LFSR Sequence Generation

Long Division in GF2

Juan Chamero, jach_spain@yahoo.es lectures
Discrete Structures & Algorithms, ·009, as of April 2006

Long division is an algorithm for dividing two numbers, obtaining the quotient, one digit at a time. The same applies for polynomials. Let's see an example with integers and their binary representation (base 2 polynomials): 63/5 = [12 3] that reads" prime number 64 divided by prime number 7 gives the pair [12 as quotient and 3 as its residual]". It holds true because 12x5 + 3 = 63. As a positional number 63 ⇔ [111111] and 5 ⇔ [101] with their associated positional polynomials

[111111] ⇔ x^5 + x^4 + x^3 + x^2 + x^1 + x^0
[101] ⇔ x^2 + x^0

Note: Replacing x's by base/module 2 the polynomials results reproduce 63 and 5 as you may check.

Warning: In this example we use regular arithmetic operators by "product" and "sum". In most cryptanalysis and logical applications the pair "AND", "XOR" operators are used instead.

[111111]/[[101] ⇔
x^5 + x^4 + x^3 + x^2 + x^1 + 1 divided by x^2 + 1

Step	Oper	x^5	x^4	x^3	x^2	X^1	1			x^2	0	1
1	Mult	x^5	0	x^3	0	0	0		quotient	x^3	x^2	
1	Substr	0	x^4	0	x^2	X^1	1					
2	Mult		x^4	0	x^2	0	0					
2	Substr		0	0	0	X^1	1					

Giving as result:

[(1100), (11)] ⇔ [12 3]
[(1.x^3 + 1.x^2 + 0.x^1 + 0.x^0), (1.x^1 + 1)]

GF2

LFSR operations can be mathematically understood finite fields, specifically GF2, Galois Fields 2. Fields require that numbers that belong to them remain within the "family" when operated by addition, subtraction, multiplication and division. As you may easily check positive integers do not satisfy because when subtracted appear negative Integers. On the contrary Real numbers will apply if we ignore the case of division by 0. Finite fields deals with finite number of members. GF2 deals with only two members: 0 and 1. These filed is well suited to understand mathematically the functioning of LFSR;s, Linear Feedback Shift Registers (see our document about LFSR's).

Let's see its use to interpret a five bit LFSR ⇔ [B4 B3 B2 B1 B0] with Tap bits in positions 2 and 4. Remember that Tap positions are determined by the "Generator Polynomial" of the whole series, in this case of (2^5 – 1) five bits states terms.

The generator polynomial is determined from the location of the tap positions on the LFSR. The polynomial is fifth order because there are five memory units. Bits 0 and 2 have taps meaning the following polynomial: P(x) = x^5 = x^2 + x^0 = x^2 + 1 = x2 + 1. But on GF2 add and subtract is the same giving the Generator Polynomial G(x) = x^5 + x^2 + 1 = 0. Let's see now how this LFSR works starting with the "seed" [00001].

Functioning: a) shifting to the left; b) feedback bit entering as bit B4; c) output bit: always B0 of "old" state that virtually "falls" outside.

Step j	S(j)	Old B0	P(x)	Numerator Polynomial = (x^j) modulo generator polynomial (x^5 + x^2 + 1)
0	00001	-	-	X^0 = 1
1	00010	0	0	X^1
2	00100	0	0	X^2
3	01001	0	1	X^3
4	10010	0	0	X^4
5	00101	1	1	X^5 = x^2 + 1
6	01011	0	1	X^6 = x^3 +x
7	10110	0	0	X^7 = x^4 + x^2
8	01100	1	0	X^8 = x^3 + x^2 + 1
9	11001	0	1	X^9 = x^4 + x^3 + x
10	10011	1	1	X^10 = x^4 + 1
11	00111	1	1	X^11 = x^2 + x + 1
12	01111	0	1	X^12 = x^3 + x^2 + x
13	11111	0	1	X^13 = x^4 + x^3 + x^2
14	11110	1	0	X^14 = x^4 + x^3 + x^2 + 1
15	11100	1	0	X^15 = x^4 + x^3 + x^2 + 1
16	11000	1	0	X^16 = x^4 + x^3 + x + 1
17	10001	1	1	X^17 = x^4 + x + 1
18	00011	1	1	X^18 = x + 1
19	00110	0	0	X^19 = x^2 + x
20	01101	0	1	X^20 = x^3 + x^2
21	11011	0	1	X^21 = x^4 + x^3
22	10111	1	1	X^22 = x^4 + x^2 + 1
23	01110	1	0	X^23 = x^3 + x^2 + x + 1
24	11101	0	1	X^24 = x^4 + x^3 + x^2 + x
25	11010	1	0	X^25 = x^4 + x^3 + 1
26	10101	1	1	X^26 = x^4 + x^2 + x + 1
27	01010	1	0	X^27 = x^3 + x + 1
28	10100	0	0	X^28 = x^4 + x^2 + x
29	01000	1	0	X^29 = x^3 + 1
30	10000	0	0	X^30 = x^4 + x
31	00001	1	1	X^31 = 1

Recursively any state S(j) could be obtained by multiplying x by S(j-1), for example:

x^25 = x.(x^24) = x.(x^4 + x^3 + x^2 + x^1) = x^5 + x^4 + x^3 + x^2 = (1 + x^2) + x^4 + x^3 + x^2 = x^4 + x^3 + 1
11010111101
Correspondence between states, remainders and "precise quotients" could be now analyzed. To accomplish this task we proceed to divide the numerator polynomial by the generator polynomial obtaining a "fraction" as follows:

x^4 + x^3 + 1 divided by x^5 + x^2 + 1 gives 0,1xxxxxxxxxxx… where the "mantissa" [1xxxxxxxxxxx…] bounded to five bits will give us the "state" S(j), in this case S(25).

Long Division Mechanics in GF2

Numerator					Negative Powers Zone – Virtual zeroed														Generator Polynomial					
4	3	2	1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	5	4	3	2	1	0
1	1	0	0	1	0														1	0	0	1	0	1
1	0	0	1	0	1														1	2	4	6	11	14
0	1	0	1	1	1	0																		
	1	0	0	1	0	1																		
	0	0	1	0	1	1	0	0																
			1	0	0	1	0	1																
			0	0	1	0	0	1	0	0														
					1	0	0	1	0	1														
					0	0	0	0	0	1	0	0	0	0	0									
										1	0	0	1	0	1									
										0	0	0	1	0	1	0	0	0						
													1	0	0	1	0	1						

Being the quotient: [11010100001001……] with 1's in positions 1, 2, 4, 6, 11, 14, ….. as computed above representing the following x powers: x^-1, x^-2, x^-4, x^-6, x^-11, x^-14, …, respectively. The division executes classically as we were told when Childs, adding zeroes to the right of the virtual limit of units and fractions (from the black zone in the table). In summary it is only one long division, namely: in the first step x^j is divided by the generator polynomial to define the Numerator Polynomial module the Generator Polynomial, and then step 2 continuing the division of the "remainder", generating the corresponding "state" S(j) and if continuing the whole states sequence. To illustrate this fact let's execute the first step classically taking into account that S(j) is x^j|mod(x^5 + x^2 + 1). In the table below we compute x^25| mod(x^5 + x^2 + 1), depicted in black background. Quotient polynomial is not shown here.

Powers of x until computed coefficients of x^4, x^3, x^2, x^1, and x^0 ⇔ [11001]																									Generator Polynomial					
2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	5	4	3	2	1	0
5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0						
1	0	0	0	0	0																										
1	0	0	1	0	1																										
0	0	0	1	0	1	0	0	0																							
			1	0	0	1	0	1																							
			0	0	1	1	0	1	0	0																					
					1	0	0	1	0	1																					
					0	1	0	0	0	1	0																					
						1	0	0	1	0	1																					
						0	0	0	1	1	1	0	0	0																	
									1	0	0	1	0	1																	
									0	1	1	1	0	1	0																
										1	0	0	1	0	1																
										0	1	1	1	1	1	0															
											1	0	0	1	0	1															
											0	1	1	0	1	1	0														
												1	0	0	1	0	1														
												0	1	0	0	1	1	0													
													1	0	0	1	0	1													
													0	0	0	0	1	1	0	0	0	0									
																	1	0	0	1	0	1									
																	0	1	0	1	0	1	0								
																		1	0	0	1	0	1								
																		0	0	1	1	1	1	0	0						
																				1	0	0	1	0	1						
																				0	1	1	0	0	1						

We invite you to fill the quotient below Generator Polynomial.