SeMicro-PR 2022

Estudo e Implementação de um TRNG em Hardware

Leonardo Pelanda Jarek, Sibilla Batista da Luz França
Departamento de Engenharia Elétrica , Curitiba, Brasil
Universidade Federal do Paraná (UFPR)
leopelanda4@gmail.com

***Resumo*—Números aleatórios são amplamente utilizados em criptografia, por este motivo, o estudo de geradores de números verdadeiramente aleatórios continua sendo um tema em ascensão. Neste trabalho o objetivo é apresentar uma arquitetura para um gerador verdadeiramente aleatórios (TRNG), baseado em anéis osciladores, com anéis controlados, buscando aumentar a aleatoriedade da sequência gerada, operando em frequências mais elevadas. Os geradores baseados em anéis osciladores exploram o fenômeno do jitter das portas lógicas inversoras. A arquitetura proposta adiciona um controle que faz o chaveamento do anel, ligando e desligando a oscilação. Para implementação do TRNG foi utilizada uma FPGA Stratix II, realizando a síntese com a linguagem de descrição de hardware VHDL, no software Quartus II. As sequências de bits extraídas foram validadas pelo software estatístico NIST, que por meio de vários testes verifica se a sequência de bits produzida pelo gerador é satisfatoriamente randômica.**

***Palavras-chave*—TRNG, FPGA, Anéis osciladores, VHDL, NIST.**

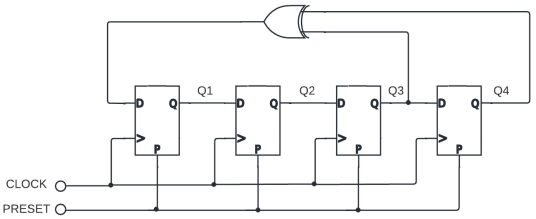
I. Introdução

Os números aleatórios têm grande importância para a computação, sendo utilizados em diversas aplicações. Em sistemas de criptografia os números aleatórios são amplamente utilizados como geradores de chaves confidenciais, chaves simétricas, chaves públicas e como fontes de senhas. Em todas essas aplicações, a segurança depende fortemente da aleatoriedade da fonte [1].

Os números aleatórios podem ser classificados como pseudo-aleatórios ou verdadeiramente aleatórios. A característica que diferencia a aleatoriedade é derivada da forma que os geradores operam. Números aleatórios são obtidos a partir dos geradores de números pseudo-aleatórios (*Pseudo Random Number Generators* - PRNGs) ou de geradores de números verdadeiramente aleatórios (*True Random Number Generators* - TRNGs).

PRNGs possuem uma característica cíclica e determinística, justamente por esse princípio se tornam frágil em sistemas de segurança. A alternativa mais eficiente é usar um número que possui uma fonte de aleatoriedade física e indeterminada, para não haver uma tendência de repetição dentro da sequência de números.

Um PRNG pode ser implementado utilizando um LFSR (*Linear Feedback Shift Register*) e portas xor [2]. A partir deste circuito, como exemplificado na figura 1, obtém-se uma sequência pseudo-aleatória, que após todas as combinações possíveis passa a repeti-las.



{15, 7, 3, 1, 8, 4, 2, 9, 12, 6, 11, 5, 10, 13, 14, 15, ... ,}

Figura 1. Exemplo de LFSR e sua sequência gerada.

Os TRNGs dependem de fenômenos físicos para extrair a fonte de aleatoriedade. Entre as fontes de entropia que trazem esse comportamento estão o ruído térmico, ruído de diodos, turbulência em discos rígidos, flutuação em conversores A/D e *jitter* [1].

Os geradores utilizados neste trabalho baseiam-se em anéis osciladores e são implementados em FPGA (*Field Programmable Gate Array*) Stratix II. Para a síntese desses geradores é utilizado a linguagem de descrição de hardware VHDL.

II. TRNG baseado em anéis osciladores

Um TRNG baseado em anéis osciladores usa o efeito de *jitter* das portas inversoras para assim produzir uma oscilação interna em uma frequência indeterminada, a FPGA é capaz de reproduzir esse efeito internamente, como demonstrado por [3]. A figura 2 ilustra o modelo simples do anel oscilador, proposto por [4].

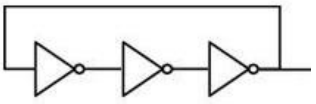


Figura 2. Estrutura básica de um anel oscilador.

O *jitter* é a característica física que gera a aleatoriedade nos anéis osciladores. Esse fenômeno pode ser descrito como a variação no tempo de transição dos níveis de tensão em uma porta lógica [5].

Modelos propostos na literatura constroem geradores com anéis osciladores, como este mostrado na figura 3, contendo múltiplos anéis em paralelo de forma que a oscilação é combinada através de portas XOR, e assim a sequência de bits é gerada e registrada.

Na implementação feita por Sunar et al [6], foram utilizados anéis assíncronos. Em um trabalho posterior, realizado por [4], um *flip-flop* do tipo D foi adicionado a saída de cada anel oscilador, melhorando a qualidade dos números gerados, permitindo que o gerador trabalhe em uma frequência mais elevada e tornando desnecessária a etapa de pós-processamento.

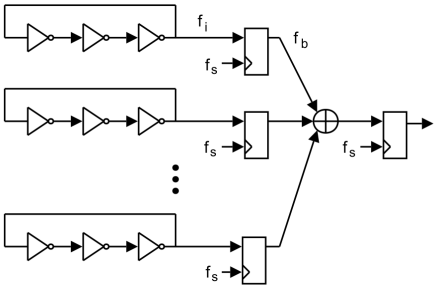


Figura 3. TRNG com registradores nos anéis [4].

Em 2016, [5] propôs uma arquitetura de TRNGs que utilizava registradores na árvore de XOR (Figura 4). Desta maneira, aumentou a performance do gerador, de modo a produzir números aleatórios em 100 MHz.

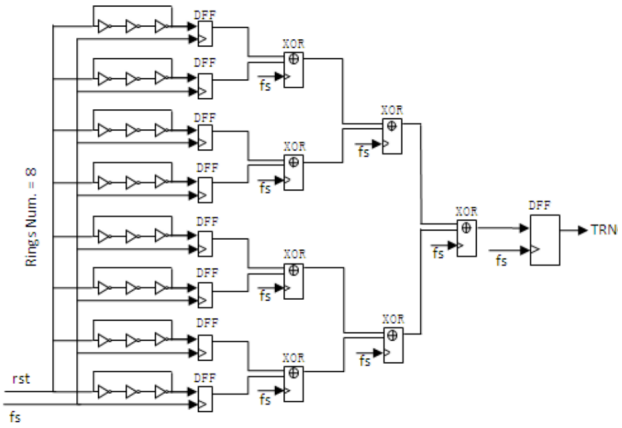


Figura 4. TRNG com árvore de xor síncrona. [4]

As arquiteturas propostas por [7] adotam os conceitos de [5], modificando a ordem e os conjuntos entre os anéis osciladores. Em uma de suas arquiteturas [7] adiciona um anel oscilador na saída do outro anel. No primeiro estágio cada anel oscilador é inicializado com um sinal de reset, e no segundo, o nó da saída do *flip-flop* D e da saída do anel é revezado a cada ciclo de *clock*. Com esse conjunto de anéis em série, foi possível gerar sequências aleatórias em 200MHz. A figura 5, mostra essa arquitetura com anéis em série.

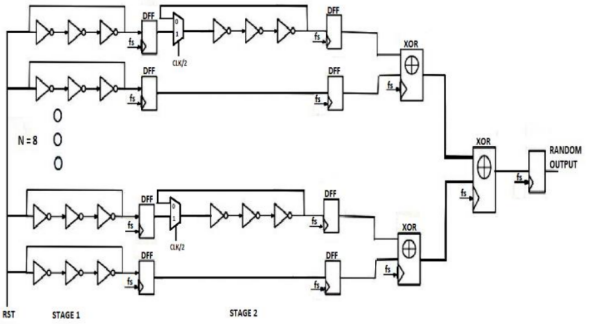


Figura 5. Implementação 1 de [7].

Um outro estudo apresenta uma modificação na estrutura do anel oscilador de modo a aumentar a aleatoriedade [8]. O conceito principal apresentado nesta abordagem é ligar e desligar o anel oscilador através da alteração do número de portas inversoras no anel. Quando for um número ímpar o anel oscila e quando for um número par o anel para de oscilar.

Em 2018, [9] utilizou desse princípio para implementar em um gerador do formato proposto por [4]. [9] passa a utilizar a nomenclatura de anéis osciladores com *shutdown* e *wake-up*, para o desligamento e acionamento do anel, respectivamente. No modo de *wake-up* a oscilação se comporta da maneira que mostra a figura 6. [10] usa a metaestabilidade para obter a aleatoriedade. O sinal *Vctrl* controla o anel, de modo a desligar a oscilação, nesse momento há o evento de clock, registrando o valor, que por sua vez, no momento do desligamento possui um valor indeterminado, capturando o fator de aleatoriedade.

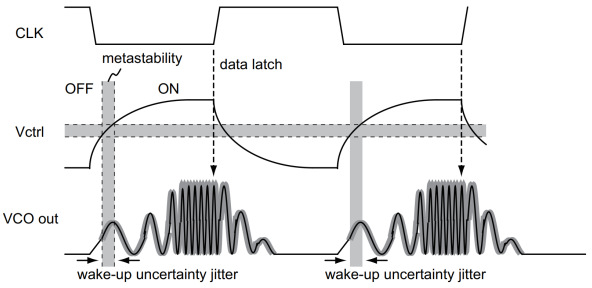


Figura 6. Comportamento do Wake-up ilustrado por [10].

III. Desenvolvimento e Resultados

O trabalho tem como objetivo propor uma nova arquitetura, utilizando a arquitetura proposta por [7], com adição dos anéis controlados de [8]. Diferentes modificações foram analisadas com intuito de obter o gerador que apresentasse o melhor desempenho em relação a aleatoriedade e frequência de operação.

Para construção dos geradores deste trabalho, foram utilizados anéis como representado na figura 7, utilizando 13 portas lógicas inversoras. Tal configuração mostrou-se a composição com o melhor resultado, da mesma forma que [9] utiliza.

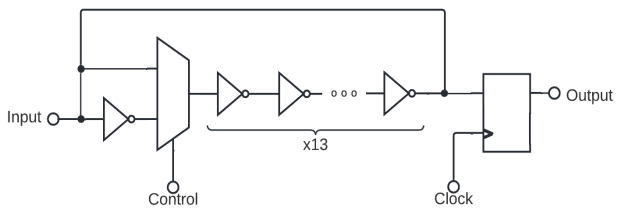


Figura 7. Anel oscilador com controle.

O anel é controlado por um sinal denominado *control*, que oscila na mesma frequência do clock, mas com *duty cycle* de 10%. Quando o sinal *control* está nível lógico alto o anel fica com um número par de portas inversoras e, portanto, deixa de oscilar. Para o *shutdown* a saída do anel é registrada no instante em que o anel para de oscilar, buscando capturar a instabilidade desse momento. No *wake-up*, o registro ocorre no instante em que o anel inicia a oscilação.

Diferentes arquiteturas foram analisadas baseando-se em [7] e incluindo os anéis controlados. Após avaliar através do teste NIST as várias sequências geradas, verificou-se que algumas não obtiveram sucesso nos testes, visto que não geravam uma sequência verdadeiramente aleatória.

O gerador desenvolvido foi baseado na implementação 1 (figura 5). Nos primeiros testes seus resultados mostraram que era capaz de gerar números aleatórios em 200 MHz. A fim de obter uma maior frequência de operação, algumas alterações foram feitas, chegando na arquitetura com os melhores resultados nos testes de aleatoriedade.

A arquitetura proposta para o TRNG está ilustrada na figura 8, simplificando o anel controlado da figura 7, em único bloco, anotando como *control***.** O gerador com o *wake-up* se mostrou melhor que o *shutdown*, e ainda utilizando o controle de anel apenas no primeiro estágio. O circuito é composto por 8 anéis controlados em paralelo conectados a 4 anéis simples (sem controle) intercalados. Cada anel utilizou internamente 13 inversores. A árvore de xor consistiu em 7 portas *xor* síncronas. A saída da última porta *xor* é registrada por um *flip-flop*, como mostra a figura 8. Este gerador produziu sequências binárias aleatórias operando a uma frequência máxima de 320 MHz. Essa arquitetura utilizou 184 ALUTs (*Adaptive Look-Up Table*) e 20 *Flip-Flops*.

As sequências geradas foram validadas pelo teste de aleatoriedade NIST. Para que este procedimento fosse realizado os números aleatórios gerados em simulação, na ferramenta Quartus II, eram salvos em um arquivo de formato *.tbl*, contendo os valores lógicos do gerador em cada instante de tempo da simulação. Utilizando um programa em C/C++, apenas os valores lógicos foram extraídos do arquivo *.tbl* para que a sequência fosse submetida aos testes de aleatoriedade.

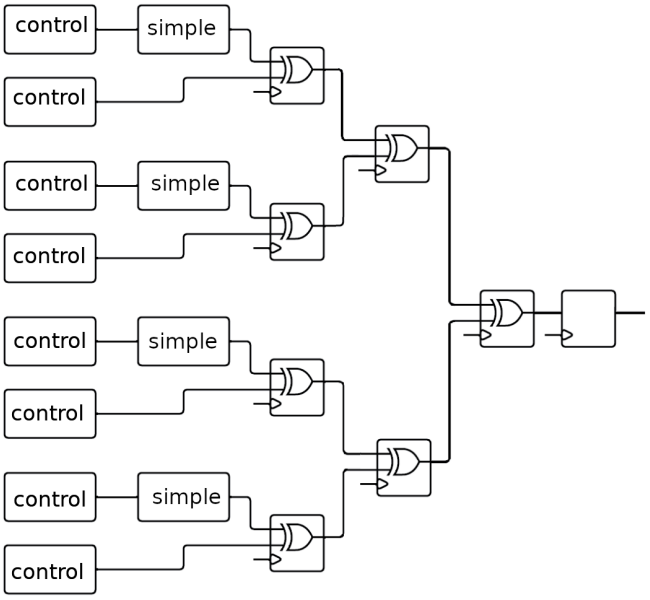
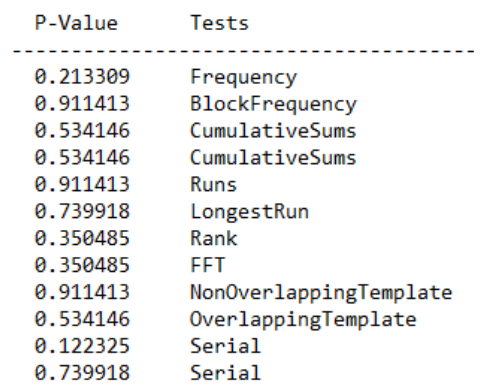


Figura 8. Gerador com controle de anel.

Para verificar a aleatoriedade das sequências numéricas geradas é usado o *software* estatístico NIST Test Suite 800-22. Os resultados dos testes feitos pelo *software* são apresentados em um relatório no qual são mostrados os P-values, indicando a probabilidade da sequência ser verdadeiramente aleatória. Dado a magnitude, quanto mais próximo de 1 o valor P for, maior é a densidade de probabilidade de que a sequência produzida pelo gerador seja satisfatoriamente randômica. Caso contrário, a sequência falha no teste, concluindo que o gerador não é um TRNG[11].

Para os testes realizados no NIST foram utilizados 10 blocos (bitstreams) de 10 Kbits cada, extraídos do gerador. Através deste procedimento diferentes arquiteturas para geradores, usando anéis controlados, foram analisadas. Durante a busca pelo melhor TRNG várias alterações foram realizadas como, a quantidade de inversores nos anéis, entre 3, 7, 13, 15 e 17. Em diversas frequências de 100 a 550 MHz. Com variações na quantidade de linhas, e na forma de aplicar o controle dos anéis, entre *shutdown* e *wake-up*. Alguns testes falharam, o motivo disso é que esses testes necessitam de uma quantidade muito maior de bits do que foi utilizado.

O gerador com o melhor resultado foi o gerador apresentado nas figuras 7 e 8, operando a frequência máxima de 320 MHz e com sucesso nos testes do NIST. Os resultados estão na figura 9, com todos os testes com valor-P acima de 0,01, demonstrando aleatoriedade na sequência binária gerada.



P-Value	Tests
0.213309	Frequency
0.911413	BlockFrequency
0.534146	CumulativeSums
0.534146	CumulativeSums
0.911413	Runs
0.739918	LongestRun
0.350485	Rank
0.350485	FFT
0.911413	NonOverlappingTemplate
0.534146	OverlappingTemplate
0.122325	Serial
0.739918	Serial

Figura 9. Resultado do P-value do Gerador II.

IV. CONCLUSÃO

Os geradores de números verdadeiramente aleatórios utilizados neste trabalho usam o *jitter* dos anéis osciladores como fonte de aleatoriedade. Na arquitetura proposta foram utilizados anéis controlados de modo a aumentar a aleatoriedade. O controle é dado por alteração no número de portas inversoras no anel. Quando for um número ímpar o anel oscila e quando for um número par o anel para de oscilar.

Diferentes estruturas para o TRNG foram testadas, alterando, por exemplo, número de anéis osciladores, números de inversoras no anel e frequência de operação, *shutdown* e *wake-up*. O gerador com o melhor resultado foi construído no formato *wake-up*, para o qual foi possível obter sequências binárias aleatórias em 320 MHz, superior ao gerador de [7] que operava em uma frequência máxima de 200 MHz. As sequências aleatórias geradas foram validadas pelo software NIST. Portanto, a arquitetura proposta se qualifica como um TRNG.

REFERÊNCIAS

[1] Sammy H. M. Kwok and Edmund Y. Lam. ”FPGA-based Highspeed True Random Number Generator for Cryptographic Applications”. IEEE, 2006.

[2] Pedroni, Volnei A.” Digital electronics and design with VHDL”, 2008.

[3] Fischer, Viktor & Drutarovský, Miloš. “True Random Number Generator Embedded in Reconfigurable Hardware”. Cryptographic Hardware and Embedded Systems”. CHES 2002.

[4] Knut Wold and Chik How Tan. “Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings”. International Conference on Reconfigurable Computing and FPGAs, 2008.

[5] Xiufeng Xu, Yuyang Wang. “High Speed True Random Number Generator Based on FPGA”. International Conference on Information Systems Engineering, 2016.

[6] Berk Sunar, William J. Martin, Douglas R. Stinson. “A provably secure true random number generator with built-in tolerance to active attacks”. IEEE Transaction Computers, 2007.

[7] Lima, E. , França, S. B. L.. “Gerador de Números Verdadeiramente Aleatórios Implementado em FPGA”. II Seminários de Microeletrônica do Paraná (SeMicro-PR), 2019.

[8] Varchola, M. “New Method of Randomness Extraction Based on a Modified Ring Oscillator for Cryptographic TRNGs Embedded in FPGAs”. 2009.

[9] M.A. Şarkışla and S. Ergün , “An Area Efficient True Random Number Generator Based on Modified Ring Oscillators” , 2018 IEEE Asia Pacific Conference on Circuits and Systems, Chengdu, 2018.

[10] T. Nakura, M. Ikeda and K. Asada, "Ring oscillator based random number generator utilizing wake-up time uncertainty," IEEE Asian Solid-State Circuits Conference, 2009.

[11] NIST Special Publication 800-22, Revision 1 ”A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,”Aug..2008.