The "Multiplexed and Twisted" GFSR, a Flexible Scheme for the Creation of Pseudo-random Generators

Thierry Moreau

April 2000
Revised April 2005 and October 2005

CONNOTECH Experts-conseils Inc.
9130 Place de Montgolfier
Montreal, Qc
Canada H2M 2A1

Tel.: +1-514-385-5691
Fax: +1-514-385-5900
e-mail: thierry.moreau@connotech.com

Table of contents

1. Introduction . . . . . - 2 -

2. From LFSR to GFSR to TGFSR to MTGFSR . . . . . - 2 -
   2.1 The TGFSR Generators . . . . . - 2 -
   2.2 Multiplexing TGFSR Generators . . . . . - 4 -

3. Applications . . . . . - 6 -
   3.1 A General Purpose 32-Bit Generator . . . . . - 6 -
   3.2 A Large Period Generator . . . . . - 6 -
   3.3 A Frogbit Application Example . . . . . - 7 -

4. References . . . . . - 8 -

1. Introduction

We describe the arrangement of the *Multiplexed and Twisted Generalized Feedback Shift Register* (MTGFSR) pseudo-random generator. The first part of this description is mathematical and serves as the justification of this arrangement. The second part of the description is more procedural, and focuses on efficient implementation issues. The third part gives specific implementation details for a few application.

2. From LFSR to GFSR to TGFSR to MTGFSR

2.1 The TGFSR Generators

The TGFSR generators are defined in two articles by Makoto Matsumoto and by Yoshiharu Kurita ([1], [2]). They represent a noteworthy improvement in the field of generators based on linear feedback shift registers (LFSR).

The Twisted GFSR is definitely more attractive than the plain GFSR, so we will not review the latter. In reviewing the TGFSR, we will not delve into the details of the mathematical justification. The TGFSR allows an efficient software implementation of an LFSR corresponding to a polynomial of relatively large degree, and with a reasonable number of non-zero coefficients.

The TGFSR uses an internal state of N computer words, each with a word size of w bits. Given these quantities, the critical parameters are a number M, where $1<M<N$, and a polynomial $A=a_w t^w+a_{w-1}t^{w-1}+ \ldots +a_2 t^2+a_1 t+a_0$, with binary coefficients a_i. The polynomial A has degree w (hence $a_w=1$), and has specific properties as follows. The polynomial A must be irreductible (hence $a_0=1$), but need not be primitive. A more stringent requirement is that the following derived polynomial

$$B=\sum_{i=0}^{w} a_i(t^N+t^M)^i,$$

of degree $N \times w$, must be primitive. The TGFSR is defined by the parameter quadruple $<w,N,M,A>$.

Then, the TGSFR algorithm operates on computer words treated as positive integers (notation x) in the range $0..2^w-1$, as follows:

Initialization of generator seed:

initialize the N computer words of state $x_{[0]}$, $x_{[1]}$, … $x_{[N-1]}$ so that they are not all equal to zero.

Generator step:

to draw a pseudo-random bit, use

$$x_{[i]}=x_{[i-N+M]}\oplus\lfloor x_{[i-N]}/2\rfloor\oplus\alpha \qquad \text{for } i=N, N+1, \ldots,$$

where $\alpha=0$ if the least significant bit of $x_{[i-N]}$ is 0, and

$$\alpha=a_0\times 2^{w-1}+a_1\times 2^{w-2}+ \ldots +a_{w-2}\times 2+a_{w-1} \text{ otherwise;}$$

the output bit is a subset of the bits of $x_{[i]}$.

(The notation $\oplus$ represents the "bit-wise exclusive or" operation on the binary representation of numbers).

When implementing the generator step, the integer division by 2 is a right shift by one bit. At first glance, the determination of α requires a conditional jump (or an "if" statement in a structured programming language). This is avoided by using a two-entry table for α, and using the least significant bit of $x_{[i]}$ as a table index. This jump optimization technique is necessary for our efficient multiplexed implementation explained below. It is expressed mathematically as a function p(x) defined as follows

$$p(0)=0$$
$$p(1)=a_0\times 2^{w-1}+a_1\times 2^{w-2}+ \ldots +a_{w-2}\times 2+a_{w-1},$$

and a re-statement of the generator step as follows:

$$x_{[i]}=x_{[i-N+M]}\oplus\lfloor x_{[i-N]}/2\rfloor\oplus p(x_{[i-N]}\odot 1) \qquad \text{for } i=N, N+1, \ldots,$$

(The notation $\odot$ represents the "bit-wise and" operation on the binary representation of numbers).

For comparaison purposes, the GFSR uses $x_{[i]}=x_{[i-N+M]}\oplus x_{[i-N]}$ as the generator step function. When used to generate a floating point number in the interval $[0,1]$, the TGFSR was first proposed with the output $x_{[i]}/2^w$. There is a statistical defect with this simple scheme, which was corrected by the second article by Matsumoto and Kurita. This issue is irrelevant in cases where a single bit is used at each step, because then the output sequence is the exactly the one produced by an LFSR based on the (primitive) equivalent polynomial B (with no better nor worse statistical properties). In other cases, the multiplexing of TGFSRs described below is intended to enhance the pseudo-random properties of the original TGFSR. When implementing a TGFSR, it is wise to test its output using the Berlekamp-Massey algorithm [3], as a verification that an LFSR based on the equivalent polynomial B is indeed implemented.

A few remarks about the procedural aspect of TGSFR parameter generation. The "i" exponent refers to repeated polynomial multiplication. Since we are handling binary polynomials, the "$\sum$" sign refers to "sum modulo 2", or exclusive or of coefficients of the same degree. The test for irreducibility (say for $w\leq 32$) can be made by exhaustive search of (irreducible) polynomial

divisors (like a test of integer primality). The test for primitivity requires the knowledge of the prime factors P_i of $2^{N\times w}-1$. A polynomial B is primitive if

$$(t)^{2^{N\times w}-1} \bmod B=1, \text{ and}$$

$$(t)^{(2^{N\times w}-1)/P_i} \bmod B\neq 1 \text{ for every } P_i.$$

This test can be efficiently implemented with the divide-and-conquer exponentiation algorithm.

2.2 Multiplexing TGFSR Generators

Multiplexing of TGFSR is made possible by combining elementary TGFSRs with common values for N and M. The number of TGFSRs is the interleave factor F. We expand the above notation with the interleave index j, where $0\leq j<F$. Thus for each elementary TGFSR, the symbols w, A, and B replaced respectively by w_j, A_j (with polynomial coefficients $a_{w_j,j}$, $a_{w_j-1,j}$, ... $a_{2,j}$, $a_{1,j}$, $a_{0,j}$), and B_j in the definition of the multiplexed TGFSR.

In the multiplexing scheme, the computer word representation is based on a combined word of length W bits. In practice, W might be a multiple of the native computer word size. The output bits are the t least significant bits after a generator state transition, where t is bounded by a truncation limit T (thus $0<t\leq T$). The value of t need not be constant for every successive draw of the generator (e.g. a random byte draw might set $t=8$, while an IEEE single precision floating point draw might need $t=23$ bits of mantissa before a normalization operation takes place). To ensure that at least t bits are pseudo-random (i.e. not a constant zero) and that W is large enough to accommodate every w_j, the parameters W and T must be selected such that

$$w_j\times F+j\geq T \text{ for every } j, \text{ and}$$

$$(w_j-1)\times F+j\leq W-1 \text{ for every } j.$$

Presumably, a smaller t would yield better statistical properties for the PRNG, but it is a waste of computing resources to select $t<F$.

The MTGFSR generator period can be determined. Each elementary TGFSR has the same maximum length period as an LFSR with B_j as the generating polynomial, that is $2^{(N\times w_j)}-1$ (assuming the generator parameters were selected according to [1]). In its multiplexed form, the period of our generator is

$$\text{lcm}(2^{(N\times w_0)}-1,\ 2^{(N\times w_1)}-1,\ \ldots,\ 2^{(N\times w_{F-1})}-1).$$

This suggests that longer periods (and perhaps better statistical properties) can be obtained when

the w_j are different.

We now introduce a notation for computer words. An interleaved computer word x_j is defined for each elementary TGFSR as

$$x_j=\sum x_{k,j}\times 2^k, \text{ where}$$

$$0\le x_{k,j}\le 1 \text{ for } 0\le k<w_j, \text{ and}$$

$$x_{k,j}=0 \text{ otherwise (i.e. } w_j\le k\le (W-1-j)/F).$$

The combined computer word is then

$$X=\sum x_{k,j}\times 2^{k\times F+j}, \qquad \text{for } 0\le j<F,\ 0\le k<w_j,$$

which reads as follows when bits are read left to right from the most significant to the least significant bit:

$$\cdots x_{k+1,0}\ x_{k,F-1}\ x_{k,F-2}\ \cdots\ x_{k,j}\ \cdots\ x_{k,1}\ x_{k,0}\ x_{k-1,F-1}\ \cdots\ \cdots\ x_{1,1}\ x_{1,0}\ x_{0,F-1}\ \cdots\ x_{0,2}\ x_{0,1}\ x_{0,0}$$

The multiplexed algorithm is thus formalized as follows

Initialization of generator seed:

initialize the N combined computer words of state $X_{[0]}, X_{[1]}, \ldots X_{[N-1]}$ so that every interleaved computer word j is not all zero (i.e. for each j, $0\le j<F$, there is at least one pair (k,i) such that $x_{[i]k,j}=1$, $0\le k<w_j$, and $0\le i<N$).

Generator step:

to draw a pseudo-random bit, use

$$x_{[i]}=x_{[i-N+M]}\oplus\lfloor x_{[i-N]}/2^F\rfloor\oplus P(x_{[i-N]}\odot(2^F-1)) \qquad \text{for } i=N, N+1, \ldots,$$

where $P(y)$ is a function defined on $0\le y<2^F$ as

$$P(2^F-1)=\sum a_{(w_j-1-k),j}\times 2^{k\times F+j}, \qquad \text{for } 0\le j<F,\ 0\le k<w_j,$$

$$P(y)=P(2^F-1)\odot\sum y\times 2^{k\times F}, \qquad \text{for } 0\le y<2^F-1,\ 0\le k\le (W-1)/F.$$

Typically, the function $P()$ is pre-computed and implemented as a table lookup operation.

The generator output is the subset of the bits of $x_{[i]}$, defined as $x_{[i]} \odot (2^t-1)$.

3. Applications

3.1 A General Purpose 32-Bit Generator

A 32-bit implementation is suggested with $N=21$, $M=19$, $F=3$, $W=32$, $T=32$, and the following polynomials:

j	w_j	A_j ($a_{w_j,j}$, $a_{w_j-1,j}$, ... $a_{2,j}$, $a_{1,j}$, $a_{0,j}$ in hexadecimal notation)
		B_j ($b_{(N\times w_j),j}$, $b_{(N\times w_j)-1,j}$, ... $b_{1,j}$, $b_{0,j}$ in hexadecimal notation)
0	11	one of B33, B8B, CC7, CD3, CE3, D0F, or EF3
		resp. AA00AA000028002900010000000000282801010000000000000280001, AA00AA000028002900010AAAA00000000000000000AA00000000280001, AA00AE400440000000000AAAA044440000000000000000044000280001, AA00AE400440000000000AAAA044440000001010000000000000280001, AA00AE400440000000000AAAA044440282800000000000000000280001, AA00AE400440000100010000000000000000000000AA00044000280001, AA00AE400468002800000AAAA044440282801010000000000000280001
1	11	see note
2	10	one of 465, 5A1, or 6B5
		resp. 4400440000000000000044440282800000000000044000000001, 4400440000100010AAAA00000028280000000000000000000001, or 4400468002800000AAAA00000028280101000000044000000001

Note: for $j=1$, use one of the unused choice of polynomial for $j=0$.

3.2 Large Period Generators

When looking for a large period generator, N=13, M=2, F=8, W=128, T=72, and the following polynomials:

j	w_j	A_j ($a_{w_j,j}$, $a_{w_j-1,j}$, ... $a_{2,j}$, $a_{1,j}$, $a_{0,j}$ in hexadecimal notation)
		B_j ($b_{(N\times w_j),j}$, $b_{(N\times w_j)-1,j}$, ... $b_{1,j}$, $b_{0,j}$ in hexadecimal notation)
0	9	3CD
		2005000801402005008094360750 51
1	10	7F3
		4002015000801402205508414223575 05
2	11	FE5
		8014022055000801402205C0941626755411
3	12	1897
		1000801002014000080100300490901207406115
4	13	3BBB
		2005000801022254500080300720 4D090B237676545
5	14	6BF9
		4002014000841022344500480341620 5D490B626675541
6	15	DFEB
		8014020051080941602715D014821453285D4969663777445
7	16	1EEDB
		1000801402205408094162274 5C0148234560858496B766567145

The above generator has been extended to even larger period: N=13, M=2, F=8, W=248, T=176, and the following polynomials:

j	w_j	A_j ($a_{w_j,j}$, $a_{w_j-1,j}$, ... $a_{2,j}$, $a_{1,j}$, $a_{0,j}$ in hexadecimal notation)
		B_j ($b_{(N\times w_j),j}$, $b_{(N\times w_j)-1,j}$, ... $b_{1,j}$, $b_{0,j}$ in hexadecimal notation)
0	22	6FDF6F
		400201400084142234550008014020051080941602711D034160459695F622D365773455
1	23	EFD7B1
		80140220540809416227455000801402005108014060870 5D234D6185A69266F24573354501
2	26	6EFFAED
		400201400080140220540849436367455000801402201 50A080122A740C234D63C1E6B3E2E6E363665451
3	27	DEFFBDD
		80140200510008014022 05C0949616737455000801 40A2115280C5122A740D234DE3D1E693F7E6E363475151
4	28	1E09E8B9
		1000801402215400000000100080148231560041000 81140A21142A1C0122254481148A311E0B59292F346624541
5	29	3DE40ECB
		2005000801422055100080340720 5C0943603715410020050008014220 5D1140A2314720DC1D41606F1C08632567045
6	30	7FE40D85
		400201500084142235550048034172 05C4943633755450020150008414223 5D5144A0310720DC5941637F5C00206514011
7	31	EFE40D6B
		80140220540809416227 45D014823456285D49636727D55502205408094162274DD154A23146285D4D63473B5409610733445

3.3 A Frogbit Application Example

The Frogbit data integrity algorithm requires 10 independent pseudo-random sources, numbered 0 to 9. Each one is utilized at a different pace, and two pseudo-random bits are extracted from any one at each step, hence $t=2$. The least significant bit in the pair provides the Frogbit keystream $k1$. The other fixed MTGFSR parameters are $N=3$, $M=1$, $F=2$, $W=26$, and $T=22$. The elementary TGFSRs widths are $w_0=13$ and $w_1=11$ for frogbit generators 0, 2, 4, 6, and 8, while $w_0=11$ and $w_1=13$ for frogbit generators 1, 3, 5, 7, and 9.

With these parameters, there are 49 valid polynomials for $w_j=11$ and 176 for $w_j=13$ (the same would be true if we selected $M=2$). The default selection is based on an equal spacing in the sorted list of polynomials, i.e. ranks 2, 7, 12, 17, 22, 27, 32, 37, 42, and 47 for $w_j=11$ and ranks 9, 26, 44, 62, 79, 97, 114, 132, 150, and 167 for $w_j=13$ (zero-based ranking). These ranks are respectively allocated to frogbit generators from 0 to 9.

The parameter $T=22$ accommodates a Frogbit key schedule in which 22 bits may be initialized in each table entry (initializing the four higher bits require special provision for the two unused bit positions in each table entry).

4. References

[1] Matsumoto, Makoto, and Kurita, Yoshiharu, ***Twisted GFSR Generators***, ACM Transactions on Modeling and Computer Simulations, Vol. 2, No. 3, July 1992, pp 179-194

[2] Matsumoto, Makoto, and Kurita, Yoshiharu, ***Twisted GFSR Generators II***, ACM Transactions on Modeling and Computer Simulations, Vol. 4, No. 3, July 1994, pp 254-266

[3] Massey, James L., ***Shift-Register Synthesis and BCH Decoding***, IEEE Transactions on Information Theory, Vol. IT-15, no. 1, January 1969, pp 122-127