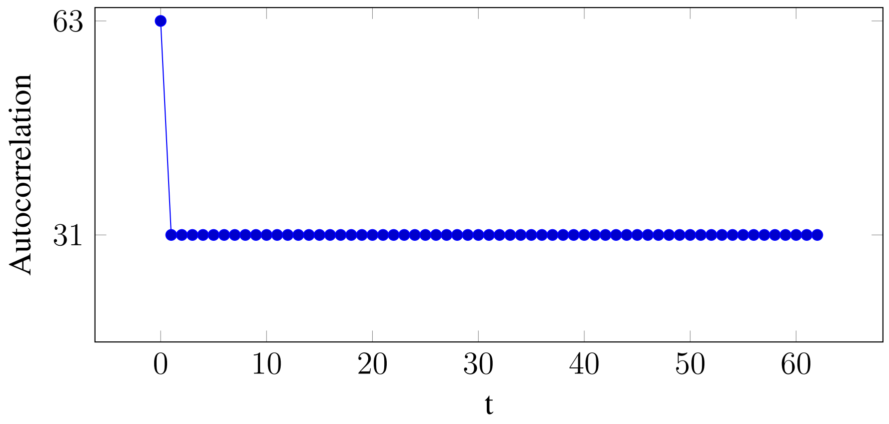


Figure 8: Autocorrelation function of the bitstream shown in Figure 7

3 A non-random behavior in LFSRs

Golomb's second postulate defines a balance of size and counts between 0-runs and 1-runs. However it is silent about the position and distribution of the runs in a period of the bitstream generated by an LFSR. Consider the valid 28-bit LFSR shown in Figure 9, When the LFSR is in the state of all bits being 1, the feedback circuit generates and injects a 0 to the MSB bit b_{27}. It keeps generating 0s for 25 clocks. And, this sequence of 25 0s will generates a sequence of 22 0s and so on until we get to the runs with small lengths. The graph in Figure 10 shows the first 100 runs after all-1 state. The negative numbers are used to indicate 0-runs. You see after every large 1-run there are a few large 0-runs with reducing the length. This type of deterministic behavior with the concentration of the large length runs in one point may not be acceptable for a system that needs more random behavior from a sequence.

The main factor in this non-randomness behavior is the big distance between the last input of the feedback circuit at b_3 and the last bit of the LFSR at b_{27}. Then, to eliminate this kind of non-randomness we need to select LFSRs with more number of feedback inputs (taps), some tap(s) at the MSB bits and more evenly distributed taps.

Another method used in industry to achieve better randomness is to use more than one LFSR (usually 3) in parallel with different length and XORing their outputs.

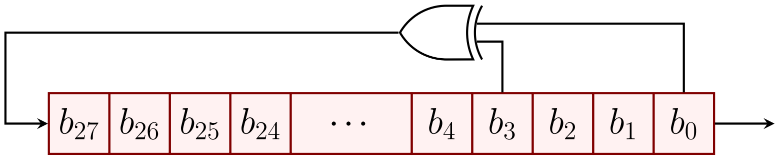


Figure 9: A 28-bit LFSR

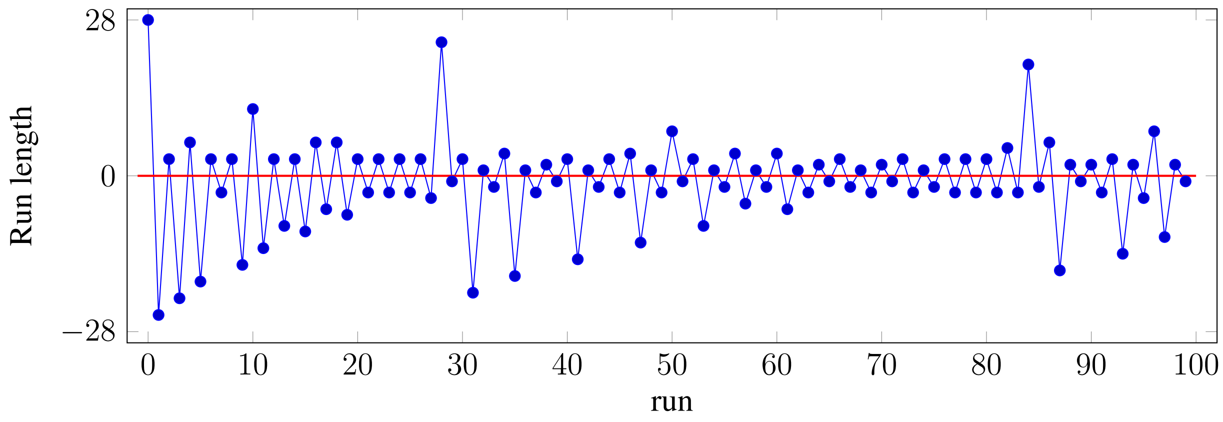


Figure 10: The first 100 runs after all-1 state. The negative numbers are used to indicate 0-runs.

4 Characteristic polynomials and Taps

In this section we discuss the proper feedback circuit[3]. The complete name for the LFSR circuits we discussed in the previous sections is *maximal length sequence*, or *m-sequence LFSRs*. In literature, including this article, LFSR implies *m-sequence LFSR*. In order to get maximal length sequence out of an LFSR, the feedback circuit must be carefully selected. The feedback circuit is always XOR of a few bits of the registers. The location of the registers selected for the input of the XORs is called **taps**. For the LFSR in Figure 1, taps are 0 and 3 and, for the LFSR in Figure 7, taps are 0 and 1.

For every LFSR size, there are a few valid combination of the taps. There are documents that provide tables of valid taps for each LFSR size. However, there 2 traditions to number the bits of the registers. We use the tradition of numbering from $n-1$ to 0. In some literature, they use numbering from 1 to n. If you see for a 6-bit LFSR a tap at 6 is mentioned, it means they are using 1 to n tradition.

When Golomb was studying LFSRs, he noticed similarity of LFSRs behavior with Galois Field (GF) polynomials. He defined a characteristic polynomial in $GF(2)$ for an LFSR as follows: The polynomial degree is the size of the LFSR. So, always a term x^n exists. In addition, there is one polynomial term for each tap with the degree corresponding to the tap. For example, for the 4-bit LFSR in Figure 1 with taps at 0 and 3, the characteristic polynomial is

$$x^4 + x^3 + 1$$

For the 6-bit LFSR in Figure 6 with taps at 0 and 1, the characteristic polynomial is

$$x^6 + x + 1$$

By knowing the characteristic polynomial of an LFSR, you know both the size and the tap locations of the LFSR.

All the properties of an LFSR we discussed before, including being a maximal length sequence generator, relies on the condition that its characteristic polynomial being *primitive*. We don't go to the mathematical meaning of it. Now, It's up to mathematicians to find *primitive* polynomials of different degrees in $GF(2)$ field. We will just take the coefficients of those polynomials as the taps in our feedback circuit.

An important property of a primitive polynomial is that its **reciprocal** is also a primitive polynomial. to find the reciprocal of a polynomial subtract the power of each term from the polynomial degree.

$$x^8 + x^6 + x^5 + x^4 + 1 \quad \Rightarrow \quad x^8 + x^4 + x^3 + x^2 + 1$$

$$8 \Rightarrow 0, \quad 6 \Rightarrow 2, \quad 5 \Rightarrow 3, \quad 4 \Rightarrow 4, \quad 0 \Rightarrow 8,$$

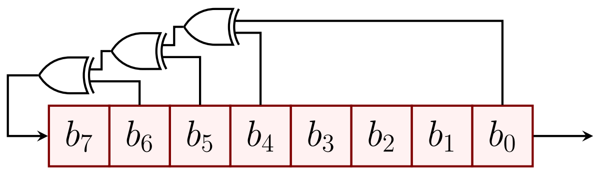


Figure 11: The implementation of $x^8 + x^6 + x^5 + x^4 + 1$

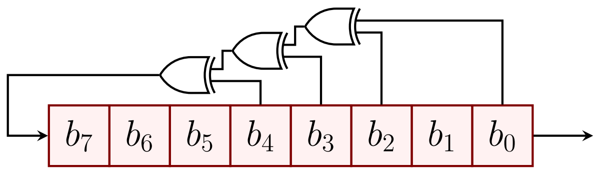


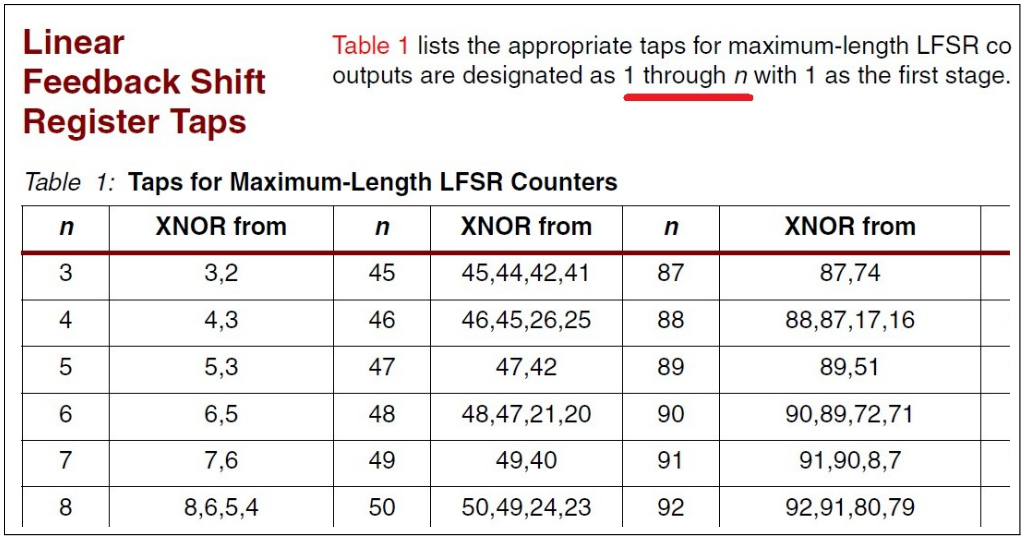
Figure 12: The implementation of $x^8 + x^4 + x^3 + x^2 + 1$

5 Where to find valid tap numbers

Not many engineering documents I found with a table of valid taps. A Xilinx document [4] is the only one so far. There are more references in mathematics world. However you should search for *Primitive Binary Polynomials*. The document may not even mention LFSR word. I found three documents in this subject [5] [6] [7]. Note that for any LFSR size there are many valid group of taps, but the documents usually publish only one or two of them. So, the documents could choose different groups and the numbers be different in each document.

As the format of the tables are different in each document and confusing, I explain briefly the format in these 4 document.

1. A part of taps table in Xilinx document[4] is shown in Figure 13. This document uses the convention of 1 to n numbering. To convert the tap number to $n-1$ to 0 convention, the tap numbers must be subtracted from LFSR size (n). For example for LFSR size 4 the table gives us numbers $4, 3$. To convert it to the convention we used in early sections they becomes $0, 1$. Another example: for LFSR size 47 the tap numbers in our convention is $0, 5$.



Linear Feedback Shift Register Taps

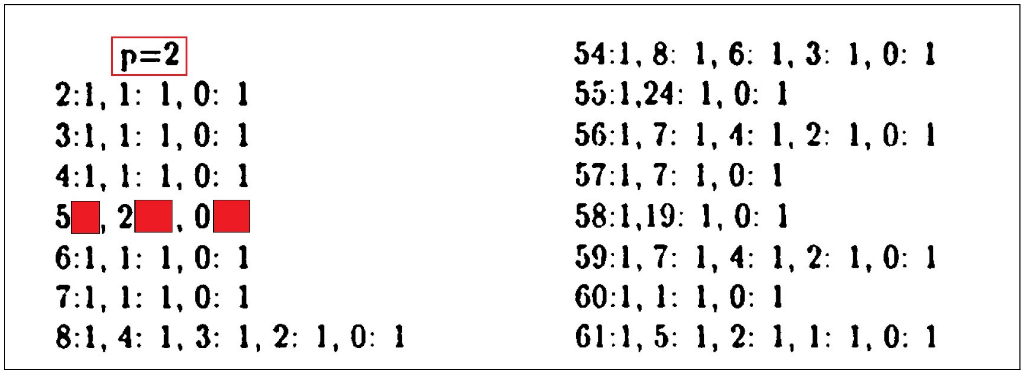
Table 1 lists the appropriate taps for maximum-length LFSR co
outputs are designated as 1 through *n* with 1 as the first stage.

Table 1: **Taps for Maximum-Length LFSR Counters**

n	XNOR from	*n*	XNOR from	*n*	XNOR from
3	3,2	45	45,44,42,41	87	87,74
4	4,3	46	46,45,26,25	88	88,87,17,16
5	5,3	47	47,42	89	89,51
6	6,5	48	48,47,21,20	90	90,89,72,71
7	7,6	49	49,40	91	91,90,8,7
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79

Figure 13: A part of taps table in [4]

2. A part of taps table in [5] is shown in Figure 14. Although, they don't call it taps table. The document uses the same $n-1$ to 0 convention that we use. However, the table has a strange format. There is a ": 1" in front of every number. Just ignore them as shown for LFSR size 5. So, For LFSR size 5, the taps are at bit 0 and 2. Another note about this table is that use only the section dedicated for $p = 2$. The table has other sections for other prime numbers $p = 3$, $p = 5$, etc that we don't use.



p=2

2:1, 1: 1, 0: 1
3:1, 1: 1, 0: 1
4:1, 1: 1, 0: 1
5[redacted], 2[redacted], 0[redacted]
6:1, 1: 1, 0: 1
7:1, 1: 1, 0: 1
8:1, 4: 1, 3: 1, 2: 1, 0: 1

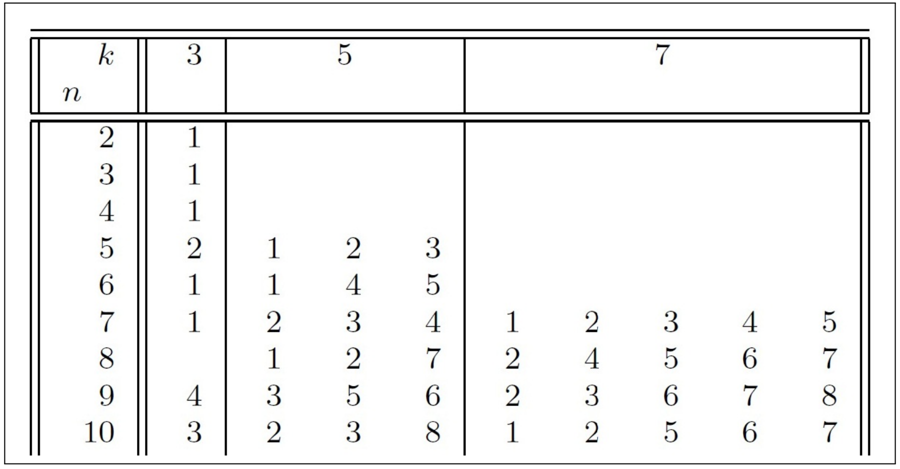
54:1, 8: 1, 6: 1, 3: 1, 0: 1
55:1,24: 1, 0: 1
56:1, 7: 1, 4: 1, 2: 1, 0: 1
57:1, 7: 1, 0: 1
58:1,19: 1, 0: 1
59:1, 7: 1, 4: 1, 2: 1, 0: 1
60:1, 1: 1, 0: 1
61:1, 5: 1, 2: 1, 1: 1, 0: 1

Figure 14: A part of taps table in [5]

3. A part of taps table in [6] is shown in Figure 15. This table provides 2-tap, 4-tap and 5-tap values for each LFSR size, if exists. To save space, the table doesn't show the tap 0 which always exists. So, For LFSR size 5, the taps are at bit 0 and 2, if we want 2 taps. If we want 4 taps, the taps are at bit 0, 1, 2 and 3. The reason the title of the 2-tap values is $k = 3$ is that k indicates the number of terms in the characteristic polynomials. In the characteristic polynomials we have an extra term for x^n. So, the characteristic polynomials of LFSR size 5 according to this table are

$$x^5 + x^2 + 1$$

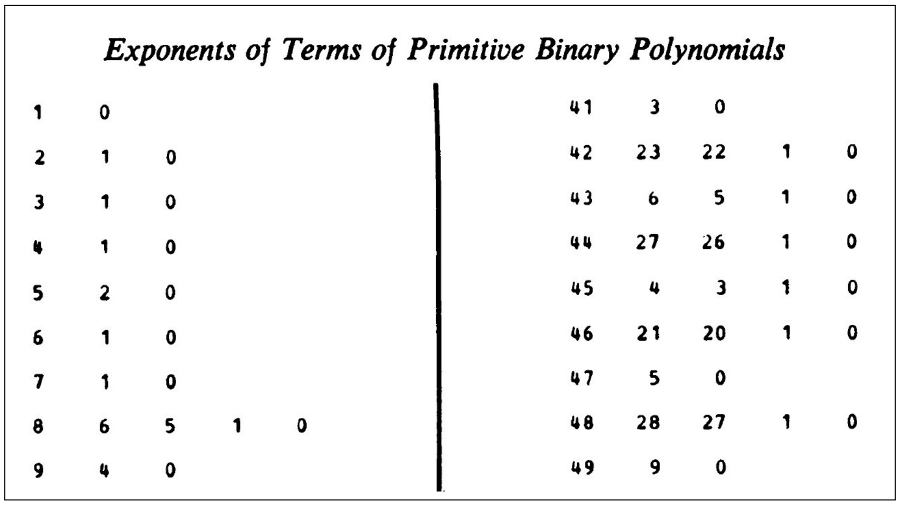
$$x^5 + x^3 + x^2 + x + 1$$



k / n	3	5			7				
2	1								
3	1								
4	1								
5	2	1	2	3					
6	1	1	4	5					
7	1	2	3	4	1	2	3	4	5
8		1	2	7	2	4	5	6	7
9	4	3	5	6	2	3	6	7	8
10	3	2	3	8	1	2	5	6	7

Figure 15: A part of taps table in [6]

4. A part of taps table in [7] is shown in Figure 16. This is the most straight forward table among all we saw. It provides only one 2-tap or 4-tap values for each LFSR size.



Exponents of Terms of Primitive Binary Polynomials

1	0				41	3	0		
2	1	0			42	23	22	1	0
3	1	0			43	6	5	1	0
4	1	0			44	27	26	1	0
5	2	0			45	4	3	1	0
6	1	0			46	21	20	1	0
7	1	0			47	5	0		
8	6	5	1	0	48	28	27	1	0
9	4	0			49	9	0		

Figure 16: A part of taps table in [7]

6 Multi-bit throughput - example

An LFSR, in its basic circuit shifts one bit per clock and generates one random bit output. In this section we will see how we can modify the basic circuit to shift and generate more bits per clock. We start with an example and then will generalize the concept. Consider the LFSR in Figure 17. The state of the circuit at clock t $S(t)$ is the value of each bit at clock t

$$S(t) = [\ b_5(t),\ b_4(t)\ , b_3(t),\ b_2(t),\ b_1(t),\ b_0(t)\ ]$$

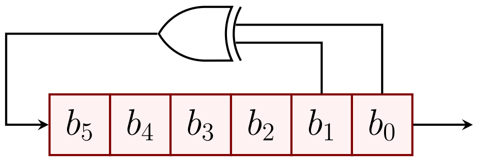


Figure 17: A basic 6-bit LFSR

The state after 1 and 2 clock can be written based on the state at clock t

$$\begin{array}{lllllll}
S(t+1) = [\ b_5(t+1), & b_4(t+1), & b_3(t+1), & b_2(t+1), & b_1(t+1), & b_0(t+1) &]\textbf{(1)} \\
\qquad\quad = [\ b_1(t) \oplus b_0(t), & b_5(t), & b_4(t), & b_3(t), & b_2(t), & b_1(t) &]\textbf{(2)} \\
 & & & & & & \\
S(t+2) = [\ b_1(t+1) \oplus b_0(t+1), & b_5(t+1), & b_4(t+1), & b_3(t+1), & b_2(t+1), & b_1(t+1) &]\textbf{(3)} \\
\qquad\quad = [\ b_2(t) \oplus b_1(t), & b_1(t) \oplus b_0(t), & b_5(t), & b_4(t), & b_3(t), & b_2(t) &]\textbf{(4)}
\end{array}$$

The equation **(1)** is just the definition of $S(t+1)$. The equation **(2)** is based on the structure of the circuit, how the value of each bit at $(t+1)$ is related to the value of other bits at the clock before. The equation **(3)** is also based on the structure of the circuit, how the value of each bit at $(t+2)$ is related to the value of bits at the clock before $(t+1)$. The equation **(3)** is constructed by substituting the value of bits at $(t+1)$ by the value of the bits at (t) using equation **(2)**.

It's better to show the equation **(3)** relationships in a table:

$b_x(t+2)$	$b_x(t)$
b_5	$b_2 \oplus b_1$
b_4	$b_1 \oplus b_0$
b_3	b_5
b_2	b_4
b_1	b_3
b_0	b_2

Table 2: Value of each bit based on the values 2 clock before in LFSR in Figure 17

If we re-wire the circuit according to Table 2, we will have the transition in one clock, instead of 2 clocks. Figure 18 shows the new circuit. We need two feedback circuits (XORs) to feed the two most significant bits. And, we need to shift two bits to the right at each clock. This circuit generates the same sequence that circuit in Figure 17 generates, but two bits at each clock.

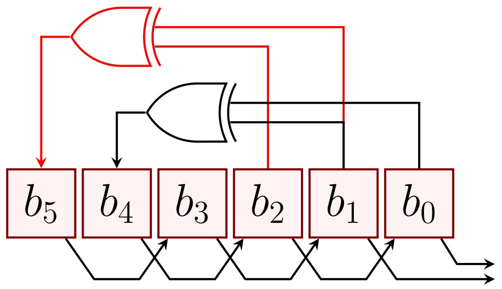


Figure 18: A 6-bit LFSR with 2-bit shift each clock

7 Multi-bit throughput - generalization and limitation

Figure 19 shows the generalization of the concept we saw in the last section. The feedback circuit[1] shown as a box. We want our LFSR generate k bits every clock, so we need to repeat feedback circuit k times. They are numbered from 0 to $k-1$. The input taps of the feedback 0 are at original bit locations. The output of feedback 0 is fed to bit $n-k$. The location of inputs and outputs of other feedback circuits can be find by adding 1 to the location of the previous circuit. Until we get to the feedback $k-1$, the output of the last circuit is connected to $(n-k)+(k-1)=(n-1)$th register which is the last register..

Then, all registers will shift right k bits. It means the old values of registers k to $n-1$ move to registers 0 to $n-k-1$. Note that in the Figure 19, we show two k bits at the MSB registers and two k bits at the LSB registers. It does not imply that k needs to divide n.

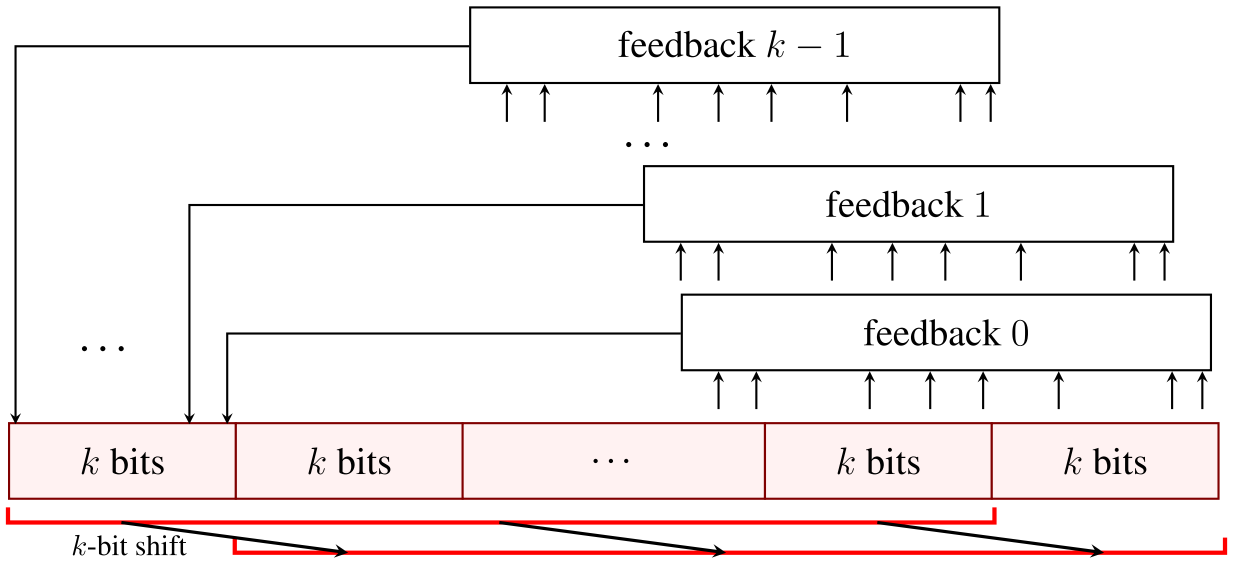


Figure 19: A general LFSR with k-bit shift and throughput

The question is that how large k can be. How many bits can we generate from a given LFSR using this parallel feedback circuit method? To answer The question, let's see what happens when we increase k. The output of the $(k-1)$th circuit is always at the last register $(n-1)$ and the inputs of the circuit 0 is always at the original taps of the LFSR. As k increases, the output location of each feedback circuit gets closer to the location of its inputs. The maximum k is when the output of the 0th circuit is at the location of the the biggest LFSR tap. As we saw earlier in this section

[1]it is worth mentioning that, this method works for any kind of feedback circuit not just "linear feedback". If the feedback circuit consists of only XOR gates, it would be "linear". However, LFSRs don't have much applications in cryptography because they can be hacked easily. Instead, "Non-linear" feedback shift registers are popular in cryptography, meaning that the feedback circuit would have logic gates other then XORs.

the output of the 0th circuit is at bit $n - k$. We can write:

$$\text{MAX}(TAP) \quad = \quad n - \text{MAX}(k)$$

Or:

$$\begin{aligned}\text{MAX}(TAP) &\leq \text{WIDTH } (n) - \text{THROUGHPUT } (k) \\ \text{THROUGHPUT } (k) &\leq \text{WIDTH } (n) - \text{MAX}(TAP)\end{aligned}$$

An interesting case would be the LFSR in Figure 9. The width is 28 bits and the highest tap is at bit 3. So, according to the above formula we can get 25 bits throughput at each clock. **Imagine a 28-bit LFSR that shifts 25 bits at each clock!** It is quite valid and our implementation supports it. It is one of the tests in our implementation testbench.

Although, I haven't seen any literature explicitly defining the formula for the maximum throughput, the selection of the numbers in some literature implies that they believe in the above formula too. For example in the design defined in [8], a 128-bit LFSR and a 128-bit NLFSR[1] is used. For both the maximum tap number is 96. It results in $128 - 96 = 32$ maximum bits of throughput. And, they provide data for different *levels of parallelization* of 1, 2, and 32.

When we discussed non-random behavior of LFSRs, we suggested that using LFSRs with taps closer to MSB bits can reduce this behavior. Here, we see that taps at the higher bits reduces the number of throughput bits we get. So the optimum solution is what they did in [8], Selecting an LFSR with the maximum tap at highest possible location of (WIDTH - THROUGHPUT).

8 Verilog implementation

The LFSR with multi-bit throughput that discussed in the previous section is implemented in Verilog and is available in a GitHub public repository[9] under MIT free software licence.

There is only one module named *mbt_lfsr* in the file with the name *mbt_lfsr.v*. There is also a corresponding testbench file named *mbt_lfsr_tb.v*. Both RTL and the testbench intentionally written in plain Verilog without using SystemVerilog constructs to be compatible with all EDAs. It has been tested using Icarus free Verilog compiler.

Listing 1 shows the header of the module. All aspects of the LFSR is parametrized. The default parameters values define a valid 28-bit LFSR with 2 taps and 8-bit throughput. It is the same LFSR shown in Figure 9.

The testbench *mbt_lfsr_tb.v* includes some basic tests and shows some features and examples of instantiation. It is not meant for the complete verification of the module. There are three groups of instantiation and test:

G1: Three similar LFSRs with different throughput sizes instantiated. The outputs of them are collected for a limited number of clocks. At the end of the simulation the collected outputs are compared for the size of the smallest one.

[1] Nonlinear-feedback shift register

G2: The outputs of the same LFSR with different throughput size are displayed on screen during simulation for a limited number of clocks for visual check. Here you can see the bitstream generated by 1-, 8- and 25-bit throughput of 28-bit LFSR in Figure 9.

G3: The outputs of the LFSRs are not connected. They only show the instantiation options.

Some examples of instantiation is provided in Listing 2.

```verilog
module mbt_lfsr(out, clk, reset, seed_str, seed_val);

  parameter WIDTH = 28; //# of LFSR bits (n)
  parameter TPUT = 8;   //# of TPUT bits (k)
  // Up to 6 taps can be defined. Leave unused TAPs to 0
  parameter TAP0 = 0;
  parameter TAP1 = 3;
  parameter TAP2 = 0;
  parameter TAP3 = 0;
  parameter TAP4 = 0;
  parameter TAP5 = 0;

  output  wire [TPUT-1:0] out;
  input   wire clk, reset;
  input   wire seed_str;
  input   wire [WIDTH-1:0] seed_val;
```

Listing 1: mbt_lfsr Verilog module header

```verilog

  // 1 bit throughput from the default 28-bit LFSR
  wire def_1b;
  mbt_lfsr #(.TPUT(1)) lfsr_def_1b (def_1b, clk, reset, 1'b0, 28'b0);
  // 8 bit throughput from the default 28-bit LFSR
  wire [7:0] def_8b;
  mbt_lfsr #(.TPUT(8)) lfsr_def_8b (def_8b, clk, reset, 1'b0, 28'b0);
  // 25 bit throughput from the default 28-bit LFSR
  wire [24:0] def_25b;
  mbt_lfsr #(.TPUT(25)) lfsr_def_25b (def_25b, clk, reset, 1'b0, 28'b0);
  // seed_str is inserted during clk# 4-8
  wire [15:0] def_16b_s;
  mbt_lfsr #(.TPUT(16)) lfsr_def_16b_s
      (def_16b_s, clk, reset, seed_str, 28'b10101010_11001100_11111111_1111);

  // 50 bit throughput from 6-tap 72-bit LFSR
  wire [49:0] value_50b;
  mbt_lfsr      #(.WIDTH(72), .TPUT(24), .TAP5(22), .TAP4(14), .TAP3(11),
                                         .TAP2(10), .TAP1(6), .TAP0(0) )
      lfsr_72_24 (value_50b, clk, reset, 1'b0, 72'b0);
```

Listing 2: Examples of mbt_lfsr module instantiation

References

[1] S.W. Golomb, *"Shift Register Sequences"*, Holden-Day, San Francisco, CA., 1967

[2] S.W. Golomb, *"Shift Register Sequences"*, Revised ed., Laguna Hills, Aegean Park Press, June 1981

[3] S.W. Golomb, G. Guang, *"Signal Design for Good Correlation For Wireless Communication, Cryptography, and Radar"*, Cambridge, Cambridge University Press, July 2005

[4] Xilinx, *"Linear Feedback Shift Registers in Virtex Devices"*, XAPP210 (v1.3) April 30, 2007 https://docs.amd.com/v/u/en-US/xapp210

[5] T. Hansen, G. Mullen, *"Primitive Polynomials Over Finite Fields"* Mathematics of Computation Vol. 59, No. 200 (Oct., 1992), pp. 639-643 https://www.ams.org/journals/mcom/1992-59-200/S0025-5718-1992-1134730-7/S0025-5718-1992-1134730-7.pdf

[6] M. Zivkovic, *"A Table of Primitive Binary Polynomials"*, 1985 https://poincare.matf.bg.ac.rs/~ezivkovm/publications/primpol1.pdf

[7] W. Stahnke, *"Primitive Binary Polynomials"*, mathematics of computation, volume 27, number 124, October 1973 https://www.ams.org/journals/mcom/1973-27-124/S0025-5718-1973-0327722-7/S0025-5718-1973-0327722-7.pdf

[8] Martin Hell et al., *"Grain-128AEADv2- A lightweight AEAD stream cipher"*, NIST Computer Security Resource Center https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf

[9] Kaveh Fazli, *"MBT_LFSR, Multi-bit-throughput LFSR"*, Github repository https://github.com/Kaveh-Fazli/MBT_LFSR/tree/main

Appendix A - licensing and liability