PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA

DESENVOLVIMENTO DE UM MÓDULO DE CRIPTOGRAFIA LEVE PARA FPGA UTILIZANDO O ALGORITMO SIMON AND SPECK

CRISTOVAM LAGE DA SILVA

Monografia apresentada como requisito parcial à obtenção do grau de Bacharel em Engenharia de Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre
2019

DESENVOLVIMENTO DE UM MÓDULO DE CRIPTOGRAFIA LEVE PARA FPGA UTILIZANDO O ALGORITMO SIMON AND SPECK

RESUMO

A era da Internet das Coisas (IoT) trouxe um aumento considerável no número de dispositivos interconectados, e consequentemente o volume de dados compartilhados entre eles. Além da maior quantidade de aparelhos, a cada ano o poder computacional destes dispositivos também se eleva, e seu tamanho e preço diminuem. O crescimento da quantidade de mensagens trocadas contendo os mais variados tipos de informação gera uma preocupação significativa em relação à segurança e à privacidade dos usuários. Em razão disso, estão sendo estudadas e desenvolvidas técnicas de segurança da informação adequadas para este novo cenário.O objetivo deste documento é apresentar o trabalho de Conclusão de Curso (TCC) para o curso de Engenharia de Computação da Pontifícia Universidade Católica do Rio Grande do Sul. Neste TCC são descritos a implementação e a validação, de um módulo de criptografia leve utilizando o algoritmo Simon da família Simon e Speck para um dispositivo FPGA, posteriormente comparando-o à um core AES disponibilizado por Hemanth Satyanarayana em 2004 na plataforma *opencores*.

Palavras-Chave: criptografia leve, VHDL, FPGA, Simon and Speck.

DESIGN OF A LIGHTWEIGHT CRYPTOGRAPHY MODULE FOR FPGA BASED ON SIMON AND SPECK ALGORITHMS

ABSTRACT

The Internet of Things (IoT) era brought a considerable increase in the number of interconnected devices and consequently the volume of data shared among them. In addition to the increased number of devices, the computational power of these escalates whilst their size and price reduces. This increase in the amount of messages exchanged containing the most varied types of information raises a significant concern regarding the users' security and privacy. Considering that, new security information techniques suitable for this new cenario are being studied and developed. The goal of this document is to present the the End-of-Term work for the Computer Engineering course of the Pontifical Catholic University of Rio Grande do Sul. This End-of-Term work describes the implementation and validation, of a lightweight cryptography module using the Simon of the Simon and Speck algorithms family for a FPGA device and thereafter a comparison with an AES core release by Hemanth Satyanarayana on 2004 on the *opencores* platform.

Keywords: lightweight cryptography, VHDL, FPGA, Simon and Speck.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AES – *Advanced Encryption Standard*

ASIC – *Application-Specific Integrated Circuit*

DES – *Default Encryption Standard*

DH – *Diffie-Hellmann*

ECC – *Elliptic Curve Criptography*

ES – *Embedded System*

FPGA – *Field Programmable Gate Array*

GE – *Gate Equivalent*

IOT – *Internet of Things*

IC – *Integrated Circuit*

LFSR – *Linear-Feedback Shift Register*

LWC – *lightweight cryptography*

MSS – *Merkle Signature Scheme*

MPSOC – *Multiprocessor System on Chip*

NSA – *National Security Agency*

RSA – *Rivest, Shamir and Adleman*

RNG – *Random Number Generator*

SOC – *System on Chip*

SPN – *Substitution Permutation Network*

VHDL – *VHSIC Hardware Description Language*

1. INTRODUÇÃO

Há alguns anos vivenciamos a terceira era da informática, após os *mainframes* e os computadores pessoais, inciou-se a era da Internet das Coisas – *Internet of Things* (IoT) em inglês –, a fase dos denominados "objetos inteligentes". Este novo ambiente envolve computadores que vem adentrando em praticamente todas as áreas do nosso cotidiano, estando presentes em uma gama enorme de diferentes aparelhos produzidos em grande escala.

Essas aplicações podem ser industriais, vestíveis, operando em espaços públicos ou privados, contendo informações sensíveis ou até críticas. Estes sistemas são instalados em lugares hostis e/ou remotos, muitas vezes interagindo com o meio externo, de forma monitorada ou não [Rabaiei and Harous, 2016].

Em cada dispositivo existe uma certa quantidade de informações, o que vem trazendo grandes preocupações em relação à segurança e à privacidade de todos. Por isso, há algum tempo pesquisadores vêm estudando os desafios no desenvolvimento de sistemas cada vez mais baratos, com recursos mais limitados e em ambientes mais heterogêneos sem abrir mão de sua segurança.

A criptografia é uma área bastante desenvolvida da segurança da informação, entretanto, em sua forma tradicional pode não ser adequada às capacidades de processamento, de memória e, principalmente, de gasto energético dos dispositivos utilizados em IoT.

A *criptografia leve* (LWC) é a vertente da criptografia destinada a suprir as necessidades dessa nova realidade. Sua padronização vem sendo discutida nas ISO/IEC 29192 [Hanley and ONeill, 2012].

Entre os mais promissores algoritmos dessa nova linha de criptografia está a família *Simon and Speck*. Apresentada pela *National Security Agency* (NSA), tem como principais objetivos o desenvolvimento de algoritmos LWC de alto desempenho com flexibilidade e baixo custo, tanto em hardware quanto em software [Aysu et al., 2014, Gulcan et al., 2014, Beaulieu et al., 2015b].

Os sistemas embarcados – *embedded systems*, em inglês [Kleidermacher and Kleidermacher, 2012], diferentemente dos computadores pessoais, são sistemas desenvolvidos com um propósito específico e são esses que disponibilizarão especificações mais restritas de desempenho. Entre as plataformas mais comuns para implementação de um sistema embarcado estão os microcontroladores, ASICs e FPGAs.

Um microcontrolador é um microprocessador de menor complexidade, implementado em um único circuito integrado, possuindo um menor custo quando comparado a microprocessadores convencionais. Os microcontroladores são usados em produtos e dispositivos controlados automaticamente, como sistemas de controle de motores de automóveis,

dispositivos médicos, eletrodomésticos, e outros sistemas embarcados. Microcontroladores de sinais mistos (com conversores AD e DA) são comuns, integrando componentes analógicos necessários para controlar sistemas eletrônicos não digitais [Heath, 2002].

Os *Application-Specific Integrated Circuits* (ASICs) são dispositivos de hardware projetados frequentemente como um *System-on-Chip* (SoC), que possuem função específica. Comparando-se os ASICs às demais soluções tecnológicas, este proveem maior desempenho e menor consumo de energia, porém com maior custo de desenvolvimento e menor flexibilidade [Smith, 1997].

Os *Field Programmable Gate Arrays* (FPGA) são circuitos integrados configuráveis, sem um conjunto de instruções pré-definidas. Por essa razão representam uma solução com melhor desempenho que os microcontroladores, e uma maior flexibilidade e menor custo comparando-se a projetos baseados em ASIC [Brown et al., 1992].

Atualmente, as pesquisas relacionadas a LWC são, em sua grande maioria, focadas em projetos para ASICs, porém considerando características como a capacidade de reconfiguração, a diminuição dos preços dos FPGAs, e o aumento da capacidade de lógica configurável a cada ano, o Autor, baseado nos artigos [Katagi and Moriai, 2008, Yalla and Kaps, 2009, Aysu et al., 2014], acredita que o desenvolvimento de LWC para sistemas embarcados FPGAs representa uma importante contribuição para estes projetos.

1.1 Objetivos

Este trabalho tem por objetivo principal implementar um módulo de criptografia leve em VHDL. O foco será no desenvolvimento e validação por simulação do módulo de forma isolada, comprovando sua capacidade de encriptação e decriptação. Mais especificamente, consiste na implementação do algoritmo Simon da família *Simon and Speck*.

Um segundo objetivo é realização da síntese deste módulo para FPGA e ASIC e a comparação de suas características com o módulo *Advanced Encryption Standard* (AES) disponibilizado por Hemanth Satyanarayana em 2004 na plataforma *opencores*.

1.2 Estrutura do Documento

Este documento é organizado como segue. Os Capítulos 1 e 2 abordam os conceitos básicos sobre criptografia. O Capítulo 3 detalha o funcionamento do algoritmo Simon que será implementado. Nos capítulos 4 e 5 são apresentados as etapas de desenvolvimento, validação e análise do módulo, respectivamente. Finalmente, no capítulo 6 são discutidas as conclusões sobre o projeto e trabalhos futuros.

2. CRIPTOGRAFIA

A criptografia, palavra originada do grego e que tem como significado “grafia escondida”, é a parte da criptologia em que se estudam métodos de comunicação segura na presença de terceiros.

Nas áreas da computação, este estudo é aplicado para fins de segurança da informação, que tem como propriedades básicas, segundo os padrões internacionais (ISO/IEC 17799:2005), a confidencialidade, integridade e disponibilidade, como principais, e a autenticidade, não-repúdio e conformidade, como secundárias.

Já as propriedades básicas de sistemas de criptografia são estabelecidas pelos princípios de Kerckhoffs [Petitcolas, 1883]:

- Uma informação codificada não pode ser decodificada sem o conhecimento da chave.
- Os interlocutores não podem ser prejudicados caso o sistema de encriptação seja descoberto.
- A chave deve ser simples e modificável quando necessário.
- O sistema de encriptação e os criptogramas devem ser portáveis.
- O sistema deve ser de simples utilização.
- O sistema de codificação deve ser validado por especialistas.

Resumidamente, um sistema deve ser seguro mesmo que todas as suas partes, exceto a chave, se tornem conhecidas pelo atacante.

Na criptografia computacional é necessário distinguir entre os dois tipos existentes: os algoritmos de chave pública, ou assimétricos, e os de chave privada, ou simétricos. A escolha das técnicas utilizadas varia de acordo com as necessidades e recursos disponíveis em cada aplicação a ser desenvolvida. Os dois serão abordados em seguida.

A parte complementar da criptologia é a criptoanálise, em que os cripto-analistas dedicam seus estudos a desvendar a lógica por trás dos sistemas e/ou o texto encriptado. Ou seja, é onde são desenvolvidas técnicas para ‘quebrar’ o mascaramento feito pela criptografia. São eles que irão determinar, através de análises e testes, o nível de segurança de cada sistema.

É importante salientar que o comprimento da chave (em bits) difere do nível de segurança, ou da ‘força’, de um algoritmo de criptografia. Este valor é calculado pela quantidade de tempo que um ataque mais rápido que o uso de força bruta requer para ‘quebrá-lo’. O valor máximo de segurança (também em bits) não pode ser maior que o comprimento de chave. Os algoritmos de chave privada mais utilizados atualmente podem alcançar o valor

máximo, isto não acontece para os de chave pública, que tem em seu caso mais promissor a metade do comprimento de chave como nível de segurança [Schinianakis, 2017].

2.1 Chave Pública

Os algoritmos de chave pública oferecem melhores mecanismos de gerenciamento de chaves, de não-repúdio e de autenticação quando comparados aos de chave privada [Schinianakis, 2017]. São úteis em ambientes em que a geração de chaves deve ser dinâmica, como em sistemas heterogêneos, por exemplo.

Em contraponto às vantagens em relação aos algoritmos simétricos, os assimétricos podem ser de 2 a 3 ordens de grandeza mais custosos [Manifavas et al., 2013]. Essa característica se dá pela necessidade de chaves de comprimento muito grandes pois, pela natureza dos problemas matemáticos que são utilizados, como mencionado anteriormente, existe uma baixa relação força/comprimento de chave na implementação destes algoritmos.

Quanto ao funcionamento dos algoritmos, são utilizadas duas chaves, uma pública, para encriptação e uma privada para decriptação da mensagem. Quando um usuário A deseja enviar uma mensagem a um usuário B, este utiliza a chave pública de B para encriptar a mensagem. Por sua vez, B poderá decriptá-la com sua própria chave privada ao receber e vice-versa. Faz-se necessário ressaltar que A não pode decriptar a própria mensagem com a chave pública de B nem com a sua chave privada. Apesar de existir relação entre a chave privada e pública de cada usuário, os sistemas de encriptação e decriptação são distintos.

Este tipo de algoritmo é realizado com funções *trap-door* unidirecionais, que são fáceis de computar apenas em uma direção e baseadas em complexos problemas matemáticos. Os três mais conhecidos são o Diffie-Hellman (DH), o Elliptic Curve Criptography (ECC), e o Rivest, Shamir and Adleman (RSA), sendo o ECC aquele que possui a maior 'força' relativa ao comprimento de chave.

O RSA é o mais conhecido deste tipo, sendo utilizado em servidores de emails, browsers, ssh, entre outros. É o benchmark para novos algoritmos. Seu tamanho de chave varia entre 1024 e 4096 bits.

Cripto-sistemas alternativos têm sido estudados utilizando diferentes problemas matemáticos, empregando, por exemplo, hash e redes diagonais. Um dos mais promissores para aplicações lightweight é o Merkle Signature Scheme (MSS) baseado no Advanced Encryption System (AES) e hash, possuindo menor código e verificação mais rápida que o RSA e o EEC, além de geração de assinatura mais rápida que o RSA e próxima a do ECC [Schinianakis, 2017].

Apesar dos avanços em novos tipos de sistemas assimétricos, os elevados números de bits nas chaves destes algoritmos ainda geram uma grande área, atraso e consumo de energia quando implementados em hardware. E, apesar de existirem otimizações para este tipo de algoritmo, especialmente em software, o objetivo deste projeto é gerar um módulo de criptografia de baixo custo em uma FPGA com recursos limitados. Portanto este grupo de algoritmos não será mais abordado neste documento.

2.2 Chave Privada

Estes algoritmos são implementados para gerar mensagens de autenticação, verificação de integridade e encriptação de payloads (por serem muito mais rápidos que os assimétricos) [Schinianakis, 2017]. A desvantagem está no gerenciamento da chave, ou seja, no fato de todos os interlocutores necessitarem da chave antecipadamente. Para uma conversa segura entre *n* interlocutores um número de *n*(n-1)/2* chaves devem ser utilizadas, por exemplo.

Os algoritmos de chave privada podem ser implementados de duas maneiras, por fluxo ou bloco, e devem ser escolhidos de acordo com a aplicação. Em ambos os casos uma chave privada é conhecida pelas partes comunicantes e utilizada para encriptar e decriptar as mensagens devido à reversibilidade das funções. Ou seja, usa-se o mesmo algoritmo e inverte-se a chave nos respectivos processos.

Nos algoritmos de fluxo, os dados a serem enviados são encriptados bit-a-bit ou byte-a-byte enquanto, nos por bloco um bloco com *n* bits é utilizado como entrada e um bloco de *n* bits é obtido como saída.

2.2.1 Fluxo

Nos algoritmos de encriptação por fluxo, os dados são manipulados bit-a-bit ou byte-a-byte, sendo encriptados através de uma operação *XOR* com um *key-stream*. São muito úteis quando o tamanho da mensagem não é conhecido, justamente por operarem nas menores unidades possíveis, além de serem bastante rápidos e fáceis de implementar [Stallings, 2015].

O *key-stream* é obtido através de um gerador de números pseudo-aleatórios, *random number generator* (RNG) em inglês, e a segurança depende da boa implementação deste. Estes geradores são formados, normalmente, por LFSRs, que são essencialmente lineares, mas técnicas são utilizadas para torná-los não-lineares.

Fundamentalmente, deve ser impossível recuperar a chave e/ou o estado dos geradores em qualquer momento, além de dever-se evitar usar a mesma semente mais de uma vez, o que muitas vezes acontece.

Alguns exemplos de algoritmo são o RC4 e o AC5/1, encontrados frequentemente em sistemas que utilizam GSM. Recentemente, estes foram declarados não mais seguros e sua utilização foi proibida em novos sistemas [Manifavas et al., 2013].

Apesar de serem algoritmos viáveis para implementação *lightweight*, ainda existem dois aspectos relevantes para a sua não utilização: os problemas de segurança relacionados a estes sistemas, e muitas vezes a necessidade de um período de inicialização da execução que gera um atraso alto [Schinianakis, 2017]. Por estas razões, esses algoritmos não serão posteriormente abordados no decorrer deste documento.

2.2.2 Bloco

Os algoritmos simétricos que operam por blocos são mais rápidos que os assimétricos e mais seguros que os por fluxo, sendo a opção adequada para projetos visando LWC. Estes são baseados nos princípios de confusão e difusão que, de acordo com Shannon [Kocarev, 2001]:

> "A confusão refere-se a tornar a relação entre o texto original e o cifrado a mais complexa possível, e a difusão em dissipar as estruturas estatísticas que relacionam os dois"

Esses processos são, geralmente, automatizados através de sequências de passos bem definidos e repetitivos de substituição e permutação. Grande parte dos algoritmos deste grupo utilizam as chamadas Redes de Substituição-Permutação ou Redes de Feistel, apresentadas abaixo.

Mesmo com a dificuldade de implementação em hardware de lógicas de substituição de bytes, é possível utilizar algumas técnicas para mitigar essa característica e alcançar excelentes resultados de implementações leves e de baixo custo.

A. Redes de Substituição-Permutção (SPN)

Como supracitado, algoritmos de chave simétrica baseados em blocos realizam um conjunto de passos repetidas vezes, denominados *rounds*. Esses são necessários para atingir o nível de confusão e difusão desejado. Um algoritmo que atinge baixos níveis de difusão e confusão a cada iteração necessitará de um número maior de rounds, prejudicando seu desempenho [Stallings, 2015].

O algoritmo AES, baseado em SPN, servirá como exemplo do funcionamento deste tipo de estrutura. O AES possui 11 *rounds*, sendo o primeiro e o último diferente dos 9 restantes. No primeiro round, uma operação *XOR* é feita entre a chave e o bloco. Durante os 9 rounds intermediários são realizadas 4 operações: substituição por *s-box*, deslocamento de colunas, multiplicação por uma matriz estado e, finalmente, operação *XOR* com a chave de round, como no primeiro. O 11º *round* realiza apenas as três primeiras operações dos nove precedentes. A Figura 2.1 ilustra este processo.

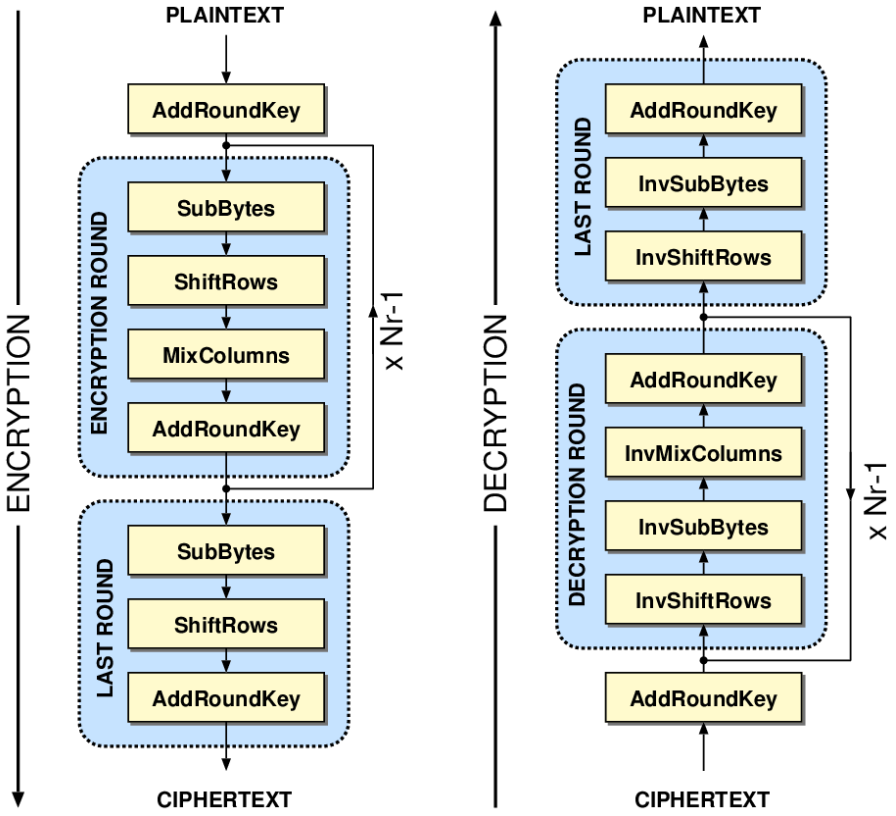

Figura 2.1 – Funcionamento do AES [Herigemblong, 2018].

As operações que tem por objetivo aplicar confusão são: a substituição com *s-box* (*SubBytes*), que realiza uma substituição direta dos dados na matriz estado pelos de uma tabela pré-definida e deve ser completamente inversível para a decriptação, e a multiplicação por uma matriz constante (*MixColumns*), que multiplica os valores da matriz estado por constantes de um campo de Galois GF(2).

Os deslocamentos de colunas *ShiftRows* (uma procedimento de permutação dos bytes do bloco) e uma operação *XOR* com a chave do round (*AddRoundKey*) são responsáveis pela adição de difusão ao bloco.

As chaves usadas na etapa *AddRoundKey*, denominadas chaves de *round*, são obtidas a partir da chave 'mestra' compartilhada pelos interlocutores e obtidas através do sistema de geração de chaves próprio do algoritmo.

B. Redes de Feistel

Diferentemente das SPNs, nas redes de Feistel o bloco é dividido em duas partes, esquerda e direita e, apenas uma destas é utilizada a cada *round*, sendo os lados invertidos no *round* seguinte. A Figura 2.2 ilustra a operação da rede de Feistel presente no algoritmo DES.

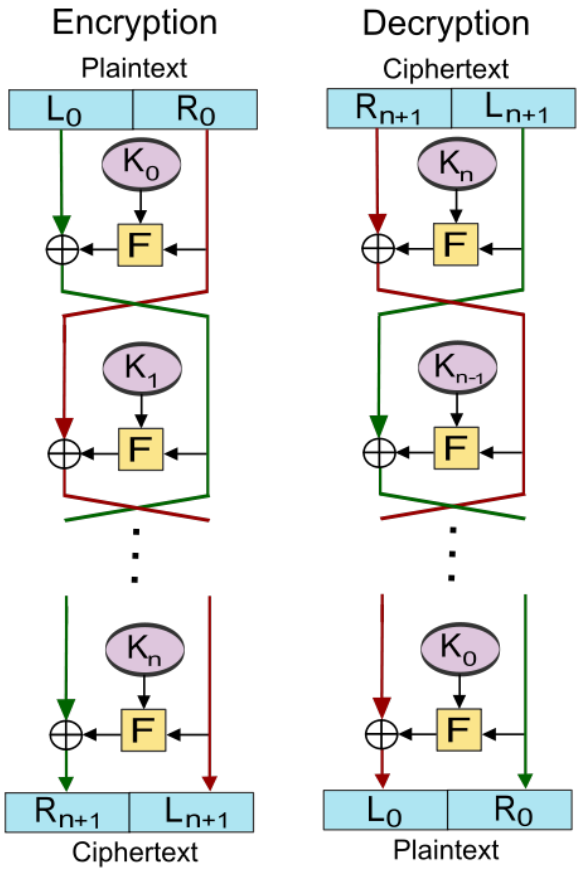

Figura 2.2 – Funcionamento do DES [Wikipedia, 2018].

O bloco amarelo na Figura 2.2 representa uma função não-linear *F* - também baseada em mecanismos de substituição e permutação - mas, neste caso, não há a necessidade de ser inversível para a decriptação. Após ser submetido à função, o lado sendo computado ainda realiza uma última operação *XOR* com o lado não utilizado.

Usualmente, pelo fato de trabalhar apenas com a metade do bloco a ser encriptado um número maior de *rounds* pode ser necessário, o que pode comprometer o desempenho.

2.3 Criptografia Leve

Os algoritmos previamente apresentados para criptografia buscam obter altos níveis de segurança, com pouca ou nenhuma preocupação em relação ao consumo ener-

gético, área do circuito, quantidade de memória, entre outras características. A área de criptografia leve é direcionada à segurança de dispositivos em que estes recursos são limitados. Muitos estudos tem sido realizados, especialmente utilizando algoritmos de chave privada com encriptação por blocos.

Uma das técnicas utilizadas por pesquisadores é melhorar o desempenho de algoritmos já conhecidos e com a segurança comprovada, como o AES, que no estado-da-arte de uma de suas implementações utiliza 2400 portas lógicas equivalentes (*GE* - gate equivalente, correspondendo a uma porta NAND de duas entradas), sendo utilizado como benchmark para novos algoritmos.

Uma segunda abordagem é desenvolver algoritmos inteiramente focados nas propriedades da LWC. É o caso do SIMON, do PRINCE e do PRESENT, que chegam a ter implementações menores que 1000 *GEs* . Estes algoritmos porém, devem ter seu nível de segurança verificado por cripto-analistas e serem submetidos a diversos ataques e testes.

Em um terceiro cenário está a criação de algoritmos híbridos, que utilizam características de diversos algoritmos já comprovadamente seguros, juntamente com as novas técnicas desenvolvidas [Manifavas et al., 2013]. Nas tabelas 2.1 e 2.2 são apresentados alguns algoritmos e suas respectivas características ASIC e FPGA.

Tabela 2.1 – Comparação dos resultados ASIC dos algoritmos LWC.

Algoritmo	Chave (bits)	Vazão (Kbps à 100 KHz)	*GEs*
AES [Katagi and Moriai, 2008]	128	72.4	3100
AES [Manifavas et al., 2013]	128	56.6	2400
PRESENT [Katagi and Moriai, 2008]	128	11.5	1391
SIMON [Beaulieu et al., 2015a]	128	2.9	1234

A implementação do algoritmo LWC deve sempre levar em consideração os compromissos de área e vazão. Por exemplo, há duas implementação do AES, uma com vazão aproximadamente 27% mais elevada ao custo de uma área 29% maior.

Tabela 2.2 – Comparação dos resultados FPGA dos algoritmos LWC.

Algoritmo	Chave (bits)	Slices	Vazão (Mbps)
SIMON [Beaulieu et al., 2015a]	128	24	9.6
PRESENT [Beaulieu et al., 2015a]	128	117	28.4
AES [Beaulieu et al., 2015a]	128	184	36.5

Assim como para ASIC, a implementação para FPGA pode ser otimizada de acordo com o projeto e devem ser levados em consideração as necessidades e recursos disponíveis. O Simon possui implementações com 24 slices, 15% do necessário para o AES, mas com vazão quase 4 vezes menor.

O Simon além de apresentar características adequadas para o conceito de LWC é um algoritmo parametrizável e baseado em funções de fácil implementação, sendo uma boa escolha para o desenvolvimento deste projeto.

3. SIMON AND SPECK

Divulgado em 2013 pela Agência Nacional de Segurança (NSA) dos Estados Unidos da América, a família de algoritmos *Simon and Speck* tem por objetivo trazer alternativas para a área de criptografia leve com algoritmos flexíveis e otimizados. Os criadores afirmam que ambos os algoritmos (Simon e Speck) alcançam ótimos resultados, independentemente da plataforma. As implementações Speck foram elaboradas para obterem melhores resultados em software, enquanto o Simon é mais eficiente em implementações de hardware [Beaulieu et al., 2015b].

Estes algoritmos são parametrizáveis e utilizados de acordo com as necessidades de desempenho e/ou nível de segurança do projeto, possuindo diversas combinações de tamanho de chave e bloco, como mostrado na Tabela 3.1.

Tabela 3.1 – Combinações de Chave/Bloco Simon and Speck.

Tamanho do Bloco *(2n)*	Tamanho da Chave *(mn)*	Sequencia Constante *(zj)*	Rounds *(T)*
32	64	z_0	32
48	72	z_0	36
48	96	z_1	36
64	96	z_2	42
64	128	z_3	44
96	96	z_2	52
96	144	z_3	54
128	128	z_2	68
128	192	z_3	69
128	256	z_4	72

Existem 10 combinações possíveis de serem implementadas. As opções que oferecem menor e maior segurança possuem tamanho de chave de 64 e 256 bits, respectivamente. Em LWC o tamanho de chave mínimo para um sistema ser considerado seguro é de 80 bits [Manifavas et al., 2013], ou seja, apenas 8 combinações das apresentados na tabela possuem esta característica.

3.1 Simon

O método denominado Simon foi desenvolvido para ser aplicado em hardware, e sendo um algoritmo de criptografia leve, de chave privada, por bloco e otimizado para o objetivo deste trabalho, foi o escolhido para ser implementado.

O algoritmo Simon, simbolizado por *Simon 2n/mn*, realizada a encriptação de 2n-bits por execução, cada bloco (lado da rede de Feistel) possui n-bits, e o tamanho da chave é de mn-bits, sendo m um valor a ser discutido na seção 3.1.1, quando abordado o sistema

de geração das chaves de round. *O Simon 128/128*, por exemplo, opera sobre blocos de 64 bits e tamanho de chave de 128 bits.

3.1.1 Geração das Chaves

A escolha do Simon a ser utilizado definirá o número de palavras de chave m, o número de *rounds* e a sequência constante z_j de acordo com a Tabela 3.1. O mecanismo de geração das chaves de round utiliza os blocos $K[0]$ à $K[m-1]$ como ilustrado na Figura 3.1.

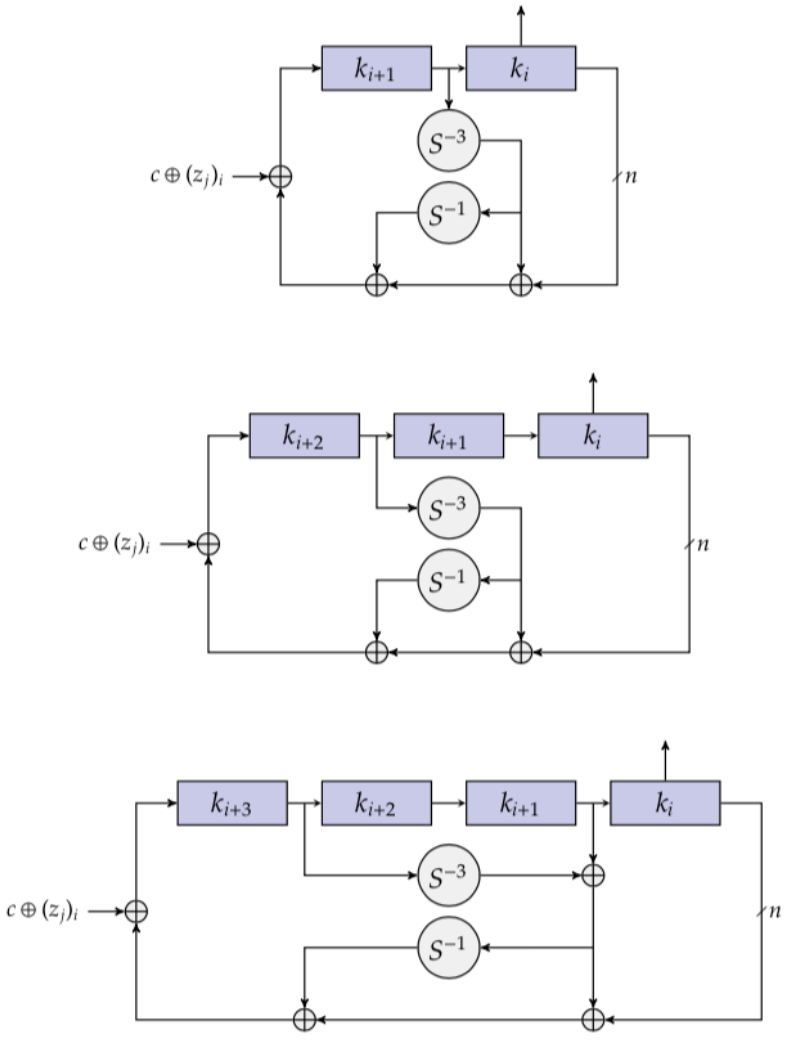

Figura 3.1 – Estruturas de Geração de chaves do algoritmo Simon [Beaulieu et al., 2015b].

O sistema de produção das sequencias z_j se baseia no uso de campos finitos de Galois GF(2) e de LFSRs, o detalhamento deste pode ser encontrado em [Beaulieu et al., 2015b]. As diferentes sequências tem por objetivo prover separação criptográfica entre as versões do algoritmo.

Na Figura 3.1 são mostrados os diferentes mecanismos de geração relativos ao valor de m. A costante $c = (2^n - 4) = 0xFF...FC$ é utilizada para a extração de bits da sequencia z_j, os símbolos S^{-j} representam o deslocamento circular (para a direita) de J bits, e $\oplus$ a operação *XOR*.

As chaves de *round* começam a ser geradas a partir da chave pré-definida e conhecida pelos interlocutores, sendo o bloco $K[m-1]$ carregado no registrador mais à esquerda e $K[0]$ no mais à direita. A cada round é gerada a chave K_i, onde $m \leqslant i < T$.

3.1.2 Operação do Algortimo

O baixo custo de implementação é resultado da natureza das operações. São utilizadas apenas *XOR bitwise, AND bitwise* e deslocamento circular (para a esquerda) para gerar a função F da rede de Feistel, Equação 3.1. Um diagrama de blocos é apresentado na Figura 3.2, onde o lado esquerdo será denominado *Xupper* e o direito *Xlower*.

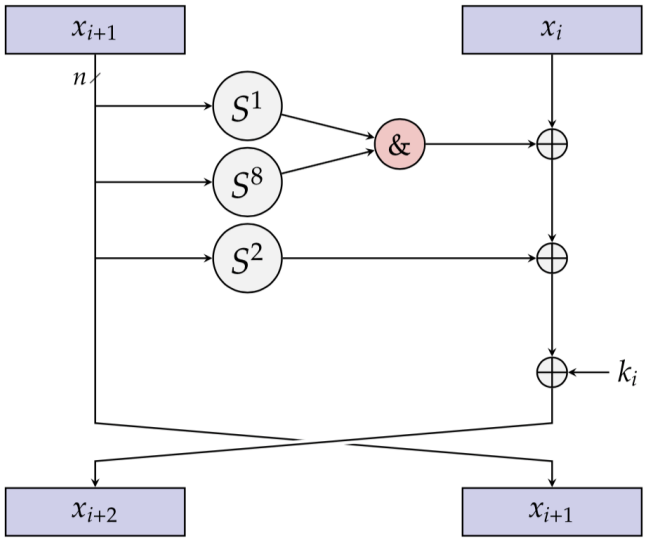

Figura 3.2 – Rede de Feistel do Algoritmo Simon [Beaulieu et al., 2015b]

A cada *round* são feitas 3 operações *XOR* representadas pelo símbolo $\oplus$, 3 deslocamentos circulares de j bits representados por S^J e 1 operação *AND* representada por *&*. Um pseudo-código do funcionamento – contendo a geração das chaves mostradas na seção 3.1.1, está disponível no Anexo A. Formalmente, as funções F, de encriptação e de decriptação são definidas respectivamente pelas equações abaixo:

$$f(X) = (XS^1 \ \& \ XS^8) \oplus XS^2 \tag{3.1}$$

$$R_k(X_{upper}\ ,\ X_{lower}) = (\ K_i \oplus f(X_{upper}) \oplus X_{lower},\ X_{upper}\ ) \tag{3.2}$$

$$R_k^{-1}(X_{upper}\ ,\ X_{lower}) = (\ X_{lower},\ K_i \oplus f(X_{lower}) \oplus X_{upper}\ ) \tag{3.3}$$

Assim como no algoritmo DES mostrado em 2.2.2. A função de encriptação tem como entrada e saída os dois lados da rede de Feistel. Na saída, o lado esquerdo receberá o valor computado pelas operações *XOR* da chave com a função *F* e o lado direito. E o lado direito receberá o valor que estava no lado esquerdo.

A decriptação nada mais é do que a inversão do processo, o lado esquerdo receberá o lado direito anterior, e o lado direito o valor computado. Utilizando as chaves de round também invertidas, começando pela última.

Utilizando a propriedades de comutatividade da função *XOR*, *F* foi expressa em função apenas dos escorregamentos laterais para a esquerda, podendo ser utilizada nas expressões de encriptação e de decriptação.

3.1.3 Criptoanálise

Para se estudar o nível de segurança de um algoritmo de criptografia simétrico são utilizados os ataques diferenciais e lineares em rounds reduzidos do algoritmo, ou seja, são analisadas características do circuito ao longo da operação (rounds) de encriptação.

É dito que um ataque teve sucesso quando se obtém informação sobre o estado do circuito em determinado round. Quanto mais rounds são atacados com sucesso, menor é a força do algoritmo.

A maioria dos algoritmos de criptografia por bloco é suscetível à ataques de canais laterais devido as emissões de energia fortemente relacionadas a chave compartilhada. Descrever a natureza e funcionamento dos ataques e técnicas para mitiga-los não é o escopo deste projeto.

O trabalho realizado por Farzaneh Abed e outros em [Abed et al., 2013] fornece algumas informações sobre o estudo cripto-analítico do algoritmo Simon, focado principalmente nos ataques diferenciais.

Os autores consideram o algoritmo seguro principalmente devido a estrutura de seu gerador de chaves que "protege muito eficientemente contra escorregamentos, rotações e ataques *meet-in-the-middle* por um número considerável de rounds".

As principais preocupações estão em relação a falta de operações não-lineares, tornando o algoritmo suscetível à análise diferencial, sendo possível obter sucesso em ataques em aproximadamente 30% dos rounds chegando a mais de 50% em alguns casos.

4. IMPLEMENTAÇÃO

Os capítulos anteriores corresponderam ao estudo necessário para a execução do presente trabalho. Os próximos capítulos apresentam a principal contribuição deste TCC, que é o desenvolvimento do módulo de criptografia leve.

O sistema foi implementado em duas entidades: *key scheduler* e *core*, que executam os funcionamentos descritos em 3.1.1 e 3.1.2, respectivamente. A organização das entidades está disposta como apresentado na Figura 4.1 abaixo.

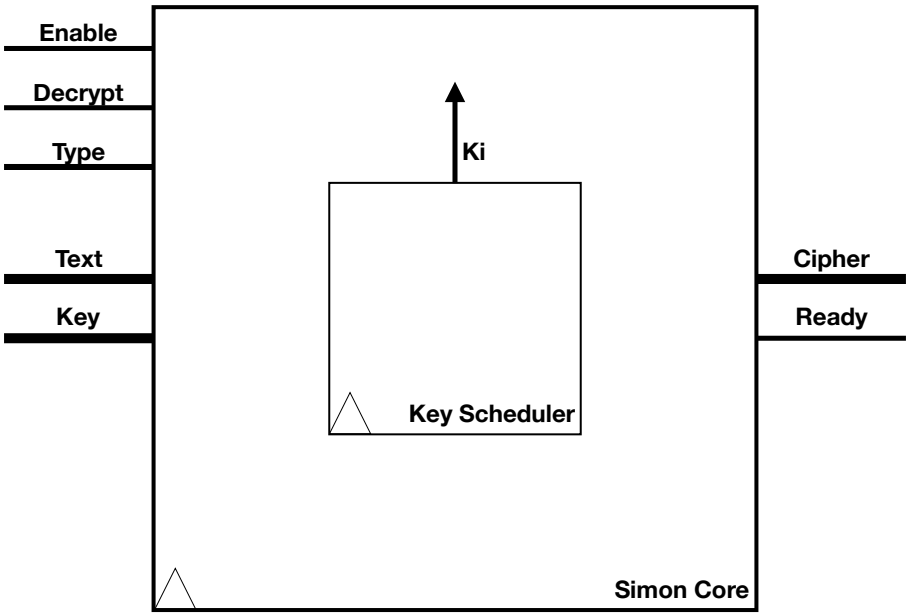

Figura 4.1 – Estrutura do módulo Simon. Fonte: O Autor.

São entradas do sistema os sinais *Decrypt e Type* que são utilizados para fins de configuração do modo de operação (encriptação ou decriptação) e qual dos algoritmos apresentados na tabela 3.1 será executado.

Os sinais *Text e Key* definem o texto a ser encriptado ou decriptado e a chave compartilhada a ser utilizada. *Enable* inicia a operação. A saída *Cipher* conterá o texto encriptado/decriptado e *Ready* indica o final da operação.

4.1 Gerador de Chaves

O comportamento inicial do gerador de chaves depende do cenário em que o sistema se encontra ao inicializá-lo. Após passar por uma das situações abaixo ele gerará as chaves normalmente seguindo o que foi abordado na seção 3.1.1 juntamente com o que está explicado a seguir. São três os possíveis cenários e as reações relativas à eles:

- Cenário 1 - Há uma encriptação a ser realizada: Começa a geração das chaves imediatamente.
- Cenário 2 - Há uma decriptação a ser realizada e esta é a primeira vez que a operação é solicitada ou os parâmetros (chave e/ou tipo) mudaram desde a última execução: Realiza-se o processo de *setup*, e começa a geração das chaves.
- Cenário 3 - Há uma decriptação a ser realizada e esta não é a primeira vez que a operação é solicitada e os parâmetros não mudaram desde a última execução. Carrega-se as chaves salvas ao final do último *setup*, e começa a geração das chaves.

O *setup* consiste em gerar todas as chaves de round para poder realizar a inversão do processo de geração de chaves, ou seja, obtêm-se a última chave e começa-se a geração utilizando-a como chave primária. Evitando assim o uso de memória, diminuindo a área do circuito. Ao final do *setup* a última chave é guardada em registradores para evitar o reprocessamento a cada decriptação.

Apesar do algoritmo de encriptação implementado no *Core* ser totalmente inversível, a geração das chaves não é. A figura 4.2 mostra a nova organização do gerador de chaves para o caso $m = 4$.

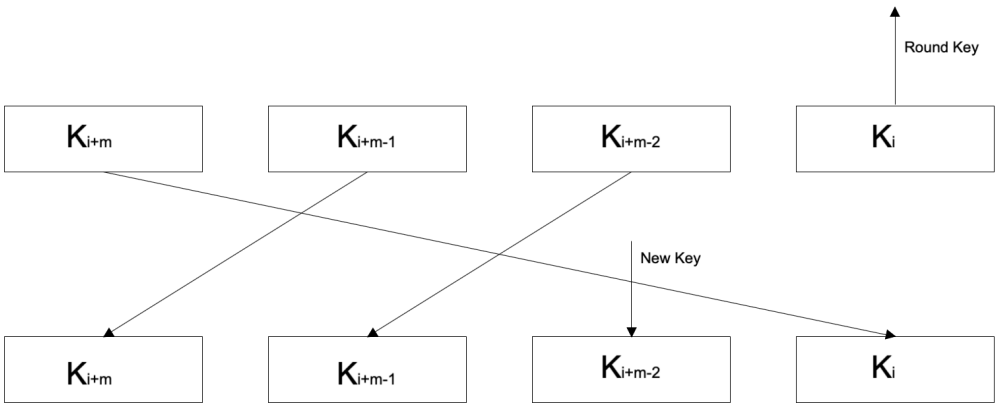

Figura 4.2 – Organização do gerador de chaves na decriptação. Fonte: O Autor.

Além da necessidade de iniciar a partir da última chave de round, é preciso alterar o uso dos registradores para reutilizar o hardware do gerador de chaves para a decriptação.

Da forma que para o gerador invertido $K_i = K_{i+m}$, $K_{i+m} = K_{i+m-1}$ e K_{i+1} recebe a nova chave gerada a cada iteração, tendo $0 \leqslant i < T - m$. Um trecho do código VHDL implementado é mostrado abaixo.

```
when M4 =>  key_left    <= key_2;
            key_2       <= key_1;
            key_1       <= reg_2 xor reg_1;
            key_right   <= key_left;
```

A necessidade de estruturas como esta implica na instanciação de diversos multiplexadores em hardware, aumentando sua área e consumo energético.

A máquina de estados (FSM – *Finite State Machine*), apresentada na Figura 4.3 corresponde ao funcionamento do módulo *key scheduler*. Os estados *CHECK e INIT DECRYPT* são utilizados para verificação de cenário e salvamento das chaves após o *setup* mencionados acima.

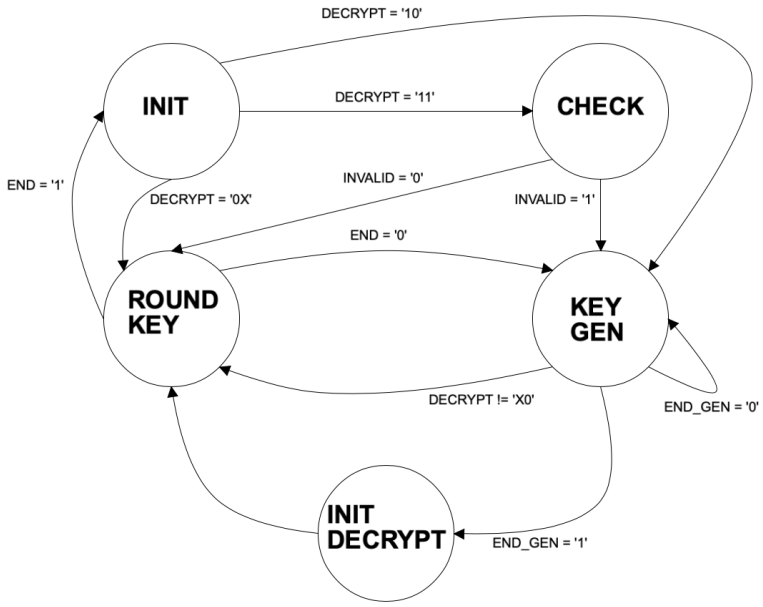

Figura 4.3 – Máquina de estados da entidade *KEY SCHEDULER*. Fonte: O Autor.

Os estados *ROUND KEY e KEY GEN* concentram a operação de geração de chaves apresentada em 3.1.1 e complementada da forma descrita nesta seção, como mostram os trechos do código da descrição do hardware abaixo.

```
round_key <= key_right;
reg_3  <= key_1 xor (key_left(2 downto 0) & key_left(63 downto 3)); -- SHIFT_3
reg_11 <= reg_3 xor key_right;
reg_22 <= reg_3(0) & reg_3(63 downto 1); -- SHIFT_1
reg_2  <= C_CONSTANT (63 downto 1) & Z_SEQUENCE(mod_counter); -- C xor Z(i)
reg_1  <= reg_11 xor reg_22;
```

```
when S_KEY_GEN    => key_right <= key_left;
                     key_left  <= reg_2 xor reg_1;
when S_ROUND_KEY => if key_request = '1' then
                         NEXT_STATE <= S_KEY_GEN;
                     end if;
```

Para melhor compreensão do código, o processo combinacional foi separado nos sinais *reg_1, reg_2, reg_3, reg_11 e reg_22*. A figura 4.4 apresenta a posição dos sinais criados no diagrama de blocos do caso em que *m = 4* usado como exemplo. A cada nova chave computada é feita a rotação indicada pelas setas e é salvo no registrador *key_left* a nova chave, resultado da operação *reg_2* $\oplus$ *reg_1*.

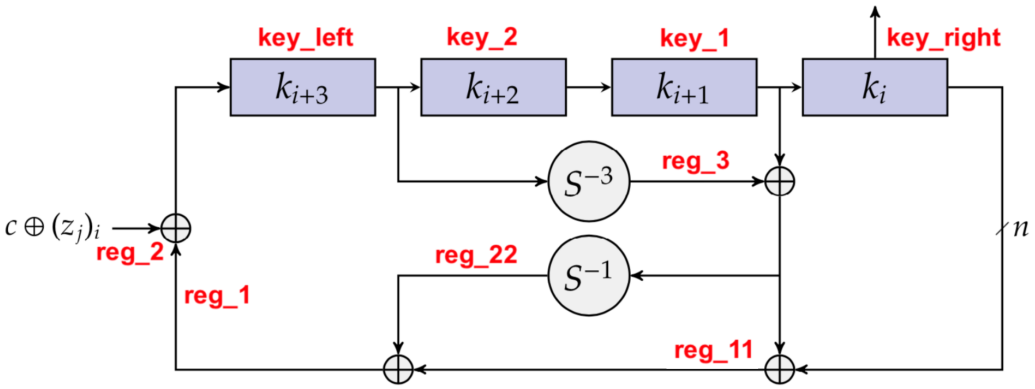

Figura 4.4 – Geração das chaves conforme o VHDL. Fonte: O Autor.

No estado *ROUND KEY* é feita a leitura da chave de round (*round key*) pelo *core*, indicada pela ativação do sinal *key request* oriundo do mesmo. A chave de round será sempre o valor contido no registrador *key_right*. No estado *KEY GEN* são atualizados os registradores do escalonador. A cada leitura a nova chave é gerada combinacionalmente quando os registradores são atualizados.

4.2 Core

A entidade *core* funciona de forma análoga a do *key scheduler*, concentrando sua operação em dois estados e alternando entre eles, os estados *SHIFT* e *NEXT*. Um melhor entendimento da operação pode ser alcançado através da análise das figuras 4.5 e 4.6, que ilustram funcionamento descrito pelo trecho do código mostrado abaixo.

```
shift_1     <= left_block(62 downto 0) & left_block(63);
shift_2     <= left_block(61 downto 0) & left_block(63 downto 62);
shift_8     <= left_block(55 downto 0) & left_block(63 downto 56);
```

```
temp_block  <= round_key xor (shift_2 xor (right_block xor (shift_1 and shift_8)));

when S_ENCRYPT_SHIFT => key_request <= '1';

when S_ENCRYPT_NEXT  => key_request <= '0';
                        right_block <= left_block;
                        left_block  <= temp_block;
```

Também por uma melhor compreensão do código, o processo combinacional do *core* foi separado nos sinais *shift_1, shift_2, shift_8 e temp_block*. A figura 4.5 ilustra a posição dos sinais criados no diagrama de blocos mostrado em 3.1.2.

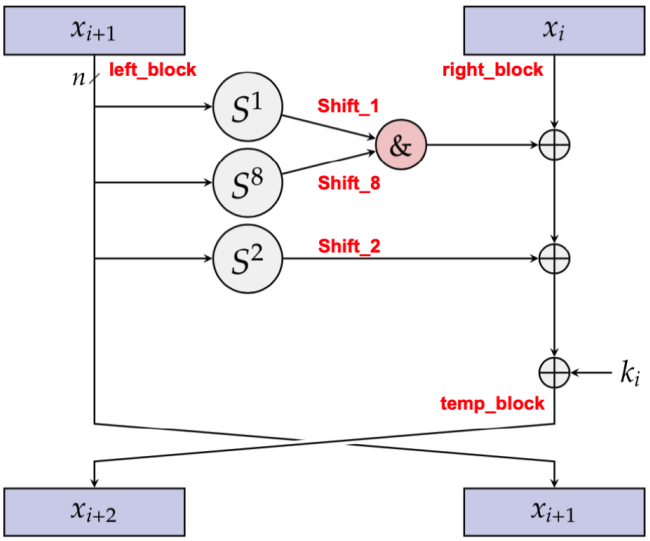

Figura 4.5 – Geração das cifras conforme o VHDL. Fonte: O Autor.

Para cada versão do *Simon 2n/mn*, os *n* bits mais significativos da entrada *TEXT* serão salvos no registrador *left_block* e os *n* bits menos significativos serão salvos no registrador *right_block*. A cada round será registrado em *left_block* a encriptação ou decriptação na forma descrita pelas equações 3.1, 3.2 e 3.3 e invertidos os lados.

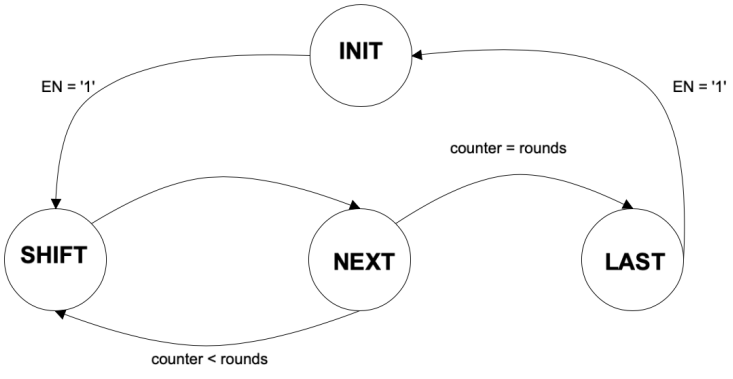

Figura 4.6 – Máquina de estados da entidade *CORE*. Fonte: O Autor.

No estado *SHIFT* é feita a leitura da chave de round presente no barramento do *key scheduler* e a computação da cifra do round no sinal *temp_block*, de forma combinacional.

No estado *NEXT* são feitos os registros do novo lado esquerdo e direito. Estes estados alternam entre si round vezes, de acordo com o tipo de Simon escolhido e os valores da tabela 3.1.

No estado *LAST* são ativados o sinal *READY*, colocadas na saída *CIPHER* as cifras com o lado esquerdo e direito invertidos, e o módulo colocado em espera enquanto uma nova requisição não é realizada.

5. VALIDAÇÃO E ANÁLISE

A validação do módulo foi feita através da simulação e síntese para FPGA e ASIC da implementação descrita no capítulo anterior. Neste capítulo são discutidos os resultados das simulações, e posteriormente é feita uma comparação com uma implementação do algoritmo AES obtida na plataforma *opencores*. Os relatórios das sínteses de ambos os módulos podem ser encontrados nos anexos B e C.

As formas de onda apresentadas nas seções seguintes são referentes à uma versão do hardware desenvolvida para ser comparada ao AES. É a opção Simon 128/128 pura, ou seja, foram retiradas do projeto do hardware as outras 9 versões e a possibilidade de escolha.

Essa ação foi tomada para obter uma comparação mais justa entre os dois módulos, tendo em vista que a implementação com 10 versões utiliza uma área e potência muito maior devido ao número elevado de multiplexadores e registradores, e o AES não possui uma versão parametrizável.

5.1 Validação Funcional - Testbench

Para a verificação do correto funcionamento do sistema foi desenvolvido um *testbench* e realizada a simulação do circuito no software ModelSim, verificando as formas de onda e a coerência de arquivos de entrada e saída gerados. A estrutura do *testbench* é apresentada na figura 5.1 abaixo.

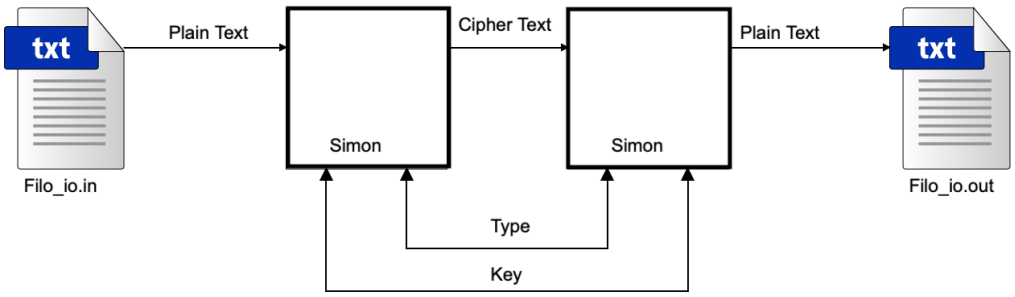

Figura 5.1 – Estrutura do Testbench. Fonte: O Autor.

São feitas leituras de blocos de dados de 128 bits de um arquivo de entrada, e os vetores utilizados como entrada de um módulo Simon instanciado em modo de encriptação. Um segundo módulo é instanciado no modo de decriptação e recebe como entrada a saída do primeiro. A Figura 5.2 apresenta uma iteração da simulação.

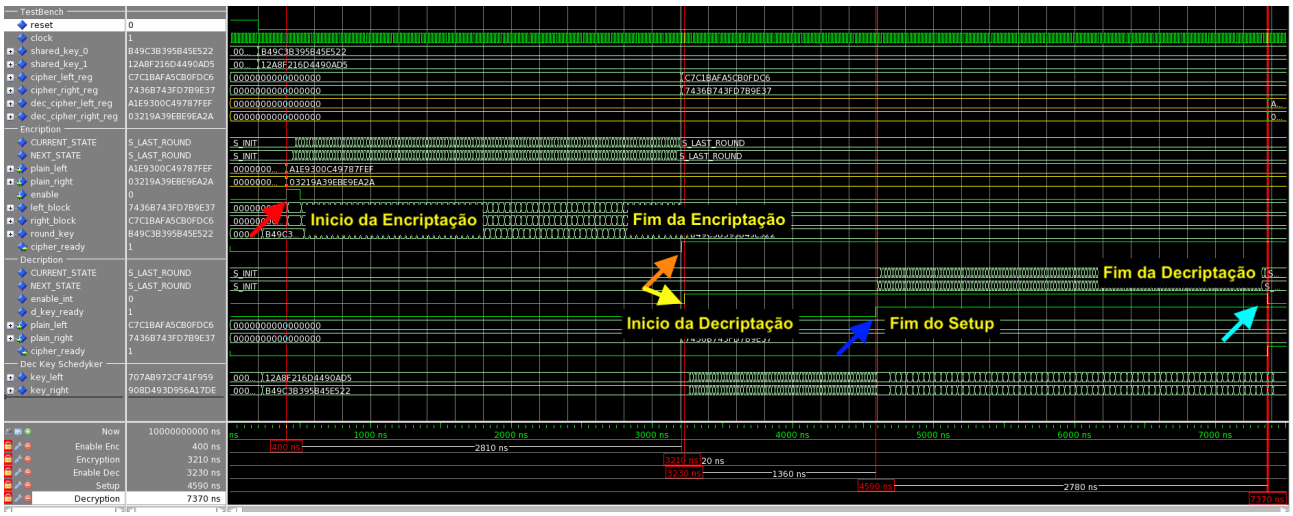

Figura 5.2 – Iteração encriptação/decriptação. Fonte: O Autor.

Primeiramente é acionado o sinal *enable* do módulo Simon em modo de criptografia com, assim que a encriptação termina - indicada pelo sinal *ready* do mesmo módulo -, a saída deste é utilizada como entrada do segundo módulo Simon, em modo de decriptação, e o sinal *enable* deste é acionado. Após o fim do setup, indicado pelo sinal *d_key_ready* incia a operação de decriptação como explicado anteriormente. Finalmente, é aguardado o acionamento do sinal *ready* do segundo módulo, indicando o fim da iteração.

O número de cifras a ser gerado para cada opção do Simon pode ser definido no testbench, os dados e chaves são gerados aleatoriamente por uma aplicação externa presente no diretório do projeto.

Ao final de cada iteração, a saída do segundo módulo é gravada em um segundo arquivo. Após o encerramento da simulação os dois arquivos devem possuir tantos valores iguais quanto o número de testes realizado, provando que o sistema é inversível e que realiza ambas as operações corretamente. A tabela 5.1 contém os resultados referentes aos ciclos de relógio necessários para *setup*, encriptação e decriptação para cada tipo do algoritmo.

Tabela 5.1 – Resultados da Simulação para a versão com todos os tipos.

Simon *(2n/mn)*	Encriptação	Setup	Decriptação
32/64	68	31	69
48/72	78	38	79
48/96	78	37	79
64/96	90	44	91
64/128	94	45	95
96/96	110	55	111
96/144	114	56	115
128/128	142	71	143
128/192	144	71	145
128/256	150	69	151

Para todos os tipos definidos o tempo de decriptação apresenta 1 ciclo a mais que o tempo de encriptação. Isto ocorre devido a necessidade de realizar a verificação da alteração dos parâmetros como explicado em 4.1. Os tempos de *setup* estão relacionados tanto ao número de palavras de chave quanto ao número de rounds.

5.2 Comparação com AES

Foram levados em consideração para a comparação a área e potência quando sintetizados os circuitos para ASIC, a quantidade de *Look-Up-Tables* (LUT) e *Flip-Flops* (FF) quando sintetizados para FPGA e os tempos de operação (em ciclos de relógio) obtidos na simulação. Um resumo dos resultados é apresentado na tabela 5.2.

- Tempo para Encriptação/Decriptação. Ambos os módulos operam de forma sequencial, sem qualquer técnica de *pipeline* ou paralelização. Foram contabilizados os ciclos do inicio da operação até o aparecimento da cifra.

  Simon: Para a encriptação da versão do Simon analisada são necessários 140 ciclos de relógio. A decriptação leva 1 ciclo a mais, totalizando 141.

  AES: O AES leva os mesmos 19 ciclos para encriptação e decriptação. Mais especificamente, 6 ciclos para carregar as entradas e gerar as saídas, e 13 ciclos para realizar os 11 rounds.

  O AES é aproximadamente 7,3 vezes mais rápido que o Simon.

- Tempo de *Setup*. O tempo de *setup* foi abordado na seção 4.1.

  Simon: O Simon $128/128$ necessita de 68 ciclos para realizar o setup.

  AES: O AES não possui tempo de *setup*, pois não possui um sistema de geração próprio das chaves de decriptação, ele utiliza sempre a chave fornecida como entrada para realizar as operações.

- ASIC: Área e Potência. Ambos os circuitos foram sintetizados em tecnologia de 65 *nm*, utilizando o software *Genus*, da *Cadence*.

  Simon: possui área de 22.371 μm^2 e consome 16.033.950 nW.

  AES: possui uma área de 113.355 μm^2 e consome 399.233.314 nW.

  o AES é aproximadamente 5 vezes maior em área e 25 vezes menos eficiente energeticamente.

- FPGA. A síntese para FPGA foi realizada no software Vivado da Xilinx.

  Simon: O Simon utiliza 527 LUTs e 812 FFs em sua implementação.

AES: O AES utiliza 2912 LUTs e 914 FFs em sua implementação. O AES utiliza 5,5 vezes mais LUTs e uma quantidade similar de FFs.

Tabela 5.2 – Comparação Simon vs AES.

Algoritmo / Métrica	Simon	AES
Tempo Encriptação (ciclos)	140	19
Tempo Decriptação (ciclos)	141	19
Tempo Setup (ciclos)	68	0
Área (μm)	22.371	113.355
Potência (nW)	16.033.950	399.233.314

O algoritmo Simon atende as características de LWC definidas na seção 2.3, possuindo valores consideravelmente menores que a do AES utilizado para comparação em termos de área, potência, LUTs e FFs quando analisados os resultado das sínteses para ASIC e FPGA. Em contraponto, é bastante mais lento.

É importante ressaltar que é possível aplicar técnicas de paralelismo e/ou *pipeline* ao algoritmo como mostrado na figura 5.3, impactando em sua área, consumo energético e performance.

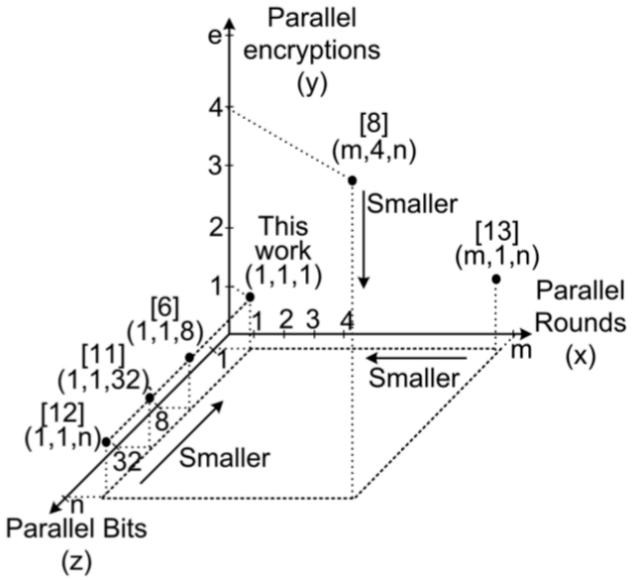

Figura 5.3 – Relação entre técnicas de paralelismo e área do Simon [Aysu et al., 2014].

A paralelização de encriptação consiste em instanciar o hardware diversas vezes e executar diversas encriptações simultaneamente. Paralelizar rounds resume-se a expandir o bloco combinacional, computando *m* rounds por ciclo de relógio. O paralelismo de bits é utilizado para diminuir ainda mais a área, operando apenas em *b* bits de cada bloco a cada ciclo de relógio.

6. CONCLUSÃO E TRABALHOS FUTUROS

A era da Internet das Coisas trouxe um aumento considerável no número de dispositivos interconectados, com cada vez menos recursos devido às funcionalidades que agora atendem. O crescimento da quantidade de mensagens trocadas entre eles contém os mais variados tipos de informação, gerando grande preocupação em relação à segurança e à privacidade dos usuários.

O trabalho descrito neste documento apresentou os conceitos necessários para a implementação de um módulo de criptografia leve que pode ser inserido em dispositivos com recursos limitados. Durante a realização deste trabalho foram exploradas áreas de grande importância dentro da Engenharia de Computação, como a segurança da informação e o desenvolvimento de sistemas embarcados. Enfatizando a oportunidade de estudar e aplicar conceitos de criptografia, que não são abordados durante o curso.

O escopo principal do projeto era implementar um módulo em VHDL que atendesse os critérios de criptografia leve. Foi implementado, simulado e sintetizado para ASIC e FPGA o algoritmo Simon da família Simon e Speck.

Um segundo objetivo era a comparação do módulo desenvolvido com um módulo já existente do algoritmo Advanced Encryption Standard. O Simon desenvolvido neste trabalho atendeu aos critérios desejados e se mostrou menor em área e mais eficiente energeticamente do que o AES, apesar da maior latência devido ao maior número de *rounds*.

Esse estudo e desenvolvimento pode servir como base para outros projetos a serem realizados em trabalhos futuros. Podendo ser modificado de forma a serem inseridas técnicas para melhoramento da velocidade e/ou diminuição da área e consumo.

REFERÊNCIAS BIBLIOGRÁFICAS

[Abed et al., 2013] Abed, F., List, E., Lucks, S., and Wenzel, J. (2013). Differential and linear cryptanalysis of reduced-round simon. *Cryptology ePrint Archive, Report 2013/526*, 2013:31.

[Aysu et al., 2014] Aysu, A., Gulcan, E., and Schaumont, P. (2014). Simon says, break the area records for symmetric key block ciphers on fpgas. *IACR Cryptology ePrint Archive*, 2014:237.

[Beaulieu et al., 2015a] Beaulieu, R., Shors, D., Smith, J., and Stefan (2015a). SIMON and SPECK: Block Ciphers for the Internet of Things. https://eprint.iacr.org/2015/585.

[Beaulieu et al., 2015b] Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., and Wingers, L. (2015b). The simon and speck lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE.

[Brown et al., 1992] Brown, S. D., Francis, R. J., Rose, J., and Vranesic, Z. G. (1992). *Field-programmable gate arrays*. Springer.

[Gulcan et al., 2014] Gulcan, E., Aysu, A., and Schaumont, P. (2014). A flexible and compact hardware architecture for the simon block cipher. In *International Workshop on Lightweight Cryptography for Security and Privacy*, pages 34–50. Springer.

[Hanley and ONeill, 2012] Hanley, N. and ONeill, M. (2012). Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers. In *ISVLSI*, pages 57–62.

[Heath, 2002] Heath, S. (2002). *Embedded systems Design*. Elsevier, 1st edition.

[Herigemblong, 2018] Herigemblong (2018). AES Decryption Flow Chart Fresh GALS System Design Side Channel Attack Secure Cryptographic. Online; acessado em Agosto, 2018, http://dailyrevshare.com/aes-decryption-flow-chart/aes-decryption-flow-chart-fresh-gals-system-design-side-channel-attack-secure-cryptographic.

[Katagi and Moriai, 2008] Katagi, M. and Moriai, S. (2008). Lightweight cryptography for the internet of things. *Sony Corporation*, 2008:7–10.

[Kleidermacher and Kleidermacher, 2012] Kleidermacher, D. and Kleidermacher, M. (2012). *Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development*. Newnes, 1st edition.

[Kocarev, 2001] Kocarev, L. (2001). Chaos-based cryptography: a brief overview. *IEEE Circuits and Systems Magazine*, 1(3):16.

[Manifavas et al., 2013] Manifavas, C., Hatzivasilis, G., Fysarakis, K., and Rantos, K. (2013). Lightweight cryptography for embedded systems–a comparative analysis. In Garcia-Alfaro, J., L. G. C.-B. N. F. S. F. W., editor, *Data Privacy Management and Autonomous Spontaneous Security*, chapter 20, pages 333–349. Springer.

[Petitcolas, 1883] Petitcolas, F. (1883). La cryptographie militaire. Online; acessado em Agosto, 2018, http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf.

[Rabaiei and Harous, 2016] Rabaiei, K. A. A. and Harous, S. (2016). Internet of things: Applications and challenges. In *IIT*, pages 1–6.

[Schinianakis, 2017] Schinianakis, D. (2017). Alternative security options in the 5G and IoT era. *IEEE Circuits and Systems Magazine*, 17(4):6–28.

[Smith, 1997] Smith, M. J. S. (1997). *Application-specific integrated circuits*. Addison-Wesley.

[Stallings, 2015] Stallings, W. (2015). *Criptografia e segurança de redes: Princípios e Práticas*. Pearson Education do Brasil, 6th edition.

[Wikipedia, 2018] Wikipedia (2018). Feistel cipher. Online; acessado em Agosto, 2018, https://en.wikipedia.org/wiki/Feistel_cipher#/media/File:Feistel_cipher_diagram_en.svg.

[Yalla and Kaps, 2009] Yalla, P. and Kaps, J.-P. (2009). Lightweight cryptography for FPGAs. In *RECONFIG*, pages 225–230.

ANEXO A – Pseudo-Código do Simon

Este anexo detalha na Figura A.1 a operação do algoritmo Simon, detalhado na seção 3.1.2, .

```
------------------------- definitions -------------------------
  n = word size (16, 24, 32, 48, or 64)
  m = number of key words (must be      4 if n = 16,
                                    3 or 4 if n = 24 or 32,
                                    2 or 3 if n = 48,
                                 2, 3, or 4 if n = 64)

  z = [11111010001001010110000111001101111101000100101011000011100110,
       10001110111110010011000010110101000111011111001001100001011010,
       10101111011100000011010010011000101000010001111110010110110011,
       11011011101011000110010111100000010010001010011100110100001111,
       11010001111001101011011000100000010111000011001010010011101111]

  (T, j) = (32,0)                    if n = 16
         = (36,0) or (36,1)          if n = 24, m = 3 or 4
         = (42,2) or (44,3)          if n = 32, m = 3 or 4
         = (52,2) or (54,3)          if n = 48, m = 2 or 3
         = (68,2), (69,3), or (72,4) if n = 64, m = 2, 3, or 4

  x,y          = plaintext words
  k[m-1]..k[0] = key words
------------------------ key expansion ------------------------
for i = m..T-1
   tmp  ← S^-3 k[i-1]
   if (m = 4) tmp ← tmp ⊕ k[i-3]
   tmp  ← tmp ⊕ S^-1 tmp
   k[i] ← ~k[i-m] ⊕ tmp ⊕ z[j][(i-m) mod 62] ⊕ 3
end for
------------------------- encryption -------------------------
for i = 0..T-1
   tmp ← x
   x   ← y ⊕ (Sx & S^8 x) ⊕ S^2 x ⊕ k[i]
   y   ← tmp
end for
```

Figura A.1 – Pseudo-código do algoritmo Simon. [Beaulieu et al., 2015b]

ANEXO B – Relatórios Síntese ASIC

Este anexo detalha os relatórios de consumo energético e área do circuito dos algoritmos AES e Simon citados na seção 5.

```
-- AES POWER
                  Leakage     Dynamic       Total
Instance   Cells  Power(nW)   Power(nW)    Power(nW)
-------------------------------------------------------
aes128_fast  19102 988246.144 38935067.667 39923313.811
  expand_key  4302 278606.729  7619742.856  7898349.586

-- AES AREA

Instance      Module     Cells  Cell Area  Net Area     Total Area
-------------------------------------------------------------------
aes128_fast               19102      67921     45434        113355
  expand_key key_expander  4302      16005      9169         25174


-- SIMON POWER
                 Leakage     Dynamic       Total
Instance Cells  Power(nW)   Power(nW)    Power(nW)
-----------------------------------------------------
simon     2601 195277.630 15838672.170 16033949.800

-- SIMON AREA

Instance Module  Cells  Cell Area  Net Area   Total Area
-----------------------------------------------------------
simon             2601      15986      6385        22371
```

Figura B.1 – Relatórios ASIC.

ANEXO C – Relatórios Síntese FPGA

Este anexo detalha os relatórios de utilização de LUTs e FFs dos algoritmos AES e Simon citados na seção 5.

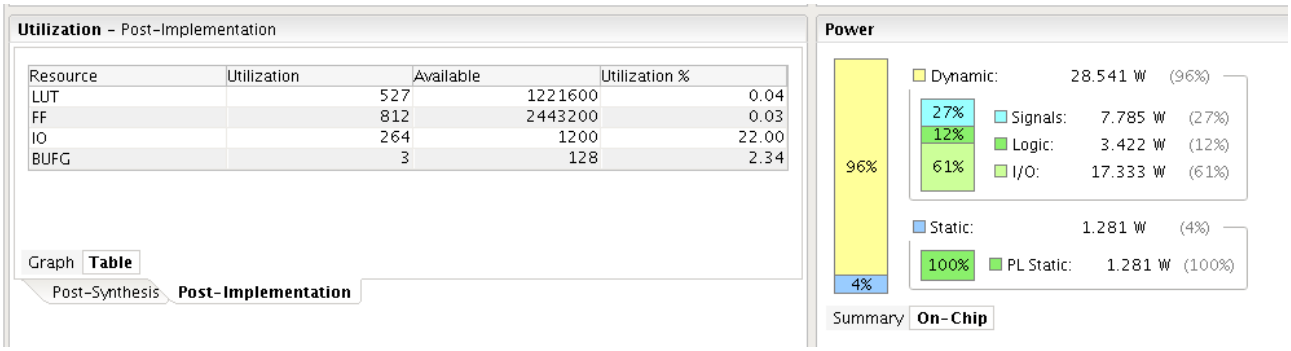

Figura C.1 – Relatório da síntese FPGA do Simon.

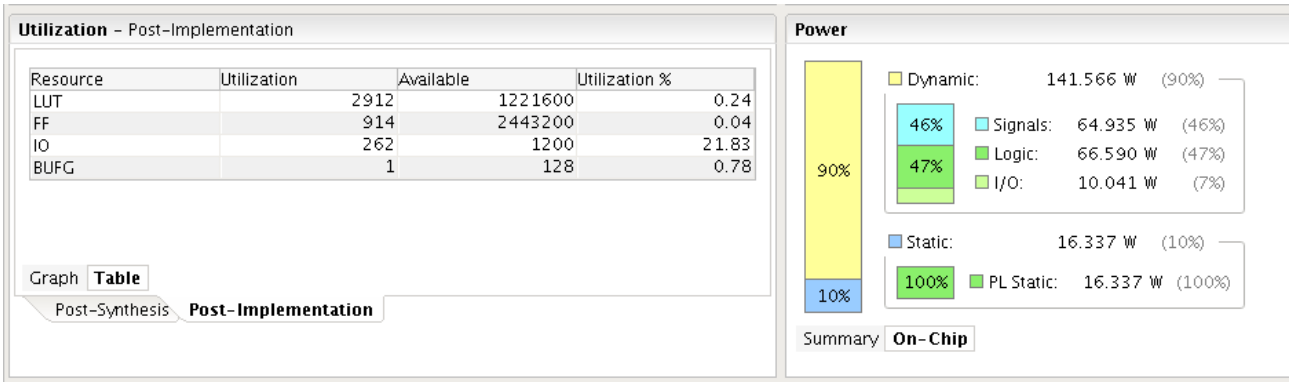

Figura C.2 – Relatório da síntese FPGA do AES.