Some aspects of program implementation pseudorandom sequence generators

O. Geiko
Department of Computer Science
Vasyl Stefanyk Precarpathian
National University
Ivano-Frankivsk, Ukraine
ifgo69@gmail.com

S. Dolinovska
Department of Computer Science
Vasyl Stefanyk Precarpathian
National University
Ivano-Frankivsk, Ukraine
sdolinovska@gmail.com

Деякі аспекти програмної реалізації генераторів псевдовипадкових послідовностей

О. Гейко
кафедра інформатики
Прикарпатський національний університет
імені Василя Стефаника
Івано-Франківськ, Україна
ifgo69@gmail.com

С. Доліновська
кафедра інформатики
Прикарпатський національний університет
імені Василя Стефаника
Івано-Франківськ, Україна
sdolinovska@gmail.com

***Abstract* — This article considers some aspects of the software implementation of pseudorandom generators, for example generator according to the scheme on the Galois shift registers with linear feedback, with using high-level programming languages, high-level languages with low level of abstraction and language of Assembler. It is proved the efficiency of the software generation of pseudorandom sequences of maximum length, which is based on the algorithms by using the following commands manipulate bits.**

***Анотація* — В статті розглянуто деякі аспекти програмної реалізації генераторів псевдовипадкових послідовностей, на прикладі генератора за схемою Галуа на регістрах зсуву з лінійними зворотними зв'язками, з використанням мов програмування високого рівня, мов високого рівня з низьким рівнем абстракції і мови Assembler. Доведено ефективність програмного забезпечення генерування псевдовипадкових послідовностей максимальної довжини, яке базується на алгоритмах з використанням команд маніпулювання бітами.**

Keywords—pseudorandom number generator; maximum length sequence; a Galois field; algorithm; Assembler.

Ключові слова—генератор псевдовипадкових чисел; М-послідовність; поле Галуа; алгоритм; Асемблер.

I. INTRODUCTION

The pseudorandom number generators are widely used in different areas of scientific and practical activities, simulation modeling, methods of statistical testing, probabilistic testing, and applied cryptography.

One of the ways of obtaining random numbers is the use of mathematical transformations which allow to obtain a numeric sequence, which by their characteristics is close to real random processes.

There are a number of methods for constructing pseudorandom number generators [1-2]. Mainly focuses on hardware implementation of digital elements, although many of the software modules is the need to generate pseudorandom numbers it programmatically.

The aim of the report is the analysis of the algorithmic features of generating pseudo-random sequences over a Galois field GF(2) by means of programming languages high-level programming language of high level with low abstraction level and the Assembler language.

II. DESCRIPTION OF BASIC MATERIAL

Pseudorandom number generators are widely used in various areas of scientific and practical activities, such as simulation, methods of statistical tests, probability testing and applied cryptography.

Mathematical transformations are one of the ways to generate random numbers. This method provides a numerical sequence with characteristics similar to real random processes.

There are several methods for constructing pseudorandom number generator (PRG). Basically it is a hardware implementation based on digital elements, which allows to provide high speed characteristics of the generator, although many software modules needs to generate pseudorandom sequences using software methods. Algorithmic generator component implementation is given insufficient attention.

The purpose of the report is the analysis of algorithmic software features to generate pseudorandom sequence over a Galois field GF (2) by high-level languages, high-level programming language with low level of abstraction and Assembler language.

A key problem in implementing generators pseudorandom binary sequences is the problem of the formation of pseudorandom sequence (PRS) maximum length $L=2^n-1$ with acceptable statistical characteristics.

One of the main ways to implement the generators PRS is the use of linear shift registers (LSR) a maximum period of linear feedback.

This way, under certain conditions, provides a pseudo-random binary M-sequences or maximum length sequences (MLS).

When using transformations over Galois fields GF (p^n) for M-sequence must fulfill certain conditions [3]:

- p - prime number.
- All initial values are not necessarily equal to 0.
- Generating polynomial is irreducible (not decomposed into multipliers lower degree).
- Generating polynomial is primitive (minimal polynomial of a primitive element of the field GF (p^m) for a positive integer m).

The figure shows the block diagram of the PRG configuration over Galois primitive polynomial linear feedback is formed based on a irreducible primitive polynomial $f32= x^{32}+x^{22}+x^2+x+1$ [4]:

Fig. 1. Block diagram PRG

Galois generator compares each nonzero element of the field GF (2^{32}) corresponding degree primitive element W = | 1; 0 |, modulo $f32= x^{32}+x^{22}+x^2+x+1$.

Depending on the outcome of the command and the installation ROL and value CF, bats reverse connection will be formed according to the following table (operation XOR):

TABLE I. CONVERTING BITS BASED ON THE CARRY FLAG

CF	Previous bit	Runny bits	Action
1	0	1	Inversion p.bits
1	1	0	
0	0	0	Duplication p.bits
0	1	1	

To generate PRS effective methods are [5-6]:
1. Using bit instruction.
2. Using bit mask.
3. Using bits field.

1. *Use the bit instruction*

With the implementation of this oscillator manner appropriate to have used the shift command and groups intended for inspection or installation of specific bits register using Carry Flag processor architecture IA-32. So for a status register 20 bit and 21 bit setting (for the circuit shown in Figure 1), you can use the following command sequence (MASM32):

```
NextCode proc ;START CONVERSION
      ROL EAX,1 ; Rotate left EAX trough carry
      .IF CARRY?; Test carry flag (CF) == 1 ?
        BT EAX, 20; Insert bit №20 EAX in CF
        .IF CARRY? ; CF == 1
              BTR EAX, 21 ; reset bit №21 EAX 0
        .else
          BTS EAX, 21 ; set bit №21 EAX 1
        .endif
          ...
      .else       ; CF == 0
      ...
        BT EAX, 20 ;Insert bit №20 EAX in CF
        .IF CARRY? ; CF == 1
              BTS EAX, 21 ; set bit №21 EAX 1
        .else
          BTR EAX, 21 ;reset bit №21 EAX 0
        .endif
```

2. Using bits field

High level languages allow you to use structures that represent a specified number of bits is "bit field". The bit field is interpreted as an integer type.

```
struct BitSet {
      unsigned short m00 : 1;
      unsigned short m01 : 1;
      …
      unsigned short m30 : 1;
      unsigned short m31 : 1;
};
```

This arrangement allows you to get easy access to single bits by performing the necessary manipulations with them, but characterized redundancy command processor to check individual fields.

To perform shift operations is advisable to create the type of union bit field and a variable of type integer (C language):

```
union mKode {
        unsigned int kode;
        BitSet mGalua;
};
```

Then shift operation and establishing certain bit may be represented by the following code:

```
if (t.mGalua.m31==1)CF=true; //check bits №31
    t.kode=t.kode << 1; //shift left by 1 bit
if (CF) {
        if (t.mGalua.m20 == 1) //check bits №20
         t.mGalua.m21 = 0;//reset bits №21
        else t.mGalua.m21 = 1; //set bits №21
    else {
        if (t.mGalua.m20 == 1) //check bits №20
         t.mGalua.m21 = 1; //set bits №21
        else t.mGalua.m21 = 0; //reset bits №21
    …}
```

3. Using bit mask

The bulk of the high-level languages, can perform bitwise operations on operands that are the basis for the program PRG.

Checking and setting specific bit can be performed using bit-wise operations.

For example, the block diagram in Figure 1:

check bit №20 - mask 0x00100000, and the operation "logical AND";

setting bit №21 - mask 0x00200000 and operation "logical OR";

reset bit №21 - mask 0xFFDFFFFF and operation "logical AND".

Using masks for bitwise operations in the bit field for program PRG, may presented as code (C#):

```
bool CF = false;
if ((t&0x80000000)>0)CF=true;//check bits №31
  t = t << 1;
  if (CF){
    …
   if((t & 0x00100000) > 0) //check bits №20
      t &= 0xFFDFFFFF;    //reset bits №21
   else t |= 0x00200000;   //set bits №21
    …
  else {
    …
   if((t&0x00100000) == 0) // check bits №20
      t &= 0xFFDFFFFF;   //reset bits №21
   else t |= 0x00200000;  //set bits №21
    …
}
```

III. TEST CONDITIONS

Software implementation of the PRG of the field GF (2^{32}) based on a primitive irreducible polynomial $f_{32}= x^{32}+x^{22}+x^2+x+1$ and length 2^{31}

Software:
OS – Windows 10 Home x32
bits instruction: MASM32
bits mask: Visual C# Express 2015
bits field: Visual C++ Express 2015

Hardware
Intel® Core™ i5-4200U (up to 2.60 GHz) 4GB DDR3
AMD Phenom II X4 955 Black Edition 3.2GHz, RAM 4GB DDR2

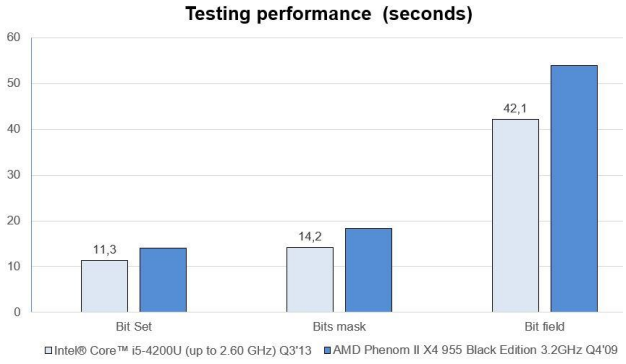

Fig. 2. Testing performance (seconds)

CONCLUSION

Thus, studies have found that program implementation to generate the PRS over a Galois field GF (2) offer a number of ways that include algorithmic complexity and speed implementation.

Using bit instruction, are available in the language Assembler have the highest performance, though inferior to hardware implementation and are complex algorithmic solution.

Using bit mask available in almost all programming languages are characterized by simplicity and algorithmic implementation is their best solution, providing sufficient performance.

Using bits field in high-level languages with a low level of abstraction can significantly simplify the algorithmic complexity, but do not provide sufficient performance generation.

It should be mentioned that speed software implementation significantly depends on the CPU architecture, speed RAM.

REFERENCES

[1] Л.Б. Петришин, Теоретичні основи перетворення форми та цифрової обробки інформації, К.: ІЗіМН МОУ, 1997.

[2] В. М. Кузнецов Генераторы случайных и псевдослучайных последовательностей на цифровых элементах задержки (основы теории и методы построения): дис. … докт. техн. наук: 05.13.05 : захищена 28.01.2012. / Кузнецов, Валерий Михайлович; Казань, 2011.- 347 с.

[3] Donald E. Knuth. Art of Computer Programming, Volume 2: Seminumerical Algorithms. (3rd Edition). Addison-Wesley. Professional; 3 edition, 1997.- 784 p.

[4] А. Я. Белецкий Примитивные матрицы и генераторы псевдослучайных последовательностей Галуа / / Белецкий.А.Я., Белецкий.Е.А. // Информационные технологии в образовании. - 2014. - № 18. – С. 14-29.

[5] Schneier, Bruce. Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth) (Publisher: John Wiley & Sons, Inc. 816 p.

[6] Н. Ф. Казакова Программная реализация универсального статистического теста Маурера для анализа псевдослучайных последовательностей / Н. Ф. Казакова, Ю. В. Щербина // Інформаційна безпека. - 2011. - №7(161). – С. 289-296.