Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 1. pp. 313–320
doi: 10.14794/ICAI.9.2014.1.313

Efficient random number generation on FPGA-s[*]

Tamás Herendi, Roland Major

University of Debrecen
`herendi.tamas@inf.unideb.hu`
`major.sandor@inf.unideb.hu`

Abstract

The presented work uses the massive parallelism made possible by FPGA chips to greatly speed up a computationally heavy algorithm for constructing uniformly distributed linear recurring sequences. Two approaches are detailed, based on the exponentiation of large matrices and polynomials. FPGA implementations of these approaches are compared to solutions on traditional architectures.

Keywords: random numbers, FPGAs, optimization

1. Introduction

Linear recurring sequences over residue class rings can be basic components of particular pseudo random number generators. Finding such recurring sequences may be based on checking some well defined but computationally rather expensive properties of some related object.

Let a sequence $\{x_n\}_{n=0}^{\infty}$ of integers be a linear recurring sequence of order k. The coefficients of the sequence determine the corresponding characteristic polynomial and companion matrix.

In [2] sufficient condition is given for the uniform distribution of x_n modulo powers of 2. However, this requires the computation of the 2^kth power of the $k \times k$ companion matrix modulo 4. In [3] a highly parallel solution is presented, based on FPGA computation, which achieved a speedup factor of 200.

In the present work we provide a refined condition on the uniform distribution of linear recurring sequences. This condition requires the computation of high powers of a given polynomial reduced by the characteristic polynomial. Denoting

[*]The research is supported by the TMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund. P

the characteristic polynomial by $p(x)$, we have to find the smallest N, such that $x^N - 1$ is divisible by $p(x)$ modulo 2.

Based on this method a parallel solution is presented. This improved FPGA implementation allows for the computation of sequences with periods at least one order of magnitude larger. The presented work uses custom built hardware that contains multiple FPGA chips to further improve the scale of parallelism.

2. Hardware used in the implementation

The modules detailed below are implemented on a Xilinx XUPV505-LX110T development platform. The board features a the Virtex-5 XC5VLX110T FPGA. The FPGA's main tool for computation is the array of 6-input look-up tables. The FPGA features a total of 69120 LUTs. A single 6-input LUT can store 64 bits of data, where its six input bits are used as an address to identify the single bit of data that is to be outputted. By manipulating the 64 bit content of the look-up table, it can be configured to carry out arbitrary boolean functions with at most six input bits. One out of four LUTs on the device can also be used as a 32 bit deep shift register.

A 256MB DDR2 SODIMM was also attached to the board, to be used for storing data exceeding the amount that can be practically stored on the FPGA.

3. Mathematical background

The present work is initiated by [2] about the construction of uniformly distributed pseudo random number sequences with large period lengths.

Definition 3.1. Let $a_0, \ldots, a_{d-1} \in \mathbb{Z}$ and $u = \{u_n\}_{n=0}^{\infty}$ be a sequence in $\mathbb{Z}$ satisfying the recurrence relation

$$u_{n+d} = a_{d-1}u_{n+d-1} + \cdots + a_0 u_n \quad \text{for} \quad n = 0, 1, \ldots \tag{3.1}$$

u is a linear recurring sequence (LRS) with defining coefficients $a_0, \ldots, a_{d-1}$ and initial values $u_0, \ldots, u_{d-1}$. The order of the recurrence is d.

$$P(x) = x^d - a_{d-1}x^{d-1} - \cdots - a_0 \tag{3.2}$$

is a characteristic polynomial of u.

The following theorem about the connection between the distribution of linear recurring sequences and their characteristic polynomials can be found in detail in [2].

Theorem 3.2. *Let $P, Q, P_i \in \mathbb{Z}[x]$ where $i = 1, 2, 3, 4$,*

a. Q be monic irreducible modulo 2 of degree k

b. $P(x) \equiv (x^2 - 1)Q(x) \mod 2$

c. $P_1(x) = P(x)$
$P_2(x) = P(x) - 2$
$P_3(x) = P(x) - 2x$
$P_4(x) = P(x) - 2x - 2$

d. $u^{(i)}$ *are LRS corresponding to* P_i*, with the greatest minimal period length modulo* 2*.*

Then at least one of the $u^{(i)}$'s has uniform distribution modulo 2^s with minimal period length $2^s\mathrm{ord}(Q)$ for any $s \in \mathbb{N}$.

We present two approaches for finding the sequence with uniform distribution.

4. Approach based on matrix exponentiation

For a charactersitic polynomial P_i constructed according to the above theorem, let u be the corresponding LRS.

Let

$$M(u) = \begin{pmatrix} 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \\ a_0 & a_1 & \dots & a_{d-2} & a_{d-1} \end{pmatrix}$$

be the companion matrix of sequence u. To find which of the above sequences has a uniform distribution, we need to compute $M(u)^{2^{d+1}-2}$. If $M(u)^{2^{d+1}-2}$ equals the identity matrix, then the period length of u_n is $2^{d+1} - 2$, which means it is not the sequence we are searching for.

The exponentiation of matrices to high powers can quickly become time consuming on traditional computers. Using FPGA devices, the computation can be done in a greatly parallel way, which results in a large speed up in comparison to traditional architectures.

4.1. Structure of modules used in the computation

The FPGA implementation for the approach based on matrix exponentiation can be found in more detail in [3].

The basic elements of the design are the LUTs denoted by $L(a, b, s) = c$, where a, b, c and s are two-digit binary numbers. The function performed by L is a multiply-accumulate (for short: MA) function, i.e.:

$$c \equiv (a \cdot b) + s \mod 4 \tag{4.1}$$

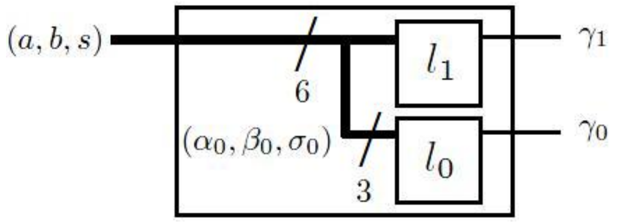

Figure 1: The structure of $L(a, b, s)$

Let $a = 2\alpha_1+\alpha_0$, $b = 2\beta_1+\beta_0$, $s = 2\sigma_1+\sigma_0$, $c = 2\gamma_1+\gamma_0$, where $\alpha_0, \alpha_1, \beta_0, \beta_1, \sigma_0, \sigma_1, \gamma_0, \gamma_1 \in \{0, 1\}$, and $L = (l_1, l_0)$ where l_1 and l_0 are two single bit LUTs.

With the help of these basic units one can compute the dot product w of two vectors $u = (u_0, u_1, ..., u_{n-1})$ and $v = (v_0, v_1, ..., v_{n-1})$. Let us define a module m by cascading n MA units. In this module m we use the output of a given MA unit as the sum input of the next unit.

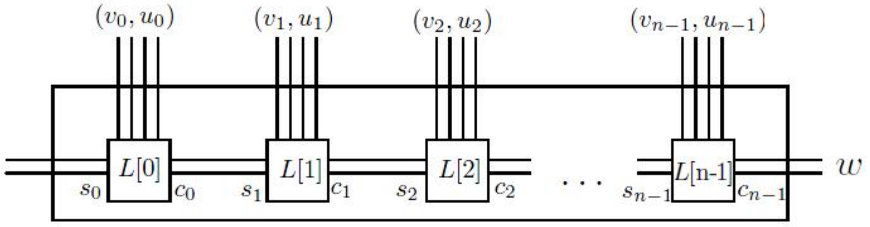

Figure 2: The structure of $m(u, v)$

Therefor m is a function that accepts a pair of vectors u, v of two-digit numbers of length n and outputs on c_{n-1} the two-digit dot-product of the two vectors, i.e. $m(u, v) = w$.

Note that vectors of arbitrary length can be used in the computation if we connect the output of module m to the sum input of $L[0]$ ($c_{n-1} = s_0$), and then iteratively shift u and v onto the module's input by n elements at a time.

The number chosen for n is critical in setting many technical characteristics of the modules. The optimal number depends on the hardware used and the clock frequency. The experiment used for determining n can be found in [3].

Our aim is to obtain a module that performs the matrix multiplication of $A, B \in \mathbb{Z}_4^{k\times k}$, where $\mathbb{Z}_4$ is the mod 4 residue class ring. In the following, let $C \in \mathbb{Z}_4^{k\times k}$ be the output matrix, such that $C = A \times B$. Furthermore, let a_i be the ith row of matrix A and let b_j be the jth column of matrix B.

The multiplier units denoted by m are used to create more complex modules in a hierarchical manner. In the implementation, 400 such modules were created on the FPGA chip, arranged such that it computes a 20×20 block of C in one iteration of the computation.

$$M_{20\times 20}(a_i, a_{i+1}, ..., a_{i+19}, b_j, b_{j+1}, ..., b_{j+19}) =$$

$$= \begin{pmatrix} c_{i,j} & c_{i,j+1} & \cdots & c_{i,j+19} \\ c_{i+1,j} & c_{i+1,j+1} & \cdots & c_{i+1,j+19} \\ \vdots & & \ddots & \\ c_{i+19,j} & c_{i+19,j+1} & \cdots & c_{i+19,j+19} \end{pmatrix} \tag{4.2}$$

The $M_{20\times 20}$ unit can be used iteratively to multiply matrices of arbitrary size, producing 20×20 sub-matrices of the output matrix C with each iteration. After inputting twenty rows from matrix A and twenty columns from matrix B and obtaining the desired output, we can simply repeat the process for a set of rows and columns of A and B respectively, until we obtain the entire output matrix C. Since for almost all practical cases the size k of matrices $A, B \in \mathbb{Z}_4^{k\times k}$ will be greater than the parameter n, the vectors taken from these matrices will need to be iteratively shifted onto the input of the multiplier $M_{20\times 20}$, n elements at a time.

The multiplier module described here is a greatly parallel design that speeds up the computation of matrix multiplication. On the hardware used, the modules achieve a speed up factor of about 200 compared to more traditional software solutions.

The exponentiation of matrices can be done efficiently via the Montgomery Ladder fast exponentiation method.

```
Function $montgomery\_ladder(M, k)$
        1. Let $R_1 = M, R_0$ the unit matrix
        2. for $j = t - 1$ downto 0 do
                3. if $k_j = 0$ then $R_1 = R_0 * R_1$, $R_0 = R_0^2$
                4. else $R_0 = R_0 * R_1$, $R_1 = R_1^2$
        5. end for
        6. return $R_0$
end Function
```

where M is the matrix we wish to exponentiate and k is the exponent, with k_i denoting the ith bit of k.

The approach based on matrix exponentiation faces certain technical difficulties. To efficiently utilize the multiplication module described above, complex data movement and storage are required. For large matrices (on the order of magnitude of about 1000×1000), these make the hardware implementations increasingly complex, making it difficult to modify the design according to needs.

5. Approach based on polynomial exponentiation

To solve the above issue, a different approach is considered, which can be implemented in hardware in a much simpler and more elegant way. The algorithm detailed in this section and the FPGA implementation of it are the current direction of the research.

5.1. Mathematical background

For the recurring sequence construction we have to find irreducible polynomials in $\mathbb{Z}/2\cdot\mathbb{Z}$.

This can be done (see e.g. [1], Corollary 14.35) by applying iterative modular composition.

Lemma 5.1. *A polynomial $Q(x) \in \mathbb{F}_2$ of degree k is irreducible if and only if*

1. $Q(x)|x^{2^k} - x$ *and*

2. $\gcd\left(x^{2^{k/t}} - x, Q(x)\right) = 1$ *for all prime divisors t of k.*

To check property **1.** one has to compute

$$x^{2^k} \mod (Q(x), 2)$$

and if the result is x, then **1.** holds.
To check property **2.** one has to compute

$$R_t(x) = x^{2^{k/t}} \mod (Q(x), 2)\ , \text{ for all prime } t|k\ .$$

If $\gcd(R_t(x) - x, Q(x)) = 1$ for all t, then **2.** holds.

Lemma 5.2. *Let $Q(x) \in \mathbb{Z}[x]$ be a polynomial of degree k and irreducible with maximal order* $\bmod 2$ *and let $P(x) \equiv (x^2 - 1)Q(x) \mod 2$.*

The impulse response sequence corresponding to $P(x)$ has period length $4\cdot ord(Q)$ $\bmod 4$ *if and only if $P(x) \not| \left(x^{2\cdot ord(Q)} - 1\right) \mod 4$.*

To check property the above property, one has to compute

$$x^{2\cdot\mathrm{ord}(Q)} \mod (Q(x), 4)$$

and if the result is not equal to 1, then the property holds. Assume

$$x^{\mathrm{ord}(Q)} \equiv 2\cdot Q_1(x) + Q_0(x) \mod (Q(x), 4)\ ,$$

where $Q_0(x)$ and $Q_1(x)$ has coefficients 0 and 1.

Then

$$\begin{aligned} x^{2\cdot\mathrm{ord}(Q)} &= \left(x^{\mathrm{ord}(Q)}\right)^2 \\ &\equiv (2\cdot Q_1(x) + Q_0(x))^2 \\ &\equiv 4\cdot Q_1^2(x) + 4\cdot Q_1(x)Q_0(x) + Q_0^2(x) \\ &\equiv Q_0^2(x) \mod (Q(x), 4) \end{aligned}$$

This means, that it is enough to know

$$Q_0(x) \equiv x^{\mathrm{ord}(Q)} \mod (Q(x), 2) \ ,$$

and then from this the final power can be computed.

The advantage of the above simplification is that the same powering and reduction method can be used as during the irreducibility test.

Since polynomial reduction is very simple componentwise XOR operation on vectors of the polynomial coefficients, thus it can be highly parallelized on FPGA devices.

5.2. Algorithm for constructing a sequence with uniform distribution

- **Step 1** Choose modulo 2 irreducible $Q(x) \in \mathbb{Z}[x]$ of degree k.

- **Step 2** Calculate

  $$P(x) \equiv (x+1)^2 Q(x) \mod 2 \ ,$$
  $$P'(x) \equiv (x+1) Q(x) \mod 2$$

  and set

  $$\begin{aligned} P_1(x) &= P(x), \\ P_2(x) &= P_1(x) - 2, \\ P_3(x) &= P_1(x) - 2x, \\ P_4(x) &= P_1(x) - 2x - 2 \ . \end{aligned}$$

- **Step 3** Check which P_i is characteristic polynomial for the sequence $1, 1, \dots$ mod 4. Keep the two polynomials which fulfill the property. Assume, they are P_1 and P_2.

- **Step 4** Determine $\varrho = \mathrm{ord}(Q)$ modulo 2 and check whether $P_1 | x^\varrho - 1$ modulo 4 or not. If not, then P_1, otherwise P_2 is chosen for the characteristic polynomial.

- **Step 5** Choose initial values of the sequence. This can be done by the following: choose random $u_0, u_1, \dots, u_k$. Set these values as initial values of the linear recurring sequence with characteristic polynomial $P'(x)$. Compute the next element of the sequence u'_{k+1}. Find a random number u_{k+1} satisfying $u_{k+1} \not\equiv u'_{k+1} \mod 2$. The set $u_0, u_1, \dots, u_k, u_{k+1}$ are suitable initial values for the sequence.

The algorithm detailed above requires computations using only polynomials. In the hardware implementation, the polynomials are stored in shift registers. This greatly reduces the amount of data needed to be stored compared to the previous approach, as the space requirement of matrices is quadratic, and the space requirement of polynomials is linear. The polynomial reduction operation requires simple XOR gates. Data movement operations, that significantly increased the complexity of the modules in the previous approach, are much less resource-intensive using the method detailed in this section.

These improvements mean that linear recurring sequences of much larger period lengths can be created using the implementation of the polynomial approach.

6. Comparisons

The FPGA implementation found in [3] achieved a speed up factor of about ~ 200 over an efficient C++ implementation, and ~ 600 compared to a Matlab implementation of the same algorithm. The algorithm based on the newer approach provides further improvement. The computationally intensive step of the first algorithm, the matrix exponentiation described in the previous sections, on the FPGA implementation achieves a running time of ~ 0.6 seconds using matrices of size 896×896. The critical step of the second algorithm, the polynomial reduction, computes for ~ 0.014 seconds using polynomials of size 1021.

References

[1] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra* Cambridge University Press, (2003)

[2] T. Herendi, Construction of uniformly distributed linear recurring sequences modulo powers of 2 (201?)

[3] T. Herendi, R.S. Major, Modular exponentiation of matrices on FPGA-s *Acta Univ. Sapientiae, Informatica*, 3, 2 (2011) 172 - 191