Implementação em FPGA de Geradores de Números Pseudo-aleatórios sobre Anéis de Inteiros

Davi C. M. de Almeida, Carlos E. C. Souza, Daniel P. B. Chaves e Cecilio Pimentel

***Resumo*— Uma forma eficaz de projetar geradores de números pseudo-aleatórios é através de sistemas caóticos, fazendo-se uso do seu determinismo agregado à aparência aleatória de sua saída, a despeito da geração simples dessa. A implementação desses sistemas é facilitada quando são definidos sobre estruturas discretas adequadas, como anéis de inteiros. Neste trabalho propõe-se uma arquitetura otimizada para geradores de números pseudo-aleatórios baseados no mapa de Arnold sobre o anel $\mathbb{Z}_{3^m}$. Especifica-se uma unidade aritmética ternária empregando codificação binária. Por fim, compara-se a proposta com o uso do multiplicador modular de Montgomery, através da implementação de ambos em FPGA e da avaliação dos correspondentes consumos de hardware.**

***Palavras-Chave*— Sequências pseudo-aleatórias, algoritmo de Montgomery, caos discreto, FPGA.**

***Abstract*— An effective way to design pseudo-random number generators makes use of chaotic systems, taking advantage of their determinism associated with an output that looks random in spite of its easy generation. The implementation of these systems is facilitated when they are defined on suitable discrete structures, such as integer rings. In this work, we propose an optimized architecture for pseudo-random number generators based on Arnold's Cat map over the ring $\mathbb{Z}_{3^m}$. An arithmetic unit is specified using binary-coded ternary representation. Finally, the PRNG is implemented using the Montgomery modular multiplier, which is compared with the proposal through the implementation of both in FPGA and the evaluation of the corresponding hardware consumptions.**

***Keywords*— Pseudo-random sequences, Montgomery's algorithm, discrete chaos, FPGA.**

I. Introdução

Mapas caóticos vêm apresentando recorrente sucesso em aplicações como criptografia e comunicações [1]–[4]. Apesar disto, a dinâmica caótica gerada por mapas reais pode sofrer degradações causadas pelas operações de arredondamento típicas de unidades de hardware digitais [5]–[7] devido à resolução finita das unidades aritméticas [8]. Esse fenômeno implica potencialmente na deterioração das características originais das estatísticas vinculadas aos sistemas dinâmicos contínuos, sendo estas essenciais para as aplicações pretendidas. Em particular, o uso de mapas caóticos definidos sobre anéis de inteiros não apresenta essa desvantagem pois estão definidos sobre uma estrutura discreta. Quando apropriadamente projetados, estes mapas podem herdar algumas das propriedades estatísticas dos mapas caóticos reais. Desta forma, revelam-se como uma alternativa em aplicações criptográficas quando é exigido o uso de geradores de números pseudo-aleatórios (PRNG - *Pseudorandom Number Generator*).

Uma proposta de mapas caóticos definidos sobre estruturas discretas é apresentada em [9]. Nesse trabalho, os autores propõem um processo de discretização de mapas caóticos reais que culmina em uma dinâmica discreta periódica. Em sentido restrito, os sistemas obtidos por esse processo não apresentam caos. Contudo, as características observadas dentro de um período assemelham-se àquelas associadas a uma dinâmica caótica, sendo que, no limite em que o período vai ao infinito o mapa recupera as características do mapa original. Como exemplo, determina-se o expoente de Lyapunov, que é interpretado como a taxa de dispersão entre pontos vizinhos no conjunto discreto [9]. Este tipo de comportamento é denominado de caos discreto ou pseudo-caos [8], [9].

Os mapas caóticos definidos sobre anéis de inteiros constituem exemplos de mapas discreto [8], [10]–[12]. Destacam-se por permitirem a reprodução exata da dinâmica e por envolverem operações em ponto fixo, o que reduz sua complexidade computacional em relação aos mapas sobre os reais. A despeito da periodicidade das sequências geradas por esses mapas, são alternativas vantajosas em relação aos mapas reais, pois a escolha de anéis com cardinalidade suficientemente grande os torna essencialmente aperiódicos no contexto de algumas aplicações. Em [13], é proposta uma família de mapas caóticos discretos unidimensionais baseados no mapa de Arnold sobre $\mathbb{Z}_{3^m}$. Destaca-se nesses mapas a alta taxa de geração de bits quando utilizados no projeto de geradores de números pseudo-aleatórios. Quando representado por bits, cada amostra caótica (ou seja, um elemento de $\mathbb{Z}_{3^m}$) pode ser representada por $b = \lceil m \times \log_2(3) \rceil$ bits, obtendo-se uma taxa de $(b - 8)$ bits/amostra para formar a sequência do PRNG. Essa taxa contrasta com a observada em alguns trabalhos na literatura que empregam mapas reais, nos quais é utilizada aritmética em ponto flutuante e a taxa adotada é de 1 bit/amostra [6], [14].

O desafio no caso de mapas sobre anéis $\mathbb{Z}_{3^m}$ está na implementação dessas unidades, já que a aritmética modular não é a nativa dos dispositivos de hardware. Esses dispositivos são mais eficientes para aritméticas módulo uma potência de dois. Podemos destacar duas estratégias para a implementação de PRNGs sobre $\mathbb{Z}_{3^m}$: customizar unidades aritméticas ternárias ou aplicar algoritmos rápidos para operações modulares. No primeiro caso emprega-se codificação binárias de trits (dígitos ternários) [15], assim a representação numérica é ternária e o processo de redução modular é trivial. O segundo caso aplica o algoritmo de multiplicação modular de Montgomery (MMM - *Montgomery Modular Multiplication*) para reduzir a complexidade computacional na implementação das operações

Os autores são do Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife-PE, Recife-PE, e-mails: {davi.moreno, carlos.ecsouza, daniel.chaves, cecilio.pimentel}@ufpe.br. Este trabalho foi parcialmente financiado pelo CAPES, CNPq e FACEPE.

de multiplicação sobre $\mathbb{Z}_{3^m}$ empregadas no PRNG. Neste trabalho, ambas são implementadas e o resultado é comparado. Em particular, o algoritmo MMM implementado é a versão clássica [16], já que há diversas customizações voltadas para aplicações criptográficas específicas [17]–[20], em particular, voltadas para sistemas criptográficos sobre curvas elípticas e, portanto, definidos sobre corpos finitos e não sobre anéis.

Há três contribuições principais neste trabalho. Primeiramente, definimos uma unidade aritmética que dispensa a realização de operações de divisão para determinar os resíduos provenientes das somas e produtos realizados em $\mathbb{Z}_{3^m}$. Desenvolvemos uma unidade multiplicativa em aritmética ternária empregando codificação binária dos dígitos ternários (BCT, *Binary-Coded Ternary*); para reduzir sua complexidade e tempo de resposta, adota-se representação em árvore [21] e computam-se previamente os produtos de um dos operandos pelos dígitos ternários não nulos. Por fim, o consumo de recursos de uma FPGA (*Field-Programmable Gate Array*) pela arquitetura proposta é comparado com o consumo gerado pela implementação do algoritmo MMM, que é a forma clássica de implementar multiplicação modular.

As demais seções deste trabalho estão organizadas como segue. Na Seção II abordamos o mapa de Arnold discreto e suas aplicações, além do algoritmo MMM. A Seção III é dedicada à apresentação da arquitetura proposta neste trabalho. Na Seção IV explanamos a implementação em FPGA da proposta e do algoritmo de Montgomery e trazemos as comparações entre esses. Por fim, na Seção V tecemos as conclusões.

II. Preliminares

A. O mapa de Arnold discreto

O mapa de Arnold discreto (DACM - *Discrete Arnold's Cat Map*) é a aplicação $\Gamma : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q \times \mathbb{Z}_q$ definida por

$$\Gamma(x, y) = (2x + y, x + y) \pmod q, \tag{1}$$

em que $q = p^m$ é uma potência de primo e $\mathbb{Z}_q$ é o anel de inteiros módulo q. Neste trabalho adotaremos $p = 3$, em decorrência das propriedades já conhecidas do DACM sobre o anel $\mathbb{Z}_{3^m}$ [10], [13]. A aplicação iterativa de Γ a partir de uma condição inicial $(x_0, y_0) \in \mathbb{Z}_{3^m} \times \mathbb{Z}_{3^m}$ gera uma sequência bidimensional $s = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots\}$ em que (x_n, y_n) é definido recursivamente como

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pmod{3^m}. \tag{2}$$

Portanto, ao denominarmos

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix},$$

pode-se determinar qualquer ponto da sequência a partir da condição inicial e da matriz $\mathbf{A}$ pela relação

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \mathbf{A}^n \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \pmod{3^m}.$$

O período do DACM é definido como o menor expoente T para o qual $A^T \equiv \mathbb{I}$, em que $\mathbb{I}$ é a matriz identidade [10]. Portanto, segue a equivalência $(x_{n+T}, y_{n+T}) \equiv (x_n, y_n) \pmod q \; \forall n$. Podendo ser demonstrado pela relação entre a sequência de Fibonacci e o DACM, que o período deste é $T = 4 \times 3^{m-1}$ [11], [13].

1) Sequências-z: A sequência-z é uma estrutura discreta unidimensional $\{z_n\} = \{z_0, z_1, z_2, \ldots\}$, $z_n \in \mathbb{Z}_{3^m}$, tal que

$$z_n \triangleq x_n y_n \pmod{3^m},$$

em que (x_n, y_n) é o par obtido pela n-ésima iteração do DACM a partir de uma condição inicial $(x_0, y_0) \in \mathbb{Z}_{3^m} \times \mathbb{Z}_{3^m}$. Como demonstrado em [13, Teorema 1], desde que (x_0, y_0) não gerem a sequência nula e 3 não seja fator de x_0 e y_0, então o período de $\{z_n\}$ é $T_z = 2 \times 3^{m-1}$, sendo este o período máximo de uma sequência-z gerada a partir do DACM sobre $\mathbb{Z}_{3^m}$.

2) PRNG baseado no DACM: Para obter um PRNG precisamos inicialmente gerar sequências de período virtualmente infinito para a aplicação pretendida. Portanto, escolhemos x_0, y_0 de forma que T_z possua valor máximo e m seja suficientemente grande. Cada símbolo z_i da sequência $\{z_n\}$ é representado em binário por um bloco de comprimento

$$b = \lceil m \times \log_2 3 \rceil. \tag{3}$$

A saída do PRNG é formada pela concatenação do subconjunto de bits extraídos dessas representações binárias. As propriedades estatísticas da saída do PRNG são analisadas com a bateria de testes NIST versão SP800-22 [22]. Foram testadas 10^2 subsequências de comprimento 10^6 cada, empregando nível de confiança $\alpha = 0,01$, como recomendado em [22].

Ao empregar-se os b bits por amostras caóticas $z_i \in \{z_n\}$ para compor a sequência de saída do PRNG, não se obtém aprovação na bateria de testes NIST [13]. Contudo, obtém-se aprovação com o emprego de $(b-8)$ bits/amostra para qualquer valor de m em (3); portanto, quanto maior o valor de m maior a taxa de geração de bits por amostra caótica do PRNG. Os melhores resultados nos testes ocorrem com o descarte dos três bits menos significativos e dos cinco bits mais significativos.

B. Algoritmo MMM

Neste trabalho, o algoritmo MMM é empregado como referencial para avaliar a arquitetura proposta quanto à complexidade de implementação em hardware. O algoritmo implementado é baseado na descrição disponibilizada em [16], com o correspondente diagrama apresentado na Fig. 1. Observa-se no diagrama à esquerda que o primeiro passo é calcular as formas de Montgomery de x e y, sendo representadas por X e Y. Feito isso, a forma de Montgormery do produto entre x e y módulo q é calculada por $Z = XYR^{-1} (\mathrm{mod}\ q)$. Para realizar essa redução, emprega-se o diagrama à direita, em que a redução módulo q é substituída pela redução módulo R. Tipicamente, escolhe-se R da forma 2^k, $k \geq 2$, e $R > q$; assim, a redução módulo R de um inteiro $a_0 + a_1 \times 2^1 + a_2 \times 2^2 + \cdots + a_t \times 2^t$ resume-se a descartar os bits a_j, tal que, $j \geq k$. Observa-se que a conversão de Z para z também faz uso dessa redução módulo simplificada.

Este ponto é propício para enfatizarmos a estratégia da proposta dos autores. Assim como no algoritmo MMM, em que os operandos foram reescritos em um formato adequado para realizar a redução módulo 2^k, $k \geq 2$, na arquitetura

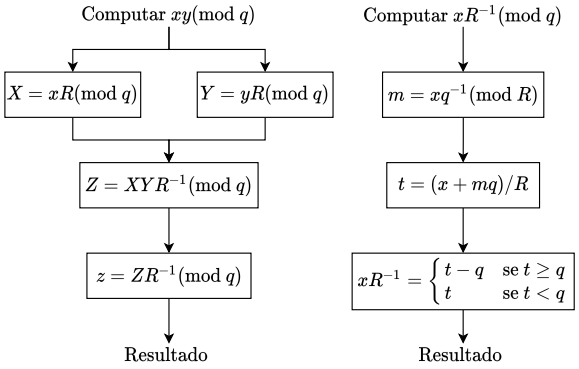

Fig. 1: Diagrama do algoritmo MMM.

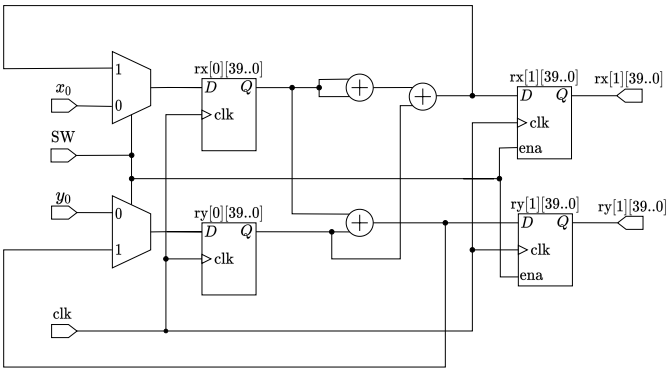

Fig. 2: Implementação do DACM sobre $\mathbb{Z}_{3^m}$ com codificação BCT.

proposta emprega-se a representação ternária para simplificar a operação de redução módulo 3^m, $m \geq 2$. Assim como no algoritmo MMM, no caso ternário a redução do número $b_0 + b_1 \times 3^1 + b_2 \times 3^2 + \ldots + b_t \times 3^t$ módulo 3^m resume-se a descartar os trits b_j para $j \geq m$.

III. Arquitetura proposta do PRNG sobre $\mathbb{Z}_{3^m}$

O PRNG proposto é formado pela concatenação de três unidades que operam em pipeline, de forma que a cada ciclo de relógio um bloco de $(b-8)$ bits é disponibilizado na saída do PRNG.

A. Implementação do DACM

A primeira unidade implementa o DACM sobre $\mathbb{Z}_{3^m}$ empregando codificação BCT. Neste caso, como exemplo, os valores ternários 21 e 22 são representados em hardware como 1001 e 1010, respectivamente, em que cada trit é associado a um par de bits, conforme a Tabela V do Apêndice. Enquanto $21+22 = 120$ em ternário, ao empregarmos a codificação binária dos trits teremos $1001 + 1010 = 011000$, conforme a Tabela III do Apêndice. Semelhante ao que ocorre na codificação BCD.

O RTL desta unidade é apresentado na Fig. 2 e implementa a expressão iterativa (2). Inicialmente, a coluna de MUX's seleciona os valores x_n, y_n para os quais x_{n+1}, y_{n+1} serão calculados, em que $x_n = (x_{n0}, \ldots, x_{n(m-1)})$ e $y_n = (y_{n0}, \ldots, y_{n(m-1)})$ são representados como uma sequência de m trits codificados em binário. Se $n = 0$ então $\mathsf{sw} = 0$ e x_1, y_1 é calculado para a condição inicial. Caso $n > 0$ então $\mathsf{sw} = 1$ e a saída é calculada a partir dos valores realimentados para as entradas 1 dos MUX's. Observe que esses valores são mantidos estáveis pelos registradores $\mathsf{rx}[0], \mathsf{ry}[0]$ e serão as saídas desses, assim como dos registradores $\mathsf{rx}[1], \mathsf{ry}[1]$, na próxima borda de subida do relógio. Esse ciclo se repete, com novos valores de x_n, y_n sendo calculados todas as vezes que as saídas dos registradores $\mathsf{rx}[0], \mathsf{ry}[0]$ são atualizadas.

B. Multiplicador de valores no formato BCT

O bloco multiplicador sobre $\mathbb{Z}_{3^m}$ recebe as entradas T1, T2 e T1x2 de m trits cada, ou seja, $2m$ bits já que estão codificadas em BCT. A multiplicação dos valores nas entradas T1 e T2 tem como resultado um número com até $2m$ trits, porém como a multiplicação é módulo 3^m, só os m trits

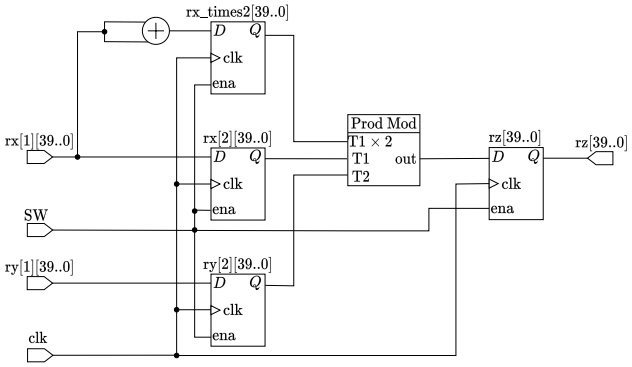

Fig. 3: RTL do Multiplicador ternário com codificação BCT.

menos significativos são necessários. Por isso, a arquitetura do multiplicador é simplificada por só requerer a implementação do circuito aritmético que calcule os m trits menos significativos do produto $x_n y_n (\mathrm{mod}\ 3^m)$. O valor na entrada T1x2 corresponde ao dobro do valor na entrada $T1$, sendo calculado previamente e representa o produto do valor em T1 pelo trit 2. As entradas $T1$ e $T2$ recebem os valores x_n e y_n, respectivamente. Dessa forma, a saída é determinada por

$$\sum_{i=0}^{m-1} y_{ni} \times (x_{n0} + \cdots + x_{n(m-1-i)} 3^{m-1-i})\, 3^i, \qquad (4)$$

em que $x_n = x_{n0} + x_{n1}\, 3^1 + \cdots + x_{n(m-1)}\, 3^{m-1}$ e $y_n = y_{n0} + y_{n1}\, 3^1 + \cdots + y_{n(m-1)}\, 3^{m-1}$ são as representações ternárias de x_n e y_n. Todos os termos do somatório (4) estão disponíveis previamente na unidade '`Prod Mod`', cuja saída é o produto $x_n y_n (\mathrm{mod}\ 3^m)$. Internamente essa unidade é implementada utilizando somadores ternários, com codificação BCT, dispostos em uma configuração de árvore binária [21], de forma a reduzir o número de estágios somadores de m (implementação direta) para $\lceil \log_2(m) \rceil$. Devido ao atraso reduzido obtido com a configuração em árvore, é possível realizar as operações nesta unidade em menos de um ciclo de relógio.

C. Conversor do formato BCT para o binário

A etapa de multiplicação fornece o resultado no formato BCT, sendo necessário a conversão para binário e posterior

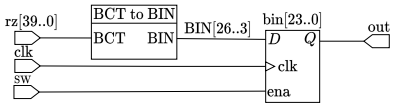

Fig. 4: RTL do bloco conversor do formato BCT para o binário.

extração dos bits que formam a sequência do PRNG. O conversor foi gerado através de uma modificação no código gerado pelo superotimizador proposto por Henry Massalin [23] para o conversor do formato BCT para o binário. No RTL apresentado na Fig. 4 essa conversão é realizada pelo bloco "BCT to BIN", cujo código é descrito em linguagem Verilog no ALGORITMO 1. A cada pulso de relógio um novo valor em formato BCT (proveniente do estágio de multiplicação) adentra o conversor, com sua representação em base 2 disponível na saída após alguns ciclos de relógio. O número exato de ciclos para que a conversão seja concluída depende do parâmetro m. No entanto, para qualquer valor adotado, o conversão é implementado em pipeline. Dessa forma, a cada ciclo de relógio a saída é atualizada com a representação binária de um produto $x_{n-j}y_{n-j} (\mathrm{mod}\ 3^m)$, em que j está associado ao valor m e corresponde ao número de ciclos de relógio necessários para concluir uma conversão do formato BCT para o binário. Após estarem disponíveis na saída do conversor, os bits que formam a sequência do PRNG são a entrada do registrador `bin[23..0]`, os quais, no próximo ciclo de relógio, encontram-se acessíveis na saída do PRNG.

IV. Implementação em FPGA

A arquitetura proposta do PRNG descrita na Seção III é implementada em uma FPGA Cyclone® IV EP4CE115F29C7 da Intel®, para os anéis de inteiros $\mathbb{Z}_{3^{20}}$ e $\mathbb{Z}_{3^{25}}$. Contudo, pode ser facilmente escalonável para outros valores de m em $\mathbb{Z}_{3^m}$. As mesmas condições são empregadas para implementar o algoritmo MMM. A Tabela I apresenta o consumo de elementos lógicos, registradores e multiplicadores de 9 bits para o PRNG proposto.

1) Implementação do algoritmo MMM: Para avaliar o consumo de hardware da proposta, adota-se a comparação com uma arquitetura clássica para o PRNG. O algoritmo MMM é utilizado como unidade de multiplicação para o PRNG baseado no DACM. Para o melhor aproveitamento do MMM, sabe-se que o cálculo da forma de Montgomery dos operandos só deve ocorrer, preferencialmente, no início de uma sequência

ALGORITMO 1: Conversor BCT para Binário.

```
// Converter BCT of at most 32 trits to Binary
bct_to_bin( bct ) {
acc = bct;

acc = acc-(((acc>> 2) & 0x3333333333333333)*(2**2-3**1));
acc = acc-(((acc>> 4) & 0x0F0F0F0F0F0F0F0F)*(2**4-3**2));
acc = acc-(((acc>> 8) & 0x00FF00FF00FF00FF)*(2**8-3**4));
acc = acc-(((acc>>16) & 0x0000FFFF0000FFFF)*(2**16-3**8));
acc = acc-(((acc>>32) & 0x00000000FFFFFFFF)*(2**32-3**16));

return acc;}
```

TABELA I: Recursos de hardware para a implementação do PRNG em $\mathbb{Z}_{3^m}$ com aritmética ternária para $m = 20$ e $m = 25$.

Recurso de Hardware	$m = 20$	$m = 25$
Elementos Lógicos (LE)	2439	3834
Registradores	377	530

TABELA II: Recursos de hardware para a implementação do PRNG em $\mathbb{Z}_{3^m}$ utilizando o algoritmo MMM com aritmética binária para $m = 20$ e $m = 25$.

Recurso de Hardware	$m = 20$	$m = 25$
Elementos Lógicos (LE)	2865	4881
Registradores	252	316

de operações que envolva necessariamente uma multiplicação modular. Uma vez que, o cálculo da forma de Montgomery é custosa computacionalmente. Portanto, o PRNG com o algoritmo MMM já é iniciado com os valores $x_0R \,(\mathrm{mod}\ 3^m)$ e $y_0R \,(\mathrm{mod}\ 3^m)$. Na implementação do DACM, como as operações são unicamente somas, a saída da unidade correspondente já se encontra na forma de Montgomery. Assim, os valores $x_nR(\mathrm{mod}\ 3^m)$ e $y_nR(\mathrm{mod}\ 3^m)$ ficam disponíveis na entrada do bloco multiplicador na n-ésima iteração do PRNG; que após alguns ciclos de relógio gera a saída z_n. Os bits que formam a saída do PRNG são selecionados de z_n, como descrito na Seção II-A. A taxa de geração de bits do PRNG é adotada como fator de normalização para a comparação. Portanto, assim como a arquitetura proposta, neste caso gera-se uma amostra caótica por ciclo de relógio através da implementação em pipeline do PRNG baseado no MMM. A Tabela II apresenta o consumo de recursos desta implementação para $m = 20$ e $m = 25$, os quais correspondem a blocos de bits de comprimento 32 bits e 40 bits, respectivamente.

2) Comparação dos algoritmos: Comparando as Tabelas I e II, observa-se que tanto para $m = 20$ quanto para $m = 25$ o PRNG proposto apresenta menor consumo de elementos lógicos, enquanto o PRNG com o MMM consome menos registradores. Contudo, como ressaltado na Seção II-A, para que o DACM possua período máximo é necessário que $3 \nmid x_0$ e $3 \nmid y_0$. Para o PRNG proposto isso é verificado observando se o trit menos significativo da representação BCT é 01 ou 10. Já para o PRNG com MMM, até onde é de conhecimento dos autores, não há uma forma tão direta de fazer essa verificação, o que requer o cálculo da forma de Montgomery de valores x_0 e y_0 para os quais se verifica previamente que não são divisíveis por três. Como o cálculo da forma de Montgomery consome muito hardware, em comparação as demais unidades do PRNG, a etapa de verificação traz uma vantagem para PRNG proposto.

V. Conclusões

Este trabalho aborda o projeto em FPGA de um PRNG baseado no mapa DACM sobre o anel de inteiros $\mathbb{Z}_{3^m}$. O

período da sequência $\{z\}$ utilizada nesse PRNG é de $2\times3^{m-1}$, ou seja, possui um crescimento exponencial com o número de trits necessários para a representação ternária dos elementos do anel. Assim, para $m = 25$ o período da sequência $\{z\}$ é de $T_z = 564.859.072.962$, já o do PRNG deve considerar o número de bits extraídos por amostra caótica, elevando o período para $(b - 8) \times T_z = 18.075.490.334.784$, em que $b = 40$. Observa-se que com valores modestos de m obtém-se implementações com longos períodos. Isso torna esta proposta viável para plataformas de baixo consumo de energia e de baixo custo como a FPGA Intel® MAX® 10 10M04; ainda, competitiva para aplicações criptográficas (ver [13]), tendo como atributos uma modesta exigência de hardware e sendo de implementação simples.

APÊNDICE

As tabelas utilizadas para implementar as operações aritméticas de soma e produto na arquitetura proposta são apresentadas nas Tabelas III e IV. Essas são baseadas nas tabelas apresentadas em [15] para operações balanceadas, aqui adaptadas para operações ternárias não balanceadas.

TABELA III: Tabela do somador ternário.

Sum	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

C_{out}	0	1	2
0	0	0	0
1	0	0	1
2	0	1	1

TABELA IV: Tabela do multiplicador ternário.

Mult	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

C_{out}	0	1	2
0	0	0	0
1	0	0	0
2	0	0	1

Para implementar a aritmética ternária em FPGA, a qual manipula bits, emprega-se a codificação BCT na Tabela V.

TABELA V: Codificador BCT.

Trit sem sinal	BCT
0	00
1	01
2	10

REFERÊNCIAS

[1] M. Preishuber, T. Hütter, S. Katzenbeisser, and A. Uhl, "Depreciating motivation and empirical security analysis of chaos-based image and video encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 9, pp. 2137–2150, Sept. 2018.

[2] N. A. Loan, N. N. Hurrah, S. A. Parah, J. W. Lee, J. A. Sheikh, and G. M. Bhat, "Secure and robust digital image watermarking using coefficient differencing and chaotic encryption," *IEEE Access*, vol. 6, pp. 19 876–19 897, Mar. 2018.

[3] K. W. Wong, Q. Lin, and J. Chen, "Simultaneous arithmetic coding and encryption using chaotic maps," *IEEE Trans. Circuits and Syst. II: Exp. Briefs*, vol. 57, no. 2, pp. 146–150, Feb. 2010.

[4] D. P. Chaves, C. E. Souza, and C. Pimentel, "A smooth chaotic map with parameterized shape and symmetry," *EURASIP Journal on Advances in Signal Processing*, no. 122, pp. 1 – 10, Nov. 2016.

[5] Z. Hua, B. Zhou, and Y. Zhou, "Sine chaotification model for enhancing chaos and its hardware implementation," *IEEE Trans. Ind. Electron.*, vol. 66, no. 2, pp. 1273–1284, Feb. 2019.

[6] M. Garcia-Bosque, A. Pérez-Resa, C. Sánchez-Azqueta, C. Aldea, and S. Celma, "Chaos-based bitwise dynamical pseudorandom number generator on FPGA," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 1, pp. 291–293, Jan. 2019.

[7] J. V. Evangelista, J. A. Artiles, D. P. Chaves, and C. Pimentel, "Emitter-coupled pair chaotic generator circuit," *AEÜ - International Journal of Electronics and Communications*, vol. 77, pp. 112 – 117, July 2017.

[8] B. Chirikov and F. Vivaldi, "An algorithmic view of pseudochaos," *Physica D: Nonlinear Phenomena*, vol. 129, no. 3, pp. 223 – 235, May 1999.

[9] L. Kocarev, J. Szczepanski, J. M. Amigo, and I. Tomovski, "Discrete chaos-I: Theory," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 53, no. 6, pp. 1300–1309, June 2006.

[10] F. Chen, K. Wong, X. Liao, and T. Xiang, "Period distribution of generalized discrete Arnold cat map for $N = p^e$," *IEEE Trans. Inf. Theory*, vol. 58, no. 1, pp. 445–452, Jan. 2012.

[11] F. Chen, K. Wong, X. Liao, and T. Xiang, "Period distribution of the generalized discrete Arnold cat map for $N = 2^e$," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 3249–3255, May 2013.

[12] B. Yang and X. Liao, "Some properties of the logistic map over the finite field and its application," *Signal Processing*, vol. 153, pp. 231 – 242, Dec. 2018.

[13] C. E. C. Souza, D. P. B. Chaves, and C. Pimentel, "One-dimensional pseudo-chaotic sequences based on the discrete arnold's cat map over Z3m," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 491–495, Jan. 2021.

[14] R. Lan, J. He, S. Wang, Y. Liu, and X. Luo, "A parameter-selection-based chaotic system," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 3, pp. 492–496, March 2019.

[15] B. Parhami and M. McKeown, "Arithmetic with binary-encoded balanced ternary numbers," in *2013 Asilomar Conference on Signals, Systems and Computers*, 2013, pp. 1130–1133.

[16] R. E. Blahut, *Fast Algorithms for Signal Processing*. Cambridge University Press, 2010.

[17] A. A. Abd-Elkader, M. Rashdan, E.-S. A. Hasaneen, and H. F. Hamed, "Advanced implementation of Montgomery Modular Multiplier," *Microelectronics Journal*, vol. 106, p. 104927, Dec. 2020.

[18] A. A. H. Abd-Elkader, M. Rashdan, E.-S. A. M. Hasaneen, and H. F. A. Hamed, "FPGA-Based Optimized Design of Montgomery Modular Multiplier," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, pp. 2137–2141, Jun. 2021.

[19] B. Li, J. Wang, G. Ding, H. Fu, B. Lei, H. Yang, J. Bi, and S. Lei, "A High-performance and Low-cost Montgomery Modular Multiplication Based on Redundant Binary Representation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2021.

[20] S. R. Pillutla and L. Boppana, "Low-complexity bit-serial sequential polynomial basis finite field GF(2^m) Montgomery multipliers," *Microprocessors and Microsystems*, vol. 84, p. 104053, Jul. 2021.

[21] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, ser. Signals and Communication Technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[22] L. E. B. III *et al.*, *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Gaithersburg, MD, United States: Nat. Inst. Std. & Technol., 2010.

[23] H. Massalin, "Superoptimizer: A look at the smallest program," in *Proceedings of the Second International Conference on Architectual Support for Programming Languages and Operating Systems*, ser. ASPLOS II. Washington, DC, USA: IEEE Computer Society Press, 1987, p. 122–126.