PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós-Graduação em Engenharia Elétrica

Pedro Henrique de Oliveira Santos

GERADOR DE NÚMEROS PSEUDOALETÓRIOS PARAMETRIZÁVEL COMPOSTO POR AUTÔMATO CELULAR E LFSR: proposta e implementação em FPGA

Belo Horizonte
2017

Pedro Henrique de Oliveira Santos

GERADOR DE NÚMEROS PSEUDOALETÓRIOS PARAMETRIZÁVEL COMPOSTO POR AUTÔMATO CELULAR E LFSR: proposta e implementação em FPGA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Carlos Augusto Paiva da Silva Martins

Coorientadora: Prof.ª Dr.ª Flávia Magalhães Freitas Ferreira

Área de Concentração: Sistemas de Engenharia Elétrica e de Computação

Belo Horizonte
2017

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Santos, Pedro Henrique de Oliveira
S237g Gerador de números pseudoaletórios parametrizável composto por autônomo celular e LFSR: proposta e implementação em FPGA / Pedro Henrique de Oliveira Santos. Belo Horizonte, 2017.
69 f.: il.

Orientador: Carlos Augusto Paiva da Silva Martins
Coorientadora: Flávia Magalhães Freitas Ferreira
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica

1. Autômato celular. 2. Sistemas de controle por realimentação. 3. Algorítmos genéticos. 4. VHDL (Linguagem descritiva de hardware). 5. Computadores. 6. Simulação (Computadores). I Martins, Carlos Augusto Paiva da Silva. II. Ferreira, Flávia Magalhães Freitas. III. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDU: 621.38.032
Ficha catalográfica elaborada por Rosemary Socorro Hosken - CRB 6/3170

Pedro Henrique de Oliveira Santos

GERADOR DE NÚMEROS PSEUDOALETÓRIOS PARAMETRIZÁVEL COMPOSTO POR AUTÔMATO CELULAR E LFSR: proposta e implementação em FPGA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Sistemas de Engenharia Elétrica e de Computação

_____

Prof. Dr. Carlos Augusto Paiva da Silva Martins (Orientador) – PUC Minas

_____

Prof.ª Dr.ª Flávia Magalhães Freitas Ferreira (Coorientadora) – PUC Minas

_____

Prof. Dr. Omar Paranaíba Vilela Neto – UFMG

_____

Prof. Dr. Luís Fabrício Wanderley Góes – PUC Minas

Belo Horizonte, 24 de novembro de 2017

AGRADECIMENTOS

A Deus pelas forças para a caminhada de 2 anos.

A minha esposa Thais, meus filhos André e Laura por estarmos juntos e com amor, superando as incontáveis dificuldades que apareceram ao longo deste trabalho.

Aos meus pais Sady e Rosana e meu irmão Paulo, por todo o suporte que precisei e por acolher minha família. Em especial a meu pai por sempre me incentivar a começar e concluir o Mestrado.

Ao Prof. Carlos Augusto pela confiança nos meus resultados, por todas as horas que foram investidas e pelos ensinamentos.

À Prof. Flávia Magalhães pelo apoio no desenvolvimento e nas várias revisões de textos.

Aos amigos, colegas do PPGEE e trabalho, que de alguma forma, contribuíram para a conclusão dessa etapa de minha vida.

À PUC Minas pelo ensino humanizado e de qualidade e a todos professores que foram tutores em algum momento dessa caminhada.

À FAPEMIG pelo investimento na bolsa de estudos.

A todos que de forma direta e/ou indireta contribuíram para a conclusão desta pesquisa.

RESUMO

Esta dissertação apresenta uma implementação em FPGA de um gerador de números pseudoaleatórios (PRNG) através da combinação de um Autômato Celular (CA) e Registradores de deslocamento de com Realimentação Linear (LFSR) usando uma estrutura parametrizável. A bateria de testes Diehard foi utilizada para avaliar a qualidade do PRNG. O PRNG foi primeiramente parametrizado com base em trabalhos relacionados presentes na literatura. Depois de verificar a qualidade do gerador inicial, foram feitas alterações nos seus parâmetros visando melhor qualidade e velocidade. Uma exploração do espaço de projeto dos parâmetros do CA e do LFSR foi feita para avaliar centenas de configurações diferentes para o PRNG. Além disso, foi implementado um PRNG que usa o conceito de Hardware Evolutivo, de modo que o sistema é capaz de avaliar o PRNG e alterar seus parâmetros em tempo de execução para evoluir sua qualidade. Os resultados mostram que as combinações de mudanças em diferentes parâmetros podem aumentar ou diminuir o número de testes bem-sucedidos na ferramenta Diehard. O PRNG parametrizável foi capaz de ter melhor qualidade e velocidade do que os apresentados nos trabalhos relacionados após a alteração de seus parâmetros.

Palavras-chave: PRNG. FPGA. Autômato Celular. LFSR. Hardware Evolutivo.

ABSTRACT

This paper presents a FPGA implementation of a Pseudo-Random Number Generator (PRNG) by combining a Cellular Automaton (CA) and Linear Feedback Shift Register (LFSR) using a parametrizable structure. The Diehard battery of tests was used to evaluate the quality the PRNG. The PRNG was first parameterized based on related works present in the literature. After checking the quality of the initial generator, changes were made to its parameters aiming for better quality and speed. An exploration of the design space of the CA and LFSR parameters was done to evaluate hundreds of different configurations for the PRNG. Also, a PRNG that uses the concept of Evolvable Hardware was implemented, so the system can evaluate the PRNG and change its parameters at run time to evolve its quality. The results show that the combinations of changes on different parameters may increase or decrease the number of successful tests on the Diehard tool. The parameterizable PRNG was able to have better quality and speed than those presented in the related works after changing their parameters.

Keywords: PRNG. FPGA. Cellular Automata. LFSR. Evolvable Hardware.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ARM	Advanced RISC Machine
ASG	Alternating Step Generator
CA	Cellular Automaton (singular), Cellular Automata (plural)
CAM	Cellular Automata with Memory
CASR	Cellular Automaton Shift Registe
FIFO	First-in-First-out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
HPS	Hard Processor System
LFSR	Linear Feedback Shift Registers
LUT	Look Up Table
LUT-SR	Look Up Tables Shift Registers
MSE	Mean Square Error
PLD	Progammable Logic Devices
PRNG	Pseudo-Random Number Generator
SG	Shrinking Generator
SoC	System on a Chip
SPCA	Self-Programmable Cellular Automata

SUMÁRIO

1 INTRODUÇÃO

Números aleatórios são aqueles escolhidos por acaso dentro de uma distribuição específica, de modo que não exista correlação entre números sucessivos (WEISSTEIN, 2017). São essenciais em várias aplicações, como, por exemplo, algoritmos de otimização evolucionários, criptografia, redes neurais, simulações de Monte Carlo, entre outras (GUAN; TAN, 2004; ISHIMURA *et al.*, 2015; KHALAF; EL-KARIM; HAMED, 2016).

Os geradores de números aleatórios são classificados como “Verdadeiros” (*True Random Number Generators* - TRNG) ou “Pseudo” (*Pseudo-Random Number Generator* - PRNG). Enquanto os TRNGs utilizam fenômenos físicos para gerar um número aleatório, como ruídos de temperatura, radiação cósmica ou decaimento de partículas, os PRNGs utilizam uma fórmula que pode ser descrita matematicamente e, por isso, sempre geram a mesma sequência quando se utiliza um mesmo número inicial (semente). As sequências podem ser caracterizadas por sua qualidade (descrita na seção 1.1), tamanho e tempo demandado na geração de cada número pseudoaleatório variando para cada PRNG. Para avaliar a qualidade existem algumas ferramentas, como as baterias de testes Diehard (MARSAGLIA, 1996), TestU01 (L'ECUYER; SIMARD, 2007) e NIST SP800-22 (RUKHIN *et al.*, 2001).

Trabalhos presentes na literatura apresentam PRNGs que utilizam Autômatos Celulares (*Celullar Automaton*, no singular ou *Cellular Automata,* no plural - CA) (GONCU; YALCIN, 2014; MAITI; CHOWDHURY, 2017; MORENO-ARMENDÁRIZ *et al.*, 2013; SHACKLEFORD *et al.*, 2002) ou Registradores de Deslocamento com Realimentação Linear (*Linear Feedback Shift Registers* - LFSR) (ALFKE, 1996; KHALAF; EL-KARIM; HAMED, 2016; PAROL; DĄBAL; SZPLET, 2016; THOMAS; LUK, 2013). Há ainda pesquisas que utilizam uma combinação de CA e LFSR para criar um PRNG (CERDA *et al.*, 2012; TKACIK, 2002).

Os CA são sistemas que possuem grau de paralelismo total, isto é, todas as células podem ser processadas simultaneamente. Portanto, sua implementação em *Field Programmable Gate Array* (FPGA) é vantajosa, por ser um dispositivo capaz de processar cada célula paralelamente em cada Tabela Verdade (*Look Up Table* - LUT) do FPGA (CERDA *et al.*, 2012). Os LFSRs também já foram implementados em FPGAs utilizando LUTs (THOMAS; LUK, 2013).

O problema motivador deste trabalho é descobrir quais são os melhores conjuntos de valores para os parâmetros do CA, do LFSR e da integração deles para implementar um PRNG. E ainda, como as variações nesses parâmetros podem produzir ganhos ou perdas significativas de qualidade, tamanho da sequência e tempo de geração do número pseudoaleatório.

1.1 Objetivos

O objetivo deste artigo é explorar o espaço de projeto da combinação de parâmetros de CA unidimensionais e LFSRs através da construção de um PRNG parametrizável implementado em FPGA. A ferramenta DIEHARD será utilizada para mensurar a qualidade do PRNG, em termos da quantidade de testes avaliados com sucesso. Além da qualidade da sequência gerada, outras análises serão apresentadas. A primeira é o desempenho do PRNG, que será avaliado pelo tempo de geração de um número pseudoaleatório. E a segunda é a quantidade de elementos lógicos utilizados, uma vez que a demanda de um número pequeno de elementos lógicos é desejável por permitir que o PRNG parametrizável possa ser implementado em FPGAs simples e de menor custo.

1.2 Contribuição

Como contribuição, esta pesquisa propõe a criação de um PRNG que possa ter os parâmetros do CA e do LFSR alterados sem que seja necessária uma nova síntese do circuito. Serão utilizados critérios adicionais na avaliação da qualidade do PRNG, além dos resultados dos testes do *software* Diehard. São investigados também os ganhos de se utilizar os conceitos de *Hardware* Evolutivo (*Evolvable Hardware* - EHW) no PRNG parametrizável, de forma a avaliar a qualidade do PRNG e mudar sua configuração em tempo de execução.

1.3 Justificativa

Os trabalhos relacionados que serão apresentados no Capítulo 3, assim como outros trabalhos que utilizaram CA ou LFSRs em PRNGs, são compostos por estruturas semelhantes, mas com combinações de parâmetros diferentes, além de

serem fixas. O PRNG proposto é capaz de modificar esses parâmetros dinamicamente, o que o torna versátil para diversas aplicações. Por exemplo, alternar entre melhor qualidade e maior velocidade, à medida em que o tempo evolui.

1.4 Estrutura do trabalho

O restante desta dissertação está estruturado da seguinte maneira: o Capítulo 2 apresenta uma fundamentação teórica de CA e LFSR no contexto de PRNG e EHW, o Capítulo 3 mostra os resultados obtidos por trabalhos relacionados presentes na Literatura. O Capítulo 4 mostra a metodologia utilizada neste trabalho para os testes dos PRNGs. Os resultados obtidos estão divididos no Capítulos 4, 5 e 6, da seguinte maneira: o Capítulo 4 apresenta o PRNG parametrizável proposto, o Capítulo 5 mostra a exploração do espaço de parâmetros do PRNG parametrizável e o Capítulo 6 apresenta a combinação de EHW com o PRNG Parametrizável. O Capítulo 7 apresenta a conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

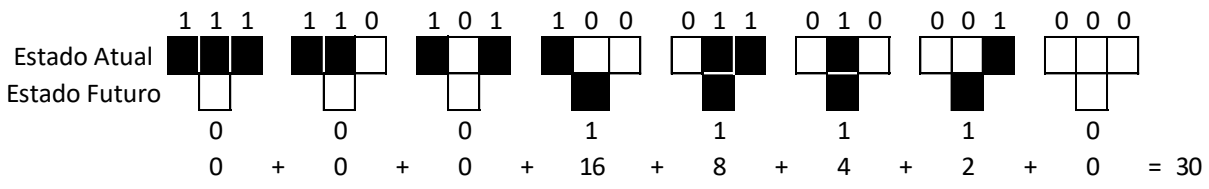
Para uma boa compreensão desta dissertação é necessário o entendimento de alguns conceitos básicos sobre CA, LFSR e EHW. A fundamentação teórica descreve alguns conceitos de CA e LFSR no contexto de geradores de números aleatórios e apresenta a definição de *Hardware* Evolutivo e seus diversos modelos.

2.1 Autômato Celular

Os Autômatos Celulares foram propostos na década de 1940 por Stanisław Ulam e John von Neumann como possíveis modelos para sistemas autorreplicáveis (VON NEUMANN; BURKS, 1966). Em seu livro, *New Kind of Science*, publicado na década de 1980, S. Wolfram apresenta uma coleção de resultados e descobertas do uso de autômatos celulares (WOLFRAM, 1986a).

Um CA é um conjunto de células em uma grade de forma geométrica específica, em n dimensões, que evolui em passos discretos, de acordo com uma série de regras que se baseiam no estado de células vizinhas. Grades unidimensionais são os modelos mais simples e serão utilizados neste trabalho. O número de estados (k) que uma célula pode assumir é tipicamente um número inteiro, sendo k=2 (binário) a escolha mais comum (HALBACH; HOFFMANN; RÖDER, 2004; MAITI; CHOWDHURY, 2017). O modelo de vizinhança, ou seja, quais células afetam uma célula genérica, é outro parâmetro de cada CA. O uso dos vizinhos mais próximos é o mais simples. Nele apenas células diretamente adjacentes a uma célula afetam seu estado futuro a cada passo. Uma célula e suas duas vizinhas formam uma vizinhança de 3 células, como mostrado na Figura 1, e por isso existem $2^3 = 8$ padrões possíveis para uma vizinhança. Há então 28 = 256 regras possíveis.

Figura 1 – Autômato Celular Unidimensional utilizando a Regra 30 para a determinação do estado futuro de vizinhanças de 3 células



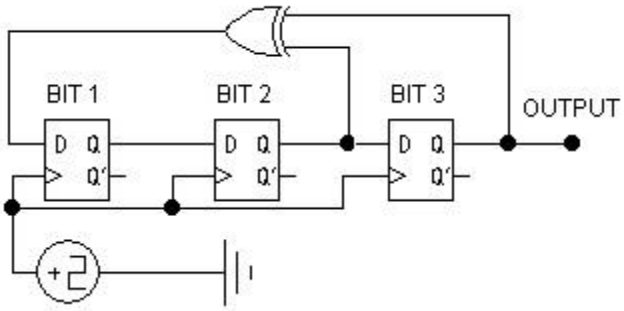
Fonte: Elaborado pelo autor

Nos autômatos celulares unidimensionais, as regras recebem como rótulo o número que representa, em binário, os estados futuros de uma célula. No caso da vizinhança de 3 células, portanto, os rótulos são números decimais correspondentes aos códigos de 8 bits, como também mostra a Figura 1 onde a linha superior apresenta o estado atual de uma célula (centro) e de seus vizinhos (esquerda e direita) e a linha inferior mostra o próximo estado da célula (no exemplo dado, o decimal 30, que corresponde a 00011110 em binário). Com regras e estruturas simples, um autômato celular é capaz de sequências de números pseudoaleatórios, como descrito por Wolfram (1986b).

2.2 Registradores de Deslocamento com Realimentação Linear

Os LFSRs são registradores de deslocamento onde algumas saídas de registradores selecionados, chamadas de *taps*, são direcionadas para uma porta lógica (normalmente uma XOR ou XNOR), que tem sua saída conectada na entrada do registrador de deslocamento, como mostrado na Figura 2. A localização dos *taps* determina o padrão gerado pelo LFSR, essa localização é chamada de polinômio do LFSR (PAROL; DĄBAL; SZPLET, 2016).

Figura 2 – Exemplo de um LFSR de 3 bits



Fonte: Elaborado pelo autor

Os n registradores do LFSR são denominados Q1 a Qn ou Bit 1 a Bit n (ALFKE, 1996). Qualquer LFSR gera uma sequência binária que é matematicamente previsível, mas pode ser considerada randômica na maioria das aplicações. A seleção dos *taps* determina o tamanho da sequência, ou seja, quantos valores podem ser gerados até que ela se repita. O maior tamanho de sequência que um LFSR pode gerar é $2^n - 1$, uma vez que a sequência que causaria um bloqueio (somente 0 para XOR e somente

1 para XNOR) é excluída. Um LFSR sempre tem um *tap* no registrador n, caso contrário ele se comportaria como um LFSR com menos registradores, de acordo com a posição do último *tap*. As combinações de *taps* que geram a maior sequência para LFSRs de 3 até 168 registradores são apresentadas por Alfke (1996).

2.3 *Hardware* Evolutivo

Hardware Evolutivo é um tipo de *hardware* que pode mudar sua arquitetura e comportamento dinamicamente e automaticamente através da interação com seu ambiente (SALVADOR, 2016). Ao contrário do *hardware* convencional, em que a estrutura é irreversível e definida no processo de design, o EHW é desenhado para se adaptar as mudanças nos requisitos de tarefas ou mudanças no ambiente através de sua habilidade de reconfigurar seu *hardware* (HIGUCHI *et al.*, 1999).

O projeto de um *Hardware* Evolutivo consiste de duas tarefas principais. A primeira apresenta a seleção de estruturas reconfiguráveis úteis. A segunda procura por uma configuração adequada para essa estrutura, ou seja, o sistema tem que resolver um problema de otimização. A busca pela melhor configuração é, na maior parte dos casos, feita por algoritmos evolucionários (BURIAN, 2009).

Um EHW pode ser classificado em duas categorias: extrínseco e intrínseco. EHW extrínseco simula uma evolução por *software* e apenas carrega a melhor configuração para o hardware em cada geração. Já o EHW intrínseco, simula a evolução diretamente no seu *hardware*. EHWs normalmente são implementados em Dispositivos Lógicos Programáveis (*Progammable Logic Devices* - PLD), como FPGAs, onde a arquitetura do PLD e sua função são determinados por uma série de bits que podem ser reconfigurados (HADDOW; TYRRELL, 2011).

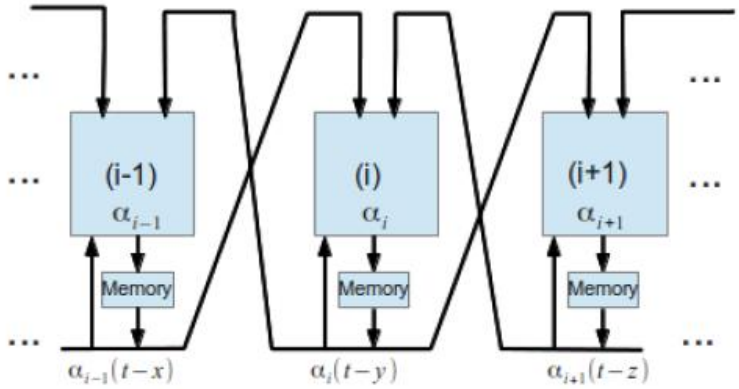
3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma pesquisa de vários trabalhos com temas relacionados a Autômatos Celulares, Registradores de Deslocamento com Realimentação Linear, PRNGs compostos por CAs e/ou LFSRs e Hardware Evolutivo em FPGA de medição não invasivos. A bateria de testes Diehard é utilizada em alguns desses trabalhos.

3.1 Autômato Celular com Memória

Em um autômato celular convencional, os estados futuros das células dependem apenas do estado atual das células vizinhas. No modelo de autômato celular com memória (*Cellular Automata with Memory* - CAM), os estados futuros dependem também dos estados anteriores (Figura 3). Um novo modelo de CAM para ser utilizado como gerador de números pseudoaleatórios foi proposto por Goncu e Yalcin (2014).

Figura 3 – Autômato Celular com Memória



Fonte: (GONCU; YALCIN, 2014)

O CAM foi implementado em um FPGA Spartan 6, onde a aleatoriedade do modelo pode ser analisada pela medida da sua entropia. No modelo proposto é possível modificar qual estado anterior será considerado para então analisar o efeito da memória na aleatoriedade. Os resultados mostraram que para obter valores elevados de entropia é necessário que apenas uma célula na vizinhança tenha memória. Foi observado também que se todas as células tiverem memória, o valor da entropia diminui.

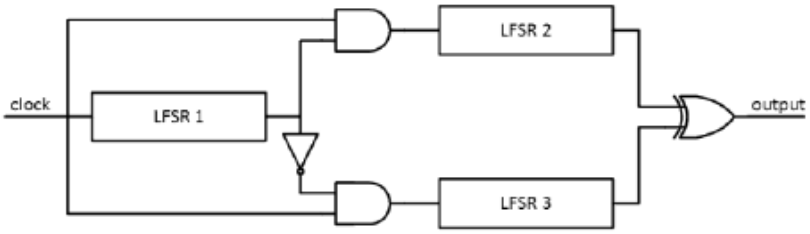
3.2 Gerador de Bits Aleatórios em FPGA

Parol, Dąbal e Szplet (2016) utilizaram a bateria de testes NIST SP800-22 para avaliar três Geradores de Bits Pseudoaleatórios, cada um composto por diferentes tipos de LFSR. Os geradores foram implementados em um FPGA Spartan 6 XC6SLX16 da Xilinx.

O primeiro é um LFSR puro, como apresentado no Capítulo 2, o segundo é chamado de Gerador de Passo Alternado (*Alternating Step Generator* - ASG), por fim, o terceiro é chamado de Gerador de Encolhimento (*Shrinking Generator* - SG).

O esquema do ASG é apresentado na Figura 4. Esse gerador consiste em três LFSRs. O LFSR1 controla os outros dois - LFSR2 e LFSR3. O bit de saída do LFSR1 determina qual dos os outros dois LFSRs podem executar a operação de mudança na saída do PRNG. Se a saída do LFSR1 é igual a 1, o LFSR2 é ativado, enquanto o LFSR3 não é. O LFSR1 é chamado de registrador de controle e os outros LFSRs são conhecidos como registradores geradores.

Figura 4 – Gerador de Passo Alternado



Fonte: (PAROL; DĄBAL; SZPLET, 2016)

O SG envolve dois LFSRs (A e B). A Figura 3 mostra a estrutura de um SG. A sequência gerada pelo LFSR A é usada para selecionar bits da sequência gerada pelo LFSR B que cria a sequência de saída. Se a saída do LFSR A é 1 o bit de saída do LFSR B é aceito, caso contrário este bit é descartado.

Figura 5 – Gerador de Encolhimento

Fonte: (PAROL; DĄBAL; SZPLET, 2016)

Os resultados obtidos dos testes mostraram que o gerador composto por um LFSR não passou nenhuma variante dos testes de complexidade linear. Foram observados resultados melhores para o SG, que falhou apenas um teste. O ASG passou todos os testes aplicados.

3.3 Autômatos Celulares com Super Regra

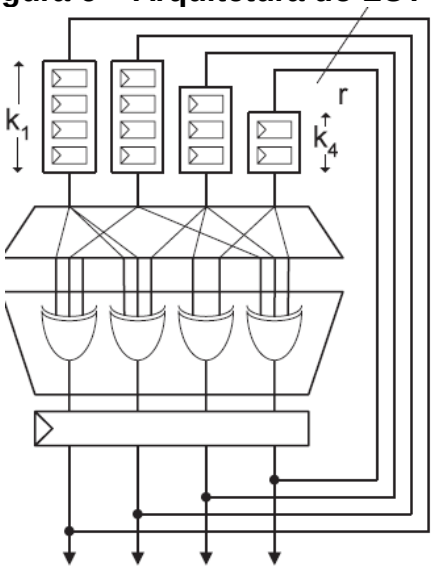
Guan e Tan (2004) propuseram uma nova classe de CA chamada *Self-Programmable Cellular Automata* (SPCA) onde cada célula do CA passa a considerar uma super-regra, que determina quando o SPCA deve fazer uma mudança dinâmica na regra vigente em cada célula do CA, baseado no estado de vizinhos até três células distantes.

A super-regra pode ser considerada um segundo CA trabalhando em paralelo com o CA principal, para evitar a ocorrência de padrões estáticos, que são indicativos de PRNGs de baixa qualidade. Os autores testaram o SPCA usando os pares de regras 90/165 e 105/150 (regra base / super regra). Para essas regras, SPCAs de 16 até 24 bits foram aprovados em toda a bateria de testes Diehard.

3.4 PRNGs com LUTs como Registradores de Deslocamento em FPGA

Thomas et al. (2013) descreveram em seu artigo uma família de geradores chamada *Look Up Tables Shift Registers* (LUT-SR) RNGs, que usa *Look Up Tables* como registradores de deslocamento (Figura 6) para obter qualidade alta e longos períodos (2^{1024}-1, para 32 bits), enquanto usa poucos recursos. Os autores também forneceram ferramentas e um ambiente de testes de código aberto para verificar os resultados obtidos.

Figura 6 – Arquitetura do LUT-SR



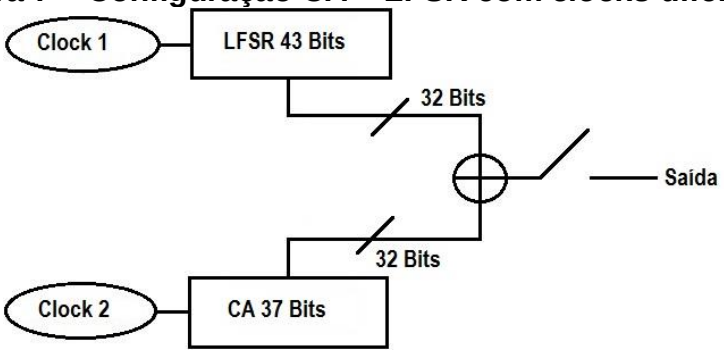
Fonte: (THOMAS; LUK, 2013)

A ferramenta de testes TestU01 foi utilizada para comprovar a eficácia do LUT-SR RNG. Essa ferramenta é um aprimoramento do Diehard, que não permitia adicionar testes ou modificar os parâmetros do teste. Os resultados mostraram que o LUT-SR foi capaz de passar nos testes *Crush* e *BigCrush* do TestU01. Quando comparado ao LUT-OPT, que utiliza o mínimo de recursos, e o LUT-FIFO, que gera longos períodos, o LUT-SR é um meio termo entre os dois.

3.5 PRNG em Hardware

Tkacik (2002) utilizou uma combinação de CA e LFSR para criar um PRNG. Em sua implementação, um *Cellular Automaton Shift Register* (CASR), um CA com os dois extremos interligados) de 37 bits e um LFSR de 43 bits são conectados a osciladores de *clocks* diferentes. São selecionados 32 bits paralelamente de cada estrutura, os quais são permutados e combinados através de uma porta XOR, como pode ser visto na Figura 7, onde o símbolo $\oplus$ representa a porta XOR.

Figura 7 – Configuração CA + LFSR com *clocks* diferentes



Fonte: Modificado de (TKACIK, 2002)

São escolhidos 32 bits de cada estrutura para gerar um PRNG com 1 bit de saída. A cada solicitação de um novo número aleatório, existe um tempo mínimo de amostragem que permite que ambas as estruturas recebam pulsos de *clock* de pelo menos o dobro de seu tamanho (86 pulsos de *clock* para o LFSR e 74 para o CA). O autor avaliou a qualidade de um PRNG composto apenas por um CA ou apenas por um LFSR e o PRNG composto pela combinação CA + LFSR. Os PRNGs que utilizaram CA ou LFSR isolados foram reprovados em diversos testes da bateria de testes Diehard. A combinação de CA + LFSR foi reprovada apenas em 15 de 216 testes, mostrando um melhor resultado que as estruturas isoladas.

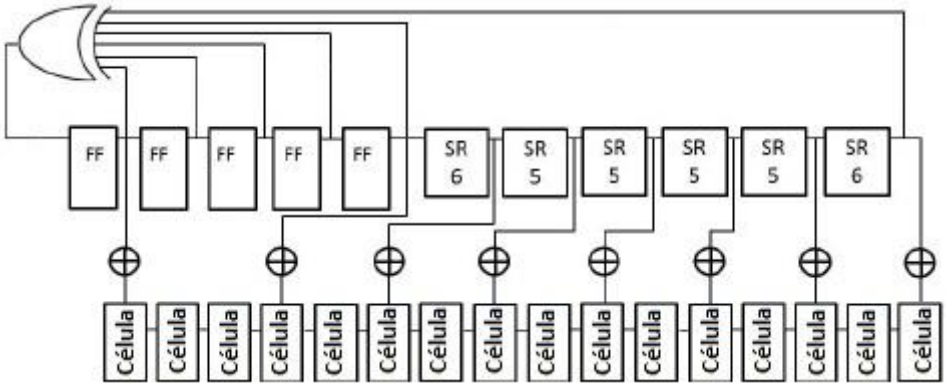
3.6 PRGNs utilizando LFSRs e CAs em FPGA

Cerda et al. (2012) utilizaram a bateria de testes Diehard para avaliar PRNGs construídos através de diferentes combinações de CA e LFSRs. Os autores implementaram os PRNGs em um FPGA Xilinx Spartan 3E a 50 MHz. Algumas combinações de modelos de CA com modelos de LFSR não foram aprovadas em todos os testes. Porém, foi possível obter configurações que satisfazem todos os testes.

O foco dos autores se deu na configuração CA 16 Bits + LFSR 37 Bits, que foi a configuração com resultados mais promissores entre as diversas testadas. Foram utilizadas conexões de 1, 2, 4 e 8 bits entre o CA e o LFSR. O CA de 16 Bits utilizou as regras 90 e 150, conforme esquema proposto por Hortensius *et al.* (1989), visando a obtenção de sequências de números pseudoaleatórios de tamanho máximo (2^n - 1). Os *taps* do LFSR foram conectados aos bits 37, 5, 4, 3, 2 e 1, de acordo com o

recomendado por Alfke (1996). A combinação de 8 bits pode ser vista na Figura 8, onde o símbolo ⊕ representa uma porta XOR.

Figura 8 – Configuração CA + LFSR com saída de 8 bits



Fonte: Modificado de (CERDA *et al.*, 2012)

Nos resultados apresentados pelos autores a combinação de CA + LFSR foi aprovada em toda a bateria de testes Diehard utilizando 1, 2, 4 e 8 bits gerados a cada pulso de *clock*.

3.7 *Hardware* Evolutivo Implementado em FPGA

Burian (2009) examinou a influência de parâmetros de algoritmos genéticos para o uso em EHW. Uma implementação deste algoritmo em FPGA foi realizada para dois exemplos práticos de *hardware* evolutivo, um circuito lógico combinacional e um filtro *Finite Impulse Response* (FIR) evolucionário.

Neste circuito, a evolução pode acontecer indefinidamente, permitindo que implemente várias funções ao longo do processo. A evolução começa a partir da escolha de uma nova função. Se a evolução encontrar uma configuração adequada, o sistema passa a executar a nova função. Porém, existem alguns problemas para esta técnica. O elemento chave destes sistemas é a função objetivo do algoritmo genético, cujo cálculo pode consumir tempo de processamento, fazendo com que a velocidade de evolução (busca por uma configuração) dependa especialmente dela.

Para o circuito lógico combinacional, o autor utilizou um sistema com quatro entradas e duas saídas, em que a tabela verdade era alterada e o *hardware* deve se alterar para executar a nova tabela. Para o filtro FIR, a resposta ao impulso é obtida através de evolução. Em um filtro FIR convencional, há uma entrada e uma saída de

dados, já o filtro evolucionário possuí uma entrada adicional, que é a entrada para amostras ideais na saída (Figura 9). A evolução deve encontrar uma resposta ao impulso que garante uma correspondência adequada entre o sinal de saída real e ideal.

Figura 9 – Filtro FIR Evolucionário

Fonte: Modificado de (BURIAN, 2009)

Com os resultados obtidos, o autor apresenta quatro recomendações para o uso de algoritmos evolucionários para evolução de *hardware* em FPGA: Utilizar algoritmos simples; empregar populações pequenas; não realizar cruzamento, pois não é eficiente; otimizar a função objetivo.

3.8 Considerações Finais

A Tabela 1 apresenta uma síntese dos temas tratados nos trabalhos relacionados e apresenta em sua última linha os temas tratados neste trabalho, para comparação.

Tabela 1 – Síntese dos Temas dos Trabalhos Relacionados

Trabalho Relacionado (Seção)	PRNG	Implementado em Hardware	CA	LFSR	Metodologia de Teste	EHW
3.1	X	X	X		Entropia	
3.2	X	X		X	NIST	
3.3	X	X	X		Diehard	
3.4	X	X		X	TestU01	
3.5	X	X	X	X	Diehard	
3.6	X	X	X	X	Diehard	
3.7		X				X
PRNG Parametrizável	X	X	X	X	Diehard	X

Fonte: Elaborado pelo autor

É possível observar na tabela que todos os trabalhos relacionados que utilizaram um PRNG tiveram sua implementação em *hardware*, mas nenhum deles

usou o conceito de EHW em sua implementação, que permite o PRNG evoluir e melhorar sua qualidade. A bateria de testes Diehard é a mais utilizada e será usada neste trabalho como critério de avaliação da qualidade do PRNG.

4 PRNG PARAMETRIZÁVEL

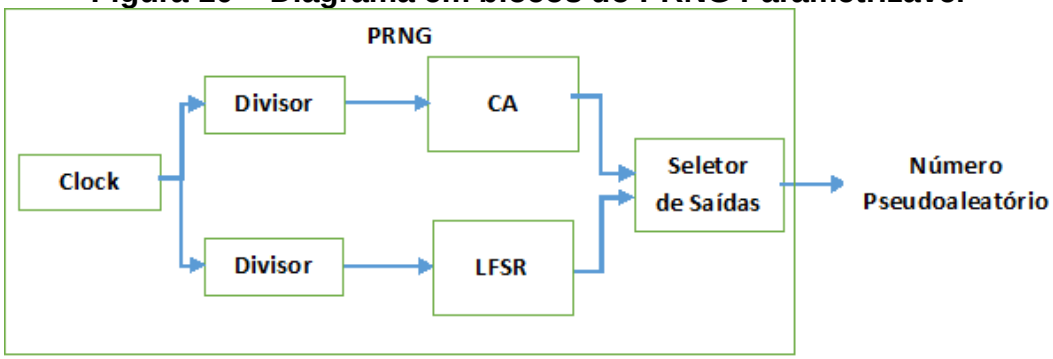
Os trabalhos relacionados apresentados no Capítulo 3, assim como outros trabalhos que utilizaram CA ou LFSRs em PRNGs, são compostos por estruturas semelhantes, mas com combinações de parâmetros diferentes. Diante desse cenário, este capítulo apresenta a proposta da criação de um PRNG que seja capaz de ser parametrizado em tempo de execução, ou seja, tempo de geração dos números pseudoaleatórios, para que sua qualidade pudesse ser avaliada e de acordo com o resultado, alterar novamente os parâmetros até a obtenção dos resultados desejados.

Neste trabalho, os experimentos ficaram restritos aos PRNGs implementados puramente com um CA ou com um LFSR, além da combinação de um CA com um LFSR.

4.1 Diagrama do PRNG Parametrizável

O diagrama em blocos da Figura 10 apresenta a estrutura do PRNG parametrizável proposto.

Figura 10 – Diagrama em blocos do PRNG Parametrizável



Fonte: Elaborado pelo autor

A arquitetura é composta por uma fonte de *clock* que é conectada a dois divisores de *clock*, que são sinais de entrada para o CA e para o LFSR, permitindo que o CA e o LFSR tenham *clocks* diferentes. Para as duas estruturas, são selecionados bits de acordo com uma quantidade especificada, os quais são combinados um a um através de portas XOR no bloco seletor de saídas que, então, produz um número pseudoaleatório com essa quantidade de bits a cada pulso de *clock*.

No PRNG parametrizável é possível configurar os seguintes parâmetros: o fator de divisão de cada *clock*, a regra e valor inicial de cada célula do CA (semente), o polinômio do LFSR que representa quais registradores serão conectados à porta lógica (*taps*) e o número de bits *n* que o PRNG gera a cada pulso de *clock*. Valores diferentes para *n* geram sequências diferentes, mesmo com uma mesma semente. Os bits selecionados são distribuídos o mais uniformemente possível ao longo do CA e do LFSR. Para alterar o tamanho do CA ou do LFSR é necessária uma nova síntese do circuito no FPGA.

Como contribuição adicional deste trabalho foi proposta uma alteração na estrutura do PRNG parametrizável, para que o Autômato Celular e o LFSR fossem conectados a sinais de *clock* diferentes, também parametrizáveis, mas sem a limitação do tempo mínimo de amostragem em que cada estrutura deve receber pulsos de clock de pelo menos o dobro de seu tamanho, diferentemente do modelo proposto por Tkacik (2002). Como consequência, a velocidade de geração de um novo número pseudoaleatório pelo PRNG aumenta e é determinada apenas pelo *clock* mais lento, ou seja, o que possui o fator de divisão de *clock* com maior valor.

4.2 Implementação em VHDL

A implementação do PRNG parametrizável em *hardware* foi dividida em quatro partes, ou entidades: os divisores de *clock*, o LFSR, o CA e o Seletor de Saídas.

Na entidade ClockDivider, que faz a divisão do *clock* principal, as entradas são o *clock* principal e o valor do divisor e a saída é o *clock* de saída, cuja frequência está dividida (Figura 11).

Figura 11 – Código VHDL da entidade ClockDivider

```
entity ClockDivider is
  port (
    clk_in  :in  std_logic;
    divisor :in  std_logic_vector (7 downto 0);
    clk_out :out std_logic
  );
end entity;
```

Fonte: Elaborado pelo autor

Para o LFSR, todos os registradores são processados em uma única entidade, pois é necessário fazer apenas duas operações: o deslocamento de bits e a operação

da porta lógica na realimentação. O LFSR possui como entradas os sinais de *clock* e de *reset*, além do polinômio para a determinação dos *taps*, como mostrado no trecho de código VHDL da Figura 12.

Figura 12 – Código VHDL da entidade LFSR

```
entity LFSR is
  generic(constant N: integer := 37);
  port (
    clk         :in  std_logic;
    reset       :in  std_logic;
    polynome    :in  std_logic_vector (N-1 downto 0);
    lfsr_out    :out std_logic_vector (N-1 downto 0)
  );
end entity;
```

Fonte: Elaborado pelo autor

A implementação do CA foi realizada com paralelismo total, ou seja, cada célula do autômato celular é processada individualmente. Com isso o cálculo do próximo estado do CA pode ser feito com apenas um pulso de clock. A Figura 13 apresenta o código em VHDL que representa uma entidade de uma célula individual. Cada célula recebe como entrada um sinal de *clock*, o *reset*, a semente quando é feito o *reset*, o estado das células vizinhas e a regra do CA a ser executada. Como saída tem-se o estado atual da célula. A primeira e a última célula são interligadas semelhantemente a um anel.

Figura 13 – Código VHDL da entidade CA1D

```
entity CA1D is
  generic(constant N: integer := 8);
  port (
    clk   :in  std_logic;
    reset :in  std_logic;
    init  :in  std_logic;
    west  :in  std_logic;
    east  :in  std_logic;
    rule  :in  std_logic_vector (N-1 downto 0);
    state :out std_logic
  );
end entity;
```

Fonte: Elaborado pelo autor

Na entidade OutputSelector, são entradas o clock mais lento (do CA ou LFSR), o número de bits a serem selecionados de cada estrutura, os bits do CA e os bits do LFSR. A saída são os bits gerados pelo PRNG. No trecho de código da Figura 14, por exemplo, o tamanho da saída é de 8 bits.

Figura 14 – Código VHDL da entidade OutputSelector

```
entity OutputSelector is
  generic(constant CA_SIZE:   integer := 16;
          constant LFSR_SIZE: integer := 16);
  port (
    clk    in  std_logic; -- slowest (CA/LFSR)
    n_bits in  integer range 1 to 8;
    ca     in  std_logic_vector (CA_SIZE-1   downto 0);
    lfsr   in  std_logic_vector (LFSR_SIZE-1 downto 0);
    rn     out std_logic_vector (7 downto 0)
  );
end entity;
```

Fonte: Elaborado pelo autor

4.3 Metodologia de Testes

Para utilizar a bateria de testes DIEHARD, é necessário que seja criado um arquivo com 11 MB (88 Mb) de dados gerados pelo PRNG (MARSAGLIA, 1996). Para gerar essa quantidade de dados pseudoaleatórios quando o PRNG gera 1 bit por vez, são necessários 88 milhões de pulsos de *clock*. A utilização de mais bits na saída do PRNG, por sua vez, diminuirá a quantidade de pulsos de *clock* necessária. Por exemplo, com 2 bits de saída são necessários 44 milhões de pulsos de *clock*.

O arquivo contendo a sequência de bits gerada pelo PRNG alimenta o *software* DIEHARD, que executa toda a bateria de testes e salva os resultados em um arquivo de texto. Para a análise desse arquivo de texto, foi escrito um código em Python para fazer seu *parser* e extrair todos os *p-values*, que são uma nota entre 0 a 1 de cada um dos 216 testes executados (MARTIN, 2002). Um *p-value* igual a 0 ou igual a 1 representa um teste reprovado (MARTIN, 2002). Em um PRNG ideal, os *p-values* estão distribuídos uniformemente entre 0 e 1, de forma que suas amostras constituam a reta passando pela origem, com inclinação de 45°, mostrada na Figura 15. Na avaliação da qualidade do PRNG, foi considerado o número de testes reprovados (MARTIN, 2002).

Como contribuição metodológica em relação aos trabalhos relacionados, foram ainda avaliados o coeficiente de correlação de Pearson (r^2) e o Erro Médio Quadrático (*Mean Square Error* - MSE) entre o PRNG testado e o PRNG ideal, que demonstram o quanto os *p-values* obtidos estão próximos do PRNG ideal. O coeficiente r^2 e o MSE foram escolhidos para se determinar o melhor PRNG para os casos que dois ou mais

deles apresentem número de testes reprovados bem similares. Ao se aproximar do PRNG ideal, o valor do MSE tende a 0 e o valor do r^2 tende a 1.

Para a obtenção dos resultados, foi utilizada a placa de desenvolvimento Altera DE2-70 com o FPGA Cyclone II EP2C70F896, o ambiente de desenvolvimento Quartus II e a linguagem VHDL. Os bits gerados pelo PRNG no FPGA são enviados via pinos de entrada e saída para uma segunda placa com um processador Cortex M4f, que gera o arquivo com os dados para serem utilizados no *software* DIEHARD.

Essa metodologia de testes será a mesma utilizada nos Capítulos posteriores.

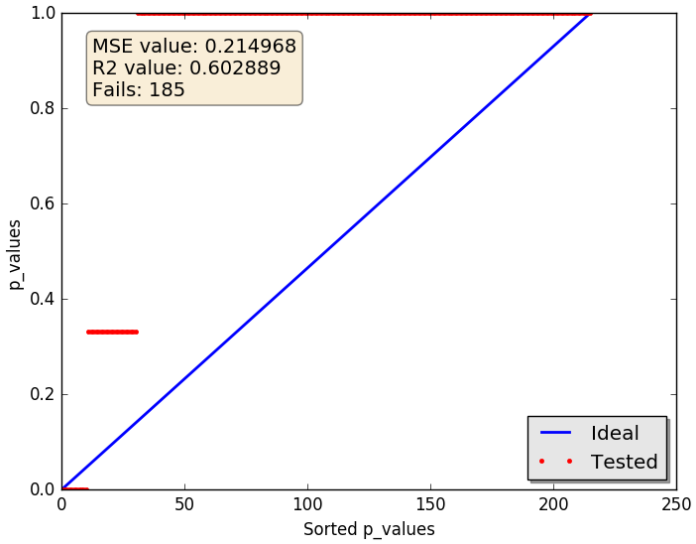
4.4 Resultados Experimentais

Os testes foram divididos de maneira a verificar se um PRNG composto por CA + LFSR é melhor que um composto pelas estruturas isoladas, tendo como base o modelo proposto por Cerda et al. (2012). Em seguida foram feitas mudanças nos parâmetros para tentar obter resultados melhores. A avaliação utilizará a quantidade de testes reprovados pelo DIEHARD (falhas) e as métricas MSE e r^2. O polinômio do LFSR foi mantido fixo em todos os testes, apesar de ser um parâmetro, e teve seu valor igual ao utilizado por Cerda et al. (2012), que tem taps nos registradores 37, 5, 4, 3, 2 e 1.

4.4.1 CA e LFSR Isolados

O primeiro teste realizado foi parametrizar o PRNG proposto para que ele fosse composto apenas por um CA de 16 bits ou apenas por um LFSR de 37 bits, e avaliar a sua qualidade. O resultado da avaliação da qualidade do PRNG parametrizado como um CA de 16 bits mostrou que a sequência de números pseudoaleatórios gerada pelo PRNG foi reprovada em 185 testes (aqueles que obtiveram nota 0 ou 1), apresentando MSE de 0.214968 e r^2 de 0.602889 (linha vermelha na Figura 15).

Figura 15 – Resultado dos testes do PRNG composto por CA de 16 bits comparados com o PRNG ideal

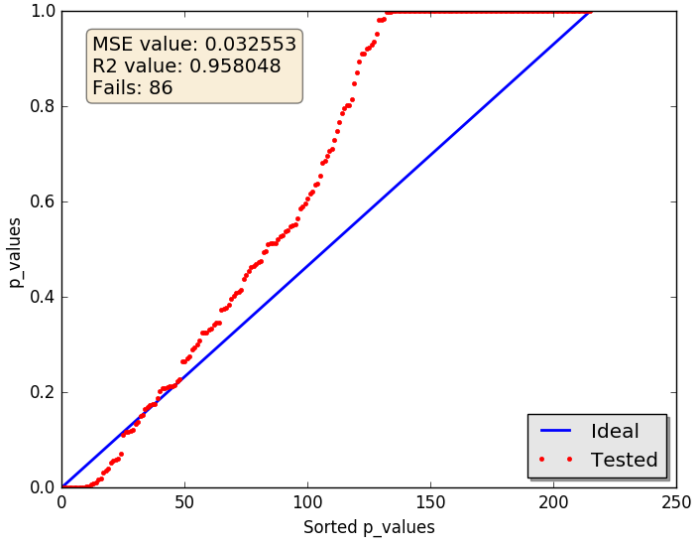


Fonte: Elaborado pelo autor

Os valores de MSE e r^2 estão distantes dos valores ideais, uma vez que as curvas constituídas pelas amostras de *p-values* do PRNG sob avaliação e do PRNG ideal diferem-se enormemente. Quase todos os pontos estão em 0 ou 1 (são as falhas) e alguns em aproximadamente 0,3. Isso indica que um CA de 16 bits isolado não é adequado para ser utilizado como um gerador de números pseudoaleatórios.

Para o PRNG parametrizado como um LFSR de 37 bits, a sequência foi reprovada em 86 testes, com MSE de 0.032553 e r^2 de 0.958048 (Figura 16).

Figura 16 – Resultado dos testes do PRNG composto por LFSR de 37 bits comparados com o PRNG ideal



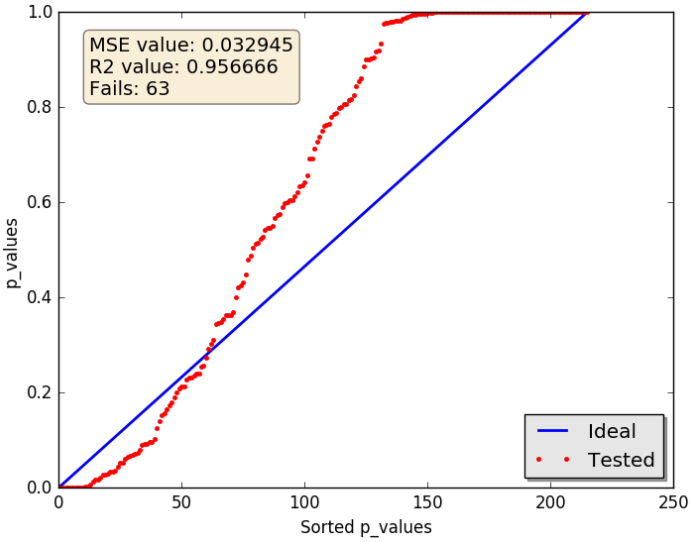
Fonte: Elaborado pelo autor

Apesar de a sequência ter sido reprovada, houve uma melhoria significativa em relação ao método anterior. Em relação ao PRNG com CA isolado, o número de testes reprovados diminuiu e as métricas MSE e r^2 melhoraram, mas ainda de forma insatisfatória. A curva constituída pelas amostras de *p-values* na Figura 16 ainda está distante da curva do PRNG ideal.

4.4.2 CA e LFSR Combinados

Em seguida o PRNG foi parametrizado de maneira a reproduzir exatamente o modelo proposto por Cerda et al. (2012). Nesse modelo, um autômato celular de 16 bits (regras 90/150) é combinado com um LFSR de 37 bits (*taps* em 37, 5, 4, 3, 2 e 1). Com essa configuração de CA e LFSR foram feitos testes com o seletor de saída configurado com 1 bit de saída e com 2 bits de saída. Para o caso de 1 bit, foi selecionada a célula 16 do CA e o registrador 37 do LFSR. Para o caso de 2 bits, foram selecionadas as células 8 e 16 do CA e os registradores 18 e 37 do LFSR. O resultado do teste mostrou que a sequência de números pseudoaleatórios foi reprovada em 63 testes com MSE de 0.032945 e r^2 de 0.956666 (Figura 17).

Figura 17 – Resultado dos testes da combinação CA de 16 bits + LFSR de 37 bits comparados com o PRNG ideal.



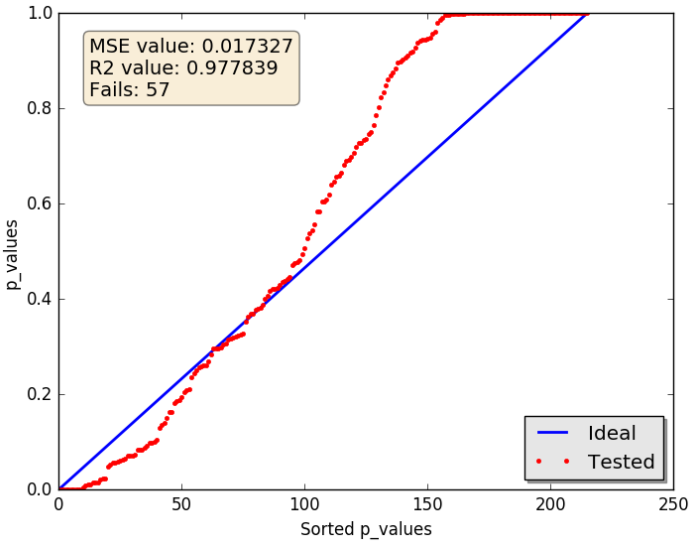
Fonte: Elaborado pelo autor

Com a combinação do CA e do LFSR o PRNG foi aprovado em um maior número de testes em relação ao CA e LFSR isolados, porém os valores de MSE e r^2

ficaram muito próximos dos valores obtidos pelo PRNG configurado com apenas um LFSR de 37 bits. A curva constituída pelas amostras de *p-values* ainda não se aproxima da curva do PRNG ideal. Em relação ao desempenho, o PRNG parametrizável gerou a sequência em um tempo menor que a implementação de Tkacik (2002), uma vez que não há tempo mínimo de amostragem para cada estrutura.

Para o PRNG parametrizado para reproduzir o modelo de Cerda et al. (2002), mas com o seletor de saída configurado com 2 bits, teve-se como resultado 57 testes reprovados, MSE de 0.017327 e r^2 de 0.977839, como pode ser visto na Figura 18.

Figura 18 – Resultado dos testes do PRNG CA de 16 bits + LFSR de 37 bits com saída de 2 bits comparados com o PRNG ideal.



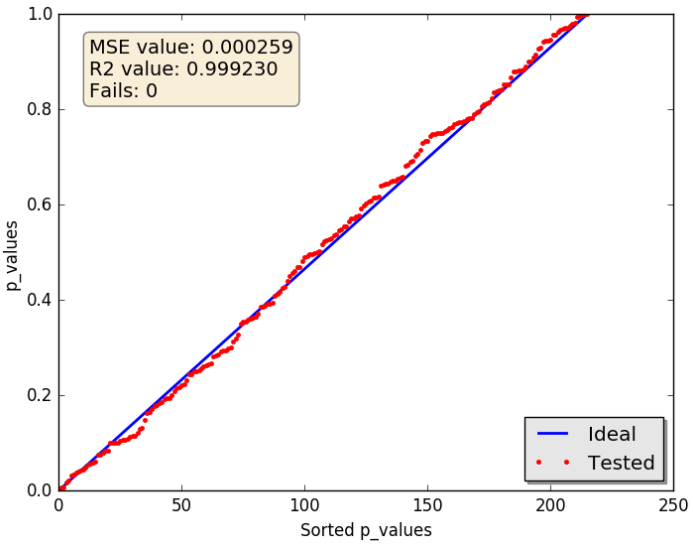
Fonte: Elaborado pelo autor

A utilização de 2 bits acelerou a geração da sequência de números pseudoaleatórios, pois como 2 bits são gerados simultaneamente, os 11MB de dados são obtidos na metade do tempo. O aumento da quantidade de bits de saída resultou na redução do número de testes reprovados, bem como na melhoria das métricas MSE e r^2, fazendo a curva de distribuição das amostras de *p-values* do PRNG testado aproximar-se ainda mais a curva do PRNG ideal.

4.4.3 CA e LFSR Combinados com Clocks Diferentes

A seguir, o PRNG proposto foi parametrizado novamente como o modelo de Cerda et al. (2012), com 1 bit de saída e, adicionalmente, foram configurados fatores de divisão de *clock* diferentes para o CA e o LFSR, sendo 2 para o CA e 3 para o LFSR. Desse modo, o PRNG parametrizável foi aprovado em toda a bateria de testes DIEHARD, como apresentado na Figura 19, onde nenhum teste falhou e obtiveram-se MSE de 0.000259 e r^2 de 0.999230.

Figura 19 – Resultado dos testes do PRNG CA de 16 bits + LFSR 37 de bits com *clocks* diferentes comparados com o PRNG ideal.



Fonte: Elaborado pelo autor

O PRNG com essa configuração apresenta curva praticamente igual à do PRNG ideal, o que é retratado nas métricas (MSE < 0,001) e (r^2 > 0,999). Os fatores de divisão de *clock* tornam esse PRNG 3 vezes mais lento (maior divisor tem fator 3) que a configuração anterior, mas a qualidade do PRNG está muito próxima do PRNG ideal.

A Tabela 2 apresenta a quantidade de falhas do DIEHARD nos resultados dos testes do PRNG parametrizado com outros fatores de divisão de *clock* e números de bits no bloco seletor de saída. O polinômio do LFSR tem taps nos registradores 37, 5, 4, 3, 2 e 1.

Tabela 2 – Quantidade de Falhas na Execução da Bateria de Testes Diehard na Modificação do PRNG CA 16 Bits + LFSR 37 Bits

Divisores/NºBits	1 Bit	2 Bits	4 Bits	8 Bits
1:1 (CA:LFSR) Cerda et al. (2012)	63	57	54	92
2:3 (CA:LFSR)	0	33	69	55
3:2 (CA:LFSR)	61	49	71	65
5:7 (CA:LFSR)	0	0	13	20
7:5 (CA:LFSR)	0	0	20	90

Fonte: Elaborado pelo autor

A primeira linha da Tabela 2 corresponde aos resultados obtidos pelo PRNG configurado como o modelo de Cerda et al. (2012). Todas as mudanças nos divisores de *clock* seguiram uma relação de números primos (com exceção de quando parametrizado com valores iguais). É possível observar que ao aumentar a quantidade de bits de saída a quantidade de falhas tende a aumentar, de maneira contrária, ao aumentar os valores dos divisores de *clock* a quantidade de falhas tende a diminuir.

A Tabela 3 apresenta os valores de r^2 e a Tabela 4 apresenta os valores de MSE para as mesmas variações nos parâmetros do PRNG da Tabela 2. Todas as configurações que foram aprovadas em todos os testes possuem valores de MSE menores que 0,0003 e r^2 maiores que 0,999 (valores em negrito nas Tabelas).

Tabela 3 – Valores de R^2 na Modificação do PRNG CA 16 Bits + LFSR 37 Bits

Divisores/NºBits	1 Bit	2 Bits	4 Bits	8 Bits
1:1 (CA:LFSR) Cerda et al. (2012)	0.956666	0.977839	0.983377	0.949317
2:3 (CA:LFSR)	**0.999230**	0.996020	0.963342	0.985352
3:2 (CA:LFSR)	0.955974	0.972501	0.968910	0.972797
5:7 (CA:LFSR)	**0.999244**	**0.999474**	0.998427	0.995513
7:5 (CA:LFSR)	**0.999343**	**0.999415**	0.998081	0.878458

Fonte: Elaborado pelo autor

Tabela 4 – Valores de MSE na Modificação do PRNG CA 16 Bits + LFSR 37 Bits

Divisores/NºBits	1 Bit	2 Bits	4 Bits	8 Bits
1:1 (CA:LFSR) Cerda et al. (2012)	0.032945	0.017327	0.013589	0.039828
2:3 (CA:LFSR)	**0.000259**	0.006796	0.035910	0.013986
3:2 (CA:LFSR)	0.033094	0.027939	0.030902	0.023377
5:7 (CA:LFSR)	**0.000191**	**0.000095**	0.000639	0.004539
7:5 (CA:LFSR)	**0.000115**	**0.000146**	0.002361	0.087345

Fonte: Elaborado pelo autor

Nas configurações testadas, nenhum PRNG parametrizado para saída com 4 ou 8 bits foi aprovado em toda a bateria de testes, independente dos valores dos divisores de *clock*. Por outro lado, apesar da configuração de 2:3 (CA:LFSR) para os divisores de *clock* e 1 bit de saída ter sido aprovada em todos os testes, a configuração

3:2 (CA:LFSR) com 1 bit de saída não foi. De maneira semelhante, a parametrização 5:7 (CA:LFSR) obteve melhores resultados que a 7:5 (CA:LFSR), quando se configuram 4 ou 8 bits no seletor de saída.

A parametrização 2:3 (CA:LFSR), não foi aprovada no DIEHARD a partir de 2 bits e as configurações 5:7 (CA:LFSR) e 7:5 (CA:LFSR), não foram aprovadas a partir de 4 bits de saída. Apesar de conseguir uma velocidade maior na geração dos números pseudoaleatórios quando se aumenta o número de bits, a quantidade de reprovações nos testes aumentou.

4.4.4 Testes de Uso de Elementos Lógicos do FPGA

A Tabela 5 apresenta a quantidade de elementos lógicos do FPGA utilizados (de um total de 68416) para PRNGs com 8, 18 e 32 bits no CA/LFSR. Foram avaliados PRNGs utilizando Autômatos Celulares e LFSRs isolados, bem como a combinação das duas estruturas.

Tabela 5 – Uso de Elementos Lógicos do FPGA Para CAs e LFSRs com Diferentes Tamanhos

Uso de EL	8 Bits	16 Bits	32 Bits
PRNG – CA isolado	35	43	59
PRNG – LFSR isolado	35	43	59
PRNG - CA e LFSR combinados	43	59	91

Fonte: Elaborado pelo autor

Na síntese do *hardware*, a quantidade de elementos lógicos em estruturas com 16 e 32 bits corresponde à quantidade de elementos lógicos em estruturas com 8 bits, somada ao incremento de bits entre eles, o que significa que cada célula do CA ou registrador do LFSR gasta um elemento lógico do FPGA ou *Look Up Table*.

4.4.5 Considerações Finais

Os resultados apresentados mostram uma contribuição em relação aos trabalhos de Cerda *et al.* (2012) e Tkacik (2002) pois foi possível fazer o PRNG composto por um CA de 16 de bits e um LFSR de 37 bits passar em toda a bateria de testes DIEHARD alterando apenas o divisor de *clock* de cada estrutura e não sendo

necessário aguardar pulsos de *clock* em quantidade igual ao dobro do tamanho de cada estrutura para então amostrar um bit.

Foram apresentados resultados com a exploração de alguns parâmetros do PRNG, mas não todos. Esta exploração será feita no Capítulo 5.

5 EXPLORAÇÃO DE PARÂMETROS DO PRNG

No capítulo anterior, o PRNG parametrizável tinha uma estrutura fixa, composta por CA de 16 bits e um LFSR de 37 bits, onde foi feita uma exploração da mudança dos parâmetros do PRNG. Neste capítulo serão avaliados tamanhos diferentes do CA e do LFSR. Além desses, outros parâmetros do PRNG também serão alterados.

5.1 Planejamento da Exploração de Parâmetros do PRNG

O Quadro 1 apresenta as variações de tamanho de CA e do LFSR que serão testadas no PRNG Parametrizável, tanto para o caso do PRNG composto por apenas CA ou apenas LFSR, como para o PRNG composto pela combinação de CA e LFSR. O quadro também mostra a quantidade de bits de saída e os divisores de *clock* de cada estrutura. Finalmente, o quadro apresenta o esquema de regras do CA (somente para PRNGs que usam o CA): Regra 30 ou Regras 90/150 alternando de acordo com Hortensius *et al.* (1989) com o objetivo de alcançar o tamanho de sequência máxima do CA. Os *taps* do LFSR foram posicionados para cada tamanho de LFSR de acordo com Alfke (1996).

A combinação das configurações de parâmetros apresentadas no Quadro 1 produzem 800 variações do PRNG, que serão testadas na bateria de testes DIEHARD avaliando a quantidade de testes com falha e os valores das métricas propostas no contexto deste trabalho, MSE e r^2. Também serão avaliados o uso de elementos lógicos do PRNG e o tempo de geração de um número aleatório através do *clock* relativo de cada configuração, que é calculado a partir do maior valor utilizado nos blocos divisores de *clock,* dividido pela quantidade de bits de saída.

O valor do *clock* relativo é usado para avaliar o tempo de geração do PRNG pois desta maneira a avaliação não dependerá do valor real do *clock*, que pode variar dependendo da plataforma que implementar o PRNG Parametrizável. O *clock* relativo significa quantos pulsos de *clock* são necessários para se obter 1 bit na saída do PRNG.

Quadro 1– Valores Utilizados nos Parâmetros do PRNG

Quantidade de bits (se apenas CA ou apenas LFSR)	**16** **20** **32** **37** **48** **52**
Quantidade de bits (se CA e LFSR conjugados)	16x37 16x52 20x20 24x16 24x26 32x16 32x37 32x52 37x16 48x37 48x52
Quantidade de bits de saída	1 2 4 8
Divisores de *clock* (CAxLFSR)	1x1 2x3 3x2 5x7 7x5
Regra do CA (em configurações que tenham CA)	30 90/150

Fonte: Elaborado pelo autor

5.2 Resultados Experimentais

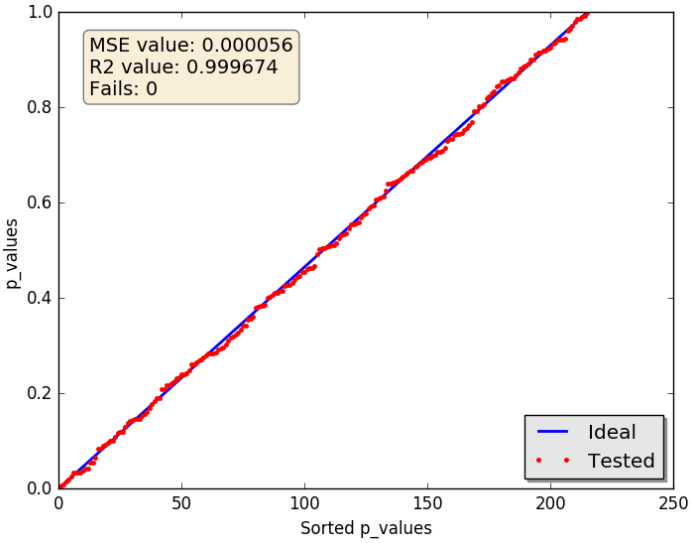
Durante os testes, foi observado que os resultados de uma mesma configuração do PRNG podem variar significativamente de acordo com o valor da semente, ou valor inicial das células do CA e registradores do LFSR. Para reduzir a influência da semente no resultado, foram realizados 30 testes para cada configuração e aquele com o melhor resultado foi escolhido para compor este trabalho.

Dentro de todas as 800 configurações diferentes para o PRNG, 339 conseguiram passar em todos os testes DIEHARD. Essas configurações foram posteriormente classificadas por *clock* relativo, uso elementos lógicos, MSE e r^2, nesta ordem, e apresentados no Apêndice A.

O melhor resultado considerando apenas o valor de MSE como critério foi para o PRNG Parametrizável composto por um CA de 48 bits (Regras 90/150) e um LFSR

de 52 bits usando os divisores de *clock* com valores 5x7 e 1 bit de saída (*clock* relativo de 7). Essa configuração usou 127 Elementos Lógicos e obteve 0.000056 e 0.999674 como valores de MSE e r^2 respectivamente (Figura 20).

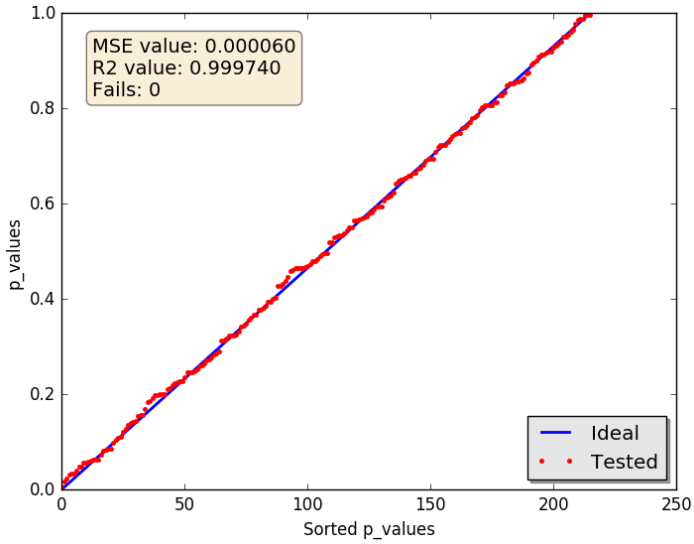
Figura 20 – Resultado dos testes da combinação CA de 48 bits + LFSR de 52 bits com 1 bit de saída.



Fonte: Elaborado pelo autor

O melhor resultado considerando apenas o valor de r^2 como critério foi o PRNG Parametrizável composto apenas por um CA de 48 bits (Regra 30), divisor de *clock* com valor 5 e 4 bits como saída (*clock* relativo de 1,125). Essa configuração usou 75 Elementos Lógicos e obteve 0.000060 e 0.999740 como valores de MSE e r^2 respectivamente (Figura 21).

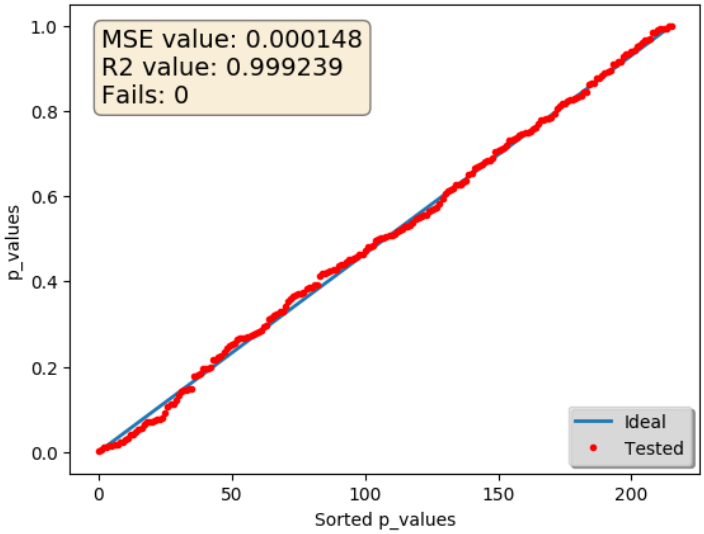
Figura 21 – Resultado dos testes do PRNG composto por apenas um CA de 48 bits com 2 bits de saída.



Fonte: Elaborado pelo autor

O melhor resultado considerando os múltiplos critérios de *clock* relativo, uso de elementos lógicos, MSE e r^2, nesta ordem, foi para o PRNG Parametrizável composto por um CA de 24 bits (Regras 90/150) e um LFSR de 26 bits usando os divisores de *clock* com valores 1x1 e 8 bits de saída (*clock* relativo de 0,125). Essa configuração usou 77 Elementos Lógicos e obteve 0.000148 e 0.999239 como valores de MSE e r^2 respectivamente (Figura 22).

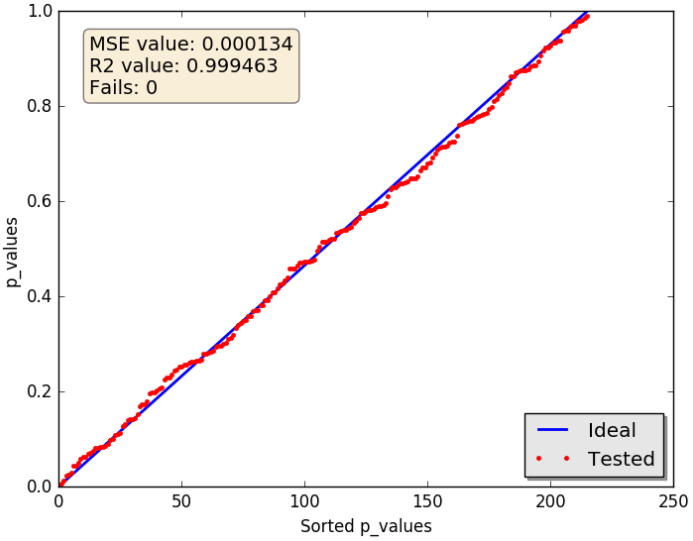
Figura 22 – Resultado dos testes da combinação CA de 24 bits + LFSR de 26 bits com 8 bits de saída.



Fonte: Elaborado pelo autor

Ao considerar apenas PRNGs compostos somente por um CA, o melhor resultado considerando os múltiplos critérios de *clock* relativo, uso de elementos lógicos, MSE e r^2, nesta ordem, foi para o PRNG Parametrizável composto por um CA de 48 bits (Regra 30), usando divisor de *clock* com valor 1 e 4 bits de saída (*clock* relativo de 0,25). Essa configuração usou 75 Elementos Lógicos e obteve 0.000134 e 0.999463 como valores de MSE e r^2 respectivamente (Figura 23).

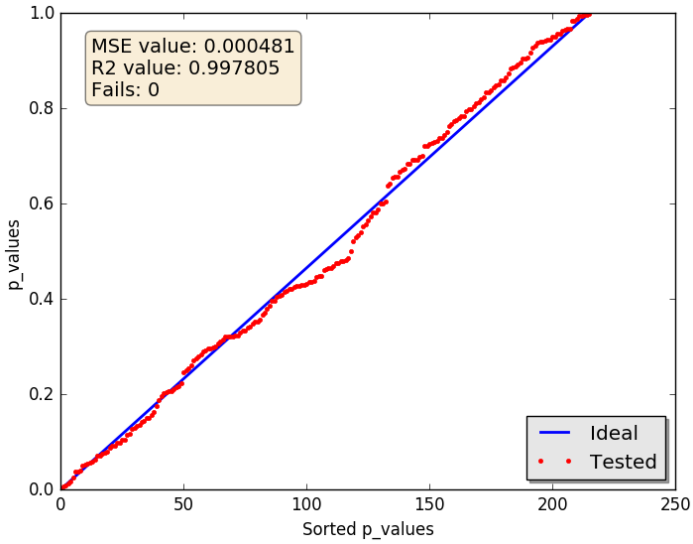
Figura 23 – Resultado dos testes do PRNG composto por apenas um CA de 48 bits com 4 bits de saída.



Fonte: Elaborado pelo autor

Finalmente, ao considerar apenas PRNGs compostos somente por um LFSR, o melhor resultado considerando os múltiplos critérios de *clock* relativo, uso de elementos lógicos, MSE e r^2, nesta ordem, foi para o PRNG Parametrizável composto por um LFSR de 48 bits, usando divisor de *clock* com valor 1 e 2 bits de saída (*clock* relativo de 0,5). Essa configuração usou 75 Elementos Lógicos e obteve 0.000481 e 0.997805 como valores de MSE e r^2 respectivamente (Figura 24).

Figura 24 – Resultado dos testes do PRNG composto por apenas um LFSR de 48 bits com 2 bits de saída.



Fonte: Elaborado pelo autor

O valor de MSE obtido na Figura 23 é melhor que o da Figura 22, mostrando que um PRNG composto por apenas um CA pode ser melhor que a combinação de CA + LFSR dependendo do critério de avaliação utilizado. Outros resultados semelhantes são apresentados no Apêndice A, onde PRNGs compostos por apenas um CA ou apenas um LFSR, são melhores que a combinação dessas estruturas.

5.3 Considerações Finais

Foi verificado que nenhuma configuração que gerou na saída um *bitstream* com quantidade de bits menor que a necessária para gerar uma sequência de 11MB sem repetir foi capaz de passar em toda bateria de testes DIEHARD. Ou seja, apenas PRGNs com mais de 24 bits (somando CA e LFSR) conseguiram passar em todos os testes.

Quando se utilizam 48 bits ou mais em cada estrutura, a combinação do CA com LFSR não tem ganho de qualidade significativo, quando comparado a PRNGs compostos pelas estruturas separadas, com o mesmo tamanho. Verificou-se também que quando a CA e a LFSR têm tamanhos diferentes, melhores resultados foram obtidos ao usar o maior valor dos divisores de *clock* na estrutura com maior tamanho.

A classificação de um PRNG como melhor ou pior que outro depende de qual critério é usado, por exemplo: o PRNG da Figura 22 tem MSE pior que o da Figura 20

e r^2 pior que o da Figura 21, mas tem *clock* relativo melhor que ambas. É uma questão de otimização multicritério decidir qual o melhor PRNG.

6 PRNG COM HARDWARE EVOLUTIVO INTRÍNSECO

Neste capítulo apresenta-se a utilização dos conceitos de *Hardware* Evolutivo no PRNG Parametrizável, com o propósito de fazer os parâmetros do PRNG serem alterados em tempo de execução, para obter uma configuração que tenha resultados melhores segundo o critério de avaliação.

Para avaliar o uso do conceito de EHW no PRNG parametrizável, foi escolhida a plataforma *Mpression Helio Evaluation Base Board*, que contém um Altera Cyclone V *System on a Chip* (SoC). Esse SoC possui dentro dele um FPGA e um *Hard Processor System* (HPS) ARM Cortex-A9 Dual Core, que executa uma distribuição de Linux Embarcado chamada Ångström, com a versão de kernel 3.10.31-ltsi. O ARM é capaz de se comunicar com o FPGA através de barramentos de dados que mapeiam conexões internas do FPGA a endereços de memória virtual no Linux.

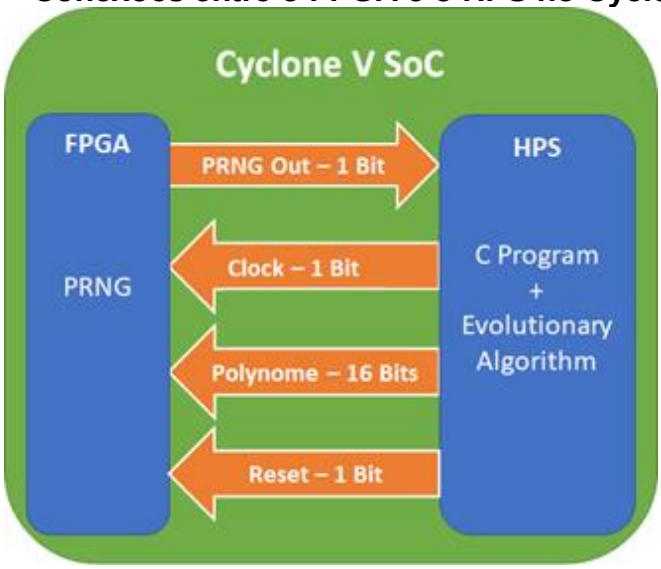
Com esse SoC é possível fazer com que o FPGA implemente o PRNG Parametrizável e o processador ARM seja o responsável por avaliar a sequência de números aleatórios gerada. A partir dessa avaliação, o ARM então muda os parâmetros do PRNG através de um algoritmo evolucionário. Como o *software* que avalia o PRNG agora é executado dentro do próprio *chip*, o sistema é classificado como EHW Intrínseco (SALVADOR, 2016).

6.1 Arquitetura do PRNG com EHW

A evolução do *hardware* do PRNG não terá como objetivo selecionar a melhor estrutura, mas apenas otimizar os parâmetros de uma estrutura previamente definida, apresenta-se como uma prova de conceito de um PRNG evolutivo. Portanto, foi escolhida inicialmente uma estrutura mais simples de PRNG, com apenas 1 bit de saída, e que é composto apenas por um LFSR de 18 bits.

Através da ferramenta QSys do *software* Altera Quartus II, o mapeamento das conexões entre o PRNG e o HPS foi feito conforme mostrado na Figura 25.

Figura 25 – Conexões entre o FPGA e o HPS no Cyclone V Soc



Fonte: Elaborado pelo autor

O HPS gera o polinômio do LFSR, pode *resetar* o PRNG para o estado inicial (também chamado de semente, com o registrador menos significativo em nível 1 e os demais em nível 0) e gera um sinal de *clock*, para que o valor da saída do PRNG seja atualizado apenas quando o HPS estiver pronto para ler o novo bit. O sinal de entrada no HPS é o bit de saída do PRNG.

A Figura 26 mostra uma imagem do *software* Qsys, onde uma entidade chamada “lfsr” é a responsável por fazer a conexão entre o FPGA e o HPS. É uma entidade do tipo *Parallel Input/Output* (PIO) bidirecional, ou seja, possuí entradas e saídas em ambos sentidos entre FPGA e HPS, justamente como mostrado na Figura 25. É possível ver também quais os endereços de memória no Linux estão reservados para a escrita e leitura dos bits compartilhados. No lado do FPGA são criados dois *arrays* de bits para escrita e leitura separadamente.

Figura 26 – Conexões entre o FPGA e o HPS no Software Qsys

lfsr	PIO (Parallel I/O)		
clk	Clock Input		
reset	Reset Input		
s1	Avalon Memory Mapped Slave	0x0001_00e0	0x0001_00ff
external_connection	Conduit		

Fonte: Elaborado pelo autor

Um *software* escrito em linguagem C é o responsável por fazer a evolução do algoritmo evolucionário, onde a função objetivo utilizada é o tamanho da sequência

gerada pelo PRNG. A bateria de testes DIEHARD não foi escolhida como função objetivo devido à limitação da velocidade da execução do Sistema Operacional, pois o *software* só está pronto para analisar mudanças no bit de saída do PRNG a cada 1,3 ms, no mínimo (valor obtido experimentalmente). Com essa velocidade, seria necessário mais de 1 dia para gerar a sequência de 11MB de dados necessária para executar a bateria de testes DIEHARD.

O *software* faz a evolução de um único indivíduo e por isso o algoritmo evolucionário utiliza apenas a mutação como técnica de evolução. A mutação no polinômio do PRNG tem 50% de chance de ocorrer em 1 bit e 50% de chance de ocorrer em 2 bits. O critério de parada da evolução é atingir o tamanho mínimo da sequência, que é um parâmetro escolhido. Há também um elitismo para preservar o melhor polinômio já obtido: caso o tamanho da sequência gerada utilizando-se o polinômio atual seja menor que aquela obtida com o melhor polinômio obtido em gerações passadas, a mutação é feita a partir desse melhor polinômio e não a partir do polinômio corrente.

6.1.1 Resultados Experimentais no SoC

Foram feitas 10 execuções da evolução do PRNG Parametrizável utilizando a *Mpression Helio Evaluation Base Board*, onde o critério de parada foi alcançar uma sequência de tamanho mínimo igual a 250.000 (o máximo possível com 18 bits é 262143). As 10 execuções encontraram polinômios diferentes e todos atenderam ao critério de parada.

A Tabela 6 apresenta o número de gerações para atingir o critério de parada e o polinômio obtido (em hexadecimal) ao final da evolução para os 10 testes do PRNG Parametrizável com EHW.

Tabela 6 – Polinômios obtidos nas execuções de EHW no PRNG Parametrizável

Nº da Execução	1	2	3	4	5	6	7	8	9	10
Nº de Gerações	24	16	25	21	26	25	44	32	39	29
Polinômio Obtido	2B641	3DD90	21042	2A98A	24056	2A40C	30EA2	2ABC0	2A206	2E220

Fonte: Elaborado pelo autor

A Figura 27 apresenta as mensagens de log do programa que faz a evolução do PRNG para a execução de número 10, em que foram necessárias 16 gerações para atingir o critério de parada.

Figura 27 – Log do programa para evoluir o PRNG

```
root@cyclone5:~/lfsr# ./evolve 20000
250000

LFSR EHW

Initial Polynome: 0x00020000
Sequence Minimal Size: 250000
LFSR Size: 18

Evaluating polynome 0x00020000
Polynome 0x00020000 MAX size 1

Evaluating polynome 0x00028010
Polynome 0x00028010 MAX size 4445
Better polynome found!

Evaluating polynome 0x00028811
Polynome 0x00028811 MAX size 2555
Polynome max size is lower than best,
mutating best again

Evaluating polynome 0x00029810
Polynome 0x00029810 MAX size 9271
Better polynome found!

Evaluating polynome 0x00028810
Polynome 0x00028810 MAX size 2387
Polynome max size is lower than best,
mutating best again

Evaluating polynome 0x00029C10
Polynome 0x00029C10 MAX size 32767
Better polynome found!

Evaluating polynome 0x00029E18
Polynome 0x00029E18 MAX size 29127
Polynome max size is lower than best,
mutating best again

Evaluating polynome 0x00021C50
Polynome 0x00021C50 MAX size 28985
Polynome max size is lower than best,
mutating best again

Evaluating polynome 0x0002DC90
Polynome 0x0002DC90 MAX size 75565
Better polynome found!

Evaluating polynome 0x0002DC94
Polynome 0x0002DC94 MAX size 43307
Polynome max size is lower than
best, mutating best again

Evaluating polynome 0x0002DE90
Polynome 0x0002DE90 MAX size 32764
Polynome max size is lower than
best, mutating best again

Evaluating polynome 0x0002DD90
Polynome 0x0002DD90 MAX size 131071
Better polynome found!

Evaluating polynome 0x0002DF90
Polynome 0x0002DF90 MAX size 341
Polynome max size is lower than
best, mutating best again

Evaluating polynome 0x0003FD90
Polynome 0x0003FD90 MAX size 10922
Polynome max size is lower than
best, mutating best again

Evaluating polynome 0x00029D90
Polynome 0x00029D90 MAX size 29127
Polynome max size is lower than
best, mutating best again

Evaluating polynome 0x0003DD90
Polynome 0x0003DD90 reached minimum
size 250000

Condition reached with polynome
0x0003DD90, after 16 generations
```

Fonte: Elaborado pelo autor

O polinômio inicial escolhido foi 0x20000, que é o polinômio mais simples. Pode ser visto na Figura 27 que o algoritmo evolutivo faz o uso de elitismo. Quando o polinômio atual gera uma sequência com o maior tamanho até o momento a mensagem "*Better polynome found!*" aparece no log. Ao contrário, quando o polinômio

em teste obtém um resultado pior, a mensagem "*Polynome max size is lower than best, mutating best again*" aparece no log, indicando que o melhor polinômio sofrerá mutação novamente, para tentar obter um resultado melhor a partir dele.

O experimento realizado demonstra que o objetivo da prova de conceito do PRGN Evolutivo foi alcançado, pois nas 10 execuções o algoritmo evolutivo foi capaz de encontrar um polinômio para o PRNG que fosse capaz de gerar uma sequência com mais de 250.000 números antes de se repetir, partindo de um polinômio (0x20000) que gera apenas um número.

6.2 Simulação do PRNG com EHW

Na seção anterior, foram apresentados a proposta, a implementação e os resultados da prova de conceito do PRNG com *hardware* evolutivo intrínseco, utilizando a plataforma *Mpression Helio Evaluation Base Board*. Por limitação dessa plataforma, a bateria de testes DIEHARD não pode ser utilizada na formulação da função objetivo do algoritmo evolucionário implementando no HPS. Além disso, como o objetivo era apenas apresentar uma prova de conceito, foi utilizada uma estrutura de PRNG mais simples, consistindo apenas de um LFSR de 18 bits, cujo parâmetro de configuração a ser definido pelo algoritmo evolucionário era apenas o polinômio com os *taps* dos registradores.

Nesta seção, tomando como ponto de partida os resultados satisfatórios da prova de conceito apresentada na seção anterior, o que se propõe é a simulação do hardware evolutivo do PRNG em computador, utilizando, como função objetivo do algoritmo evolucionário, a quantidade de testes bem-sucedidos no DIEHARD. Nessa simulação, o PRNG foi inicialmente configurado de acordo com o modelo de (CERDA *et al.*, 2012), onde um CA de 16 bits (com a regra alternando entre as de número 90 e 150) é combinado com um LFSR de 37 bits, ambos com o mesmo *clock* e tomando apenas 1 bit à saída do PRNG.

Três parâmetros do PRNG podem ser definidos pelo algoritmo evolucionário e alterados na simulação: o divisor de *clock* do CA, o divisor de *clock* do LFSR (ambos podendo assumir valores de 1 a 8) e a regra do CA (regra 30, 90, 105, 150, 165, 225 ou as regras 90 e 150 alternadamente).

O polinômio do LFSR nesta simulação é o mesmo utilizado por (CERDA *et al.*, 2012), com *taps* nos registradores 37, 5, 4, 3, 2 e 1, o que garante o maior tamanho

de sequência. Os valores possíveis para a regra do CA foram escolhidos de acordo com os resultados de Cerda *et al.* (2012) e Szaban (2014). Através de um sorteio, cada um dos três parâmetros tem 50% de chance de serem alterados dentre os valores possíveis. Todos os três parâmetros podem ser alterados para a próxima geração, e, o sorteio ocorre até pelo menos um parâmetro ser alterado.

O critério de parada da evolução é atingir a quantidade especificada de testes DIEHARD bem-sucedidos, ou o limite do número de gerações da evolução do indivíduo, sendo ambos parâmetros da simulação. A quantidade de bits de saída (N = 1, 2, 4 ou 8 bits) é fixada previamente como um requisito do PRNG, ou seja, os demais parâmetros devem evoluir até encontrar uma configuração do PRNG que minimize a quantidade de falhas nos testes DIEHARD para o requisito de N bits de saída.

6.2.1 Resultados Experimentais da Simulação

A simulação feita em computador e que utilizou a quantidade de testes DIEHARD bem-sucedidos como função objetivo foi parametrizada para interromper sua execução apenas quando todos os 216 testes não apresentassem falhas, ou quando se alcancem 100 gerações. Foram realizadas 28 simulações (7 para cada quantidade de bits de saída) e todas foram capazes de obter uma evolução da parametrização do PRNG capaz de passar em todos os testes da bateria antes do limite de gerações. Ressalta-se que a configuração inicial, que é o modelo proposto por CERDA *et al.* (2012), foi reprovada em 63 testes, com MSE de 0.032945 e r^2 de 0.956666, conforme mostrado na Figura 17.

A Tabela 7 apresenta o resultado das 28 simulações, onde também se apresentam os valores das métricas *clock* relativo, MSE e r^2. O uso de elementos lógicos é sempre 80, já que os tamanhos do CA e do LFSR são fixos. Cada uma dessas configurações buscou atender ao requisito de um determinado tamanho de saída (N = 1, 2, 4 e 8).

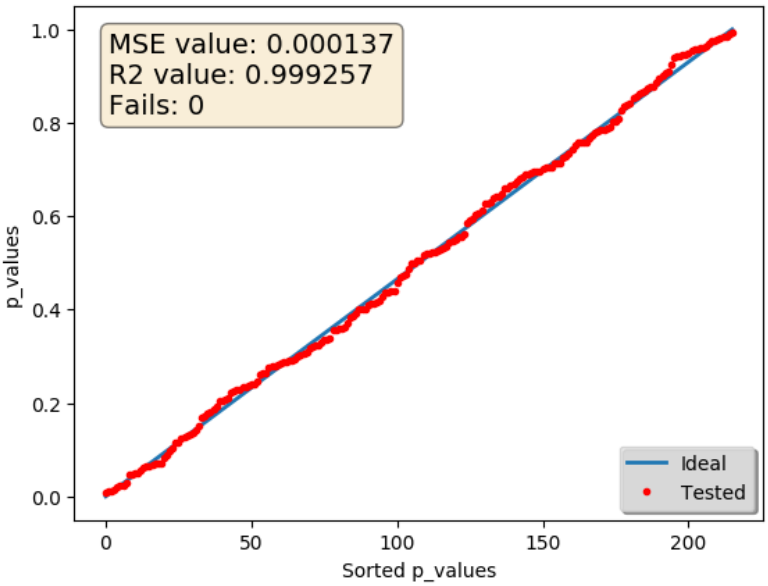
Tabela 7 – Resultados da Simulação do PRNG Parametrizável Evolutivo

Bits de saída	Regra CA	Divisor de *clock* CA	Divisor de *clock* LFSR	Clock Relativo	MSE	r^2
1	30	1	6	6	0.001317	0.997030
1	30	2	5	5	0.000417	0.998451
1	150	1	7	7	0.000759	0.996962
1	165	5	7	7	0.000262	0.998758
1	225	1	3	3	0.000717	0.998514
1	225	1	5	5	0.000519	0.997915
1	**225**	**3**	**3**	**3**	**0.000137**	**0.999257**
2	**30**	**7**	**4**	**3.5**	**0.000175**	**0.999575**
2	90	3	7	3.5	0.000838	0.998771
2	90/150	3	5	2.5	0.000168	0.999214
2	105	3	7	3.5	0.000204	0.999091
2	105	6	7	3.5	0.000388	0.998653
2	165	7	8	4	0.001065	0.999051
2	225	1	8	4	0.000683	0.999293
4	30	1	6	1.5	0.000612	0.998933
4	30	7	6	1.75	0.000408	0.998783
4	90/150	2	8	2	0.000950	0.998913
4	105	1	8	2	0.000488	0.997825
4	150	6	8	2	0.000827	0.998709
4	225	3	7	1.75	0.001009	0.997561
4	225	8	8	2	0.000354	0.998457
8	30	2	7	0.875	0.000383	0.998612
8	30	3	6	0.75	0.000847	0.997801
8	30	3	7	0.875	0.000365	0.998425
8	**30**	**5**	**4**	**0.625**	**0.001086**	**0.998763**
8	30	6	7	0.875	0.000439	0.998636
8	30	7	2	0.875	0.001492	0.997657
8	30	8	7	0.875	0.000376	0.999394

Fonte: Elaborado pelo autor

O melhor resultado das simulações considerando apenas o valor de MSE como critério, realçado em negrito na Tabela 7, foi para o PRNG Parametrizável usando a Regra 225 com os divisores de *clock* com valores 3x3 e 1 bit de saída (*clock* relativo de 3). Essa configuração obteve 0.000137 e 0.999257 como valores de MSE e r^2 respectivamente (Figura 28).

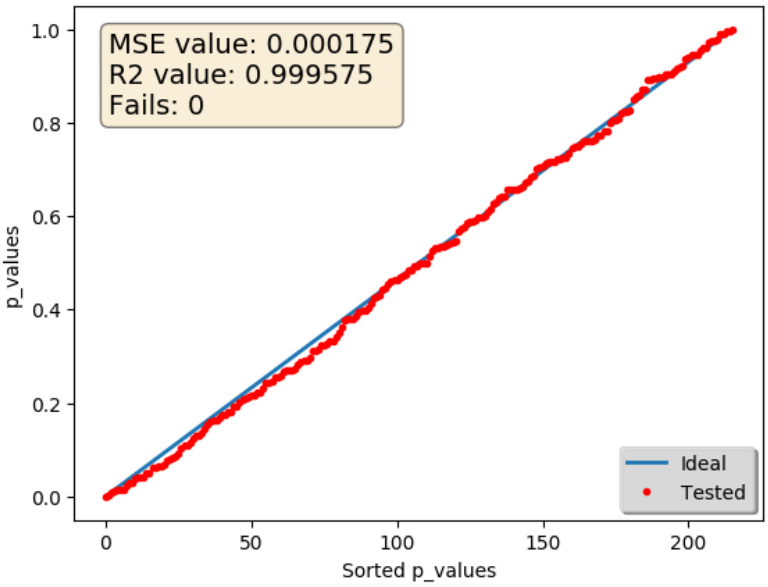
Figura 28 – Resultado dos testes da simulação do PRNG com Regra 225, 1 bit de saída e divisores de *clock* com valores 3x3.



Fonte: Elaborado pelo autor

O melhor resultado das simulações considerando apenas o valor de r^2 como critério, também realçado em negrito na Tabela 7, foi para o PRNG Parametrizável usando a Regra 30 com os divisores de *clock* com valores 7x4 e 2 bits de saída (*clock* relativo de 3,5). Essa configuração obteve 0.000175 e 0.999575 como valores de MSE e r^2 respectivamente (Figura 29).

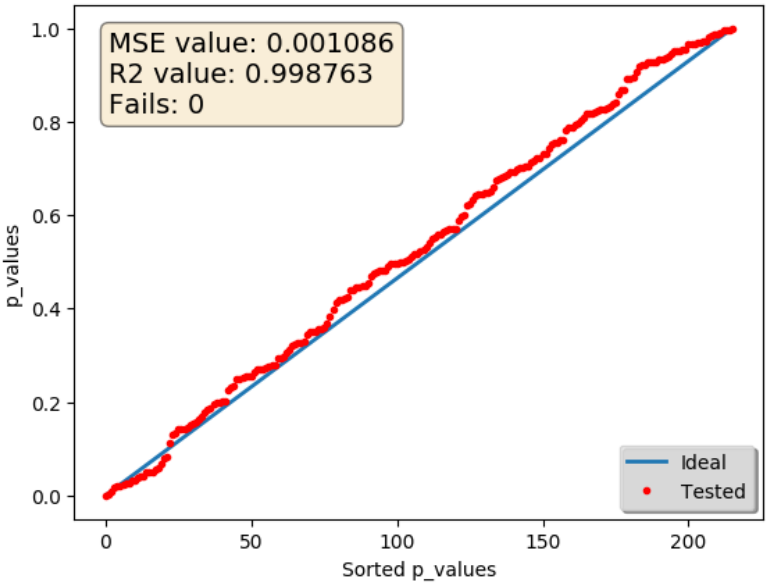
Figura 29 – Resultado dos testes da simulação do PRNG com Regra 30, 2 bits de saída e divisores de *clock* com valores 7x4.



Fonte: Elaborado pelo autor

Por último, o melhor resultado considerando apenas o valor do *clock* relativo como critério, em negrito na Tabela 7, foi para o PRNG Parametrizável usando a Regra 30 com os divisores de *clock* com valores 5x4 e 8 bits de saída (*clock* relativo de 0,625). Essa configuração obteve 0.001086 e 0.998763 como valores de MSE e r^2 respectivamente (Figura 30).

Figura 30 – Resultado dos testes da simulação do PRNG com Regra 30, 8 bits de saída e divisores de *clock* com valores 5x4.



Fonte: Elaborado pelo autor

6.3 Considerações Finais

Apesar da limitação de velocidade do HPS impedir a utilização da bateria de testes DIEHARD para compor a função objetivo do algoritmo evolucionário, foi possível evoluir um parâmetro de um PRNG simples (polinômio de um LFSR de 18 bits), até satisfazer a função objetivo correspondente ao tamanho mínimo da sequência de números pseudoaleatórios.

Para avaliar a evolução de mais parâmetros de um PRNG um pouco mais complexo, equivalente ao modelo de Cerda *et al.* (2012) foi feita uma simulação em computador, onde foi possível avaliar o PRNG utilizando a bateria de testes DIEHARD. Nessa simulação, o algoritmo evolutivo foi capaz de fazer o PRNG passar em toda a bateria de testes ao evoluir seus parâmetros: divisor de *clock* do CA, divisor de *clock* do LFSR e regra do CA.

Diferentemente do realizado no Capítulo 4, onde um projetista humano foi o responsável por escolher parâmetros para o PRNG, na simulação os parâmetros são obtidos pelo próprio *Hardware* Evolutivo. Foram realizadas 28 simulações e todas foram capazes de obter uma evolução da parametrização do PRNG capaz de passar em todos os testes da bateria antes do limite de gerações, com parâmetros diferentes daqueles usados nos Capítulos 4 e 5 para o PRNG composto por um CA de 16 bits e um LFSR de 37 bits.

O uso de EHW tem a vantagem de ser um sistema que pode se adaptar a falhas de *hardware,* por exemplo, uma célula do CA que tem sempre saída em 1, e buscar uma configuração que continue se comportando de maneira semelhante antes da falha ocorrer.

7 CONCLUSÃO

Neste trabalho, foi proposto e implementado em FPGA um PRNG parametrizável construído por meio da combinação de CA, LFSR, divisores de *clock* e um seletor de saída, cujo conjunto de parâmetros que podem ser configurados em tempo de execução são: os divisores, a quantidade de bits de saída, a regra e a semente do CA e o polinômio do LSFR.

O PRNG foi inicialmente avaliado tendo uma estrutura semelhante ao PRNG proposto por Cerda *et al.* (2012), que combina um CA de 16 bits e um LFSR de 37 bits, com 1 bit de saída e o mesmo *clock* (divisores de clock com valores 1x1). Essa configuração do PRNG teve sua qualidade avaliada através da bateria de testes DIEHARD, onde obtiveram-se mais testes bem-sucedidos em relação ao PRNG composto pelas estruturas isoladas, mas que ainda foi reprovado em 63 dos 216 testes. Quando o PRNG foi parametrizado para selecionar 2 bits do CA com 2 bits do LFSR por meio de uma porta XOR e gerar 2 bits de saída, aumentou-se ainda mais a quantidade de testes bem-sucedidos, mas foi reprovado em 54 dos 216 testes. Com os 2 bits de saída o PRNG é 2 vezes mais rápido do que com 1 bit de saída, mantendo-se os valores dos divisores de *clock*, de maneira semelhante ao que foi relatado nos trabalhos relacionados de Cerda *et al.* (2012) e Tkacik (2002).

Quando o PRNG foi parametrizado para que os blocos divisores de *clock* tivessem valores 2:3 (CA:LFSR), não necessitando receber uma quantidade de pulsos de *clock* de acordo com o dobro do tamanho do CA ou LFSR, este foi aprovado em todos os 216 testes e teve ganhos significativos de velocidade em relação ao PRNG proposto por Tkacik (2002).

Com a exploração da variação dos parâmetros do PRNG proposto neste trabalho, 339 PRNGs com combinações diferentes de parâmetros foram capazes de ser aprovados em toda a bateria de testes DIEHARD. Diante dessas 339 diferentes configurações de um total de 800, foi verificado que quando se utilizam 48 bits ou mais em cada estrutura, a combinação do CA com LFSR tem quantidade de testes DIEHARD bem-sucedidos semelhante a PRNGs compostos pelas estruturas separadas, com o mesmo tamanho (Resultados no Apêndice A). Verificou-se também que quando o CA e o LFSR têm tamanhos diferentes, melhores resultados foram obtidos ao usar o maior valor dos divisores de *clock* na estrutura com maior tamanho.

Durante os testes, observou-se um *trade-off* entre a qualidade e o *clock* relativo do PRNG, onde o *clock* relativo depende do número de bits de saída dividido pelo valor do maior divisor de *clock*. Quanto maior a qualidade do PRNG, mais lento se torna e vice-versa, como apresentado na Tabela 2. É uma decisão de projeto escolher uma melhor qualidade ou maior velocidade, e o fato de o PRNG proposto ser parametrizável facilita essa escolha.

Ao aplicar os conceitos de *Hardware* Evolutivo Intrínseco no PRNG, foi possível analisar o tamanho da sequência gerada pelo PRNG em tempo de execução e otimizar o polinômio do LFSR de acordo com os resultados obtidos. Ao utilizar o Cortex-A9 presente no Cyclone V SoC para executar o algoritmo evolutivo, a velocidade do PRNG fica limitada à velocidade de processamento do Cortex-A9.

A simulação que foi feita para evoluir o PRNG Parametrizável com configuração inicial como o modelo proposto por Cerda *et al.* (2012) foi capaz de encontrar 28 configurações diferentes que passaram em toda a bateria de testes DIEHARD, com parâmetros diferentes daqueles usados nos Capítulos 4 e 5.

O uso de EHW tem a vantagem de ser um sistema que pode se adaptar a falhas de *hardware*, mudando o polinômio para buscar uma configuração que continue se comportando de maneira semelhante a de antes da falha. Outra vantagem é que ao mudar o polinômio do PRNG, muda-se também a equação que define como os números aleatórios são gerados. Isso é útil em aplicações em que a equação do PRNG não pode ser descoberta.

As principais contribuições deste trabalho são a proposta, o projeto e a implementação de um PRNG parametrizável que não está limitado a valores fixos de regra do CA, ao polinômio do LSFR, ao *clock* de cada estrutura e ao tamanho de saída, sendo possível alterar esses valores em tempo de geração dos números pseudoaleatórios para adaptar o PRNG da melhor maneira aos requisitos e restrições da aplicação onde será utilizado. Foram utilizadas outras métricas para avaliação da qualidade do PRNG, como o valor de MSE e r^2. Com esse PRNG foi possível fazer o modelo proposto por Cerda *et al.* (2012) passar em toda a bateria de testes DIEHARD e diferentemente do modelo proposto por Tkacik (2002), não há a limitação de que o tempo mínimo de amostragem em que cada estrutura deve receber pulsos de *clock* seja pelo menos o dobro de seu tamanho.

Trabalhos futuros poderiam acrescentar um parâmetro para mudar o modelo de vizinhança do CA, utilizar o ASG ao invés de um LFSR simples e na parte de

Hardware Evolutivo, implementar o algoritmo evolutivo dentro do próprio FPGA. Dessa maneira o sistema não dependeria de um sistema operacional como o Linux e toda implementação seria feita puramente em VHDL, eliminando a limitação da integração do FPGA com o HPS. Futuras implementações poderiam ter vários PRNGs dentro do SoC, para avaliar mais combinações de parâmetros de uma só vez, além de utilizar a combinação CA + LFSR e outras funções objetivo para avaliar a qualidade do PRNG. O PRNG poderia ser capaz de aumentar ou diminuir o tamanho do CA e do LFSR em tempo de execução, de acordo com o necessário, assim o PRNG poderia ser configurado como qualquer modelo apresentado no Capítulo 5.

REFERÊNCIAS

ALFKE, P. Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. **Xilinx, Inc.**, v. Version 1.1, n. XAPP 052, July 1996.

BURIAN, P. Evolvable Hardware Implemented by FPGA. **Computer Applications in Electrical Engineering**, v. 2, p. 100-117, 2009.

CERDA, J. C. et al. An Efficient FPGA Random Number Generator Using LFSRs and Cellular Automata. **Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on**, p. 912-915, 2012.

GONCU, E.; YALCIN, M. E. A New Cellular Automata Model with Memory and its FPGA Implementation. **Cellular Nanoscale Networks and their Applications (CNNA), 2014 14th International Workshop on**, p. 1-2, 2014.

GUAN, S.-U.; TAN, S. K. Pseudorandom NumberGeneration With Self-Programmable Cellular Automata. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 23, n. 7, p. 1095-1101, 2004.

HADDOW, P. C.; TYRRELL, A. M. Challenges of Evolvable Hardware: Past, Present and the Path to a Promising Future. **Genetic Programming and Evolvable Machines**, v. 12, n. 3, p. 183-215, 2011.

HALBACH, M.; HOFFMANN, R.; RÖDER, P. FPGA Implementation of Cellular Automata Compared to Software Implementation. **ARCS Workshops**, v. 41, p. 309-317, 2004.

HIGUCHI, T. et al. Real-World Applications of Analog and Digital Evolvable Hardware. **Evolutionary Computation, IEEE Transactions on**, v. 3, n. 3, p. 220-235, 1999.

HORTENSIUS, P. D. et al. Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 8, n. 8, p. 842-859, 1989.

ISHIMURA, K. et al. FPGA Implementation of Hardware-Oriented Reaction-Diffusion Cellular Automata Models. **Nonlinear Theory and Its Applications, IEICE**, v. 6, n. 2, p. 252-262, 2015.

KHALAF, A. A. M.; EL-KARIM, M. S. A.; HAMED, H. F. A. A Triple Hill Cipher Algorithm Proposed to Increase the Security of Encrypted Binary Data and its Implementation Using FPGA. **18th International Conference on Advanced Communication Technology (ICACT)**, IEEE, p. 752-759, 2016.

L'ECUYER, P.; SIMARD, R. TestU01: AC library for empirical testing of random number generators. **ACM Transactions on Mathematical Software (TOMS)**, v. 33, n. 4, p. 22, 2007.

MAITI, S.; CHOWDHURY, D. R. Study of Five-Neighborhood Linear Hybrid Cellular Automata and Their Synthesis. **International Conference on Mathematics and Computing**, 2017. 68-83.

MARSAGLIA, G. DIEHARD, A Battery of Tests for Random Number Generators. **CD-ROM, Department of Statistics and Supercomputer Computations Research Institute, Florida State University**, Florida State University, 1996. Disponivel em: <http://stat.fsu.edu/~geo/diehard.html>. Acesso em: 19 Março 2016.

MARTIN, P. An Analysis Of Random Number Generators For A Hardware Implementation Of Genetic Programming Using FPGAs And Handel-C. **Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation**, Morgan Kaufmann Publishers Inc., p. 837-844, 2002.

MORENO-ARMENDÁRIZ, M. A. et al. Hardware Implementation of the Elitist Compact Genetic Algorithm Using Cellular Automata Pseudo-Random Number Generator. **Computers & Electrical Engineering**, v. 39, n. 4, p. 1367-1379, 2013.

PAROL, M.; DĄBAL, P.; SZPLET, R. Pseudo-Random Bit Generators Based on Linear-Feedback Shift Registers in a Programmable Device. **Measurement Automation Monitoring**, 62, 2016.

RUKHIN, A. et al. **A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications**. Booz-Allen and Hamilton Inc Mclean Va. [S.l.]. 2001.

SALVADOR, R. Evolvable Hardware in FPGAs: Embedded Tutorial. **Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2016 International Conference on**, p. 1-6, 2016.

SHACKLEFORD, B. et al. FPGA Implementation of Neighborhood-of-Four Cellular Automata Random Number Generators. **Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays**, p. 106-112, 2002.

SZABAN, M. How to Find the Best Rules for CA-Based PRNG. **Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)**, p. 1, 2014.

THOMAS, D. B.; LUK, W. The LUT-SR Family of Uniform Random Number Generators for FPGA Architectures. **Very Large Scale Integration (VLSI) Systems, IEEE Transactions on**, v. 21, n. 4, p. 761-770, 2013.

TKACIK, T. E. A Hardware Random Number Generator. **Cryptographic Hardware and Embedded Systems-CHES**, v. 2523, p. 450-453, 2002.

VON NEUMANN, J.; BURKS, A. W. Theory of Self-Reproducing Automata. **IEEE Transactions on Neural Networks**, v. 5, n. 1, p. 3-14, 1966.

WEISSTEIN, E. W. Random Number. **MathWorld--A Wolfram Web Resource**, 2017. Disponivel em: <http://mathworld.wolfram.com/RandomNumber.html>. Acesso em: 29 ago. 2017.

WOLFRAM, S. **Theory and Applications of Cellular Automata**. Singapore: World scientific, v. 1, 1986a.

WOLFRAM, S. Random Sequence Generation by Cellular Automata. **Advances in applied mathematics**, v. 7, n. 2, p. 123-169, 1986b.

APÊNDICE A

A seguir, são apresentados os resultados de todos as 339 configurações do PRNG que passaram na bateria de testes DIEHARD, classificadas por clock relativo, uso de elementos lógicos, valor de MSE e valor de r^2, nesta ordem.

Configuration	Bits	CA Rule	CA Clock	LFSR Clock	Relative Clock	Logic Elements	MSE	r^2
CA_24_LFSR_26	8	90/150	1	1	0.125	77	0.000148	0.999239
CA_37_LFSR_16	8	30	1	1	0.125	80	0.000145	0.999301
CA_16_LFSR_52	8	30	1	1	0.125	95	0.000161	0.999215
CA_32_LFSR_37	8	90/150	1	1	0.125	96	0.000179	0.999239
CA_32_LFSR_37	8	30	1	1	0.125	96	0.000313	0.999230
CA_32_LFSR_52	8	30	1	1	0.125	111	0.000102	0.999437
CA_32_LFSR_52	8	90/150	1	1	0.125	111	0.000116	0.999442
CA_48_LFSR_37	8	30	1	1	0.125	112	0.000124	0.999345
CA_48_LFSR_52	8	90/150	1	1	0.125	127	0.000119	0.999608
CA_48_LFSR_52	8	30	1	1	0.125	127	0.000189	0.998980
CA_48_LFSR_0	4	30	1	0	0.250	75	0.000134	0.999463
CA_32_LFSR_16	4	30	1	1	0.250	75	0.000383	0.999367
CA_24_LFSR_26	4	90/150	1	1	0.250	77	0.000193	0.999124
CA_24_LFSR_26	4	30	1	1	0.250	77	0.001187	0.998986
CA_52_LFSR_0	4	30	1	0	0.250	79	0.000136	0.999260
CA_37_LFSR_16	4	30	1	1	0.250	80	0.000122	0.999306
CA_32_LFSR_37	4	30	1	1	0.250	96	0.000129	0.999312
CA_32_LFSR_37	4	90/150	1	1	0.250	96	0.000165	0.999096
CA_32_LFSR_52	4	30	1	1	0.250	111	0.000101	0.999446
CA_32_LFSR_52	4	90/150	1	1	0.250	111	0.000236	0.999309
CA_48_LFSR_37	4	90/150	1	1	0.250	112	0.000120	0.999352
CA_48_LFSR_37	4	30	1	1	0.250	112	0.000157	0.999462
CA_48_LFSR_52	4	30	1	1	0.250	127	0.000222	0.999359
CA_48_LFSR_52	4	90/150	1	1	0.250	127	0.000381	0.999154
CA_32_LFSR_16	8	30	3	2	0.375	75	0.000685	0.999175
CA_24_LFSR_26	8	90/150	3	2	0.375	77	0.000140	0.999247
CA_24_LFSR_26	8	30	3	2	0.375	77	0.000190	0.999129
CA_24_LFSR_26	8	30	2	3	0.375	77	0.000322	0.999440
CA_37_LFSR_16	8	30	3	2	0.375	80	0.000090	0.999491
CA_32_LFSR_37	8	30	2	3	0.375	96	0.000092	0.999592
CA_32_LFSR_37	8	90/150	2	3	0.375	96	0.000131	0.999613
CA_32_LFSR_37	8	30	3	2	0.375	96	0.000159	0.999203
CA_32_LFSR_37	8	90/150	3	2	0.375	96	0.000239	0.999300
CA_32_LFSR_52	8	90/150	3	2	0.375	111	0.000110	0.999458
CA_32_LFSR_52	8	30	2	3	0.375	111	0.000119	0.999377
CA_32_LFSR_52	8	30	3	2	0.375	111	0.000254	0.999299

CA_32_LFSR_52	8	90/150	2	3	0.375	111	0.000439	0.999238
CA_48_LFSR_37	8	30	3	2	0.375	112	0.000152	0.999444
CA_48_LFSR_37	8	30	2	3	0.375	112	0.000243	0.999388
CA_48_LFSR_37	8	90/150	3	2	0.375	112	0.000243	0.999277
CA_48_LFSR_37	8	90/150	2	3	0.375	112	0.000414	0.999373
CA_48_LFSR_52	8	30	2	3	0.375	127	0.000079	0.999546
CA_48_LFSR_52	8	90/150	2	3	0.375	127	0.000102	0.999395
CA_48_LFSR_52	8	30	3	2	0.375	127	0.000151	0.999443
CA_48_LFSR_52	8	90/150	3	2	0.375	127	0.000375	0.999416
CA_37_LFSR_0	4	30	2	0	0.500	64	0.000182	0.999320
CA_20_LFSR_20	2	30	1	1	0.500	67	0.000193	0.999079
CA_48_LFSR_0	2	30	1	0	0.500	75	0.000097	0.999558
CA_32_LFSR_16	2	30	1	1	0.500	75	0.000202	0.999114
CA_0_LFSR_48	2	0	0	1	0.500	75	0.000481	0.997805
CA_24_LFSR_26	2	30	1	1	0.500	77	0.000163	0.999351
CA_24_LFSR_26	2	90/150	1	1	0.500	77	0.000236	0.999493
CA_52_LFSR_0	4	30	2	0	0.500	79	0.000147	0.999277
CA_52_LFSR_0	2	30	1	0	0.500	79	0.000265	0.999340
CA_37_LFSR_16	2	30	1	1	0.500	80	0.000112	0.999495
CA_16_LFSR_52	2	30	1	1	0.500	95	0.000111	0.999419
CA_32_LFSR_37	2	90/150	1	1	0.500	96	0.000161	0.999193
CA_32_LFSR_37	2	30	1	1	0.500	96	0.000264	0.999300
CA_32_LFSR_52	2	30	1	1	0.500	111	0.000168	0.999243
CA_32_LFSR_52	2	90/150	1	1	0.500	111	0.000196	0.999149
CA_48_LFSR_37	2	90/150	1	1	0.500	112	0.000152	0.999284
CA_48_LFSR_37	2	30	1	1	0.500	112	0.000156	0.999142
CA_48_LFSR_52	2	30	1	1	0.500	127	0.000092	0.999536
CA_48_LFSR_52	2	90/150	1	1	0.500	127	0.000158	0.999187
CA_0_LFSR_48	8	0	0	5	0.625	75	0.000149	0.999282
CA_48_LFSR_0	8	30	5	0	0.625	75	0.000221	0.999436
CA_52_LFSR_0	8	30	5	0	0.625	79	0.000123	0.999560
CA_32_LFSR_16	4	30	3	2	0.750	75	0.000179	0.999048
CA_24_LFSR_26	4	90/150	3	2	0.750	77	0.000083	0.999556
CA_24_LFSR_26	4	30	2	3	0.750	77	0.000163	0.999255
CA_24_LFSR_26	4	90/150	2	3	0.750	77	0.000341	0.999046
CA_24_LFSR_26	4	30	3	2	0.750	77	0.000476	0.998878
CA_0_LFSR_52	4	0	0	3	0.750	79	0.000155	0.999214
CA_52_LFSR_0	4	30	3	0	0.750	79	0.000156	0.999491
CA_37_LFSR_16	4	30	3	2	0.750	80	0.000084	0.999532
CA_16_LFSR_37	4	30	3	2	0.750	80	0.000175	0.999208
CA_37_LFSR_16	4	30	2	3	0.750	80	0.000185	0.998909
CA_16_LFSR_52	4	30	3	2	0.750	95	0.000120	0.999542
CA_16_LFSR_52	4	30	2	3	0.750	95	0.000134	0.999555
CA_16_LFSR_52	4	90/150	2	3	0.750	95	0.000195	0.998921
CA_32_LFSR_37	4	30	3	2	0.750	96	0.000073	0.999616
CA_32_LFSR_37	4	90/150	3	2	0.750	96	0.000082	0.999530

CA_32_LFSR_37	4	90/150	2	3	0.750	96	0.000178	0.999498
CA_32_LFSR_37	4	30	2	3	0.750	96	0.000238	0.999473
CA_32_LFSR_52	4	90/150	2	3	0.750	111	0.000159	0.999388
CA_32_LFSR_52	4	30	3	2	0.750	111	0.000177	0.999395
CA_32_LFSR_52	4	90/150	3	2	0.750	111	0.000197	0.999353
CA_32_LFSR_52	4	30	2	3	0.750	111	0.000216	0.999186
CA_48_LFSR_37	4	30	2	3	0.750	112	0.000102	0.999614
CA_48_LFSR_37	4	90/150	2	3	0.750	112	0.000181	0.999441
CA_48_LFSR_37	4	30	3	2	0.750	112	0.000253	0.998997
CA_48_LFSR_37	4	90/150	3	2	0.750	112	0.000262	0.999393
CA_48_LFSR_52	4	90/150	3	2	0.750	127	0.000060	0.999663
CA_48_LFSR_52	4	30	3	2	0.750	127	0.000114	0.999375
CA_48_LFSR_52	4	30	2	3	0.750	127	0.000142	0.999695
CA_48_LFSR_52	4	90/150	2	3	0.750	127	0.000144	0.999486
CA_37_LFSR_0	8	30	7	0	0.875	64	0.000078	0.999562
CA_32_LFSR_16	8	30	7	5	0.875	75	0.000136	0.999354
CA_48_LFSR_0	8	30	7	0	0.875	75	0.000264	0.999271
CA_0_LFSR_48	8	0	0	7	0.875	75	0.000307	0.998884
CA_32_LFSR_16	8	30	5	7	0.875	75	0.000399	0.999675
CA_24_LFSR_26	8	30	5	7	0.875	77	0.000072	0.999609
CA_24_LFSR_26	8	90/150	7	5	0.875	77	0.000081	0.999546
CA_24_LFSR_26	8	90/150	5	7	0.875	77	0.000127	0.999363
CA_24_LFSR_26	8	30	7	5	0.875	77	0.000157	0.999398
CA_37_LFSR_16	8	30	5	7	0.875	80	0.000103	0.999606
CA_16_LFSR_37	8	30	5	7	0.875	80	0.000164	0.999381
CA_37_LFSR_16	8	30	7	5	0.875	80	0.000197	0.999299
CA_32_LFSR_37	8	30	5	7	0.875	96	0.000117	0.999479
CA_32_LFSR_37	8	90/150	5	7	0.875	96	0.000149	0.999551
CA_32_LFSR_37	8	30	7	5	0.875	96	0.000173	0.999413
CA_32_LFSR_37	8	90/150	7	5	0.875	96	0.000312	0.998425
CA_32_LFSR_52	8	90/150	5	7	0.875	111	0.000123	0.999389
CA_32_LFSR_52	8	30	5	7	0.875	111	0.000155	0.999246
CA_32_LFSR_52	8	90/150	7	5	0.875	111	0.000216	0.999229
CA_32_LFSR_52	8	30	7	5	0.875	111	0.000266	0.999398
CA_48_LFSR_37	8	90/150	5	7	0.875	112	0.000118	0.999376
CA_48_LFSR_37	8	30	7	5	0.875	112	0.000126	0.999388
CA_48_LFSR_37	8	30	5	7	0.875	112	0.000146	0.999169
CA_48_LFSR_37	8	90/150	7	5	0.875	112	0.000178	0.999580
CA_48_LFSR_52	8	30	7	5	0.875	127	0.000115	0.999488
CA_48_LFSR_52	8	90/150	7	5	0.875	127	0.000153	0.999263
CA_48_LFSR_52	8	30	5	7	0.875	127	0.000159	0.999365
CA_48_LFSR_52	8	90/150	5	7	0.875	127	0.000697	0.999111
CA_37_LFSR_0	1	30	1	0	1.000	64	0.000127	0.999286
CA_0_LFSR_48	1	0	0	1	1.000	75	0.000100	0.999449
CA_48_LFSR_0	2	30	2	0	1.000	75	0.000165	0.999320
CA_48_LFSR_0	1	30	1	0	1.000	75	0.000183	0.999457

CA_32_LFSR_16	1	30	1	1	1.000	75	0.000184	0.999506
CA_24_LFSR_26	1	30	1	1	1.000	77	0.000155	0.999437
CA_24_LFSR_26	1	90/150	1	1	1.000	77	0.000207	0.999063
CA_52_LFSR_0	1	30	1	0	1.000	79	0.000119	0.999427
CA_52_LFSR_0	2	30	2	0	1.000	79	0.000157	0.999329
CA_37_LFSR_16	1	30	1	1	1.000	80	0.000135	0.999392
CA_16_LFSR_52	1	30	1	1	1.000	95	0.000146	0.999305
CA_32_LFSR_37	1	90/150	1	1	1.000	96	0.000333	0.999399
CA_32_LFSR_37	1	30	1	1	1.000	96	0.000343	0.999054
CA_32_LFSR_52	1	90/150	1	1	1.000	111	0.000127	0.999269
CA_32_LFSR_52	1	30	1	1	1.000	111	0.000207	0.999065
CA_48_LFSR_37	1	90/150	1	1	1.000	112	0.000092	0.999577
CA_48_LFSR_37	1	30	1	1	1.000	112	0.000139	0.999452
CA_48_LFSR_52	1	30	1	1	1.000	127	0.000110	0.999452
CA_48_LFSR_52	1	90/150	1	1	1.000	127	0.000157	0.999536
CA_37_LFSR_0	4	30	5	0	1.250	64	0.001083	0.999324
CA_48_LFSR_0	4	30	5	0	1.250	75	0.000099	0.999575
CA_0_LFSR_48	4	0	0	5	1.250	75	0.000248	0.999249
CA_52_LFSR_0	4	30	5	0	1.250	79	0.000109	0.999384
CA_0_LFSR_52	4	0	0	5	1.250	79	0.000207	0.998922
CA_37_LFSR_0	2	30	3	0	1.500	64	0.000174	0.999035
CA_32_LFSR_16	2	30	2	3	1.500	75	0.000152	0.999312
CA_48_LFSR_0	2	30	3	0	1.500	75	0.000181	0.999043
CA_32_LFSR_16	2	30	3	2	1.500	75	0.000191	0.999132
CA_24_LFSR_26	2	90/150	2	3	1.500	77	0.000146	0.999251
CA_24_LFSR_26	2	90/150	3	2	1.500	77	0.000173	0.999401
CA_24_LFSR_26	2	30	2	3	1.500	77	0.000208	0.999389
CA_24_LFSR_26	2	30	3	2	1.500	77	0.000231	0.998959
CA_0_LFSR_52	2	0	0	3	1.500	79	0.000121	0.999336
CA_52_LFSR_0	2	30	3	0	1.500	79	0.000150	0.999598
CA_37_LFSR_16	2	30	3	2	1.500	80	0.000138	0.999295
CA_37_LFSR_16	2	30	2	3	1.500	80	0.000186	0.999312
CA_16_LFSR_37	2	30	2	3	1.500	80	0.001778	0.998619
CA_16_LFSR_52	2	30	2	3	1.500	95	0.000197	0.999378
CA_16_LFSR_52	2	90/150	2	3	1.500	95	0.000208	0.999521
CA_32_LFSR_37	2	30	3	2	1.500	96	0.000126	0.999380
CA_32_LFSR_37	2	90/150	3	2	1.500	96	0.000127	0.999361
CA_32_LFSR_37	2	90/150	2	3	1.500	96	0.000167	0.999489
CA_32_LFSR_37	2	30	2	3	1.500	96	0.000246	0.998695
CA_32_LFSR_52	2	90/150	2	3	1.500	111	0.000133	0.999587
CA_32_LFSR_52	2	30	2	3	1.500	111	0.000217	0.999500
CA_32_LFSR_52	2	90/150	3	2	1.500	111	0.000297	0.999230
CA_32_LFSR_52	2	30	3	2	1.500	111	0.000379	0.999417
CA_48_LFSR_37	2	90/150	2	3	1.500	112	0.000103	0.999600
CA_48_LFSR_37	2	90/150	3	2	1.500	112	0.000125	0.999426
CA_48_LFSR_37	2	30	2	3	1.500	112	0.000130	0.999354

CA_48_LFSR_37	2	30	3	2	1.500	112	0.000177	0.999606
CA_48_LFSR_52	2	90/150	3	2	1.500	127	0.000102	0.999482
CA_48_LFSR_52	2	30	2	3	1.500	127	0.000115	0.999549
CA_48_LFSR_52	2	90/150	2	3	1.500	127	0.000136	0.999508
CA_48_LFSR_52	2	30	3	2	1.500	127	0.000230	0.999275
CA_37_LFSR_0	4	30	7	0	1.750	64	0.000189	0.999197
CA_32_LFSR_16	4	30	7	5	1.750	75	0.000155	0.999485
CA_32_LFSR_16	4	30	5	7	1.750	75	0.000200	0.999380
CA_0_LFSR_48	4	0	0	7	1.750	75	0.000258	0.999517
CA_48_LFSR_0	4	30	7	0	1.750	75	0.000392	0.999317
CA_24_LFSR_26	4	30	7	5	1.750	77	0.000202	0.999425
CA_24_LFSR_26	4	90/150	7	5	1.750	77	0.000485	0.997784
CA_0_LFSR_52	4	0	0	7	1.750	79	0.000107	0.999458
CA_52_LFSR_0	4	30	7	0	1.750	79	0.000125	0.999271
CA_37_LFSR_16	4	30	5	7	1.750	80	0.000109	0.999450
CA_16_LFSR_37	4	30	5	7	1.750	80	0.000153	0.999307
CA_37_LFSR_16	4	30	7	5	1.750	80	0.000159	0.999359
CA_16_LFSR_37	4	30	7	5	1.750	80	0.000260	0.998745
CA_16_LFSR_52	4	30	7	5	1.750	95	0.000123	0.999574
CA_16_LFSR_52	4	90/150	7	5	1.750	95	0.000161	0.999195
CA_16_LFSR_52	4	90/150	5	7	1.750	95	0.000173	0.999030
CA_16_LFSR_52	4	30	5	7	1.750	95	0.000174	0.999468
CA_32_LFSR_37	4	90/150	5	7	1.750	96	0.000127	0.999437
CA_32_LFSR_37	4	30	5	7	1.750	96	0.000134	0.999361
CA_32_LFSR_37	4	30	7	5	1.750	96	0.000149	0.999234
CA_32_LFSR_37	4	90/150	7	5	1.750	96	0.000192	0.999317
CA_32_LFSR_52	4	90/150	5	7	1.750	111	0.000094	0.999593
CA_32_LFSR_52	4	90/150	7	5	1.750	111	0.000101	0.999462
CA_32_LFSR_52	4	30	7	5	1.750	111	0.000142	0.999536
CA_32_LFSR_52	4	30	5	7	1.750	111	0.000161	0.999150
CA_48_LFSR_37	4	90/150	5	7	1.750	112	0.000079	0.999716
CA_48_LFSR_37	4	30	7	5	1.750	112	0.000126	0.999260
CA_48_LFSR_37	4	90/150	7	5	1.750	112	0.000160	0.999433
CA_48_LFSR_37	4	30	5	7	1.750	112	0.000202	0.999302
CA_48_LFSR_52	4	90/150	5	7	1.750	127	0.000072	0.999575
CA_48_LFSR_52	4	30	5	7	1.750	127	0.000159	0.999279
CA_48_LFSR_52	4	90/150	7	5	1.750	127	0.000160	0.999329
CA_48_LFSR_52	4	30	7	5	1.750	127	0.000167	0.999250
CA_37_LFSR_0	1	30	2	0	2.000	64	0.000155	0.999149
CA_0_LFSR_48	1	0	0	2	2.000	75	0.000094	0.999500
CA_48_LFSR_0	1	30	2	0	2.000	75	0.000151	0.999447
CA_52_LFSR_0	1	30	2	0	2.000	79	0.000089	0.999517
CA_0_LFSR_37	2	0	0	5	2.500	64	0.000095	0.999445
CA_48_LFSR_0	2	30	5	0	2.500	75	0.000060	0.999740
CA_0_LFSR_48	2	0	0	5	2.500	75	0.000127	0.999507
CA_52_LFSR_0	2	30	5	0	2.500	79	0.000109	0.999438

CA_0_LFSR_52	2	0	0	5	2.500	79	0.000192	0.999433
CA_37_LFSR_0	1	30	3	0	3.000	64	0.000159	0.999397
CA_20_LFSR_20	1	30	3	2	3.000	67	0.000182	0.999165
CA_20_LFSR_20	1	30	2	3	3.000	67	0.000601	0.998310
CA_32_LFSR_16	1	30	2	3	3.000	75	0.000081	0.999547
CA_0_LFSR_48	1	0	0	3	3.000	75	0.000105	0.999453
CA_32_LFSR_16	1	30	3	2	3.000	75	0.000133	0.999581
CA_48_LFSR_0	1	30	3	0	3.000	75	0.000334	0.998889
CA_24_LFSR_26	1	30	2	3	3.000	77	0.000068	0.999598
CA_24_LFSR_26	1	90/150	3	2	3.000	77	0.000197	0.999362
CA_24_LFSR_26	1	90/150	2	3	3.000	77	0.000203	0.999434
CA_24_LFSR_26	1	30	3	2	3.000	77	0.000218	0.998755
CA_52_LFSR_0	1	30	3	0	3.000	79	0.000147	0.999330
CA_0_LFSR_52	1	0	0	3	3.000	79	0.000157	0.999255
CA_16_LFSR_37	1	90/150	2	3	3.000	80	0.000112	0.999524
CA_37_LFSR_16	1	30	2	3	3.000	80	0.000149	0.999394
CA_16_LFSR_37	1	30	2	3	3.000	80	0.000189	0.999507
CA_37_LFSR_16	1	30	3	2	3.000	80	0.000317	0.998918
CA_16_LFSR_52	1	30	2	3	3.000	95	0.000093	0.999456
CA_16_LFSR_52	1	30	3	2	3.000	95	0.000114	0.999475
CA_16_LFSR_52	1	90/150	2	3	3.000	95	0.000335	0.998937
CA_32_LFSR_37	1	30	2	3	3.000	96	0.000179	0.999360
CA_32_LFSR_37	1	30	3	2	3.000	96	0.000216	0.998942
CA_32_LFSR_37	1	90/150	2	3	3.000	96	0.000219	0.999454
CA_32_LFSR_52	1	90/150	2	3	3.000	111	0.000086	0.999558
CA_32_LFSR_52	1	30	2	3	3.000	111	0.000094	0.999472
CA_32_LFSR_52	1	90/150	3	2	3.000	111	0.000100	0.999418
CA_32_LFSR_52	1	30	3	2	3.000	111	0.000139	0.999329
CA_48_LFSR_37	1	30	2	3	3.000	112	0.000140	0.999411
CA_48_LFSR_37	1	90/150	2	3	3.000	112	0.000201	0.998821
CA_48_LFSR_37	1	90/150	3	2	3.000	112	0.000248	0.999201
CA_48_LFSR_37	1	30	3	2	3.000	112	0.000263	0.999387
CA_48_LFSR_52	1	30	3	2	3.000	127	0.000098	0.999493
CA_48_LFSR_52	1	30	2	3	3.000	127	0.000118	0.999455
CA_48_LFSR_52	1	90/150	3	2	3.000	127	0.000159	0.999381
CA_48_LFSR_52	1	90/150	2	3	3.000	127	0.000228	0.999200
CA_37_LFSR_0	2	30	7	0	3.500	64	0.000214	0.999224
CA_0_LFSR_37	2	0	0	7	3.500	64	0.000228	0.998939
CA_48_LFSR_0	2	30	7	0	3.500	75	0.000078	0.999581
CA_32_LFSR_16	2	30	7	5	3.500	75	0.000120	0.999460
CA_0_LFSR_48	2	0	0	7	3.500	75	0.000154	0.999314
CA_32_LFSR_16	2	30	5	7	3.500	75	0.000198	0.999353
CA_24_LFSR_26	2	30	7	5	3.500	77	0.000078	0.999625
CA_24_LFSR_26	2	30	5	7	3.500	77	0.000233	0.999039
CA_24_LFSR_26	2	90/150	7	5	3.500	77	0.000249	0.999358
CA_24_LFSR_26	2	90/150	5	7	3.500	77	0.000612	0.998785

CA_52_LFSR_0	2	30	7	0	3.500	79	0.000162	0.999369
CA_0_LFSR_52	2	0	0	7	3.500	79	0.000209	0.999125
CA_37_LFSR_16	2	30	5	7	3.500	80	0.000102	0.999415
CA_37_LFSR_16	2	30	7	5	3.500	80	0.000115	0.999390
CA_16_LFSR_37	2	90/150	5	7	3.500	80	0.000126	0.999464
CA_16_LFSR_37	2	30	5	7	3.500	80	0.000164	0.999389
CA_16_LFSR_37	2	90/150	7	5	3.500	80	0.000180	0.999011
CA_16_LFSR_37	2	30	7	5	3.500	80	0.000421	0.999332
CA_16_LFSR_52	2	90/150	7	5	3.500	95	0.000092	0.999526
CA_16_LFSR_52	2	30	7	5	3.500	95	0.000148	0.999527
CA_16_LFSR_52	2	30	5	7	3.500	95	0.000182	0.999003
CA_16_LFSR_52	2	90/150	5	7	3.500	95	0.000186	0.999228
CA_32_LFSR_37	2	30	7	5	3.500	96	0.000095	0.999463
CA_32_LFSR_37	2	90/150	5	7	3.500	96	0.000114	0.999325
CA_32_LFSR_37	2	30	5	7	3.500	96	0.000176	0.999173
CA_32_LFSR_37	2	90/150	7	5	3.500	96	0.000187	0.999135
CA_32_LFSR_52	2	30	5	7	3.500	111	0.000126	0.999509
CA_32_LFSR_52	2	90/150	5	7	3.500	111	0.000134	0.999507
CA_32_LFSR_52	2	90/150	7	5	3.500	111	0.000165	0.999431
CA_32_LFSR_52	2	30	7	5	3.500	111	0.000436	0.999426
CA_48_LFSR_37	2	30	7	5	3.500	112	0.000083	0.999520
CA_48_LFSR_37	2	30	5	7	3.500	112	0.000124	0.999325
CA_48_LFSR_37	2	90/150	5	7	3.500	112	0.000149	0.999451
CA_48_LFSR_37	2	90/150	7	5	3.500	112	0.000447	0.999573
CA_48_LFSR_52	2	90/150	5	7	3.500	127	0.000078	0.999599
CA_48_LFSR_52	2	30	7	5	3.500	127	0.000156	0.999528
CA_48_LFSR_52	2	90/150	7	5	3.500	127	0.000164	0.999422
CA_48_LFSR_52	2	30	5	7	3.500	127	0.000179	0.999212
CA_37_LFSR_0	1	30	5	0	5.000	64	0.000169	0.999392
CA_0_LFSR_37	1	0	0	5	5.000	64	0.000238	0.999318
CA_0_LFSR_48	1	0	0	5	5.000	75	0.000146	0.999543
CA_48_LFSR_0	1	30	5	0	5.000	75	0.000493	0.999055
CA_0_LFSR_52	1	0	0	5	5.000	79	0.000234	0.999447
CA_52_LFSR_0	1	30	5	0	5.000	79	0.000241	0.999343
CA_37_LFSR_0	1	30	7	0	7.000	64	0.000103	0.999466
CA_0_LFSR_37	1	0	0	7	7.000	64	0.000312	0.998996
CA_20_LFSR_20	1	30	7	5	7.000	67	0.000942	0.998547
CA_0_LFSR_48	1	0	0	7	7.000	75	0.000174	0.999493
CA_32_LFSR_16	1	30	7	5	7.000	75	0.000229	0.999468
CA_32_LFSR_16	1	30	5	7	7.000	75	0.000240	0.999458
CA_48_LFSR_0	1	30	7	0	7.000	75	0.000278	0.999183
CA_24_LFSR_26	1	90/150	7	5	7.000	77	0.000146	0.999183
CA_24_LFSR_26	1	30	5	7	7.000	77	0.000150	0.999250
CA_24_LFSR_26	1	30	7	5	7.000	77	0.000170	0.999245
CA_24_LFSR_26	1	90/150	5	7	7.000	77	0.000202	0.999223
CA_0_LFSR_52	1	0	0	7	7.000	79	0.000080	0.999603

CA_52_LFSR_0	1	30	7	0	7.000	79	0.000219	0.999387
CA_37_LFSR_16	1	30	5	7	7.000	80	0.000092	0.999460
CA_16_LFSR_37	1	30	5	7	7.000	80	0.000113	0.999384
CA_16_LFSR_37	1	90/150	5	7	7.000	80	0.000184	0.999219
CA_16_LFSR_37	1	90/150	7	5	7.000	80	0.000191	0.999160
CA_16_LFSR_37	1	30	7	5	7.000	80	0.000255	0.999341
CA_37_LFSR_16	1	30	7	5	7.000	80	0.000289	0.999354
CA_16_LFSR_52	1	90/150	7	5	7.000	95	0.000105	0.999421
CA_16_LFSR_52	1	30	5	7	7.000	95	0.000120	0.999511
CA_16_LFSR_52	1	90/150	5	7	7.000	95	0.000128	0.999444
CA_16_LFSR_52	1	30	7	5	7.000	95	0.000174	0.999258
CA_32_LFSR_37	1	30	5	7	7.000	96	0.000079	0.999598
CA_32_LFSR_37	1	30	7	5	7.000	96	0.000143	0.999627
CA_32_LFSR_37	1	90/150	7	5	7.000	96	0.000174	0.999009
CA_32_LFSR_37	1	90/150	5	7	7.000	96	0.000189	0.999339
CA_32_LFSR_52	1	30	7	5	7.000	111	0.000127	0.999434
CA_32_LFSR_52	1	90/150	5	7	7.000	111	0.000148	0.999273
CA_32_LFSR_52	1	90/150	7	5	7.000	111	0.000163	0.999277
CA_32_LFSR_52	1	30	5	7	7.000	111	0.000230	0.999532
CA_48_LFSR_37	1	30	5	7	7.000	112	0.000107	0.999552
CA_48_LFSR_37	1	30	7	5	7.000	112	0.000155	0.999503
CA_48_LFSR_37	1	90/150	5	7	7.000	112	0.000164	0.999467
CA_48_LFSR_37	1	90/150	7	5	7.000	112	0.000179	0.999221
CA_48_LFSR_52	1	90/150	5	7	7.000	127	0.000056	0.999674
CA_48_LFSR_52	1	30	7	5	7.000	127	0.000150	0.999258
CA_48_LFSR_52	1	30	5	7	7.000	127	0.000181	0.999368
CA_48_LFSR_52	1	90/150	7	5	7.000	127	0.000232	0.999276