Generation of special form shift registers using a dedicated software platform

Paweł AUGUSTYNOWICZ

Institute of Mathematics and Cryptology, Faculty of Cybernetics,
Military University of Technology
ul. Gen. S. Kaliskiego 2, 00-908 Warszawa
pawel.augustynowicz@wat.edu.pl

ABSTRACT: This article describes crucial functionalities of a Unified Framework for Nonlinear Feedback Shift Register Generation (UFfNG). The core of UFfNG framework is a unified algorithm for Nonlinear Feedback Shift Registers (NLFSR) enumeration which can be effectively implemented in heterogeneous environments including CPUs, GPUs and FPGAs. For the sake of completeness, implementation and efficiency results for each platform are discussed and presented.

KEYWORDS: Nonlinear Feedback Shift Registers, de Bruijn sequences, pseudo-random number generation

1. Introduction

The problem of the effective generation of pseudorandom binary sequences with good statistical properties is a common issue that finds many practical applications, e.g.: in cryptography, software testing, and simulations [8], [10]. In his renowned work on pseudorandom sequence generation, S.W. Golomb proposed using shift registers with special forms of feedback for this purpose, in order to obtain bit sequences with the desired statistical or structural properties [8]. This approach has found many proponents due to its

effectiveness and simplicity and is successfully used to this day. In particular, designers of stream ciphers eagerly use shift registers in their designs [3], [5], [7], although both for safety and performance reasons, it is necessary to ensure appropriate feedback functions. Generating feedback functions with special forms for shift registers is the task of the UFfNG software platform presented in this paper. Due to the fact that such investigations are characterised by significant complexity, the UFfNG software platform includes full support for parallel and distributed computing and graphics accelerator cards, as well as a module that can be run on FPGA systems (Field Programmable Logic Array).

2. Theoretical introduction

Definition 1

A shift register with a feedback function of the f *order* n *can formally be defined as a representation of vector space* $\mathbf{F}_2^n$ *on the same vector space with the form:*

$$(x_0, x_1, ..., x_{n-1}) \mapsto (x_1, ..., x_{n-1}, f(x_0, x_1, ..., x_{n-1})), \tag{1}$$

where function f*, having* n *variables is referred to as the feedback function.*

A given shift register with a feedback function f is referred to as linear if function f is linear, and nonlinear if the corresponding feedback function f is nonlinear.

Definition 2

Consider a sequence of binary values s*. Sequence* s *is referred to as periodic if for a certain value* $p > 0$*, relation* $\forall_{i \geq 0} : s_i = s_{i+p}$ *is true. The lowest value of* p *that has the property discussed is defined as the period of sequence S.*

It must be added that the maximum period of a sequence generated by a shift register with a feedback function of the order n is 2^n. From the point of view of applicability, long periods of sequences generated by shift registers are an extremely important property.

Definition 3

A de Bruijn sequence of the order n *is a binary sequence with period* 2^n, *in which every* n*-element tuple occurs precisely once.*

Definition 4

A modified de Bruijn sequence is a sequence obtained from a proper de Bruijn sequence by removing one zero from an ordered tuple containing zero-elements only.

Modified de Bruijn sequences can be generated by both linear and nonlinear-feedback shift registers. For linear feedback registers, it has been proven that they generate modified de Bruijn sequences with a maximum period $2^n - 1$ only when the feedback function is defined by a primitive polynomial over $\mathbf{F}_2[x]$. However, no similar statements have been constructed in relation to nonlinear-feedback registers, and the only method of finding nonlinear functions that generate sequences with the properties discussed is comprehensive searching of the prospective feedback functions. On the one hand, it is an extremely exacting computational task, on the other, for the given order n, there are $B_n = 2^{2^{n-1}-n}$ different de Bruijn sequences, the fact of which was used as early as 1894 by the French mathematician Flye Sainte-Marie [4], and independently by G. de Bruijn in 1946 [3].

Another sequence type, interesting from the standpoint of practical and theoretical applications, are so called square m-sequences. Their statistical and structural properties were first comprehensively described at the beginning of the 1990s, and in 1994 were presented at the Fast Software Encryption conference in Leuven, Belgium [2], and have since been keenly studied [1].

Definition 5

A square m-sequence is a bit sequence generated by a shift register with a feedback function with the following form:

$$f(x_0, x_1, ..., x_{n-1}) = \sum_{0 \le i \le j \le n-1} a_{ij} x_i x_j \tag{2}$$

Square *m*-sequences are characterised by a very interesting form of the generated sequence, which can be presented as:

$$\forall_{k\geq 0} : s_{n+k} = \sum_{0\leq i\leq j\leq n-1} a_{i,j} s_{i+k} s_{j+k} \,. \tag{3}$$

Studies of square m-sequences have determined the method for algorithmically generating such sequences. It has been ascertained, for example, that it is possible to construct Boolean functions that generate the sequence form by introducing nonlinear disturbances into linear functions. Consider a Boolean function with the following form:

$$f(x_0, x_1, ..., x_{n-1}) = g(x_0, x_1, ..., x_{n-1}) + x_i + x_i x_j \tag{4}$$

for which $i \neq j, 1 \leq i, j \leq n-1$, and $g(x_0, x_1, ..., x_{n-1})$ is a linear function that generates a complete-period sequence. It is an obvious conclusion that $g(x_0, x_1, ..., x_{n-1})$ must be defined by a primitive polynomial over $\mathbf{F}_2[x]$. For the given degree, there are $\frac{\varphi(2^n - 1)}{n}$ primitive polynomials, and consequently also different sequences that they generate (often referred to as linear sequences).

3. The UFfNG software platform

The primary task of the UFfNG software platform is to enable its users to effectively generate both linear- and nonlinear-feedback shift registers with the maximal periods. The most important premise that the UFfNG software platform meets is the fact that it alone enables performing parallel computations on multiple CPUs (Central Processing Unit) using MPI (Message Passing Interface) or on graphics accelerators with OpenCL (Open Computing Platform) or CUDA (Compute Unified Device Architecture) support. Only running VHDL code (Very High Speed Integrated Circuits Hardware Description Language) requires additional steps related to compiling and uploading software to an appropriate FPGA system, which is a standard procedure for such systems.

3.1. Key features and the interface

The UFfNG software platform provides the user with the following features:

- testing the periods of shift registers with any feedback functions;
- constructing primitive polynomials over binary bodies, i.e. the corresponding linear feedback functions, the use of which in shift registers enables sequences with the maximal periods to be generated;
- generating feedback functions with a specific number of polynomials, with the assumption that the functions can be generated in lexicographical order or reverse lexicographical order;
- generating feedback functions that generate square *m*-sequences with the maximal periods;
- checking whether a register with a specific feedback function generates square *m*-sequences.

3.2. Feedback function testing algorithm

The core of the presented UFfNG software platform is a versatile period test algorithm for linear- and nonlinear-feedback shift registers. It checks the given function's period completeness by enumerating successive states and verifying their uniqueness. When designing the algorithm, the fact that to demonstrate the maximality of the given shift register, it suffices to demonstrate that its initial state will be generated after exactly $2^n - 1$ steps.

For the purpose of accurate representation and analysis of the feedback function testing algorithm, assume the following data designations:

- LFSR – bit representation of the linear part of the feedback function;
- NLFSR – bit representation of the nonlinear part of the feedback function;
- N – order of the shift register tested;

//DATA: LFSR; NLFSR; N;

//ALGORITHM

1. state := 0x1;
2. **for i=1,…,(2^N)-1:**
   2.1. b_LFSR := (popcount(state and LFSR)) mod 2;
   2.2. b_NLFSR := (popcount(state and NLFSR))/popcount(NLFSR);
   2.3. bit := b_LFSR xor b_NLFSR;

2.4. state := (state<<1) xor bit;
2.5. **if** state==1 **then:**
2.5.1. **return false;**
2.6. **end if;**
3. **end for;**
4. **if** state==1 **then:**
4.1. **return true;**
5. **else**
5.1. **return false;**
6. **end if;**

Algorithm 1. Pseudocode of the period testing algorithm for sequences generated by a shift register with a specific feedback function

To complement the designations assumed in the pseudocode of algorithm 1, it should be added that *popcount* means an operation returning the number of ones in the given string, *mod* – an operation of division with remainder by a number, and the division used in point 3.2. is done integrally, rounding the result down.

3.2.1. Implementation of the algorithm on CPUs

The effective implementation of the feedback shift register period testing algorithm assumes complete parallelisation both at the level of the processor's threads and the use of SIMD (Single Instruction Multiple Data) vector instructions. With the use of the SIMD vector instructions offered by the latest processors, simple operations, such as xor, bit shifts or counting ones in a string can be done even on 512 bits concurrently, instead of the typical 32 or 64 bits. For example, thanks to this approach, each of the 8 threads in a Intel Core i7-6700 CPU can process up to 256 bits in parallel. It appears that the performance increase offered by vector instructions fully justifies their utility. If the given processor does not support vector instructions, the algorithm is run without this support.

Another improvement used in the algorithm is the ability to force the compiler to use the `popcntq` assembler instruction to count the ones in a given string. The effectiveness of this solution is significantly greater even than the best algorithmic solutions for determining the Hamming weight.

3.2.2. Implementation of the algorithm on graphics accelerators

The use of the OpenCL (Open Computing Language) programming platform assumes a universal approach to programming graphics accelerators [10]. However, version 1.1 of the standard, commonly supported by current graphics accelerators, does not enable using the `popcntq` instruction to count the ones in a given vector. It must be noted that this ability will be available in the latest version of the OpenCL platform (2.0) [1] and in the NVIDIA solution CUDA (Compute Unified Device Architecture), supported only by graphics cards manufactured by this company. Given the above, it was decided to use two separate implementations, one in the OpenCL standard, and the other in the CUDA C language, whose use depends on the graphics accelerator available. This approach enables using the computing power of the available computing environment to the fullest. Another improvement, available only in the CUDA implementation, is asynchronous data feeding to the card without the need to pause calculations. This way, at practically no point during computing does the card wait for new data to be sent to the processing unit, which eliminates delays related to the communication interface.

3.2.3. Implementation of the algorithm on FPGA systems

Presently, the implementation of the algorithm on FPGA systems assumes support only for Intel Altera systems due to the use of the Nios II microprocessor to handle the communication interfaces [10]. For FPGA systems, there are feedback function period testing algorithms better adjusted to their specific character, although they require recompiling the system each time the structure of the feedback function changes. Algorithm 1 bypasses this problem and does not require recompiling at any stage of its operation, which greatly streamlines its use for people unfamiliar with the hardware description languages and FPGA systems.

Using the Nios II microprocessor was necessary to conceal the process of communication with the CPU. In the implementation prepared for FPGA systems, the microprocessor receives data from an external interface and stores them in its buffer, waiting for the system to complete its current calculations. When they are done, it immediately feeds an appropriate data portion, eliminating the issue of delays related to the communication interface.

4. Results

Table 1 shows the durations of testing 100 MB of potential feedback functions for various shift register lengths for selected CPUs, GPUs and FPGA systems, and for different orders of shift registers.

Table 1. Expected time to test a 100 MB package of potential feedback functions for shift registers

n	Intel Core i7-6700 [s]	NVDIA GeForce 980GTX [s]	Intel Altera Cyclone V DE0-CV [s]
23	114	40	48
24	245	134	101
25	445	203	206
26	857	405	410
27	1626	838	823
28	3136	2179	1763
29	6056	3812	6127
30	11687	-	-

When using the computing platform discussed, searches are performed for feedback functions with special forms that generate m-sequences, among others. Sample functions with such properties, with a degree of 30 are:

1. $x_0 + x_1 + x_4 + x_6 + x_8 + x_{12} + x_{14} + x_{16} + x_{23} + x_{28} + x_9 x_{22}$
2. $x_0 + x_2 + x_3 + x_{11} + x_{16} + x_{21} + x_{23} + x_{24} + x_{28} + x_{29} + x_{25} x_{29}$
3. $x_0 + x_2 + x_7 + x_{14} + x_{16} + x_{18} + x_{22} + x_{24} + x_{26} + x_{29} + x_8 x_{21}$

Detailed results are presented in [1].

5. Summary and further studies

To summarise, the paper presented the main features and capabilities of the UFfNG software platform. In particular, the feedback shift register period testing algorithm and its implementations on different computing platforms was discussed.

The development of the UFfNG software platform presented is intended to be continued. In particular, as part of further work on the platform and its use, the following are intended:

- support for version 2.0 of the OpenCL programming platform;
- implementation of the shift register period testing algorithm on Xilinx FPGA systems;
- rewriting of the most critical sections of the processor code in a low-level language;
- further studies on square m-sequences and generating all feedback functions that generate square m-sequences up to degree 32.

References

[1] AUGUSTYNOWICZ P., KANCIAK K., SZMIDT J., *Performance evaluation of NLFSRs enumeration on heterogenous environments*. preprint 2018.

[2] CHAN A., GAMES J., RUSHANAN J., *On the quadratic m-sequences.* Proceedings of Fast Software Encrypton. LNCS, vol. 809, 1994, pp. 166-173.

[3] DE BRUIJN N.G., *A combinatorial problem*. Indag. Math., 8, 1946, pp. 461-467.

[4] DĄBROWSKI P., ŁABUZEK G., RACHWALIK T., SZMIDT J., *Searching for Nonlinear Feedback Shift Registers with Parallel Computing*. Information Processing Letters, Vol. 114, No. 5, May, 2014, pp. 268-272.

[5] DE CANNIÈRE C., PRENEEL B., *Trivium specifications*. eSTREAM, ECRYPT Stream Cipher Project, 2006.

[6] FLYE SAINT-MARIE C., *Solution to question nr. 48*. L'intermèdiaire des Mathématiciens 1, 1894, pp. 107-110.

[7] GAMMEL B., GÖTTFERT R., KNIFFLER O., *Achterbahnd-128/80*. ECRYPT Stream Cipher Project Report, 2006.

[8] GOLOMB S.W., *Shift Register Sequences.* Holden-Day, Inc., San Francisco, 1967.

[9] HELL M., JOHANSSON T., MEIER W., *Grain – A Stream Cipher for Constrained Environments*. International Journal of Wireless and Mobile Computing, Vol. 2, No. 1, 2007, pp. 86-93.

[10] RIVAT J., SÁRKÖZY A., *On Pseudorandom Sequences and Their Application*. In: General Theory on Information Transfer and Combinatorics, LNCS, Vol. 4123, Springer, 2006, pp. 343-361.

Electronic sources

[11] https://www.khronos.org/opencl/

[12] https://www.altera.com/products/processors/overview.html

Generacja rejestrów przesuwnych szczególnych postaci przy wykorzystaniu odpowiedniej platformy programistycznej

STRESZCZENIE: W niniejszym artykule zaprezentowano i omówiono przygotowaną przez autora platformę programistyczną, umożliwiającą generację rejestrów przesuwnych z funkcjami sprzężenia zwrotnego szczególnych postaci, o nazwie UFfNG (ang *Unified Framework for Nonlinear Feedback Shift Register Generation)*. Rdzeniem zaprojektowanej platformy programistycznej jest uniwersalny algorytm sprawdzania okresu rejestrów przesuwnych o liniowych i nieliniowych funkcjach sprzężenia zwrotnego, który został z powodzeniem zaimplementowany do użytkowania w heterogenicznych środowiskach obliczeniowych, zawierających procesory CPU, akceleratory graficzne bądź układy FPGA. W celu zilustrowania efektywności i uniwersalności prezentowanego rozwiązania zamieszczono przykładowe wyniki wydajności dla przedstawicieli poszczególnych platform.

SŁOWA KLUCZOWE: rejestry przesuwne, sekwencje de Bruijna, generacja ciągów pseudolosowych

Received by the editorial staff on: 14.05.2018