Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Modelagem de um PUF em HDL para implementação em FPGA

Felipe Augusto Sodré Ferreira de Sousa

Campina Grande, PB
Outubro de 2019

Felipe Augusto Sodré Ferreira de Sousa

Modelagem de um PUF em HDL para implementação em FPGA

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Sistemas Embarcados e Segurança de Hardware

Orientador: Gutemberg Gonçalves dos Santos Júnior

Campina Grande, PB
Outubro de 2019

Felipe Augusto Sodré Ferreira de Sousa

Modelagem de um PUF em HDL para implementação em FPGA

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovado em ___/___ /___

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Gutemberg Gonçalves dos Santos Júnior
Universidade Federal de Campina Grande
Orientador

”Il n’y a qu’une seule façon d’échouer,
c’est d’abandonner avant d’avoir réussi.”
Georges Clemenceau

Agradecimentos

Agradeço primeiramente aos meus pais, Ianna e Antonio, que sempre fizeram de tudo para que nada me faltasse. A todos da minha família que sempre estiveram ao meu lado. Meus irmãos Fabrício, Marianna e Marinna, que sempre tornaram o ambiente em casa mais leve e que não seria possível imaginar minha vida sem eles.

Agradeço aos meus amigos Arthur, Ariadne, Breno, Baiano, Indio, Vini, Samuel, Rafael e muitos outros, que me deram vários momentos felizes nesses 5 anos de curso, e aos meus amigos Cabral, Iury, João, Júlio, Ricardo, Roberto, Lucas, Japa, que me acompanham desde minha infância.

Ao professor Gutemberg Gonçalves dos Santos Júnior, pela orientação deste Trabalho de Conclusão de Curso, por toda paciência, por ter me dado o apoio necessário, sanado minhas dúvidas, disponibilizado os equipamentos para meus experimentos, e todos os ensinamentos que me passou durante toda a graduação, seja dentro ou fora de sala de aula. Agradeço ao mestrando Rubens por ter me ajudado e participado de algumas etapas do projeto.

Agradeço a todos que fizeram parte dessa etapa da minha vida.

Resumo

Os investimentos em segurança de CI vem aumentando cada vez mais. A grande preocupação são os ataques cibernéticos que ocasionam grandes prejuízos financeiros e muitas vezes agravados com o roubo e a pirataria de IP. Uma forma de proteger os dispositivos é por meio da autenticação. Neste trabalho são apresentados o projeto e desenvolvimento de um Physical Unclonable Function em HDL, baseado no design de um circuito oscilador (Ring Oscillator), com o propósito de ser implementado em um módulo FPGA para demonstrar os ganhos relacionados a segurança do dispositivo.

Palavras chave:Autenticação, Segurança de Hardware, PUF, RO, HDL

Abstract

Investments in device security are increasing steadily. The major concern is cyber attacks that cause financial losses and often aggravated by IP piracy. One way to secure devices is through authentication. In this work we present the design and development of a Physical Unclonable Function in HDL, based on the design of a Ring Oscillator, with the purpose of being implemented in a FPGA to demonstrate the safety related gains of the device.

Keywords: Autentication, Hardware Security, PUF, RO,HDL

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CI	Circuito Integrado
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IP	Intellectual property
LCELL	Logic Cell
LE	Logic Element
LUT	Lookup Tables
PUF	Physical unclonable function
RO	Ring Oscillator
RTL	Register-Transfer Level
SoC	System on a chip

Sumário

1 Introdução

O uso de aparelhos conectados a Internet vem crescendo em todo o mundo. No Brasil cerca de 70% da população teve acesso a esta tecnologia em 2018 [1]. Com o aumento de pessoas conectadas, aumenta-se o interesse de hackers de buscarem informações. São constantes os casos de ameaças cibernéticas, desde simples perfis pessoais hackeados até ataques a grandes centros financeiros como no caso do Banco do Chile em 2018, onde milhões de dólares foram desviados para uma conta [2]. A diversidade e a sofisticação das técnicas utilizadas pelos hackers forçou o desenvolvimento de novas formas de proteção para dispositivos e sistemas.

A criptografia continua sendo uma ferramenta importante para garantir a segurança de um dispositivo, tal técnica é eficaz contra algumas formas de ataques externos, como para proteção de espaços da memória e ataques visando recuperar senhas. Porém algumas formas de ataques não são mais originadas de agentes externos, podendo ocorrer através programas maliciosos implantados em componentes utilizados na concepção do produto.

A globalização fortaleceu o intercâmbio de informações, integrando mercados e aproximando empresas. Este processo repercutiu no ciclo de vida dos circuitos integrados, onde diferentes empresas intervêm na concepção, no design, na fabricação e na montagem destes componentes visando obter o melhor custo/benefício.

Nos processos de produção de *System on a Chip* (SoC) há a presença de diferentes fornecedores de diferentes países tendo seus componentes/tecnologias implementados em um único dispositivo, fabricado em outro país, acarretando na melhoria do time to Market do produto, na complexidade do seu design e diminuição do seu custo de fabricação. Entretanto, este fenômeno culminou no crescimento de várias novas empresas investindo no setor, e com isso, maior vulnerabilidade das informações.

Para garantir a segurança de dispositivos várias empresas investem em formas de aumentar a complexidade e robustez dos sistemas contra ataques. A autenticação e autorização são procedimentos básicos que podem ocasionar uma maior segurança e evitar fragilidades nos protocolos de comunicação, principalmente conexão com a internet. Um exemplo da importância da segurança de dispositivos é o caso em que foram detectadas vulnerabilidades no sistema de segurança de marcapassos, dispositivo de aplicação médica que tem o objetivo de regular os batimentos cardíacos, de tal forma que um hacker conseguiria controlar o funcionamento deste dispositivo, podendo ocasionar a morte do paciente [3].

A pirataria e o roubo de IP (Propriedade Intelectual), falsificação de componentes e inserção de circuitos maliciosos são apenas algumas formas que podem por em cheque a confiabilidade dos CI, necessitando assim técnicas capazes de minimizar essas ameaças e garantir a autenticidade dos SoC e proteção dos sistemas.

Diante do exposto este Trabalho de Conclusão de Curso tem como objetivo o desenvolvimento de uma medida de melhoramento da segurança de dispositivos em nível de hardware.

1.1 Objetivos

Modelagem de *Physical Unclonable Function* em linguagem de Hardware para implementação em uma placa FPGA de modo a caracterizar os ganhos relacionados a segurança e autenticação de dispositivos.

1.1.1 Objetivos Específicos

- Pesquisar e analisar as principais arquiteturas de PUF presentes na literatura.
- Realizar o projeto em Verilog de PUF para implementar em um modulo FPGA Altera® Cyclone V.
- Validar a implementação e demonstrar os ganhos relacionados a aplicação.
- Implementar o mesmo projeto em um segundo modulo FPGA Altera® Cyclone V para comprovar a autenticação e unicidade dos dispositivos.

1.2 Organização do Trabalho

O trabalho está estruturado em 5 capítulos, incluindo este introdutório, conforme a seguir.

- Capítulo 2: Fundamentação Teórica. Neste capítulo são apresentados os conceitos relevantes para o entendimento do trabalho;
- Capítulo 3: Modelagem: Aqui é descrito detalhadamente como projeto de modelagem foi desenvolvido;
- Capítulo 4: Resultados: Nesse capítulo são apresentados e discutidos os resultados do trabalho.
- Capítulo 5: Conclusão: Aqui são apresentadas as conclusões do trabalho.

2 Fundamentação Teórica

Este capítulo tem como objetivo apresentar os principais conceitos e fundamentos relacionados aos conceitos de segurança de hardware, as principais técnicas de criptografia, alguns tipos de ataques utilizados para recuperar informações de dispositivos e apresentação da definição e de algumas categorias de PUF.

2.1 Segurança de Sistemas Embarcados

Sistema embarcado é um conjunto de hardware e software desenvolvidos com o intuito de realizar uma ou mais tarefas de forma mais específica, visando otimizar o consumo e a performance.

Para que um sistema embarcado seja considerado seguro é necessário que todas as camadas sejam confiáveis, ou seja, o sistema operacional, hardware e software, possuam garantias que as informações processadas estejam protegidas. Os três pilares para a segurança da informação são: confidencialidade, autenticidade e integridade.

- Confidencialidade: garantir que as informações sejam acessadas apenas por processos autorizados;
- Autenticidade: garantir que as fontes de dados sejam confiáveis;
- Integridade: garantir que as informações não sejam modificadas.

Quando todas essas condições são satisfeitas o sistema torna-se mais resiliente a ataques, garantindo que seja mais difícil recuperar informações e limitando o dano potencial de ataques cibernéticos. Para se obter um sistema mais robusto é preciso que a segurança seja considerada em todas as etapas, como já mencionado no capítulo 1, o processo de produção dos SoC tem a intervenção de diferentes empresas nas etapas do ciclo de vida, sendo usual a concepção, fabricação e montagem realizadas separadamente, tornando um processo vulnerável à falhas de segurança.

Nas etapas de concepção e fabricação o sistema está sujeito ao roubo e modificações de IP, seja por um algoritmo mal implementado, uso de bibliotecas de terceiros com falhas de segurança ou robustez do protocolo escolhido. O desenvolvimento de sistemas operacionais personalizados e utilização de protocolos de comunicação seguros são alguns dos fatores que podem ajudar na proteção dos sistemas mas não garantem a total confiabilidade.

Os principais desafios da segurança estão ligados a constante evolução da tecnologia dos SoC, desenvolvendo sistemas mais complexos e mais heterogêneos. A internet das coisas (IoT) acelerou ainda mais a necessidade do investimento em segurança. A grande interconectividade é a motivação deste trabalho para estudar uma forma de garantir a autenticidade dos dispositivos.

A forma mais usual de esquemas de autentificação é utilizando chaves armazenadas em unidades de memória não-volátil. Uma forma de tentar garantir a segurança dessas chaves é por meio da criptografia.

2.2 Criptografia

A técnica de criptografar teve grande evolução com os avanços da telecomunicação, porém podemos considerar que trata-se de uma forma de comunicação muito importante na história da humanidade.

Durante as guerras já se usava métodos de esconder informações durante as transmissões das mensagens, e ao mesmo tempo já se buscavam métodos de desvendar as mensagens, como por exemplo o uso da Máquina de Turing. Fazendo a analogia, as pequisas de como se defender de ataques de ”hackers”já existia.

Código Morse e sinais de fumaça também podem ser considerados como formas de criptografar a informação, dado que é necessário o conhecimento de um ”segredo”(chave), para traduzir os sinais em palavras.

A criptografia é uma função matemática que codifica dados para que apenas dispositivos/usuários que conheçam a chave possam ter acesso a informação correta. Existem dois métodos de criptografia: criptografia simétrica e assimétrica. Os principais termos relacionados ao processo de criptografia são[4]:

- Plaintext - texto puro: mensagem em seu formato natural;
- Cyphertext - texto cifrado: mensagem criptografada;
- Key - chave: sequência que controla as operações matemáticas e o comportamento do algoritmo de criptografia;
- Keyspace - espaço de chaves: Conjunto dos possíveis valores da chave no algoritmo.

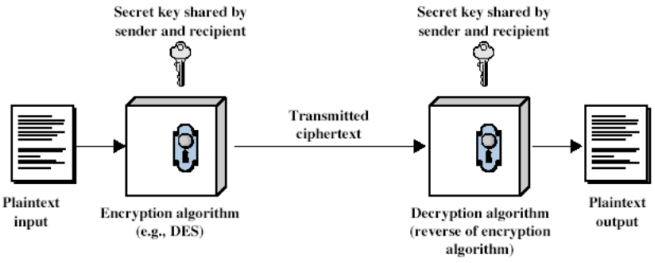
Pelo princípio de Kerckhoff, o único segredo no processo de criptografia deve ser a chave. O objetivo é garantir a confiabilidade, autenticação e integrabilidade do processo de comunicação.

2.2.1 Chave Simétrica

Algoritmos de chave simétrica são algoritmos para criptografia que usam a mesma chave criptográfica para encriptação de texto puro e descriptografar de texto cifrado. A chave, na prática, representa um segredo compartilhado entre duas ou mais partes que pode ser usado para manter uma ligação de informação privada. A figura 1 apresenta a estrutura básica da arquitetura da criptografia usando chave simétrica.

A operação de chave simétrica é mais simples, pois existe uma única chave entre as operações. A chave, na prática, representa um segredo, partilhado entre duas ou mais partes, que podem ser usadas para manter um canal confidencial de informação. Usa-se uma única chave, partilhada por ambos os interlocutores, na premissa de que esta é conhecida apenas por eles.

Figura 1: Estrutura do fluxo de informação na criptografia simétrica



Fonte: [4]

Entre os vários algoritmos de criptografia simétrica já desenvolvidos, podemos destacar DES e AES:

- DES - *Data Encryption Standard*

  É uma das primeiras criptografias utilizadas e é considerada uma proteção básica de poucos bits (cerca de 56). O seu algoritmo é o mais difundido mundialmente e realiza 16 ciclos de codificação para proteger uma informação.

  A complexidade e o tamanho das chaves de criptografia são medidos em bits, quanto maior o número, mais elevada será a segurança. Estima-se para cada bit adicionado o tempo necessário para decifrar a chave é multiplicado por 2. Se para descriptografar a chave de um algoritmo de 2 bits é necessário 2h, se considerarmos um acréscimo de 1 bit no espaço de chaves desse mesmo algoritmo, o mesmo tipo de ataque necessitaria de 4h.

- AES - *Advanced Encryption Standard*

  É o algoritmo padrão do governo dos Estados Unidos e de várias outras organizações. Ele é confiável e excepcionalmente eficiente na sua forma em 128 bits, mas também é possível usar chaves e 192 e 256 bits para informações que precisam de proteção maior.

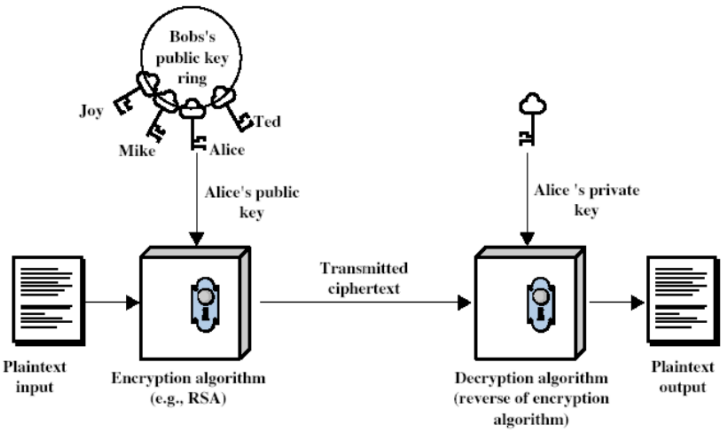
  O AES é amplamente considerado imune a todos os ataques, exceto aos ataques de injeção de faltas, que tentam decifrar o código em todas as combinações possíveis em 128, 192 e 256 bits, o que é imensamente difícil na atualidade.

Este requisito de que ambas as partes possuam acesso à mesma chave secreta é uma das principais desvantagens da criptografia de chave simétrica, em comparação com a criptografia de chave pública (também conhecida como criptografia de chave assimétrica) pois utilizam duas chaves (pública e privada).

2.2.2 Chave Assimétrica

A criptografia assimétrica, ou criptografia de chave pública usa o que é chamado de um ”par de chaves”– uma chave pública para criptografar a mensagem e uma chave privada para decifrar a mensagem. A figura 2 apresenta a estrutura básica da arquitetura da criptografia usando chave assimétrica com o uso do algoritmo RSA.

Figura 2: Estrutura do fluxo de informação na criptografia assimétrica



Fonte: [4]

Esse é considerado um dos algoritmos mais seguros do mercado, por essa razão também foi o primeiro a possibilitar a criptografia na assinatura digital.

O RSA funciona da seguinte forma: ele cria duas chaves diferentes, uma pública e outra privada (que deve ser mantida em sigilo). Todas as mensagens podem ser cifradas pela pública, mas somente decifradas pela privada.

2.3 Tipos de Ataques

A criptoanálise são técnicas que visam a obtenção da chave de criptografia, visando decifrar os cyphertext. Os métodos podem ser classificados em ataques passivos ou ataques ativos.

Ataques passivos são aqueles que os parâmetros físicos do sistema são observados sem interferir no funcionamento. Alguns exemplos são os ataques de Canal Lateral (*Side Channel Attack*) que se baseiam na observação do consumo, tempo de execução e emissão eletromagnética.

Os ataques passivos podem modificar de forma temporária, ou até mesmo definitiva, o estado de funcionamento do sistema. Utiliza-se de meios que possam ocasionar mudanças do ponto de operação, para então recuperar a chave. Com esses tipos de ataque, busca-se interferir no tempo do clock e nos flip-flops utilizados.

Alguns tipos de ataques são feitos durante o processo de produção, como a inserção de circuitos maliciosos, pirataria dos IP, ocasionando problemas de confiabilidade. Outra forma de recuperar informações é a reutilização de portas físicas de debug/teste dos circuitos, como JTAG.

2.3.1 Ataques Canal Lateral

Os parâmetros de um circuito integrado, consumo, tempo de execução, entre outros, estão correlacionados com o tipo de tarefa a ser executada. Estes tipos de ataques são muito comuns por não necessitarem de muitos recursos e por não ser invasivo ao sistema, não ocasionando danos físicos. É possível a realização destes ataques com o uso de um osciloscópio e um processador.

A análise consiste em recuperar informações sobre a atividade do sistema, como na escritura em espaços da memória, e analisar as medições de acordo com os dados processados no sistema.

Uma maneira amplamente difundida é a utilização de curvas elípticas para criação de um algoritmo para análise das curvas, recuperando assim a chave[5]. Em resumo, se mede o consumo do microcontrolador durante a execução do algoritmo e correlaciona com a sequência de bits de teste utilizados no código. A curva total de consumo permite identificar os bits que compõe a chave[6].

2.3.2 Ataques Injeção de Faltas

Baseados em inserir perturbações nos circuitos para provocar um erro interno que será em seguida explorado para recuperar a chave. O invasor possui como principais focos a modificação do fluxo de execução do programa e modificação dos dados durante escritura e leitura em blocos de memória.

Os principais métodos de injeção de faltas em SoC é a variação da tensão de alimentação, variação da frequência do clock, exposição a fortes campos eletromagnéticos e uso de laser.

O laser é utilizado pois a corrente gerada pelos feixes de carga permitem de controlar a polarização dos transistores, logo, potencialmente modificar os valores de um sinal.

Em alguns casos a porta JTAG é acessível após a concepção do produto. Para tentar mitigar a atuação dessas ameaças, além de outros tipos de ataques que podem tornar a parte Hardware do sistema embarcado vulnerável, é comum a implementação de PUF (Physical Unclonable Function).

2.4 Physical Unclonable Function

Com os avanços da IoT é essencial para os CI poderem executar operações como autenticação de dispositivos, proteção de informações confidenciais e comunicação segura de uma maneira barata e altamente segura. Como já mencionado, a técnica de autenticação armazenando chaves em espaço da memória não-volátil tornam o dispositivo vulnerável a ataques invasivos. O estudo de PUFs aumentou substancialmente tornando-os um tópico importante no campo da segurança de hardware.

O PUF consiste em um circuito desenvolvido para garantir a autenticação dos CI e melhorar na segurança dos dados. Como o próprio nome define, serve como uma identidade única para um CI. Devido a inerentes variações aleatórias do processo de fabricação, nenhum circuito integrado, mesmo com os mesmos layouts, é idêntico[7].

Na literatura existem diversas topologias ou formas de implementação de uma PUF [8]. Basicamente é uma função que mapeia suas entradas (desafios) em saídas (resposta), que apresentam um comportamento pseudoaleatório, sendo assim imprevisível mesmo para um invasor com acesso físico, como por exemplo a porta JTAG.

Um desafio aplicado e sua resposta medida geralmente são chamados de par desafio-resposta ou CRP e a relação imposta entre desafios e respostas de um PUF em particular é referido como seu comportamento de CRP[8]. Em um cenário típico de aplicativo, um PUF é usado em duas fases distintas:

1. *Inscrição*, um número de CRPs é coletado de um PUF específico e armazenado no chamado banco de dados de CRP.

2. *Verificação*, um desafio do banco de dados CRP é aplicado ao PUF e a resposta produzida pelo PUF é comparada com a resposta correspondente do banco de dados.

A forma em que o mapeamento é feito pode ser implícita na construção da arquitetura ou, como para a maioria dos PUFs, são aplicadas várias etapas de pós-processamento, fazendo com que nem sempre seja claro em que ponto a resposta é considerada [8].

Para se analisar a robustez do PUF são consideradas 2 váriaveis principais[8]:

- Inter-chip (μ_{inter}): Variação de bits da resposta de 2 PUF para um mesmo desafio;
- Intra-chip (μ_{intra}): Variação de bits da resposta do mesmo PUF para o mesmo desafio;

Interpretando os conceitos, podemos concluir que μ_{intra} expressa a noção de ruído médio nas respostas, isto é, mede a reprodutibilidade média de uma resposta medida em relação a uma observação anterior da mesma resposta.

Por outro lado, μ_{inter} expressa uma noção de singularidade, isto é, a unicidade da respostas PUF, reafirmando as variações inerentes do processo de produção.

Para um sistema robusto, espera-se um μ_{intra} próximo à 0% e μ_{intra} próximo de 50%.

Os PUF podem ser utilizados para proteção dos IP, geração e armazenamento de chaves criptográficas, identificação de dispositivo e autenticação. As principais propriedades do PUF são descritas abaixo [9]:

- Singularidade;
- Não clonável;
- Imprevisibilidade;
- Robustez.

A impossibilidade de clonagem é uma característica fundamental de um PUF e indica que é uma tarefa muito difícil e demorada construir dois circuitos PUF idênticos, que respondem da mesma forma aos mesmos desafios. Além disso, a impossibilidade de clonagem indica que, é muito difícil e praticamente impossível construir um modelo matemático preciso de um PUF, que calcule as respostas aos desafios escolhidos sem usar o próprio circuito do PUF. Como as variações do processo são incontroláveis, o sistema consegue garantir que mesmo infinitos CRPs conhecidos, a resposta a um novo desafio ainda deve ser imprevisível.

Os PUF podem ser classificados em categorias de acordo com seus princípios de funcionamento e arquitetura do design. Os modelos mais mencionados nas literaturas são baseados em características intrínsecas do CI SPUF (silicon PUF). Esta categoria ainda pode ser dividida em dois grupos, baseados na memória e baseados no delay [8].

Como o objetivo deste trabalho é a implementação deste circuito em um FPGA, foi concluído que a utilização de PUF baseados na memória não apresentavam resultados satisfatórios para esta aplicação. Ao se alimentar o FPGA, a configuração das células de memória são inicializadas, não garantindo a estabilidade da resposta do PUF.

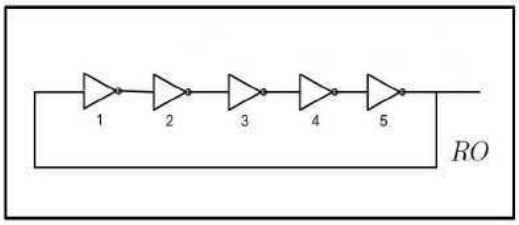
Dentre as arquiteturas baseadas em delay, podemos destacar Arbiter PUF[10], Butterfly PUF[11] e RO PUF[10]. Foi escolhido a implementação de um RO por apresentar uma arquitetura mais simples, facilitando a análise. Entretanto, apresenta frequências menores em relação as outras duas arquiteturas e consome mais do processador. No próximo capítulo será apresentada detalhadamente o funcionamento do circuito RO.

2.4.1 Ring-Oscillator

O RO é um circuito simples que oscila com uma frequência particular. Devido à variação de fabricação, cada oscilador oscila com uma frequência ligeiramente diferente. Para gerar um número fixo de bits, uma sequência fixa de pares de osciladores é selecionada e suas frequências são comparadas para gerar um bit de saída[9].

Os bits de saída da mesma sequência de comparação de pares de osciladores variam de chip para chip. Dado que os osciladores são dispostos de forma idêntica, as diferenças de frequência são determinadas pela variação de fabricação. Um bit de saída é igualmente provável que seja um ou zero se as variações aleatórias dominarem [12].

Figura 3: Arquitetura do Ring Oscillator

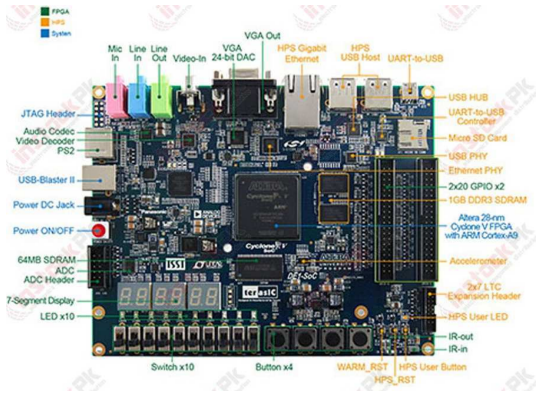


Fonte: [9]

2.5 Altera® Cyclone V DE1-Soc

Neste trabalho foram utilizados FPGA Altera® Cyclone V. Os FPGAs do Altera Cyclone V são compostos por uma matriz de blocos de matriz lógica (LAB). Cada LAB contém 10 módulos lógicos adaptativos (ALM), cada um contendo 2 LUT [13].

Figura 4: FPGA Altera® Cyclone V



Fonte: [13].

Foi escolhido utilizar a linguagem Verilog para para descrever o hardware. Esta linguagem permite a descrição em três níveis diferentes de abstração:

- Nível comportamental - descreve apenas como o hardware deve se comportar sem qualquer referência ao hardware digital.
- Register-Transfer-Level (RTL) - Aqui a descrição assume a existência de registradores e estes são sincronizados com o sinal do relógio. Portanto, os dados digitais são transferidos de um registro para o próximo em ciclos de relógio subsequentes.
- Nível de porta (Gate-level) - esta é a descrição de baixo nível em que cada porta é descrita e como elas são conectadas é especificada.

Verilog não é apenas uma linguagem de especificação que informa ao sistema CAD o que o hardware deve fazer, mas também inclui um ambiente de simulação completo. Um compilador Verilog faz mais do que mapear seu código para hardware, mas também pode simular (ou executar) seu design para prever o comportamento do seu circuito. É a linguagem predominante usada para o design de CI.

3 Modelagem

Neste capítulo estão descritas as etapas relativas ao projeto de desenvolvimento da arquitetura para implementação. Na primeira seção será apresentado o estudo realizado para a codificação e testes de frequências para os RO. Em seguida será apresentado a lógica utilizada para o mapeamento dos desafios e a estrutura do módulo denominado PUF para obtenção dos bits de respostas e, por fim, a modelagem e arquitetura do projeto.

3.1 Arquitetura Ring-Oscillator

A primeira etapa consiste da concepção de um Ring-Oscillator para ser implementado no módulo FPGA. Como apresentado anteriormente no capítulo 2, a arquitetura e o funcionamento desse circuito é relativamente simples.

A primeira alternativa para implementação o RO foi realizar uma combinação de um número impar de inversores. No entanto, os primeiros resultados dessa implementação não foram satisfatórios pois a saída dos RO obtida era sempre a entrada barrada, ou seja, toda arquitetura era concatenada a um uma porta lógica inversora.

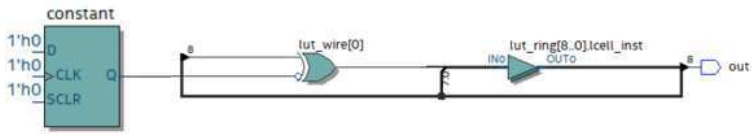
Foi possível concluir que as ferramentas FPGA não são projetadas para esse tipo de circuito buscando otimizar a lógica e reduzindo a arquitetura para uma forma mais simplificada quando analisado o RTL. Para superar este funcionamento poderia ser utilizado um PLL para gerar um clock interno de frequência controlada, porém esta solução reduz a variabilidade decorrente dos processos físicos de produção.

Analisando os guias de utilização da Altera Cooporation encontrou-se uma solução para implementação da arquitetura desejada.

O design HDL de baixo nível é a prática de usar primitivas e atribuições de baixo nível no código HDL para ditar uma implementação de hardware específica para uma parte da lógica. Primitivas de baixo nível são pequenos blocos de arquitetura que ajudam a obter uma melhor utilização de recursos ou resultados de tempo mais rápidos. A primitiva LCELL permite a divisão do projeto em partes gerenciáveis e impede que o mecanismo de síntese do software Quartus® simplifique a lógica desejada [14].

Foi utilizado um projeto encontrado em fóruns no site da Intel como base para o desenvolvimento do nosso RO. O projeto do fórum propunha uma arquitetura baseada em uma associação de uma sequencia de inversores com uma porta XOR capaz de ser implementada em um módulo FPGA e gerar uma frequência de oscilação própria[15]. Este projeto foi escolhido por utilizar LCELL no controle das portas lógicas.

Figura 5: Arquitetura do RO baseado no código do fórum



Fonte: Autoria Própria.

O oscilador é constituído de um loop de 8 inversores com uma realimentação em uma porta XOR. Utiliza-se uma variável *constant* para controlar a porta XOR (Figura 5). O valor desta variável está travado em nível baixo 0, que é invertido para 1 na porta XOR, fazendo com que a porta XOR se comporte como o nono inversor no oscilador. A Tabela 1 apresenta que, considerando *constant* como sendo a entrada A, e a entrada B como sendo a realimentação do loop de inversores, como o valor lido pela XOR sempre será 1, é possível concluir o comportamento semelhante ao de um inversor.

Tabela 1: Tabela verdade porta XOR

A	B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Autoria Própria.

Para o estudo do RO foi desenvolvido um projeto em Verilog que consiste em comparar o número de oscilações produzidas pelo oscilador em relação à um período de tempo predeterminado baseado na frequência interna do FPGA de 50 MHz (Figura 6). As principais variáveis do projeto são descritas abaixo:

- *CLOCK_50* - clock 50 MHz do FPGA;
- *out* - saída do RO;
- *rst* - reset assíncrono acionado por um botão;
- *constant* - registrador de controle de oscilação do RO;
- *flag* - variável de interrupção do RO;
- *cnt* - contador de borda de subida do clock 50 MHz do FPGA;
- *cnt1* - contador de borda de subida da saída do RO;
- *display* - 6 displays hexadecimal de 7 segmentos disponíveis no módulo FPGA;

Dois contadores de 32 bits são utilizados para contar o número de vezes que o clock interno de 50 MHz e a saída do RO passam do nível baixo ao alto. Quando o contador *cnt* atinge um número de oscilação de referência, o *flag* recebe valor 0 interrompendo o funcionamento do RO. *Flag* e o valor barrado de *constant* estão associados por uma porta AND, tal que a saída pilota o RO, se Saída=0 o RO oscila e se Saída = 1, RO para de oscilar. A tabela verdade apresentada abaixo demonstra o comportamento da saída(lut_wire)(Figura 6).

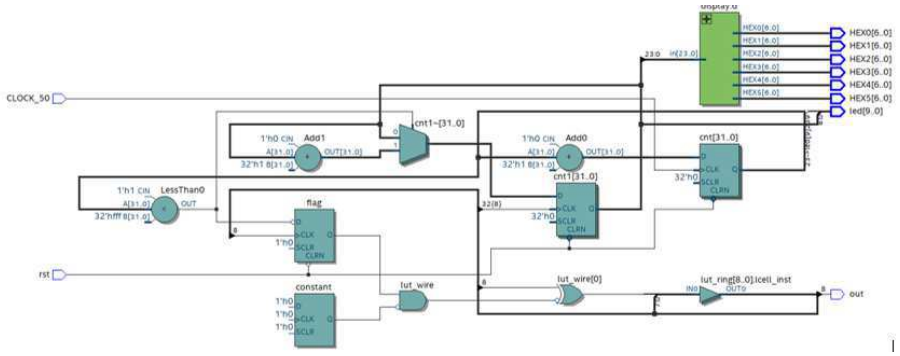
Tabela 2: Tabela verdade

constant	*flag*	Saída
0	0	0
0	1	1

Fonte: Autoria Própria.

Quando *flag* recebe valor 1, o valor *cnt1* é exibido no display. Foram analisados diversos cenários variando o valor de referência do clock de 50 MHz. Para calcular a frequência do RO é comparado o valor exibido no display com o valor de referencia adotado.

Figura 6: Projeto contendo 1 RO com frequência exibida nos Displays 7 segmentos



Fonte: Autoria Própria.

Supondo que seja analisado um período correspondente a 600 pulsos de clock de 50 MHz, ou seja, se um ciclo tem 20ηs, será analisado um período total T de 12μs. Utilizando a Tabela 3, calcula-se o valor de *cnt1* em decimal. Dividindo o valor calculado pelo valor de referência e multiplicando por 50x10^6, é estimado o valor da frequência do RO.

$$freq\_RO = \frac{cnt1 \cdot 50MHz}{cnt} \quad (1)$$

Tabela 3: Conversão dos valores exibidos no display para binário.

Representação Hexadecimal	Representação em Binário
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Fonte: Autoria Própria.

Alguns testes foram realizados variando o valor de referência entre 15, 30, 60, 120, 300 e 600 ciclos do clock de 50 MHz. Os valores apresentados nas Tabelas 4 a 9 foram os resultados exibidos nos displays para cada teste. Foram escolhidos o menor valor de frequência e o maior valor de frequência medido para o cálculo de $\Delta freq$ e o valor que mais vezes foi mostrado no display.

- 15 ciclos $\rightarrow$ cnt=4'b1111

Tabela 4: Análise da frequência do RO para 15 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
3F	0011 1111	63	210 MHz
40	0100 0000	64	213.33 MHz
42	0100 0010	66	220 MHz

Fonte: Autoria Própria.

$\Delta freq = 10MHz$ e frequência de 213.33 MHz o valor mais recorrente.

- 30 ciclos → cnt=5'b11110

Tabela 5: Análise da frequência do RO para 30 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
7F	0111 1111	127	211 MHz
80	1000 0000	128	213 MHz
82	1000 0010	130	216 MHz

Fonte: Autoria Própria.

$\Delta freq = 5MHz$ e frequência de 213 MHz o valor mais recorrente.

- 60 ciclos → cnt=6'b111100

Tabela 6: Análise da frequência do RO para 60 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
DF	1101 1111	223	185 MHz
E0	1110 0000	224	186 MHz
E1	1110 0001	225	187.5 MHz

Fonte: Autoria Própria.

$\Delta freq = 2.5MHz$ e frequência de 186 MHz o valor mais recorrente.

- 120 ciclos → cnt=7'b1111000

Tabela 7: Análise da frequência do RO para 120 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
1FE	0001 1111 1110	510	212 MHz
1FF	0001 1111 1111	511	212.9 MHz
202	0010 0000 0010	514	214.1 MHz

Fonte: Autoria Própria.

$\Delta freq = 2.1MHz$ e frequência de 212.9 MHz o valor mais recorrente.

- 300 ciclos → cnt=9'b100101100

Tabela 8: Análise da frequência do RO para 300 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
4AD	0100 1010 1101	1197	199.5 MHz
4AF	0100 1010 1111	1198	199.8 MHz
4B4	0100 1011 0100	1204	200.6 MHz

Fonte: Autoria Própria.

$\Delta freq = 1.1MHz$ e frequência de 199.8 MHz o valor mais recorrente.

- 600 ciclos → cnt=10'b1001011000

Tabela 9: Análise da frequência do RO para 600 ciclos do clock

Displays	Representação em binário	Número de ciclos	Frequência estimada
828	1000 0010 1000	2088	174 MHz
82A	1000 0010 1010	2090	174.16 MHz
82E	1000 0010 1110	2094	174.5 MHz

Fonte: Autoria Própria.

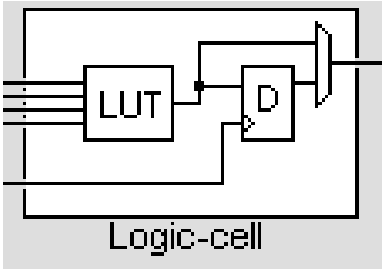
$\Delta freq = 0.5MHz$ e frequência de 174.16 MHz o valor mais recorrente.

A partir da análise dos testes realizados foi possível concluir que com o aumento do período de amostragem, a variação do valor da frequência calculada diminuía ($\Delta freq$). O menor valor de frequência calculado foi de 174.5 MHz e o maior valor foi de 220 MHz, havendo uma variação de 45 MHz para o mesmo RO.

Duplicando o número de pulsos do clock de 50 MHz, o *cnt1* não tem seu valor dobrado, ou seja, não há um comportamento linear. Tal comportamento, assim como a variação de 45 MHz dos valores calculados, é explicado pela alocação do RO no hardware do FPGA que é uma etapa do processo de implementação do projeto no módulo.

O processo de compilação no software Quartus® possui várias etapas, começando com a compilação do código HDL e termina com a configuração do hardware no FPGA. Como o próprio nome ”FPGA”sugere, é possível a programação de como as portas lógicas serão conectadas. Os FPGA são formados por um conjunto de elementos lógicos (LE). Estes blocos configuráveis são tipicamente constituídos de 4 entradas LUT, um flip-flop tipo D e um mux 2:1 (figura 7).

Figura 7: Estrutura dos Elementos Lógicos



Fonte: [16].

Um LUT trata-se de uma matriz ou vetor de dados para mapear valores de entrada para valores de saída. Ao se configurar o FPGA, o *Bitstream* é transferido a placa, determinando as funções lógicas que cada elemento lógico irá efetuar, assim como todas as interconexões, ou seja, a configuração dos LUT.

As etapas para compilação do software Quartus®, ou seja, geração do Bitstream são resumidamente apresentadas abaixo:

1. Compilação e análise: é verificado se há erros de syntax no código Verilog. Passagem da descrição em nível comportamental para estrutural (gate level);
2. Síntese: Otimização lógica e tradeoff entre utilização de elementos físicos e velocidade;
3. Mapeamento de Hardware: mapeamento da descrição estrutural para LE, flip-flops, blocos de memória, etc;
4. Place & Route: alocação e interconexão dos blocos;
5. Assembler: geração do *Bitstream* para ser transferido ao FPGA.

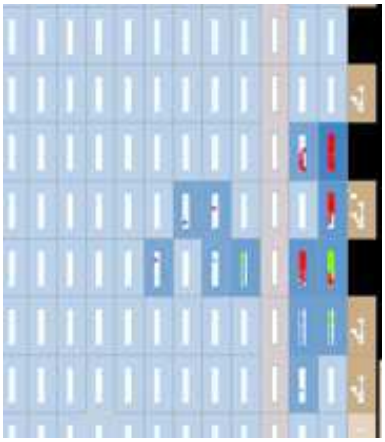
Para validar que a variação da frequência do RO corresponde ao local onde os LUTs serão configurados, foi realizado um teste com 2 RO. Com este projeto é possível de analisar que cada RO tem uma frequência de oscilação diferente e que mudanças na configuração são capazes de alterar a resposta de cada.

Foram utilizado os LEDs do FPGA para receberam os 5 primeiros bits do registrador *cnt1* de cada RO. Alterando apenas a ordem dos LEDS utilizados, o espaço onde os RO foram implementados mudam, variando a frequência de cada, como observado na figura 8.

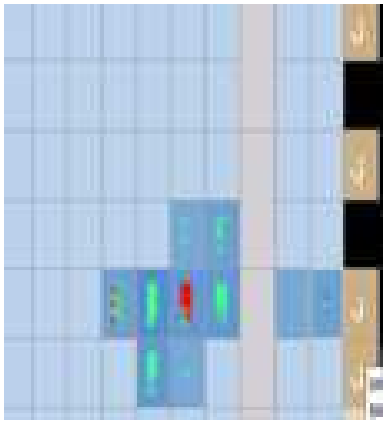
O Chip Planner é uma ferramenta de interface gráfica que permite criar regiões, editar regiões e fazer atribuições lógicas às regiões, ou seja, exibe os limites da arquitetura concebida e permite editar recursos do silício, como os pinos de entrada e saídas da placa, fornecendo o máximo controle sobre o posicionamento do projeto no hardware acarretando uma melhorara no desempenho.

Figura 8: Análise do Chip planner

(a) Chip planner para uma sequência de LEDS



(b) Chip planner com uma inversão de 2 LEDS



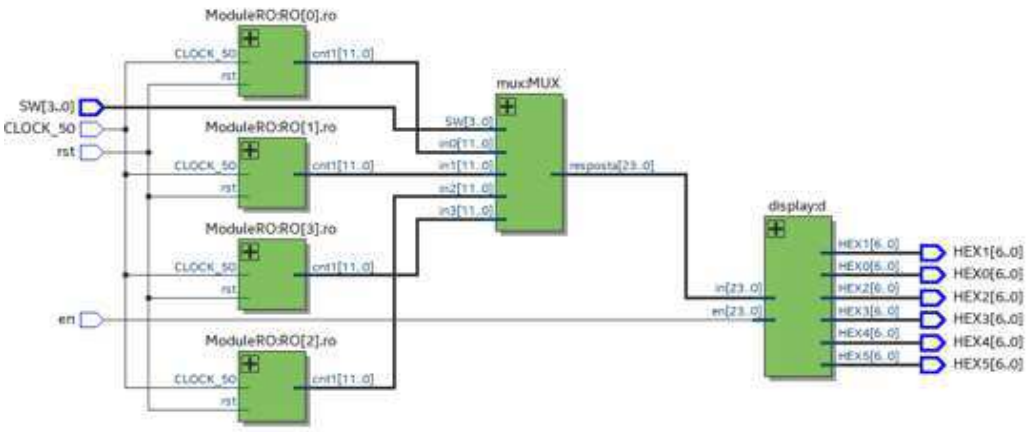
Fonte: Autoria Própria.

Os pontos vermelhos da figura 8 correspondem aos espaços onde os LUTs, utilizados para concepção do design do primeiro RO, foram configurados na placa FPGA. Os pontos verdes se referem aos LUTs utilizados no segundo RO.

As frequências obtidas para os RO em cada configuração foram diferentes, ou seja, comprovando que o espaço onde os LUTs são configurados influenciam na frequência de oscilação.

Afim de avaliar a arquitetura final dos RO e já iniciar o processo de desenvolvimento do mapeamento desafio/resposta do PUF, foi realizado um projeto com 4 RO conectados a um MUX 4:1 com chave seletora controlado por 2 switches (SW0 e SW1) do FPGA (Figura 9).

Figura 9: Comparação da frequência de 4 RO



Fonte: Autoria Própria.

A combinação dos switches permite selecionar qual RO será exibido no display de 7 segmentos, mostrando assim o valor correspondente ao seu *cnt1*. Para um valor de referência de 600 pulsos de clock, foi observado 60 MHz de diferença entre o RO mais rápido para o mais lento.

Tabela 10: Análise da frequência dos 4 RO selecionados pelo switches SW

SW	Display	Representação em binário	Número de ciclos	Frequência estimada
00	747	0111 0100 0111	1863	155. 25 MHz
01	882	1000 1000 0010	2178	181.5 MHz
10	A18	1010 0001 1000	2584	215.33 MHz
11	8C0	1000 1100 0000	2240	186.66 MHz

Fonte: Autoria Própria.

O mesmo projeto foi implementado em um segundo módulo FPGA para confirmar a variabilidade no processo de produção do SoC. Os resultados para um FPGA A e um FPGA B foram comparados na tabela 11. Nota-se que os valores das frequências são diferentes, sendo possível a utilização dessa unicidade para desenvolver o PUF.

Tabela 11: Comparação dos resultados da implementação do projeto em 2 FPGA

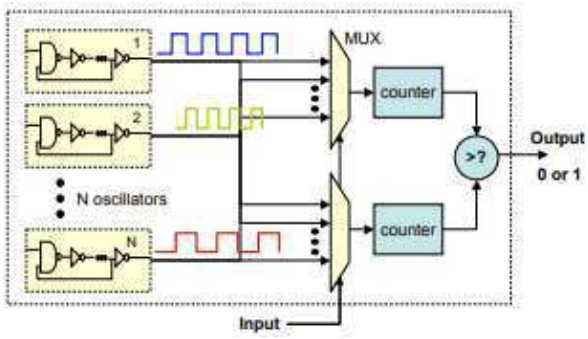
Ring Oscillator - SW	FPGA A	FPGA B
00	989	9BB
01	8A8	8CE
10	99E	9C0
11	A2A	A6A

Fonte: Autoria Própria.

3.2 Arquitetura PUF

Na literatura a estrutura mais utilizada para o mapeamento desafio/resposta de um PUF-RO é a utilização de um MUX N:1, onde N é o número de RO gerados na arquitetura. O desafio é utilizado como chave seletora do MUX. Ao se selecionar a saída do MUX, é implementado um contador de borda de subida para o cálculo da frequência estimada do RO. Duplica-se este circuito, de tal forma que se tenha 2 MUX N:1, para que as saídas sejam comparadas para se obter um bit de resposta (Figura 10).

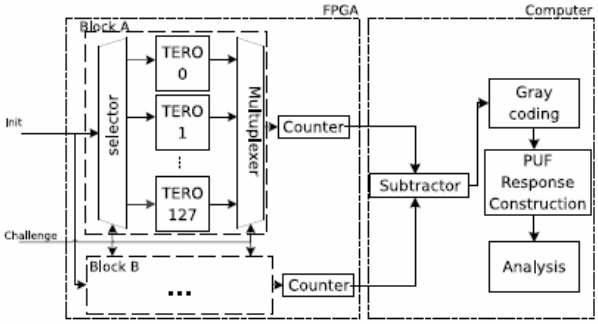
Figura 10: PUF RO arquitetura mais utilizada na literatura



Fonte: [10].

Os projetistas se baseiam nesta arquitetura para propor variações visando obter os melhores resultados para os valores da μ_{inter} e μ_{intra}. As principais alterações são na forma em que as saídas dos MUX serão utilizadas, divisão das frequências[8], ou até mesmo soluções mais complexas para aumentar a segurança e a imprevisibilidade da reposta(Figura 11)[17].

Figura 11: Arquitetura do PUF RO da referência [17]



Fonte: [17].

Na figura 11 é observado uma separação do PUF RO em uma parte de hardware implementada no FPGA e uma parte de software rodando em um computador. O mapeamento é feito de 64 bits de desafio para 128 bits de resposta sendo realizado pela diferença das frequências dos

RO. O resultado é então codificado no código Gray. É utilizada uma estrutura diferenciada de RO denominada TERO cell - Transient effect Ring Oscillator[17].

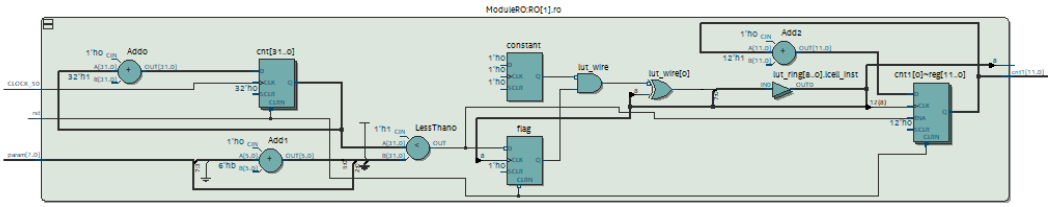
A proposta deste trabalho é implementar um PUF RO totalmente em hardware. Foi desenvolvido um módulo em Verilog denominado *PUF*. A arquitetura do módulo foi inicialmente baseada no mapeamento de 8 bits de entrada(desafio) para a obtenção de 24 bits de saída(resposta) que são exibidos nos 6 displays de 7 segmentos do FPGA, ou seja, cada display representa 4 bits da resposta. Abstraindo o módulo,temos:

- 16 entradas, sendo estas as frequências dos RO;
- 2 MUX 16:1 para selecionar as frequências dos RO à serem comparadas;
- Bloco comparador de frequência;
- Desafio - 8 bits conectados aos switches SW do FPGA;
- Resposta - N bits que serão exibidos no display de 7 segmentos;

A arquitetura do RO proposta na seção 3.1 faz a implementação do contador de borda de subida dentro do módulo RO de forma sícrona com o clock interno de 50 MHz do FPGA, garantindo que todos os RO sejam iniciados ao mesmo momento.

Para aumentar a faixa de frequências que podem ser geradas pelos RO, optou-se por utilizar os 8 bits do desafio como parâmetro de referência para o contador de borda de subida *cnt1* do RO. Como para 600 pulsos de clock os valores da variação da frequência foram os menores para os testes realizados, foi definido o valor de referência de 600 mais o valor em decimal dos 8 bits de entrada, ou seja, os RO oscilarão entre um período de 600 à 855 pulsos do clock de 50 MHz do FPGA.

Figura 12: Arquitetura do RO proposto



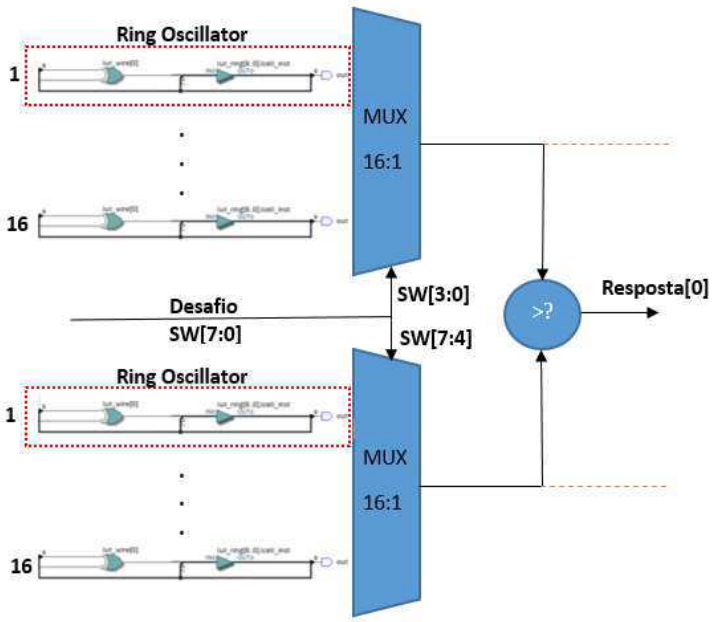
Fonte: Autoria Própria.

A forma em que o mapeamento desafio/resposta foi feito se diferencia da usual pois é utilizado um desafio para gerar uma resposta, onde o mais usual é utilizar uma sequência de diferentes desafios para gerar uma resposta. Para cada desafio se obtém um par de bits de saída, concatenando esses dados é gerado uma assinatura, resposta.

Para que cada desafio gerasse 1 reposta, o processo de mapeamento proposto foi divido em 2 etapas:

1. O **desafio** é recuperado na forma de 8 bits. Como cada MUX 16:1 necessita de uma chave seletora de 4 bits para que seja possível percorrer todas as possíveis entradas, o **desafio** é separado em 2 chaves de 4 bits, onde cada chave irá selecionar a saída do MUX. Se compara as saídas e se obtem o primeiro bit da **resposta** (*resposta[0]*) (Figura 13). Para um intervalo de 600 a 855 pulsos de clock, observou-se que as frequências utilizam 12 bits para sua representação.

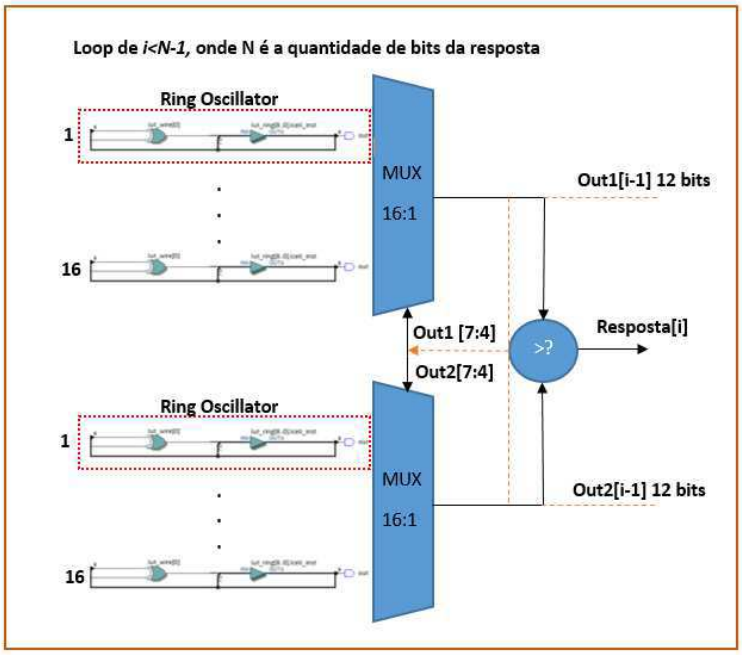
Figura 13: Arquitetura do PUF proposto - Etapa 1



Fonte: Autoria Própria.

2. As saídas dos MUX selecionadas pelo switches são armazenadas e utilizadas agora como chave seletora do MUX, ou seja, uma forma de realimentação estocásticas pois existem diversos fatores que tornam esta arquitetura não preditiva. Dos 12 bits de saída, os 4 bits correspondentes as posições intermediarias são analisados, pois foram estes bits que apresentaram a maior constância nos resultados. Este loop ocorre até que o número total de bits da resposta seja completado. Se a resposta é representada em 24 bits, este loop ocorrerá 23 vezes, dado que o primeiro bit é definido diretamente pelo desafio (Figura 13).

Figura 14: Arquitetura do PUF proposto - Etapa 2



Fonte: Autoria Própria.

A aleatoriedade é influenciada pelo desafio pois sua sequência de bits interfere no número de pulsos do clock que servirão como parâmetros para o cálculo da frequência dos módulos RO e como primeira chave seletora para resposta.

O desafio representado na forma de 8 bits é capaz de gerar 255 possibilidades diferentes de entradas. Caso não fosse considerado sua influência no cálculo da frequência, haveria 256 possibilidades de comparação entre os 2 MUX, que equivale a 256 bits independentes. Como o desafio é utilizado no módulo RO, ocasiona um aumento de 256 possibilidades para 65280 comparações possíveis.

4 Resultados

Neste capítulo são apresentados os resultados obtidos até o final do projeto e são discutidos alguns fatores que foram relevantes na obtenção destes resultados. Foram realizados, basicamente o mapeamento de 8 bits para 24 bits (desafio/resposta), com o RO configurado em 2 tipos de teste: com loop de 3 inversores e com o loop de 9 inversores.

O intuito dos testes são de ao se aplicar diferentes desafios definidos pela sequência de switches, calcular μ_{inter} e μ_{intra} da arquitetura do PUF implementada em 2 módulos FPGA.

As configurações de testes utilizando 2 quantidades diferentes de inversores no loop foi para analisar a influência da frequência de oscilação no cálculo μ_{intra}. Esta análise é uma forma de testar o comportamento do PUF para diferentes condições de funcionamento do FPGA, pois variações de tensão de alimentação e temperatura podem interferir na frequência de oscilação dos RO.

Para o cálculo de μ_{inter} e μ_{intra} utilizou-se o Excel. Foram desenvolvidas Macros funções em VBA para maior velocidade e praticidade na obtenção da comparação das sequências de 24 bits da resposta.

- RO com loop de 9 inversores

  Analisando os desafios, 0011 0101 (Figura 15); 1010 1010(Figura 16) e 1010 1111(Figura 17), é possível notar que a quantidade de respostas geradas para o mesmo desafio em cada FPGA é diferente. Tal comportamento é um primeiro indício que as variações dos processos de fabricação nos SoC são levadas em consideração pela arquitetura do PUF proposto.

  Para os resultados do desafio da Figura 15, FPGA A apresentou uma variação média dos bits que se repetem na resposta de 12.875, logo $\mu_{intra} = 46.43\%$. Para o FPGA B, observou-se uma repetição de 15 bits entre as 2 únicas respostas geradas pelo módulo, $\mu_{intra} = 37.5\%$ No cálculo do Inter-chip foi observado que em média 11.625 bits se repetem da resposta do FPGA A para o FPGA B, logo $\mu_{inter} = 51.56\%$.

Figura 15: Análise das respostas de 2 FPGA para o desafio **0011 0101**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
0011 0101	D6B5AD	249249	110101101011010110101101	001001001001001001001001
	BB76E9	0842B1	101110110111011011101001	000010000100001000010001
	39CE73		001110011100111001110011	
	0064A3		000000000110010010100011	
	52A543		010100101010010101000011	
	5F97E5		010111111001011111100101	

Fonte: Autoria Própria.

Para os resultados do desafio da Figura 16, foi calculado que a quantidade média de bits que se repetem na resposta para o FPGA A e B é de 17. A estabilidade da resposta foi $\mu_{intra} = 29\%$ para os dois FPGA. No cálculo do Inter-chip foi observado que em média 11.125 bits se repetem da resposta do FPGA A para o FPGA B, logo $\mu_{inter} = 53.65\%$.

Figura 16: Análise das respostas de 2 FPGA para o desafio **1010 1010**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
1010 1010	DB6C36	18CB7A	11011011011011000011011O	00011000110010110111101O
	F7BC36	AD6B5A	11110111101111000011011O	10101101011010110101101O
	E38E36	A0477A	11100011100011100011011O	10100000010001110111101O
	E79E70	50237A	11100111100111001110000	01010000001000110111101O

Fonte: Autoria Própria.

Para os resultados do desafio da Figura 16, FPGA A exibiu 3 respostas possíveis, enquanto que o FPGA B apresentou 5. FPGA A com 13.33 bits em média que se repetem entre suas respostas ($\mu_{intra} = 44.44\%$) e o FPGA B apresentou 17 bits ($\mu_{intra} = 29.17\%$), sendo o melhor cenário apresentado. A variabilidade da resposta foi de $\mu_{inter} = 58.85\%$, apresentando média 9.875 bits iguais entre as respostas dos dois FPGA para o mesmo desafio.

Figura 17: Análise das respostas de 2 FPGA para o desafio **1010 1111**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
1010 1111	6DB6DD	924927	01101101101101101101101	10010010010010010010011
	4F64F9	D141A3	01001111011001001111100	11010001010000011010001
	E38E3D	108423	11100011100011100011101	00010000100001000010001
		18C637		00011000110001100011011
		D141A3		11010001010000011010001

Fonte: Autoria Própria.

A Tabela 12 apresenta o valor médio das variáveis do PUF para os 3 desafios lançados.

Tabela 12: Resumo do valor médio de μ_{intra} e μ_{inter} para os testes realizados com loop de 9 inversores.

	FPGA A	FPGA B
μ_{intra}	40.40%	22.22%
μ_{inter}	54.68%	

Fonte: Autoria Própria.

- RO com loop de 3 inversores

  Com alteração no número de inversores utilizados no loop do RO, foram refeitos os cálculos das variáveis para os mesmo desafios.

  – Para o desafio 0011 0101: $\mu_{inter} = 31.25\%$

  Para os resultados do desafio da Figura 18, FPGA A apresentou $\mu_{intra} = 39.58\%$ e o FPGA B $\mu_{intra} = 41.16\%$.

  Figura 18: Análise das respostas de 2 FPGA para o desafio **0011 0101**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
0011 0101	B6DBC0	AAC508	10110110110110111000000	10101010110001010000001000
	C083F4	996D18	11000000100000011111110100	10011001011011011000001000
	5ADC90	233118	01011010110111001001000	00100011001100011000010000
	EFBE20	65CD9C	11101111101111100010000	01100101110011011011001100

  Fonte: Autoria Própria.

  – Para o desafio 1010 1010: $\mu_{inter} = 48.43\%$

  Para os resultados do desafio da Figura 19, FPGA A apresentou $\mu_{intra} = 47.91\%$ e o FPGA B $\mu_{intra} = 39.58\%$.

  Figura 19: Análise das respostas de 2 FPGA para o desafio **0011 0101**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
1010 1010	5294B8	55442A	01010010100101001011000	01010101010001000001010
	5AD6B4	55798A	01011010110101101110100	01010101011110011001010
	492494	2C58B0	01001001001001001001010100	00101100010110001011000
	6DE62A		01101101111001100010101	

  Fonte: Autoria Própria.

  – Para o desafio 1010 1111: $\mu_{inter} = 43.75\%$

  Para os resultados do desafio da Figura 20, FPGA A apresentou $\mu_{intra} = 50\%$ e o FPGA B $\mu_{intra} = 47.7\%$.

  Figura 20: Análise das respostas de 2 FPGA para o desafio **1010 1111**

Desafio	Resposta		Binário	
	FPGA A	FPGA B	FPGA A	FPGA B
1010 1111	492493	660CC1	01001001001001001001011	01100110000011001100001
	AAC557	B366CD	10101010110001010101111	10110011011001101101101
	5294A7	D6B5A1	01010010100101001010111	11010110101101011010000
		79B5EB1		01111001101101011101011

  Fonte: Autoria Própria.

Tabela 13: Resumo do valor médio de μ_{intra} e μ_{inter} para os testes realizados com loop de 3 inversores.

	FPGA A	FPGA B
μ_{intra}	45.83%	43.05%
μ_{inter}	43.14%	

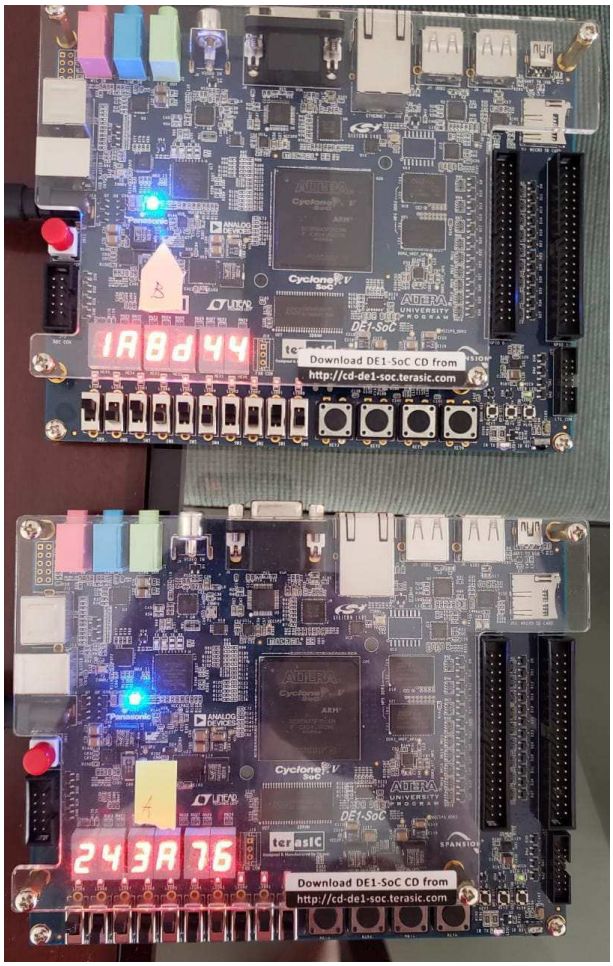
Fonte: Autoria Própria.

O valor médio de μ_{inter} para o teste utilizando 9 inversores no loop está de acordo com as especificações para se obter a unicidade dos dispositivos. Como a análise da resposta é feita em relação a 24 bits, a diferença de 1 bit entre respostas já equivale à $\mu_{intra} = 4.16\%$, valor já muito superior ao que se é esperado $\mu_{intra} = 0\%$.

Uma forma de melhorar a resposta desta variável é aumento o número de bits da representação da resposta. Os trabalhos [10] e [17] utilizam 128 bits para representar a resposta do sistema, de tal forma que a variação de um bit corresponde à um $\mu_{intra} = 0.78\%$.

Quanto menor o número de inversores utilizados no loop do RO, maior a frequência de oscilação, ocasionando maior variação no cálculo da frequência. Isto é refletido no aumento do valor dos μ_{intra} para os testes utilizando 3 inversores no loop.

Figura 21: Comparação FPGA A e FPGA B para um mesmo desafio



Fonte: Autoria Própria.

5 Conclusão

Ataques cibernéticos tem se tornado uma grande preocupação em virtude dos prejuízos financeiros que podem acarretar. Uma forma de aumentar a segurança de sistemas e a autenticação de dispositivos .

Este trabalho teve como objetivo desenvolver a modelagem de um PUF RO em Verilog para ser implementado em um módulo FPGA. A arquitetura proposta é 100% integrada e configurada na parte física do módulo. Foi possível, com a utilização de LCELL, conceber um circuito oscilador com frequência variável em relação às variações do processo de fabricação do SoC.

Em função dos resultados obtidos, pode-se concluir que as respostas de dois módulos FPGA com o mesmo bitstream gerado pelo projeto foram diferentes, atingindo o objetivo de demonstrar a unicidade dos SoC. Porém, o projeto ainda deve ser melhorado para se obter menos variações na resposta a um mesmo desafio. Os valores de μ_{intra} obtidos ainda não permitem uma aplicação direta da solução proposta, como por exemplo a autenticação de dispositivos em IoT, pois as respostas ainda não são suficientemente estáveis.

Referências

[1] Centro Regional de Estudos para o Desenvolvimento da Sociedade da Informação. Tic domicílios 2018. Disponível em: https://cetic.br/noticia/ Acesso em: 30 Ago. 2019.

[2] CryptoID. Disponível em: https://cryptoid.com.br/banco-de-noticias/conheca-os-detalhes-do-ataque-ao-banco-do-chile-via-sistema-swift/ Acesso em: 30 Ago. 2019.

[3] Disponível em: https://www.healthline.com/health-news/are-pacemakers-defibrillators-vulnerable-to-hackers Acesso em: 03 Set. 2019.

[4] David HELY. Secure vlsi design - introduction to cryptography. In *Arquivo pessoal - material de professores Grenoble-INP, ESISAR*.

[5] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography*, 19(2-3):173–193, 2000.

[6] David HELY. Introduction à la sécurtié matérielle. In *Arquivo pessoal - material de professores Grenoble-INP, ESISAR*.

[7] Muhammad Asim, Jorge Guajardo, Sandeep S Kumar, and Pim Tuyls. Physical unclonable functions and their applications to vehicle system security (full paper).

[8] Roel Maes and Ingrid Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*, pages 3–37. Springer, 2010.

[9] Giray Kömürcü, Ali Pusane, and Günhan Dündar. A ring oscillator based puf implementation on fpga. *Istanbul University-Journal of Electrical & Electronics Engineering*, 13(2):1647–1652, 2013.

[10] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. pages 9–14, 2007.

[11] Sandeep S Kumar, Jorge Guajardo, Roel Maes, Geert-Jan Schrijen, and Pim Tuyls. The butterfly puf protecting ip on every fpga. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 67–70. IEEE, 2008.

[12] David HELY. Secure vlsi design - trusted hardware. In *Arquivo pessoal - material de professores Grenoble-INP, ESISAR*.

[13] Altera. Cyclone v device handbook. Disponível em: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_5v2.pdf. Acesso em: 15 Ago. 2019.

[14] Disponível em: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_low_level.pdf Acesso em: 26 Ago. 2019.

[15] Disponível em: https://forums.intel.com/s/createarticlepage?language=en_US&articleid=a3g0P0000005RSiQAM&artTopicId=0TO0P000000MWKBWA4&action=view Acesso em: 27 Ago. 2019.

[16] Disponível em: https://www.fpga4fun.com/FPGAinfo2.html Acesso em: 30 Ago. 2019.

[17] Cédric Marchand, Lilian Bossuet, Ugo Mureddu, Nathalie Bochard, Abdelkarim Cherkaoui, and Viktor Fischer. Implementation and characterization of a physical unclonable function for iot: a case study with the tero-puf. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):97–109, 2017.